Welcome to the Krita 4.4 Manual!

Welcome to Krita's documentation page.

Krita is a sketching and painting program designed for digital artists. Our vision for Development of Krita is —

> Krita is a free and open source cross-platform application that offers an end-to-end solution for creating digital art files from scratch. Krita is optimized for frequent, prolonged and focused use. Explicitly supported fields of painting are illustrations, concept art, matte painting, textures, comics and animations. Developed together with users, Krita is an application that supports their actual needs and workflow. Krita supports open standards and interoperates with other applications.

Krita's tools are developed keeping the above vision in mind. Although it has features that overlap with other raster editors its intended purpose is to provide robust tool for digital painting and creating artworks from scratch. As you learn about Krita, keep in mind that it is not intended as a replacement for Photoshop. This means that the other programs may have more features than Krita for image manipulation tasks, such as stitching together photos, while Krita's tools are most relevant to digital painting, concept art, illustration, and texturing. This fact accounts for a great deal of Krita's design.

You can download this manual as an epub here
[https://docs.krita.org/en/epub/KritaManual.epub].

User Manual

Discover Krita's features through an online manual. Guides to help you transition from other applications.

Tutorials and Howto's

Learn through developer and user generated tutorials to see Krita in action.

Getting Started

New to Krita and don't know where to start?

Reference Manual

A quick run-down of all of the tools that are available.

General Concepts

Learn about general art and technology concepts that are not specific to Krita.

Krita FAQ

Find answers to the most common questions about Krita and what it offers.

Index

An index of the manual for searching terms by browsing.

Resources

Textures, brush packs, and python plugins to help add variety to your artwork.

User Manual

Discover Krita’s features through an online manual. Guides to help you transition from other applications.

Contents:

Getting Started

Welcome to the Krita Manual! In this section, we'll try to get you up to speed.

If you are familiar with digital painting, we recommend checking out the Introduction Coming From Other Software category, which contains guides that will help you get familiar with Krita by comparing its functions to other software.

If you are new to digital art, just start with Installation, which deals with installing Krita, and continue on to Starting Krita, which helps with making a new document and saving it, Basic Concepts, in which we'll try to quickly cover the big categories of Krita's functionality, and finally, Navigation, which helps you find basic usage help, such as panning, zooming and rotating.

When you have mastered those, you can look into the dedicated introduction pages for functionality in the User Manual, read through the overarching concepts behind (digital) painting in the General Concepts section, or just search the Reference Manual for what a specific button does.

Contents:

Installation

Windows

Windows users can download Krita from the website, the Windows Store, or Steam.

The versions on the Store and Steam cost money, but are functionally identical [https://krita.org/en/item/krita-available-from-the-windows-store/] to the (free) website version. Unlike the website version, however, both paid versions get automatic updates when new versions of Krita comes out. After deduction of the Store fee, the purchase cost supports Krita development.

Website:

The latest version is always on our website [https://krita.org/download/].

The page will try to automatically recommend the correct architecture (64- or 32-bit), but you can select "All Download Versions" to get more choices. To determine your computer architecture manually, go to *Settings* ▸ *About*. Your architecture will be listed as the *System Type* in the *Device Specifications* section.

Krita by default downloads an **installer EXE**, but you can also download a **portable ZIP file** version instead. Unlike the installer version, this portable version does not show previews in Windows Explorer automatically. To get these previews with the portable version, also install Krita's **Windows Shell Extension** extension (available on the download page).

These files are also available from the KDE download directory [https://download.kde.org/stable/krita/].

Windows Store:

For a small fee, you can download Krita from the Windows Store [https://www.microsoft.com/store/productId/9N6X57ZGRW96]. This version requires

Windows 10.

Steam:

For a small fee, you can also download Krita from Steam [https://store.steampowered.com/app/280680/Krita/].

To download a portable version of Krita go to the KDE [https://download.kde.org/stable/krita/] download directory and get the ZIP file instead of the setup.exe installer.

Note

Krita requires Windows 7 or newer. The Store version requires Windows 10.

Linux

Many Linux distributions package the latest version of Krita. Sometimes you will have to enable an extra repository. Krita runs fine under most desktop enviroments such as KDE, Gnome, LXDE, Xfce etc. – even though it is a KDE application and needs the KDE libraries. You might also want to install the KDE system settings module and tweak the gui theme and fonts used, depending on your distributions.

Nautilus/Nemo file extensions

Since April 2016, KDE's Dolphin file manager shows KRA and ORA thumbnails by default, but Nautilus and it's derivatives need an extension. We recommend Moritz Molch's extensions for XCF, KRA, ORA and PSD thumbnails [https://moritzmolch.com/1749].

Appimages

For Krita 3.0 and later, first try out the appimage from the website. **90% of the time this is by far the easiest way to get the latest Krita.** Just download

the appimage, and then use the file properties or the bash command chmod to make the appimage executable. Double click it, and enjoy Krita. (Or run it in the terminal with ./appimagename.appimage)

- Open the terminal into the folder you have the appimage.
- Make it executable:

```
chmod a+x krita-3.0-x86_64.appimage
```

- Run Krita!

```
./krita-3.0-x86_64.appimage
```

Appimages are ISOs with all the necessary libraries bundled inside, that means no fiddling with repositories and dependencies, at the cost of a slight bit more disk space taken up (And this size would only be bigger if you were using Plasma to begin with).

Ubuntu and Kubuntu

It does not matter which version of Ubuntu you use, Krita will run just fine. However, by default, only a very old version of Krita is available. You should either use the appimage, flatpak or the snap available from Ubuntu's app store. We also maintain a ppa for getting latest builds of Krita, you can read more about the ppa and install instructions here [https://launchpad.net/~kritalime/+archive/ubuntu/ppa].

OpenSUSE

The latest stable builds are available from KDE:Extra repo:

- https://download.opensuse.org/repositories/KDE:/Extra/

Note

Krita is also in the official repos, you can install it from Yast.

Fedora

Krita is in the official repos, you can install it by using packagekit (Add/Remove Software) or by writing the following command in terminal.

```
dnf install krita
```

You can also use the software center such as gnome software center or Discover to install Krita.

Debian

The latest version of Krita available in Debian is 3.1.1. To install Krita type the following line in terminal:

```
apt install krita
```

Arch

Arch Linux provides krita package in the Extra repository. You can install Krita by using the following command:

```
pacman -S krita
```

You can also find Krita pkgbuild in arch user repositories but it is not guaranteed to contain the latest git version.

Flatpak

We also have Flatpak for nightlies and stable builds, these builds are not maintained by the core developers themselves. You can either get the builds from the KDE community website [https://binary-factory.kde.org] or from the Flathub Maintainers [https://flathub.org/apps/details/org.kde.krita].

To install flatpak build from the software centre just open the flatpakrepo files with Discover or the software center provided by your distribution:

- Flathub Repo [https://flathub.org/repo/flathub.flatpakrepo]

KDE Flatpak Repo [https://distribute.kde.org/kdeapps.flatpakrepo]

After adding one of the above repos you can then search for Krita and the software center will show you the flatpak version for installation.

If you prefer doing it from terminal you can use the following commands to install Krita's flatpak build

For KDE Flatpak Repo:

```
flatpak remote-add --if-not-exists kdeapps --from
https://distribute.kde.org/kdeapps.flatpakrepo

flatpak install kdeapps org.kde.krita
```

For installing it from Flathub Repo:

```
flatpak remote-add --if-not-exists flathub
https://flathub.org/repo/flathub.flatpakrepo

flatpak install kdeapps org.kde.krita
```

Snaps

There are snap packages provided by the ubuntu snap developers, these are generally not up to date. The Krita Developers do not provide or build the snap packages themselves. To install Krita as a snap package, first install snapd application. Snapd is installed by default on ubuntu distributions.

If you are on ubuntu distribution then Krita's snap package may show up in the software center or you can run the following command in terminal

```
sudo snap install krita
```

Note

The Flatpak and Snap builds are not tested by the core developers of Krita, so you may encounter some bugs while running Krita installed from them.

OS X

You can download the latest binary from our website [https://krita.org/download/krita-desktop/]. The binaries work only with Mac OSX version 10.12 and newer.

Source

While it is certainly more difficult to compile Krita from source than it is to install from prebuilt packages, there are certain advantages that might make the effort worth it:

- You can follow the development of Krita on the foot. If you compile Krita regularly from the development repository, you will be able to play with all the new features that the developers are working on.
- You can compile it optimized for your processor. Most pre-built packages are built for the lowest-common denominator.
- You will be getting all the bug fixes as soon as possible as well.
- You can help the developers by giving us your feedback on features as they are being developed and you can test bug fixes for us. This is hugely important, which is why our regular testers get their name in the about box just like developers.

Of course, there are also some disadvantages: when building from the current development source repository you also get all the unfinished features. It might mean less stability for a while, or things shown in the user interface that don't work. But in practice, there is seldom really bad instability, and if it is, it's easy for you to go back to a revision that does work.

So… If you want to start compiling from source, begin with the latest build instructions from the guide here.

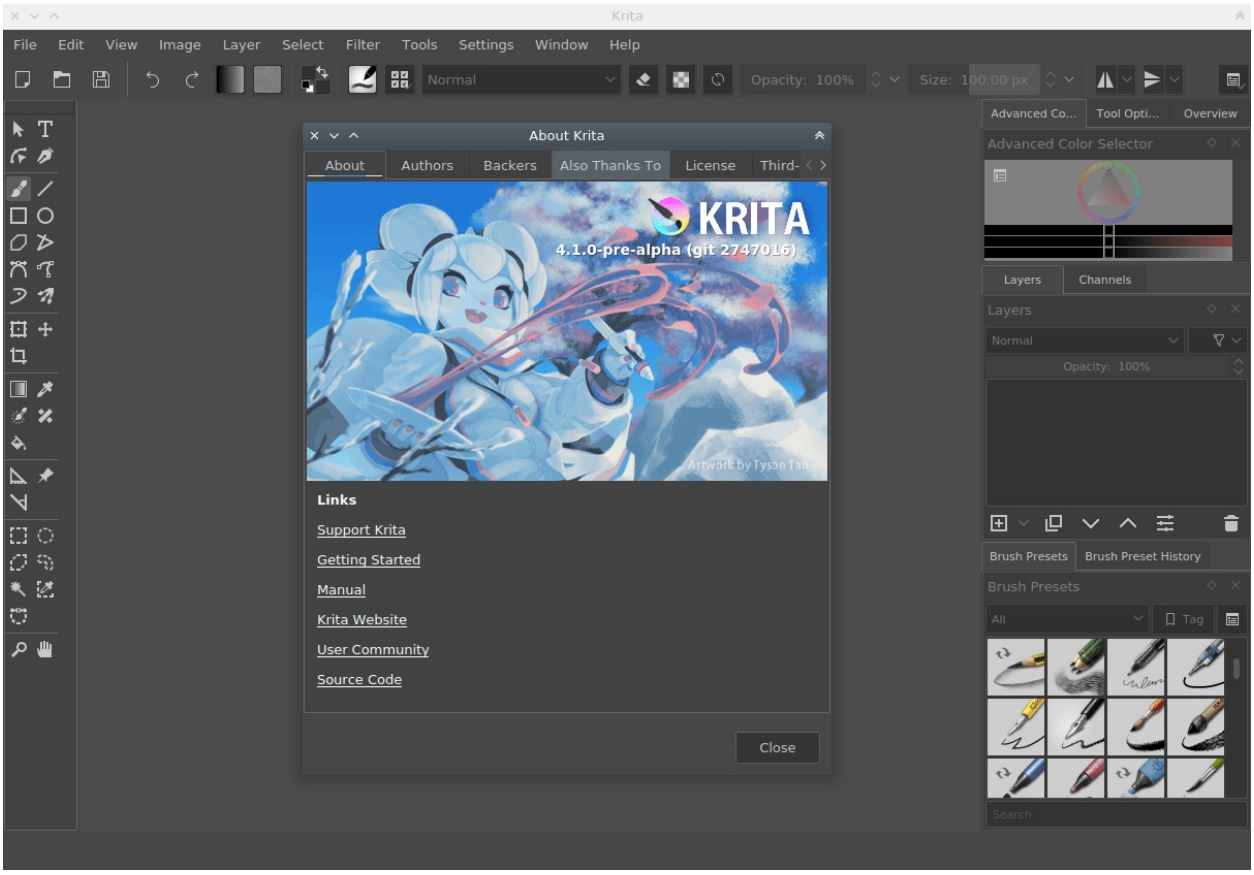
If you encounter any problems, or if you are new to compiling software, don't hesitate to contact the Krita developers. There are three main communication channels:

- irc: webchat.freenode.net, channel #krita

- mailing list [https://mail.kde.org/mailman/listinfo/kimageshop]
- forums [https://forum.kde.org/viewforum.php?f=136]

Starting Krita

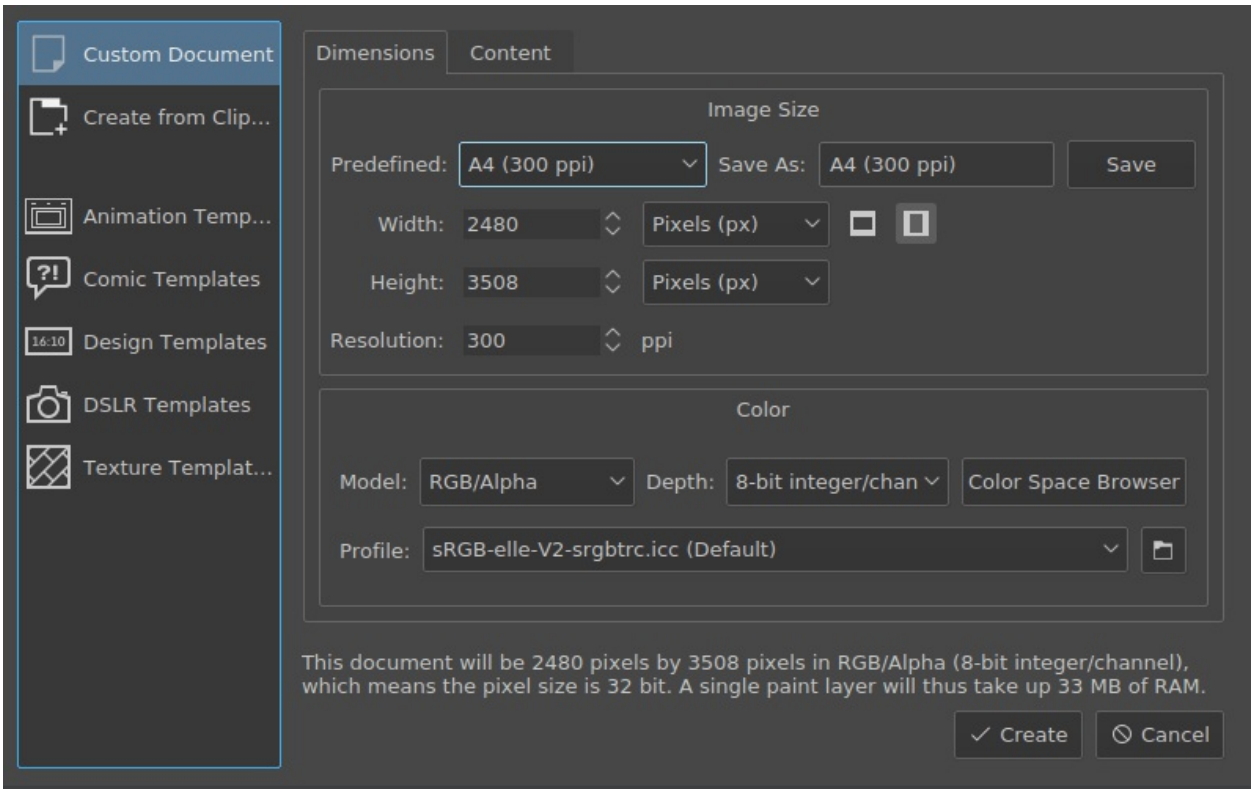
When you start Krita for the first time there will be no canvas or new document open by default. You will be greeted by a welcome screen, which will have option to create a new file or open existing document. To create a new canvas you have to create a new document from the *File* menu or by clicking on *New File* under start section of the welcome screen. This will open the new file dialog box. If you want to open an existing image, either use *File* ▸ *Open…* or drag the image from your computer into Krita's window.



Creating a New Document

A new document can be created as follows.

1. Click on *File* from the application menu at the top.
2. Then click on *New*. Or you can do this by pressing the `Ctrl + N` shortcut.
3. Now you will get a New Document dialog box as shown below:



Click on the *Custom Document* section and in the *Dimensions* tab choose A4 (300ppi) or any size that you prefer from the *Predefined* drop down To know more about the other sections such as create document from clipboard and templates see Create New Document.

Make sure that the color profile is RGB and depth is set to 8-bit integer/channel in the color section. For advanced information about the color and color management refer to Colors.

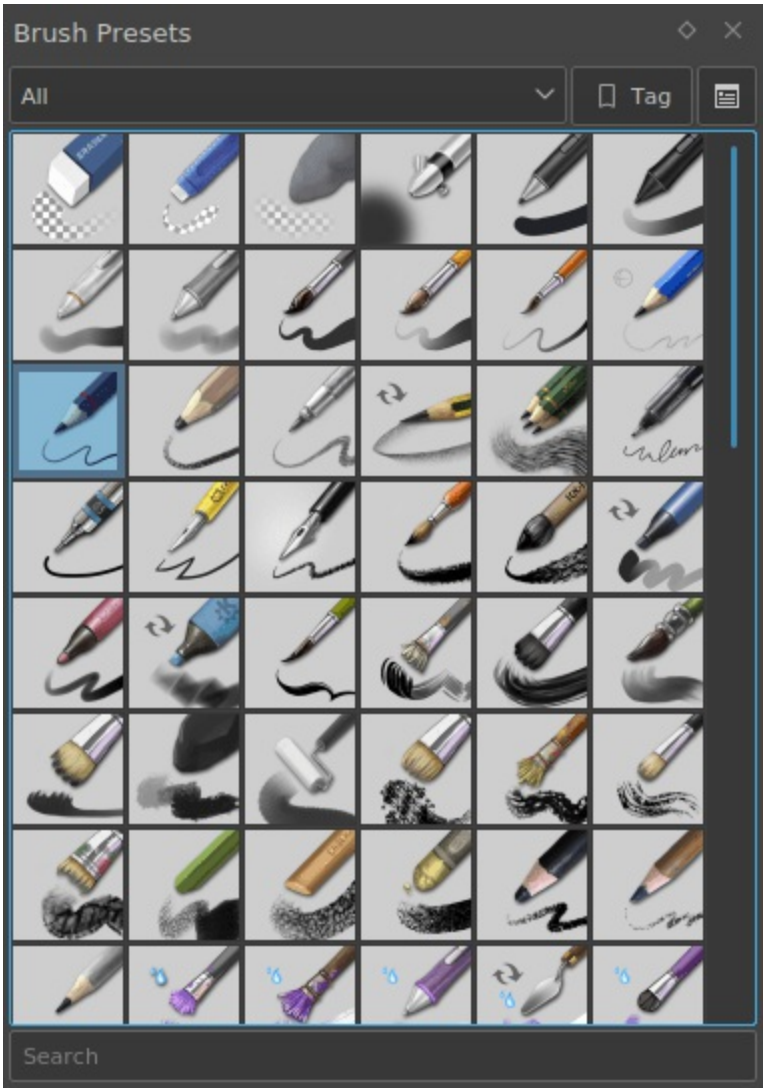
How to use brushes

Now, on the blank white canvas, just left click with your mouse or draw with the pen on a graphic tablet. If everything's correct, you should be able to draw on the canvas! The brush tool should be selected by default when you

start Krita, but if for some reason it is not, you can click on this  icon from the toolbox and activate the brush tool.

Of course, you'd want to use different brushes. On your right, there's a docker named Brush Presets (or on top, press the `F6` key to find this one) with all these cute squares with pens and crayons.

If you want to tweak the presets, check the Brush Editor in the toolbar. You can also access the Brush Editor with the `F5` key.



Tick any of the squares to choose a brush, and then draw on the canvas. To change color, click the triangle in the Advanced Color Selector docker.

Erasing

There are brush presets for erasing, but it is often faster to use the eraser toggle. By toggling the `E` key, your current brush switches between erasing and painting. This erasing method works with most of the tools. You can erase using the line tool, rectangle tool, and even the gradient tool.

Saving and opening files

Now, once you have figured out how to draw something in Krita, you may want to save it. The save option is in the same place as it is in all other computer programs: the top-menu of *File*, and then *Save*. Select the folder you want to have your drawing, and select the file format you want to use (`.kra` is Krita's default format, and will save everything). And then hit *Save*. Some older versions of Krita have a bug and require you to manually type the extension.

If you want to show off your image on the internet, check out the Saving For The Web tutorial.

Check out Navigation for further basic information, Basic Concepts for an introduction as Krita as a medium, or just go out and explore Krita!

Basic Concepts

If this is your first foray into digital painting, this page should give you a brief introduction to the basic but important concepts required for getting started with digital painting in Krita.

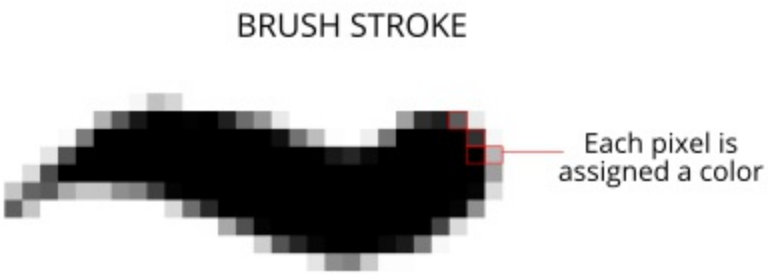
Although very lengthy, this page tries to give a brief overview of some of the Krita's most important functionality; it tries to help you grasp the functions of various menu and buttons in Krita without going into minute details.

Contents

Raster and Vector

Even though Krita is regarded primarily a raster based application, it has some vector editing capabilities as well. If you are new to digital painting medium, it is necessary that you first get yourself acquainted with the concepts of raster and Vector based images.

In digital imaging, a pixel (Picture Element) is a basic and lowest element of an Image. It is basically a grid of points each displaying specific color. Raster editing is manipulating and editing these pixels. For example when you take a 1-pixel brush which is colored black and painting on the white canvas in Krita you are actually changing the color of the pixel beneath your brush from white to black. When you zoom in and see a brush stroke you can notice many small squares with colors, these are pixels:



In contrast to raster images, vector graphic images are based on mathematical expressions. They are independent of the pixels. For example, when you draw a rectangle on a vector layer in Krita you are actually drawing paths passing through points that are called nodes, which are located on specific coordinates on the ‘x’ and ‘y’ axes. When you re-size or move these points the computer calculates and redraws the path and displays the newly formed shape to you. Hence you can re-size the vector shape to any extent without any loss in quality. In Krita, everything which is not on a vector layer is raster

based.

Images, Views and Windows

In a painting program, there are three major containers that make up your work-space.

Image

The most important one is the **Image**.

This is an individual copy of the image that you can open or create via the file dialog. Krita allows you to open the file as a new copy via the *File* menu, or to save it as a new file, or make an incremental copy.

An image contains data regarding layers, color space of image and layers, canvas size and metadata such as creator, date created and DPI et cetera. Krita can open multiple images at once, you can switch between them via the *Window* menu.

Because the image is a working copy of the image on the hard drive, you can do a lot of little saving tricks with it:

New
: Makes a new image. When you press *Save*, you make a new file on the hard drive.

Open…
: Makes an internal copy of an existing image. When you press *Save*, you will overwrite the original existing image with your working copy.

Open existing Document as Untitled Document…
: Similar to *Open…*, however, *Save* will request you to specify a saving location: you're making a new copy. This is similar to *Import…* in other programs.

Create Copy From Current Image

Similar to *Open existing Document as Untitled Document…* but with the currently selected image.

Save Incremental Version
: Allows you to quickly make a snapshot of the current image by making a new file with a version number added to it.

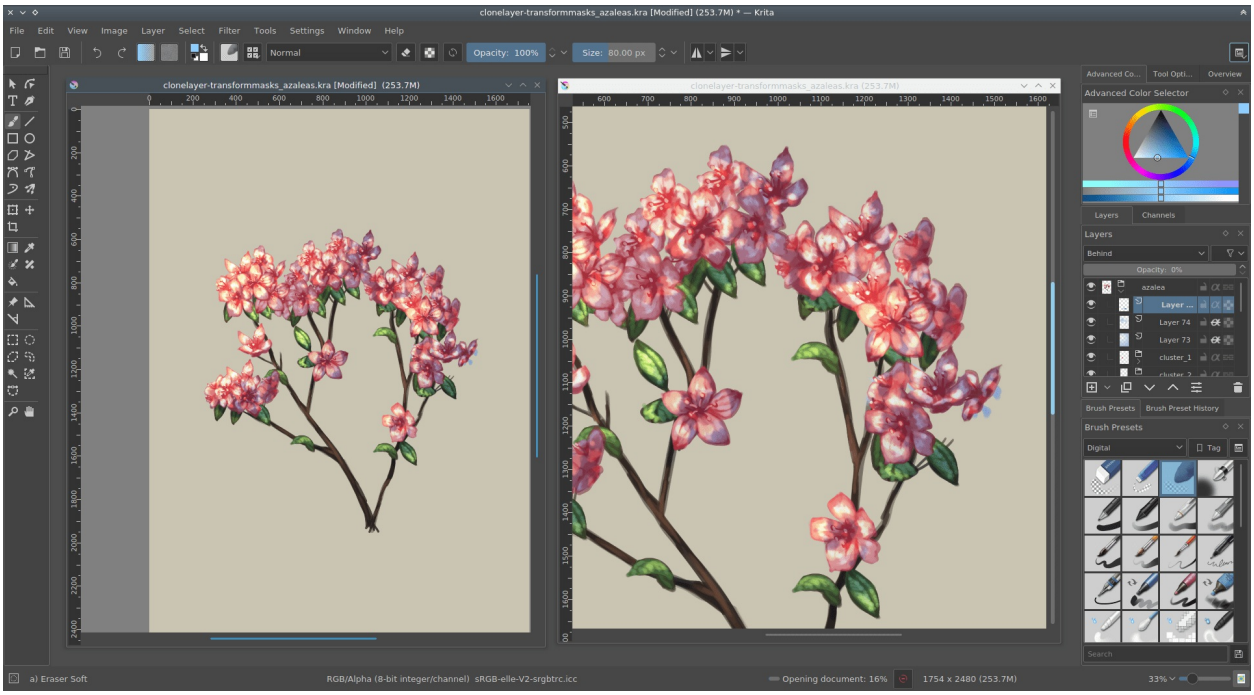
These options are great for people doing production work, who need to switch between files quickly or have backup files in case they do something extreme. Krita also has a file backup system in the form of auto-saves, backup files and crash recovery. You can configure the option for these features in the general settings.

You view the image via a **View**.

View

A view is a window onto your image. Krita allows you to have multiple views, and you can manipulate the view to zoom, rotate and mirror and modify the color of the way you see an image without editing the image itself. This is very useful for artists, as changing the way they view the image is a common way to diagnose some common mistakes, like a drawing which is skewed towards one side. Mirroring with the `M` key makes such skewing easy to identify.

If you have trouble drawing certain curves you will enjoy using rotation for drawing, and of course, there is zooming in and out for precision and rough work.

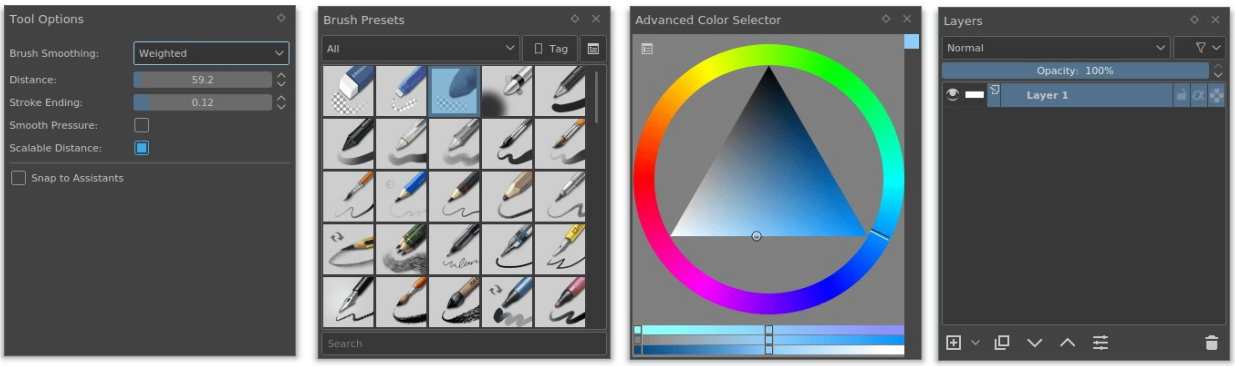


Multiple views of the same image in Krita

Multiple views are possible in Krita via *Window* ▸ *New view* ▸ *image name*. You can switch between them via the *Window* menu, or the `Ctrl + Tab` shortcut, or keep them in the same area when **subwindow** mode is active in the settings, via *Window* ▸ *Tile*.

Dockers

Dockers are little subwindows in Krita's interface. They contain useful tools, like the color selector, layer stack, tool options, et cetera.



The image above shows some of the dockers in Krita.

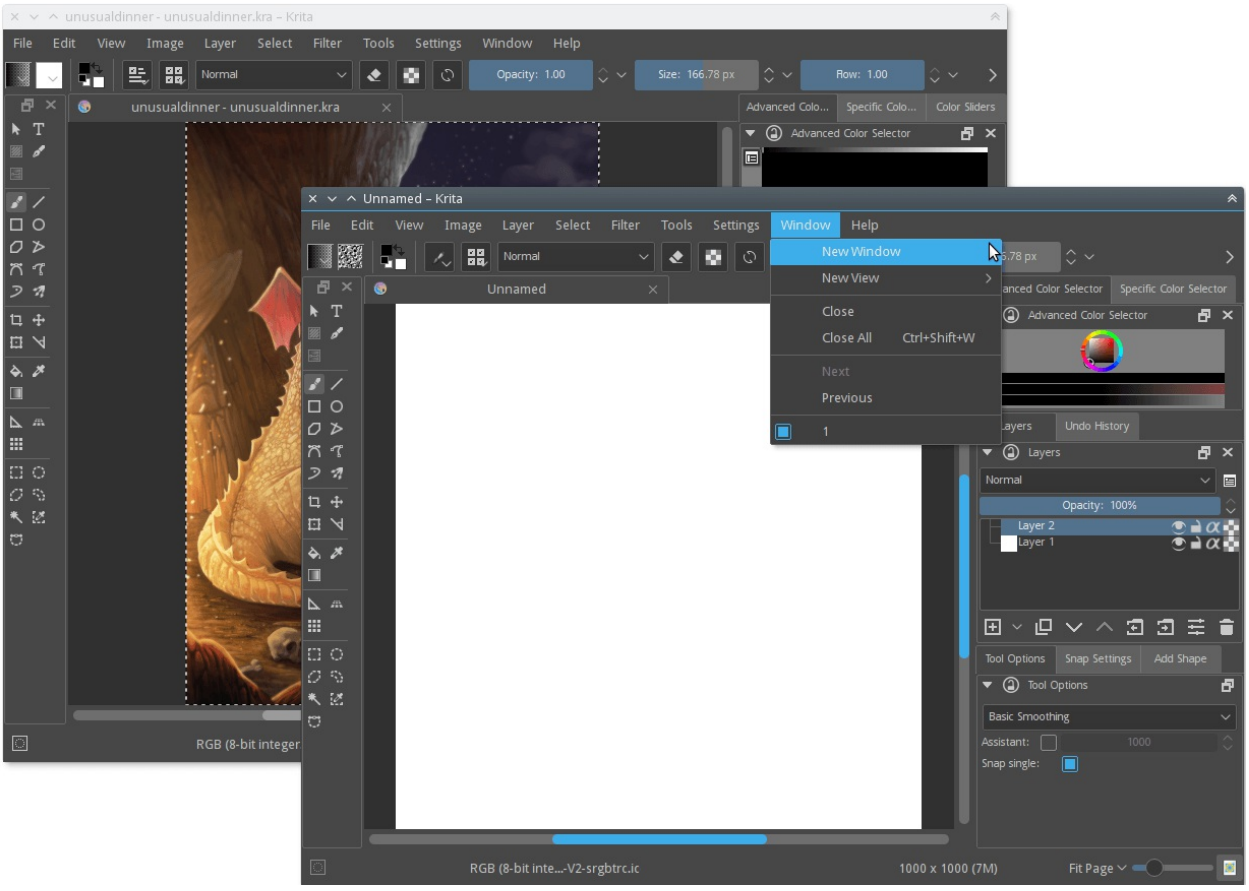
All the views and the dockers are held inside **Windows**.

Window

If you've used a computer before, you know what windows are: They are big containers for your computer programs.

Krita allows you to have multiple windows via *Window* ▸ *New Window*. You can then drag this to another monitor for multi-monitor use.

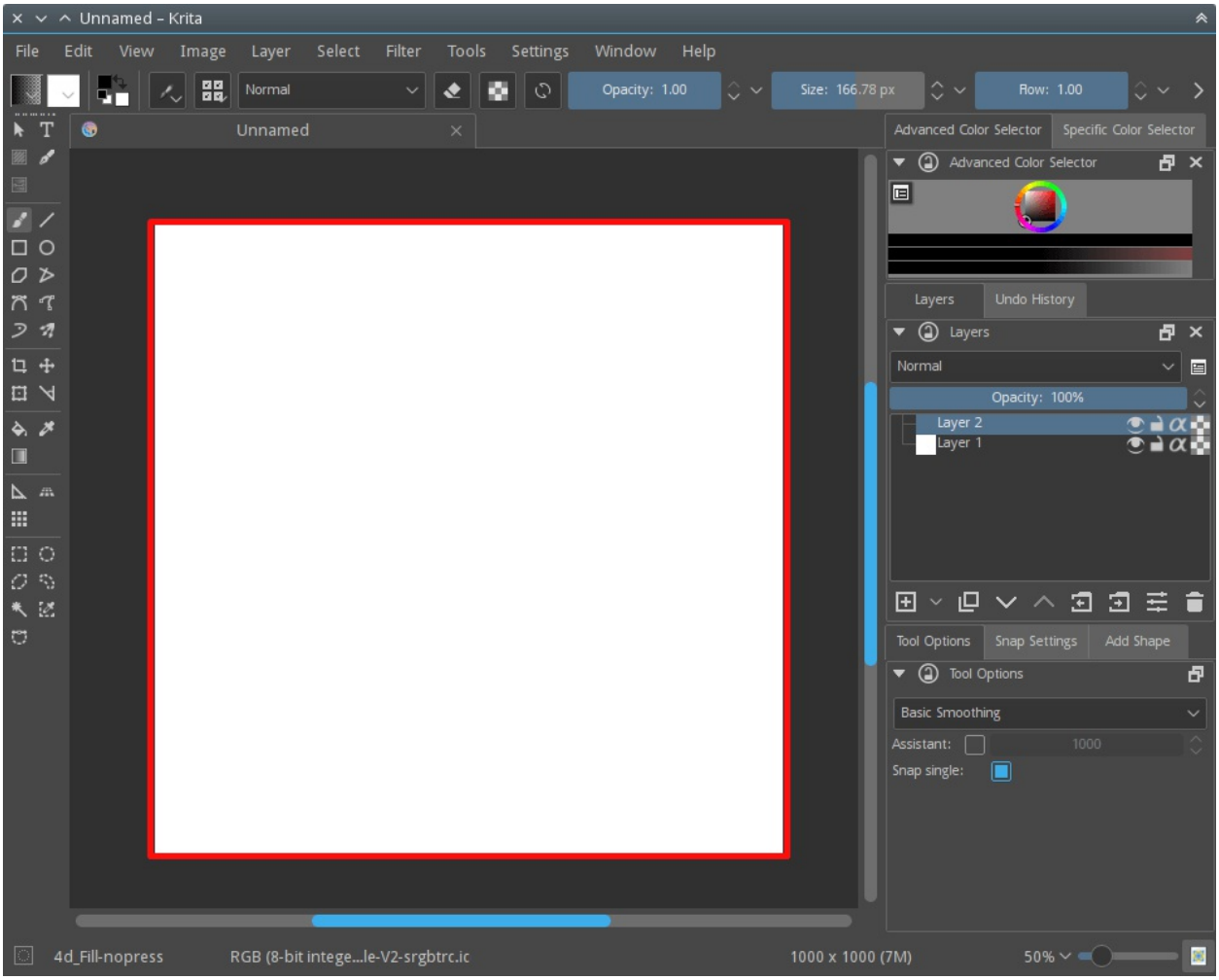
The image below shows an example of multiple windows in Krita.



Canvas in Krita

When you create a new document in Krita for the first time you will see a

rectangular white area. This is called a canvas. You can see it in the image below. The area marked by a red rectangle is a canvas.



When you save the painting as JPG, PNG et cetera or take a print out of the painting, only the content inside this area is taken into consideration. Anything beyond it is ignored. Krita does store information beyond this area, you just won't be able to see it. This data is stored in the **Layers**.

Layers and Compositing

Like a landscape painter will first paint the sky and then the furthest away elements before slowly working his way to the foreground elements, computers will do the same with all the things you tell them to draw. So, if you tell them to draw a circle after a square on the same spot, the circle will

always be drawn later. This is called the **Drawing Order**.

The layer stack is a way for you to separate elements of a drawing and manipulate the drawing order by showing you which layers are drawn when and allowing you to change the order they are drawn in and also apply all sorts of other effects. This is called **Compositing**.

This allows you to have line art above the colors, or trees before the mountains, and edit each without affecting the other.

Krita has many layer-types, each layer type is unique and has its own usecase:

Paint Layers
: These are raster layers, and the most common and default layer type in Krita, you will be painting on these.

Vector Layers
: This is a layer type on which you draw vector graphics. Vector graphics are typically more simple than raster graphics and with the benefit that you can deform them with less blurriness.

Group Layers
: These allow you to group several layers via drag and drop, so you can organize, move, apply masks and perform other actions on them together.

Clone Layers
: These are copies of the layer you selected when making them. They get updated automatically when changing the original.

File Layers
: These refer to an existing image outside of Krita and update as soon as the outside image updates. Useful for logos and emblems that change a lot.

Fill Layers
: These layers are filled with something that Krita can make up on the fly, like colors or patterns.

Filter Layer
: These layers help us to apply some filters which will affect a composite image made from all the layers beneath them.

You can manipulate the content of the layers with **Tools**.

Tools

Tools help you manipulate the image data. The most common one is of course, the freehand brush, which is the default when you open Krita. There are roughly five types of tools in Krita:

Paint Tools
: These are tools for painting on paint layers. They describe shapes, like rectangles, circles and straight lines, but also freehand paths. These shapes then get used by the Brush engines to make shapes and drawing effects.

Vector Tools
: This is the upper row of tools, which are used to edit vectors. Interestingly enough, all paint tools except the freehand brush allow you to draw shapes on the vector layers. The resulting object won't use the brush preset for outline unlike the ones made with paint tools on normal layer.

Selection Tools
: Selections allow you to edit a very specific area of the layer you are working on without affecting the others. The selection tools allow you to draw or modify the current selection. This is like using masking-fluids in traditional painting method, but whereas using masking fluids and film is often messy and delicate, selections are far easier to use.

Guide Tools
: These are tools like grids and assistants.

Transform Tools
: These are tools that allow you to transform your layer or object on the

canvas.

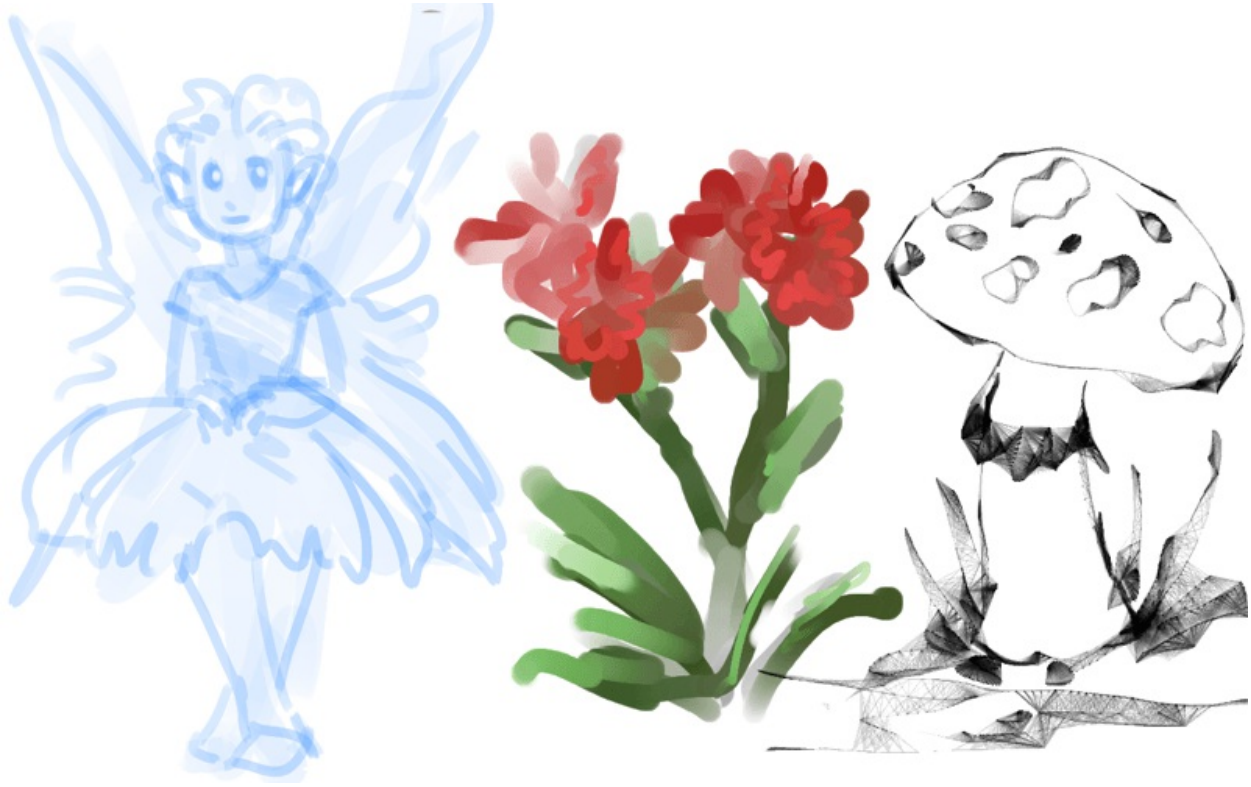
All tools can be found in the toolbox, and information about individual tools can be found in the tools section of the manual.

Brush Engines

Brush engines, as mentioned before, take a path and tablet information and add effects to it, making a stroke.

Engine is a term Krita developers use to describe a complex interacting set of code, that is the core for certain functionality and is highly configurable. In short, like the engine of your car drives your car, and the type of engine and its configuration affects how you use your car, the brush engine drives the look and feel of the brush, and different brush engines have different results.

Krita has a LOT of different brush engines, all with different effects.



Left: pixel brush, **Center:** color smudge brush, **Right:** sketch brush.

For example, the pixel-brush engine is simple and allows you to do most of your basic work, but if you do a lot of painting, the color smudge brush engine might be more useful. Even though it's slower to use than the Pixel Brush engine, its mixing of colors allows you to work faster when you need to blend and mix colors.

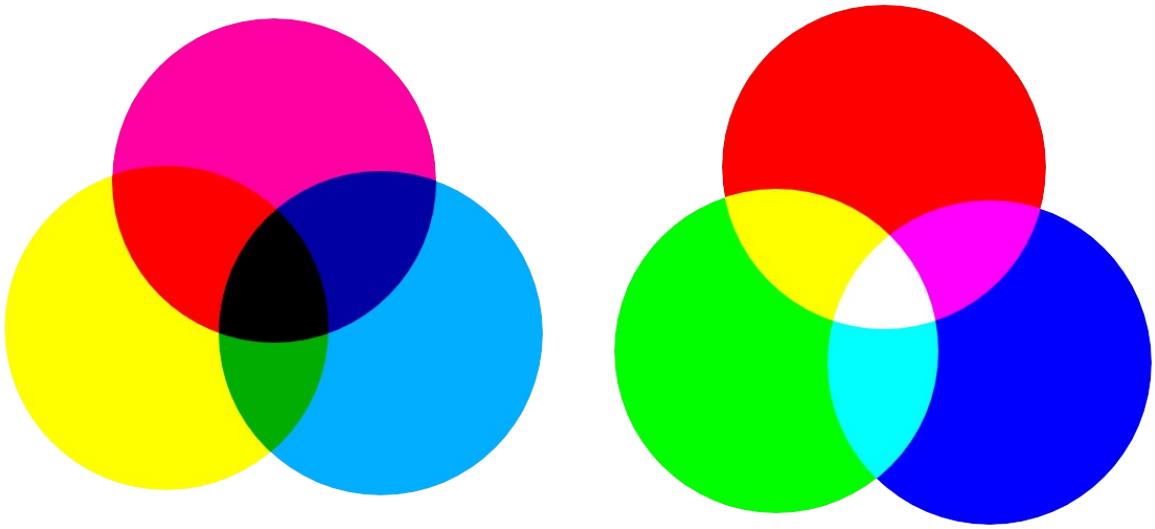
If you want something totally different from that, the sketch brush engine helps with making messy lines, and the shape brush engine allows you to make big flats quickly. There are a lot of cool effects inside Krita's brush engines, so try them all out, and be sure to check the chapters on each.

You can configure these effects via the Brush Settings drop-down, which can be quickly accessed via the `F5` key. These configurations can then be saved into presets, which you can quickly access with the `F6` key or the Brush Presets docker.

Brushes draw with colors, but how do computers understand colors?

Colors

Humans can see a few million colors, which are combinations of electromagnetic waves (light) bouncing off a surface, where the surface absorbs some of it.



Subtractive CMY colors on the left and additive RGB colors on the right. This difference means that printers benefit from color conversion before printing.

When painting traditionally, we use pigments which also absorb the right light-waves for the color we want it to have, but the more pigments you combine, the more light is absorbed, leading to a kind of murky black. This is why we call the mixing of paints **subtractive**, as it subtracts light the more pigments you put together. Because of that, in traditional pigment mixing, our most efficient primaries are three fairly light colors: Cyan blue and Magenta red and Yellow (CMY).

A computer also uses three primaries and uses a specific amount of each primary in a color as the way it stores color. However, a computer is a screen that emits light. So it makes more light, which means it needs to do **additive** mixing, where adding more and more colored lights result in white. This is why the three most efficient primaries, as used by computers are Red, Green and Blue (RGB).

Per pixel, a computer then stores the value of each of these primaries, with the maximum depending on the bit-depth. These are called the **components** or **channels** depending on who you talk to.

This is the red-channel of an image of a red rose. As you can see, the petals are white here, indicating that those areas contain full red. The leaves are much darker, indicating a lack of red, which is to be expected, as they are green.

Though by default computers use RGB, they can also convert to CMYK (the subtractive model), or a perceptual model like LAB. In all cases this is just a different way of indicating how the colors relate to each other, and each time it usually has 3 components. The exception here is grayscale, because the computer only needs to remember how white a color is. This is why grayscale is more efficient memory-wise.

In fact, if you look at each channel separately, they also look like grayscale images, but instead white just means how much Red, Green or Blue there is.

Krita has a very complex color management system, which you can read more about here.

Transparency

Just like Red, Green and Blue, the computer can also store how transparent a pixel is. This is important for **compositing** as mentioned before. After all, there's no point in having multiple layers if you can't have transparency.

Transparency is stored in the same way as colors, meaning that it's also a channel. We usually call this channel the **alpha channel** or **alpha** for short. The reason behind this is that the letter 'α' is used to represent it in programming.

Some older programs don't always have transparency by default. Krita is the opposite: it doesn't understand images that don't track transparency, and will always add a transparency channel to images. When a given pixel is completely transparent on all layers, Krita will instead show a checkerboard pattern, like the rose image shown above.

Blending modes

Because colors are stored as numbers you can do maths with them. We call this **Blending Modes** or **Compositing Modes**.

Blending modes can be done per layer or per brush stroke, and thus are also part of the compositing of layers.

Multiply
: A commonly used blending mode is for example *Multiply* which multiplies the components, leading to darker colors. This allows you to simulate the subtractive mixing, and thus makes painting shadows much easier.

Addition
: Another common one is *Addition*, which adds one layer's components to the other, making it perfect for special glow effects.

Erasing
: *Erasing* is a blending mode in Krita. There is no eraser tool, but you can toggle on the brush quickly with the `E` key to become an eraser. You can also use it on layers. Unlike the other blending modes, this one only affects the alpha channel, making things more transparent.

Normal
: The *Normal* blend mode just averages between colors depending on how transparent the topmost color is.

Krita has 76 blending modes, each doing slightly different things. Head over to the Blending Modes to learn more.

Because we can see channels as grayscale images, we can convert grayscale images into channels. Like for example, we can use a grayscale image for the transparency. We call these **Masks**.

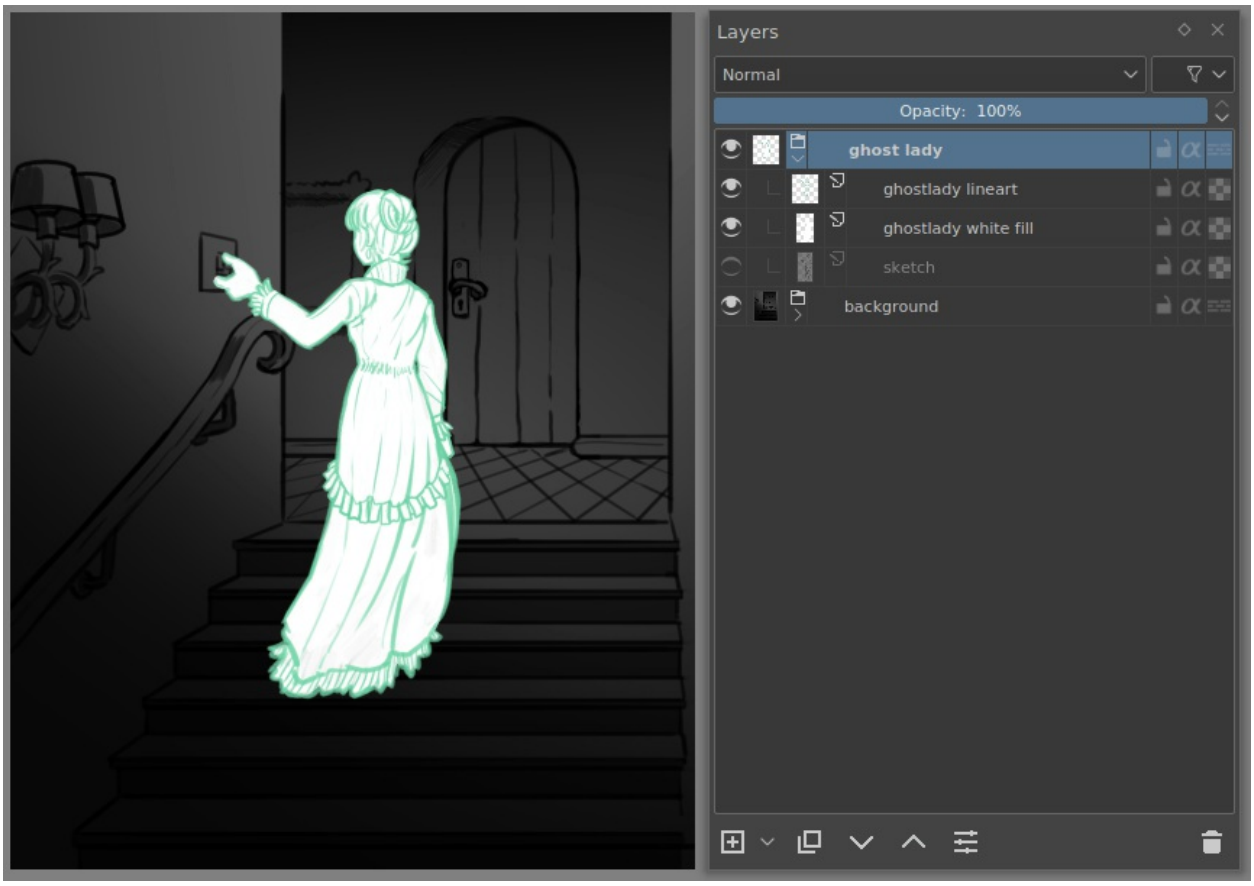
Masks

Masks are a type of sub-effect applied to a layer, usually driven by a grayscale image.

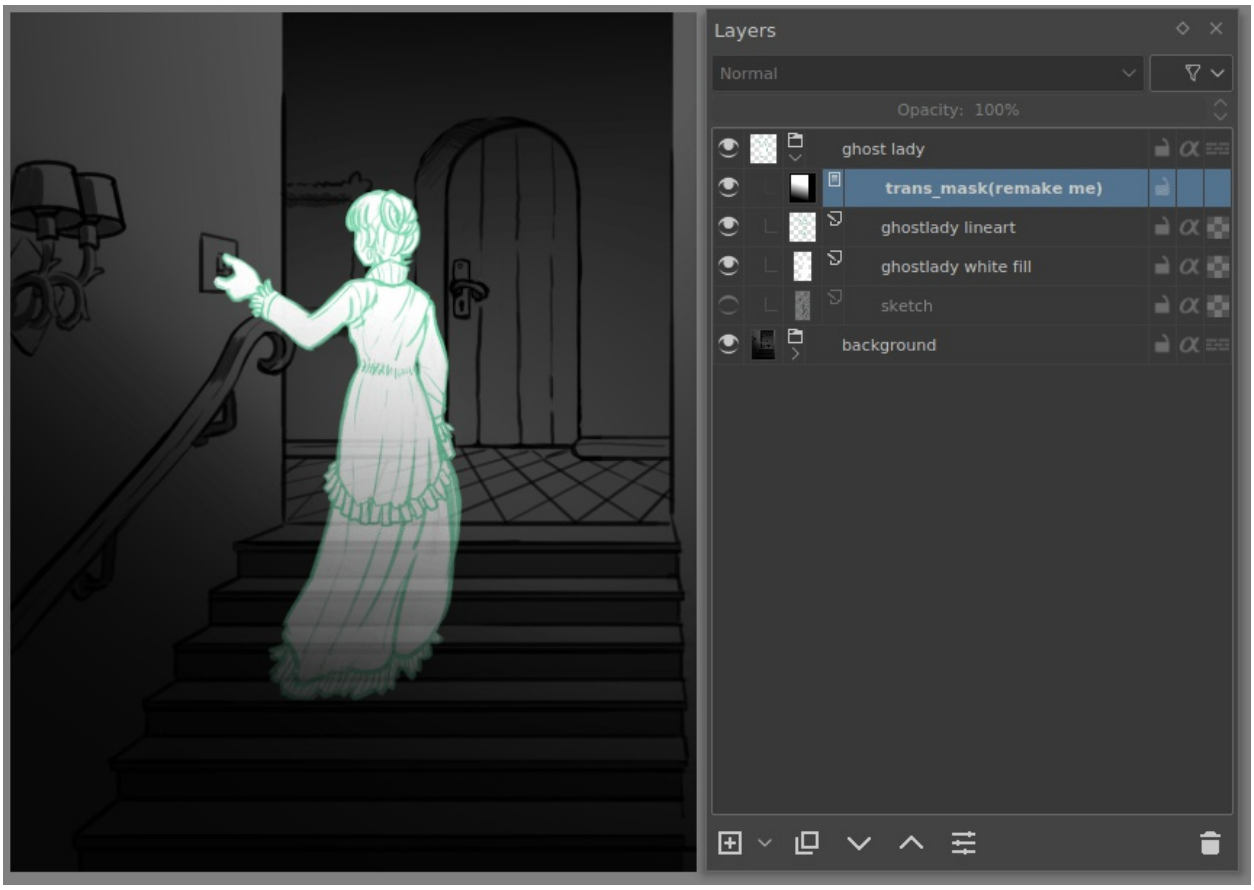
The primary types of mask are Transparency Masks, which allow you to use a grayscale image to determine transparency, where black makes everything transparent and white makes the pixel fully opaque.

You can paint on masks with any of the brushes, or convert a normal paint-layer to a mask. The big benefit of masks is that you can make things transparent without removing the underlying pixels. Furthermore, you can use masks to reveal or hide a whole group layer at once!

For example, we have a white ghost lady here:



But you can’t really tell whether she’s a ghost lady or just really really white. If only we could give the idea that she floats. We right-click the layer and add a transparency mask. Then, we select that mask and draw with a black and white linear gradient so that the black is below.



Wherever the black is, there the lady now becomes transparent, turning her into a real ghost!

The name mask comes from traditional masking fluid and film. You may recall the earlier comparison of selections to traditional masking fluid. Selections too are stored internally as grayscale images, and you can save them as a local selection which is kind of like a mask, or convert them to a transparency mask.

Filters

We mentioned earlier that you can do maths with colors. But you can also do maths with pixels, or groups of pixels or whole layers. In fact, you can make Krita do all sorts of little operations on layers. We call these operations **Filters**.

Examples of such operations are:

Desaturate
: This makes all the pixels turn gray.

Blur
: This averages the pixels with their neighbors, which removes sharp contrasts and makes the whole image look blurry.

Sharpen
: This increases the contrast between pixels that had a pretty high contrast to begin with.

Color to Alpha
: A popular filter which makes all of the chosen color transparent.

Different filter brushes being used on different parts of the image.

Krita has many more filters available: you can read about them here.

Filter Brush Engine

Because many of these operations are per pixel, Krita allows you to use the filter as part of the Filter Brush Engine.

In most image manipulation software, these are separate tools, but Krita has it as a brush engine, allowing much more customization than usual.

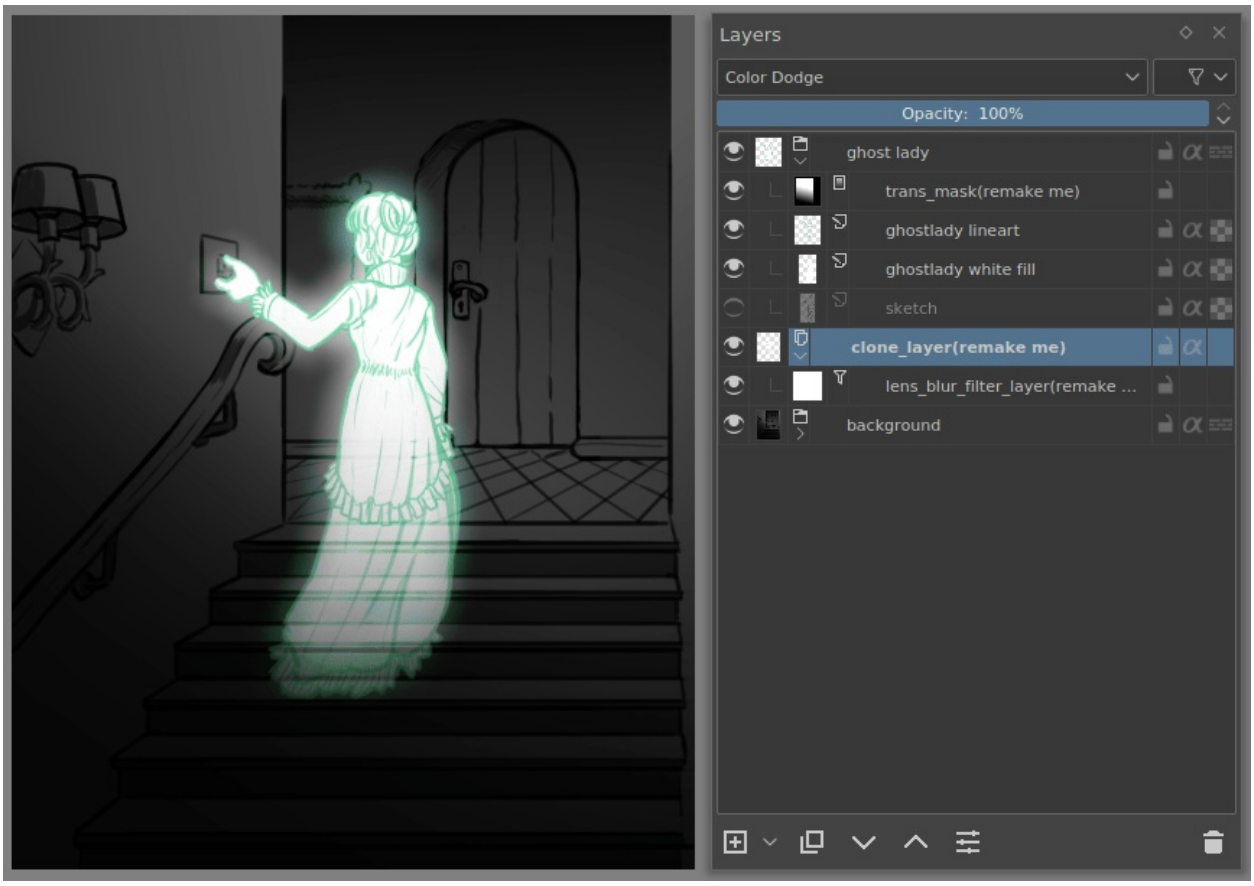
This means you can make a brush that desaturates pixels, or a brush that changes the hue of the pixels underneath.

Filter Layers, Filter Masks and Layer Styles

Krita also allows you to let the Filters be part of the layer stack, via Filter Layer and Filter Masks. Filter Layers affect all the layers underneath it in the same hierarchy. Transparency and transparency masks on Filter Layers affect where the layer is applied.

Masks, on the other hand, can affect one single layer and are driven by a grayscale image. They will also affect all layers in a group, much like a transparency mask.

We can use these filters to make our ghost lady look even more ethereal, by selecting the ghost lady's layer, and then creating a clone layer. We then right click and add a filter mask and use gaussian blur set to 10 or so pixels. The clone layer is then put behind the original layer, and set to the blending mode '**Color Dodge**', giving her a definite spooky glow. You can keep on painting on the original layer and everything will get updated automatically!

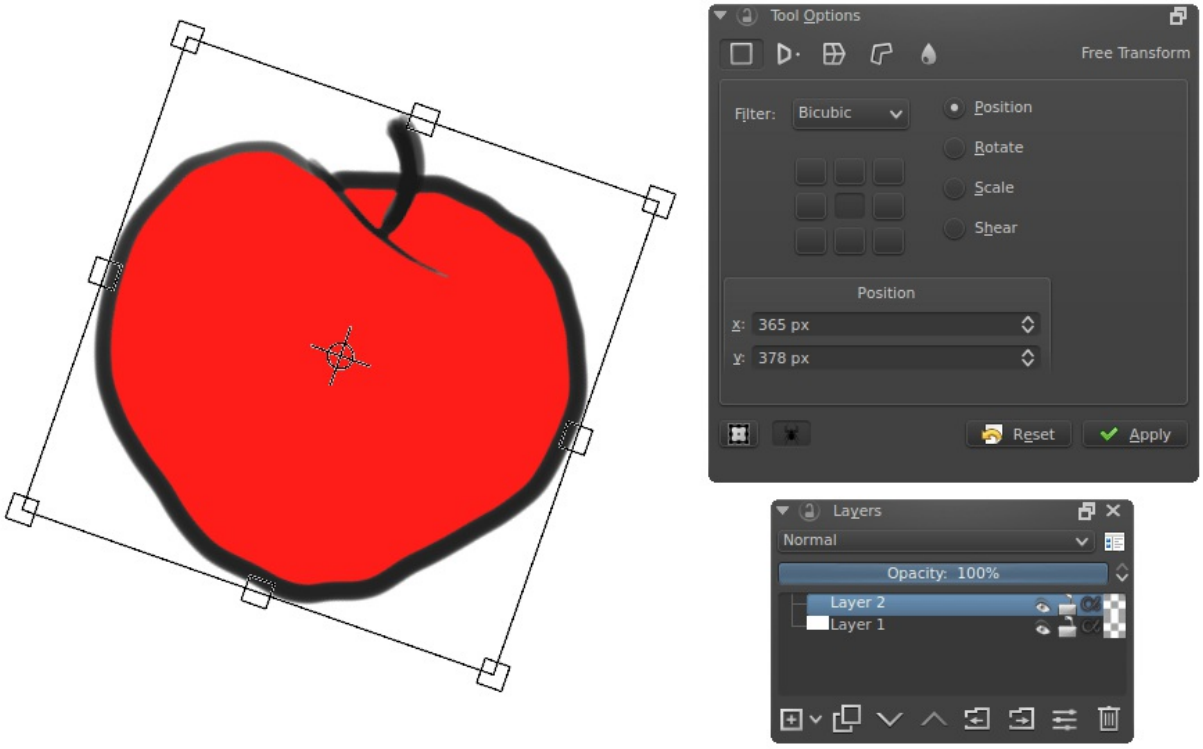


Layer Effects or Layer Styles are filter masks popularised by **Photoshop's** that are a little faster than regular masks, but not as versatile. They are available by right clicking a layer and selecting ‘layer style’.

Transformations

Transformations are kind of like filters, in that these are operations done on the pixels of an image. We have a regular image and layer wide transformations in the image and layer top menus, so that you may resize, flip and rotate the whole image.

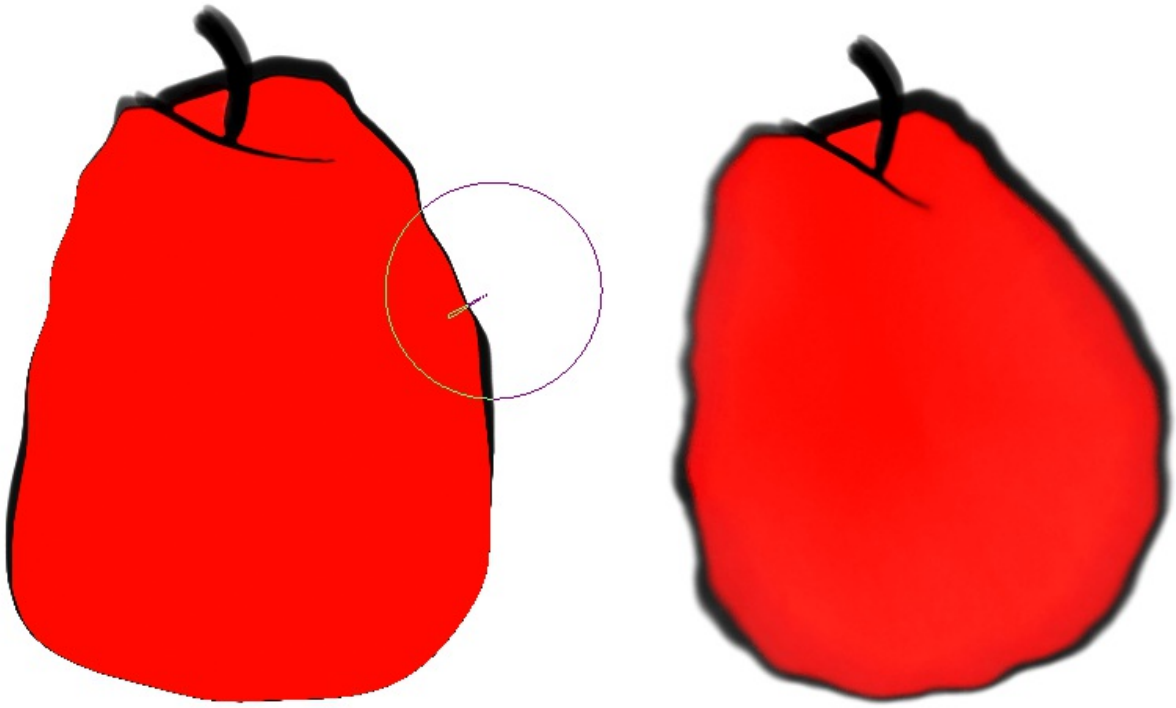
We also have the Crop Tool, which only affects the canvas size, and the Move Tool which only moves a given layer. However, if you want more control, Krita offers a Transform Tool.



With this tool you can rotate and resize on the canvas, or put it in perspective. Or you can use advanced transform tools, like the warp, cage and liquify, which allow you to transform by drawing custom points or even by pretending it’s a transforming brush.

Deform Brush Engine

Like the filter brush engine, Krita also has a Deform Brush Engine, which allows you to transform with a brush. The deform is like a much faster version of the Liquify transform tool mode, but in exchange, its results are of much lower quality.



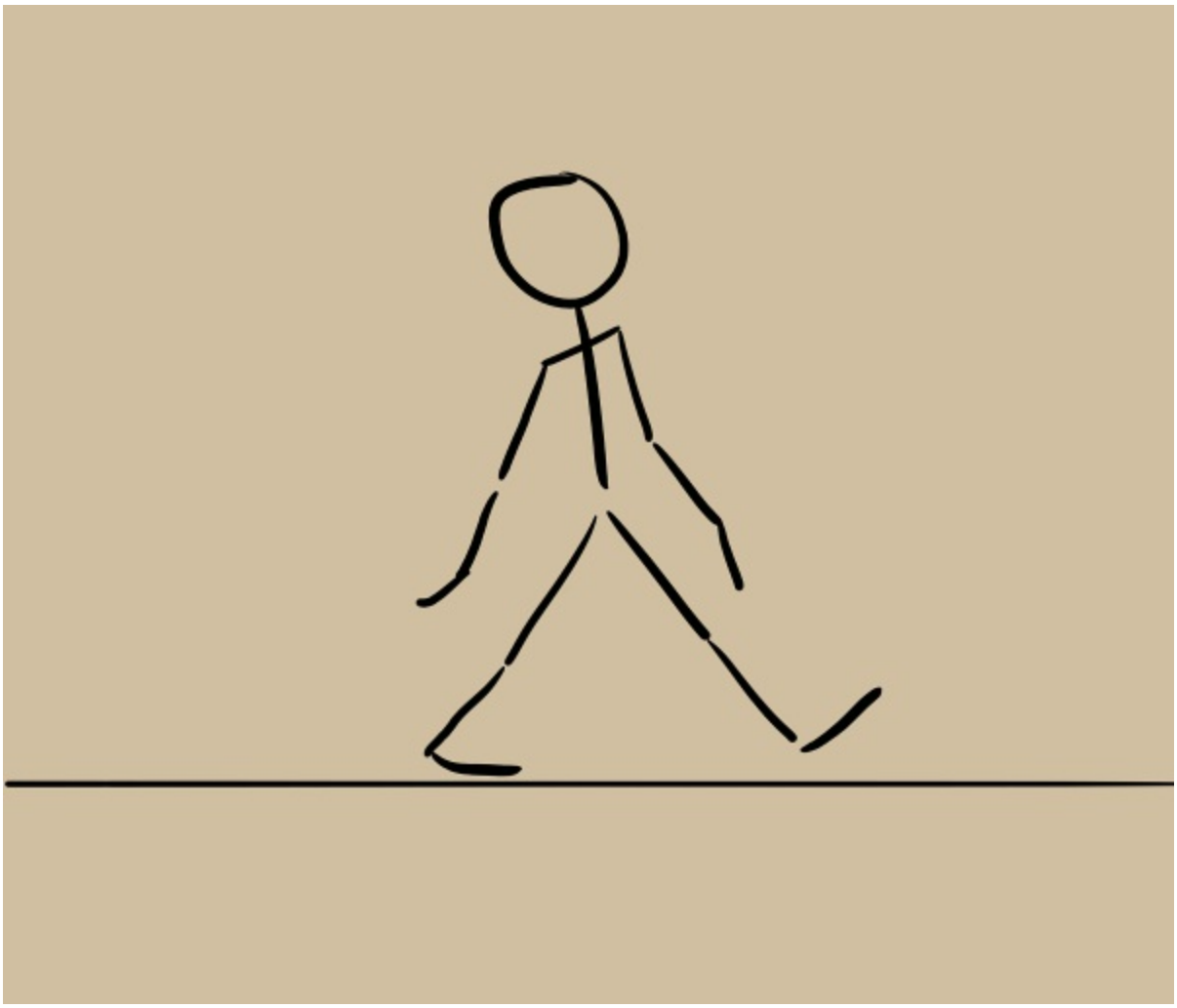
Apple transformed into a pear with liquify on the left and deform brush on the right.

Furthermore, you can’t apply the deform brush as a non-destructive mask.

Transformation Masks

Like filters, transforms can be applied as a non-destructive operation that is part of the layer stack. Unlike filter and transparency masks however, transform masks can’t be driven by a grayscale image, for technical reasons. You can use transform masks to deform clone and file layers as well.

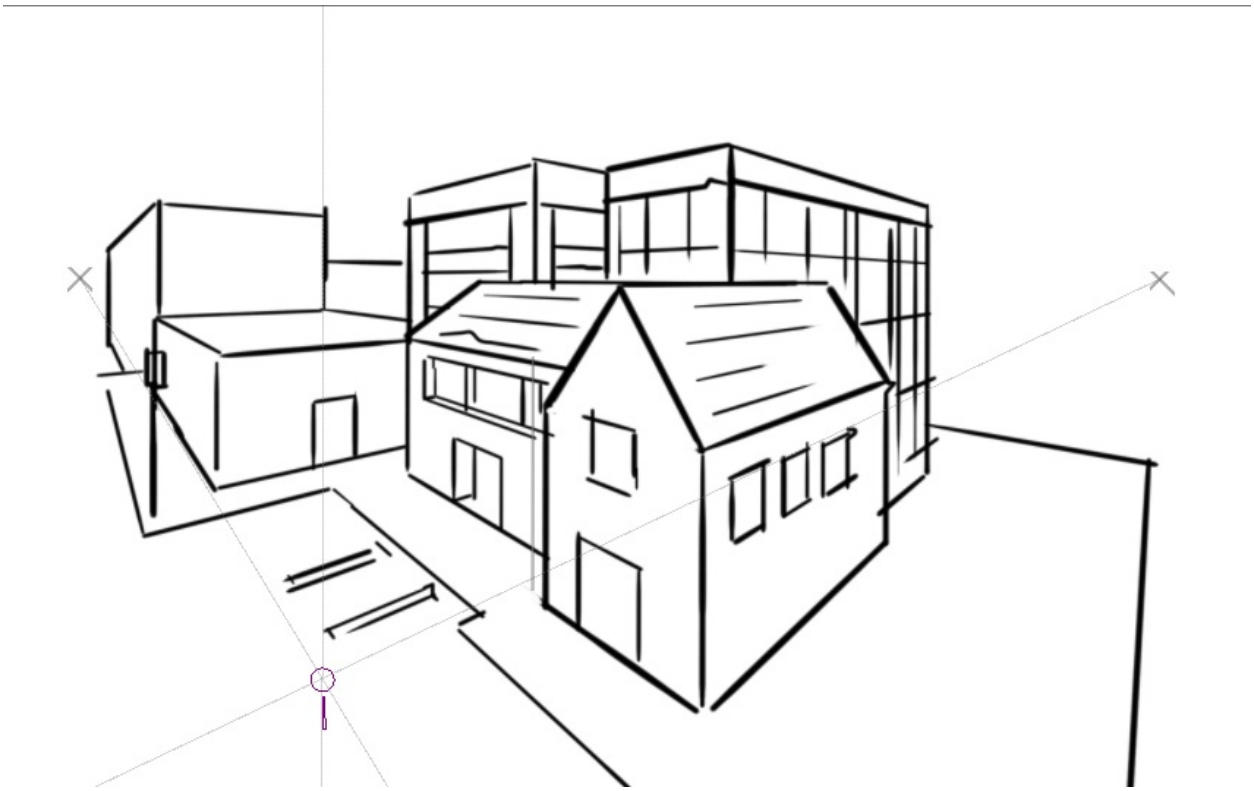
Animation with Krita



From version 3.0 onwards, Krita got raster animation support. You can use the timeline, animation and onionskin dockers, plus Krita's amazing variety of brushes to do raster based animations, export those, and then turn them into movies or GIFs.

Assistants, Grids and Guides

With all this technical stuff, you might forget that Krita is a painting program. Like how when working with traditional medium, as an illustrator, you can have all sorts of equipment to make drawing easier, Krita also offers a variety of tools:



Krita’s vanishing point assistants in action.

Grids and Guides Docker
: A very straightforward guiding tool which shows grids or guiding lines that can be configured.

Snapping
: You can snap to all sorts of things. Grids, guides, extensions, orthogonals, image centers and bounding boxes.

Painting with Assistants
: Because you can hardly put a ruler against your tablet to help you draw, the assistants are there to help you draw concentric circles, perspectives, parallel lines and other easily forgotten but tricky to draw details. Krita allows you to snap to these via the tool options as well.

These guides are saved into Krita’s native format, which means you can pick up your work easily afterward.

Customization

This leads to the final concept: customization.

In addition to rearranging the dockers according to your preferences, Krita provides and saves your configurations as Workspaces. This is the button at the top right.

You can also configure the toolbar via *Settings* ▸ *Configure Toolbars…*, as well as the shortcuts under both *Settings* ▸ *Configure Krita…* ▸ *Shortcuts* and *Settings* ▸ *Configure Krita…* ▸ *Canvas Input Settings*.

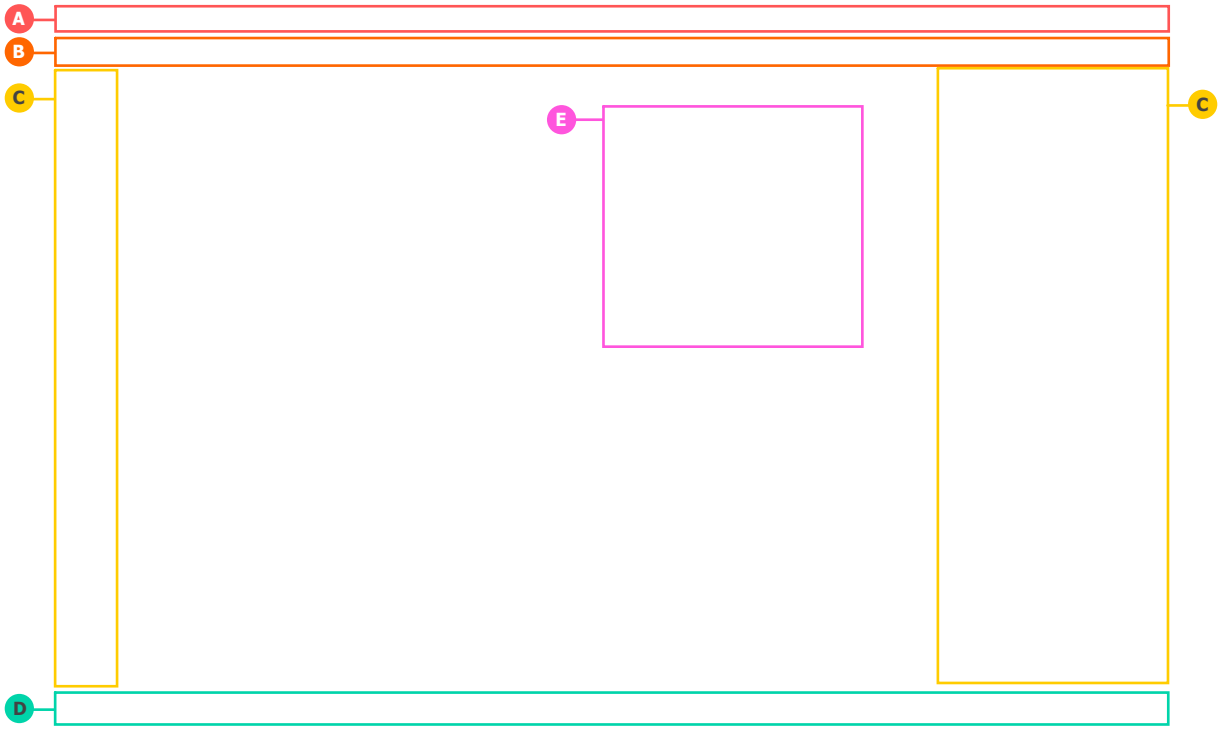
Navigation

Interface

Krita’s interface is very flexible and provides an ample choice for the artists to arrange the elements of the workspace. An artist can snap and arrange the elements, much like snapping together Lego blocks. Krita provides a set of construction kit parts in the form of Dockers and Toolbars. Every set of elements can be shown, hidden, moved and rearranged that let the artists easily customize their own user interface experience.

A Tour of the Krita Interface

As we’ve said before, the Krita interface is very malleable and the way that you choose to configure the work surface may not resemble those shown below, but we can use these as a starting point.



- **A** – Traditional *File* or action menu found in most windowed applications.
- **B** – Toolbar - This is where you can choose your brushes, set parameters such as opacity and size and other settings.
- **C** – Sidebars for the Movable Panels/Dockers. In some applications, these are known as Dockable areas. Krita also allows you to dock panels at the top and/or bottom as well.
- **D** – Status Bar - This space shows the preferred mode for showing selection i.e. marching ants or mask mode, your selected brush preset, Color Space, image size and provides a convenient zoom control.
- **E** – Floating Panel/Docker - These can be "popped" in and out of their docks at any time in order to see a greater range of options. A good example of this would be the Preset Docker or the Palette Docker.

Your canvas sits in the middle and unlike traditional paper or even most digital painting applications, Krita provides the artist with a scrolling canvas of infinite size (not that you'll need it of course!). The standard navigation tools are as follows:

Navigation

Many of the canvas navigation actions, like rotation, mirroring and zooming have default keys attached to them:

Panning

This can be done through  , `Space +`  and `the directional keys`.

Zooming

Discrete zooming can be done through `+` and `-` keys. Using the `Ctrl + Space` or `Ctrl +`  shortcuts can allow for direct zooming with the stylus.

Mirroring

You can mirror the view can be quickly done via `M` key. Mirroring is a great technique that seasoned digital artists use to quickly review the composition of their work to ensure that it "reads" well, even when

flipped horizontally.

Rotating
: You can rotate the canvas without transforming. It can be done with the `Ctrl + [` shortcut or `4`key and the other way with :kbd:`Ctrl + ]` shortcut or `6` key. Quick mouse based rotation is the `Shift + Space` and `Shift +`  shortcuts. To reset rotation use the `5` key.

You can also find these under *View* ▸ *Canvas*.

Dockers

Krita subdivides many of its options into functional panels called Dockers (also known as Docks).

Dockers are small windows that can contain, for example, things like the layer stack, Color Palette or list of Brush Presets. Think of them as the painter's palette, or his water, or his brush kit. They can be activated by choosing the *Settings* menu and the *Dockers* sub-menu. There you will find a long list of available options.

Dockers can be removed by clicking the **x** in the upper-right of the docker-window.

Dockers, as the name implies, can be docked into the main interface. You can do this by dragging the docker to the sides of the canvas (or top or bottom if you prefer).

Dockers contain many of the "hidden", and powerful, aspects of **Krita** that you will want to explore as you start delving deeper into the application.

You can arrange the dockers in almost any permutation and combination according to the needs of your workflow, and then save these arrangements as Workspaces.

Dockers can be prevented from docking by pressing the `Ctrl` key before starting to drag the docker.

Sliders

Krita uses these to control values like brush size, opacity, flow, Hue, Saturation, etc… Below is an example of a Krita slider.

The total range is represented from left to right and blue bar gives an indication of where in the possible range the current value is. Clicking anywhere, left or right, of that slider will change the current number to something lower (to the left) or higher (to the right).

To input a specific number,  the slider. A number can now be entered directly for even greater precision.

Pressing the `Shift` key while dragging the slider changes the values at a smaller increment, and pressing the `Ctrl` key while dragging the slider changes the value in whole numbers or multiples of 5.

Toolbars

Toolbars are where some of the important actions and menus are placed so that they are readily and quickly available for the artist while painting.

You can learn more about the Krita Toolbars and how to configure them in over in the Toolbars section of the manual. Putting these to effective use can really speed up the Artist's workflow, especially for users of Tablet-Monitors and Tablet-PCs.

Workspace Chooser

The button on the very right of the Toolbar is the workspace chooser. This allows you to load and save common configurations of the user interface in Krita. There are a few common workspaces that come with Krita.

Pop-up Palette

Pop-up Palette is a feature unique to Krita, designed to increase the productivity of the artist. It is a circular menu for quickly choosing brushes, foreground and background colors, recent colors while painting. To access the palette you have to just  on the canvas. The palette will spawn at the position of the brush tip or cursor.

By tagging your brush presets you can add particular sets of brushes to this palette. For example, if you add some inking brush presets to inking tag you can change the tags to inking in the pop-up palette and you'll get all the inking brushes in the palette.

You can tag brush presets via the Preset Docker, check out the resource

overview page to know more about tagging in general.

If you call up the pop-up palette again, you can click the tag icon, and select the tag. In fact, you can make multiple tags and switch between them. When you need more than ten presets, go into *Settings* ▸ *Configure Krita…* ▸ *General* ▸ *Miscellaneous* ▸ *Number of Palette Presets* and change the number of presets from 10 to something you feel comfortable.

Introduction Coming From Other Software

Krita is not the only digital painting application in the world. Because we know our users might be approaching Krita with their experience from using other software, we have made guides to illustrate differences.

Contents:

Introduction to Krita coming from Photoshop

Introduction

This document gives an introduction to Krita for users who have been using Photoshop. The intention is to make you productive in Krita as fast as possible and ease the conversion of old habits into new ones. This introduction is written with Krita version 2.9 and Photoshop CS2 and CS3 in mind. But even though things may change in the future, the basics will most likely remain the same. The first thing to remember is that Krita is a 2D paint application while Photoshop (PS) is an image manipulation program. This means that PS has more features than Krita in general, but Krita has the tools that are relevant to digital painting. When you get used to Krita, you will find that Krita has some features that are not part of PS.

Krita Basics

This chapter covers how you use Krita in the basic operations compared to PS.

View and Display

Navigation

In Krita you can navigate your document using all these methods:

1. '*Mouse wheel*':  down and up for zoom, and press  down to pan your document.

2. '*Keyboard*': with the `+` and `-` keys on your numpad keyboard.

3. As in Photoshop, Painter, Manga Studio: use the `Ctrl + Space` shortcut to zoom, and the `Space` key to pan.

    Note

    If you add use the `Alt` key and so do a `Ctrl + Alt + Space` shortcut you'll have a discrete zoom.

Rotation

Rotate the canvas with the `Shift + Space`, or `Ctrl + [` and `Ctrl + ]` shortcuts or with the `4` or `6` keys. Reset the rotation with the `5` key.

Mirror

Press the `M` key to see your drawing or painting mirrored in the viewport.

Move and Transform

Moving and Transformation of contents is done using tools in Krita. You can then find them in the toolbar. If you are familiar with the way to move layers in PS by holding down the `Ctrl` key, you can do the same in Krita by pressing the `T` key for the move tool (think 'T'ranslate) or the `Ctrl + T` shortcut for transform tool.

Press the `B` key to go back to the brush tool when the transformation or translation is done. To find how to make advanced deformations using the *Transform* tool, do not right-click on the on-canvas widget: all the option are in the *Tool Options* docker.

Changes can be applied with the `Enter` key for the Transform tool.

Note

Move tool changes are auto-applied.

Selections

Like in PS, you can use the `Alt` or `Shift` keys during a selection to remove or add selection to the active selection. Krita also offers sub tools for this, and you can select them in the *Tool Options* if a select tool is active. These sub tools are represented as icons. You can switch to those sub modes by pressing:

- `R` to replace selection
- `T` to intersect
- `A` to add to the selection (this is the one you will want to use often)
- `S` to subtract from the selection (the other one popular)

Or hold:

- `Alt` to subtract from the selection
- `Shift` to add to the selection
- `Alt + Shift` to intersect

Note

You cannot press the `Ctrl` key to move the content of the selection (you have to press the `T` key or select the *Move Tool*).

Some other tips:

- If you want to convert a layer to a selection (to select the visible pixels), right-click on the layer docker, and choose *Select Opaque*.
- If you use a polygonal selection tool, or a selection which needs to be 'closed', you will be able to do it or by using a double-click, or by using a `Shift +`  shortcut.

You can scale selection. To do this, choose *Select* ▸ *Scale*.

Note

Also, in the *Select* menu there are more classical options to grow, shrink, feather, border, etc.

If you enable *Show Global Selection Mask* (*Select* menu) you can scale/rotate/transform/move or paint on selection like on regular grayscale layer.

- `Ctrl + H`: Show / Hide selection (same shortcut)
- `Ctrl + A`: Select All
- `Ctrl + Shift + A`: deselect All (and not the `Ctrl + D` shortcut as in PS)

Note for Gimp user: Krita auto-expands and auto defloats new layers created from a selection after pressing the `Ctrl + C`, `Ctrl + V` shortcuts so you do not have to worry about not being able to paint outside the pasted element.

Note

This doesn't work as intended right now. Intersect is a selection mode which use the `T` key as the shortcut. However the `T` key is also used to switch to the *Move tool* so this shortcut is not functional right now. You have to use the button on the *Tool Options*.

Layer Handling

The most common default shortcuts are very similar in PS and Krita:

- `Ctrl + J`: duplicate
- `Ctrl + E`: merge down
- `Ctrl + Shift + E`: flattens all (not the `Ctrl + Shift + M` shortcut as in PS)
- `Ins`: insert a new paint layer
- `Ctrl + G`: create new layer group and move selected layers to this group

Groups and Blending Mode (Composite Mode):

The group blending mode in Krita has priority over child layers and overrides it. This can be surprising for Photoshop users. On Photoshop you can use groups to just clean your layer stack and keep blending mode of your layer compositing through all the stack. In Krita the compositing will happen at first level inside the group, then taking into account the blending mode of the group itself. Both systems have pros and cons. Krita's way is more predictable according to some artists, compositing-wise. The PS way leads to a cleaner and better ordered layer stack visually wise.

Multi Layer Transform or Move

You can select multiple layers on the stack by holding down the `Shift` key as in PS, and if you move the layer inside a group you can move or transform the whole group - including doing selection on the group and cut all the sub layers inside on the fly. You can not apply filters to group to affect multiple layers.

Clipping Masks

Krita has no clipping mask, but there is a simpler workaround involving layer groups and *Inherit alpha* (see the alpha icon). Place a layer with the shape you want to clip the other with at the bottom of a group and layers above with the *Inherit alpha* option. This will create the same effect as the "clipping mask" PS feature, and also keeps the layer stack cleaner than the clipping mask implementation does.

This process of arranging groups for inherit alpha can be done automatically by `Ctrl + Shift + G` shortcut. It creates a group with base layer and a layer above it with inherit alpha option checked by default.

Pass-through mode

This is available in Krita, but not implemented as a blending mode. Rather, it is an option next to 'inherit alpha' on group layers.

Smart Layers

Instead of having smart layers that you can do non-destructive transforms on, Krita has the following set of functionality:

File Layers
: These are layers which point to an outside file, and will get automatically updated if the outside file changes. Starting from version 4.0 users can convert an existing layer into a file layer by  clicking on it and doing *Convert ▸ to File Layer* or by going to *Layer ▸ Convert ▸ to File Layer*. It will then open a save prompt for the file location and when done will save the file and replace the layer with a file layer pointing at that file.

Clone Layers
: These are layers that are an ‘instance’ of the layer you had selected when creating them. They get updated automatically when the original layer updates.

Transform Masks
: These can be used to non-destructive transform all layer types, including the file and clone layers.

Filter Masks
: Like adjustment layers, these can apply filters non-destructively to all layer types, including file and clone layers.

Layer styles

You can apply Photoshop layerstyles in Krita by right clicking any given layer type and selecting ‘layer style’ from the context menu. Krita can open and save ASL files, but not all layer style functionality is there yet.

Other

Layers and groups can be exported. See the *Layer* top menu for this and many other options.

Note

Krita has at least 5 times more blending modes than PS. They are sorted by categories in the drop-down menu. You can use the checkbox to add your most used to the Favorite categories.

Paint tools

This is Krita's strong point. There are many paint tools and they have a lot of options.

Tools

In Krita, there is a totally different paradigm for defining what 'tools' are compared to PS. Unlike in PS, you will not find the brush, eraser, clone, blur tool, etc. Instead, you will find a *way to trace* your strokes on the canvas: freehand, line, rectangle, circle, multiple brush, etc. When you have selected the 'way to trace' you can choose the *way to paint*: erasing / cloning / blurring, etc are all part of *way it paint* managed by the brush-engines options. These brush engine options are saved into so-called *presets*, which you can find on *Brush presets*. You can fine tune, and build your own presets using the *Edit Brush Settings* icon on the top tool bar.

Erasing

In Krita, the eraser is not its own tool; it is a Blending mode (or Composite mode). You can toggle between erase mode and paint mode by pressing the `E` key, individually for each of your brushes.

Useful shortcuts

- `Shift`: Grow or Shrink the brush size (or the `[` and `]` keys).
- `/`: Switch last preset selected and current (ex: a pencil preset, and an eraser preset).
- `K` and `L`: Increment Darker and Lighter value of the active color.

- I and O: Increment opacity plus or minus.
- D: Reset color to black/foreground and white/background.
- X: Switch background and foreground colors.
- Shift + I / Shift + N / Shift + M: A set of default keyboard shortcuts for accessing the on-canvas color selector.

Note

Some people regard these shortcuts as somewhat unfortunate. The reason is that they are meant to be used during painting and the left Shift key is at the opposite end of the keyboard from the I, M and N keys. So for a right-handed painter, this is very difficult to do while using the stylus with a right hand. Note that you can reassign any shortcut by using the shortcut configuration in *Settings* ▸ *Configure Krita…* ▸ *Shortcuts*.

Stabilization / Path Smoothing

Using the freehand 'paint with brush' tool that you can find on the Tool Options, more settings for smoothing the path and stabilization of your brush strokes are available.

Global pressure curve

If you find the feeling of Krita too hard or too soft regarding the pressure when you paint, you can set a softer or harder curve here: *Settings* ▸ *Configure Krita…* ▸ *Tablet settings*

Adjustment

Like in PS, you can use the classic filters to adjust many things while painting:

- Ctrl + L: Levels
- Ctrl + U: HSV adjustment
- Ctrl + I: Invert

Dodge / Burn / Blur Tools

Unlike Photoshop, where these are separate tools, in Krita, they are available via the Filter Brush Engine, which allows you to apply the majority of Krita's filters in brush form.

Themes

If you don't like the dark default theme of Krita go to: *Settings* ▸ *Themes*, and choose a brighter or darker theme. If you don't like the color outside your viewport go to: *Settings* ▸ *Configure Krita…* ▸ *Display*, and change the Canvas border color.

What Krita Has Over Photoshop

As mentioned in the introduction, Krita is a specialized paint application. Thus, it has specialized tools for painting. Similar tools are not found in more generalized image manipulation applications such as PS. Here is a short list of the most important ones.

Brush Engines

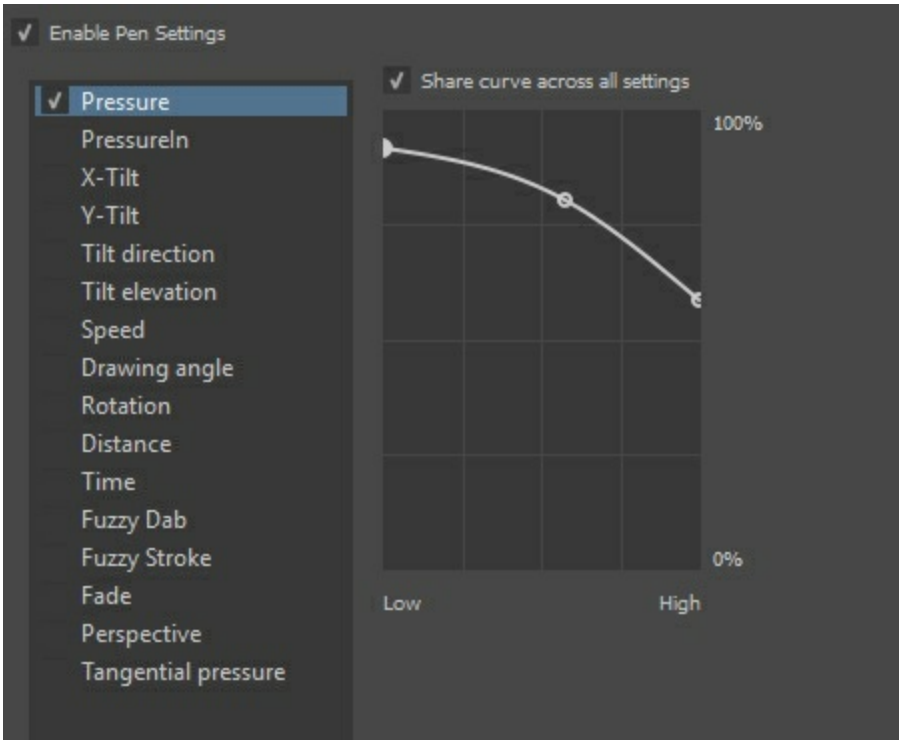
Krita has a lot of different so-called brush engines. These brush engines define various methods on how the pixels end up on your canvas. Brush engines with names like Grid, Particles, Sketch and others will bring you new experiences on how the brushes work and a new landscape of possible results. You can start customizing brushes by using the brush-settings editor, which is accessible via the toolbar, but it's much easier to just press the `F5` key.

Tags for brush presets

This is a very useful way to configure brush presets. Each brush can have any amount of tags and be in any group. You can make tag for blending brushes, for texture brushes, for effect brushes, favorites etc.

Settings curves

You can set setting to pressure (speed/distance/tilt/random/etc.) relation for each brush setting.

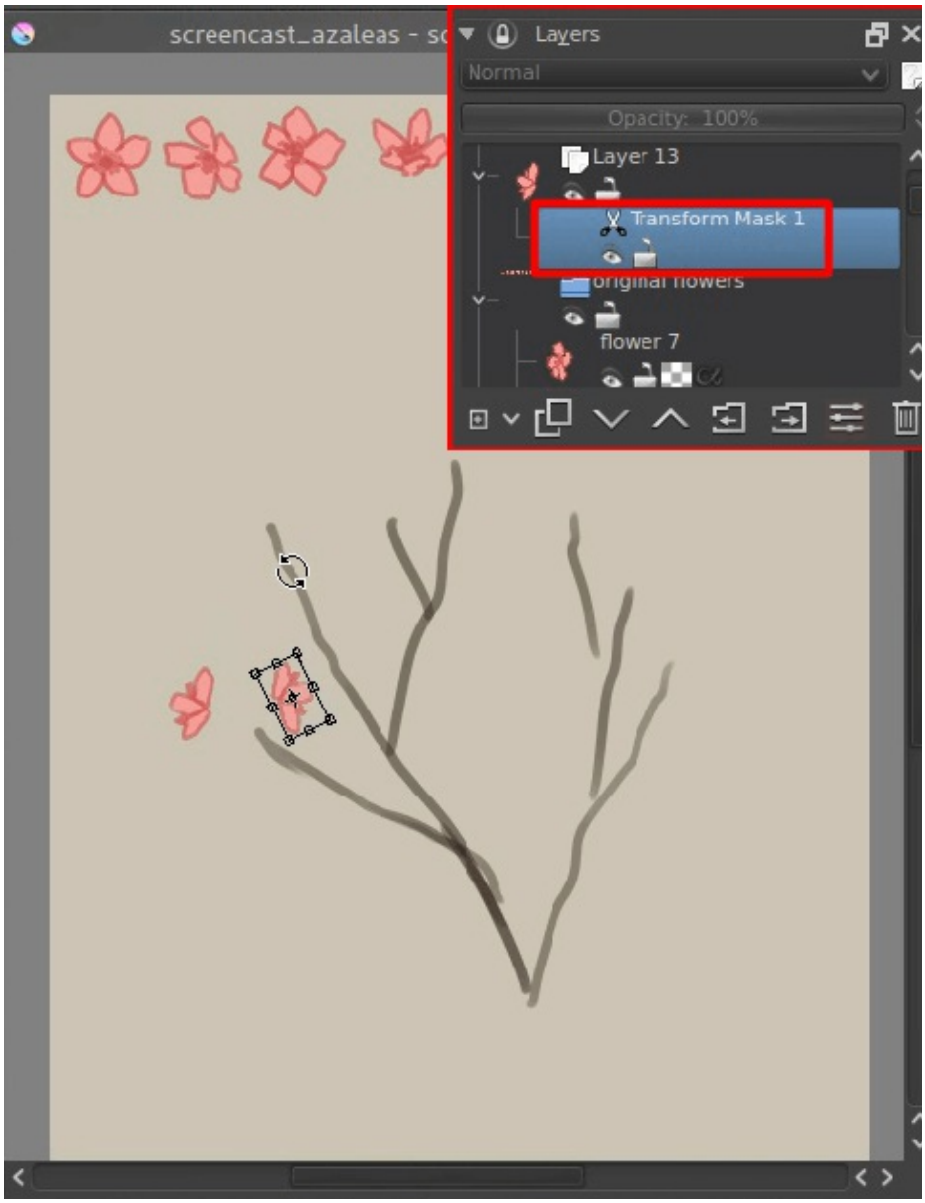


The Pop-up Palette

Easily to be found on , the pop-up palette allows you to quickly access brushes, color history and a color selector within arm’s reach. The brushes are determined by tag, and pressing the lower-right configure button calls a drop-down to change tags. This allows you to tag brushes in the preset docker by workflow, and quickly access the right brushes for the workflow you need for your image.

Transformations

The Krita transformation tool can perform transformations on a group and affect child layers. There are several modes, like free, perspective, warp, the powerful cage and even liquify. Furthermore, you can use transformation masks to apply transforms non-destructively to any layer type, raster, vector group, you name it.



Transform masks allows non-destructive transforms

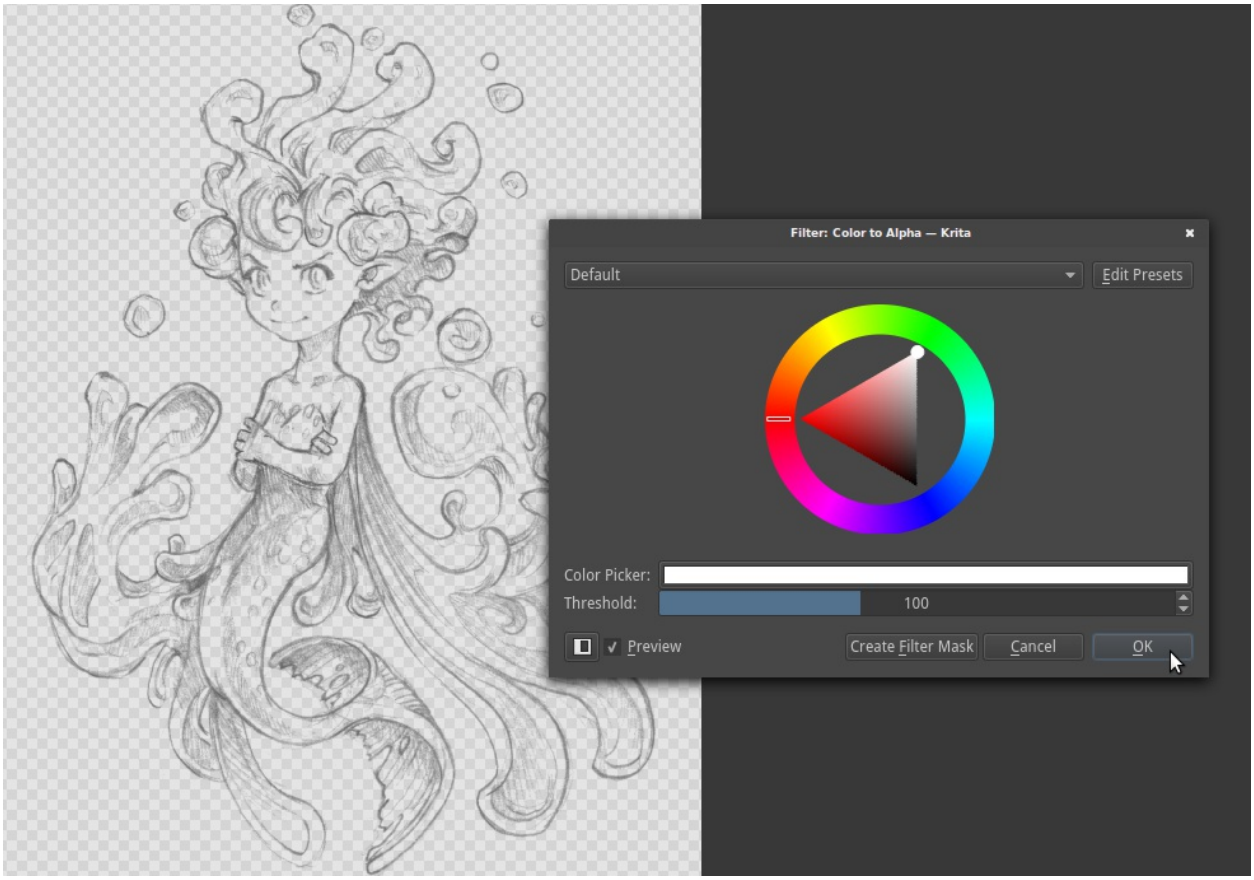
Incremental Save

You can save your artwork with the pattern : *myartworksname_001.kra*, *myartworksname_002.kra*, *myartworksname_003.kra* etc., by pressing a single key on the keyboard. Krita will increment the final number if the pattern "_XXX" is recognized at the end of the file's name.

This feature allows you to avoid overwriting your files, and keep track to your older version and work in progress steps.

Filter: Color to alpha

If you want to delete the white of the paper from a scanned artwork, you can use this filter. It takes a color and turns it into pure transparency.



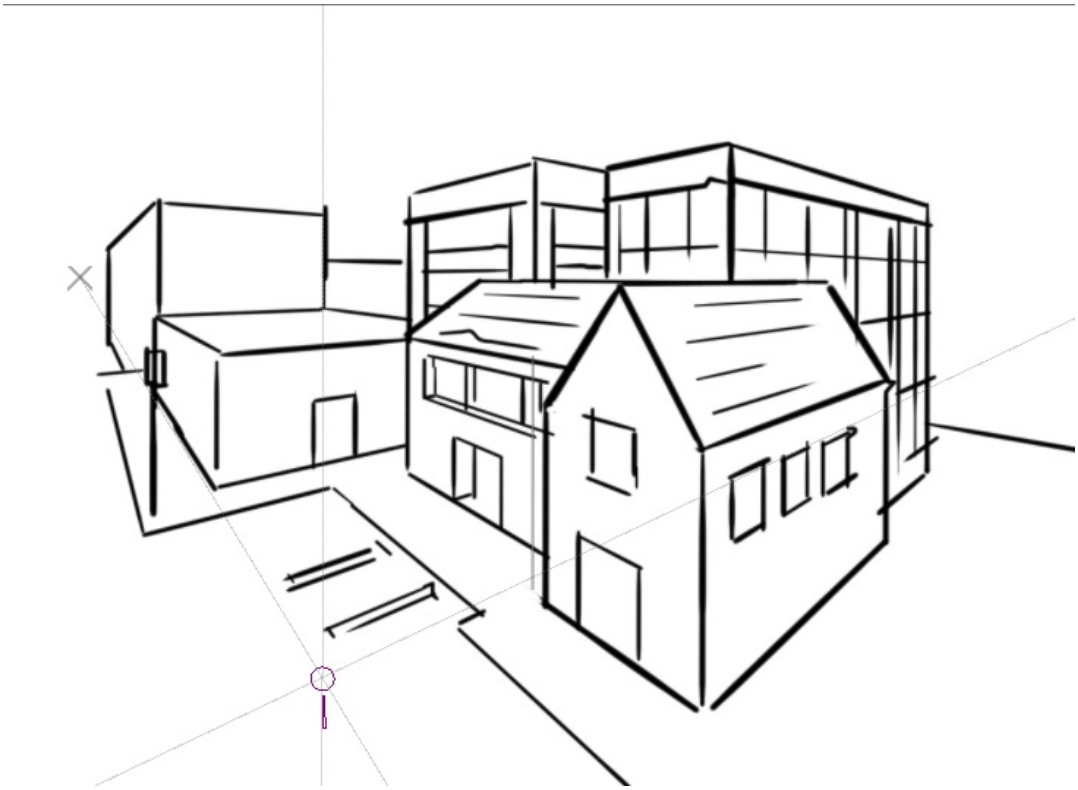
Many Blending Modes

If you like using blending modes, Krita has many of them – over 70! You

have plenty of room for experimentation. A special system of favorite blending modes has been created to let you have fast access to the ones you use the most.

Painting Assistants

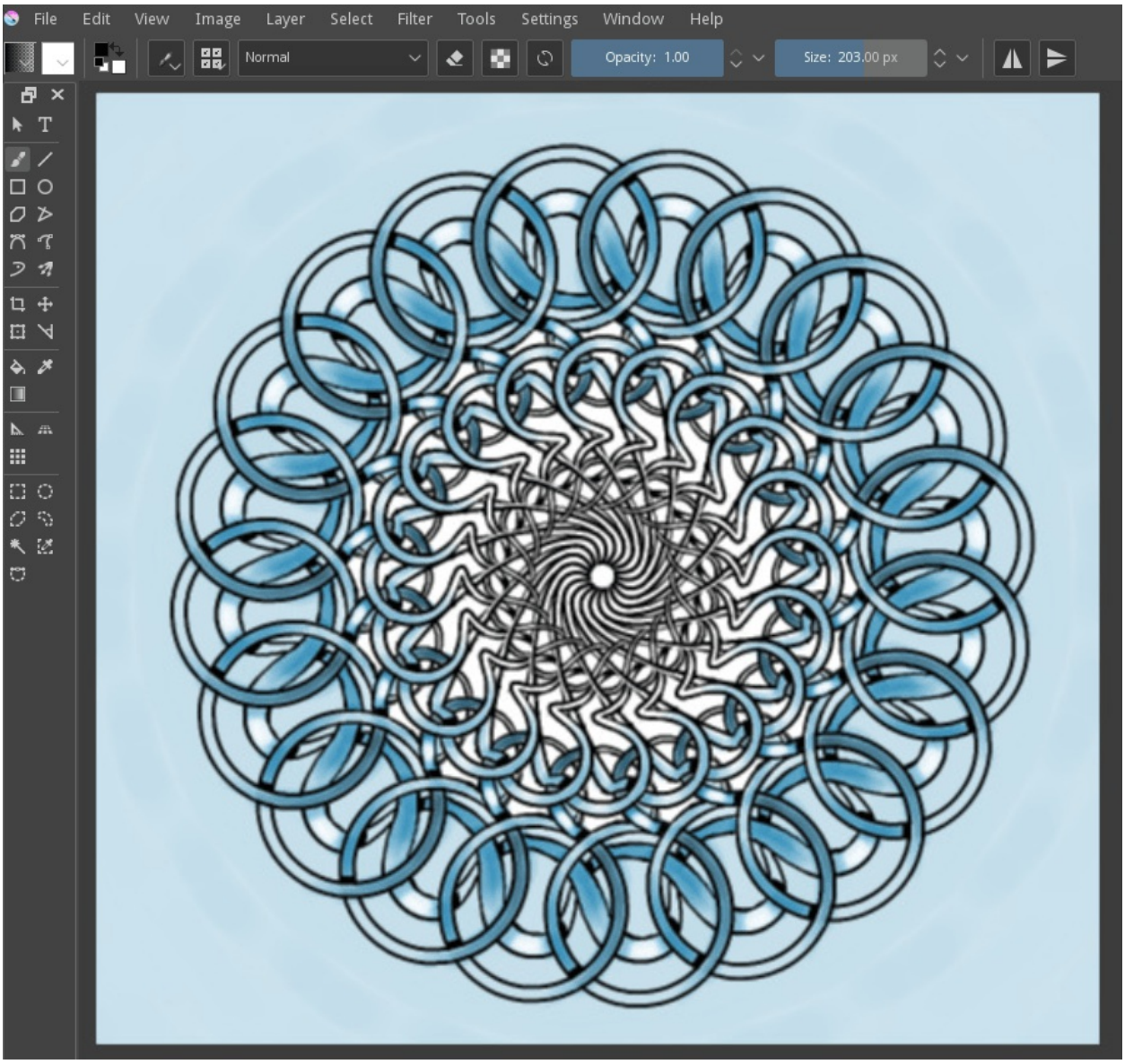
Krita has many painting assistants. This is a special type vector shapes with a magnetic influence on your brush strokes. You can use them as rulers, including with shapes other than just straight.



Krita’s vanishing point assistants in action

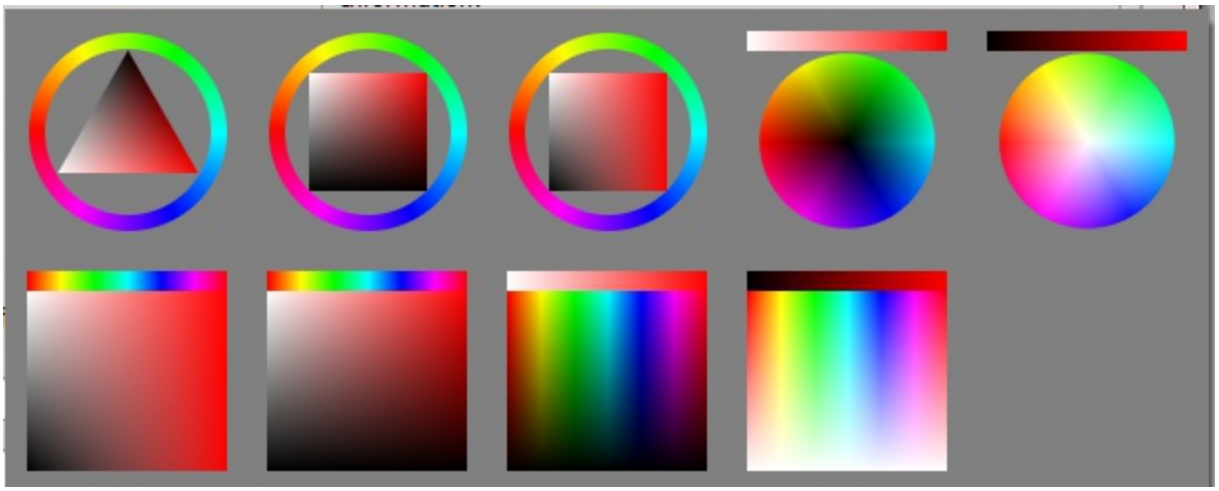
Multibrushes: Symmetry / Parallel / Mirrored / Snowflake

Krita’s Multibrush tool allows you to paint with multiple brushes at the same time. Movements of the brushes other than the main brush is created by mirroring what you paint, or by duplicating it by any number around any axis. They can also be used in parallel mode.



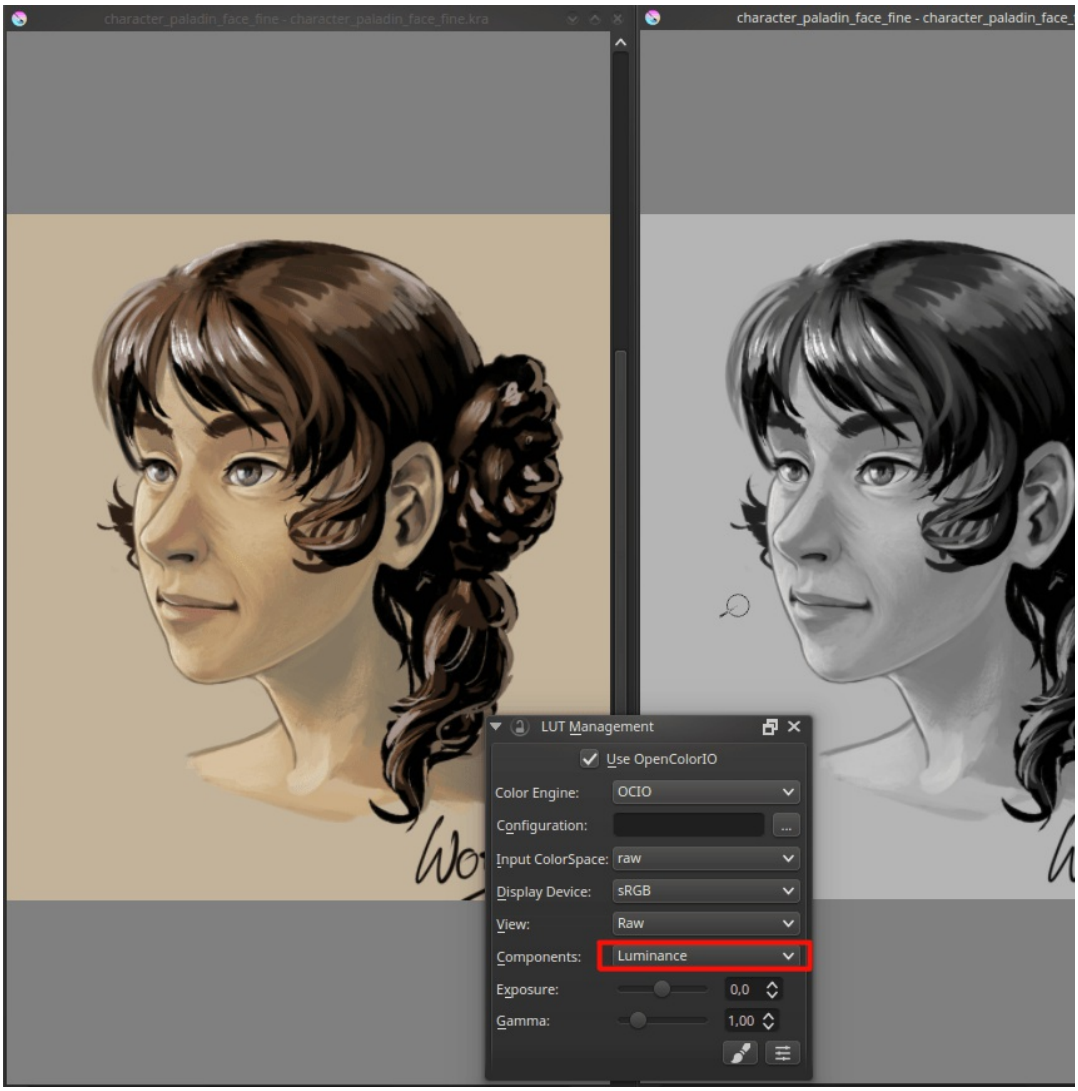
A Wide Variety of Color Selectors

The *Advanced Color Selector* docker offer you a wide choice of color selectors.



View dependent color filters

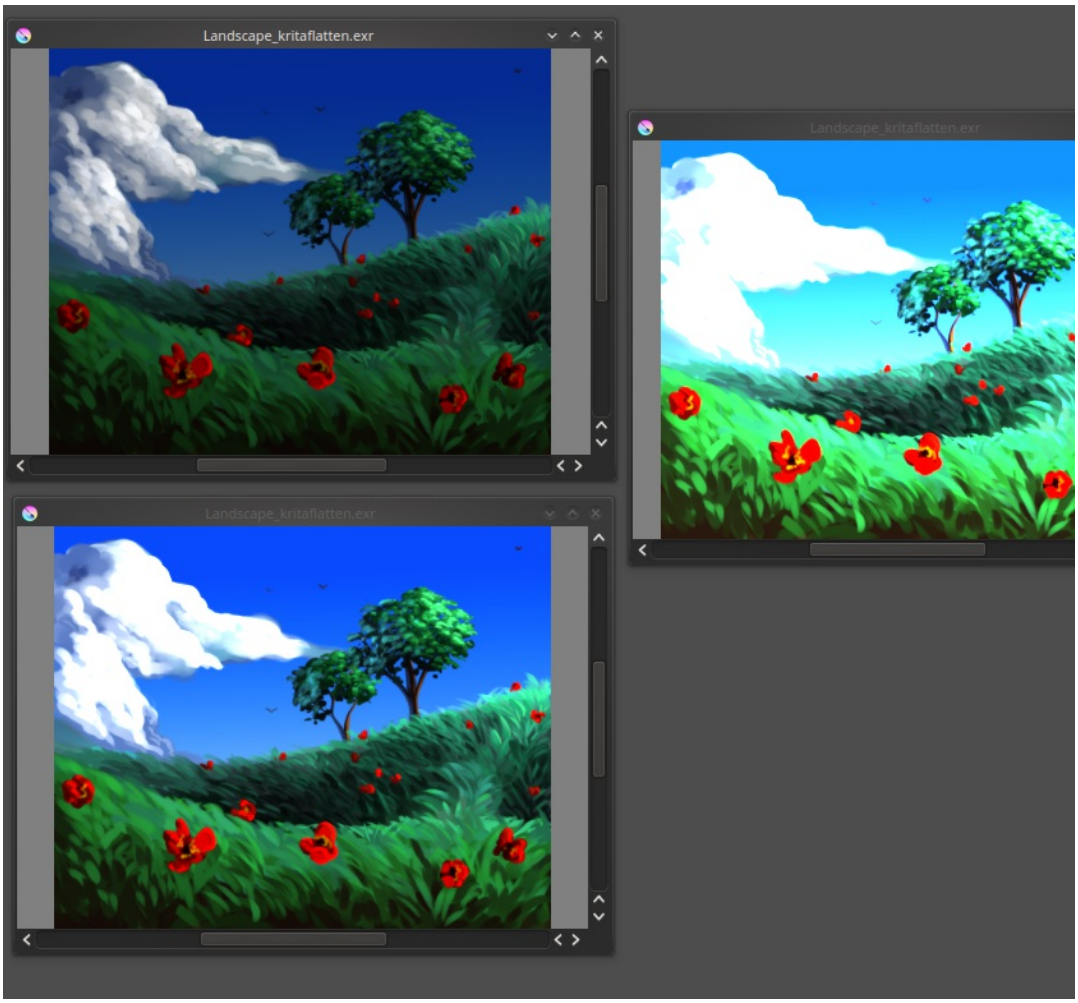
Using the LUT docker, Krita allows you to have a separate color correction filter per view. While this is certainly useful to people who do color correction in daily life, to the artist this allows for seeing a copy of the image in luminance grayscale, so that they instantly know the values of the image.



Using the LUT docker to change the colors per view

HDR color painting

This same LUT docker is the controller for painting with HDR colors. Using the LUT docker to change the exposure on the view, Krita allows you to paint with HDR colors, and has native OpenEXR support!



Painting with HDR colors

What Krita Does Not Have

Again, Krita is a digital paint application and Photoshop is an image manipulation program with some painting features. This means that there are things you can do in PS that you cannot do in Krita. This section gives a short list of these features.

Filters

Krita has a pretty impressive pack of filters available, but you will probably miss one or two of the special filters or color adjustment tools you often use in Photoshop. For example, there is no possibility to tweak a specific color in

HSV adjustment.

Automatic healing tool

Krita does not have an automatic healing tool. It does, however, have a so-called clone tool which can be used to do a healing correction, although not automatically.

Macro Recording

Macro recording and playback exists in Krita, but it is not working well at this time.

Text Tool

The text tool in Krita is less advanced than the similar tool in Photoshop.

Blending Modes While Transforming

When you transform a layer or a selection in Krita, the transformation appears on the top of your layer stack ignoring the layer blending mode.

Photomerge

You may have used this tool in Photoshop to seamlessly and automatically stitch together a drawing that was scanned in segments. Krita does not have an equivalent, though an alternative is to use Hugin, which is cross-platform and free, just like Krita.

Hugin Website [http://hugin.sourceforge.net]

Tutorial for Using Scans in Hugin [https://www.davidrevoy.com/article314/autostiching-scan-with-hugin]

Other

Also, you cannot ‘Export for web’, ‘Image Ready’ for GIF frame or slicing

web image, etc.

Conclusion

Using these tips you will probably be up to speed with Krita in a short time. If you find other things worth mentioning in this document we, the authors, would be interested in hearing about them. Krita develops fast, so we believe that the list of things possible in Photoshop but not in Krita will become shorter in time. We will maintain this document as this happens.

Introduction to Krita coming from Paint Tool Sai

How do you do that in Krita?

This section goes over the functionalities that Krita and Paint Tool Sai share, but shows how they slightly differ.

Canvas navigation

Krita, just like Sai, allows you to flip, rotate and duplicate the view. Unlike Sai, these are tied to keyboard keys.

Mirror
: This is tied to `M` key to flip.

Rotate
: There's a couple of possibilities here: either the `4` and `6` keys, or the `Ctrl + [` and `Ctrl + ]` shortcuts for basic 15 degrees rotation left and right. But you can also have more sophisticated rotation with the `Shift + Space + drag` or `Shift +`  `+ drag` shortcuts. To reset the rotation, press the `5` key.

Zoom
: You can use the `+` and `-` keys to zoom out and in, or use the `Ctrl +`  shortcut. Use the `1`, `2` or `3` keys to reset the zoom, fit the zoom to page or fit the zoom to page width.

You can use the Overview docker in *Settings* ▸ *Dockers* to quickly navigate over your image.

You can also put these commands on the toolbar, so it'll feel a little like Sai.

Go to *Settings* ▸ *Configure Toolbars…* menu item. There are two toolbars, but we'll add to the Main Toolbar.

Then, you can type in something in the left column to search for it. So, for example, 'undo'. Then select the action 'undo freehand stroke' and drag it to the right. Select the action to the right, and click *Change text*. There, toggle *Hide text when toolbar shows action alongside icon* to prevent the action from showing the text. Then press *OK*. When done right, the *Undo* should now be sandwiched between the save and the gradient icon.

You can do the same for *Redo*, *Deselect*, *Invert Selection*, *Zoom out*, *Zoom in*, *Reset zoom*, *Rotate left*, *Rotate right*, *Mirror view* and perhaps *Smoothing: basic* and *Smoothing: stabilizer* to get nearly all the functionality of Sai's top bar in Krita's top bar. (Though, on smaller screens this will cause all the things in the Brushes and Stuff Toolbar to hide inside a drop-down to the right, so you need to experiment a little).

Hide Selection, *Reset Rotation* are currently not available via the Toolbar configuration, you'll need to use the shortcuts `Ctrl + H` and `5` to toggle these.

Note

Krita 3.0 currently doesn't allow changing the text in the toolbar, we're working on it.

Right click color picker

You can actually set this in *Settings* ▸ *Configure Krita…* ▸ *Canvas input settings* ▸ *Alternate invocation*. Just double-click the entry that says `Ctrl +`  shortcut before *Pick Foreground Color from Merged Image* to get a window to set it to  .

Note

Krita 3.0 actually has a Paint-tool Sai-compatible input sheet shipped by

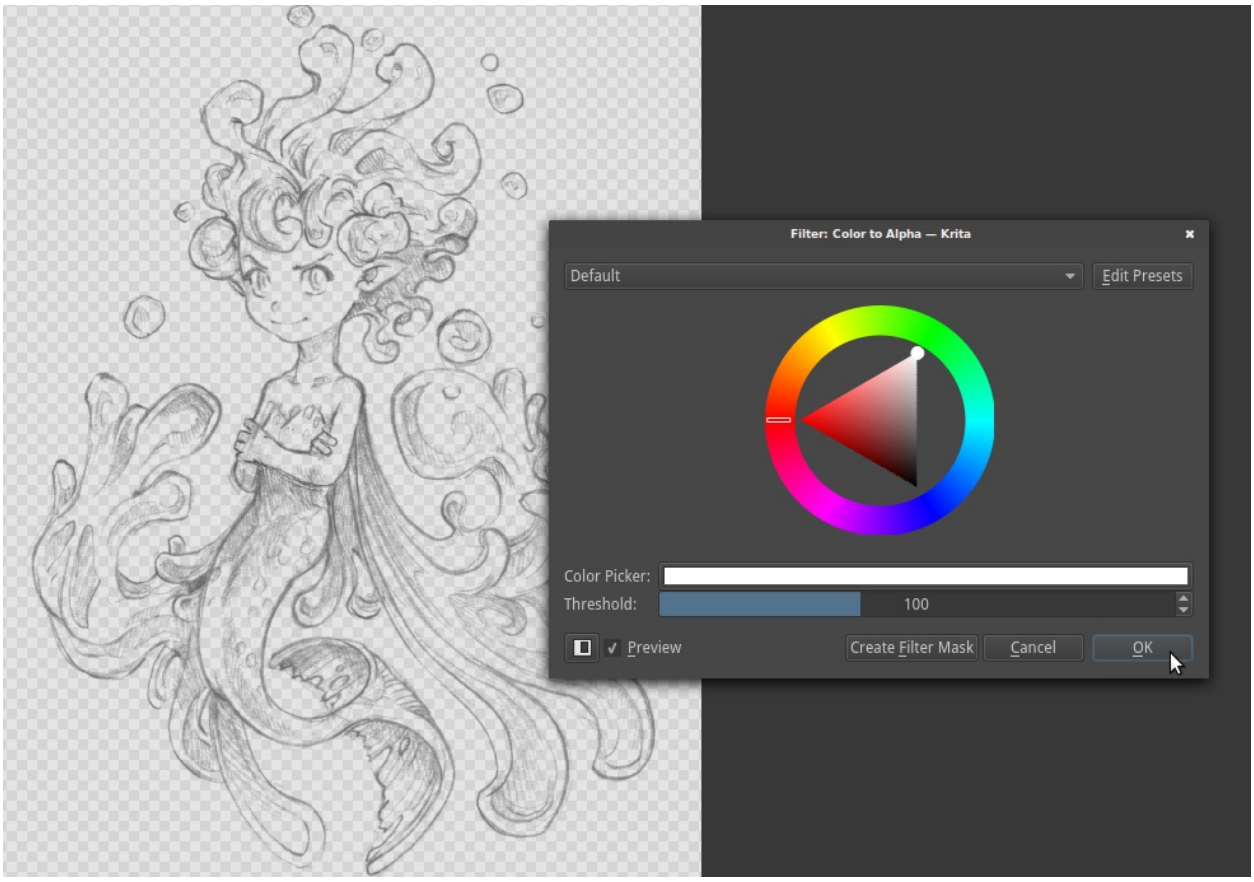
default. Combine these with the shortcut sheet for Paint tool Sai to get most of the functionality on familiar hotkeys.

Stabilizer

This is in the tool options docker of the freehand brush. Use Basic Smoothing for more advanced tablets, and Stabilizer is much like Paint Tool Sai's. Just turn off *Delay* so that the dead-zone disappears.

Transparency

So one of the things that throw a lot of Paint Tool Sai users off is that Krita uses checkers to display transparency, which is actually not that uncommon. Still, if you want to have the canvas background to be white, this is possible. Just choose *Background: As Canvas Color* in the new image dialogue and the image background will be white. You can turn it back to transparent via *Image* ▸ *Image Background Color and Transparency…* menu item. If you export a PNG or JPG, make sure to uncheck *Store alpha channel (transparency)* and to make the background color white (it's black by default).



Like Sai, you can quickly turn a black and white image to black and transparent with the *Filter: Color to Alpha* dialog under *Filters* ▸ *Colors* ▸ *Color to Alpha…* menu item.

Brush Settings

Another, somewhat amusing misconception is that Krita's brush engine is not very complex. After all, you can only change the Size, Flow and Opacity from the top bar.

This is not quite true. It's rather that we don't have our brush settings in a docker but a drop-down on the toolbar. The easiest way to access this is with the `F5` key. As you can see, it's actually quite complex. We have more than a dozen brush engines, which are a type of brush you can make. The ones you are used to from Paint Tool Sai are the Pixel Brush (ink), The Color Smudge Brush (brush) and the filter brush (dodge, burn).

A simple inking brush recipe for example is to take a pixel brush, uncheck the *Enable Pen Settings* on opacity and flow, and uncheck everything but size from the option list. Then, go into brush-tip, pick Auto Brush from the tabs, and set the size to 25 (right-click a blue bar if you want to input numbers), turn on anti-aliasing under the brush icon, and set fade to 0.9. Then, as a final touch, set spacing to 'auto' and the spacing number to 0.8.

You can configure the brushes in a lot of detail, and share the packs with others. Importing of packs and brushes can be done via the *Settings* ▸ *Manage Resources…*, where you can import `.bundle` or `.kpp` files.

Erasing

Erasing is a blending mode in Krita, much like the transparency mode of Paint Tool Sai. It's activated with the `E` key or you can select it from the *Blending Mode* drop-down box.

Blending Modes

Krita has a lot of Blending modes, and thankfully all of Paint Tool Sai's are amongst them except binary. To manage the blending modes, each of them has a little check-box that you can tick to add them to the favorites.

Multiple, Screen, Overlay and Normal are amongst the favorites. Krita's Luminosity is actually slightly different from Paint Tool Sai's and it replaces the relative brightness of color with the relative brightness of the color of the layer.

Sai's Luminosity mode (called Shine in Sai2) is the same as Krita's *Luminosity/Shine (SAI)* mode, which is new in Krita 4.2.4. The Sai's Shade mode is the same as *Color Burn* and *Hard Mix* is the same as the Luminosity and Shade modes.

Layers

Lock Alpha
: This is the checker box icon next to every layer.

Clipping group
: For Clipping masks in Krita you'll need to put all your images in a single layer, and then press the 'a' icon, or press the `Ctrl + Shift + G` shortcut.

Ink layer
: This is a vector layer in Krita, and also holds the text.

Masks
: These grayscale layers that allow you to affect the transparency are called transparency masks in Krita, and like Paint Tool Sai, they can be applied to groups as well as layers. If you have a selection and make a transparency mask, it will use the selection as a base.

Clearing a layer
: This is under *Edit* ▸ *Clear*, but you can also just press the `Del` key.

Mixing between two colors

If you liked this docker in Paint Tool Sai, Krita's Digital Color Selector docker will be able to help you. Dragging the sliders will change how much of a color is mixed in.

What do you get extra when using Krita?

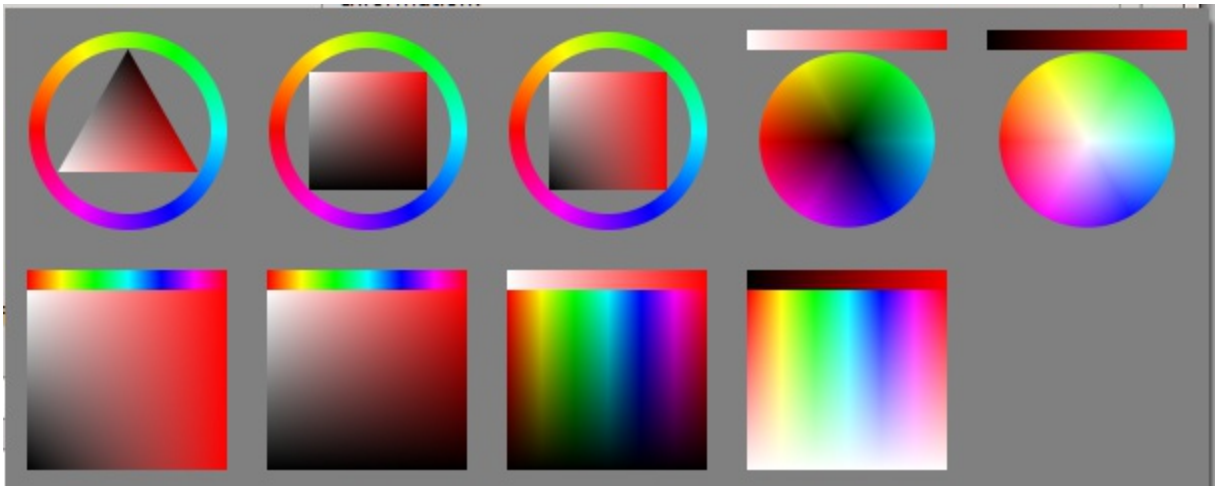
More brush customization

You already met the brush settings editor. Sketch brushes, grid brushes, deform brushes, clone brushes, brushes that are textures, brushes that respond to tilt, rotation, speed, brushes that draw hatches and brushes that deform the colors. Krita's variety is quite big.

More color selectors

You can have HSV sliders, RGB sliders, triangle in a hue ring. But you can also have HSI, HSL or HSY' sliders, CMYK sliders, palettes, round selectors, square selectors, tiny selectors, big selectors, color history and

shade selectors. Just go into *Settings* ▸ *Configure Krita…* ▸ *Color Selector Settings* ▸ *Color Selector tab*, select an option in the *Docker:* drop-down box, to change the shape and type of your main color selector.



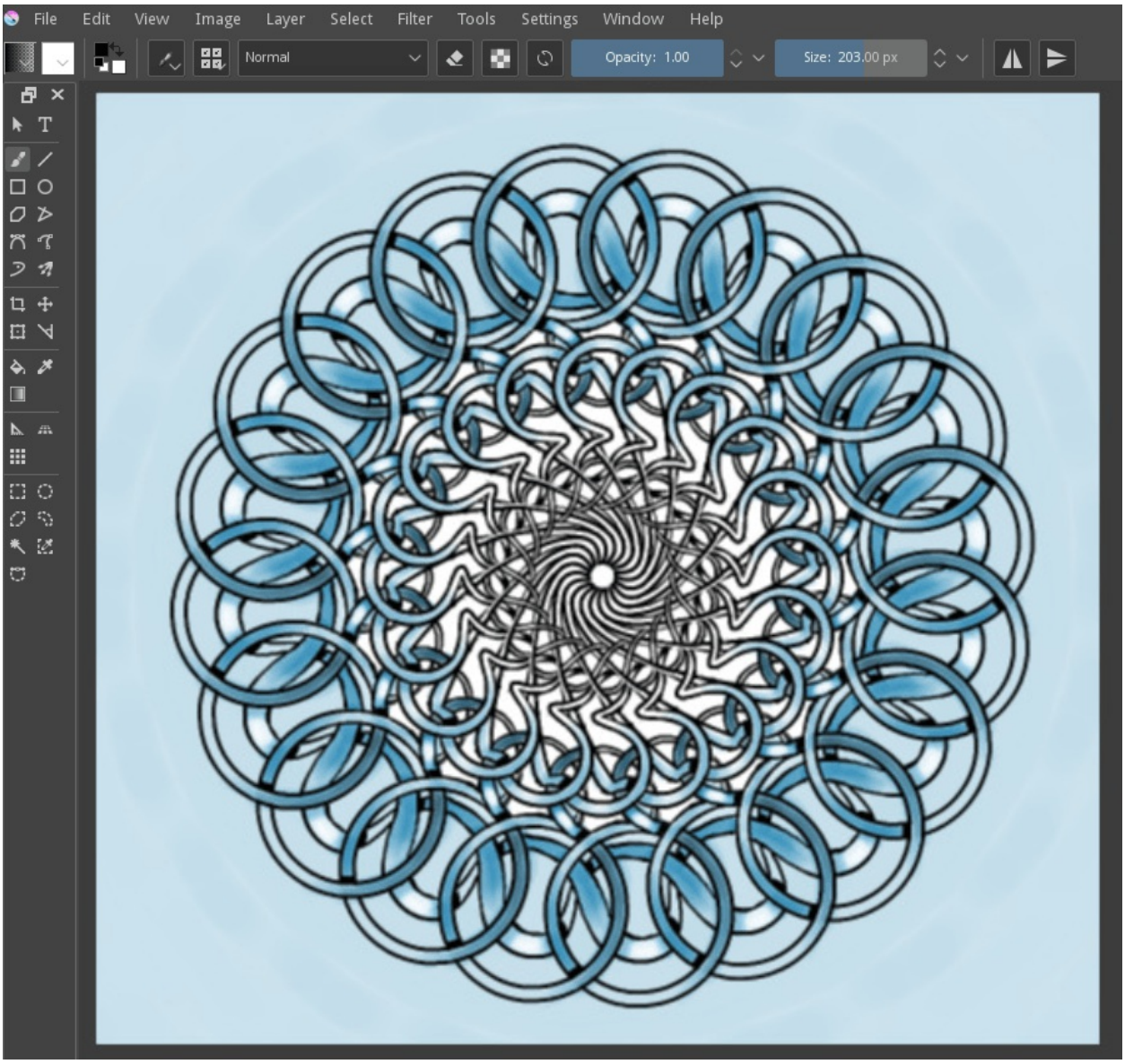
You can call the color history with the `H` key, common colors with the `U` key and the two shade selectors with the `Shift + N` and `Shift + M` shortcuts. The big selector can be called with the `Shift + I` shortcut on canvas.

Geometric Tools

Circles, rectangles, paths, Krita allows you to draw these easily.

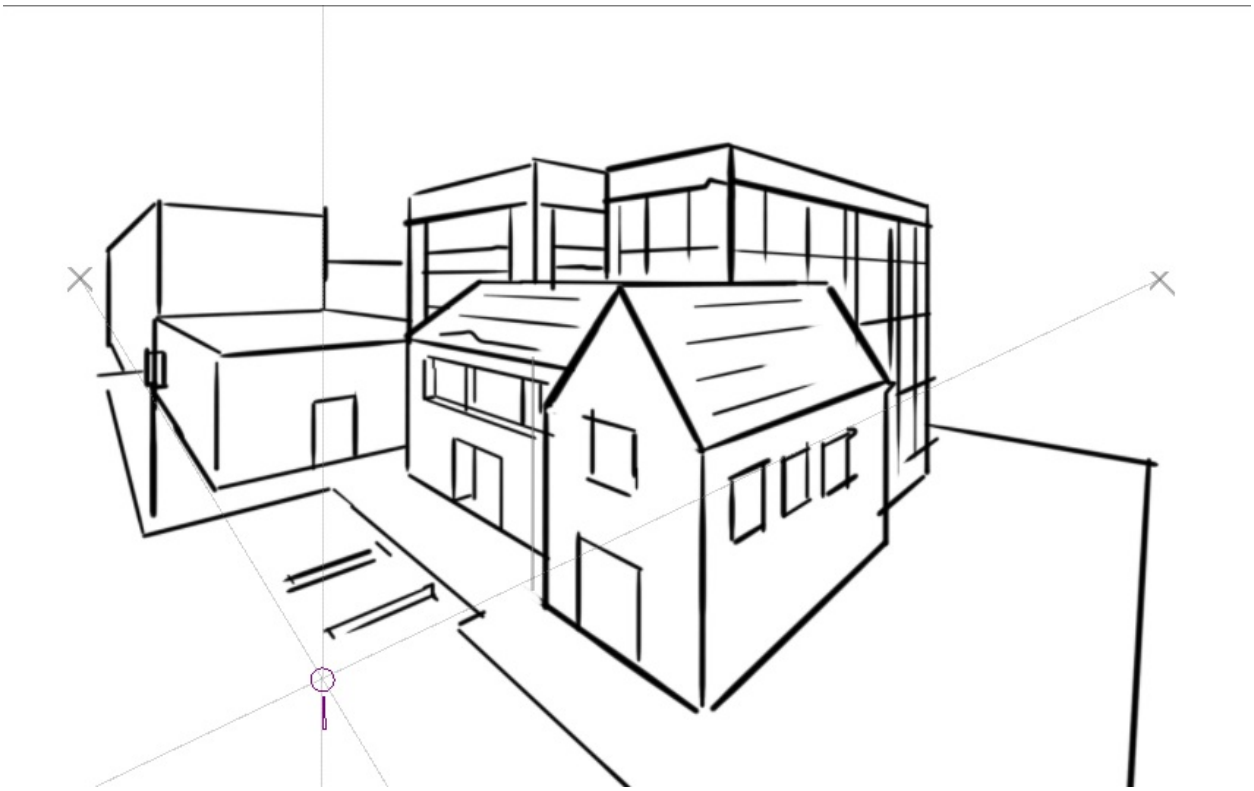
Multibrush, Mirror Symmetry and Wrap Around

These tools allow you to quickly paint a mirrored image, mandala or tiled texture in no time. Useful for backgrounds and abstract vignettes.



Assistants

The painting assistants can help you to set up a perspective, or a concentric circle and snap to them with the brush.



Krita’s vanishing point assistants in action.

Locking the Layer

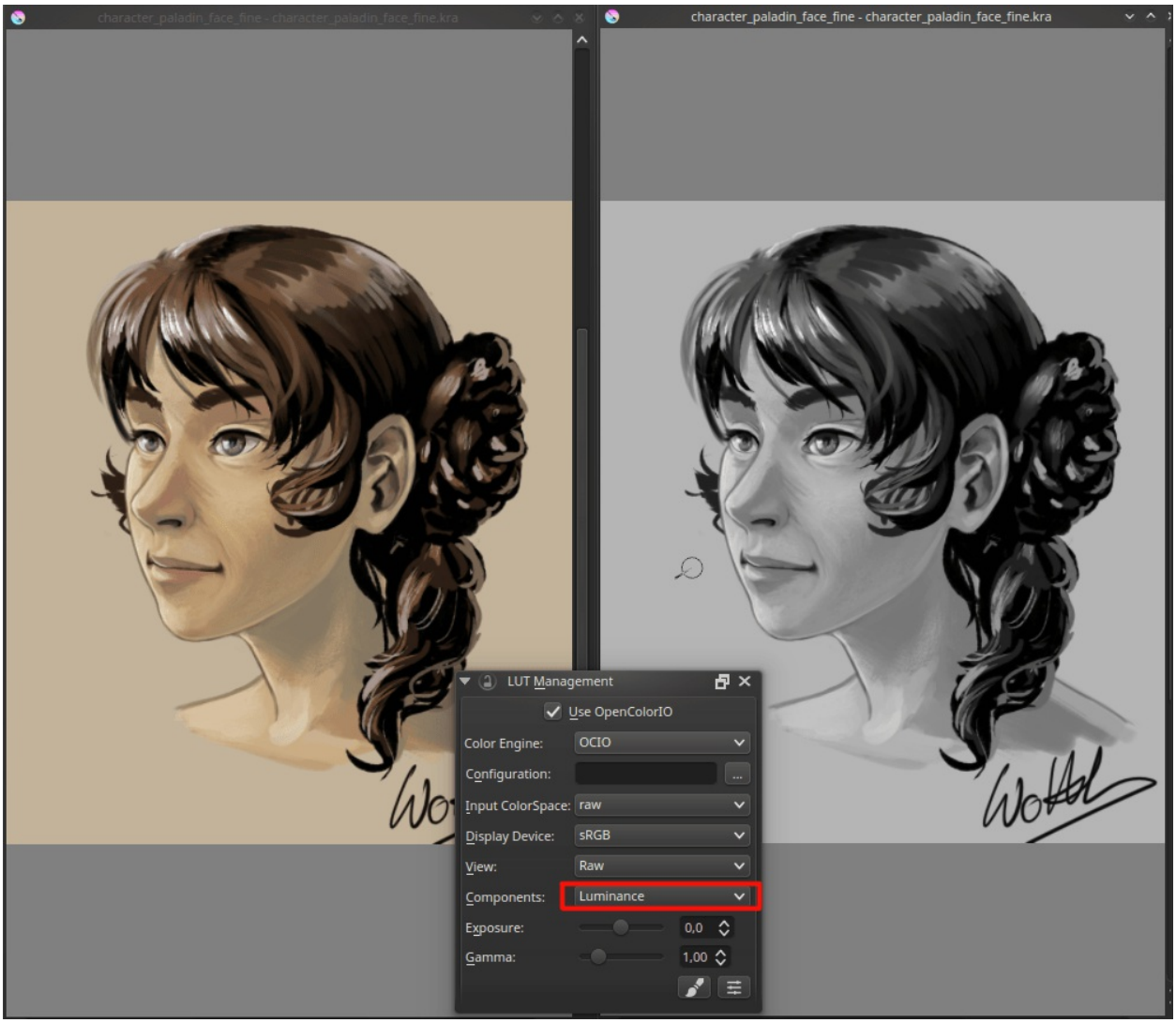
Lock the layer with the padlock so you don’t draw on it.

Quick Layer select

If you hold the R key and press a spot on your drawing, Krita will select the layer underneath the cursor. Really useful when dealing with a large number of layers.

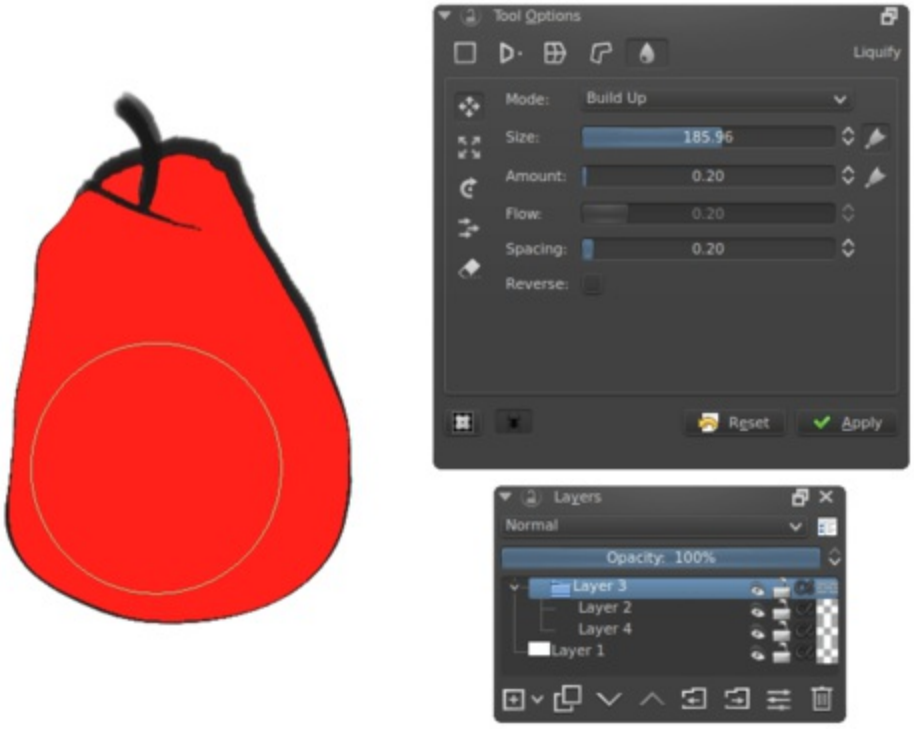
Color Management

This allows you to prepare your work for print, or to do tricks with the LUT docker so you can diagnose your image better. For example, using the LUT docker to turn the colors grayscale in a separate view, so you can see the values instantly.



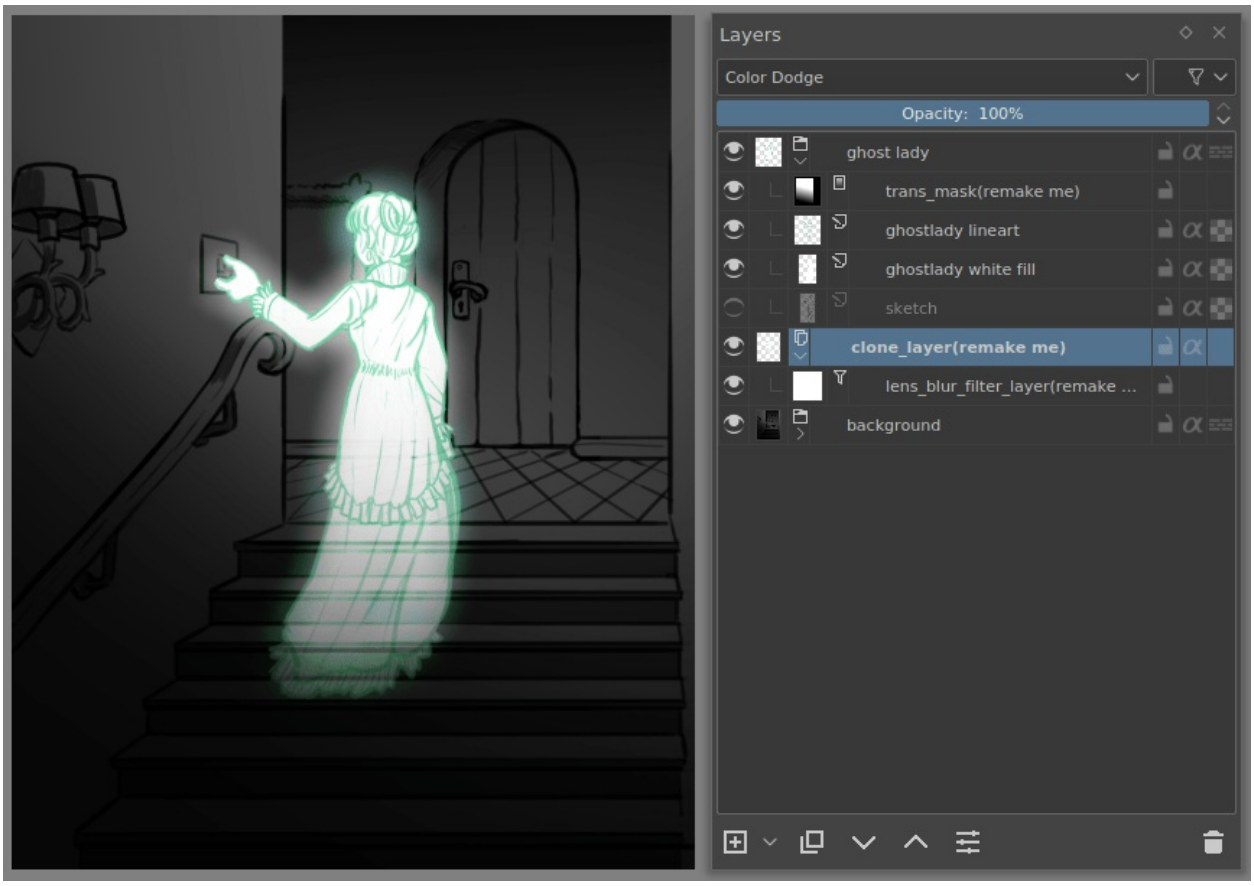
Advanced Transform Tools

Not just rotate and scale, but also cage, wrap, liquify and non-destructive transforms with the transform tool and masks.



More Filters and non-destructive filter layers and masks

With filters like color balance and curves you can make easy shadow layers. In fact, with the filter layers and layer masks you can make them apply on the fly as you draw underneath.



Pop-up palette

This is the little circular thing that is by default on the right click. You can organize your brushes in tags, and use those tags to fill up the pop-up palette. It also keeps a little color selector and color history, so you can switch brushes on the fly.

What does Krita lack compared to Paint Tool Sai?

- Variable width vector lines
- The selection source option for layers
- Dynamic hard-edges for strokes (the fringe effect)
- No mix-docker
- No Preset-tied stabilizer
- No per-preset hotkeys

Conclusion

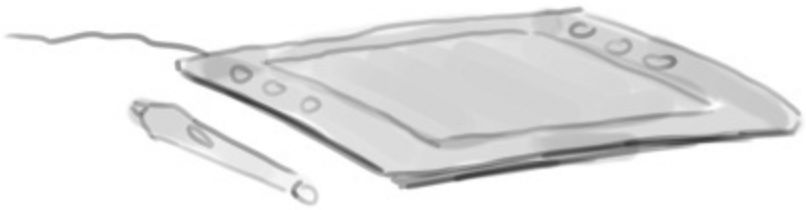
I hope this introduction got you a little more excited to use Krita, if not feel a little more at home.

Drawing Tablets

This page is about drawing tablets, what they are, how they work, and where things can go wrong.

What are Tablets?

Drawing with a mouse can be unintuitive and difficult compared to pencil and paper. Even worse, extended mouse use can result in carpal tunnel syndrome. That’s why most people who draw digitally use a specialized piece of hardware known as a drawing tablet.

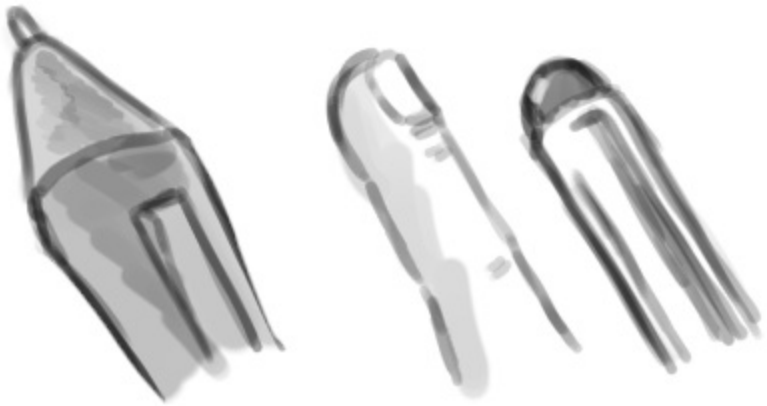


A drawing tablet is a piece of hardware that you can plug into your machine, much like a keyboard or mouse. It usually looks like a plastic pad, with a stylus. Another popular format is a computer monitor with stylus used to draw directly on the screen. These are better to use than a mouse because it's more natural to draw with a stylus and generally better for your wrists.

With a properly installed tablet stylus, Krita can use information like pressure sensitivity, allowing you to make strokes that get bigger or smaller depending on the pressure you put on them, to create richer and more interesting strokes.

Note

Sometimes, people confuse finger-touch styluses with a proper tablet. You can tell the difference because a drawing tablet stylus usually has a pointy nib, while a stylus made for finger-touch has a big rubbery round nib, like a finger. These tablets may not give good results and a pressure-sensitive tablet is recommended.



Supported Tablets

Supported tablets are owned by Krita developers themselves so they can reliably diagnose and fix bugs. We maintain a list of those here.

If you're looking for information about iPad or Android tablets, look here.

Drivers and Pressure Sensitivity

So you have bought a tablet, a real drawing tablet. And you wanna get it to work with Krita! So you plug in the USB cable, start up Krita and… It doesn’t work! Or well, you can make strokes, but that pressure sensitivity you heard so much about doesn’t seem to work.

This is because you need to install a program called a ‘driver’. Usually you can find the driver on a CD that was delivered alongside your tablet, or on the website of the manufacturer. Go install it, and while you wait, we’ll go into the details of what it is!

Running on your computer is a basic system doing all the tricky bits of running a computer for you. This is the operating system, or OS. Most people use an operating system called Windows, but people on an Apple device have an operating system called MacOS, and some people, including many of the developers use a system called Linux.

The base principle of all of these systems is the same though. You would like to run programs like Krita, called software, on your computer, and you want Krita to be able to communicate with the hardware, like your drawing tablet. But to have those two communicate can be really difficult - so the operating system, works as a glue between the two.

Whenever you start Krita, Krita will first make connections with the operating system, so it can ask it for a lot of these things: It would like to display things, and use the memory, and so on. Most importantly, it would like to get information from the tablet!

But it can’t! Turns out your operating system doesn’t know much about tablets. That’s what drivers are for. Installing a driver gives the operating system enough information so the OS can provide Krita with the right information about the tablet. The hardware manufacturer’s job is to write a proper driver for each operating system.

Warning

Because drivers modify the operating system a little, you will always need to restart your computer when installing or deinstalling a driver, so don’t forget to do this! Conversely, because Krita isn’t a driver, you don’t need to even deinstall it to reset the configuration, just rename or delete the configuration file.

Where it can go wrong: Windows

Krita automatically connects to your tablet if the drivers are installed. When things go wrong, usually the problem isn't with Krita.

Surface pro tablets need two drivers

Certain tablets using n-trig, like the Surface Pro, have two types of drivers. One is native, n-trig and the other one is called wintab. Since 3.3, Krita can use Windows Ink style drivers, just go to *Settings* ▸ *Configure Krita…* ▸ *Tablet Settings* and toggle the *Windows 8+ Pointer Input (Windows Ink)* there. You don't need to install the wintab drivers anymore for n-trig based pens.

Windows 10 updates

Sometimes a Windows 10 update can mess up tablet drivers. In that case, reinstalling the drivers should work.

Wacom Tablets

There are three known problems with Wacom tablets and Windows.

The first is that if you have customized the driver settings, then sometimes, often after a driver update, but that is not necessary, the driver breaks. Resetting the driver to the default settings and then loading your settings from a backup will solve this problem.

The second is that for some reason it might be necessary to change the display priority order. You might have to make your Cintiq screen your primary screen, or, on the other hand, make it the secondary screen. Double check in the Wacom settings utility that the tablet in the Cintiq is associated with the Cintiq screen.

The third is that if you have a display tablet like a cintiq and a wacom expresskeys remote, and you have disabled Windows Ink in the calibration

page of the stylus settings dialog so you have the full set of Wintab features, the cintiq needs to be the first item in Wacom's desktop application list. Otherwise you will have an offset between stylus and mouse that will get worse the more displays there are to the left of the cintiq display.

Broken Drivers

Tablet drivers need to be made by the manufacturer. Sometimes, with really cheap tablets, the hardware is fine, but the driver is badly written, which means that the driver just doesn't work well. We cannot do anything about this, sadly. You will have to send a complaint to the manufacturer for this, or buy a better tablet with better quality drivers.

Conflicting Drivers

On Windows, you can only have a single wintab-style driver installed at a time. So be sure to deinstall the previous driver before installing the one that comes with the tablet you want to use. Other operating systems are a bit better about this, but even Linux, where the drivers are often preinstalled, can't run two tablets with different drivers at once.

Interfering software

Sometimes, there's software that tries to make a security layer between Krita and the operating system. Sandboxie is an example of this. However, Krita cannot always connect to certain parts of the operating system while sandboxed, so it will often break in programs like Sandboxie. Similarly, certain mouse software, like Razer utilities can also affect whether Krita can talk to the operating system, converting tablet information to mouse information. This type of software should be configured to leave Krita alone, or be deinstalled.

The following software has been reported to interfere with tablet events to Krita:

1. Sandboxie
2. Razer mouse utilities

3. AMD Catalyst:sup:*TM* “game mode” (this broke the right click for someone)

Flicks (Wait circle showing up and then calling the popup palette)

If you have a situation where trying to draw keeps bringing up the pop-up palette on Windows, then the problem might be flicks. These are a type of gesture, a bit of Windows functionality that allows you to make a motion to serve as a keyboard shortcut. Windows automatically turns these on when you install tablet drivers, because the people who made this part of Windows forgot that people also draw with computers. So you will need to turn it off in the Windows flicks configuration.

Wacom Double Click Sensitivity (Straight starts of lines)

If you experience an issue where the start of the stroke is straight, and have a wacom tablet, it could be caused by the Wacom driver double-click detection.

To fix this, go to the Wacom settings utility and lower the double click sensitivity.

Supported Tablets

Supported tablets are the ones of which Krita developers have a version themselves, so they can reliably fix bugs with them. We maintain a list of those here.

Loading and Saving Brushes

In the real world, when painting or drawing, you don’t just use one tool. You use pencils, erasers, paintbrushes, different types of paint, inks, crayons, etc. All these have different ways of making marks.

In a digital program like Krita you have something similar. We call this a brush engine. And much like how cars have different engines that give different feels when driving, or how pencils make distinctly different marks than rollerball pens, different brush engines have totally different feels.

The brush engines have a lot of different settings as well. So, you can save those settings into presets.

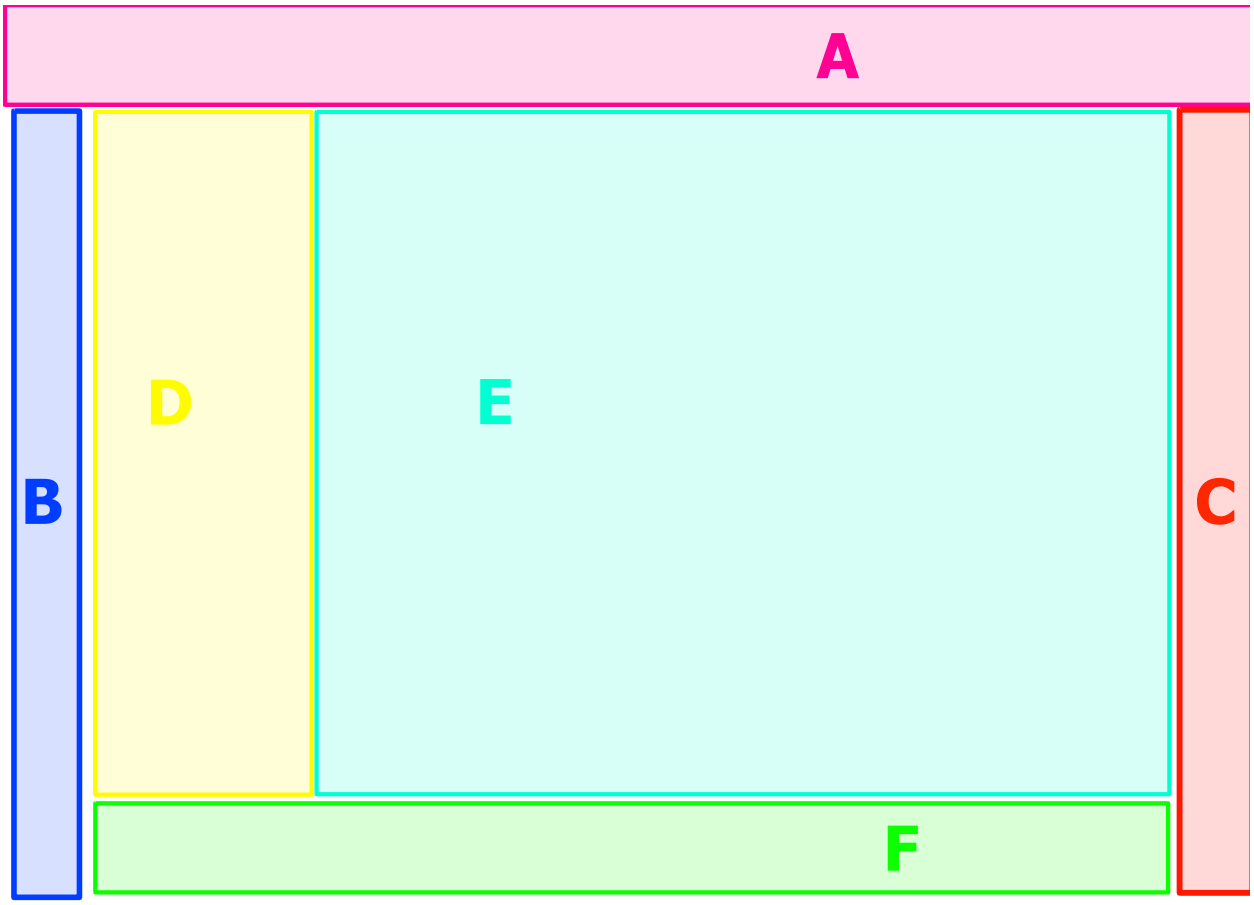
Unlike Photoshop, Krita makes a difference between brush-tips and brush-presets. Tips are only a stamp of sorts, while the preset uses a tip and many other settings to create the full brush.

The Brush settings drop-down

To start, the Brush Settings Editor panel can be accessed in the toolbar, between the *Choose brush preset* button on the right and the *Fill Patterns* button on the left. Alternately, you can use the `F5` key to open it.

When you open Brush Settings Editor panel you will see something like this:

Tour of the brush settings drop-down



The brush settings drop-down is divided into six areas,

Section A - General Information

This contains the **Preset Icon**, **Live Brush Preview**, the **Preset Name**, the **Engine** name, and several buttons for saving, renaming, and reloading.

Krita's brush settings are stored into the metadata of a 200x200 PNG (the KPP file), where the image in the PNG file becomes the preset icon. This icon is used everywhere in Krita, and is useful for differentiating brushes in ways that the live preview cannot.

The live preview shows a stroke of the current brush as a little s-curve wiggle, with the pressure being non-existent on the left, and increasing to full pressure as it goes to the right. It can thus show the effect of the Pressure, Drawing Angle, Distance, Fade and Fuzzy Dab sensors, but none of the others. For some brush engines it cannot show anything. For the color

smudge, filter brush and clone tool, it shows an alternating line pattern because these brush engines use the pixels already on canvas to change their effect.

After the preset name, there's a button for **renaming** the brush. This will save the brush as a new brush and blacklist the previous name.

Engine

The engine of a brush is the underlying programming that generates the stroke from a brush. What that means is that different brush engines have different options and different results. You can see this as the difference between using crayons, pencils and inks, but because computers are maths devices, most of our brush engines produce different things in a more mathematical way.

For most artists the mathematical nature doesn't matter as much as the different textures and marks each brush engine, and each brush engine has its own distinct flavor and use, and can be further customized by modifying the options.

Reloading

If you change a preset, an icon will appear behind the engine name. This is the *Reload the brush preset* button. You can use it to revert to the original brush settings.

Saving a preset

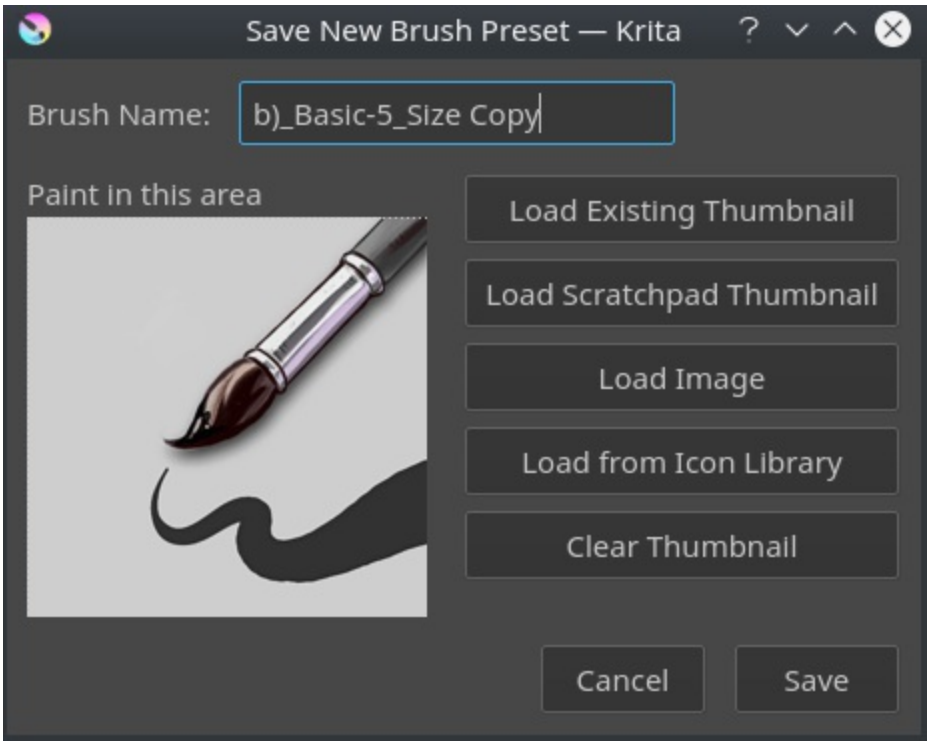
On the right, there's *Save New Brush Preset…* and *Overwrite Brush* buttons.

Save New Brush Preset…
: Will take the current preset and all its changes and save it as a new preset. If no change was made, you will be making a copy of the current preset.

Overwrite Brush
: This will only enable if there are any changes. Pressing this will override the current preset with the new settings, keeping the name and the icon

intact. It will always make a timestamped back up in the resources folder.

Save new preset will call up the following window, with a mini scratch pad, and all sorts of options to change the preset icon:



The image on the left is a mini scratch pad, you can draw on it with the current brush, allowing small modifications on the fly.

Brush Name:
: The Name of your brush. This is also used for the KPP file. If there's already a brush with that name, it will effectively overwrite it.

Load Existing Thumbnail
: This will load the existing thumbnail inside the preset.

Load Scratch Pad Thumbnail
: This will load the dashed area from the big scratch pad (Section C) into the thumbnail area.

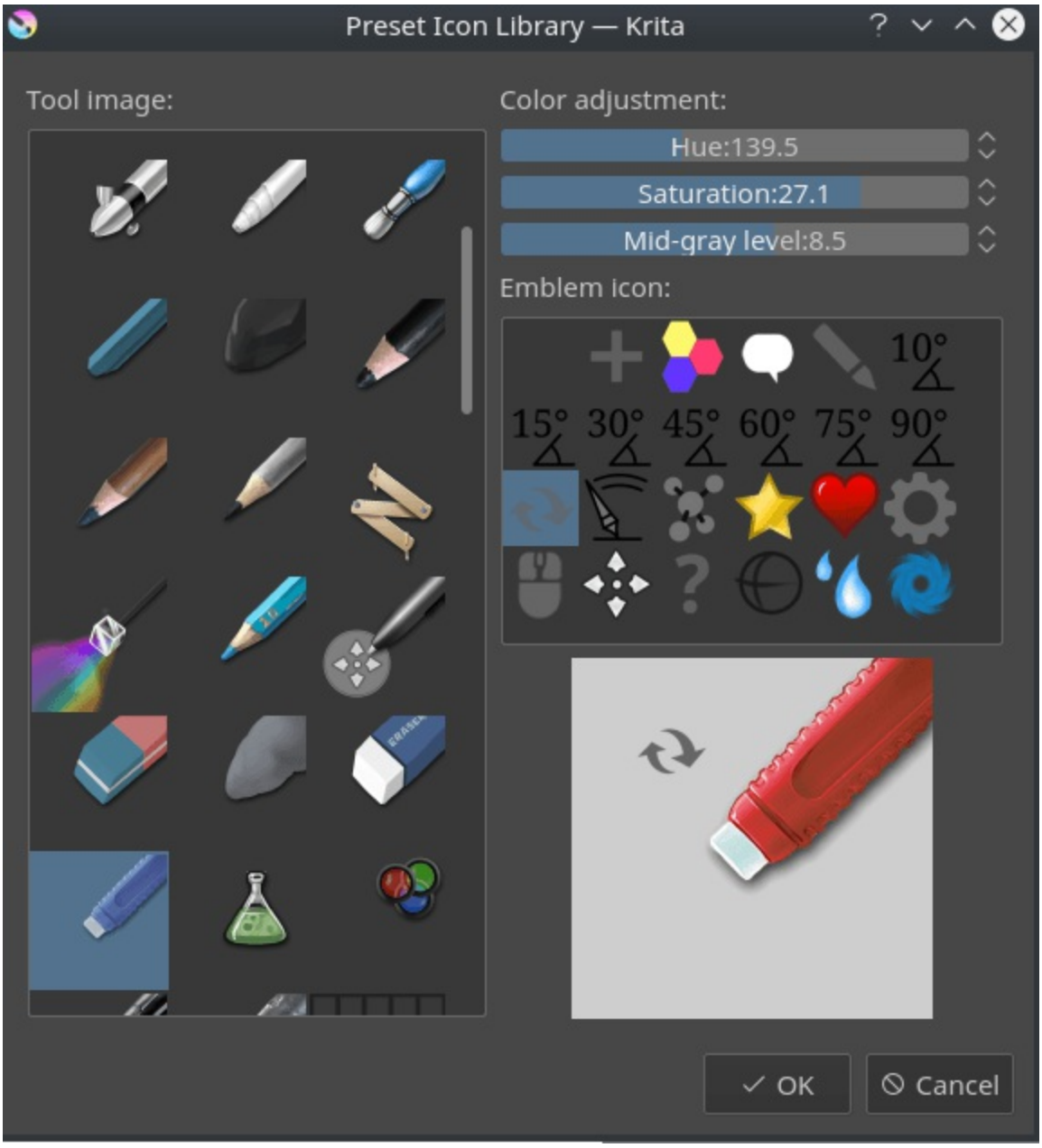
Load Image
: With this you can choose an image from disk to load as a thumbnail.

Load from Icon Library
    This opens up the icon library.

Clear Thumbnail
    This will make the mini scratch pad white.

The Icon Library

To make making presets icons faster, Krita got an icon library.



It allows you to select tool icons, and an optional small emblem. When you press *OK* it will load the resulting combination into the mini scratch pad and you can draw in the stroke.

If you go to your resources folder, there's a folder there called "preset_icons", and in this folder there are "tool_icons" and "emblem_icons". You can add semi-transparent PNGs here and Krita will load those into the icon library as well so you can customize your icons even more!

At the top right of the icon library, there are three sliders. They allow you to adjust the tool icon. The top two are the same Hue and Saturation as in HSL adjustment, and the lowest slider is a super simple levels filter. This is done this way because the levels filter allows maintaining the darkest shadows and brightest highlights on a tool icon, making it much better for quick adjustments.

If you're done with everything, you can press *Save* in the *Save New Brush Preset* dialog and Krita will save the new brush.

Section B - The Preset Chooser

The preset chooser is much the same as the preset docker and the preset drop-down on the `F6` key. It's unique in that it allows you to filter by engine and this is also where you can create brushes for an engine from scratch.

It is by default collapsed, so you will need to press the arrow at the top left of the brush engine to show it.

The top drop-down is set to "all" by default, which means it shows all engines. It then shows a tag section where you can select the tags, the preset list and the search bar.

Underneath that there's a plus icon, which when pressed gives you the full list of Krita's engines. Selecting an engine from the list will show the brushes for that engine.

The trashcan icon does the same as it does in the preset docker: delete, or rather, blacklist a preset so it won't show up in the list.

Section C - The Scratch pad

When you tweak your brushes, you want to be able to check what each setting does. That's why, to the right of the settings drop-down, there is a scratch pad.

It is by default collapsed, so you will have to press the arrow at the top right of the brush settings to show it.

When saving a new preset, you can choose to get the icon from the scratch pad, this will load the dash area into the mini scratch pad of the *Save New Brush Preset* dialog.

The scratch pad has five buttons underneath it. These are in order for:

1. Fill area with brush preset icon
2. Fill area with current image
3. Fill area with gradient (useful for smudge brushes)
4. Fill area with background color
5. Reset area to white

Section D - The Options List

The options, as stated above, are different per brush engine. These represent the different parameters, toggles and knobs that you can turn to make a brush preset unique. For a couple of options, the main things to change are sliders and check boxes, but for a lot of them, they use curves instead.

Some options can be toggled, as noted by the little check boxes next to them, but others, like flow and opacity are so fundamental to how the brush works, that they are always on.

The little padlock icon next to the options is for locking the brush. This has its own page.

Section E - Option Configuration Widget

Where section D is the list of options, section E is the widget where you can change things.

Using sensor curves

One of the big important things that make art unique to the artist who created it is the style of the strokes. Strokes are different because they differ in speed, rotation, direction, and the amount of pressure put onto the stylus. Because these are so important, we would want to customize how these values are understood in detail. The best way to do this is to use curves.

Curves show up with the size widget for example. With an inking brush, we want to have size mapped to pressure. Just toggling the size option in the option list will do that.

However, different people have different wrists and thus will press differently on their stylus. Someone who presses softly tends to find it easy to make thin strokes, but very difficult to make thick strokes. Conversely, someone who presses hard on their stylus naturally will have a hard time making thin strokes, but easily makes thick ones.

Such a situation can be improved by using the curves to map pressure to output thinner lines or thicker ones.

The brush settings curves even have quick curve buttons for these at the top. Someone who has a hard time making small strokes should try the second to last concave button, while someone who has a hard time making thick strokes should try the third button, the S shape.

Underneath the curve widget there are two more options:

Share curve across all settings
: This is for the list of sensors. Toggling this will make all the sensors use the same curve. Unchecked, all checked sensors will have separate curves.

Curves calculation mode:
: This indicates how the multiple values of the sensor curves are used. The

curves always go from 0 to 1.0, so if one curve outputs 0.5 and the other 0.7, then…

Multiply
: Will multiply the two values, 0.5*0.7 = 0.35.

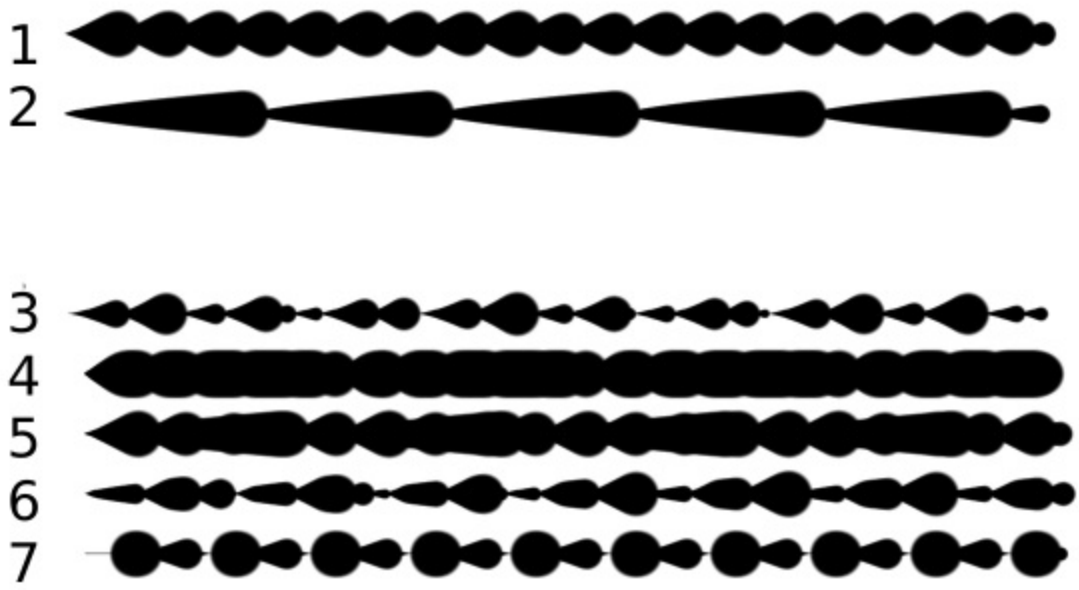
Addition
: Will add the two to a maximum of 1.0, so 0.5+0.7 = 1.2, which is then capped at 1.0.

Maximum
: Will compare the two and pick the largest. So in the case of 0.5 and 0.7, the result is 0.7.

Minimum
: Will compare the two and pick the smallest. So in the case of 0.5 and 0.7, the result is 0.5.

Difference
: Will subtract the smallest value from the largest, so 0.7-0.5 = 0.2.

It’s maybe better to see with the following example:



The first two are regular, the rest with different multiplication types.

1. Is a brush with size set to the distance sensor.
2. Is a brush with the size set to the fade sensor.
3. The size is calculated from the fade and distance sensors multiplied.
4. The size is calculated from the fade and distance sensors added to each other. Notice how thick it is.
5. The size takes the maximum value from the values of the fade and distance sensors.
6. The size takes the minimum value from the values of the fade and distance sensors.
7. The size is calculated by having the largest of the values subtracted with the smallest of the values.

Section F - Miscellaneous options

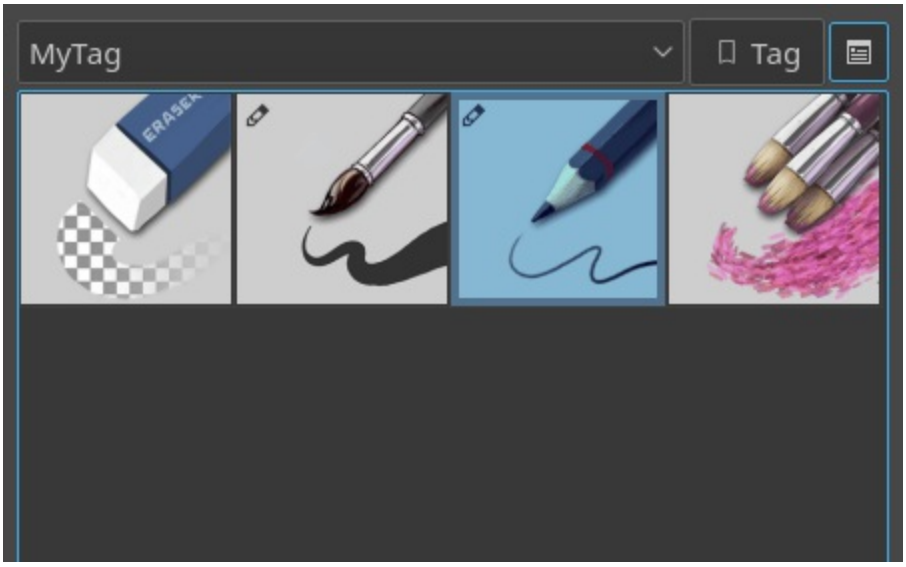
Eraser switch size

This switches the brush to a separately stored size when using the E key.

Eraser switch opacity

Same as above, but then with Eraser opacity.

Temporarily save tweaks to preset
: This enables dirty presets. Dirty presets store the tweaks you make as long as this session of Krita is active. After that, they revert to default. Dirtied presets can be recognized by the icon in the top-left of the preset.



The icon in the top left of the first two presets indicate it is “Dirty”, meaning there are tweaks made to the preset.

Instant preview
: This allows you to toggle instant preview on the brush. The Instant Preview has a super-secret feature: when you press the instant preview label, and then right click it, it will show a threshold slider. This slider determines at what brush size instant preview is activated for the brush. This is useful because small brushes can be slower with instant preview, so the threshold ensures it only activates when necessary.

The On-canvas brush settings

There are on-canvas brush settings. If you open up the pop-up palette, there should be an icon on the bottom-right. Press that to show the on-canvas brush settings. You will see several sliders here, to quickly make small changes.

At the top it shows the currently active preset. Next to that is a settings button, click that to get a list of settings that can be shown and organized for the given brush engine. You can use the up and down arrows to order their position, and then left and right arrows to add or remove from the list. You can also drag and drop.

Making a Brush Preset

Now, let's make a simple brush to test the waters with:

Getting a default for the brush engine.

First, open the settings with the `F5` key.

Then, press the arrow on the upper left to open the preset chooser. There, press the "+" icon to get a list of engines. For this brush we're gonna make a pixel brush.

Example: Making an inking brush

1. Draw on the scratch pad to see what the current brush looks like. If done correctly, you should have a 5px wide brush that has pressure set to opacity.
2. Let us turn off the opacity first. Click on the opacity option in the right-hand list. The settings should now be changed to a big curve. This is the sensor curve.
3. Uncheck the *Enable Pen Settings* checkbox.
4. Test on the scratch pad… there still seems to be something affecting opacity. This is due to the flow option.
5. Select the Flow option from the list on the right hand. Flow is like Opacity, except that Flow is per dab, and opacity is per stroke.
6. Uncheck the *Enable Pen Settings* checkbox here as well. Test again.
7. Now you should be getting somewhere towards an inking brush. It is still too small however, and kinda grainy looking. Click Brush Tip in the brush engine options.
8. Here, the diameter is the size of the brush-tip. You can touch the slider

change the size, or right-click it and type in a value. Set it to 25 and test again. It should be much better.

9. Now to make the brush feel a bit softer, turn down the fade parameter to about 0.9. This'll give the *brush mask* a softer edge.
10. If you test again, you'll notice the fade doesn't seem to have much effect. This has to do with the spacing of the dabs: The closer they are together, the harder the line is. By default, this is 0.1, which is a bit low. If you set it to 10 and test, you'll see what kind of effect spacing has. The Auto checkbox changes the way the spacing is calculated, and Auto Spacing with a value of 0.8 is the best value for inking brushes. Don't forget that you can use right-click to type in a value.
11. Now, when you test, the fade seems to have a normal effect… except on the really small sizes, which look pixelly. To get rid of that, check the anti-aliasing check box. If you test again, the lines should be much nicer now.

Saving the new Brush

When you're satisfied, go to the upper left and select *Save New Brush Preset…* button.

You will get the save preset dialog. Name the brush something like "My Preset". Then, select *Load from Icon Library* to get the icon library. Choose a nice tool icon and press *OK*.

The icon will be loaded into the mini scratch pad on the left. Now doodle a nice stroke next to it. If you feel you messed up, just go back to the icon library to load a new icon.

Finally press *Save*, and your brush should be done.

You can further modify your inking brush by…

Changing the amount of pressure you need to put on a brush to make it full size.

To do this, select the size option, and press the pressure sensor from the list next to the curve. The curve should look like a straight line. Now if you want a brush that gets big with little pressure, tick on the curve to

make a point, and drag the point to the upper-left. The more the point is to the upper-left, the more extreme the effect. If you want instead a brush that you have to press really hard on to get to full size, drag the dot to the lower-right. Such a brush is useful for fine details. Don't forget to save the changes to your brush when done.

Making the fine lines look even softer by using the flow option.
To do this, select the flow option, and turn back on the *Enable Pen Settings* check box. Now if you test this, it is indeed a bit softer, but maybe a bit too much. Click on the curve to make a dot, and drag that dot to the top-left, half-way the horizontal of the first square of the grid. Now, if you test, the thin lines are much softer, but the hard your press, the harder the brush becomes.

Sharing Brushes

Okay, so you've made a new brush and want to share it. There are several ways to share a brush preset.

The recommended way to share brushes and presets is by using the resource bundle system. We have detailed instructions on how to use them on the resource management page.

However, there are various old-fashioned ways of sharing brushes that can be useful when importing and loading very old packs:

Sharing a single preset

There are three types of resources a single preset can take:

1. A Paintoppreset file: This is the preset proper, with the icon and the curves stored inside.
2. A Brush file: This is the brush tip. When using masked brushes, there are two of these.
3. A Pattern file: this is when you are using textures.

So when you have a brush that uses unique predefined tips for either brush tip

or masked brush, or unique textures you will need to share those resources as well with the other person.

To find those resources, go to *Settings* ▸ *Manage Resources…* ▸ *Open Resource Folder*.

There, the preset file will be inside paintoppresets, the brush tips inside brushes and the texture inside patterns.

Importing a single KPP file.

Now, if you want to use the single preset, you should go to the preset chooser on the `F6` key and press the folder icon there. This will give a file dialog. Navigate to the kpp file and open it to import it.

If there are brush tips and patterns coming with the file, do the same with pattern via the pattern docker, and for the brush-tip go to the settings drop-down (`F5`) and then go to the “brush-tip” option. There, select predefined brush, and then the “import” button to call up the file dialog.

Sharing via ZIP (old-fashioned)

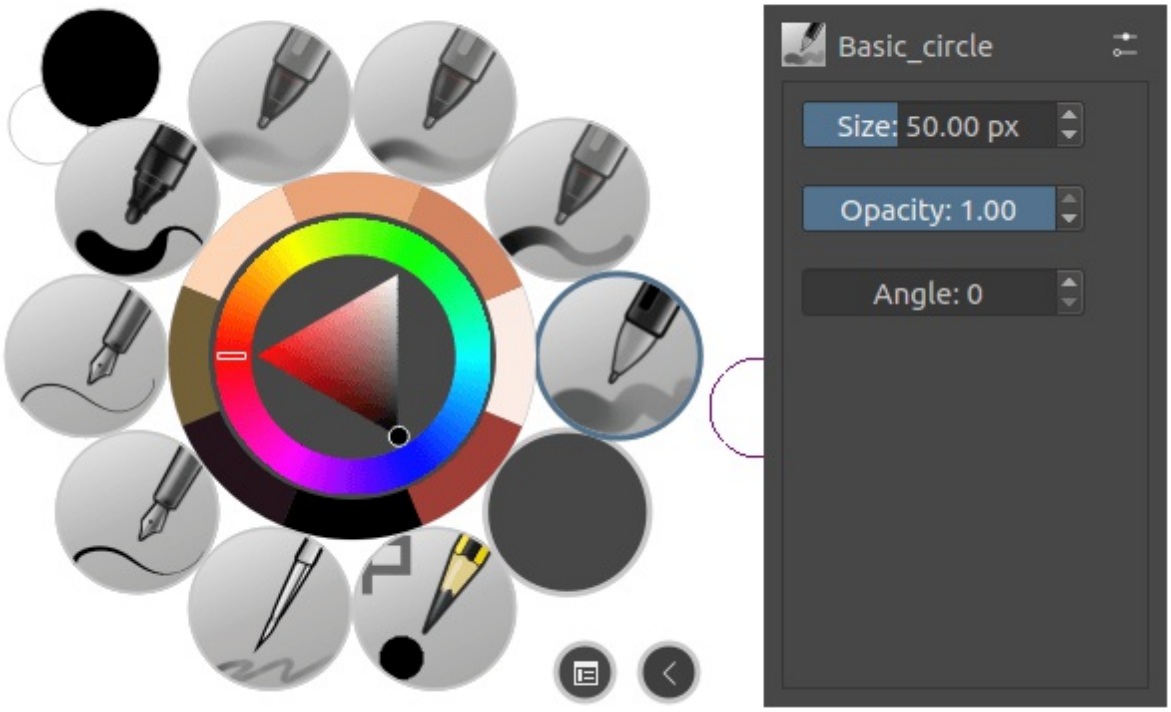
Sharing via ZIP should be replaced with resource bundles, but older brush packs are stored in ZIP files.

Using a ZIP with the relevant files.

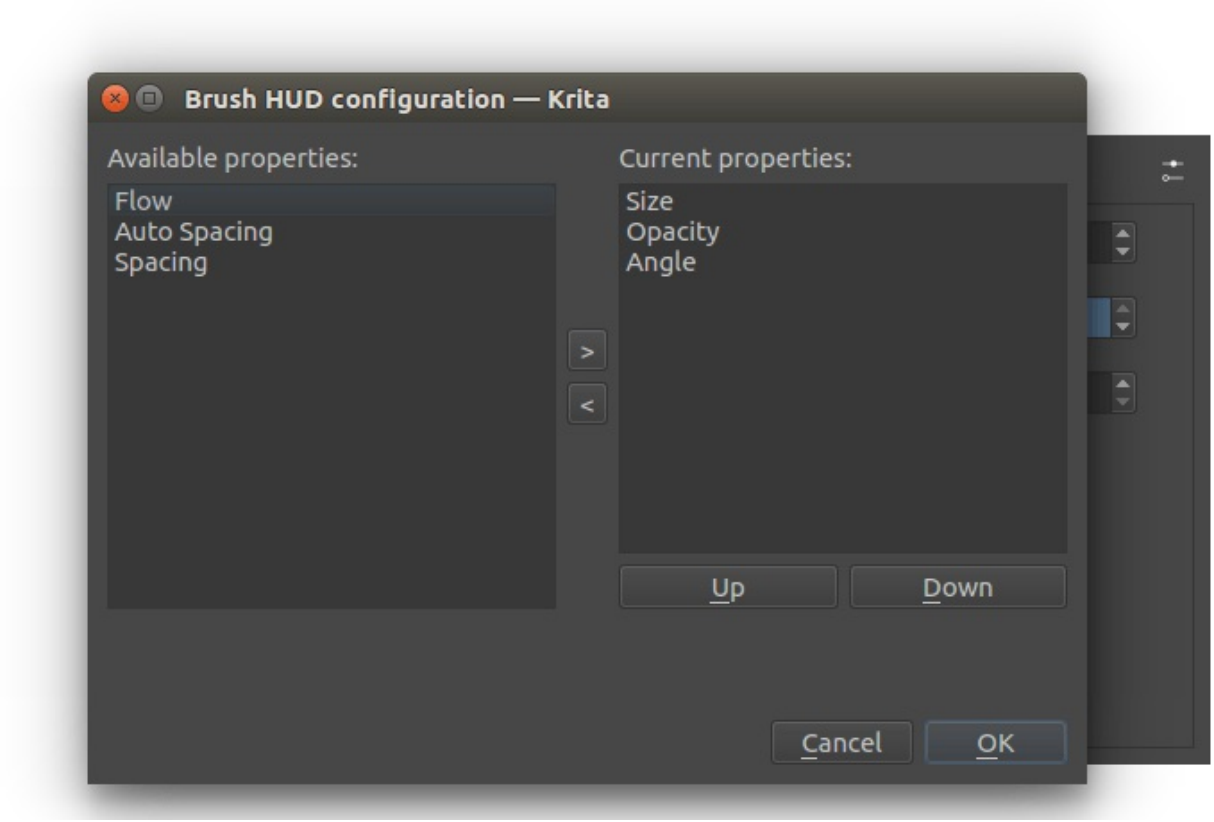
1. Go to *Settings* ▸ *Manage Resources…* ▸ *Open Resource Folder* to open the resource folder.
2. Then, open up the ZIP file.
3. Copy the brushes, paintoppresets and patterns folders from the ZIP file to the resource folder. You should get a prompt to merge the folders, agree to this.
4. Restart Krita.
5. Enjoy your brushes!

On-Canvas Brush Editor

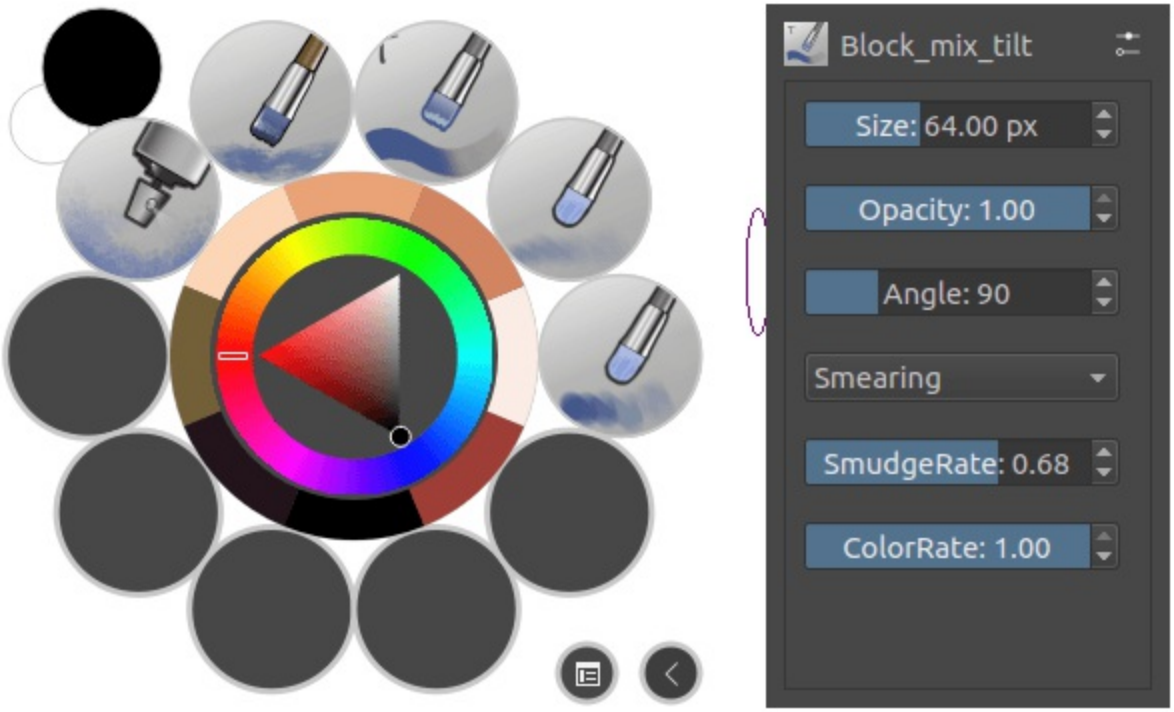
Krita's brush editor is, as you may know, on the F5 key. However, sometimes you just want to modify a single parameter quickly. Perhaps even in canvas-only mode. The on canvas brush editor or brush HUD allows you to do this. It's accessible from the pop-up palette, by ticking the lower-right arrow button.



You can change the amount of visible settings and their order by clicking the settings icon next to the brush name.



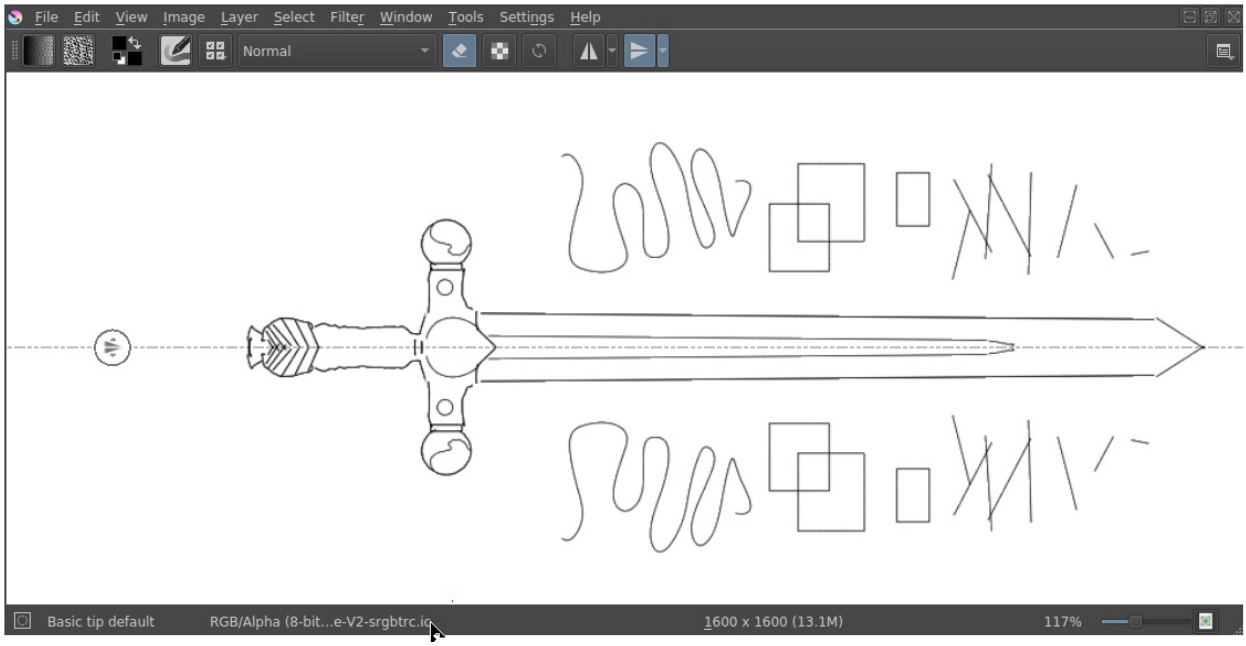
On the left are all unused settings, on the right are all used settings. You use the > and < buttons to move a setting between the two columns. The *Up* and *Down* buttons allow you to adjust the order of the used settings, for when you think flow is more important than size.



These set-ups are PER brush engine, so different brush engines can have different configurations.

Mirror Tools

Draw on one side of a mirror line while the Mirror Tool copies the results to the other side. The Mirror Tools are accessed along the toolbar. You can move the location of the mirror line by grabbing the handle.



Mirror Tools give a similar result to the Multibrush Tool, but unlike the Multibrush which only traces brush strokes like the Freehand Brush Tool, the Mirror Tools can be used with any other tool that traces strokes, such as the Straight Line Tool and the Bezier Curve Tool, and even with the Multibrush Tool.

Horizontal Mirror Tool - Mirror the results along the horizontal axis.

Vertical Mirror Tool - Mirror the results along the vertical axis.

There are additional options for each tool. You can access these by the clicking the drop-down arrow located on the right of each tool.

- Hide Mirror X/Y Line (toggle) – Locks the mirror axis and hides the axis line.

- Lock X/Y Line (toggle) - hides the move icon on the axis line.
- Move to Canvas Center X/Y - Moves the axis line to the center of the canvas.

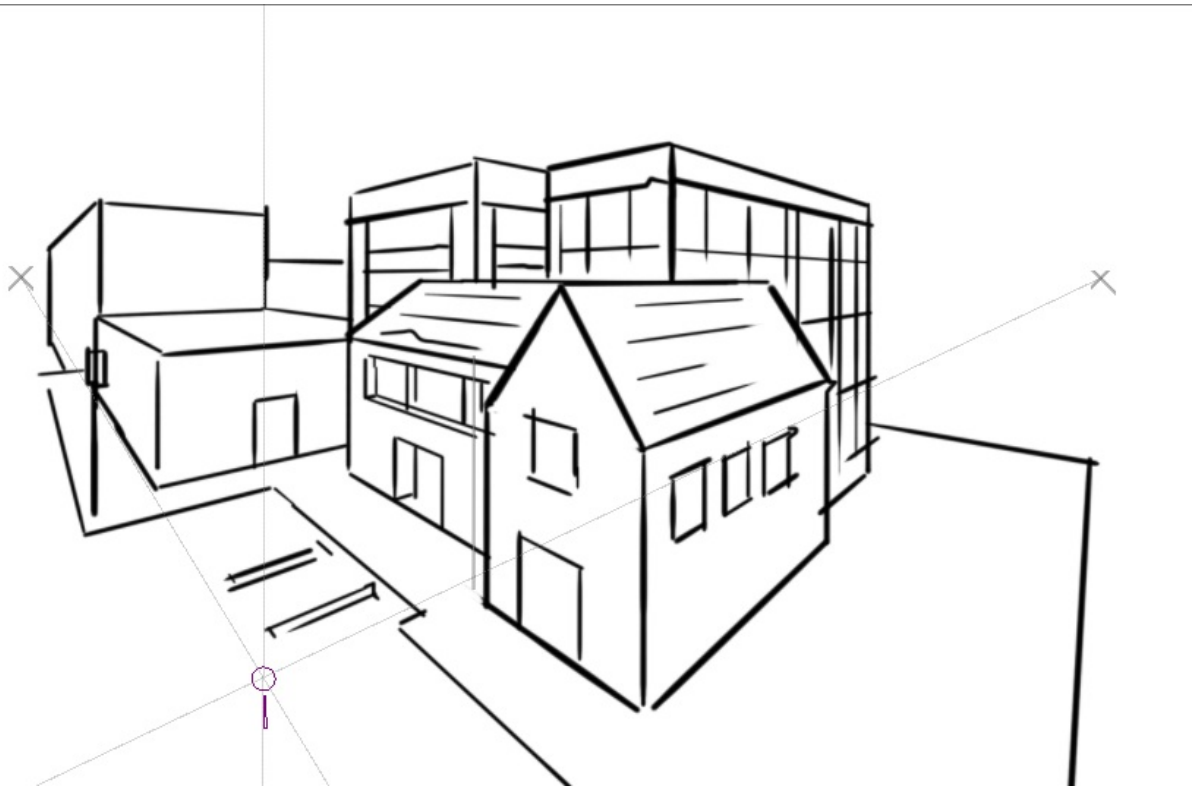
Mirroring along a rotated line

The Mirror Tool can only mirror along a perfectly vertical or horizontal line. To mirror along a line that is at a rotated angle, use the Multibrush Tool and its various parameters, it has more advanced options besides basic symmetry.

Painting with Assistants

The assistant system allows you to have a little help while drawing straight lines or circles.

They can function as a preview shape, or you can snap onto them with the freehand brush tool. In the tool options of free hand brush, you can toggle *Snap to Assistants* to turn on snapping.



Krita’s vanishing point assistants in action.

The following assistants are available in Krita:

Types

There are several types in Krita. You can select a type of assistant via the tool options docker.

Ellipse

An assistant for drawing ellipses and circles.

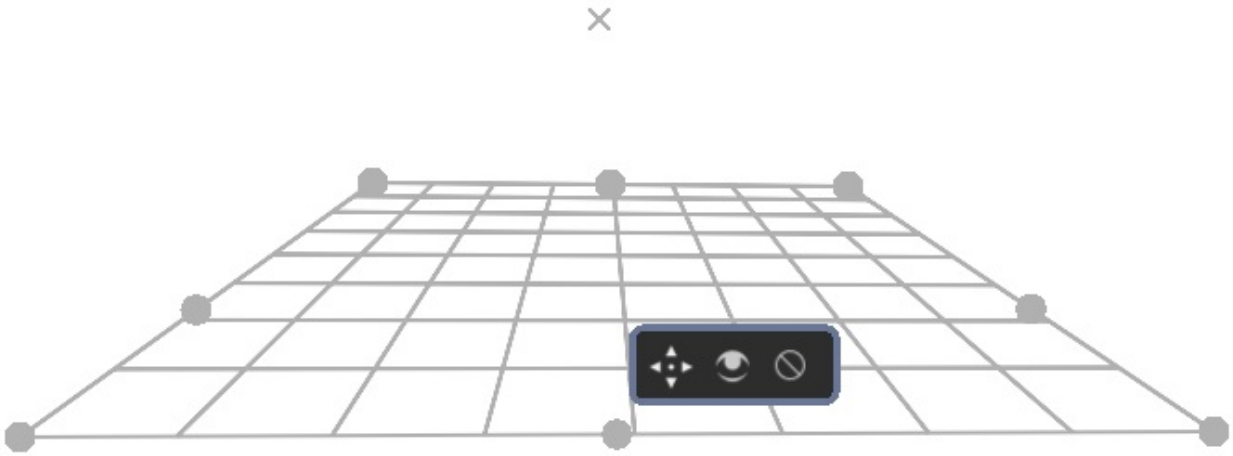
This assistant consists of three points: the first two are the axis of the ellipse, and the last one is to determine its width.

Concentric Ellipse
: The same an ellipse, but allows for making ellipses that are concentric to each other.

If you press the `Shift` key while holding the first two handles, they will snap to perfectly horizontal or vertical lines. Press the `Shift` key while holding the third handle, and it'll snap to a perfect circle.

Perspective

This ruler allows you to draw and manipulate grids on the canvas that can serve as perspective guides for your painting. A grid can be added to your canvas by first clicking the tool in the tool bar and then clicking four points on the canvas which will serve as the four corners of your grid.



This grid can be used with the 'perspective' sensor, which can influence brushes.

The grid can be manipulated by pulling on any of its four corners. The grid

can be extended by clicking and dragging a midpoint of one of its edges. This will allow you to expand the grid at other angles. This process can be repeated on any subsequent grid or grid section.If you press the `Shift` key while holding any of the corner handles, they'll snap to one of the other corner handles, in sets. You can delete any grid by clicking on the cancel button at its center. This tool can be used to build reference for complex scenes.

Ruler

There are three assistants in this group:

Ruler
: Helps create a straight line between two points.

Infinite Ruler
: Extrapolates a straight line beyond the two visible points on the canvas.

Parallel Ruler
: This ruler allows you to draw a line parallel to the line between the two points anywhere on the canvas.

If you press the `Shift` key while holding the first two handles, they will snap to perfectly horizontal or vertical lines.

Spline

This assistant allows you to position and adjust four points to create a cubic bezier curve. You can then draw along the curve, snapping your brush stroke directly to the curve line. Perfect curves every time!

If you press the `Shift` key while holding the first two handles, they will snap to perfectly horizontal or vertical lines. Press the `Shift` key while holding the third or fourth handle, they will snap relative to the handle they are attached to.

Vanishing Point

This assistant allows you to create a vanishing point, typically used for a horizon line. A preview line is drawn and all your snapped lines are drawn to this line.

It is one point, with four helper points to align it to previously created perspective lines.

They are made and manipulated with the Assistant Tool.

If you press the `Shift` key while holding the center handle, they will snap to perfectly horizontal or vertical lines depending on the position of where it previously was.

Changed in version 4.1: The vanishing point assistant also shows several general lines.

When you've just created, or when you've just moved a vanishing point assistant, it will be selected. This means you can modify the amount of lines shown in the tool options of the Assistant Tool.

Fish Eye Point

Like the vanishing point assistant, this assistant is per a set of parallel lines in a 3d space. So to use it effectively, use two, where the second is at a 90 degrees angle of the first, and add a vanishing point to the center of both. Or combine one with a parallel ruler and a vanishing point, or even one with two vanishing points. The possibilities are quite large.

This assistant will not just give feedback/snapping between the vanishing points, but also give feedback to the relative left and right of the assistant. This is so you can use it in edge-cases like panoramas with relative ease.

If you press the `Shift` key while holding the first two handles, they will snap to perfectly horizontal or vertical lines. Press the `Shift` key while holding the third handle, and it'll snap to a perfect circle.

Tutorials

Check out this in depth discussion and tutorial on https://www.youtube.com/watch?v=OhEv2pw3EuI

Setting up Krita for technical drawing-like perspectives

So now that you've seen the wide range of drawing assistants that Krita offers, here is an example of how using these assistants you can set up Krita for technical drawing.

This tutorial below should give you an idea of how to set up the assistants for specific types of technical views.

If you want to instead do the true projection, check out the projection category.

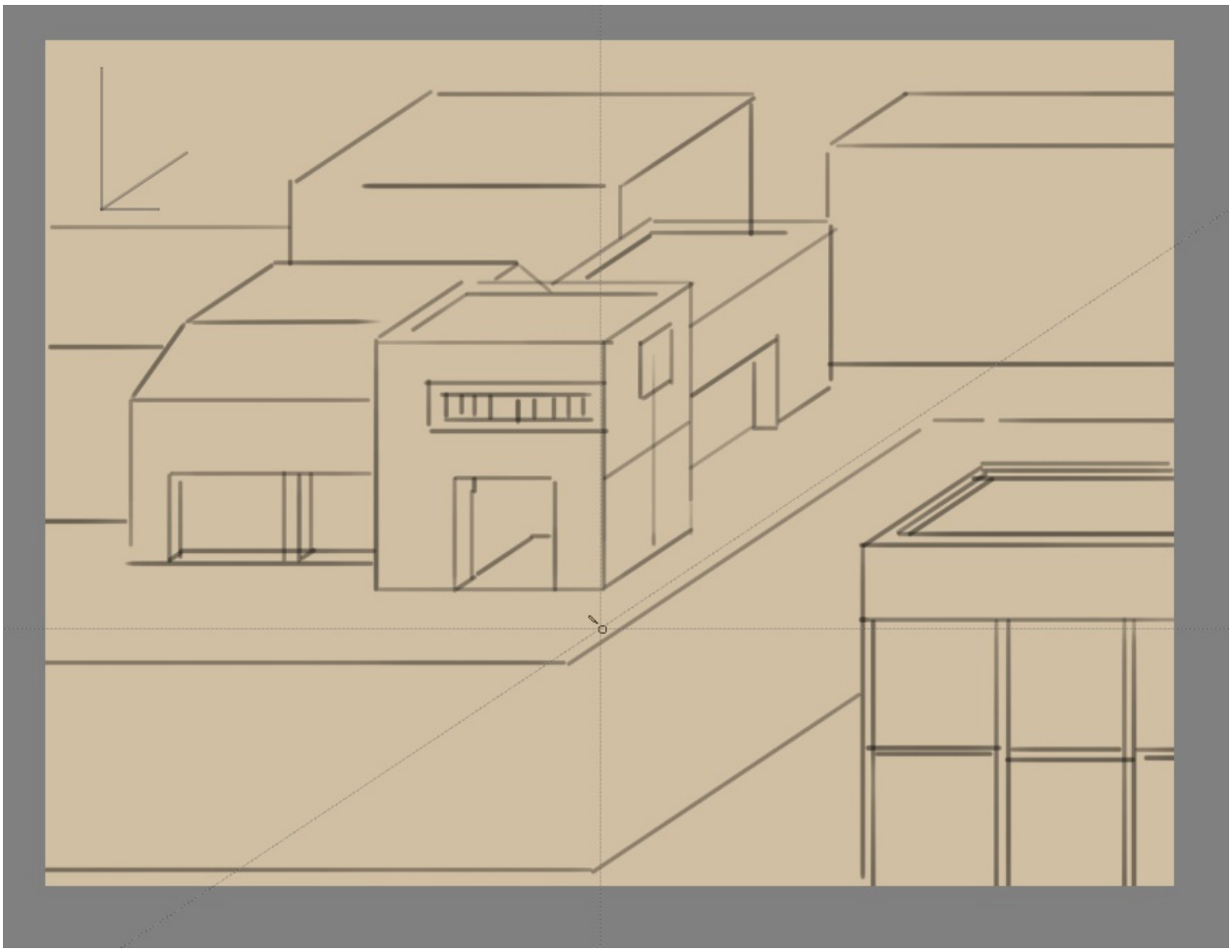
Orthographic

Orthographic is a mode where you try to look at something from the left or the front. Typically, you try to keep everything in exact scale with each other, unlike perspective deformation.

The key assistant you want to use here is the Parallel Ruler. You can set these up horizontally or vertically, so you always have access to a Grid.

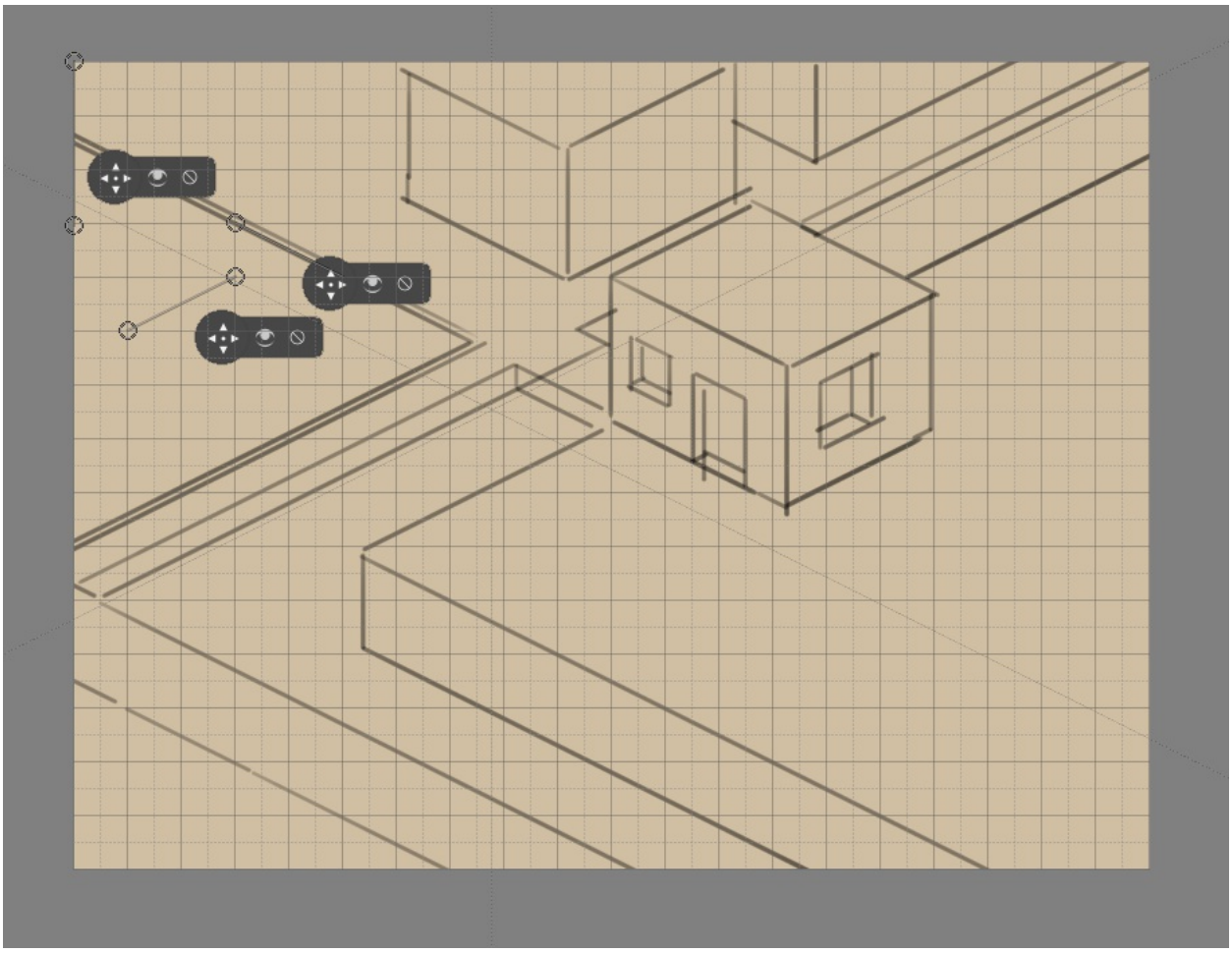
Axonometric

All of these are set up using three Parallel Rulers.



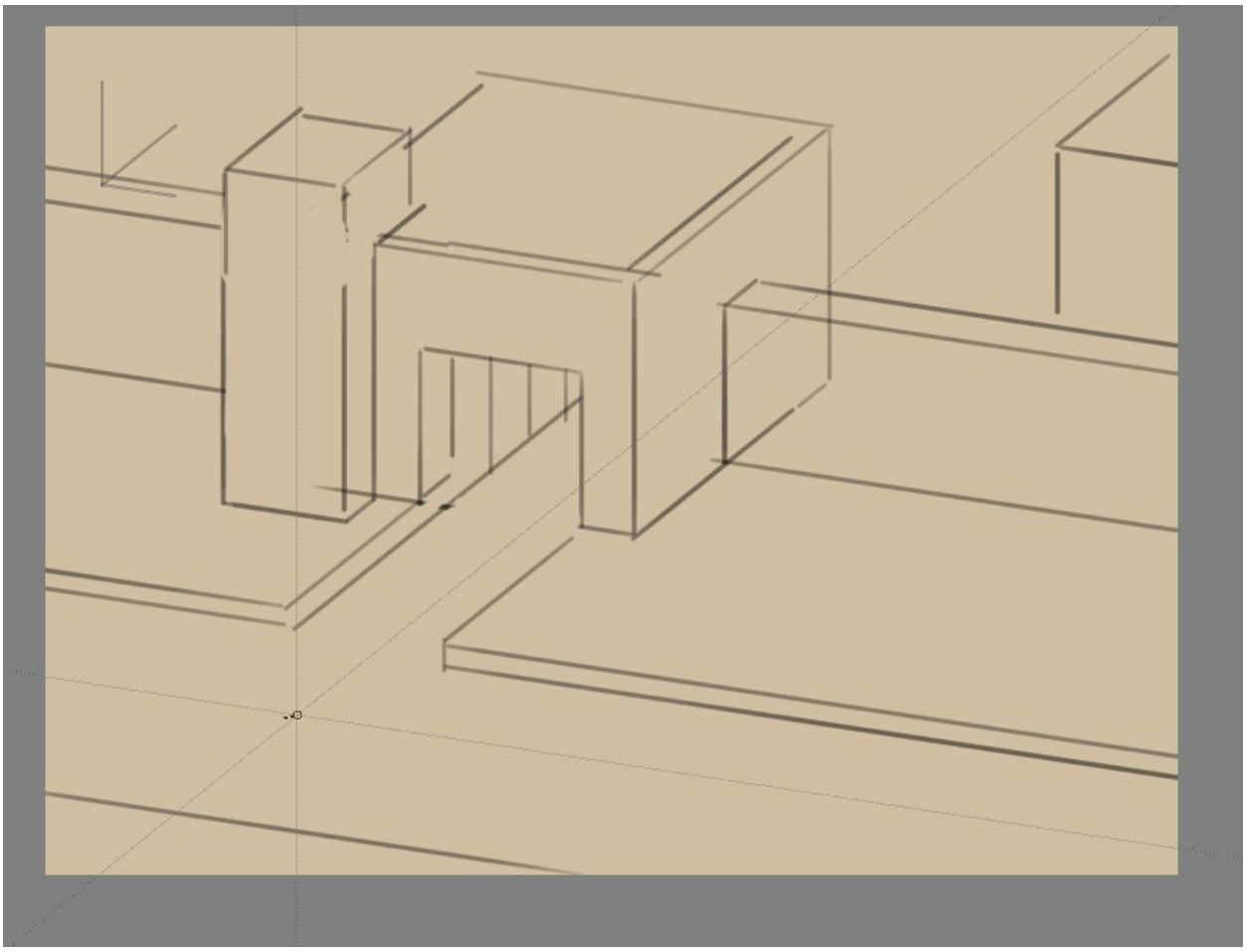
Oblique

For oblique, set two parallel rulers to horizontal and vertical, and one to an angle, representing depth.



Dimetric & Isometric

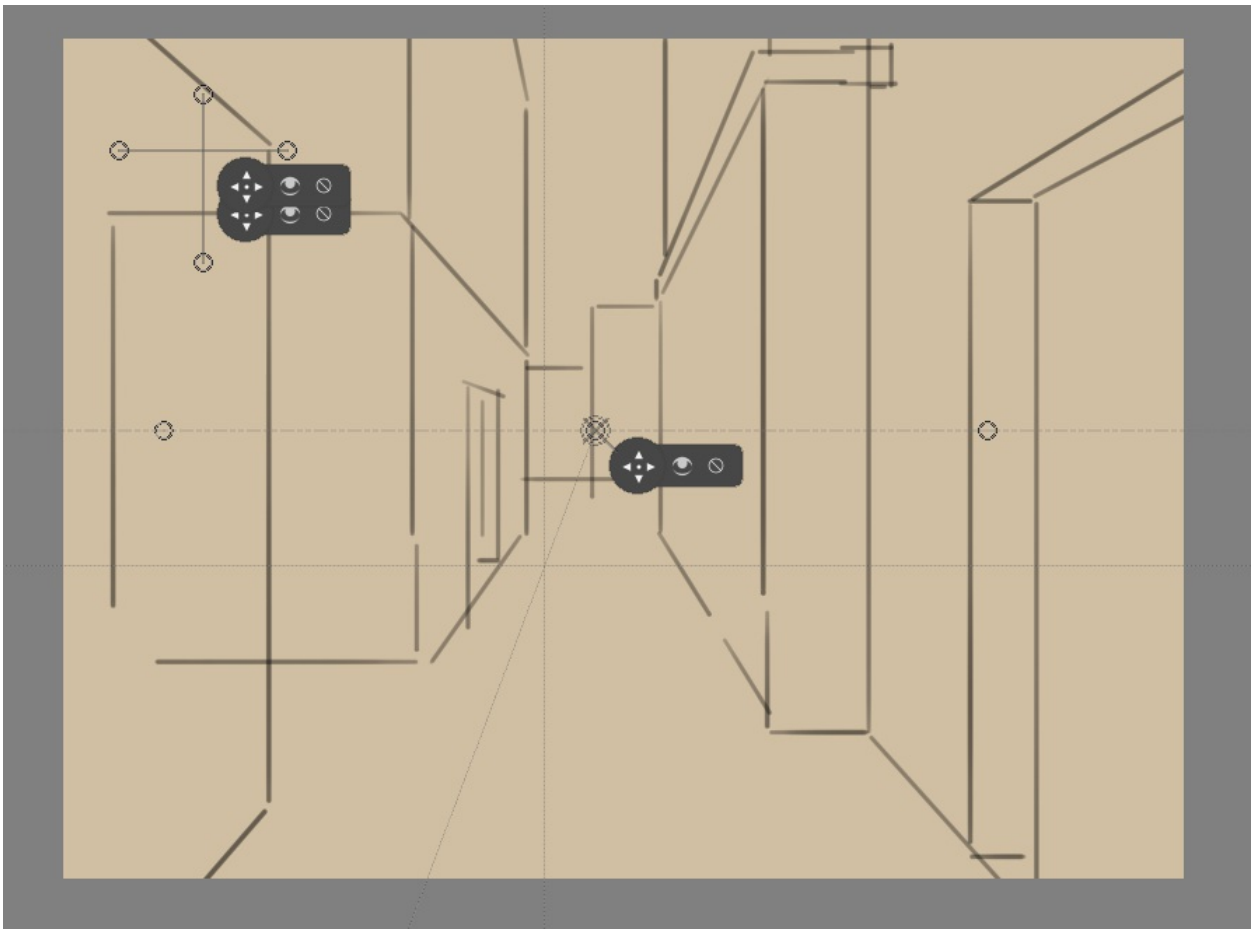
Isometric perspective has technically all three rulers set up at 120° from each other. Except when it’s game isometric, then it’s a type of dimetric projection where the diagonal values are a 116.565° from the main. The latter can be easily set up by snapping the assistants to a grid.



Trimetric

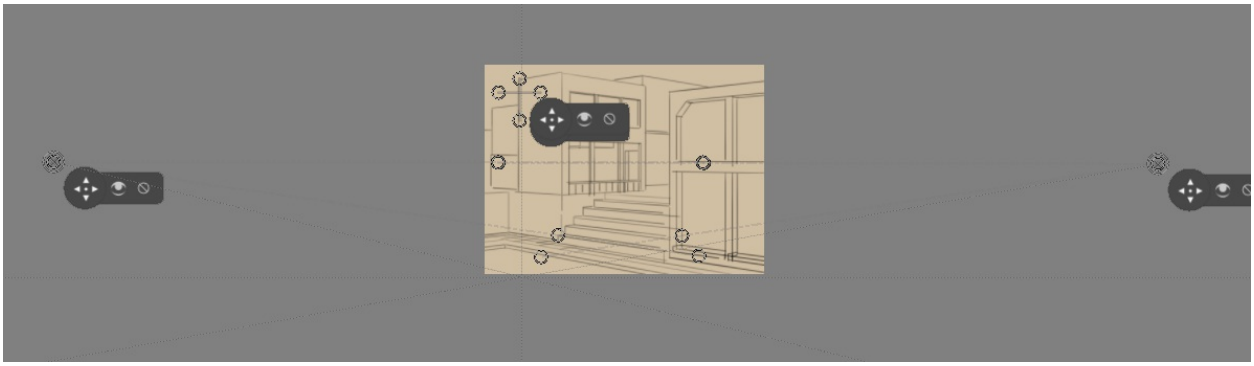
Is when all the angles are slightly different. Often looks like a slightly angled isometric.

Linear Perspective



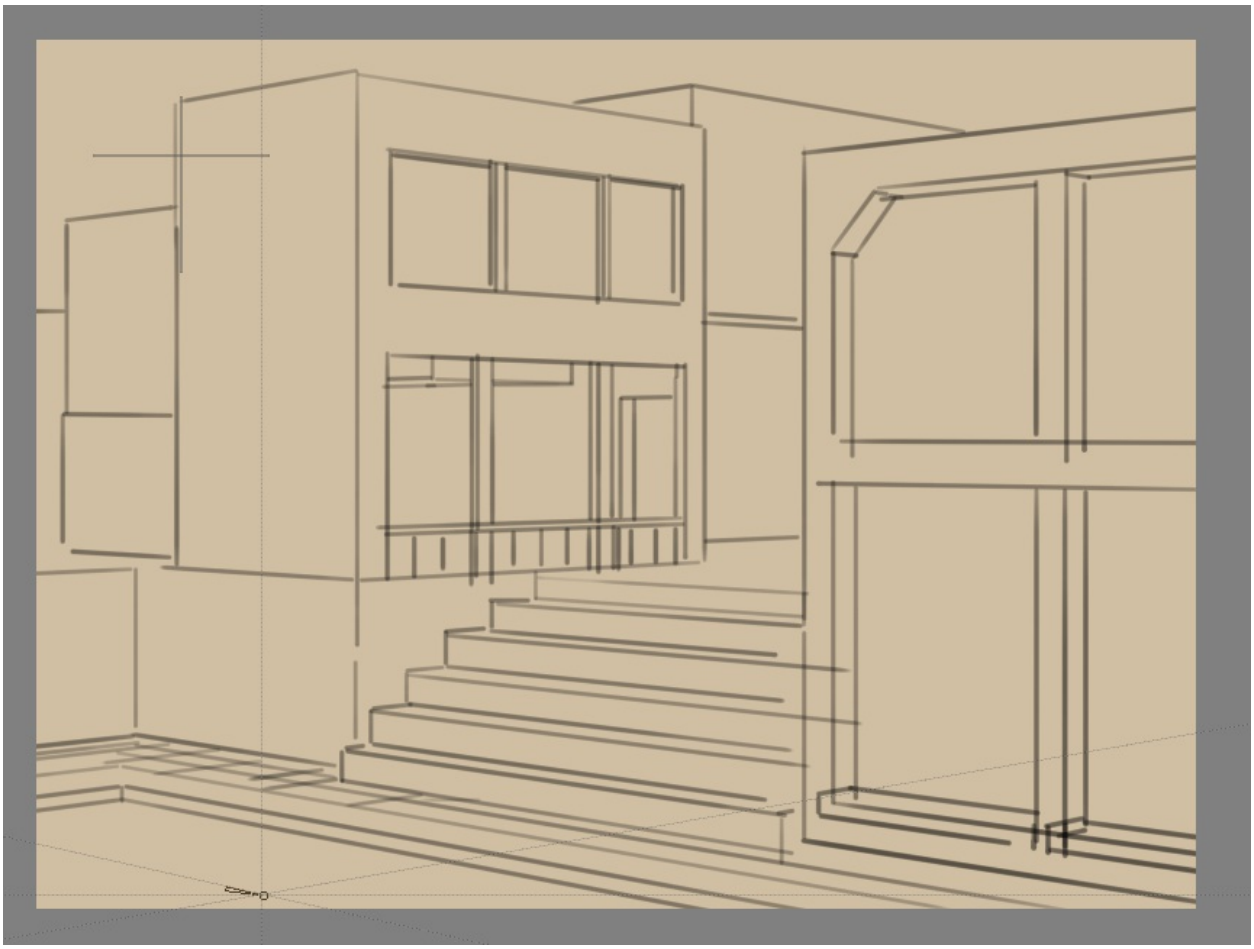
1 Point Perspective

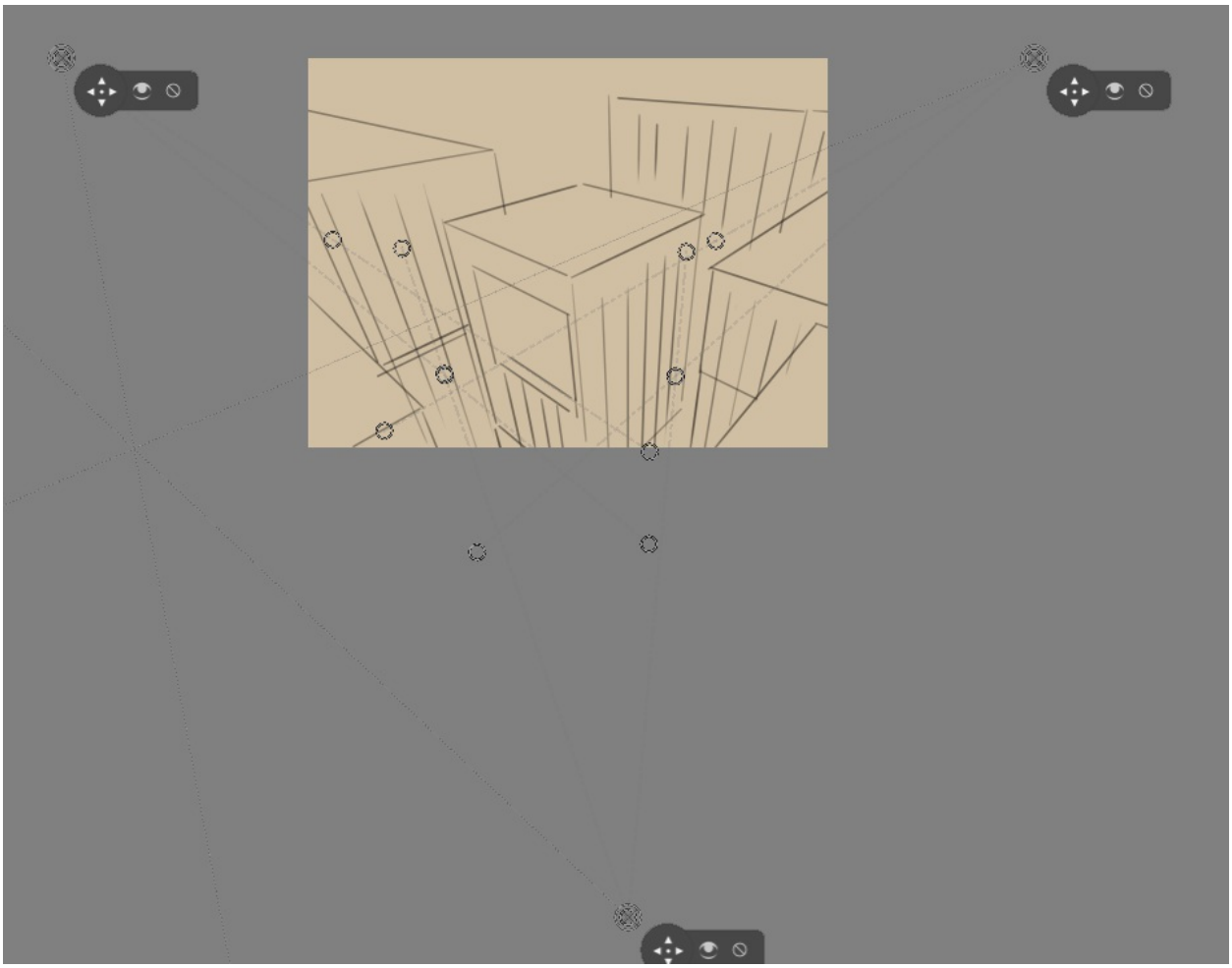
A 1 point perspective is set up using 1 vanishing point, and two crossing perpendicular parallel rulers.



2 Point Perspective

A 2 point perspective is set up using 2 vanishing point and 1 vertical parallel ruler. Often, putting the vanishing points outside the frame a little can decrease the strength of it.





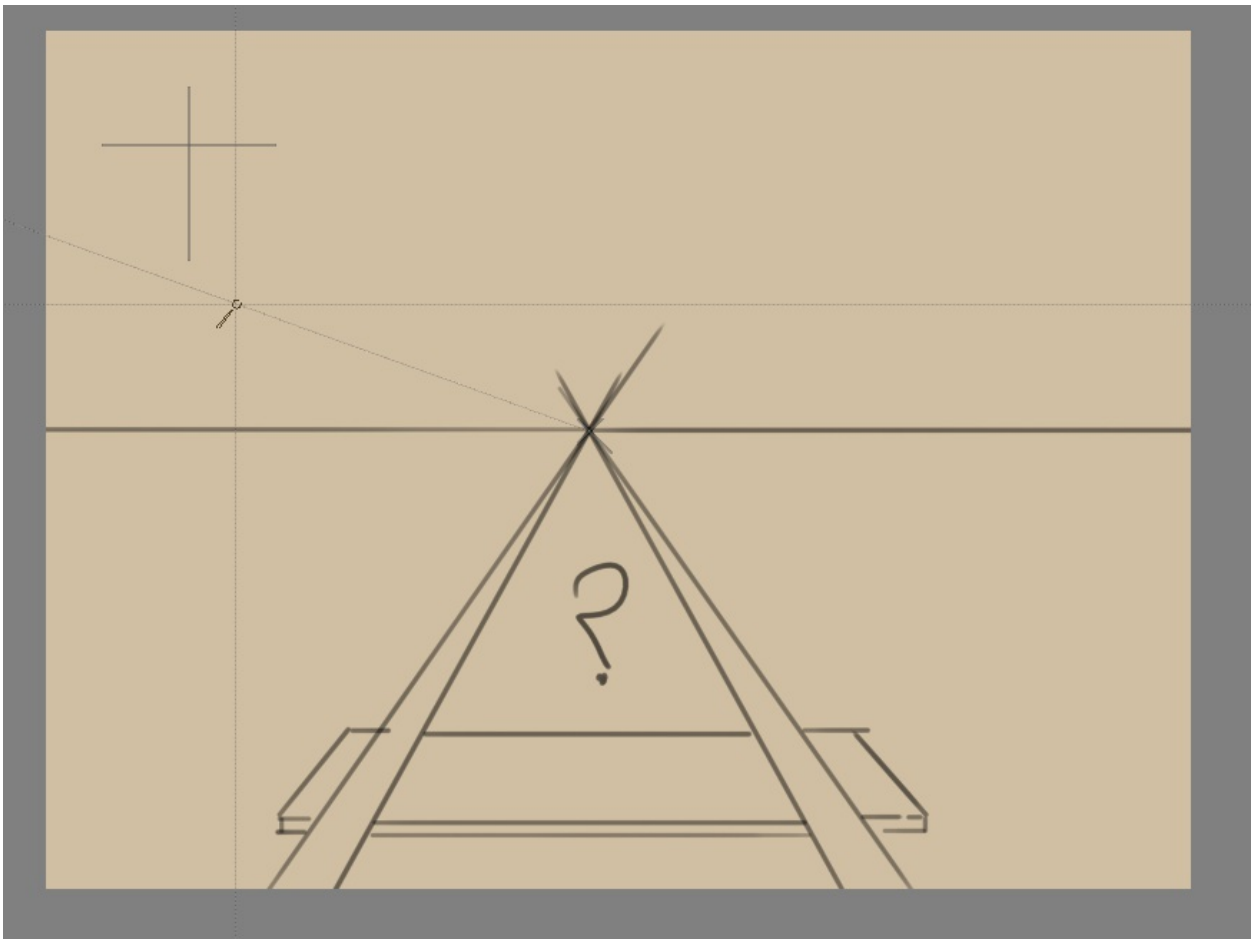
3 Point Perspective

A 3 point perspective is set up using 3 vanishing point rulers.

Logic of the vanishing point

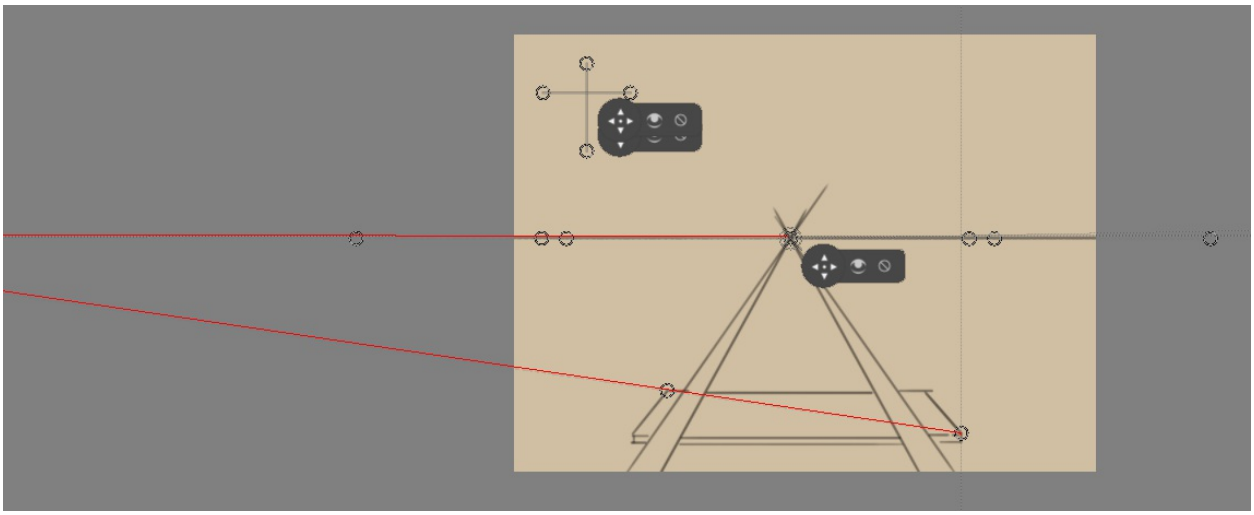
There’s a little secret that perspective tutorials don’t always tell you, and that’s that a vanishing point is the point where any two parallel lines meet. This means that a 1 point perspective and 2 point perspective are virtually the same.

We can prove this via a little experiment. That good old problem: drawing a rail-road.



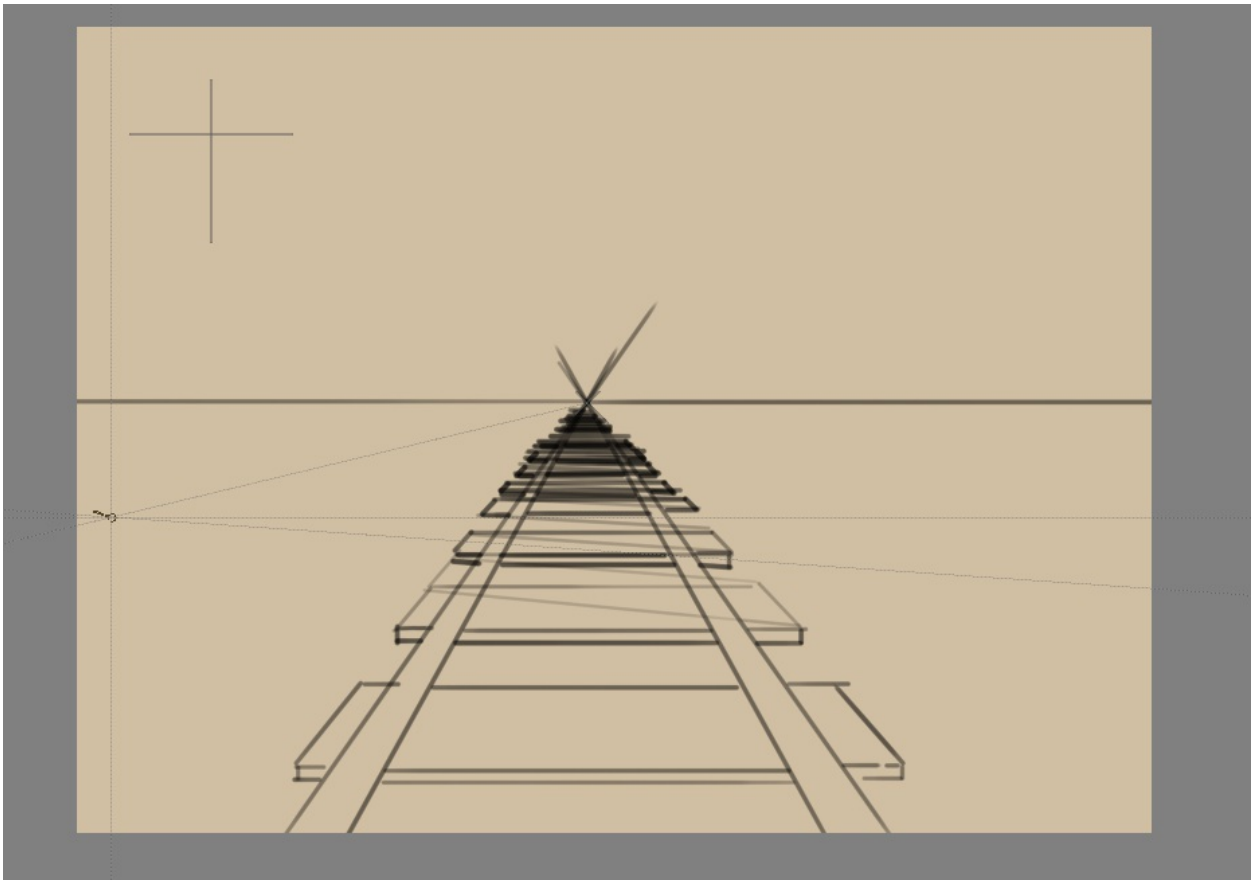
You are probably familiar with the problem: How to determine where the next beam is going to be, as perspective projection will make them look closer together.

Typically, the solution is to draw a line in the middle and then draw lines diagonally across. After all, those lines are parallel, meaning that the exact same distance is used.



But because they are parallel, we can use a vanishing point assistant instead, and we use the alignment handles to align it to the diagonal of the beam, and to the horizontal (here marked with red).

That diagonal can then in turn be used to determine the position of the beams:

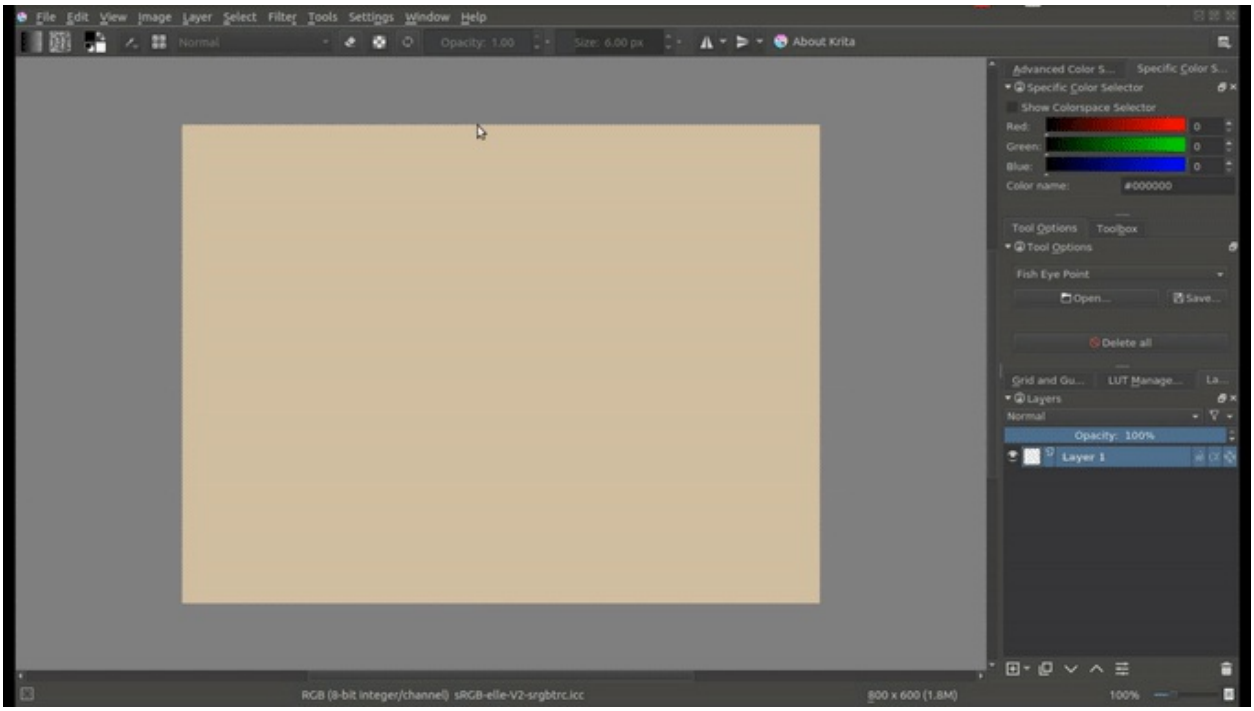


Because any given set of lines has a vanishing point (outside of the ones flat on the view-plane), there can be an infinite amount of vanishing points in a linear perspective. Therefore, Krita allows you to set vanishing points yourself instead of forcing you to only use a few.

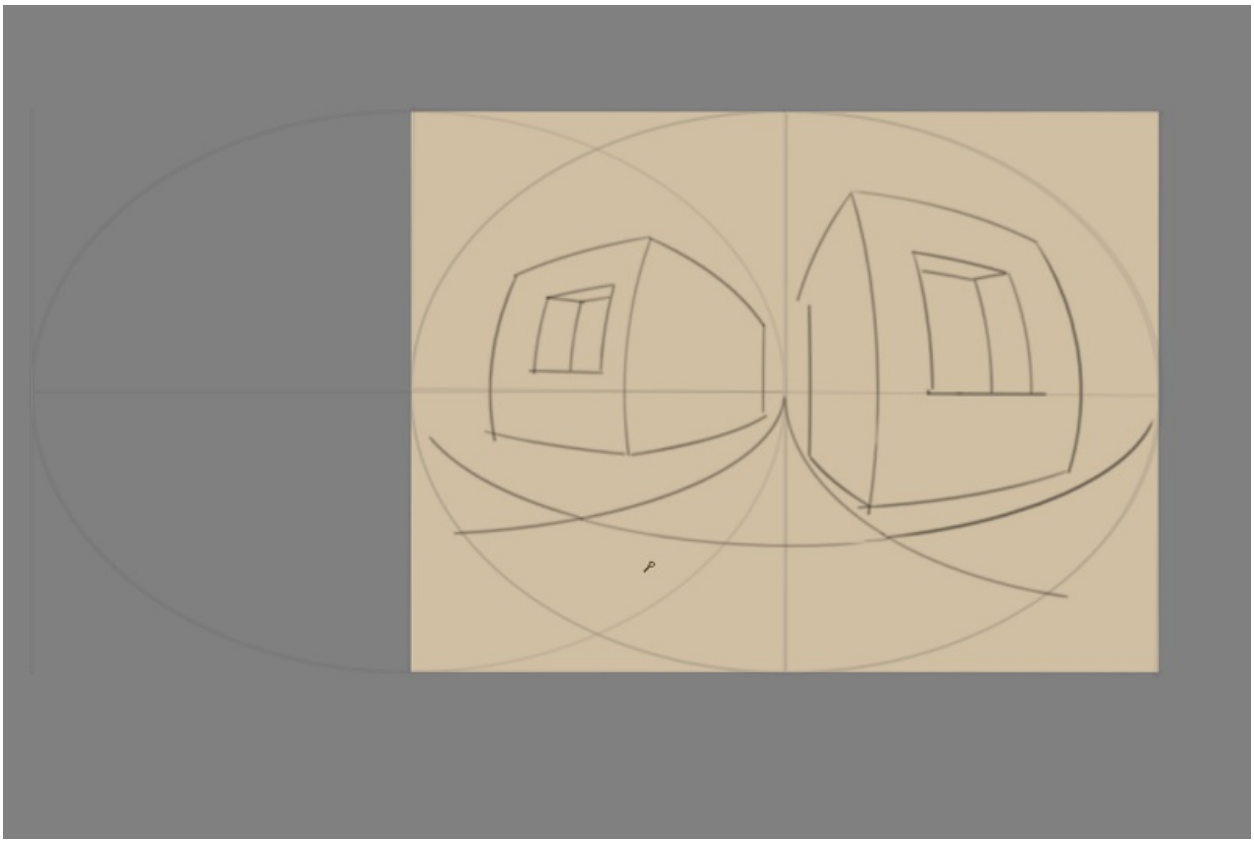
Fish Eye perspective

Fish eye perspective works much the same as the linear perspective, the big difference being that in a fish-eye perspective, any parallel set of lines has two vanishing points, each for one side.

So, to set them up, the easiest way is one horizontal, one vertical, on the same spot, and one vanishing point assistant in the middle.



But, you can also make one horizontal one that is just as big as the other horizontal one, and put it halfway:



Working with Images

Computers work with files and as a painting program, Krita works with images as the type of file it creates and manipulates.

What do Images Contain?

If you have a text document, it of course contains letters, strung in the right order, so the computer loads them as coherent sentences.

Raster Data

This is the main data on the paint layers you make. So these are the strokes with the paint brush and look pixely up close. A multi-layer file will contain several of such layers, that get overlaid on top of each other so make the final image.

A single layer file will usually only contain raster data.

Vector Data

These are mathematical operations that tell the computer to draw pixels on a spot. This makes them much more scalable, because you just tell the operation to make the coordinates 4 times bigger to scale it up. Due to this vector data is much more editable, lighter, but at the same time it’s also much more CPU intensive.

Operation Data

Stuff like the filter layers, that tells Krita to change the colors of a layer, but also transparency masks, group layer and transformation masks are saved to multi-layer files. Being able to load these depend on the software that initially made the file. So Krita can load and save groups, transparency masks and layer effects from PSD, but not load or save transform masks.

Metadata

Metadata is information like the creation date, author, description and also information like DPI.

Image size

The image size is the dimension and resolution of the canvas. Image size has direct effect file size of the Krita document. The more pixels that need to be remembered and the higher the bit depth of the color, the heavier the resulting file will be.

DPI/PPI

DPI stands for *Dots per Inch*, **PPI** stands for *Pixels per Inch*. In printing industry, suppose if your printer prints at 300 **DPI**. It means it is actually putting 300 dots of colors in an area equal to an Inch. This means the number of pixels your artwork has in a relative area of an inch.

DPI is the concern of the printer, and artists while creating artwork should keep **PPI** in mind. According to the **PPI** you have set, the printers can decide how large your image should be on a piece of paper.

Some standards:

72 PPI
: This is the default PPI of monitors as assumed by all programs. It is not fully correct, as most monitors these days have 125 PPI or even 300 PPI for the retina devices. None the less, when making an image for computer consumption, this is the default.

120 PPI
: This is often used as a standard for low-quality posters.

300 PPI
: This is the minimum you should use for quality prints.

600 PPI
: The quality used for line art for comics.

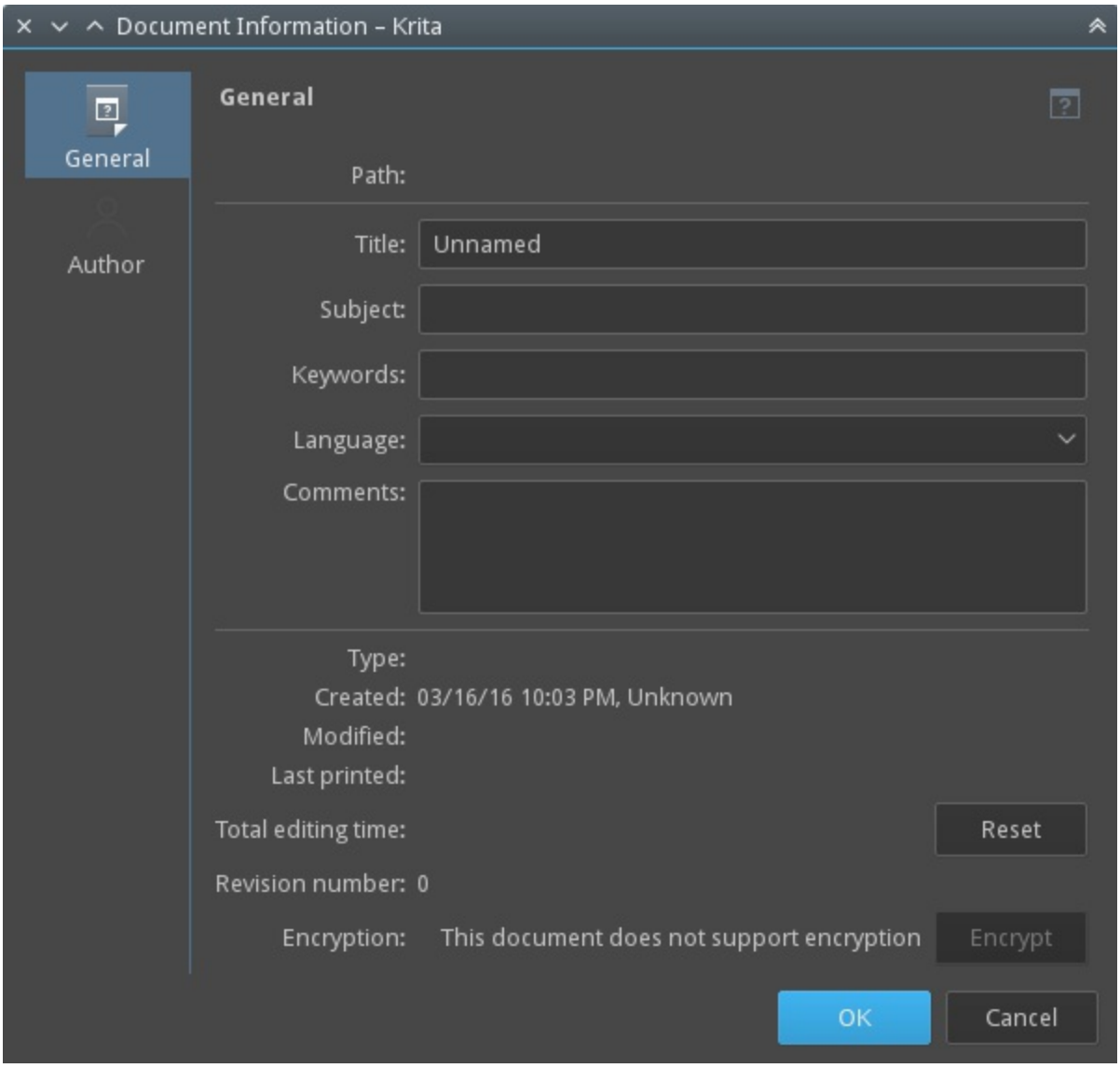
Color depth

We went over color depth in the Color Management page. What you need to understand is that Krita has image color spaces, and layer color spaces, the latter which can save memory if used right. For example, having a line art layer in grayscale can half the memory costs.

Image color space vs layer color space vs conversion.

Because there's a difference between image color space and layer color space, you can change only the image color space in *Image* ▸ *Properties…* which will leave the layers alone. But if you want to change the color space of the file including all the layers you can do it by going to *Image* ▸ *Convert Image Color Space…* this will convert all the layers color space as well.

Author and Description



Krita will automatically save who created the image into your image's metadata. Along with the other data such as time and date of creation and modification, Krita also shows editing time of a document in the document information dialog, useful for professional illustrators, speed-painters to keep track of the time they worked on artwork for billing purposes. It detects when you haven't performed actions for a while, and has a precision of ±60 seconds. You can empty it in the document info dialog and of course by unzipping you `.kra` file and editing the metadata there.

These things can be edited in *File* ▸ *Document Information*, and for the author's information *Settings* ▸ *Configure Krita…* ▸ *Author*. Profiles can be switched under *Settings* ▸ *Active Author Profile*.

Setting the canvas background color

You can set the canvas background color via *Image* ▸ *Image Background Color and Transparency…* menu item. This allows you to turn the background color non-transparent and to change the color. This is also useful for certain file formats which force a background color instead of transparency. PNG and JPG export use this color as the default color to fill in transparency if you do not want to export transparency.

If you come in from a program like **Paint Tool Sai**, then using this option, or using *As canvas color* radio button at *Background:* section in the new file options, will allow you to work in a slightly more comfortable environment, where transparency isn't depicted with checkered boxes.

Basic transforms

There are some basic transforms available in the Image menu.

Shear Image…
: This will allow you to skew the whole image and its layers.

Rotate
: This show a submenu that will allow you to rotate the image and all its layers quickly.

Mirror Image Horizontally/Vertically
: This will allow you to mirror the whole image with all its layers.

But there are more options than that…

Cropping and resizing the canvas

You can crop and image with the Crop Tool, to cut away extra space and improve the composition.

Trimming

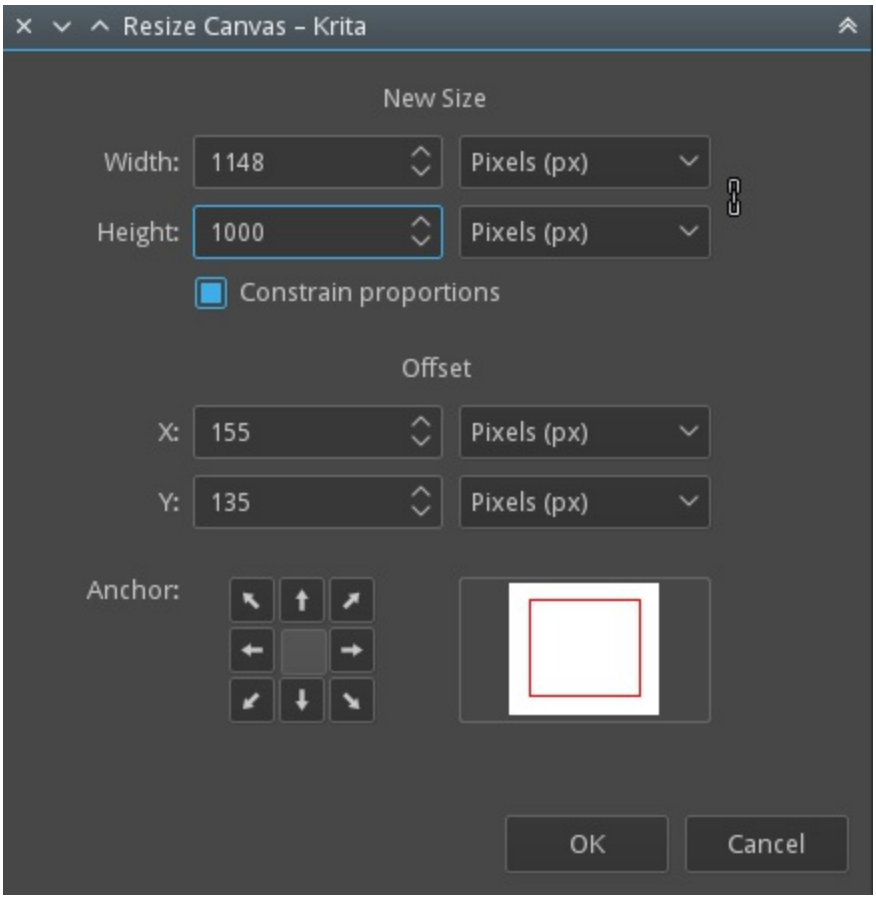
Using *Image* ▸ *Trim to Current Layer*, Krita resizes the image to the dimensions of the layer selected. Useful for when you paste a too large image into the layer and want to resize the canvas to the extent of this layer.

Image ▸ *Trim to Selection* is a faster cousin to the crop tool. This helps us to resize the canvas to the dimension of any active selection. This is especially useful with right clicking the layer on the layer stack and choosing *Select Opaque*. *Image* ▸ *Trim to Selection* will then crop the canvas to the selection bounding box.

Image ▸ *Trim to Image Size* is actually for layers, and will trim all layers to the size of the image, making your files lighter by getting rid of invisible data.

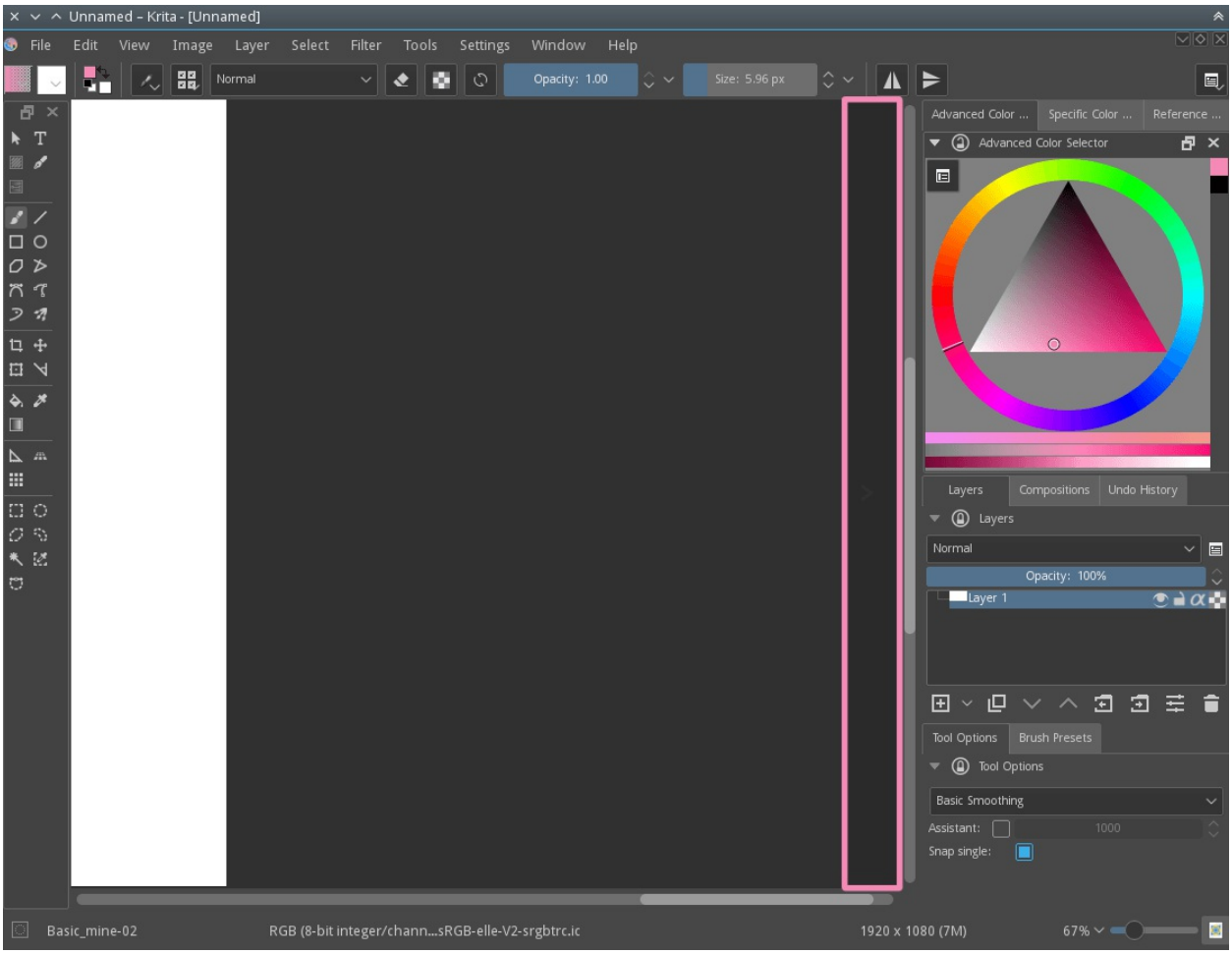
Resizing the canvas

You can also resize the canvas via *Image* ▸ *Resize Canvas…* (or the `Ctrl + Alt + C` shortcut). The dialog box is shown below.



In this, *Constrain proportions* checkbox will make sure the height and width stay in proportion to each other as you change them. Offset indicates where the new canvas space is added around the current image. You basically decide where the current image goes (if you press the left-button, it'll go to the center left, and the new canvas space will be added to the right of the image).

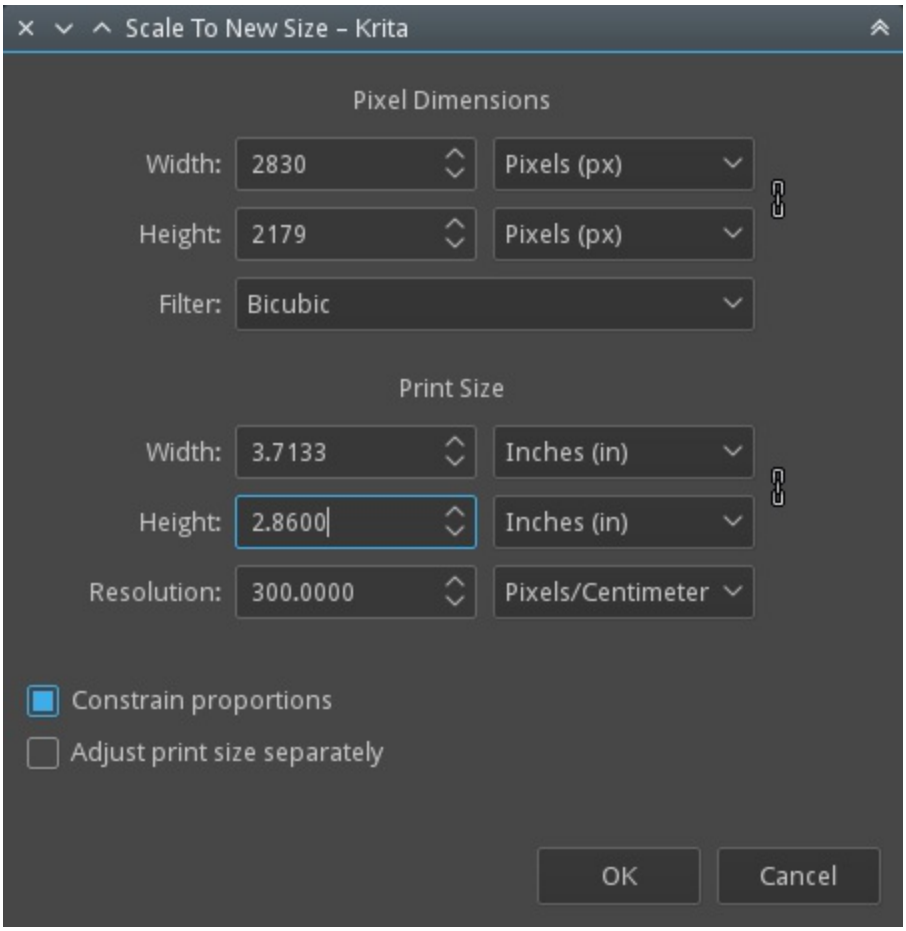
Another way to resize the canvas according to the need while drawing is when you scroll away from the end of the canvas, you can see a strip with an arrow appear. Clicking this will extend the canvas in that direction. You can see the arrow marked in red in the example below:



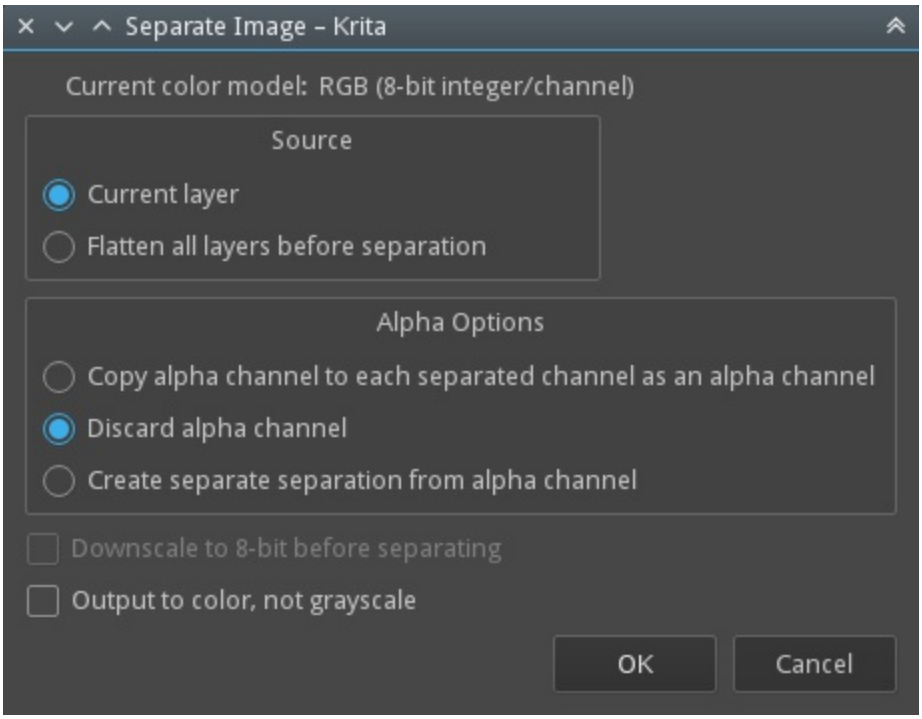
Resizing the image

Scale Image to New Size… allows you to resize the whole image. Also, importantly, this is where you can change the resolution or *upres* your image. So for instance, if you were initially working at 72 PPI to block in large shapes and colors, images, etc… And now you want to really get in and do some detail work at 300 or 400 PPI this is where you would make the change.

Like all other dialogs where a chain link appears, when the chain is linked the aspect ratio is maintained. To disconnect the chain, just click on the link and the two halves will separate.



Separating Images



This powerful image manipulation feature lets you separate an image into its different components or channels.

This is useful for people working in print, or people manipulating game textures. There's no combine functionality, but what you can do, if using colored output, is to set two of the channels to the addition Blending Modes.

For grayscale images in the RGB space, you can use the Copy Red, Copy Green and Copy Blue blending modes, with using the red one for the red channel image, etc.

Saving, Exporting and Opening Files

When Krita creates or opens a file, it has a copy of the file in memory, that it edits. This is part of the way how computers work: They make a copy of their file in the RAM. Thus, when saving, Krita takes its copy and copies it over the existing file. There's a couple of tricks you can do with saving.

Save
: Krita saves the current image in its memory to a defined place on the hard-drive. If the image hadn't been saved before, Krita will ask you

where to save it.

Save As…
: Make a copy of your current file by saving it with a different name. Krita will switch to the newly made file as its active document.

Open…
: Open a saved file. Fairly straightforward.

Export…
: Save a file to a new location without actively opening it. Useful for when you are working on a layered file, but only need to save a flattened version of it to a certain location.

Open Existing Document as Untitled Document…
: This is a bit of an odd one, but it opens a file, and forgets where you saved it to, so that when pressing ‘save’ it asks you where to save it. This is also called ‘import’ in other programs.

Create Copy from Current Image
: Makes a new copy of the current image. Similar to *Open Existing Document as Untitled Document…*, but then with already opened files.

Save Incremental Version
: Saves the current image as *‘filename’_XXX.kra* and switches the current document to it.

Save Incremental Backup
: Copies and renames the last saved version of your file to a back-up file and saves your document under the original name.

Note

Since Krita’s file format is compressed data file, in case of a corrupt or broken file you can open it with archive managers and extract the contents of the layers. This will help you to recover as much as possible data from the file. On Windows, you will need to rename it to *filename.zip* to open it.

Saving, AutoSave and Backup Files

Krita does its best to keep your work safe. But if you want to make sure that you won't lose work, you will need to understand how Saving, AutoSave and Backup Files work in Krita.

Saving

Krita does not store your images somewhere without your intervention. You need to save your work, or it will be lost, irretrievably. Krita can save your images in many formats. You should always save your work in Krita's native format, `.kra` because that supports all Krita's features.

Additionally, you can export your work to other formats, for compatibility with other applications or publication on the Web or on paper. Krita will warn which aspects of your work are going to be lost when you save to another format than `.kra` and offers to make a `.kra` file for you as well.

If you save your work, Krita will ask you where it should save on your computer. By default, this is the Pictures folder in your User folder: this is true for all operating systems.

If you use *Save As…* your image will be saved under a new name. The original file under its own name will not be deleted. From now on, your file will be saved under the new name.

If you use *Export…* using a new filename, a new file will be created with a new name. The file you have open will keep the old name, and the next time you save it, it will be saved under the old name.

You can *Save*, *Save As…* and *Export…* to any file format.

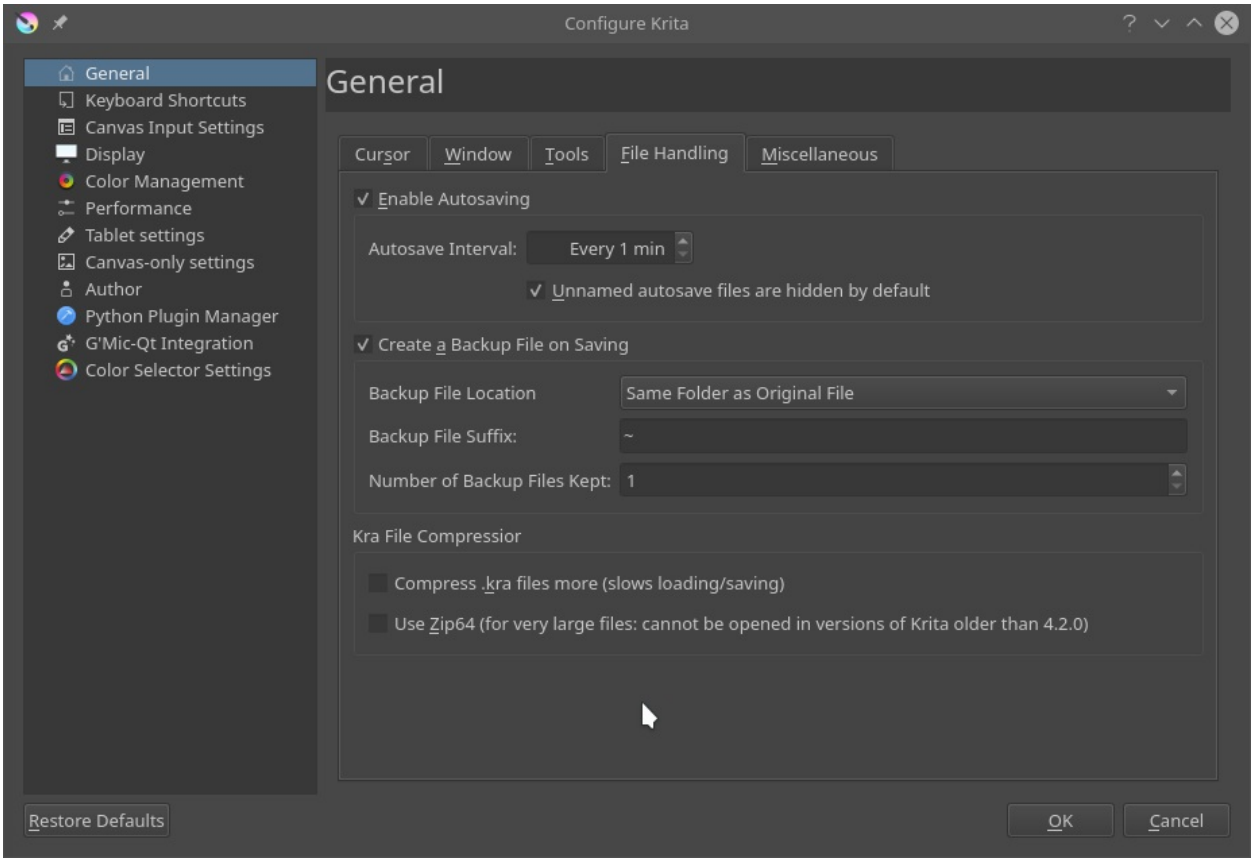
See also

Saving for the Web

AutoSave

AutoSave is what happens when you've worked for a bit and not saved your work yourself: Krita will save your work for you. Autosave files are by default hidden in your file manager. You can configure Krita 4.2 and up to create autosave files that are visible in your file manager. By default, Krita autosaves every fifteen minutes; you can configure that in the File tab of the General Settings page of the *Configure Krita* dialog, which is in the Settings menu (Linux, Windows) or in the Application menu (macOS).

If you close Krita without saving, your unsaved work is lost and cannot be retrieved. Closing Krita normally also means that autosave files are removed.



There are two possibilities:

- You hadn't saved your work at all
- You had saved your work already

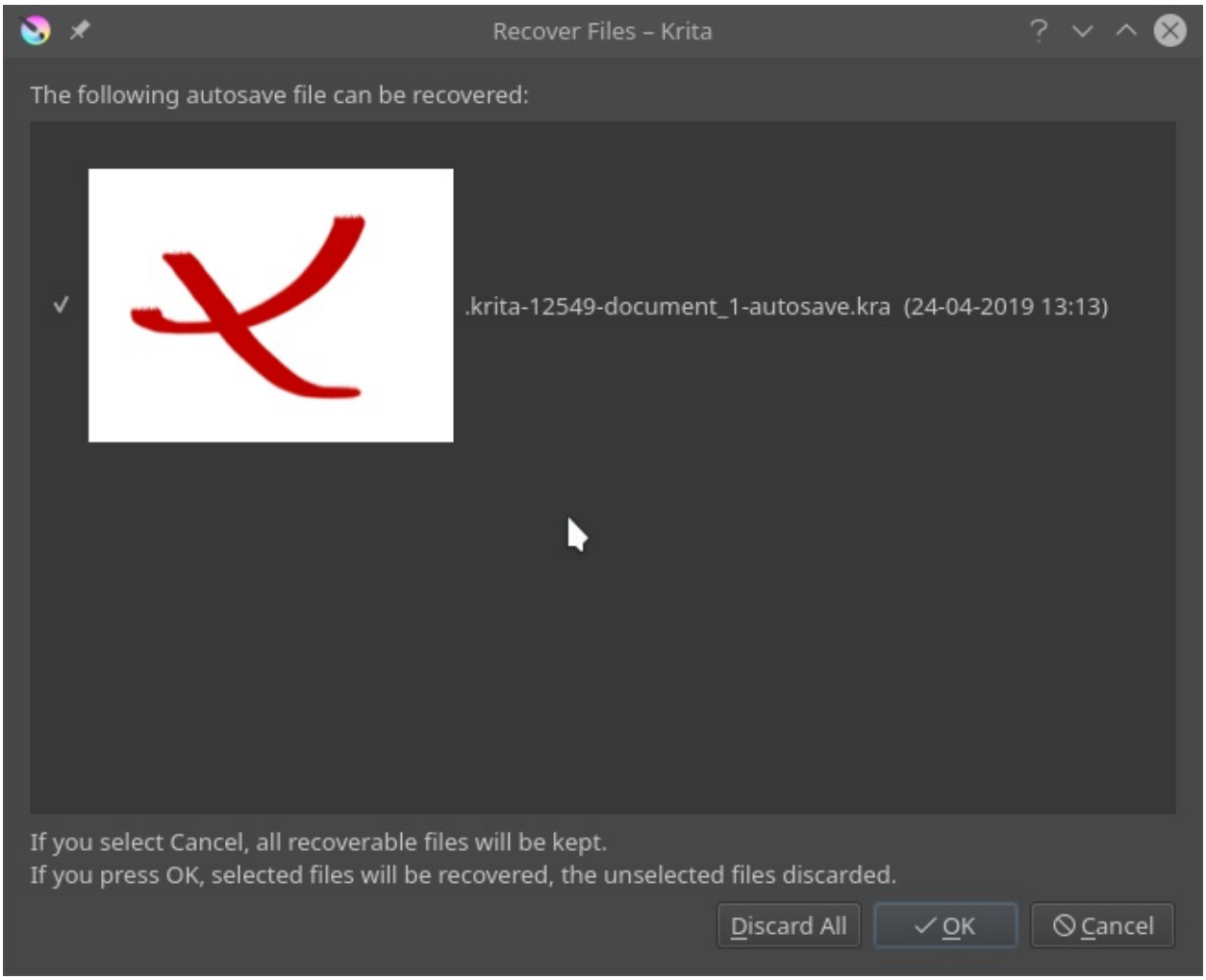
AutoSave for Unsaved Files

If you had not yet saved your work, Krita will create an unnamed AutoSave file.

If you're using Linux or macOS, the AutoSave file will be a hidden file in your home directory. If you're using Windows, the AutoSave file will be a file in your user's `%TEMP%` folder. In Krita 4.2 and up, you can configure Krita to make the AutoSave files visible by default.

A hidden autosave file will be named like `.krita-12549-document_1-autosave.kra`

If Krita crashes before you had saved your file, then the next time you start Krita, you will see the file in a dialog that shows up as soon as Krita starts. You can select to restore the files, or to delete them.



If Krita crashed, and you're on Windows and your `%TEMP%` folder gets cleared, you will have lost your work. Windows does not clear the `%TEMP%` folder by default, but you can enable this feature in Settings. Applications like Disk Cleanup or cCleaner will also clear the `%TEMP%` folder. Again, if Krita crashes, and you haven't saved your work, and you have something enabled that clear your `%TEMP%` folder, you will have lost your work.

If Krita doesn't crash, and you close Krita without saving your work, Krita will remove the AutoSave file: your work will be gone and cannot be retrieved.

If you save your work and continue, or close Krita and do save your work, the AutoSave file will be removed.

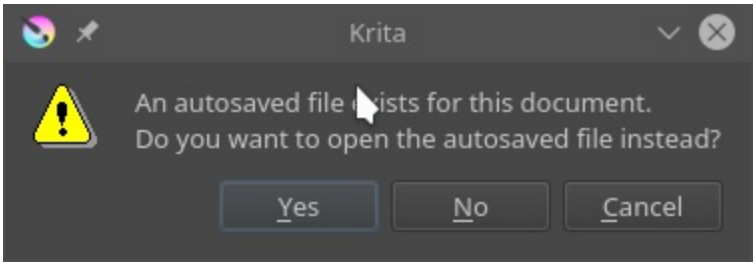
AutoSave for Saved Files

If you had already saved your work, Krita will create a named AutoSave file.

A hidden named autosave file will look like `.myimage.kra-autosave.kra`.

By default, named AutoSave files are hidden. Named AutoSave files are placed in the same folder as the file you were working on.

If you start Krita again after it crashed and try to open your original file, Krita will ask you whether to open the AutoSave file instead:



If you choose “no”, the AutoSave file will be removed. The work that has been done since the last time you saved your file yourself will be lost and cannot be retrieved.

If you choose “yes”, the AutoSave file will be opened, then removed. The file you have open will have the name of your original file. The file will be set to Modified, so the next time you try to close Krita, Krita will ask you whether you want to save the file. If you choose No, your work is irretrievably gone. It cannot be restored.

If you use *Save As…* your image will be saved under a new name. The original file under its own name and its AutoSave file are not deleted. From now on, your file will be saved under the new name; if you save again, an AutoSave file will be created using the new filename.

If you use *Export…* using a new filename, a new file will be created with a new name. The file you have open will keep the new name, and the next time you save it, the AutoSave file will be created from the last file saved with the current name, that is, not the name you choose for *Export…*.

Backup Files

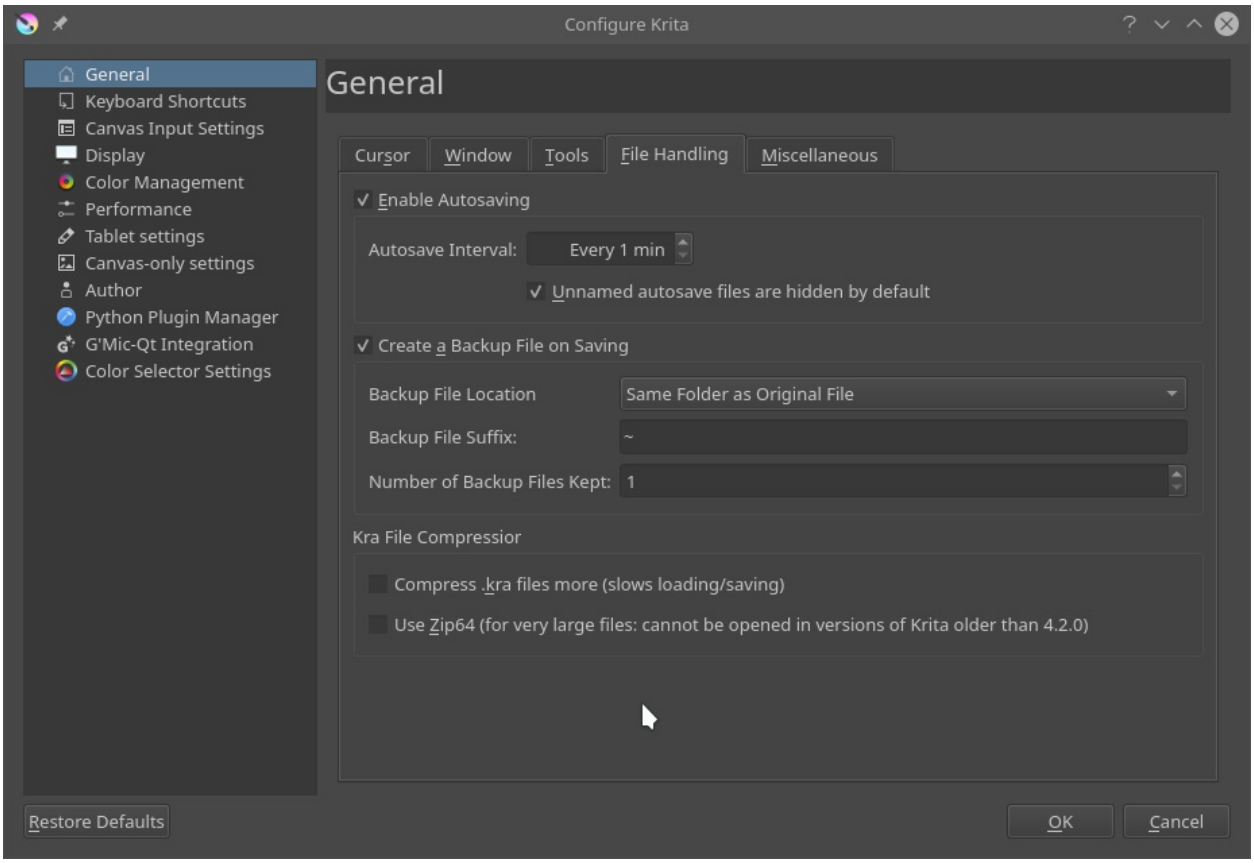
There are three kinds of Backup files

- Ordinary Backup files that are created when you save a file that has been opened from disk
- Incremental Backup files that are copies of the file as it is on disk to a numbered backup, and while your file is saved under the current name
- Incremental Version files that are saves of the file you are working on with a new number, leaving alone the existing files on disk.

Ordinary Backup Files

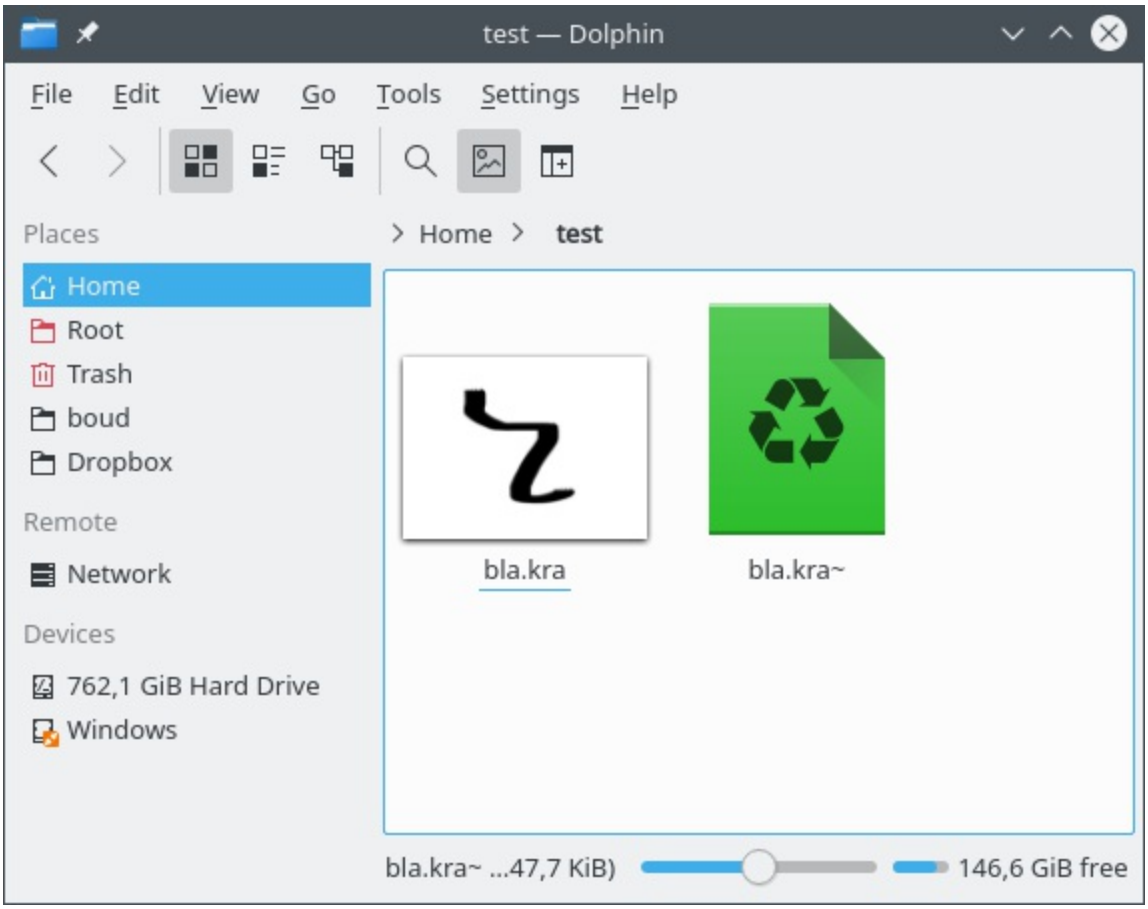
If you have opened a file, made changes, then save it, or save a new file after the first time you've saved it, Krita will save a backup of your file.

You can disable this mechanism in the File tab of the General Settings page of the *Configure Krita* dialog, which is in the Settings menu (Linux, Windows) or in the Application menu (macOS). By default, Backup files are enabled.



By default, a Backup file will be in the same folder as your original file. You can also choose to save Backup files in the User folder or the `%TEMP%` folder; this is not as safe because if you edit two files with the same name in two different folders, their backups will overwrite each other.

By default, a Backup file will have ~ as a suffix, to distinguish it from an ordinary file. If you are using Windows, you will have to enable “show file extensions” in Windows Explorer to see the extension.

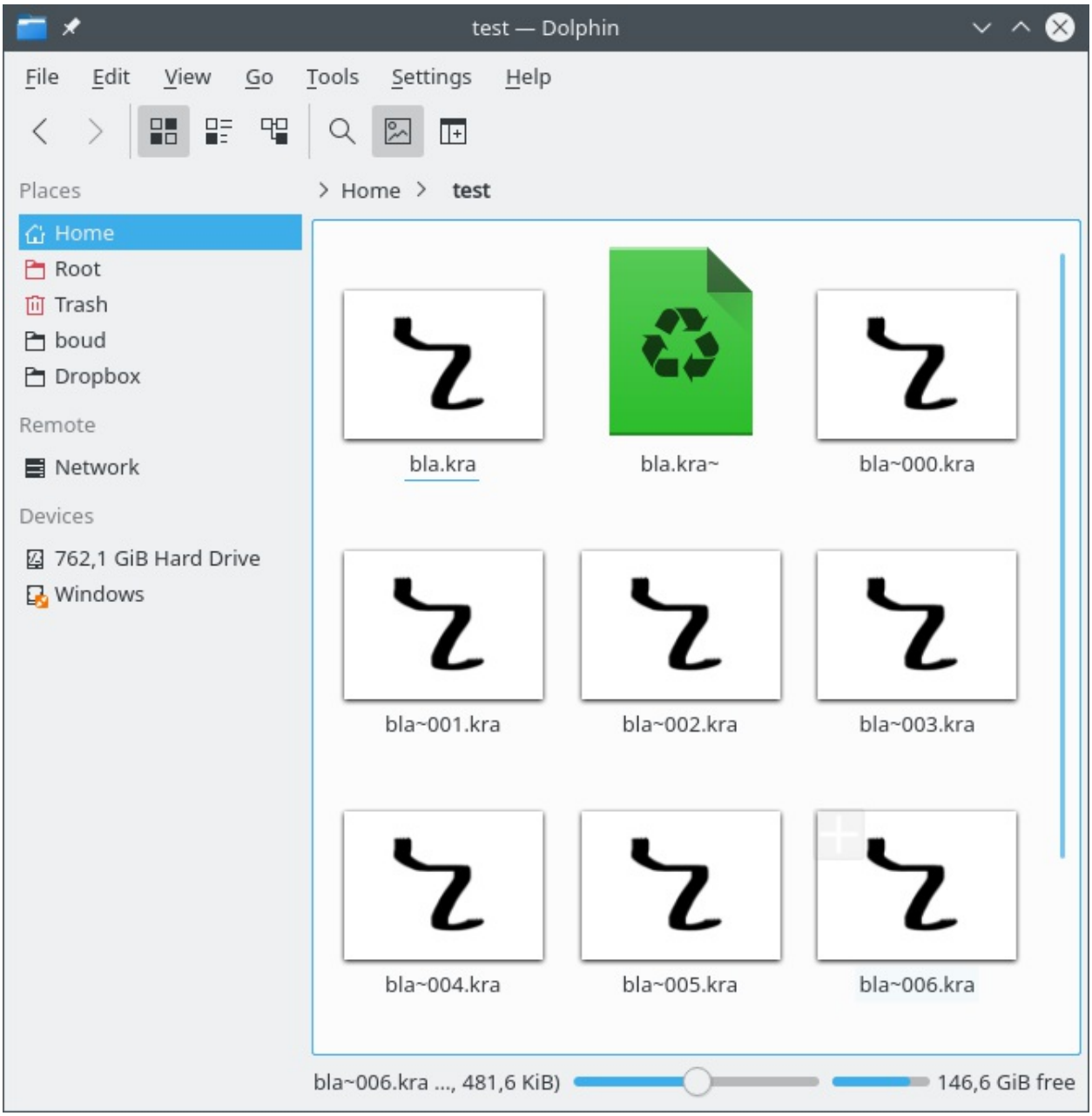


If you want to open the Backup file, you will have to rename it in your file manager. Make sure the extension ends with `.kra`.

Every time you save your file, the last version without a ~ suffix will be copied to the version with the ~ suffix. The contents of the original file will be gone: it will not be possible to restore that version.

Incremental Backup Files

Incremental Backup files are similar to ordinary Backup files: the last saved state is copied to another file just before saving. However, instead of overwriting the Backup file, the Backup files are numbered:

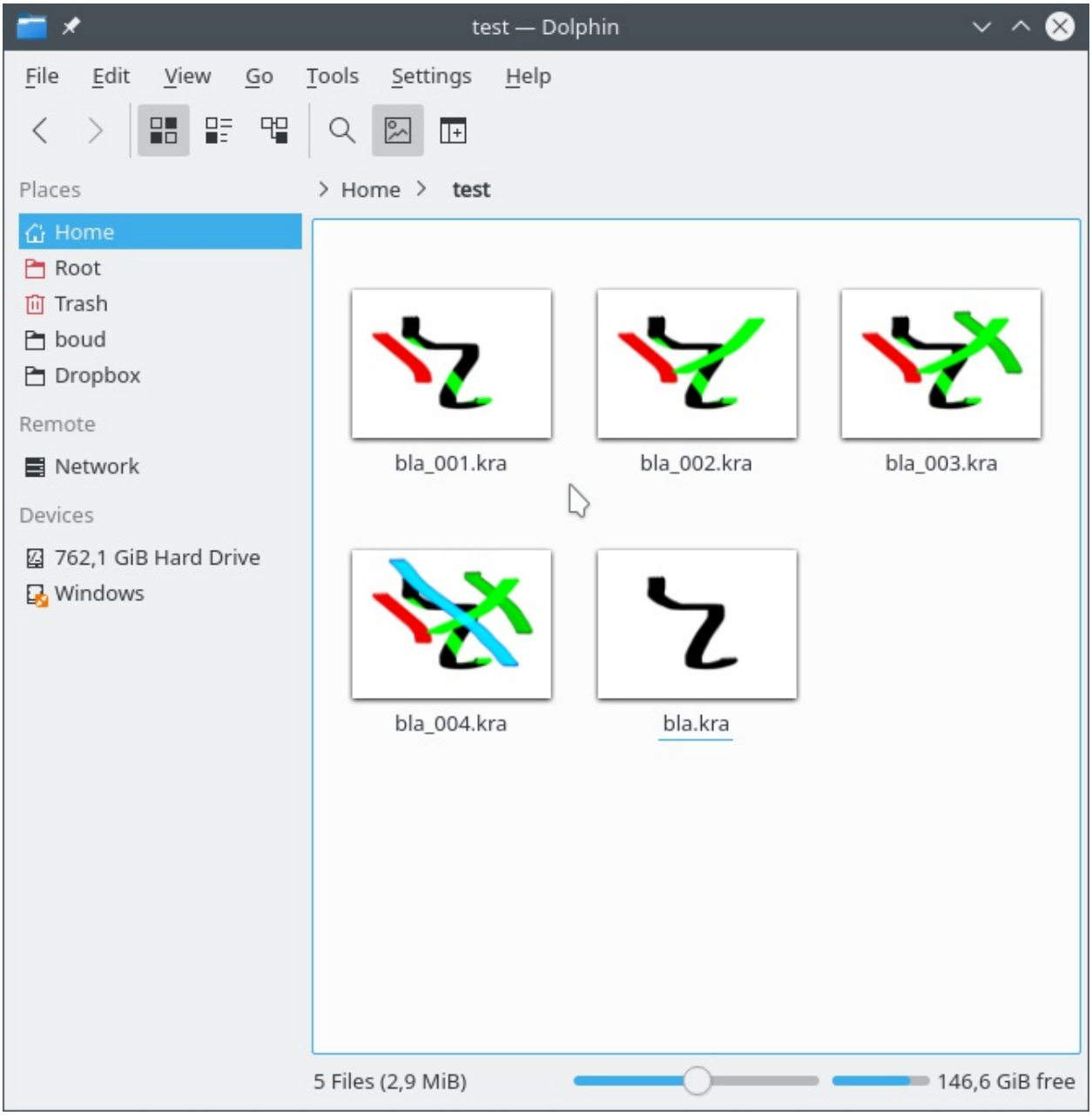


Use this when you want to keep various known good states of your image throughout your painting process. This takes more disk space, of course.

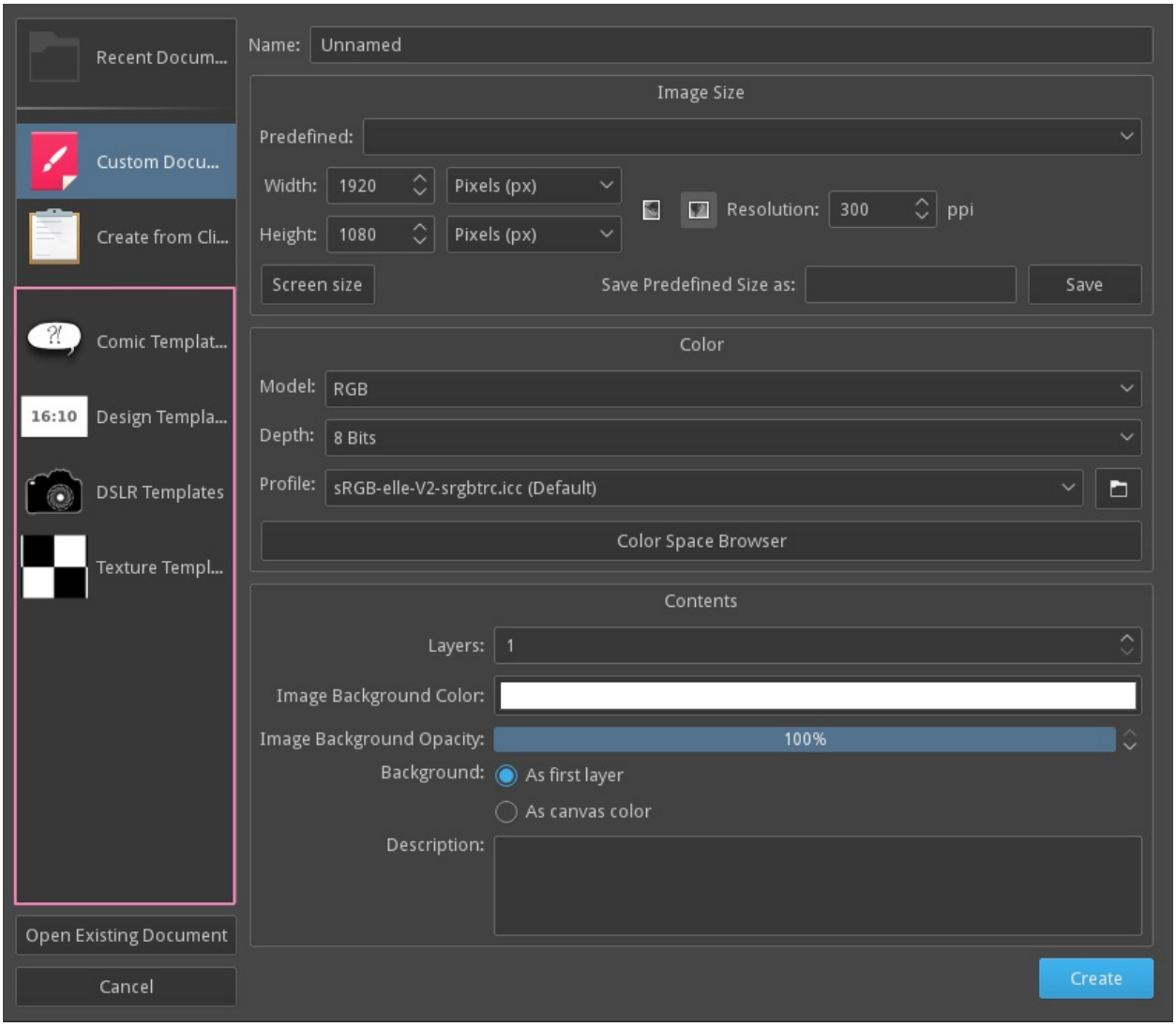
Do not be confused: Krita does not save the current state of your work to the latest Incremental file, but copies the last saved file to the Backup file and then saves your image under the original filename.

Incremental Version Files

Incremental Version works a bit like Incremental Backup, but it leaves the original files alone. Instead, it will save a new file with a file number:



Templates



Templates are just `.kra` files which are saved in a special location so it can be pulled up by Krita quickly. This is like the *Open Existing Document as Untitled Document…* but then with a nicer place in the UI.

You can make your own template file from any `.kra` file, by using *File* ▸ *Create Template from Image…* menu item. This will add your current document as a new template, including all its properties along with the layers and layer contents.

We have the following defaults:

Comic Templates

These templates are specifically designed for you to just get started with drawing comics. The comic template relies on a system of vectors and clones of those vector layers which automatically reflect any changes made to the vector layers. In between these two, you can draw your picture, and not fear them drawing over the panel. Use *Inherit Alpha* to clip the drawing by the panel.

European Bande Desinée Template.
: This one is reminiscent of the system used by for example TinTin or Spirou et Fantasio. These panels focus on wide images, and horizontal cuts.

US-style comics Template.
: This one is reminiscent of old DC and Marvel comics, such as Batman or Captain America. Nine images for quick story progression.

Manga Template.
: This one is based on Japanese comics, and focuses on a thin vertical gutter and a thick horizontal gutter, ensuring that the reader finished the previous row before heading to the next.

Waffle Iron Grid
: 12 little panels at your disposal.

Design Templates

These are templates for design and have various defaults with proper ppi at your disposal:

- Cinema 16:10
- Cinema 2.93:1
- Presentation A3-landscape
- Presentation A4 portrait

- Screen 4:3
- Web Design

DSLR templates

These have some default size for photos:

- Canon 55D
- Canon 5DMK3
- Nikon D3000
- Nikon D5000
- Nikon D7000

Texture Templates

These are for making 3D textures, and are between 1024, to 4092.

Introduction to Layers and Masks

Krita supports layers which help to better control parts and elements of your painting.

Think of an artwork or collage made with various stacks of papers with some paper cut such that they show the paper beneath them while some hide what's beneath them. If you want to replace an element in the artwork, you replace that piece of paper instead of drawing the entire thing. In Krita instead of papers we use **Layers**. Layers are part of the document which may or may not be transparent, they may be smaller or bigger than the document itself, they can arrange one above other, named and grouped.

Layers can give better control over your artwork for example you can re-color an entire artwork just by working on the separate color layer and thereby not destroying the line art which will reside above this color layer.

You can edit individual layers, you can even add special effects to them, like Layer styles, blending modes, transparency, filters and transforms. Krita takes all these layers in its layer stack, including the special effects and combines or composites together a final image. This is just one of the many digital image manipulation tricks that **Krita** has up its sleeve!

Usually, when you put one paint layer on top of another, the upper paint layer will be fully visible, while the layer behind it will either be obscured, occluded or only partially visible.

Managing layers

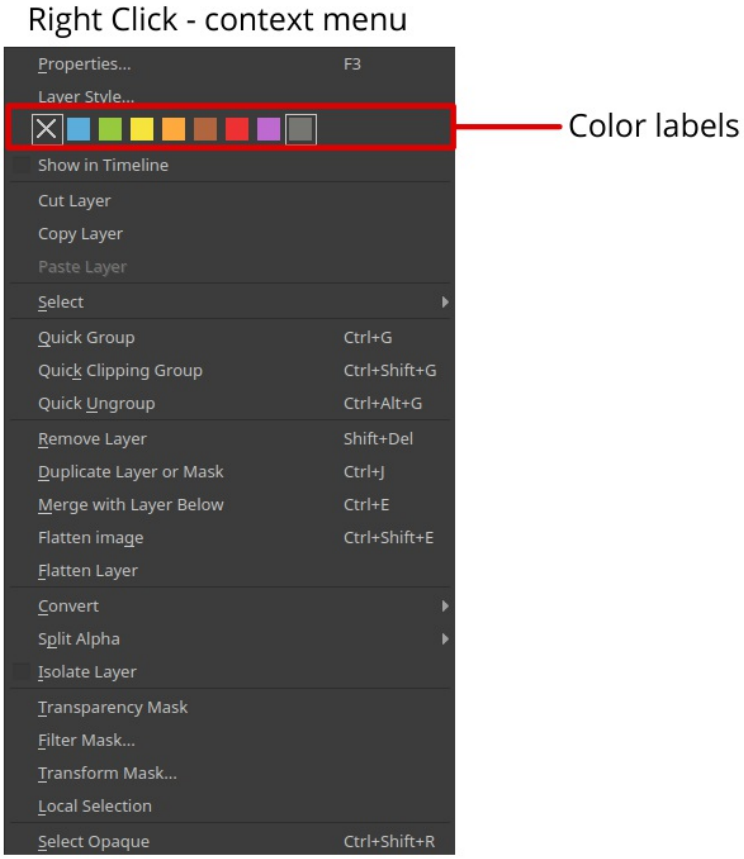
Some artists draw with limited number of layers but some prefer to have different elements of the artwork on separate layer. Krita has some good layer management features which make the layer management task easy.

You can group layers and organise the elements of your artwork.

The layer order can be changed or layers can be moved in and out of a group in the layer stack by simply holding them and dragging and dropping. Layers can also be copied across documents while in the subwindow mode, by dragging and dropping from one document to another.

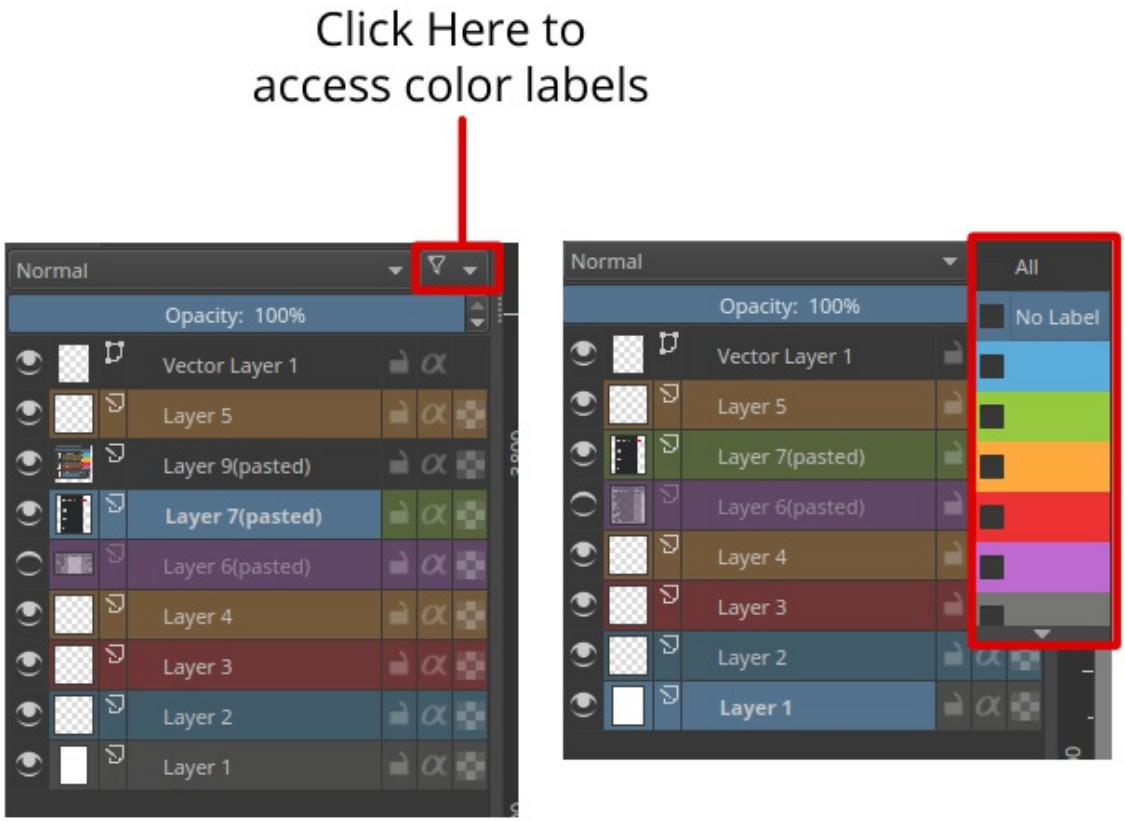
These features save time and also help artists in maintaining the file with a layer stack which will be easy to understand for others who work on the same file. In addition to these layers and groups can both be labeled and filtered by colors, thus helping the artists to visually differentiate them.

To assign a color label to your layer or layer group you have to right click on the layer and choose one of the given colors from the context menu. To remove an already existing color label you can click on the ‘x’ marked box in the context menu.

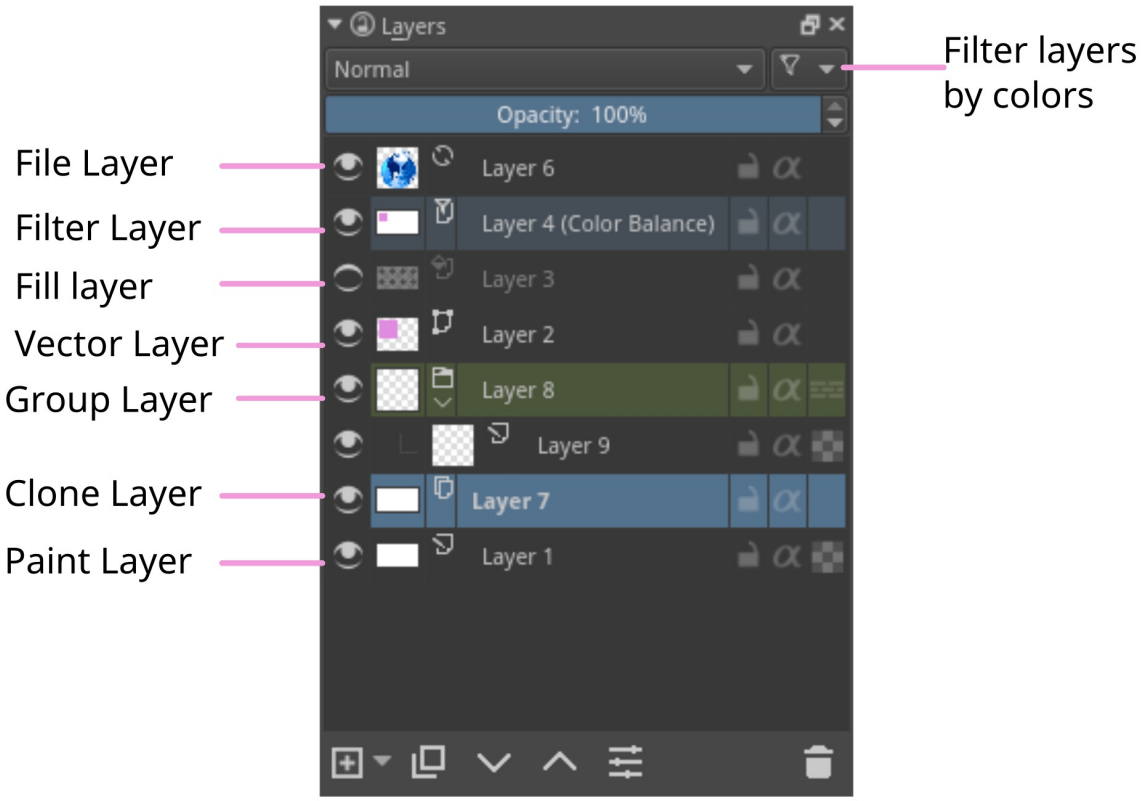


Once you assign color labels to your layers, you can then filter layers having similar color label by clicking on one or more colors in the list from the drop-

down situated at the top-right corner of the layer docker.



Types of Layers



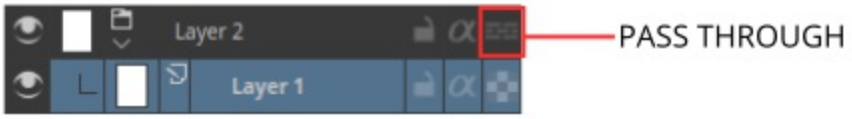
The image above shows the various types of layers in Layers. Each layer type has a different purpose for example all the vector elements can be only placed on a vector layer and similarly normal raster elements are mostly on the paint layer, Layers and Masks page contains more information about these types layers.

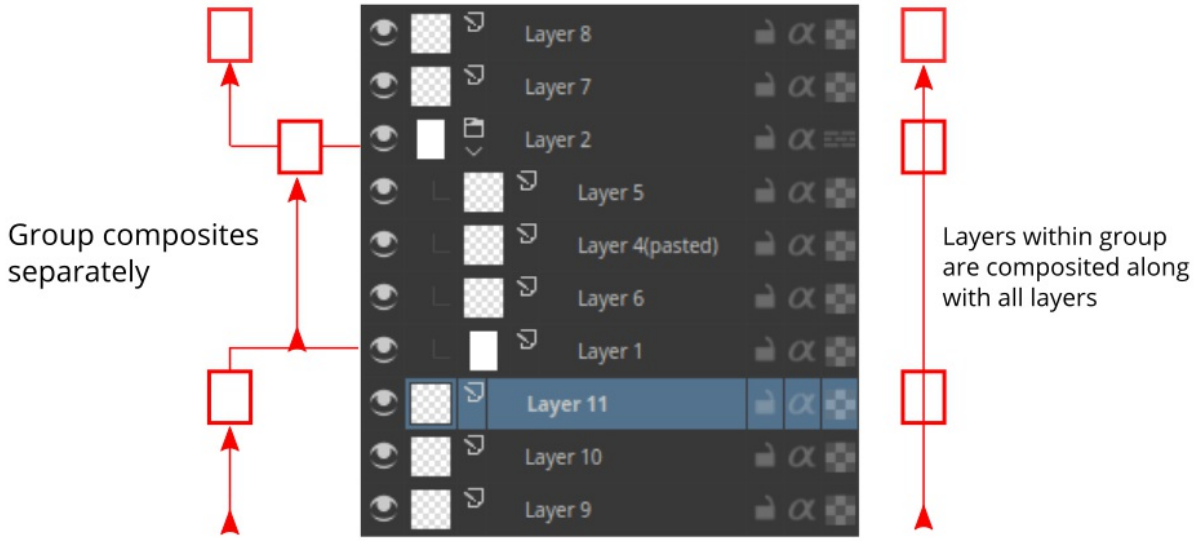
Now Let us see how these layers are composited in Krita.

How are layers composited in Krita ?

In Krita, the visible layers form a composite image which is shown on the canvas. The order in which Krita composites the layers is from bottom to top, much like the stack of papers we discussed above. As we continue adding layers, the image we see changes, according to the properties of the newly added layers on top. Group Layers composite separately from the other layers in the stack, except when pass through mode is activated. The layers inside a

group form a composite image first and then this composite is taken into consideration while the layer stack is composited to form a whole image. If the pass through mode is activated by pressing the icon similar to bricked wall, the layers within the group are considered as if they are outside of that particular group in the layer stack, however, the visibility of the layers in a group depends on the visibility of the group.

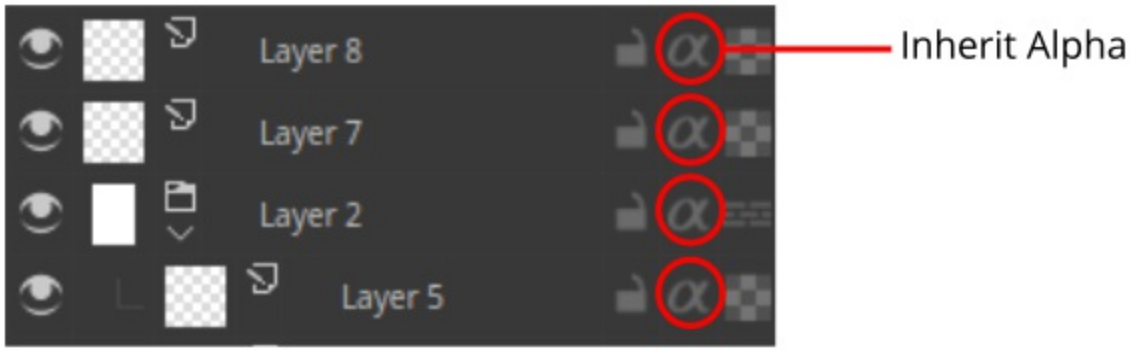




The groups in a PSD file saved from Photoshop have pass-through mode on by default unless they are specifically set with other blending modes.

Inherit Alpha or Clipping layers

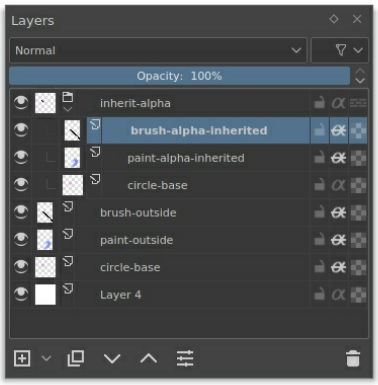
There is a clipping feature in Krita called inherit alpha. It is denoted by an alpha icon in the layer stack.

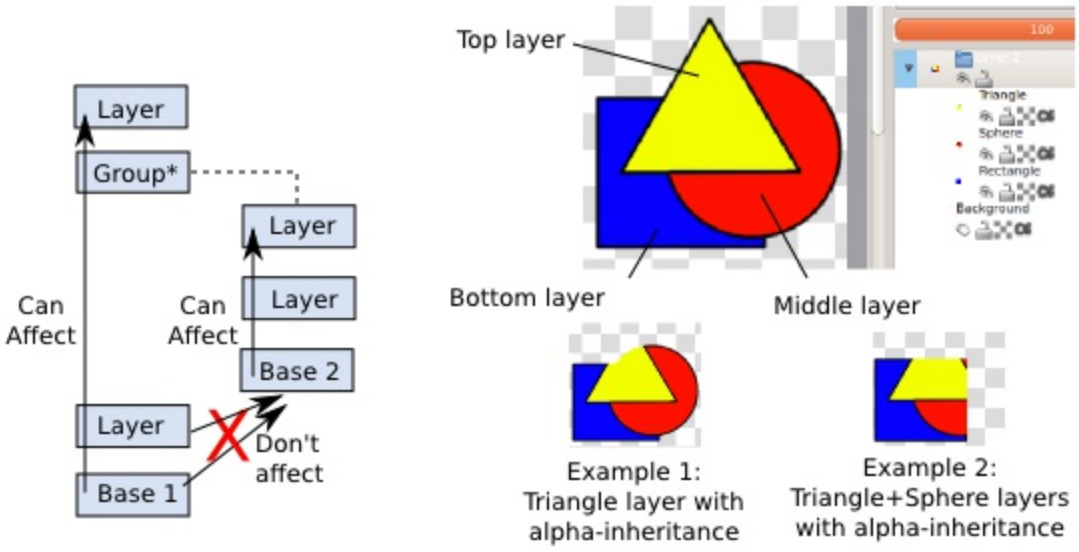


It can be somewhat hard to figure out how the inherit alpha feature works in Krita for the first time. Once you click on the inherit alpha icon on the layer stack, the pixels of the layer you are painting on are confined to the combined pixel area of all the layers below it. That means if you have the default white background layer as first layer, clicking on the inherit alpha icon and painting on any layer above will seem to have no effect as the entire canvas is filled with white. Hence, it is advised to put the base layer that you want the pixels to clip in a group layer. As mentioned above, group layers are composited separately, hence the layer which is the lowest layer in a group becomes the bounding layer and the content of the layers above this layer clips to it if inherit alpha is enabled.

Inherit Alpha Outside Group

Inherit Alpha Inside Group





You can also enable alpha inheritance to a group layer.

Masks and Filters

Krita supports non-destructive editing of the content of the layer. Non-destructive editing means editing or changing a layer or image without actually changing the original source image permanently, the changes are just added as filters or masks over the original image while keeping it intact, this helps a lot when your workflow requires constant back and forth. You can go back to original image with a click of a button. Just hide the filter or mask you have your initial image.

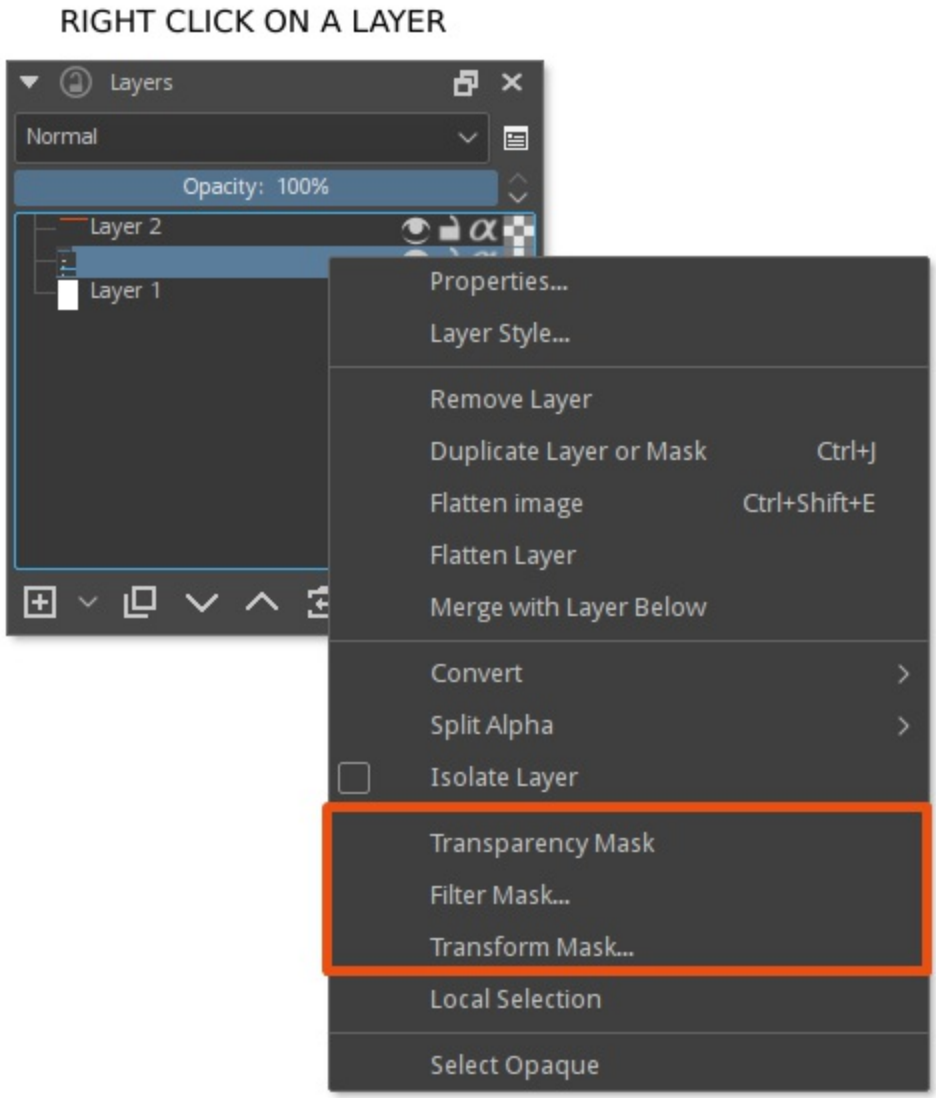
You can add various filters to a layer with Filter mask, or add Filter layer which will affect the whole image. Layers can also be transformed non-destructively with the transformation masks, and even have portions temporarily hidden with a Transparent Mask. Non-destructive effects like these are very useful when you change your mind later, or need to make a set of variations of a given image.

Note

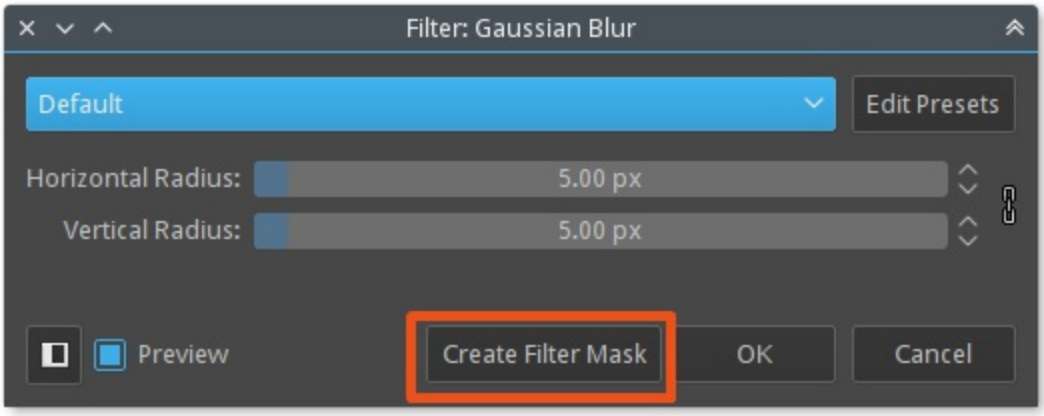
You can merge all visible layers by selecting everything first *Layer* ▸ *Select* ▸ *Visible Layers*. Then Combine them all by merging *Layer* ▸ *Merge with*

Layer Below.

These filters and masks are accessible through the right click menu (as shown in the image below) and the Plus icon on the layer docker.



You can also add a filter as a mask from filter dialog itself, by clicking on the *Create Filter Mask* button.



All the filters and masks can also be applied over a group too, thus making it easy to non-destructively edit multiple layers at once. In the category Layers and masks you can read more about the individual types of layers and masks.

Layer Docker has more information about the shortcuts and other layer management workflows.

Selections

Selections allow you to pick a specific area of your artwork to change. This is useful when you want to move a section of the painting, transform it, or paint on it without affecting the other sections. There are many selection tools available that select in different ways. Once an area is selected, most tools will stay inside that area. On that area you can draw or use gradients to quickly get colored and/or shaded shapes with hard edges. The selections in Krita are not limited to the canvas boundary, so you can also selection portions of the painting that are beyond the canvas boundary.

Creating Selections

The most common selection tools all exist at the bottom of the toolbox. Each tool selects things slightly differently. The links for each tool go into a more detailed description of how to use it.

Rectangular Selection Tool		Select the shape of a square.
Elliptical Selection Tool		Select the shape of a circle.
Polygonal Selection Tool		Click where you want each point of the Polygon to be. Double click to end your polygon and finalize your selection area. Use the `Shift + Z` shortcut to undo last point.

Outline Selection Tool		Outline/Lasso tool is used for a rough selection by drawing the outline.
Similar Color Selection Tool		Similar Color Selection Tool.
Contiguous Selection Tool		Contiguous or “Magic Wand” selects a field of color. Adjust the *Fuzziness* to allow more changes in the field of color, by default limited to the current layer.
Path Selection Tool		Path select an area based on a vector path, click to get sharp corners or drag to get flowing lines and close the path with the `Enter` key or connecting back to the first point.
Magnetic Selection Tool		Magnetic selection makes a free hand selection where the selection snaps to sharp contrasts in the image.

Note

You can also use the transform tools on your selection, a great way to try different proportions on parts of your image.

Editing Selections

The tool options for each selection tool gives you the ability to modify your selection.

Action	Modifier	Shortcut	Description
Replace	Ctrl	R	Replace the current selection.
Intersect	Shift + Alt	–	Get the overlapping section of both selections.
Add	Shift	A	Add the new selection to the current selection.
Subtract	Alt	S	Subtract the selection from the current selection.
Symmetric Difference	–	–	Make a selection where both the new and current do not overlap.

You can change this in Tools Settings.

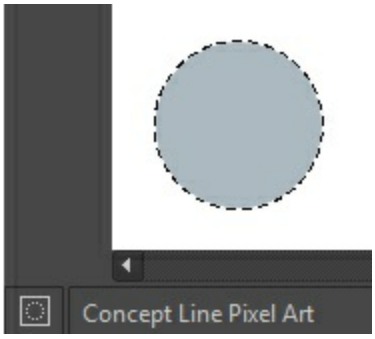
If you hover over a selection with a selection tool and no selection is activated, you can move it. To quickly go into transform mode,  and select *Edit Selection*.

Removing Selections

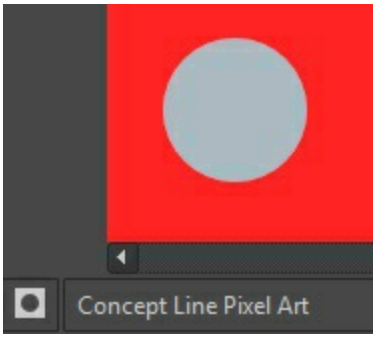
If you want to delete the entire selection, the easiest way is to deselect everything. *Select* ▸ *Deselect*. Shortcut `Ctrl + Shift + A`. When you have one of the selection tool active, and the mode of selection is in intersect, replace or symmetric difference then you can also deselect by just  anywhere on the canvas.

Display Modes

In the bottom left-hand corner of the status bar there is a button to toggle how the selection is displayed. The two display modes are the following: (Marching) Ants and Mask. The red color with Mask can be changed in the preferences. You can edit the color under *Settings* ▸ *Configure Krita…* ▸ *Display* ▸ *Selection Overlay*. If there is no selection, this button will not do anything.



Ants display mode (default) is best if you want to see the areas that are not selected.

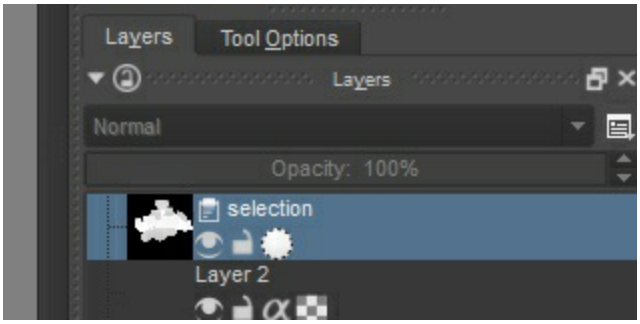


Mask display mode is good if you are interested in seeing the various transparency levels for your selection. For example, when you have a selection with very soft edges due using feathering.

Changed in version 4.2: Mask mode is activated as well when a selection mask is the active layer so you can see the different selection levels.

Global Selection Mask (Painting a Selection)

The global Selection Mask is your selection that appears on the layers docker. By default, this is hidden, so you will need to make it visible via *Select* ▸ *Show Global Selection Mask*.



Once the global Selection Mask is shown, you will need to create a selection. The benefit of using this is that you can paint your selection using any of the normal painting tools, including the transform and move. The information is saved as grayscale.

You can enter the global selection mask mode quickly from the selection tools by doing  and select *Edit Selection*.

Selection from layer transparency

You can create a selection based on a layer's transparency by right-clicking on the layer in the layer docker and selecting *Select Opaque* from the context menu.

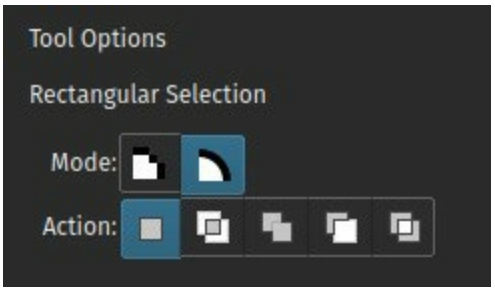
New in version 4.2: You can also do this for adding, subtracting and

intersecting by going to *Select* ▸ *Select Opaque*, where you can find specific actions for each.

If you want to quickly select parts of layers, you can hold the `Ctrl +` shortcut on the layer *thumbnail*. To add a selection do `Ctrl + Shift +` , to remove `Ctrl + Alt +` and to intersect `Ctrl + Shift + Alt +` . This works with any mask that has pixel or vector data (so everything but transform masks).

Pixel and Vector Selection Types

Vector selections allow you to modify your selection with vector anchor tools. Pixel selections allow you to modify selections with pixel information. They both have their benefits and disadvantages. You can convert one type of selection to another.



When creating a selection, you can select what type of selection you want from the Mode in the selection tool options: Pixel or Vector. By default this will be Vector.

Vector selections can be modified as any other vector shape with the Shape Selection Tool, if you try to paint on a vector selection mask it will be converted into a pixel selection. You can also convert vector shapes to selection. In turn, vector selections can be made from vector shapes, and vector shapes can be converted to vector selections using the options in the *Selection* menu. Krita will add a new vector layer for this shape.

One of the most common reasons to use vector selections is that they give

you the ability to move and transform a selection without the kind of resize artifacts you get with a pixel selection. You can also use the Shape Edit Tool to change the anchor points in the selection, allowing you to precisely adjust bezier curves or add corners to rectangular selections.

If you started with a pixel selection, you can still convert it to a vector selection to get these benefits. Go to *Select* ▸ *Convert to Vector Selection*.

Note

If you have multiple levels of transparency when you convert a selection to vector, you will lose the semi-transparent values.

Common Shortcuts while Using Selections

- Copy – `Ctrl + C` or `Ctrl + Ins`
- Paste – `Ctrl + V` or `Shift + Ins`
- Cut – `Ctrl + X`, `Shift + Del`
- Copy From All Layers – `Ctrl + Shift + C`
- Copy Selection to New Layer – `Ctrl + Alt + J`
- Cut Selection to New Layer – `Ctrl + Shift + J`
- Display or hide selection with `Ctrl + H`
- Select Opaque – `Ctrl +`  on layer thumbnail.
- Select Opaque (Add) – `Ctrl + Shift +`  on layer thumbnail.
- Select Opaque (Subtract) – `Ctrl + Alt +`  on layer thumbnail.
- Select Opaque (Intersect) – `Ctrl + Shift + Alt +`  on layer thumbnail.

Python Scripting

This section covers python scripting.

Contents:

Managing Python plugins

How to install a Python plugin

Caution

Custom Python plugins are made by users of Krita and the Krita team does not guarantee that they work, that they are useful or that they are *safe*. Note that a Python plugin can do everything that Krita can do, which means for example access to your files. Krita team isn't responsible for any damage you might suffer from the plugin and you install it on your own risk.

Using Python plugin importer

Note

This method doesn't always import action files (responsible for shortcuts) correctly.

You need to ensure that you have the plugin in a `*.zip` file. Inside the zip file there should be a file `pluginname.desktop` and a folder `pluginname` (instead of *pluginname* there should be an actual unique name of the plugin).

Go to *Tools* ▸ *Scripts* ▸ *Import Python Plugin…*, find the `*.zip` file and press *OK*. Restart Krita.

Go to *Configure Krita* ▸ *Python Plugins Manager*, find the plugin and enable it. Restart Krita.

Now the plugin should be available.

Manually

If the plugin is inside a `*.zip` archive, you need to extract it first.

Go to *Settings* ▸ *Manage Resources* ▸ *Open Resource Folder*. Put file `pluginname.desktop` and folder `pluginname` (instead of *pluginname* there should be an actual unique name of the plugin) inside the `pykrita` folder. Put file `pluginname.action` into the `actions` folder. Restart Krita.

Now the plugin should be available.

How to get to the plugin?

Plugins in Krita are either dockers or extensions.

If it's an extension, it will be available in the menu *Tools* ▸ *Scripts*.

If it's a docker, you can find it in *Settings* ▸ *Dockers*.

If the plugin has any shortcuts and you imported the action file properly, you can change the shortcuts in *Configure Krita* ▸ *Keyboard Shortcuts*.

How to enable and disable a plugin?

You can enable and disable all plugins (no matter if they're pre-installed or custom) in *Configure Krita* ▸ *Python Plugins Manager*.

Introduction to Python Scripting

New in version 4.0.

When we offered python scripting as one of Kickstarter Stretchgoals we could implement next to vectors and text, it won the backer vote by a landslide. Some people even only picked python and nothing else. So what exactly is python scripting?

What is Python Scripting?

Python is a scripting language, that can be used to automate tasks. What python scripting in Krita means is that we added an API to krita, which is a bit of programming that allows python to access to parts of Krita. With this we can make dockers, perform menial tasks on a lot of different files and even write our own exporters. People who work with computer graphics, like VFX and video game artists use python a lot to make things like sprite sheets, automate parts of export and more.

It is outside the scope of this manual to teach you python itself. However, as python is an extremely popular programming language and great for beginners, there's tons of learning material around that can be quickly found with a simple 'learn python' internet search.

This manual will instead focus on how to use python to automate and extend Krita. For that we'll first start with the basics: How to run Python commands in the scripter.

How to Enable the Scripter Plugin

The scripter plugin is not necessary to use python, but it is very useful for testing and playing around with python. It is a python console, written in python, which can be used to write small scripts and execute them on the fly.

To open the scripter, navigate to *Tools* ▸ *Scripts* ▸ *Scripter*. If you don't see it

listed, go to *Settings* ▸ *Configure Krita…* ▸ *Python Plugin Manager* and toggle “Scripter” in the list to enable it. If you don’t see the scripter plugin, make sure you are using an up-to-date version of Krita.

The scripter will pop up with a text editor window on top and an output window below. Input the following in the text area:

```
print("hello world")
```

Press the big play button or press the `Ctrl + R` shortcut to run the script. Then, below, in the output area the following should show up:

```
==== Warning: Script not saved! ====
hello world
```

Now we have a console that can run functions like print() from the Python environment - but how do we use it to manage Krita?

Running basic Krita commands

To allow Python to communicate with Krita, we will use the Krita module. At the top of every script, we will write: `from krita import *`

This allows us to talk to Krita through `Krita.instance()`. Let’s try to double our coding abilities with Python.

```
from krita import *

Krita.instance().action('python_scripter').trigger()
```

You should see a second scripter window open. Pretty neat! Here is a slightly more advanced example.

```
from krita import *

d = Krita.instance().createDocument(512, 512, "Python test docume
Krita.instance().activeWindow().addView(d)
```

This will open up a new document. Clearly Python gives you quite a lot of control to automate Krita. Over time we expect the community to write all

kinds of scripts that you can use simply by pasting them in the scripter.

But what if you want to write new commands for yourself? The best place to start is very simple: search for examples written by other people! You can save a lot of time if someone else has written code that you can base your work on. It's also worth looking through the python plugins, which are located in /share/krita/pykrita. There's also a step by step guide for How to make a Krita Python plugin here in the manual.

But it's likely that you need more information. For that, we will need see what's hidden behind the asterisk when you `import * from Krita`. To learn what Krita functions that are available and how to use them, you will want to go for Krita API reference documentation.

Krita's API

- LibKis API Overview [https://api.kde.org/extragear-api/graphics-apidocs/krita/libs/libkis/html/index.html]
- Krita class documentation [https://api.kde.org/extragear-api/graphics-apidocs/krita/libs/libkis/html/classKrita.html]

Those pages may look like a lot of jargon at first. This is because Krita's API documentation comes from the underlying C++ language that Krita is written in. The magic happens because of a Python tool called SIP, which makes it possible for python speak in C++ and talk to Krita. The end result is that when we `import krita` and call functions, we're actually using the C++ methods listed in that documentation.

Let's see how this stuff works in more detail. Let's take a look at the second link, the Krita class reference [https://api.kde.org/extragear-api/graphics-apidocs/krita/libs/libkis/html/classKrita.html#aa55507903d088013ced2df8c74f28a63]. There we can see all the functions available to the Krita instance. If you type dir(Krita.instance()) in Python, it should match this page very closely - you can view the documentation of the functions createDocument(), activeWindow(), and action() which we used above.

One of the more confusing things is seeing all the C++ classes that Krita uses, including the Qt classes that start with Q. But here is the beauty of SIP: it

tries to make the translation from these classes into Python as simple and straightforward as possible. For example, you can see that the function filters() returns a QStringList. However, SIP converts those QStringLists into regular python list of strings!

```
from krita import *

print(Krita.instance().filters())
```

Outputs as:

```
['asc-cdl', 'autocontrast', 'blur', 'burn', 'colorbalance', 'colo
'desaturate', 'dodge', 'edge detection', 'emboss', 'emboss all di
'emboss horizontal only', 'emboss laplascian', 'emboss vertical o
'gradientmap', 'halftone', 'height to normal', 'hsvadjustment', '
'maximize', 'mean removal', 'minimize', 'motion blur', 'noise', '
'pixelize', 'posterize', 'raindrops', 'randompick', 'roundcorners
'wave', 'waveletnoisereducer']
```

However, sometimes the conversion doesn't go quite as smoothly.

```
from krita import *

print(Krita.instance().documents())
```

gives something like this:

```
[<PyKrita.krita.Document object at 0x7f7294630b88>,
<PyKrita.krita.Document object at 0x7f72946309d8>,
<PyKrita.krita.Document object at 0x7f7294630c18>]
```

It is a list of something, sure, but how to use it? If we go back to the Krita apidocs page and look at the function, documents() we'll see there's actually a clickable link on the 'Document' class. If you follow that link [https://api.kde.org/extragear-api/graphics-apidocs/krita/libs/libkis/html/classDocument.html], you'll see that the document has a function called name() which returns the name of the document, and functions width() and height() which return the dimensions. So if we wanted to generate an info report about the documents in Krita, we could write a script like this:

```
from krita import *
```

```
for doc in Krita.instance().documents():
    print(doc.name())
    print(" "+str(doc.width())+"x"+str(doc.height()))
```

We get an output like:

```
==== Warning: Script not saved! ====
Unnamed
 2480x3508
sketch21
 3508x2480
Blue morning
 1600x900
```

Hopefully this will give you an idea of how to navigate the API docs now.

Krita's API has many more classes, you can get to them by going to the top-left class list, or just clicking their names to get to their API docs. The functions print() or dir() are your friends here as well. This line will print out a list of all the actions in Krita - you could swap in one of these commands instead of 'python_scripter' in the example above.

```
[print([a.objectName(), a.text()]) for a in Krita.instance().acti
```

The Python module `inspect` was designed for this sort of task. Here's a useful function to print info about a class to the console.

```
import inspect
def getInfo(target):
    [print(item) for item in inspect.getmembers(target) if not it

getInfo(Krita.instance())
```

Finally, in addition to the LibKis documentation, the Qt documentation, since Krita uses PyQt to expose nearly all of the Qt API to Python. You can build entire windows with buttons and forms this way, using the very same tools that Krita is using! You can read the Qt documentation [https://doc.qt.io/] and the PyQt documentation [https://www.riverbankcomputing.com/static/Docs/PyQt5/] for more info about this, and also definitely study the included plugins as well to see

how they work.

Technical Details

Python Scripting on Windows

To get Python scripting working on Windows 7/8/8.1, you will need to install the Universal C Runtime from Microsoft's website [https://www.microsoft.com/en-us/download/details.aspx?id=48234]. (Windows 10 already comes with it.)

Python 2 and 3

By default Krita is compiled for python 3.

However, it is possible to compile it with python 2. To do so, you will need to add the following to the cmake configuration line:

```
-DENABLE_PYTHON_2=ON
```

How to make a Krita Python plugin

You might have some neat scripts you have written in the Scripter Python runner, but maybe you want to do more with it and run it automatically for instance. Wrapping your script in a plugin can give you much more flexibility and power than running scripts from the Scripter editor.

Okay, so even if you know python really well, there are some little details to getting Krita to recognize a python plugin. So this page will give an overview how to create the various types of python script unique to Krita.

These mini-tutorials are written for people with a basic understanding of python, and in such a way to encourage experimentation instead of plainly copy and pasting code, so read the text carefully.

Getting Krita to recognize your plugin

A script in Krita has two components - the script directory (holding your script's Python files) and a ".desktop" file that Krita uses to load and register your script. For Krita to load your script both of these must put be in the pykrita subdirectory of your Krita resources folder (on Linux ~/.local/share/krita/pykrita). To find your resources folder start Krita and click the *Settings* ▸ *Manage Resources…* menu item. This will open a dialog box. Click the *Open Resources Folder* button. This should open a file manager on your system at your Krita resources folder. See the API [https://api.kde.org/extragear-api/graphics-apidocs/krita/libs/libkis/html/index.html] docs under "Auto starting scripts". If there is no pykrita subfolder in the Krita resources directory use your file manager to create one.

Scripts are identified by a file that ends in a .desktop extension that contain information about the script itself.

Therefore, for each proper plugin you will need to create a folder, and a desktop file.

The desktop file should look as follows:

```
[Desktop Entry]
Type=Service
ServiceTypes=Krita/PythonPlugin
X-KDE-Library=myplugin
X-Python-2-Compatible=false
X-Krita-Manual=myPluginManual.html
Name=My Own Plugin
Comment=Our very own plugin.
```

Type
: This should always be service.

ServiceTypes
: This should always be Krita/PythonPlugin for python plugins.

X-KDE-Library
: This should be the name of the plugin folder you just created.

X-Python-2-Compatible
: Whether it is python 2 compatible. If Krita was built with python 2 instead of 3 (`-DENABLE_PYTHON_2=ON` in the cmake configuration), then this plugin will not show up in the list.

X-Krita-Manual
: An Optional Value that will point to the manual item. This is shown in the Python Plugin manager. If it's an HTML file it'll be shown as rich text [https://doc.qt.io/qt-5/richtext-html-subset.html], if not, it'll be shown as plain text.

Name
: The name that will show up in the Python Plugin Manager.

Comment
: The description that will show up in the Python Plugin Manager.

Krita python plugins need to be python modules, so make sure there's an __init__.py script, containing something like…

```
from .myplugin import *
```

Where .myplugin is the name of the main file of your plugin. If you restart Krita, it now should show this in the Python Plugin Manager in the settings, but it will be grayed out, because there's no myplugin.py. If you hover over disabled plugins, you can see the error with them.

Note

You need to explicitly enable your plugin. Go to the Settings menu, open the *Configure Krita* dialog and go to the Python Plugin Manager page and enable your plugin.

Summary

In summary, if you want to create a script called *myplugin*:

- in your Krita *resources/pykrita* directory create
    - a folder called *myplugin*
    - a file called *myplugin.desktop*
- in the *myplugin* folder create
    - a file called *__init__.py*
    - a file called *myplugin.py*
- in the *__init__.py* file put this code:

```
from .myplugin import *
```

- in the desktop file put this code:

```
[Desktop Entry]
Type=Service
ServiceTypes=Krita/PythonPlugin
X-KDE-Library=myplugin
X-Python-2-Compatible=false
Name=My Own Plugin
Comment=Our very own plugin.
```

- write your script in the ‘’myplugin/myplugin.py’’ file.

Creating an extension

Extensions [https://api.kde.org/extragear-api/graphics-apidocs/krita/libs/libkis/html/classExtension.html] are relatively simple python scripts that run on Krita start. They are made by extending the Extension class, and the most barebones extension looks like this:

```
from krita import *

class MyExtension(Extension):

    def __init__(self, parent):
        # This is initialising the parent, always important when
        super().__init__(parent)

    def setup(self):
        pass

    def createActions(self, window):
        pass

# And add the extension to Krita's list of extensions:
Krita.instance().addExtension(MyExtension(Krita.instance()))
```

This code of course doesn’t do anything. Typically, in createActions we add actions to Krita, so we can access our script from the *Tools* menu.

First, let’s create an action [https://api.kde.org/extragear-api/graphics-apidocs/krita/libs/libkis/html/classAction.html]. We can do that easily with Window.createAction() [https://api.kde.org/extragear-api/graphics-apidocs/krita/libs/libkis/html/classWindow.html#a72ec58e53844076c1461966c34a9115c]. Krita will call createActions for every Window that is created and pass the right window object that we have to use.

So…

```
def createActions(self, window):
    action = window.createAction("myAction", "My Script", "tools/
```

“myAction”
: This should be replaced with a unique id that Krita will use to find the action.

“My Script”
: This is what will be visible in the tools menu.

If you now restart Krita, you will have an action called “My Script”. It still doesn’t do anything, because we haven’t connected it to a script.

So, let’s make a simple export document script. Add the following to the extension class, make sure it is above where you add the extension to Krita:

```
def exportDocument(self):
    # Get the document:
    doc =  Krita.instance().activeDocument()
    # Saving a non-existent document causes crashes, so lets chec
    if doc is not None:
        # This calls up the save dialog. The save dialog returns
        fileName = QFileDialog.getSaveFileName()[0]
        # And export the document to the fileName location.
        # InfoObject is a dictionary with specific export options
        doc.exportImage(fileName, InfoObject())
```

And add the import for QFileDialog above with the imports:

```
from krita import *
from PyQt5.QtWidgets import QFileDialog
```

Then, to connect the action to the new export document:

```
def createActions(self, window):
    action = window.createAction("myAction", "My Script")
    action.triggered.connect(self.exportDocument)
```

This is an example of a signal/slot connection [https://doc.qt.io/qt-5/signalsandslots.html], which Qt applications like Krita use a lot. We’ll go over how to make our own signals and slots a bit later.

Restart Krita and your new action ought to now export the document.

Creating configurable keyboard shortcuts

Now, your new action doesn't show up in *Settings* ▸ *Configure Krita* ▸ *Keyboard Shortcuts*.

Krita, for various reasons, only adds actions to the shortcuts menu when they are present in an .action file. The action file to get our action to be added to shortcuts should look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<ActionCollection version="2" name="Scripts">
    <Actions category="Scripts">
        <text>My Scripts</text>

        <Action name="myAction">
        <icon></icon>
        <text>My Script</text>
        <whatsThis></whatsThis>
        <toolTip></toolTip>
        <iconText></iconText>
        <activationFlags>10000</activationFlags>
        <activationConditions>0</activationConditions>
        <shortcut>ctrl+alt+shift+p</shortcut>
        <isCheckable>false</isCheckable>
        <statusTip></statusTip>
        </Action>
    </Actions>
</ActionCollection>
```

<text>My Scripts</text>
: This will create a sub-category under scripts called "My Scripts" to add your shortcuts to.

name
: This should be the unique id you made for your action when creating it in the setup of the extension.

icon
: the name of a possible icon. These will only show up on KDE plasma, because Gnome and Windows users complained they look ugly.

text

The text that it will show in the shortcut editor.

whatsThis
: The text it will show when a Qt application specifically calls for ‘what is this’, which is a help action.

toolTip
: The tool tip, this will show up on hover-over.

iconText
: The text it will show when displayed in a toolbar. So for example, “Resize Image to New Size” could be shortened to “Resize Image” to save space, so we’d put that in here.

activationFlags
: This determines when an action is disabled or not.

activationConditions
: This determines activation conditions (e.g. activate only when selection is editable). See the code [https://cgit.kde.org/krita.git/tree/libs/ui/kis_action.h#n76] for examples.

shortcut
: Default shortcut.

isCheckable
: Whether it is a checkbox or not.

statusTip
: The status tip that is displayed on a status bar.

Save this file as “myplugin.action” where myplugin is the name of your plugin. The action file should be saved, not in the pykrita resources folder, but rather in a resources folder named “actions”. (So, share/pykrita is where the python plugins and desktop files go, and share/actions is where the action files go) Restart Krita. The shortcut should now show up in the shortcut action list.

Creating a docker

Creating a custom docker [https://api.kde.org/extragear-api/graphics-apidocs/krita/libs/libkis/html/classDockWidget.html] is much like creating an extension. Dockers are in some ways a little easier, but they also require more use of widgets. This is the barebones docker code:

```
from PyQt5.QtWidgets import *
from krita import *

class MyDocker(DockWidget):

    def __init__(self):
        super().__init__()
        self.setWindowTitle("My Docker")

    def canvasChanged(self, canvas):
        pass

Krita.instance().addDockWidgetFactory(DockWidgetFactory("myDocker
```

The window title is how it will appear in the docker list in Krita. canvasChanged always needs to be present, but you don't have to do anything with it, so hence just 'pass'.

For the addDockWidgetFactory…

"myDocker"
: Replace this with an unique ID for your docker that Krita uses to keep track of it.

DockWidgetFactoryBase.DockRight
: The location. These can be DockTornOff, DockTop, DockBottom, DockRight, DockLeft, or DockMinimized

MyDocker
: Replace this with the class name of the docker you want to add.

So, if we add our export document function we created in the extension section to this docker code, how do we allow the user to activate it? First,

we'll need to do some Qt GUI coding: Let's add a button!

By default, Krita uses PyQt, but its documentation is pretty bad, mostly because the regular Qt documentation is really good, and you'll often find that the PyQt documentation of a class, say, QWidget [https://www.riverbankcomputing.com/static/Docs/PyQt5/api/qtwidgets/qwidget.html] is like a weird copy of the regular Qt documentation [https://doc.qt.io/qt-5/qwidget.html] for that class.

Anyway, what we need to do first is that we need to create a QWidget, it's not very complicated, under setWindowTitle, add:

```
mainWidget = QWidget(self)
self.setWidget(mainWidget)
```

Then, we create a button:

```
buttonExportDocument = QPushButton("Export Document", mainWidget)
```

Now, to connect the button to our function, we'll need to look at the signals in the documentation. QPushButton [https://doc.qt.io/qt-5/qpushbutton.html] has no unique signals of its own, but it does say it inherits 4 signals from QAbstractButton [https://doc.qt.io/qt-5/qabstractbutton.html#signals], which means that we can use those too. In our case, we want clicked.

```
buttonExportDocument.clicked.connect(self.exportDocument)
```

If we now restart Krita, we'll have a new docker and in that docker there's a button. Clicking on the button will call up the export function.

However, the button looks aligned a bit oddly. That's because our mainWidget has no layout. Let's quickly do that:

```
mainWidget.setLayout(QVBoxLayout())
mainWidget.layout().addWidget(buttonExportDocument)
```

Qt has several layouts [https://doc.qt.io/qt-5/qlayout.html], but the QHBoxLayout and the QVBoxLayout [https://doc.qt.io/qt-5/qboxlayout.html] are the easiest to use, they just arrange widgets horizontally or vertically.

Restart Krita and the button should now be laid out nicely.

PyQt Signals and Slots

We've already been using PyQt signals and slots already, but there are times where you want to create your own signals and slots. [As PyQt's documentation is pretty difficult to understand](https://www.riverbankcomputing.com/static/Docs/PyQt5/signals_slots.html), and the way how signals and slots are created is very different from C++ Qt, we're explaining it here:

All python functions you make in PyQt can be understood as slots, meaning that they can be connected to signals like Action.triggered or QPushButton.clicked. However, QCheckBox has a signal for toggled, which sends a boolean. How do we get our function to accept that boolean?

First, make sure you have the right import for making custom slots:

```
from PyQt5.QtCore import pyqtSlot
```

(If there's `from PyQt5.QtCore import *` already in the list of imports, then you won't have to do this, of course.)

Then, you need to add a PyQt slot definition before your function:

```
@pyqtSlot(bool)
def myFunction(self, enabled):
    enabledString = "disabled"
    if (enabled == True):
        enabledString = "enabled"
    print("The checkbox is"+enabledString)
```

Then, when you have created your checkbox, you can do something like myCheckbox.toggled.connect(self.myFunction).

Similarly, to make your own PyQt signals, you do the following:

```
# signal name is added to the member variables of the class
signal_name = pyqtSignal(bool, name='signalName')
```

```
def emitMySignal(self):
    # And this is how you trigger the signal to be emitted.
    self.signal_name.emit(True)
```

And use the right import:

```
from PyQt5.QtCore import pyqtSignal
```

To emit or create slots for objects that aren't standard python objects, you only have to put their names between quotation marks.

A note on unit tests

If you want to write unit tests for your plugin, have a look at the mock krita module [https://github.com/rbreu/krita-python-mock].

Conclusion

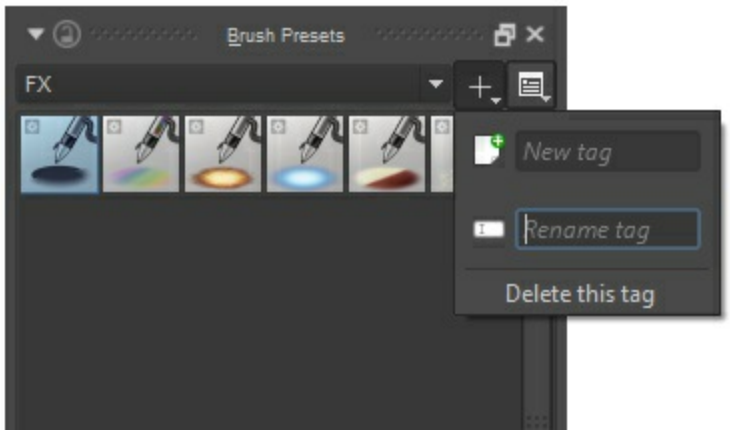
Okay, so that covers all the Krita specific details for creating python plugins. It doesn't handle how to parse the pixel data, or best practices with documents, but if you have a little bit of experience with python you should be able to start creating your own plugins.

As always, read the code carefully and read the API docs for python, Krita and Qt carefully to see what is possible, and you'll get pretty far.

Tag Management

Tags are how you organize common types of resources. They can be used with brushes, gradients, patterns, and even brush tips. You can select them from a drop-down menu above the resources. Selecting a tag will filter all the resources by that tag. Selecting the tag of *All* will show all resources. Krita comes installed with a few default tags. You can create and edit your own as well. The tags are managed similarly across the different types of resources.

You can use tags together with the Pop-up Palette for increased productivity.



Note

You can select different brush tags in the pop-up palette. This can be a quick way to access your favorite brushes.

Adding a New Tag for a Brush

By pressing the + next to the tag selection, you will get an option to add a tag. Type in the name you want and press the `Enter` key. You will need to go back to the *All* tag to start assigning brushes.

Assigning an Existing Tag to a Brush

Right-click on a brush in the Brush Presets Docker. You will get an option to assign a tag to the brush.

Changing a Tag's Name

Select the existing tag that you want to have changed from the drop-down. Press the + icon next to the tag. You will get an option to rename it. Press the `Enter` key to confirm. All the existing brushes will remain in the newly named tag.

Deleting a Tag

Select the existing tag that you want to have removed from the drop-down. Press the + icon next to the tag. You will get an option to remove it.

Note

The default brushes that come with Krita cannot have their default tags removed.

Soft Proofing

When we make an image in Krita, and print that out with a printer, the image tends to look different. The colors are darker, or less dark than expected, maybe the reds are more aggressive, maybe contrast is lost. For simple documents, this isn’t much of a problem, but for professional prints, this can be very sad, as it can change the look and feel of an image drastically.

The reason this happens is simply because the printer uses a different color model (CMYK) and it has often access to a lower range of colors (called a gamut).

A naive person would suggest the following solution: do your work within the CMYK color model! But there are three problems with that:

- Painting in a CMYK space doesn’t guarantee that the colors will be the same on your printer. For each combination of Ink, Paper and Printing device, the resulting gamut of colors you can use is different. Which means that each of these could have a different profile associated with them.
- Furthermore, even if you have the profile and are working in the exact color space that your printer can output, the CMYK color space is very irregular, meaning that the color maths isn’t as nice as in other spaces. Blending modes are different in CMYK as well.
- Finally, working in that specific CMYK space means that the image is stuck to that space. If you are preparing your work for different a CMYK profile, due to the paper, printer or ink being different, you might have a bigger gamut with more bright colors that you would like to take advantage of.

So ideally, you would do the image in RGB, and use all your favorite RGB tools, and let the computer do a conversion to a given CMYK space on the fly, just for preview. This is possible, and is what we call ‘’Soft Proofing’’.

On the left, the original, on the right, a view where soft proofing is turned o
subtle due to the lack of really bright colors, but the soft proofed version is sl
the whites of the flowers and slightly less saturated in the greens of

You can toggle soft proofing on any image using the `Ctrl + Y` shortcut. Unlike other programs, this is per-view, so that you can look at your image non-proofed and proofed, side by side. The settings are also per image, and saved into the `.kra` file. You can set the proofing options in *Image* ▸ *Image Properties* ▸ *Soft Proofing*.

There you can set the following options:

Profile, Depth, Space
: Of these, only the profile is really important. This will serve as the profile you are proofing to. In a professional print workflow, this profile should be determined by the printing house.

Intent

Set the proofing Intent. It uses the same intents as the intents mentioned in the color managed workflow.

Left: Soft proofed image with Adaptation state slider set to max. Right: S
Adaptation State set to minimum.

Adaptation State

A feature which allows you to set whether *Absolute Colorimetric* will make the white in the image screen-white during proofing (the slider set to max), or whether it will use the white point of the profile (the slider set to minimum). Often CMYK profiles have a different white as the screen, or amongst one another due to the paper color being different.

Black Point Compensation

Set the black point compensation. Turning this off will crunch the shadow values to the minimum the screen and the proofing profile can handle, while turning this on will scale the black to the screen-range, showing you the full range of grays in the image.

Gamut Warning
: Set the color of the out-of-gamut warning.

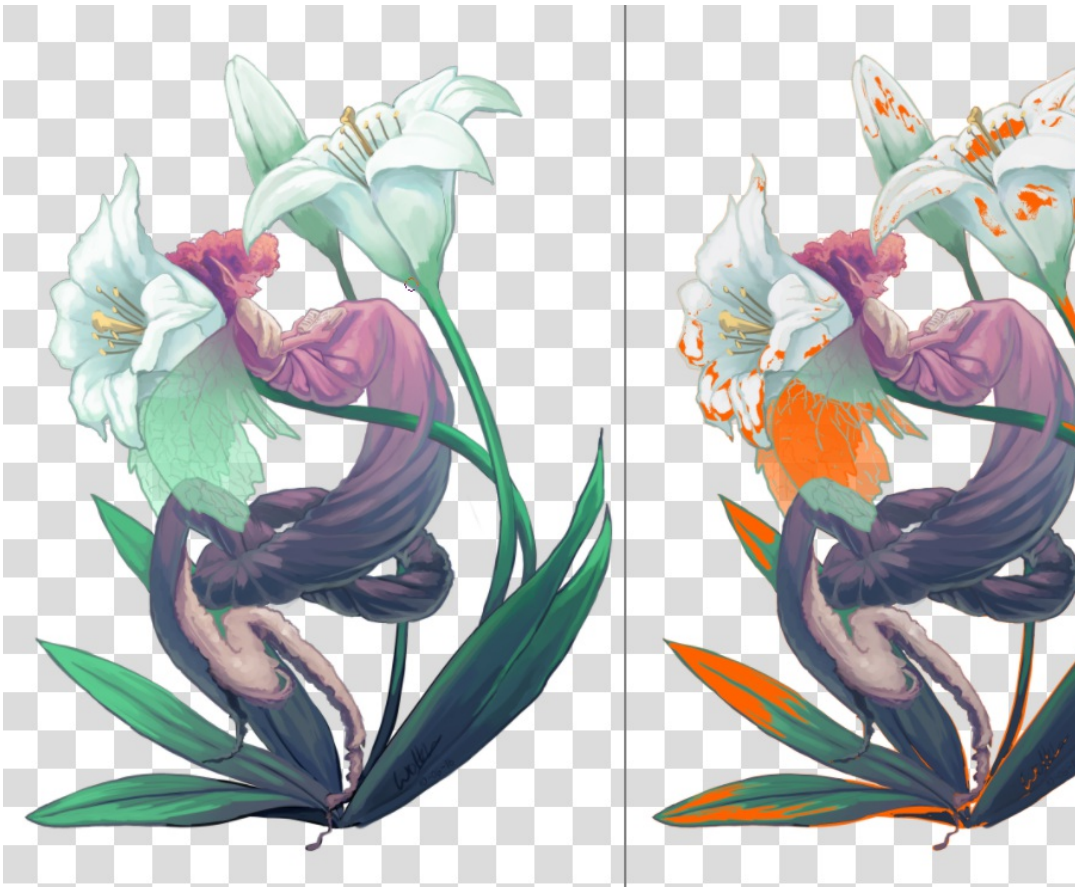
You can set the defaults that Krita uses in *Settings* ▸ *Configure Krita…* ▸ *Color Management*.

To configure this properly, it’s recommended to make a test image to print (and that is printed by a properly set-up printer) and compare against, and then approximate in the proofing options how the image looks compared to the real-life copy you have made.

Out of Gamut Warning

The out of gamut warning, or gamut alarm, is an extra option on top of Soft-Proofing: It allows you to see which colors are being clipped, by replacing the resulting color with the set alarm color.

This can be useful to determine where certain contrasts are being lost, and to allow you to change it slowly to a less contrasted image.



Left: View with original image, Right: View with soft proofing and gamut v
Krita will save the gamut warning color alongside the proofing options into th
color that you think will stand out for your current image

You can activate Gamut Warnings with the `Ctrl + Shift + Y` shortcut, but it needs soft proofing activated to work fully.

Note

Soft Proofing doesn’t work properly in floating-point spaces, and attempting to force it will cause incorrect gamut alarms. It is therefore disabled.

Warning

Gamut Warnings sometimes give odd warnings for linear profiles in the

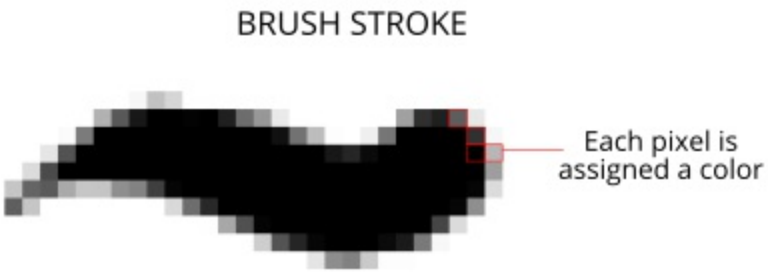
shadows. This is a bug in LCMS, see here [https://ninedegreesbelow.com/bug-reports/soft-proofing-problems.html] for more info.

Vector Graphics

Krita 4.0 has had a massive rewrite of the vector tools. So here's a page explaining the vector tools:

What are vector graphics?

Krita is primarily a raster graphics editing tool, which means that most of the editing changes the values of the pixels on the raster that makes up the image.



Vector graphics on the other hand use mathematics to describe a shape. Because it uses a formula, vector graphics can be resized to any size.

On one hand, this makes vector graphics great for logos and banners. On the other hand, raster graphics are much easier to edit, so vectors tend to be the domain of deliberate design, using a lot of precision.

Tools for making shapes

You can start making vector graphics by first making a vector layer (press the arrow button next to the + in the layer docker to get extra layer types). Then, all the usual drawing tools outside of the freehand, dynamic and the multibrush tool can be used to draw shapes.

The path and polyline tool are the tools you used most often on a vector layer, as they allow you to make the most dynamic of shapes.

On the other hand, the *Ellipse* and *Rectangle* tools allow you to draw special shapes, which then can be edited to make special pie shapes, or for easy rounded rectangles.

The calligraphy and text tool also make special vectors. The calligraphy tool is for producing strokes that are similar to brush strokes, while the text tool makes a text object that can be edited afterwards.

All of these will use the current brush size to determine stroke thickness, as well as the current foreground and background color.

There is one last way to make vectors: the *Vector Image* tool. It allows you to add shapes that have been defined in an SVG file as symbols. Unlike the other tools, these have their own fill and stroke.

Arranging Shapes

A vector layer has its own hierarchy of shapes, much like how the whole image has a hierarchy of layers. So shapes can be in front of one another. This can be modified with the arrange docker, or with the *Select Shapes* tool.

The arrange docker also allows you to group and ungroup shapes. It also allows you to precisely align shapes, for example, have them aligned to the center, or have an even spacing between all the shapes.

Editing shapes

Editing of vector shapes is done with the *Select Shapes* tool and the *Edit Shapes* tool.

The *Select Shapes* tool can be used to select vector shapes, to group them (via ), ungroup them, to use booleans to combine or subtract shapes from one another (via ), to move them up and down, or to do quick transforms.

Fill

You can change the fill of a shape by selecting it and changing the active foreground color.

You can also change it by going into the tool options of the *Select Shapes* tool and going to the *Fill* tab.

Vector shapes can be filled with a solid color, a gradient or a pattern.

Stroke

Strokes can be filled with the same things as fills.

However, they can also be further changed. For example, you can add dashes and markers to the line.

Coordinates

Shapes can be moved with the *Select Shapes* tool, and in the tool options you can specify the exact coordinates.

Editing nodes and special parameters

If you have a shape selected, you can double click it to get to the appropriate tool to edit it. Usually this is the *Edit Shape* tool, but for text this is the *Text* tool.

In the *Edit Shape* tool, you can move around nodes on the canvas for regular paths. For special paths, like the ellipse and the rectangle, you can move nodes and edit the specific parameters in the *Tool Options* docker.

Working together with other programs

One of the big things Krita 4.0 brought was moving from ODG to SVG. What this means is that Krita saves as SVG inside KRA files, and that means we can open SVGs just fine. This is important as SVG is the most popular

vector format.

Inkscape

You can copy and paste vectors from Krita to Inkscape, or from Inkscape to Krita. Only the SVG 1.1 features are supported, so don’t be surprised if a mesh gradient doesn’t cross over very well.

Snapping

In Krita 3.0, we now have functionality for Grids and Guides, but of course, this functionality is by itself not that interesting without snapping.

Snapping is the ability to have Krita automatically align a selection or shape to the grids and guides, document center and document edges. For Vector layers, this goes even a step further, and we can let you snap to bounding boxes, intersections, extrapolated lines and more.

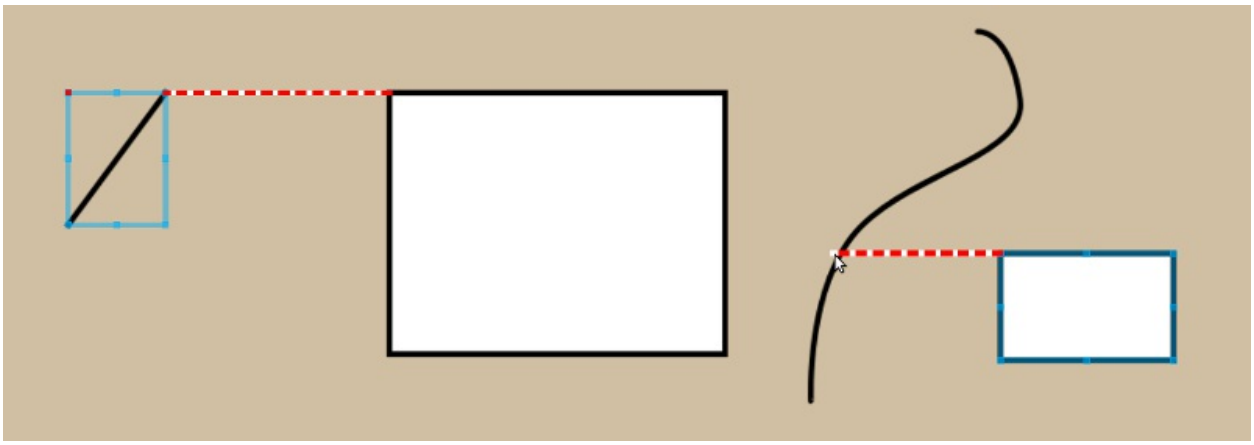
All of these can be toggled using the snap pop-up menu which is assigned to `Shift + S` shortcut.

Now, let us go over what each option means:

Grids
: This will snap the cursor to the current grid, as configured in the grid docker. This doesn't need the grid to be visible. Grids are saved per document, making this useful for aligning your art work to grids, as is the case for game sprites and grid-based designs.

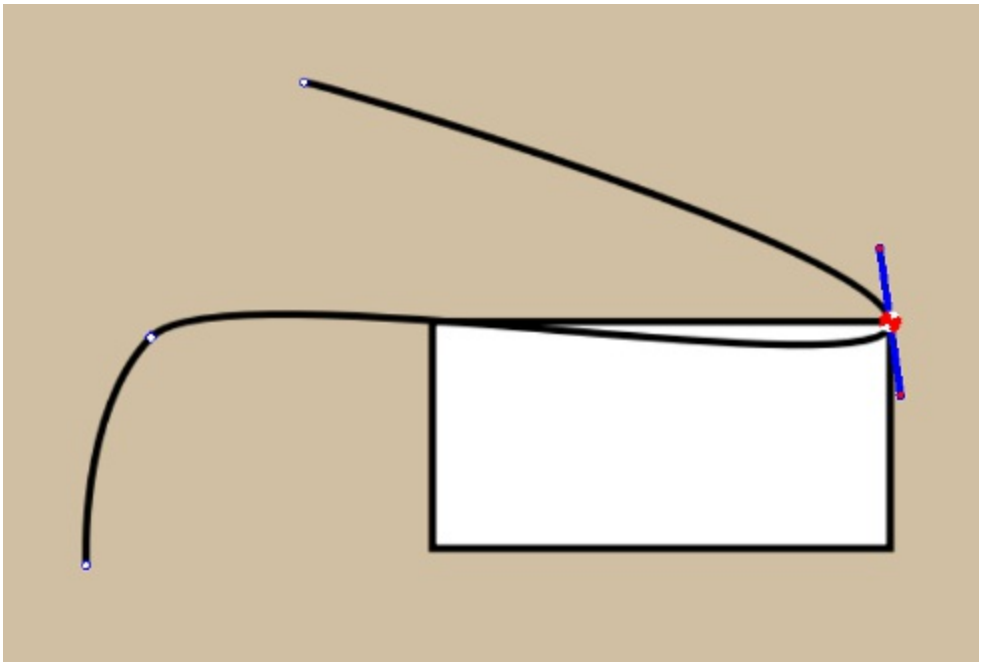
Pixel
: This allows to snap to every pixel under the cursor. Similar to Grid Snapping but with a grid having spacing = 1px and offset = 0px.

Guides
: This allows you to snap to guides, which can be dragged out from the ruler. Guides do not need to be visible for this, and are saved per document. This is useful for comic panels and similar print-layouts, though we recommend Scribus for more intensive work.



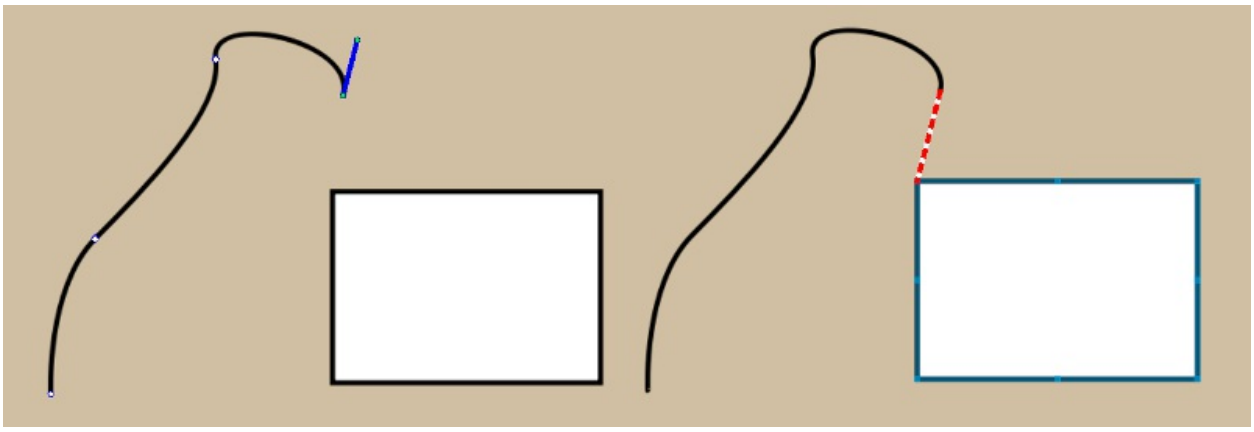
Orthogonal (Vector Only)

This allows you to snap to a horizontal or vertical line from existing vector objects's nodes (Unless dealing with resizing the height or width only, in which case you can drag the cursor over the path). This is useful for aligning object horizontally or vertically, like with comic panels.



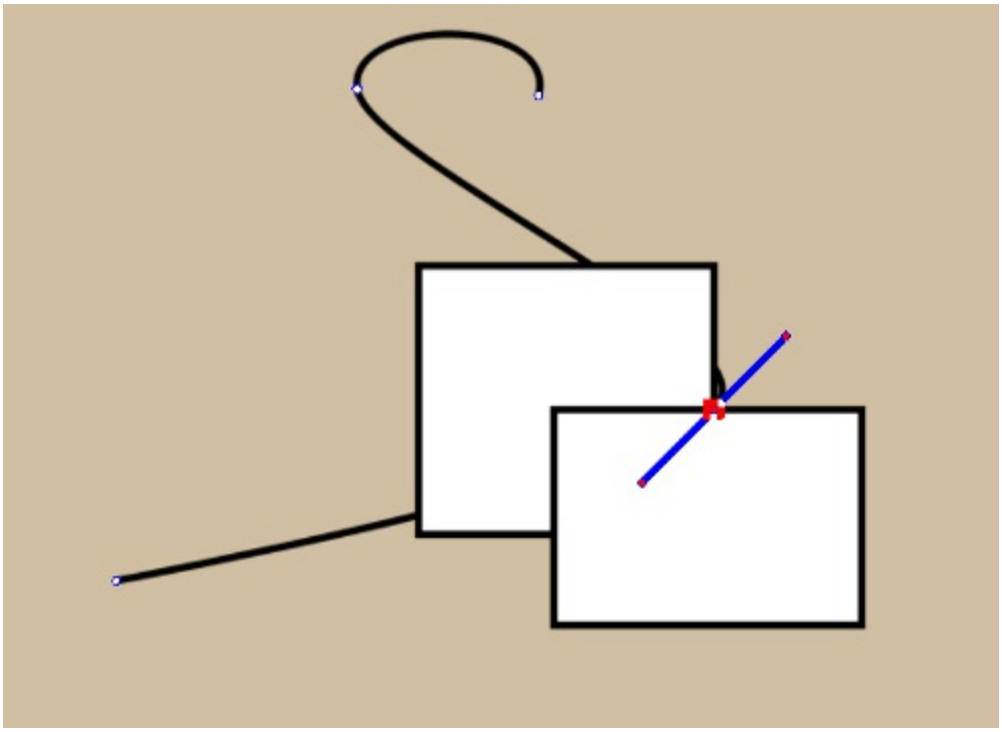
Node (Vector Only)

This snaps a vector node or an object to the nodes of another path.



Extension (Vector Only)

When we draw an open path, the last nodes on either side can be mathematically extended. This option allows you to snap to that. The direction of the node depends on its side handles in path editing mode.



Intersection (Vector Only)

This allows you to snap to an intersection of two vectors.

Bounding box (Vector Only)

This allows you to snap to the bounding box of a vector shape.

Image bounds

Allows you to snap to the vertical and horizontal borders of an image.

Image center

Allows you to snap to the horizontal and vertical center of an image.

The snap works for the following tools:

1. Straight line
2. Rectangle
3. Ellipse
4. Polyline
5. Path
6. Freehand path
7. Polygon
8. Gradient
9. Shape Handling tool
10. The Text-tool
11. Assistant editing tools
12. The move tool (note that it snaps to the cursor position and not the bounding box of the layer, selection or whatever you are trying to move)
13. The Transform tool
14. Rectangle select
15. Elliptical select
16. Polygonal select
17. Path select
18. Guides themselves can be snapped to grids and vectors

Snapping doesn’t have a sensitivity yet, and by default is set to 10 screen pixels.

Animation with Krita

Thanks to the 2015 Kickstarter, **Krita** has animation. In specific, **Krita** has frame-by-frame raster animation. There's still a lot of elements missing from it, like tweening, but the basic workflow is there.

To access the animation features, the easiest way is to change your workspace to Animation. This will make the animation dockers and workflow appear.

Note

New in version 4.1: The Timeline docker looks a bit different from the screenshots shown in this tutorial, however you should be able to follow it if you take care to select options mentioned in text.

Animation curves

To create an animation curve (currently only for opacity) expand the *New Frame* button in the *Animation* dock and click *Add Opacity Keyframe*. You can now edit the keyframed value for opacity directly in the "Layers" dock, adding more keyframes will by default fade from the last to the next upcoming keyframe in the timeline over the frames between them. See animation curves for details.

Workflow

In traditional animation workflow, what you do is that you make *key frames*, which contain the important poses, and then draw frames in between (*tweening* in highly sophisticated animator's jargon).

For this workflow, there are three important dockers:

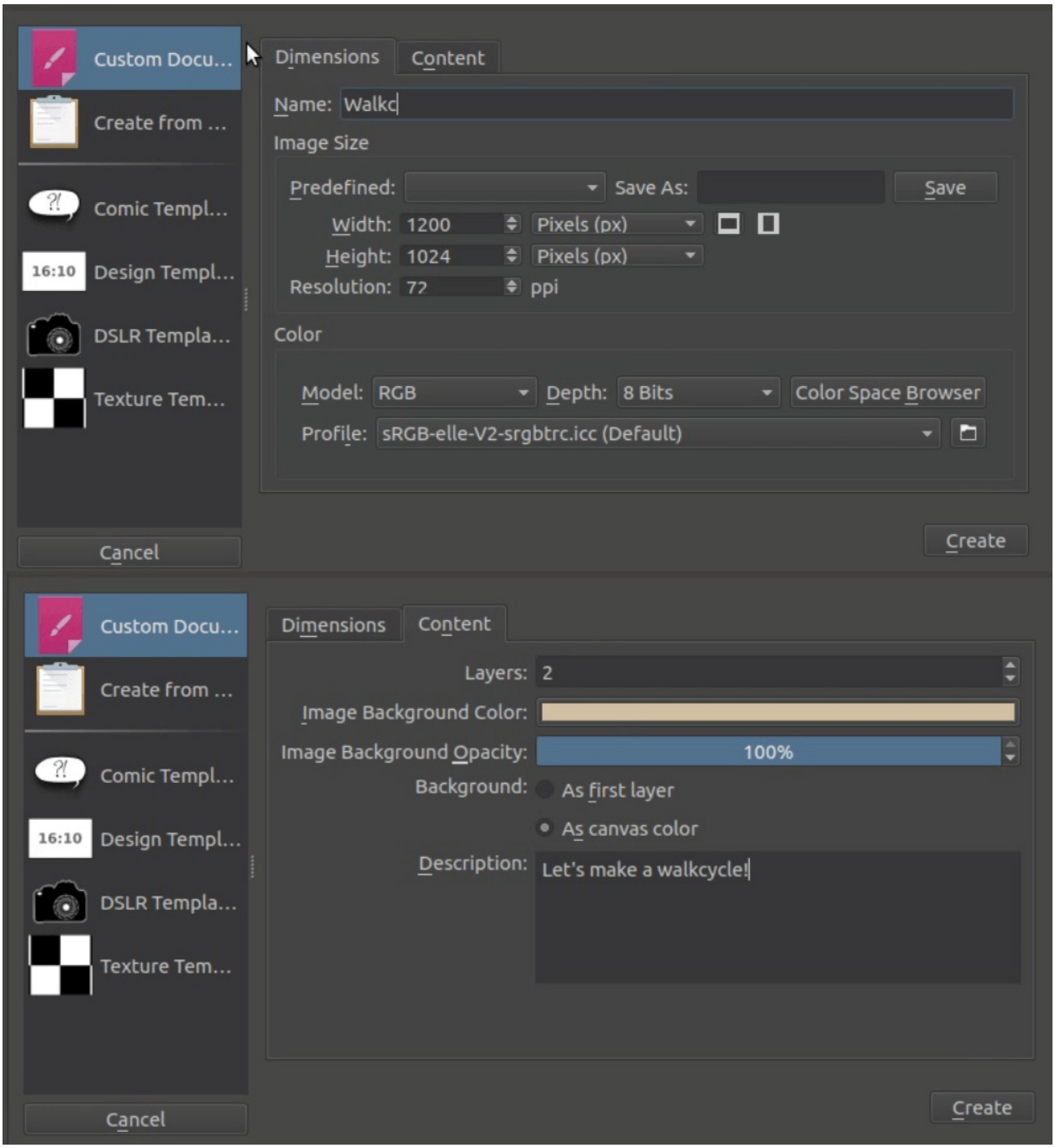
1. The Timeline Docker. View and control all of the frames in your animation. The timeline docker also contains functions to manage your layers. The layer that are created in the timeline docker also appear on the normal Layer docker.
2. The Animation Docker. This docker contains the play buttons as well as the ability to change the frame-rate, playback speed and useful little options like *auto-key framing*.
3. The Onion Skin Docker. This docker controls the look of the onion skin, which in turn is useful for seeing the previous frame.

Introduction to animation: How to make a walkcycle

The best way to get to understand all these different parts is to actually use them. Walk cycles are considered the most basic form of a full animation, because of all the different parts involved with them. Therefore, going over how one makes a walkcycle should serve as a good introduction.

Setup

First, we make a new file:



On the first tab, we type in a nice ratio like 1280x1024, set the dpi to 72 (we're making this for screens after all) and title the document 'walkcycle'.

In the second tab, we choose a nice background color, and set the background to canvas-color. This means that Krita will automatically fill in any transparent bits with the background color. You can change this in *Image* ▸ *Image Properties*. This seems to be most useful to people doing animation, as

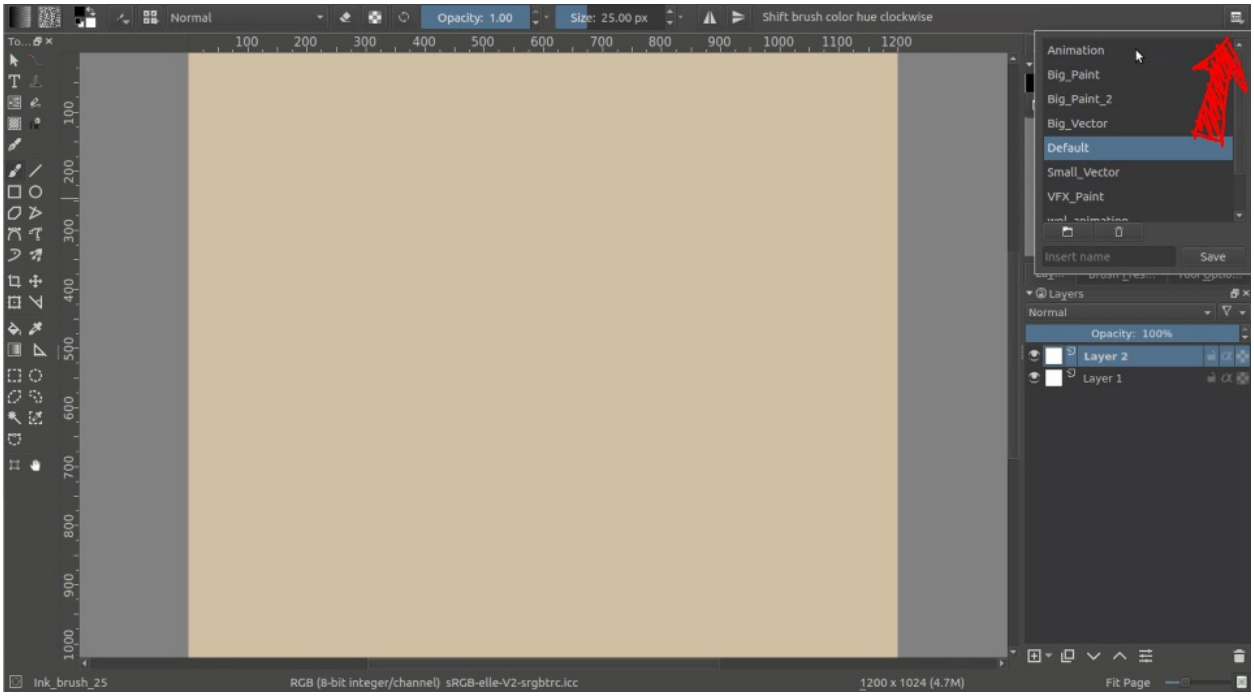
the layer you do animation on MUST be semi-transparent to get onion skinning working.

Note

Krita has a bunch of functionality for meta-data, starting at the *Create Document* screen. The title will be automatically used as a suggestion for saving and the description can be used by databases, or for you to leave comments behind. Not many people use it individually, but it can be useful for working in larger groups.

Then hit *Create*!

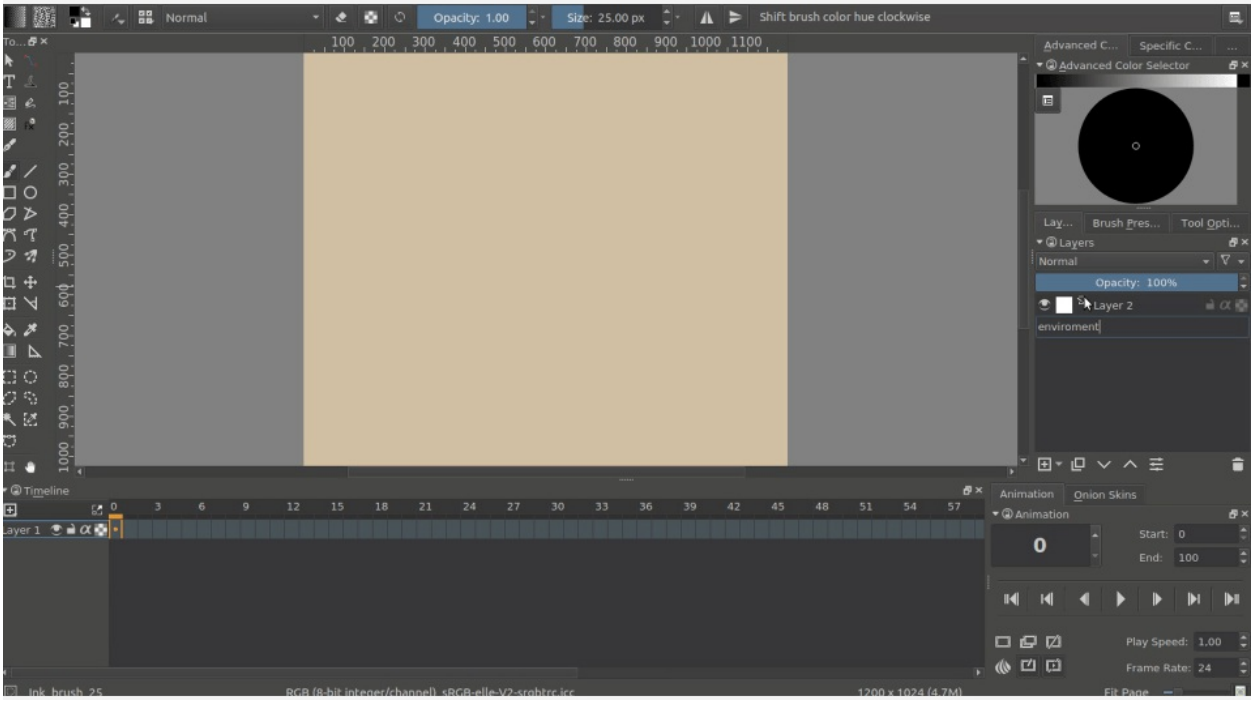
Then, to get all the necessary tools for animation, select the workspace switcher:



The red arrow points at the workspace switcher.

And select the animation workspace.

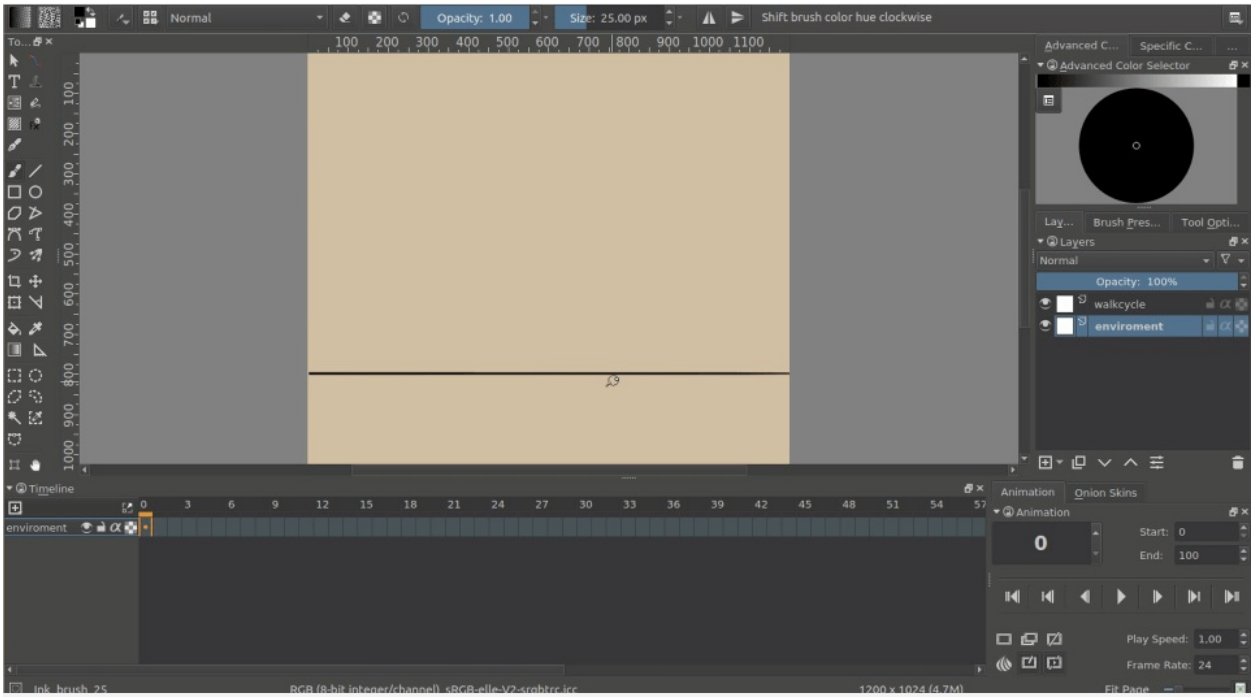
Which should result in this:



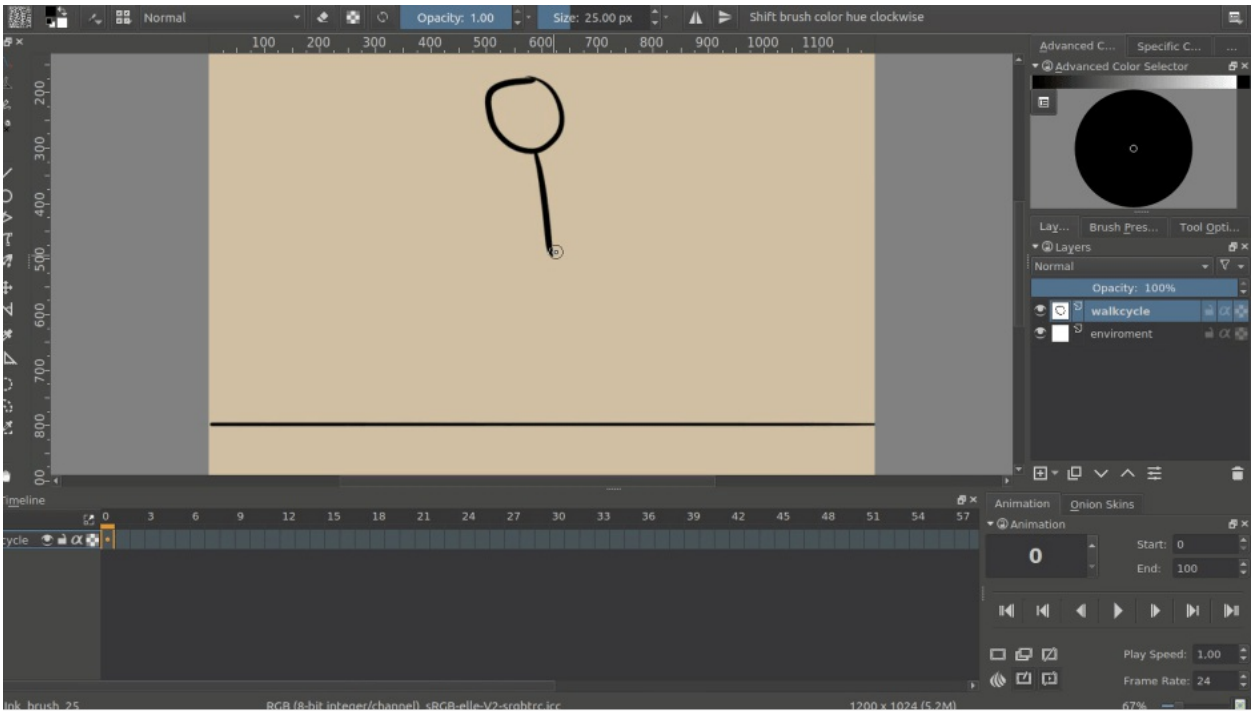
The animation workspace adds the timeline, animation and onion skin dockers at the bottom.

Animating

We have two transparent layers set up. Let’s name the bottom one ‘environment’ and the top ‘walkcycle’ by double clicking their names in the layer docker.

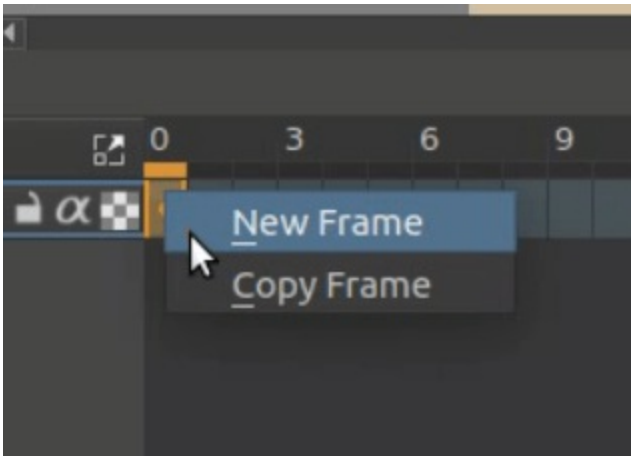


Use the straight line tool to draw a single horizontal line. This is the ground.



Then, select the ‘walkcycle’ layer and draw a head and torso (you can use any brush for this).

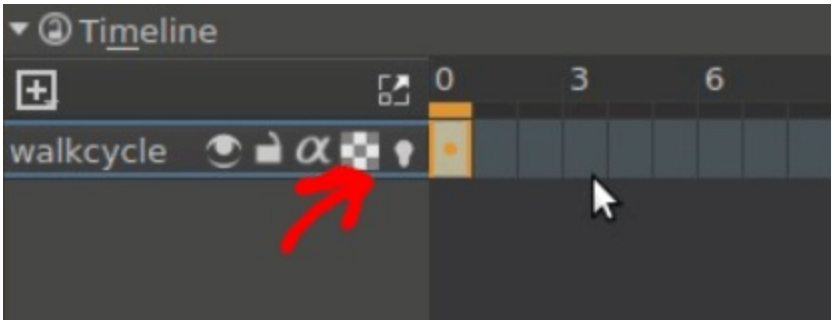
Now, selecting a new frame will not make a new frame automatically. Krita doesn't actually see the 'walkcycle' layer as an animated layer at all!



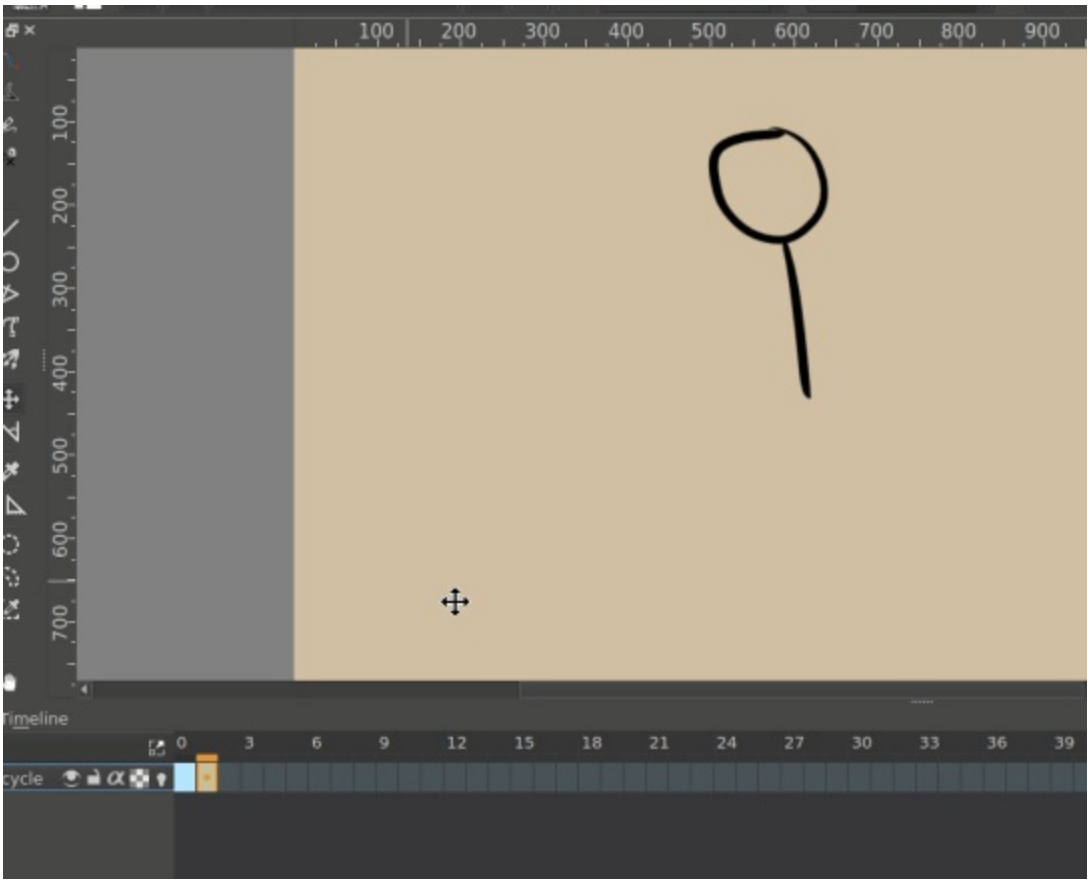
We can make it animatable by adding a frame to the timeline.  a frame in the timeline to get a context menu. Choose *Create Duplicate Frame*.

Attention

If you select *Create Blank Frame*, the content of the layer will be dropped and a new blank frame will appear; since you want to preserve the image, you need to use *Create Duplicate Frame*.

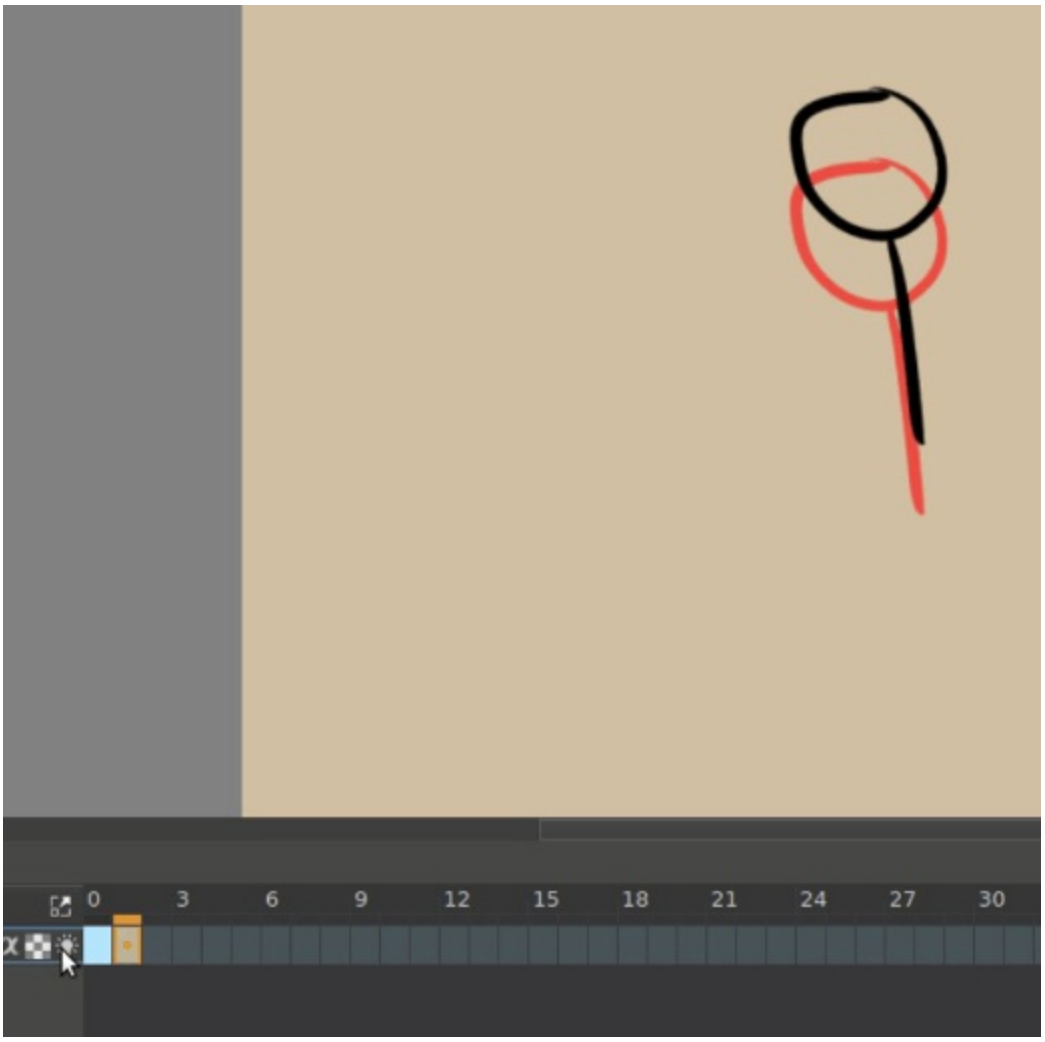


You can see it has become an animated layer because of the onion skin icon showing up in the timeline docker.



Use the *Create Duplicate Frame* button to copy the first frame onto the second. Then, use the `Move Tool` (switch to it using the `T` shortcut) with the `Shift + ↑` shortcut to move the frame contents up.

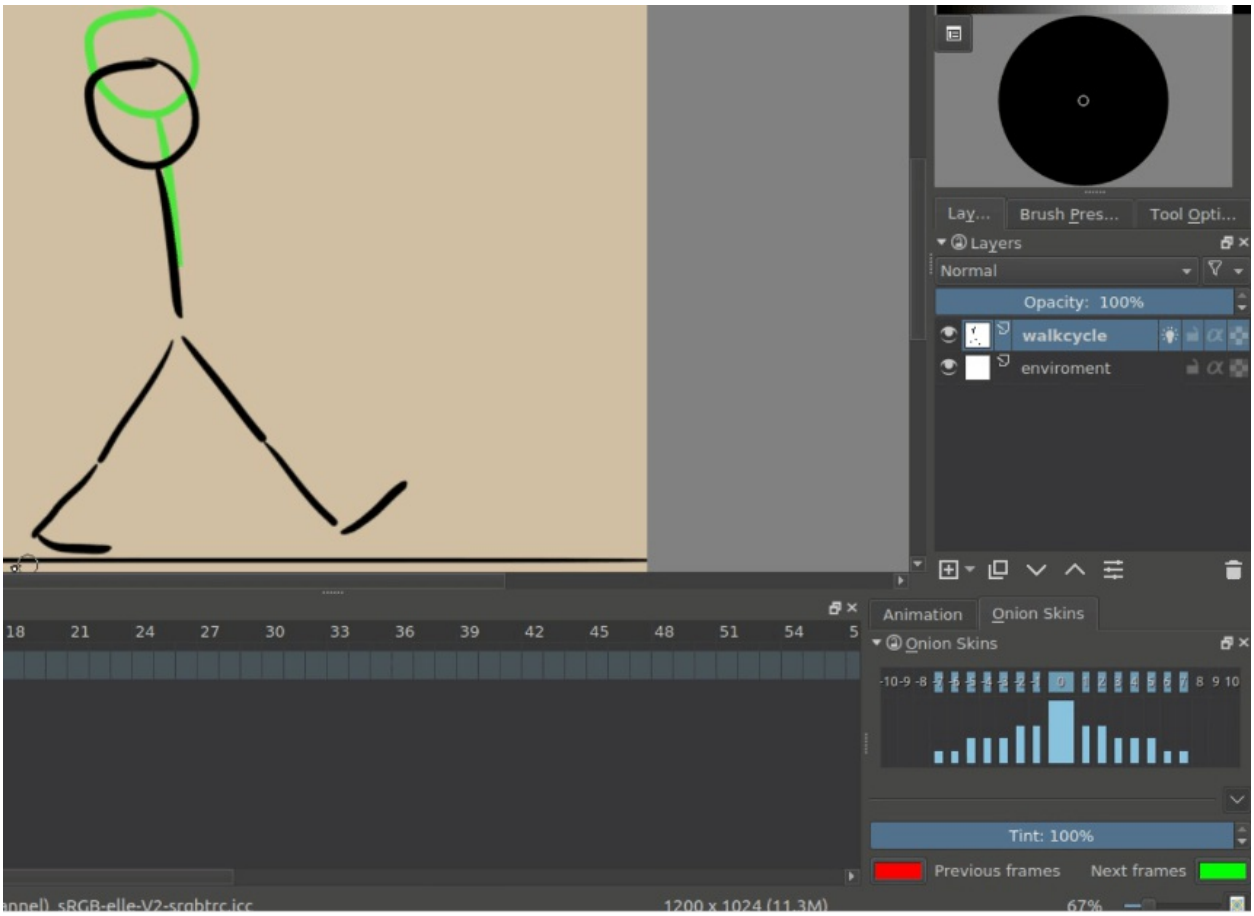
We can see the difference by turning on the onionskinning:



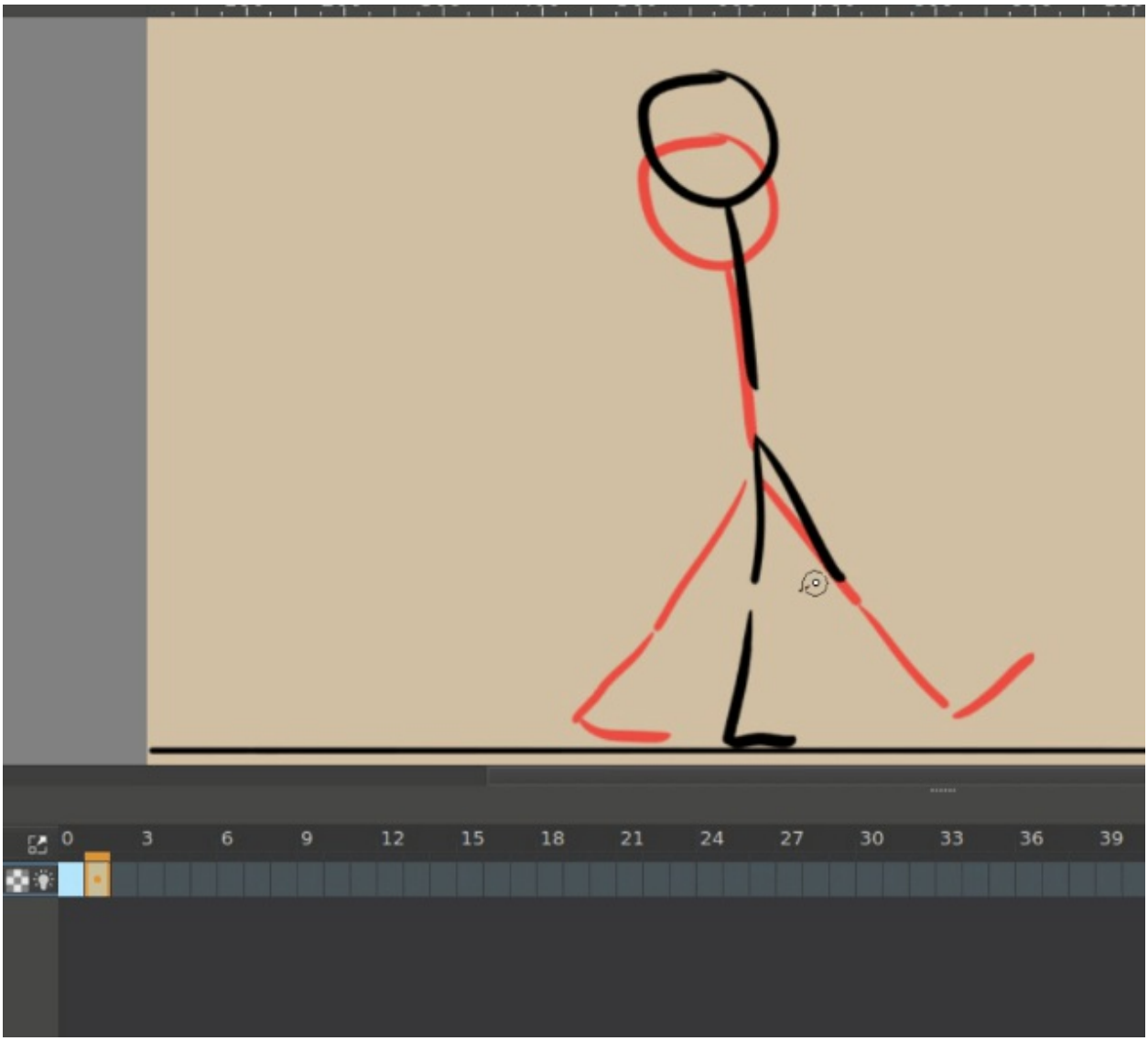
Now, you should see the previous frame as red.

Warning

Krita sees white as a color, not as transparent, so make sure the animation layer you are working on is transparent in the bits where there’s no drawing. You can fix the situation by use the Color to Alpha filter, but prevention is best.

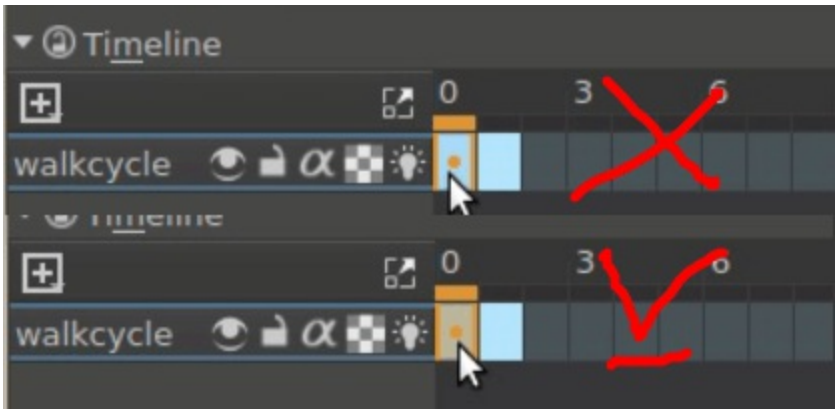


Future frames are drawn in green, and both colors can be configured in the onion skin docker.



Now, we’re gonna draw the two extremes of the walkcycle. These are the pose where both legs are as far apart as possible, and the pose where one leg is full stretched and the other pulled in, ready to take the next step.

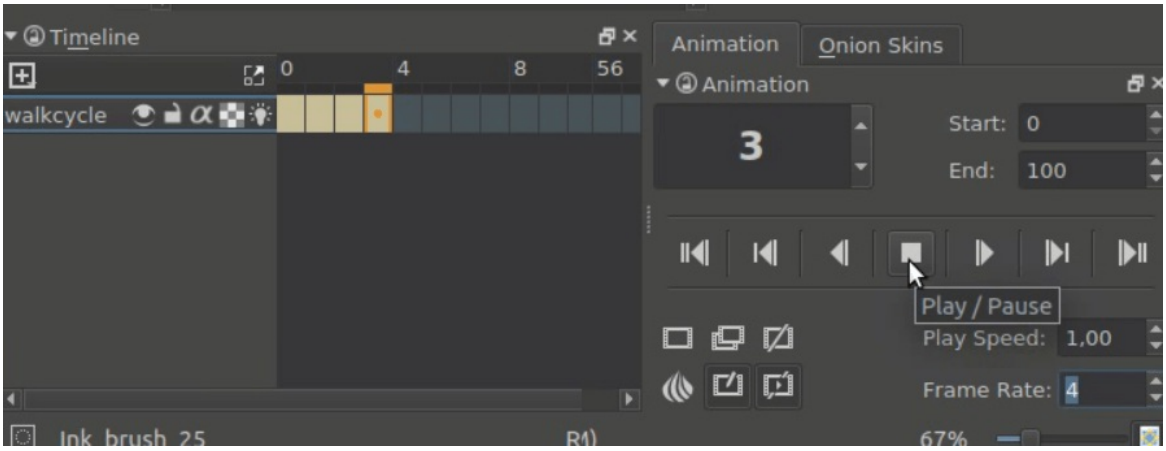
Now, let’s copy these two… We could do that with the `Ctrl + drag` shortcut, but here comes a tricky bit:



`Ctrl +`  also selects and deselects frames, so to copy…

- `Ctrl +`  to select all the frames you want to select.
- `Ctrl + drag`. You need to make sure the first frame is ‘orange’, otherwise it won’t be copied along.

Now then…



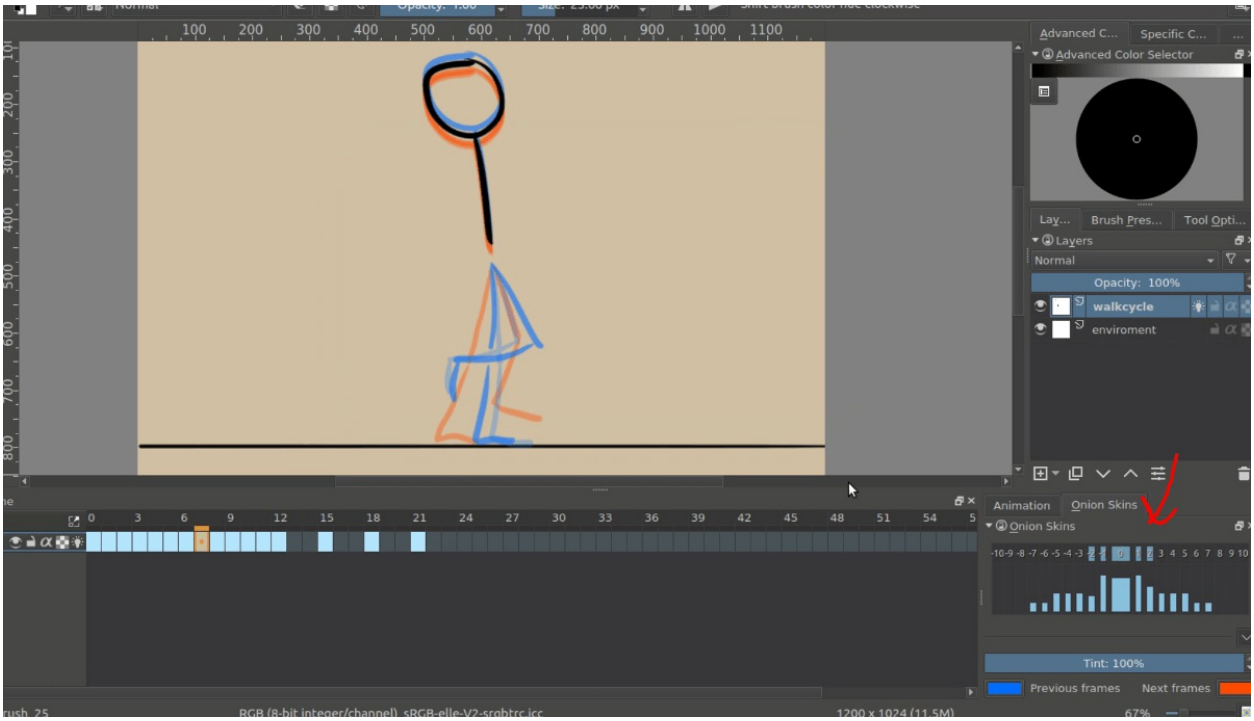
Squashed the timeline docker a bit to save space.

1. Copy frame 0 to frame 2.
2. Copy frame 1 to frame 3.
3. In the animation docker, set the frame-rate to 4.
4. Select all frames in the timeline docker by dragging-selecting them.
5. Press play in the animation docker.
6. Enjoy your first animation!

Expanding upon your rough walkcycle

You can quickly make some space by the `Alt + drag` shortcut on any frame. This'll move that frame and all others after it in one go.

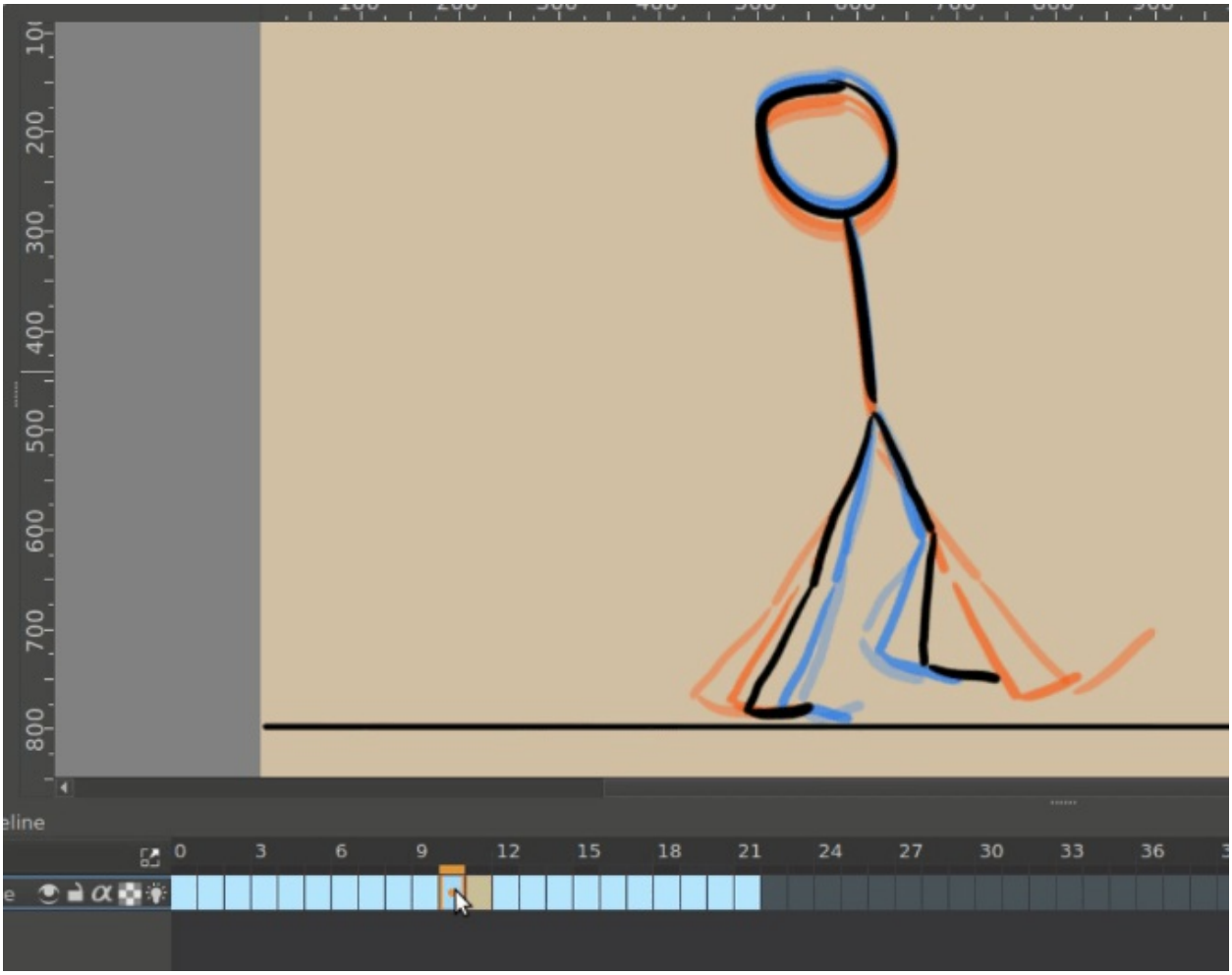
Then draw inbetweens on each frame that you add.



You'll find that the more frames you add, the more difficult it becomes to keep track of the onion skins.

You can modify the onion skin by using the onion skin docker, where you

can change how many frames are visible at once, by toggling them on the top row. The bottom row is for controlling transparency, while below there you can modify the colors and extremity of the coloring.

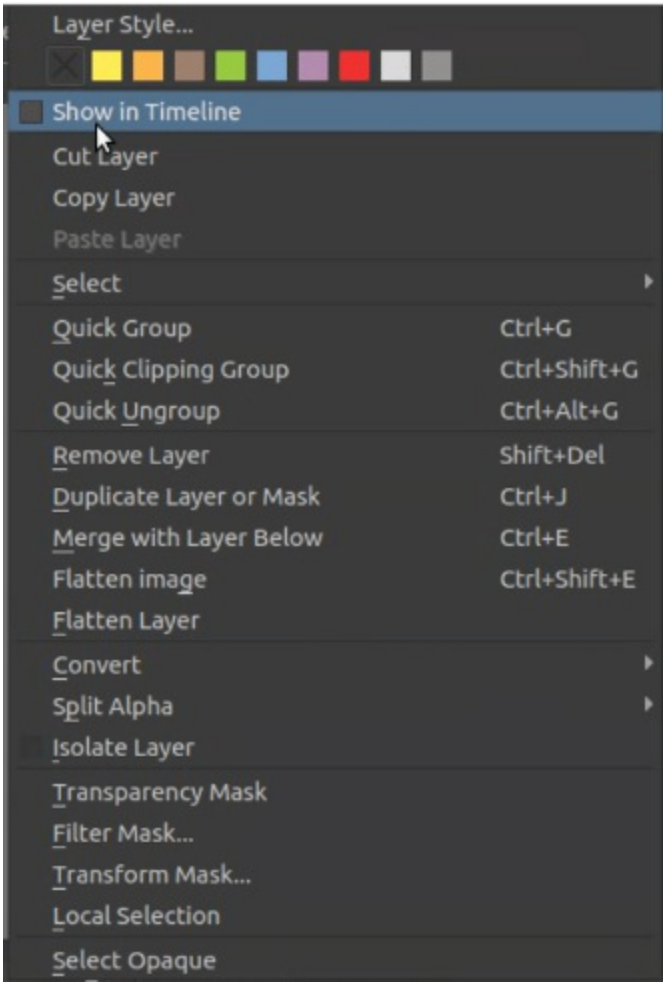


Animating with multiple layers

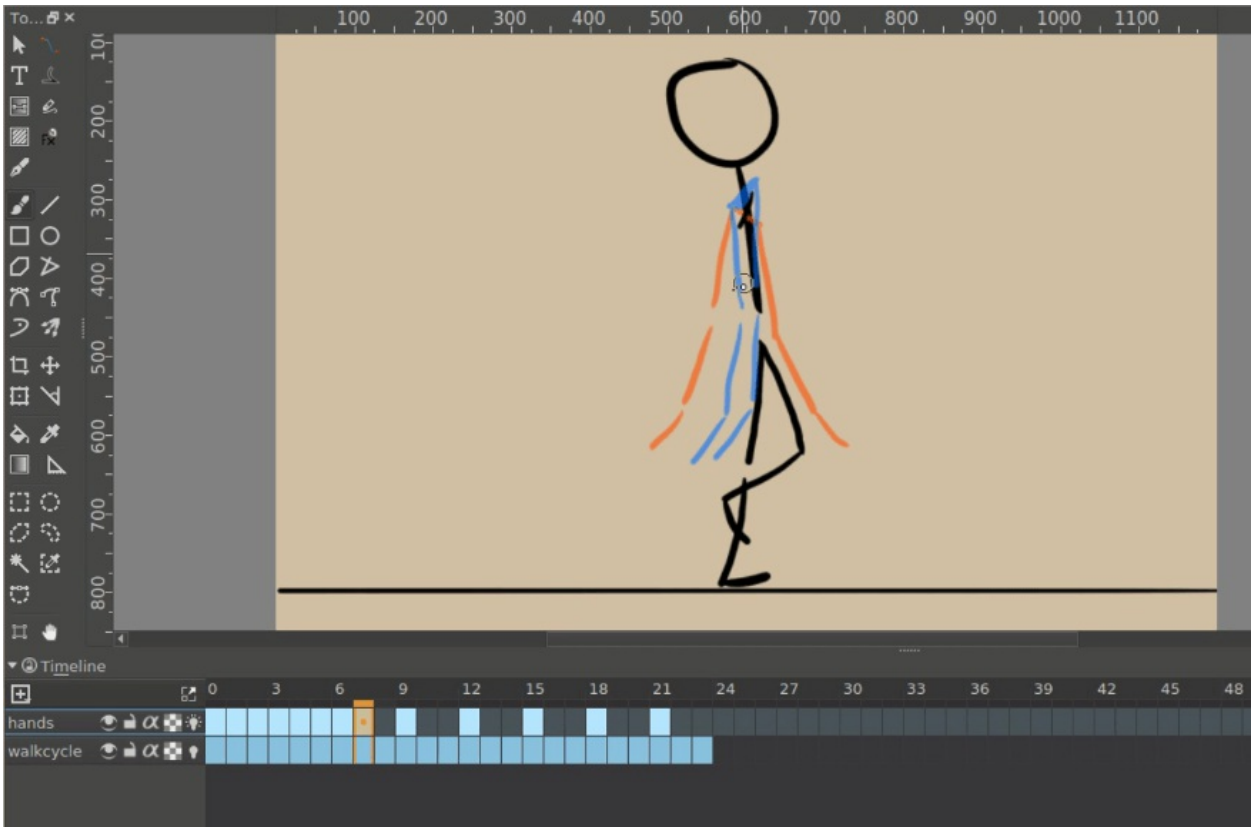
Okay, our walkcycle is missing some hands, let’s add them on a separate layer. So we make a new layer, and name it hands and…

Our walkcycle is gone from the timeline docker! This is a feature actually. A full animation can have so many little parts that an animator might want to remove the layers they're not working on from the timeline docker. So you manually have to add them.

New in version 4.3.0: In **Krita 4.3.0** and later, all new layers are pinned to the timeline by default.

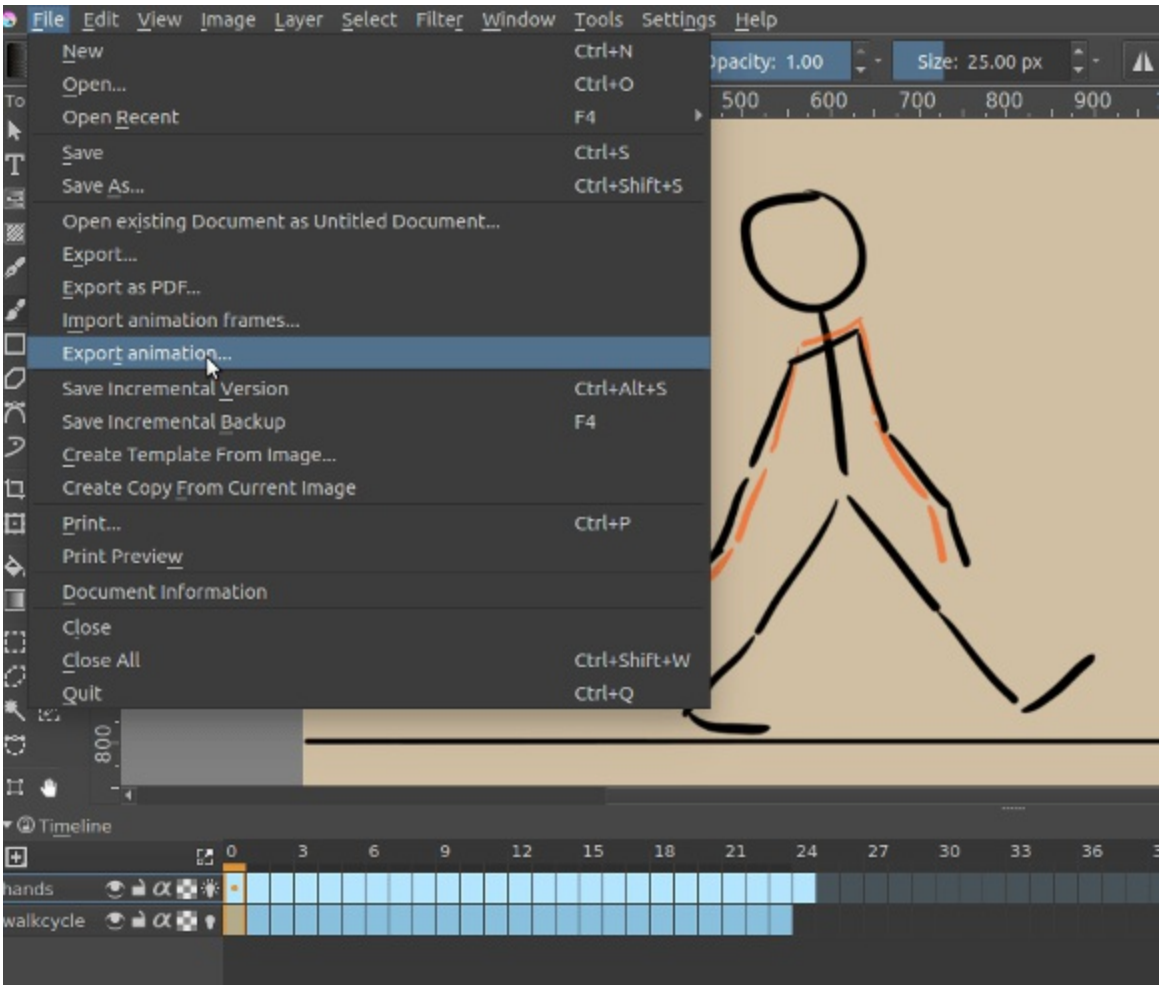


To show a layer whether it's active or not, you can "pin" it to the timeline by right-clicking  on the layer in the layer docker, and toggling *Pin to Timeline*. We recommend pinning any layers that you're currently animating on.

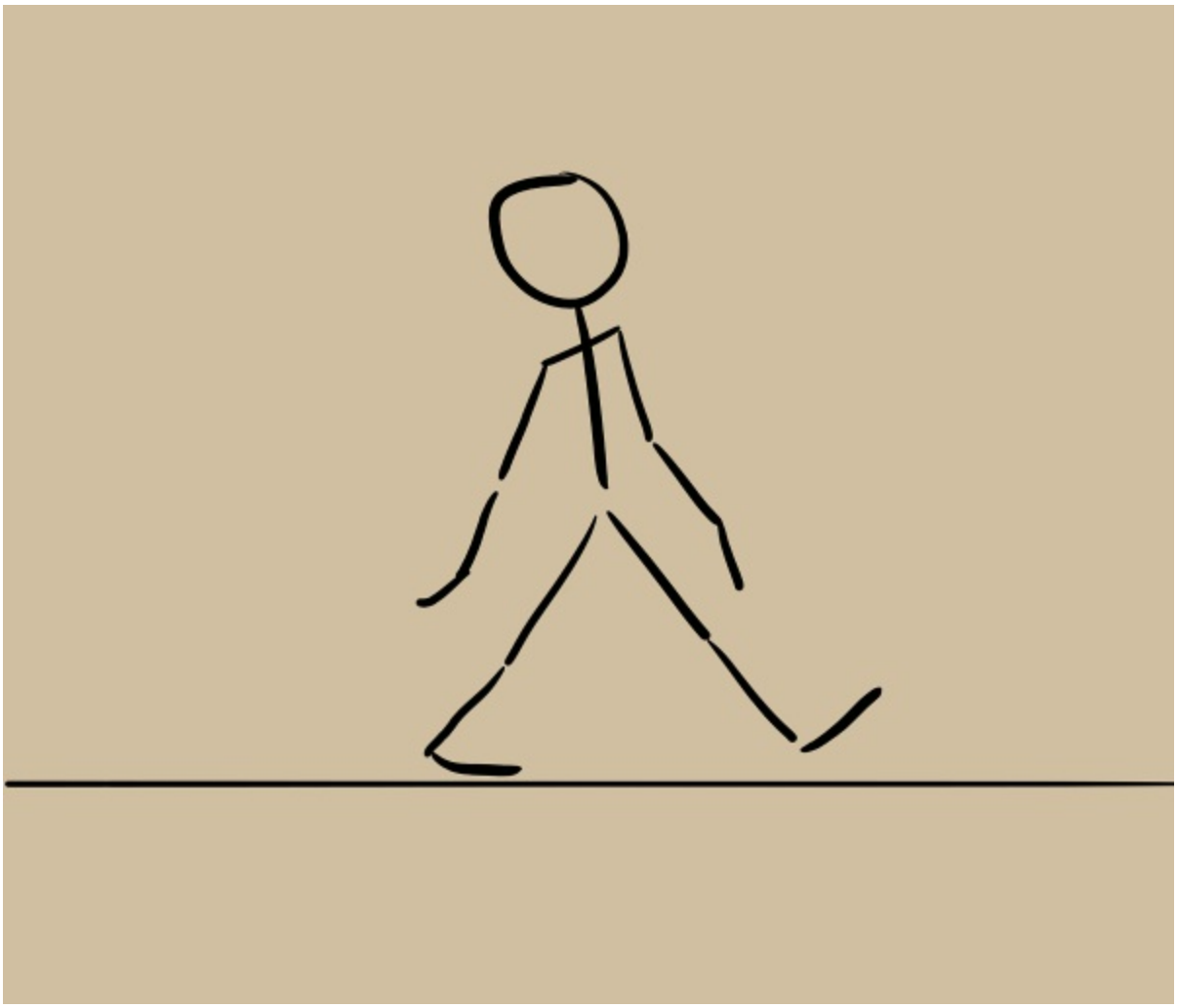


Exporting

When you are done, select *File* ▸ *Render Animation*. To render to a video file, you'll need a program called `FFmpeg`. To learn more, please read Render Animation.



Enjoy your walkcycle!

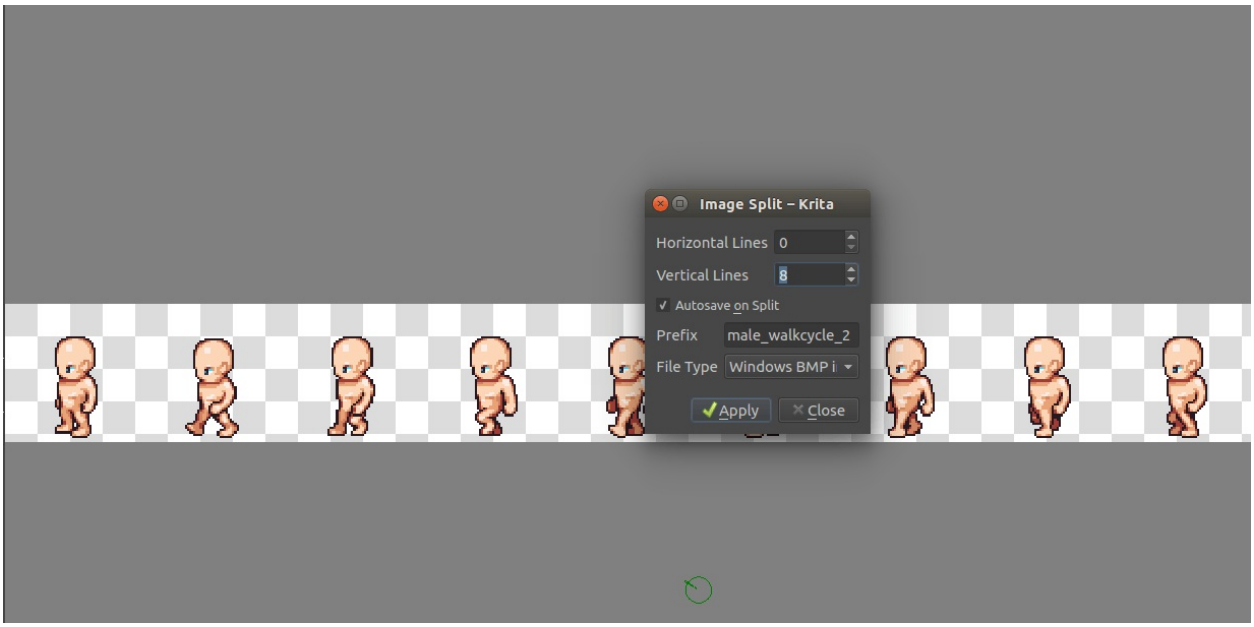


Importing animation frames

In Krita you can import animation frames.

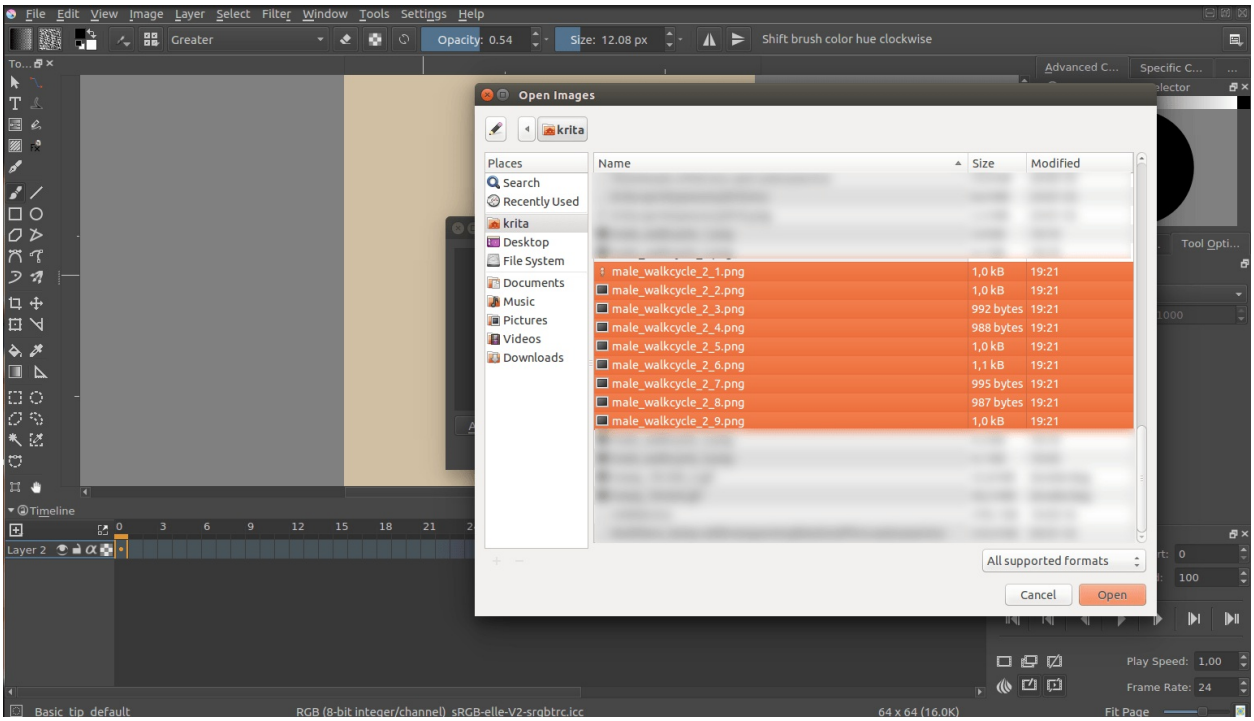
First let us take a sprite sheet from Open Game Art. (This is the Libre Pixel Cup male walkcycle).

We'll use *Image* ▸ *Split Image* to split up the sprite sheet.

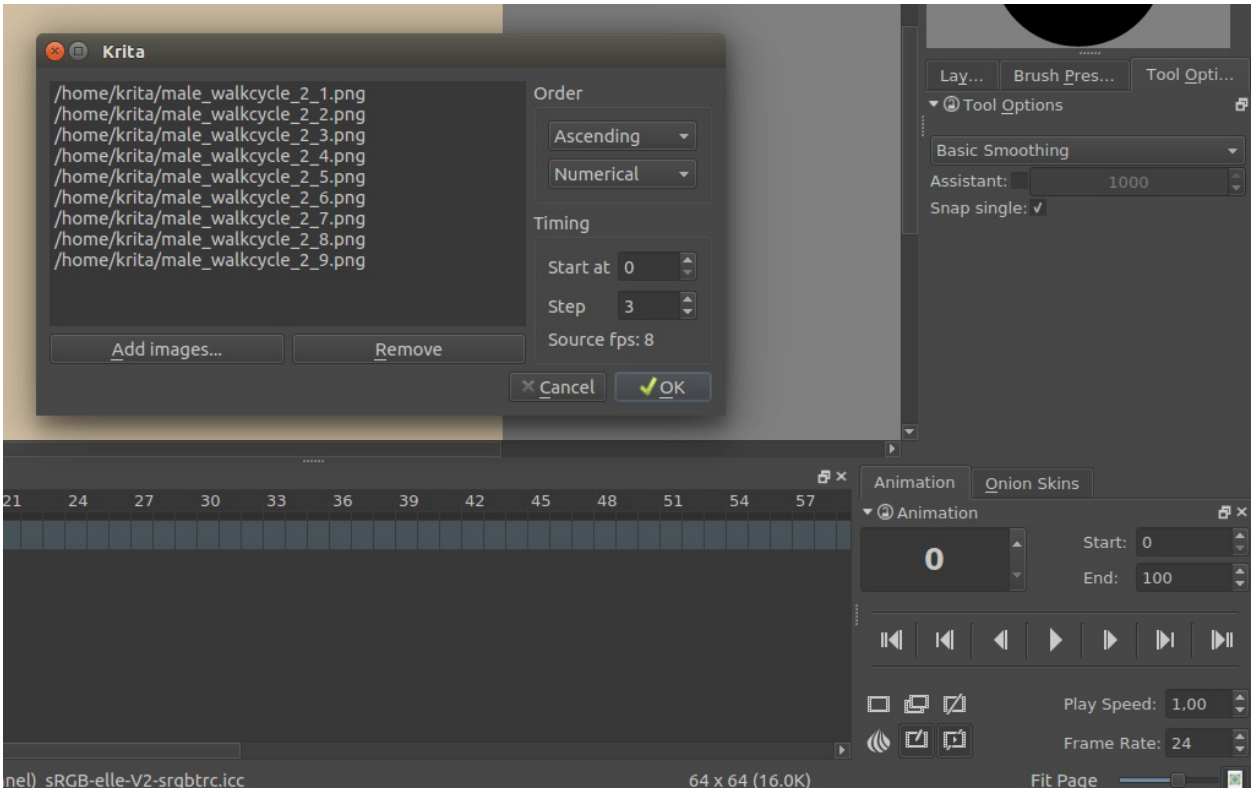


The slices are even, so for a sprite sheet of 9 sprites, use 8 vertical slices and 0 horizontal slices. Give it a proper name and save it as png.

Then, make a new canvas, and select *File* ▸ *Import Animation Frames*. This will give you a little window. Select *Add images*. This should get you a file browser where you can select your images.



You can select multiple images at once.

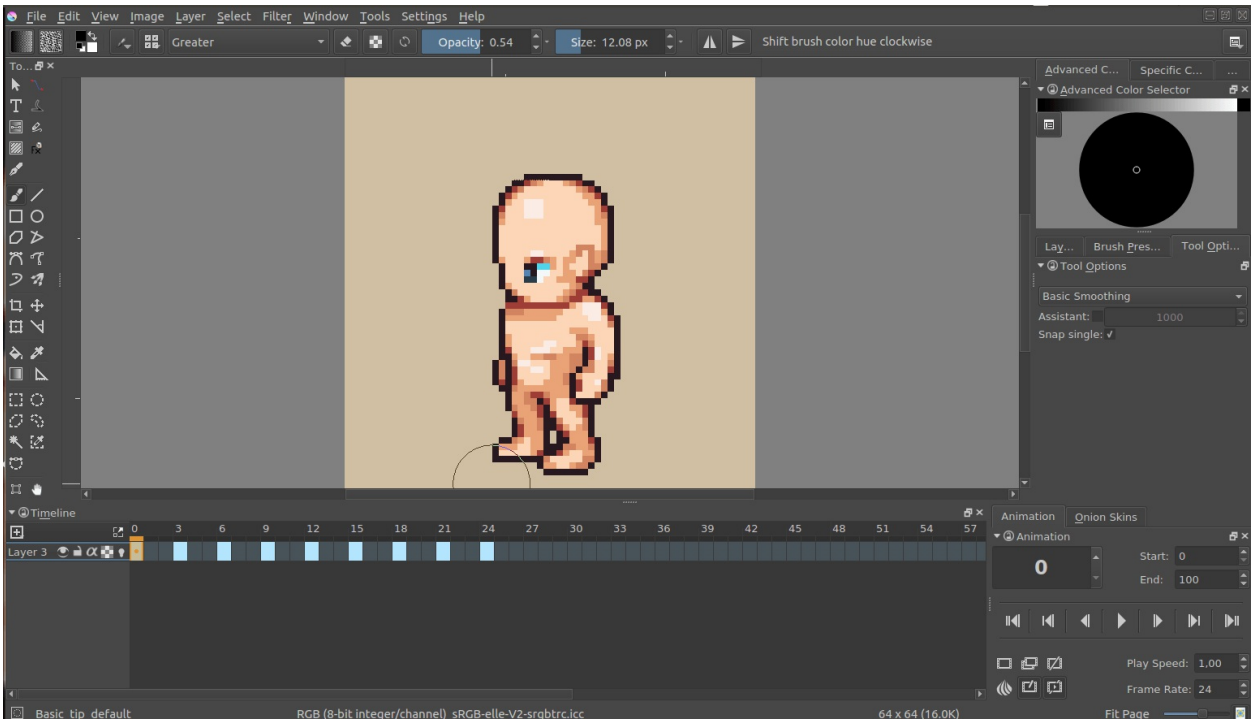


The frames are currently automatically ordered. You can set the ordering with the top-left two drop-down boxes.

Start
: Indicates at which point the animation should be imported.

Step
: Indicates the difference between the imported animation and the document frame rate. This animation is 8 frames big, and the fps of the document is 24 frames, so there should be a step of 3 to keep it even. As you can see, the window gives feedback on how much fps the imported animation would be with the currently given step.

Press *OK*, and your animation should be imported as a new layer.



Reference

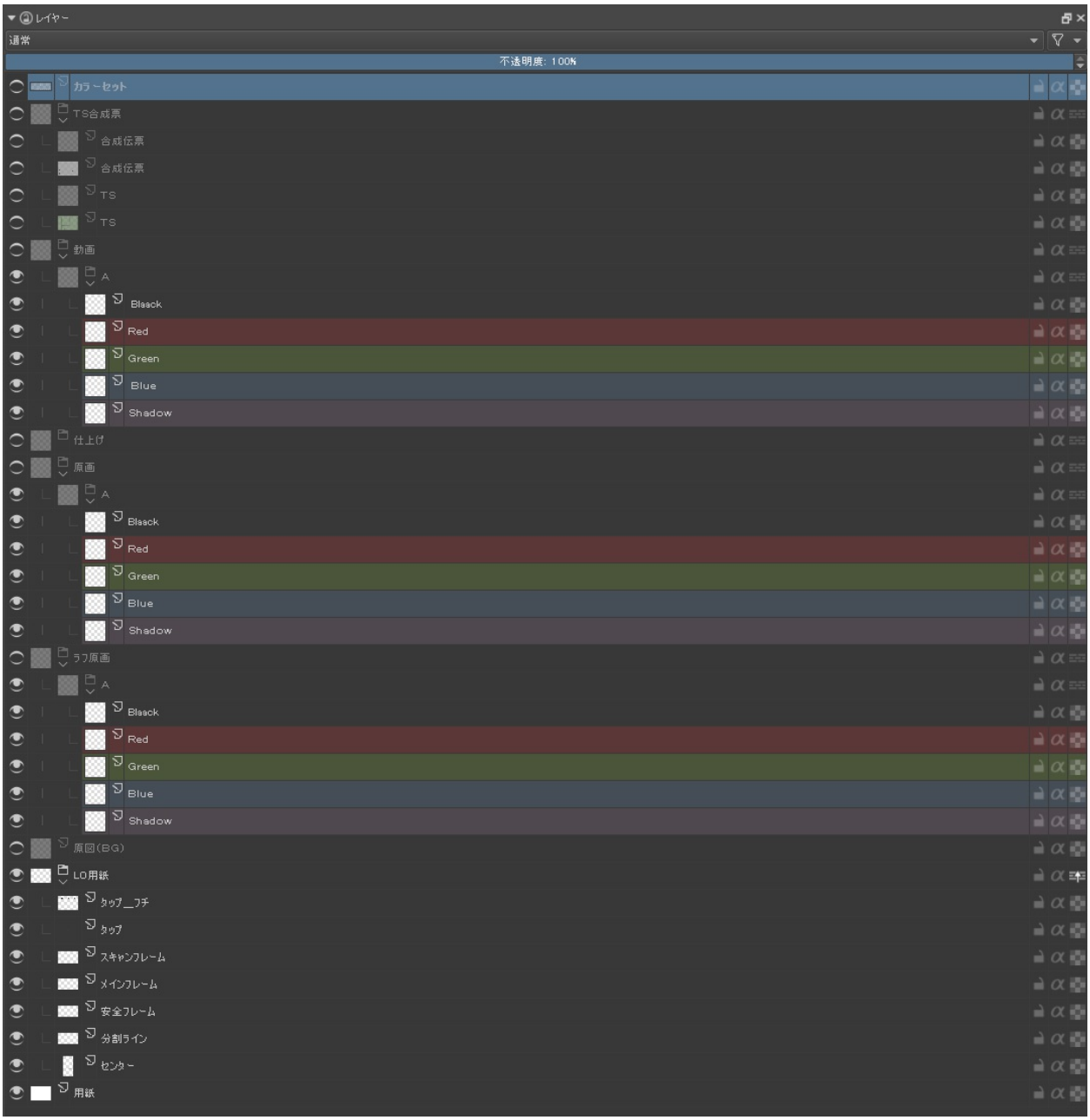
- https://community.kde.org/Krita/Docs/AnimationGuiFeaturesList
- The source for the libre pixel cup male walkmediawiki cycle
  [https://opengameart.org/content/liberated-pixel-cup-lpc-base-assets-sprites-map-tiles]

Japanese Animation Template

This template is used to make Japanese-style animation. It is designed on the assumption that it was used in co-production, so please customize its things like layer folders according to scale and details of your works.

Basic structure of its layers

Layers are organized so that your work will start from lower layers go to higher layers, except for coloring layers.



Its layer contents

from the bottom

Layout Paper

These layers are a form of layout paper. Anime tap holes are prepared on separate layers in case you have to print it out and continue your drawing traditionally.

Layout (Background)

These layers will contain background scenery or layouts which are scanned from a traditional drawing. If you don’t use them, you can remove them.

Key drafts

These layers are used to draw layouts digitally.

Keys

Where you add some details to the layouts and arrange them to draw “keys” of animation.

Inbetweening

Where you add inbetweens to keys for the process of coloring, and remove unnecessary details to finalize keys (To be accurate, I finish finalization of keys before beginning to add inbetweens).

Coloring (under Inbetweening)

Where you fill areas with colors according to specification of inbetweens.

Time Sheet and Composition sheet

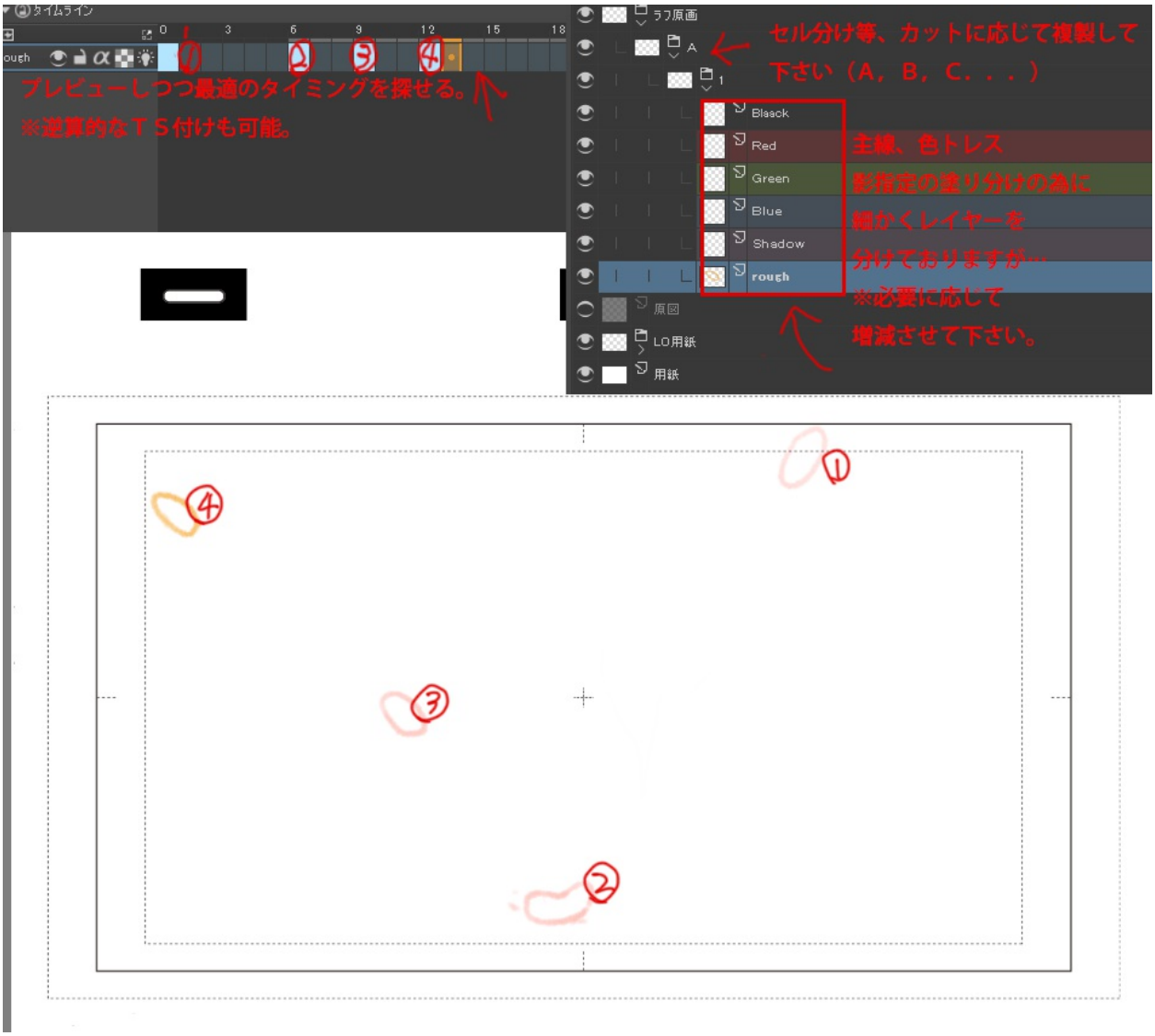
This contains a time sheet and composition sheet. Please rotate them before using.

Color set

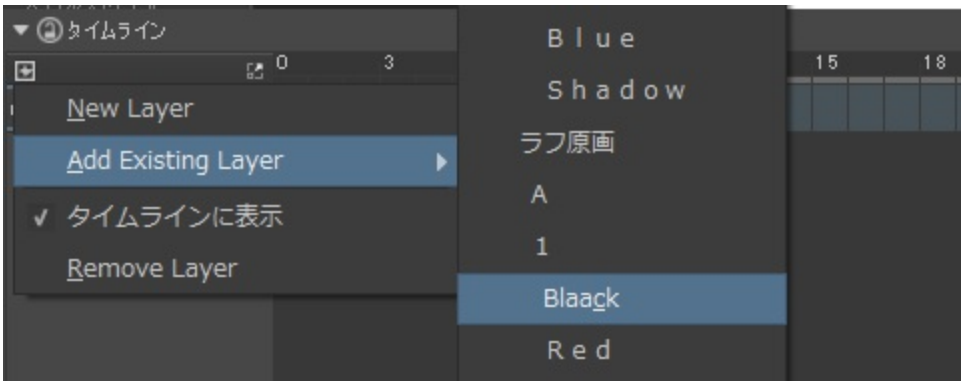
This contains colors used to draw main and auxiliary line art and fill highlight or shadows. You can add them to your palette.

Basic steps to make animation

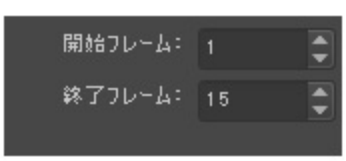
Key draft –> assign them into Time sheet (or adjust them on Timeline, then assign them into Time sheet) –> adjust them on Timeline –> add frames to draw drafts for inbetweening if you need them –> Start drawing Keys



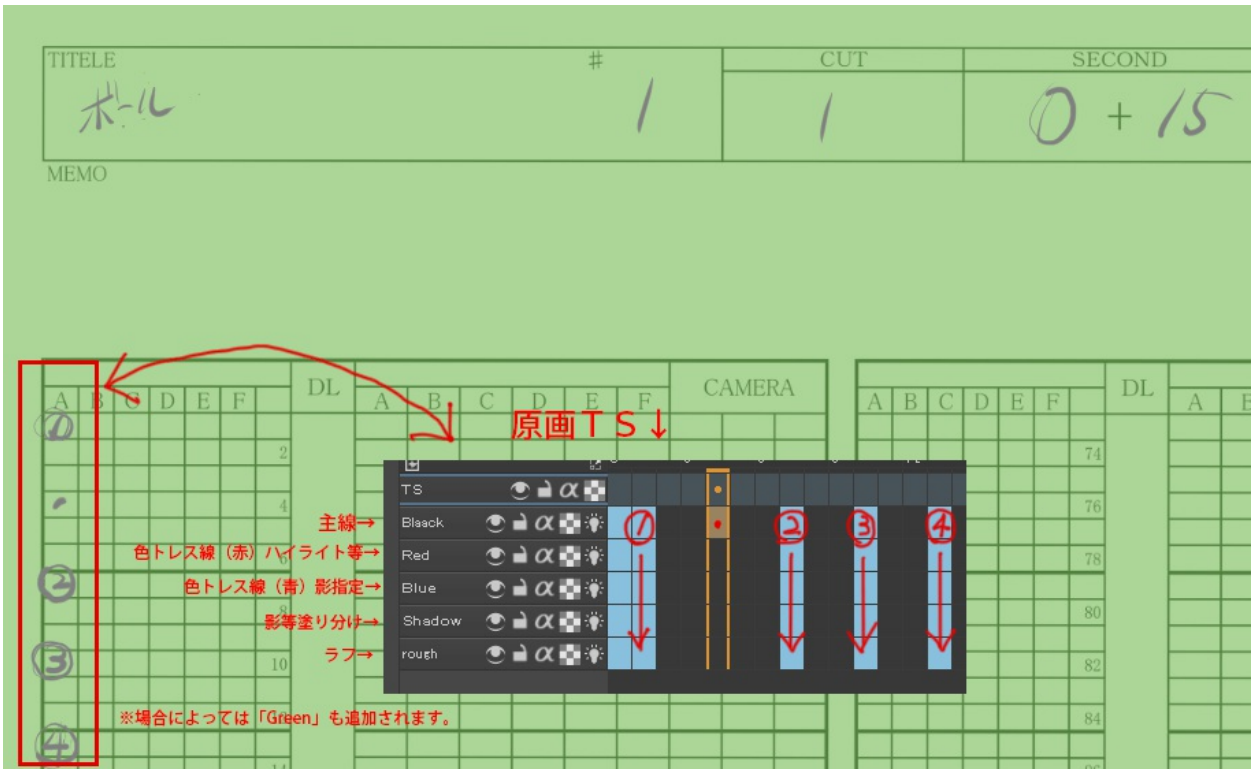
You can add layers and add them to timeline.



This is due difference between 24 drawing per second, which is used in Full Animation, and 12 drawing per second and 8 drawings per second, which are used in Limited Animation, on the Timeline docker.



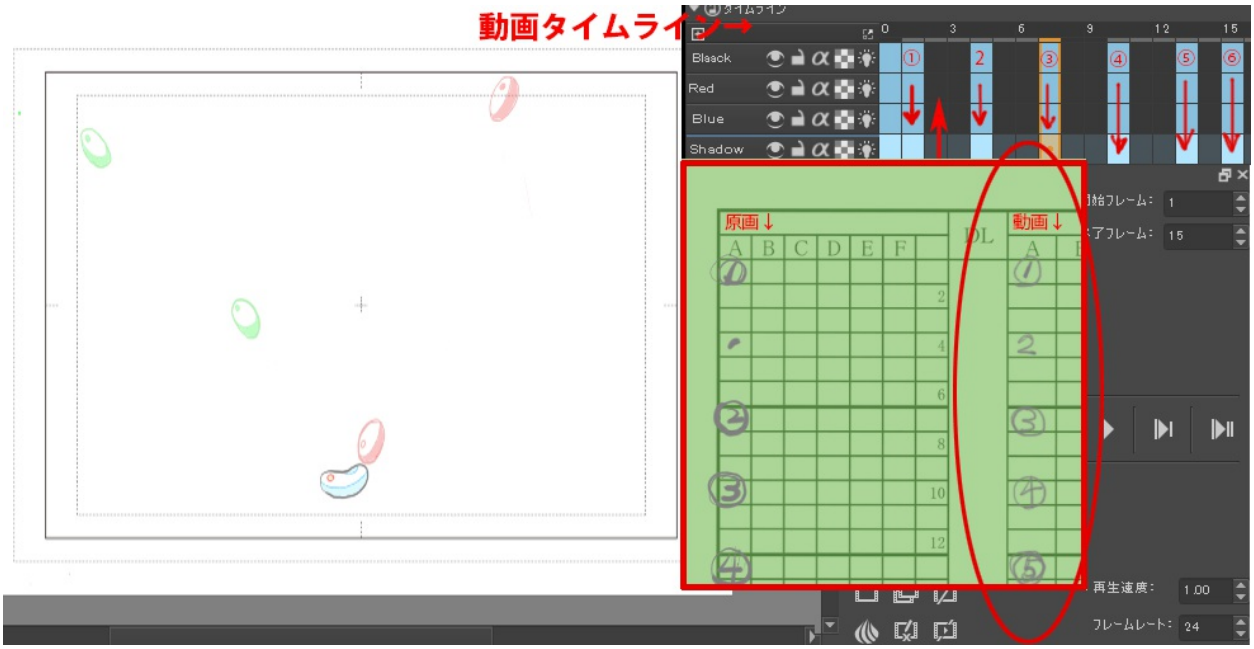
This is correspondence between Timeline and Time sheet. “Black” layer is to draw main line art which are used ordinary line art, “Red” layer is to draw red auxiliary linearts which are used to specify highlights, “Blue” layer is to draw blue auxiliary linearts which are used to specify shadows, and “Shadow” layer is to draw light green auxiliary line art which are used to specify darker shadows. However, probably you have to increase or decrease these layers according to your work.



Finished keys, you will begin to draw the inbetweens. If you feel Krita is becoming slow, I recommend you to merge key drafts and keys, as well as to remove any unnecessary layers.

After finalizing keys and cleaning up unnecessary layers, add inbetweenings, using Time sheet and inbetweening drafts as reference.

This is its correspondence with Time sheet.



Once the vector functionality of Krita becomes better, I recommend you to use vector to finalize inbetweening.

If you do the colors in Krita, please use Coloring group layer. If you do colors in other software, I recommend to export frames as .TGA files.

Resolution

I made this template in 300 dpi because we have to print them to use them in traditional works which still fill an important role in Japanese Anime Studio. However, if you stick to digital, 150-120 dpi is enough to make animation. So you can decrease its resolution according to your need.

Originally written by Saisho Kazuki, Japanese professional animator, and translated by Tokiedian, KDE contributor.

Gamut Masks

New in version 4.2.

Gamut masking is an approach to color formalized by James Gurney, based on the idea that any color scheme can be expressed as shapes cut out from the color wheel.

It originates in the world of traditional painting, as a form of planning and premixing palettes. However, it translates well into digital art, enabling you to explore and analyze color, plan color schemes and guide your color choices.

How does it work?

You draw one or multiple shapes on top of the color wheel. You limit your color choices to colors inside the shapes. By leaving colors out, you establish a relationship between the colors, thus creating harmony.

Gamut masking is available in both the Advanced and Artistic Color Selectors.

See also

- Color Wheel Masking, Part 1 by James Gurney
  [https://gurneyjourney.blogspot.com/2008/01/color-wheel-masking-part-1.html]
- The Shapes of Color Schemes by James Gurney
  [https://gurneyjourney.blogspot.com/2008/02/shapes-of-color-schemes.html]
- Gamut Masking Demonstration by James Gourney (YouTube)
  [https://youtu.be/qfE4E5goEIc]

Selecting a gamut mask

For selecting and managing gamut masks open the Gamut Masks Docker:

Settings ▸ *Dockers* ▸ *Gamut Masks*.

Gamut Masks docker

In this docker you can choose from several classic gamut masks, like the ‘Atmospheric Triad’, ‘Complementary’, or ‘Dominant Hue With Accent’. You can also duplicate those masks and make changes to them (3,4), or create new masks from scratch (2).

Clicking the thumbnail icon (1) of the mask applies it to the color selectors.

See also

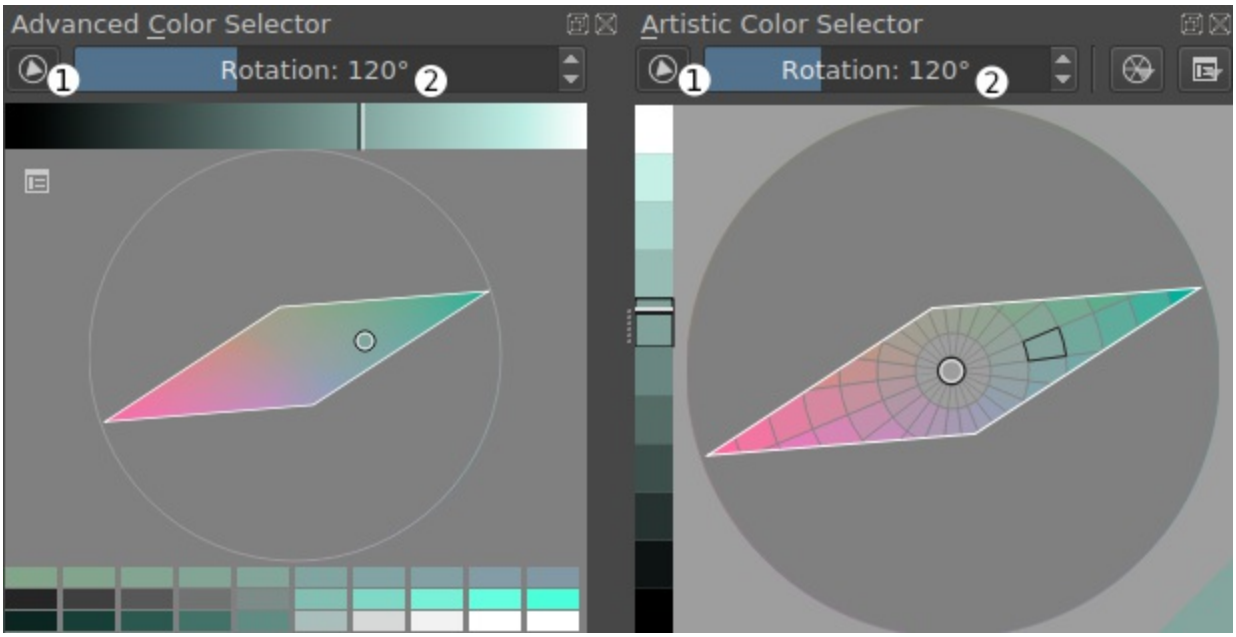
- Gamut Masks Docker

In the color selector

You can rotate an existing mask directly in the color selector, by dragging the

rotation slider on top of the selector (2).

The mask can be toggled off and on again by the toggle mask button in the top left corner (1).



Advanced and Artistic color selectors with a gamut mask

See also

- Artistic Color Selector Docker
- Advanced Color Selector

Editing/creating a custom gamut mask

Tip

To rotate a mask around the center point use the rotation slider in the color selector.

If you choose to create a new mask, edit, or duplicate selected mask, the

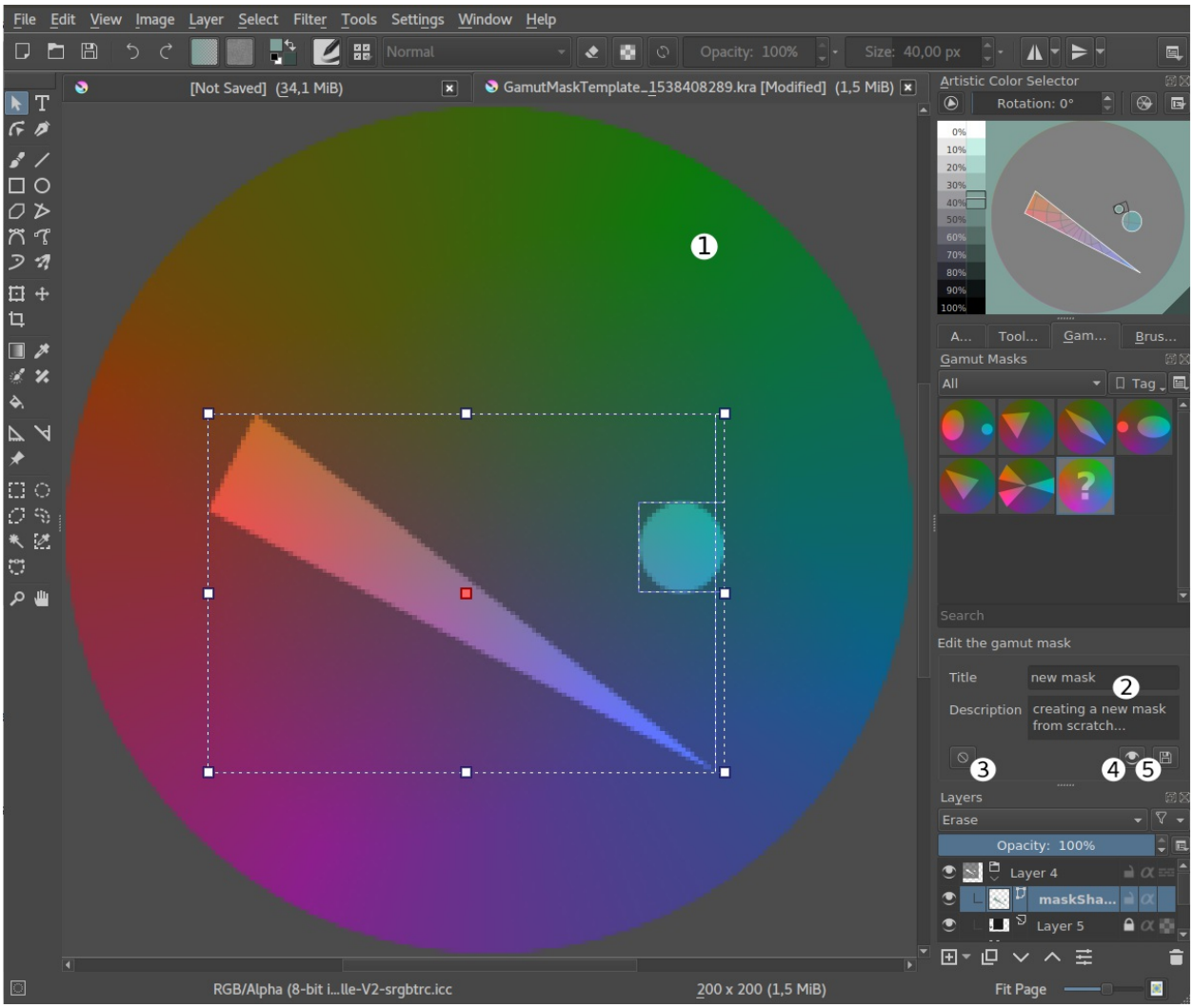
mask template documents open as a new view (1).

There you can create new shapes and modify the mask with standard vector tools (Vector Graphics). Please note, that the mask is intended to be composed of basic vector shapes. Although interesting results might arise from using advanced vector drawing techniques, not all features are guaranteed to work properly (e.g. grouping, vector text, etc.).

Warning

Transformations done through the transform tool or layer transform cannot be saved in a gamut mask. The thumbnail image reflects the changes, but the individual mask shapes do not.

You can *Preview* the mask in the color selector (4). If you are satisfied with your changes, *Save* the mask (5). *Cancel* (3) will close the editing view without saving your changes.



Editing a gamut mask

Importing and exporting

Gamut masks can be imported and exported in bundles in the Resource Manager. See Resource Management for more information.

General Concepts

Learn about general art and technology concepts that are not specific to Krita.

Contents:

Colors

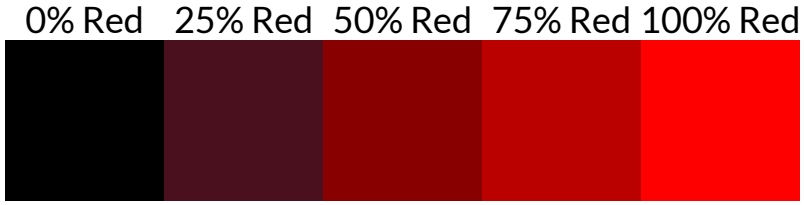
Okay, so… Let's talk **colors**!

Colors are pretty, and they're also *pretty* fundamental to painting. When painting, we want to be able to access and manipulate colors easily to do fun stuff like mixing them together or matching them to create visual **harmony** or **contrast**. We also want to be able to quickly find our favorite shades of red or favorite tints of blue without thinking or working too hard. All of this becomes even more important the more colors we have access to!

Naturally, the first thing we do is organize the colors, usually based on what we see in nature. For example, we tend to order hues in the order that they appear in a rainbow, and we think about brightness of values as a tonal range from white to black. Of course, nature itself is tied to physics, and the order of hues and the concept of brightness has everything to do with the wavelength and energy of light as it bounces around and eventually enters our eyes.

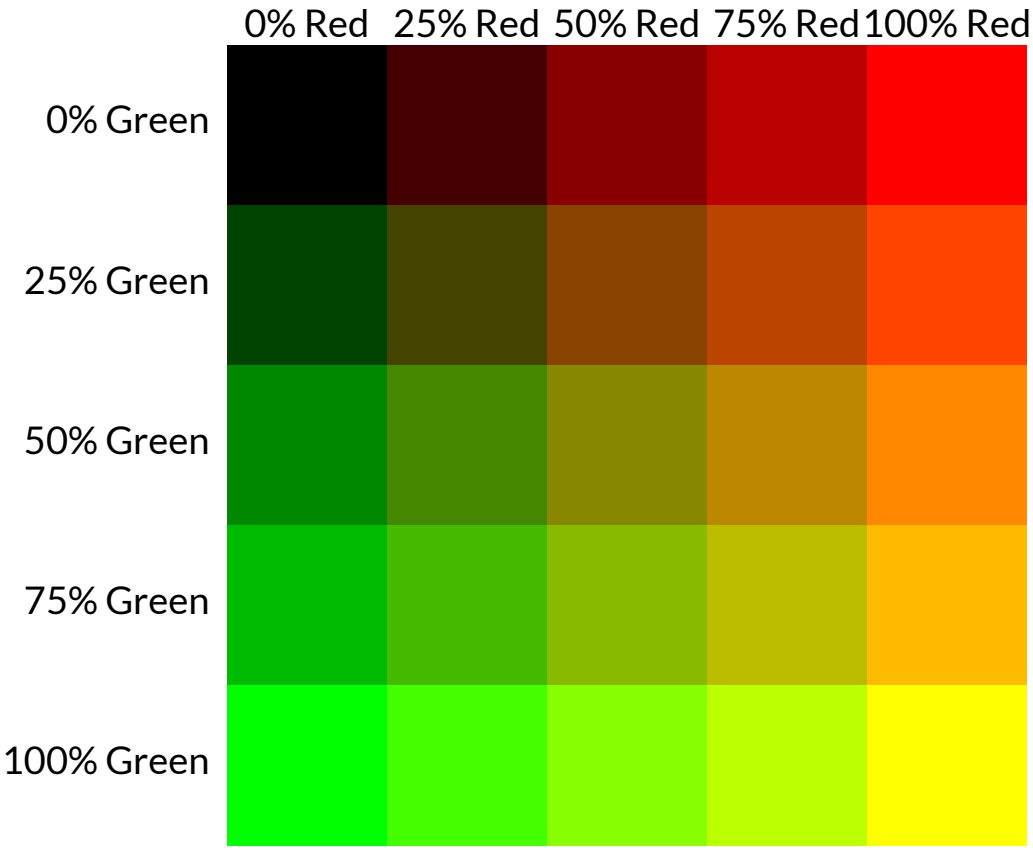
In the case of *traditional media*, we order the colors (**hues**) by how they result from mixes of other colors, starting with the *subtractive* **primary colors**: cyan, magenta, yellow. Mixing each primary color with each other reveals three secondary colors: violet, orange, and green. Mixing between those colors creates tertiary colors, and so on - the variations of hues between each named color are practically limitless! Thinking of colors in this way creates a circle of hues that artists call *"the color wheel"*! Each one of these hues can be made **lighter (tint)** or **darker (shade)** by mixing with white or black, respectively, and any color can be made **less saturated** (more gray or muted) by mixing with another color on the opposite side of the color wheel.

In the digital world of computers color is treated similarly, and we order colors by the *way the screen generates them*; each **pixel** of color on our screen is produced by combining *super tiny* red, green, and blue lights of varying intensities. Unlike mixing paint, where light intensity is subtracted by pigment and mixing all the colors together produces a muddy brown or gray, *mixing lights is additive* - no light at all is obviously black, and mixing all of the colored lights produces white. As such, we can make a list of possible primary color **intensities**:

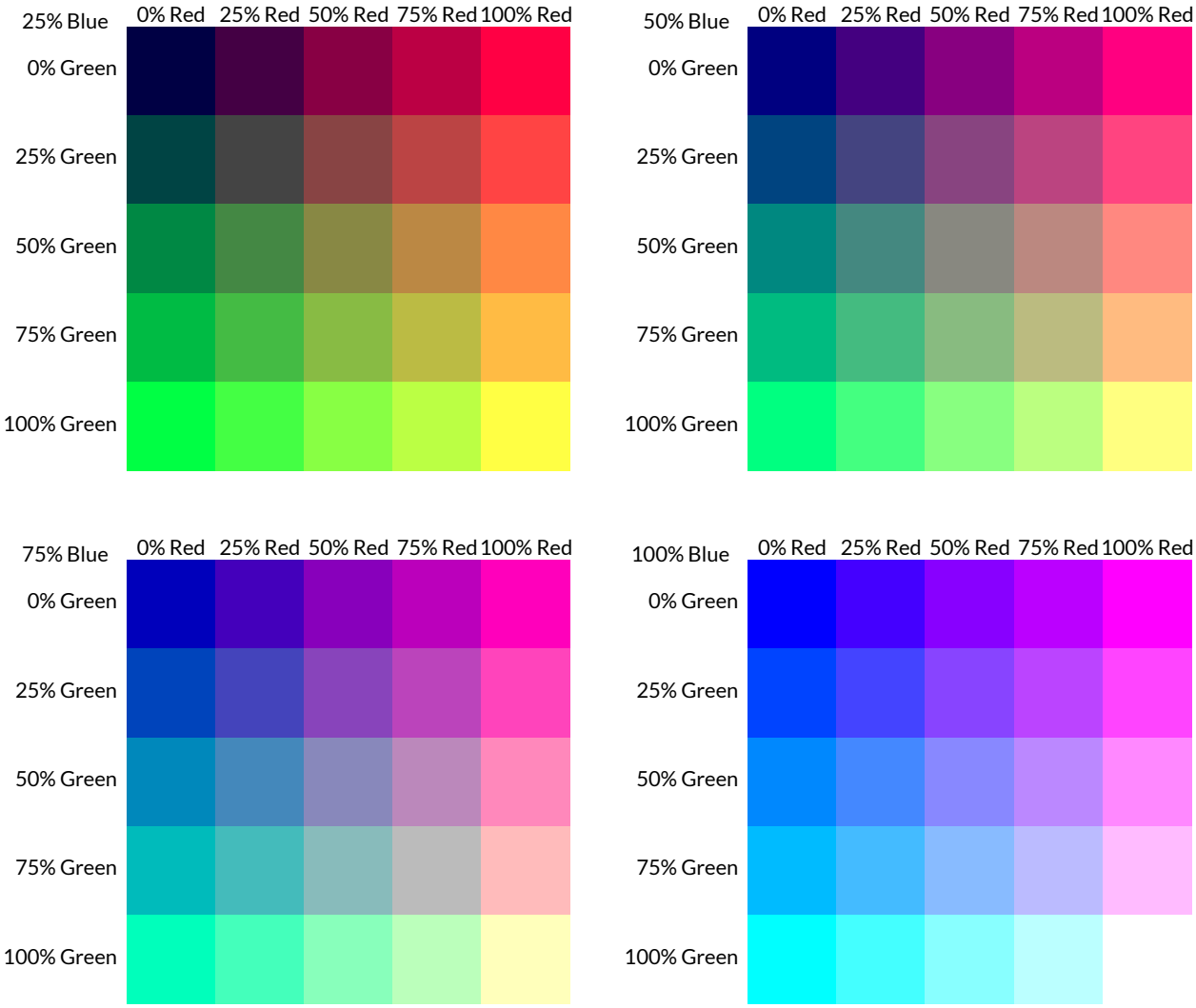


Shown above is a table of different intensities of red light. Our screens can certainly create a lot of shades of red, but we only start to see *the power of pixels* when we add in the other primary colors, green and blue, and show the

colors of light that are produced when they are added together! For example, here's a table showing various mixes of red and green:

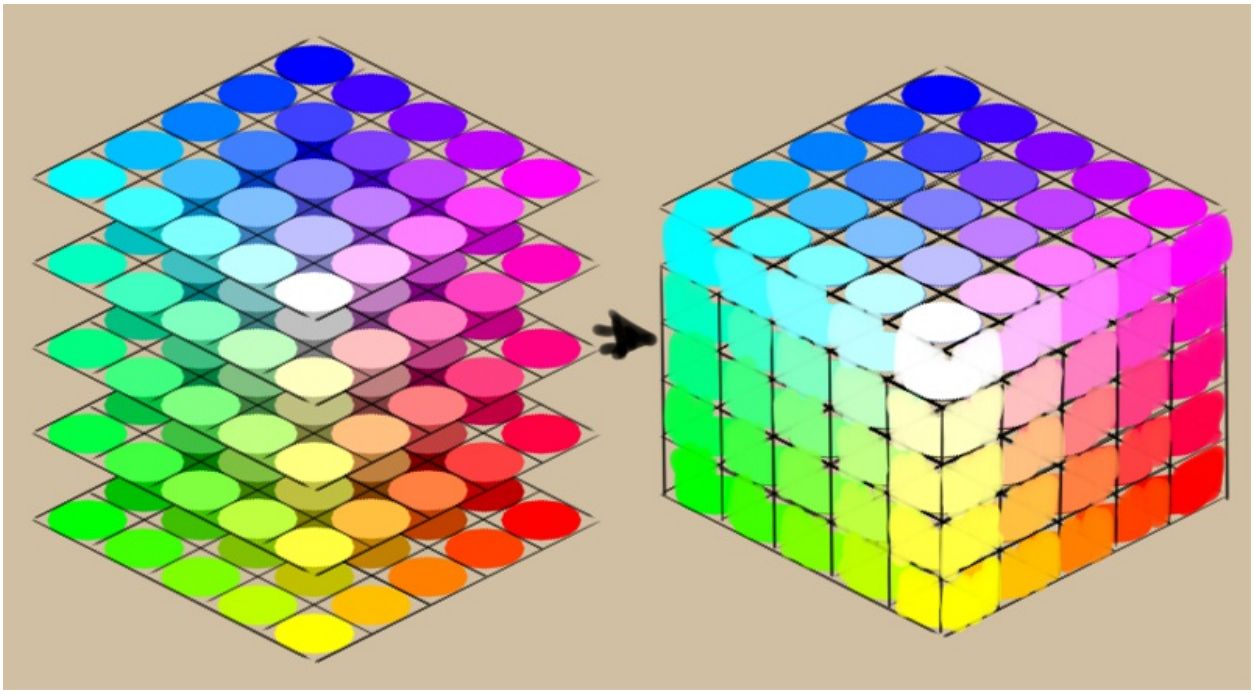


But that's just red and green, what about blue? I guess we can make *even more tables* to show what happens when different amounts of blue are added into the mix:



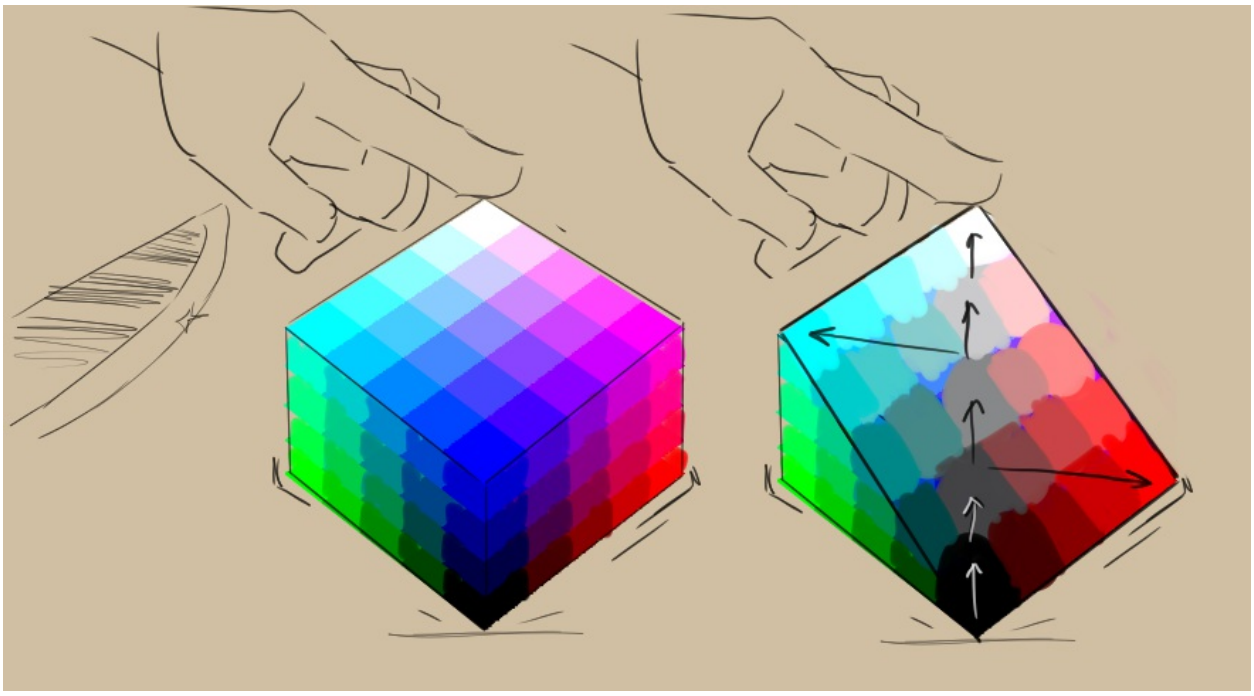
This way of ordering colors is probably familiar to you if you have used some programs for making internet applications, like Flash. In fact, if we had made 6 samples instead of 5 per “channel” (that is, per each primary color), we’d have gotten the 216 websafe colors [https://websafecolors.info/color-chart]!

Showing the colors in a bunch of tables just *feels wrong*, though, doesn’t it? That’s because, while our tables are *2D*, as we are mixing *three* primary colors, color can be thought of as *3D*! It’s a bit odd the first time you think about it this way, but you can actually stack these tables based on the amount of blue and they become a **cube**!



This cube is not filled with water, or sand, or even *concrete*, but colors! Colors are pretty *abstract*, and we typically talk about cubes and other 3D objects that represent abstract ideas as **spaces**, hence we call this cube a **color space**. Because this particular cube uses red, green, and blue as its axes, we say that our cube is in the RGB color model.

There are many more color models. For example, if we were to balance our cube on the black corner, the white corner would be right under our finger at the very top of the cube. And as geometry and maths would have it, if we were to cut the cube in half as we balanced it, the line from the white point at the top to the black point at the bottom would be the **grayscale**.



When you think about a strip of grays running through the middle of the cube, as we move farther away from that grayscale towards the *outer edges* of the cube the colors would begin to become more saturated (colorful and vivid). The circle of colors around that middle axis of gray would then define the hue, with a different color in each direction.

This is the basic idea of the HSV, HSL, HSI, and HSY color models. This particular model is called **HSI** (hue, saturation, and intensity), because it maps each unique color to the intensity of the primary colored lights that mix to create them.

There are other color models, like L*a*b*, where we look at the corresponding gray value of a color first, and then try to describe it, not it terms of hue and saturation, but by how red, green, blue, and yellow it is. Because our brains cannot really comprehend a color that is both green and red, or yellow and blue, this makes them good *polar opposites* in a sliding scale. We call this a **perceptual model**, as it is based on how *we see color instead of how the color is generated*.

Color models describe color spaces, which, in turn, are all sorts of sizes and shapes as well. Krita allows you to do operations in different models and spaces, and we call this functionality **“Color Management”**.

Color Management is necessary for CMYK (subtractive) support, but outside of that, not many drawing or painting programs offer the feature, as *some* developers believe that artists have no need for such functionality. *What a pity!* Especially because Color Management allows for far more *cool tricks* than just basic CMYK support, and the ability to *manipulate colors like a computer can* is perhaps digital painting's most unique quality!

As Krita is giving almost *unprecedented control of color*, this unfortunately means that there are little to no articles out there on how to use color management for artists or painters. And so, we made this category and hope to fill it up with relatively short articles explaining color-related concepts in a light-hearted and visual manner.

We recommend going over the color managed workflow page next - even if you don't plan on using it, it will help make sense out of the many features related to colors and Color Management. Other than that, each article should stand on its own and can be taken in at your own direction and pace!

Topics:

Bit Depth

Bit depth basically refers to the amount of working memory per pixel you reserve for an image.

Like how having a A2 paper in real life can allow for much more detail in the end drawing, it does take up more of your desk than a simple A4 paper.

However, this does not just refer to the size of the image, but also how much precision you need per color.

To illustrate this, I'll briefly talk about something that is not even available in Krita:

Indexed Color

In older programs, the computer would have per image, a palette that contains a number for each color. The palette size is defined in bits, because the computer can only store data in bit-sizes.

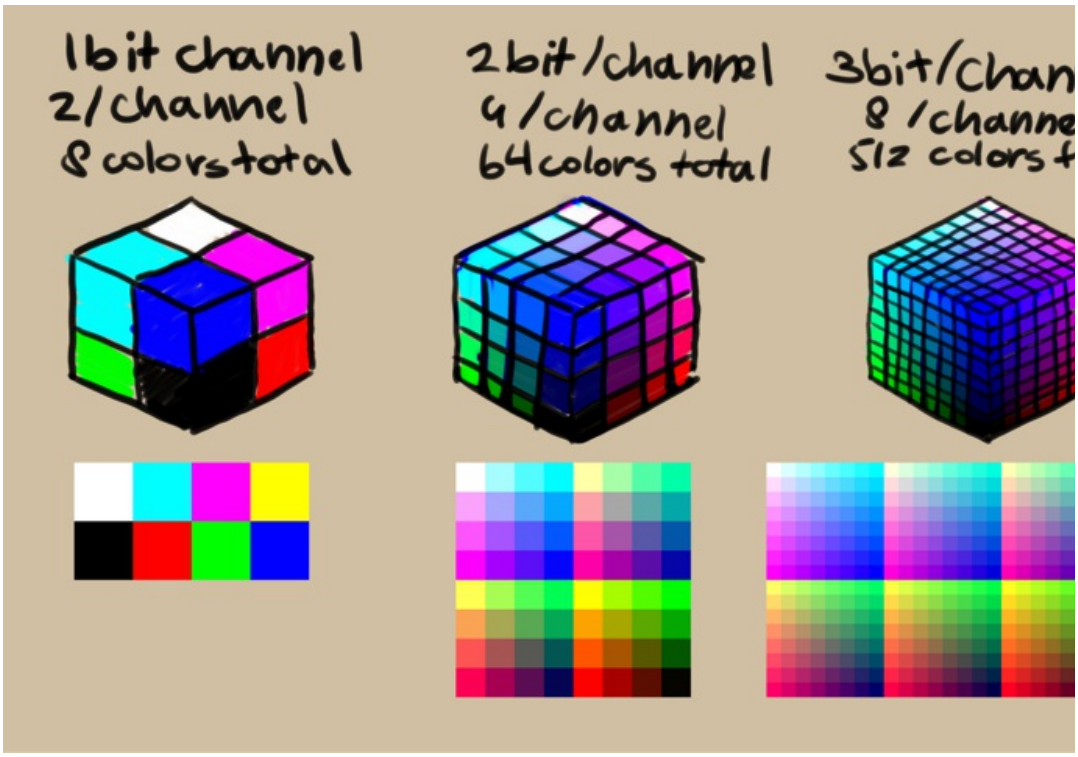
1bit
: Only two colors in total, usually black and white.

4bit (16 colors)
: 16 colors in total, these are famous as many early games were presented in this color palette.

8bit
: 256 colors in total. 8bit images are commonly used in games to save on memory for textures and sprites.

However, this is not available in Krita. Krita instead works with channels, and counts how many colors per channel you need (described in terms of ‘’bits per channel’’). This is called ‘real color’.

Real Color



1, 2, and 3bit per channel don't actually exist in any graphics application ou
imagining them, we can imagine how each bit affects the precision: Usually,
each section in the color cube meaning precision becomes a power of 2 bigg
cube.

4bit per channel (not supported by Krita)

Also known as Hi-Color, or 16bit color total. A bit of an old system, and not used outside of specific displays.

8bit per channel

Also known as "True Color", "Millions of colors" or "24bit/32bit". The standard for many screens, and the lowest bit-depth Krita can handle.

16bit per channel

One step up from 8bit, 16bit per channel allows for colors that can't be displayed by the screen. However, due to this, you are more likely to have smoother gradients. Sometimes known as "Deep Color". This color depth type doesn't have negative values possible, so it is 16bit precision, meaning that you have 65536 values per channel.

16bit float

Similar to 16bit, but with more range and less precision. Where 16bit only allows coordinates like [1, 4, 3], 16bit float has coordinates like [0.15, 0.70, 0.3759], with [1.0,1.0,1.0] representing white. Because of the differences between floating point and integer type variables, and because Scene-referred imaging allows for negative values, you have about 10-11bits of precision per channel in 16 bit floating point compared to 16 bit in 16 bit int (this is 2048 values per channel in the 0-1 range). Required for HDR/Scene referred images, and often known as ‘half floating point’.

32bit float

Similar to 16bit float but with even higher precision. The native color depth of OpenColor IO, and thus faster than 16bit float in HDR images, if not heavier. Because of the nature of floating point type variables, 32bit float is roughly equal to 23-24 bits of precision per channel (16777216 values per channel in the 0-1 range), but with a much wider range (it can go far above 1), necessary for HDR/Scene-referred values. It is also known as ‘single floating point’.

This is important if you have a working color space that is larger than your device space: At the least, if you do not want color banding.

And while you can attempt to create all your images a 32bit float, this will quickly take up your RAM. Therefore, it’s important to consider which bit depth you will use for what kind of image.

Color Managed Workflow

You may have heard that Krita has something called color-management. Or maybe you just wondered what all these ‘color model’ and ‘color profile’ things you can find in the menus mean. Color management is pretty useful for people who work in digital imaging professionally, and hopefully this page will explain why.

Basic Info

If you’ve never worked with color management before, and have no clue what it is, then know that you’ve probably been working in the 8bit RGB color space with the sRGB profile. This means you can choose for *sRGB built-in* or *sRGB-elle-v2-srgbtrc.icc*. With the new color space browser this profile is marked with *(default)* when using 8bit.

We’ll go into what these terms mean in the theory, but if you’re here only for trying to figure out which is the default, you now know it. Maybe, after reading this, you may feel like changing the default, to get new and interesting results from filters, blending modes, or just the color smudge brush.

What is the problem?

To explain the point of color management, you’d first need to learn which problem color management tries to solve.

Let us imagine a kinder garden:

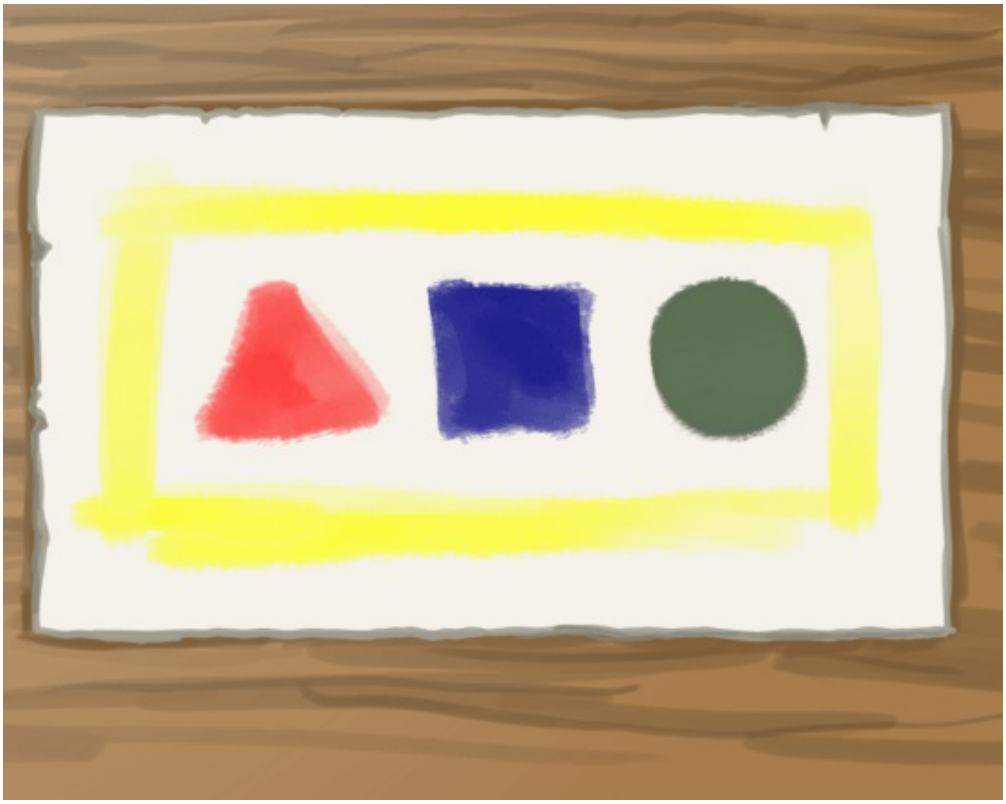
The class of 28 children is subdivided in groups of 7. Each group has its own table.

The teacher gives them a painting assignment: They need to paint a red triangle, a blue square, a green circle and put a yellow border around the

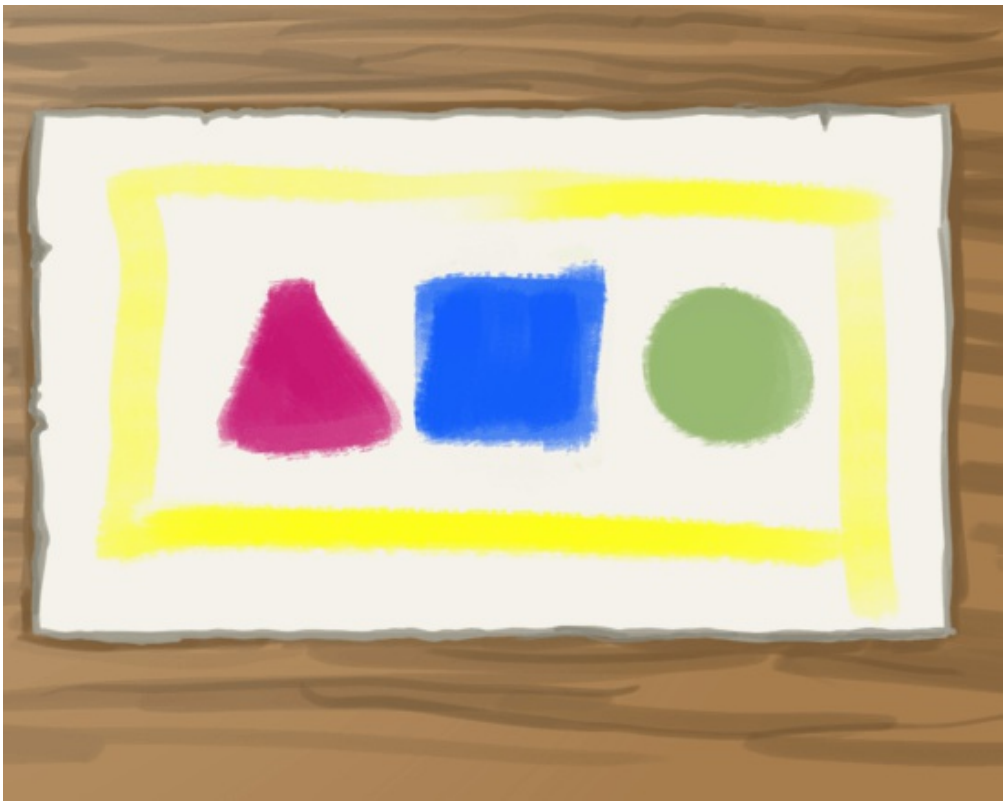
three. The kids are very experienced with painting already, so the teacher can confidently leave the smarter ones to their own devices, and spent more time on those who need help.

The following results come from painting:

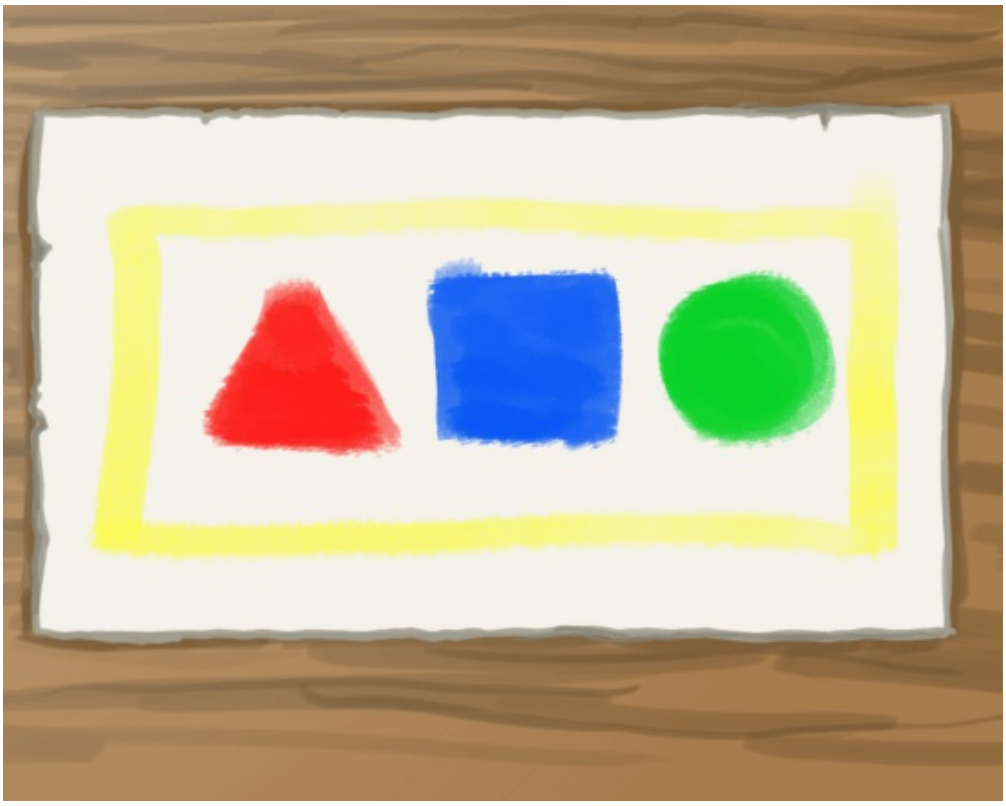
Even though all groups had the same assignment, each group’s result looks different.



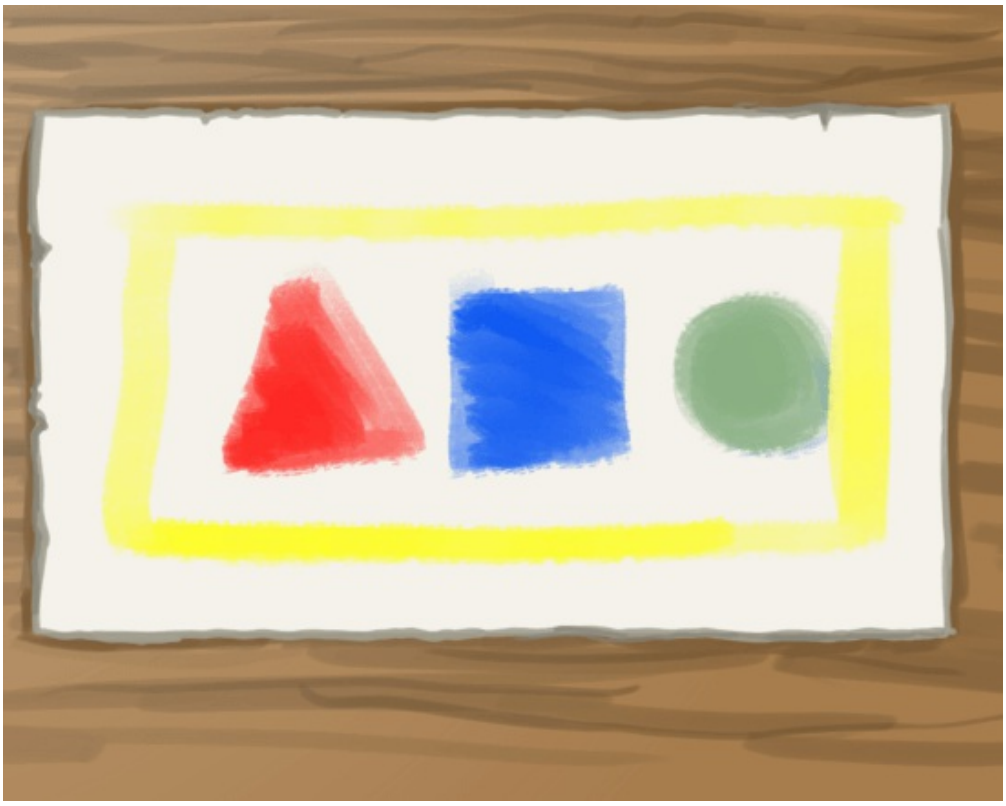
Group 1 had vermillion red, citron yellow and ultramarine blue to their disposal. This means their triangle looks nice and red, but their circle’s green is muddy. This is because ultramarine is too dark of a blue to create nice greens with.



Group 2 had magenta red, citron yellow and cerulean blue. Magenta is a type of red that is closer to pink, opposed to vermillion, which is closer to orange. However, their green looks nice because cerulean is a much lighter blue.



Group 3 had vermillion red, citron yellow, emerald green and cerulean blue. They didn’t mix their green, and thus ended up with a purer color.



Finally, group 4 has vermillion red, citron yellow and cerulean blue. Their colors probably look like what you imagined.

Now, these are kindergarteners, so this isn't the largest problem in the world. However, imagine that something like this happened at a printing company? Imagine four printers printing the same magazine with wildly different results? That would be disastrous!

For this purpose, we invented color management.

What is color management?

Color management is, dryly put, a set of systems that tries to have the same color translate properly between color devices.

It usually works by attempting to convert a color to the reference color space XYZ. XYZ is a coordinate system that has a spot for all colors that the average human eye can see.

From XYZ it can then be translated back into another device space, such as RGB (for screens), or CMYK (for printers).

Krita has two systems dedicated to color management. On the one hand, we have **lcms2**, which deal with ICC profiles, and on the other, we have **OCIO**, which deal with LUT color management.

To give a crude estimate, ICC profiles deal with keeping colors consistent over many interpretations of devices (screens, printers) by using a reference space, and OCIO deals with manipulating the interpretation of said colors.

Within both we can identify the following color spaces:

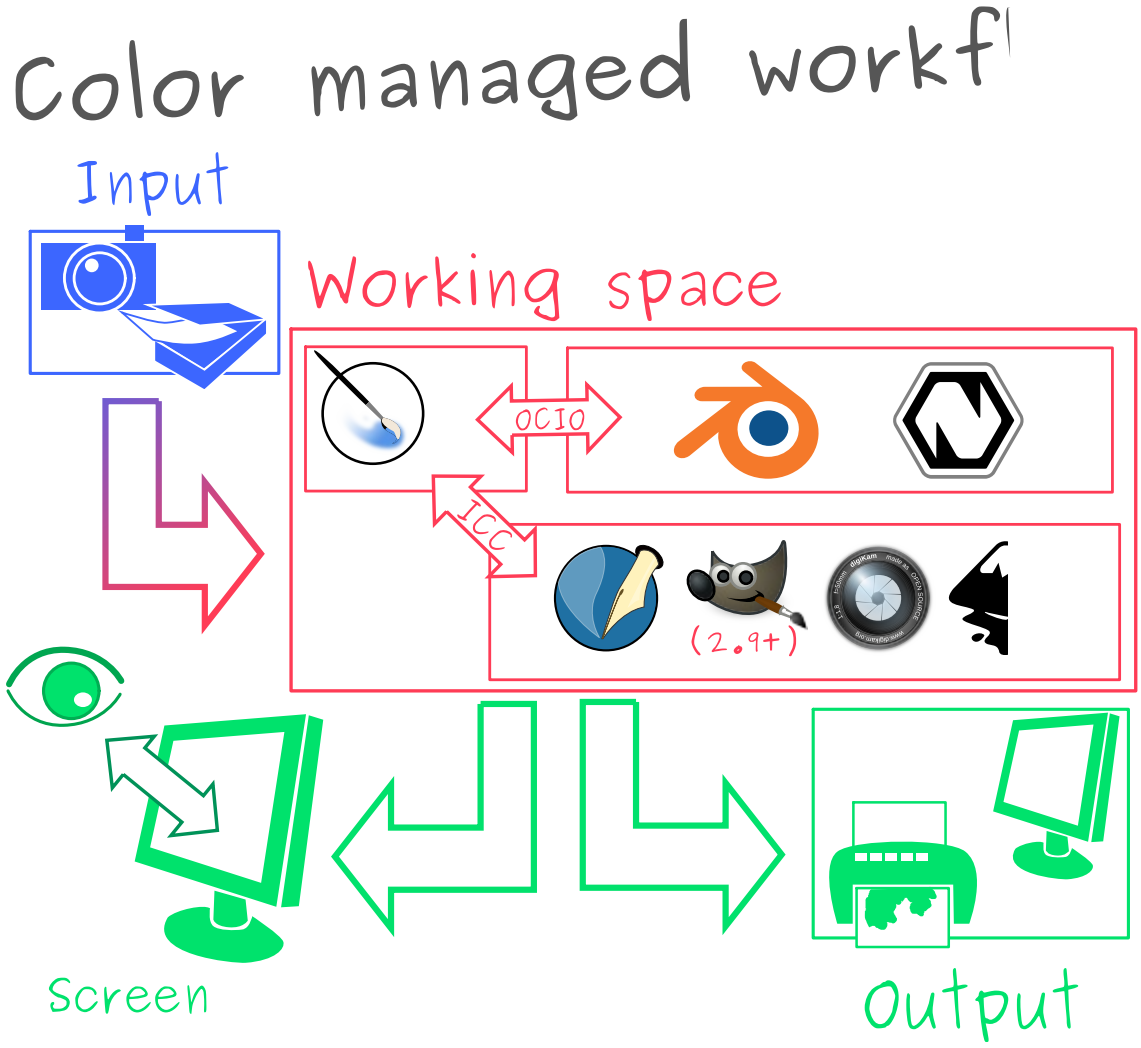
Device spaces
: Device spaces are those describing your monitor, and have to be made using a little device that is called “colorimeter”. This device, in combination with the right software, measures the strongest red, green and blue your screen can produce, as well as the white, black and gray it produces. Using these and several other measurements it creates an ICC profile unique to your screen. You set these in Krita’s color management tab. By default we assume sRGB for screens, but it’s very likely that your screen isn’t exactly fitting sRGB, especially if you have a high quality screen, where it may be a bigger space instead. Device spaces are also why you should first consult with your printer what profile they expect. Many printing houses have their own device profiles for their printers, or may prefer doing color conversion themselves. You can read more about colorimeter usage here.

Working spaces
: These are delivered alongside Krita for ICC, and downloadable from the OCIO website for OCIO. Working spaces are particularly nice to do color calculations in, which programs like Krita do often. It’s therefore recommended to have a working space profile for your image.

Aesthetic or Look spaces
: These are special spaces that have been deformed to give a certain look to an image. Krita doesn’t deliver Look profiles for ICC, nor does it yet support Look spaces for OCIO.

Color managed workflow

Knowing this about these spaces of course doesn't give you an idea of how to use them, but it does make it easier to explain how to use them. So let us look at a typical color management workflow:



A typical example of a color managed workflow. We have input from scanner we convert to a working space that can be used between different editing softv to an output space for viewing on screen or printing.

In a traditional color managed workflow, we usually think in terms of real

world colors being converted to computer colors and the other way around. So, for example photos from a camera or scanned in images. If you have a device space of such a device, we first assign said device space to the image, and then convert it to a working space.

We then do all our editing in the working space, and use the working space to communicate between editing programs. In Krita's case, due to it having two color management systems, we use ICC profiles between programs like Gimp 2.9+, Inkscape, digiKam and Scribus, and OCIO configuration between Blender and Natron.

You also store your working files in the working space, just like how you have the layers unmerged in the working file, or have it at a very high resolution.

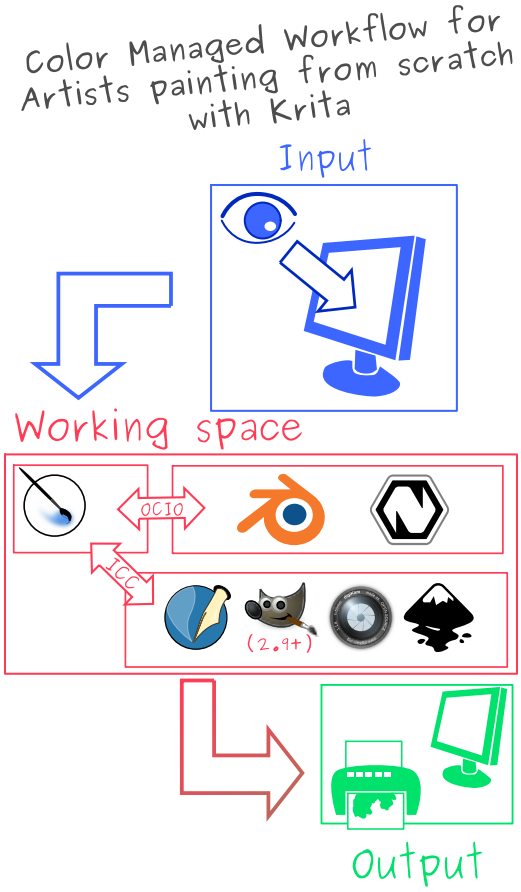
Sometimes, we apply aesthetic or 'look' spaces to an image as part of the editing process. This is rather advanced, and probably not something to worry about in Krita's case.

Then, when we're done editing, we try to convert to an output space, which is another device space. This can be CMYK for printers or a special screen RGB profile. When you are dealing with professional printing houses, it is best to ask them about this step. They have a lot of experience with doing the best conversion, and may prefer to do the conversion from your working space to the device space of their printers.

Another form of output is the way your screen displays the color. Unlike regular output, this one is done all the time during editing: After all, you need to be able to see what you are doing, but your screen is still a device with a device space, so it does distort how the image looks. In this manner, you can see your screen as a set of binoculars you have to look through to see your image at all.

Therefore, without a profiled monitor, you actually don't know what the actual colors you are working with are like, because the computer doesn't know the relevant properties of your screen. So if you profiled your monitor, give Krita the profile in the settings, and select the sRGB space to draw in, you are for the first time seeing the actual colors of the sRGB space.

So what does this mean?



When we paint from scratch, we can see our screen profile as the input space, because we use it to determine what colors to pick. This somewhat simplifies the workflow, but makes the screen profile and viewing conditions more important.

Now, photographers and people who do a tricky discipline of VFX called ‘color grading’ will go completely mad over trying to get the colors they put in to come out 100% correctly, and will even count in factors like the time of day and the color they painted their walls. For example, if the wall behind your computer is pure red, your eyes will adjust to be less sensitive to red,

which means that the colors they pick in the program could come out redder. We call these the *viewing conditions*.

Thankfully, artists have to worry a slight bit less about this. As illustrations are fully handmade, we are able to identify the important bits and make appropriate contrasts between colors. This means that even if our images turn out to be slightly redder than intended, it is less likely the whole image is ruined. If we look back at the kindergarten example above, we still understand what the image was supposed to look like, despite there being different colors on each image. Furthermore, because the colors in illustrations are deliberately picked, we can correct them more easily on a later date. Yet, at the same time, it is of course a big drag to do this, and we might have had much more flexibility had we taken viewing conditions under consideration.

That said, for artists it is also very useful to understand the working spaces. Different working spaces give different results with filters and mixing, and only some working spaces can be used for advanced technology like HDR.

Similarly, Krita, as a program intended to make images from scratch, doesn't really worry about assigning workspaces after having made the image. But because you are using the screen as a binocular to look at your image, and to pick colors, you can see your screen's device space as an input space to the image. Hence why profiling your monitor and giving the profile to Krita in the settings can help with preparing your work for print and future ventures in the long run.

Overall, it is kinda useful to keep things like viewing conditions in the back of your mind. Many professional artists use a mid-gray color as their default canvas background because they find they create much more dynamic images due to having improved their viewing conditions. It is also why a lot of graphics programs, including Krita, come with a dark theme nowadays. (Though, of course this might also be because dark themes can be considered cool, who knows.)

ICC profiles

An ICC profile is a set of coordinates describing the extremities of the device space within XYZ, and it is the color management data you use to communicate your working space to printers and applications that are designed for the print industry, such as Gimp, Scribus, Photoshop, Illustrator, Inkscape, digiKam, RawTheraphee, etc. You have two types of ICC profiles:

Matrix Shaper profiles

These are delivered alongside Krita. Matrix shaper profiles are made by setting parameters and interpolating between these to get the exact size of the color space. Due to this, Krita's color space browser can give you a lot of information on these profiles. Such profiles are also preferable as working space.

Matrix shaper profiles have a few parameters that describe the color space which are then interpolated between, this requires a lot of maths.

cLUT profiles

These are fairly rare, and primarily used to describe printer profiles, such

as CMYK. cLUT, or Color Look-up Table profiles store far more data than Matrix shaper profiles, so they can hold data of little particularities caused by, for example, unexpected results from mixing pigments. This is a far more organic approach to describing a color space, hence why a lot of programs that don't care for color management much don't support these.

cLUT profiles work by holding tables of each color in a color space and their respective coordinates in a reference space. For CMYK this is typically L*A*B* and for the rest XYZ. These tables are tricky to make, which means these profiles are a lot rarer.

The interesting thing about ICC profiles is that your working space can be larger than your device space. This is generally not bad. However, when converting, you do end up with the question of how to translate the working space values.

Perceptual

This just squishes the values of the working space into the space it's converted to. It's a nice method to see all possible values in this, but not so good if you want accurate color reproduction. Use this if you want to

see all colors in an image, or want to express all possible contrasts. Doesn’t work with Matrix Shaper profiles, defaults to relative colorimetric.

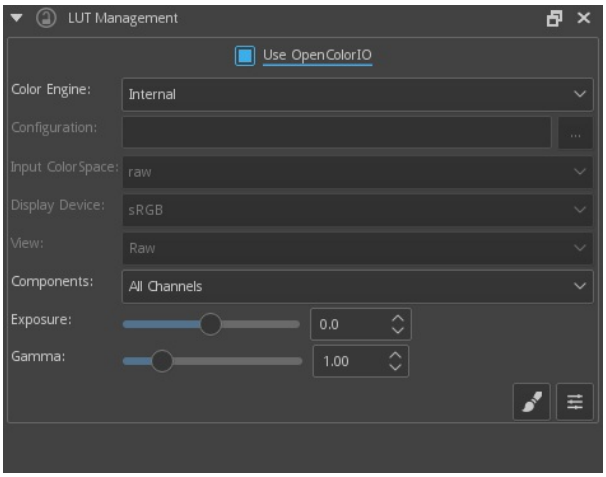
Absolute Colorimetric
: The opposite to Perceptual, Absolute colorimetric will attempt to retain all the correct colors at whatever cost, which may result in awful looking colors. Recommended only for reproduction work. Doesn’t work with Matrix Shaper profiles in Krita due to ICC v4 workflow standards.

Relative Colorimetric
: An in between solution between perceptual and absolute, relative will try to fit whatever colors it can match between color spaces. It does this by aligning the white and black points. It cuts off the rest to their respective borders. This is what all matrix shaper profiles default to during conversion, because the ICC v4 workflow specifies to only use Relative Colorimetric for matrix shaper profiles.

Saturation
: Does anything to retain colorfulness, even hue will be sacrificed. Used in infographics. Doesn’t work with Matrix Shaper profiles, defaults to relative colorimetric.

ICC profile version is the last thing to keep in mind when dealing with ICC profiles. Krita delivers both Version 2 and Version 4 profiles, with the later giving better results in doing color maths, but the former being more widely supported (as seen below in ‘Interaction with other applications’). This is also why Krita defaults to V2, and we recommend using V2 when you aren’t certain if the other programs you are using support V4.

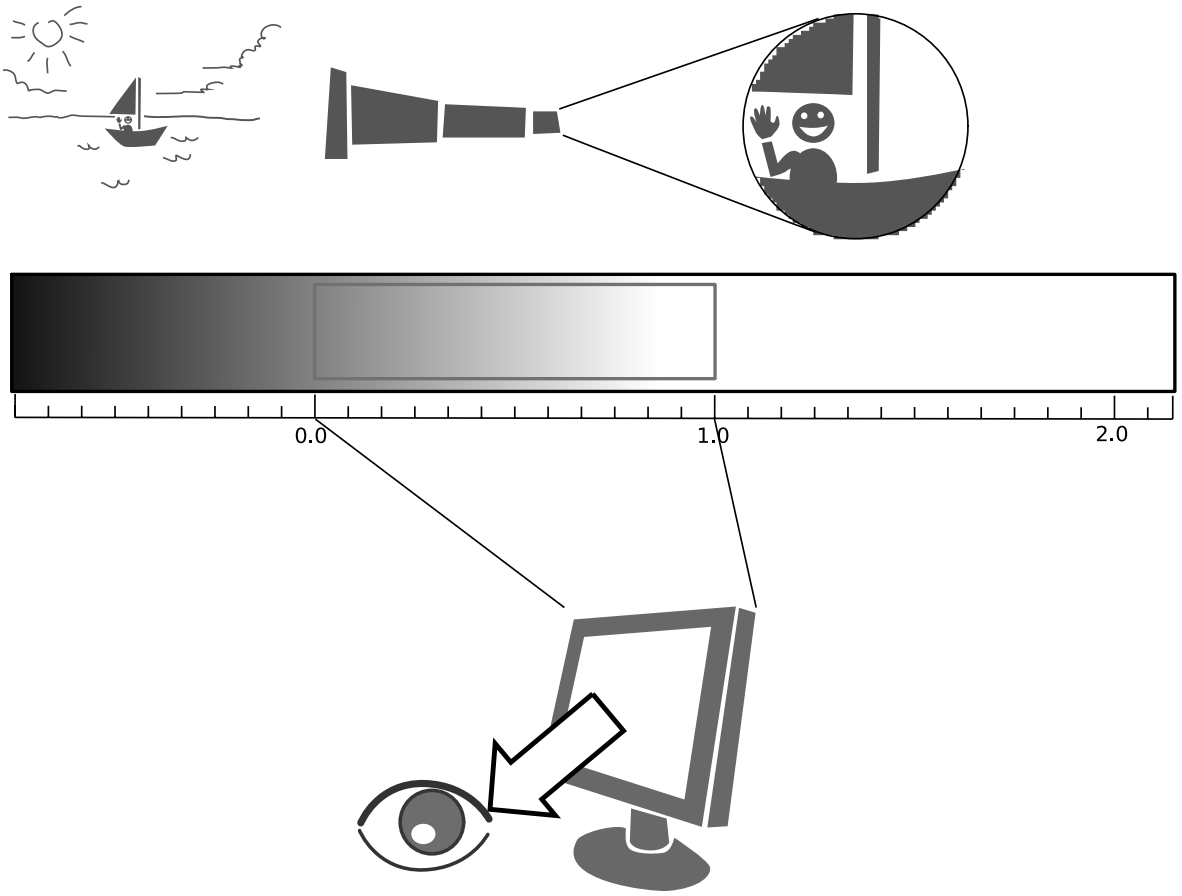
LUT docker and HDR imaging



The LUT Management is the second important bit of color management in Krita that is shared between Krita and programs like Blender, Natron and Nuke, and only uses Look Up Tables that are configured via a config file.

You can set the workingspace of the image under input color space, and the display to sRGB or your own LUT if you have added it to the configuration. View in this case is for proofing transforms to a certain display device.

Component, exposure, gamma, whitepoint and blackpoint are knobs which allows you to modify the display filter.



As explained before, we can see our monitor as a telescope or binocular into the world of our picture. Which means it distorts our view of the image a little. But we can modify this binocular, or display filter to see our image in a different way. For example, to allow us to see the white in an image that are whiter than the white of our screen. To explain what that means, we need to think about what white is.

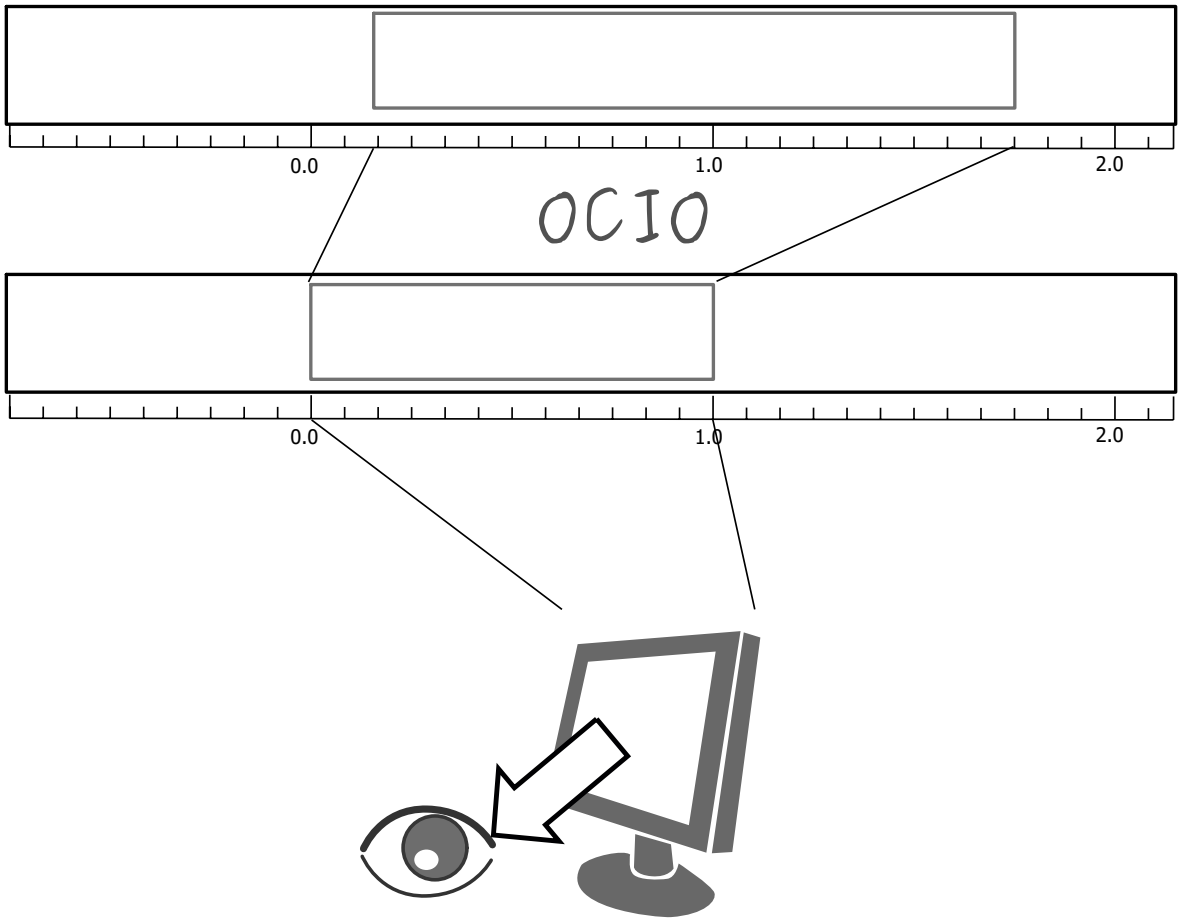
For example, white, on our monitor is full red, full green and full blue. But it's certainly different from white on our paper, or the color of milk, white from the sun, or even the white of our cell-phone displays.

Black similarly, is brighter on a LCD display than a LED one, and incomparable with the black of a carefully sealed room.

This means that there's potentially blacker blacks than screen black, and

white whites than screen white. However, for simplicity's sake we still assign the black-point and the white-point to certain values. From there, we can determine whether a white is whiter than the white point, or a black blacker than the black-point.

The LUT docker allows us to control this display-filter and modify the distortion. This is useful when we start modifying images that are made with scene referred values, such as HDR photos, or images coming out of a render engine.



So, for example, we can choose to scale whiter-than-screen-white to our screen-white so we can see the contrasts there.

The point of this is that you can take advantage of more lightness detail in an image. While you can't see the difference between screen white and whiter-

than-screen-white (because your screen can’t show the difference), graphics programs can certainly use it.

A common example is matching the lighting between a 3d model and a real world scene. Others are advanced photo retouching, with much more contrast information available to the user. In painting itself, this allows you to create an image where you can be flippant with the contrast, and allow yourself to go as bright as you’d like.

LUT docker manipulations are per view, so you can create a new view and set it to luminosity. This way you can see the image in both color as well as grayscale and keep a good eye on your values.

Another example is to carefully watch the gradients in a certain section.

Like ICC, the LUT Docker allows you to create a profile of sorts for your device. In this case it’s the ‘LUT’, which stands for ‘Look Up Table’, and which can be added to OCIO by modifying the configuration file. When OCIO is turned on, the configuration in *Settings* ▸ *Configure Krita…* ▸ *Color Management* is turned off, unless you are using the *Internal* color engine.

In summary

Krita has two modes of color management:

- ICC works in terms of spaces relative to the CIEXYZ space, and requires an ICC profile.
- OCIO works in terms of interpretation, and makes use of LUTs.
- both can be made with a colorimeter.
- If you want to have a properly color managed workflow, you have one made customary for the input device (your screen) and the output devices (your printer, or target screen). For web the output is always sRGB.
- Set up your screen profiles under *Settings* ▸ *Configure Krita…* ▸ *Color management*.
- Do NOT use screen profiles or other device profiles to draw in. Use a working space profile such as any of the ‘elle’ profiles for this, as the

color maths will be much more predictable and pleasant. Krita will convert between your screen and working space on the fly, allowing you to pick the correct colors. This turns your screen into binoculars to view the image.
- Use the appropriate color management for the appropriate workflow. If you are working with Blender, you will be better off using OCIO, than ICC. If you are working with Scribus or Photoshop, use ICC.

Krita does a lot of color maths, often concerning the blending of colors. This color maths works best in linear color space, and linear color space requires a bit depth of at the least 16bit to work correctly. The disadvantage is that linear space can be confusing to work in.

If you like painting, have a decent amount of RAM, and are looking to start your baby-steps in taking advantage of Krita's color management, try upgrading from having all your images in sRGB built-in to sRGB-v2-elle-g10.icc or rec2020-v2-elle-g10.icc at 16bit float. This'll give you better color blending while opening up the possibility for you to start working in HDR!

Note

Some graphics cards, such as those of the NVidia-brand actually have the best performance under 16bit float, because NVidia cards convert to floating point internally. When it does not need to do that, it speeds up!

Note

No amount of color management in the world can make the image on your screen and the image out of the printer have 100% the same color.

Exporting

When you have finished your image and are ready to export it, you can modify the color space to optimize it:

If you are preparing an image for the web:

- If you use 16bit color depth or higher, convert the image to 8bit color depth. This will make the image much smaller.
    - Krita doesn't have built-in dithering currently, which means that 16 to 18bit conversions can come out a bit banded. But you can simulate it by adding a fill layer with a pattern, set this fill layer to overlay, and to 5% opacity. Then flatten the whole image and convert it to 8bit. The pattern will function as dithering giving a smoother look to gradients.
- If it's a gray-scale image, convert it to gray-scale.
- If it's a color image, keep it in the working space profile: Many web browsers these days support color profiles embedded into images. Firefox, for example, will try to convert your image to fit the color profile of the other monitor (if they have one). That way, the image will look almost exactly the same on your screen and on other profiled monitors.

Note

In some versions of Firefox, the colors actually look strange: This is a bug in Firefox, which is because its color management system is incomplete [https://ninedegreesbelow.com/galleries/viewing-photographs-on-the-web.html], save your PNG, JPG or TIFF without an embedded profile to work around this.

If you are preparing for print:

- You hopefully made the picture in a working space profile instead of the actual custom profile of your screen, if not, convert it to something like Adobe RGB, sRGB or Rec. 2020.
- Check with the printer what kind of image they expect. Maybe they expect sRGB color space, or perhaps they have their own profile.

Interaction with other applications

Blender

If you wish to use Krita's OCIO functionality, and in particular in combination with Blender's color management, you can try to have it use Blender's OCIO config.

Blender's OCIO config is under `<Blender-folder>/version number/datafiles/colormanagement`. Set the LUT docker to use the OCIO engine, and select the config from the above path. This will give you Blender's input and screen spaces, but not the looks, as those aren't supported in Krita yet.

Windows Photo Viewer

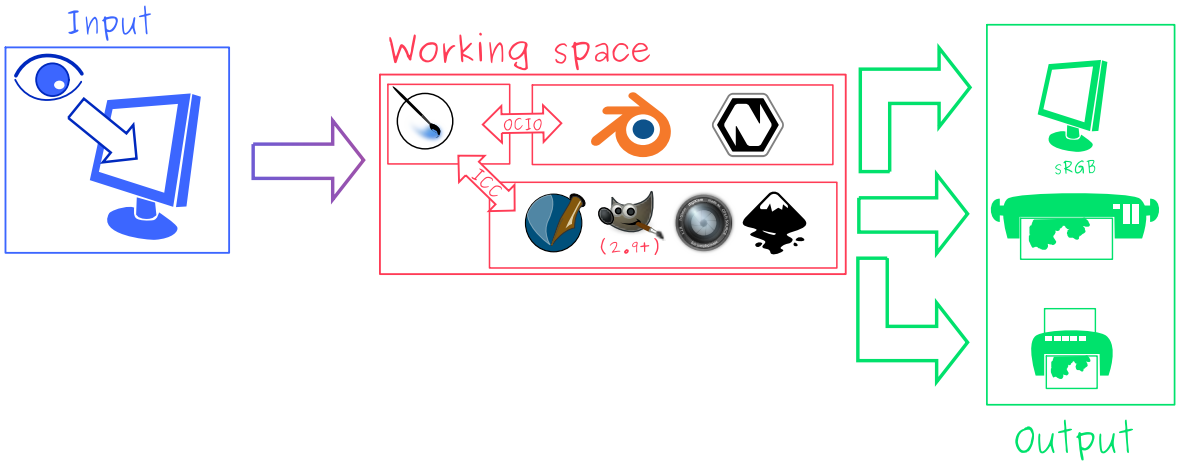
You might encounter some issues when using different applications together. One important thing to note is that the standard Windows Photo Viewer application does not handle modern ICC profiles. Krita uses version 4 profiles; Photo Viewer can only handle version 2 profiles. If you export to JPEG with an embedded profile, Photo Viewer will display your image much too dark.

Example workflows

Here are some example workflows to get a feeling of how your color management workflow may look like.

As mentioned before, input for your screen is set via *Settings* ▸ *Configure Krita…* ▸ *Color management*, or via the LUT docker's 'screen space'. Working space is set via new file per document, or in the LUT docker via 'input space'.

Webcomic



Input
Your screen profile. (You pick colors via your screen)

Workingspace
sRGB (the default screen profile) or any larger profile if you can spare the bit depth and like working in them.

Output
sRGB, ICC version 2, sRGB TRC for the internet, and a specialized CMYK profile from the printing house for the printed images.

Use the sRGB-elle-V2-srgbtrc.icc for going between Inkscape, Photoshop, Painttool Sai, Illustrator, Gimp, Manga Studio, Paintstorm Studio, MyPaint, Artrage, Scribus, etc. and Krita.

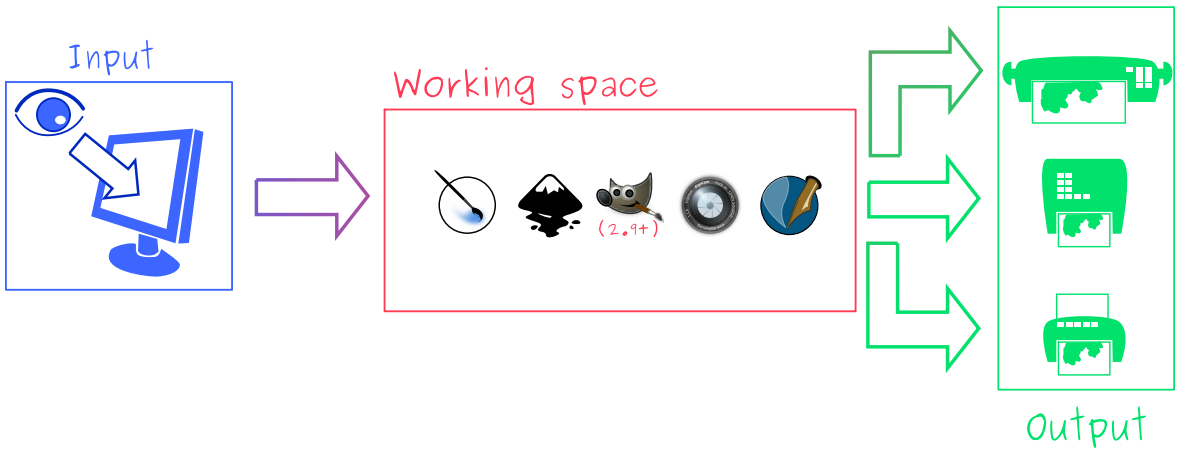
If you are using a larger space via ICC, you will only be able to interchange it between Krita, Photoshop, Illustrator, Gimp 2.9, Manga Studio and Scribus. All others assume sRGB for your space, no matter what, because they don't have color management.

If you are going between Krita and Blender, Nuke or Natron, use OCIO and set the input space to 'sRGB', but make sure to select the sRGB profile for ICC when creating a new file.

For the final for the web, convert the image to sRGB 8bit, 'srgbtrc', do not

embed the ICC profile. Then, if using PNG, put it through something like ‘pngcrush’ or other PNG optimizers. sRGB in this case is chosen because you can assume the vast majority of your audience hasn’t profiled their screen, nor do they have screens that are advanced enough for the wide gamut stuff. So hence why we convert to the screen default for the internet, sRGB.

Print



Input
: Your screen profile. (You pick colors via your screen)

Workingspace
: sRGB or Rec. 2020 if you can afford the bit-depth being 16bit.

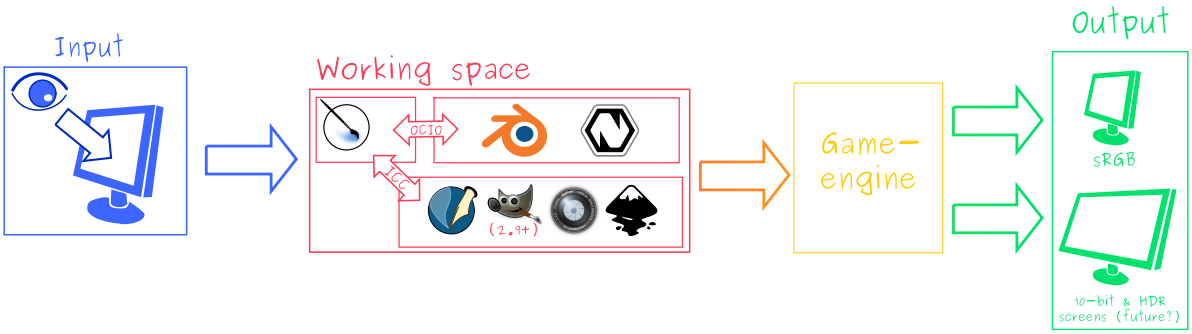
Output
: Specialized CMYK profile from the printing house for the printed images.

The CMYK profiles are different per printer, and even per paper or ink-type so don’t be presumptuous and ask ahead for them, instead of doing something like trying to paint in any random CMYK profile. As mentioned in the viewing conditions section, you want to keep your options open.

You can set the advanced color selector to transform to a given profile via *Settings* ▸ *Configure Krita…* ▸ *Color Selector Settings*. There, tick *Color*

Selector Uses Different Color Space than Image and select the CMYK profile you are aiming for. This will limit your colors a little bit, but keep all the nice filter and blending options from RGB.

Games



Input
: Your screen profile. (You pick colors via your screen)

Workingspace
: sRGB or grayscale linear for roughness and specular maps.

Output
: This one is tricky, but in the end it'll be sRGB for the regular player.

So this one is tricky. You can use OCIO and ICC between programs, but recommended is to have your images to the engine in sRGB or grayscale. Many physically based renderers these days allow you to set whether an image should be read as a linear or 'srgbtrc' image, and this is even vital to have the images being considered properly in the physically based calculations of the game renderer.

While game engines need to have optimized content, and it's recommended to stay within 8bit, future screens may have higher bit depths, and when renderers will start supporting those, it may be beneficial to develop a workflow where the working-space files are rather unnecessarily big and you run some scripts to optimize them for your current render needs, making updating the game in the future for fancier screens less of a drag.

Normal maps and heightmaps are officially supposed to be defined with a ‘non-color data’ working space, but you’ll find that most engines will not care much for this. Instead, tell the game engine not to do any conversion on the file when importing.

Specular, glossiness, metalness and roughness maps are all based on linear calculations, and when you find that certain material has a metalness of 0.3, this is 30% gray in a linear space. Therefore, make sure to tell the game engine renderer that this is a linear space image (or at the very least, should NOT be converted).

See also

- Visualizing the XYZ color space [https://www.youtube.com/watch?v=x0-qoXOCOow].
- Basics of gamma correction [https://www.cambridgeincolour.com/tutorials/gamma-correction.htm].
- Panda3D example of how an image that has gamma encoded without the 3D renderer being notified of it having gamma-encoding can result in too dark images [https://www.panda3d.org/blog/the-new-opengl-features-in-panda3d-1-9/].
- 2D examples of the effect of gamma-encoding on color maths [https://ninedegreesbelow.com/photography/linear-gamma-blur-normal-blend.html].
- Basic overview of color management from ArgyllCMS manual [https://www.argyllcms.com/doc/ColorManagement.html].

Mixing Colors

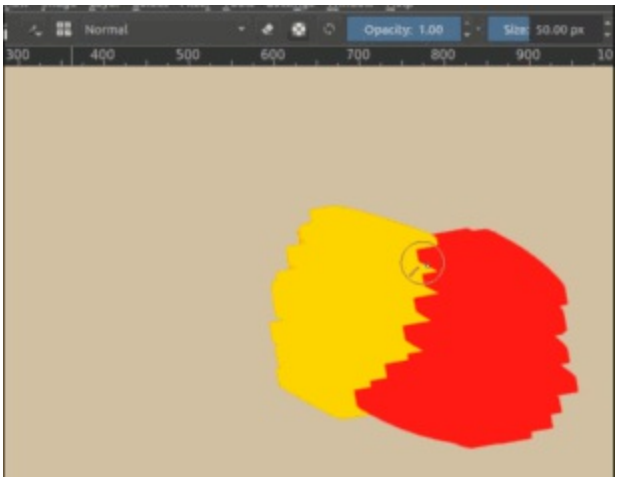
Much like physical media, there are many ways to mix colors together in Krita. Traditional painters and illustrators often use techniques like **glazing**, **scumbling**, and **hatching** to mix colors directly on their canvas, on top of mixing colors together on a **palette** or even within the hairs of their **brush**. With a little bit of practice and know-how, and thanks to the variety of powerful tools in Krita, we can mimic all of these mixing techniques in digital painting.

In both traditional and digital painting, mixing techniques can be divided into *two major categories*: let's call them **"on-canvas"** and **"off-canvas"**.

On-Canvas Mixing

On-canvas mixing techniques are ones where multiple colors are combined directly on the canvas as the artist paints. This takes a few forms, including **layering semi-transparent color** on top of another color, using **texture** to change how a color is perceived, or even in the interaction between two areas of wet paint in traditional media. Bottom line: on-canvas mixing happens right on the canvas and is a great tool for artistic exploration and "happy accidents".

Glazing



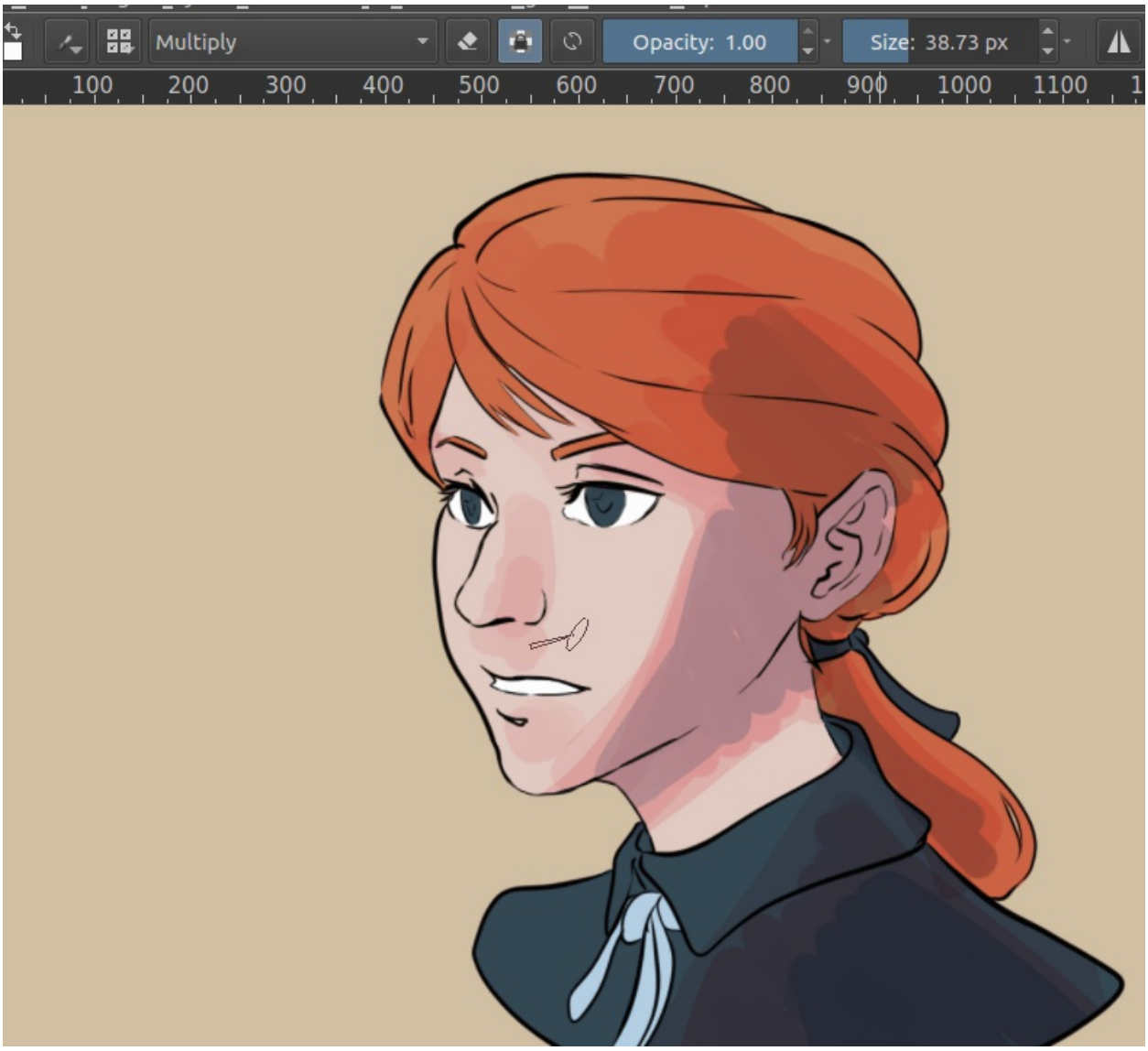
In traditional painting, **glazing** is overlaying many different semi-transparent layers to create on-canvas color mixtures. Likewise, in digital painting we can also use glazing to mix colors directly on our canvas. This is one of the most fundamental and commonly used mixing techniques in digital painting.

We first lay down a semi-transparent layer on top of another color that we intend to mix with. Then, we pick the **resultant color** with the `Ctrl +`  shortcut (this can be configured in the canvas input settings), and paint with that. Depending on our brush's **opacity setting**, each time we glaze one color over another we will get a color that is somewhere between those two colors, often leading to a nice mixture.

We can mix even more easily with glazing when we set our brush's **flow** to a lower setting. Subtly different than opacity, flow is transparency per dab instead of stroke, and so it gives us softer strokes without giving up control.

Furthermore, we can combine glazing with various **blending modes** to achieve different, interesting effects. For example, glazing with the **multiply** blending mode to create nice shadows:

Staring with line art and base colors.

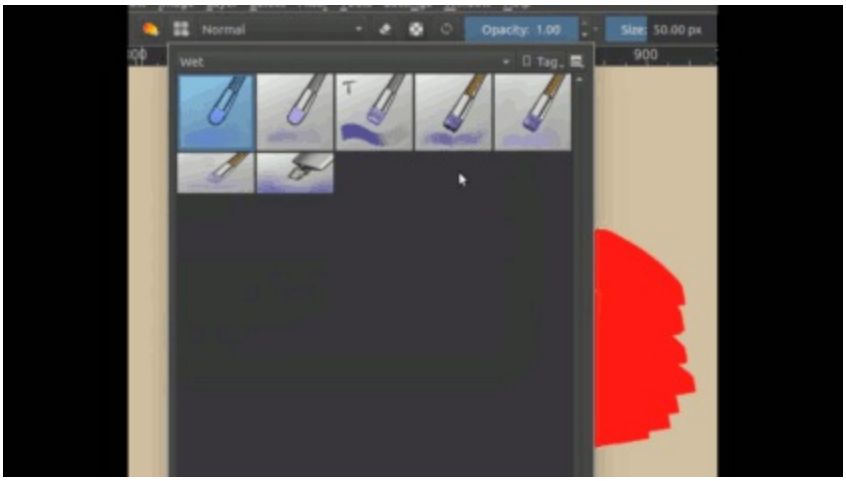


Using a semi-transparent brush that’s set to multiply, we can add colored layers to suggest shadows and mid-tones. The multiply blending mode will darken and interact with each base color differently.

Then, using a brush with low flow (~0.30), we can pick the resulting colors and lay down more layers. Not only does this help you define the different planes and forms that are so crucial for creating a sense of depth and three-dimensionality, it also gives quite a nice, painterly effect!

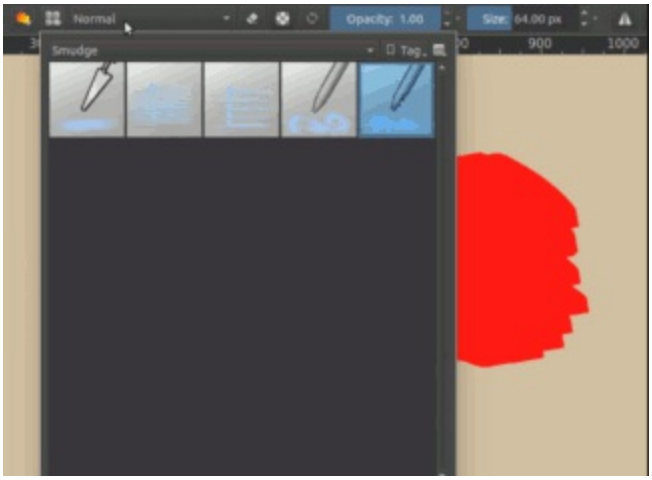
Continue with a lower opacity and flow to create even smoother gradients. Make your **edges** as sharp or smooth as your subject matter and art style demands!

Smudging



Smudge mixing is done with the Color Smudge Brush Engine, a special brush engine that allows you to mix your current brush color with the color(s) under the brush. It's a very powerful type of brush that gives a lovely painterly effect. *Performance wise, it's a bit more demanding and slower than the regular pixel brush.*

If you *remove all paint from a smudge brush*, you get a simple-yet-powerful smudge effect:

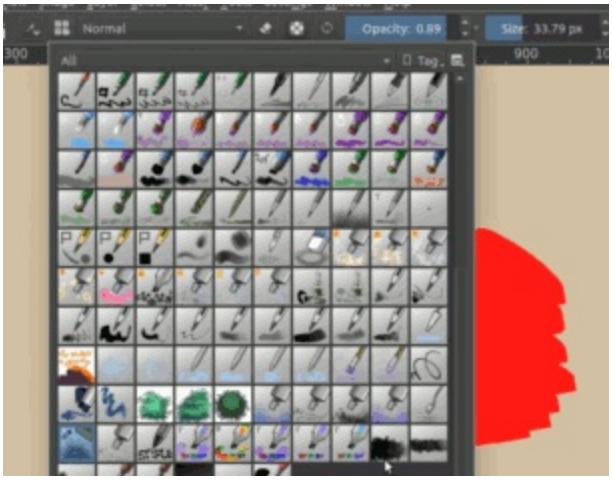


Different smudge brushes have different effects, so be sure to try them all out!

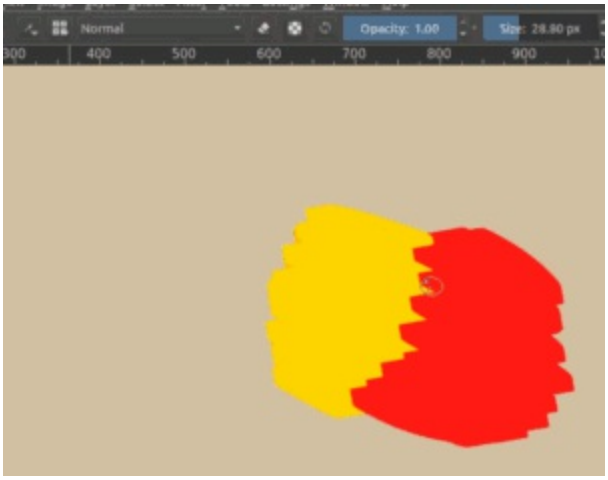
Scumbling

Scumbling is similar to glazing, except instead of having a semi-opaque

layer, we use layers of **textured** or **patterned** paint.



Like most painting programs, Krita allows you to pick a Brush Tips, which can be used to create a textured effect like that of scumbling.



Krita's brush engines also allow you to use Texture. This allows you to create interesting and stylized screentone-like effects.

With glazing can get you pretty far when it comes to *defining planes and forms*, scumbling is the best method to *create texture and to break up big pasty flats* in your painting.

Off-Canvas Mixing

Off-canvas mixing has basically always been a core tool for artists

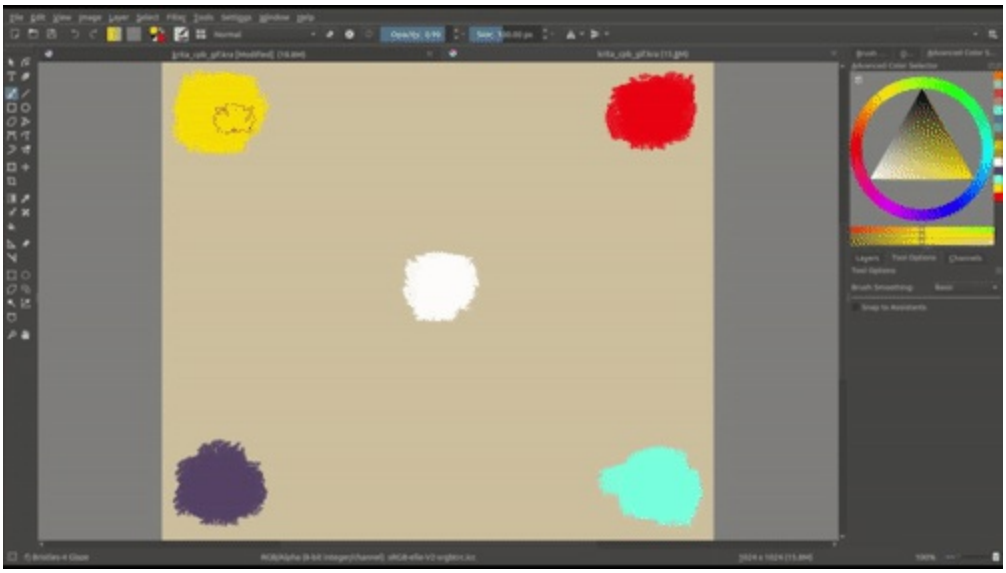
everywhere; when we think of the stereotypical artist we might imagine someone with a few **brushes** in one hand and a wooden **palette** in the other. Whether it's oils, watercolor, or other traditional media, for the artist to have absolute control over their colors it's crucial to have some kind of palette, plate, jar, or other **off-canvas area** to mix colors together. While it's easy to overlook this in digital painting (where selecting fresh new colors without mixing at all is both easy and free), Krita has a few very useful and unique features for off-canvas mixing.

Color Picker Blending

New in version 4.1.

Krita, like almost every art and graphics program, has a Color Selector Tool which allows you to very quickly sample a color from any pixel on your canvas. While this tool may seem relatively simple and humble, it is also one of the most important and commonly used tools in the digital artist's toolbox - perhaps only second to the brush! In fact, the color picker tool is at the very heart of mixing colors, and is often used in combination with on-canvas techniques like glazing and scumbling to produce smooth blends of color.

And still, there is more to this little tool than meets the eye! Not only can you configure Krita's color picker to sample from the average color of a **radius** of pixels, Krita's Color Picker also has a unique **blending** feature: a powerful and intuitive tool for off-canvas color mixing!



The Color Picker Blending feature changes the way that picking colors has traditionally worked for decades; instead of completely replacing your current brush color with the newly sampled color, *blending allows you to quickly "soak up" some portion of the sampled color*, which is then mixed with your current brush color.

You can use Color Picker Blending much like a physical paint brush in traditional media. If you were to dip your paint brush into a pool of *blue* paint, and then immediately dip again into a pool of *red* paint and paint a stroke across your canvas, the stoke wouldn't be pure red - it would be some combination of blue and red which would mix to create an intermediate purple color. Which shade of purple would depend on the ratio of paints that mix together within the hairs of your brush, and this is essentially what the Color Picker's "blend" option controls: what percentage of sampled color is mixed together with your current brush color!

Not only does Krita's Color Picker Blending feel even more like mixing paints, it is also completely off-canvas and independent of opacity, flow, shape, and other brush settings. Furthermore, unlike most on-canvas mixing techniques, Color Picker Blending works regardless of the location of colors on your canvas - enabling you to mix with colors at any position, on any layer, or even in different documents! Quickly mix lighting colors with local colors, mix the ambient sky color into shadows, create atmospheric depth, mix from a preselected palette of colors in another layer/document, etc.

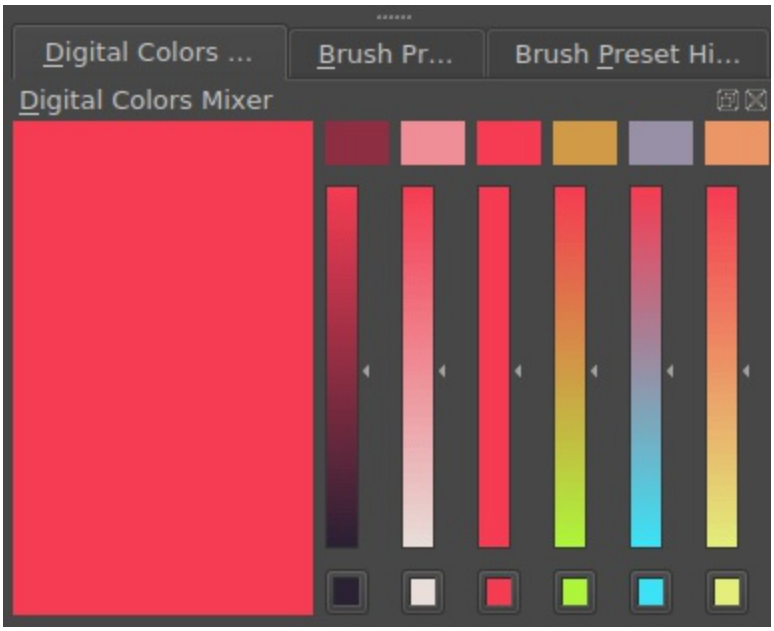
To use Color Picker Blending, simply set the “blend” option in the **Tool Options Docker** while the Color Picker Tool is active; setting blend to 100% will cause your Color Picker to work in the traditional way (completely replacing your brush color with the picked color), setting to around 50% will give you a half-way mix between colors, and setting to a lower value will create more subtle shifts in colors each click. Of course, blending affects both your dedicated Color Picker Tool as well as the `Ctrl` +  shortcut.

Note

Clicking and dragging the Color Picker around the canvas currently causes it to sample many times as it switches pixels. You can use this trait to quickly soak up more color by “dipping” your color picker into color and swirling it around. This can be pretty satisfying! However, this also means that some care must be taken to prevent from accidentally picking up more color than you want. It’s pretty easy to click a single pixel only one time using a **mouse**, but when painting with a **drawing tablet and pen** it can sometimes be desirable to use a slightly lower blend setting!

The Digital Colors Mixer

Somewhat hidden away in the **Dockers** menu (*Settings* ▸ *Dockers* ▸ *Digital Colors Mixer*), this can be a useful tool for off-canvas mixing. The Digital Colors Mixer looks a little bit like an audio mixing board that you’d see in a recording studio, but instead of mixing music it mixes colors! It contains 6 independent **color mixers** that mix your current brush color with any color of your choosing.



By clicking the **color buttons** below each mixer you can choose a palette of colors to mix with. Above each mixer is a **color patch** that will produce a color that mixes some amount of your current brush color with the palette color. Colors towards the top of the mixer will deliver subtle changes to your current color, while colors towards the bottom will be much closer to the palette color of that channel.

Other Tips

Outside of making it easier to create smooth gradients, mixing has another benefit: It allows you to create a cohesive piece.

Limiting the number of colors we use and then mixing tends to give a more cohesive palette, as we're not trying to do too much at once. This cohesive palette in turn means it will become easier to create a certain mood in an image. Sometimes, mixing in a little bit of accent color can also create unexpected results which in turn can be a little discovery for the audience to delight over as they discover the world of your image.

What we can learn from this, is that the next time we select, say, gray, instead of reaching for a random or generic gray from the Advanced Color Selector, consider using one of Krita's many wonderful mixing tools to create an

interesting and fitting gray from hues that are roughly complementary (opposite each other on the hue wheel).

While on-canvas and off-canvas techniques are fundamentally different categories of mixing colors, they are not mutually exclusive. All of the mixing methods in this article have pros and cons; different tools can be useful for different situations, and combining various techniques can be extremely powerful and fun!

Finally, mixing colors will often go far better in a higher bit-depth like 16bit, though it'll make the image take up much more working **memory** (RAM). Furthermore, using a linear color space can often give far better mixtures than a **gamma-corrected** one, though doing sketches and line art is easier to do in a gamma-corrected space.

Color Models

Krita has many different color spaces and models. Following here is a brief explanation of each, and their use-cases.

RGB

Red, Green, Blue.

These are the most efficient primaries for light-based color mixing, like computer screens. Adding Red, Green and Blue light together results in White, and is thus named the additive color wheel.

RGB is used for two purposes:

Images that are meant for viewing on a screen:

- So that could be images for the web, buttons, avatars, or just portfolio images.
- Or for Video games, both sprites and textures are best in RGB there.
- Or for 3d rendering, visual effects and cg animation.

And for the working space. A working space is an RGB gamut that is really large and predictable, meaning it’s good for image manipulation. You use this next to a profiled monitor. This way you can have precise colors while also being able to view them correctly on multiple screens.

Blending modes in RGB

Color 1			Color 2			Normal			Multiply			Screen	
R	**G**	**B**	**R**	**G**	**B**	**R**	**G**	**B**	**R**	**G**	**B**	**R**	**G**

R & G	1.0	0.0	0.0	0.0	1.0	0.0	0.5	0.5	0.0	0.0	0.0	0.0	1.0	1.0
Gray	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.25	0.25	0.25	0.75	0.75

RGB models: HSV, HSL, HSI and HSY

These are not included as their own color spaces in Krita. However, they do show up in the blending modes and color selectors, so a brief overview:

–Images of relationship rgb-hsv etc.

Hue
: The tint of a color, or, whether it's red, yellow, green, etc. Krita's Hue is measured in 360 degrees, with 0 being red, 120 being green and 240 being blue.

Saturation
: How vibrant a color is. Saturation is slightly different between HSV and the others. In HSV it's a measurement of the difference between two base colors being used and three base colors being used. In the others it's a measurement of how close a color is to gray, and sometimes this value is called **Chroma**. Saturation ranges from 0 (gray) to 100 (pure color).

Value
: Sometimes known as Brightness. Measurement of how much the pixel needs to light up. Also measured from 0 to 100.

Lightness
: Where a color aligns between white and black. This value is non-linear, and puts all the most saturated possible colors at 50. Ranges from 0 to 100.

Intensity
: Similar to lightness, except it acknowledges that yellow (1,1,0) is lighter

than blue (0,0,1). Ranges from 0 to 100.

Luma (Y’)
: Similar to lightness and Intensity, except it weights the red, green and blue components based real-life measurements of how much light a color reflects to determine its lightness. Ranges from 0 to 100. Luma is well known for being used in film-color spaces.

Grayscale

This color space only registers gray values. This is useful, because by only registering gray values, it only needs one channel of information, which in turn means the image becomes much lighter in memory consumption!

This is useful for textures, but also anything else that needs to stay grayscale, like Black and White comics.

	Color 1	Color 2	Normal	Multiply	Screen
	G	**G**	**G**	**G**	**G**
Gray	0.5	0.5	0.5	0.25	0.75

CMYK

Cyan, Magenta, Yellow, Key

This is the color space of printers. Unlike computers, printers have these four colors, and adding them all adds up to black instead of white. This is thus also called a ‘subtractive’ color space.

Color 1	Color 2	Normal	Multiply

	C	M	Y	K	C	M	Y	K	C	M	Y	K	C	M	Y
R & G	0.0	1.0	1.0	0.0	1.0	0.0	1.0	0.0	0.5	0.5	1.0	0.0	0.25	0.25	1.
Gray	0.0	0.0	0.0	0.5	0.0	0.0	0.0	0.5	0.0	0.0	0.0	0.5	0.0	0.0	0.

There’s also a difference between ‘colored gray’ and ‘neutral gray’ depending on the profile.

	25%				50%				75%			
	C	M	Y	K	C	M	Y	K	C	M	Y	K
Colored Gray	0.25	0.25	0.25	0.25	0.5	0.5	0.5	0.5	0.75	0.75	0.75	0.75
Neutral Gray	0.0	0.0	0.0	0.25	0.0	0.0	0.0	0.5	0.0	0.0	0.0	0.75

In Krita, there's also the fact that the default color is a perfect black in RGB, which then gets converted to our default CMYK in a funny manner, giving a yellow look to the strokes. Again, another good reason to work in RGB and let the conversion be done by the printing house.

While CMYK has a smaller 'gamut' than RGB, however, it's still recommended to use an RGB working space profile to do your editing in. Afterwards, you can convert it to your printer's CMYK profile using either perceptual or relative colorimetric intent. Or you can just give the workspace rgb image to your printer and let them handle the work.

YCrCb

Luminosity, Red-chroma, Blue-chroma

YCrCb stands for:

Y'/Y
: Luma/Luminosity, thus, the amount of light a color reflects.

Cr
: Red Chroma. This value measures how red a color is versus how green it is.

Cb
: Blue Chroma. This value measures how blue a color is versus how yellow it is.

This color space is often used in photography and in (correct) implementations of JPEG. As a human you're much more sensitive to the lightness of colors, and thus JPEG tries to compress the Cr and Cb channels, and leave the Y channel in full quality.

Warning

Krita doesn't bundle a ICC profile for YCrCb on the basis of there being no open source ICC profiles for this color space. It's unusable without one, and also probably very untested.

XYZ

Back in 1931, the CIE (Institute of Color and Light), was studying human color perception. In doing so, they made the first color spaces, with XYZ being the one best at approximating human vision.

It's almost impossible to really explain what XYZ is.

Y
: Is equal to green.

Z
: Akin to blue.

X
: Is supposed to be red.

XYZ is used as a baseline reference for all other profiles and models. All

color conversions are done in XYZ, and all profiles coordinates match XYZ.

L*a*b*

Stands for:

L*
: Lightness, similar to luminosity in this case.

a*
: a* in this case is the measurement of how magenta a color is versus how green it is.

b*
: b* in this case is a measurement of how yellow a color is versus how blue a color is.

L*a*b* is supposed to be a more comprehensible variety of XYZ and the most ‘complete’ of all color spaces. It’s often used as an in between color space in conversion, but even more as the correct color space to do color-balancing in. It’s far easier to adjust the contrast and color tone in L*a*b*.

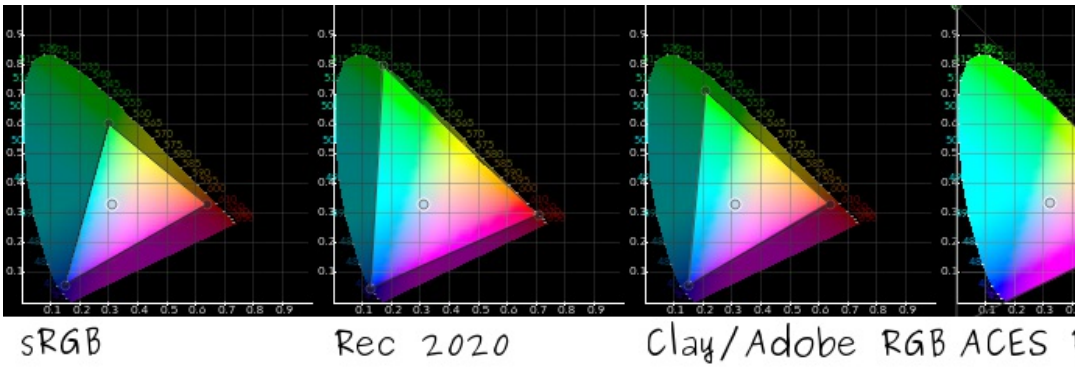
L*a*b* is technically the same as Photoshop’s LAB. Photoshop specifically uses CIELAB d50.

Filters and blending modes

Maybe you have noticed that blending modes in LAB don’t work like they do in RGB or CMYK. This is because the blending modes work by doing a bit of maths on the color coordinates, and because color coordinates are different per color space, the blending modes look different.

Color Space Size

Using Krita's color space browser, you can see that there are many different space sizes.

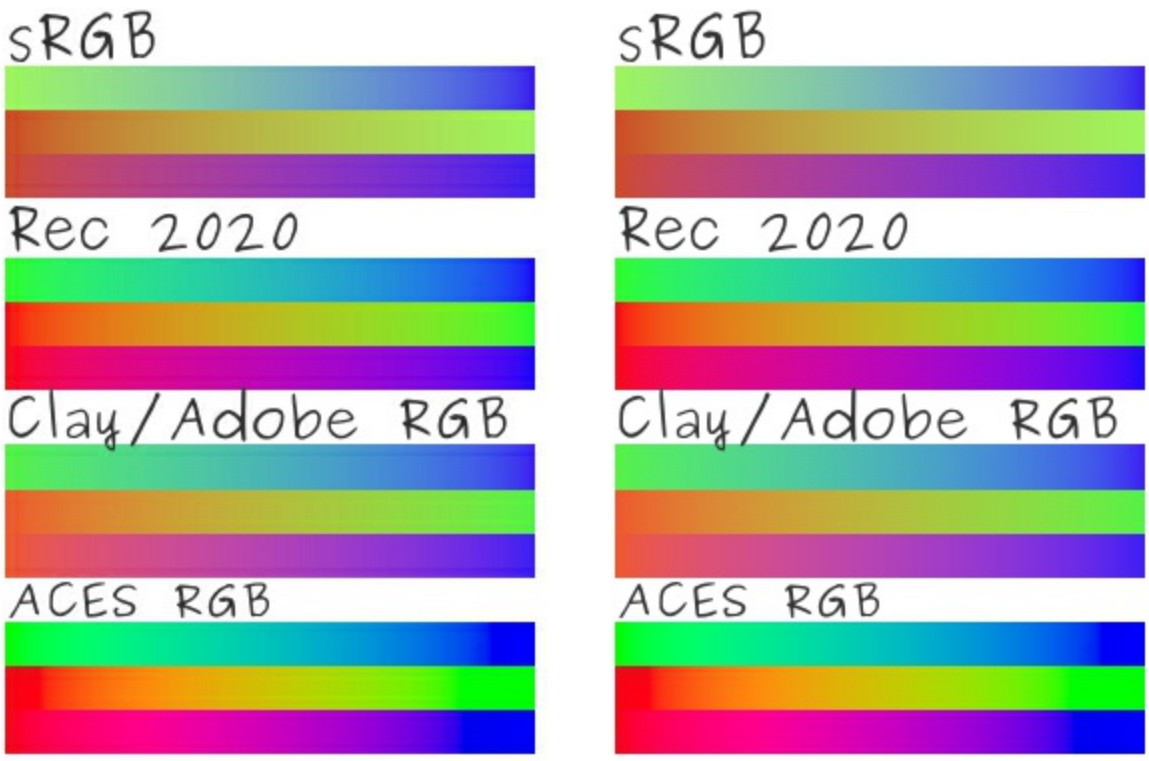


How do these affect your image, and why would you use them?

There are three primary reasons to use a large space:

1. Even though you can't see the colors, the computer program does understand them and can do color maths with it.
2. For exchanging between programs and devices: most CMYK profiles are a little bigger than our default sRGB in places, while in other places, they are smaller. To get the best conversion, having your image in a space that encompasses both your screen profile as your printer profile.
3. For archival purposes. In other words, maybe monitors of the future will have larger amounts of colors they can show (spoiler: they already do), and this allows you to be prepared for that.

Let's compare the following gradients in different spaces:



On the left we have an artifact-ridden color managed JPEG file with an ACES sRGBtrc v2 profile attached (or not, if not, then you can see the exact different between the colors more clearly). This should give an approximation of the actual colors. On the right, we have an sRGB PNG that was converted in Krita from the base file.

Each of the gradients is the gradient from the max of a given channel. As you can see, the mid-tone of the ACES color space is much brighter than the mid-tone of the RGB colorspace, and this is because the primaries are further apart.

What this means for us is that when we start mixing or applying filters, Krita can output values higher than visible, but also generate more correct mixes and gradients. In particular, when color correcting, the bigger space can help with giving more precise information.

If you have a display profile that uses a LUT, then you can use perceptual to give an indication of how your image will look.

Bigger spaces do have the downside they require more precision if you do not

want to see banding, so make sure to have at the least 16bit per channel when choosing a bigger space.

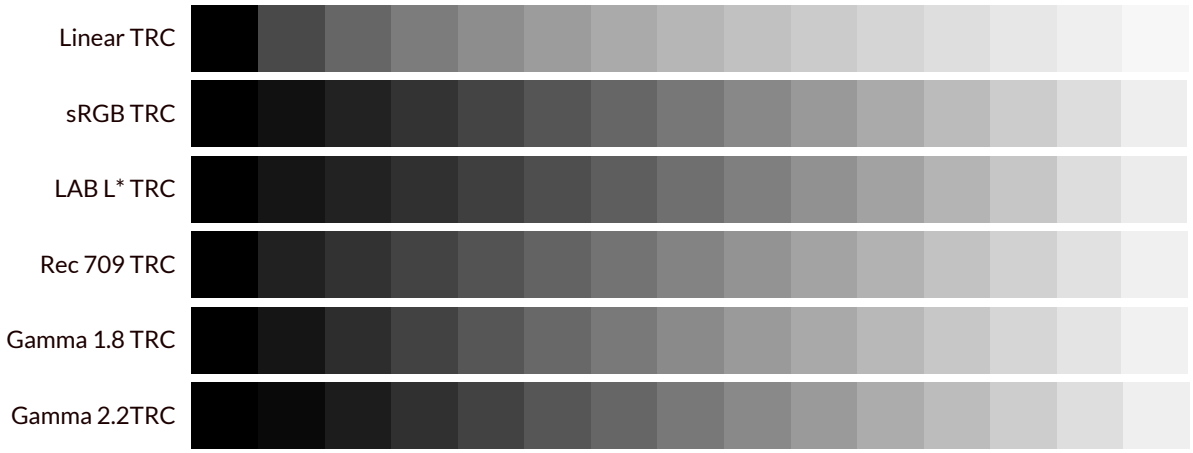
Gamma and Linear

Now, the situation we talk about when talking theory is what we would call ‘linear’. Each step of brightness is the same value. Our eyes do not perceive linearly. Rather, we find it more easy to distinguish between darker grays than we do between lighter grays.

As humans are the ones using computers, we have made it so that computers will give more room to darker values in the coordinate system of the image. We call this ‘gamma-encoding’, because it is applying a gamma function to the TRC or transfer function of an image. The TRC in this case being the Tone Response Curve or Tone Reproduction Curve or Transfer function (because color management specialists hate themselves), which tells your computer or printer how much color corresponds to a certain value.

One of the most common issues people have with Krita’s color management i
right colorspace to the encoded TRC. Above, the center Pepper is the right on
and assigned TRC are the same. To the left we have a Pepper encoded in sF
linear profile, and to the right we have a Pepper encoded with a linear TRC a
TRC. Image from Pepper & Carrot [https://www.peppercarrot.con

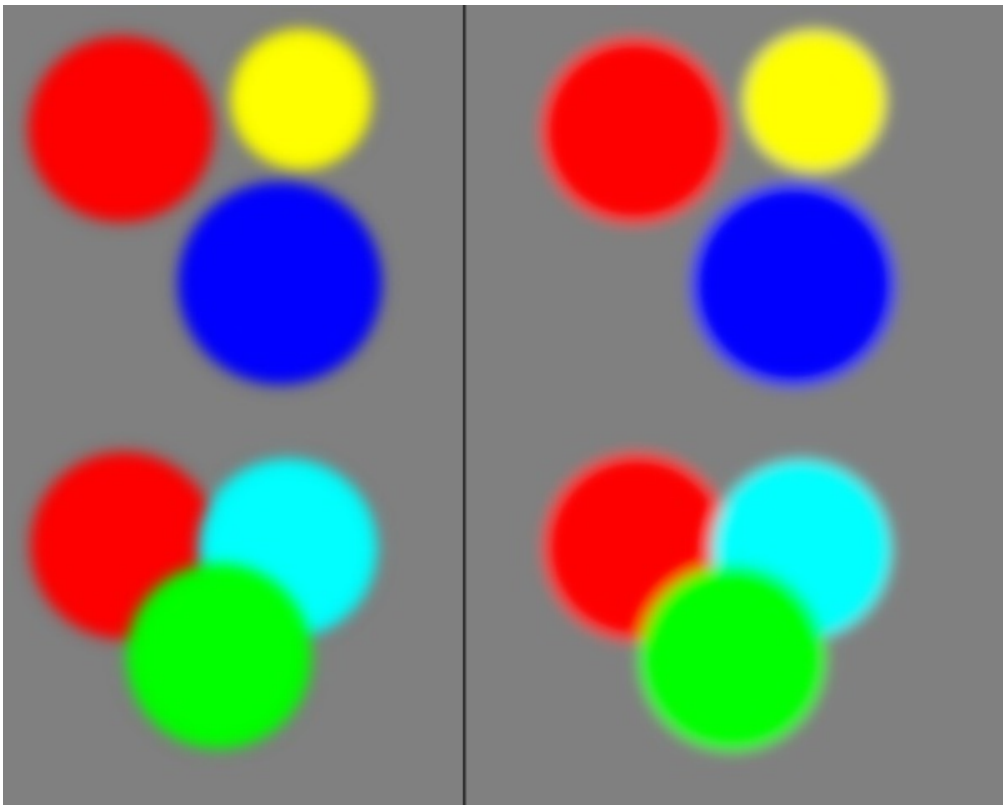
The following table shows how there’s a lot of space being used by lighter values in a linear space compared to the default sRGB TRC of our modern computers and other TRCs available in our delivered profiles:



If you look at linear of Rec. 709 TRCs, you can see there's quite a jump between the darker shades and the lighter shades, while if we look at the Lab L* TRC or the sRGB TRC, which seem more evenly spaced. This is due to our eyes' sensitivity to darker values. This also means that if you do not have enough bit depth, an image in a linear space will look as if it has ugly banding. Hence why, when we make images for viewing on a screen, we always use something like the Lab L*, sRGB or Gamma 2.2 TRCs to encode the image with.

However, this modification to give more space to darker values does lead to wonky color maths when mixing the colors.

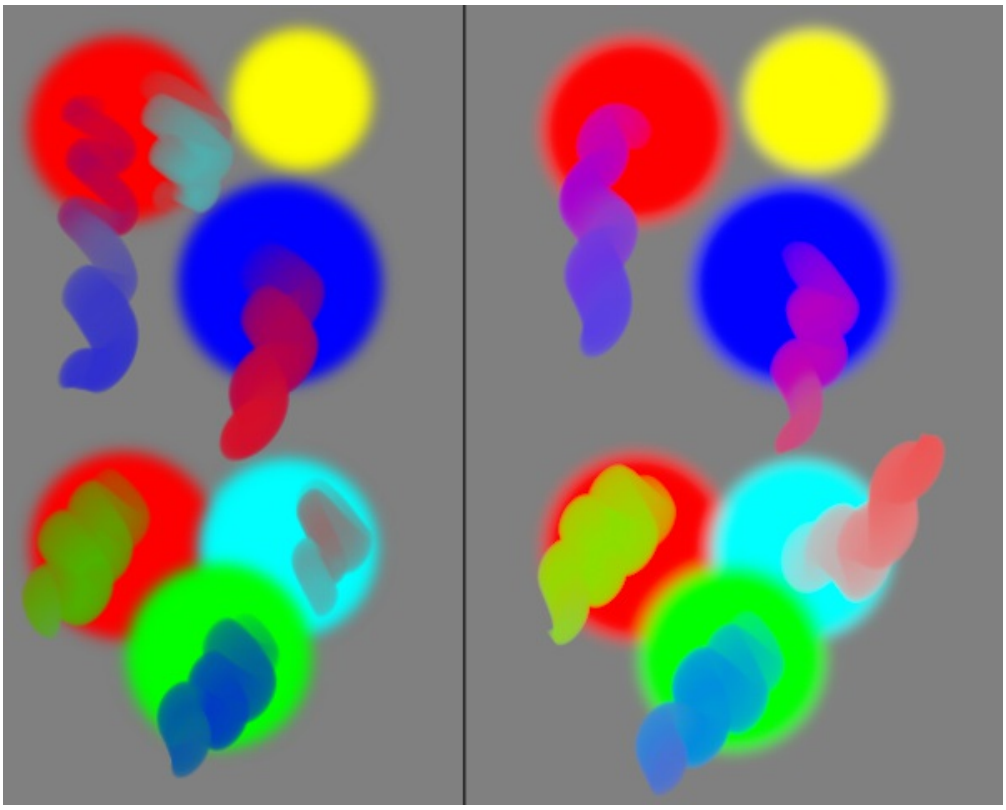
We can see this with the following experiment:



Left: Colored circles blurred in a regular sRGB space. **Right:** Colored circles blurred in a linear space.

Colored circles, half blurred. In a gamma-corrected environment, this gives an odd black border. In a linear environment, this gives us a nice gradation.

This also counts for Krita’s color smudge brush:



That’s right, the ‘muddying’ of colors as is a common complaint by digital painters everywhere, is in fact, a gamma-corrected colorspace mucking up your colors. If you had been working in LAB to avoid this, be sure to try out a linear rgb color space.

What is happening under the hood

Imagine we want to mix red and green.

First, we would need the color coordinates of red and green inside our color space’s color model. So, that’d be…

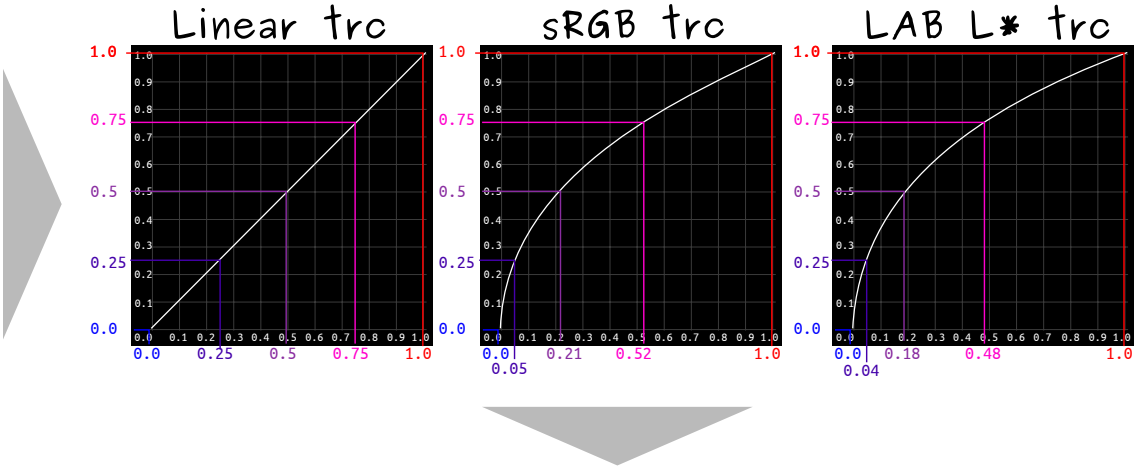
Color	Red	Green	Blue
Red	1.0	0.0	0.0

Green	0.0	1.0	0.0

We then average these coordinates over three mixes:

	Red	**Mix1**	**Mix2**	**Mix3**	**Green**
Red	1.0	0.75	0.5	0.25	0.0
Green	0.0	0.25	0.5	0.75	1.0
Blue	0.0	0.0	0.0	0.0	0.0

But to figure out how these colors look on screen, we first put the individual values through the TRC of the color-space we're working with:

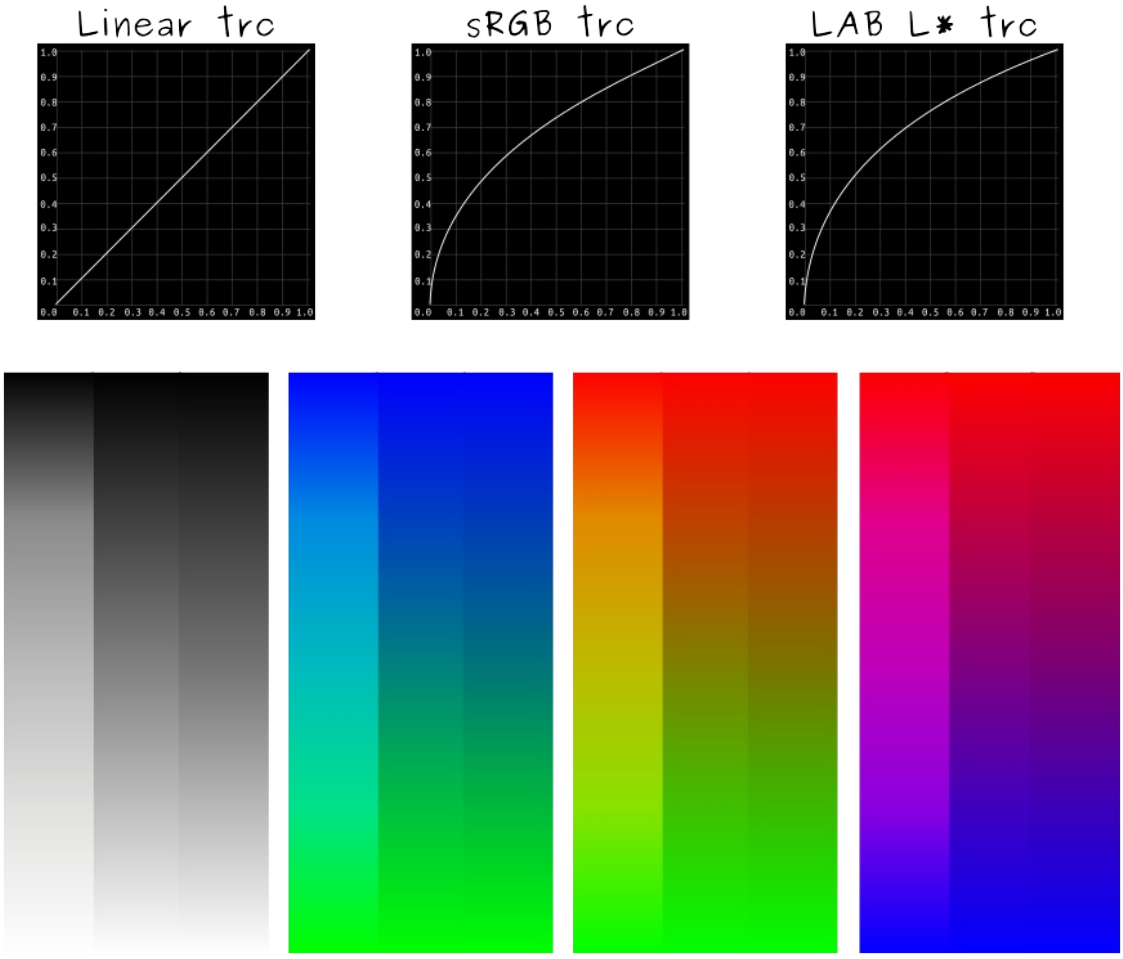


Then we fill in the values into the correct spot. Compare these to the values of the mixture table above!

	Linear TRC			sRGB TRC			Lab L* TRC			Rec 709 TRC			Gamma 1.8			Gamma 2.2		
	R	G	B	R	G	B	R	G	B	R	G	B	R	G	B	R	G	B
Red	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0
Mix 1	0.75	0.25	0.0	0.52	0.05	0.0	0.48	0.04	0.0	0.56	0.08	0.0	0.60	0.08	0.0	0.53	0.05	0.0
Mix 2	0.5	0.5	0.0	0.21	0.21	0.0	0.18	0.18	0.0	0.27	0.27	0.0	0.29	0.29	0.0	0.22	0.22	0.0
Mix 3	0.25	0.75	0.0	0.05	0.52	0.0	0.04	0.48	0.0	0.08	0.56	0.0	0.08	0.60	0.0	0.05	0.53	0.0
Green	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0

And this is why color mixtures are lighter and softer in linear space. Linear space is more physically correct, but sRGB is more efficient in terms of space, so hence why many images have an sRGB TRC encoded into them. In case this still doesn't make sense: *sRGB gives largely* **darker** *values than linear space for the same coordinates*.

So different TRCs give different mixes between colors, in the following example, every set of gradients is in order a mix using linear TRC, a mix using sRGB TRC and a mix using Lab L* TRC.



So, you might be asking, how do I tick this option? Is it in the settings somewhere? The answer is that we have several ICC profiles that can be used for this kind of work:

- scRGB (linear)
- All ‘elle’-profiles ending in ‘g10’, such as *sRGB-elle-v2-g10.icc*.

In fact, in all the ‘elle’-profiles, the last number indicates the gamma. 1.0 is linear, higher is gamma-corrected and ‘srgbtrc’ is a special gamma correction for the original sRGB profile.

If you use the color space browser, you can tell the TRC from the ‘estimated gamma’(if it’s 1.0, it’s linear), or from the TRC widget in Krita 3.0, which

looks exactly like the curve graphs above.

Even if you do not paint much, but are for example making textures for a videogame or rendering, using a linear space is very beneficial and will speed up the renderer a little, for it won’t have to convert images on its own.

The downside of linear space is of course that white seems very overpowered when mixing with black, because in a linear space, light grays get more room. In the end, while linear space is physically correct, and a boon to work in when you are dealing with physically correct renderers for videogames and raytracing, Krita is a tool and no-one will hunt you down for preferring the dark mixing of the sRGB TRC.

Profiling and Calibration

So to make it simple, a color profile is just a file defining a set of colors inside a pure XYZ color cube. This “color set” can be used to define different things:

- the colors inside an image
- the colors a device can output

Choosing the right workspace profile to use depends on how much colors you need and on the bit depth you plan to use. Imagine a line with the whole color spectrum from pure black (0,0,0) to pure blue (0,0,1) in a pure XYZ color cube. If you divide it choosing steps at a regular interval, you get what is called a linear profile, with a gamma=1 curve represented as a straight line from 0 to 1. With 8bit/channel bit depth, we have only 256 values to store this whole line. If we use a linear profile as described above to define those color values, we will miss some important visible color change steps and have a big number of values looking the same (leading to posterization effect).

This is why was created the sRGB profile to fit more different colors in this limited amount of values, in a perceptually regular grading, by applying a custom gamma curve (see picture here: https://en.wikipedia.org/wiki/SRGB) to emulate the standard response curve of old CRT screens. So sRGB profile is optimized to fit all colors that most common screen can reproduce in those 256 values per R/G/B channels. Some other profiles like Adobe RGB are optimized to fit more printable colors in this limited range, primarily extending cyan-green hues. Working with such profile can be useful to improve print results, but is dangerous if not used with a properly profiled and/or calibrated good display. Most common CMYK workspace profile can usually fit all their colors within 8bit/channel depth, but they are all so different and specific that it’s usually better to work with a regular RGB workspace first and then convert the output to the appropriate CMYK profile.

Starting with 16bit/channel, we already have 65536 values instead of 256, so

we can use workspace profiles with higher gamut range like Wide-gamut RGB or Pro-photo RGB, or even unlimited gamut like scRGB.

But sRGB being a generic profile (even more as it comes from old CRT specifications…), there are big chances that your monitor have actually a different color response curve, and so color profile. So when you are using sRGB workspace and have a proper screen profile loaded (see next point), Krita knows that the colors the file contains are within the sRGB color space, and converts those sRGB values to corresponding color values from your monitor profile to display the canvas.

Note that when you export your file and view it in another software, this software has to do two things:

- read the embed profile to know the “good” color values from the file (which most software do nowadays; when they don’t they usually default to sRGB, so in the case described here we’re safe)
- and then convert it to the profile associated to the monitor (which very few software actually does, and just output to sRGB.. so this can explain some viewing differences most of the time).

Krita uses profiles extensively, and comes bundled with many.

The most important one is the one of your own screen. It doesn’t come bundled, and you have to make it with a color profiling device. In case you don’t have access to such a device, you can’t make use of Krita’s color management as intended. However, Krita does allow the luxury of picking any of the other bundled profiles as working spaces.

Profiling devices

Profiling devices, called Colorimeters, are tiny little cameras of a kind that you connect to your computer via an usb, and then you run a profiling software (often delivered alongside of the device).

Note

If you don’t have software packaged with your colorimeter, or are unhappy with the results, we recommend ArgyllCMS [https://www.argyllcms.com/].

The little camera then measures what the brightest red, green, blue, white and black are like on your screen using a predefined white as base. It also measures how gray the color gray is.

It then puts all this information into an ICC profile, which can be used by the computer to correct your colors.

It’s recommended not to change the “calibration” (contrast, brightness, you know the menu) of your screen after profiling. Doing so makes the profile useless, as the qualities of the screen change significantly while calibrating.

To make your screen display more accurate colors, you can do one or two things: profile your screen or calibrate and profile it.

Just profiling your screen means measuring the colors of your monitor with its native settings and put those values in a color profile, which can be used by color-managed application to adapt source colors to the screen for optimal result. Calibrating and profiling means the same except that first you try to calibrate the screen colors to match a certain standard setting like sRGB or other more specific profiles. Calibrating is done first with hardware controls (lightness, contrast, gamma curves), and then with software that creates a vcgt (video card gamma table) to load in the GPU.

So when or why should you do just one or both?

Profiling only:

With a good monitor

> You can get most of the sRGB colors and lot of extra colors not inside sRGB. So this can be good to have more visible colors.

With a bad monitor

> You will get just a subset of actual sRGB, and miss lot of details, or even have hue shifts. Trying to calibrate it before profiling can help to get

closer to full-sRGB colors.

Calibration+profiling:

Bad monitors
: As explained just before.

Multi-monitor setup
: When using several monitors, and specially in mirror mode where both monitor have the same content, you can’t have this content color-managed for both screen profiles. In such case, calibrating both screens to match sRGB profile (or another standard for high-end monitors if they both support it) can be a good solution.

Soft-proofing
: When you need to match an exact rendering context for soft-proofing, calibrating can help getting closer to the expected result. Though switching through several monitor calibration and profiles should be done extremely carefully.

Scene Linear Painting

Previously referred to as HDR painting and Scene Referred painting, Scene Linear Painting is doing digital painting in a peculiar type of colorspace. It is painting in a color space that is…

1. Linear - there's no gamma encoding, or tone-mapping or whatever going on with the pixels you manipulate. (This is different from the pixels you see, but we'll get to that later)
2. Floating Point - So 16bit or 32bit floating point per channel.

These are the two important characteristics. The colorspace has a few more properties than this, such as the white point, or more importantly, the colorants that make up the gamut. But here's the thing, those two could be anything, as long as the space is linear and the color depth is floating point.

So, *Scene Linear is not a single one colorspace, but a* **TYPE** *of colorspace*. You can have a scene linear space that uses the sRGB/Rec. 709 colorants, or one that uses adobeRGB, or maybe one that uses Rec. 2020, as long as it is *linear* and in a *floating point bit depth*.

Note

If you want to create images for display on an HDR canvas, you will need to select the Rec. 2020 space profile with a linear gamma. The default profile in Krita for that is *Rec2020-elle-V4-g10.icc*.

These two factors are for one reason: To make black and white arbitrary values. This might seem a bit weird. But when you are dealing with light-sources, you are dealing with a massive range of contrasts, and will have to decide afterwards which white and black you'd like to have. This is what the scene means in scene-linear, the relevant values are unique per scene, like a real world scene: a flower field lit by moonlight, a city in twilight or a sunny beach. You want to be able to put the right emphasis on the most important

contrasting values, and being able to choose what is white and what is black is a very powerful tool here. After all, humans in the real world can see much more when they get used to the dark, or to the sun, so why not apply that to how we make our images?

This is also why it needs to be Linear. Gamma and Tone-mapped color spaces are already choosing which contrast is the most important to you. But for that, they too need to choose what is white or black. Linear doesn't make such assumptions, so much better for when you want to choose yourself. You will eventually want to stick your image through some tone-mapping or gamma correction, but only at the end after you have applied filters and mixed colors!

In fact, there's always a non-destructive sort of transform going on while you are working on your image which includes the tone-mapping. This is called a display or view transform, and they provide a sort of set of binoculars into the world of your image. Without it, your computer cannot show these colors properly; it doesn't know how to interpret it properly, often making the image too dark. Providing such a transform and allowing you to configure it is the prime function of color management.

Between different view and display transforms, there's also a difference in types. Some are really naive, others are more sophisticated, and some need to be used in a certain manner to work properly. The ICC color management can only give a certain type of view transforms, while OCIO color management in the LUT docker can give much more complex transforms easily configurable and custom settings that can be shared between programs.

Above, an example of the more naive transform provided by going from sc
regular sRGB, and to the right a more sophisticated transform coming from
OCIO configuration. Look at the difference between the paws. Image by Wo
Westerflier, License: CC-BY-SA

Conversely, transforming and interpreting your image's colors is the only thing OCIO can do, and it can do it with really complex transforms, really fast. It doesn't understand what your image's color space is originally, doesn't understand what CMYK is, and there's also no such thing as a OCIO color profile. Therefore you will need to switch to an ICC workflow if you wish to prepare for print.

Yes, but what is the point?

The point is making things easier in the long run:

1. It is easier to keep saturated non-muddy colors in a linear space.
2. The high bit depth makes it easier to get smoother color mixes.
3. Filters are more powerful and give nicer results in this space. It is far more easy to get nice blurring and bokeh results.
4. Simple Blending Modes like Multiply or Addition are suddenly black magic. This is because Scene-Linear is the closest you can get to the physical (as in, physics, not material) model of color where multiplying colors with one another is one of the main ways to calculate the effect of light.
5. Combining painting with other image results such as photography and physically based rendering is much easier as they too work in such a type of colorspace. So you could use such images as a reference with little qualms, or make textures that play nice with such a renderer.

So the advantages are prettier colors, cooler filter results, more control and easier interchange with other methods.

Okay, but why isn't this all the rage then?

Simply put, because while it's easier in the long run, you will also have to drop tools and change habits…

In particular, there are many tools in a digital painter's toolbox that have **hard-coded assumptions about black and white**.

A very simple but massive problem is one with **inversion**. Inverting colors is done code-wise by taking the color for white and subtracting the color you want to invert from it. It's used in many blending modes. But often the color white is hardcoded in these filters. There's currently no application out there that allows you to define the value range that inversion is done with, so inverting is useless. And that also means the filters and blending modes that use it, such as (but not limited to)…

- Screen (invert+multiply+invert)
- Overlay (screens values below midtone-value, in sRGB this would be middle gray)
- Color-dodge (divides the lower color with an inversion of the top one)

- Color-burn (inverts the lower color and then divides it by the top color)
- Hardlight (a different way of doing overlay, including the inversion)
- Softlight (uses several inversions along the way)

Conversely Multiply, Linear Dodge/Addition (they're the same thing), Subtract, Divide, Darker (only compares colors' channel values), Lighter (ditto), and Difference *are fine to use*, as long as the program you use doesn't do weird clipping there.

Another one is HSL, HSI and HSY algorithms. They too need to assume something about the top value to allow scaling to white. HSV doesn't have this problem. So it's best to use an HSV color selector.

For the blending modes that use HSY, there's always the issue that they tend to be hardcoded to sRGB/Rec. 709 values, but are otherwise fine (and they give actually far more correct results in a linear space). So these are not a good idea to use with wide-gamut colorspaces, and due to the assumption about black and white, not with scene linear painting. The following blending modes use them:

- Color
- Luminosity
- Saturation
- Darker Color (uses luminosity to determine the color)
- Lighter Color (Ditto)

So that is the blending modes. Many filters suffer from similar issues, and in many applications, filters aren't adjusted to work with arbitrary whites.

Speaking of filters, when using the transform tool, you should also avoid using lanczos3, it'll give a weird black halo to sharp contrasts in scene-linear. The bilinear interpolation filter will work just fine in this case.

The second big problem is that **black doesn't work quite the same**.

If you have mixed pigments before, you will know that black can quite easily overpower the other colors, so you should only add the tiniest amount of it to a mixture. White in contrast gets dirtied quite easily.

In a Scene Linear Color space, this is flipped. White is now more overpowering and black gets washed out super quickly. This relates to the additive nature of digital color theory, that becomes more obvious when working in linear.

This makes sketching a bit different, after all, it's really difficult to make marks now. To get around this, you can do the following:

- Sketch on a mid-gray background. This is recommended anyway, as it serves as a neutral backdrop. For a linear space, 18% or 22% gray would be a good neutral.
- Make a special brush that is more opaque than the regular sketching brushes you use.
- Or conversely, sketch with white instead.
- For painting, block out the shapes with a big opaque brush before you start doing your mixing.

Overall, this is something that will take a little while getting used to, but you will get used to it soon enough.

Finally, there's the **issue of size**.

16 bit float per channel images are big. 32 bit float per channel images are bigger. This means that they will eat RAM and that painting and filtering will be slower. This is something that will fix itself over the years, but not many people have such a high-end PC yet, so it can be a blocker.

So the issues are tools, expectations and size.

In Summary

Scene Linear Painting is painting an image in a color space that is linear and has a floating point bit depth. This does not assume anything about the values of black and white, so you can only use tools that don't assume anything about the values of black and white. It has the advantage of having nicer filter results and better color mixtures as well as better interoperability with other scene-linear output.

To be able to view such an image you use a view transform, also called a display conversion. Which means that if you wish to finalize your image for the web, you make a copy of the image that goes through a display conversion or view transform that then gets saved to PNG, JPEG or TIFF.

Getting to actual painting

Now we've covered the theory, let us look at a workflow for painting scene linear.

Setting up the Canvas

Select either a 16bit or 32bit image. By default Krita will select a linear sRGB profile. If you want to create images for HDR display, you will need to make sure that the profile selected is the *Rec2020-elle-V4-g10.icc* profile. HDR images are standardised to use the Rec. 2020 gamut, which is much larger than sRGB in size, so this ensures you've got access to all the colors.

If you're working on a non-HDR enabled monitor, you should enable OCIO in the LUT docker.

Keep in mind everything mentioned above. Not all filters and not all blending modes work. This will improve in the future. Other than that, everything else is the same.

Picking really bright colors

Picking regular colors is easy, but how do we pick the really bright colors? There are three ways of getting access to the really bright colors in Krita:

1. By lowering the exposure in the LUT docker. This will increase the visible range of colors in the color selectors. You can even hotkey the exposure in the canvas input settings.
2. By setting the nits slider in the Small Color Selector higher than 100.
3. Or simply by opening the internal color selector by double clicking the dual color button and typing in values higher than 1 into the input field.
4. And finally by picking a really bright color from an image that has such

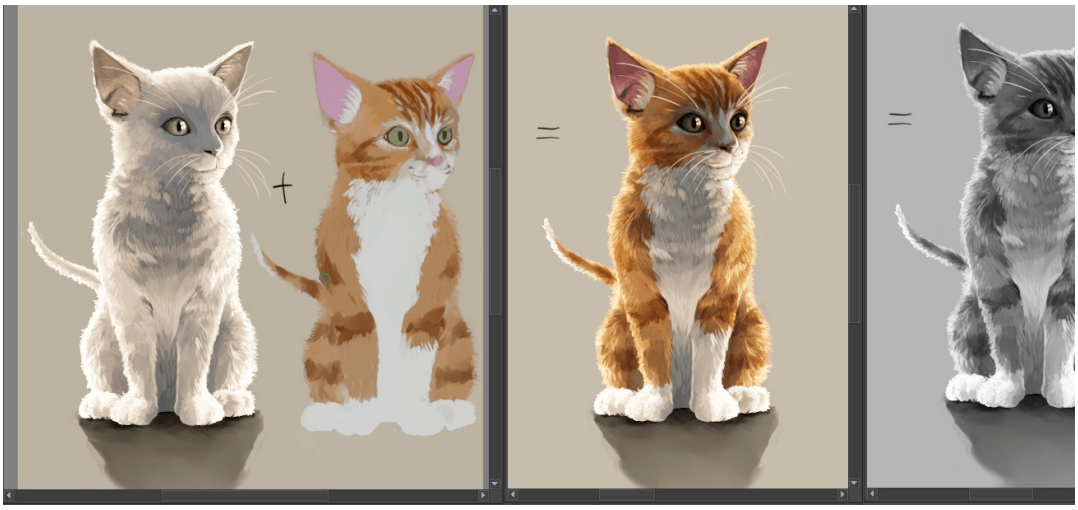
values.

Then paint. It’s recommended to make a bunch of swatches in the corner, at the least, until Krita’s new Palette docker allows you to save the values properly.

Lighting based workflow

So, we have our typical value based workflow, where we only paint the grays of the image so that we can focus on the values of the image. We can do something similar with Scene Linear Painting.

Where with the value based workflow you paint the image as if it were a grayscale of what you intended to paint, with a lighting based workflow you paint as if all the objects are white. The effect of the color of an object can be determined by multiplying its base color with the color of the light. So you could paint objects as if they were white, paint the colors on a separate layer and just use the Multiply blending mode to get the right colors.



The leftmost image is both the lighting based one and the color layer separate
two layers multiplied and the right a luminosity based view. This cat is a r
demonstrates why having textures and lighting separate could be i

You can even combine this with a value based workflow by opening a new view and setting the component to luminosity. That way you can see both the grayscale as well as the lighting based version of the image next to one

another.

The keen minded will notice that a lighting based workflow kind of resembles the idea of a light pass and a color pass in 3d rendering. And indeed, it is basically the same, so you can use lighting passes from 3d renders here, just save them as EXR and import them as a layer. One of the examples where scene linear painting simplifies combining methods.

Finishing up

When you are done, you will want to apply the view transform you have been using to the image (at the least, if you want to post the end result on the Internet)… This is called LUT baking and not possible yet in Krita. Therefore you will have to save out your image in EXR and open it in either Blender or Natron. Then, in Blender it is enough to just use the same OCIO config, select the right values and save the result as a PNG.

For saving HDR images, check the HDR Display page.

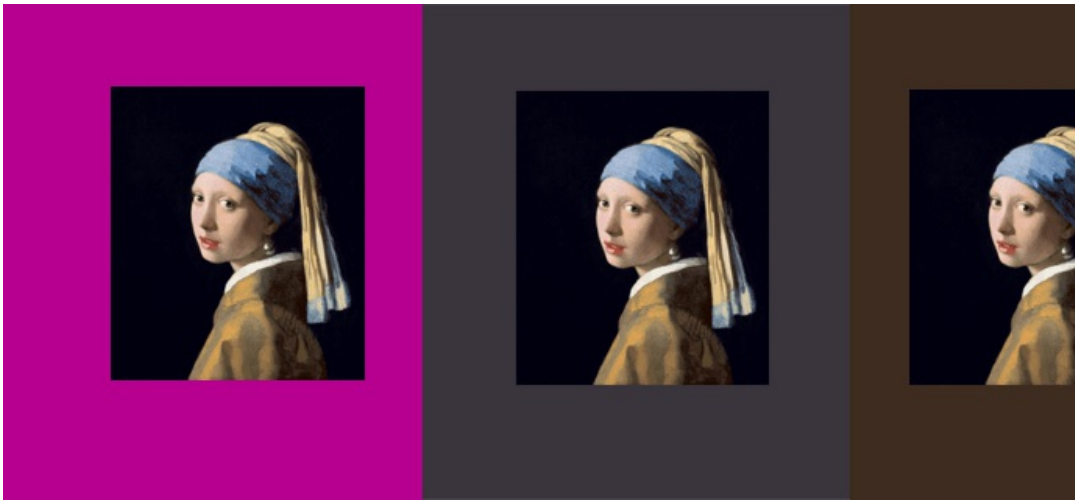
You can even use some of Blender's or Natron's filters at this stage, and when working with others, you would save out in EXR so that others can use those.

Viewing Conditions

We mentioned viewing conditions before, but what does this have to do with ‘white points’?

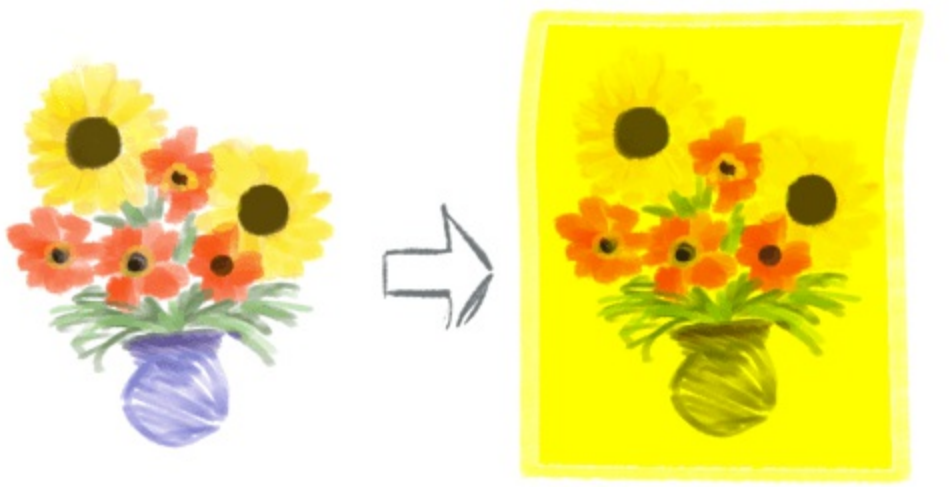
A lot actually, rather, white points describe a type of viewing condition.

So, usually what we mean by viewing conditions is the lighting and decoration of the room that you are viewing the image in. Our eyes try to make sense of both the colors that you are looking at actively (the colors of the image) and the colors you aren’t looking at actively (the colors of the room), which means that both sets of colors affect how the image looks.



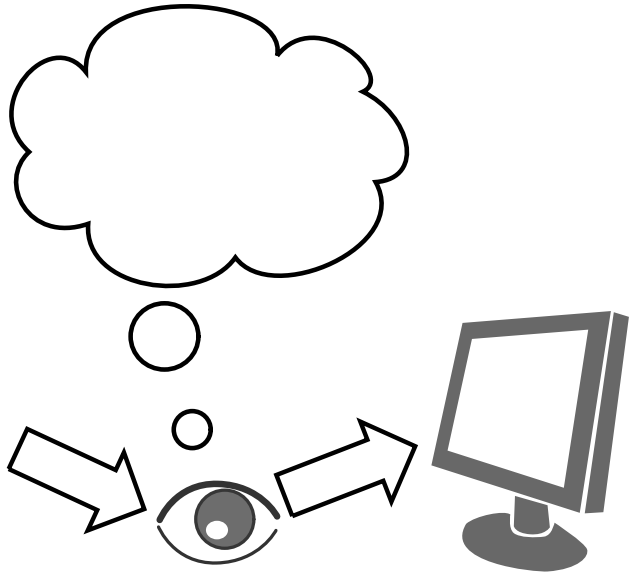
Left: Let’s ruin Vermeer by putting a bright purple background that asks for the famous painting itself. **Center**: a much more neutral backdrop that an inte hate but brings out the colors. **Right**: The approximate color that this painting in real life in the Maurits House, at the least, last time I was there. Original ir commons.

This is for example, the reason why museum exhibitioners can get really angry at the interior decorators when the walls of the museum are painted bright red or blue, because this will drastically change the way how the painting’s colors look. (Which, if we are talking about a painter known for their colors like Vermeer, could result in a really bad experience).



Lighting is the other component of the viewing condition which can have dramatic effects. Lighting in particular affects the way how all colors look. For example, if you were to paint an image of sunflowers and poppies, print that out, and shine a bright yellow light on it, the sunflowers would become indistinguishable from the white background, and the poppies would look orange. This is called metamerism [https://en.wikipedia.org/wiki/Metamerism_%28color%29], and it's generally something you want to avoid in your color management pipeline.

An example where metamerism could become a problem is when you start matching colors from different sources together.



For example, if you are designing a print for a red t-shirt that’s not bright red, but not super grayish red either. And you want to make sure the colors of the print match the color of the t-shirt, so you make a dummy background layer that is approximately that red, as correctly as you can observe it, and paint on layers above that dummy layer. When you are done, you hide this dummy layer and sent the image with a transparent background to the press.

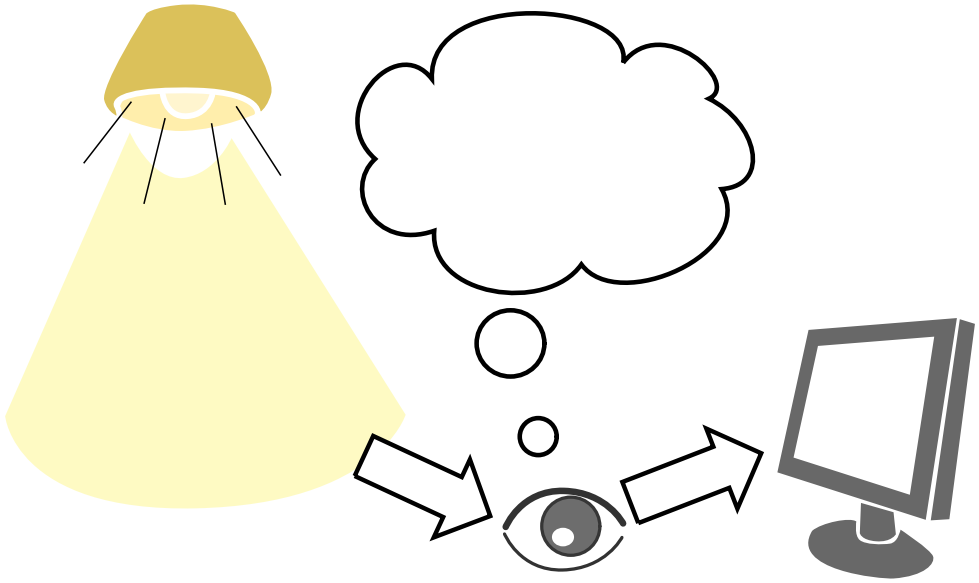
But when you get the t-shirt from the printer, you notice that all your colors look off, mismatched, and maybe too yellowish (and when did that T-Shirt become purple?).

This is where white points come in.

You probably observed the t-shirt in a white room where there were incandescent lamps shining, because as a true artist, you started your work in the middle of the night, as that is when the best art is made. However, incandescent lamps have a black body temperature of roughly 2300-2800K, which makes them give a yellowish light, officially called White Point A.

Your computer screen on the other hand, has a black body temperature of 6500K, also known as D65. Which is a far more blueish color of light than the lamps you are hanging.

What's worse, Printers print on the basis of using a white point of D50, the color of white paper under direct sunlight.



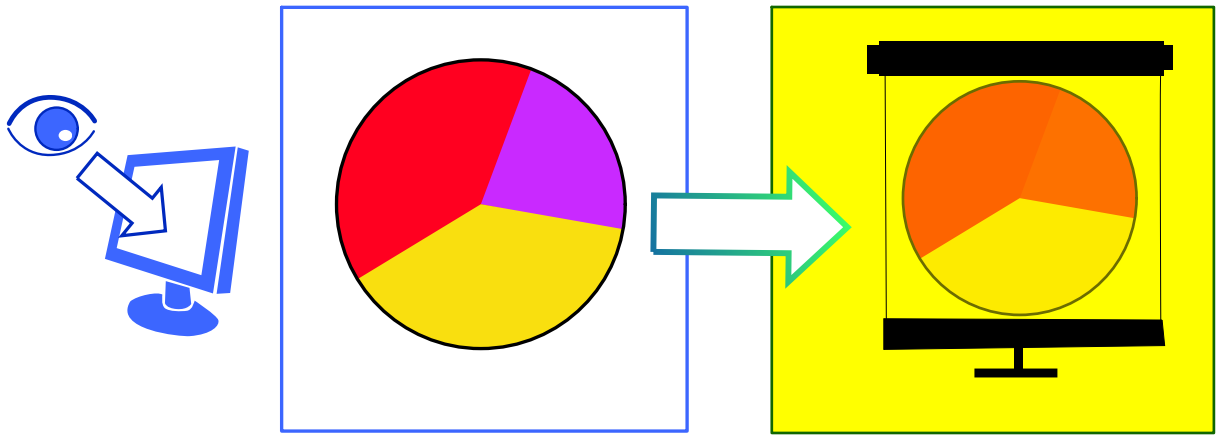
So, by eye-balling your t-shirt's color during the evening, you took its red color as transformed by the yellowish light. Had you made your observation in diffuse sunlight of an overcast (which is also roughly D65), or made it in direct sunlight light and painted your picture with a profile set to D50, the color would have been much closer, and thus your design would not be as yellowish.

Applying a white balance filter will sort of match the colors to the tone as in the middle, but you would have had a much better design had you designed against the actual color to begin with.

Now, you could technically quickly fix this by using a white balancing filter, like the ones in G'MIC, but because this error is caught at the end of the production process, you basically limited your use of possible colors when you were designing, which is a pity.

Another example where metamerism messes things up is with screen projections.

We have a presentation where we mark one type of item with red, another with yellow and yet another with purple. On a computer the differences between the colors are very obvious.



However, when we start projecting, the lights of the room aren't dimmed, which means that the tone scale of the colors becomes crunched, and yellow

becomes near indistinguishable from white. Furthermore, because the light in the room is slightly yellowish, the purple is transformed into red, making it indistinguishable from the red. Meaning that the graphic is difficult to read.

In both cases, you can use Krita's color management a little to help you, but mostly, you just need to be ''aware'' of it, as Krita can hardly fix that you are looking at colors at night, or the fact that the presentation hall owner refuses to turn off the lights.

That said, unless you have a display profile that uses LUTs, such as an OCIO LUT or a cLUT ICC profile, white point won't matter much when choosing a working space, due to weirdness in the ICC v4 workflow which always converts matrix profiles with relative colorimetric, meaning the white points are matched up.

File Formats

This category is for graphics file-formats. While most file-formats can be looked up on wikipedia, this doesn't always explain what the format can be used for and what its strengths and weaknesses are.

In this category we try to describe these in a manner that can be read by beginners.

Generally, there are the following features that people pay attention to in regards to file formats:

Compression

Compression is how the file-format tries to describe the image with as little data as possible, so that the resulting file is as small as it can get without losing quality.

What we generally see is that formats that are small on disk either lose image quality, or require the computer to spend a lot of time thinking about how the image should look.

Vector file-formats like `SVG` are a typical example of the latter. They are really small because the technology used to create them is based on mathematics, so it only stores maths-variables and can achieve very high quality. The downside is that the computer needs to spend a lot of time thinking about how it should look, and sometimes different programs have different ways of interpreting the values. Furthermore, vector file-formats imply vector graphics, which is a very different way of working than Krita is specialized in.

Lossy file formats, like `JPG` or `WebP` are an example of small on disk, but lowering the quality, and are best used for very particular types of images. Lossy thus means that the file format plays fast and loose with describing your image to reduce filesize.

Non-lossy or lossless formats, like `PNG`, `GIF` or `BMP` are in contrast, much heavier on disk, but much more likely to retain quality.

Then, there's proper working file formats like Krita's `KRA`, Gimp's `XCF`, Photoshop's `PSD`, but also interchange formats like `ORA` and `EXR`. These are the heaviest on the hard-drive and often require special programs to open them up, but on the other hand these are meant to keep your working environment intact, and keep all the layers and guides in them.

Metadata

Metadata is the ability of a file format to contain information outside of the actual image contents. This can be human readable data, like the date of creation, the name of the author, a description of the image, but also computer readable data, like an ICC profile which tells the computer about the qualities of how the colors inside the file should be read.

Openness

This is a bit of an odd quality, but it's about how easy it to open or recover the file, and how widely it's supported.

Most internal file formats, like PSD are completely closed, and it's really difficult for human outsiders to recover the data inside without opening Photoshop. Other examples are camera raw files which have different properties per camera manufacturer.

SVG, as a vector file format, is on the other end of the spectrum, and can be opened with any text-editor and edited.

Most formats are in-between, and thus there's also a matter of how widely supported the format is. JPG and PNG cannot be read or edited by human eyes, but the vast majority of programs can open them, meaning the owner has easy access to them.

Contents:

- *.bmp
- *.csv
- *.exr
- *.gbr
- *.gif
- *.gih
- *.jpg
- *.kpl
- *.kra
- *.ora
- *.pbm, *.pgm and *.ppm
- *.pdf
- *.png
- *.psd
- *.svg
- *.tiff
- Lossy and Lossless Image Compression

*.bmp

`.bmp`, or Bitmap, is the simplest raster file format out there, and, being patent-free, most programs can open and save bitmap files.

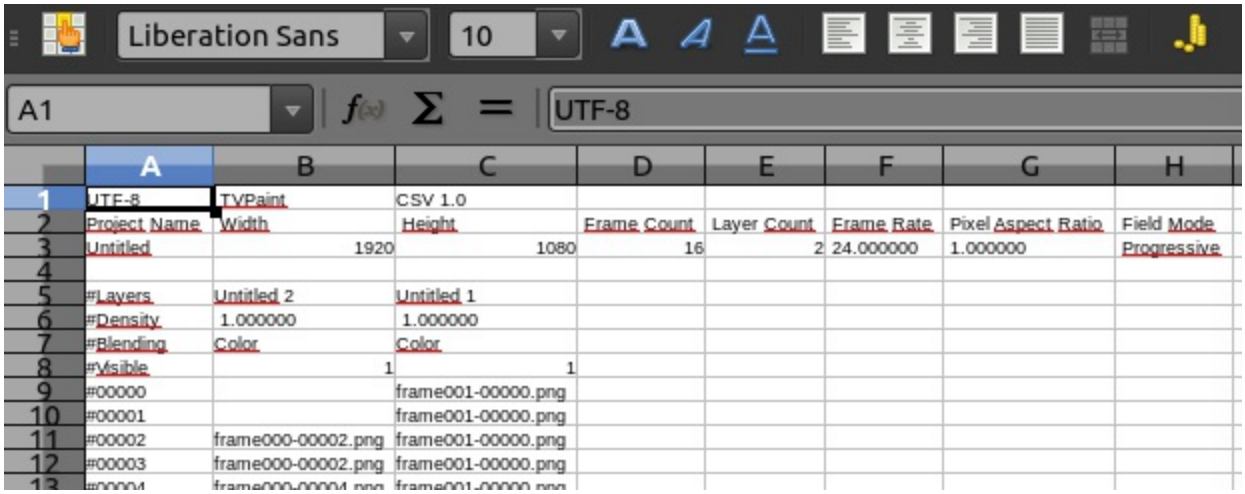
However, most programs don’t compress bitmap files, leading to BMP having a reputation for being very heavy. If you need a lossless file format, we actually recommend *.png.

*.csv

`.csv` is the abbreviation for Comma Separated Values. It is an open, plain text spreadsheet format. Since the CSV format is a plain text itself, it is possible to use a spreadsheet program or even a text editor to edit the `*.csv` file.

Krita supports the CSV version used by TVPaint, to transfer layered animation between these two softwares and probably with others, like Blender. This is not an image sequence format, so use the document loading and saving functions in Krita instead of the *Import animation frames* and *Render Animation* menu items.

The format consists of a text file with `.csv` extension, together with a folder under the same name and a `.frames` extension. The CSV file and the folder must be on the same path location. The text file contains the parameters for the scene, like the field resolution and frame rate, and also contains the exposure sheet for the layers. The folder contains *.png picture files. Unlike image sequences, a key frame instance is only a single file and the exposure sheet links it to one or more frames on the timeline.



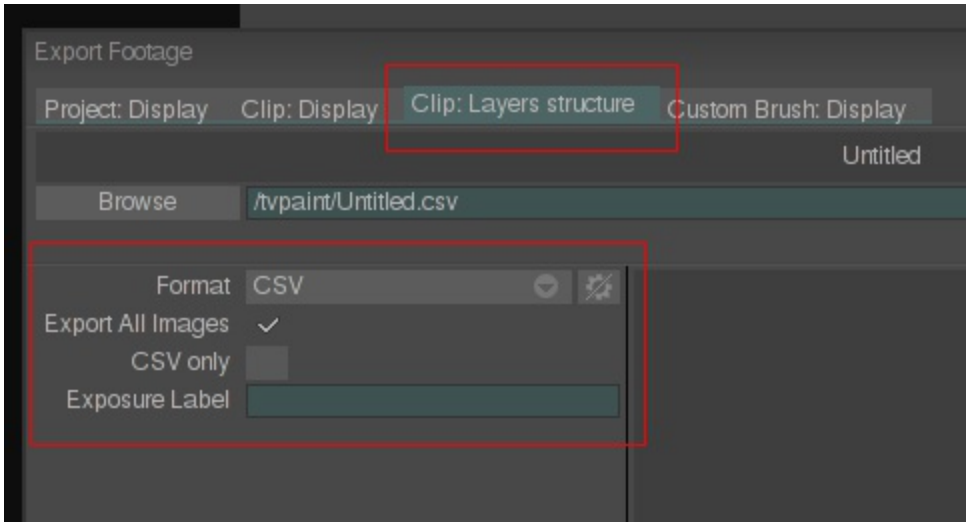
	A	B	C	D	E	F	G	H
1	UTF-8	TVPaint	CSV 1.0					
2	Project Name	Width	Height	Frame Count	Layer Count	Frame Rate	Pixel Aspect Ratio	Field Mode
3	Untitled	1920	1080	16	2	24.000000	1.000000	Progressive
4								
5	#Layers	Untitled 2	Untitled 1					
6	#Density	1.000000	1.000000					
7	#Blending	Color	Color					
8	#Visible	1	1					
9	#00000		frame001-00000.png					
10	#00001		frame001-00000.png					
11	#00002	frame000-00002.png	frame001-00000.png					
12	#00003	frame000-00002.png	frame001-00000.png					

A `.csv` file as a spreadsheet in **LibreOffice Calc**.

Krita can both export and import this format. It is recommended to use 8bit

sRGB color space because that's the only color space for **TVPaint**. Layer groups and layer masks are also not supported.

TVPaint can only export this format by itself. In **TVPaint 11**, use the *Export to…* option of the *File* menu, and on the upcoming *Export footage* window, use the *Clip: Layers structure* tab.



Exporting into `.csv` in TVPaint.

To import this format back into TVPaint there is a George language script extension. See the "Packs, Plugins, Third party" section on the TVPaint community forum for more details and also if you need support for other softwares. Moho/Anime Studio and Blender also have plugins to import this format.

See also

- CSV import script for TVPaint [https://forum.tvpaint.com/viewtopic.php?f=26&t=9759].
- CSV import script for Moho/Anime Studio [https://forum.tvpaint.com/viewtopic.php?f=26&t=10050].
- CSV import script for Blender [https://developer.blender.org/T47462].

*.exr

`.exr` is the prime file format for saving and loading floating point bit depths, and due to the library made to load and save these images being fully open source, the main interchange format as well.

Floating point bit-depths are used by the computer graphics industry to record scene referred values, which can be made via a camera or a computer renderer. Scene referred values means that the file can have values whiter than white, which in turn means that such a file can record lighting conditions, such as sunsets very accurately. These EXR files can then be used inside a renderer to create realistic lighting.

Krita can load and save EXR for the purpose of paint-over (yes, Krita can paint with scene referred values) and interchange with applications like Blender, Mari, Nuke and Natron.

*.gbr

The GIMP brush format. Krita can open, save and use these files as predefined brushes.

There's three things that you can decide upon when exporting a `*.gbr`:

Name
: This name is different from the file name, and will be shown inside Krita as the name of the brush.

Spacing
: This sets the default spacing.

Use color as mask
: This'll turn the darkest values of the image as the ones that paint, and the whitest as transparent. Untick this if you are using colored images for the brush.

GBR brushes are otherwise unremarkable, and limited to 8bit color precision.

*.gif

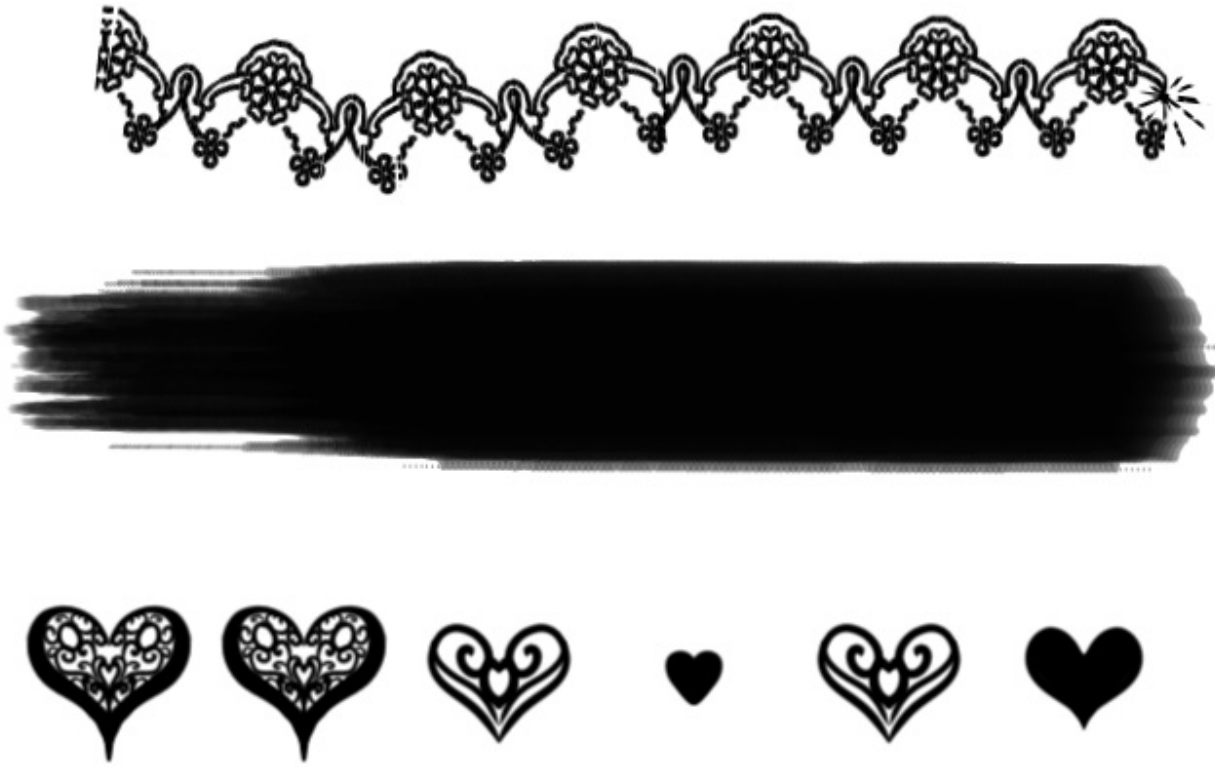
`.gif` is a file format mostly known for the fact that it can save animations. It's a fairly old format, and it does its compression by indexing the colors to a maximum of 256 colors per frame. Because we can technically design an image for 256 colors and are always able save over an edited GIF without any kind of extra degradation, this is a lossless compression technique.

This means that it can handle most grayscale images just fine and without losing any visible quality. But for color images that don't animate it might be better to use *.jpg or *.png.

*.gih

The GIMP image hose format. Krita can open and save these, as well as import via the predefined brush tab.

Image Hose means that this file format allows you to store multiple images and then set some options to make it specify how to output the multiple images.

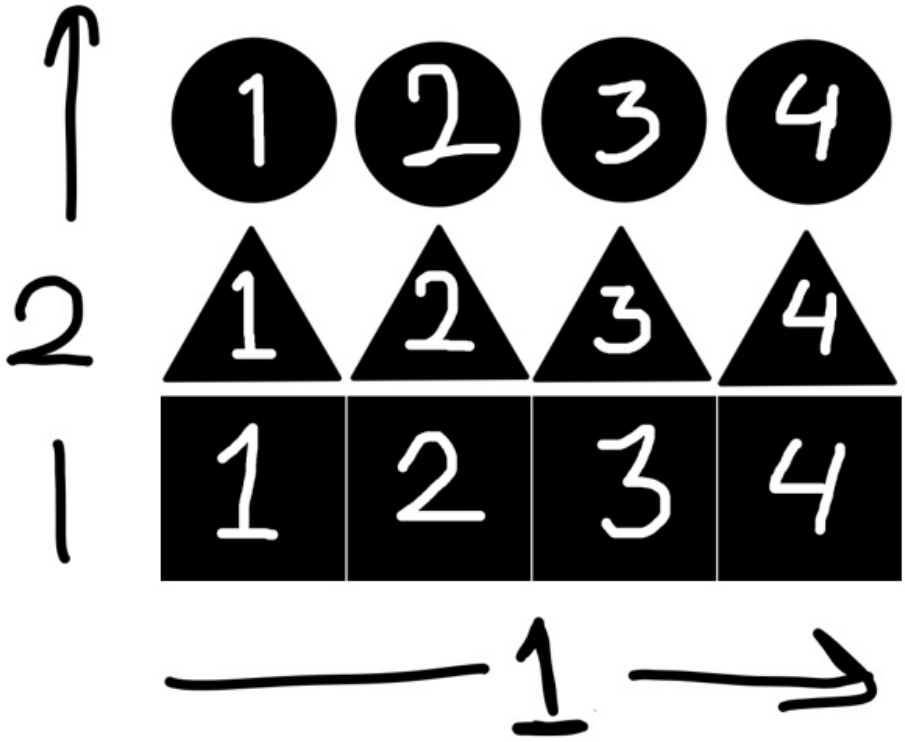


From top to bottom: Incremental, Pressure and Random

Dimension and ranks.

The GIMP image hose format allows multiple dimensions for a given brush. You could for example have a dimension that updates incrementally, and one that updates on pressure, or updates randomly. Upon export, Krita will use the ranks to subdivide the layers per dimension. If you have a 24 layer image and three ranks, and the first dimension is set to 2, the second to 4 and the third to 3, then Krita will divide 24 into 2 groups of 12, each of which have unique images for the 2 parts of the first dimension. Then those 2 groups of 12 get divided into 8 groups of 4, each of which have unique brush tips for the four parts of the second dimension, and finally, the grouped three images have each a unique brush for the final dimension.

So, the following image has a table where dimension 1 is unique in one of 4 numbers, while dimension 2 is unique in one of 3 shapes. So our ranks for dimension 1 and dimension 2 need to be 4 and 3 respectively. Now, to order the layers, you need to subdivide the table first by the first dimension, and then by the second. So we end up with three layers each for a shape in the second dimension but for the first number, then another three layers, each for a shape, but then for the second number, and so forth.



See [the GIMP documentation](https://docs.gimp.org/2.8/en/gimp-using-animated-brushes.html) for a more thorough explanation.

GIMP image hose format options:

Constant
: This'll use the first image, no matter what.

Incremental
: This'll paint the image layers in sequence. This is good for images that can be strung together to create a pattern.

Pressure
: This'll paint the images depending on pressure. This is good for brushes imitating the hairs of a natural brush.

Random
: This'll draw the images randomly. This is good for image-collections used in speedpainting as well as images that generate texture. Or perhaps more graphical symbols.

Angle
: This'll use the dragging angle to determine with image to draw.

When exporting a Krita file as a `.gih`, you will also get the option to set the default spacing, the option to set the name (very important for looking it up in the UI) and the ability to choose whether or not to generate the mask from the colors.

Use Color as Mask
: This'll turn the darkest values of the image as the ones that paint, and the whitest as transparent. Untick this if you are using colored images for the brush.

We have a Krita Brush tip page on how to create your own GIH brush.

*.jpg

`.jpg`, `.jpeg` or `.jpeg2000` are a family of file-formats designed to encode photographs.

Photographs have the problem that they have a lot of little gradients, which means that you cannot index the file like you can with *.gif and expect the result to look good. What JPEG instead does is that it converts the file to a perceptual color space (YCrCb), and then compresses the channels that encode the colors, while keeping the channel that holds information about the relative lightness uncompressed. This works really well because human eye-sight is not as sensitive to colorfulness as it is to relative lightness. JPEG also uses other lossy compression techniques, like using cosine waves to describe image contrasts.

However, it does mean that JPEG should be used in certain cases. For images with a lot of gradients, like full scale paintings, JPEG performs better than *.png and *.gif.

But for images with a lot of sharp contrasts, like text and comic book styles, PNG is a much better choice despite a larger file size. For grayscale images, *.png and *.gif will definitely be more efficient.

Because JPEG uses lossy compression, it is not advised to save over the same JPEG multiple times. The lossy compression will cause the file to reduce in quality each time you save it. This is a fundamental problem with lossy compression methods. Instead use a lossless file format, or a working file format while you are working on the image.

*.kpl

Since 4.0, Krita has a new palette file-format that can handle colors that are wide gamut, RGB, CMYK, XYZ, GRAY, or LAB, and can be of any of the available bitdepths, as well as groups. These are Krita Palettes, or `*.kpl`.

`*.kpl` files are ZIP files, with two XMLs and ICC profiles inside. The *colorset.xml* file contains the swatches as ColorSetEntry and Groups as Group. The *profiles.xml* file contains a list of profiles, and the ICC profiles themselves are embedded to ensure compatibility over different computers.

*.kra

`.kra` is Krita's internal file-format, which means that it is the file format that saves all of the features Krita can handle. It's construction is vaguely based on the open document standard, which means that you can rename your `.kra` file to a `.zip` file and open it up to look at the insides.

It is a format that you can expect to get very heavy, and isn't meant for sharing on the internet.

*.ora

`.ora`, or the Open Raster format, is an interchange format. It was designed to replace *.psd as an interchange format, as the latter isn't meant for that. Like *.kra it is loosely based on the Open Document structure, thus a ZIP file with a bunch of XMLs and PNGs, but where Krita's internal file format can sometimes have fully binary chunks, `.ora` saves its layers as *.png making it fully open and easy to support.

As an interchange format, it can be expected to be heavy and isn't meant for uploading to the internet.

See also

Open Raster Specification [https://www.openraster.org/]

*.pbm, *.pgm and *.ppm

`.pbm`, `.pgm` and `.ppm` are a series of file-formats with a similar logic to them. They are designed to save images in a way that the result can be read as an ASCII file, from back when email clients couldn’t read images reliably.

They are very old file formats, and not used outside of very specialized usecases, such as embedding images inside code.

.pbm
: One-bit and can only show strict black and white.

.pgm
: Can show 255 values of gray (8bit).

.ppm
: Can show 8bit rgb values.

*.pdf

`.pdf` is a format intended for making sure a document looks the same on all computers. It became popular because it allows the creator to make sure that the document looks the same and cannot be changed by viewers. These days it is an open standard and there is quite a variety of programs that can read and save PDFs.

Krita can open PDFs with multiple layers. There is currently no PDF export, nor is that planned. If you want to create a PDF with images from Krita, use Scribus [https://www.scribus.net/].

While PDFs can be viewed via most browsers, they can also become very heavy and are thus not recommended outside of official documents. Printhouses will often accept PDF.

*.png

.png, or Portable Network Graphics, is a modern alternative to *.gif and with that and *.jpg it makes up the three main formats that are widely supported on the internet.

PNG is a lossless file format, which means that it is able to maintain all the colors of your image perfectly. It does so at the cost of the file size being big, and therefore it is recommended to try *.jpg for images with a lot of gradients and different colors. Grayscale images will do better in PNG as well as images with a lot of text and sharp contrasts, like comics.

Like *.gif, PNG can support indexed color. Unlike *.gif, PNG doesn't support animation. There have been two attempts at giving animation support to PNG, APNG and MNG, the former is unofficial and the latter too complicated, so neither have really taken off yet.

New in version 4.2: Since 4.2 we support saving HDR to PNG as according to the W3C PQ HDR PNG standard [https://www.w3.org/TR/png-hdr-pq/]. To save as such files, toggle *Save as HDR image (Rec. 2020 PQ)*, which will convert your image to the Rec 2020 PQ color space and then save it as a special HDR PNG.

*.psd

`.psd` is Photoshop's internal file format. For some reason, people like to use it as an interchange format, even though it is not designed for this.

`.psd`, unlike actual interchange formats like *.pdf, *.tiff, *.exr, *.ora and *.svg doesn't have an official spec online. Which means that it needs to be reverse engineered. Furthermore, as an internal file format, it doesn't have much of a philosophy to its structure, as it's only purpose is to save what Photoshop is busy with, or rather, what all the past versions of Photoshop have been busy with. This means that the inside of a PSD looks somewhat like Photoshop's virtual brains, and PSD is in general a very disliked file-format.

Due to `.psd` being used as an interchange format, this leads to confusion amongst people using these programs, as to why not all programs support opening these. Sometimes, you might even see users saying that a certain program is terrible because it doesn't support opening PSDs properly. But as PSD is an internal file-format without online specs, it is impossible to have any program outside it support it 100%.

Krita supports loading and saving raster layers, blending modes, layerstyles, layer groups, and transparency masks from PSD. It will likely never support vector and text layers, as these are just too difficult to program properly.

We recommend using any other file format instead of PSD if possible, with a strong preference towards *.ora or *.tiff.

As a working file format, PSDs can be expected to become very heavy and most websites won't accept them.

*.svg

`.svg`, or Scalable Vector Graphics, is the most modern vector graphics interchange file format out there.

Being vector graphics, SVG is very light weight. This is because it usually only stores coordinates and parameters for the maths involved with vector graphics.

It is maintained by the W3C SVG working group, who also maintain other open standards that make up our modern internet.

While you can open up SVG files with any text-editor to edit them, it is best to use a vector program like Inkscape. Krita 2.9 to 3.3 supports importing SVG via the add shape docker. Since Krita 4.0, SVGs can be properly imported, and you can export singlevector layers via *Layer* ▸ *Import/Export* ▸ *Save Vector Layer as SVG…* menu item. For 4.0, Krita will also use SVG to save vector data into its internal format.

SVG is designed for the internet, though sadly, because vector graphics are considered a bit obscure compared to raster graphics, not a lot of websites accept them yet. Hosting them on your own webhost works just fine though.

*.tiff

`.tiff`, or Tagged Image File Format, is a raster interchange format that was originally designed to be a common format generated by scanners and used by printers.

It can support multiple color spaces, and even layers. However, the latter is a bit odd, as the official specs, owned by Adobe, have a different way of saving layers to TIFF than Photoshop, also owned by Adobe.

As an interchange format, `.tiff` is not meant for sharing on the internet, and you will not find many websites that do accept it. However, printhouses know the file format, and will likely accept it.

Lossy and Lossless Image Compression

When we compress a file, we do this because we want to temporarily make it smaller (like for sending over email), or we want to permanently make it smaller (like for showing images on the internet).

Lossless compression techniques are for when we want to *temporarily* reduce information. As the name implies, they compress without losing information. In text, the use of abbreviations is a good example of a lossless compression technique. Everyone knows ‘etc.’ expands to ‘etcetera’, meaning that you can half the 8 character long ‘etcetera’ to the four character long ‘etc.’.

Within image formats, examples of such compression is by for example ‘indexed’ color, where we make a list of available colors in an image, and then assign a single number to them. Then, when describing the pixels, we only write down said number, so that we don’t need to write the color definition over and over.

Lossy compression techniques are for when we want to *permanently* reduce the file size of an image. This is necessary for final products where having a small filesize is preferable such as a website. That the image will not be edited anymore after this allows for the use of the context of a pixel to be taken into account when compressing, meaning that we can rely on psychological and statistical tricks.

One of the primary things JPEG for example does is chroma sub-sampling, that is, to split up the image into a grayscale and two color versions (one containing all red-green contrast and the other containing all blue-yellow contrast), and then it makes the latter two versions smaller. This works because humans are much more sensitive to differences in lightness than we are to differences in hue and saturation.

Another thing it does is to use cosine waves to describe contrasts in an image.

What this means is that JPEG and other lossy formats using this are *very good at describing gradients, but not very good at describing sharp contrasts*.

Conversely, lossless image compression techniques are *really good at describing images with few colors thus sharp contrasts, but are not good to compress images with a lot of gradients*.

Another big difference between lossy and lossless images is that lossy file formats will degrade if you re-encode them, that is, if you load a JPEG into Krita edit a little, resave, edit a little, resave, each subsequent save will lose some data. This is a fundamental part of lossy image compression, and the primary reason we use working files.

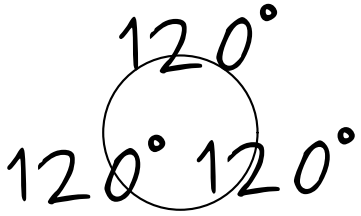
See also

If you're interested in different compression techniques, Wikipedia's page(s) on image compression [https://en.wikipedia.org/wiki/Image_compression] are very good, if not a little technical.

Perspective Projection

The Perspective Projection tutorial is one of the Kickstarter 2015 tutorial rewards. It's about something that humanity has known scientifically for a very long time, and decent formal training will teach you about this. But I think there are very very few tutorials about it in regard to how to achieve it in digital painting programs, let alone open source.

The tutorial is a bit image heavy, and technical, but I hope the skill it teaches will be really useful to anyone trying to get a grasp on a complicated pose. Enjoy, and don't forget to thank Raghukamath [https://www.raghukamath.com/] for choosing this topic!



Isometric

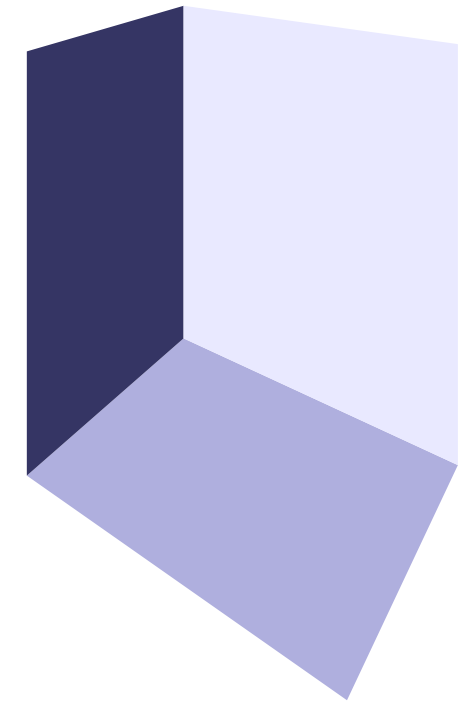
Parts:

So let’s start with the basics…

Orthographic

Despite the fancy name, you probably know what orthographic is. It is a schematic representation of an object, draw undeformed. Like the following example:

This is a rectangle. We have a front, top and side view. Put into perspective it should look somewhat like this:



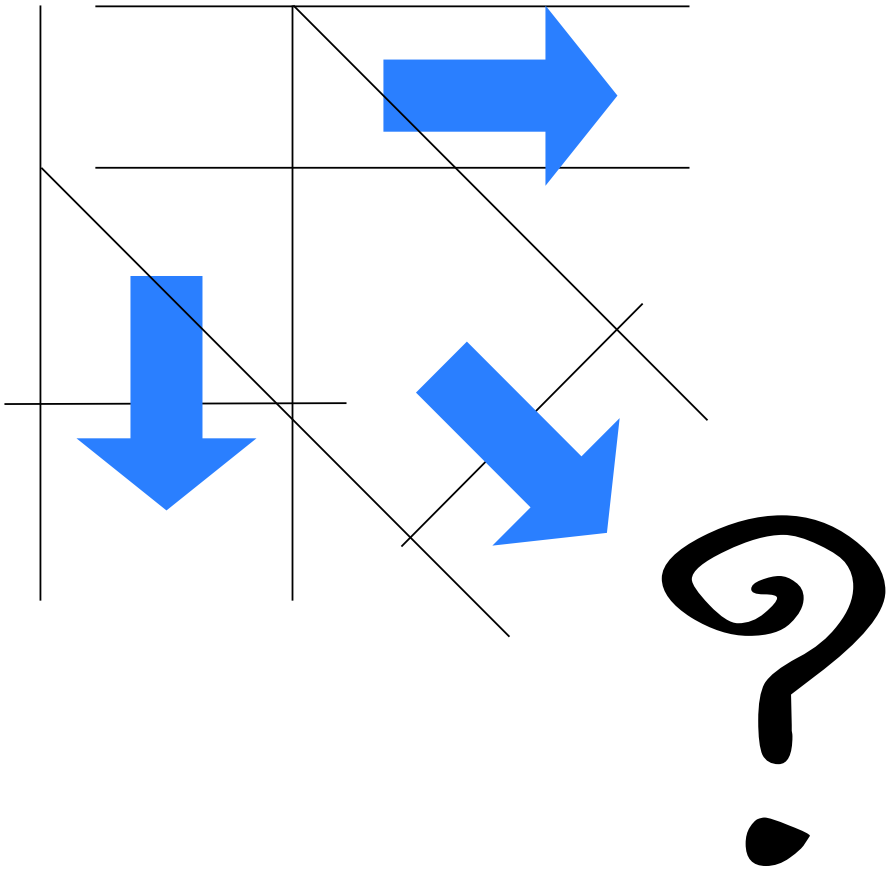
Perspective

While orthographic representations are kinda boring, they're also a good basis to start with when you find yourself in trouble with a pose. But we'll get to that in a bit.

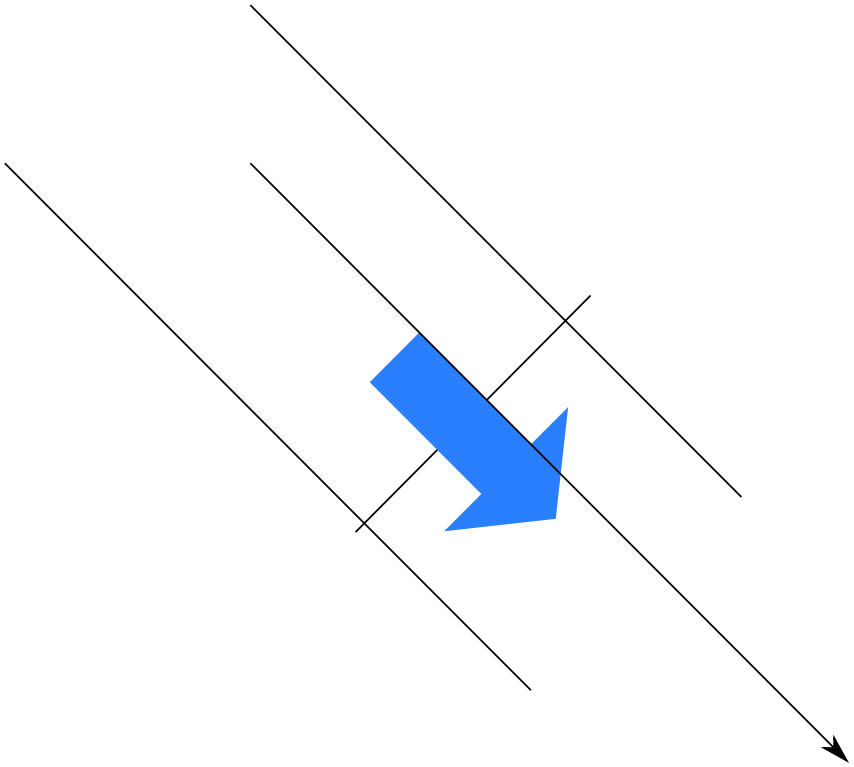
Oblique

So, if we can say that the front view is the viewer looking at the front, and the

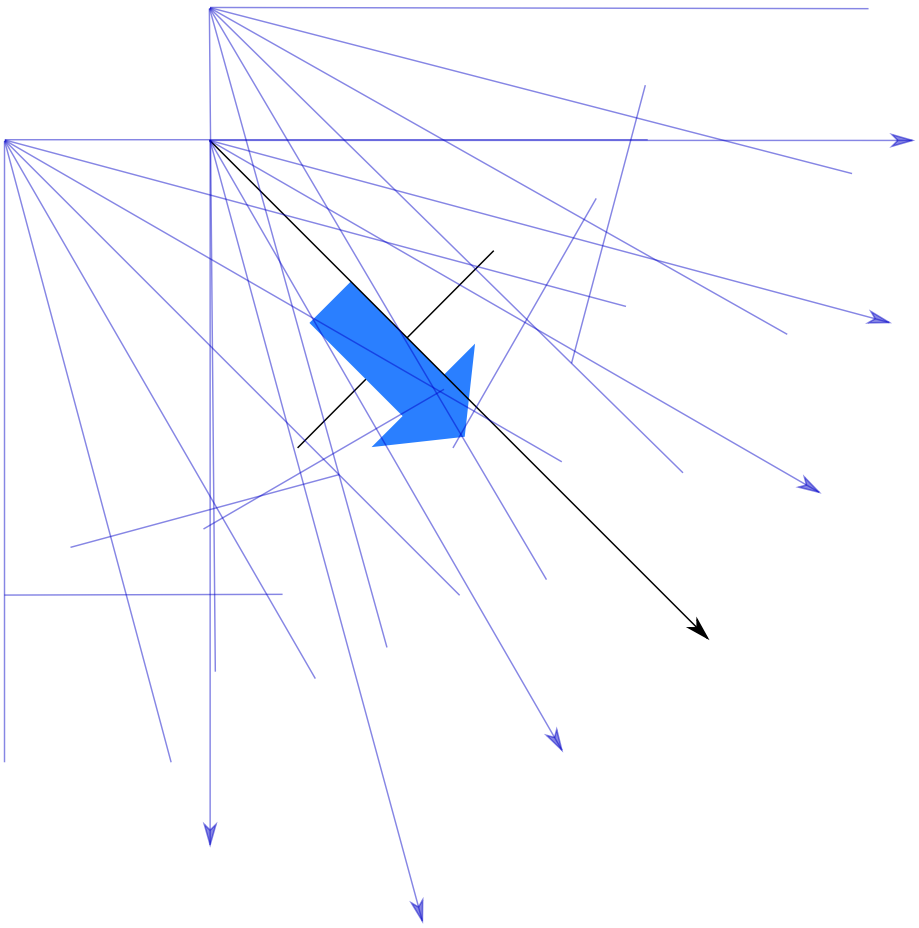
side view is the viewer directly looking at the side. (The perpendicular line being the view plane it is projected on)



Then we can get a half-way view from looking from an angle, no?



If we do that for a lot of different sides…

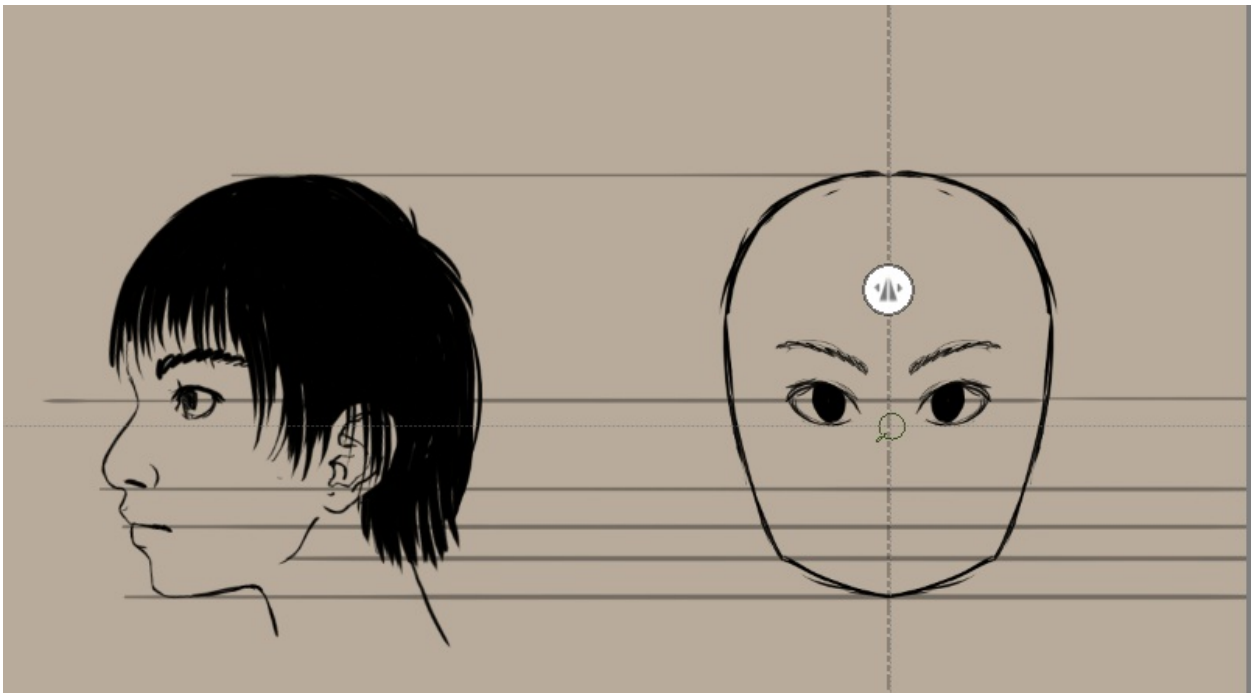


And we line up the sides we get a…

Rotation animation! :D

But cubes are boring. I am suspecting that projection is so ignored because no tutorial applies it to an object where you actually might NEED projection. Like a face.

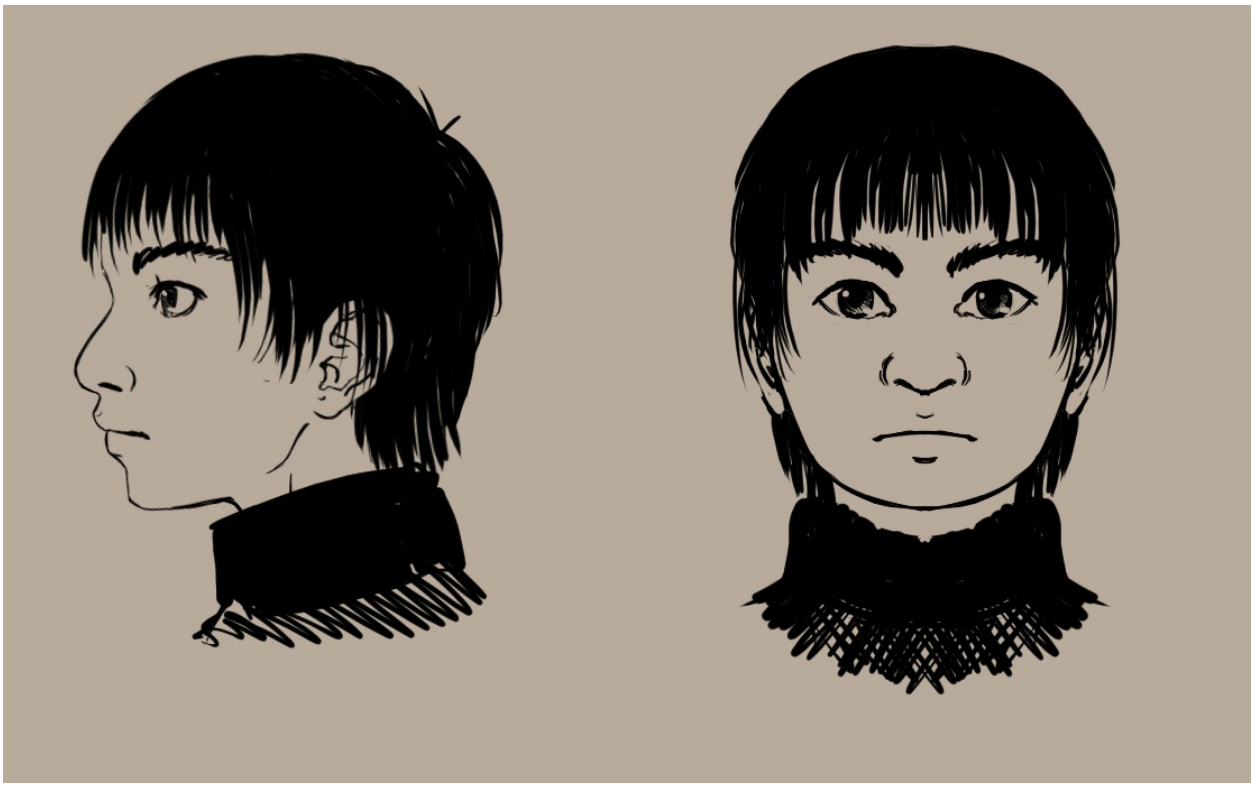
First, let's prepare our front and side views:



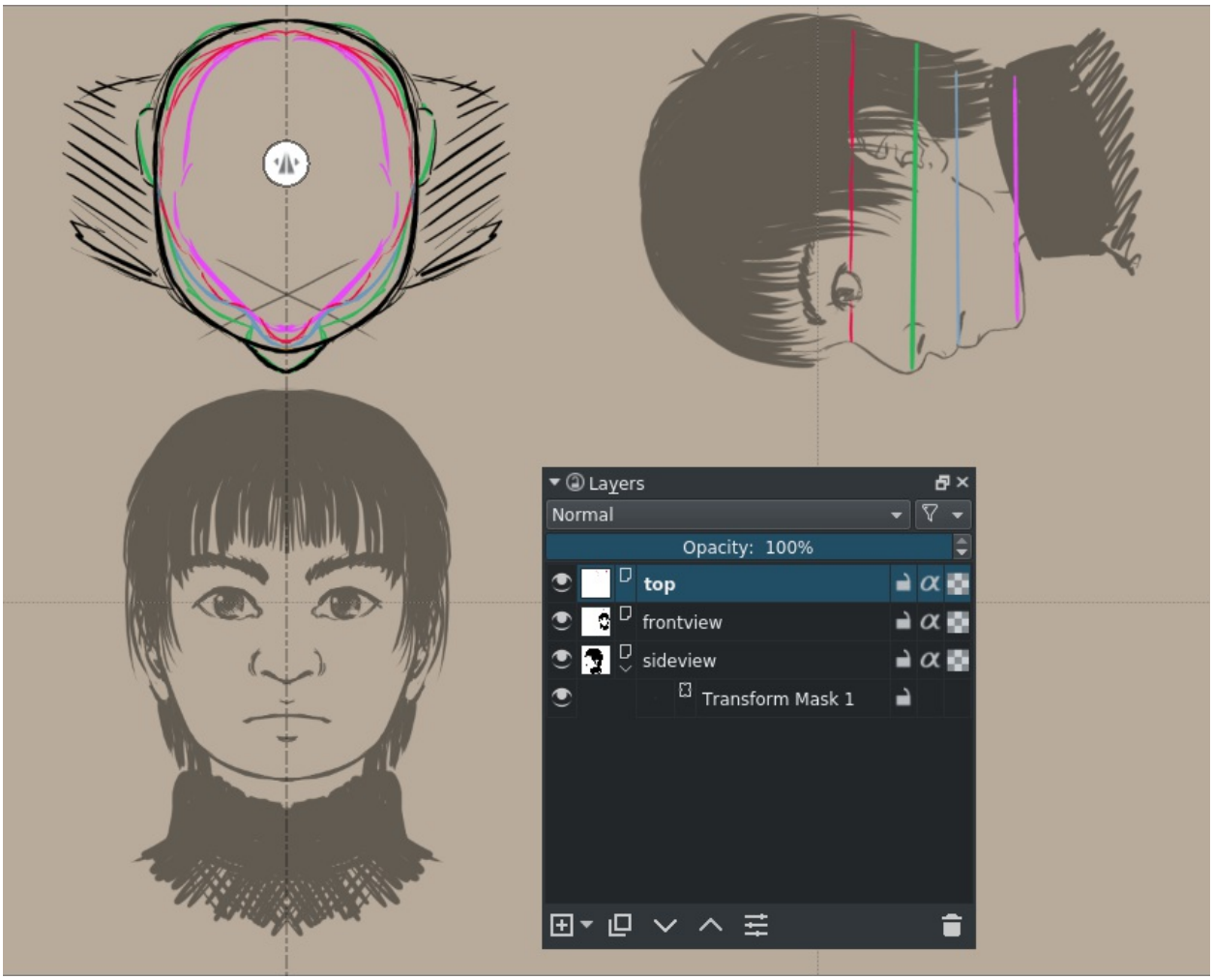
I always start with the side, and then extrapolate the front view from it. Because you are using Krita, set up two parallel rulers, one vertical and the other horizontal. To snap them perfectly, drag one of the nodes after you have made the ruler, and press the `Shift` key to snap it horizontal or vertical. In 3.0, you can also snap them to the image borders if you have *Snap Image Bounds* active via the `Shift + S` shortcut.

Then, by moving the mirror to the left, you can design a front view from the side view, while the parallel preview line helps you with aligning the eyes (which in the above screenshot are too low).

Eventually, you should have something like this:



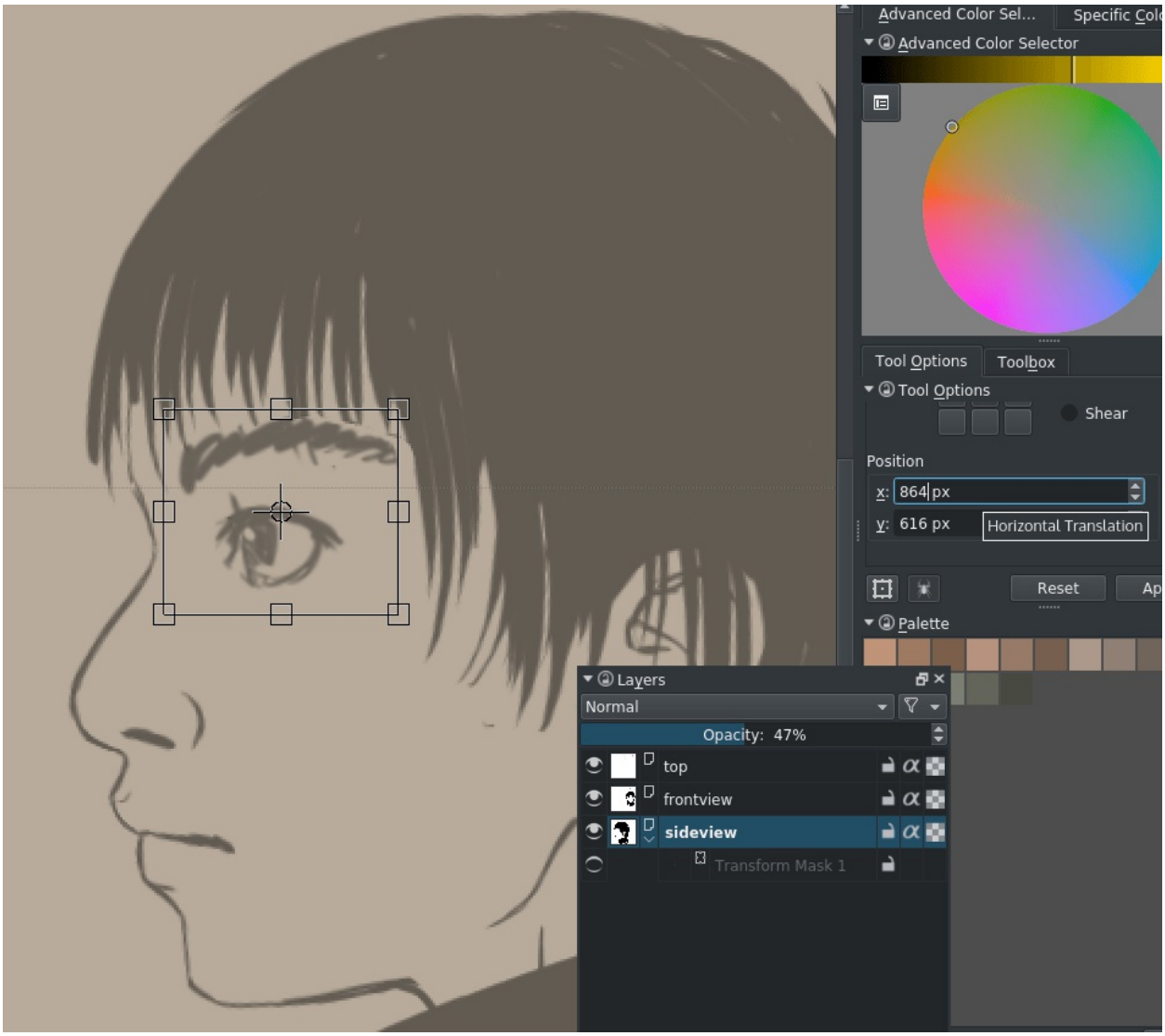
And of course, let us not forget the top, it’s pretty important:



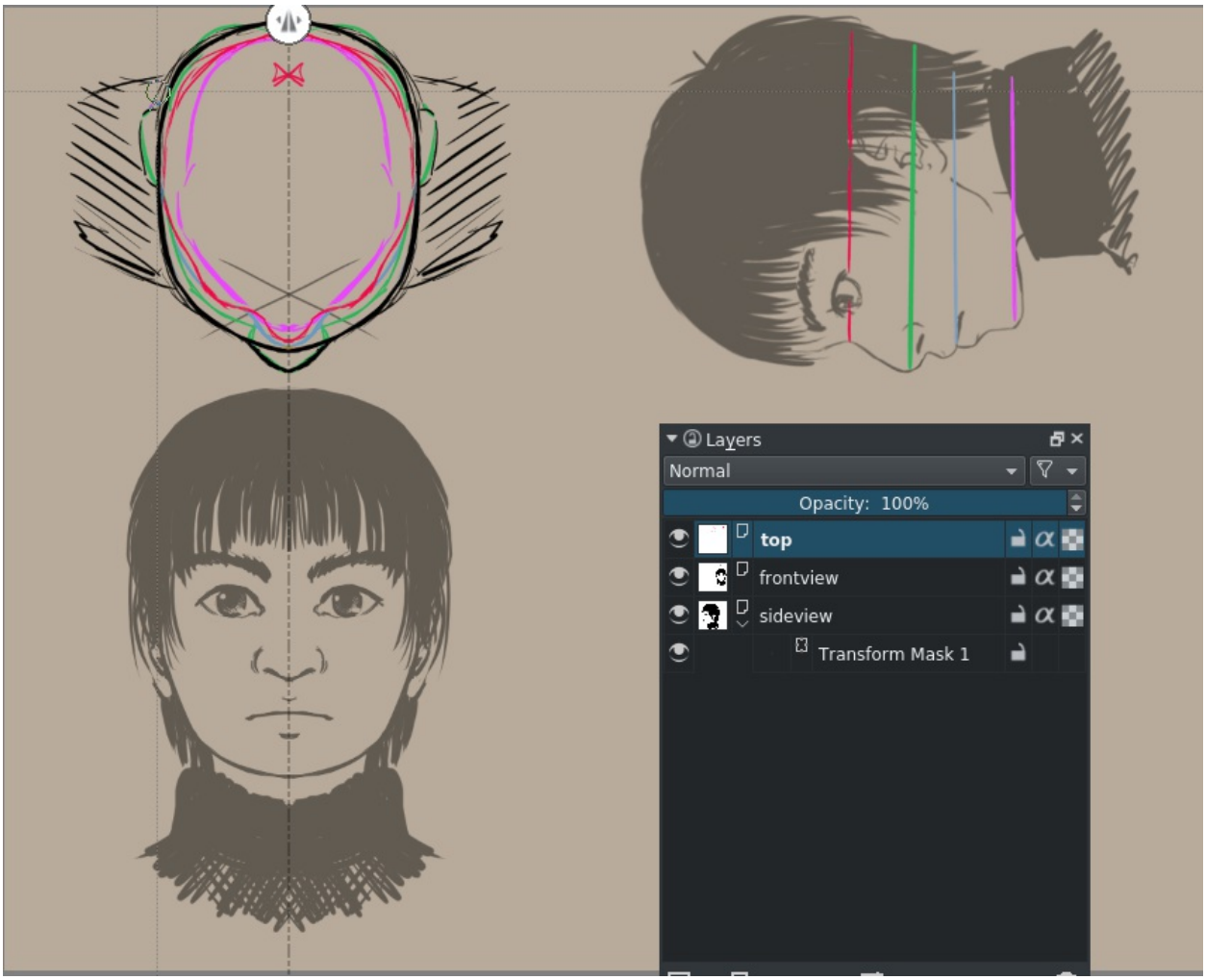
Tip

When you are using Krita, you can just use transform masks to rotate the side view for drawing the top view.

The top view works as a method for debugging your orthos as well. If we take the red line to figure out the orthographics from, we see that our eyes are obviously too inset. Let’s move them a bit more forward, to around the nose.

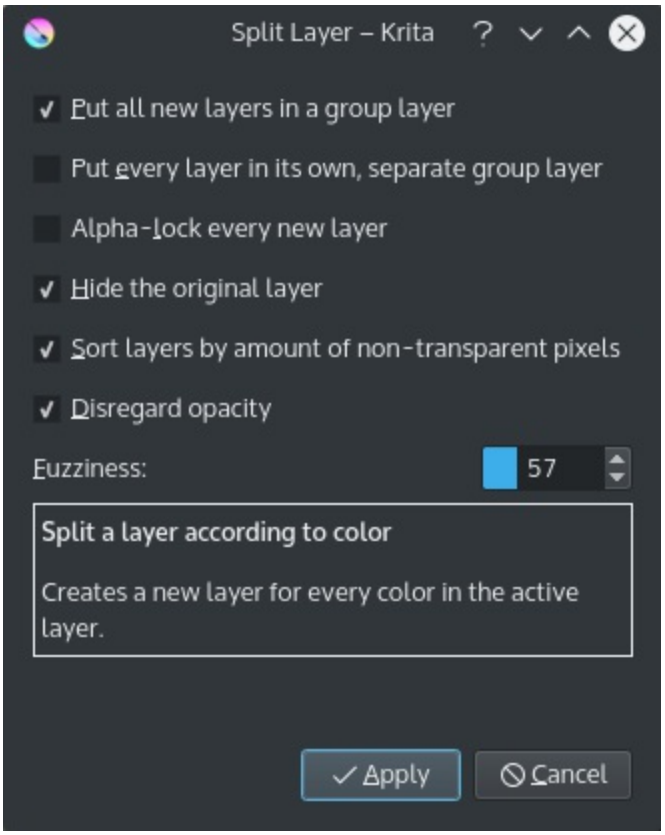


If you want to do precision position moving in the tool options docker, just select ‘position’ and the input box for the X. Pressing down then moves the transformed selection left. With Krita 3.0 you can just use the move tool for this and the arrow keys. Using transform here can be more convenient if you also have to squash and stretch an eye.

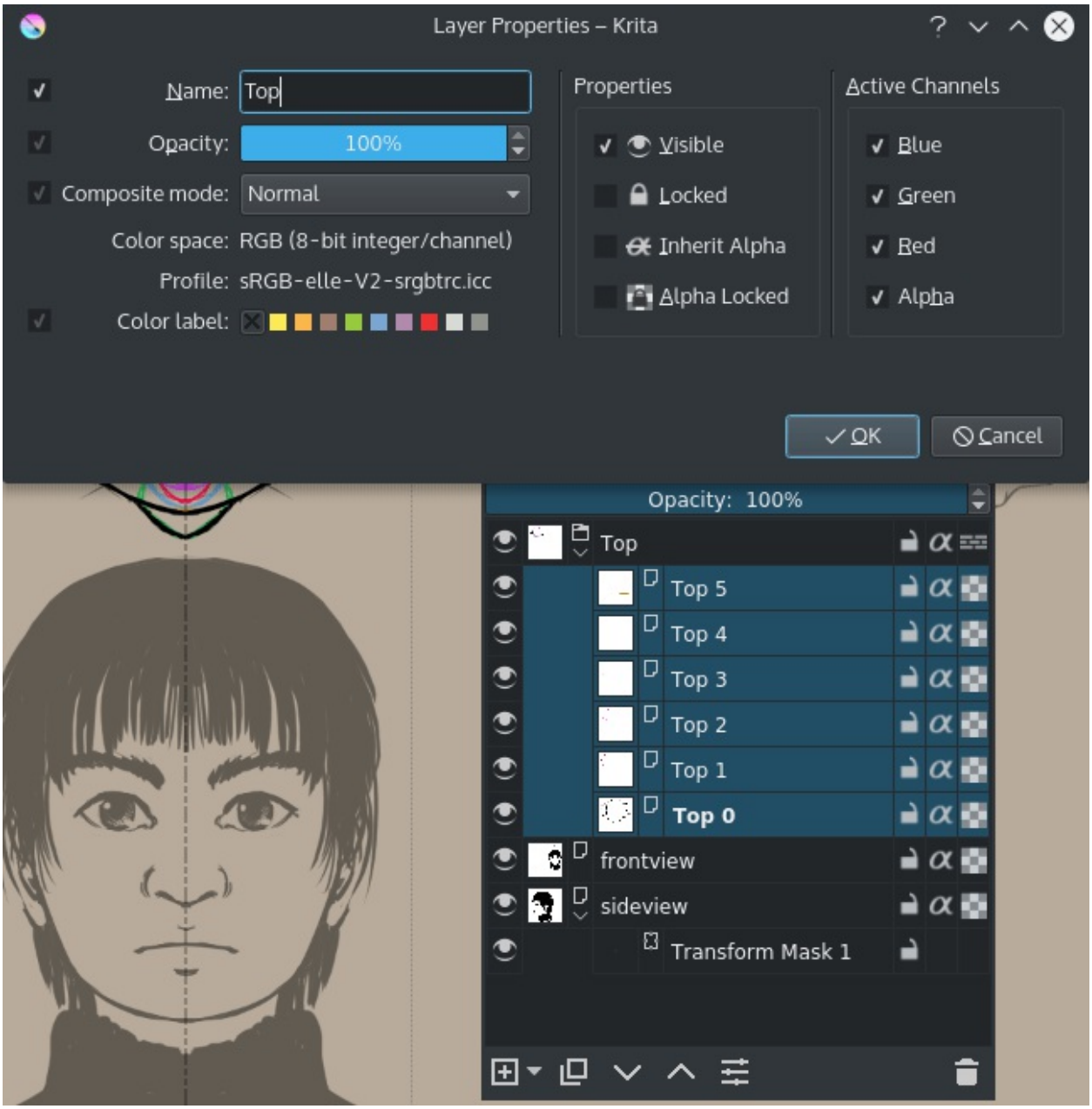


We fix the top view now. Much better.

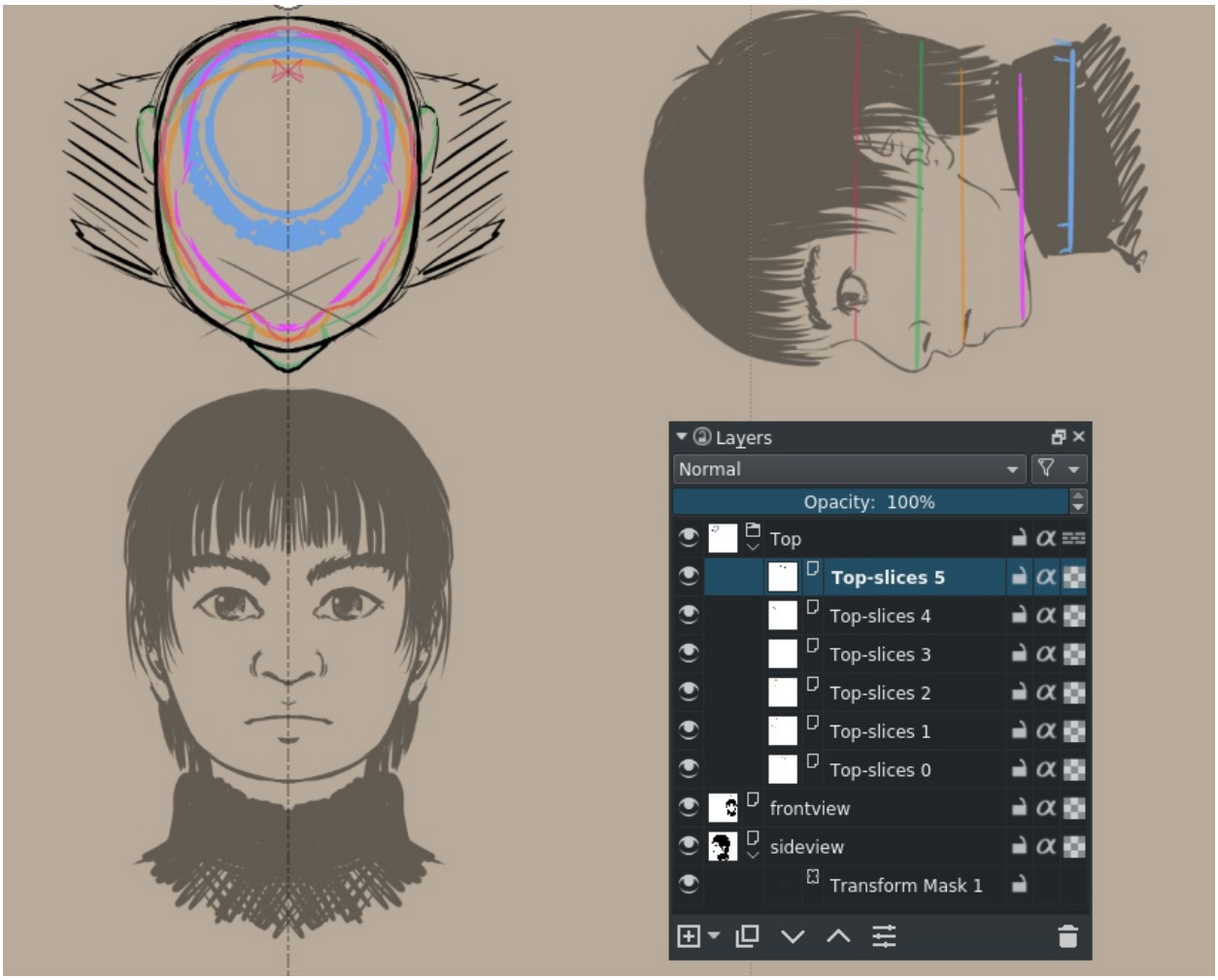
For faces, the multiple slices are actually pretty important. So important even, that I have decided we should have these slices on separate layers.
Thankfully, I chose to color them, so all we need to do is go to *Layer* ▸ *Split Layer*.



This’ll give you a few awkwardly named layers… rename them by selecting all and mass changing the name in the properties editor:

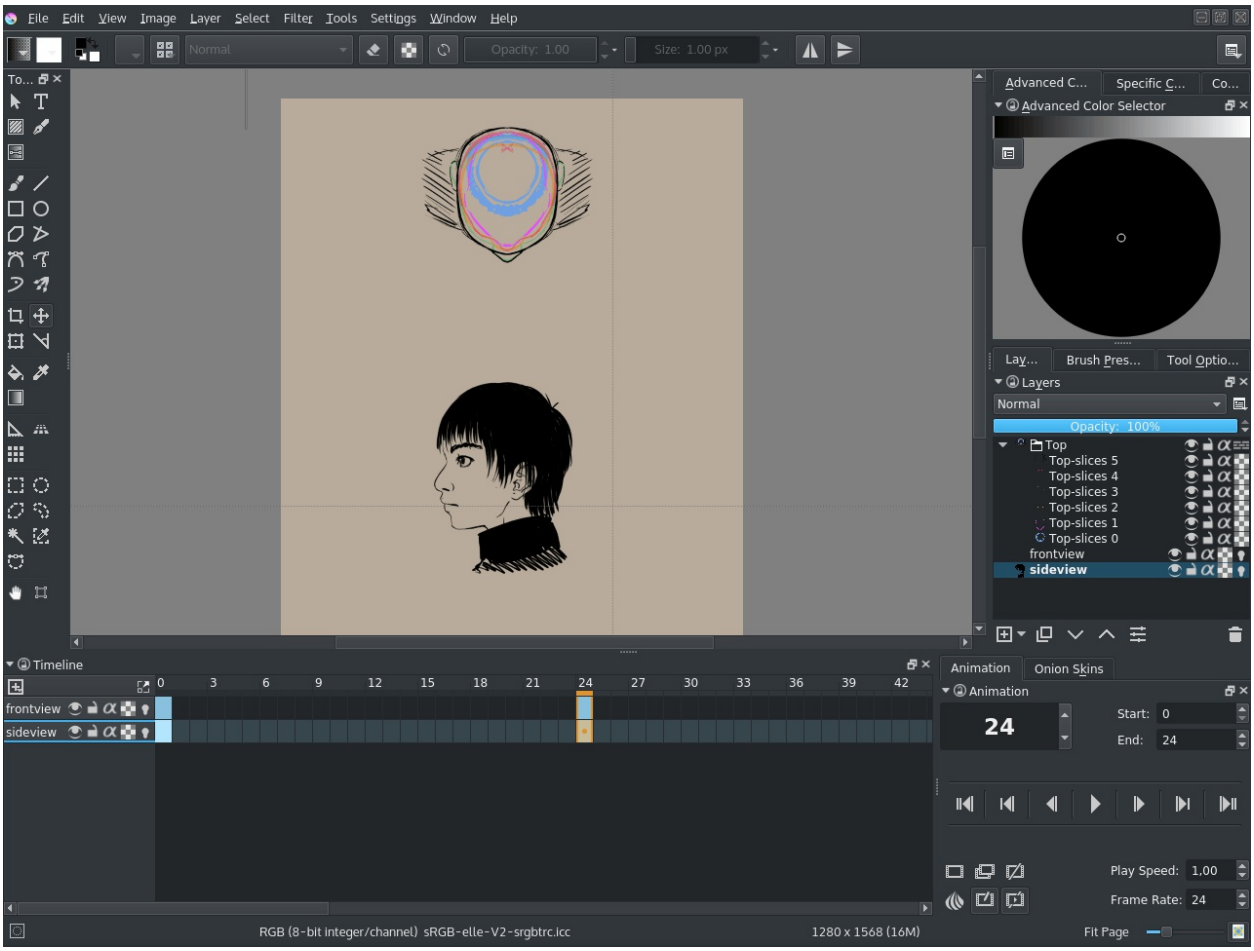


So, after some cleanup, we should have the following:

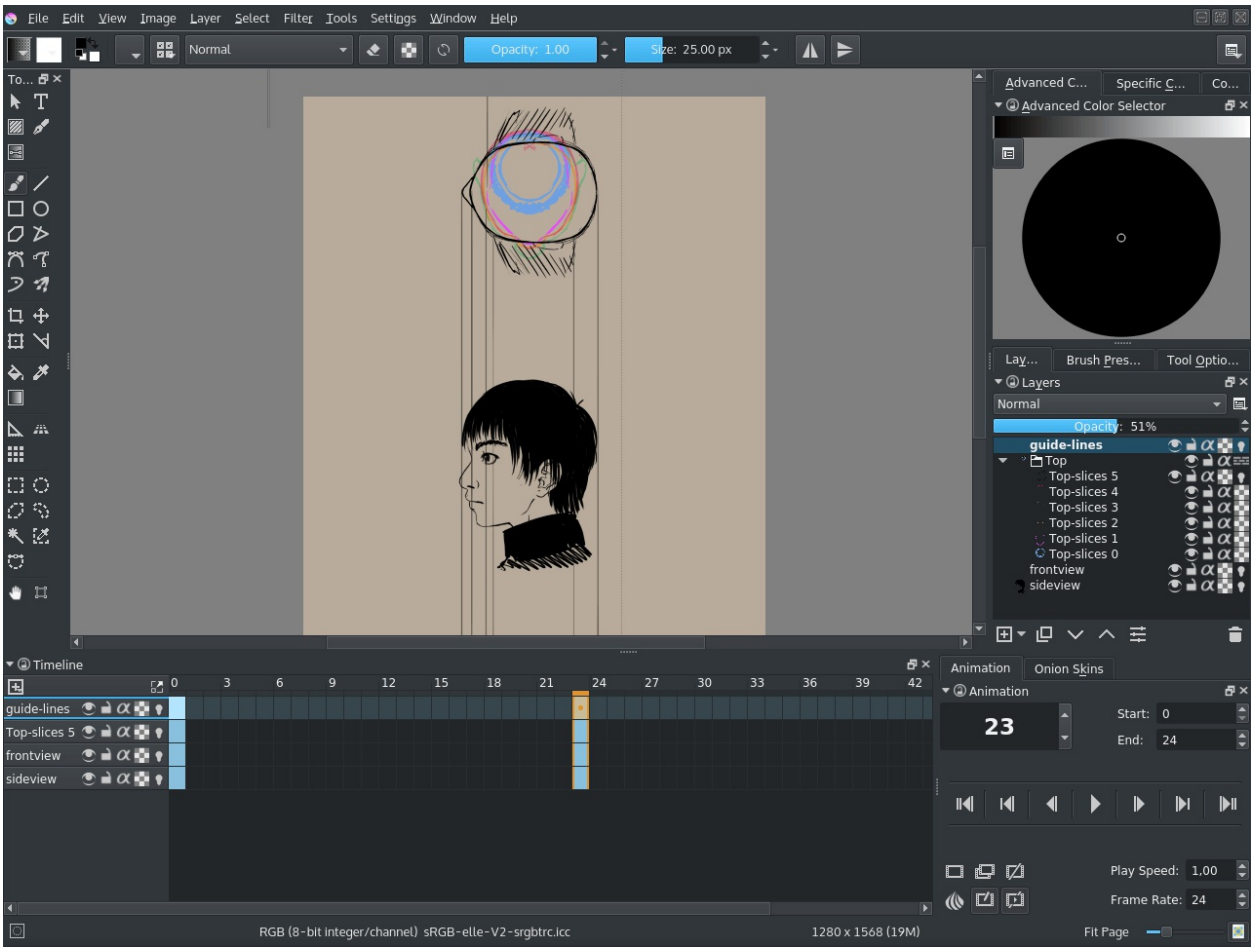


Okay, now we’re gonna use animation for the next bit.

Set it up as follows:

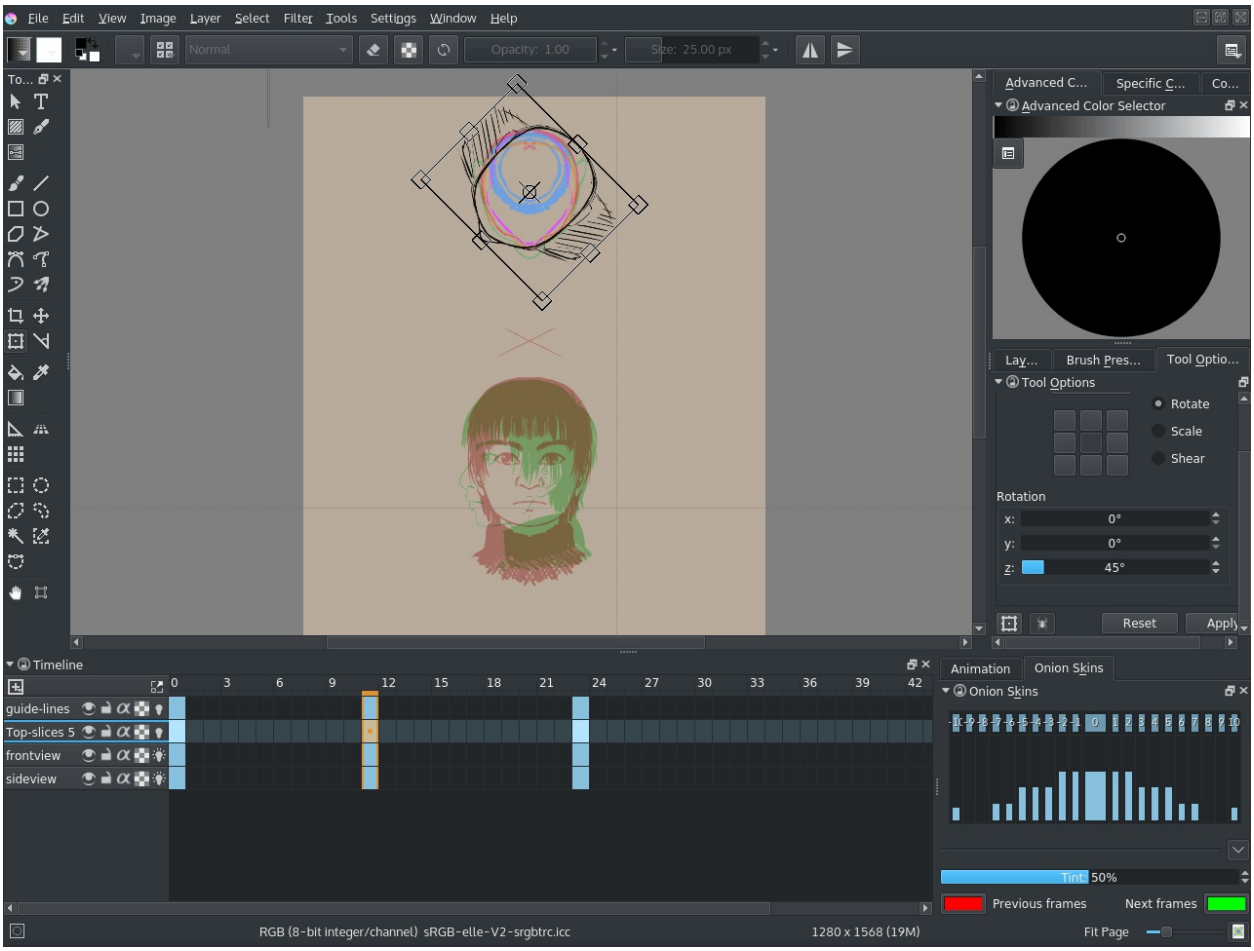


- Both front view and side view are set up as ‘visible in timeline’ so we can always see them.
- Front view has its visible frame on frame 0 and an empty frame on frame 23.
- Side view has its visible frame on frame 23 and an empty view on frame 0.
- The end of the animation is set to 23.

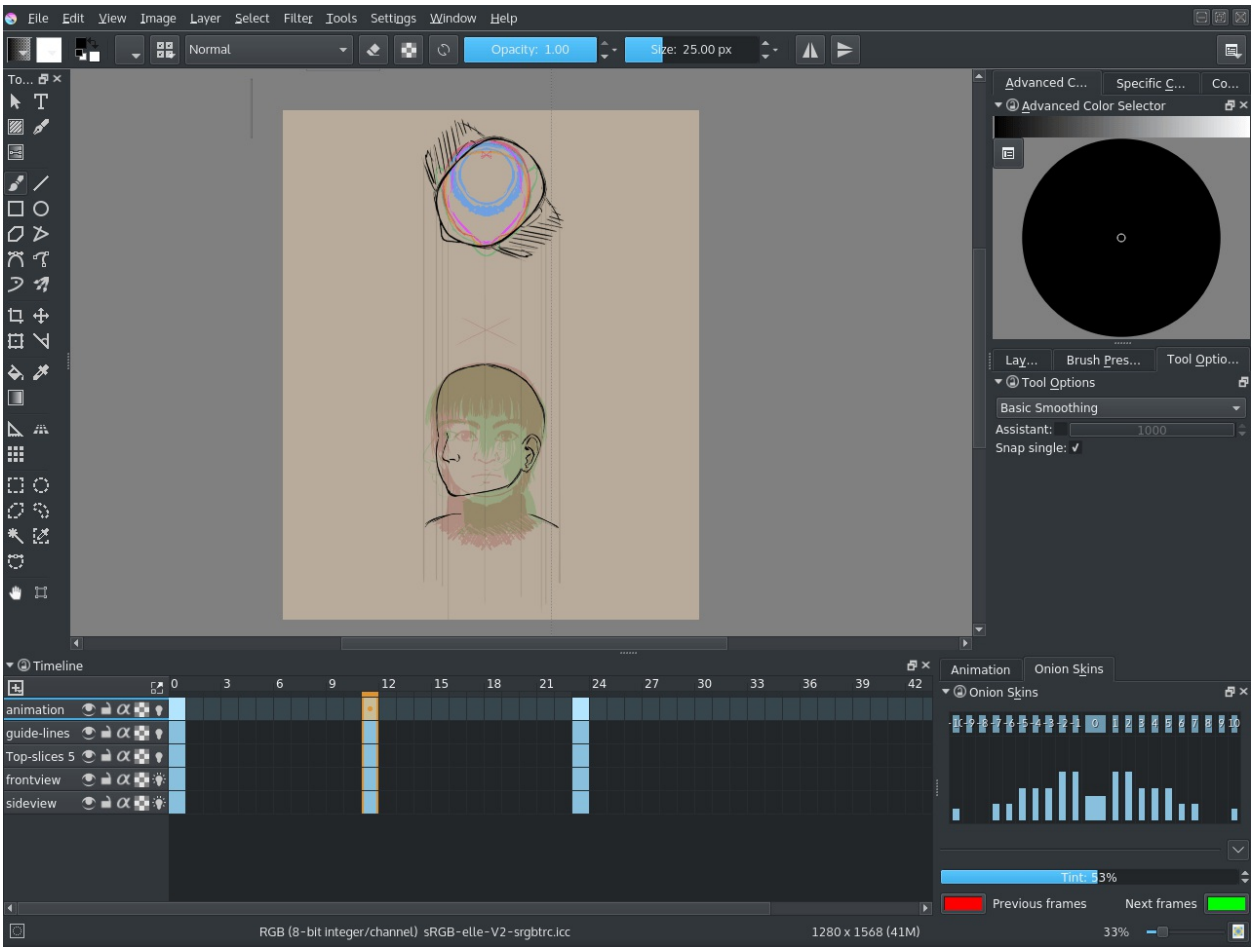


Krita can’t animate a transformation on multiple layers on multiple frames yet, so let’s just only transform the top layer. Add a semi-transparent layer where we draw the guidelines.

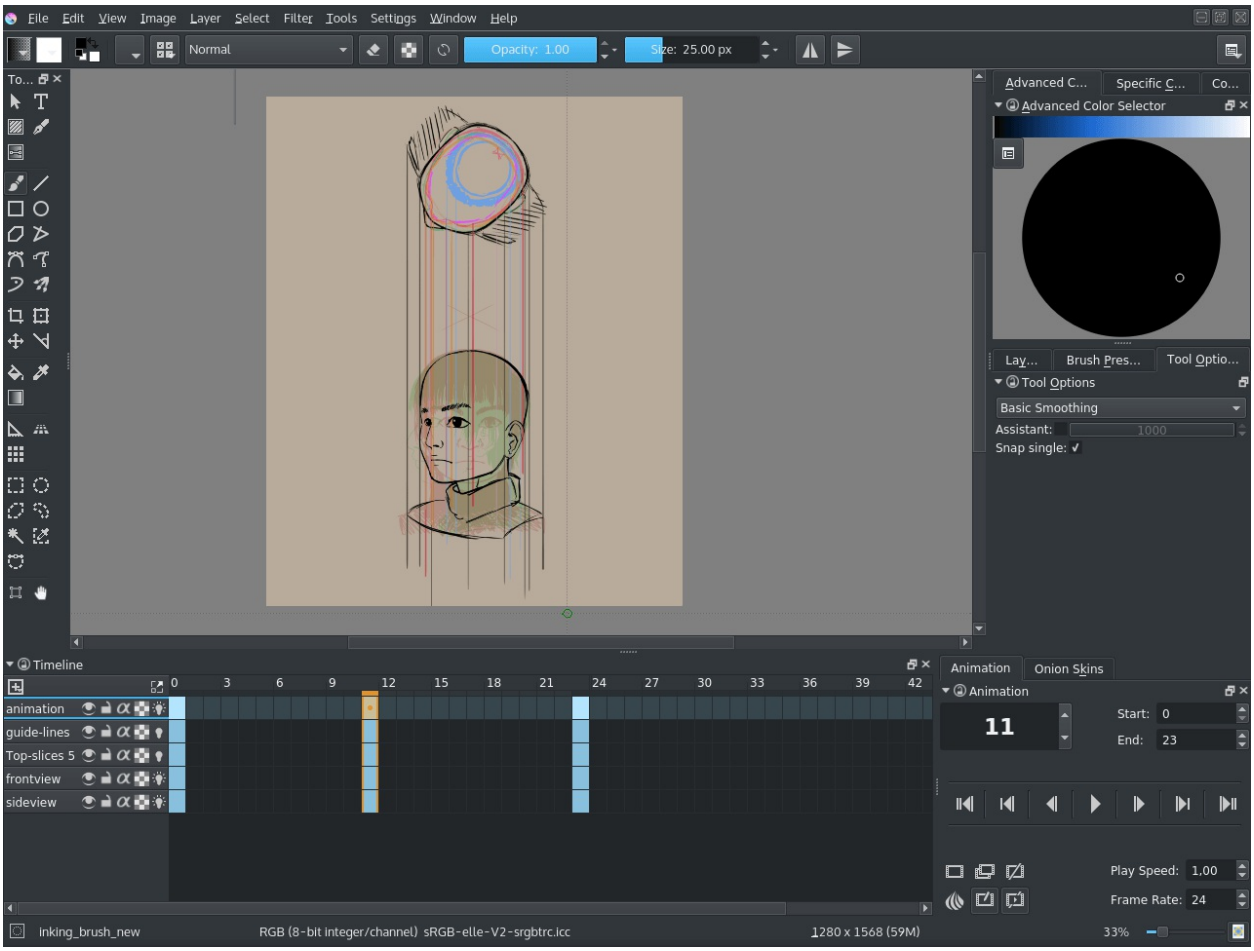
Now, select frame 11 (halfway), add new frames from front view, side view and the guidelines. And turn on the onion skin by toggling the lamp symbols. We copy the frame for the top view and use the transform tool to rotate it 45°.



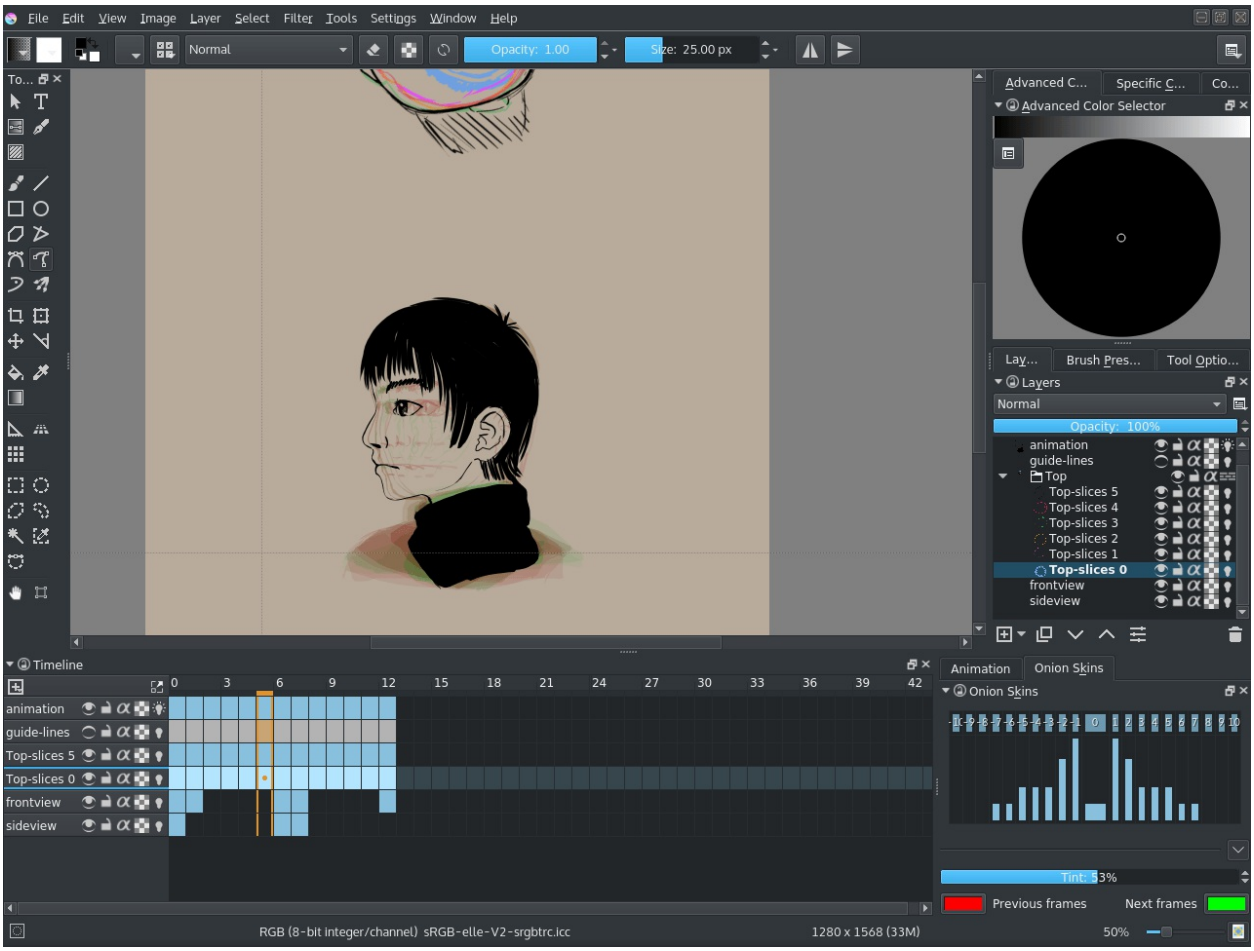
So, we draw our vertical guides again and determine a in-between…



This is about how far you can get with only the main slice, so rotate the rest as well.



And just like with the cube, we do this for all slices…



Eventually, if you have the top slices rotate every frame with 15°, you should be able to make a turn table, like this:

Because our boy here is fully symmetrical, you can just animate one side and flip the frames for the other half.

While it is not necessary to follow all the steps in the theory section to

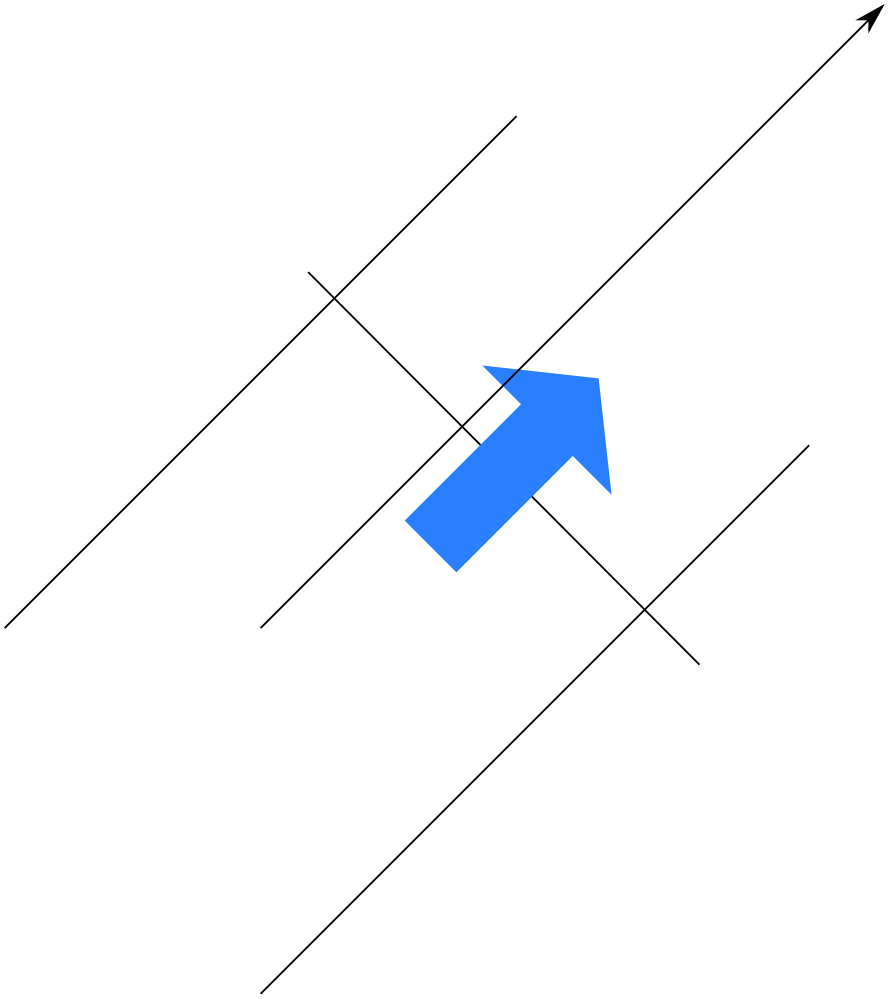
understand the tutorial, I do recommend making a turn table sometime. It teaches you a lot about drawing 3/4th faces.

How about… we introduce the top view into the drawing itself?

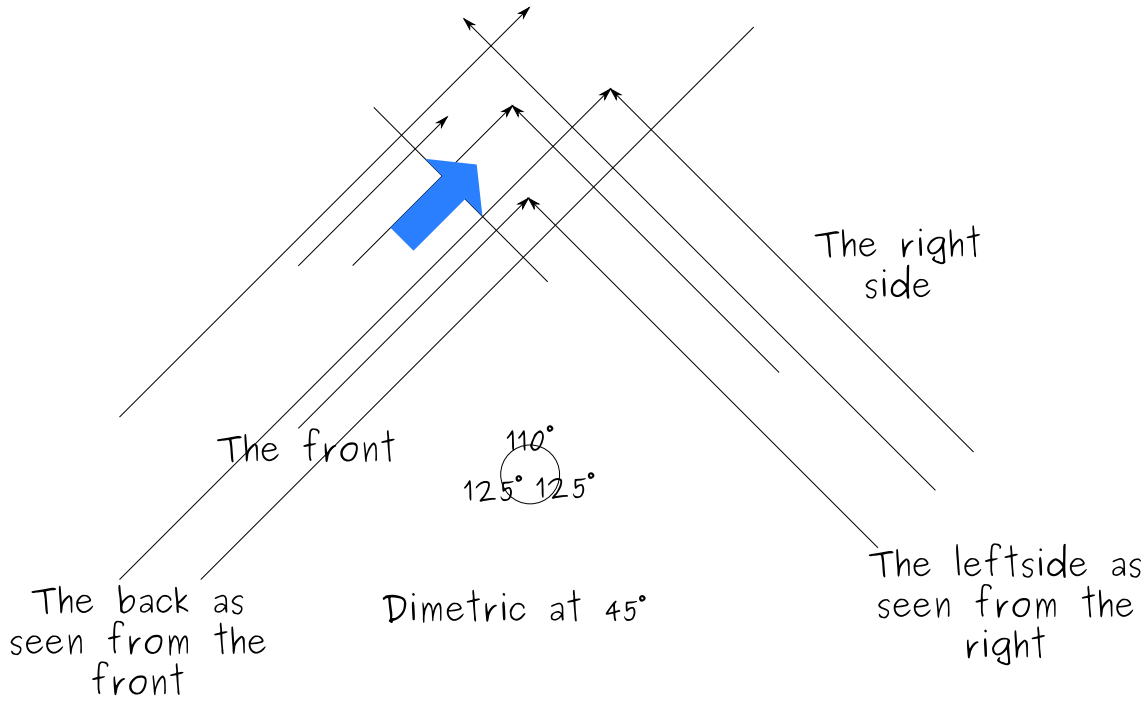
This is a continuation of the orthographic and oblique tutorial, be sure to check it out if you get confused!

Axonometric

So, the logic of adding the top is still similar to that of the side.

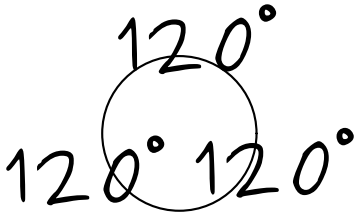


Not very interesting. But it gets much more interesting when we use a side projection:



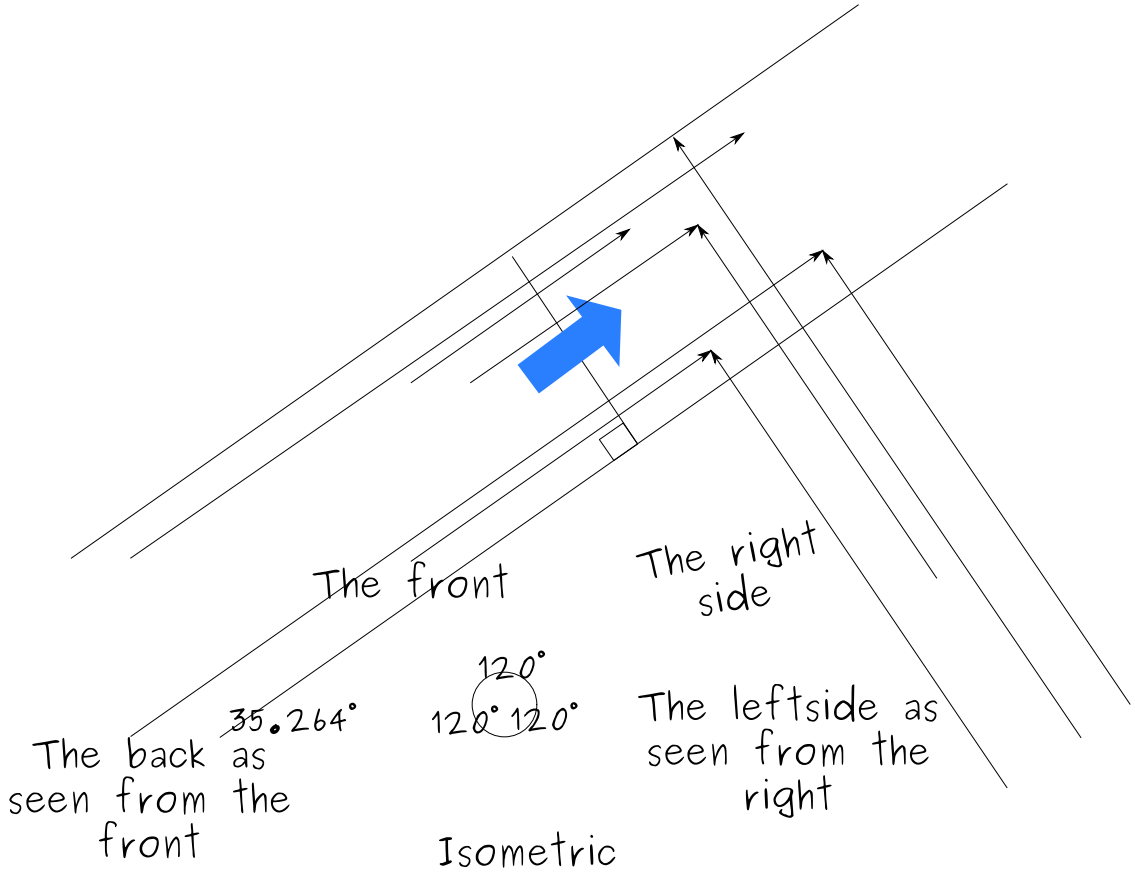
Because our cube is red on both front-sides, and blue on both left and right side, we can just use copies, this simplifies the method for cubes a lot. We call this form of axonometric projection ‘dimetric’ as it deforms two parallel lines equally.

Isometric is sorta like dimetric where we have the same angle between all main lines:



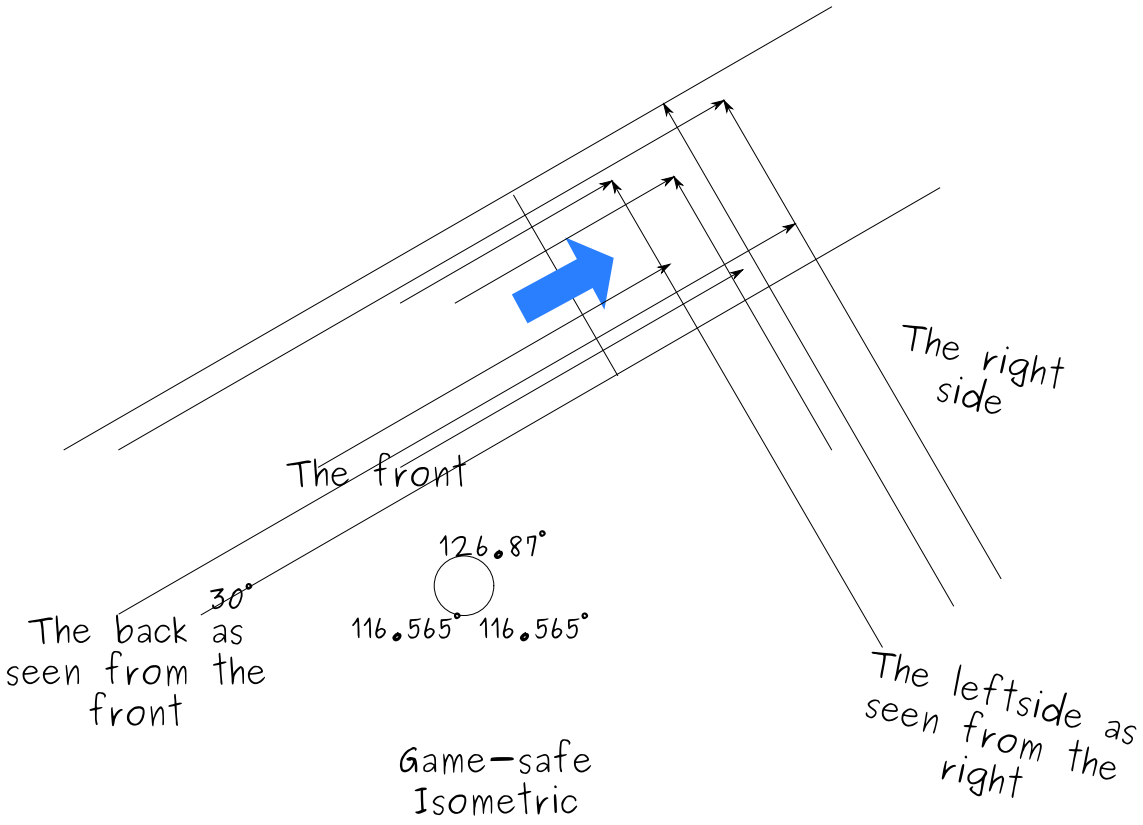
Isometric

True isometric is done with a 90-54.736=35.264° angle from ground plane:



(as you can see, it doesn't line up perfectly, because Inkscape, while more designed for making these kinds of diagrams than Krita, doesn't have tools to manipulate the line's angle in degrees)

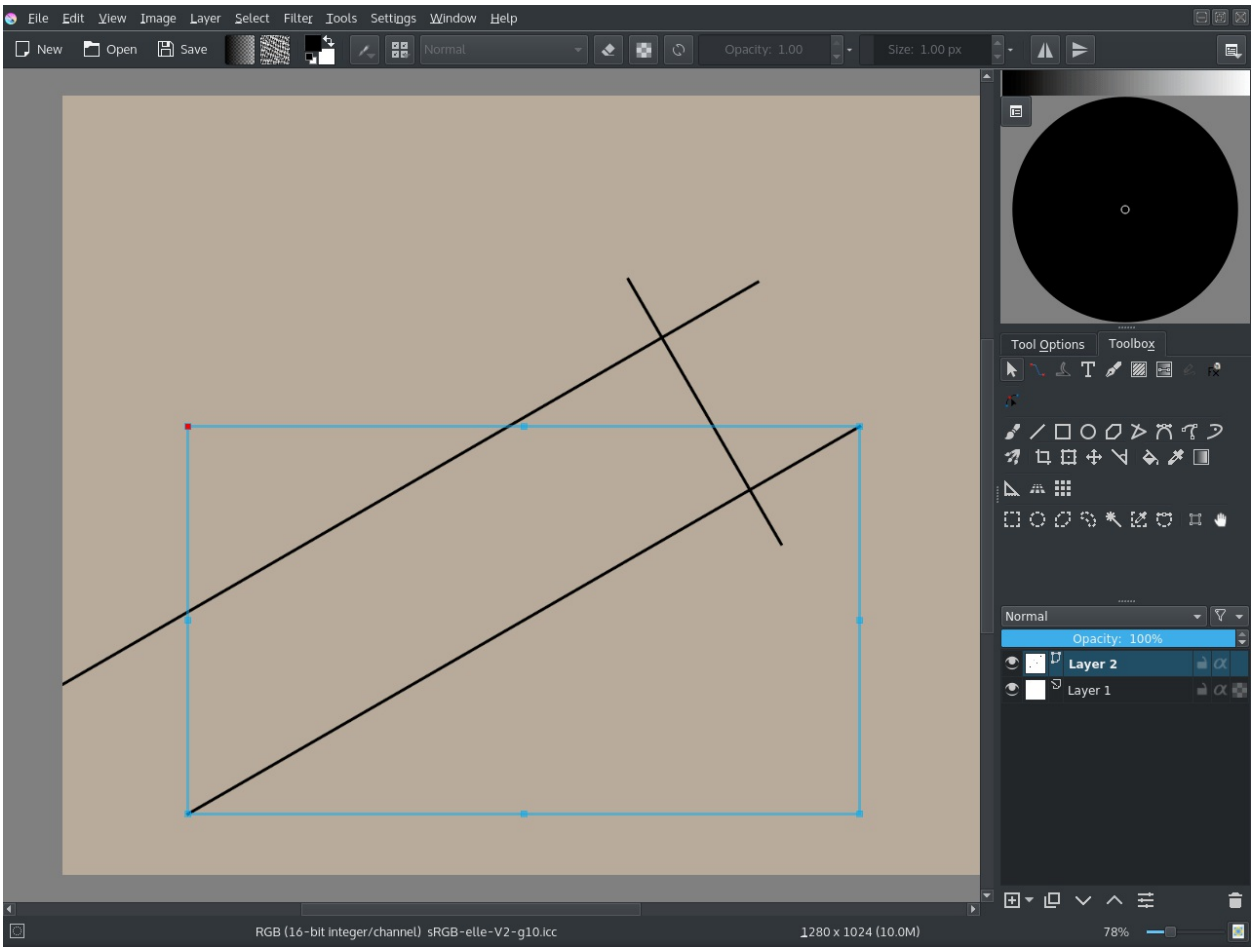
This is a bit of an awkward angle, and on top of that, it doesn't line up with pixels sensibly, so for videogames an angle of 30° from the ground plane is used.



Alright, so, let’s make an isometric out of our boy then.

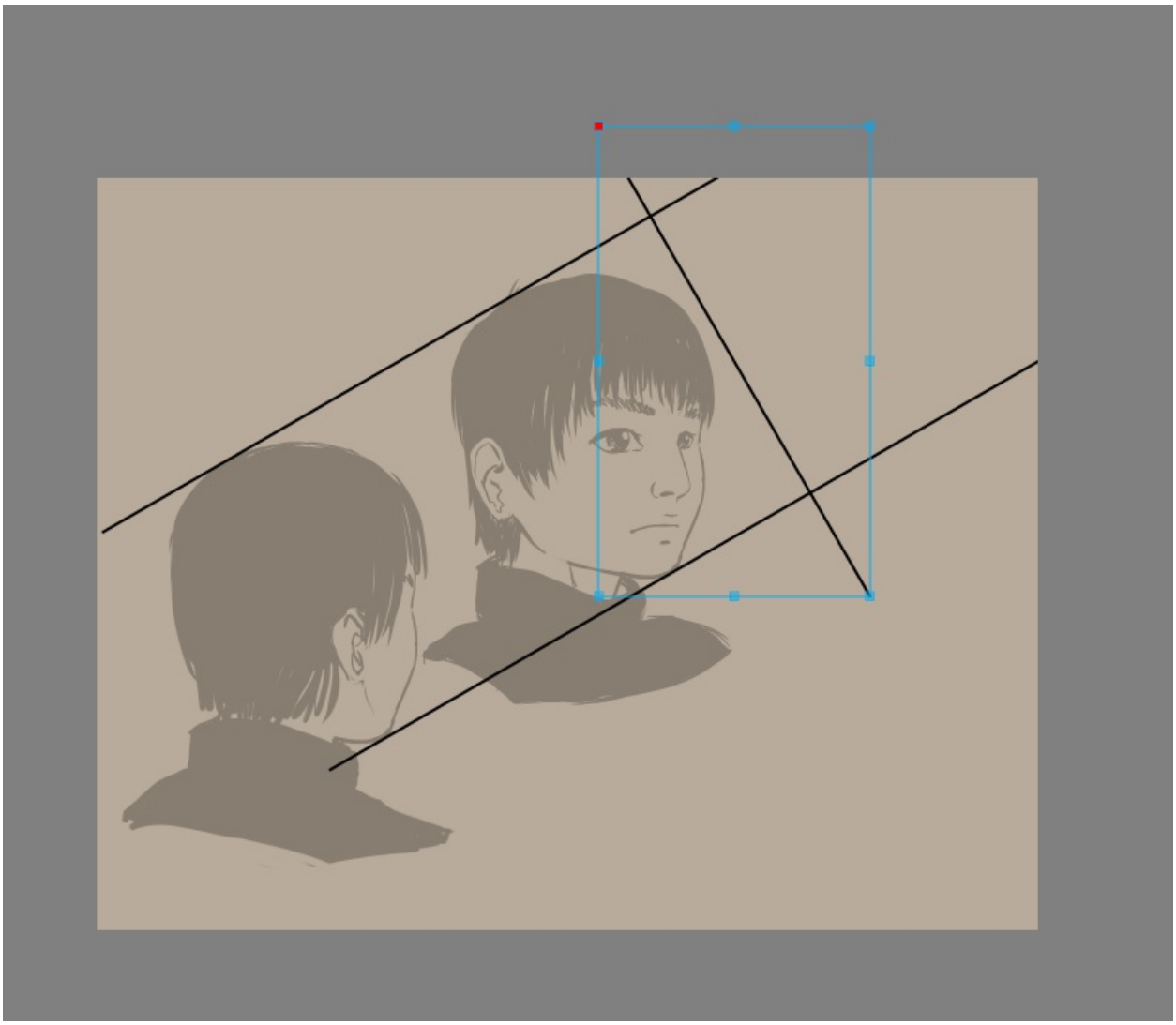
We make a new document, and add a vector layer.

On the vector layer, we select the straight line tool, start a line and then hold the `Shift` key to make it snap to angles. This’ll allow us to make a 30° setup like above:



We then import some of the frames from the animation via *Layers* ▸ *Import/Export* ▸ *Import layer*.

Then crop it by setting the crop tool to *Layer*, and use *Filters* ▸ *Colors* ▸ *Color to alpha…* to remove any background. I also set the layers to 50% opacity. We then align the vectors to them:

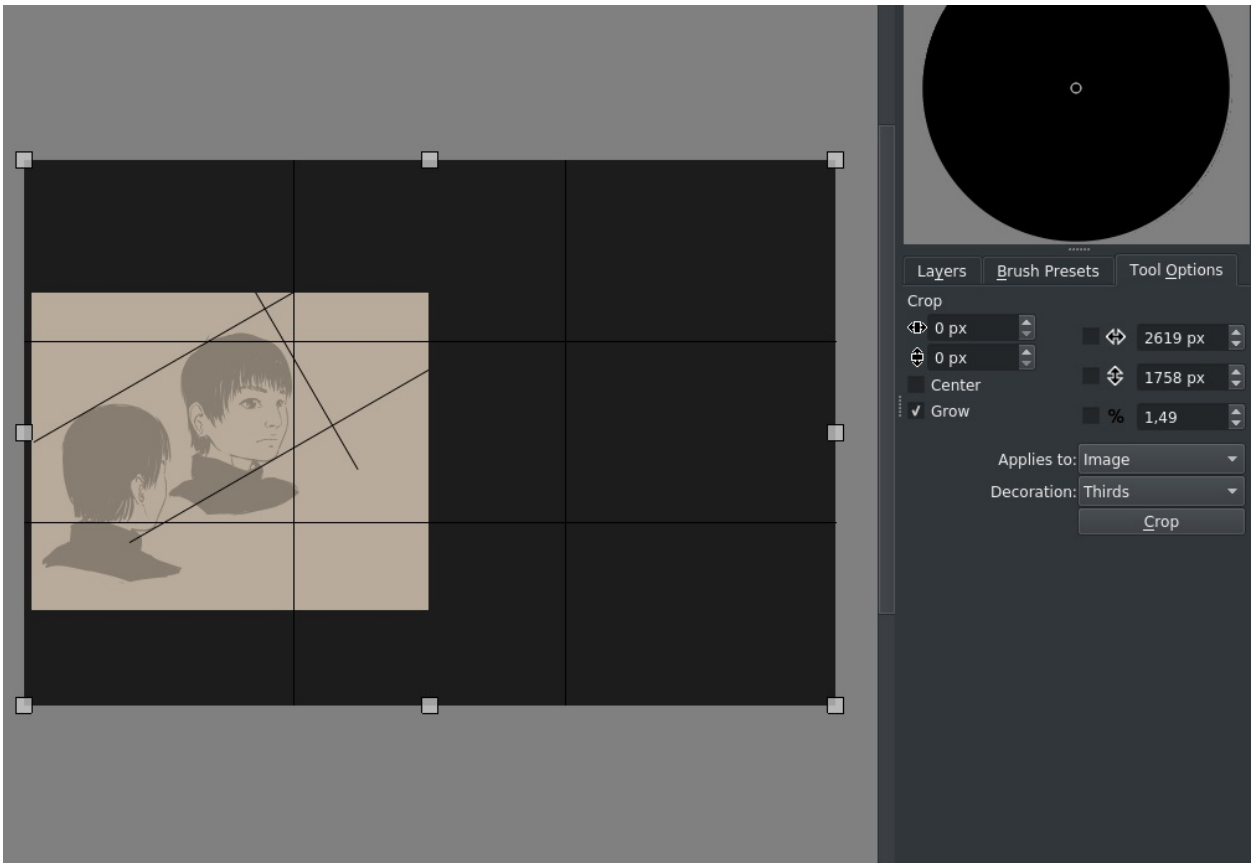


Tip

To resize a vector but keep its angle, you just select it with the shape handling tool (the white arrow) drag on the corners of the bounding box to start moving them, and then press the `Shift` key to constrain the ratio. This'll allow you to keep the angle.

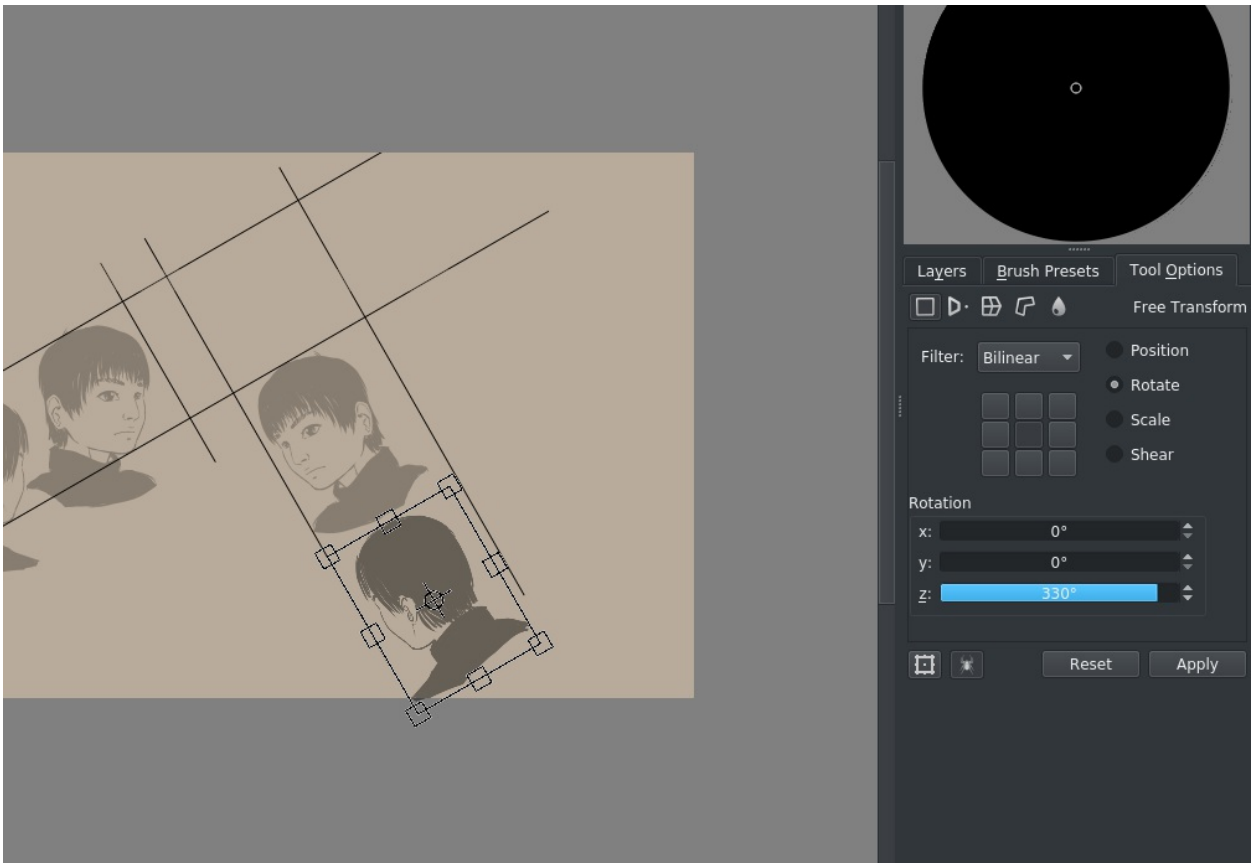
The lower image is 'the back seen from the front', we'll be using this to determine where the ear should go.

Now, we obviously have too little space, so select the crop tool, select *Image* and tick *Grow* and do the following:



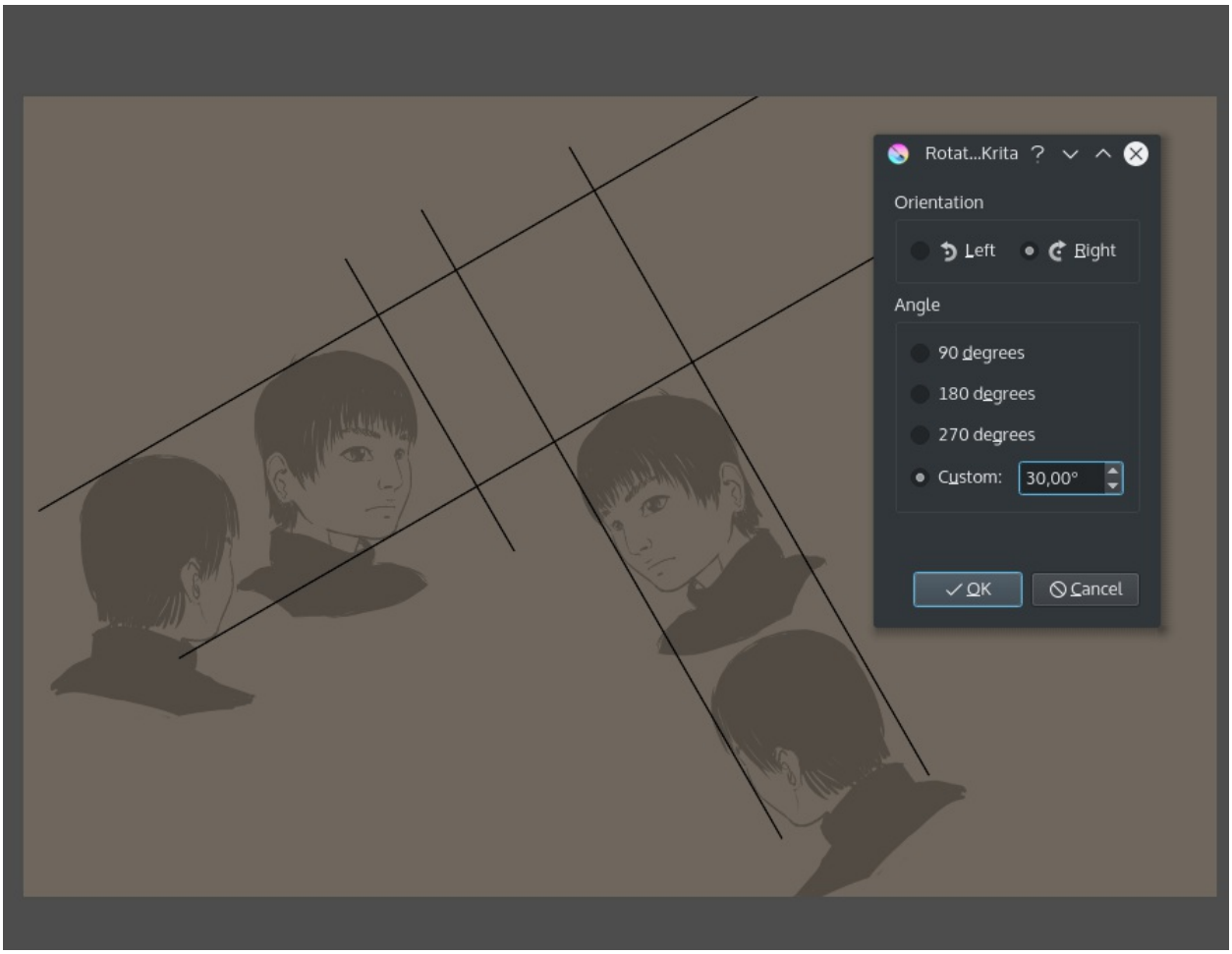
Grow is a more practical way of resizing the canvas in width and height immediately.

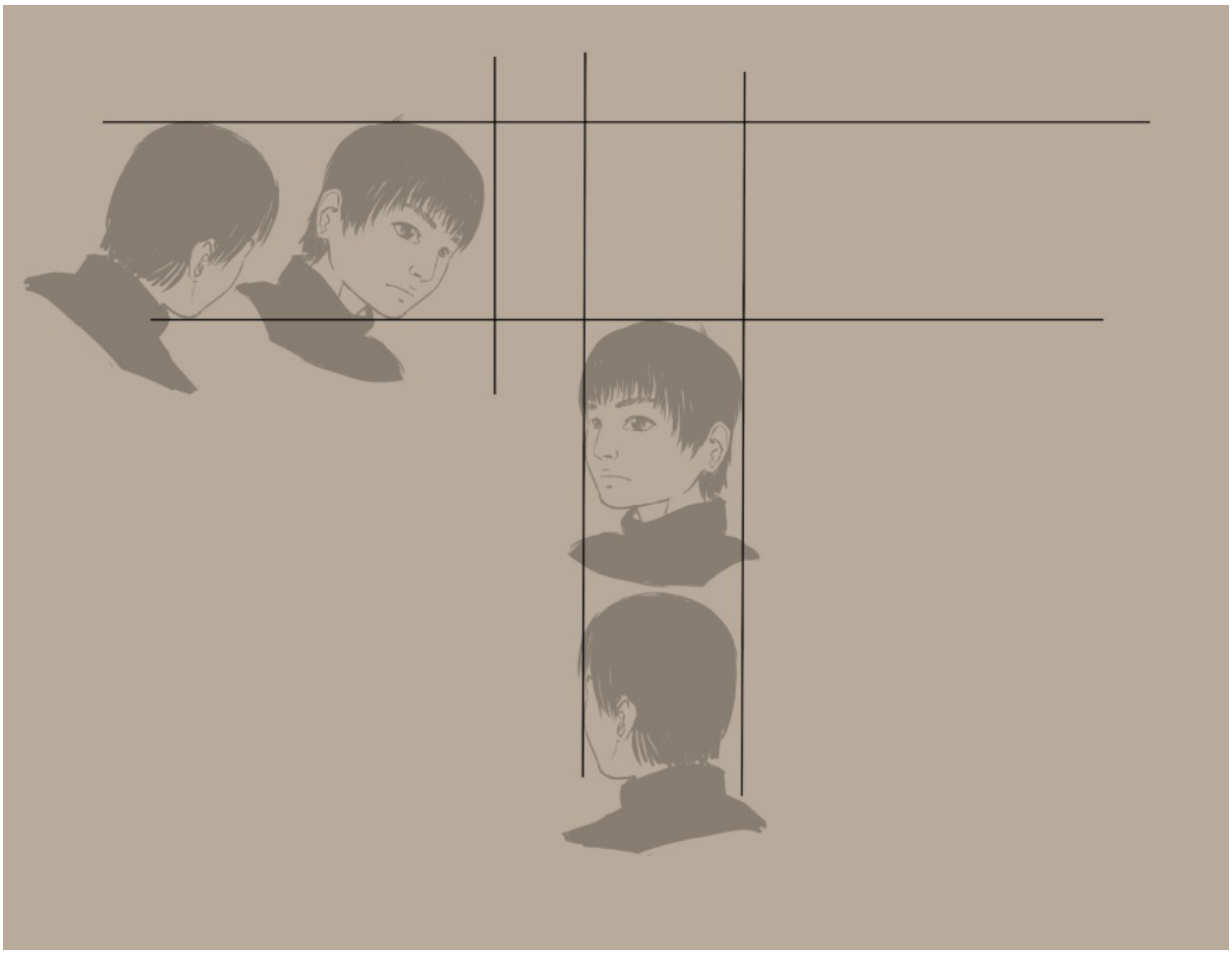
Then we align the other heads and transform them by using the transform tool options:



(330° here is 360°-30°)

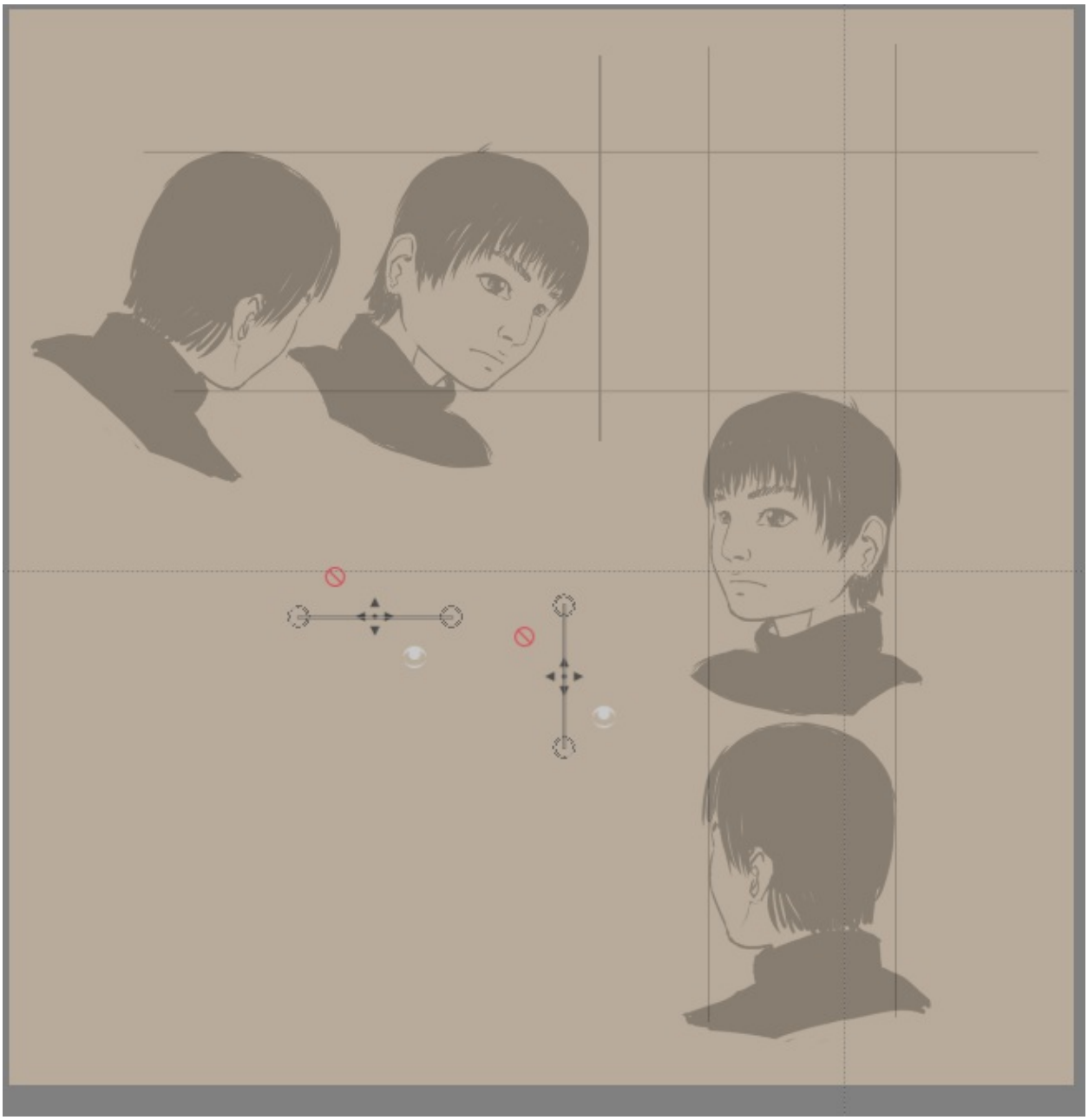
Our rectangle we'll be working in slowly becomes visible. Now, this is a bit of a difficult angle to work at, so we go to *Image* ▸ *Rotate* ▸ *Rotate Image* and fill in 30° clockwise:


Rotat...Krita
Orientation
Left
Right
Angle
90 degrees
180 degrees
270 degrees
Custom:
30,00°
OK
Cancel

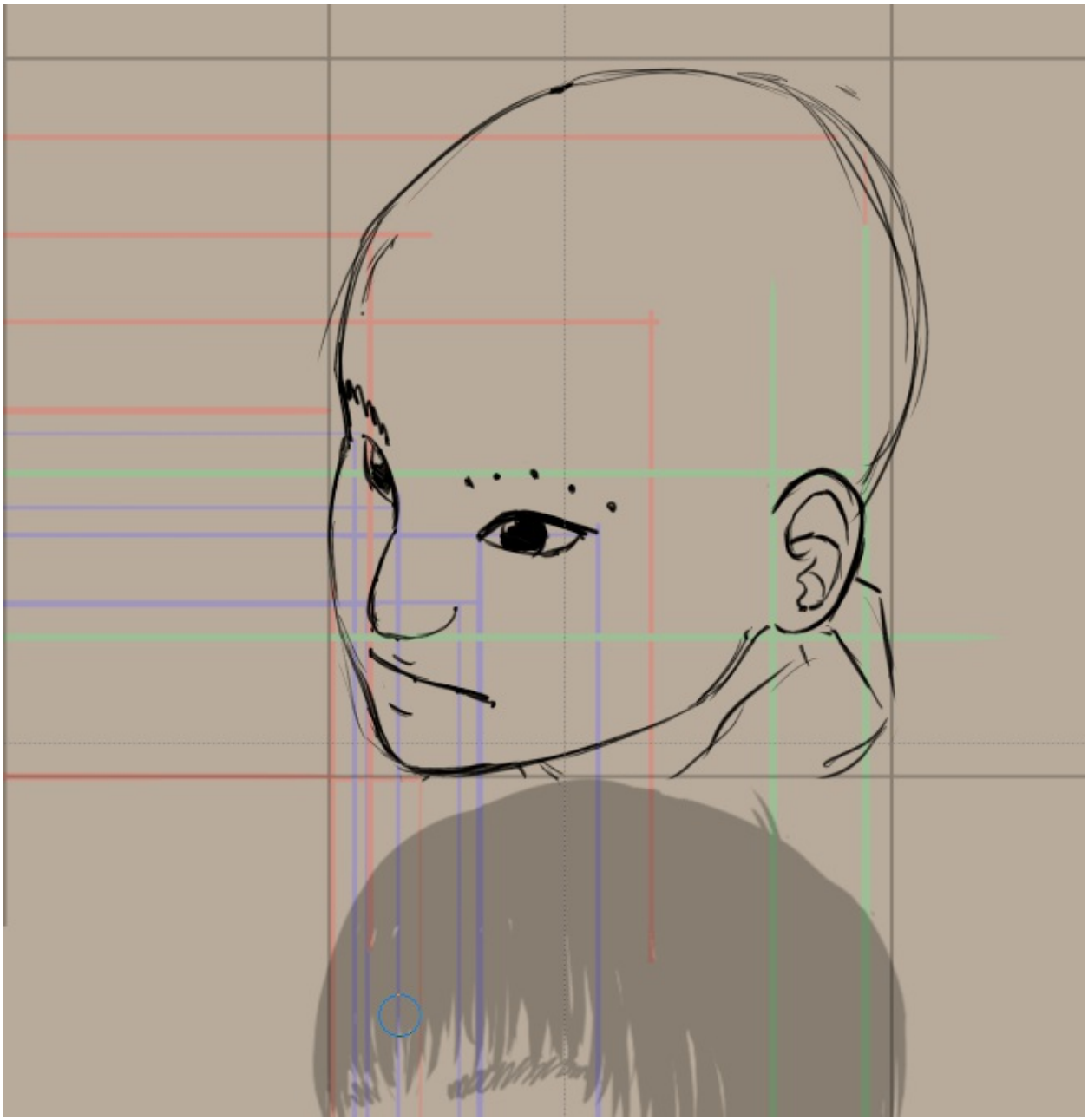


(of course, we could’ve just rotated the left two images 30°, this is mostly to be less confusing compared to the cube)

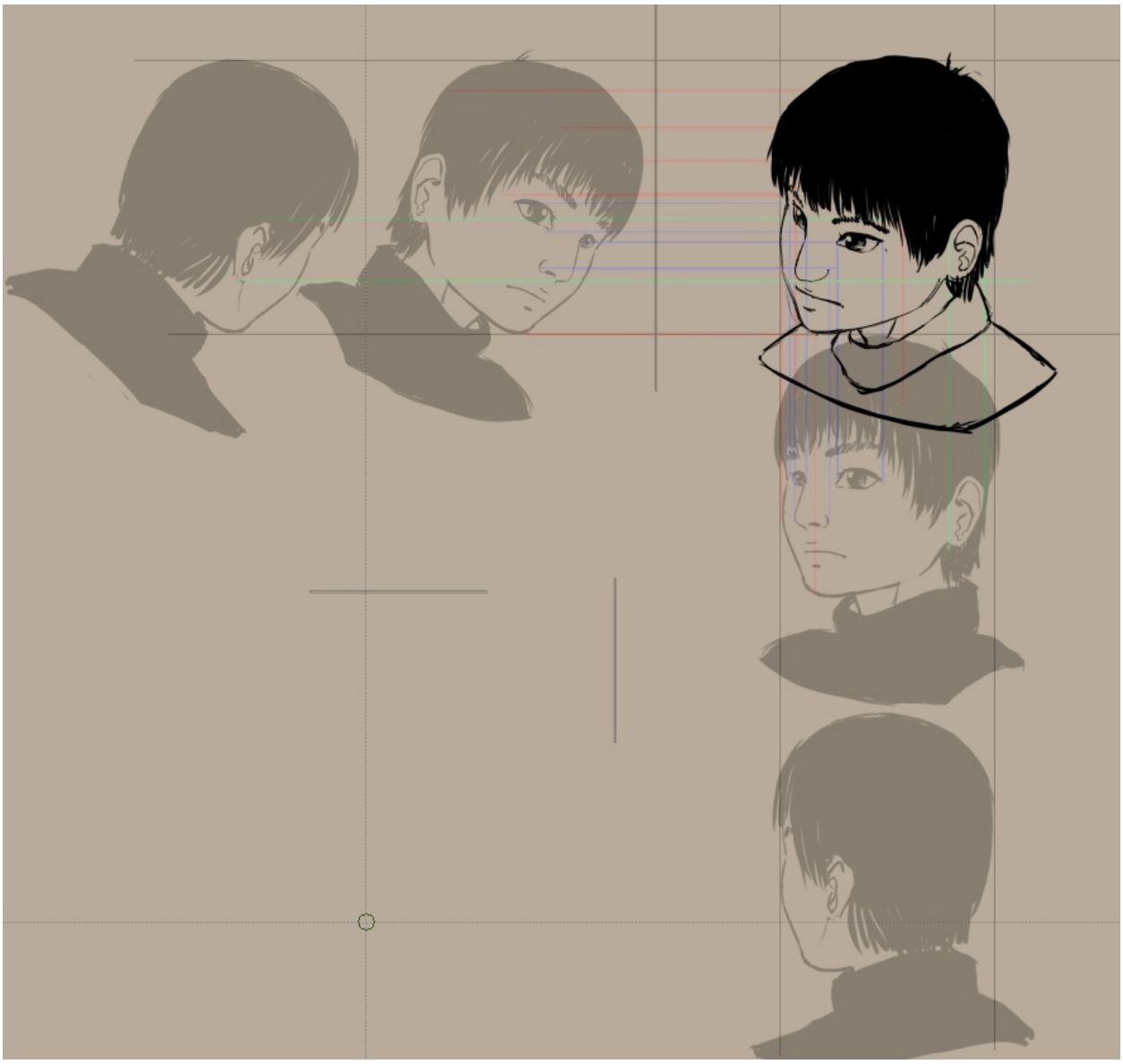
So, we do some cropping, some cleanup and add two parallel assistants like we did with the orthographic:



So the idea here is that you draw parallel lines from both sides to find points in the drawing area. You can use the previews of the assistants for this to keep things clean, but I drew the lines anyway for your convenience.



The best is to make a few sampling points, like with the eyebrows here, and then draw the eyebrow over it.



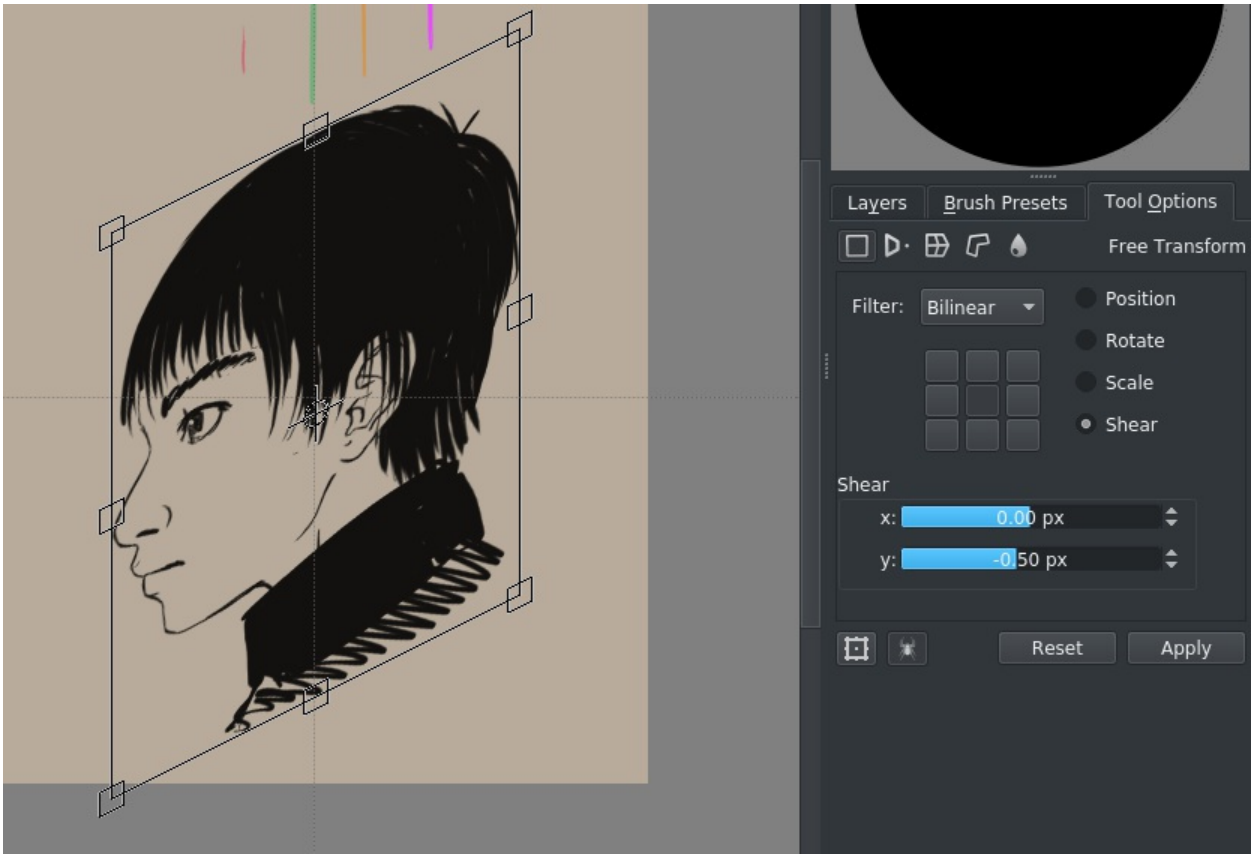
Alternative axonometric with the transform tool

Now, there’s an alternative way of getting there that doesn’t require as much space.

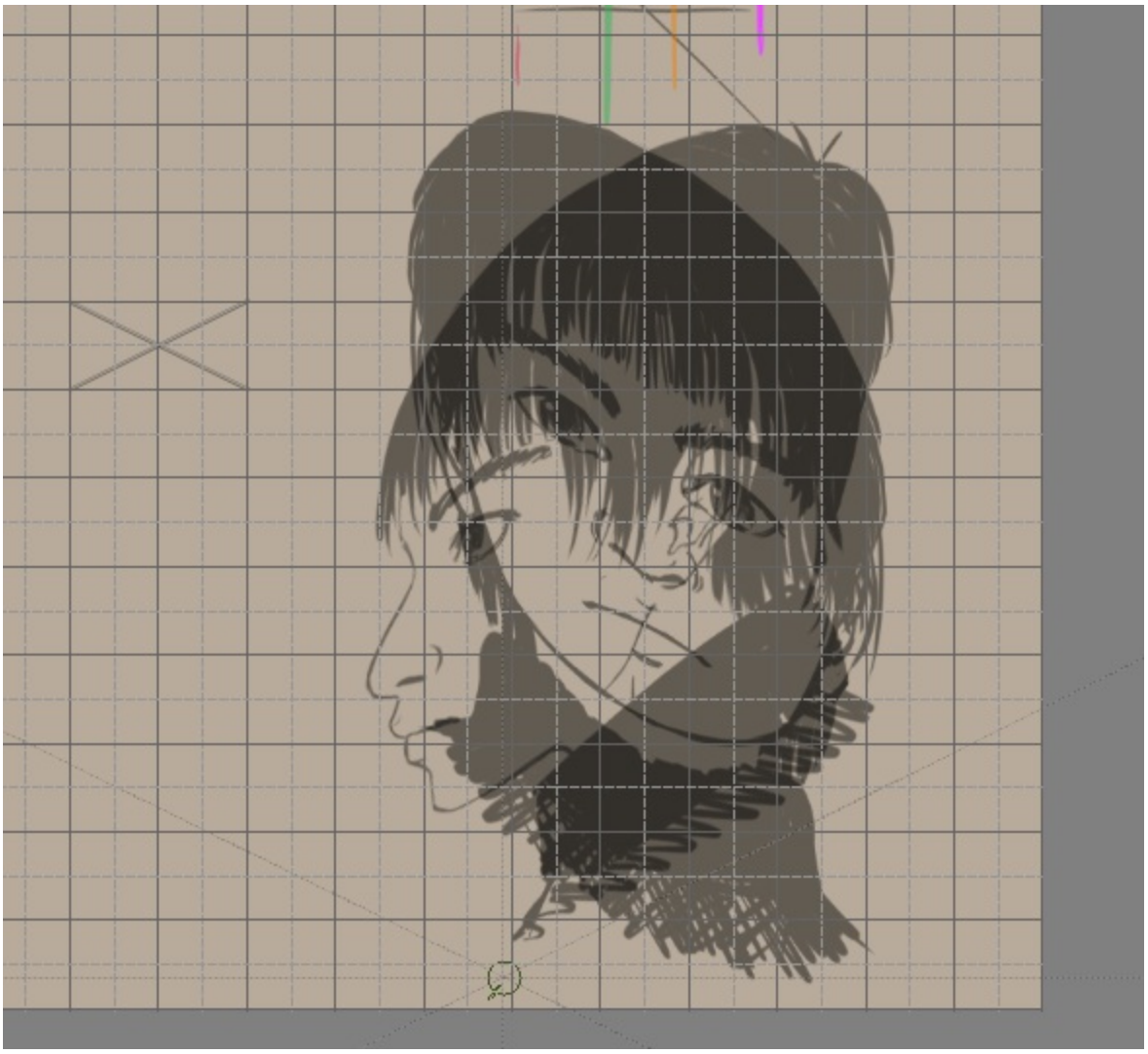
We open our orthographic with *Open existing Document as Untitled Document* so that we don’t save over it.

Our game-safe isometric has its angle at two pixels horizontal is one pixel vertical. So, we shear the ortho graphics with transform masks to -.5/+.5

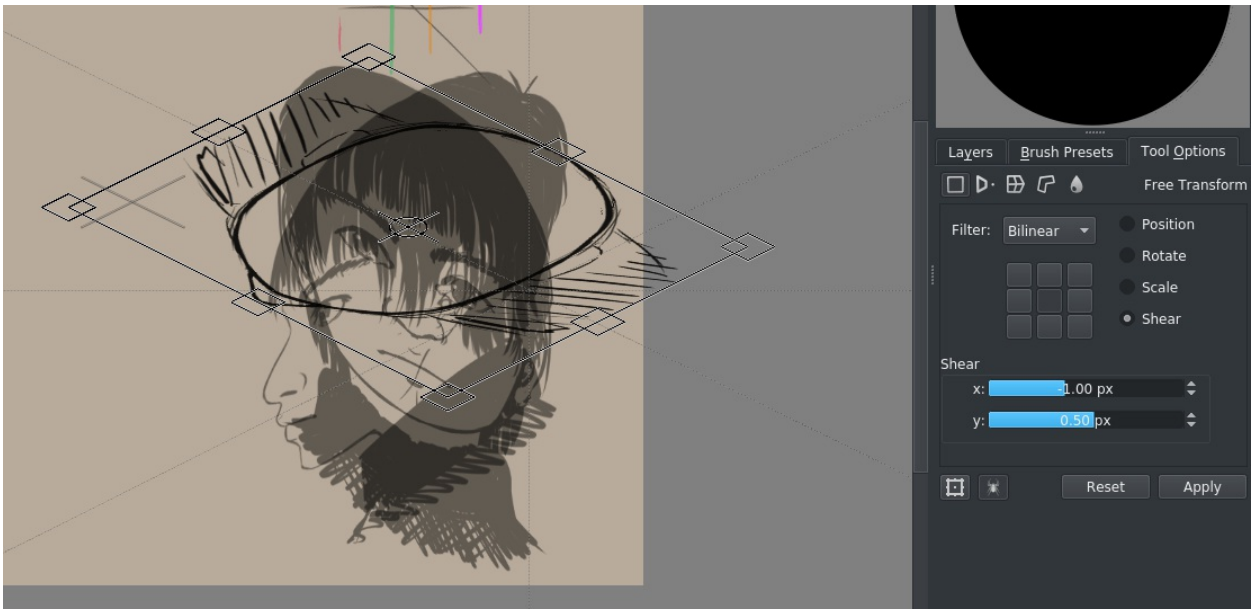
pixels (this is proportional)



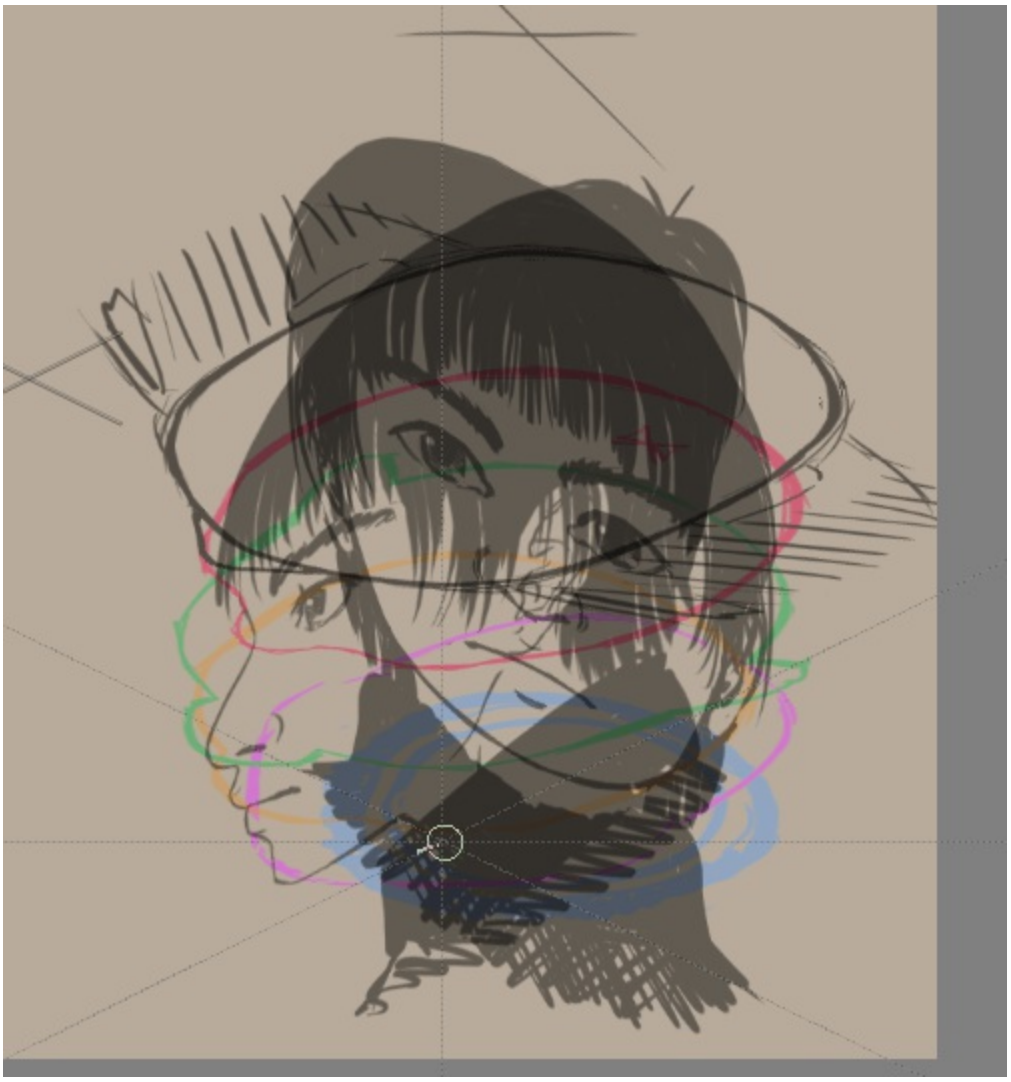
Use the grid to setup two parallel rulers that represent both diagonals (you can snap them with the `Shift + S` shortcut):



Add the top view as well:



if you do this for all slices, you get something like this:



Using the parallel rulers, you can then figure out the position of a point in 3d-ish space:

As you can see, this version both looks more 3d as well as more creepy.

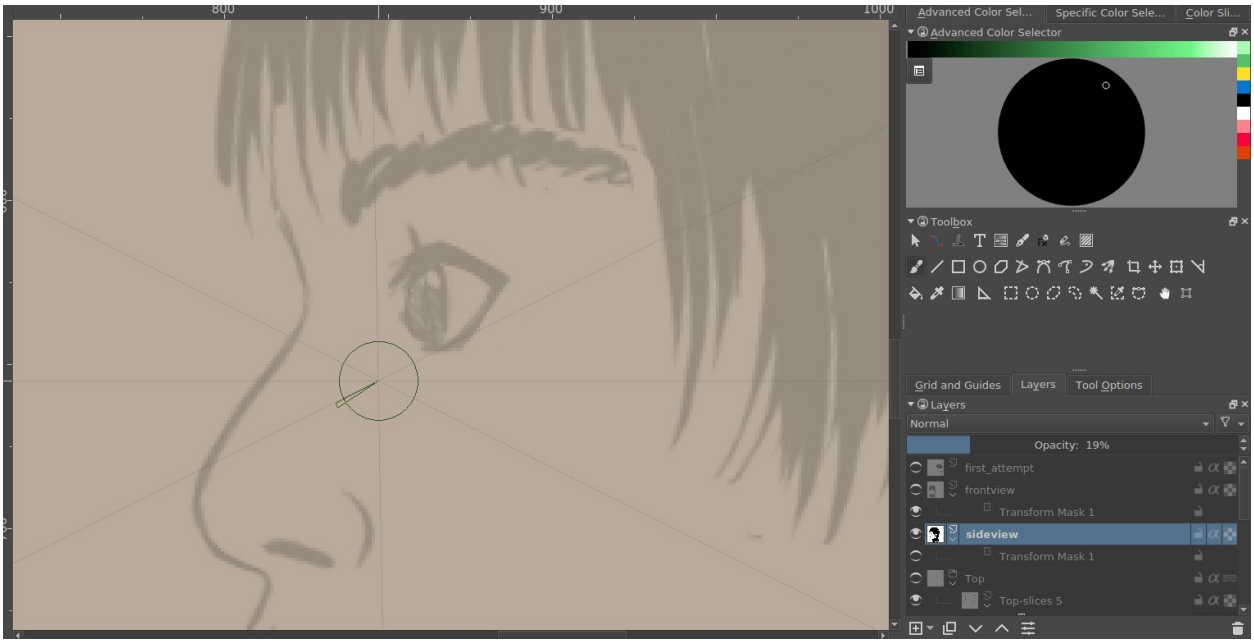
That’s because there are less steps involved as the previous version – We’re deriving our image directly from the orthographic view – so there are less errors involved.

The creepiness is because we’ve had the tiniest bit of stylisation in our side

view, so the eyes come out HUGE. This is because when we stylize the side view of an eye, we tend to draw it not perfectly from the side, but rather slightly at an angle. If you look carefully at the turntable, the same problem crops up there as well.

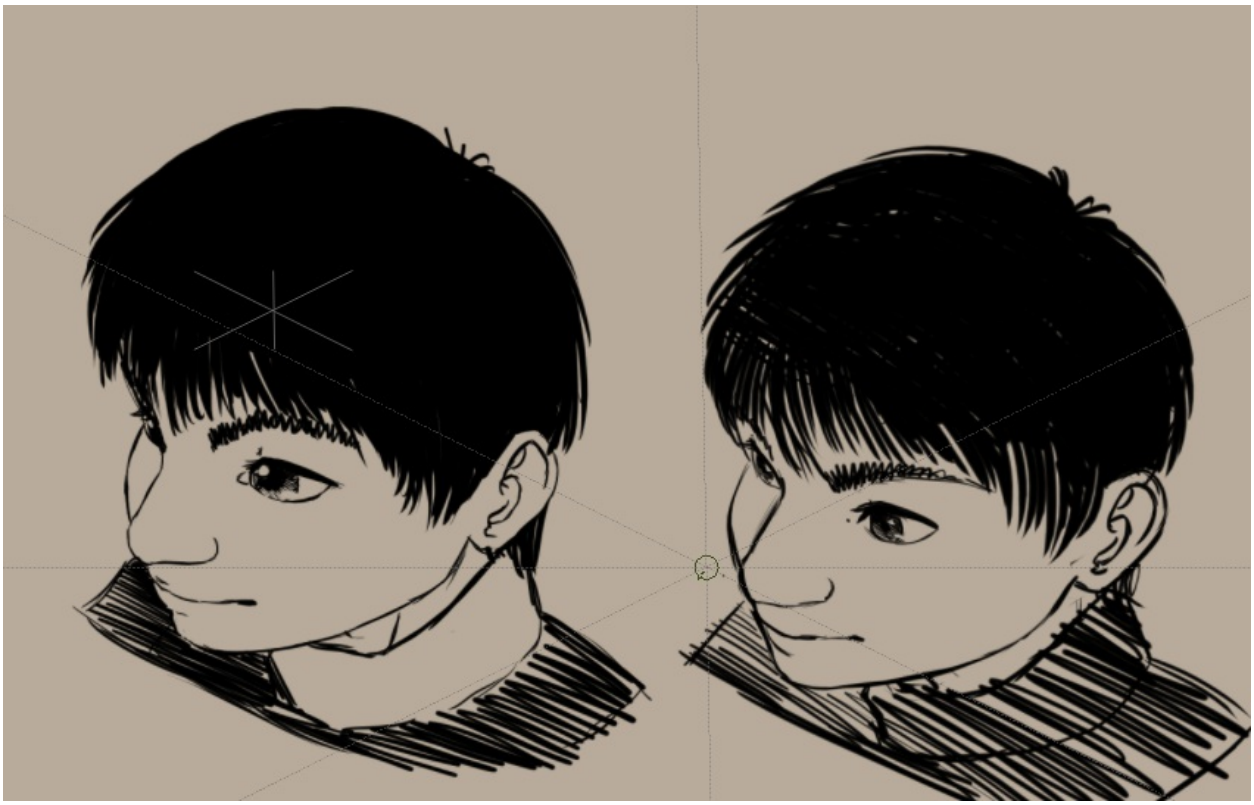
Generally, stylized stuff tends to fall apart in 3d view, and you might need to make some choices on how to make it work.

For example, we can just easily fix the side view (because we used transform masks, this is easy.)



And then generate a new drawing from that…

Compare to the old one and you should be able to see that the new result’s eyes are much less creepy:



It still feels very squashed compared to the regular parallel projection above,

and it might be an idea to not just skew but also stretch the orthos a bit.

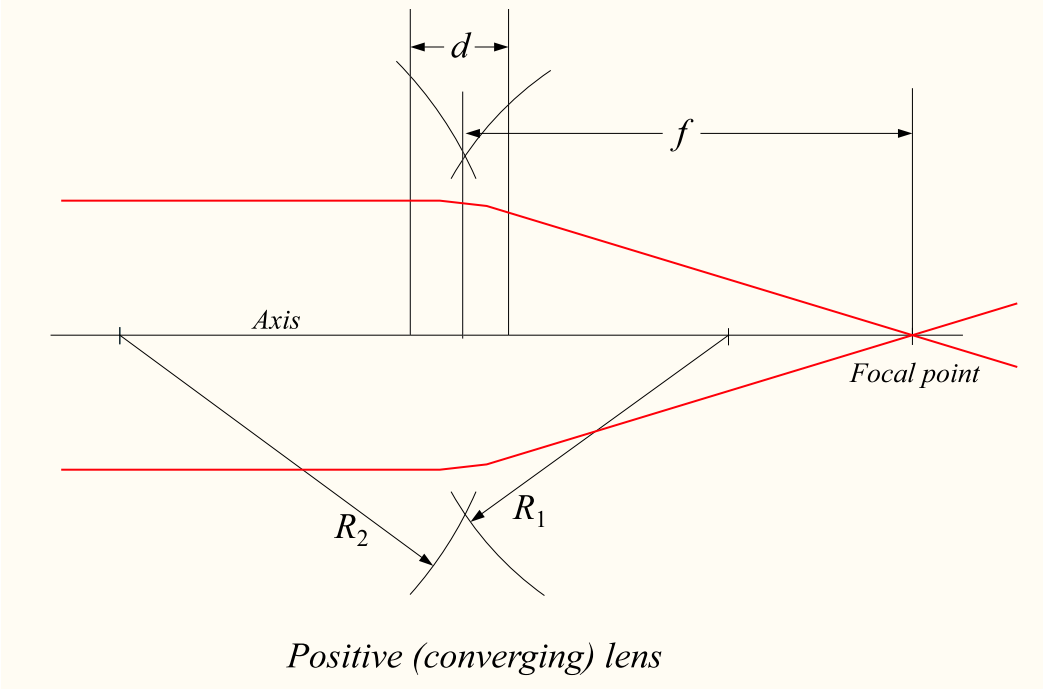
Let’s continue with perspective projection in the next one!

This is a continuation of the axonometric tutorial, be sure to check it out if you get confused!

Perspective Projection

So, up till now we've done only parallel projection. This is called like that because all the projection lines we drew were parallel ones.

However, in real life we don't have parallel projection. This is due to the lens in our eyes.



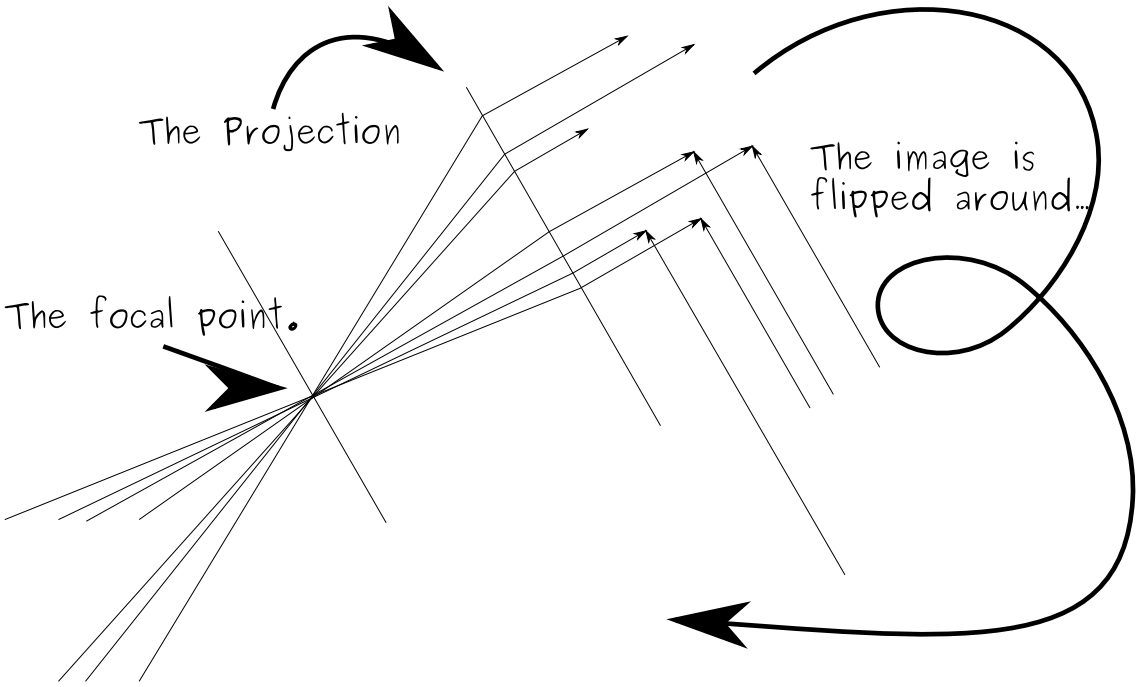
Positive (converging) lens

Convex lenses, as this lovely image from wikipedia [https://en.wikipedia.org/wiki/Lens_%28optics%29] shows us, have the ability to turn parallel lightrays into converging ones.

The point where all the rays come together is called the focal point, and the vanishing point in a 2d drawing is related to it as it's the expression of the maximum distortion that can be given to two parallel lines as they're skewed toward the focal point.

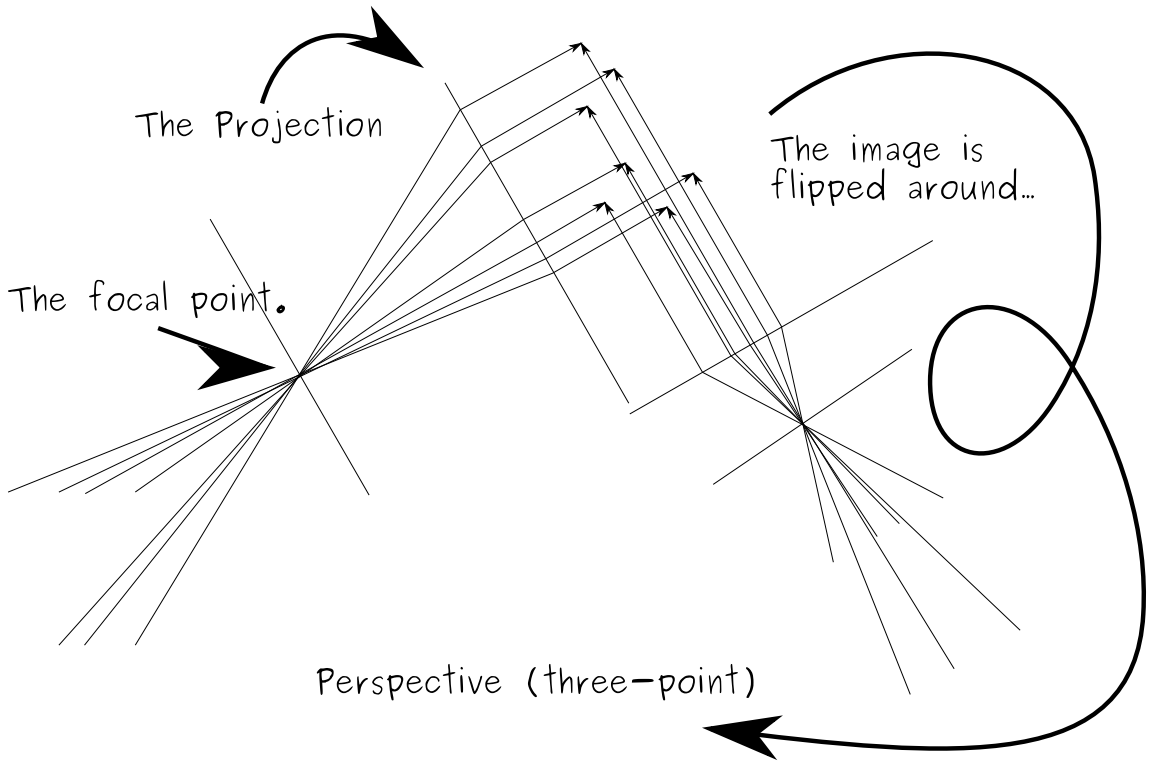
As you can see from the image, the focal point is not an end-point of the rays. Rather, it is where the rays cross before diverging again… The only difference is that the resulting image will be inverted. Even in our eyes this inversion happens, but our brains are used to this awkwardness since childhood and turn it around automatically.

Let's see if we can perspectively project our box now.



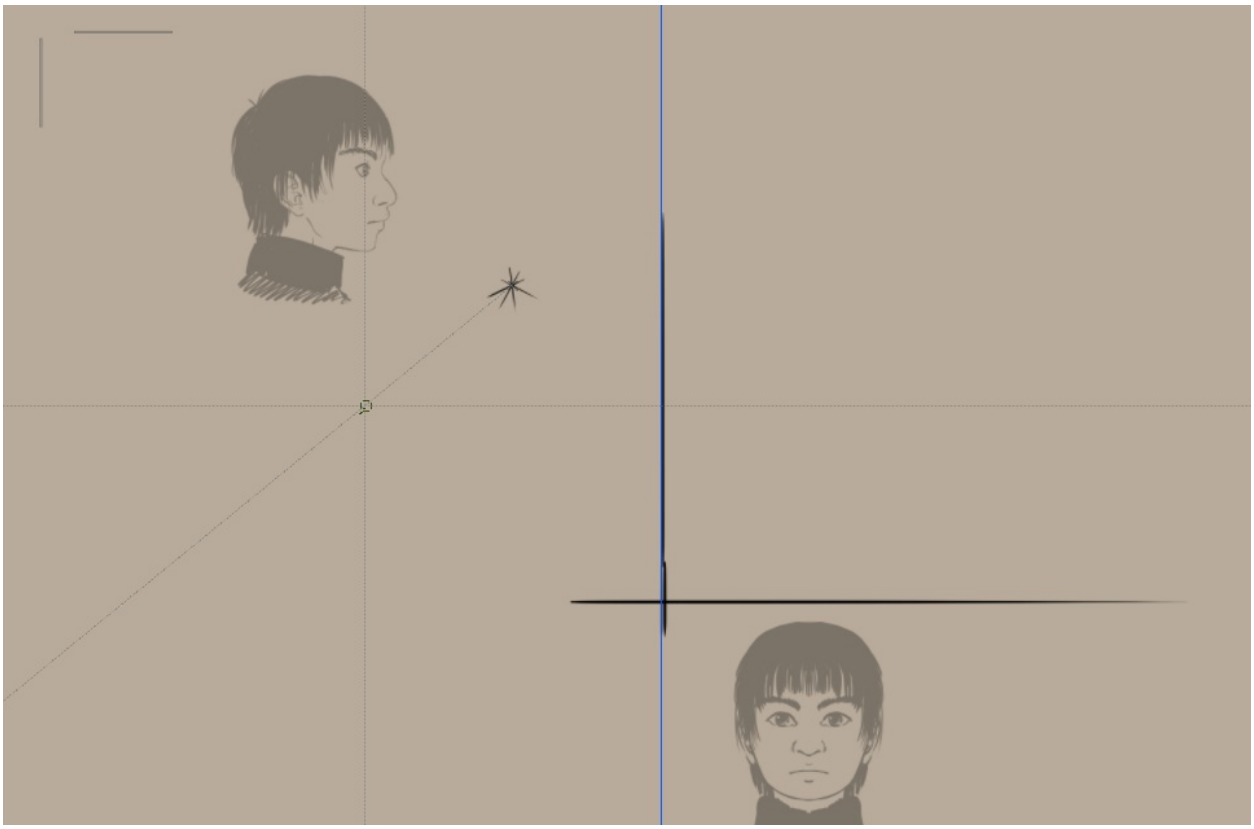
Perspective (one/two-point)

That went pretty well. As you can see we sort of *merged* the two sides into one (resulting into the purple side square) so we had an easier time projecting. The projection is limited to one or two vanishing point type projection, so only the horizontal lines get distorted. We can also distort the vertical lines



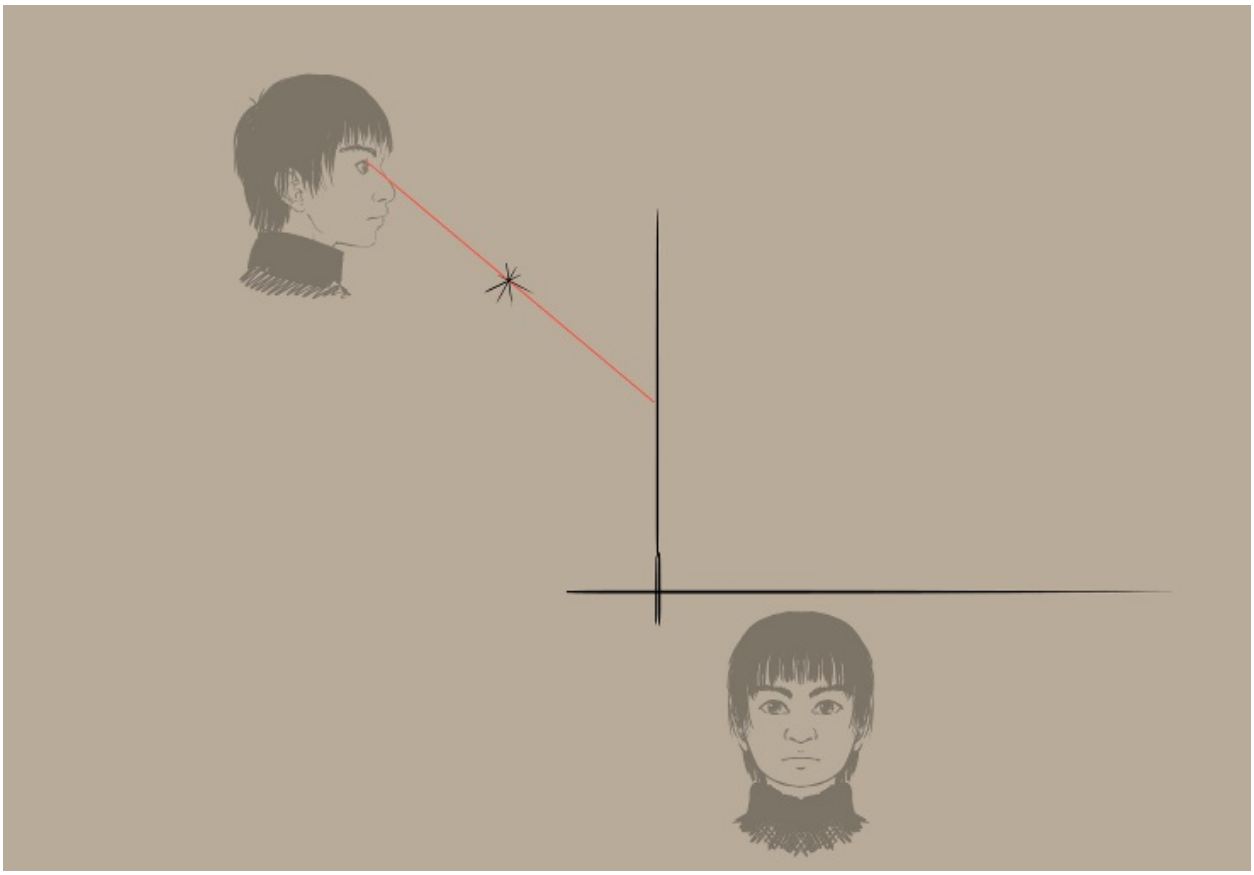
… to get three-point projection, but this is a bit much. (And I totally made a mistake in there…)

Let’s setup our perspective projection again…



We’ll be using a single vanishing point for our focal point. A guide line will be there for the projection plane, and we’re setting up horizontal and vertical parallel rules to easily draw the straight lines from the view plane to where they intersect.

And now the workflow in GIF format… (don’t forget you can rotate the canvas with the 4 and 6 keys)

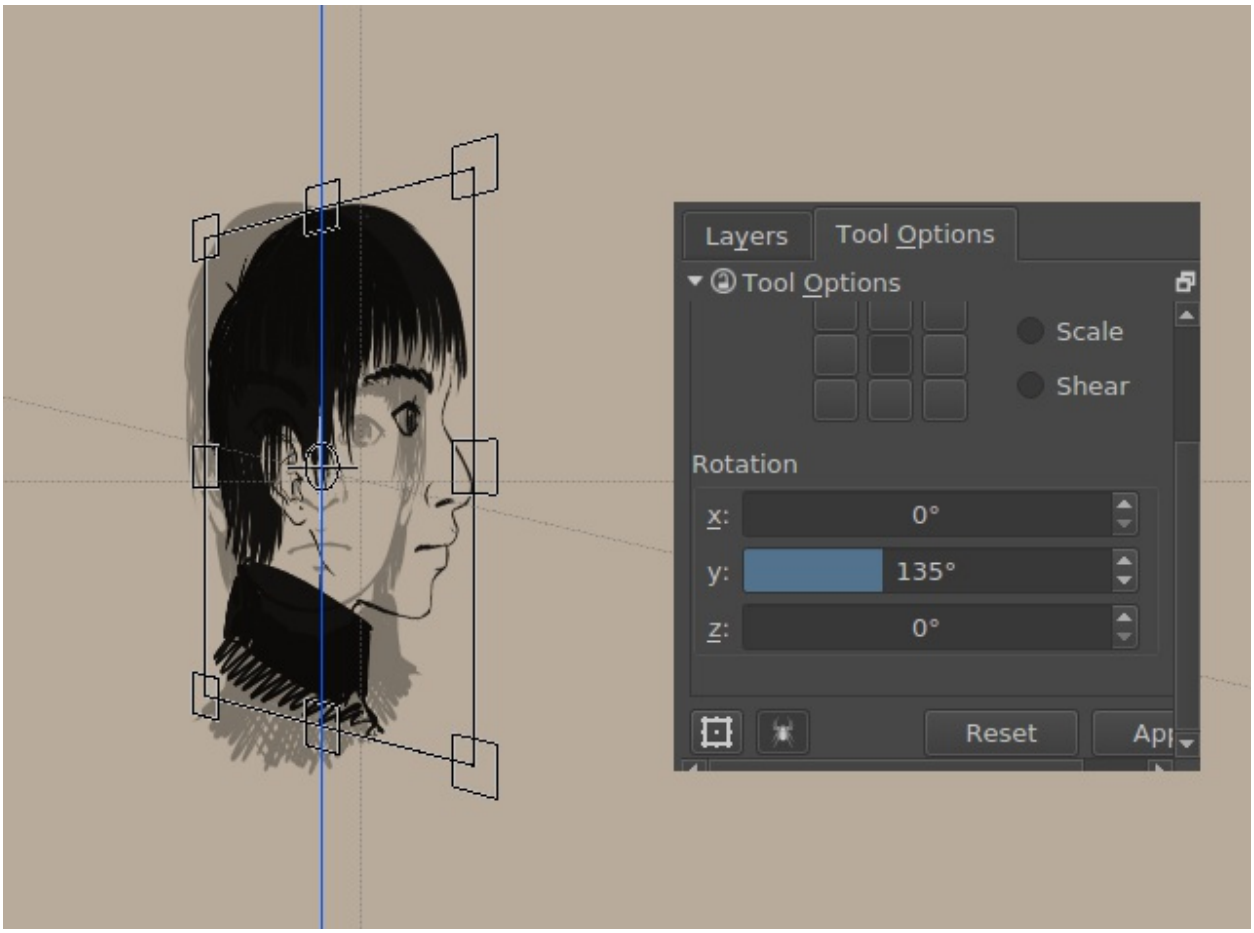


Result:

Looks pretty haughty, doesn’t he?

And again, there’s technically a simpler setup here…

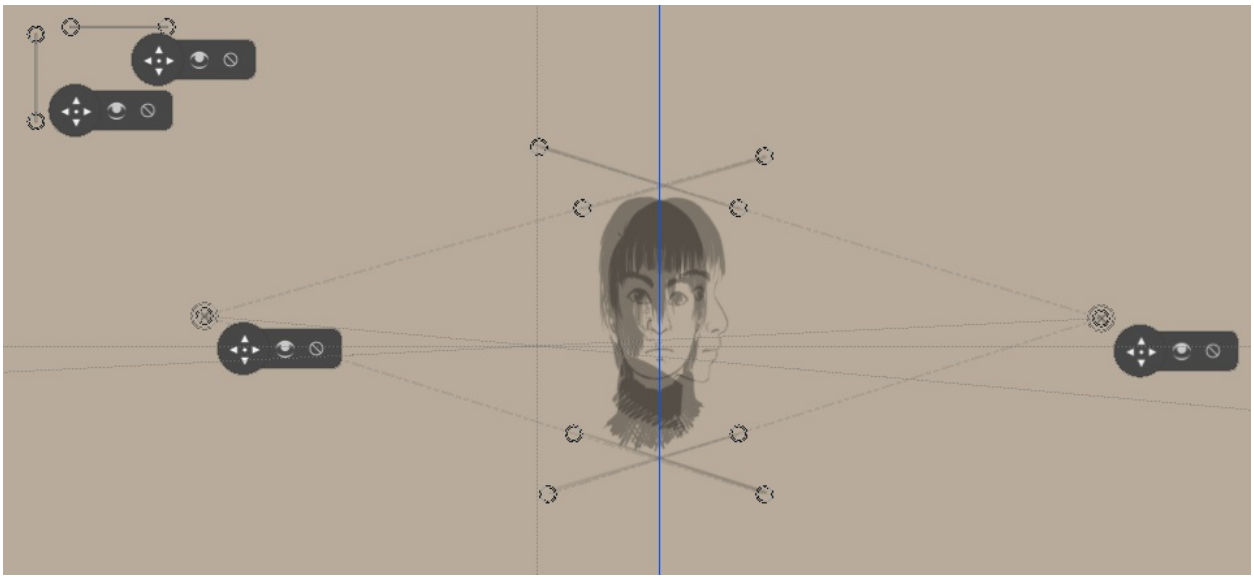
Did you know you can use Krita to rotate in 3d? No?



Well, now you do.

The ortho graphics are being set to 45 and 135 degrees respectively.

We draw horizontal lines on the originals, so that we can align vanishing point rulers to them.



And from this, like with the shearing method, we start drawing. (Don’t forget the top-views!)

Which should get you something like this:

But again, the regular method is actually a bit easier…

But now you might be thinking: gee, this is a lot of work… Can’t we make it easier with the computer somehow?

Uhm, yes, that’s more or less why people spent time on developing 3d graphics technology:

(The image above is sculpted in blender using our orthographic reference)

So let us look at what this technique can be practically used for in the next part…

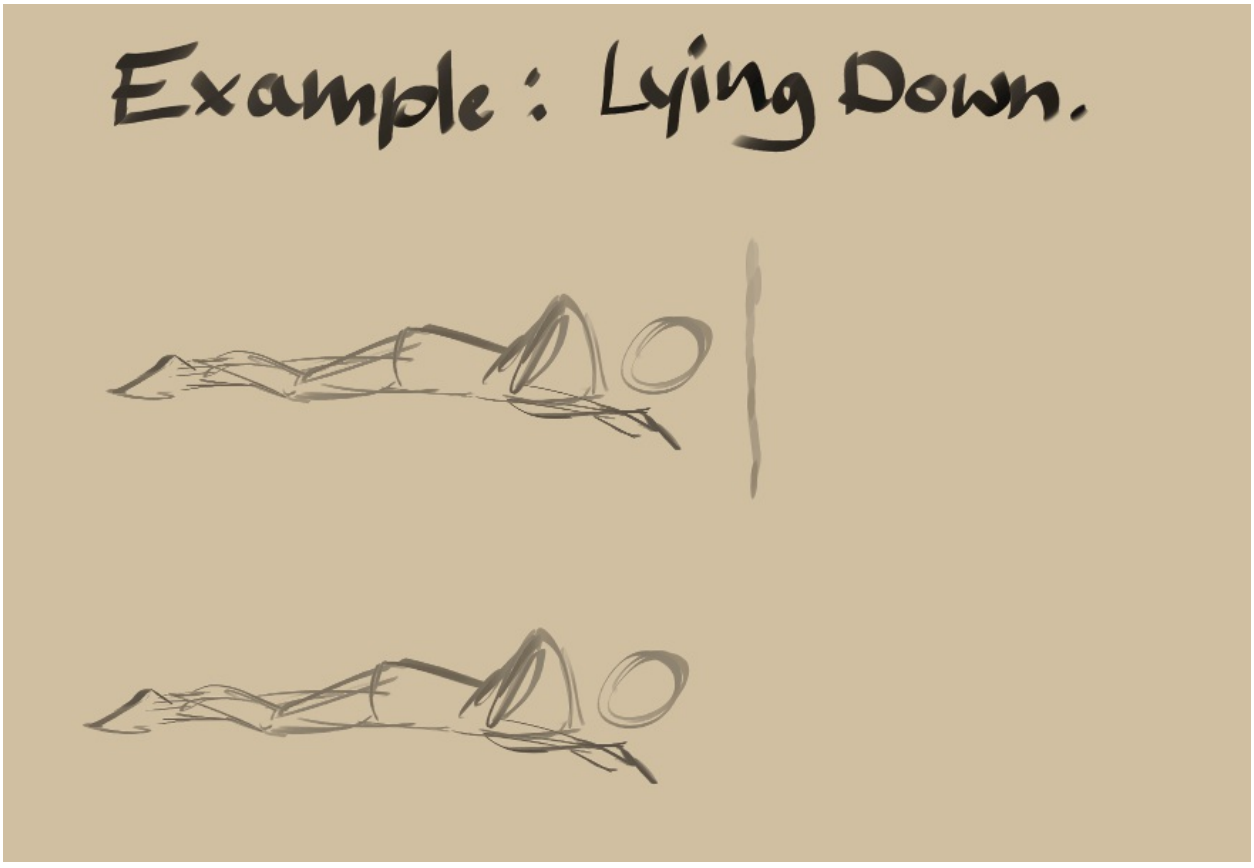
This is a continuation of the perspective projection tutorial, be sure to check it out if you get confused!

Practical

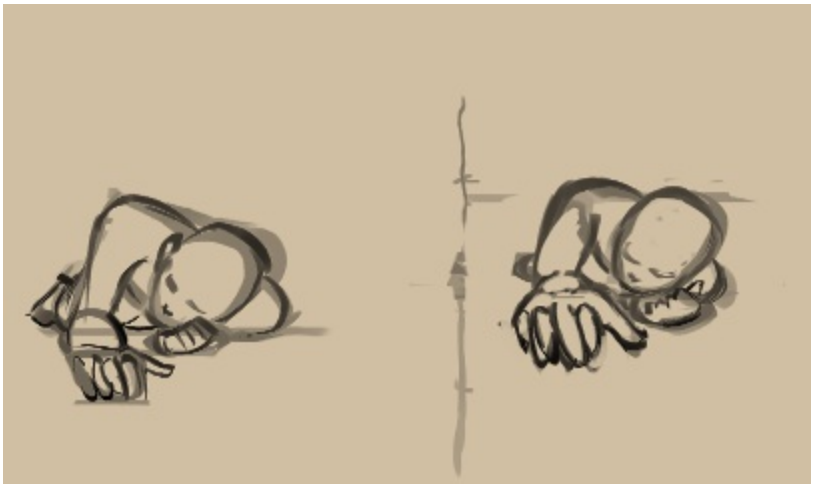
So, if computers can already automate a ton, and it is fairly complicated, is there still a use for a traditional 2d artist to learn this?

Yes, actually. The benefit that 2d art still has over 3d is that it’s plain faster for single images, especially with complicated subjects like faces and bodies.

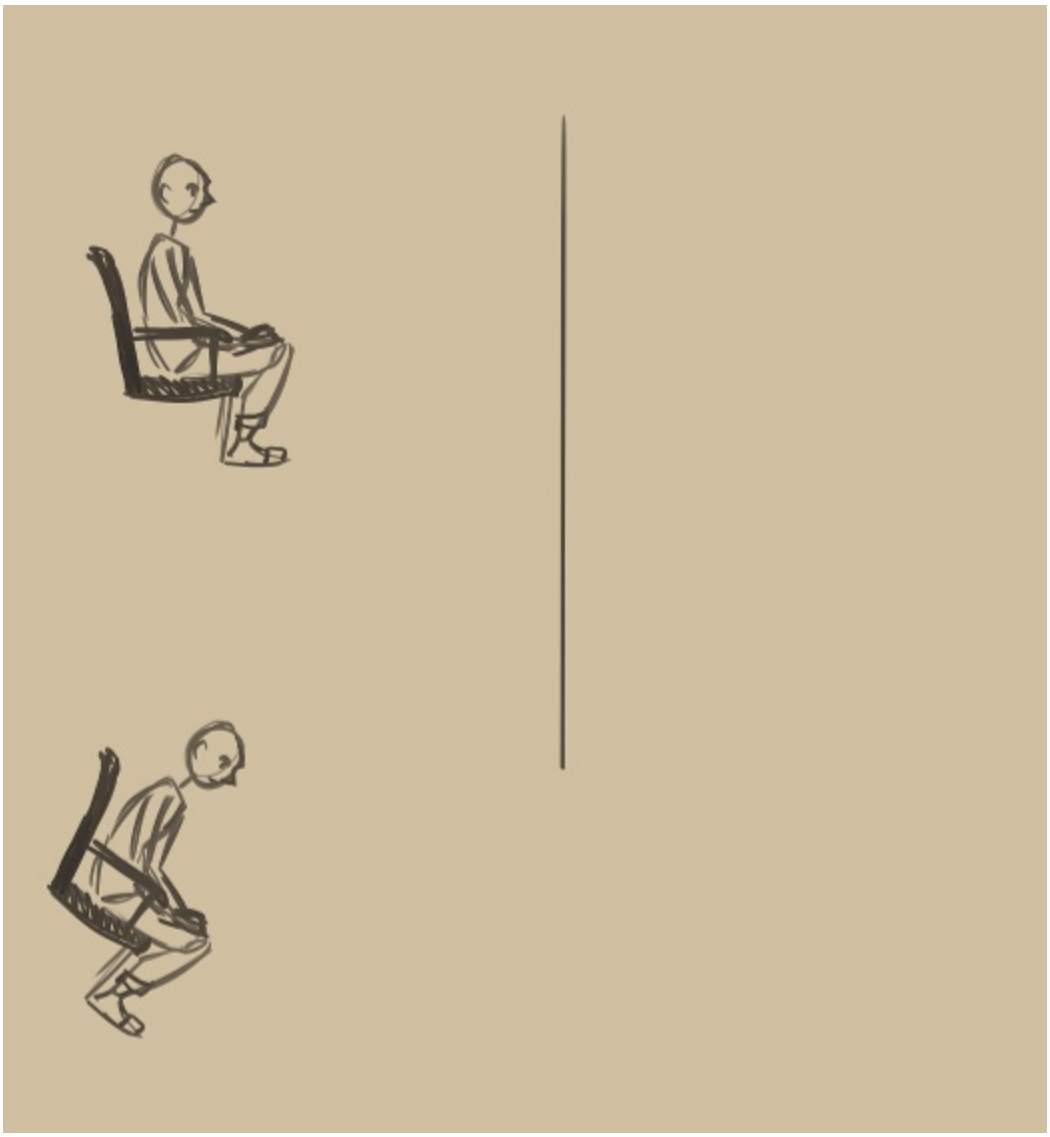
Perspective projection can help a lot getting down those annoying poses, like people lying down. It also helps when combining 2d and 3d, as when you know where the camera is in the 3d render, you can use that in a projection to get the character projected.



The side view of a person lying down is often easy to draw, but the top view or the view from the feet isn’t. Hence why we use the side view to do perspective projection on.



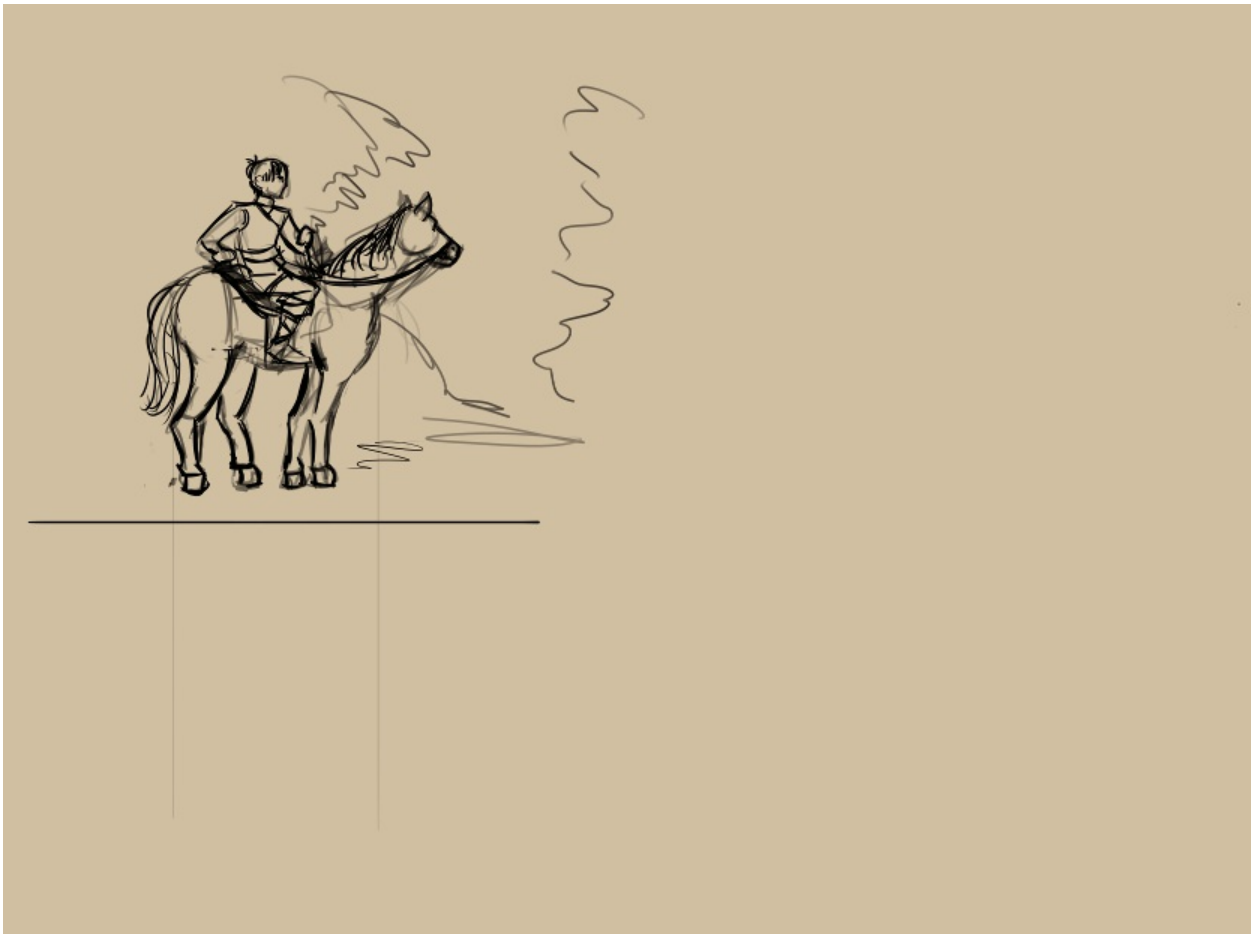
Another example with an equally epic task: sitting.



Now, with this one we have a second vanishing point above the front-view. It should be about the same distance above the front-view as it is above the head of the rotated side-view. The projection plane should also be the same distance from the vanishing point, but that doesn’t mean it has to be behind. This is something I avoided in the earlier examples, because it makes the working field really messy, but if you look up perspective projection you’ll see multiple examples of this method.

Also of note is that you actually should be having the view plane/projection plane perfectly perpendicular to the angle of the focal point, otherwise you get odd distortion, this doesn’t happen here, which means this sitting person is a bit more stretched vertically than necessary.

One more, for the road…



Here you can see that the misalignment of the vanishing point to the projection plane causes skewing which was then fixed by Krita’s transform tools, technically it’s of course correct, but what is correct doesn’t always look good. (I also mess up the position of the shoulder for a good while if you look closely.)

Conclusion and afterthoughts

I probably didn’t make as nice result images as I could have, especially if you compare it to the 3d images. However, you can still see that the main landmarks are there. The real use of this technique lies in poses though, and it allows you to iterate on a pose quite quickly once you get the hang of it.

Generally, it’s worth exploring, if only because it improves your spatial sense.

See also

- https://en.wikipedia.org/wiki/Axonometric_projection
- https://blenderartists.org/t/creating-an-isometric-camera/440743

- http://flarerpg.org/tutorials/isometric_tiles/
- https://en.wikipedia.org/wiki/Isometric_graphics_in_video_games_and_
- https://en.wikipedia.org/wiki/Lens_%28optics%29

Reference Manual

A quick run-down of all of the tools that are available.

Contents:

Audio for Animation

Caution

Audio for animation is an unfinished feature. It has multiple bugs and may not work on your system.

You can add audio files to your animation to help sync lips or music. This functionality is available in the timeline docker.

Importing Audio Files

Krita supports MP3, OGM, and WAV audio files. When you open up your timeline docker, there will be a speaker button in the top left area.

If you press the speaker button, you will get the available audio options for the animation.

- Open
- Mute
- Remove audio
- Volume slider

Krita saves the location of your audio file. If you move the audio file or rename it, Krita will not be able to find it. Krita will tell you the file was moved or deleted the next time you try to open the Krita file up.

Using Audio

After you import the audio, you can scrub through the timeline and it will play the audio chunk at the time spot. When you press the Play button, the entire the audio file will playback as it will in the final version. There is no visual display of the audio file on the screen, so you will need to use your

ears and the scrubbing functionality to position frames.

Exporting with Audio

To get the audio file included when you are exporting, you need to include it in the Render Animation options. In the *File* ▸ *Render Animation* options there is a checkbox *Include Audio*. Make sure that is checked before you export and you should be good to go.

Packages needed for Audio on Linux

The following packages are necessary for having the audio support on Linux:

For people who build Krita on Linux:

- libasound2-dev
- libgstreamer1.0-dev gstreamer1.0-pulseaudio
- libgstreamer-plugins-base1.0-dev
- libgstreamer-plugins-good1.0-dev
- libgstreamer-plugins-bad1.0-dev

For people who use Krita on Linux:

- libqt5multimedia5-plugins
- libgstreamer-plugins-base1.0
- libgstreamer-plugins-good1.0
- libgstreamer-plugins-bad1.0

Since Krita 4.4, audio works inside the appimage.

Blending Modes

Blending modes are a little difficult to explain. Basically, when one layer is above the other, the computer uses a bit of programming to decide how the combination of both layers will look.

Blending modes can not just apply to Layers, but also to individual strokes.

Favorites

These are the blending modes that have been ticked as favorites, defaulting these are:

- Addition
- Burn
- Color, HSV, HSI, HSL, HSY
- Color Dodge
- Darken
- Erase
- Lighten
- Luminosity
- Multiply
- Normal
- Overlay
- Saturation HSI, HSV, HSL, HSY

Hotkeys associated with Blending modes

Defaultly the following hotkeys are associated with blending modes used for painting. Note: these shortcuts do not change the blending mode of the current layer.

You first need to use modifiers `Alt + Shift`, then use the following hotkey to have the associated blending mode:

- A Linear Burn
- B Burn
- C Color, HSV, HSI, HSL, HSY
- D Color Dodge
- E Difference
- F Soft Light (Photoshop) & Soft Light SVG
- I Dissolve
- J Linear Light
- K Darken
- L Hard Mix
- M Multiply
- O Overlay
- Q Behind
- R Normal
- S Screen
- T Saturation HSI, HSV, HSL, HSY
- U Hue HSV, HSI, HSL, HSY
- V Vivid Light
- W Exclusion
- X Linear Dodge
- Y Luminosity
- Z Pin Light
- Next Blending Mode +
- Previous Blending Mode -

Available Blending Modes

- Arithmetic
  - Addition
  - Divide
  - Inverse Subtract
  - Multiply
  - Subtract
- Binary
  - AND
  - CONVERSE
  - IMPLICATION

    - NAND
    - NOR
    - NOT CONVERSE
    - NOT IMPLICATION
    - OR
    - XOR
    - XNOR
- Darken
    - Burn
    - Easy Burn
    - Fog Darken (IFS Illusions)
    - Darken
    - Darker Color
    - Gamma Dark
    - Linear Burn
    - Shade (IFS Illusions)
- HSX
    - HSI
    - HSL
    - HSV
    - HSY
    - HSX Blending Modes
- Lighten
    - Color Dodge
    - Gamma Illumination
    - Gamma Light
    - Hard Light
    - Lighten
    - Lighter Color
    - Linear Dodge
    - Easy Dodge
    - Flat Light
    - Fog Lighten (IFS Illusions)
    - Linear Light
    - Luminosity/Shine (SAI)
    - P-Norm A
    - P-Norm B

    - Pin Light
    - Screen
    - Soft Light (Photoshop) & Soft Light SVG
    - Soft Light (IFS Illusions) & Soft Light (Pegtop-Delphi)
    - Super Light
    - Tint (IFS Illusions)
    - Vivid Light
- Misc
    - Bumpmap
    - Combine Normal Map
    - Copy
    - Copy Red, Green, Blue
    - Dissolve
- Mix
    - Allanon
    - Interpolation
    - Interpolation - 2X
    - Alpha Darken
    - Behind
    - Erase
    - Geometric Mean
    - Grain Extract
    - Grain Merge
    - Greater
    - Hard Mix
    - Hard Mix (Photoshop)
    - Hard Overlay
    - Normal
    - Overlay
    - Parallel
    - Penumbra A
    - Penumbra B
    - Penumbra C
    - Penumbra D
- Modulo
    - Divisive Modulo
    - Divisive Modulo - Continuous

    - Modulo
    - Modulo - Continuous
    - Modulo Shift
    - Modulo Shift - Continuous
- Negative
    - Additive Subtractive
    - Arcus Tangent
    - Difference
    - Equivalence
    - Exclusion
    - Negation
- Quadratic
    - Freeze
    - Freeze-Reflect
    - Glow
    - Glow-Heat
    - Heat
    - Heat-Glow
    - Heat-Glow and Freeze-Reflect Hybrid
    - Reflect
    - Reflect-Freeze

See also

Basic blending modes:
https://en.wikipedia.org/wiki/Blend_modes

Grain Extract/Grain Merge:
https://docs.gimp.org/en/gimp-concepts-layer-modes.html

Arithmetic

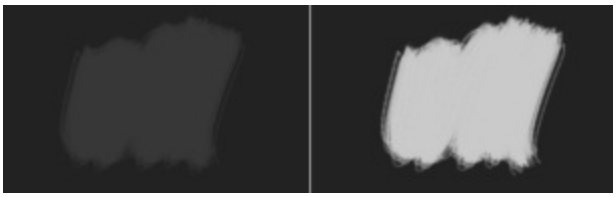
These blending modes are based on simple maths.

Addition

Adds the numerical values of two colors together:

Yellow(1, 1, 0) + Blue(0, 0, 1) = White(1, 1, 1)

Darker Gray(0.4, 0.4, 0.4) + Lighter Gray(0.5, 0.5, 0.5) = Even Lighter Gray (0.9, 0.9, 0.9)



Left: **Normal**. Right: **Addition**.

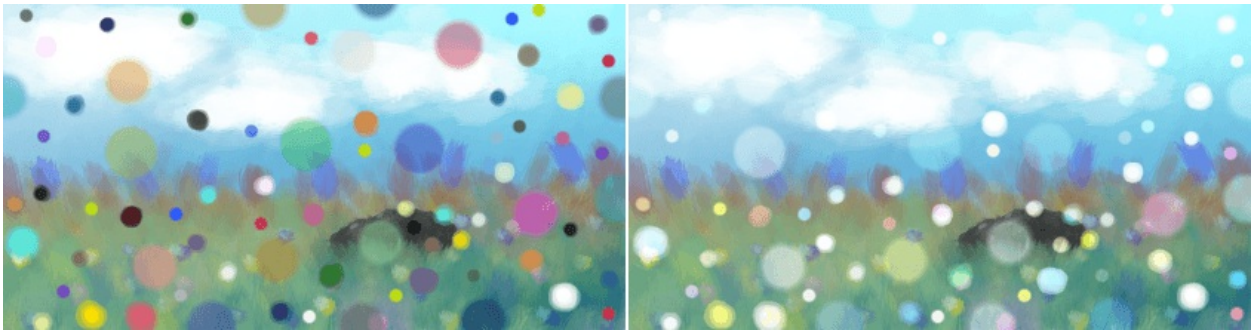
Light Blue(0.1608, 0.6274, 0.8274) + Orange(1, 0.5961, 0.0706) = (1.1608, 1.2235, 0.8980) → Very Light Yellow(1, 1, 0.8980)

Left: **Normal**. Right: **Addition**.

Red(1, 0, 0) + Gray(0.5, 0.5, 0.5) = Pink(1, 0.5, 0.5)

Left: **Normal**. Right: **Addition**.

When the result of the addition is more than 1, white is the color displayed. Therefore, white plus any other color results in white. On the other hand, black plus any other color results in the added color.



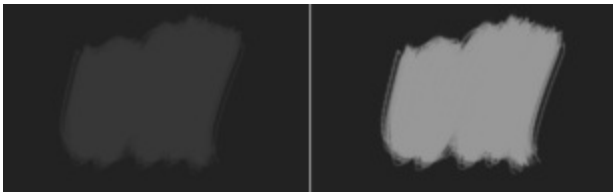
Left: **Normal**. Right: **Addition**.

Divide

Divides the numerical value from the lower color by the upper color.

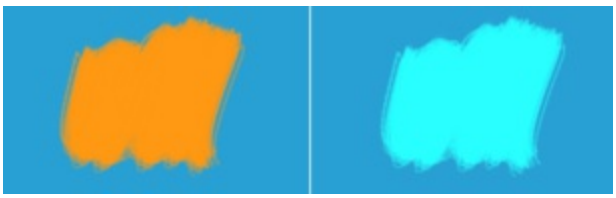
Red(1, 0, 0) / Gray(0.5, 0.5, 0.5) = (2, 0, 0) → Red(1, 0, 0)

Darker Gray(0.4, 0.4, 0.4) / Lighter Gray(0.5, 0.5, 0.5) = Even Lighter Gray (0.8, 0.8, 0.8)

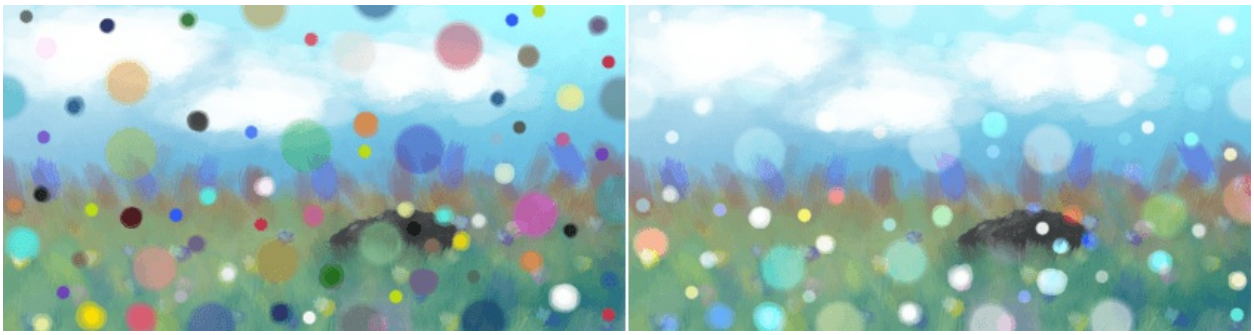


Left: **Normal**. Right: **Divide**.

Light Blue(0.1608, 0.6274, 0.8274) / Orange(1, 0.5961, 0.0706) = (0.1608, 1.0525, 11.7195) → Aqua(0.1608, 1, 1)



Left: **Normal**. Right: **Divide**.



Left: **Normal**. Right: **Divide**.

Inverse Subtract

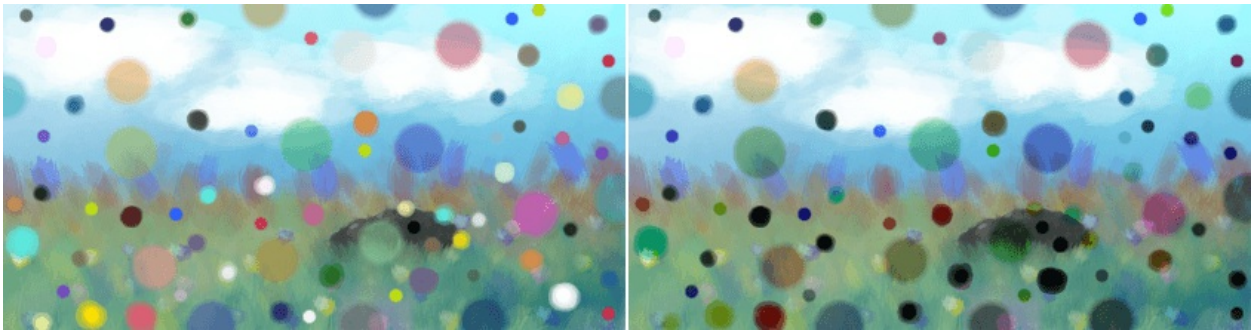
This inverts the lower layer before subtracting it from the upper layer.

Lighter Gray(0.5, 0.5, 0.5)_(1_Darker Gray(0.4, 0.4, 0.4)) = (-0.1, -0.1, -0.1) → Black(0, 0, 0)

Left: **Normal**. Right: **Inverse Subtract**.

Orange(1, 0.5961, 0.0706)_(1_Light Blue(0.1608, 0.6274, 0.8274)) = (0.1608, 0.2235, -0.102) → Dark Green(0.1608, 0.2235, 0)

Left: **Normal**. Right: **Inverse Subtract**.



Left: **Normal**. Right: **Inverse Subtract**.

Multiply

Multiplies the two colors with each other, but does not go beyond the upper limit.

This is often used to color in a black and white lineart. One puts the black and white lineart on top, sets the layer to ‘Multiply’, and then draws in color on a layer beneath. Multiply will allow all the color to go through.

White(1,1,1) x White(1, 1, 1) = White(1, 1, 1)

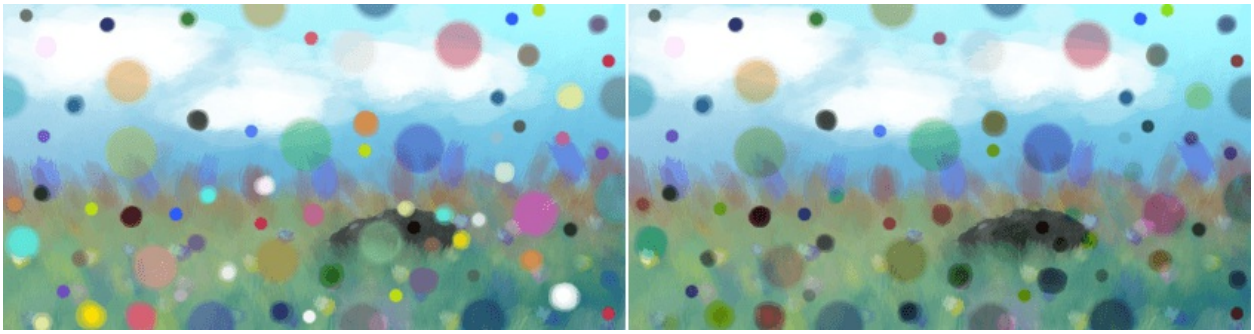
White(1, 1, 1) x Gray(0.5, 0.5, 0.5) = Gray(0.5, 0.5, 0.5)

Darker Gray(0.4, 0.4, 0.4) x Lighter Gray(0.5, 0.5, 0.5) = Even Darker Gray (0.2, 0.2, 0.2)

Left: **Normal**. Right: **Multiply**.

Light Blue(0.1608, 0.6274, 0.8274) x Orange(1, 0.5961, 0.0706) = Green(0.1608, 0.3740, 0.0584)

Left: **Normal**. Right: **Multiply**.



Left: **Normal**. Right: **Multiply**.

Subtract

Subtracts the top layer from the bottom layer.

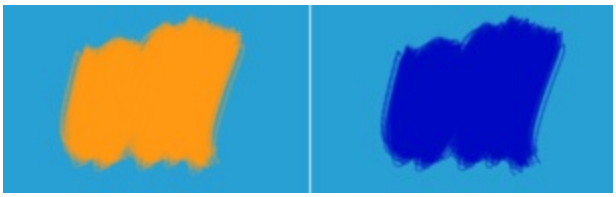
White(1, 1, 1)_White(1, 1, 1) = Black(0, 0, 0)

White(1, 1, 1)_Gray(0.5, 0.5, 0.5) = Gray(0.5, 0.5, 0.5)

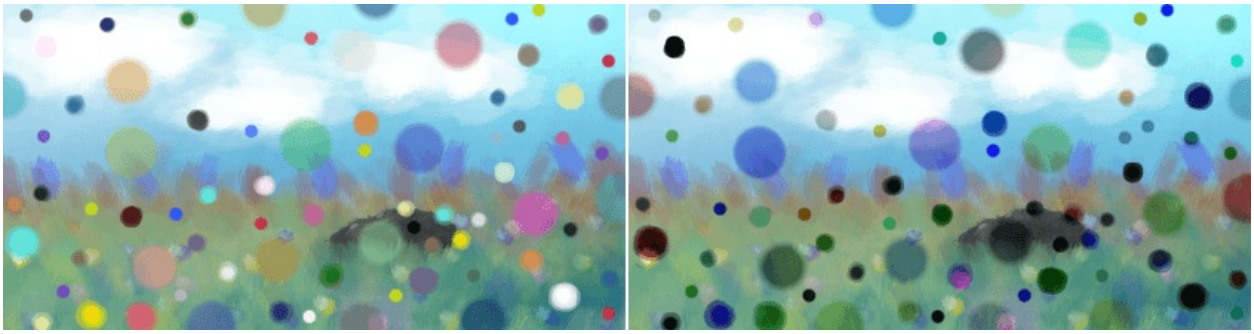
Darker Gray(0.4, 0.4, 0.4)_Lighter Gray(0.5, 0.5, 0.5) = (-0.1, -0.1, -0.1) → Black(0, 0, 0)

Left: **Normal**. Right: **Subtract**.

Light Blue(0.1608, 0.6274, 0.8274) - Orange(1, 0.5961, 0.0706) = (-0.8392, 0.0313, 0.7568) → Blue(0, 0.0313, 0.7568)



Left: **Normal**. Right: **Subtract**.



Left: **Normal**. Right: **Subtract**.

Binary

Binary modes are special class of blending modes which utilizes binary operators for calculations. Binary modes are unlike every other blending modes as these modes have a fractal attribute with falloff similar to other blending modes. Binary modes can be used for generation of abstract art using layers with very smooth surface. All binary modes have capitalized letters to distinguish themselves from other blending modes.

To clarify on how binary modes works, convert decimal values to binary values, then treat 1 or 0 as T or F respectively, and use binary operation to get the end result, and then convert the result back to decimal.

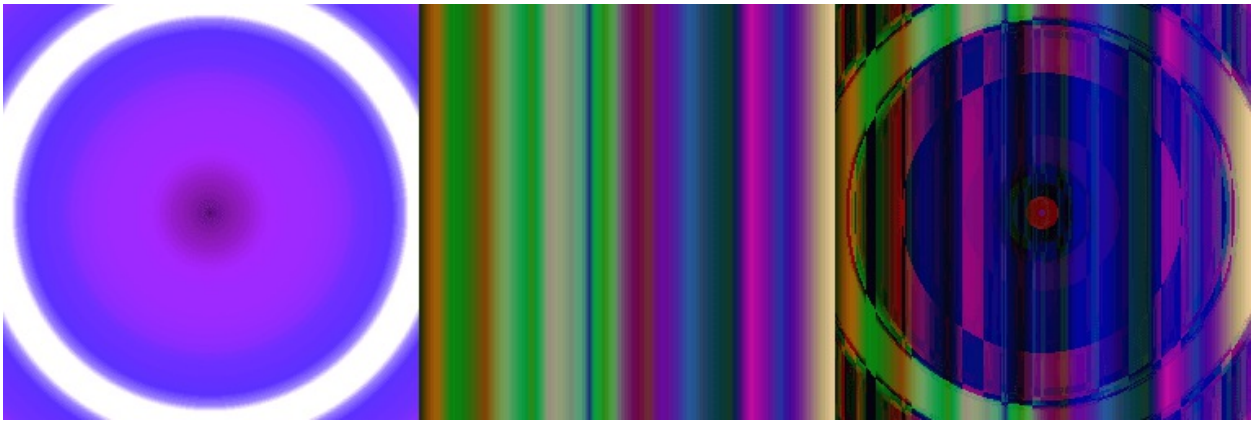
Warning

Binary blending modes do not work on float images or negative numbers! So, don’t report bugs about using binary modes on unsupported color space.

AND

Performs the AND operation for the base and blend layer. Similar to multiply blending mode.

Left: **Base Layer**. Middle: **Blend Layer**. Right: **AND**.

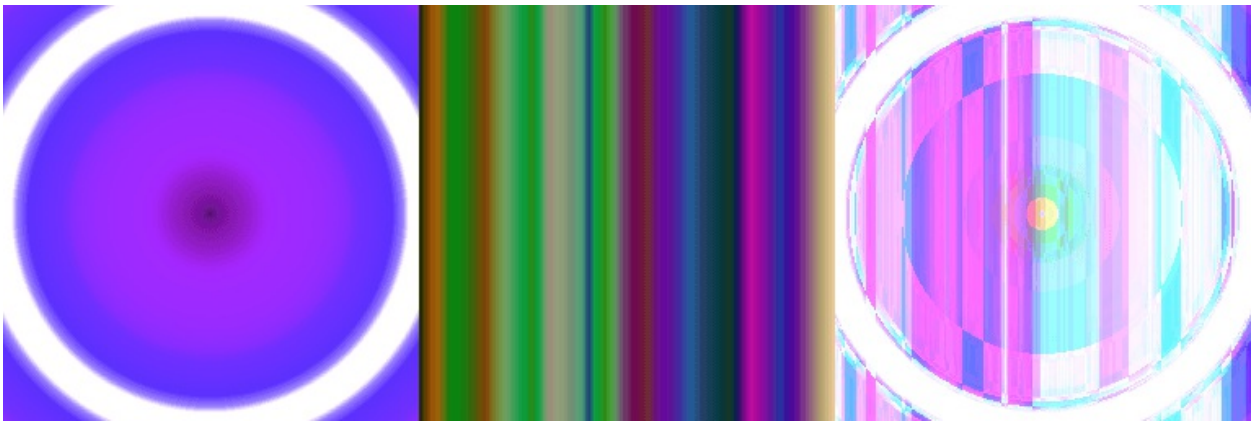


Left: **Base Layer**. Middle: **Blend Layer**. Right: **AND**.

CONVERSE

Performs the inverse of IMPLICATION operation for the base and blend layer. Similar to screen mode with blend layer and base layer inverted.

Left: **Base Layer**. Middle: **Blend Layer**. Right: **CONVERSE**.

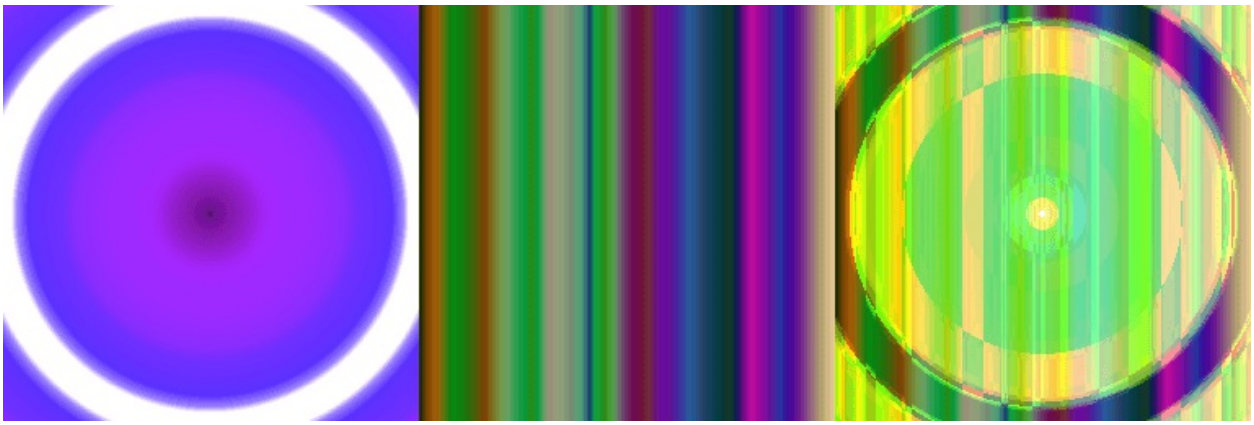


Left: **Base Layer**. Middle: **Blend Layer**. Right: **CONVERSE**.

IMPLICATION

Performs the IMPLICATION operation for the base and blend layer. Similar to screen mode with base layer inverted.

Left: **Base Layer**. Middle: **Blend Layer**. Right: **IMPLICATION**.

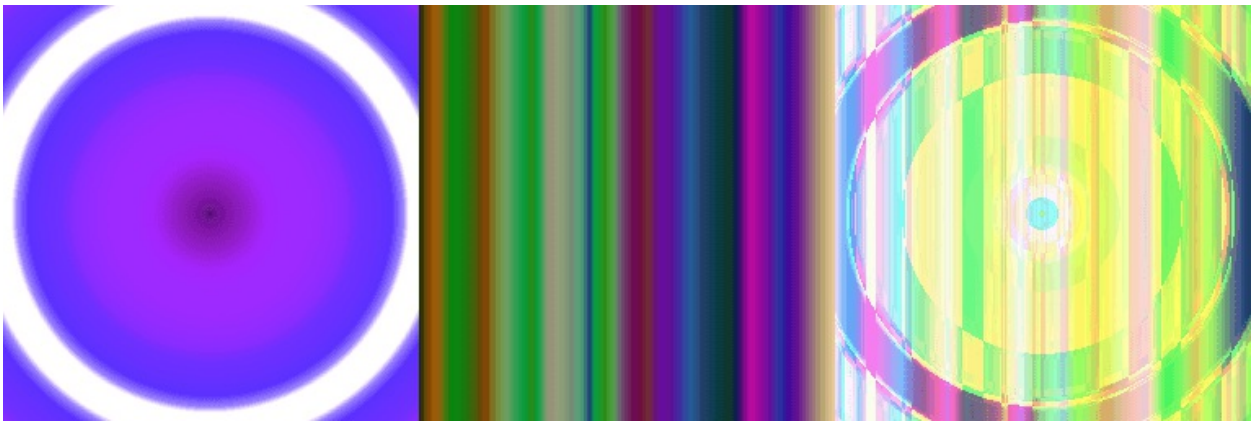


Left: **Base Layer**. Middle: **Blend Layer**. Right: **IMPLICATION**.

NAND

Performs the inverse of AND operation for base and blend layer. Similar to the inverted multiply mode.

Left: **Base Layer**. Middle: **Blend Layer**. Right: **NAND**.

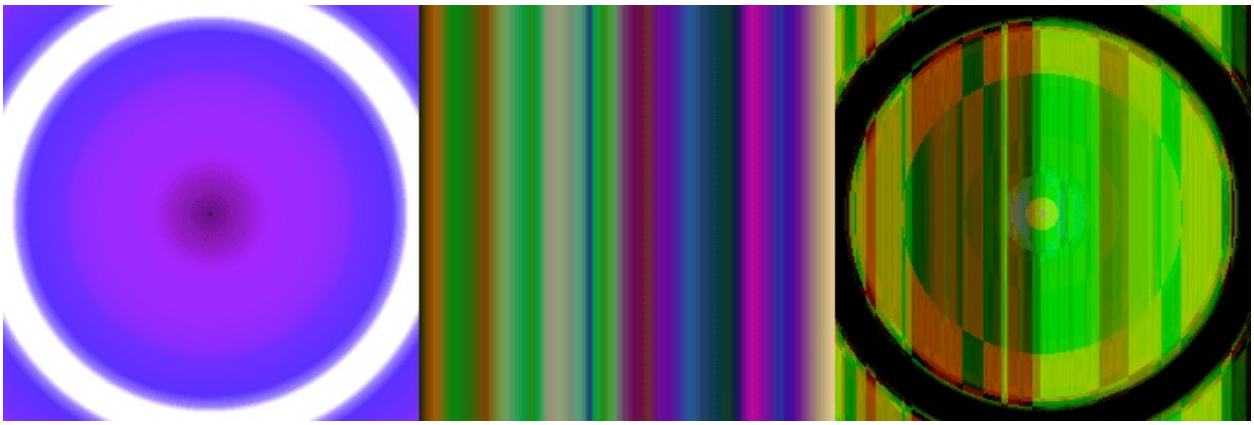


Left: **Base Layer**. Middle: **Blend Layer**. Right: **NAND**.

NOR

Performs the inverse of OR operation for base and blend layer. Similar to the inverted screen mode.

Left: **Base Layer**. Middle: **Blend Layer**. Right: **NOR**.

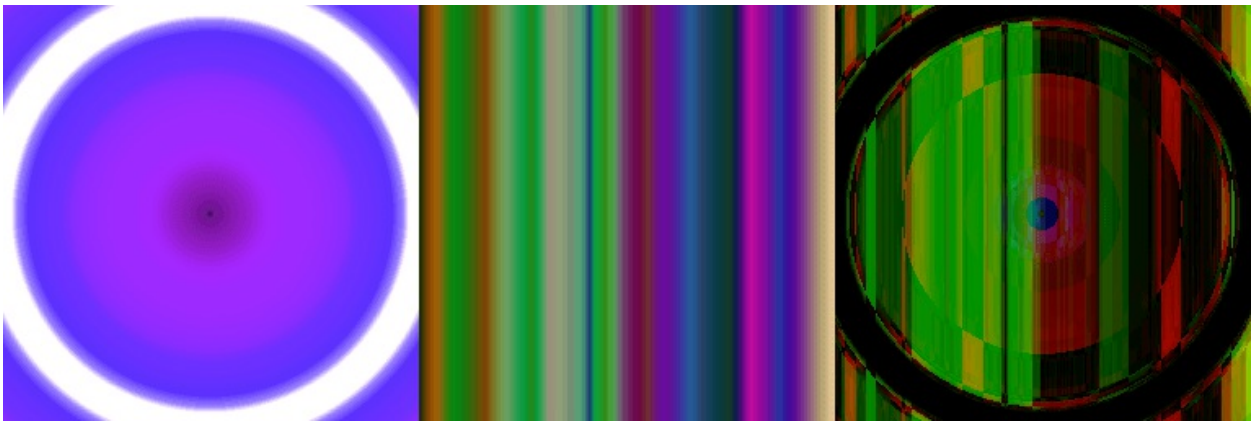


Left: **Base Layer**. Middle: **Blend Layer**. Right: **NOR**.

NOT CONVERSE

Performs the inverse of CONVERSE operation for base and blend layer. Similar to the multiply mode with base layer and blend layer inverted.

Left: **Base Layer**. Middle: **Blend Layer**. Right: **NOT CONVERSE**.

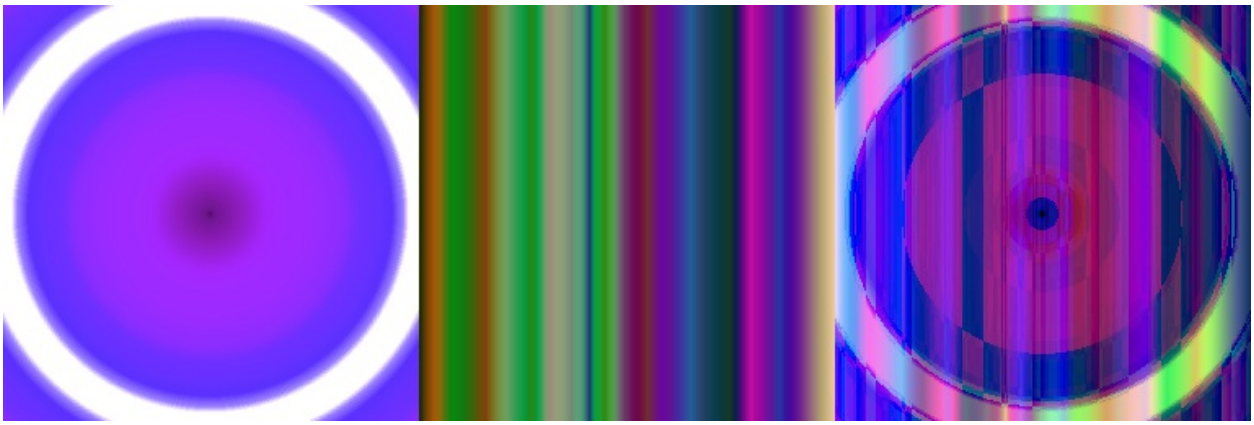


Left: **Base Layer**. Middle: **Blend Layer**. Right: **NOT CONVERSE**.

NOT IMPLICATION

Performs the inverse of IMPLICATION operation for base and blend layer. Similar to the multiply mode with the blend layer inverted.

Left: **Base Layer**. Middle: **Blend Layer**. Right: **NOT IMPLICATION**.

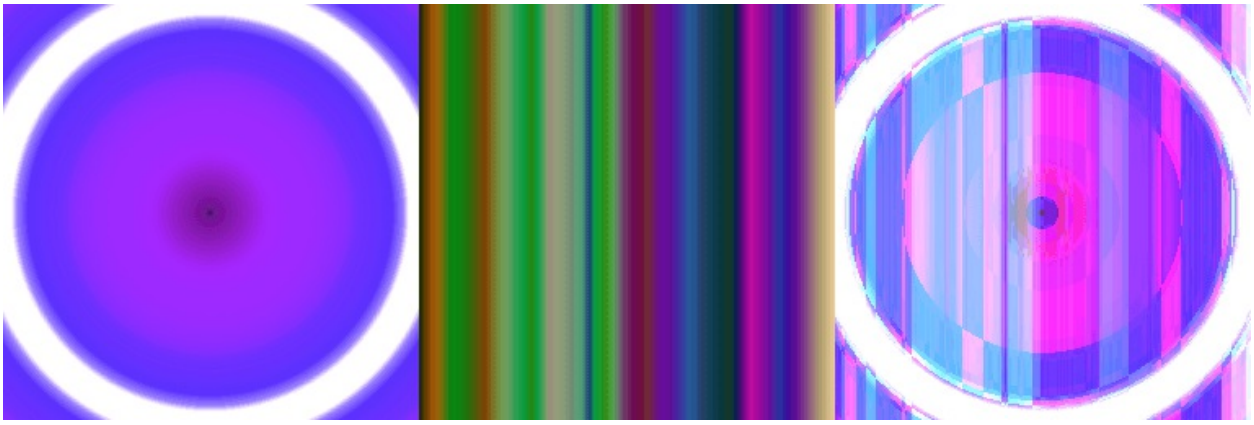


Left: **Base Layer**. Middle: **Blend Layer**. Right: **NOT IMPLICATION**.

OR

Performs the OR operation for base and blend layer. Similar to screen mode.

Left: **Base Layer**. Middle: **Blend Layer**. Right: **OR**.

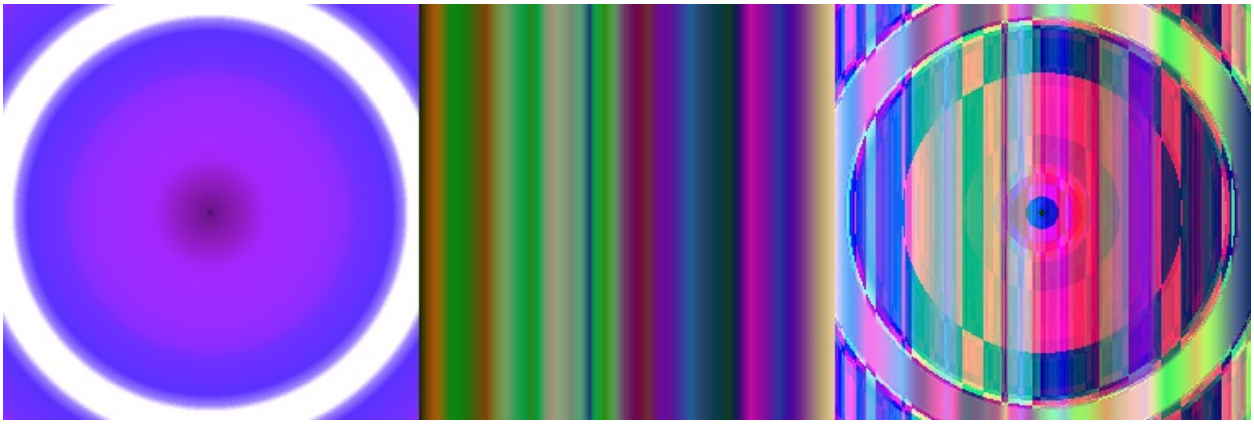


Left: **Base Layer**. Middle: **Blend Layer**. Right: **XOR**.

XOR

Performs the XOR operation for base and blend layer. This mode has a special property that if you duplicate the blend layer twice, you get the base layer.

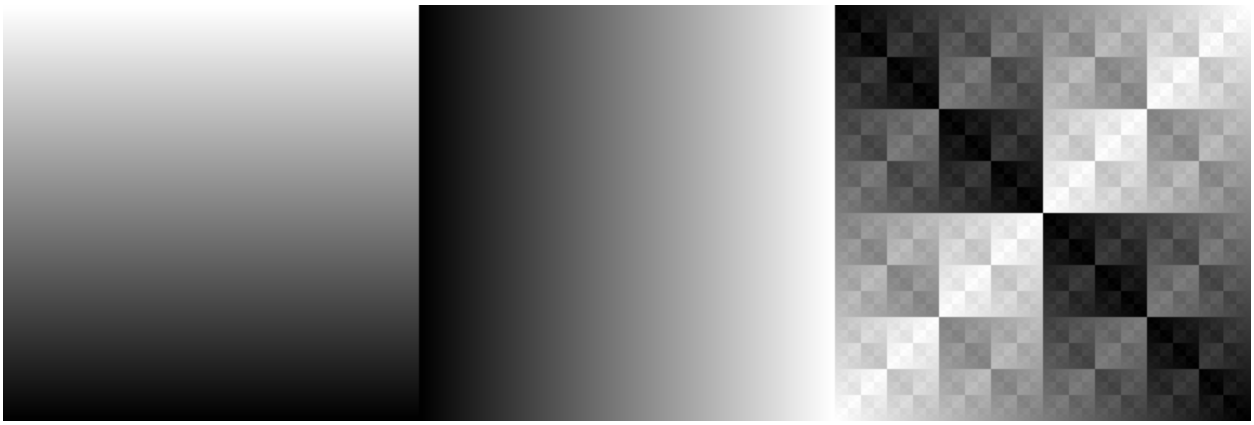
Left: **Base Layer**. Middle: **Blend Layer**. Right: **XOR**.



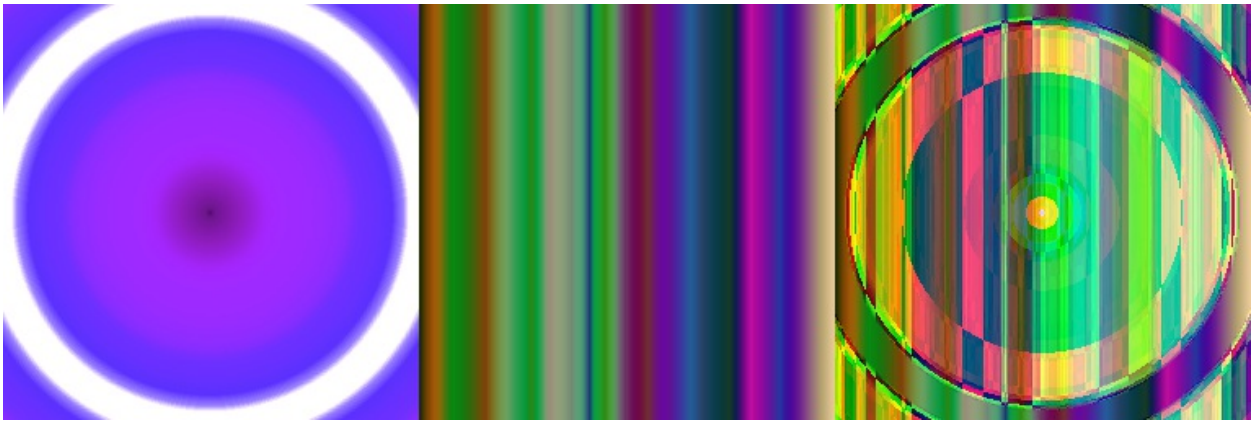
Left: **Base Layer**. Middle: **Blend Layer**. Right: **XOR**.

XNOR

Performs the XNOR operation for base and blend layer. This mode has a special property that if you duplicate the blend layer twice, you get the base layer.



Left: **Base Layer**. Middle: **Blend Layer**. Right: **XNOR**.



Left: **Base Layer**. Middle: **Blend Layer**. Right: **XNOR**.

Darken

Burn

A variation on Divide, sometimes called ‘Color Burn’ in some programs.

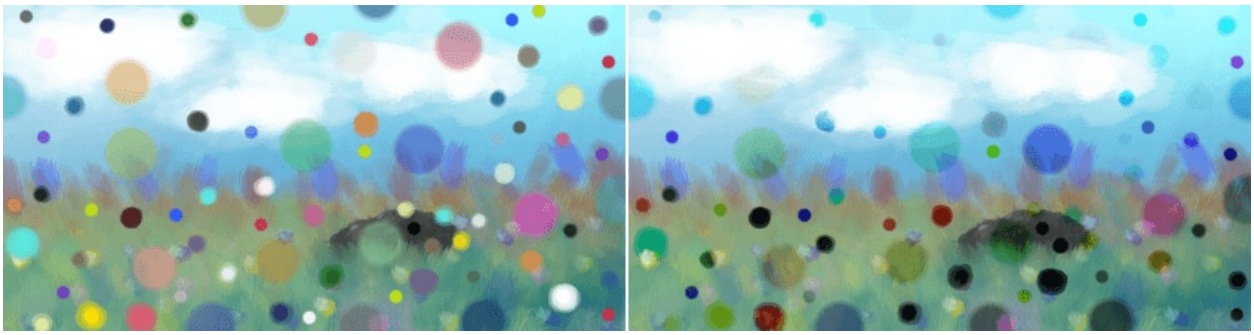
This inverts the bottom layer, then divides it by the top layer, and inverts the result. This results in a darkened effect that takes the colors of the lower layer into account, similar to the burn technique used in traditional darkroom photography.

1_{[1_Darker Gray(0.4, 0.4, 0.4)] / Lighter Gray(0.5, 0.5, 0.5)} = (-0.2, -0.2, -0.2) → Black(0, 0, 0)

Left: **Normal**. Right: **Burn**.

1_{[1_Light Blue(0.1608, 0.6274, 0.8274)] / Orange(1, 0.5961, 0.0706)} = (0.1608, 0.3749, -1.4448) → Green(0.1608, 0.3749, 0)

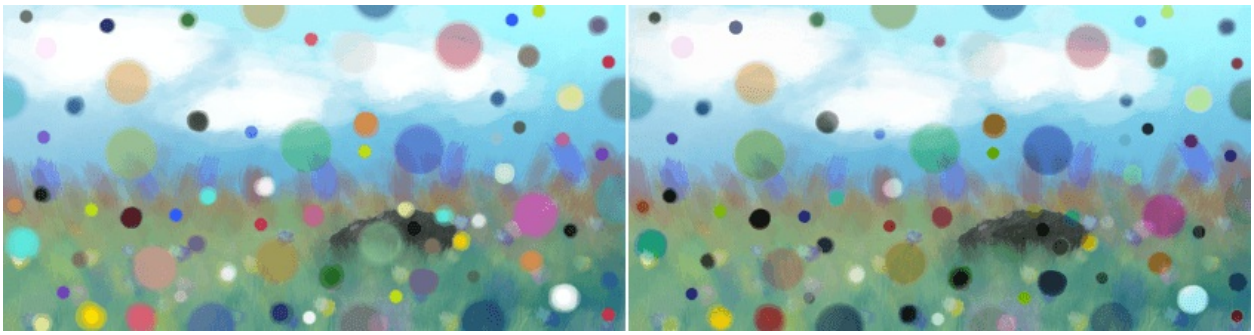
Left: **Normal**. Right: **Burn**.



Left: **Normal**. Right: **Burn**.

Easy Burn

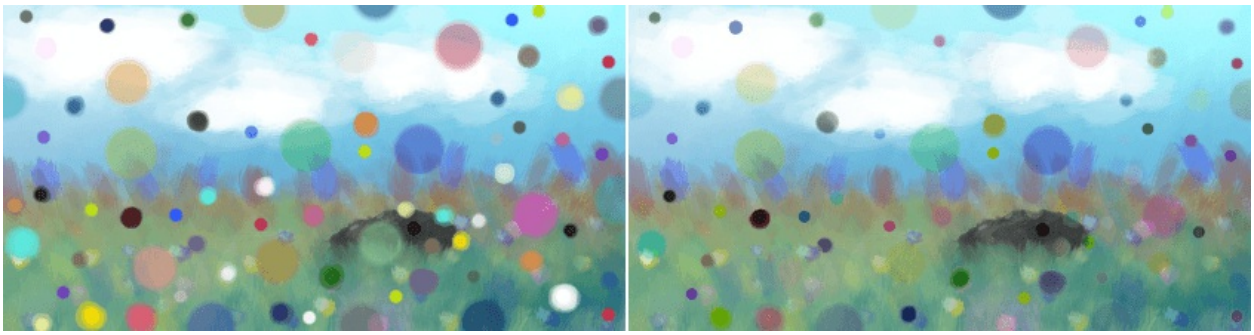
Aims to solve issues with Color Burn blending mode by using a formula which falloff is similar to Dodge, but the falloff rate is softer. It is within the range of 0.0f and 1.0f unlike Color Burn mode.



Left: **Normal**. Right: **Easy Burn**.

Fog Darken (IFS Illusions)

Darken the image in a way that there is a ‘fog’ in the end result. This is due to the unique property of fog darken in which midtones combined are lighter than non-midtones blend.



Left: **Normal**. Right: **Fog Darken** (exactly the same as Addition).

Darken

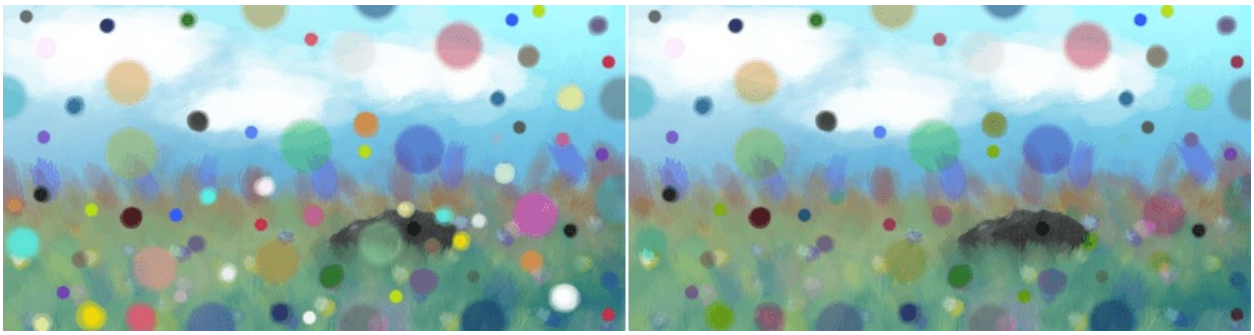
With the darken, the upper layer's colors are checked for their lightness. Only if they are darker than the underlying color on the lower layer, will they be visible.

Is Lighter Gray(0.5, 0.5, 0.5) darker than Darker Gray(0.4, 0.4, 0.4)? = (no, no, no) → Darker Gray(0.4, 0.4, 0.4)

Left: **Normal**. Right: **Darken**.

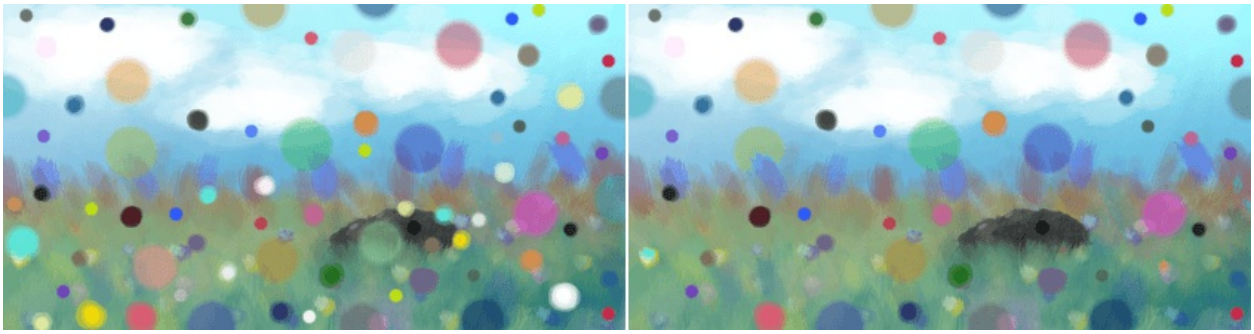
Is Orange(1, 0.5961, 0.0706) darker than Light Blue(0.1608, 0.6274, 0.8274)? = (no, yes, yes) → Green(0.1608, 0.5961, 0.0706)

Left: **Normal**. Right: **Darken**.



Left: **Normal**. Right: **Darken**.

Darker Color



Left: **Normal**. Right: **Darker Color**.

Gamma Dark

Divides 1 by the upper layer, and calculates the end result using that as the power of the lower layer.

Darker Gray(0.4, 0.4, 0.4)^[1 / Lighter Gray(0.5, 0.5, 0.5)] = Even Darker Gray(0.1600, 0.1600, 0.1600)

Left: **Normal**. Right: **Gamma Dark**.

Light Blue(0.1608, 0.6274, 0.8274)^[1 / Orange(1, 0.5961, 0.0706)] = Green(0.1608, 0.4575, 0.0683)

Left: **Normal**. Right: **Gamma Dark**.

Left: **Normal**. Right: **Gamma Dark**.

Linear Burn

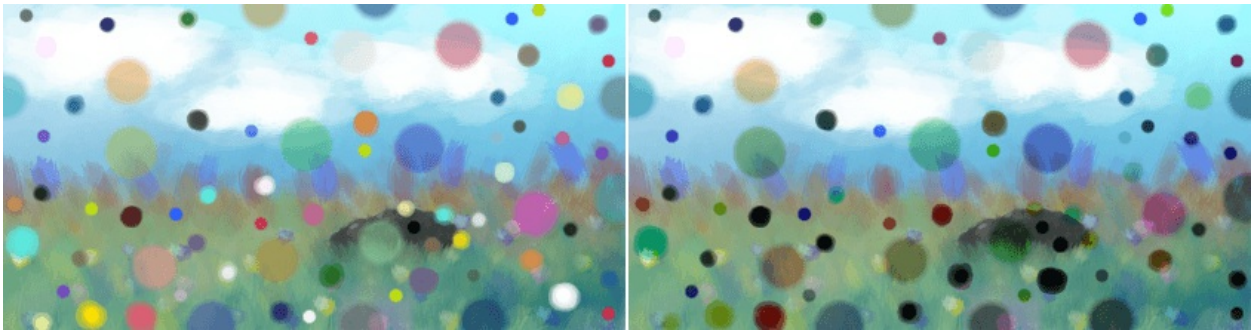
Adds the values of the two layers together and then subtracts 1. Seems to produce the same result as Inverse Subtract.

[Darker Gray(0.4, 0.4, 0.4) + Lighter Gray(0.5, 0.5, 0.5)]_1 = (-0.1000, -0.1000, -0.1000) → Black(0, 0, 0)

Left: **Normal**. Right: **Linear Burn**.

[Light Blue(0.1608, 0.6274, 0.8274) + Orange(1, 0.5961, 0.0706)]_1 = (0.1608, 0.2235, -0.1020) → Dark Green(0.1608, 0.2235, 0)

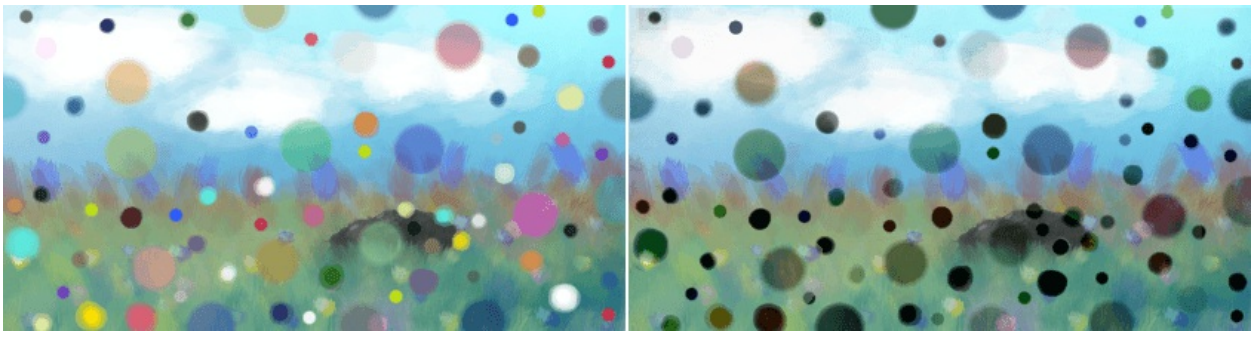
Left: **Normal**. Right: **Linear Burn**.



Left: **Normal**. Right: **Linear Burn**.

Shade (IFS Illusions)

Basically, the blending mode only ends in shades of shades. This means that it’s very useful for painting shading colors while still in the range of shades.



Left: **Normal**. Right: **Shade**.

HSX

Krita has four different HSX coordinate systems. The difference between them is how they handle tone.

HSI

HSI is a color coordinate system, using Hue, Saturation and Intensity to categorize a color. Hue is roughly the wavelength, whether the color is red, yellow, green, cyan, blue or purple. It is measure in 360°, with 0 being red. Saturation is the measurement of how close a color is to gray. Intensity, in this case is the tone of the color. What makes intensity special is that it recognizes yellow (rgb:1,1,0) having a higher combined rgb value than blue (rgb:0,0,1). This is a non-linear tone dimension, which means it's gamma-corrected.

HSL

HSL is also a color coordinate system. It describes colors in Hue, Saturation and Lightness. Lightness specifically puts both yellow (rgb:1,1,0), blue (rgb:0,0,1) and middle gray (rgb:0.5,0.5,0.5) at the same lightness (0.5).

HSV

HSV, occasionally called HSB, is a color coordinate system. It measures colors in Hue, Saturation, and Value (also called Brightness). Value or Brightness specifically refers to strength at which the pixel-lights on your monitor have to shine. It sets Yellow (rgb:1,1,0), Blue (rgb:0,0,1) and White (rgb:1,1,0) at the same Value (100%).

HSY

HSY is a color coordinate system. It categorizes colors in Hue, Saturation

and Luminosity. Well, not really, it uses Luma instead of true luminosity, the difference being that Luminosity is linear while Luma is gamma-corrected and just weights the rgb components. Luma is based on scientific studies of how much light a color reflects in real-life. While like intensity it acknowledges that yellow (rgb:1,1,0) is lighter than blue (rgb:0,0,1), it also acknowledges that yellow (rgb:1,1,0) is lighter than cyan (rgb:0,1,1), based on these studies.

HSX Blending Modes

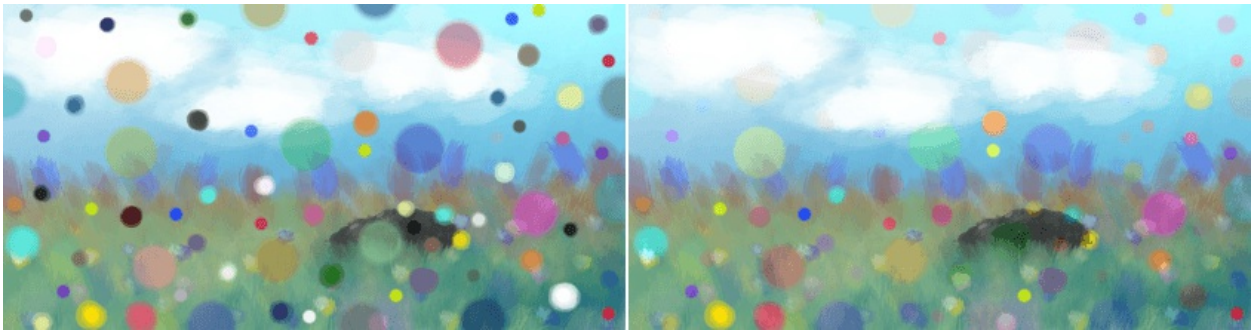
Color, HSV, HSI, HSL, HSY

This takes the Luminosity/Value/Intensity/Lightness of the colors on the lower layer, and combines them with the Saturation and Hue of the upper pixels. We refer to Color HSY as ‘Color’ in line with other applications.

Left: **Normal**. Right: **Color HSI**.

Left: **Normal**. Right: **Color HSI**.

Left: **Normal**. Right: **Color HSI**.



Left: **Normal**. Right: **Color HSL**.

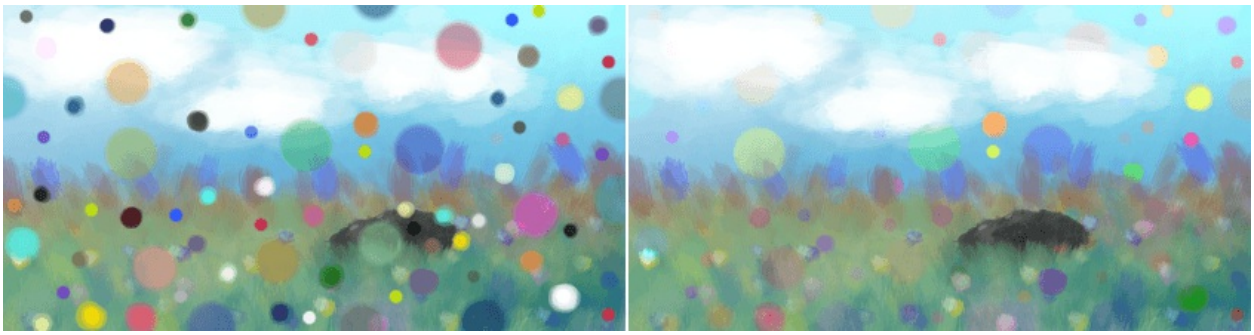
Left: **Normal**. Right: **Color HSV**.

Left: **Normal**. Right: **Color**.

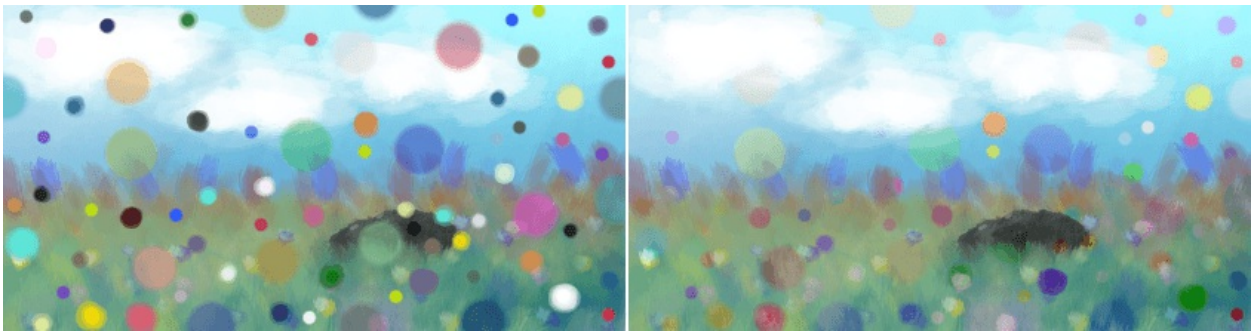
Hue HSV, HSI, HSL, HSY

Takes the saturation and tone of the lower layer and combines them with the hue of the upper-layer. Tone in this case being either Value, Lightness, Intensity or Luminosity.

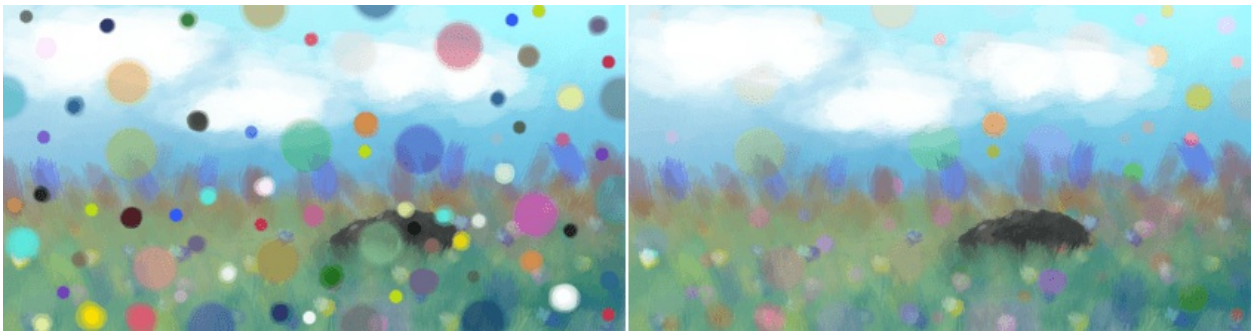
Left: **Normal**. Right: **Hue HSI**.



Left: **Normal**. Right: **Hue HSL**.



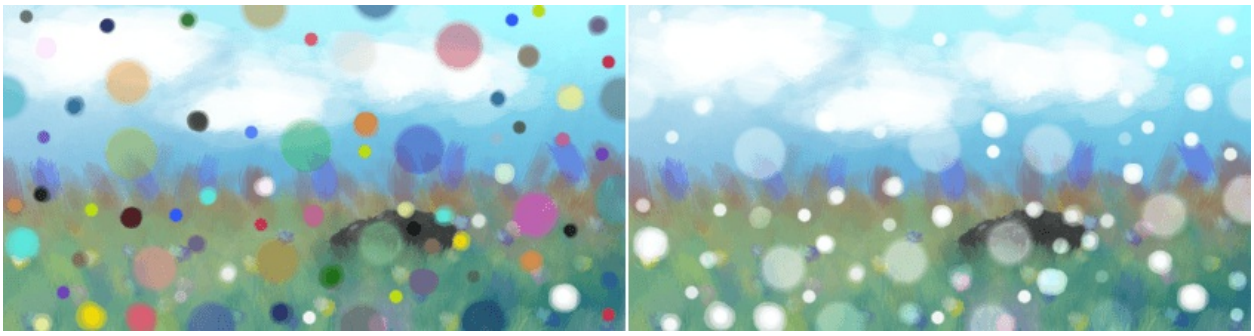
Left: **Normal**. Right: **Hue HSV**.



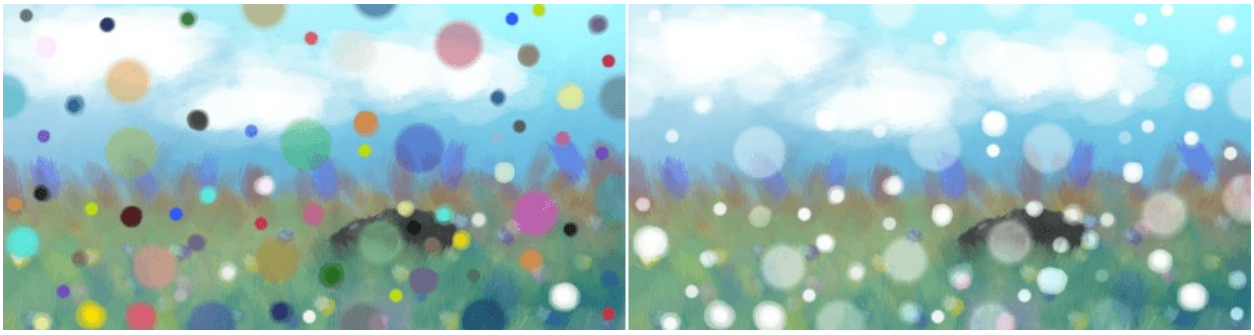
Left: **Normal**. Right: **Hue**.

Increase Value, Lightness, Intensity or Luminosity.

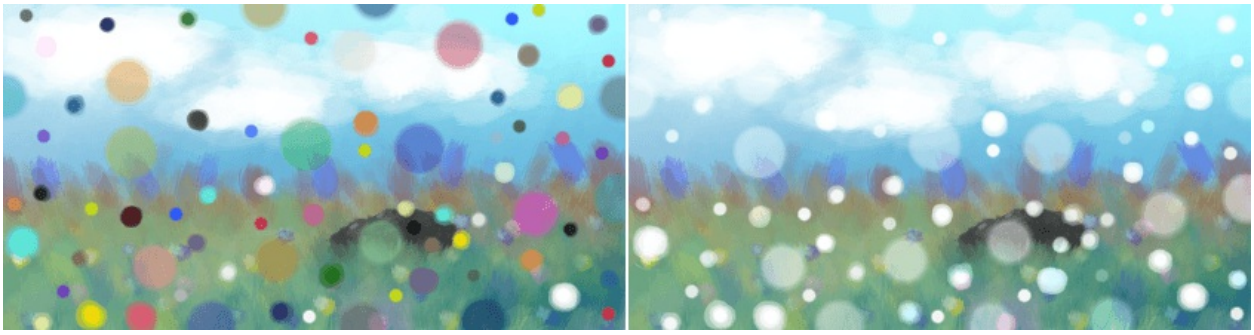
Similar to lighten, but specific to tone. Checks whether the upper layer's pixel has a higher tone than the lower layer's pixel. If so, the tone is increased, if not, the lower layer's tone is maintained.



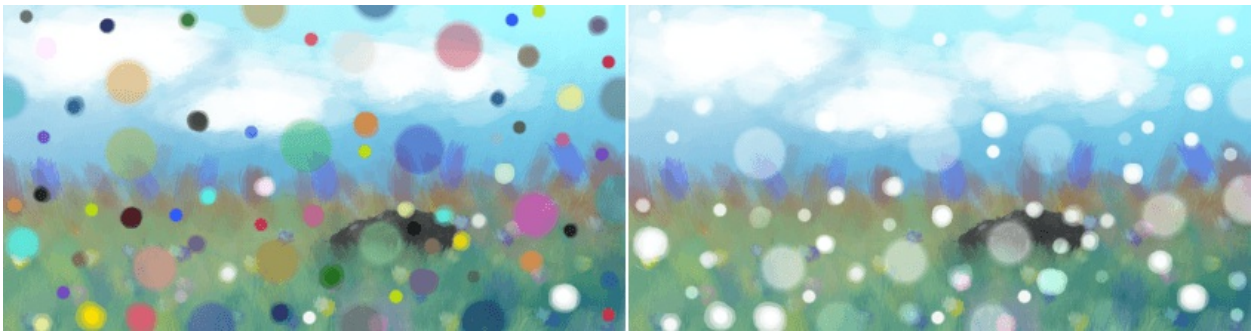
Left: **Normal**. Right: **Increase Intensity**.



Left: **Normal**. Right: **Increase Lightness**.



Left: **Normal**. Right: **Increase Value**.



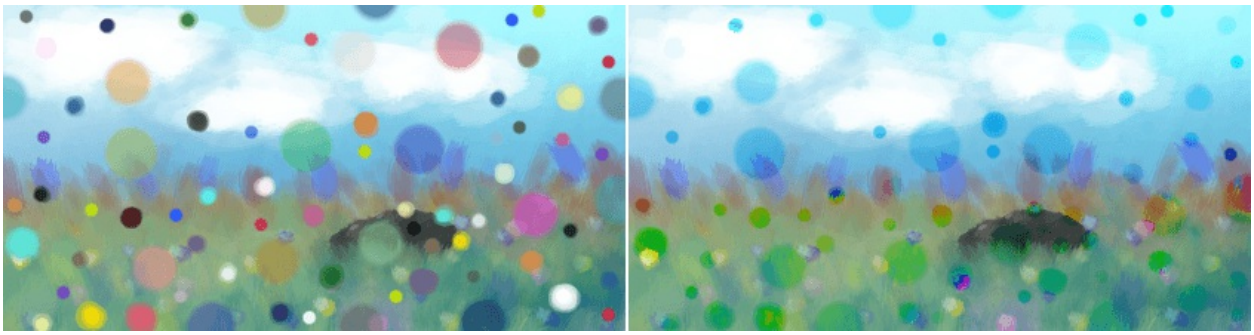
Left: **Normal**. Right: **Increase Luminosity**.

Increase Saturation HSI, HSV, HSL, HSY

Similar to lighten, but specific to Saturation. Checks whether the upper layer's pixel has a higher Saturation than the lower layer's pixel. If so, the Saturation is increased, if not, the lower layer's Saturation is maintained.

Left: **Normal**. Right: **Increase Saturation HSI**.

Left: **Normal**. Right: **Increase Saturation HSL**.

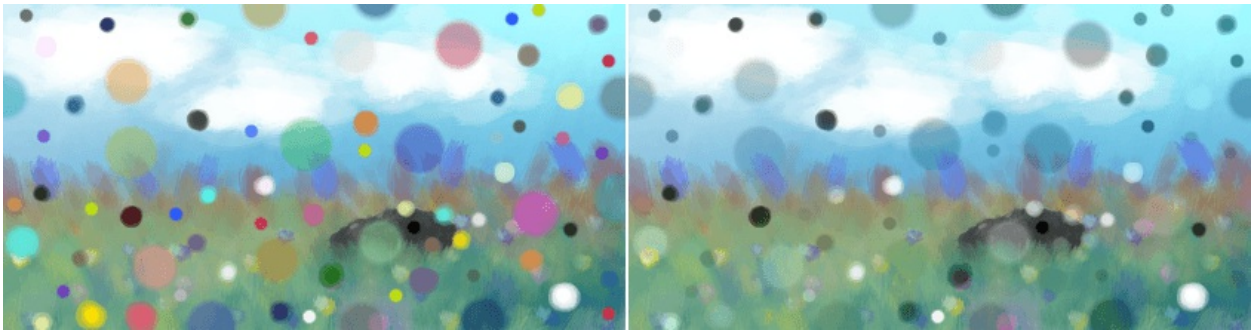


Left: **Normal**. Right: **Increase Saturation HSV**.

Left: **Normal**. Right: **Increase Saturation**.

Intensity

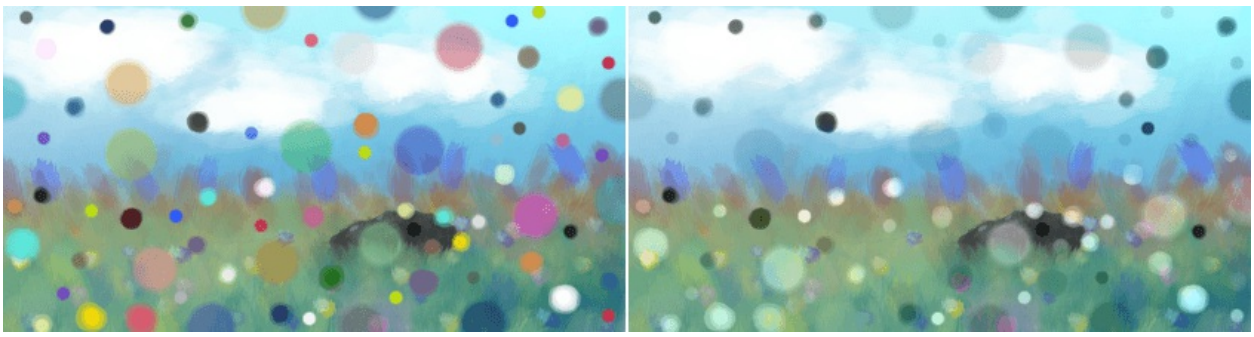
Takes the Hue and Saturation of the Lower layer and outputs them with the intensity of the upper layer.



Left: **Normal**. Right: **Intensity**.

Value

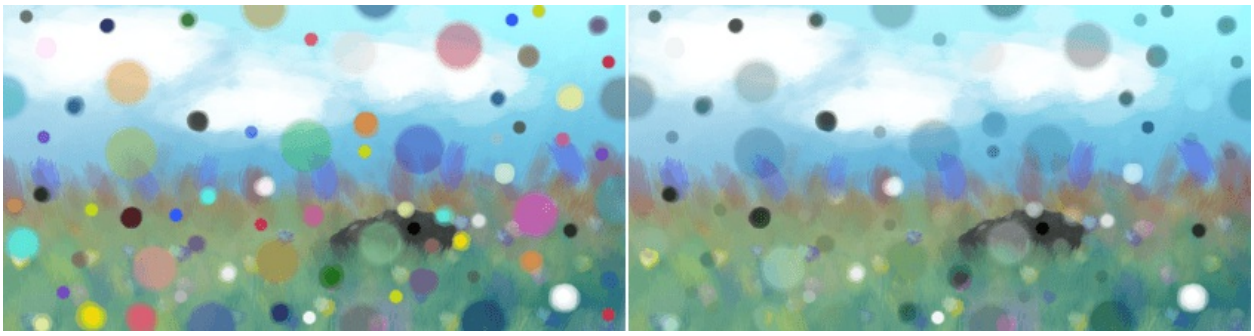
Takes the Hue and Saturation of the Lower layer and outputs them with the Value of the upper layer.



Left: **Normal**. Right: **Value**.

Lightness

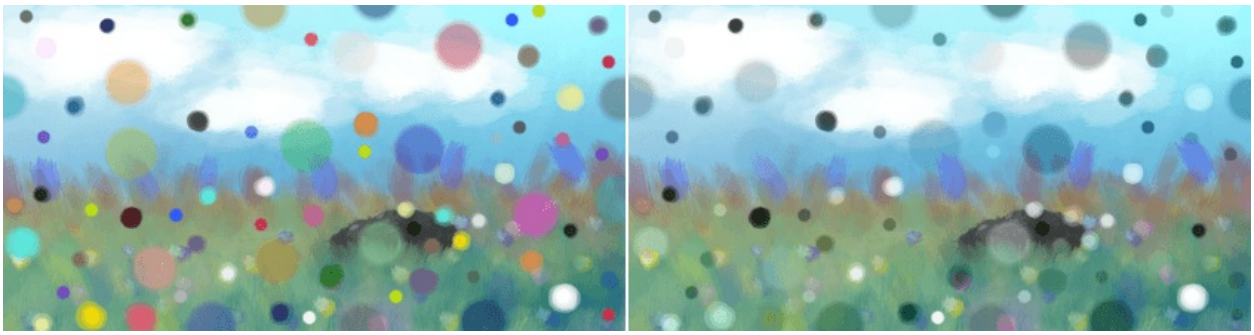
Takes the Hue and Saturation of the Lower layer and outputs them with the Lightness of the upper layer.



Left: **Normal**. Right: **Lightness**.

Luminosity

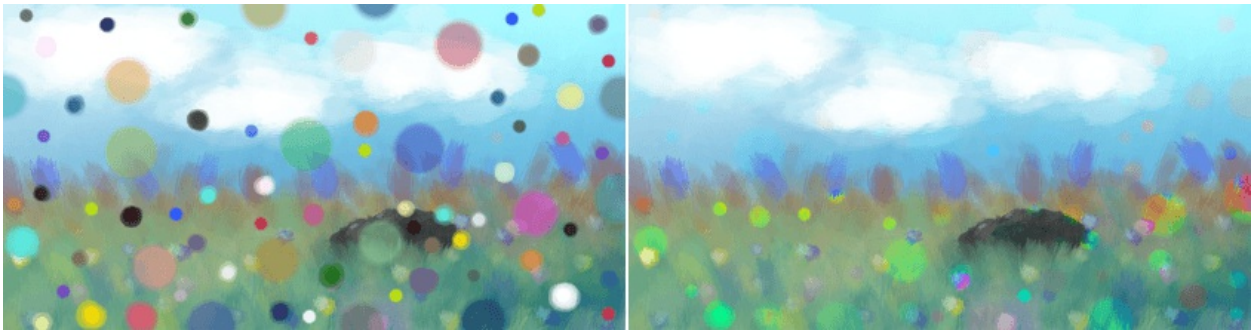
As explained above, actually Luma, but called this way as it’s in line with the terminology in other applications. Takes the Hue and Saturation of the Lower layer and outputs them with the Luminosity of the upper layer. The most preferred one of the four Tone blending modes, as this one gives fairly intuitive results for the Tone of a hue.



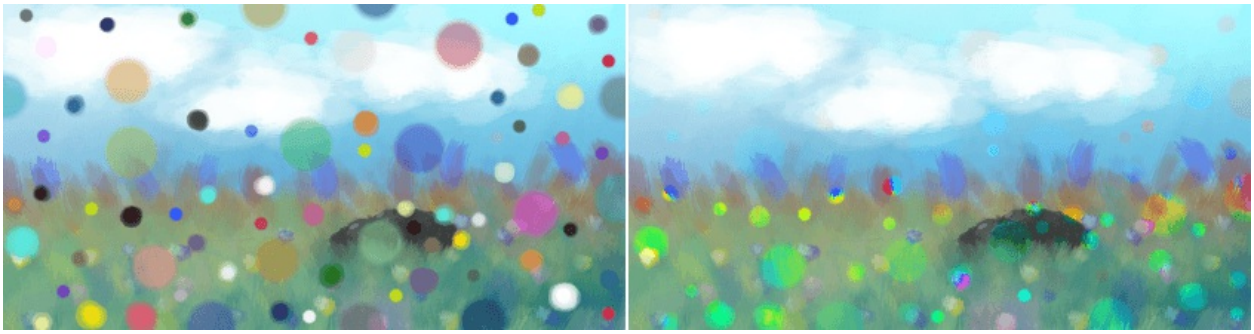
Left: **Normal**. Right: **Luminosity**.

Saturation HSI, HSV, HSL, HSY

Takes the Intensity and Hue of the lower layer, and outputs them with the HSI saturation of the upper layer.



Left: **Normal**. Right: **Saturation HSI**.



Left: **Normal**. Right: **Saturation HSL**.

Left: **Normal**. Right: **Saturation HSV**.

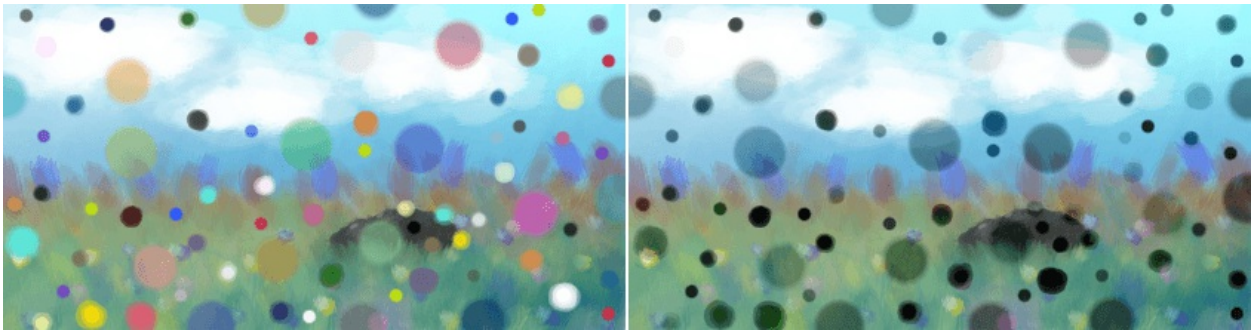
Left: **Normal**. Right: **Saturation**.

Decrease Value, Lightness, Intensity or Luminosity

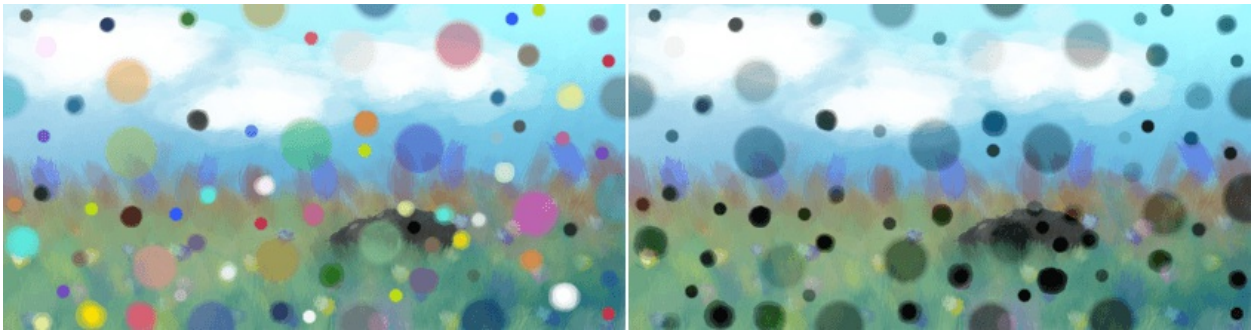
Similar to darken, but specific to tone. Checks whether the upper layer's pixel has a lower tone than the lower layer's pixel. If so, the tone is decreased, if not, the lower layer's tone is maintained.

Left: **Normal**. Right: **Decrease Intensity**.

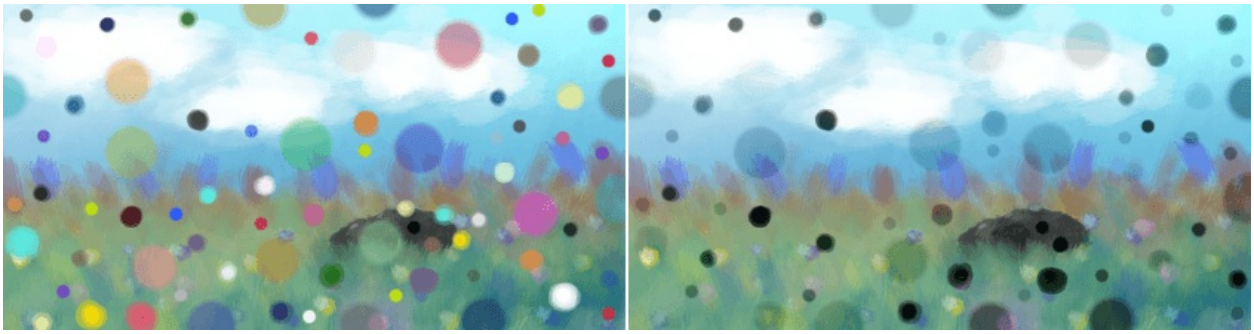
Left: **Normal**. Right: **Decrease Intensity**.



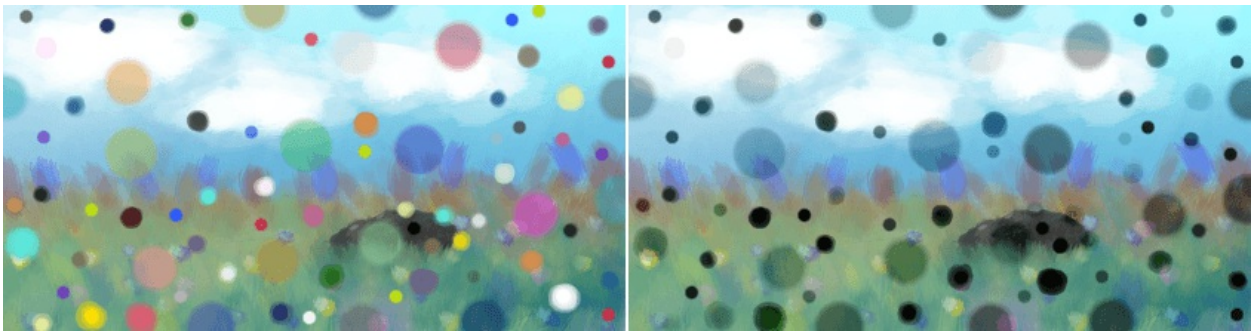
Left: **Normal**. Right: **Decrease Intensity**.



Left: **Normal**. Right: **Decrease Lightness**.



Left: **Normal**. Right: **Decrease Value**.



Left: **Normal**. Right: **Decrease Luminosity**.

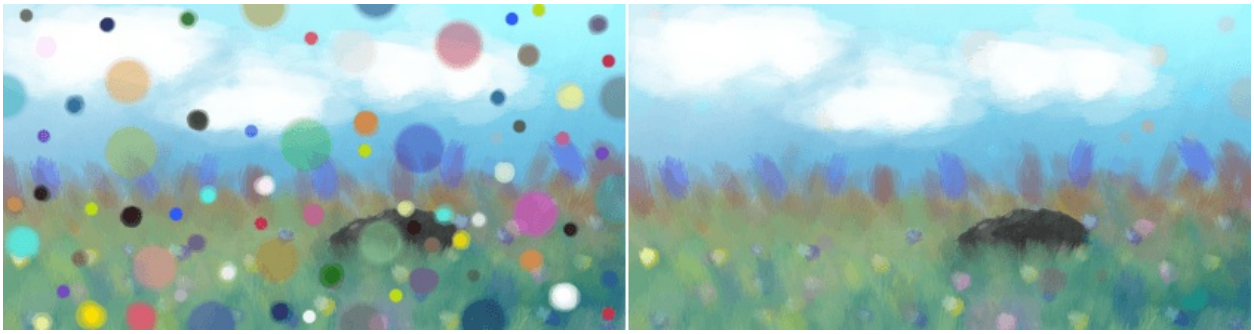
Decrease Saturation HSI, HSV, HSL, HSY

Similar to darken, but specific to Saturation. Checks whether the upper layer's pixel has a lower Saturation than the lower layer's pixel. If so, the Saturation is decreased, if not, the lower layer's Saturation is maintained.

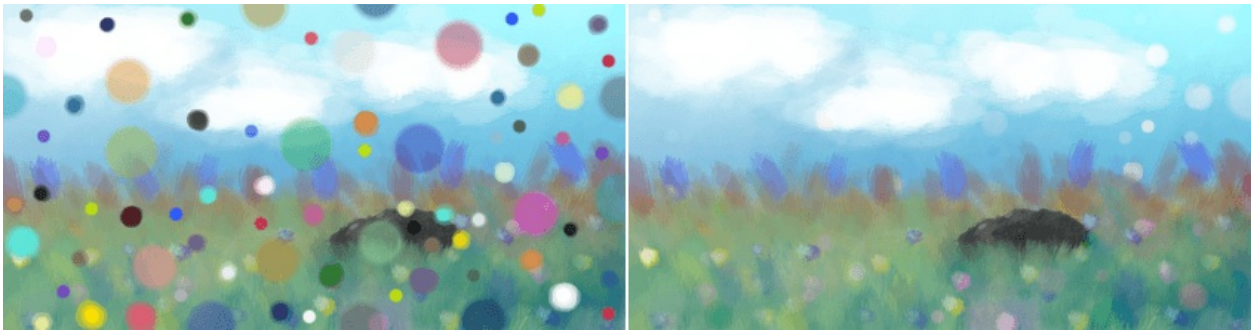
Left: **Normal**. Right: **Decrease Saturation HSI**.

Left: **Normal**. Right: **Decrease Saturation HSI**.

Left: **Normal**. Right: **Decrease Saturation HSI**.



Left: **Normal**. Right: **Decrease Saturation HSL**.



Left: **Normal**. Right: **Decrease Saturation HSV**.

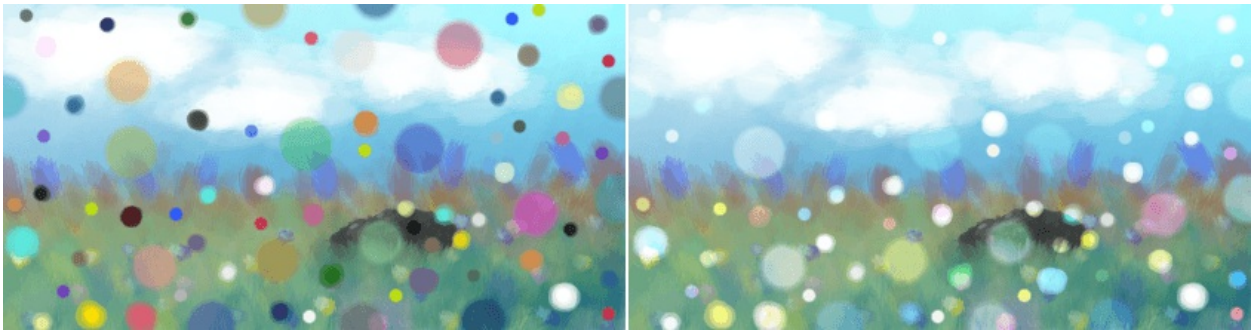
Left: **Normal**. Right: **Decrease Saturation**.

Lighten

Blending modes that lighten the image.

Color Dodge

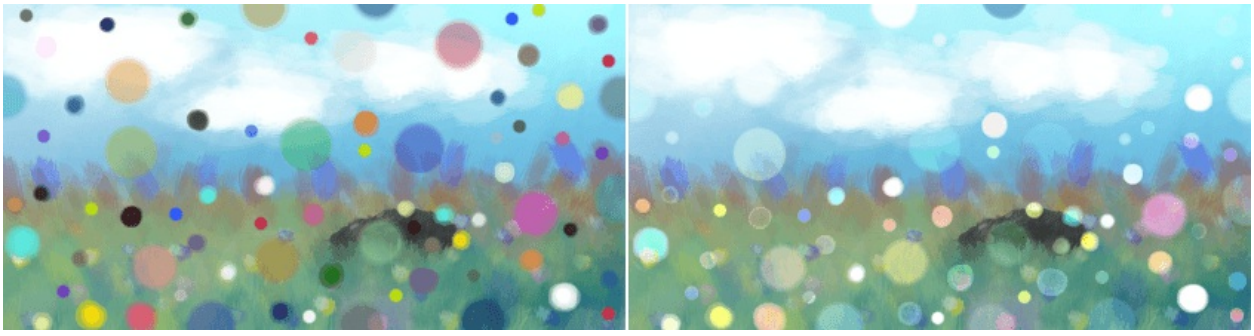
Similar to Divide. Inverts the top layer, and divides the lower layer by the inverted top layer. This results in a image with emphasized highlights, like Dodging would do in traditional darkroom photography.



Left: **Normal**. Right: **Color Dodge**.

Gamma Illumination

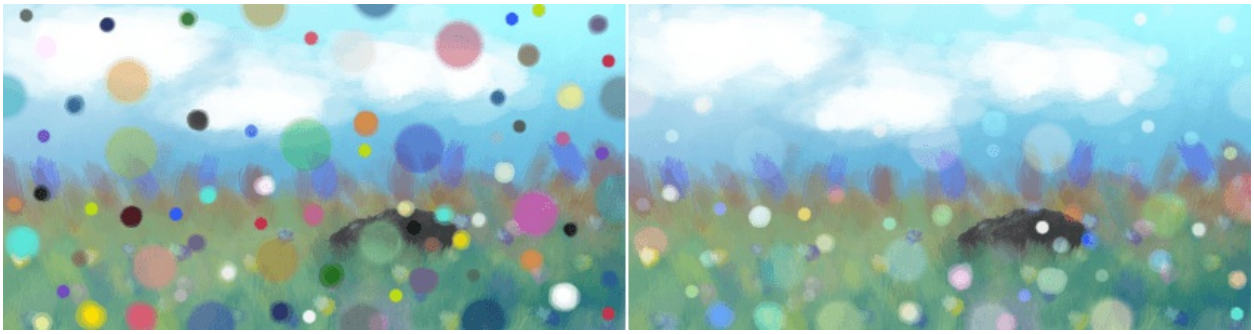
Inverted Gamma Dark blending mode.



Left: **Normal**. Right: **Gamma Illumination**.

Gamma Light

Outputs the upper layer as power of the lower layer.



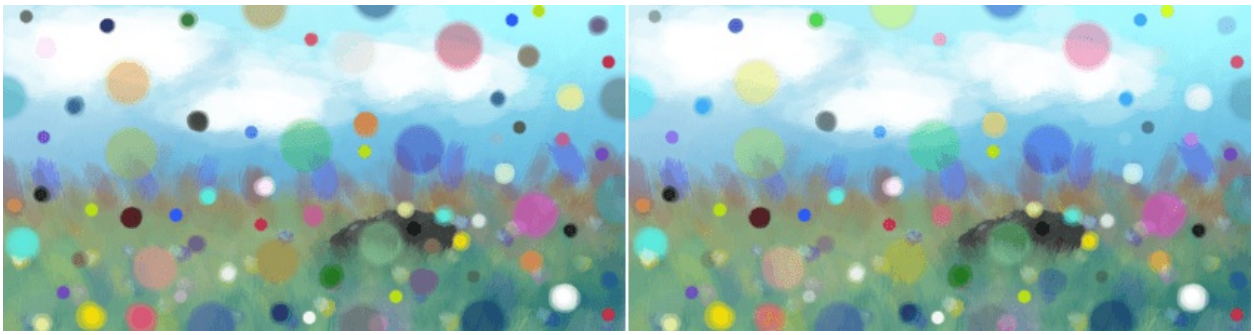
Left: **Normal**. Right: **Gamma Light**.

Hard Light

Similar to Overlay. A combination of the Multiply and Screen blending modes, switching between both at a middle-lightness.

Hard light checks if the color on the upperlayer has a lightness above 0.5. Unlike overlay, if the pixel is lighter than 0.5, it is blended like in Multiply mode, if not the pixel is blended like in Screen mode.

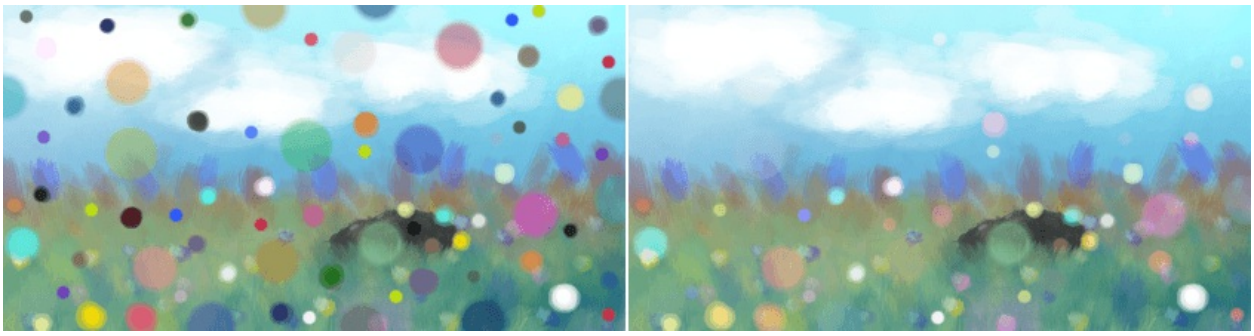
Effectively, this decreases contrast.



Left: **Normal**. Right: **Hard Light**.

Lighten

With the darken, the upper layer’s colors are checked for their lightness. Only if they are Lighter than the underlying color on the lower layer, will they be visible.



Left: **Normal**. Right: **Lighten**.

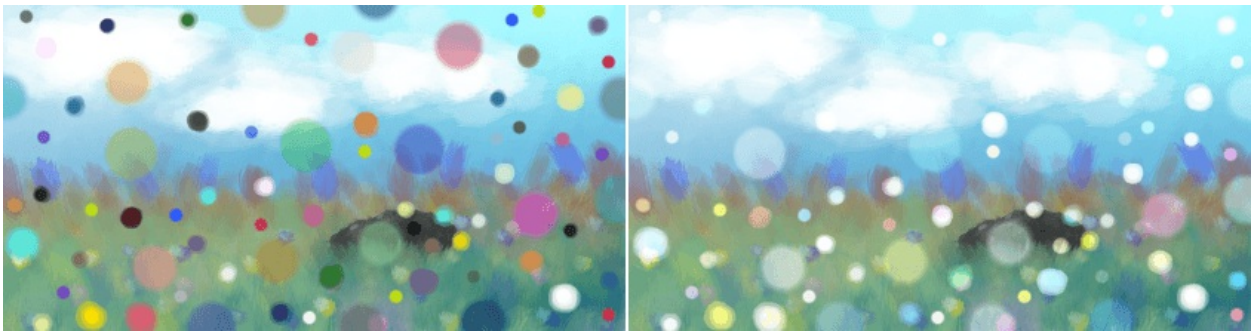
Lighter Color

Left: **Normal**. Right: **Lighter Color**.

Linear Dodge

Exactly the same as Addition.

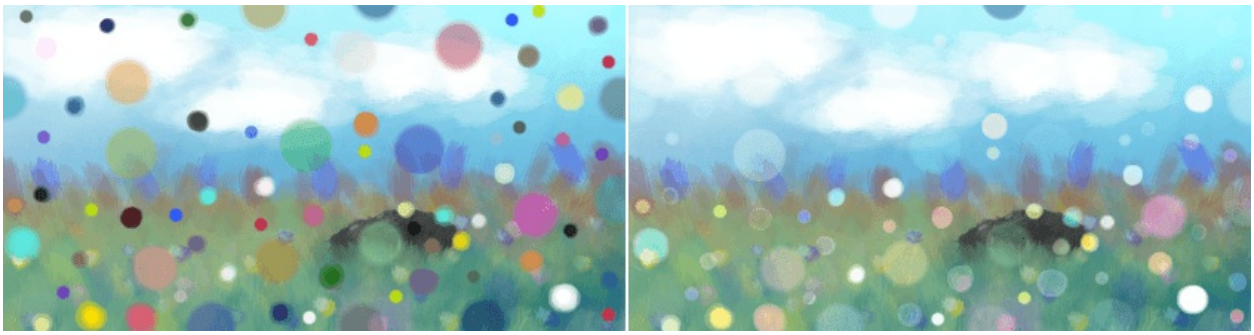
Put in for compatibility purposes.



Left: **Normal**. Right: **Linear Dodge** (exactly the same as Addition).

Easy Dodge

Aims to solve issues with Color Dodge blending mode by using a formula which falloff is similar to Dodge, but the falloff rate is softer. It is within the range of 0.0f and 1.0f unlike Color Dodge mode.



Left: **Normal**. Right: **Easy Dodge**.

Flat Light

The spreadout variation of Vivid Light mode which range is between 0.0f and 1.0f.

Left: **Normal**. Right: **Flat Light**.

Fog Lighten (IFS Illusions)

Lightens the image in a way that there is a ‘fog’ in the end result. This is due to the unique property of fog lighten in which midtones combined are lighter than non-midtones blend.

Left: **Normal**. Right: **Fog Lighten**.

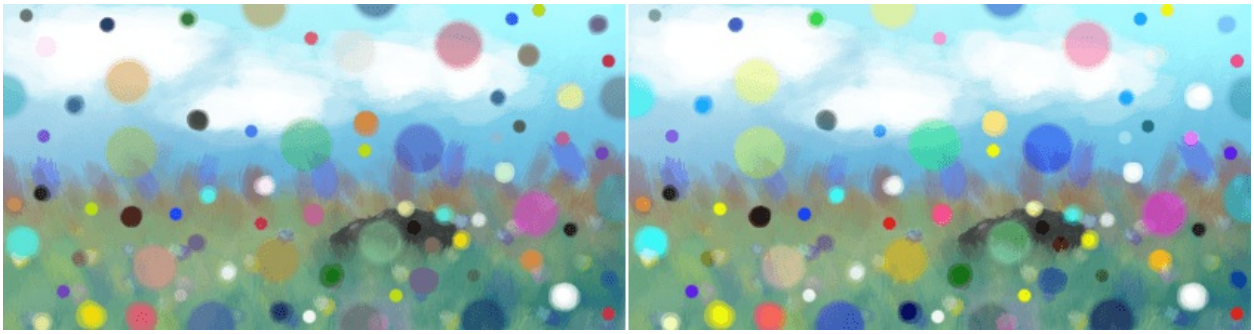
Linear Light

Similar to Overlay.

Combines Linear Dodge and Linear Burn. When the lightness of the upper-pixel is higher than 0.5, it uses Linear dodge, if not, Linear burn to blend the pixels.

Left: **Normal**. Right: **Linear Light**.

Left: **Normal**. Right: **Linear Light**.



Left: **Normal**. Right: **Linear Light**.

Luminosity/Shine (SAI)

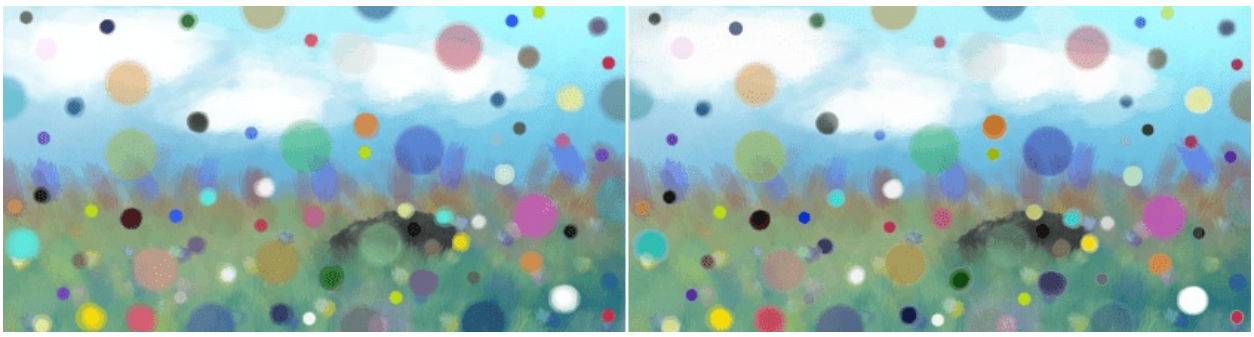
Similar to Addition.

Takes the opacity of the new color (combined opacity of the layer, the brush, any used transparency masks etc.) and multiples the color by the opacity, then adds to the original/previous color.

\[c_{new} = c_{above}*{\alpha}_{above} + c_{below}\]

The result of this operation is the same as combining the new pixels with a fully opaque black layer in a Normal mode and then combining the result with the original layer using Addition mode. It should be also the same as the

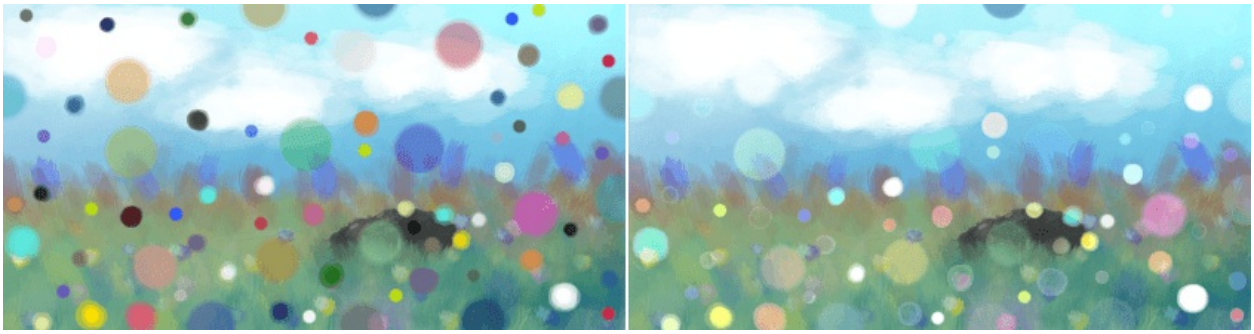
results of “Luminosity” blending mode in SAI1 or “Shine” blending mode in SAI2.



Left: **Normal**. Right: **Luminosity/Shine (SAI)**.

P-Norm A

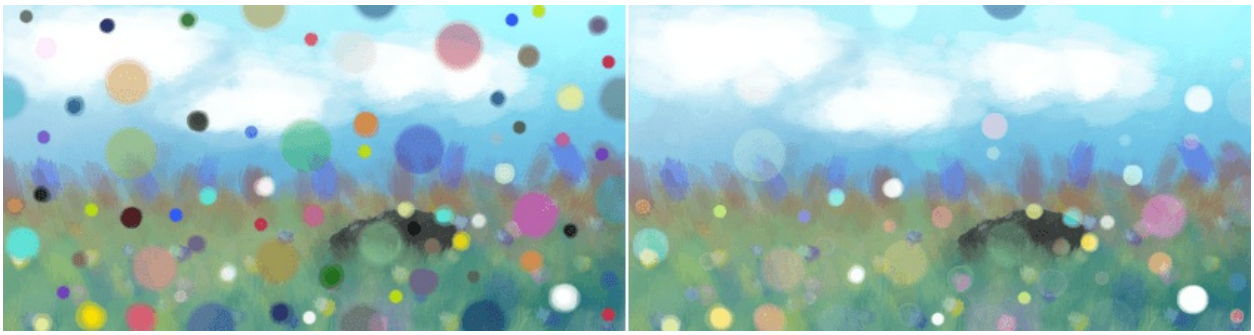
P-Norm A is similar to Screen blending mode which slightly darken images, and the falloff is more consistent all-around in terms of outline of values. Can be used an alternative to screen blending mode at times.



Left: **Normal**. Right: **P-Norm A**.

P-Norm B

P-Norm B is similar to Screen blending mode which slightly darken images, and the falloff is more consistent all-around in terms of outline of values. The falloff is sharper in P-Norm B than in P-Norm A. Can be used an alternative to screen blending mode at times.



Left: **Normal**. Right: **P-Norm B**.

Pin Light

Checks which is darker the lower layer’s pixel or the upper layer’s double so bright. Then checks which is brighter of that result or the inversion of the doubled lower layer.

Left: **Normal**. Right: **Pin Light**.

Left: **Normal**. Right: **Pin Light**.

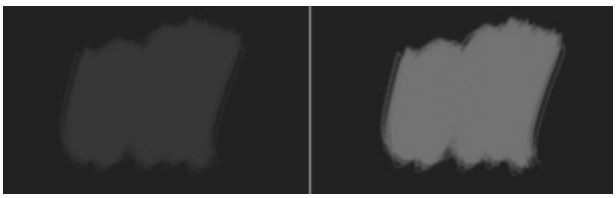
Left: **Normal**. Right: **Pin Light**.

Screen

Perceptually the opposite of Multiply.

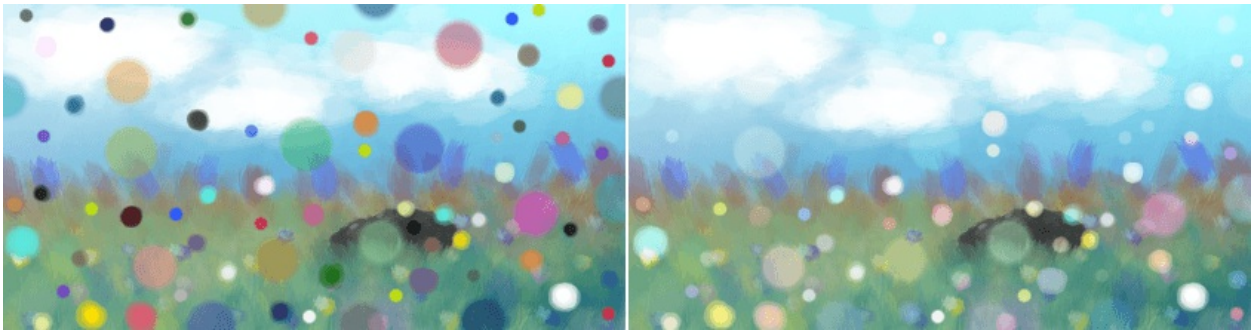
Mathematically, Screen takes both layers, inverts them, then multiplies them, and finally inverts them again.

This results in light tones being more opaque and dark tones transparent.



Left: **Normal**. Right: **Screen**.

Left: **Normal**. Right: **Screen**.

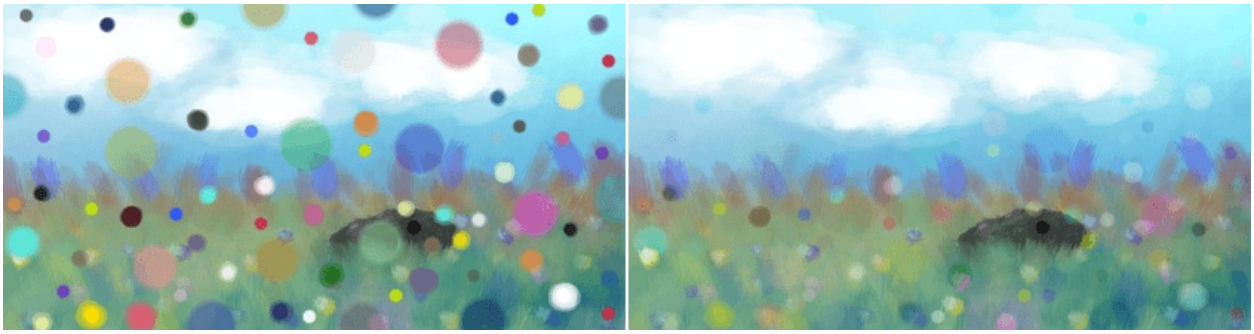


Left: **Normal**. Right: **Screen**.

Soft Light (Photoshop) & Soft Light SVG

These are less harsh versions of Hard Light, not resulting in full black or full white.

The SVG version is slightly different to the Photoshop version in that it uses a slightly different bit of formula when the lightness of the lower pixel is lower than 25%, this prevents the strength of the brightness increase.



Left: **Normal**. Right: **Soft Light (Photoshop)**.

Left: **Normal**. Right: **Soft Light (SVG)**.

Soft Light (IFS Illusions) & Soft Light (Pegtop-Delphi)

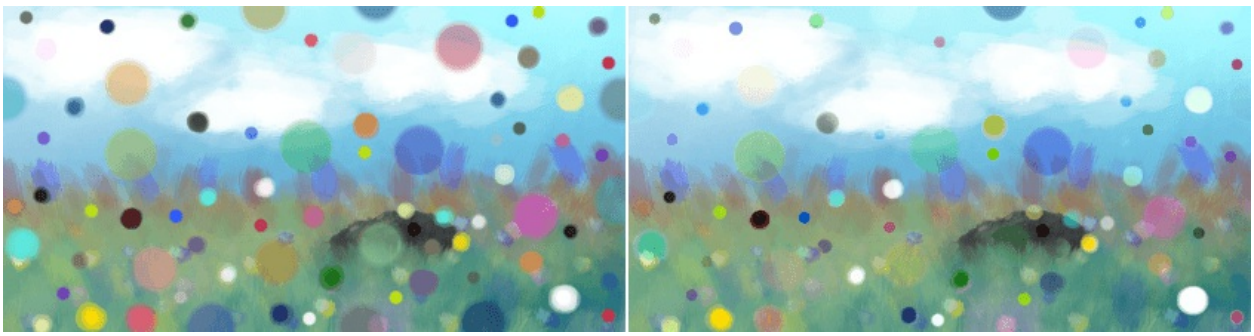
These are alternative versions of standard softlight modes which are made to solve discontinuities seen with the standard blend modes. Sometimes, these modes offer subtle advantages by offering more contrast within some areas, and these advantages are more or less noticeable within different color spaces and depth.

Left: **Normal**. Right: **Soft Light (IFS Illusions)**.

Left: **Normal**. Right: **Soft Light (Pegtop-Delphi)**.

Super Light

Smoother variation of Hard Light blending mode with more contrast in it.



Left: **Normal**. Right: **Super Light**.

Tint (IFS Illusions)

Basically, the blending mode only ends in shades of tints. This means that it’s very useful for painting light colors while still in the range of tints.

Left: **Normal**. Right: **Tint**.

Vivid Light

Similar to Overlay.

Mixes both Color Dodge and Burn blending modes. If the color of the upper layer is darker than 50%, the blending mode will be Burn, if not the blending mode will be Color Dodge.

> **Warning**
>
> This algorithm doesn’t use color dodge and burn, we don’t know WHAT it does do but for Color Dodge and Burn you need to use Hard Mix.

Left: **Normal**. Right: **Vivid Light**.

Misc

Bumpmap

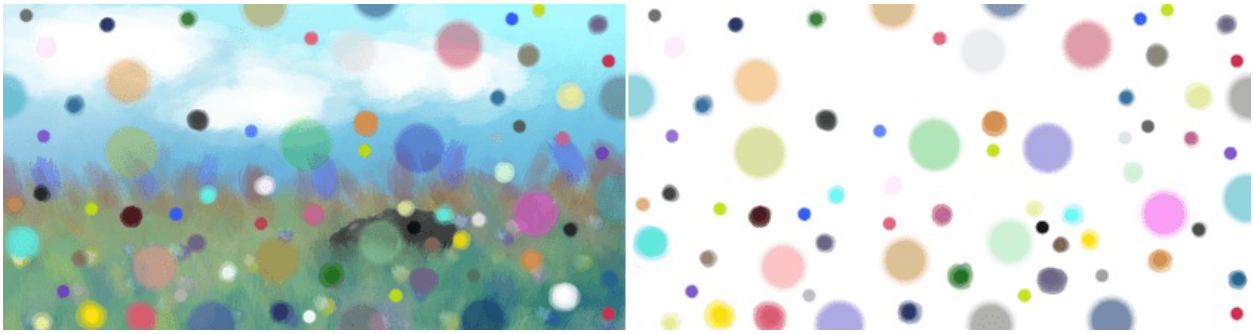
This filter seems to both multiply and respect the alpha of the input.

Combine Normal Map

Mathematically robust blending mode for normal maps, using Reoriented Normal Map Blending [https://blog.selfshadow.com/publications/blending-in-detail/].

Copy

Copies the previous layer exactly. Useful for when using filters and filter-masks.



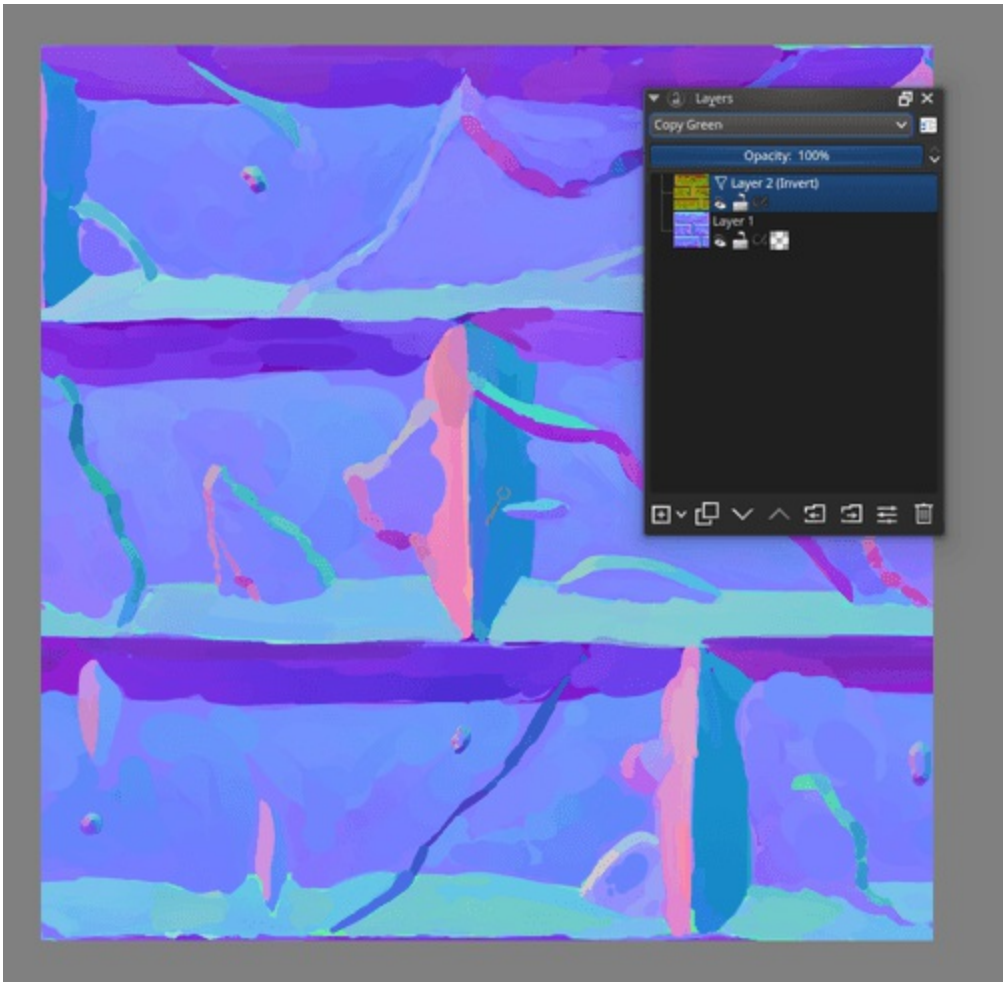
Left: **Normal**. Right: **Copy**.

Copy Red, Green, Blue

This is a blending mode that will just copy/blend a source channel to a destination channel. Specifically, it will take the specific channel from the upper layer and copy that over to the lower layers.

So, if you want the brush to only affect the red channel, set the blending

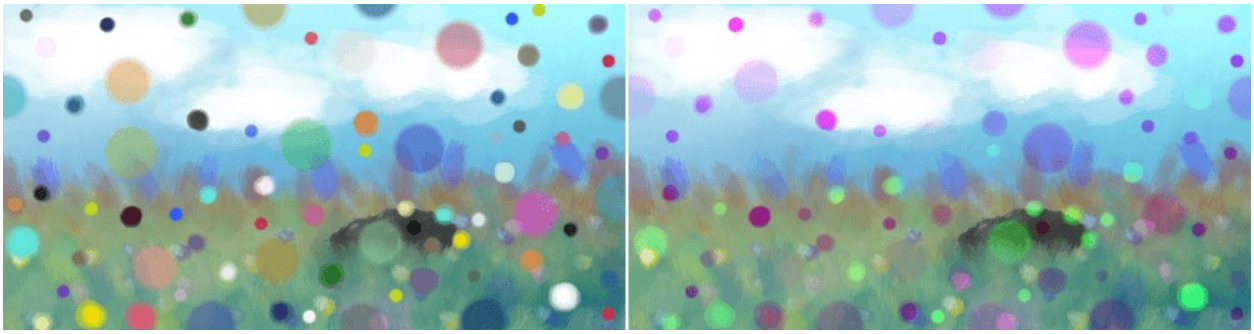
mode to ‘copy red’.



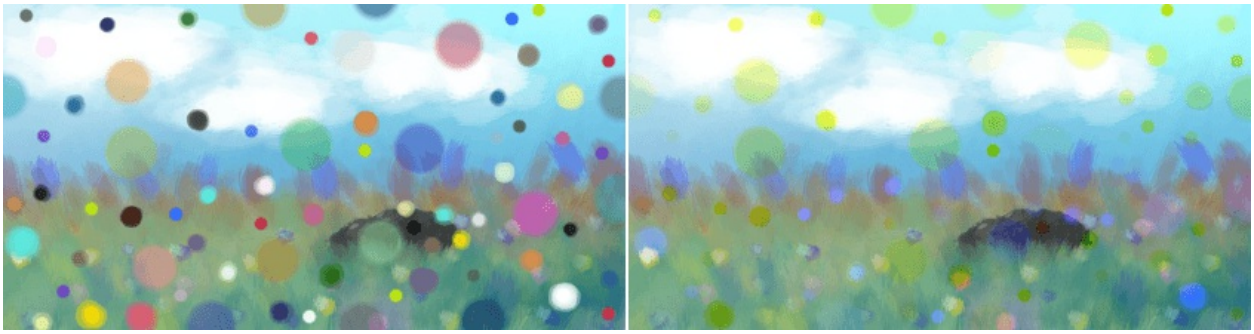
The copy red, green and blue blending modes also work on filter-layers.

This can also be done with filter layers. So if you quickly want to flip a layer’s green channel, make an invert filter layer with ‘copy green’ above it.

Left: **Normal**. Right: **Copy Red**.



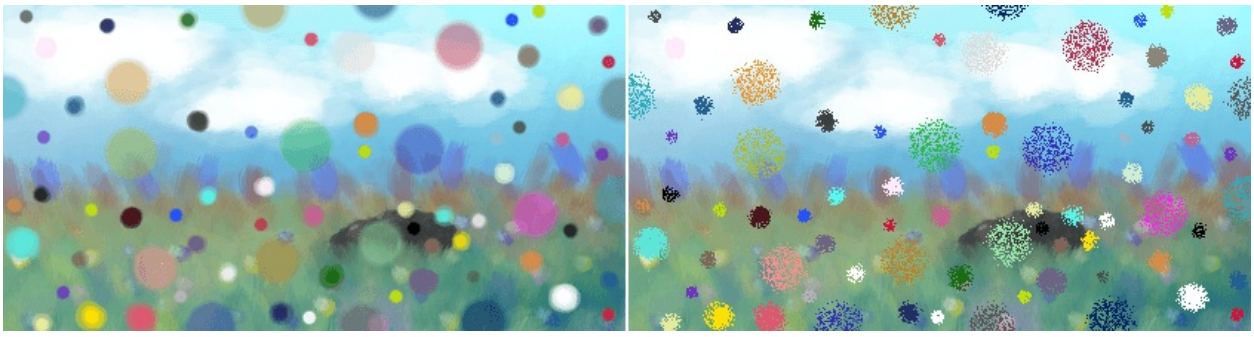
Left: **Normal**. Right: **Copy Green**.



Left: **Normal**. Right: **Copy Blue**.

Dissolve

Instead of using transparency, this blending mode will use a random dithering pattern to make the transparent areas look sort of transparent.

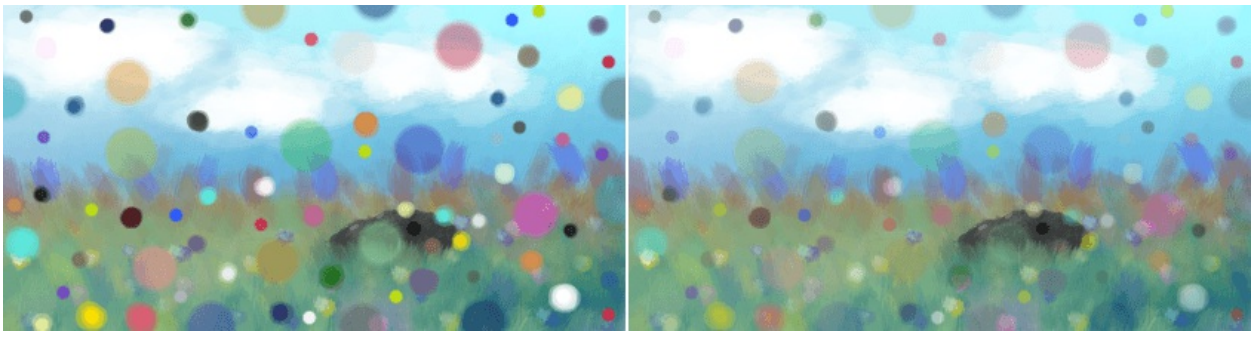


Left: **Normal**. Right: **Dissolve**.

Mix

Allanon

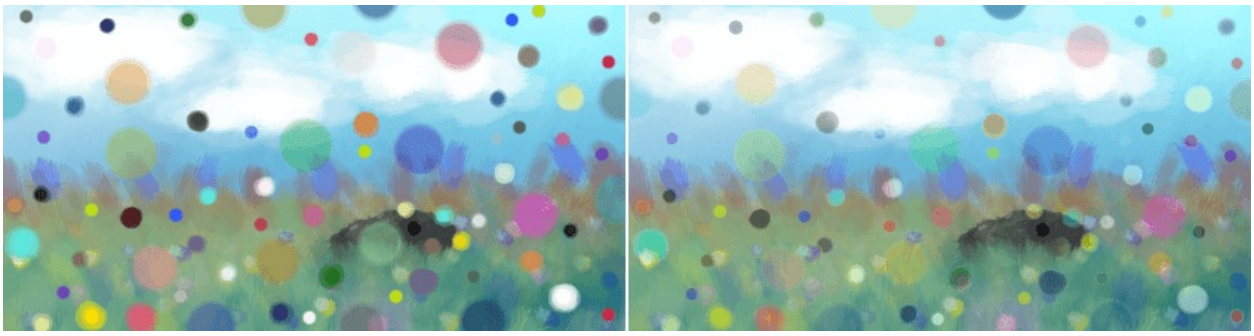
Blends the upper layer as half-transparent with the lower. (It add the two layers together and then halves the value).



Left: **Normal**. Right: **Allanon**.

Interpolation

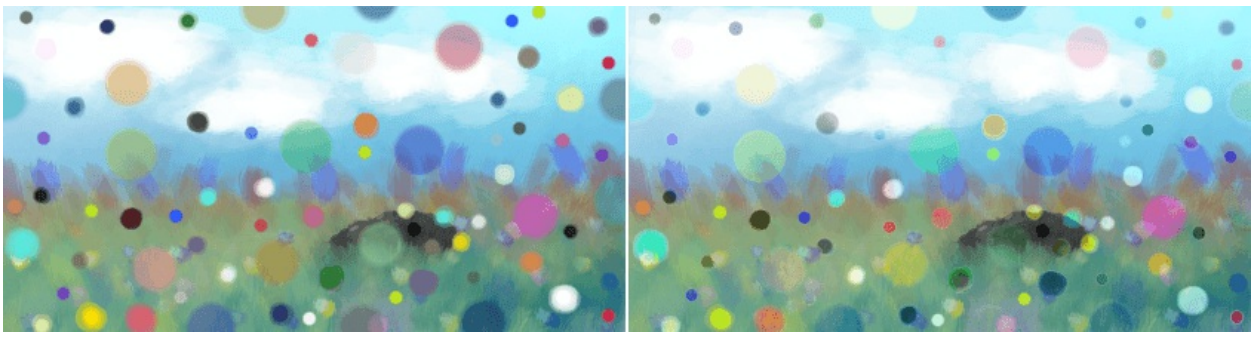
Subtract 0.5f by 1/4 of cosine of base layer subtracted by 1/4 of cosine of blend layer assuming 0-1 range. The result is similar to Allanon mode, but with more contrast and functional difference to 50% opacity.



Left: **Normal**. Right: **Interpolation**.

Interpolation - 2X

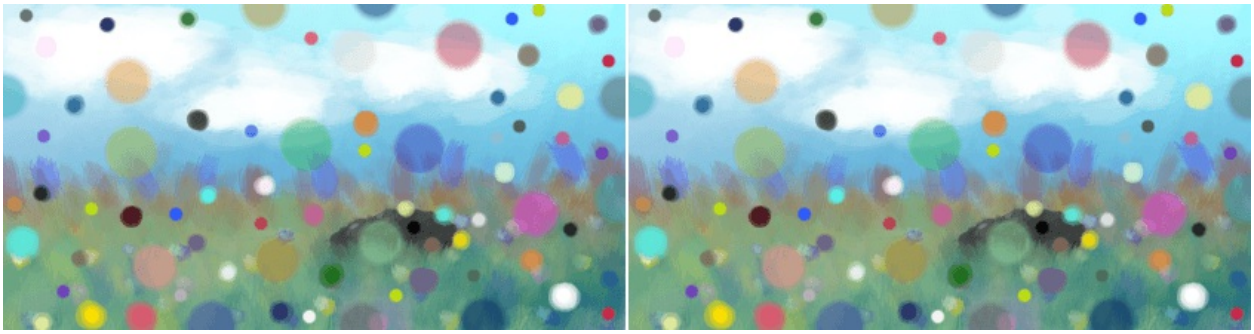
Applies Interpolation blend mode to base and blend layers, then duplicate to repeat interpolation blending.



Left: **Normal**. Right: **Interpolation - 2X**.

Alpha Darken

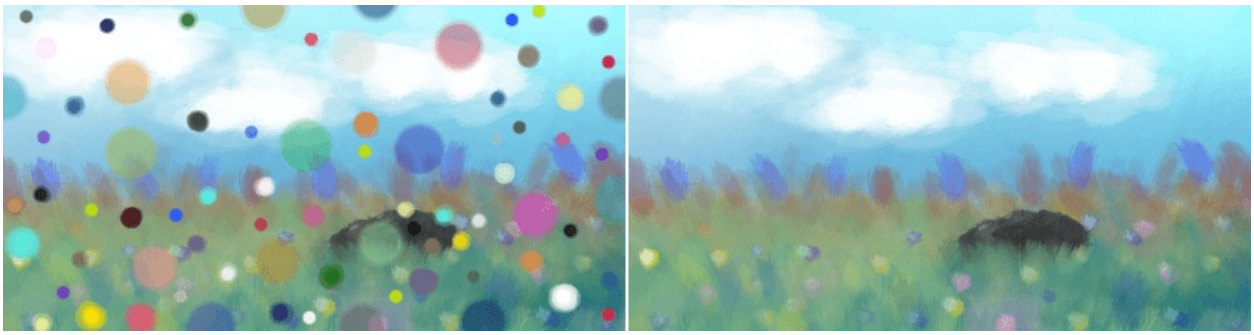
As far as I can tell this seems to premultiply the alpha, as is common in some file-formats.



Left: **Normal**. Right: **Alpha Darken**.

Behind

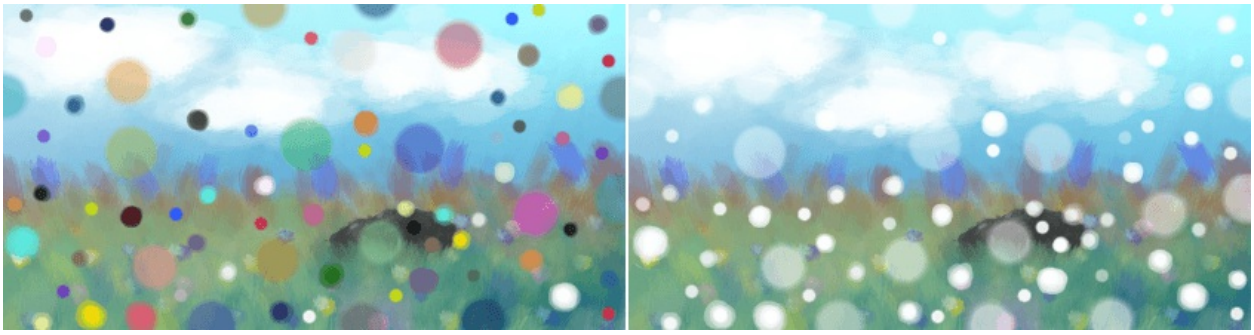
Does the opposite of normal, and tries to have the upper layer rendered below the lower layer.



Left: **Normal**. Right: **Behind**.

Erase

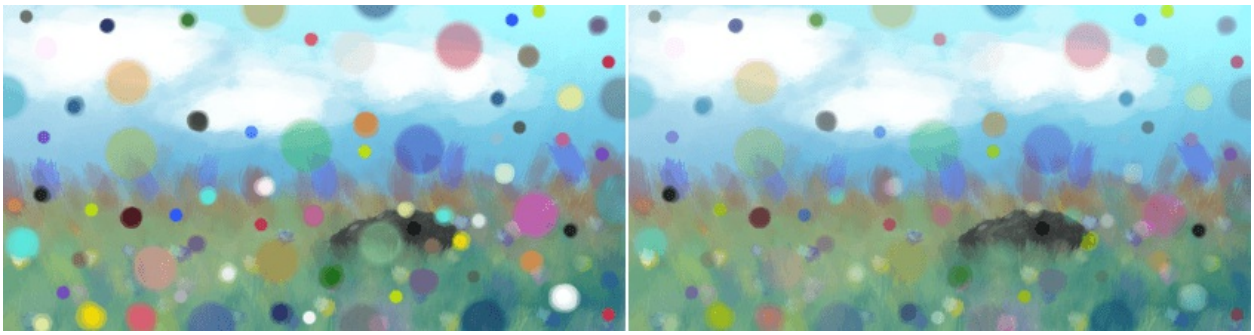
This subtracts the opaque pixels of the upper layer from the lower layer, effectively erasing.



Left: **Normal**. Right: **Erase**.

Geometric Mean

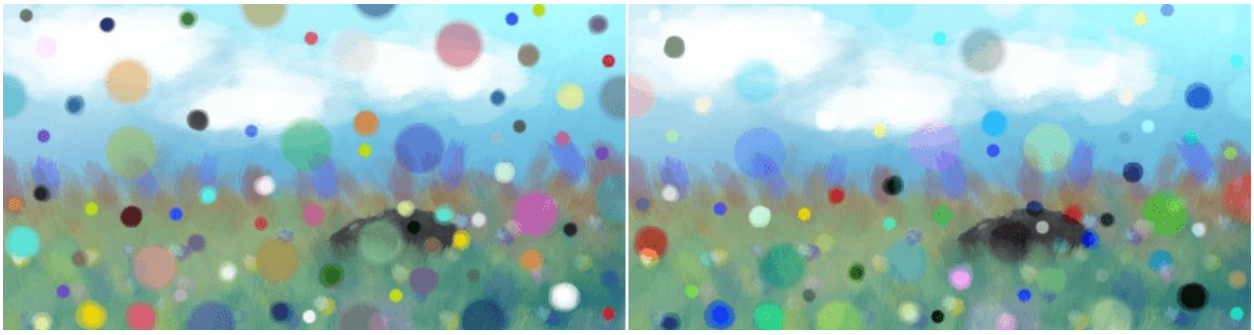
This blending mode multiplies the top layer with the bottom, and then outputs the square root of that.



Left: **Normal**. Right: **Geometric Mean**.

Grain Extract

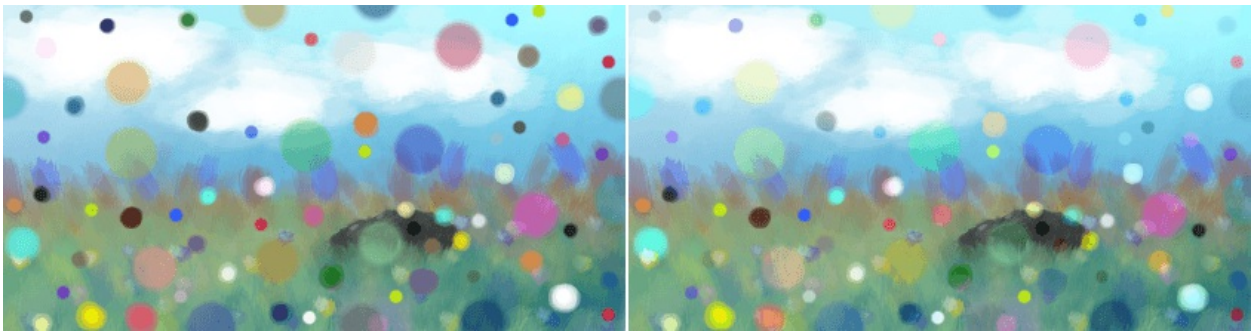
Similar to subtract, the colors of the upper layer are subtracted from the colors of the lower layer, and then 50% gray is added.



Left: **Normal**. Right: **Grain Extract**.

Grain Merge

Similar to addition, the colors of the upper layer are added to the colors, and then 50% gray is subtracted.



Left: **Normal**. Right: **Grain Merge**.

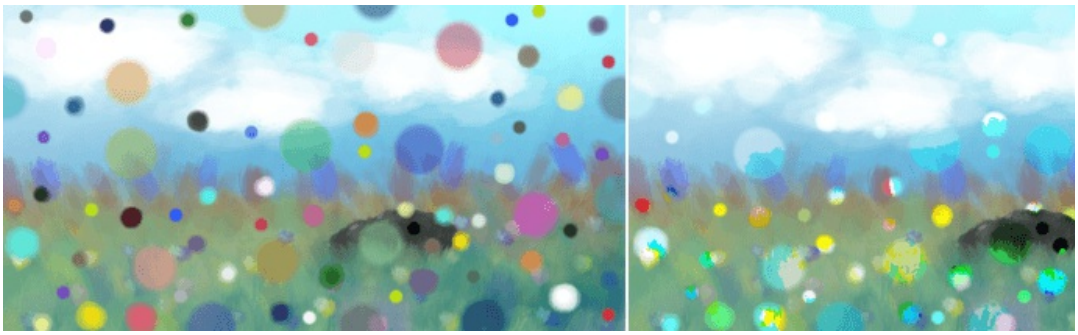
Greater

A blending mode which checks whether the painted color is painted with a higher opacity than the existing colors. If so, it paints over them, if not, it doesn’t paint at all.

Hard Mix

Similar to Overlay.

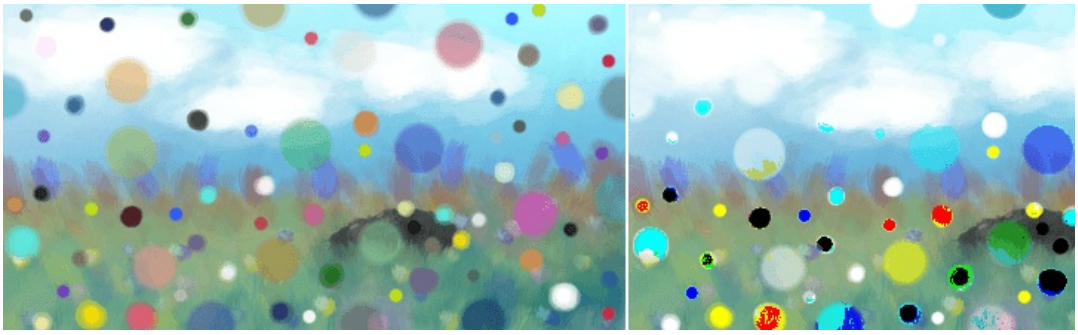
Mixes both Color Dodge and Burn blending modes. If the color of the upper layer is darker than 50%, the blending mode will be Burn, if not the blending mode will be Color Dodge.



Left: **Normal**. Right: **Hard Mix**.

Hard Mix (Photoshop)

This is the hard mix blending mode as it is implemented in photoshop.



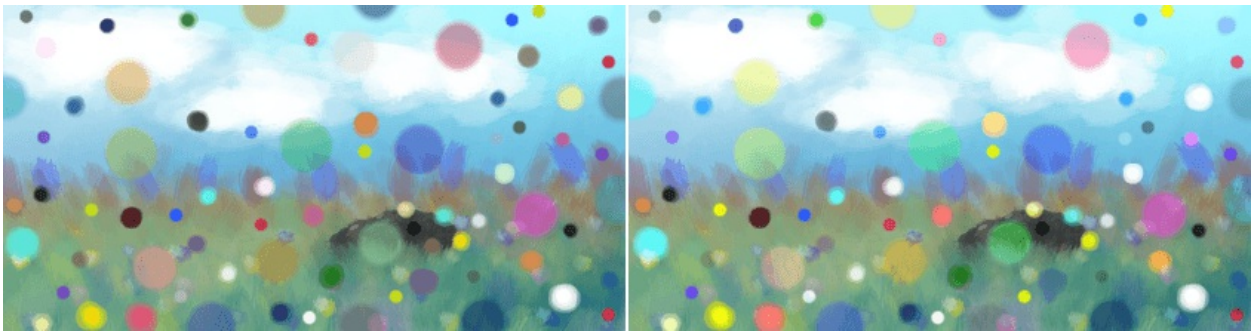
Left: Dots are mixed in with the normal blending mode, on the **Right**: Dots hardmix.

This add the two values, and then checks if the value is above the maximum. If so it will output the maximum, otherwise the minimum.

Hard Overlay

New in version 4.0.

Similar to Hard light but hard light use Screen when the value is above 50%. Divide gives better results than Screen, especially on floating point images.

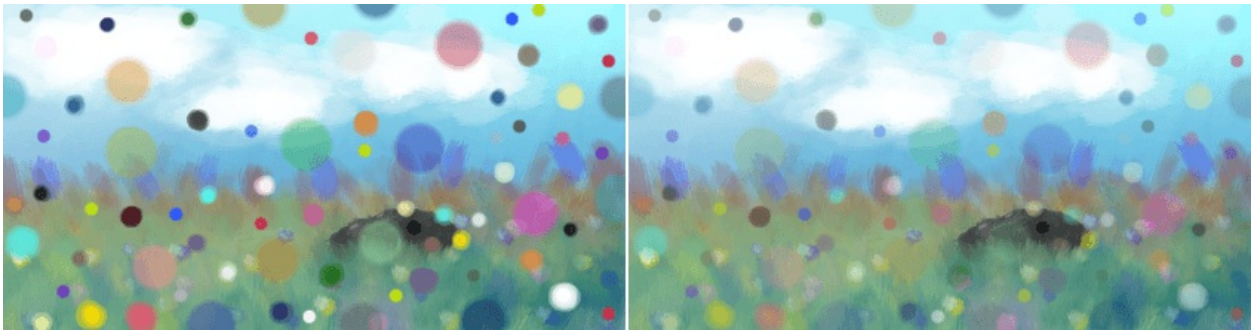


Left: **Normal**. Right: **Hard Overlay**.

Normal

As you may have guessed this is the default Blending mode for all layers.

In this mode, the computer checks on the upper layer how transparent a pixel is, which color it is, and then mixes the color of the upper layer with the lower layer proportional to the transparency.



Left: **Normal** 100% Opacity. Right: **Normal** 50% Opacity.

Overlay

A combination of the Multiply and Screen blending modes, switching between both at a middle-lightness.

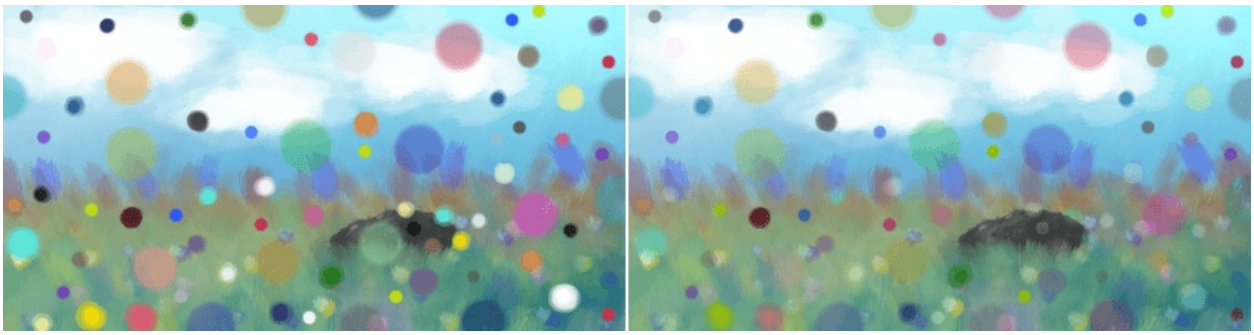
Overlay checks if the color on the upperlayer has a lightness above 0.5. If so, the pixel is blended like in Screen mode, if not the pixel is blended like in Multiply mode.

This is useful for deepening shadows and highlights.

Left: **Normal**. Right: **Overlay**.

Parallel

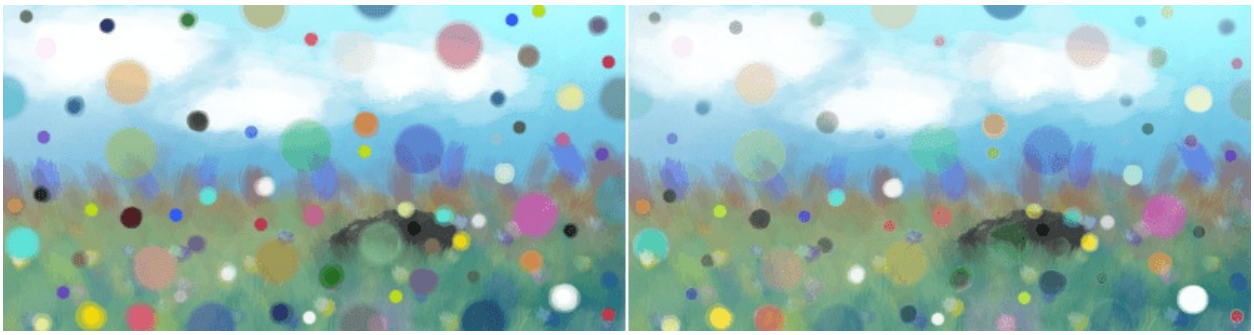
This one first takes the percentage in two decimal behind the comma for both layers. It then adds the two values. Divides 2 by the sum.



Left: **Normal**. Right: **Parallel**.

Penumbra A

Creates a linear penumbra falloff. This means most tones will be in the midtone ranges.



Left: **Normal**. Right: **Penumbra A**.

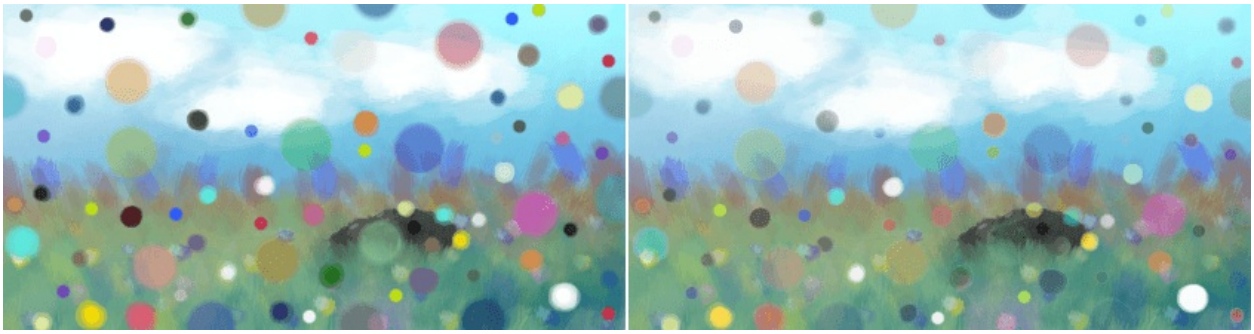
Penumbra B

Penumbra A with source and destination layer swapped.

Left: **Normal**. Right: **Penumbra B**.

Penumbra C

Creates a penumbra-like falloff using arc-tangent formula. This means most tones will be in the midtone ranges.



Left: **Normal**. Right: **Penumbra C**.

Penumbra D

Penumbra C with source and destination layer swapped.

Left: **Normal**. Right: **Penumbra D**.

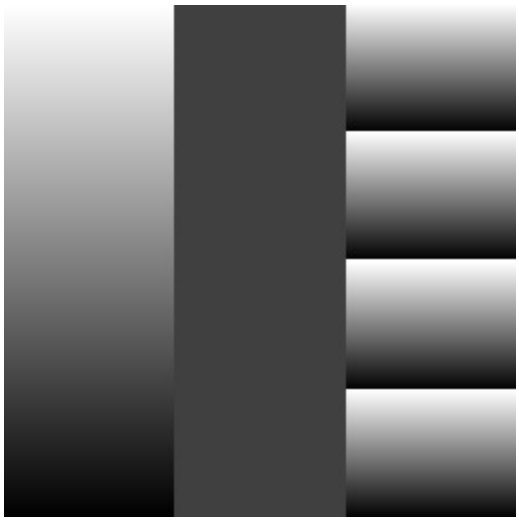
Modulo

Modulo modes are special class of blending modes which loops values when the value of channel blend layer is less than the value of channel in base layers. All modes in modulo modes retains the absolute of the remainder if the value is greater than the maximum value or the value is less than minimum value. Continuous modes assume if the calculated value before modulo operation is within the range between a odd number to even number, then values are inverted in the end result, so values are perceived to be wave-like.

Furthermore, this would imply that modulo modes are beneficial for abstract art, and manipulation of gradients.

Divisive Modulo

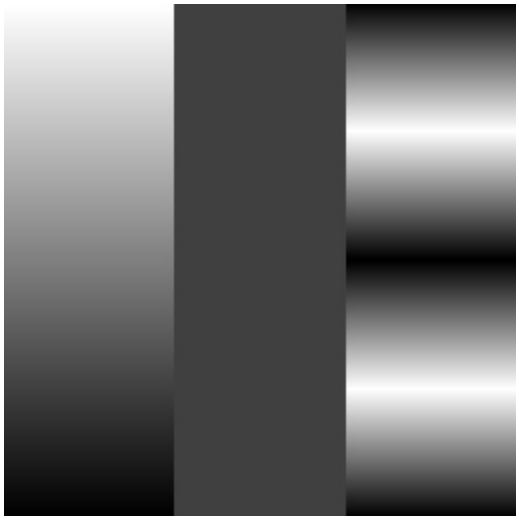
First, Base Layer is divided by the sum of blend layer and the minimum possible value after zero. Then, performs a modulo calculation using the value found with the sum of blend layer and the minimum possible value after zero.



Left: **Base Layer**. Middle: **Blend Layer**. Right: **Divisive Modulo**.

Divisive Modulo - Continuous

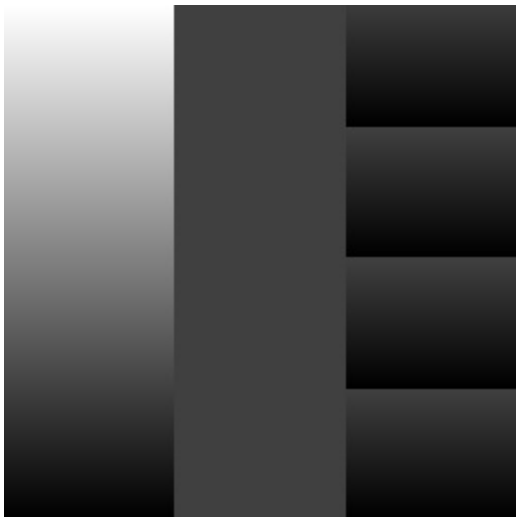
First, Base Layer is divided by the sum of blend layer and the minimum possible value after zero. Then, performs a modulo calculation using the value found with the sum of blend layer and the minimum possible value after zero. As this is a continuous mode, anything between odd to even numbers are inverted.



Left: **Base Layer**. Middle: **Blend Layer**. Right: **Divisive Modulo - Continuous**.

Modulo

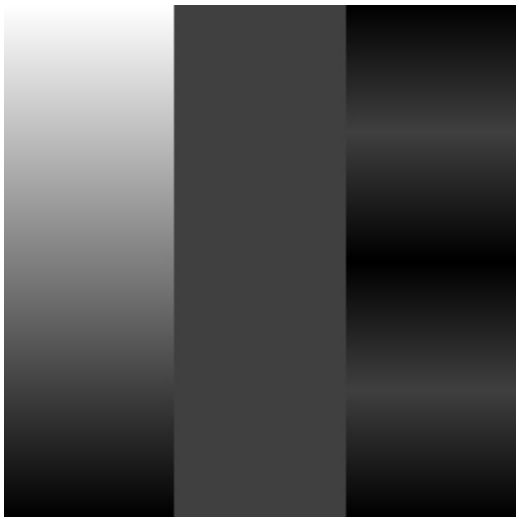
Performs a modulo calculation using the sum of blend layer and the minimum possible value after zero.



Left: **Base Layer**. Middle: **Blend Layer**. Right: **Modulo**.

Modulo - Continuous

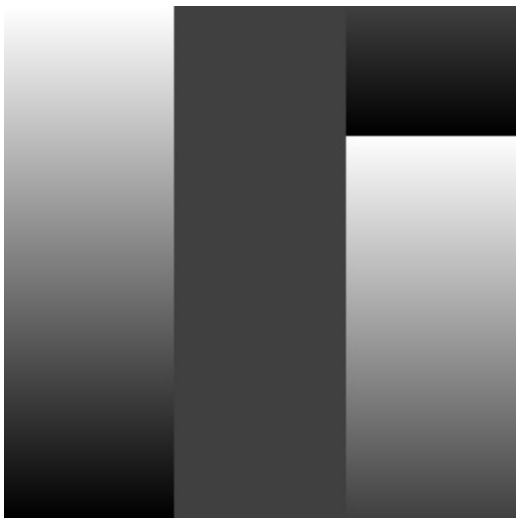
Performs a modulo calculation using the sum of blend layer and the minimum possible value after zero. As this is a continuous mode, anything between odd to even numbers are inverted.



Left: **Base Layer**. Middle: **Blend Layer**. Right: **Modulo - Continuous**.

Modulo Shift

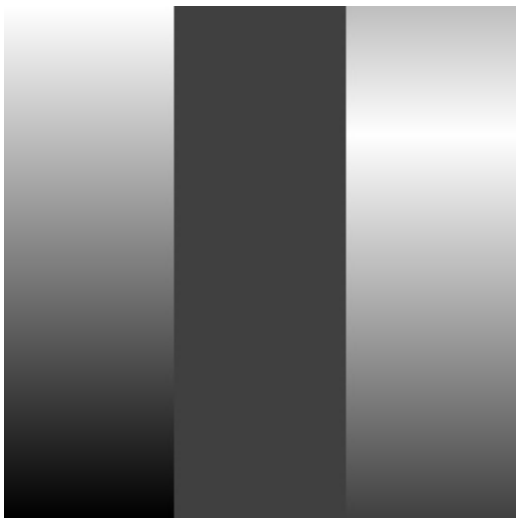
Performs a modulo calculation with the result of the sum of base and blend layer by the sum of blend layer with the minimum possible value after zero.



Left: **Base Layer**. Middle: **Blend Layer**. Right: **Modulo Shift**.

Modulo Shift - Continuous

Performs a modulo calculation with the result of the sum of base and blend layer by the sum of blend layer with the minimum possible value after zero. As this is a continuous mode, anything between odd to even numbers are inverted.



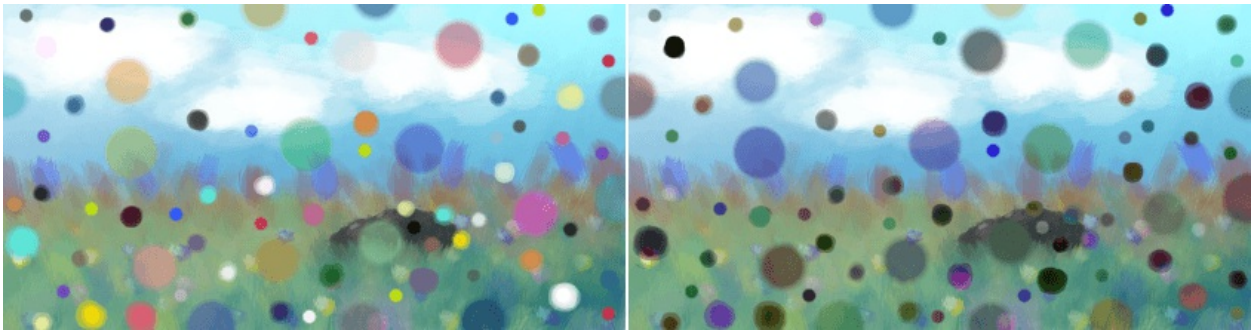
Left: **Base Layer**. Middle: **Blend Layer**. Right: **Modulo Shift - Continuous**.

Negative

These are all blending modes which seem to make the image go negative.

Additive Subtractive

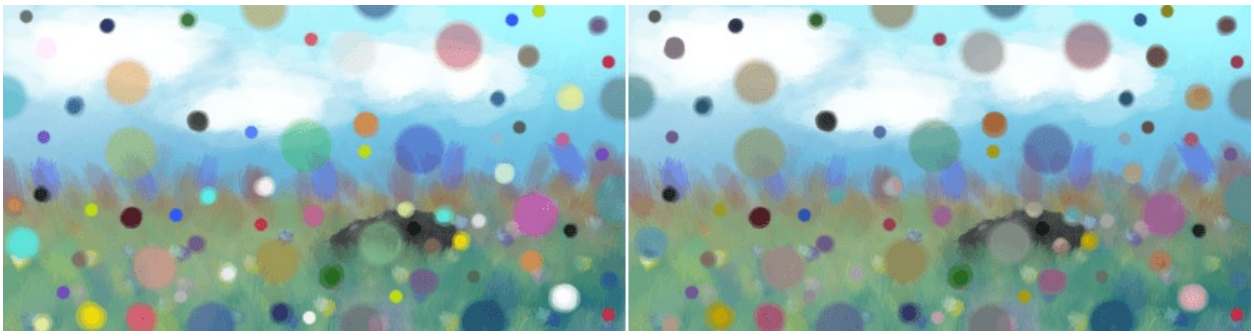
Subtract the square root of the lower layer from the upper layer.



Left: **Normal**. Right: **Additive Subtractive**.

Arcus Tangent

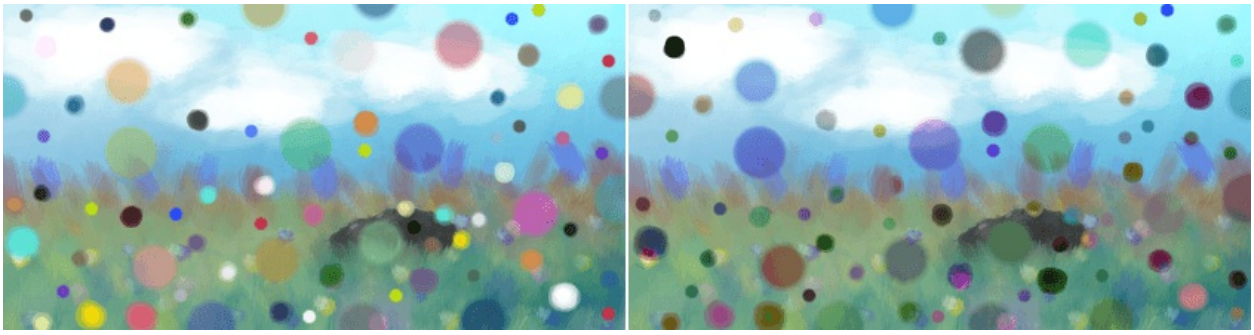
Divides the lower layer by the top. Then divides this by Pi. Then uses that in an Arc tangent function, and multiplies it by two.



Left: **Normal**. Right: **Arcus Tangent**.

Difference

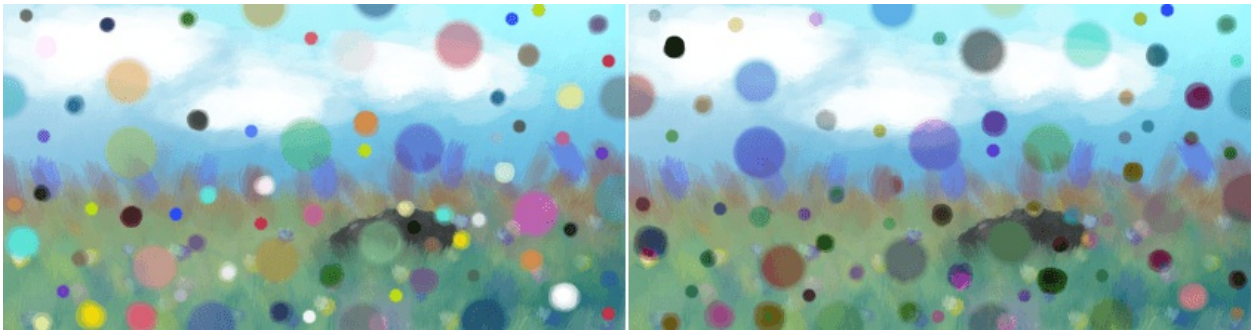
Checks per pixel of which layer the pixel-value is highest/lowest, and then subtracts the lower value from the higher-value.



Left: **Normal**. Right: **Difference**.

Equivalence

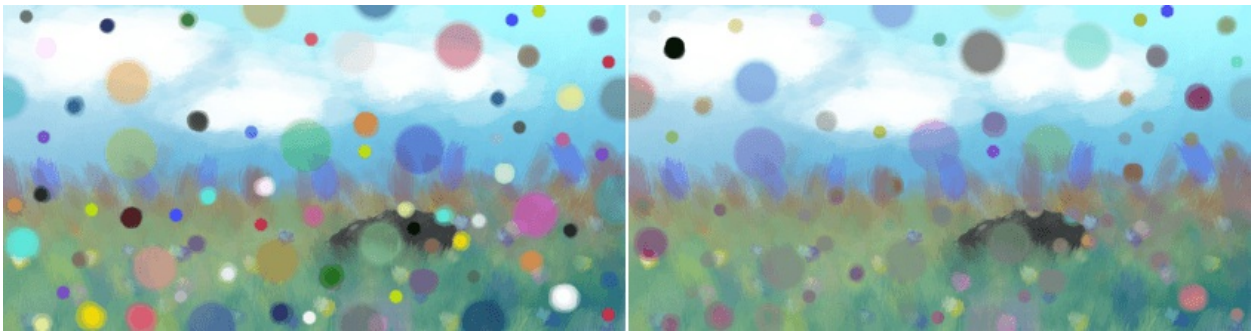
Subtracts the underlying layer from the upper-layer. Then inverts that. Seems to produce the same result as Difference.



Left: **Normal**. Right: **Equivalence**.

Exclusion

This multiplies the two layers, adds the source, and then subtracts the multiple of two layers twice.



Left: **Normal**. Right: **Exclusion**.

Negation

The absolute of the 1.0f value subtracted by base subtracted by the blend layer. abs(1.0f - Base - Blend)

Left: **Normal**. Right: **Negation**.

Quadratic

New in version 4.2.

The quadratic blending modes are a set of modes intended to give various effects when adding light zones or overlaying shiny objects.

Freeze

The freeze blending mode. Inversion of the reflect blending mode.

Left: **Normal**. Right: **Freeze**.

Freeze-Reflect

Mix of Freeze and Reflect blend mode.

Left: **Normal**. Right: **Freeze-Reflect**.

Glow

Reflect Blend Mode with source and destination layers swapped.

Left: **Normal**. Right: **Glow**.

Glow-Heat

Mix of Glow and Heat blend mode.

Left: **Normal**. Right: **Glow_Heat**.

Heat

The Heat Blend Mode. Inversion of the Glow Blend Mode.

Left: **Normal**. Right: **Heat**.

Heat-Glow

Mix of Heat, and Glow blending mode.

Left: **Normal**. Right: **Heat-Glow**.

Heat-Glow and Freeze-Reflect Hybrid

Mix of the continuous quadratic blending modes. Very similar to overlay, and sometimes provides better result than overlay.

Left: **Normal**. Right: **Heat-Glow and Freeze-Reflect Hybrid**.

Reflect

Reflect is essentially Color Dodge Blending mode with quadratic falloff.

Left: **Normal**. Right: **Reflect**.

Reflect-Freeze

Mix of Reflect and Freeze blend mode.

Left: **Normal**. Right: **Reflect-Freeze**.

Brushes

One of the most important parts of a painting program, Krita has a very extensive brush system.

Brush Engines

Information on the brush engines that can be accessed in the brush editor.

Available Engines:

- Bristle Brush Engine
- Chalk Brush Engine
- Clone Brush Engine
- Color Smudge Brush Engine
- Curve Brush Engine
- Deform Brush Engine
- Dyna Brush Engine
- Filter Brush Engine
- Grid Brush Engine
- Hatching Brush Engine
- Particle Brush Engine
- Pixel Brush Engine
- Quick Brush Engine
- Shape Brush Engine
- Sketch Brush Engine
- Spray Brush Engine
- Tangent Normal Brush Engine

Bristle Brush Engine

A brush intended to mimic real-life brushes by drawing the trails of their lines or bristles.

Brush Tip

Simply put:

- The brush tip defines the areas with bristles in them.
- Lower opacity areas have lower-opacity bristles. With this brush, this may give the illusion that lower-opacity areas have fewer bristles.
- The Size and Rotation dynamics affect the brush tip, not the bristles.

You can:

- Use different shapes for different effects. Be aware that complex brush shapes will draw more slowly though, while the effects aren't always visible (since in the end, you're passing over an area with a certain number of bristles).
- To decrease bristle density, you can also just use an autobrush and decrease the brush tip's density, or increase its randomness.

Bristle Options

The core of this particular brush-engine.

Scale

Think of it as pressing down on a brush to make the bristles further apart.

- Larger values basically give you larger brushes and larger bristle spacing. For example, a value of 4 will multiply your base brush size by 4, but the bristles will be 4 times more spaced apart.
- Use smaller values if you want a “dense” brush, i.e. you don’t want to see so many bristles within the center.
- Negative values have the same effect as corresponding positive values: -1.00 will look like 1.00, etc.

Random Offset

Adds a jaggy look to the trailing lines.

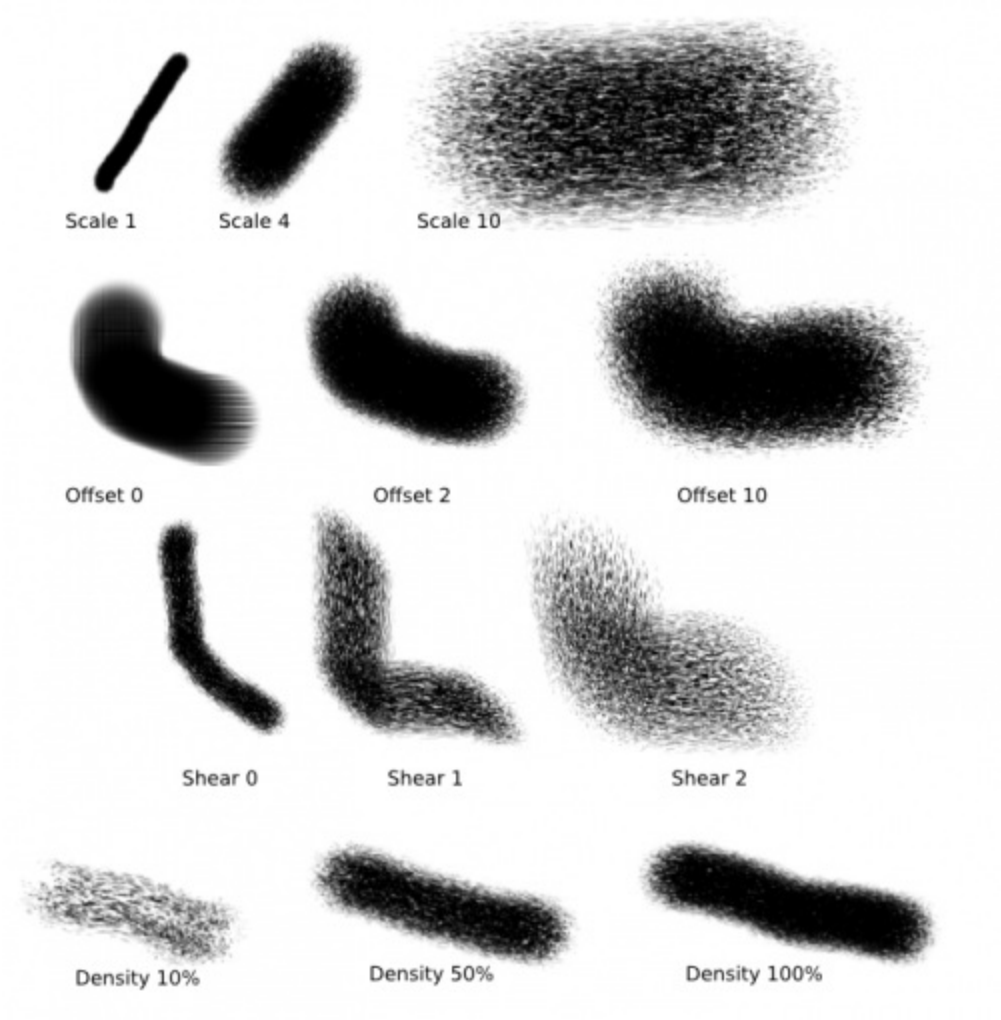
- At 0.00, all the bristles basically remain completely parallel.
- At other values, the bristles are offset randomly. Large values will increase the brush size a bit because of the bristles spreading around, but not by much.
- Negative values have the same effect as corresponding positive values.

Shear

Shear introduces an angle to your brush, as though you’re drawing with an oval brush (or the side of a round brush).

Density

This controls the density of bristles. Scale takes a number of bristles and expands or compresses them into a denser area, whereas density takes a fixed area and determines the number of bristles in it. See the difference?



Mouse Pressure

This one maps “Scale” to mouse speed, thus simulating pressure with a graphics tablet!

- Rather, it uses the “distance between two events” to determine scale. Faster drawing, larger distances.
- This doesn’t influence the “pressure” input for anything else (size, opacity, rotation etc.) so you still have to map those independently to something else.

Threshold

This is a tablet feature. When you turn this on, only bristles that are able to “touch the canvas” will be painted.

Connect Hairs

The bristles get connected. See for yourself.

Anti-Aliasing
: This will decrease the jaggy-ness of the lines.

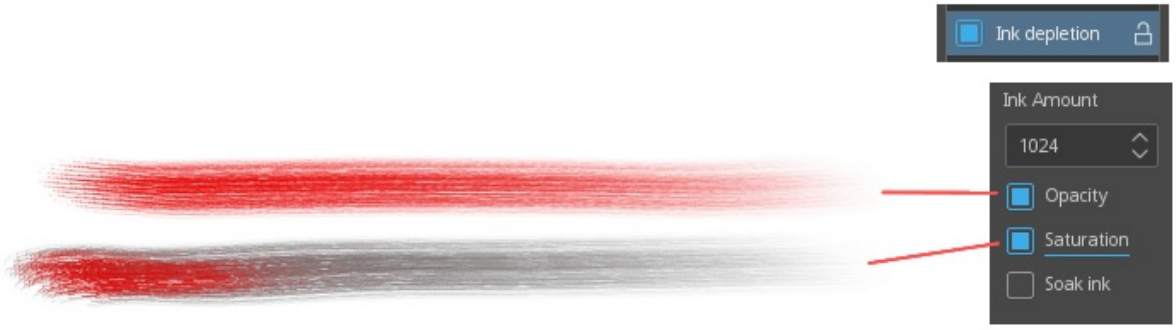
Composite Bristles
: This “composes the bristle colors within one dab,” but explains that the effect is “probably subtle”.

Ink Depletion

This simulated ink depletion over drawing time. The value dictates how long it will take. The curve dictates the speed.

Opacity
: The brush will go transparent to simulate ink-depletion.

Saturation
: The brush will be desaturated to simulate ink-depletion.

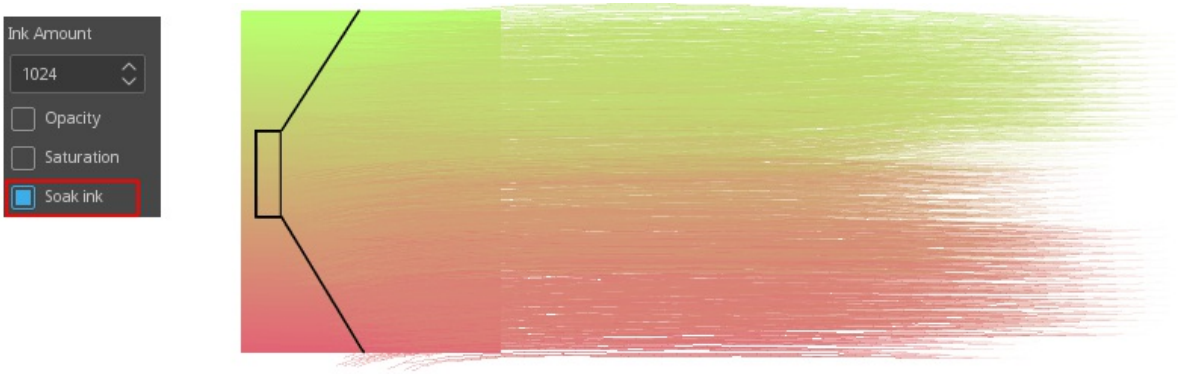


Soak Ink

The brush will pick up colors from other brushes. You don't need to have *Ink depletion* checked to activate this option, you just have to check *Soak ink*. What this does is cause the bristles of the brush to take on the colors of the first area they touch. Since the Bristle brush is made up of independent bristles, you can basically take on several colors at the same time.

Note

- It will only take colors in the unscaled area of the brush, so if you're using a brush with 4.00 scale for example, it will only take the colors in the 1/4 area closest to the center.
- When the source is transparent, the bristles take black color.



Warning

Be aware that this feature is a bit buggy though. It's supposed to take the color from the current layer, but some buggy behavior causes it to often use the last layer you've painted on (with a non-Bristle brush?) as source. To avoid these weird behaviors, stick to just one layer, or paint

something on the current active layer first with another brush (such as a Pixel brush).

Weighted saturation

Works by modifying the saturation with the following:

- Pressure weight
- Bristle length weight
- Bristle ink amount weight
- Ink depletion curve weight

Chalk Brush Engine

Deprecated since version 4.0: This brush engine has been removed in 4.0. There are other brush engines such as pixel that can do everything this can… plus more.

Apparently, the Bristle brush engine is derived from this brush engine. Now, all of **Krita's** brushes have a great variety of uses, so you must have tried out the Chalk brush and wondered what it is for. Is it nothing but a pixel brush with opacity and saturation fade options? As per the developers this brush uses a different algorithm than the Pixel Brush, and they left it in here as a simple demonstration of the capabilities of **Krita's** brush engines.

So there you go, this brush is here for algorithmic demonstration purposes. Don’t lose sleep because you can’t figure out what it’s for, it Really doesn’t do much. For the sake of description, here’s what it does:

Yeah, that’s it, a round brush with some chalky texture, and the option to fade in opacity and saturation. That’s it.

Clone Brush Engine

The clone brush is a brush engine that allows you to paint with a duplication of a section of a paint-layer. This is useful in manipulation of photos and textures. You have to select a source and then you can paint to copy or clone the source to a different area. Other applications normally have a separate tool for this, Krita has a brush engine for this.

Usage and Hotkeys

To see the source, you need to set the brush-cursor settings to brush outline.

The clone tool can now clone from the projection and it's possible to change the clone source layer. Press the `Ctrl + Alt +`  shortcut to select a new clone source on the current layer. The `Ctrl +`  shortcut to select a new clone source point on the layer that was active when you selected the clone op.

Warning

The `Ctrl + Alt +`  shortcut is temporarily disabled on 2.9.7.

Settings

- Size
- Blending Modes
- Opacity and Flow

Painting mode

Healing

This turns the clone brush into a healing brush: often used for removing blemishes in photo retouching, and maybe blemishes in painting.

Perspective correction

Only works when there’s a perspective grid visible.

Warning

This feature is currently disabled.

Source Point move

This will determine whether you will replicate the source point per dab or per stroke. Can be useful when used with the healing brush.

Source Point reset before a new stroke

This will reset the source point everytime you make a new stroke. So if you were cloning a part in one stroke, having this active will allow you to clone the same part again in a single stroke, instead of using the source point as a permanent offset.

Clone from all visible layers

Tick this to force cloning of all layers instead of just the active one.

Color Smudge Brush Engine

The Color Smudge Brush is a brush engine that allows you to mix colors by smearing or dulling. A very powerful brush engine to the painter.

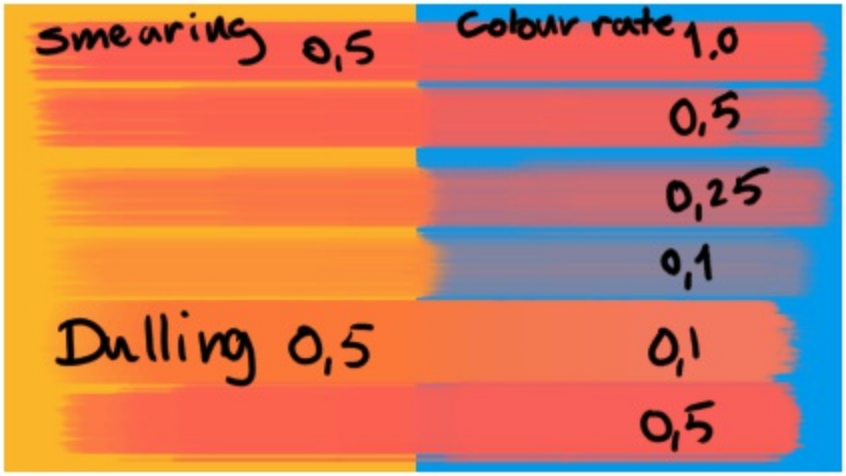
Options

- Brush Tips
- Blending Modes
- Opacity and Flow
- Size
- Ratio
- Spacing
- Mirror
- Rotation
- Scatter
- Gradient
- Airbrush
- Texture

Options Unique to the Color Smudge Brush

Color Rate

How much of the foreground color is added to the smudging mix. Works together with Smudge Length and Smudge Radius.



Smudge Length

Affects smudging and allows you to set it to Sensors.

There are two major types:

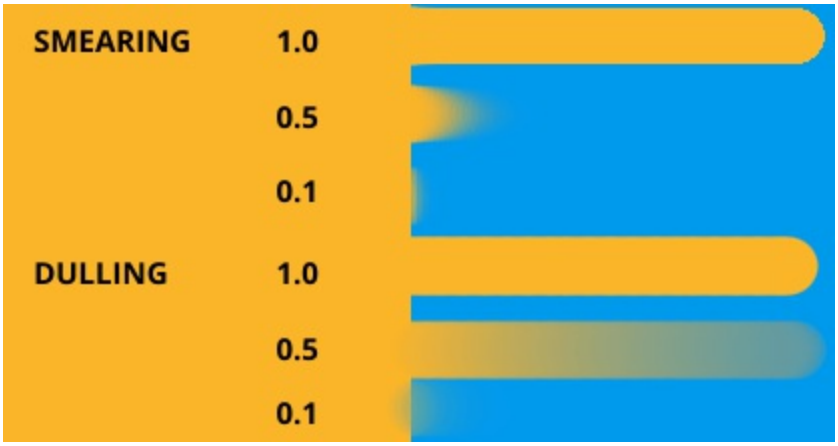
Smearing
: Great for making brushes that have a very impasto oil feel to them.

Dulling
: Named so because it dulls strong colors.

Using an arithmetic blending type, Dulling is great for more smooth type of painting.

Strength

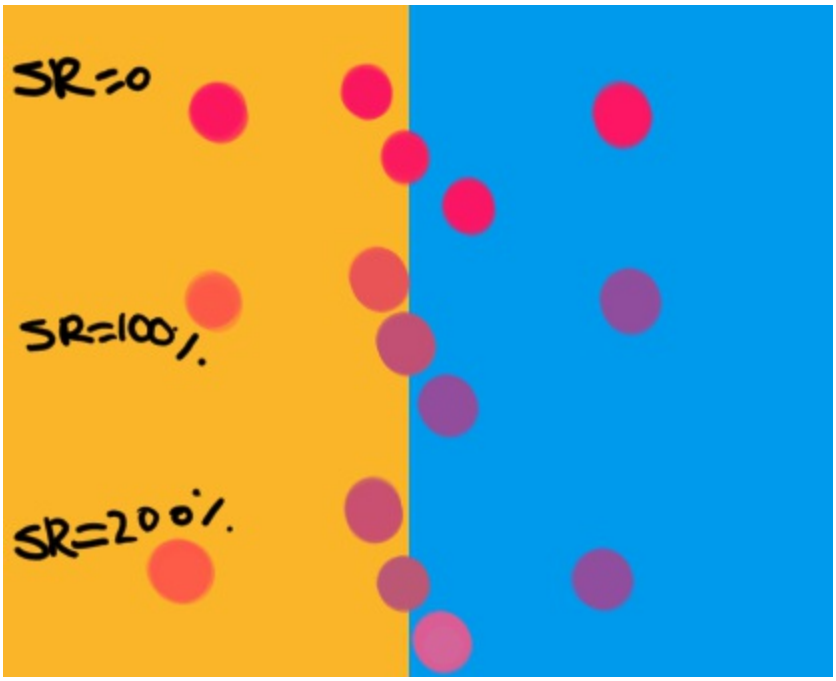
Affects how much the smudge length takes from the previous dab its sampling. This means that smudge-length at 1.0 will never decrease, but smudge-lengths under that will decrease based on spacing and opacity/flow.



Smudge Radius

The *Smudge Radius* allows you to sample a larger radius when using smudge-length in *Dulling* mode.

The slider is percentage of the brush-size. You can have it modified with *Sensors*.



Overlay

Overlay is a toggle that determine whether or not the smudge brush will sample all layers (overlay on), or only the current one.

Tutorial: Color Smudge Brushes

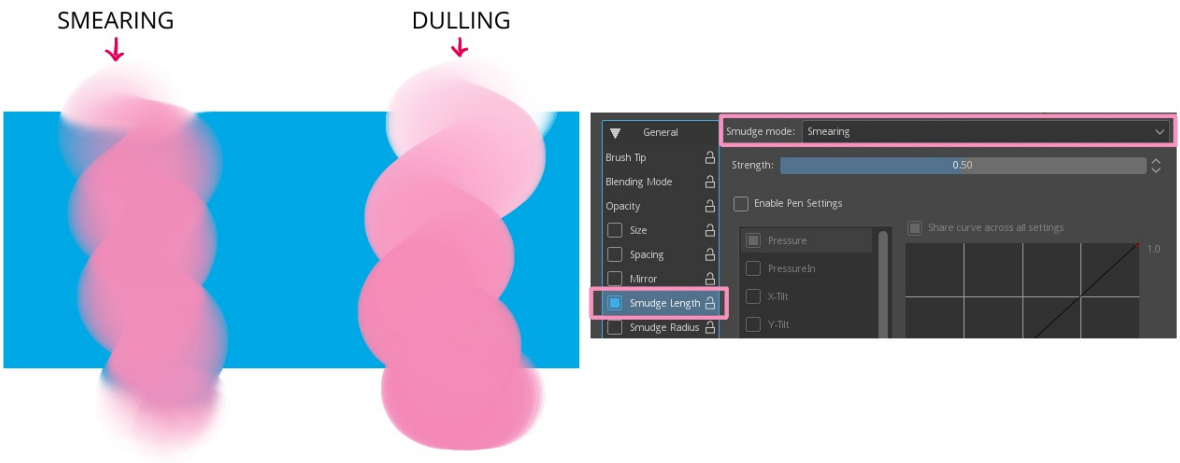
I recommend at least skimming over the first part to get an idea of what does what.

Overview and settings

Overview: Smearing and Dulling

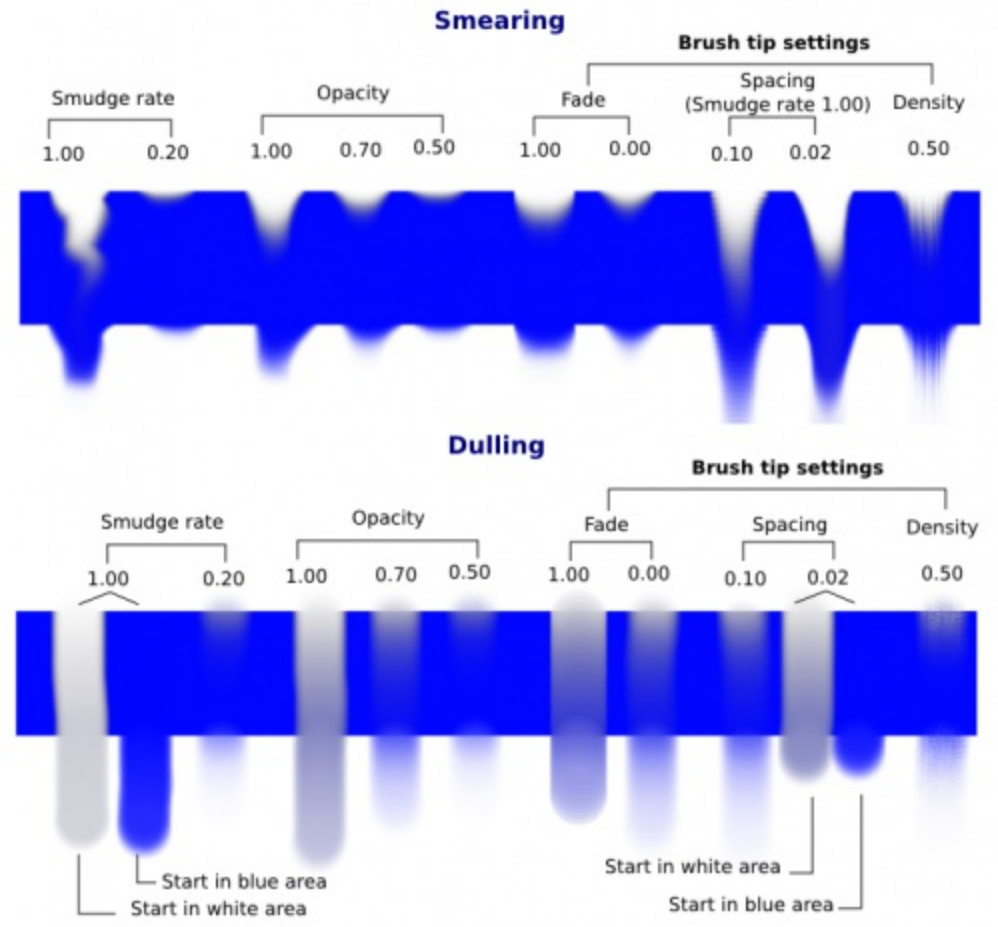
The Color Smudge Brush offers 2 modes, accessible from the *Smudge Rate* section:

- Smearing: This mode mixes colors by smudging (“smearing”) the area underneath.
- Dulling: In his mode, the brush “picks up” the color underneath it, mixes it with its own color, then paints with it.



Smudge Length

To better demonstrate the smudge function, I turned the color rate function off.



Common behaviors:

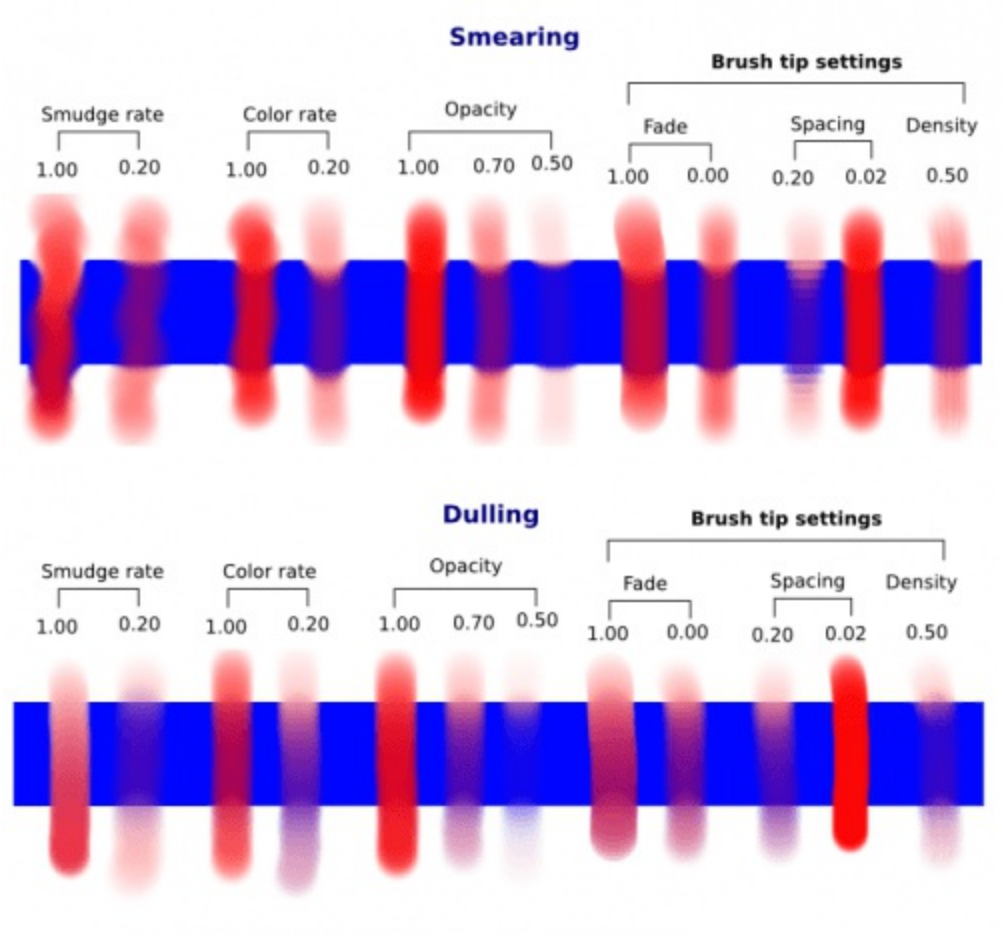
- Unchecking the smudge rate function sets smudge rate to 1.00 (not 0.00).
- Opacity: Below 0.50, there is practically no smudging left: keep opacity over 0.50.

Differences:

- Spacing with Smearing: the lower the spacing, the smoother the effect, so for smearing with a round brush you may prefer a value of 0.05 or less. Spacing affects the length of the smudge trail, but to a much lesser extent. The “strength” of the effect remains more or less the same however.
- Spacing with Dulling: the lower the spacing, the stronger the effect: lowering the spacing too much can make the dulling effect too strong (it picks up a color and never lets go of it). The length of the effect is also affected.
- Both Smearing and Dulling have a “smudge trail”, but in the case of Dulling, the brush shape is preserved. Instead the trail determines how fast the color it picked up is dropped off.

The other settings should be pretty obvious from the pictures, so I’ll spare you some walls of text.

Color Rate, Gradient and Blending modes



Again, most of the settings behaviors should be obvious from the pictures. Just remember to keep *Opacity* over 0.50.

Brush tips

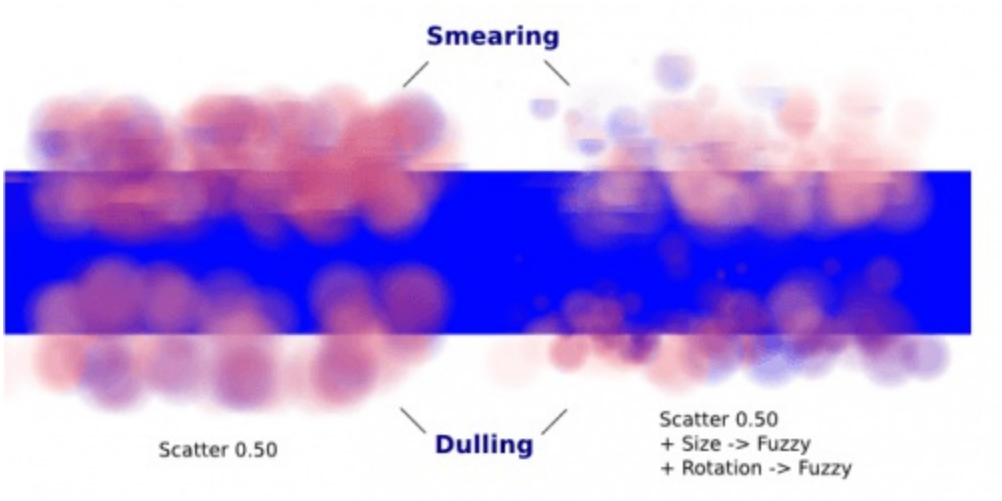
The Color Smudge Brush has all the same brush tip options as the Pixel Brush!

Just remember that the smudge effects are weaker when a brush tip's opacity is lower, so for low-opacity brush tips, increase the opacity and smudge/color rates.

Scatter and other shape dynamics

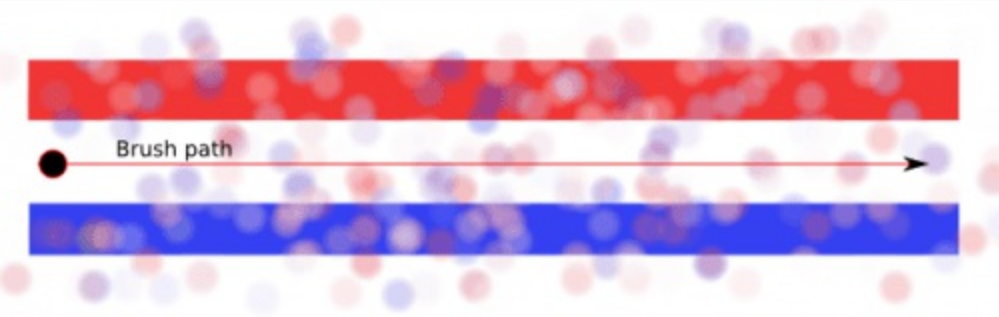
The Color Smudge Brush shares the following dynamics with the Pixel Brush: Opacity, Size, Spacing, Rotation, and Scatter.

However, because of the Smudge effects, the outcome will be different from the Pixel Brush. In particular, the Scatter option becomes much more significant.



A few things to note:

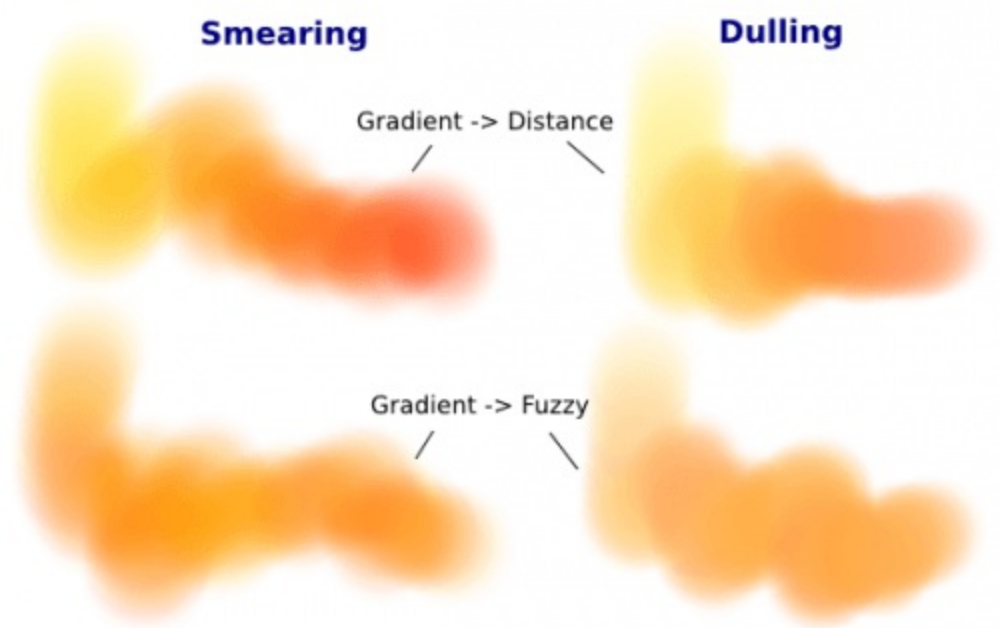
- Scattering is proportional to the brush size. It's fine to use a scattering of 5.00 for a tiny round brush, but for bigger brushes, you may want to get it down to 0.50 or less.
- You may notice the lines with the *Smearing* option. Those are caused by the fact that it picked up the hard lines of the rectangle.
- For scattering, the brush picks up colors within a certain distance, not the color directly under the paintbrush:



Other color behaviors: Gradient, Blending modes, Overlay mode

Gradient

Gradient is equivalent to the *Source* ▸ *Gradient* and *Color* ▸ *Mix* for the Pixel brush: the color will vary between the colors of the gradient.



You can either:

- Leave the default *Foreground* ▸ *Background gradient* setting, and just change the foreground and background colors
- Select a more specific gradient
- Or make custom gradients.

Blending Modes

Blending Modes work just like with the Pixel Brush. The color used though is the color from Color rate.

Color Blending modes with the smudge brush are even harder to predict than with the pixel brush, so I'll leave you to experiment on your own.

Overlay Mode

By default, the Color Smudge Brush only takes information from the layer it is on. However, if you want it to take color information from all the layers, you can turn on the Overlay mode.

Be aware though, that it does so by "picking up" bits of the layer underneath, which may mess up your drawing if you later make changes to the layer underneath.

Use cases: Smudging and blending

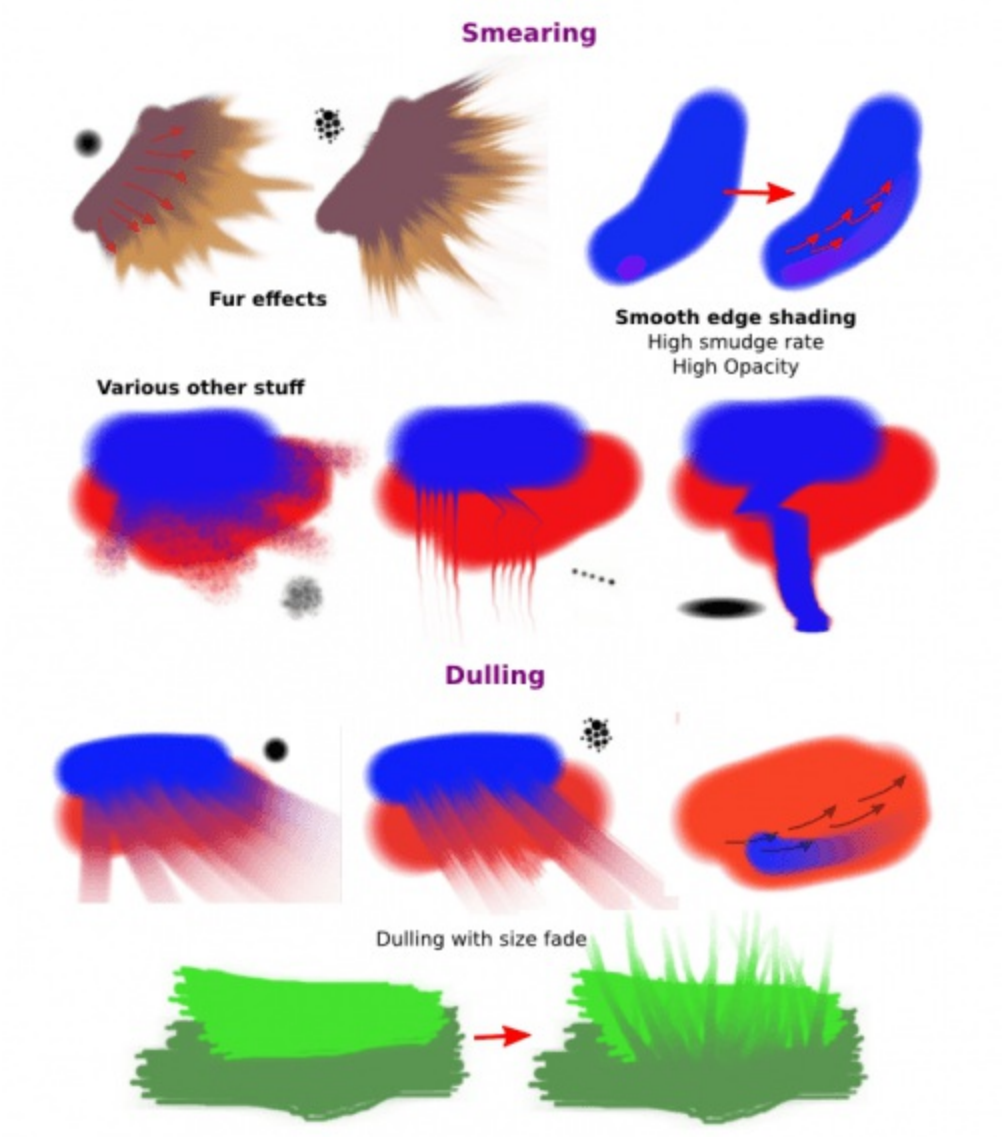
This part describes use cases with color rate off.

I won't explain the settings for dynamics in detail, as you can find the explanations in the Pixel Brush tutorial.

Smudging effects

For simple smudging:

- Pick the Color Smudge Brush. You can use either Smearing or Dulling.
- Turn off Color Rate
- Smudge away



When using lower opacity brush tips, remember to “compensate” for the less visible effects by increasing both *Smudge Rate* and *Opacity*, if necessary to maximum.

Some settings for Smearing

- For smoother smearing, decrease spacing. Remember that spacing is proportional to brush tip size. For a small round brush, 0.10 spacing is fine, but for mid-sized and large brushes, decrease spacing to 0.05 or less.

Some settings for Dulling

- Lowering the spacing will also make the smudging effect stronger, so find a right balance. 0.10 for most mid-sized round brushes should be fine.
- Unlike Smearing, Dulling preserves the brush shape and size, so it won't "fade off" in size like Smearing brushes do. You can mimic that effect through the simple size fade dynamic.

Textured blending

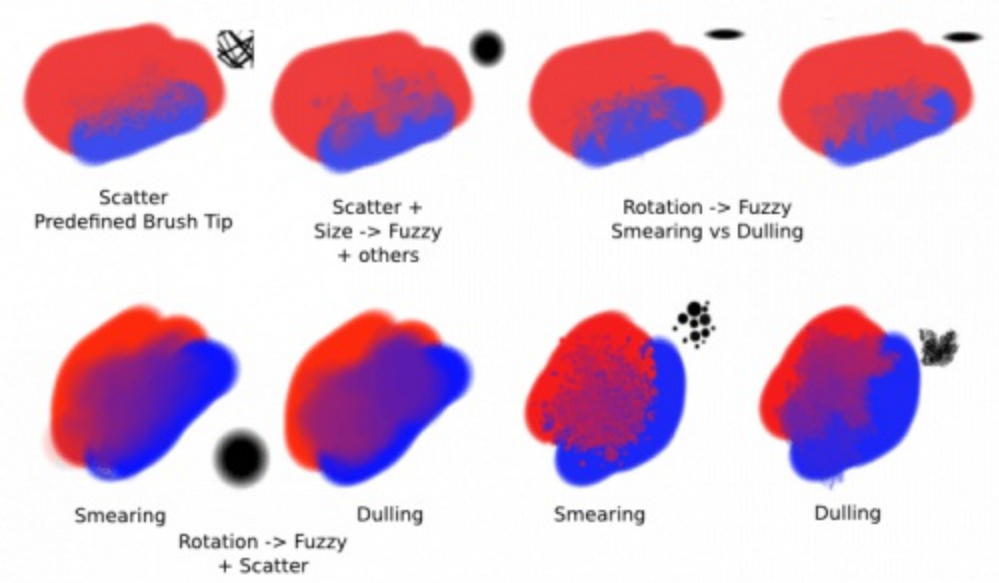
In this case, what I refer to as "Blending" here is simply using one of the following two dynamics:

- *Rotation* set to *Distance* or *Fuzzy*
- And/or Scatter:
    - For most mid-sized brushes you will probably want to lower the scatter rate to 0.50 or lower. Higher settings are okay for tiny brushes.
    - Note that Scatter picks colors within a certain distance, not the color directly under the brush (see Brush Tips).

- Optional: Pile on size and other dynamics and vary brush tips. In fact, the Color Smudge brush is not a blur brush, so smudging is not a very good method of "smooth" blending. To blend smoothly, you'll have better luck with:
- Building up the transition by painting with intermediate values, described later
- Or using the "blur with feathered selection" method that I'll briefly mention at the end of this tutorial.

I've tried to achieve smooth blending with Color Smudge brush by adding rotation and scatter dynamics, but honestly they looked like crap.

However, the Color Smudge brush is very good at "textured blending":



Basically you can paint first and add textured transitions after.

Use cases: Coloring

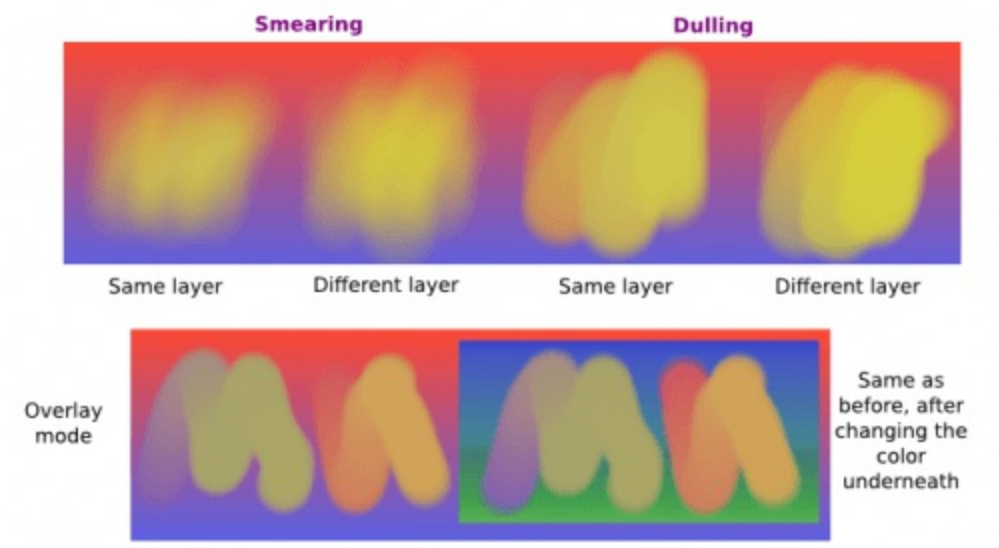
For this last section, *Color Rate* is on.

Layer options

Before we get started, notice that you have several possibilities for your set up:

- Shading on the same layer
- Shading on a separate layer, possibly making use of alpha-inheritance. The brush blends with the transparency of the layer it's on. This means:
    - If the area underneath is more of less uniform, the output is actually similar as if shading on the same layer
        - But if the area underneath is not uniform, then you'll get fewer color variations.
- Shading on a separate layer, using Overlay mode. Use this only if you're fairly sure you don't need to adjust the layer below, or the colors may

become a mess.

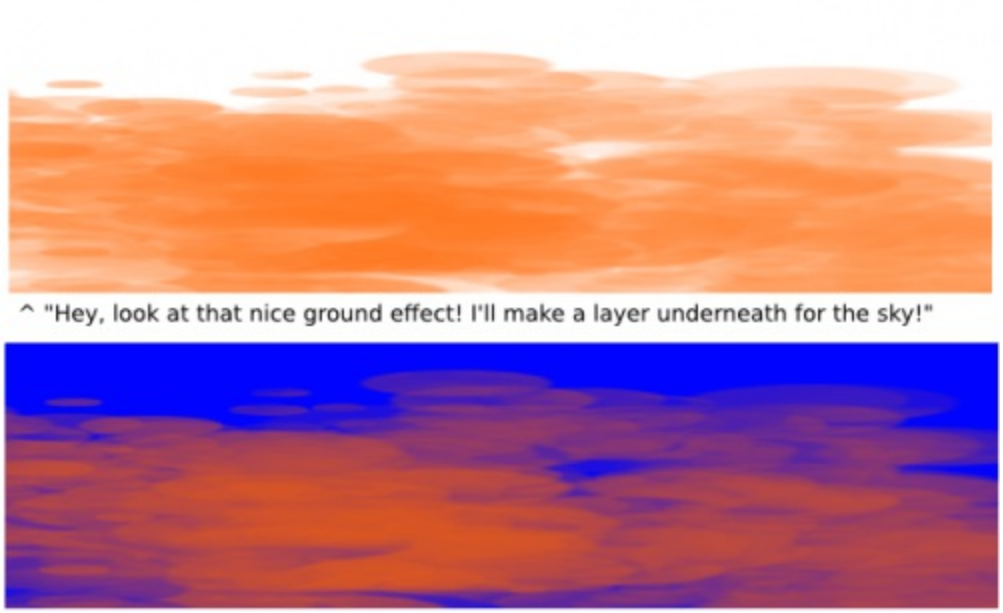


Issue with transparency

The Color Smudge Brush blends with transparency. What this means is that when you start a new, transparent layer and “paint” on this layer, you will nearly always get less than full opacity.

Basically:

- It may look great when you’re coloring on a blank canvas
- But it won’t look so great when you add something underneath



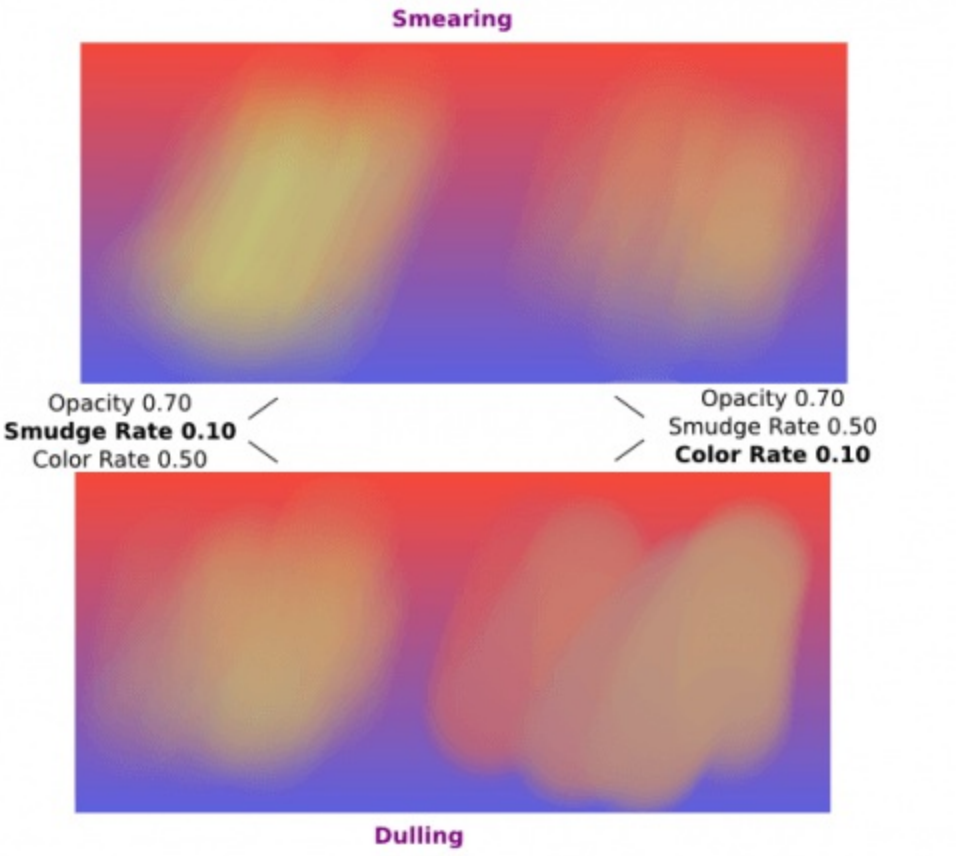
The solution is pretty simple though:

- Make sure you have the area underneath colored in first:
    - With tinting, you already have the color underneath colored, so that’s done
    - For painting, roughly color in the background layer first
    - Or color in the shape on a new layer and make use of alpha-inheritance
- For the last solution, use colors that contrast highly with what you’re using for best effect. For example, shade in the darkest shadow area first, or the lightest highlights, and use the color smudge brush for the contrasting color.

Soft-shading

Suppose you want more or less smooth color transitions. You can either:

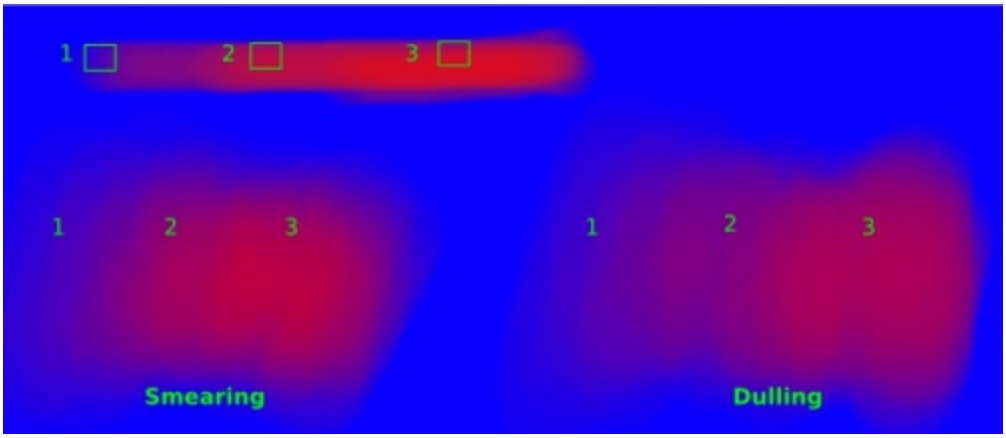
- *Color Rate* as low as 0.10 for round brushes, higher with non fully opaque brush tips.
- Or set the *Smudge Rate* as low as 0.10 instead.
- Or a combination of the two. Please try yourself for the output you like best.
- Optional: turn on *Rotation* for smoother blending.
- Optional: turn on *Scatter* for certain effects.
- Optional: fiddle with *Size* and *Opacity* dynamics as necessary.



This remains, in fact, a so-so way of making smooth transitions. It’s best to build up intermediate values instead. Here:

- I first passed over the blue area three times with a red color. I select 3 shades.
- I color picked each of these values with the `Ctrl` +  shortcut, then

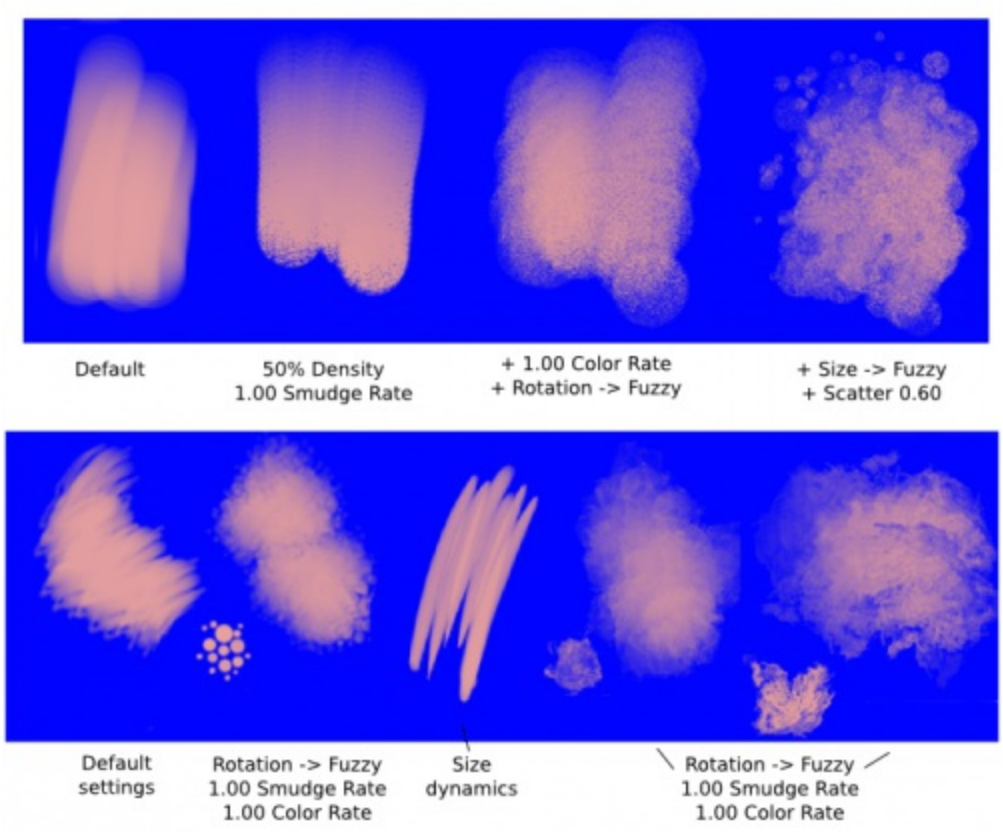
used them in succession.



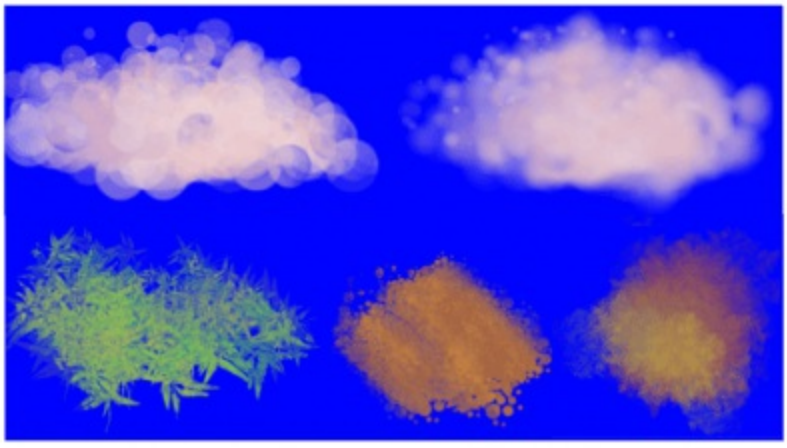
Painting: thick oil style

Many of the included color smudge brush presets produce a thick oil paint-like effect. This is mainly achieved with the Smearing mode on. Basically:

- Smearing mode with high smudge and color rates
    - Both at 0.50 are fine for normal round brushes or fully opaque predefined brushes
    - Up to 1.00 each for brushes with less density or non fully-opaque predefined brushes
- Add Size/Rotation/Scatter dynamics as needed. When you do this, increase smudge and color rates to compensate for increased color mixing.

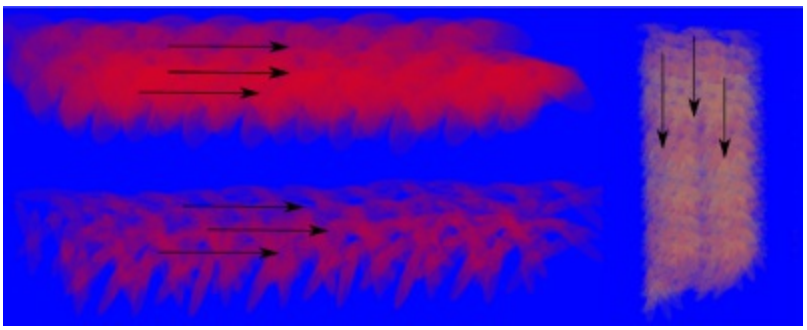


One thing I really like to do is to set different foreground and background colors, then turn on *Gradient* ▸ *Fuzzy*. Alternatively, just paint with different colors in succession (bottom-right example).



Here's some final random stuff. With pixel brushes, you can get all sorts of frill designs by using elongated brushes and setting the dynamics to rotation. You won't get that with Color Smudge Brushes. Instead you'll get something

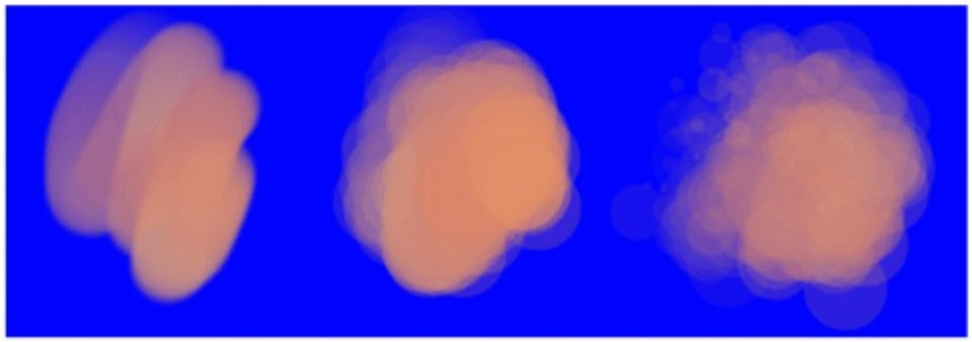
that looks more like… yarn. Which is cool too. Here, I just used oval brushes and *Rotation* ▸ *Distance*.



Painting: Digital watercolor style

When I say “digital watercolor”, it refers to a style often seen online, i.e. a soft, smooth shading style rather than realistic watercolor. For this you mostly need the Dulling mode. A few things:

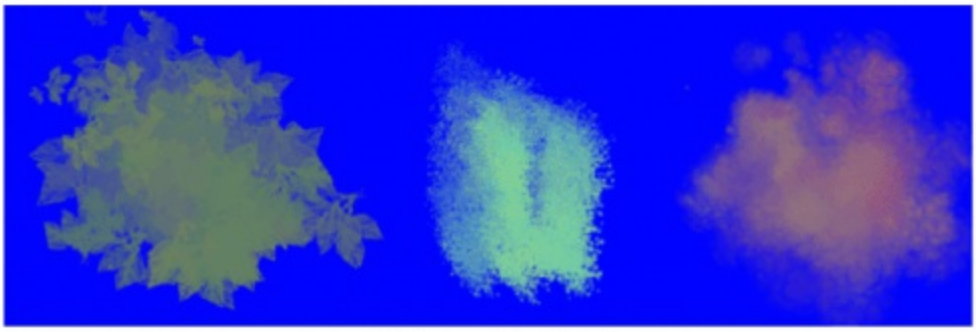
- Contrary to the Smearing mode, you may want to lower opacity for normal round brushes to get a smoother effect, to 0.70 for example.
- Vary the brush tip fade value as well.
- When using *Scatter* or other dynamics, you can choose to set smudge and color values to high or low values, for different outcomes.



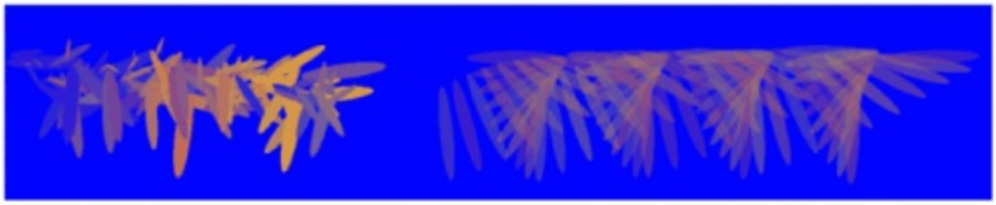
Smudge/Color rates 0.50
Opacity 0.70
Gradient -> Fuzzy

+ Rotation -> Fuzzy

+ Size -> Fuzzy
+ Scatter 0.60



Various predefined brushes
1.00 Smudge rate, Color rate and Opacity
Size and Rotation -> fuzzy



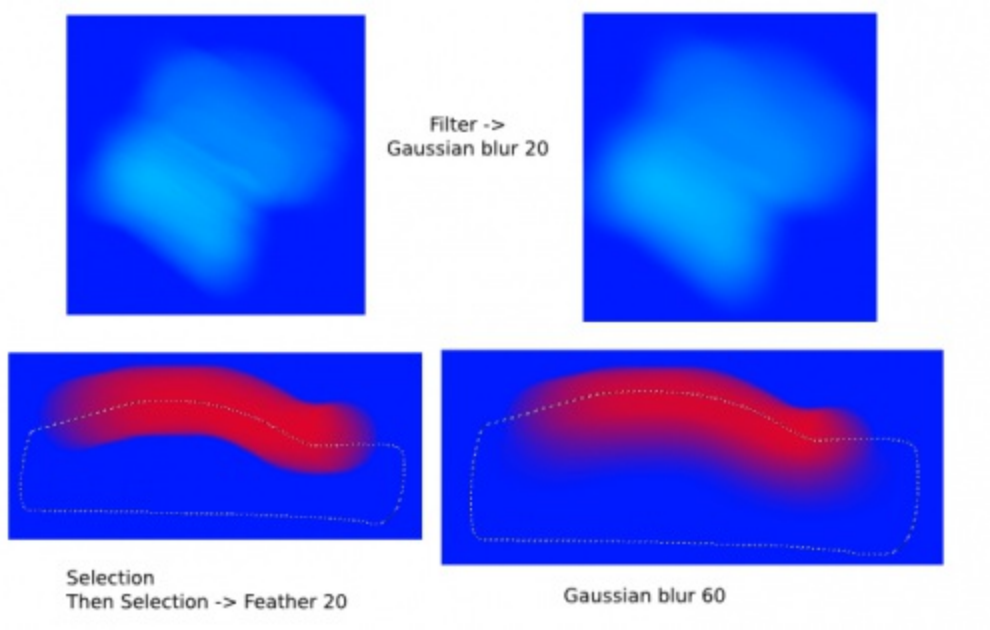
Some rotation effects with elliptical brush

Blurring

You can:

- Paint then smudge, for mostly texture transitions
- Or build up transitions by using intermediate color values

If you want even smoother effects, well, just use blur. Gaussian blur to be exact.



And there you go. That last little trick concludes this tutorial.

Curve Brush Engine

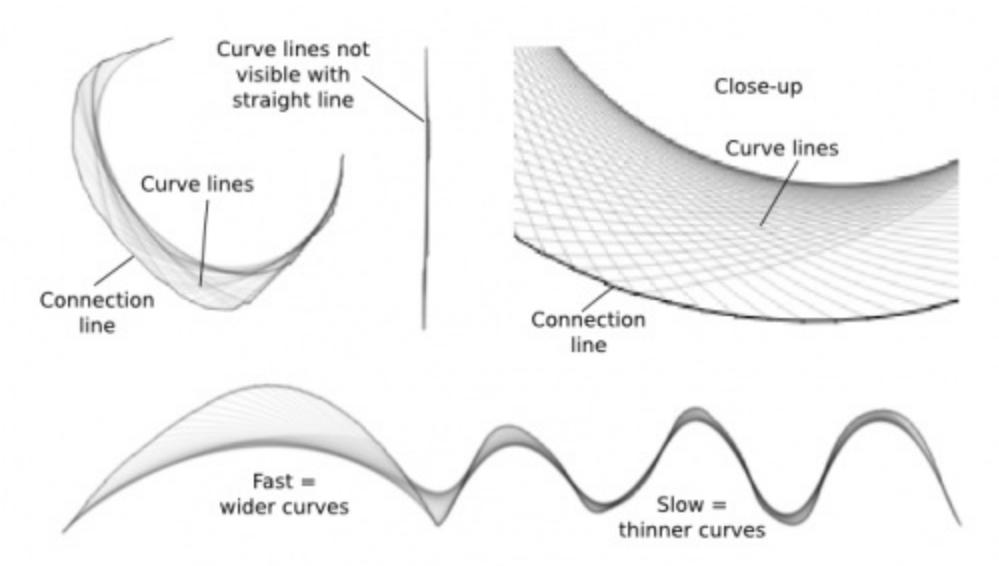
☾

The curve brush is a brush engine which creates strokes made of evenly spaced lines. It has, among other things been used as a replacement for pressure sensitive strokes in lieu of a tablet.

Settings

First off, the line produced by the Curve brush is made up of 2 sections:

- The connection line, which is the main line drawn by your mouse.
- The curve lines I think, which are the extra fancy lines that form at curves. The curve lines are formed by connecting one point of the curve to a point earlier on the curve. This also means that if you are drawing a straight line, these lines won't be visible, since they'll overlap with the connection line. Drawing faster gives you wider curves areas.

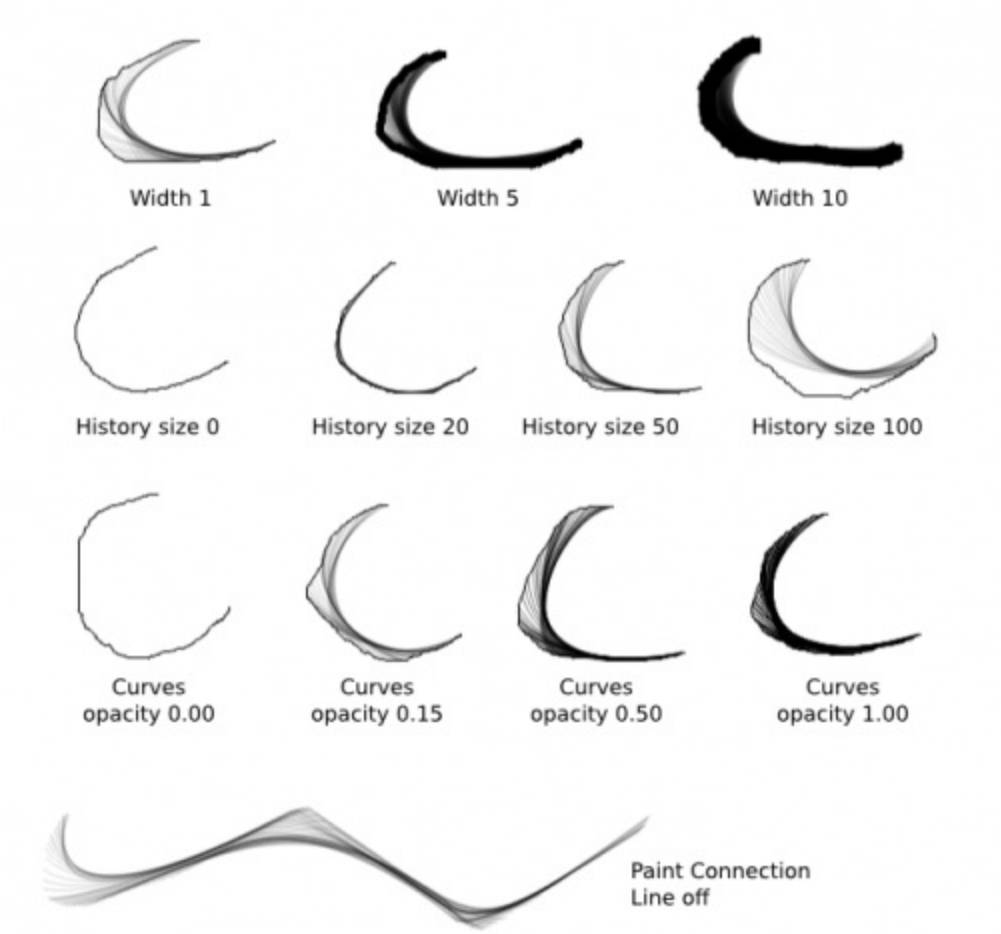


You have access to 3 settings from the Lines tab, as well as 2 corresponding dynamics:

- Line width: this applies to both the connection line and the curve lines.
    - Line width dynamics: use this to vary line width dynamically.
- History size: this determines the distance for the formation of curve lines.
    - If you set this at low values, then the curve lines can only form over a small distances, so they won't be too visible.
    - On the other hand, if you set this value too high, the curve lines will only start forming relatively "late".
    - So in fact, you'll get maximum curve lines area with a mid-value of say… 40~60, which is about the default value. Unless you're drawing at really high resolutions.
- Curves opacity: you can't set different line widths for the connection line and the curve lines, but you can set a different opacity for the curve lines. With low opacity, this will produce the illusion of thinner curve lines.
    - Curves opacity dynamics: use this to vary Curves opacity dynamically.

In addition, you have access to two checkboxes:

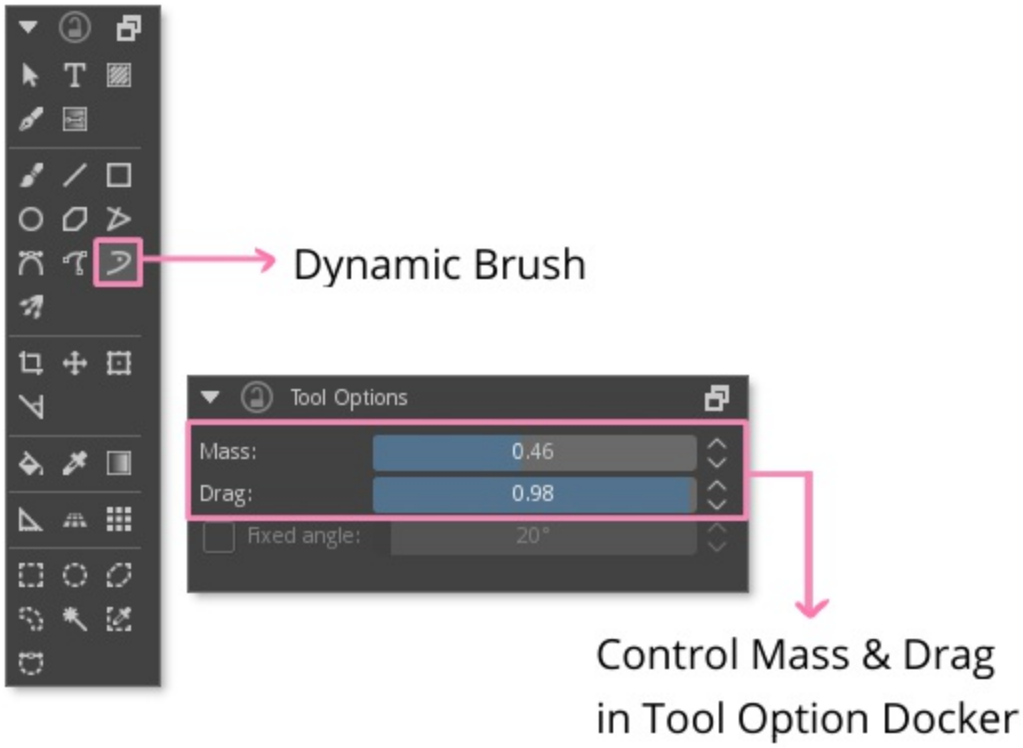
- Paint connection line, which toggles the visibility of the connection line.
- Smoothing, which… I have no idea actually. I don't see any differences with or without it. Maybe it's for tablets?



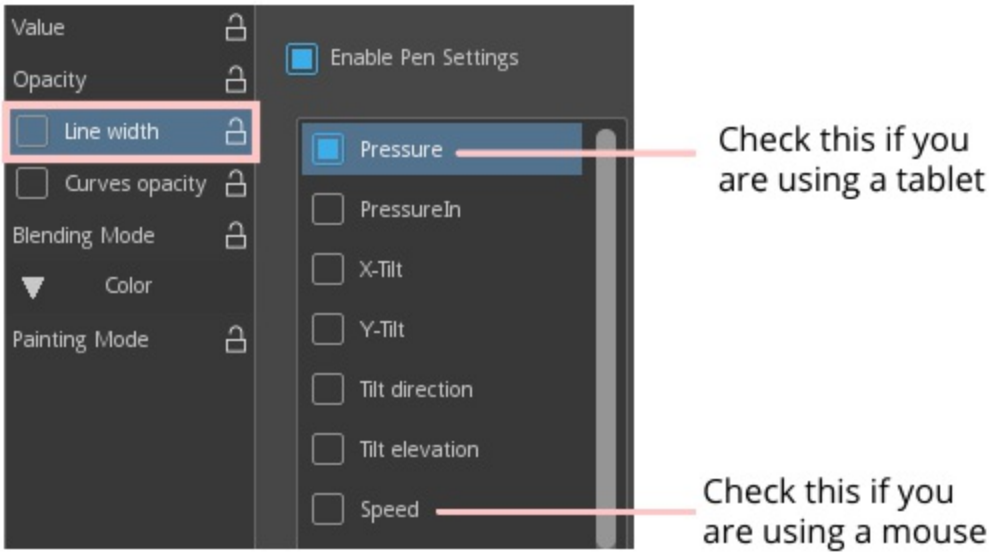
Drawing variable-width lines

And here’s the only section of this tutorial that anyone cares about: pretty lineart lines! For this:

- Use the Draw Dynamically mode: I tend to increase drag to at least 50. Vary Mass and Drag until you get the feel that’s most comfortable for you.



- Set line width to a higher value (ex.: 5), then turn line width dynamics on:
    - If you're a tablet user, just set this to Pressure (this should be selected by default so just turn on the Line Width dynamics). I can't check myself, but a tablet user confirmed to me that it works well enough with Draw Dynamically.
    - If you're a mouse user hoping to get variable line width, set the Line Width dynamics to Speed.



- Set Curves opacity to 0: This is the simplest way to turn off the Curve lines. That said, leaving them on will get you more “expressive” lines.

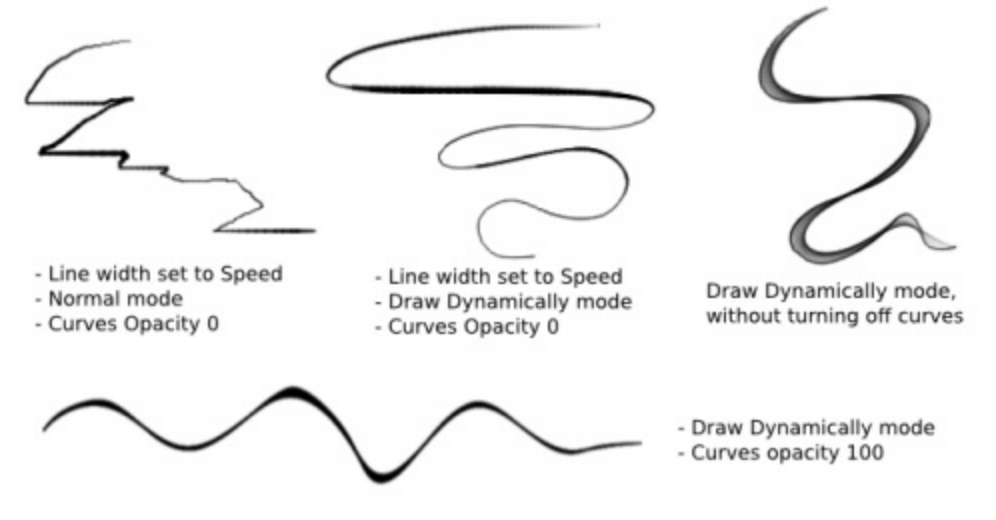
Additional tips:

- Zig-zag a lot if you want a lot of extra curves lines.
- Use smooth, sweeping motions when you’re using Draw Dynamically with Line Width set to Speed: abrupt speed transitions will cause abrupt size transitions. It takes a bit of practice, and the thicker the line, the more visible the deformities will be. Also, zoom in to increase control.
- If you need to vary between thin and thick lines, I suggest creating presets of different widths, since you can’t vary the base line width from the canvas.

Alternative:

- Use the Draw Dynamically mode
- Set Curves opacity to 100
- Optionally decrease History size to about 30

The curve lines will fill out the area they cover completely, resulting in a line with variable widths. Anyway, here are some comparisons:



And here are examples of what you can do with this brush:

Deform Brush Engine

The Deform Brush is a brush that allows you to pull and push pixels around. It's quite similar to the Liquify, but where liquify has higher quality, the deform brush has the speed.

Options

- Brush Tips
- Deform Options
- Blending Modes
- Opacity and Flow
- Size
- Rotation
- Airbrush

Deform Options

1: undeformed, 2: Move, 3: Grow, 4: Shrink, 5: Swirl Counter Clock Wise, 6: Swirl Clockwise, 7: Lens Zoom In, 8: Lens Zoom Out

These decide what strangeness may happen underneath your brush cursor.

Grow
: This bubbles up the area underneath the brush-cursor.

Shrink
: This pinches the Area underneath the brush-cursor.

Swirl Counter Clock Wise
: Swirls the area counter clock wise.

Swirl Clock Wise
: Swirls the area clockwise.

Move
: Nudges the area to the painting direction.

Color Deformation

This seems to randomly rearrange the pixels underneath the brush.

Lens Zoom In

Literally paints a enlarged version of the area.

Lens Zoom Out

Paints a minimized version of the area.

Showing color deform.

Deform Amount

Defines the strength of the deformation.

Bilinear Interpolation

Bilinear Interpolation

Smoothens the result. This causes calculation errors in 16bit.

Use Counter

Slows down the deformation subtlety.

Without ‘use undeformed’ to the left and with to the right.

Use Undeformed Image

Samples from the previous version of the image instead of the current. This works better with some deform options than others. Move for example seems to almost stop working, but it works really well with Grow.

Dyna Brush Engine

Dyna brush uses dynamic setting like mass and drag to draw strokes. The results are fun and random spinning strokes. To experiment more with this brush you can play with values in ‘dynamic settings’ section of the brush editor under Dyna Brush.

Deprecated since version 4.0: This brush engine has been removed in 4.0. This engine mostly had smoothing results that the dyna brush tool has in the toolbox. The stabilizer settings can also give you further smoothing options from the tool options.

Options

- Brush Size (Dyna)
- Blending Modes
- Opacity and Flow
- Airbrush

Brush Size (Dyna)

Dynamics Settings

Initial Width
: Initial size of the dab.

Mass
: How much energy there is in the satellite like movement.

Drag
: How close the dabs follow the position of the brush-cursor.

Width Range
: How much the dab expands with speed.

Shape

Diameter
: Size of the shape.

Angle
: Angle of the shape. Requires Fixed Angle active to work.

Circle
: Make a circular dab appear.

Two
: Draws an extra circle between other circles.

Line
: Connecting lines are drawn next to each other. The number boxes on the right allows you to set the spacing between the lines and how many are drawn.

Polygon
: Draws a black polygon as dab.

Wire
: Draws the wireframe of the polygon.

Paint Connection
: Draws the connection line.

Filter Brush Engine

Where in other programs you have a ‘dodge tool’, ‘blur tool’ and ‘sharpen tool’, Krita has a special brush engine for this: The Filter Brush engine. On top of that, due to Krita’s great integration of the filters, a huge amount of filters you’d never thought you wanted to use for a drawing are possible in brush form too!

Options

The filter brush has of course some basic brush-system parameters:

- Brush Tips
- Blending Modes
- Opacity and Flow
- Size
- Mirror
- Rotation

Grid Brush Engine

The grid brush engine draws shapes on a grid. It helps you produce retro and halftone effects.

If you're looking to setup a grid for snapping, head to Grids and Guides Docker.

Options

- Brush Size
- Particle Type
- Blending Modes
- Opacity and Flow
- Color Options

Brush Size

Grid Width
: Width of the cursor area.

Grid Height
: Height of the cursor area.

Division
: Subdivides the cursor area and uses the resulting area to draw the particles.

Division by pressure
: The more you press, the more subdivisions. Uses Division as the finest subdivision possible.

Scale

Scales up the area.

Vertical Border
: Forces vertical borders in the particle space, between which the particle needs to squeeze itself.

Horizontal Border
: Forces a horizontal borders in the particle space, between which the particle needs to squeeze itself.

Jitter Borders
: Randomizes the border values with the Border values given as maximums.

Particle Type

Decides the shape of the particle.

Ellipse
: Fills the area with an ellipse.

Rectangle
: Fills the area.

Line
: Draws lines from the lower left to the upper right corner of the particle.

Pixel
: Looks like an aliased line on high resolutions.

Anti-aliased Pixel
: Fills the area with little polygons.

Color Options

Random HSV
: Randomize the HSV with the strength of the sliders. The higher, the more

the color will deviate from the foreground color, with the direction indicating clock or counter clockwise.

Random Opacity
: Randomizes the opacity.

Color Per Particle
: Has the color options be per particle instead of area.

Sample Input Layer
: Will use the underlying layer as reference for the colors instead of the foreground color.

Fill Background
: Fills the area before drawing the particles with the background color.

Mix with background color
: Gives the particle a random color between foreground/input/random HSV and the background color.

Hatching Brush Engine

When I first tried this brush, my impression of it was “plain parallel lines” (and the award for most boring brush goes to…). Fortunately, existing presets gave me an idea of the possibilities of this brush.

Settings

Brush tip

The brush tip simply defines the area where the hatching will be rendered.

- Transparent brush tip areas give more transparent hatching, but as with a normal brush, passing over the area again will increase opacity.
- The hatching itself is mostly fixed in location, so drawing with a hatching brush usually acts more like “revealing” the hatching underneath than drawing with brushes of parallel lines. The exception is for *Moiré pattern* with *Crosshatching* dynamics on.
- Vary the brush shape or texture for a variety of effects. Decreasing the density of the autobrush will give a grainy texture to your hatching, for example.
- The *Size* dynamic affects the brush tip, not the hatching thickness.

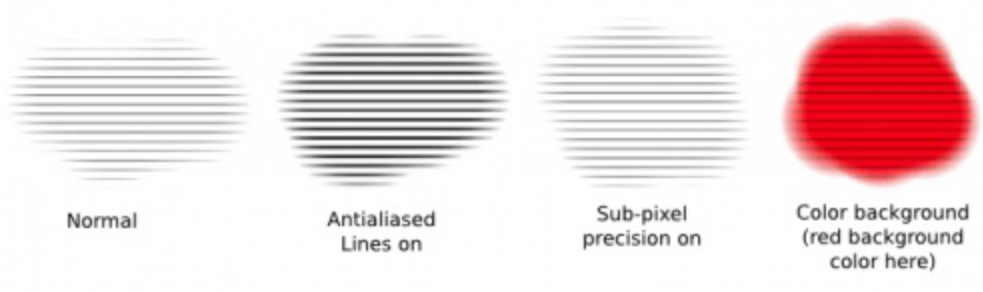
Hatching preferences

Before going on: at the time of this writing, there is a bug that causes line thickness to not vary on default settings. To get around this, go to *Hatching preferences* and check *Antialiased Lines*. Pentalis is aware of this issue so the bug may get fixed soon.

The three options are:

- *Antialiased lines*: This controls aliasing. If changing line thickness isn't working, check this option and it should work, because it switches to a different algorithm.
- *Subpixel precision*: I'm guessing this affects the rendering quality, but you won't see much of a difference. Check this if you want to.
- *Color background*: Checking this will color in the background at the back of the hatching.

The output is slightly different depending on whether the first two options are checked, but the difference isn't enough for you to worry about. I recommend just keeping the first two options checked.



Hatching options

This is where the main hatching options go. They're intuitive enough:

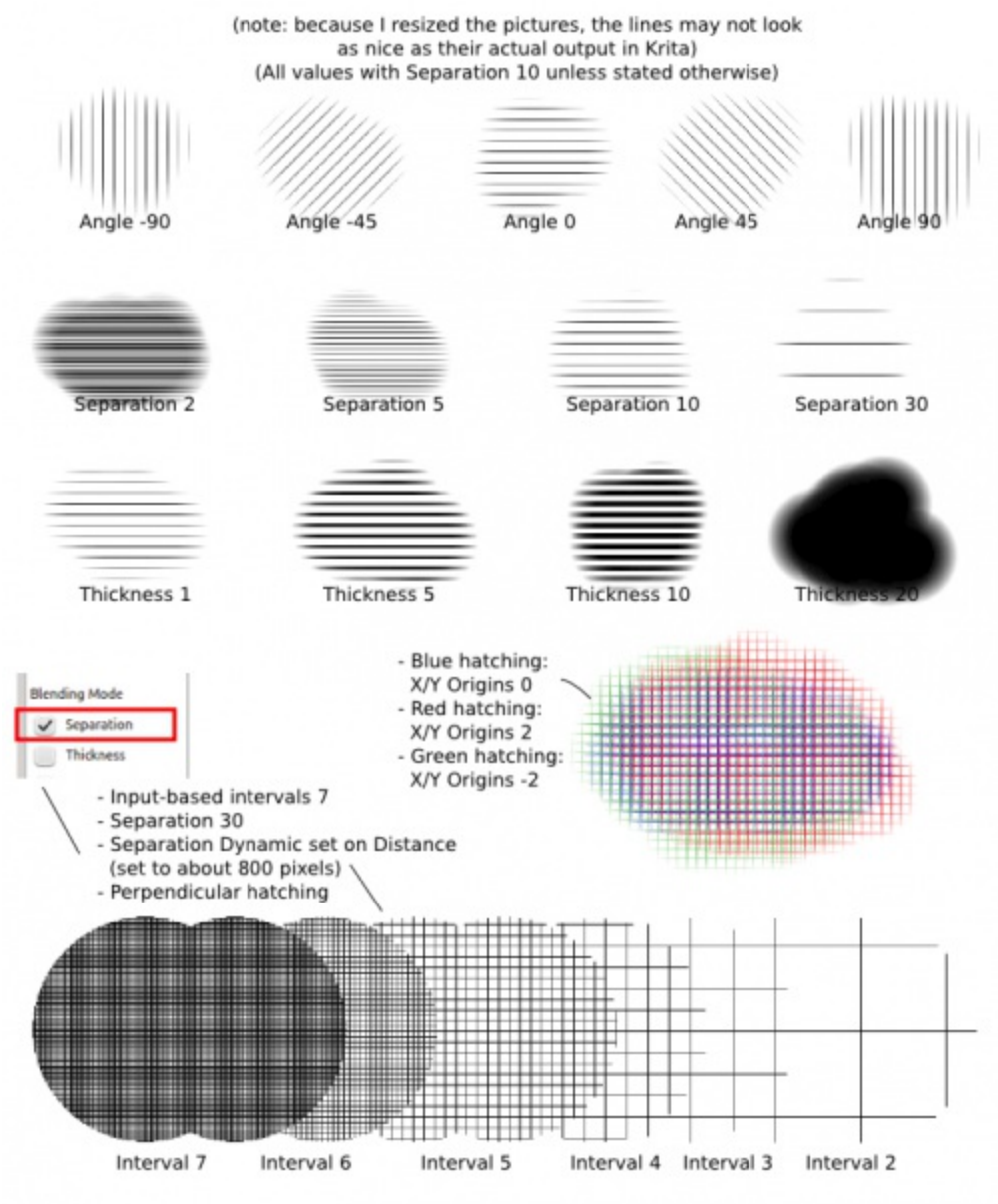
- Angle: The angle of the hatching.
- Separation: This is the distance between the centers of the lines.
    - Use a value of 2 pixels or higher, or the lines won't be distinct

anymore.
    - The Separations dynamic doesn’t actually assign random values to Separation, instead it will take the value in “Input-based intervals” to divide the grid further. “Input-based intervals” can take values between 2 and 7.

- Thickness: The line thickness.

    - Actually, this is the thickness of the line + blank area, so the line itself has a thickness of half this value.
    - If you use the same separation value and the same line thickness value, then the lines and the area between them will be of the same thickness.
    - You can vary this value dynamically with the Thickness dynamics.
    - If the line thickness isn’t changing for you, go to Hatching Preferences and check “Antialiased Lines”.

- Origin X and Origin Y: The hatching has a fixed location, painting acts as though you’re revealing the existing hatching underneath. To nudge the hatching, you can tweak these two values. You can get various grid effects this way.

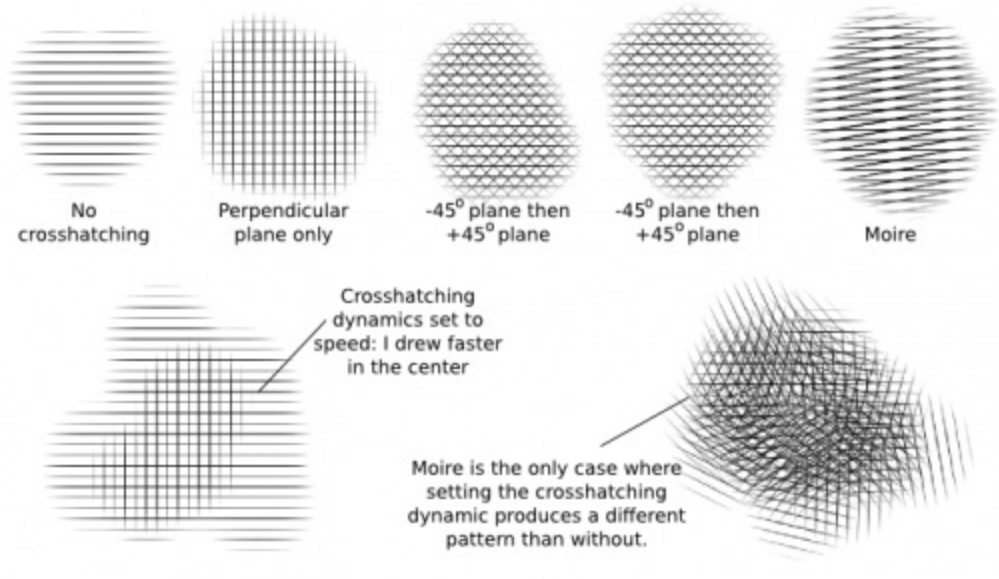


Finally, we have the hatching styles:

- No crosshatching: basic parallel lines
- Perpendicular plane only: grid lines
- -45 degrees plane then +45 degrees plane: see example.
- +45 degrees plane then -45 degrees plane: see example, actually not much different from the above, it’s mostly the order that changes when using dynamics.
- Moiré pattern: See example.

The Crosshatching dynamic only works if you have already chosen a crosshatching style. When that happens, the crosshatching only gets drawn according to the conditions of the dynamics (pressure, speed, angle…).

- With most hatching styles, using crosshatching dynamics basically gets you the same hatching style, minus the occasional line.
- The exception is with Moire, which will produce a different pattern.



Use cases

If you don’t want the edges to be fuzzy, go to Brush Tip and set the Fade values to 1.00. I recommended doing the hatching on a separate layer, then erasing the extra areas.

Now for the uses:

- You can, of course, just use this for completely normal hatching. In versions I’m using, the default Separation is 1, which is too low, so increase Separation to a value between 2 to 10.
- If you find normal hatching too boring, increase the Thickness and set the Thickness dynamic to either Pressure (if you have a tablet) or Speed (if you’re using a mouse). Doesn’t that look more natural? (When using a mouse, pass over the areas where you want thicker lines again while

drawing faster).

- Grittier texture: add some density and/or randomness to your autobrush for a grittier texture.
- You can also set Painting Mode to Build up, and Mode to Multiply, to make some colors have more depth. (see my grid example).
- Vary Origin X and Origin Y while using the same patterns.
- Use the Separations dynamic for more complex patterns. Add in Line Thickness and other dynamics for more effect.
- Now, the Moiré pattern is quite boring on its own, but it is much more interesting with Crosshatching dynamics set on Fuzzy.
- For more texture, set Line Thickness to Fuzzy, decrease Density a bit and increase Randomness and you get a nice gritty texture.

Normal hatching
(*yawn*)

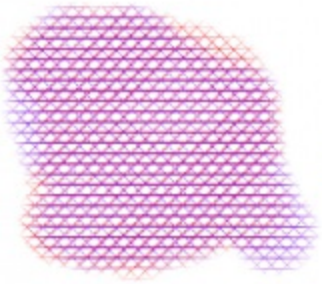
+ Thickness -> Speed
Draw over areas where you
want thicker lines faster

+Perpendicular hatching
+Brush tip: 50% density,
100% randomness, Fade 0
Build-up + Multiply modes

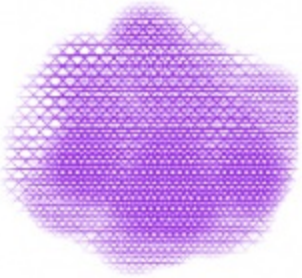
Moire
Crosshatching -> Fuzzy

Brush tip: Density 60%, Fade 0.00
Moire, Separation and Thickness 5
Crosshatching fuzzy

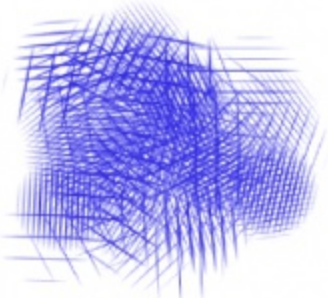
Same as left with
Thickness 1 and
Size -> Fuzzy



-45°plane then +45° plane
Blue grid, Origins X/Y 0
Red grid, Origins X/Y 1



-45°plane then +45° plane
Separations -> Fuzzy



Moire
Separations -> Fuzzy

Particle Brush Engine

A brush that draws wires using parameters. These wires always get more random and crazy over drawing distance. Gives very intricate lines best used for special effects.

Options

Brush Size

Particles
: How many particles there’s drawn.

Opacity Weight
: The Opacity of all particles. Is influenced by the painting mode.

Dx Scale (Distance X Scale)
: How much the horizontal cursor distance affects the placing of the pixel. Is unstable on negative values. 1.0 is equal.

Dy Scale (Distance Y Scale)
: How much the vertical cursor distance affects the placing of the pixel. Is unstable on negative values. 1.0 is equal.

Gravity
: Multiplies with the previous particle’s position, to find the new particle’s position.

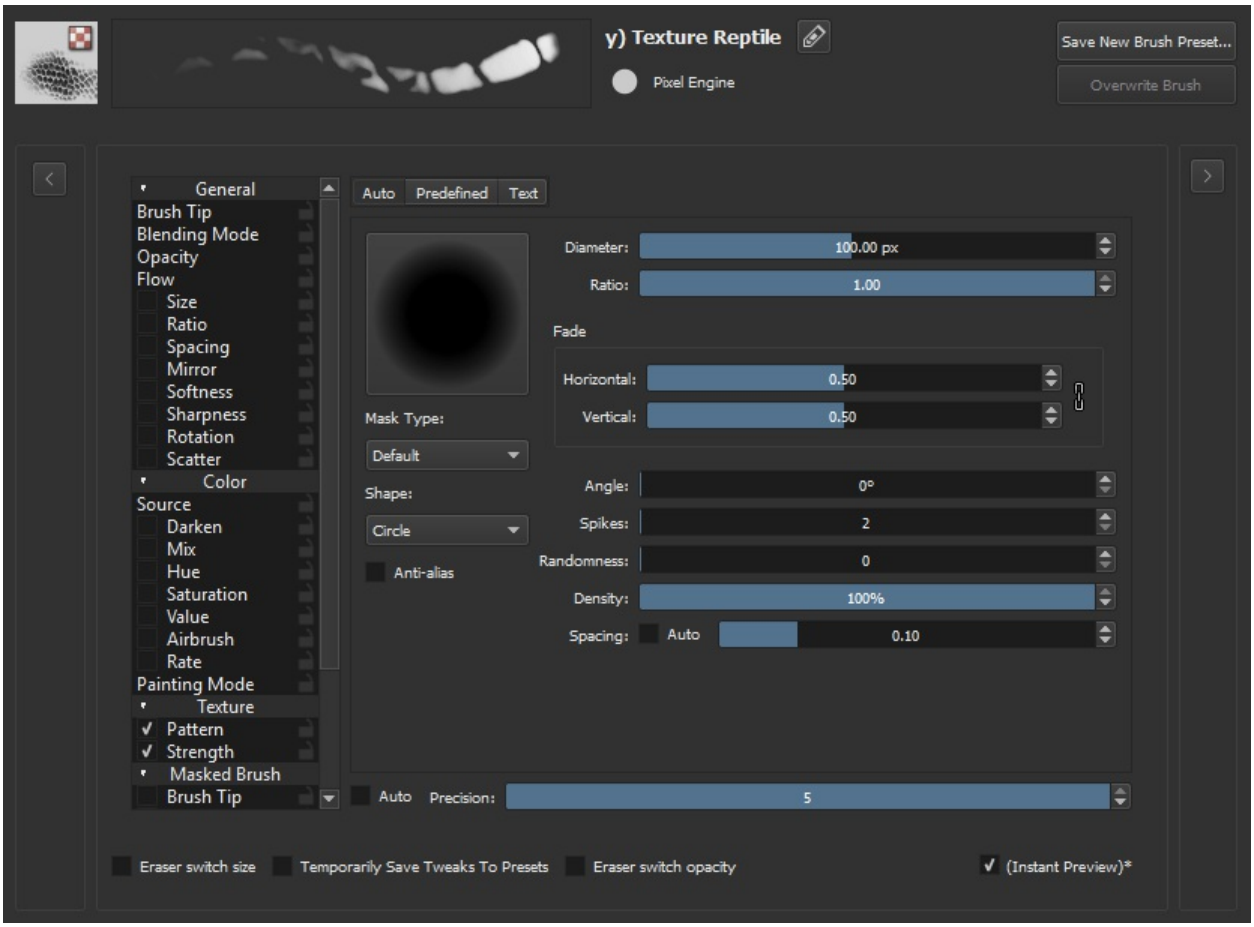
Iterations

The higher, the higher the internal acceleration is, with the furthest away particle from the brush having the highest acceleration. This means that the higher iteration is, the faster and more randomly a particle moves over time, giving a messier result.

Pixel Brush Engine

●

Brushes are ordered alphabetically. The brush that is selected by default when you start with Krita is the *Pixel Brush*. The pixel brush is the traditional mainstay of digital art. This brush paints impressions of the brush tip along your stroke with a greater or smaller density.



Let’s first review these mechanics:

1. Select a brush tip. This can be a generated brush tip (round, square, star-shaped), a predefined bitmap brush tip, a custom brush tip or a text.
2. Select the spacing: this determines how many impressions of the tip will

be made along your stroke.
3. Select the effects: the pressure of your stylus, your speed of painting or other inputs can change the size, the color, the opacity or other aspects of the currently painted brush tip instance – some applications call that a “dab”.
4. Depending on the brush mode, the previously painted brush tip instance is mixed with the current one, causing a darker, more painterly stroke, or the complete stroke is computed and put on your layer. You will see the stroke grow while painting in both cases, of course!

Since 4.0, the Pixel Brush Engine has Multithreaded brush-tips, with the default brush being the fastest mask.

Available Options:

- Brush Tips
- Blending Modes
- Opacity and Flow
- Size
- Ratio
- Spacing
- Mirror
- Softness
- Sharpness
- Rotation
- Scatter
- Source
- Mix
- Airbrush
- Texture
- Masked Brush

Specific Parameters to the Pixel Brush Engine

Darken

Allows you to Darken the source color with Sensors.

The color will always become black in the end, and will work with Plain Color, Gradient and Uniform random as source.

Hue, Saturation, Value

These parameters allow you to do an HSV adjustment filter on the Source and control it with Sensors.

Works with Plain Color, Gradient and Uniform random as source.

Uses

Having all three parameters on Fuzzy will help with rich color texture. In combination with Mix, you can have even finer control.

Quick Brush Engine

A Brush Engine inspired by the common artist's workflow where a simple big brush, like a marker, is used to fill large areas quickly, the Quick Brush engine is an extremely simple, but quick brush, which can give the best performance of all Brush Engines.

It can only change size, blending mode and spacing, and this allows for making big optimisations that aren't possible with other brush engines.

Brush

The only parameter specific to this brush.

Diameter
: The size. This brush engine can only make round dabs, but it can make them really fast despite size.

Spacing
: The spacing between the dabs. This brush engine is particular in that it's faster with a lower spacing, unlike all other brush engines.

See also

Phabricator Task [https://phabricator.kde.org/T3492]

Shape Brush Engine

An Al.chemy inspired brush-engine. Good for making chaos with!

Parameters

Experiment Option

Speed
: This makes the outputted contour jaggy. The higher the speed, the jaggier.

Smooth
: Smoothens the output contour. This slows down the brush, but the higher the smooth, the smoother the contour.

Displace
: This displaces the shape. The slow the movement, the higher the displacement and expansion. Fast movements shrink the shape.

Winding Fill
: This gives you the option to use a ‘non-zero’ fill rules instead of the ‘even-odd’ fill rule, which means that where normally crossing into the shape created transparent areas, it now will not.

Hard Edge
: Removes the anti-aliasing, to get a pixelized line.

Sketch Brush Engine

A line based brush engine, based on the Harmony brushes. Very messy and fun.

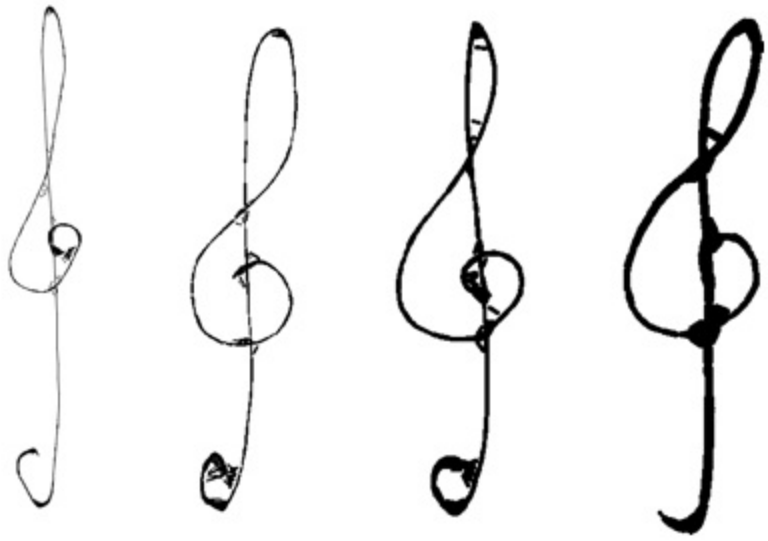
Parameters

Has the following parameters:

- Brush Tips
- Blending Modes
- Opacity and Flow
- Size
- Ratio
- Line Width
- Offset Scale
- Density
- Rotation
- Airbrush

Line Width

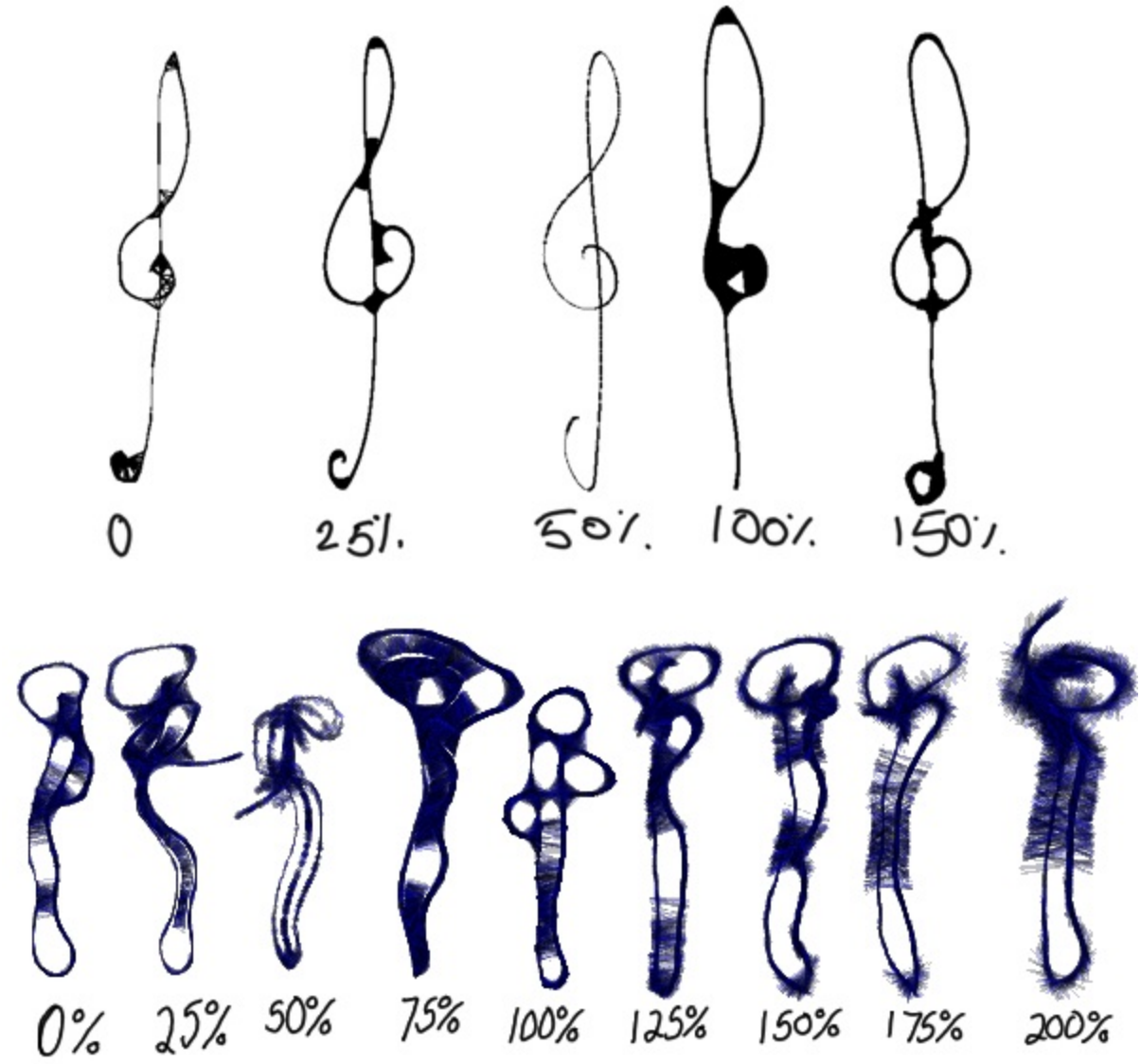
The width of the rendered lines.



Offset Scale

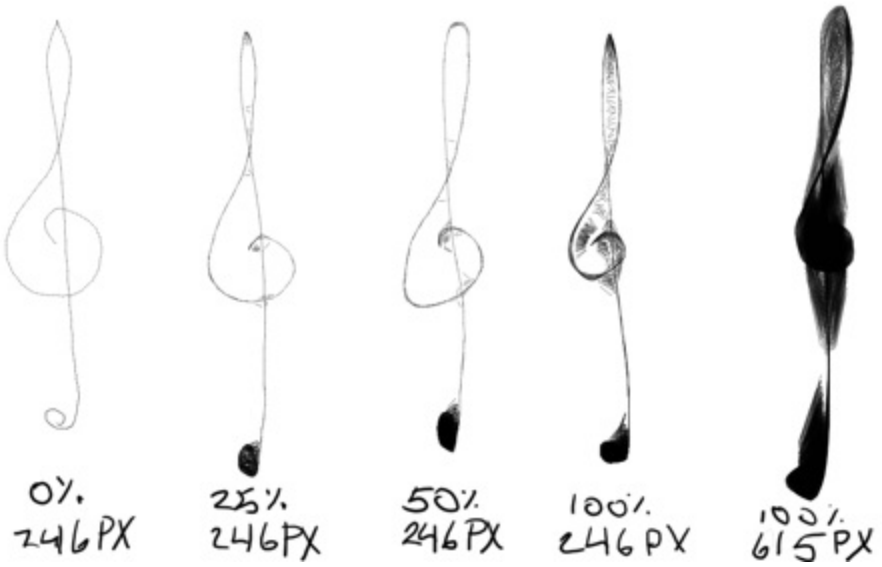
When curve lines are formed, this value roughly determines the distance from the curve lines to the connection lines:

- This is a bit misleading, because a value of 0% and a value of 100% give similar outputs, as do a value of say 30% and 70%. You could think that the actual value range is between 50% and 200%.
- 0% and 100% correspond to the curve lines touching the connection lines exactly.
- Above 100%, the curve lines will go further than the connection lines, forming a fuzzy effect.



Density

The density of the lines. This one is highly affected by the Brush-tip, as determined by the Distance Density toggle.



Use Distance Density

The further the line covered is from the center of the area of effect, the less the density of the resulting curve lines.

Magnetify

Magnetify is *on* by default. It's what causes curve lines to form between two close line sections, as though the curve lines are attracted to them like magnets. With Magnetify *off*, the curve line just forms on either side of the current active portion of your connection line. In other words, your line becomes fuzzier when another portion of the line is nearby, but the lines don't connect to said previous portion.

Random RGB

Causes some slight RGB variations.

Random Opacity

The curve lines get random opacity. This one is barely visible, so for the example I used line width 12 and 100% opacity.

Distance Opacity

The distance based opacity. When you move your pen fast when painting, the opacity will be calculated based on the distance from the center of the effect area.

Simple Mode

This mode exists for performance reasons, and doesn’t affect the output in a visible way. Check this for large brushes or thick lines for faster rendering.

Paint Connection Line

What appears to be the connection line is usually made up of an actual connection line and many smaller curve lines. The many small curve lines make up the majority of the line. For this reason, the only time this option will make a visible difference is if you’re drawing with 0% or near 0% density, and with a thick line width. The rest of the time, this option won’t make a visible difference.

Spray Brush Engine

A brush that can spray particles around in its brush area.

Options

- Spray Area
- Spray Shape
- Brush Tips (Used as particle if spray shape is not active)
- Opacity and Flow
- Size
- Blending Modes
- Shape Dynamics
- Color Options
- Rotation
- Airbrush

Spray Area

The area in which the particles are sprayed.

Diameter
: The size of the area.

Aspect Ratio
: It’s aspect ratio: 1.0 is fully circular.

Angle
: The angle of the spray size: works nice with aspect ratios other than 1.0.

Scale
: Scales the diameter up.

Spacing
: Increases the spacing of the diameter’s spray.

Particles

Count
: Use a specified amount of particles.

Density
: Use a % amount of particles.

Jitter Movement
: Jitters the spray area around for extra randomness.

Gaussian Distribution
: Focuses the particles to paint in the center instead of evenly random over the spray area.

Spray Shape

If activated, this will generate a special particle. If not, the brush-tip will be the particle.

Shape
: Can be…

- Ellipse
- Rectangle
- Anti-aliased Pixel
- Pixel
- Image

Width & Height
: Decides the width and height of the particle.

Proportional
: Locks Width & Height to be the same.

Texture
: Allows you to pick an image for the *Image shape*.

Shape Dynamics

Random Size
: Randomizes the particle size between 1x1 px and the given size of the particle in brush-tip or spray shape.

Fixed Rotation
: Gives a fixed rotation to the particle to work from.

Randomized Rotation
: Randomizes the rotation.

Follow Cursor Weight
: How much the pressure affects the rotation of the particles. At 1.0 and high pressure it’ll seem as if the particles are exploding from the middle.

Angle Weight
: How much the spray area angle affects the particle angle.

Color Options

Random HSV
: Randomize the HSV with the strength of the sliders. The higher, the more the color will deviate from the foreground color, with the direction indicating clock or counter clockwise.

Random Opacity
: Randomizes the opacity.

Color Per Particle
: Has the color options be per particle instead of area.

Sample Input Layer.
: Will use the underlying layer as reference for the colors instead of the

foreground color.

Fill Background
: Fills the area before drawing the particles with the background color.

Mix with background color.
: Gives the particle a random color between foreground/input/random HSV and the background color.

Tangent Normal Brush Engine

The Tangent Normal Brush Engine is an engine that is specifically designed for drawing normal maps, of the tangent variety. These are in turn used in 3d programs and game engines to do all sorts of lightning trickery. Common uses of normal maps include faking detail where there is none, and to drive transformations (Flow Maps).

A Normal map is an image that holds information for vectors. In particular, they hold information for Normal Vectors, which is the information for how the light bends on a surface. Because Normal Vectors are made up of 3 coordinates, just like colors, we can store and see this information as colors.

Normals can be seen similar to the stylus on your tablet. Therefore, we can use the tilt-sensors that are available to some tablets to generate the color of the normals, which can then be used by a 3d program to do lighting effects.

In short, you will be able to paint with surfaces instead of colors.

The following options are available to the tangent normal brush engine:

- Brush Tips
- Blending Modes
- Opacity and Flow
- Size
- Ratio
- Spacing
- Mirror
- Softness
- Sharpness
- Rotation
- Scatter
- Airbrush

- Texture

Specific Parameters to the Tangent Normal Brush Engine

Tangent Tilt

These are the options that determine how the normals are calculated from tablet input.

Tangent Encoding
: This allows you to set what each color channel means. Different programs set different coordinates to different channels, a common version is that the green channel might need to be inverted (-Y), or that the green channel is actually storing the x-value (+X).

Tilt Options
: Allows you to choose which sensor is used for the X and Y.

Tilt
: Uses Tilt for the X and Y.

Direction
: Uses the drawing angle for the X and Y and Tilt-elevation for the Z, this allows you to draw flowmaps easily.

Rotation
: Uses rotation for the X and Y, and tilt-elevation for the Z. Only available for specialized Pens.

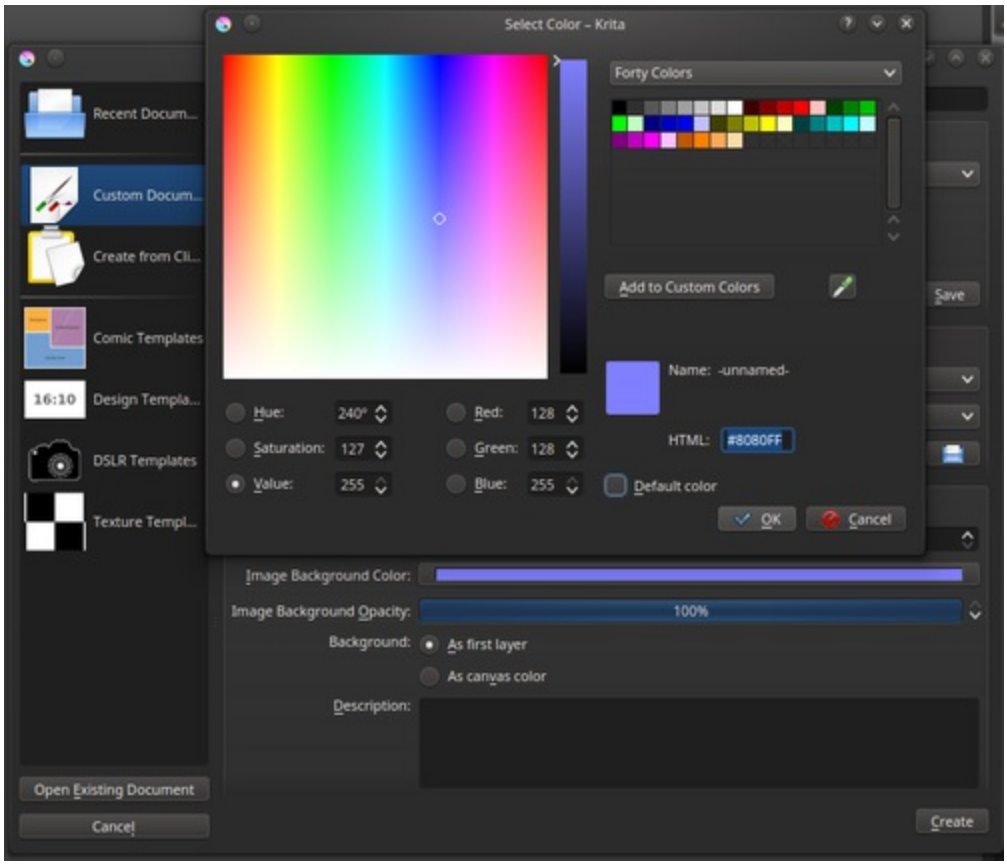
Elevation Sensitivity
: Allows you to change the range of the normal that are outputted. At 0 it will only paint the default normal, at 1 it will paint all the normals in a full hemisphere.

Usage

The Tangent Normal Map Brush Engine is best used with the Tilt Cursor, which can be set in *Settings* ▸ *Configure Krita* ▸ *General* ▸ *Outline Shape* ▸ *Tilt Outline*.

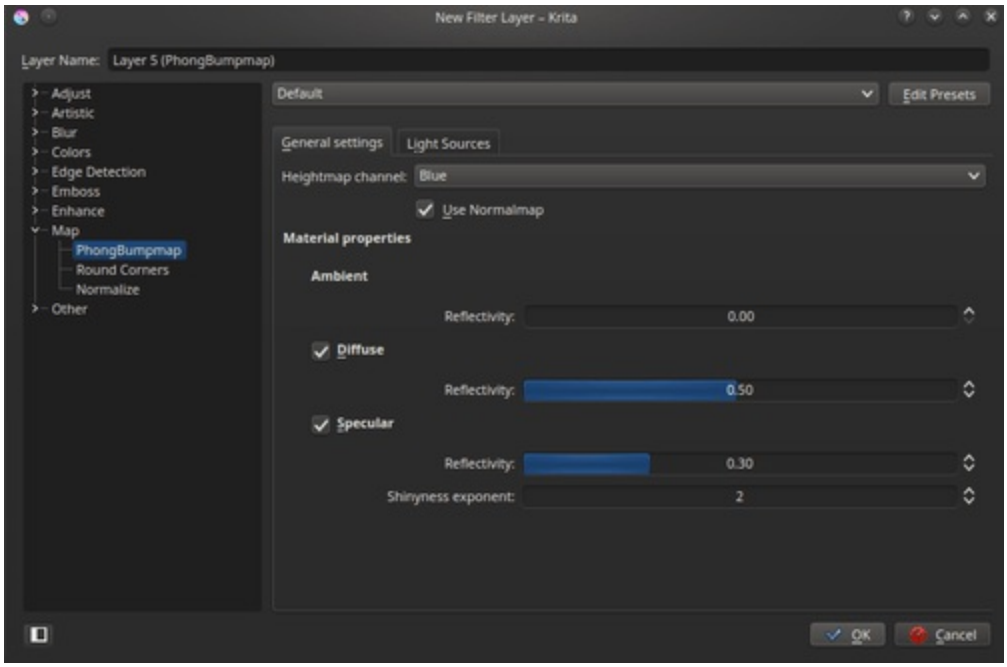
Normal Map authoring workflow

1. Create an image with a background color of (128, 128, 255) blue/purple.

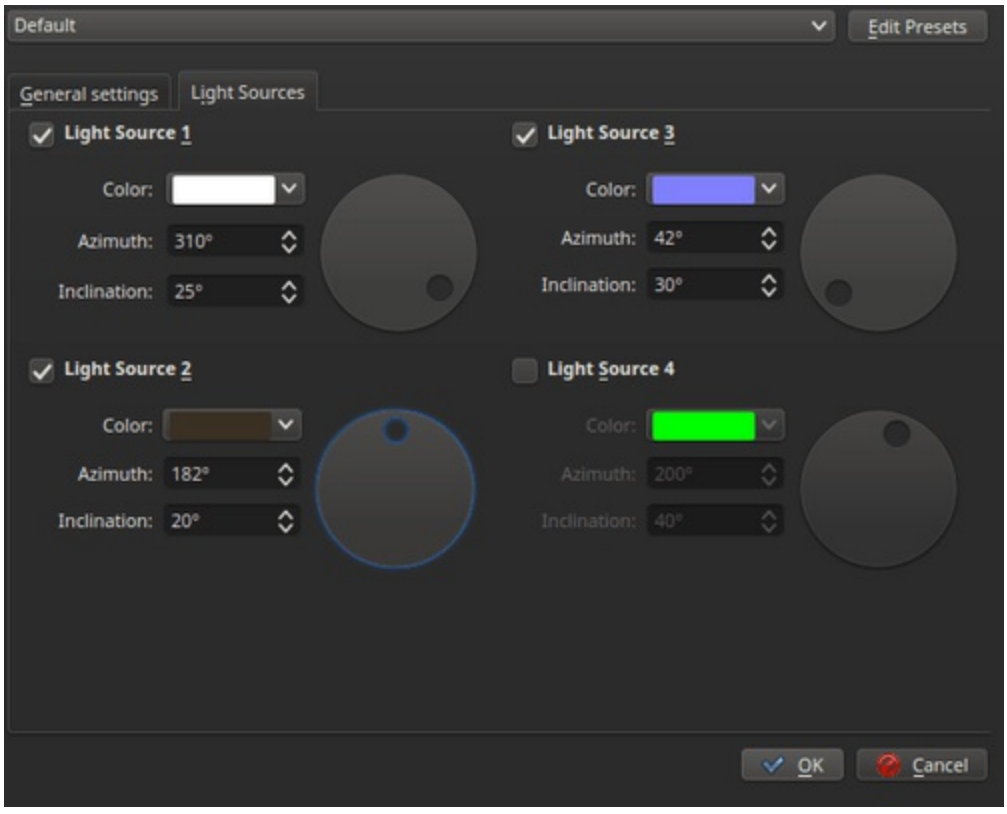


Setting up a background with the default color.

2. Set up group with a *Phong Bumpmap* filter mask. Use the *Use Normal map* checkbox on the filter to make it use normals.



Creating a phong bump map filter layer, make sure to check ‘Use Normal map’.

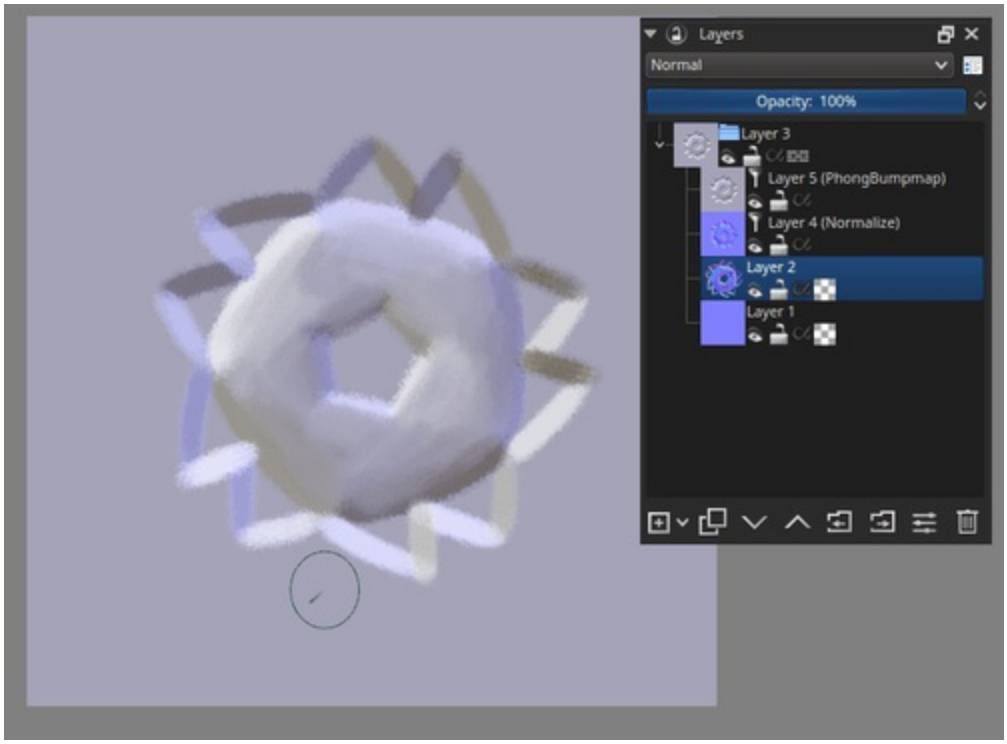


These settings give a nice daylight-esque lighting setup, with

light 1 being the sun, light 3 being the light from the sky, and light 2 being the light from the ground.

3. Make a *Normalize* filter layer or mask to normalize the normal map before feeding it into the Phong bumpmap filter for the best results.

4. Then, paint on layers in the group to get direct feedback.



Paint on the layer beneath the filters with the tangent normal brush to have them be converted in real time.

5. Finally, when done, hide the Phong bumpmap filter layer (but keep the Normalize filter layer!), and export the normal map for use in 3d programs.

Drawing Direction Maps

Direction maps are made with the *Direction* option in the *Tangent Tilt* options. These normal maps are used to distort textures in a 3d program (to simulate for example, the flow of water) or to create maps that indicate how hair and brushed metal is brushed. Krita can’t currently give feedback on how

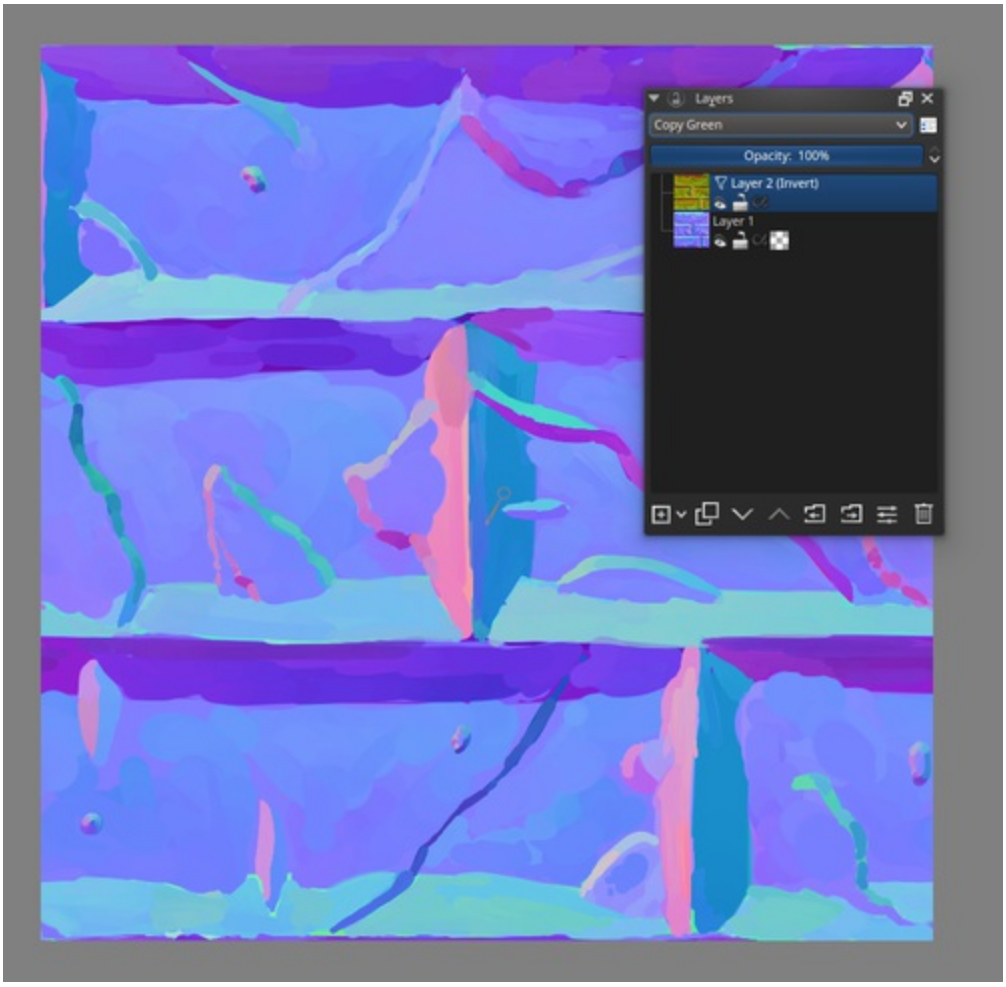
a given direction map will influence a distortion or shader, but these maps are a little easier to read.

Just set the *Tangent Tilt* option to *Direction*, and draw. The direction your brush draws in will be the direction that is encoded in the colors.

Only editing a single channel

Sometimes you only want to edit a single channel. In that case set the blending mode of the brush to *Copy <channel>*, with <channel> replaced with red, green or blue. These are under the *Misc* section of the blending modes.

So, if you want the brush to only affect the red channel, set the blending mode to *Copy Red*.



The copy red, green and blue blending modes also work on filter-layers.

This can also be done with filter layers. So if you quickly want to flip a layer's green channel, make an invert filter layer with *Copy Green* above it.

Mixing Normal Maps

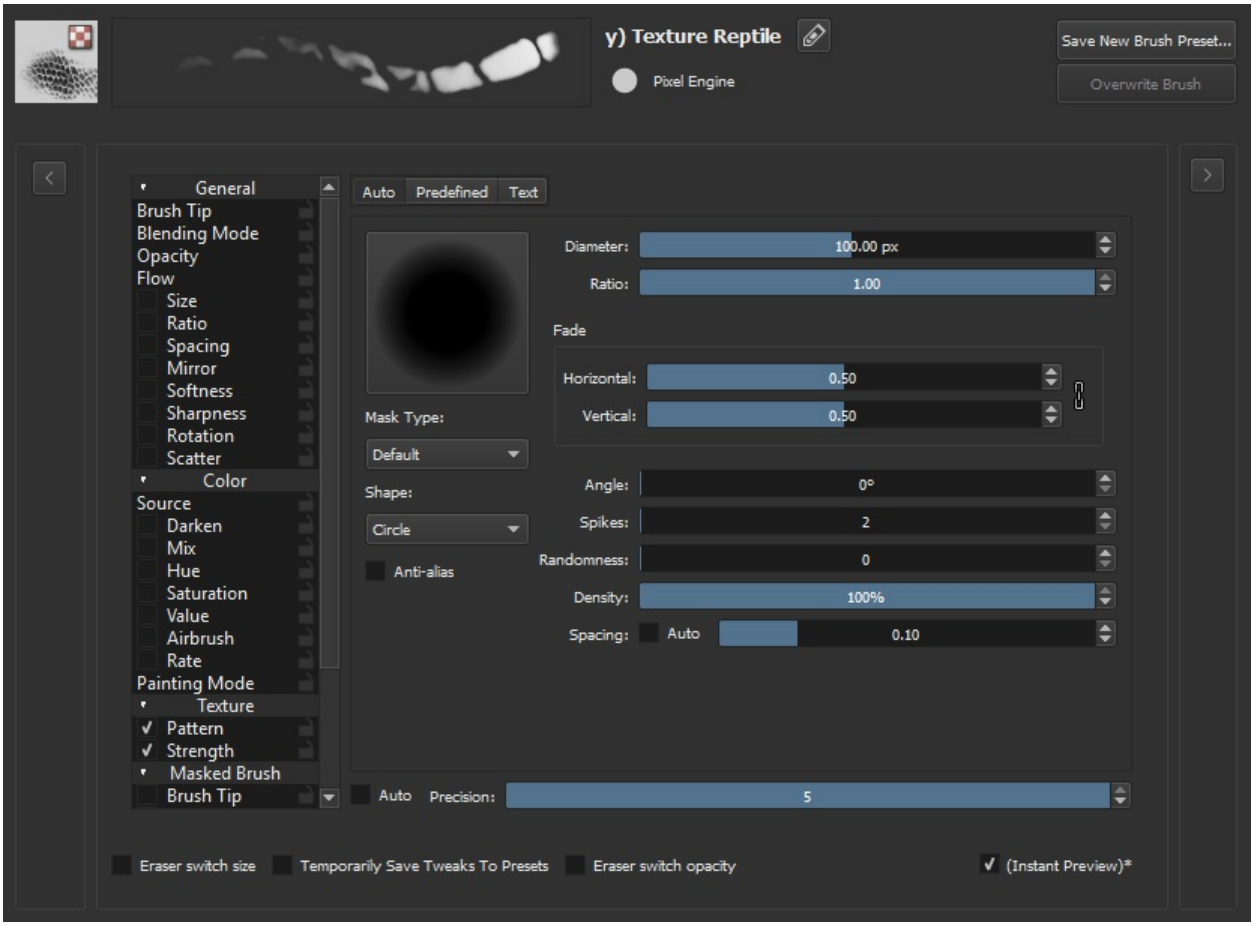
For mixing two normal maps, Krita has the *Combine Normal Map* blending mode under *Misc*.

Brush Settings

Overall Brush Settings for the various brush engines.

Contents:

Brush Tips



Auto Brush

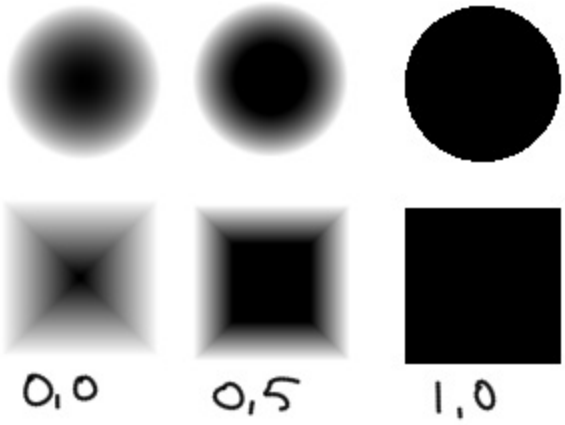
The generic circle or square. These brush tips are generated by Krita through certain parameters.

Types

First, there are three mask-types, with each the circle and square shape:

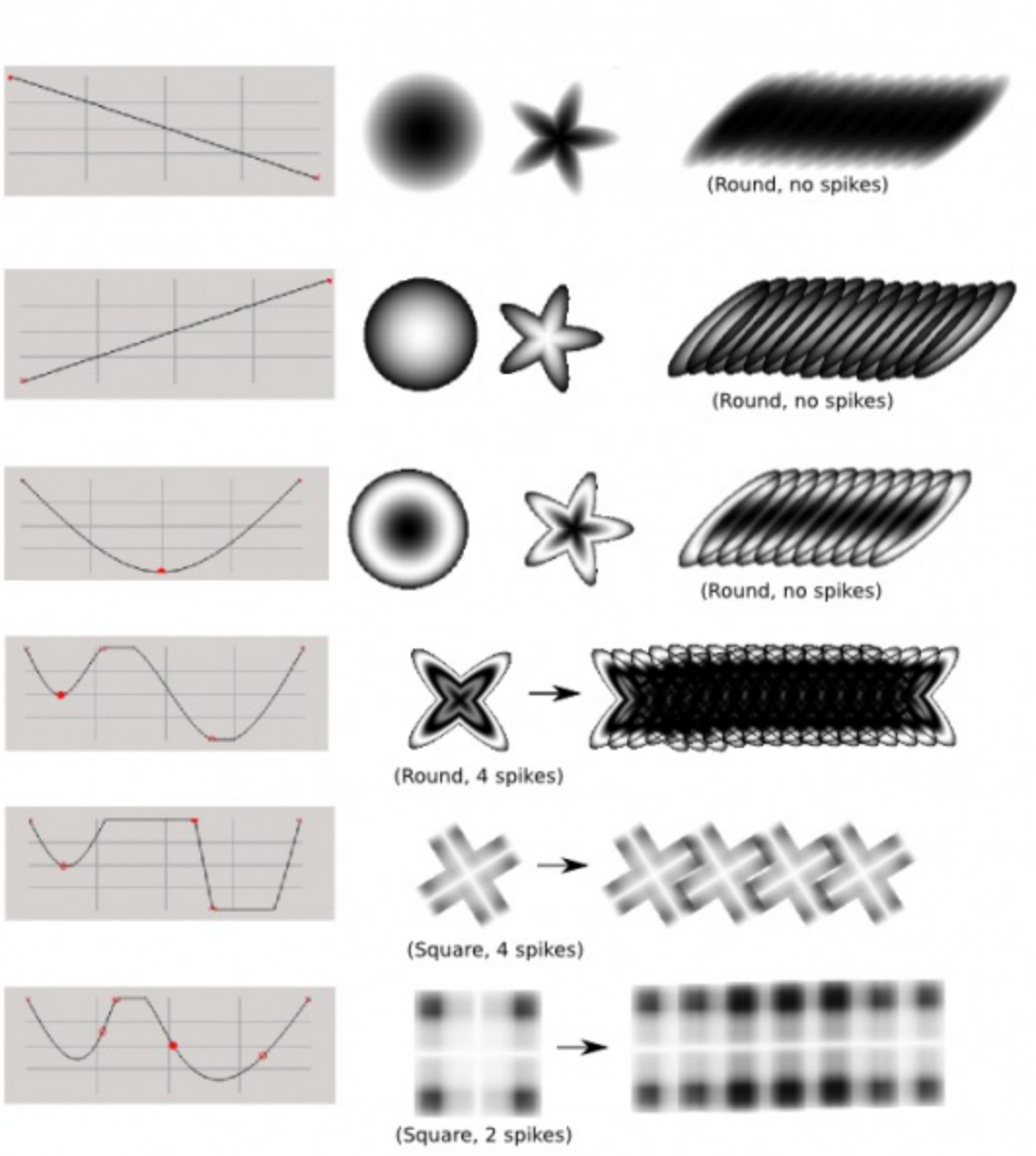
Default
: This is the ultimate generic type. The Fade parameter produces the below

results. Of the three auto brushes, this is the fastest.



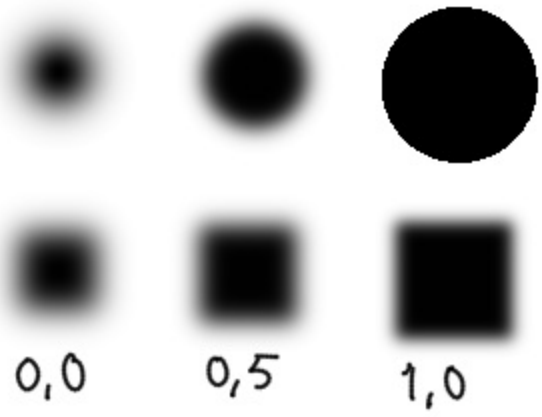
Soft

This one’s fade is controlled by a curve!



Gaussian

This one uses the gaussian algorithm to determine the fade. Out of the three auto brushes, this is the slowest.



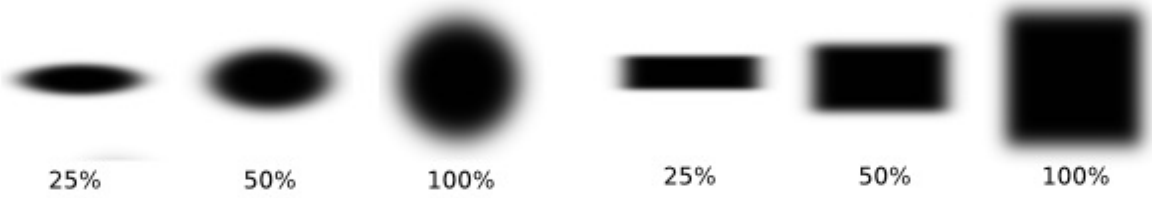
Parameters

Diameter

The pixel size of the brush.

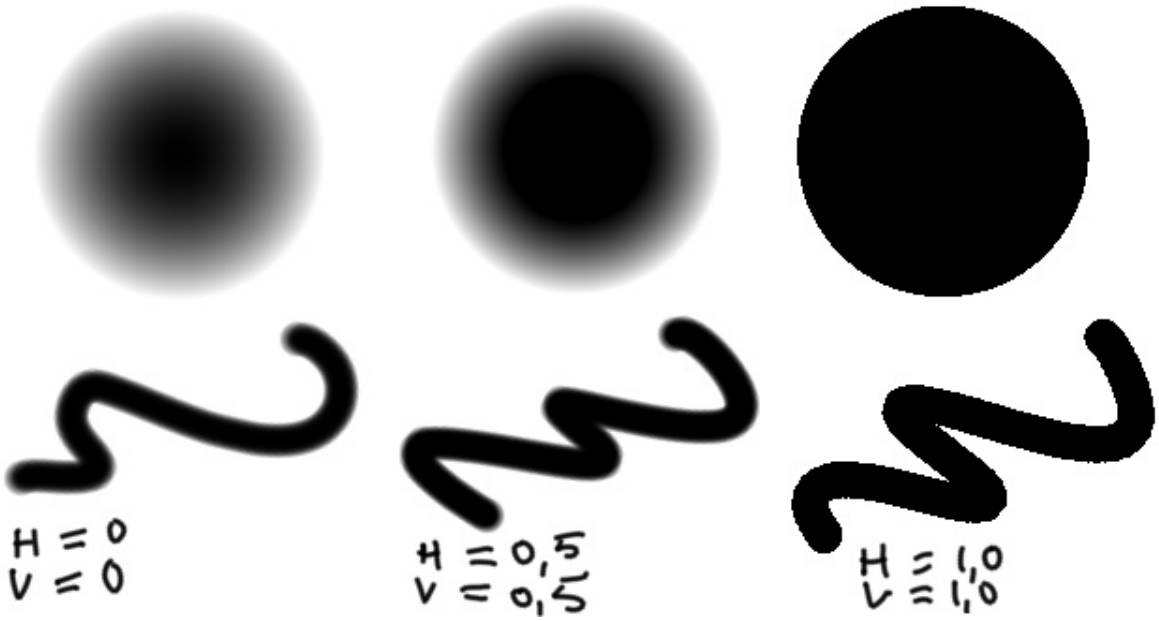
Ratio

Whether the brush is elongated or not.



Fade

this sets the softness of the brush. You can click the chain-symbol to lock and unlock these settings. Fade has a different effect per mask-type, so don’t be alarmed if it looks strange, perhaps you have the wrong mask-type.

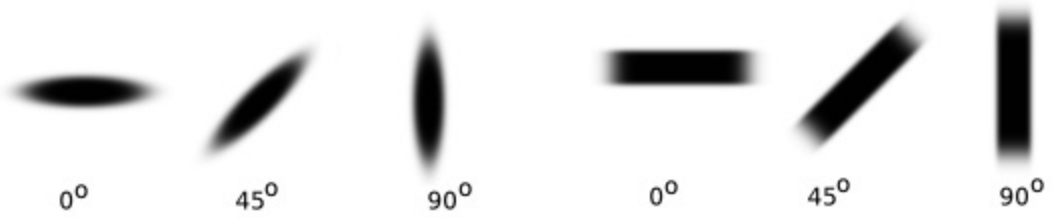


With fade locked.

With fade separately horizontal and vertical.

Angle

This changes the angle a which the brush is at.

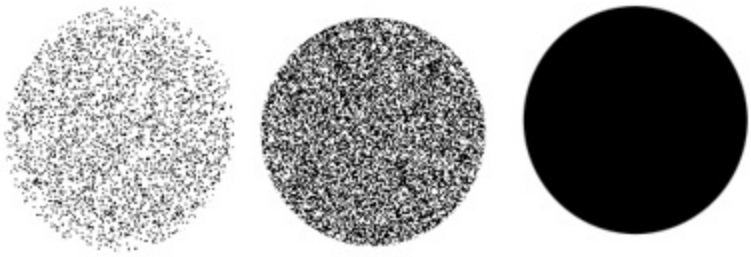


Spikes

This gives the amount of tips related to the ratio.

Density

This determines how much area the brush-covers over its size: It makes it noisy. In the example below, the brush is set with density 0%, 50% and 100% respectively.



Randomness

This changes the randomness of the density. In the example below, the brush is set with randomness 0%, 50% and 100% respectively.

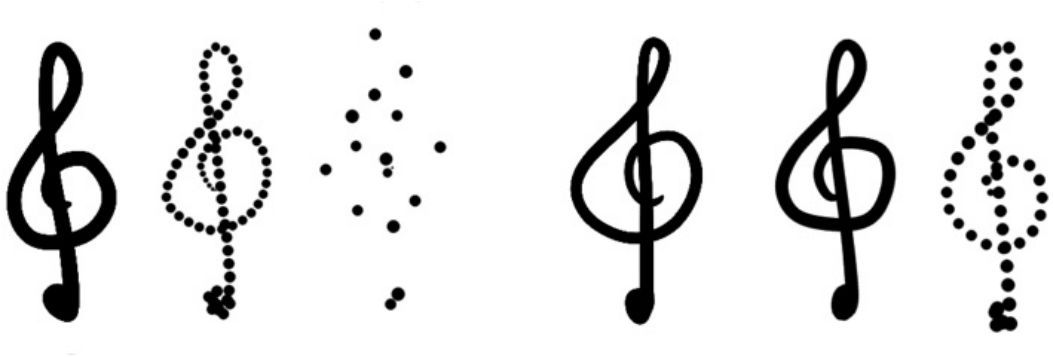
Spacing

This affects how far brushes are spaced apart. In the below picture, the three examples on the left are with spacing 0, 1 and 5.

Auto (spacing)

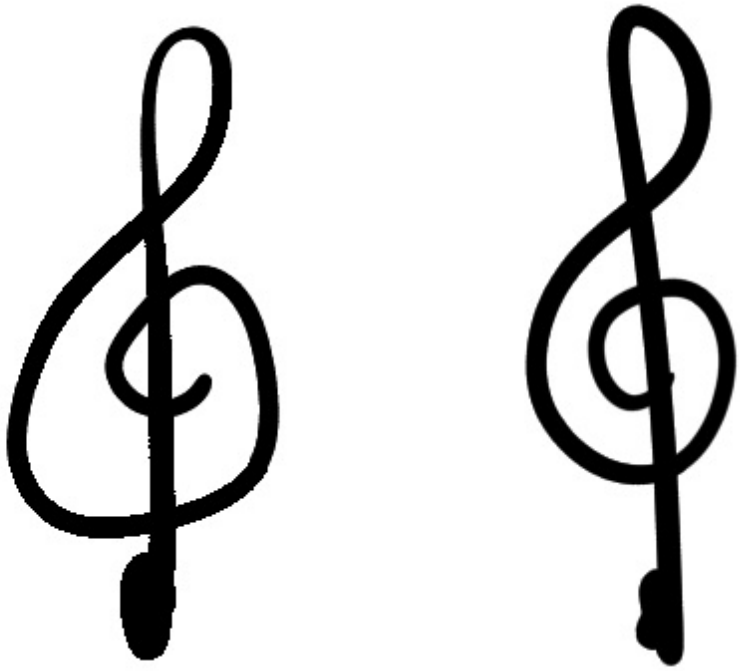
Ticking this will set the brush-spacing to a different (quadratic) algorithm. The result is fine control over the spacing. In the below

picture, the three examples on right are with auto spacing, 0, 1 and 5 respectively.



Smooth lines

This toggles the super-smooth anti-aliasing. In the below example, both strokes are drawn with a default brush with fade set to 0. On the left without smooth lines, and the right with. Very useful for inking brushes. This option is best used in combination with Auto Spacing.



Precision

This changes how smooth the brush is rendered. The lower, the faster the brush, but the worse the rendering looks. You'd want an inking brush to have a precision of 5 at all times, but a big filling brush for painting

doesn’t require such precision, and can be easily sped up by setting precision to 1.

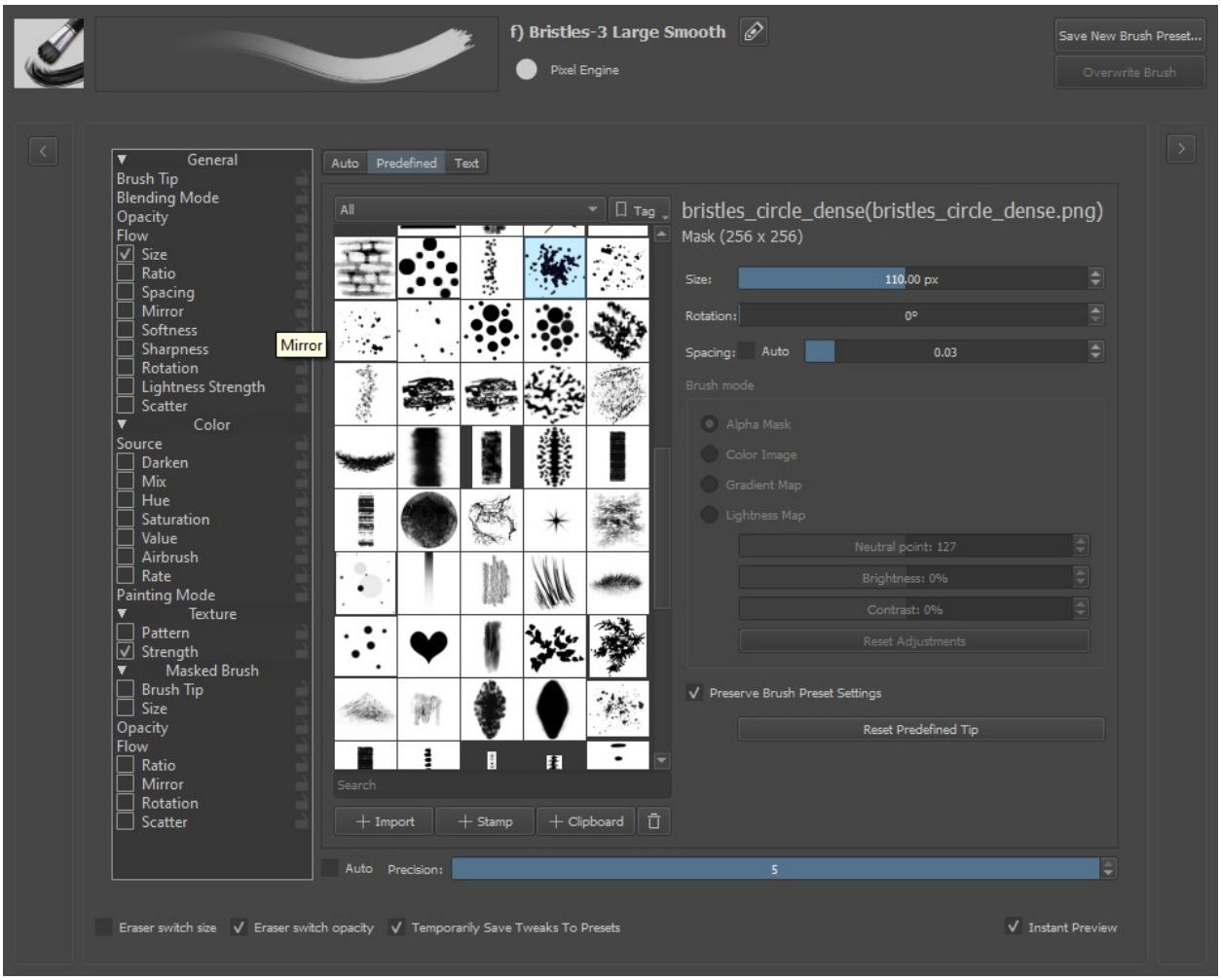
Auto (precision)

This allows you to set the precision linked to the size. The first value is the brush size at which precision is at last 5, and the second is the size-difference at which the precision will decrease.

For example: A brush with ‘’starting brush size’’ 10 and ‘’delta’’ 4, will have…

- precision 5 at size 10
- precision 4 at size 14
- precision 3 at size 18
- precision 2 at size 22
- precision 1 at sizes above 26.

Predefined Brushes



If you have used other applications like **GIMP** or **Photoshop**, you will have used this kind of brush. **Krita** is (mostly) compatible with the brush tip definitions files of these applications:

abr
: Gimp autobrush tip definitions.

*.gbr
: Gimp single bitmap brush tip. Can be black and white or colored.

*.gih
: Gimp Image Hose brush tip: contains a series of brush tips that are painted randomly or in order after each other. Can be black and white or colored. **Krita** does not yet support all the parameters yet.

abr
: Photoshop brush tip collections. We support many of the features of these brush files, though some advanced features are not supported yet.

Note that the definition of ABR brushes has been reverse engineered since Adobe does not make the specification public. We strongly recommend every **Krita** user to share brush tips in GBR and GIH format and more complex brushes as **Krita** presets.

All predefined brush tips are shown in one selector. There are four more options that influence the initial bitmap brush tip you start painting with:

Scale
: Scales the brush tip. 1.0 is the native size of the brush tip. This can be fairly large! When painting with variable size (for instance governed by pressure), this is the base for the calculations.

Rotation
: Initial rotation of the brush tip.

Spacing
: Distance between the brush tip impressions.

Brush Mode

Alpha Mask
: For colored brushes, don’t paint the actual colors, but make a grayscale brush tip that will be colored by your selected foreground/background color. Lighter areas will be interpreted as more transparent.

Color Image
: Use the brush tip image exactly as it is. Especially useful for image stamps.

Lightness Map
: *New in version 4.3:* Combines the features of Alpha Mask and Image Stamp modes. Transparency is preserved as it is in Image Stamp mode,

but colors or gray tones in the brush are replaced by the foreground color. The Lightness values of the brush tip image (if thinking in HSL mode) are preserved, so dark parts of the image are dark, and bright parts are bright. This allows image stamps where you can choose the color, but preserve highlights and shadows, and can even create an effect of thick paint in a brush stroke by simulating the highlights and shadows caused by the texture of the paint and brush stroke (sometimes called an "impasto" effect).

There are three sliders here, to control the exact feel of the current brush tip in Lightness or Gradient mode:

Neutral point
: This is the lightness level that will be the same as your current foreground color. Higher values than this will be lighter versions of the current foreground color, and lower, darker versions of the current color.

Brightness
: Makes the tip as a whole brighter or darker.

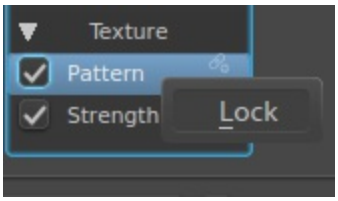
Contrast
: Increase the contrast between dark and light areas in the tip.

Gradient Map
: *New in version 4.4:* Use the lightness values of the brush tip image as a map to a gradient. Black maps to the left side of the gradient, and white to the right side of the gradient. The gradient used is the currently selected gradient in the main window, so you can change the gradient quickly and easily while painting. This mode allows image stamps with multiple colors that can be changed (great for flowers or other colorful vegetation), and can allow paint brushes that have multiple colors. Image adjustment sliders for Lightness Map mode can be used for this mode too.

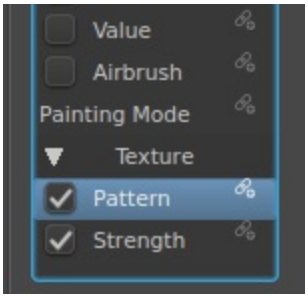
Locked Brush Settings

Normally, a changing to a different brush preset will change all brush settings. Locked presets are a way for you to prevent Krita from changing all settings. So, if you want to have the texture be that same over all brushes, you lock the texture parameter. That way, all brush-preset you select will now share the same texture!

Locking a brush parameter



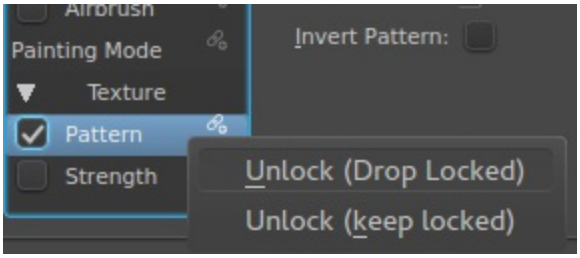
To lock an option,  the little lock icon next to the parameter name, and set it to *Lock*. It will now be highlighted to show it's locked:



And on the canvas, it will show that the texture-option is locked.

Unlocking a brush parameter

To *unlock*,  the icon again.



There will be two options:

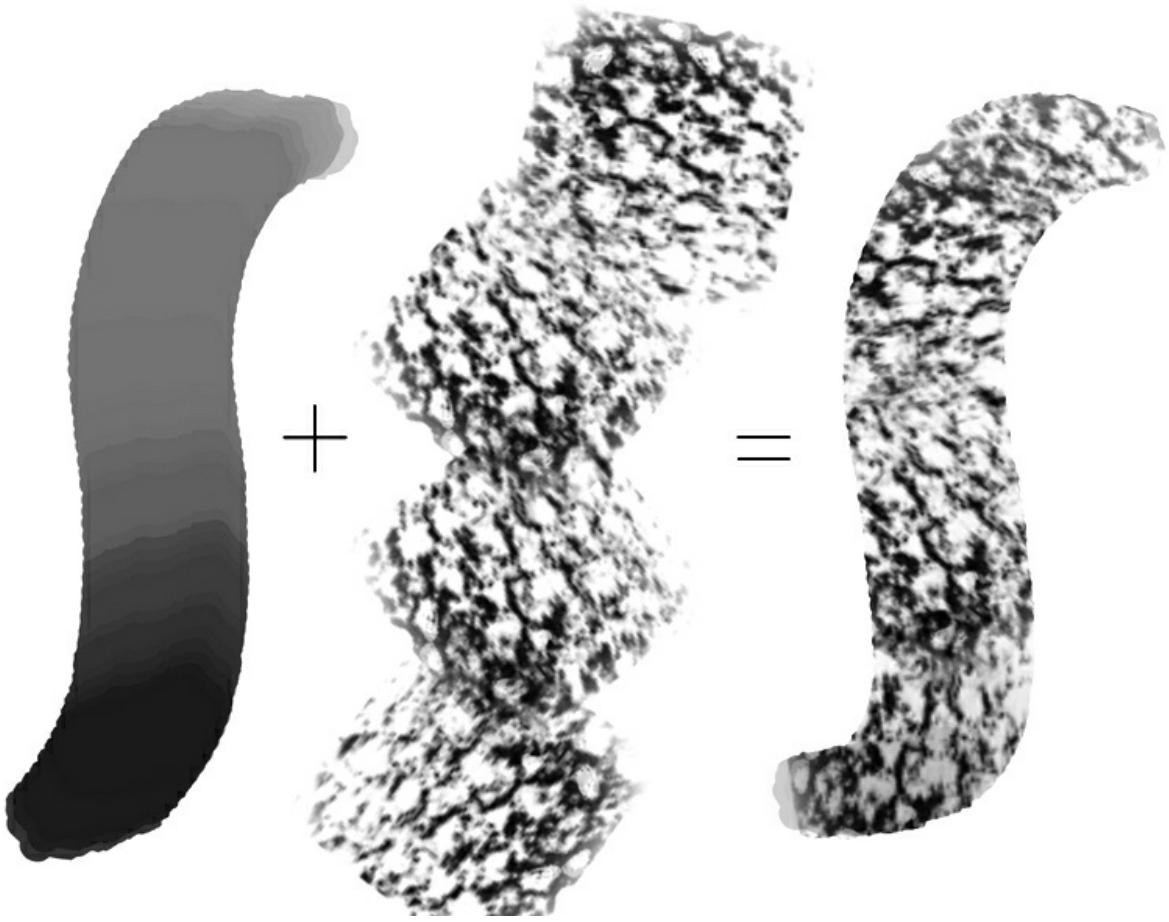
Unlock (Drop Locked)
: This will get rid of the settings of the locked parameter and take that of the active brush preset. So if your brush had no texture on, using this option will revert it to having no texture.

Unlock (Keep Locked)
: This will keep the settings of the parameter even though it’s unlocked.

Masked Brush

New in version 4.0.

Masked brush is new feature that is only available in the Pixel Brush Engine. They are additional settings you will see in the brush editor. Masked brushes allow you to combine two brush tips in one stroke. One brush tip will be a mask for your primary brush tip. A masked brush is a good alternative to texture for creating expressive and textured brushes.



Note

Due to technological constraints, the masked brush only works in the wash painting mode. However, do remember that flow works as opacity does in

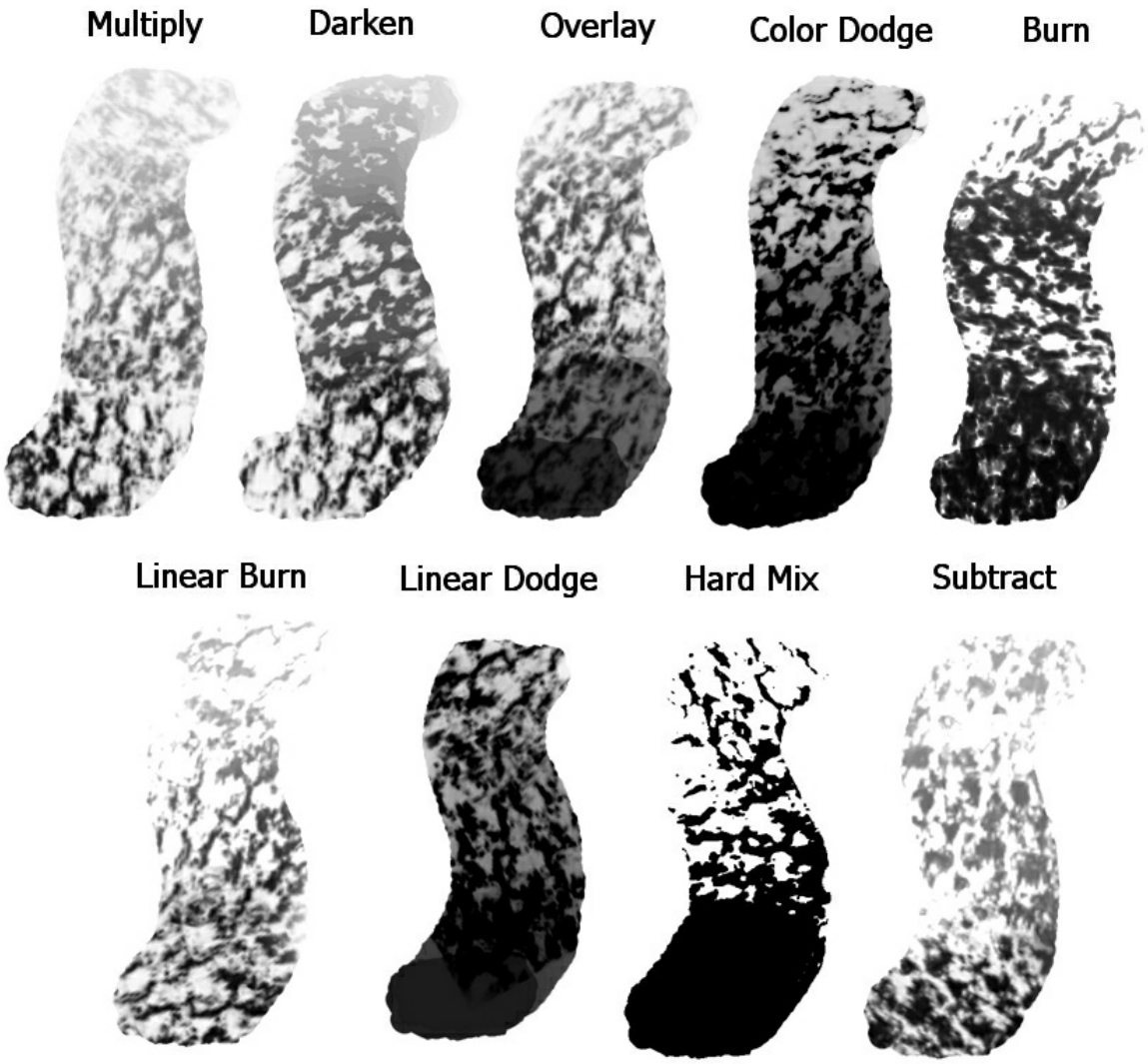
the build-up painting mode.

Brush Tips

Like with normal brush tip you can choose any brush tip and change it size, spacing, and rotation. Masking brush size is relative to main brush size. This means when you change your brush size masking tip will be changed to keep the ratio.

Blending mode (drop-down inside Brush tip):

Blending modes changes how tips are combined.



Size

The size sensor option of the second tip.

Opacity and Flow

The opacity and flow of the second tip. This is mapped to a sensor by default. Flow can be quite aggressive on subtract mode, so it might be an idea to turn it off there.

Ratio

This affects the brush ratio on a given brush.

Mirror

The Mirror option of the second tip.

Rotation

The rotation option of the second tip. Best set to “fuzzy dab”.

Scatter

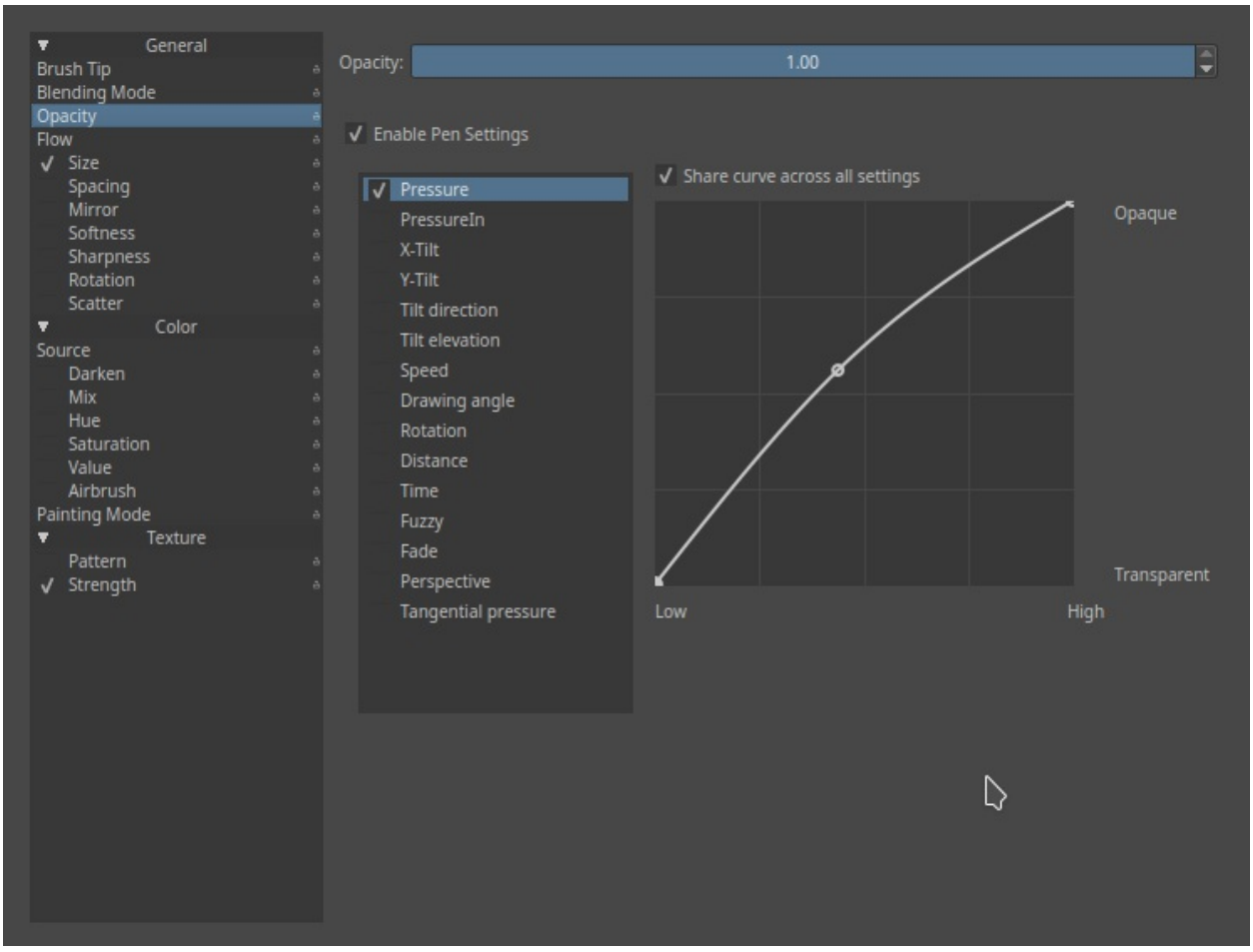
The scatter option. The default is quite high, so don’t forget to turn it lower.

Difference from Texture:

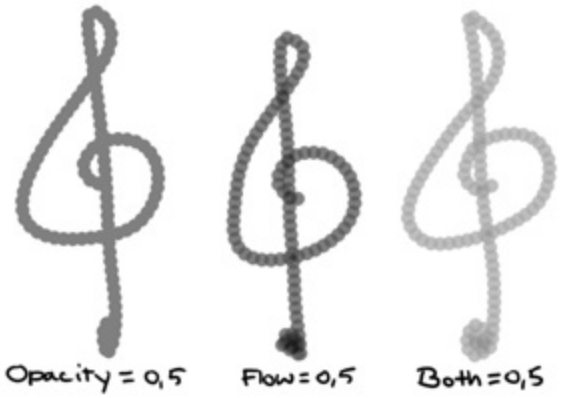
- You don’t need seamless texture to make cool looking brush.
- Stroke generates on the fly, it always different.
- Brush strokes looks same on any brush size.
- Easier to fill some areas with solid color but harder to make it hard textured.

Opacity and Flow

Opacity and flow are parameters for the transparency of a brush.

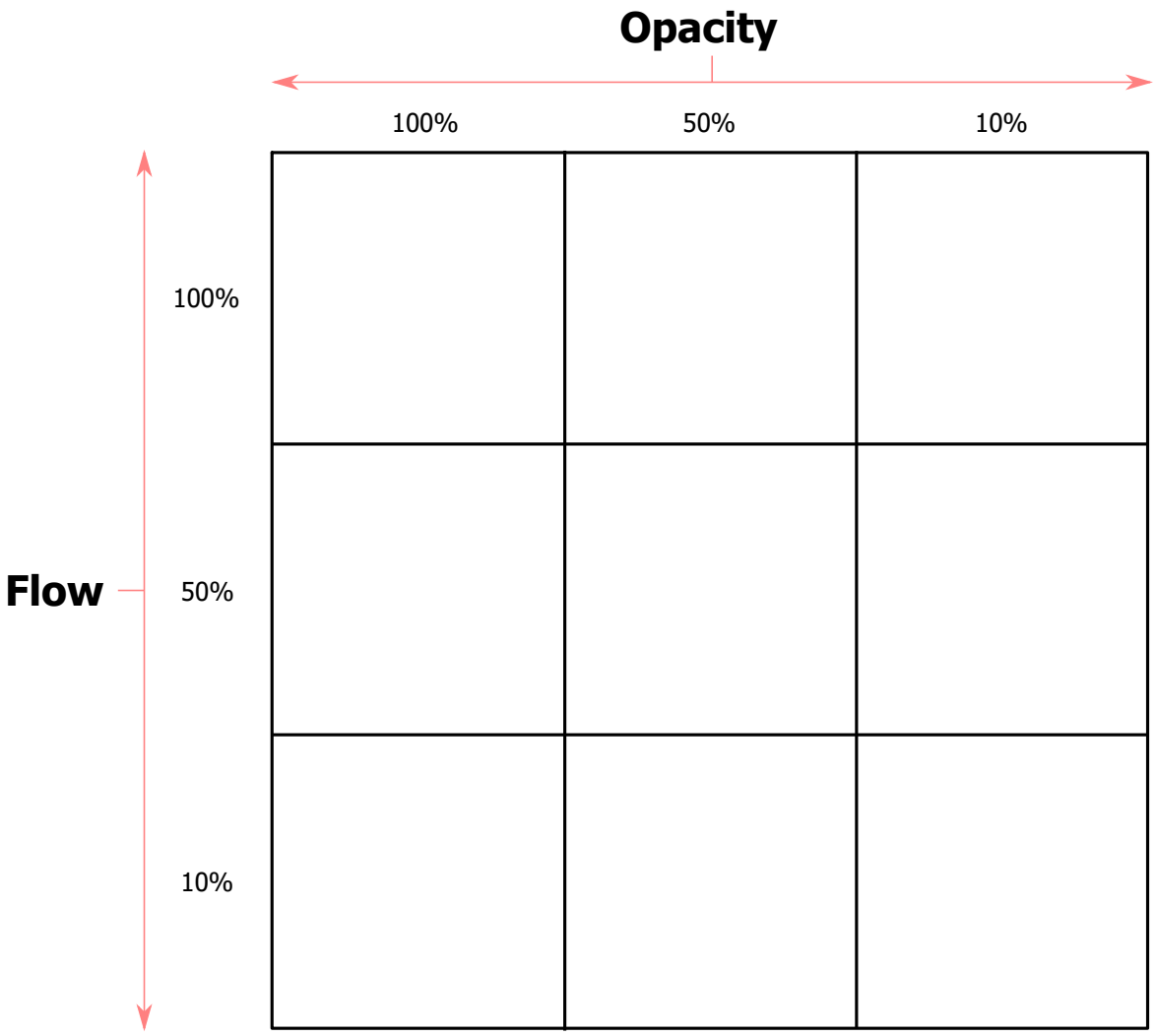


They are interlinked with the painting mode setting.



Opacity
: The transparency of a stroke.

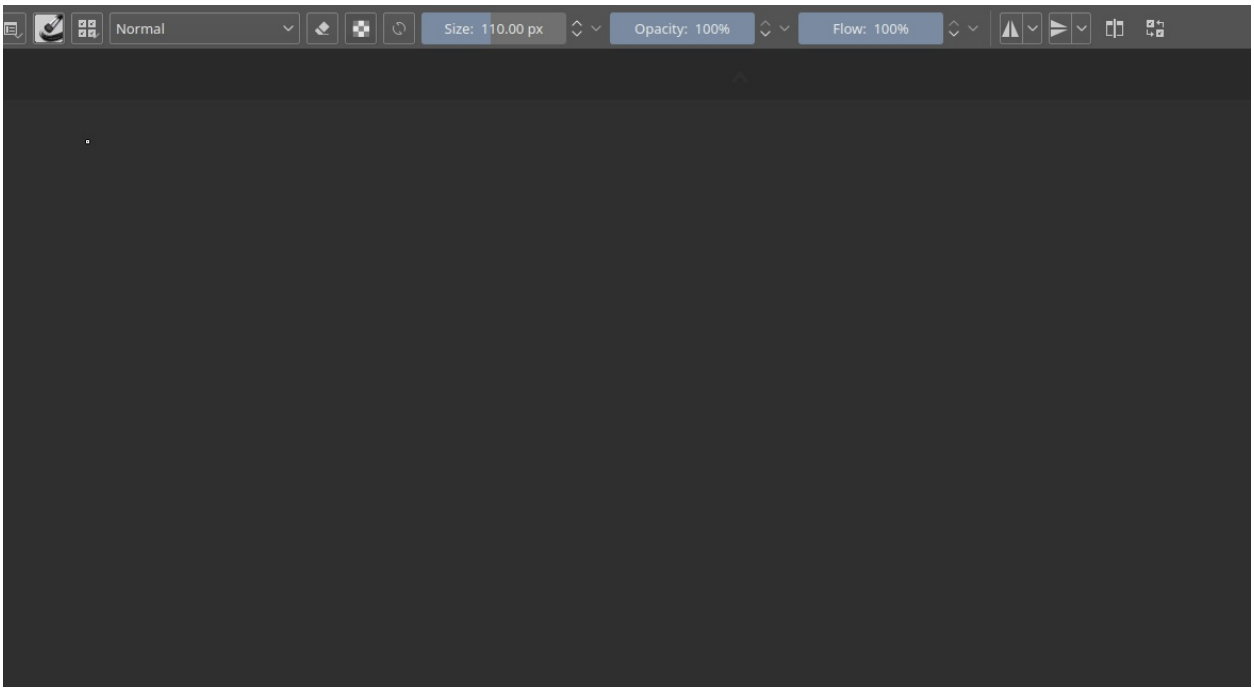
Flow
: The transparency of separate dabs. Finally separated from Opacity in 2.9.



Changed in version 4.2: In Krita 4.1 and below, the flow and opacity when combined with brush sensors would add up to one another, being only limited by the maximum opacity. This was unexpected compared to all other painting applications, so in 4.2 this finally got corrected to the flow and opacity multiplying, resulting in much more subtle strokes. This change can be switched back in the Tools Settings, but we will be deprecating the old way in future versions.

The old behavior can be simulated in the new system by…

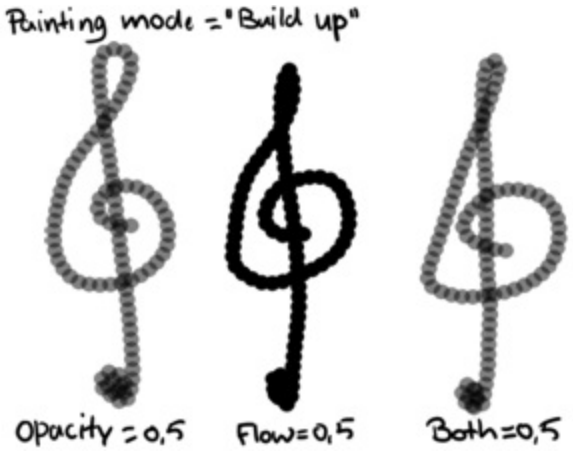
1. Deactivating the sensors on opacity.
2. Set the maximum value on flow to 0.5.
3. Adjusting the pressure curve to be concave.



Painting mode

Build-up
: Will treat opacity as if it were the same as flow.

Wash
: Will treat opacity as stroke transparency instead of dab-transparency.



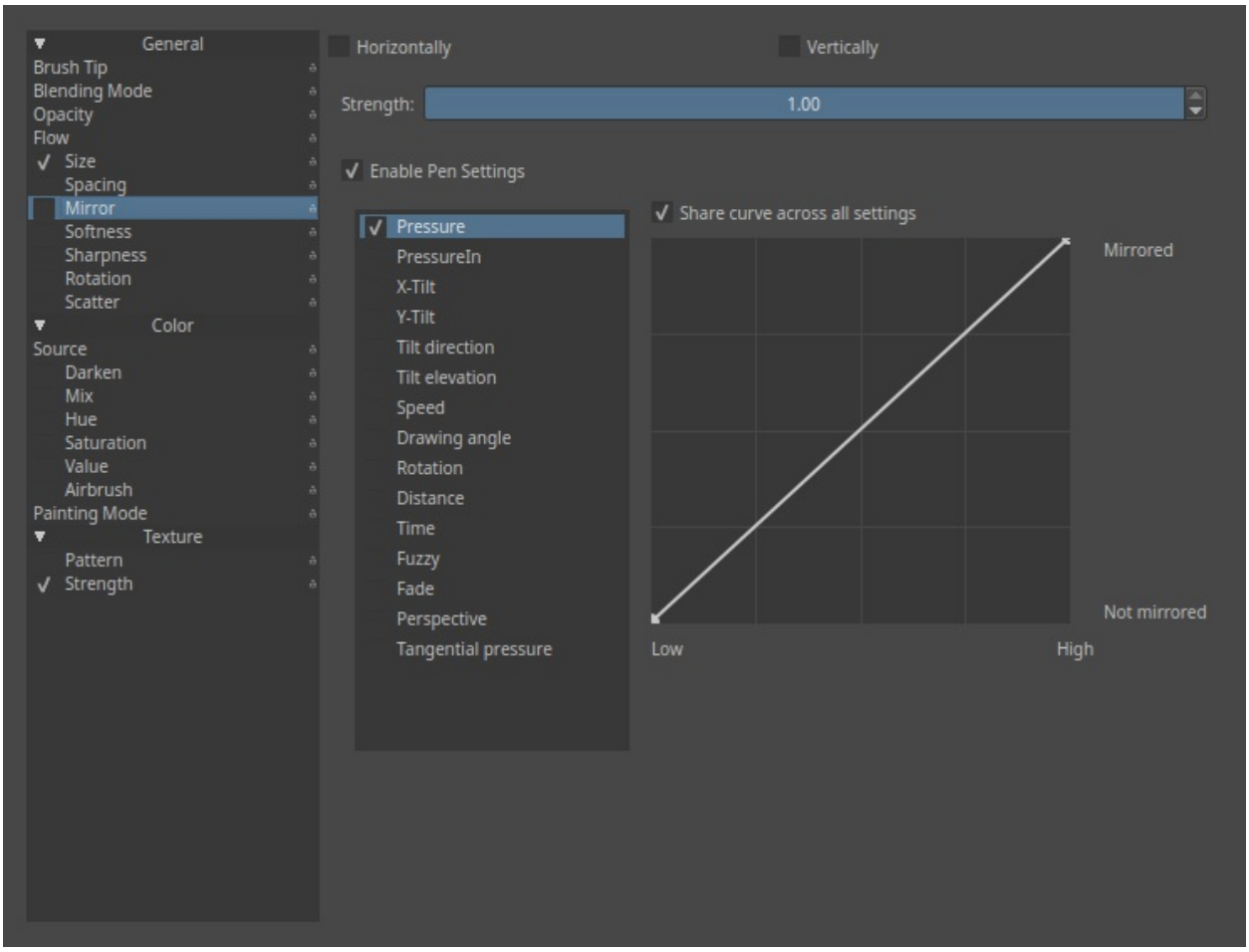
Where the other images of this page had all three strokes set to painting mode: wash, this one is set to build-up.

Options

Airbrush

If you hold the brush still, but are still pressing down, this will keep adding color onto the canvas. The lower the rate, the quicker the color gets added.

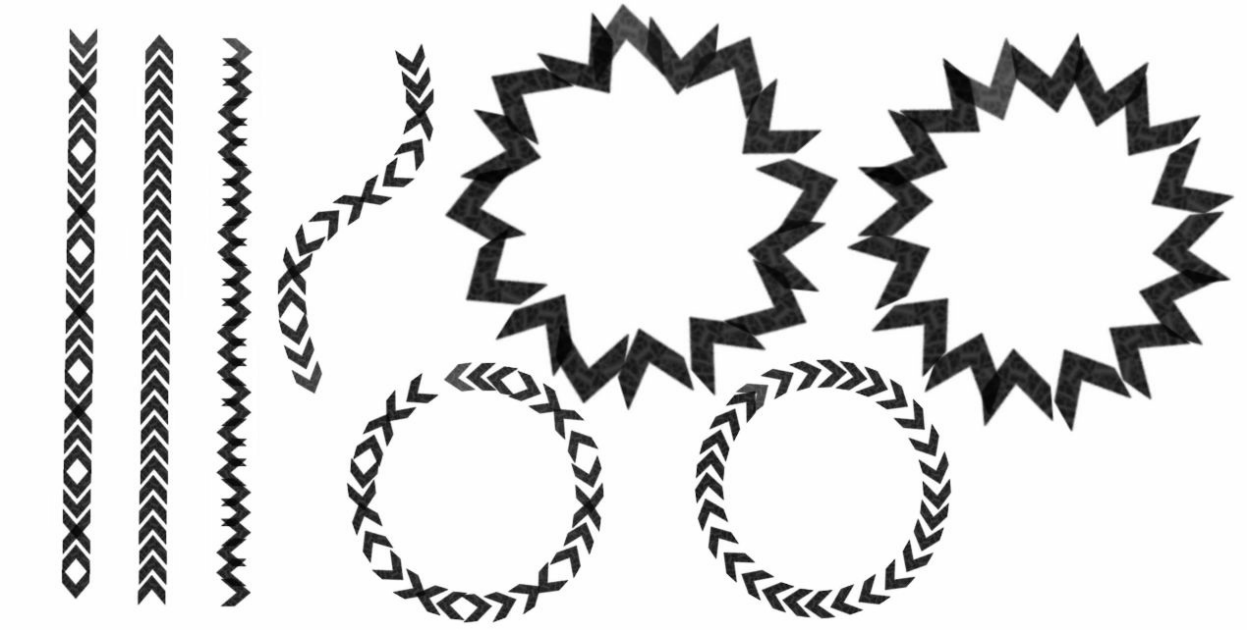
Mirror



This allows you to mirror the Brush tip with Sensors.

Horizontal
: Mirrors the mask horizontally.

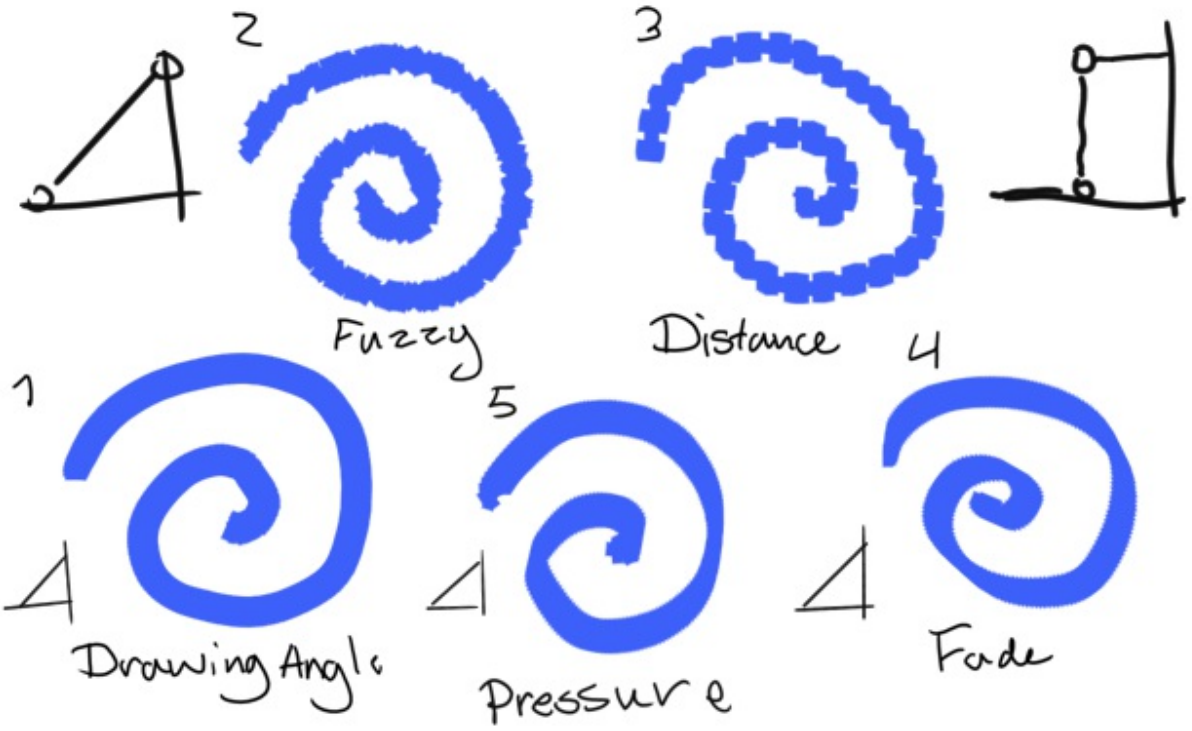
Vertical
: Mirrors the mask vertically.

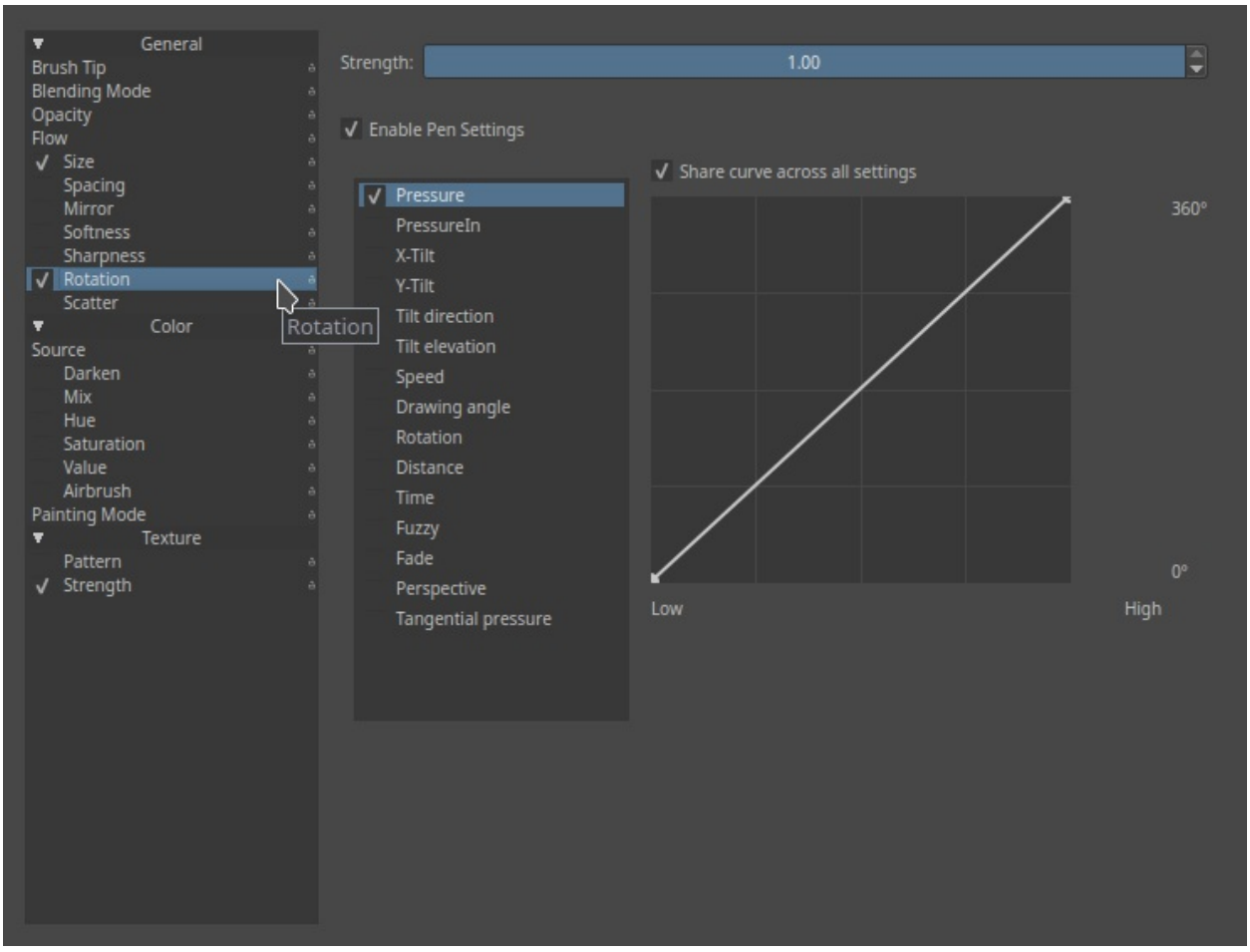


Some examples of mirroring and using it in combination with Rotation.

Rotation

This allows you to affect Angle of your brush tip with Sensors.


2
Fuzzy
3
Distance
1
Drawing Angle
5
Pressure
4
Fade

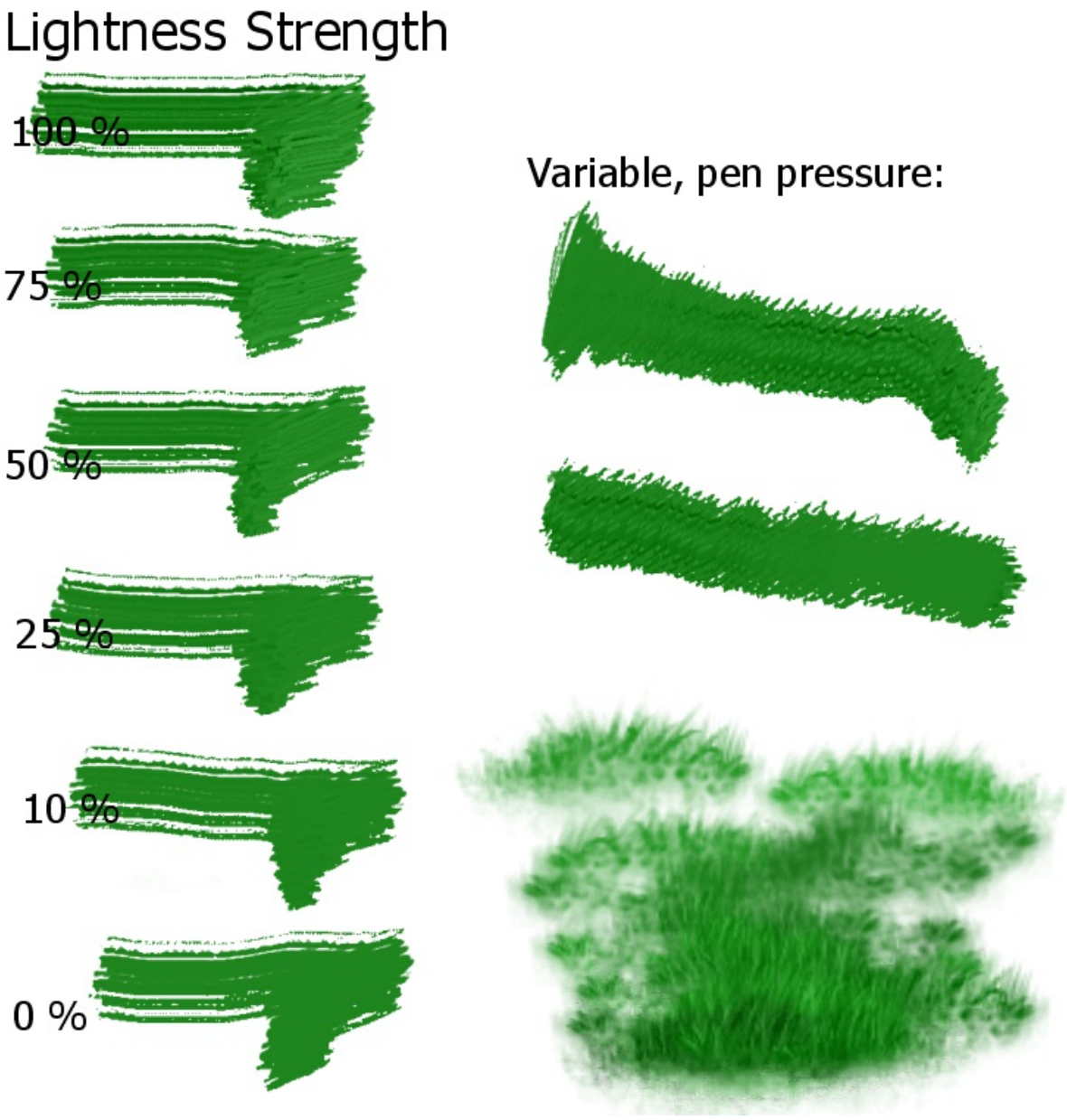

General
Brush Tip
Blending Mode
Opacity
Flow
Size
Spacing
Mirror
Softness
Sharpness
Rotation
Scatter
Color
Source
Darken
Mix
Hue
Saturation
Value
Airbrush
Painting Mode
Texture
Pattern
Strength
Rotation
Strength:
1.00
Enable Pen Settings
Pressure
PressureIn
X-Tilt
Y-Tilt
Tilt direction
Tilt elevation
Speed
Drawing angle
Rotation
Distance
Time
Fuzzy
Fade
Perspective
Tangential pressure
Share curve across all settings
360°
0°
Low
High

In the above example, several applications of the parameter.

1. Drawing Angle – A common one, usually used in combination with rake-type brushes. Especially effect because it does not rely on tablet-specific sensors. Sometimes, Tilt-Direction or Rotation is used to achieve a similar-more tablet focused effect, where with Tilt the 0° is at 12 o'clock, Drawing angle uses 3 o'clock as 0°.
2. Fuzzy – Also very common, this gives a nice bit of randomness for texture.
3. Distance – With careful editing of the Sensor curve, you can create nice patterns.
4. Fade – This slowly fades the rotation from one into another.
5. Pressure – An interesting one that can create an alternative looking line.

Lightness Strength

New in version 4.4: This allows you to affect the Lightness Strength of your brush tip with Sensors. Only available with brush tips in Lightness Map mode.



This changes the contrast of the brush tip, so that at 100%, the full effect of the lightness variation is visible in the brush, while at 0% the brush paints without any lightness variation. This allows a variable impasto effect with lightness brushes, and for variation in texture stamp brushes that use a lightness-enabled brush tip.

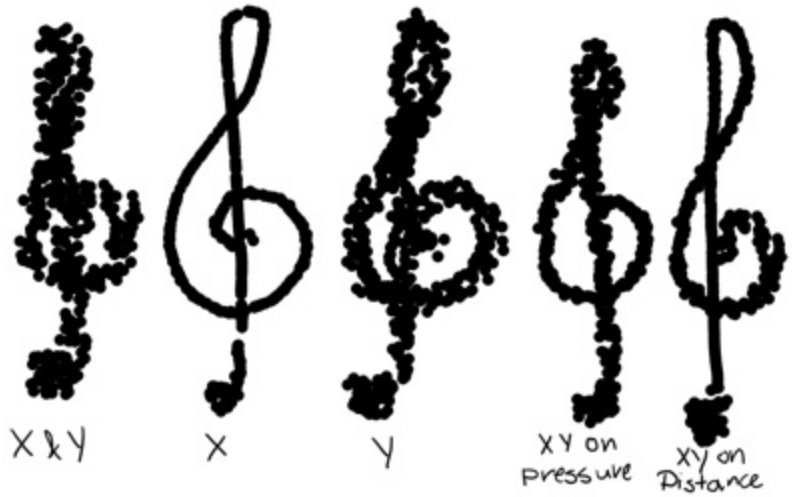
Scatter

This parameter allows you to set the random placing of a brush-dab. You can

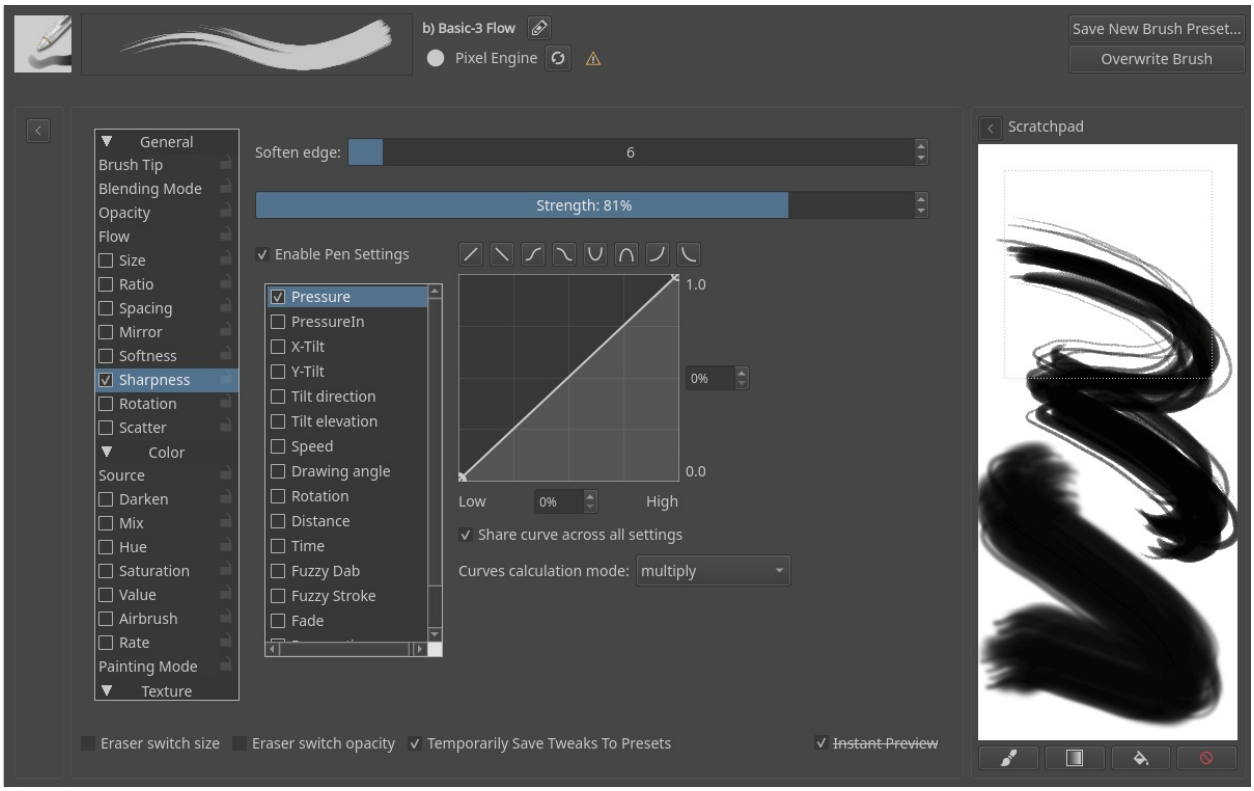
affect them with Sensors.

X
: The scattering on the angle you are drawing from.

Y
: The scattering, perpendicular to the drawing angle (has the most effect).



Sharpness



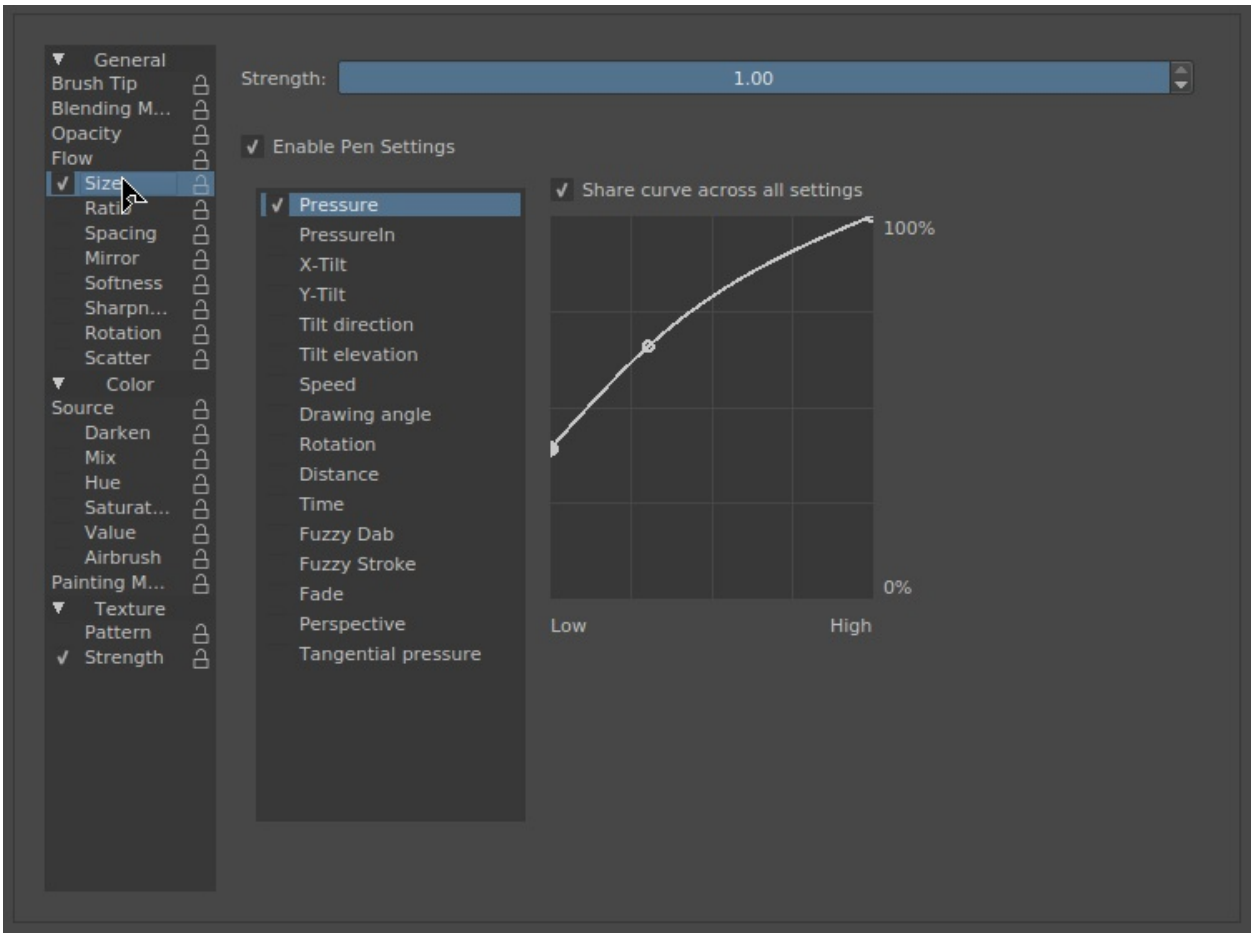
Puts a threshold filter over the brush mask. This can be used for brush like strokes, but it also makes for good pixel art brushes.

Strength
: Controls the threshold, and can be controlled by the sensors below.

Softness
: Controls the extra non-fully opaque pixels. This adds a little softness to the stroke.

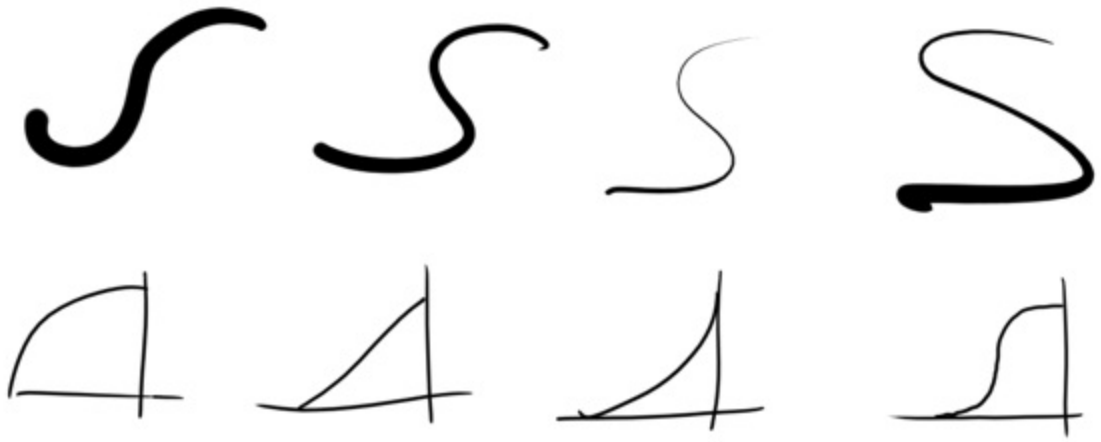
Changed in version 4.2: The sensors now control the threshold instead of the subpixel precision, softness slider was added.

Size

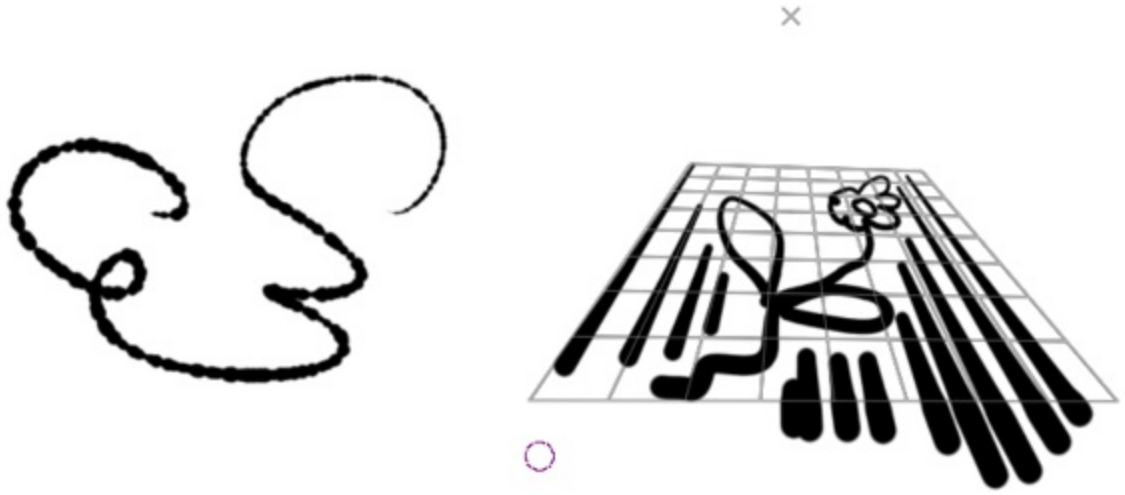


This parameter is not the diameter itself, but rather the curve for how it's affected.

So, if you want to lock the diameter of the brush, lock the Brush tip. Locking the size parameter will only lock this curve. Allowing this curve to be affected by the Sensors can be very useful to get the right kind of brush. For example, if you have trouble drawing fine lines, try to use a concave curve set to pressure. That way you'll have to press hard for thick lines.



Also popular is setting the size to the sensor fuzzy or perspective, with the later in combination with a Perspective.



Softness

This allows you to affect Fade with Sensors.

Has a slight brush-decreasing effect, especially noticeable with soft-brush, and is overall more noticeable on large brushes.

Source

Picks the source-color for the brush-dab.

Plain Color
: Current foreground color.

Gradient
: Picks active gradient.

Uniform Random
: Gives a random color to each brush dab.

Total Random
: Random noise pattern is now painted.

Pattern
: Uses active pattern, but alignment is different per stroke.

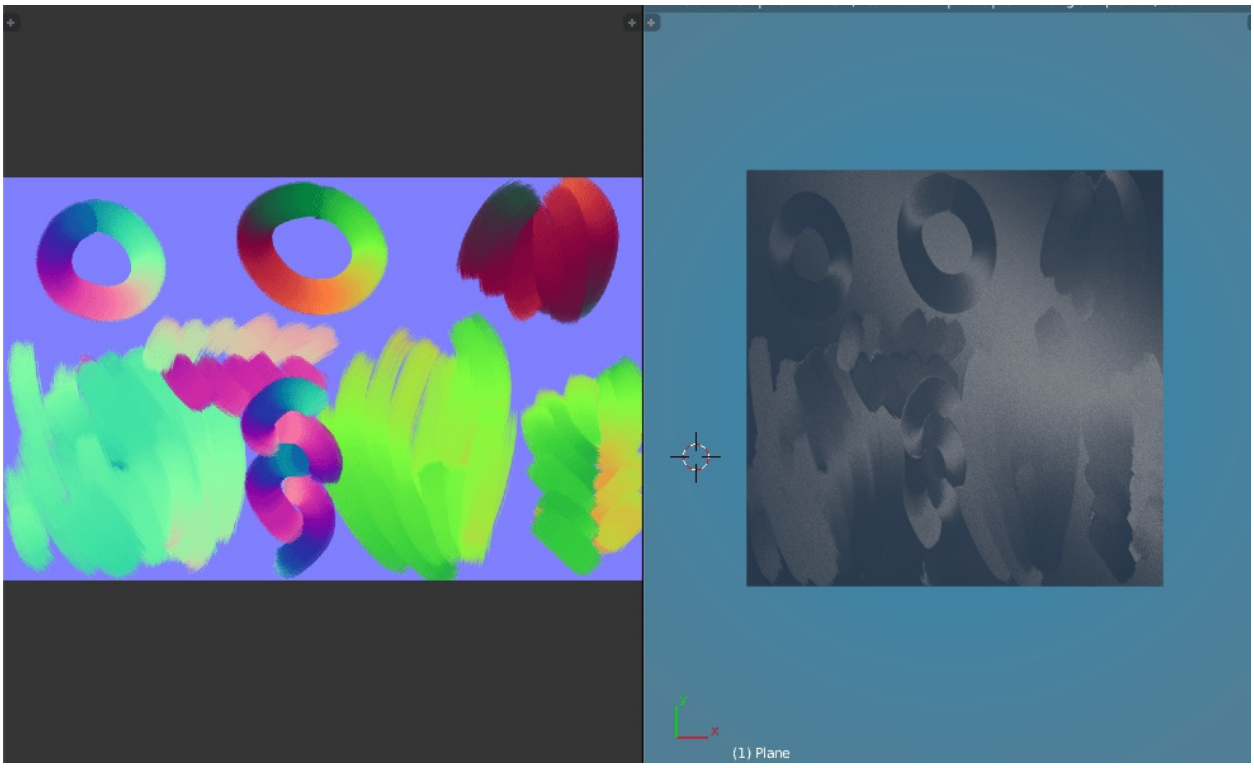
Locked Pattern

Locks the pattern to the brushdab.

Mix

Allows you to affect the mix of the Source color with Sensors. It will work with Plain Color and Gradient as source. If Plain Color is selected as source, it will mix between foreground and background colors selected in color picker. If Gradient is selected, it chooses a point on the gradient to use as painting color according to the sensors selected.

Uses



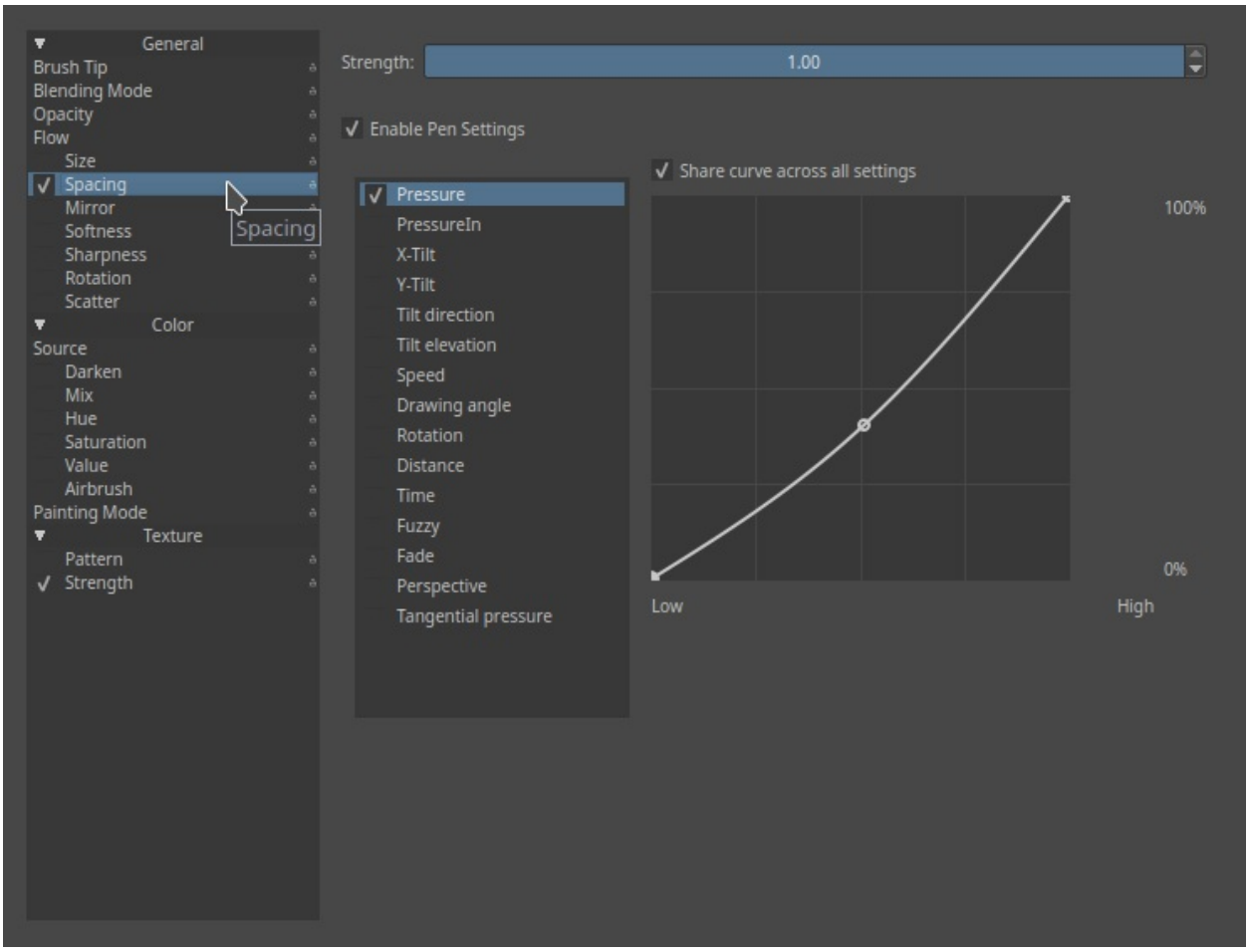
Flow map

The above example uses a **Krita** painted flowmap in the 3D program **Blender**. A brush was set to *Source* ▸ *Gradient* and *Mix* ▸ *Drawing angle*. The gradient in question contained the 360° for normal map colors. Flow maps are used in several Shaders, such as brushed metal, hair and certain river-shaders.

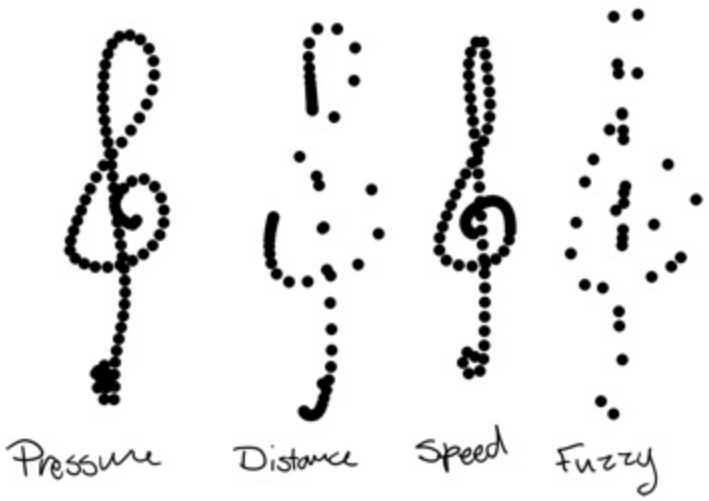
Gradient

Exactly the same as using *Source* ▸ *Gradient* with *Mix*, but only available for the Color Smudge Brush.

Spacing

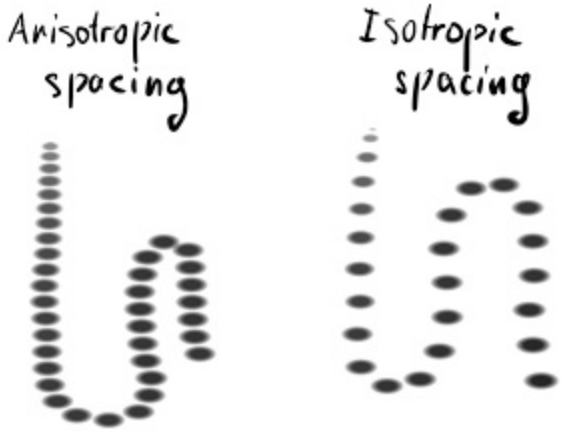


This allows you to affect Brush Tips with Sensors.



Isotropic spacing

Instead of the spacing being related to the ratio of the brush, it will be on diameter only.



Ratio

Allows you to change the ratio of the brush and bind it to parameters. This also works for predefined brushes.

Sensors

Pressure
: Uses the pressure in and out values of your stylus.

PressureIn
: Uses only pressure in values of your stylus. Previous pressure level in same stroke is overwritten *only* by applying more pressure. Lessening the pressure doesn’t affect PressureIn.

X-tilt
: How much the brush is affected by stylus angle, if supported.

Y-tilt
: How much the brush is affected by stylus angle, if supported.

Tilt-direction
: How much the brush is affected by stylus direction. The pen point pointing towards the user is 0°, and can vary from -180° to +180°.

Tilt-elevation
: How much the brush is affected by stylus perpendicularity. 0° is the stylus horizontal, 90° is the stylus vertical.

Speed
: How much the brush is affected by the speed at which you draw.

Drawing Angle
: How much the brush is affected by which direction you are drawing in. *Lock* will lock the angle to the one you started the stroke with. *Fan corners* will try to smoothly round the corners, with the angle being the angles threshold it’ll round. *Angle offset* will add an extra offset to the current angle.

Rotation
: How much a brush is affected by how the stylus is rotated, if supported

by the tablet.

Distance
: How much the brush is affected over length in pixels.

Time
: How much a brush is affected over drawing time in seconds.

Fuzzy (Dab)
: Basically the random option.

Fuzzy Stroke
: A randomness value that is per stroke. Useful for getting color and size variation in on speed-paint brushes.

Fade
: How much the brush is affected over length, proportional to the brush size.

Perspective
: How much the brush is affected by the perspective assistant.

Tangential Pressure
: How much the brush is affected by the wheel on airbrush-simulating styli.

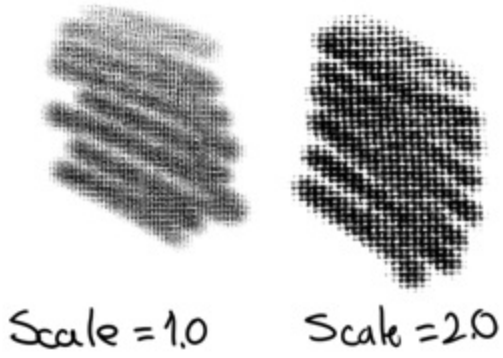
Texture

This allows you to have textured strokes. This parameter always shows up as two parameters:

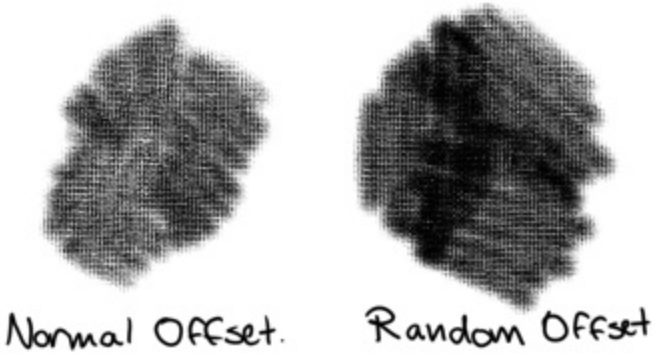
Texture

Pattern
: Which pattern you'll be using.

Scale
: The size of the pattern. 1.0 is 100%.



Horizontal Offset & Vertical Offset
: How much a brush is offset, random offset sets a new per stroke.



Texturing mode

Multiply Alpha

Uses alpha multiplication to determine the effect of the texture. Has a soft feel.

Subtract Alpha

Uses subtraction to determine the effect of the texture. Has a harsher, more texture feel.

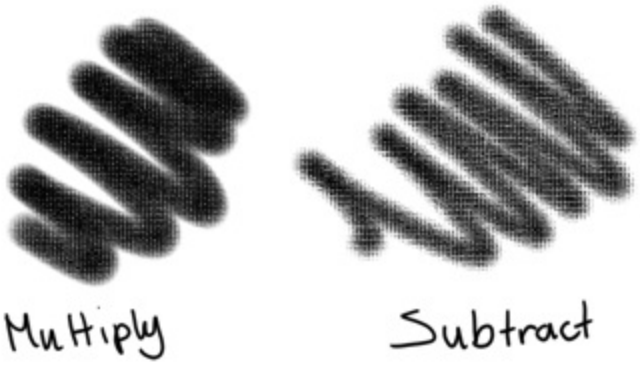
New in version 4.4: Lightness and Gradient Map options:

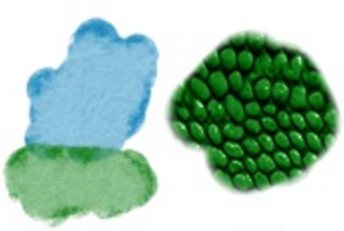
Lightness Map

Applies lightness values of the texture to the paint. Can be used to simulate paper/canvas, or for painting a texture, like reptile skin or tree bark.

Gradient Map

Maps gray/lightness values of the texture to the currently selected gradient. Useful for painting textures with multiple colors, like reptile skin, tree bark, stars, etc.





Lightness Map

Gradient Map

Cutoff policy

Cutoff policy will determine what range and where the strength will affect the textured outcome.

Disabled

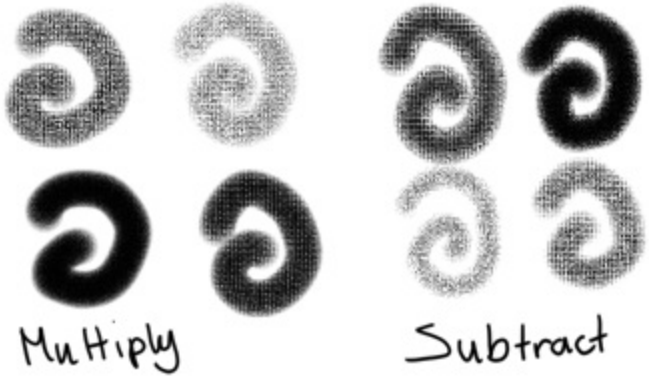
Doesn’t cut off. Full range will be used.

Pattern

Cuts the pattern off.

Brush

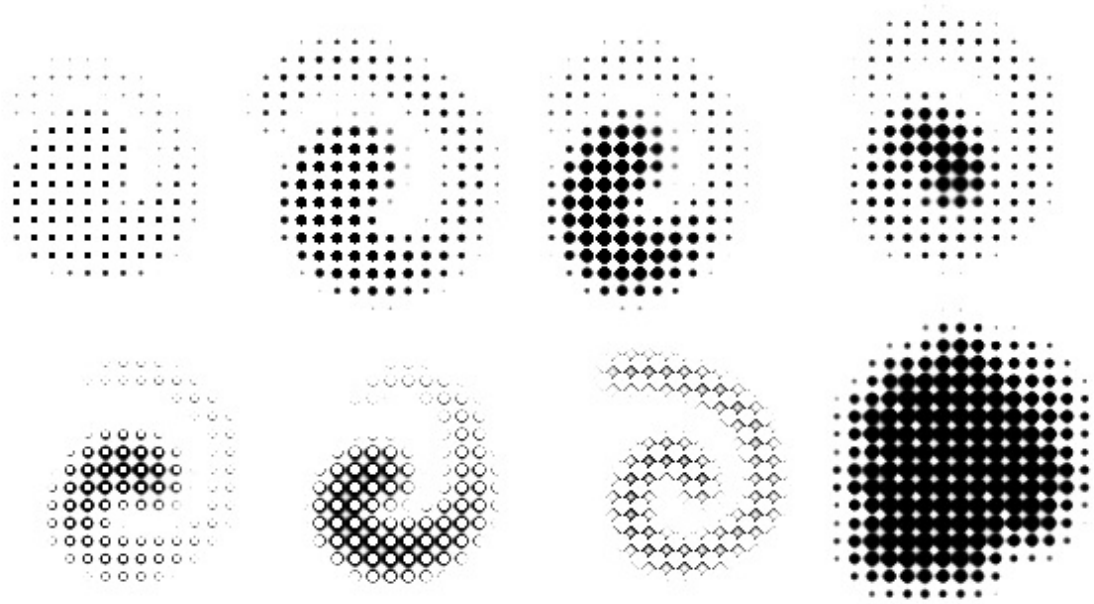
Cuts the brush tip off.



Cutoff

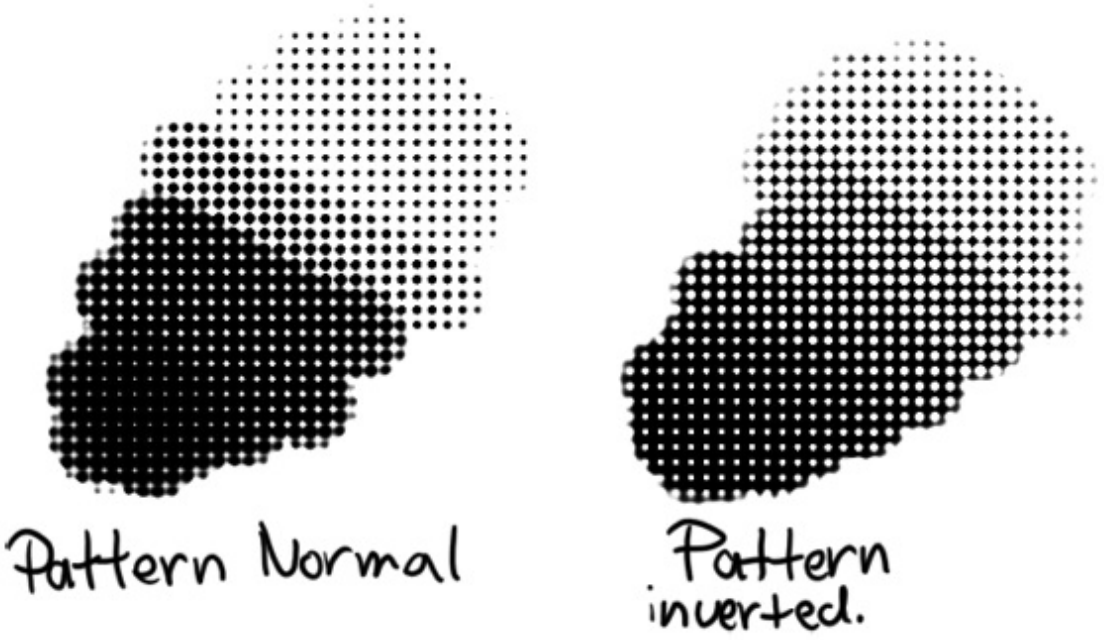
Cutoff is… the grayscale range that you can limit the texture to. This also affects the limit takes by the strength. In the below example, we move from the right arrow moved close to the left one, resulting in only the darkest values being drawn. After that, three images with larger range, and underneath that, three ranges with the left arrow moved, result in the

darkest values being cut away, leaving only the lightest. The last example is the pattern without cutoff.



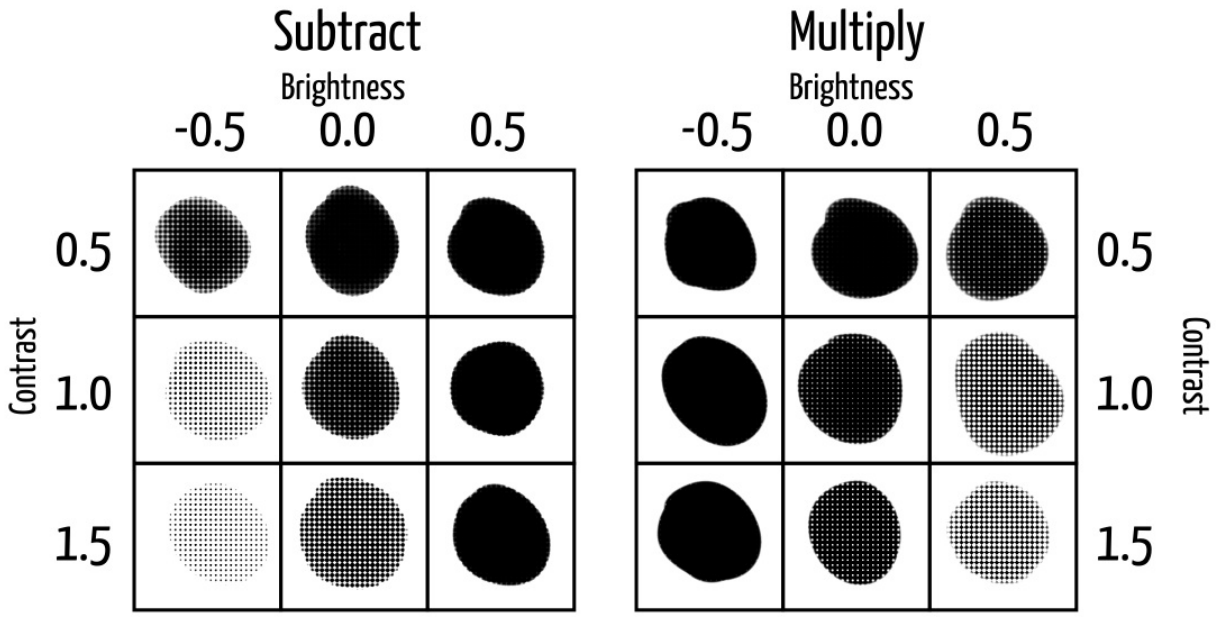
Invert Pattern

Invert the pattern.



Brightness and Contrast

New in version 3.3.1: Adjust the pattern with a simple brightness/contrast filter to make it easier to use. Because Subtract and Multiply work differently, it's recommended to use different values with each:



New in version 4.4: Neutral Point adjustment:

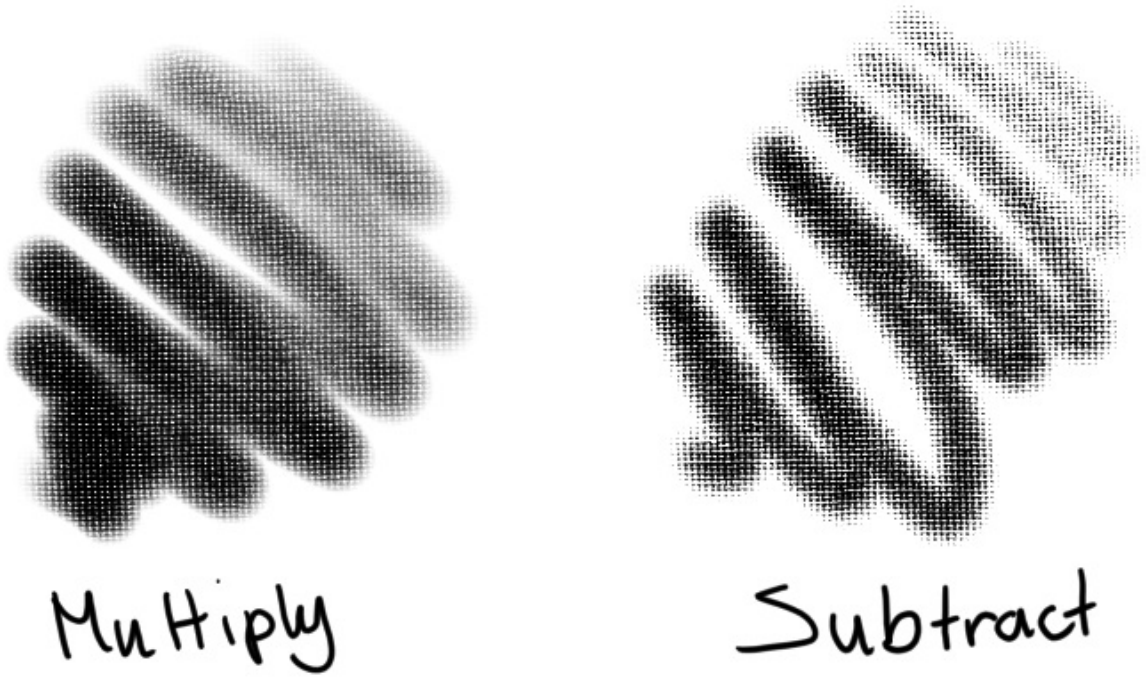
Neutral Point
: Adjust the gray value that is considered neutral in the texture. 0.5 keeps the texture as is; higher values make the texture darker, and lower values make the texture lighter. Works a bit differently from the brightness option, and is mostly useful to adjust existing textures to work well with Lightness Map and Gradient Map modes (though it can have applications with the other two modes).

Strength

This allows you to set the texture to Sensors. It will use the cutoff to continuously draw lighter values of the texture (making the result darker).

New in version 4.4: For Lightness Map and Gradient Map modes, *Strength* controls how much of the texture is applied compared to how much of the

selected paint color comes through.



See also

David Revoy describing the texture feature (old)
[https://www.davidrevoy.com/article107/textured-brush-in-floss-digital-painting].

Clones Array

Allows you to create a set of clone layers quickly. These are ordered in terms of rows and columns. The default options will create a 2 by 2 grid. For setting up tiles of an isometric game, for example, you'd want to set the X offset of the rows to half the value input into the X offset for the columns, so that rows are offset by half. For a hexagonal grid, you'd want to do the same, but also reduce the Y offset of the grids by the amount of space the hexagon can overlap with itself when tiled.

- Elements
  The amount of elements that should be generated using a negative of the offset.

+ Elements
  The amount of elements that should be generated using a positive of the offset.

X offset
  The X offset in pixels. Use this in combination with Y offset to position a clone using Cartesian coordinates.

Y offset
  The Y offset in pixels. Use this in combination with X offset to position a clone using Cartesian coordinates.

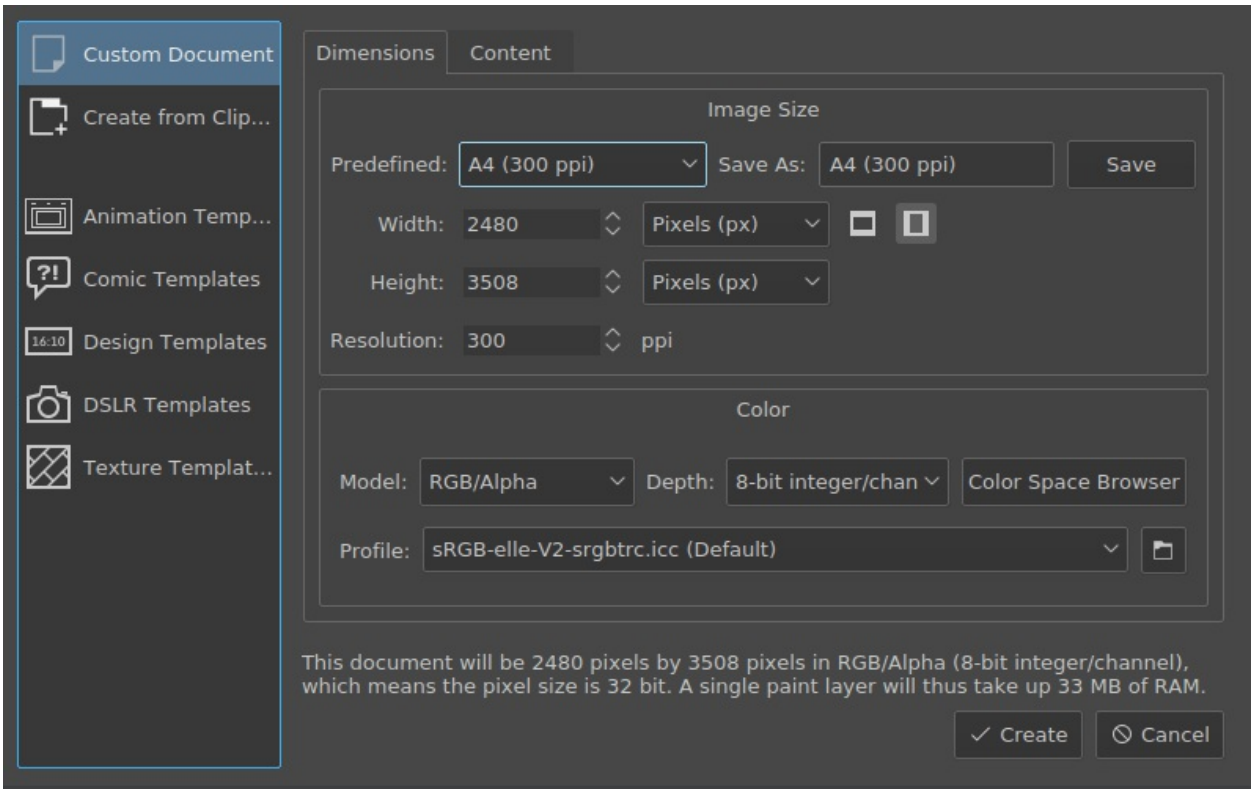
Distance
  The line-distance of the original origin to the clones origin. Use this in combination with angle to position a clone using a polar coordinate system.

Angle
  The angle-offset of the column or row. Use this in combination with distance to position a clone using a polar coordinate system.

Create New Document

A new document can be created as follows.

1. Click on *File* from the application menu at the top.
2. Then click on *New*. Or you can do this by pressing the `Ctrl + N` shortcut.
3. Now you will get a New Document dialog box as shown below:



There are various sections in this dialog box which aid in creation of new document, either using custom document properties or by using contents from clipboard and templates. Following are the sections in this dialog box:

Custom Document

From this section you can create a document according to your requirements: you can specify the dimensions, color model, bit depth, resolution, etc.

In the top-most field of the *Dimensions* tab, from the Predefined drop-down you can select predefined pixel sizes and PPI (pixels per inch). You can also set custom dimensions and the orientation of the document from the input fields below the *Predefined:* drop-down. This can also be saved as a new predefined preset for your future use by giving a name in the *Save Image Size as:* input box and clicking on the *Save* button. Below we find the Color section of the new document dialog box, where you can select the color model and the bit-depth. Check Colors for more detailed information regarding color.

On the *Content* tab, you can define a name for your new document. This name will appear in the metadata of the file, and Krita will use it for the auto-save functionality as well. If you leave it empty, the document will be referred to as 'Unnamed' by default. You can select the background color and the amount of layers you want in the new document. Krita remembers the amount of layers you picked last time, so be careful.

Finally, there's a description box, useful to note down what you are going to do.

Create From Clipboard

This section allows you to create a document from an image that is in your clipboard, like a screenshot. It will have all the fields set to match the clipboard image.

Templates:

These are separate categories where we deliver special defaults. Templates are just `.kra` files which are saved in a special location, so they can be pulled up by Krita quickly. You can make your own template file from any `.kra` file, by using *File* ▸ *Create Template from Image…* in the top menu. This will add your current document as a new template, including all its properties along with the layers and layer contents.

Once you have created a new document according to your preference, you

should now have a white canvas in front of you (or whichever background color you chose in the dialog).

Pre-installed Python plugins

This page describes all plugins that are available in Krita by default (you don’t need to install them).

See also

If you want to see a selection of custom user-made Python plugins that you can additionally download and install, see User-made Python Plugins.

To learn how to manage your plugins, see Managing Python plugins.

If you want to know more about an individual plugin, you can access the plugin’s manual by going to *Settings* ▸ *Configure Krita…* menu, and then choosing the `Python Plugin Manager` tab. Then you can click on a specific plugin and the manual will appear in the bottom text area.

Usability

Mixer Slider Docker

Docker that allows you to choose a color from gradients between the current color and other selected colors.

Palette Docker

Docker that allows you to control palettes more easily. You can add swatches, groups and export the palette settings, or even the palette itself as a GIMP Palette or Inkscape SVG.

Quick Settings Docker

Docker that allows you to quickly set the opacity, flow and size from a predefined list.

Ten Brushes

Plugin that assigns presets to one of ten configurable hotkeys. To use, go to *Tools* ▸ *Scripts* ▸ *Ten Brushes*, and a window will pop up with a preset chooser and ten boxes above it. Underneath the boxes is the hotkey the box is associated with.

Customize your shortcuts by editing the configurations in *Settings* ▸ *Configure Krita* ▸ *Keyboard Shortcuts*, and then change the "Activate Brush Preset" actions under "Ten Brushes".

Workflow Improvements

Comics Project Management Tools

Plugin that simplifies comics creation.

- Organize and quickly access their pages.
- Export to multiple formats with proper metadata.
- Random suggestions for metadata to avoid spending time on finding the perfect title before starting the project.

Batch Exporter

Plugin for Game Developers and Graphic Designers.

- Batch export of assets to multiple sizes, file types and custom paths.
- Renaming layers quickly with the smart rename tool.
- Export all layers or only selected layers.

By default, the plugin exports the images in an `export` folder next to the Krita document and follows the structure of your layer stack.

Image/Document Actions

Assign Profile Dialog

Allows you to assign a profile to an image instead of converting it to that profile. The difference is that it allows only interpreting the colors by the new profile, but not change any of the values. It can be found in *Tools* ▸ *Assign Profile to Image…*, and will present a list of profiles for the current image's color model.

Color Space

Allows you to select a document and convert its colors to a new color space, like RGBA, CMYKA or L*a*b.

Channels to Layers

Splits channels from a layer to sub-layers.

Document Tools

Allows you to select a document and scale, crop and rotate in one action.

Filter Manager

Quickly apply a filter on selected documents.

High Pass

Performs a high pass filter on the active document.

File Actions

Export Layers

Allows you to select a document and export its layers in an ordered and sensible manner.

Last Documents Docker

Script that shows the recently opened documents as a thumbnail image.

Python Scripting

Krita Script Starter

A script that helps set up the various files that Krita expects to see when it runs a script, namely:

- `.desktop` meta data file;
- the main directory for your plugin;
- `__init__.py` file;
- the main python file for your package;
- `Manual.html` file for your documentation;

Python Plugin Importer

Imports Python plugins from zip files. See Managing Python plugins.

Scripter

A small Python scripting console, allows to write code in an editor and run it, with feedback related to the output of the execution. You can also debug your code using the “Debug” button.

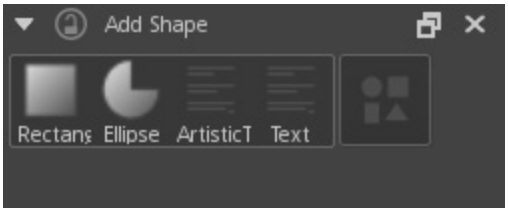
Ten Scripts

Similar to Ten Brushes, this plugin allows an assignment of Python scripts to ten configurable hotkeys.

Dockers

All of the panels that exist in Krita and what they do.

Add Shape



A docker for adding KOffice shapes to a Vector Layers.

Deprecated since version 4.0: This got removed in 4.0, the Vector Library replacing it.

Advanced Color Selector

As compared to other color selectors in Krita, Advanced color selector provides more control and options to the user. To open Advanced color selector choose *Settings* ▸ *Dockers* ▸ *Advanced Color Selector*. You can configure this docker by clicking on the little wrench icon on the top left corner. Clicking on the wrench will open a popup window with following tabs and options:

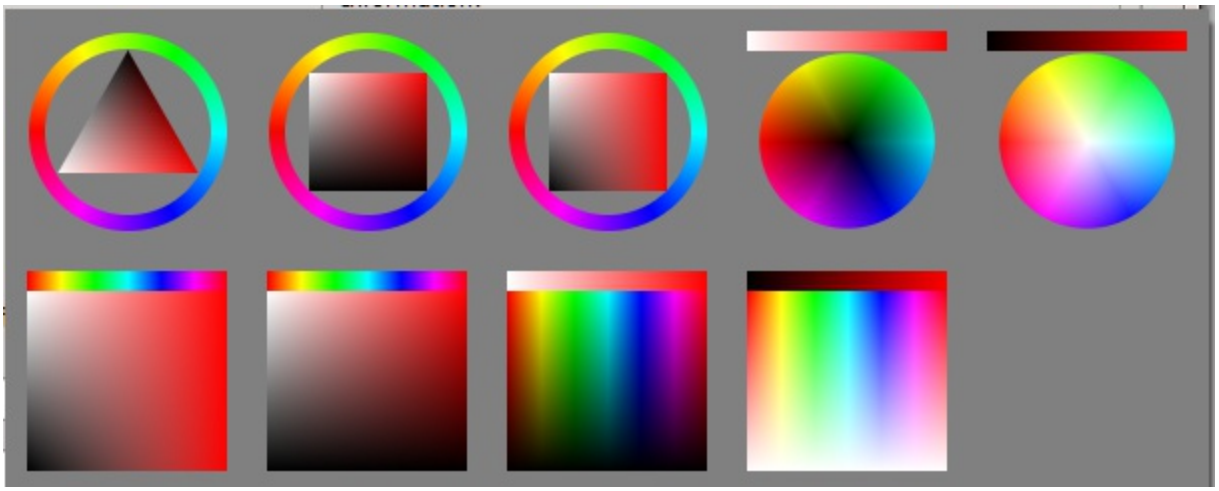
Color Selector

Here you configure the main selector.

Show Color Selector

New in version 4.2: This allows you to configure whether to show or hide the main color selector.

Type and Shape



Here you can pick the hsx model you'll be using. There's a small blurb explaining the characteristic of each model, but let's go into detail:

HSV

Stands for Hue, Saturation, Value. Saturation determines the difference between white, gray, black and the most colorful color. Value in turn measures either the difference between black and white, or the difference between black and the most colorful color.

HSL

Stands for Hue, Saturation, Lightness. All saturated colors are equal to 50% lightness. Saturation allows for shifting between gray and color.

HSI

This stands for Hue, Saturation and Intensity. Unlike HSL, this one determine the intensity as the sum of total rgb components. Yellow (1,1,0) has higher intensity than blue (0,0,1) but is the same intensity as cyan (0,1,1).

HSY'

Stands for Hue, Saturation, Luma, with Luma being an RGB approximation of true luminosity. (Luminosity being the measurement of relative lightness). HSY' uses the Luma Coefficients, like Rec. 709 [https://en.wikipedia.org/wiki/Rec._709], to calculate the Luma. Due to this, HSY' can be the most intuitive selector to work with, or the most confusing.

Then, under shape, you can select one of the shapes available within that color model.

Note

Triangle is in all color models because to a certain extent, it is a wildcard shape: All color models look the same in an equilateral triangle selector.

Luma Coefficients

This allows you to edit the Luma coefficients for the HSY model selectors to your leisure. Want to use Rec. 601 [https://en.wikipedia.org/wiki/Rec._601] instead of Rec. 709? These boxes allow you to do that!

By default, the Luma coefficients should add up to 1 at maximum.

Gamma
: The HSY selector is linearised, this setting allows you to choose how much gamma is applied to the Luminosity for the gui element. 1.0 is fully linear, 2.2 is the default.

Color Space

This allows you to set the overall color space for the Advanced Color Selector.

Warning

You can pick only sRGB colors in advanced color selector regardless of the color space of advanced color selector. This is a bug.

Behavior

When docker resizes

This determines the behavior of the widget as it becomes smaller.

Change to Horizontal
: This'll arrange the shade selector horizontal to the main selector. Only works with the MyPaint shade selector.

Hide Shade Selector.
: This hides the shade selector.

Do nothing
: Does nothing, just resizes.

Zoom selector UI

If your have set the docker size considerably smaller to save space, this option might be helpful to you. This allows you to set whether or not the selector will give a zoomed view of the selector in a size specified by you, you have these options for the zoom selector:

- when pressing middle mouse button
- on mouse over
- never

The size given here, is also the size of the Main Color Selector and the MyPaint Shade Selector when they are called with the `Shift + I` and `Shift + M` shortcuts, respectively.

Hide Pop-up on click
: This allows you to let the pop-up selectors called with the above hotkeys to disappear upon clicking them instead of having to leave the pop-up boundary. This is useful for faster working.

Shade selector

Shade selector options. The shade selectors are useful to decide upon new shades of color.

Update Selector

This allows you to determine when the shade selector updates.

MyPaint Shade Selector

Ported from MyPaint, and extended with all color models. Default hotkey is `Shift + M`.

Simple Shade Selector

This allows you to configure the simple shade selector in detail.

Color Patches

This sets the options of the color patches.

Both Color History and Colors From the Image have similar options which will be explained below.

Show
: This is a radio button to show or hide the section. It also determines whether or not the colors are visible with the advanced color selector docker.

Size
: The size of the color boxes can be set here.

Patch Count
: The number of patches to display.

Direction
: The direction of the patches, Horizontal or Vertical.

Allow Scrolling
: Whether to allow scrolling in the section or not when there are too many patches.

Number of Columns/Rows
: The number of Columns or Rows to show in the section.

Update After Every Stroke
: This is only available for Colors From the Image and tells the docker whether to update the section after every stroke or not, as after each stroke the colors will change in the image.

History patches

The history patches remember which colors you've drawn on canvas with. They can be quickly called with the H key.

Common Patches

The common patches are generated from the image, and are the most common color in the image. The hotkey for them on canvas is the U key.

Gamut masking

New in version 4.2.

Note

Gamut masking is available only when the selector shape is set to wheel.

You can select and manage your gamut masks in the Gamut Masks Docker.

In the gamut masking toolbar at the top of the selector you can toggle the selected mask off and on (left button). You can also rotate the mask with the rotation slider (right).

External Info

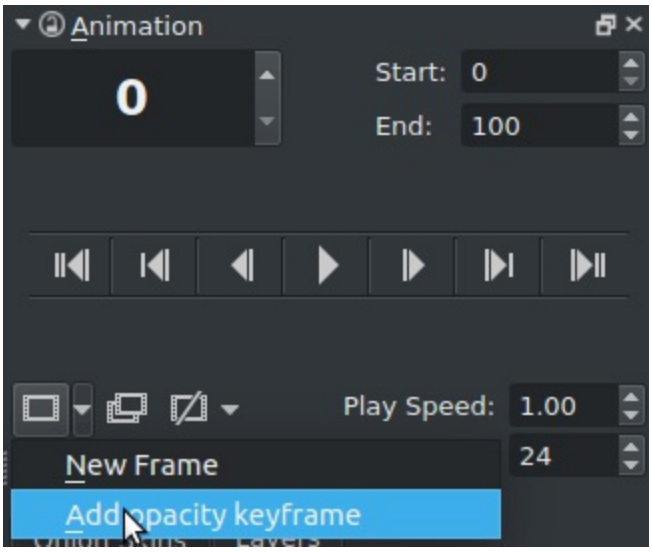
HSI and HSY for Krita's advanced color selector [https://wolthera.info/?p=726].

Animation Curves Docker

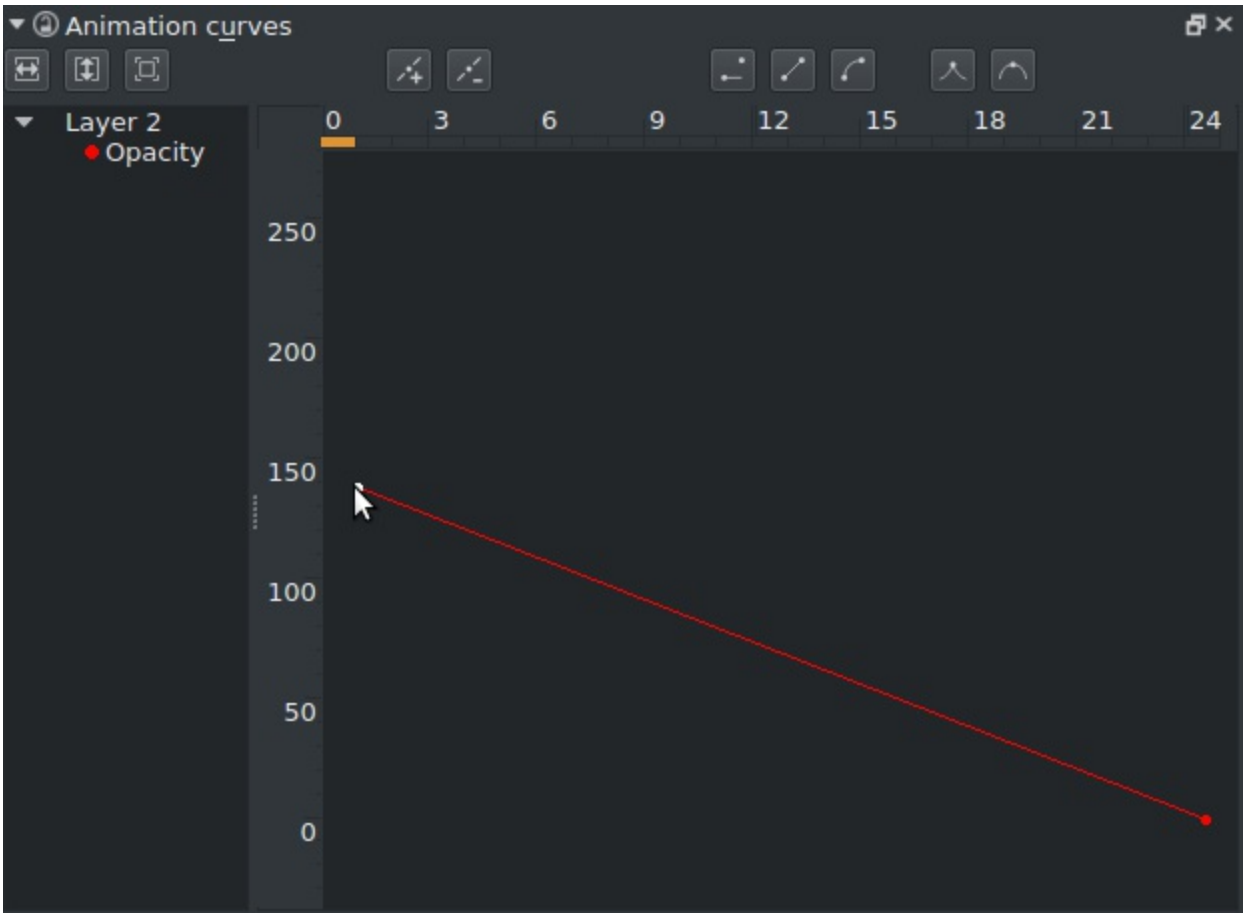
The Animation Curve docker allows you to edit tweened sections by means of interpolation curves. As of this time of writing, it can only edit opacity.

The idea is that sometimes what you want to animate can be expressed as a value. This allows the computer to do maths on the values, and automate tasks, like interpolation, also known as ‘Tweening’. Because these are values, like percentage opacity, and animation happens over time, that means we can visualize the way the values are interpolated as a curve graph, and also edit the graph that way.

But, when you first open this docker, there’s no curves visible! You will first need to add opacity keyframes to the active animation layer. You can do this by using the animation docker and selection *Add new keyframe*.



Opacity should create a bright red curve line in the docker. On the left, in the layer list, you will see that the active layer has an outline of its properties: A red *Opacity* has appeared. Pressing the red dot will hide the current curve, which’ll be more useful in the future when more properties can be animated.



If you select a dot of the curve, you can move it around to shift its place in the time-line or its value.

On the top, you can select the method of smoothing:

Hold Value
: This keeps the value the same until there's a new keyframe.

Linear Interpolation (Default)
: This gives a straight interpolation between two values.

Custom interpolation
: This allows you to set the section after the keyframe node as one that can be modified.  +dragging on the node allows you to drag out a handler node for adjusting the curving.

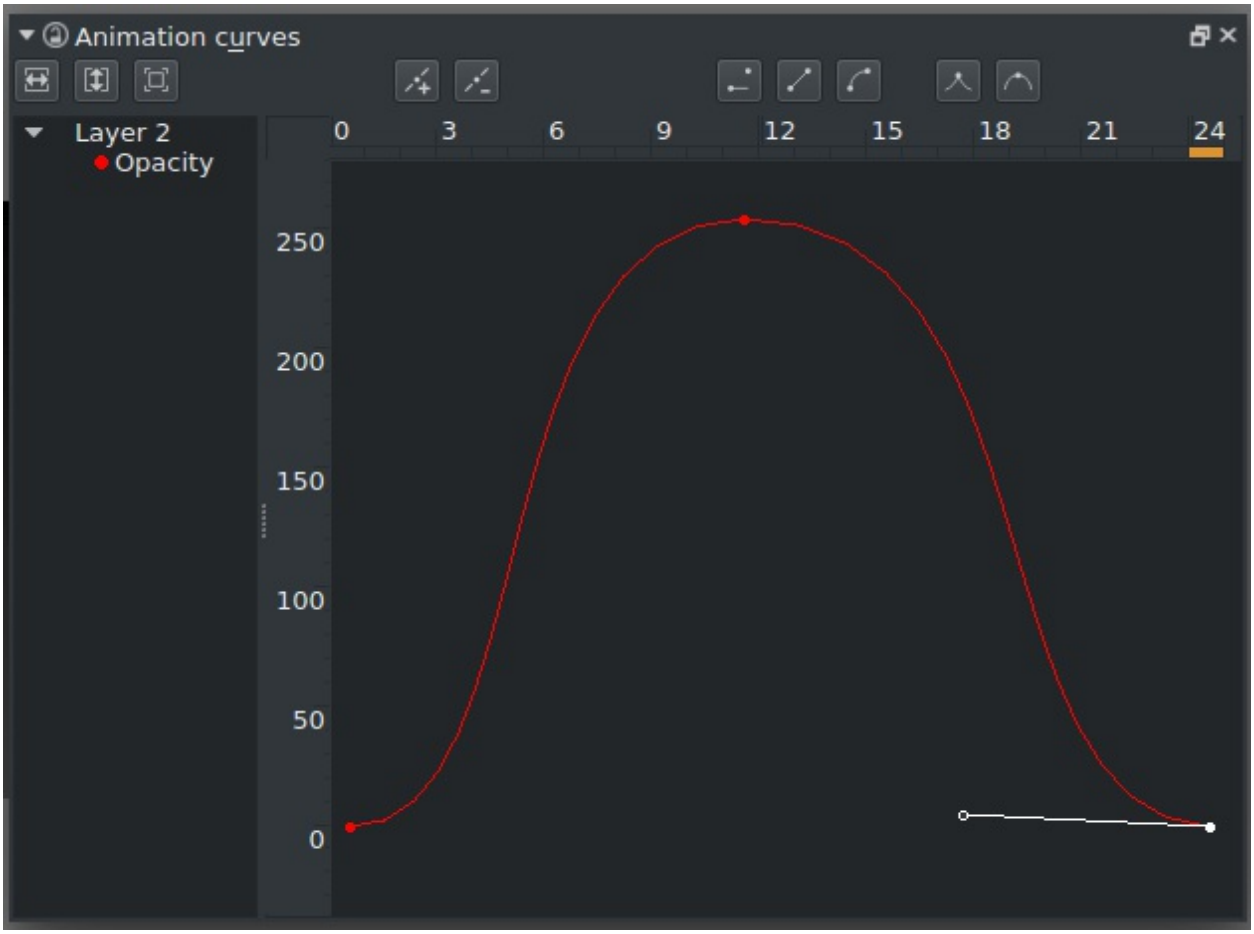
So, for example, making a 100% opacity keyframe on frame 0 and a 0%

opacity one on frame 24 gives the following result:

If we select frame 12 and press *Add New Keyframe* a new opacity keyframe will be added on that spot. We can set this frame to 100% and set frame 0 to 0% for this effect.

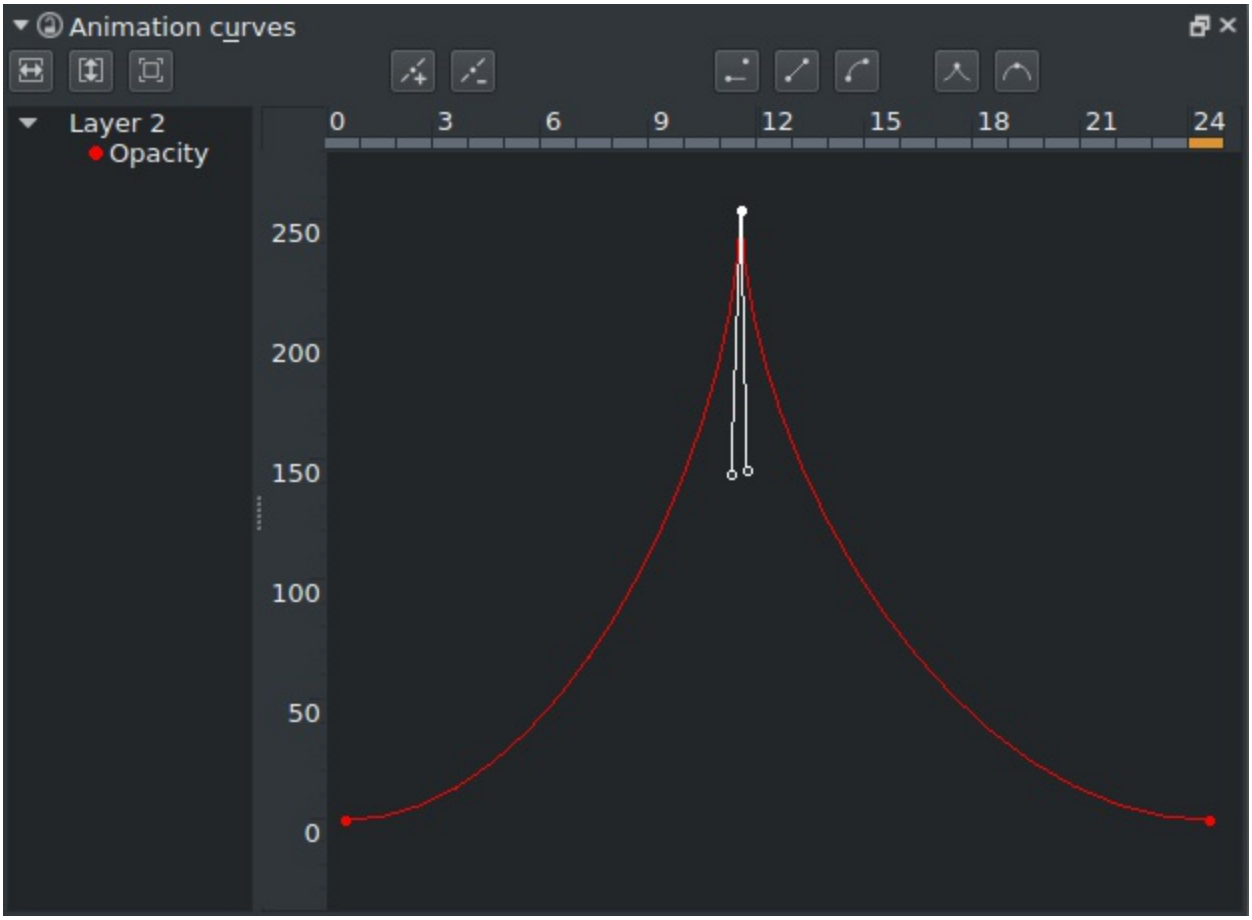
Now, if we want easing in, we select the node on frame 0 and press the *Custom Interpolation* button at the top. This will enable custom interpolation on the curve between frames 0 and 12. Doing the same on frame 12 will enable custom interpolation between frames 12 and 24. Drag from the node

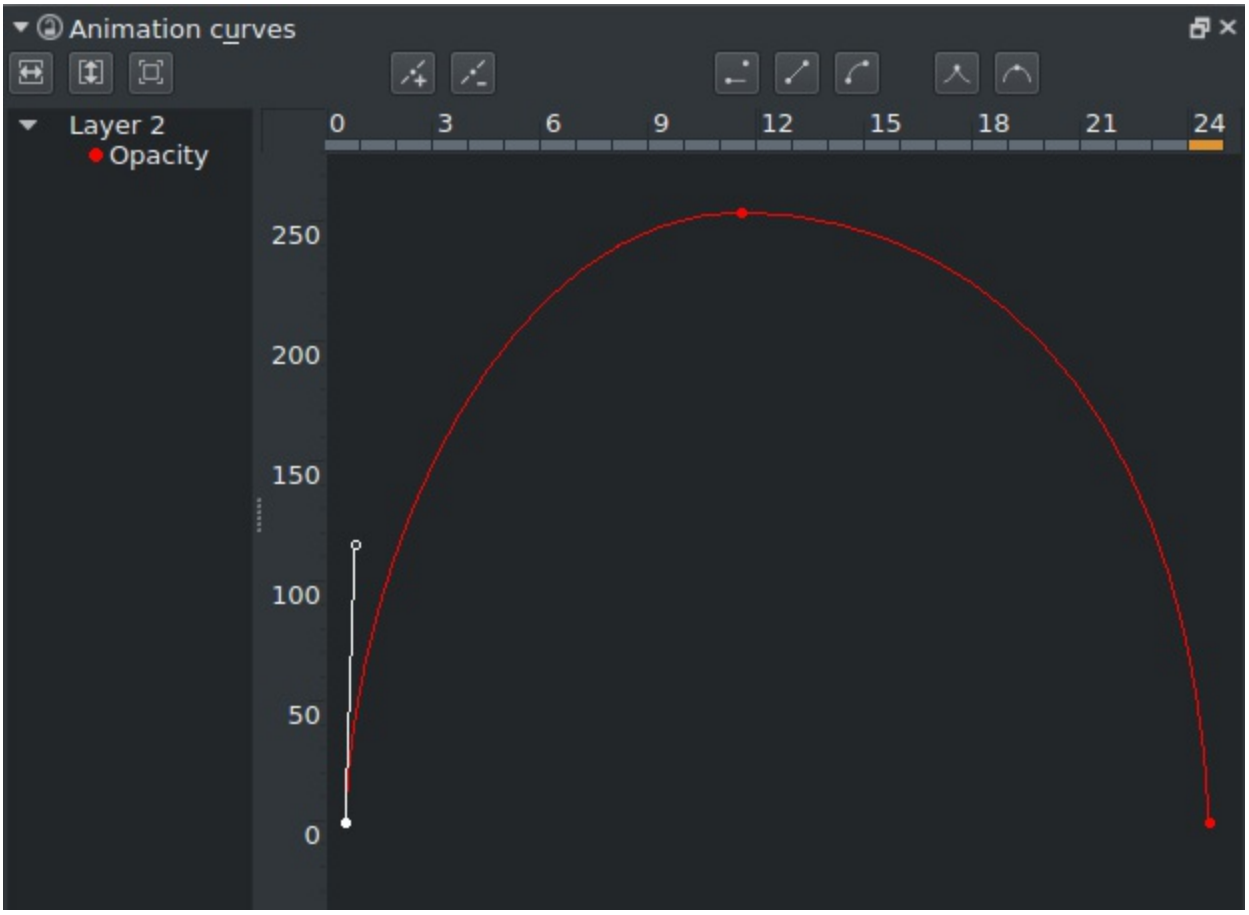
to add a handle, which in turn you can use to get the following effects:



The above shows an ease-in curve.

And convex/concave examples:





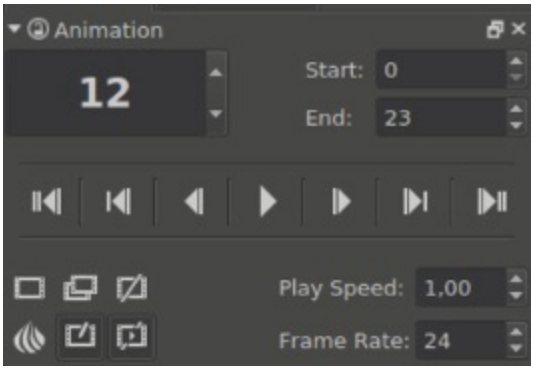
As you may be able to tell, there’s quite a different ‘texture’, so to speak, to each of these animations, despite the difference being only in the curves. Indeed, a good animator can get quite some tricks out of interpolation curves,

and as we develop Krita, we hope to add more properties for you to animate this way.

Note

Opacity has currently 255 as maximum in the curve editor, as that’s how opacity is stored internally.

Animation Docker



To have a playback of the animation, you need to use the animation docker.

The first big box represents the current Frame. The frames are counted with programmer's counting so they start at 0.

Then there are two boxes for you to change the playback range here. So, if you want to do a 10 frame animation, set the end to 10, and then Krita will cycle through the frames 0 to 10.

The bar in the middle is filled with playback options, and each of these can also be hot-keyed. The difference between a keyframe and a normal frame in this case is that a normal frame is empty, while a keyframe is filled.

Then, there's buttons for adding, copying and removing frames. More interesting is the next row:

Onion Skin
: Opens the Onion Skin Docker if it wasn't open before.

Auto Frame Mode
: Will make a frame out of any empty frame you are working on. Currently automatically copies the previous frame.

Drop frames
: This'll drop frames if your computer isn't fast enough to show all frames

at once. This process is automatic, but the icon will become red if it’s forced to do this.

You can also set the speedup of the playback, which is different from the framerate.

Arrange

A docker for aligning and arranging vector shapes. When you have the Shape Selection Tool active, the following actions will appear on this docker:

Align
: Align all selected objects.

  - Align Left
  - Horizontally Center
  - Align Right
  - Align Top
  - Vertically Center
  - Align Bottom

Distribute
: Ensure that objects are distributed evenly.

  - Distribute left edges equidistantly.
  - Distribute centers equidistantly horizontally.
  - Distribute right edges equidistantly.
  - Distribute top edges equidistantly.
  - Distribute centers equidistantly vertically.
  - Distribute bottom edges equidistantly.

Spacing
: Ensure the gaps between objects are equal.

  - Make horizontal gaps between object equal.
  - Make vertical gaps between object equal.

Order
: Change the order of vector objects.

  - Bring to front
  - Raise
  - Lower
  - Bring to back

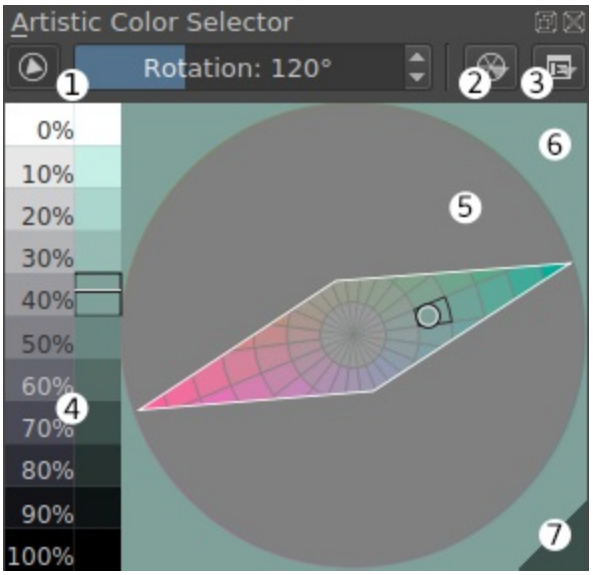
Grouping

Buttons to group and ungroup vector objects.

Artistic Color Selector Docker

A color selector inspired by traditional color wheel and workflows.

Usage



Artistic color selector with a gamut mask.

Select hue and saturation on the wheel (5) and value on the value scale (4). changes foreground color (6). changes background color (7).

The blip shows the position of current foreground color on the wheel (black&white circle) and on the value scale (black&white line). Last selected swatches are outlined.

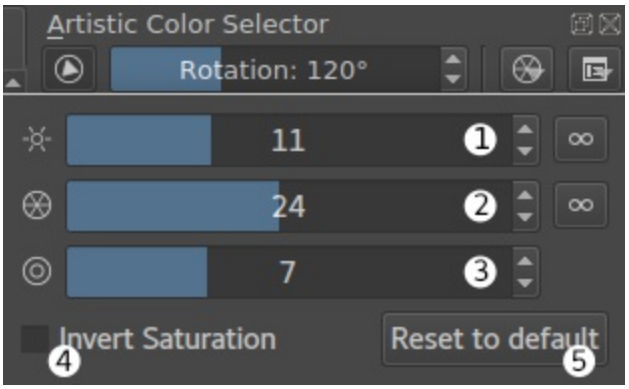
Parameters of the wheel can be set in Color wheel preferences menu (2). Selector settings are found under Selector settings menu (3).

Gamut Masking

You can select and manage your gamut masks in the Gamut Masks Docker.

In the gamut masking toolbar (1) you can toggle the selected mask off and on (left button). You can also rotate the mask with the rotation slider (right).

Color wheel preferences



Color wheel preferences.

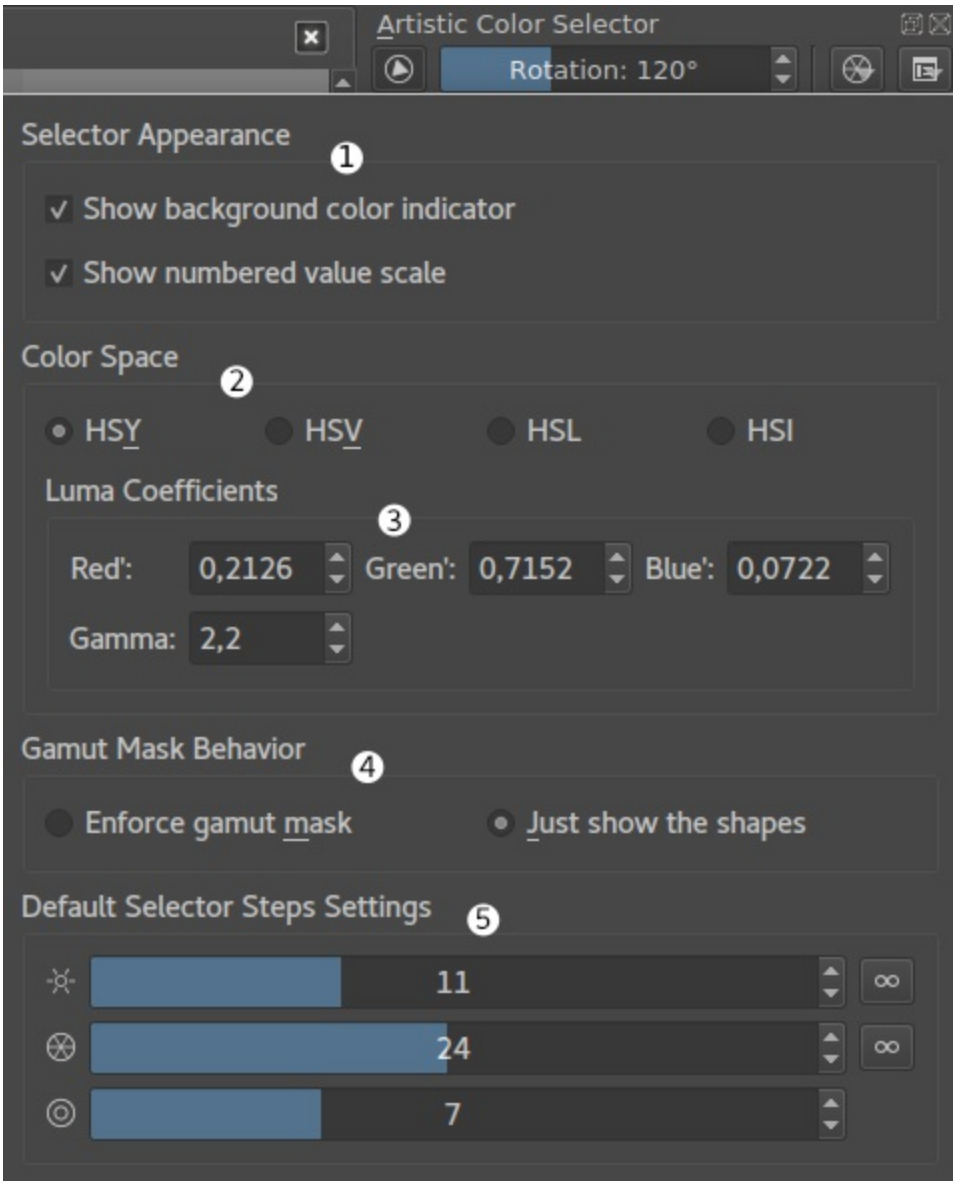
Sliders 1, 2, and 3
: Adjust the number of steps of the value scale, number of hue sectors and saturation rings on the wheel, respectively.

Continuous Mode
: The value scale and hue sectors can also be set to continuous mode (with the infinity icon on the right of the slider). If toggled on, the respective area shows a continuous gradient instead of the discrete swatches.

Invert saturation (4)
: Changes the order of saturation rings within the the hue sectors. By default, the wheel has gray in the center and most saturated colors on the perimeter. *Invert saturation* puts gray on the perimeter and most saturated colors in the center.

Reset to default (5)
: Loads default values for the sliders 1, 2, and 3. These default values are configured in selector settings.

Selector settings



Selector settings menu.

Selector Appearance (1)

Show background color indicator

Toggles the bottom-right triangle with current background color.

Show numbered value scale

If checked, the value scale includes a comparative gray scale with lightness percentage.

Color Space (2)

Set the color model used by the selector. For detailed information on color models, see Color Models.

Luma Coefficients (3)
: If the selector's color space is HSY, you can set custom Luma coefficients and the amount of gamma correction applied to the value scale (set to 1.0 for linear scale; see Gamma and Linear).

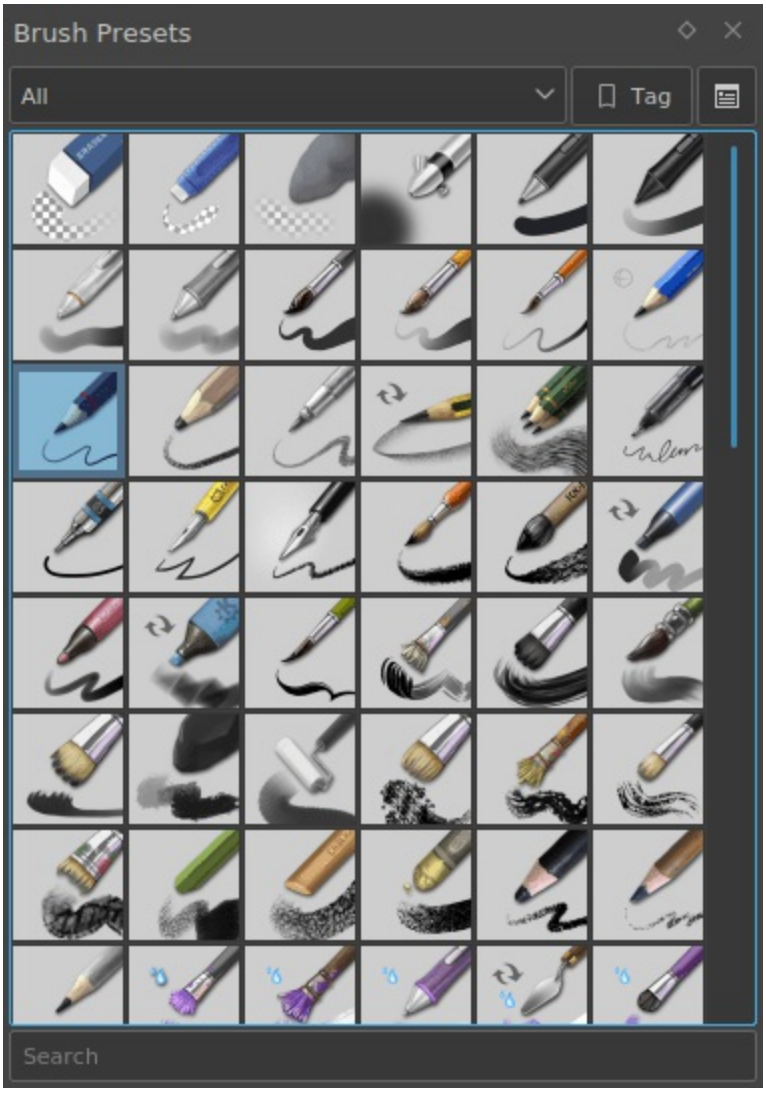
Gamut Masking Behavior (4)
: The selector can be set either to *Enforce gamut mask*, so that colors outside the mask cannot be selected, or to *Just show the shapes*, where the mask is visible but color selection is not limited.

Default Selector Steps Settings
: Values the color wheel and value scale will be reset to default when the *Reset to default* button in Color wheel preferences is pressed.

External Info

- HSI and HSY for Krita's advanced colour selector by Wolthera van Hövell tot Westerflier [https://wolthera.info/?p=726].
- The Color Wheel, Part 7 by James Gurney [https://gurneyjourney.blogspot.com/2010/02/color-wheel-part-7.html].

Preset Docker



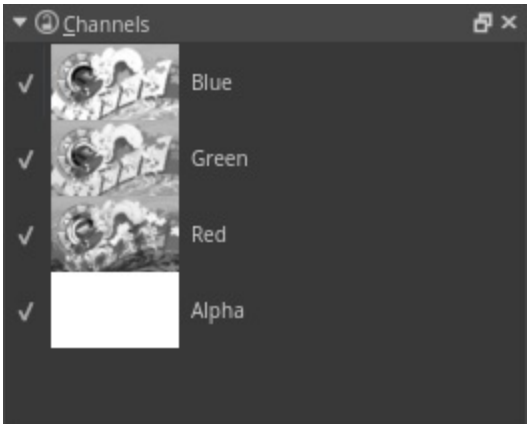
This docker allows you to switch the current brush you're using, as well as tagging the brushes.

Just  on an icon to switch to that brush!

Tagging

 a brush to add a tag or remove a tag.

Channels



The channel docker allows you to turn on and off the channels associated with the color space that you are using. Each channel has an enabled and disabled checkbox. You cannot edit individual layer channels from this docker.

Editing Channels

If you want to edit individual channels by their grayscale component, you will need to manually separate a layer. This can be done with a series of commands with the layer docker.

1. Select the layer you want to break apart.
2. Go to *Image* ▸ *Separate Image*.
3. Select the following options and click *OK*:
    1. Source: Current Layer.
    2. Alpha Options: Create separate separation from alpha channel.
    3. Output to Grayscale, not color: unchecked.
4. Hide your original layer.

5. Select All of the new channel layers and put them in a group layer (*Layer* ▸ *Quick Group*).

6. Select the Red layer and change the blending mode to “Copy Red” (these are in the Misc. category).

7. Select the Green layer and change the blending mode to “Copy Green”.

8. Select the Blue layer and change the blending mode to “Copy Blue” .

9. Make sure the Alpha layer is at the bottom of the group.

10. Enable Inherit Alpha for the Red, Green, and Blue layers.

Here is a video to see this process [https://www.youtube.com/watch?v=lWuwegJ-mIQ&feature=youtu.be] in Krita 3.0.

When working with editing channels, it can be easier to use the Isolate Layer feature to only see the channel. Right-click on the layer to find Isolate Layer.

Color Sliders

Deprecated since version 4.1: This docker has been removed in 4.1. It will return in some form in the future.

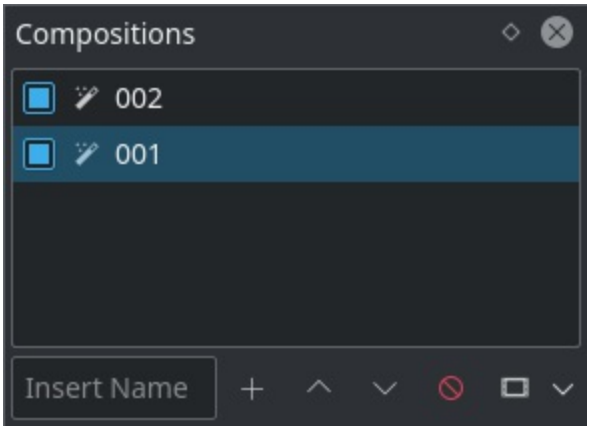
A small docker with Hue, Saturation and Lightness bars.

You can configure this docker via *Settings* ▸ *Configure Krita…* ▸ *Color Selector Settings* ▸ *Color Sliders*.

There, you can select which sliders you would like to see added, allowing you to even choose multiple lightness sliders together.

Compositions

The compositions docker allows you to save the configurations of your layers being visible and invisible, allowing you to save several configurations of your layers.



Adding new compositions
: You do this by setting your layers as you wish, then pressing the plus sign. If you had a word in the text-box to the left, this will be the name of your new composition.

Activating composition
: Double-click the composition name to switch to that composition.

..versionadded::4.4

Rearranging compositions
: You can rearrange compositions by using the up/down buttons.

Removing compositions
: The minus sign. Select a composition, and hit this button to remove it.

Exporting compositions
: The file sign. Will export all checked compositions.

New in version 4.4: It is also possible to render animations for each selected composition. This will use the settings last used in the render animation dialog, simplifying the export process.

Updating compositions

 a composition to overwrite it with the current configuration.

Rename composition

 a composition to rename it.

Digital Color Mixer

This docker allows you to do simple mathematical color mixing.

It works as follows:

You have on the left side the current color.

Next to that there are six columns. Each of these columns consists of three rows: The lowest row is the color that you are mixing the current color with. Ticking this button allows you to set a different color using a palette and the mini-color wheel. The slider above this mixing color represent the proportions of the mixing color and the current color. The higher the slider, the less of the mixing color will be used in mixing. Finally, the result color. Clicking this will change your current color to the result color.

Gamut Masks Docker

New in version 4.2: Docker for gamut masks selection and management.

Usage

 an icon (1) to apply a mask to color selectors.

Gamut Masks can be imported and exported in the resource manager.

Management Toolbar

Create new mask (2)
: Opens the mask editor with an empty template.

Edit mask (3)

Opens the the currently selected mask in the editor.

Duplicate mask (4)
: Creates a copy of the currently selected mask and opens the copy in the editor.

Delete mask (5)
: Deletes the currently selected mask.

Editing

If you choose to create a new mask, edit, or duplicate selected mask, the mask template document will be opened as a new view (1).

There you can create new shapes and modify the mask with standard vector tools (Vector Graphics).

Fill in the fields at (2).

Title (Mandatory)
: The name of the gamut mask.

Description
: A description.

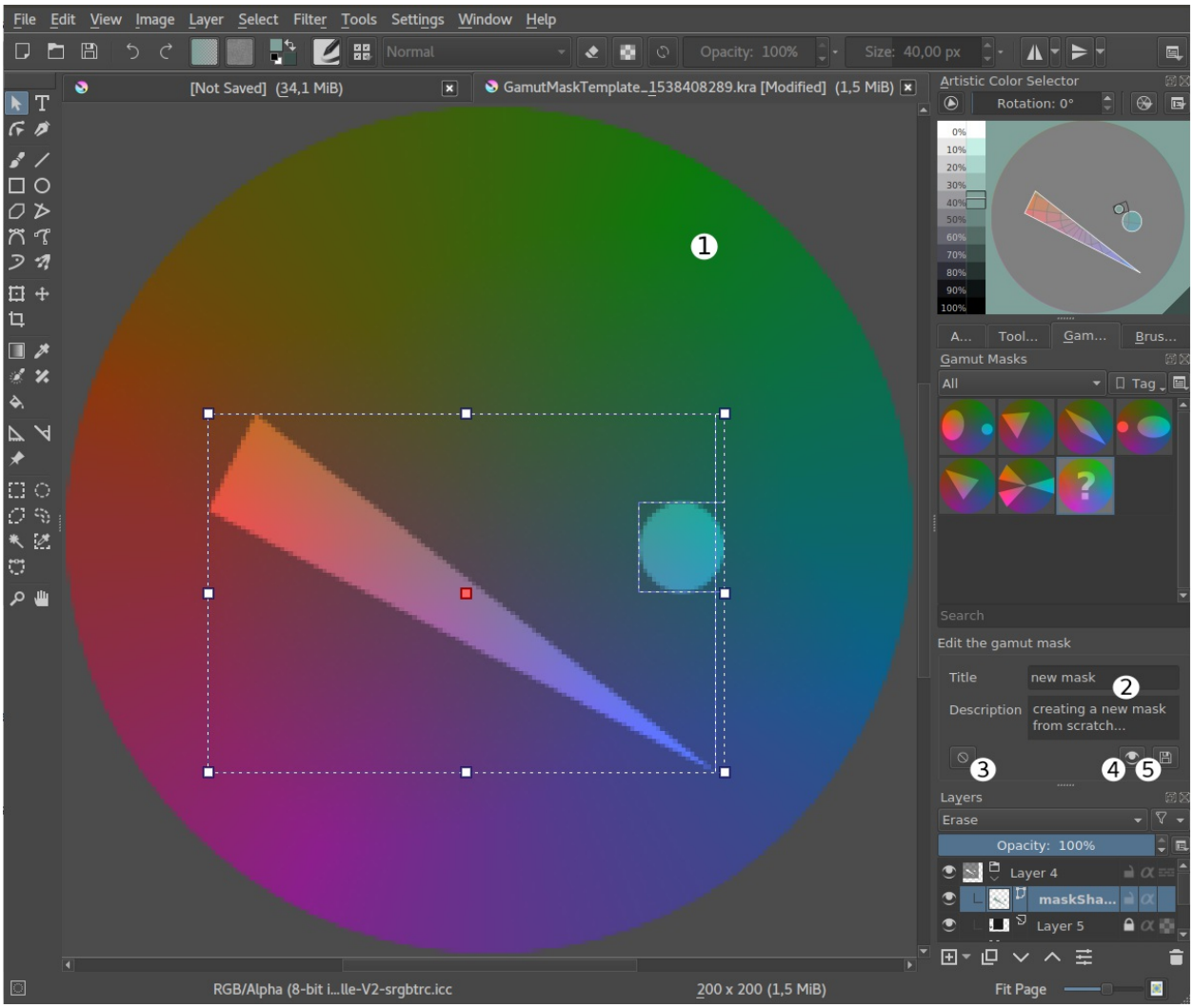
Preview the mask in the artistic color selector (4), *save* the mask (5), or *cancel* editing (3).

Warning

- The shapes need to be added to the layer named “maskShapesLayer” (which is selected by default).
- The shapes need have solid background to show correctly in the editor.
- A template with no shapes cannot be saved.

Note

The mask is intended to be composed of basic vector shapes. Although interesting results might arise from using advanced vector drawing techniques, not all features are guaranteed to work properly (e.g. grouping, vector text, etc.).



External Info

- [Color Wheel Masking, Part 1 by James Gurney] [https://gurneyjourney.blogspot.com/2008/01/color-wheel-masking-part-1.html].
- [The Shapes of Color Schemes by James Gurney] [https://gurneyjourney.blogspot.com/2008/02/shapes-of-color-schemes.html].
- [Gamut Masking Demonstration by James Gourney (YouTube)]

[https://youtu.be/qfE4E5goEIc].

Grids and Guides Docker

This docker controls the look and the visibility of both the Grid and the Guides decorations. It also features a checkbox to quickly toggle snapping on or off.

Grids

Grids in Krita can currently only be orthogonal and diagonal. There is a single grid per canvas, and it is saved within the document. Thus it can be saved in a Templates.

Show Grid
: Shows or hides the grid.

Snap to Grid
: Toggles grid snapping on or off. This can also be achieved with the `Shift + S` shortcut.

Type
: The type of Grid.

  Rectangle
  : An orthogonal grid.

    X and Y spacing
    : Sets the width and height of the grid in pixels.

    Subdivision
    : Groups cells together as larger squares and changes the look of the lines it contains. A subdivision of 2 will make cells appear twice as big, and the inner lines will become subdivisions.

  Isometric

A diagonal grid. Isometric doesn’t support snapping.

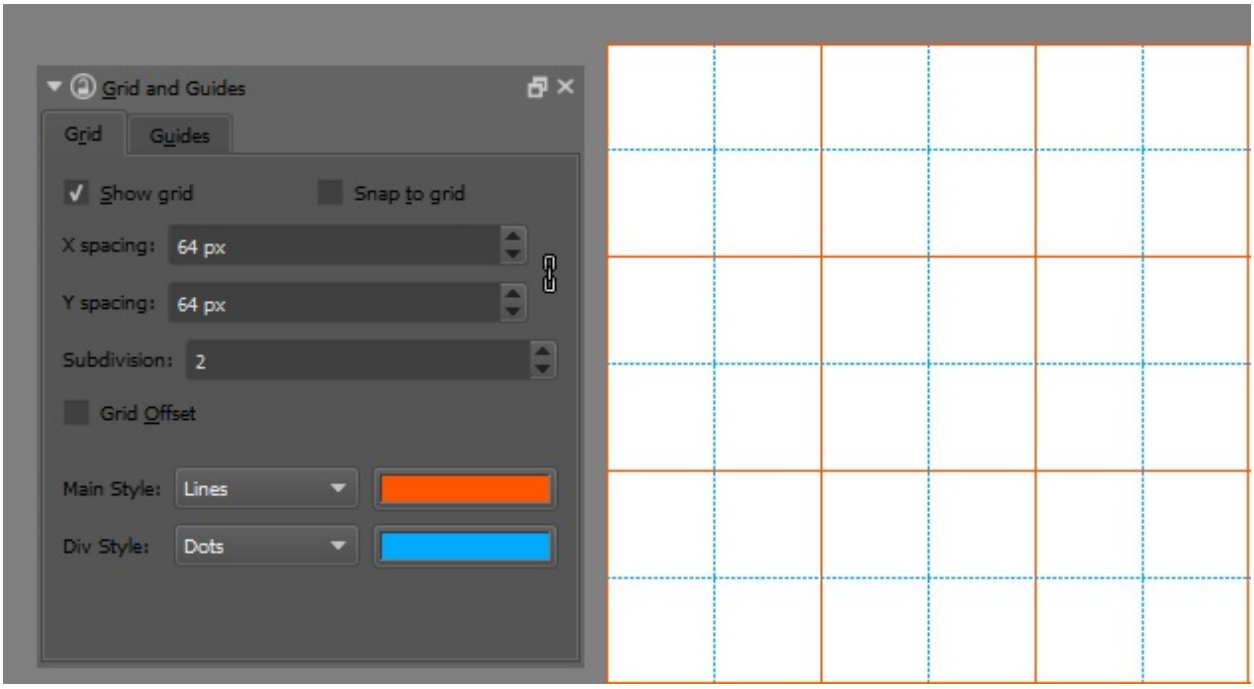
Left and Right Angle
: The angle of the lines. Set both angles to 30° for true isometric.

Cell spacing
: Determines how much both sets of lines are spaced.

Grid Offset
: Offsets the grid’s starting position from the top-left corner of the document, in pixels.

Main Style
: Controls the look of the grid’s main lines.

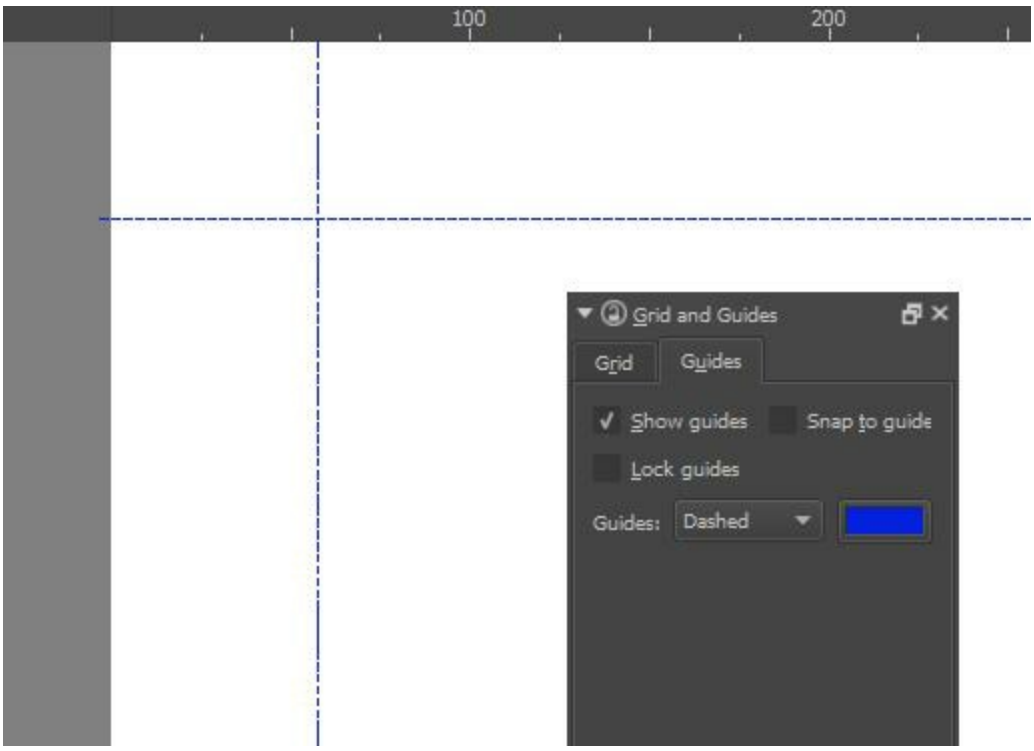
Div Style
: Controls the look of the grid’s “subdivision” lines.



The grid’s base size is 64 pixels. With a subdivision of 2, the main grid lines are 128 px away from one another, and the intermediate lines have a different look.

Guides

Guides are horizontal and vertical reference lines. You can use them to place and align layers accurately on the canvas.



Creating Guides

To create a guide, you need both the rulers and the guides to be visible.

1. Rulers. (*View* ▸ *Show Rulers*)
2. Guides. (*View* ▸ *Show Guides*)

To create a guide, move your cursor over a ruler and drag in the direction of the canvas. A line will appear. Dragging from the left ruler creates a vertical guide, and dragging from the top ruler creates a horizontal guide.

Editing Guides

Place your cursor above a guide on the canvas. If the guides are not locked, your cursor will change to a double arrow. In that case, click and drag to

move the guide. To lock and unlock the guides, open the Grid and Guides Docker. Ensure that the Guides tab is selected. From here you can lock the guides, enable snapping, and change the line style.

Note

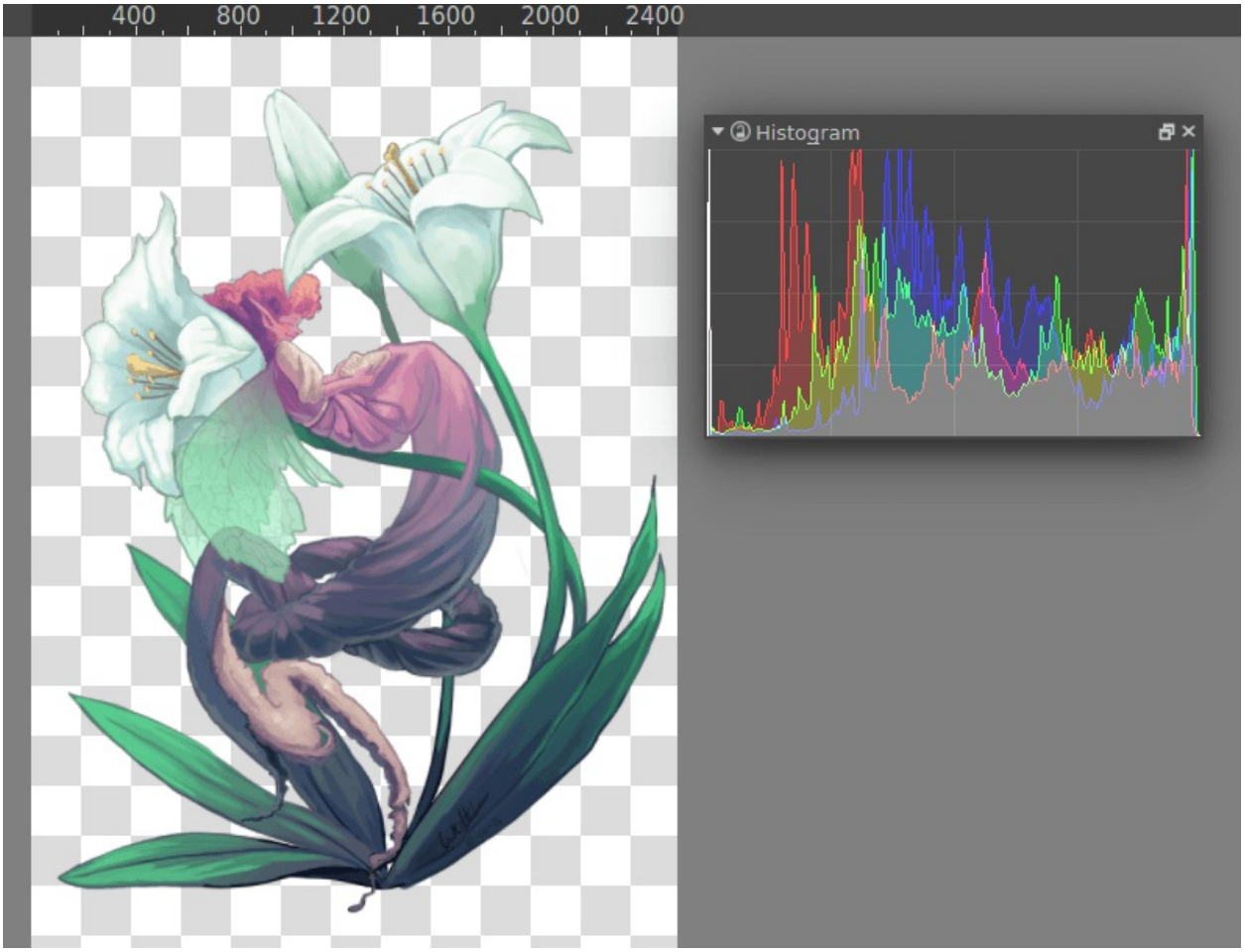
Currently, it is not possible to create or to move guides to precise positions. The only way to achieve that for now is to zoom in on the canvas, or to use the grid and snapping to place the guide.

Removing Guides

Click on the guide you want to remove and drag it outside of the canvas area. When you release your mouse or stylus, the guide will be removed.

Histogram Docker

A Histogram is a chart that shows how much of a specific channel value is used in an image. Its purpose is to give a really technical representation of the colors in an image, which can be helpful in decision making about filters.

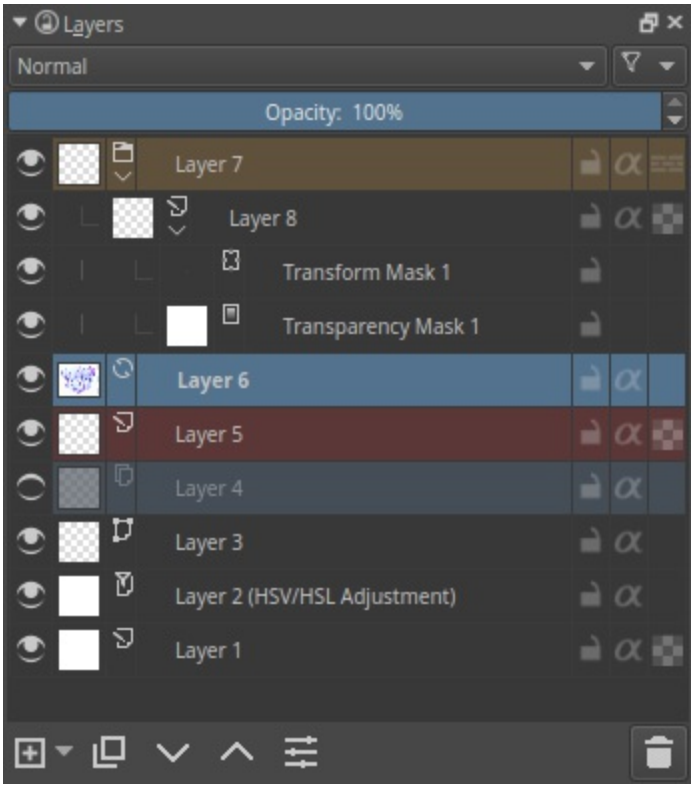


The histogram docker was already available via *Layers* ▸ *Histogram*, but it's now a proper docker.

External Links:

- Wikipedia's entry on image histograms
  [https://en.wikipedia.org/wiki/Image_histogram].

Layers



The Layers docker is for one of the core concepts of Krita: Layer Management. You can add, delete, rename, duplicate and do many other things to layers here.

The Layer Stack

You can select the active layer here. Using the `Shift` and `Ctrl` keys you can select multiple layers and drag-and-drop them. You can also change the visibility, edit state, alpha inheritance and rename layers. You can open and close groups, and you can drag and drop layers, either to reorder them, or to put them in groups.

Name

The Layer name, just do double-  to make it editable, and press the

`Enter` key to finish editing.

Label
: This is a color that you can set on the layer.  the layer to get a context menu to assign a color to it. You can then later filter on these colors.

Blending Mode
: This will set the Blending Modes of the layer.

Opacity
: This will set the opacity of the whole layer.

Visibility
: An eye-icon. Clicking this can hide a whole layer.

Edit State (Or layer Locking)
: A lock Icon. Clicking this will prevent the layer from being edited, useful when handling large amounts of layers.

Alpha Lock
: This will prevent the alpha of the layer being edited. In more plain terms: This will prevent the transparency of a layer being changed. Useful in coloring images.

Pass-through mode
: Only available on Group Layers, this allows you to have the blending modes of the layers within affect the layers outside the group. Doesn't work with masks currently, therefore these have a strike-through on group layers set to pass-through.

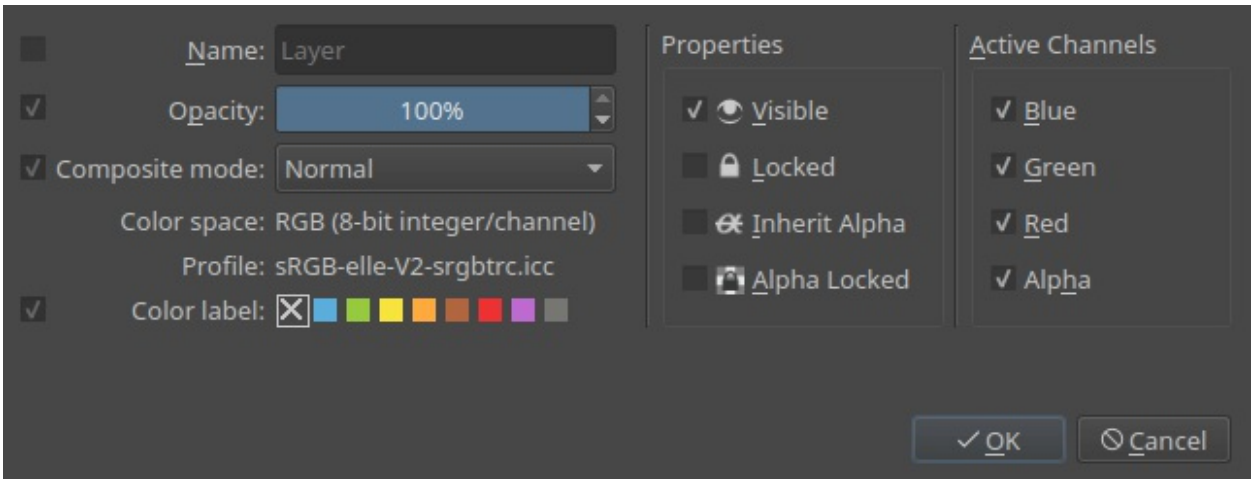
Alpha Inheritance
: This will use the alpha of all the peers of this layer as a transparency mask. For a full explanation see Introduction to Layers and Masks.

Open or Close Layers
: (An Arrow Icon) This will allow you to access sub-layers of a layer. Seen with masks and groups.

Onion Skin
: This is only available on animated layers, and toggles the onion skin feature.

Layer Style
: This is only available on layers which have a Layer Styles assigned. The button allows you to switch between on/off quickly.

To edit these properties on multiple layers at once, press the properties option when you have multiple layers selected or press the `F3` key. There, to change the names of all layers, the checkbox before *Name* should be ticked after which you can type in a name. Krita will automatically add a number behind the layer names. You can change other layer properties like visibility, opacity, lock states, etc. too.



Lower buttons

These are buttons for doing layer operations.

Add
: Will by default add a new Paint Layer, but using the little arrow, you can call a sub-menu with the other layer types.

Duplicate
: Will Duplicate the active layer(s). Can be quickly invoked with the `Ctrl`

`+`  `+ drag` shortcut.

Move layer up.
: Will move the active layer up. Will switch them out and in groups when coming across them.

Move layer down.
: Will move the active layer down. Will switch them out and in groups when coming across them.

Layer properties.
: Will open the layer properties window.

Delete
: Will delete the active layer(s). For safety reasons, you can only delete visible layers.

Hot keys and Sticky Keys

- `Shift` key for selecting multiple contiguous layers.
- `Ctrl` key for select or deselect layer without affecting other layers selection.
- `Ctrl +`  `+ drag` shortcut makes a duplicate of the selected layers, for you to drag and drop.
- `Ctrl + E` shortcut for merging a layer down. This also merges selected layers, layer styles and will keep selection masks intact. Using the `Ctrl + E` shortcut on a single layer with a mask will merge down the mask into the layer.
- `Ctrl + Shift + E` shortcut merges all layers.
- `R +`  shortcut allows you to select layers on canvas, similar to picking colors directly on canvas. Use the `Shift + R +`  shortcut for multiple layers.
- `Ins` key for adding a new layer.
- `Ctrl + G` shortcut will create a group layer. If multiple layers are selected, they are put into the group layer.

- `Ctrl + Shift + G` shortcut will quickly set-up a clipping group, with the selected layers added into the group, and a new layer added on top with alpha-inheritance turned on, ready for painting!
- `Ctrl + Alt + G` shortcut will ungroup layers inside a group.
- `Alt +`  shortcut for isolated view of a layer. This will maintain between layers till the same action is repeated again.
- `Page Up` and `Page Down` keys for switching between layers.
- `Ctrl + Page Up` and `Ctrl + Page Down` shortcuts will move the selected layers up and down.

Log Viewer

The log viewer docker allows you to see debug output without access to a terminal. This is useful when trying to get a tablet log or to figure out if Krita is spitting out errors while a certain thing is happening.

The log docker is used by pressing the *enable logging* button at the bottom.

Warning

When enabling logging, this output will not show up in the terminal. If you are missing debug output in the terminal, check that you didn't have the log docker enabled.

The docker is composed of a log area which shows the debug output, and four buttons at the bottom.

Log Output Area

The log output is formatted as follows:

White
: This is just a regular debug message.

Yellow
: This is a info output.

Orange
: This is a warning output.

Red
: This is a critical error. When this is bolded, it is a fatal error.

Options

There’s four buttons at the bottom:

Enable Logging
: Enable the docker to start logging. This caries over between sessions.

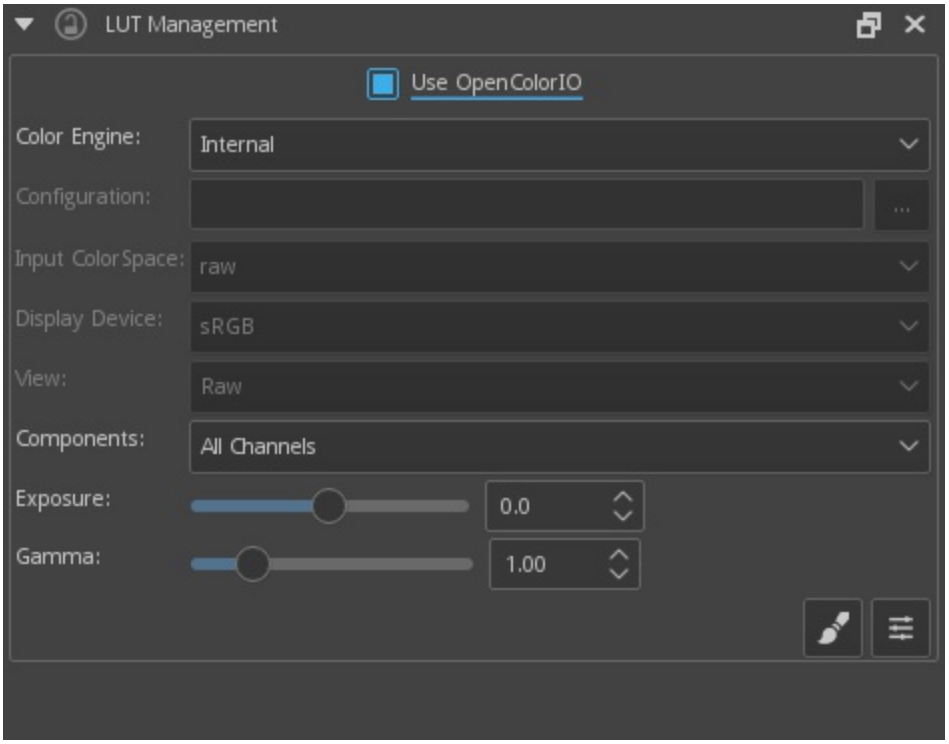
Clear the Log
: This empties the log output area.

Save the Log
: Save the log to a text file.

Configure Logging
: Configure which kind of debug is added. By default only warnings and simple debug statements are logged. You can enable the special debug messages for each area here.

    - General
    - Resource Management
    - Image Core
    - Registries
    - Tools
    - Tile Engine
    - Filters
    - Plugin Management
    - User Interface
    - File Loading and Saving
    - Mathematics and Calculations
    - Image Rendering
    - Scripting
    - Input Handling
    - Actions
    - Tablet Handing
    - GPU Canvas
    - Metadata
    - Color Management

LUT Management



The Look Up Table (LUT) Management docker controls the high dynamic range (HDR) painting functionality.

Use OpenColorIO
: Use Open Color IO instead of Krita's internal color management. Open Color IO is a color management library. It is sometimes referred to as OCIO. This is required as Krita uses OCIO for its HDR functionality.

Color Engine
: Choose the engine.

Configuration
: Use an OCIO configuration file from your computer.

Note

Some system locals don’t allow you to read the configuration files. This is due to a bug in OCIO. If you are using Linux you can fix this. If you start Krita from the terminal with the `LC_ALL=C krita` flag set, you should be able to read the configuration files.

Input Color Space
: What the color space of the image is. Usually sRGB or Linear.

Display Device
: The type of device you are using to view the colors. Typically sRGB for computer screens.

View
: –

Components
: Allows you to study a single channel of your image with LUT.

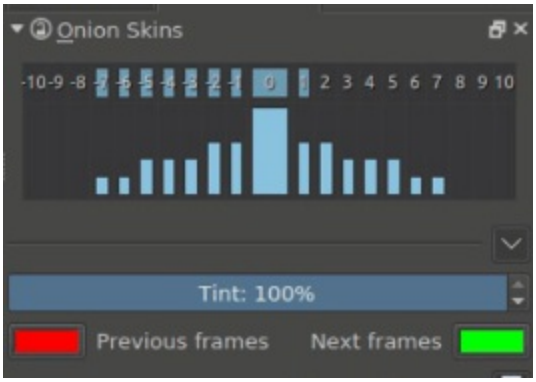
Exposure
: Set the general exposure. On 0.0 at default. There’s the `Y` key to change this on the fly on canvas.

Gamma
: Allows you to set the gamma. This is 1.0 by default. You can set this to change on the fly in canvas shortcuts.

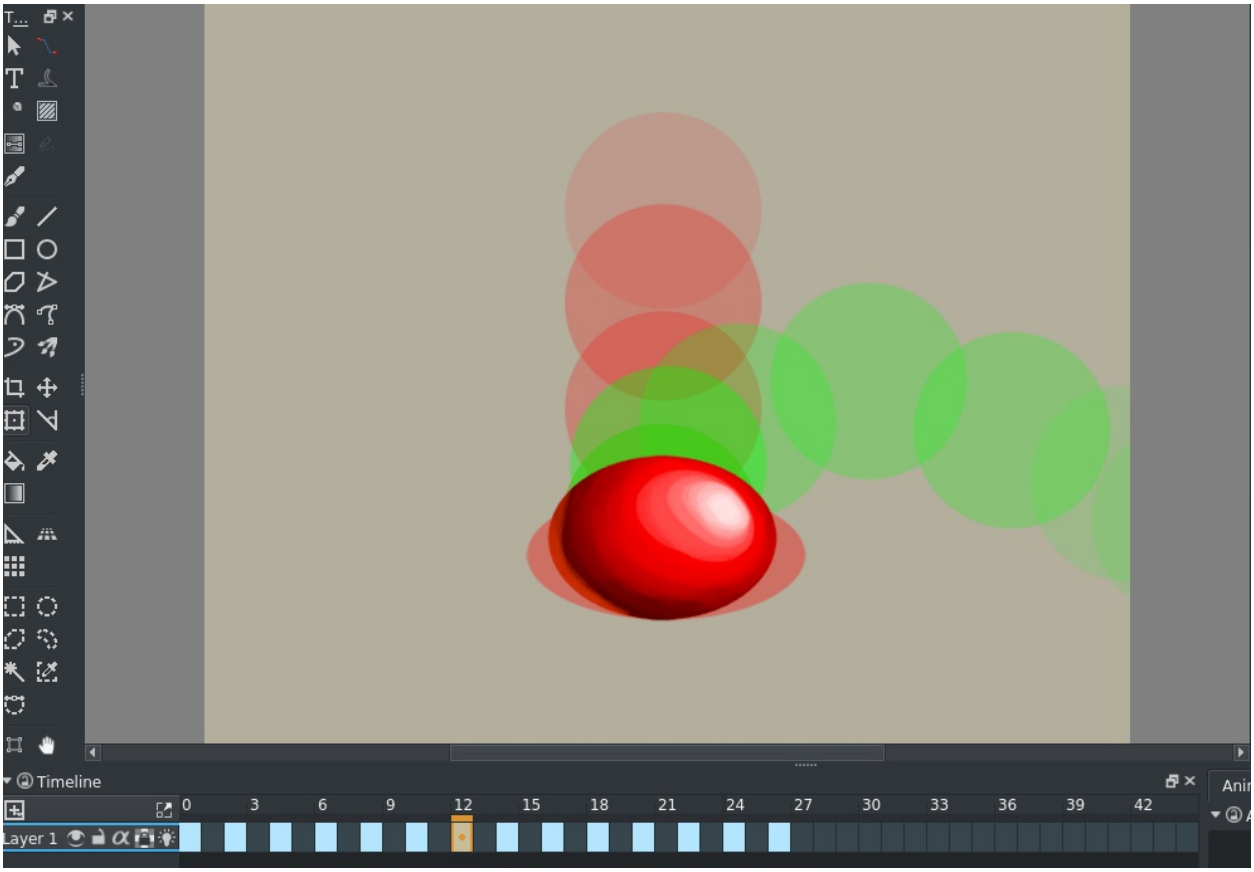
Lock color
: Locks the color to make sure it doesn’t shift when changing exposure. May not be desired.

Set white and black points
: This allows you to set the maximum and minimum brightness of the image, which’ll adjust the exposure and gamma automatically to this.

Onion Skin Docker



To make animation easier, it helps to see both the next frame as well as the previous frame sort of layered on top of the current. This is called *onion-skinning*.



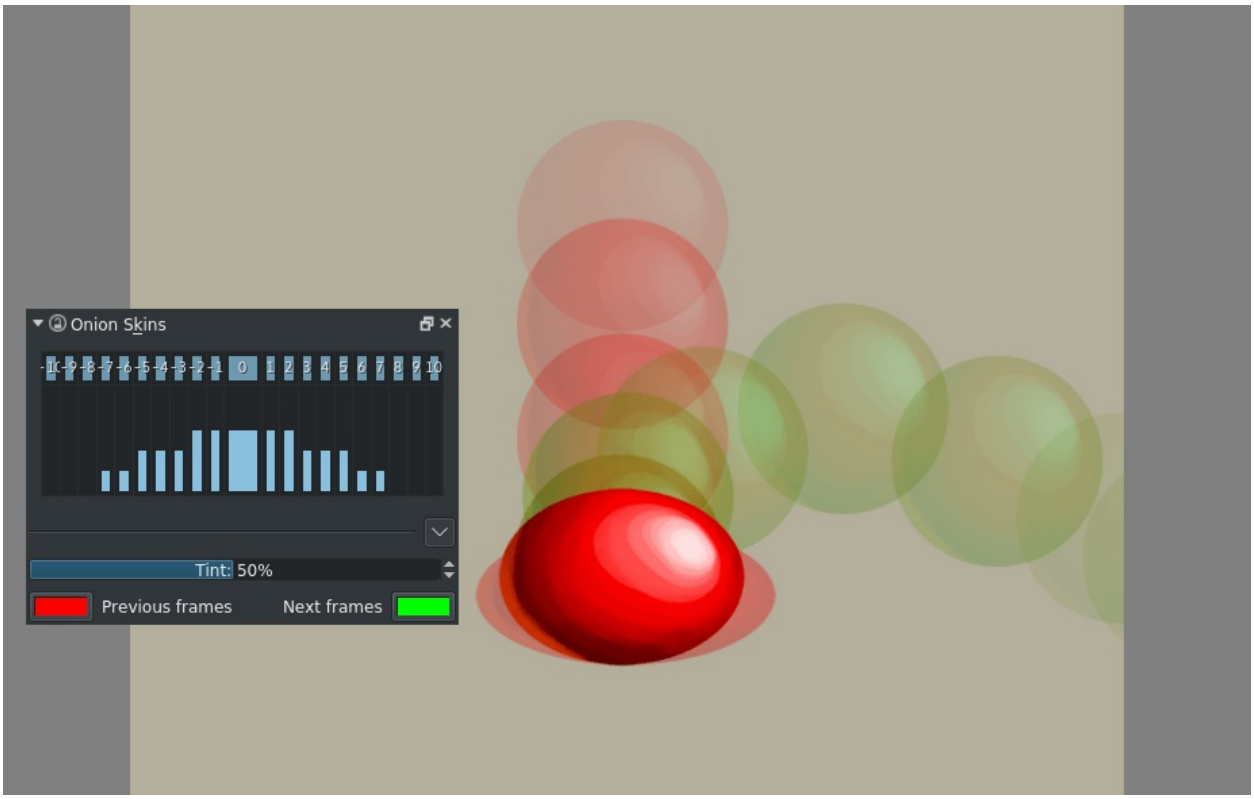
Basically, they are images that represent the frames before and after the current frame, usually colored or tinted.

You can toggle them by clicking the lightbulb icon on a layer that is animated (so, has frames), and isn't fully opaque. (Krita will consider white to be white, not transparent, so don't animated on an opaque layer if you want onion skins.)

Changed in version 4.2: Since 4.2 onion skins are disabled on layers whose default pixel is fully opaque. These layers can currently only be created by using *background as raster layer* in the *content* section of the new image dialog. Just don't try to animate on a layer like this if you rely on onion skins, instead make a new one.

The term onionskin comes from the fact that onions are semi-transparent. In traditional animation animators would make their initial animations on semitransparent paper on top of an light-table (of the special animators variety), and they'd start with so called keyframes, and then draw frames in between. For that, they would place said keyframes below the frame they were working on, and the light table would make the lines of the keyframes shine through, so they could reference them.

Onion-skinning is a digital implementation of such a workflow, and it's very useful when trying to animate.

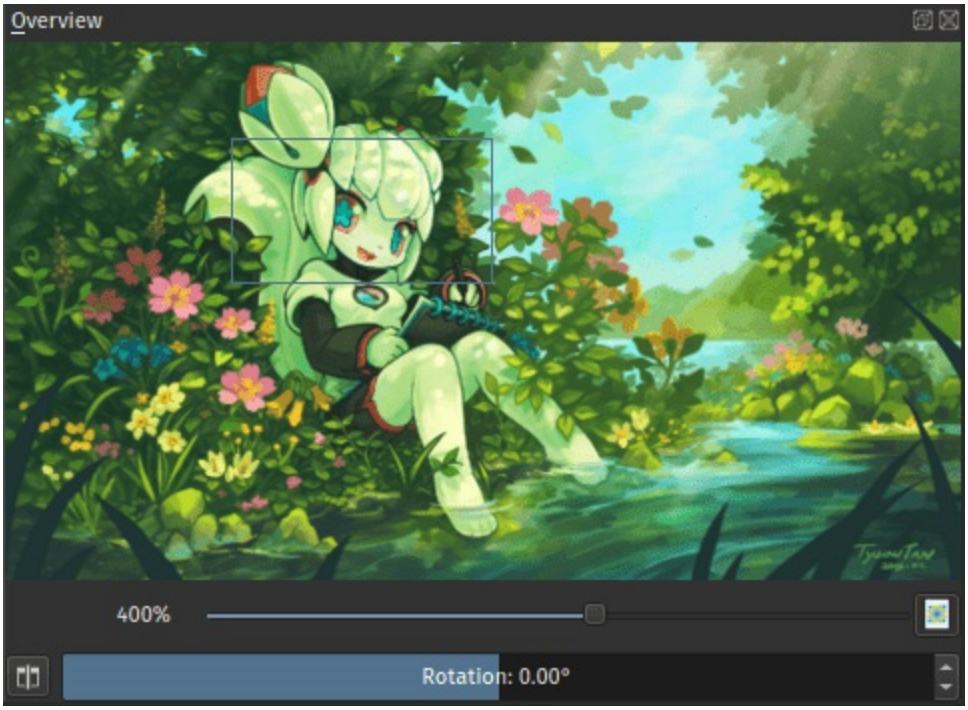


The slider and the button with zero offset control the master opacity and visibility of all the onion skins. The boxes at the top allow you to toggle them on and off quickly, the main slider in the middle is a sort of ‘master transparency’ while the sliders to the side allow you to control the transparency per keyframe offset.

Tint controls how strongly the frames are tinted, the first screen has 100%, which creates a silhouette, while below you can still see a bit of the original colors at 50%.

The *Previous Frame* and *Next Frame* color labels allows you set the colors.

Overview



This docker allows you to see a full overview of your image. You can also use it to navigate and zoom in and out quickly. Dragging the view-rectangle allows you quickly move the view.

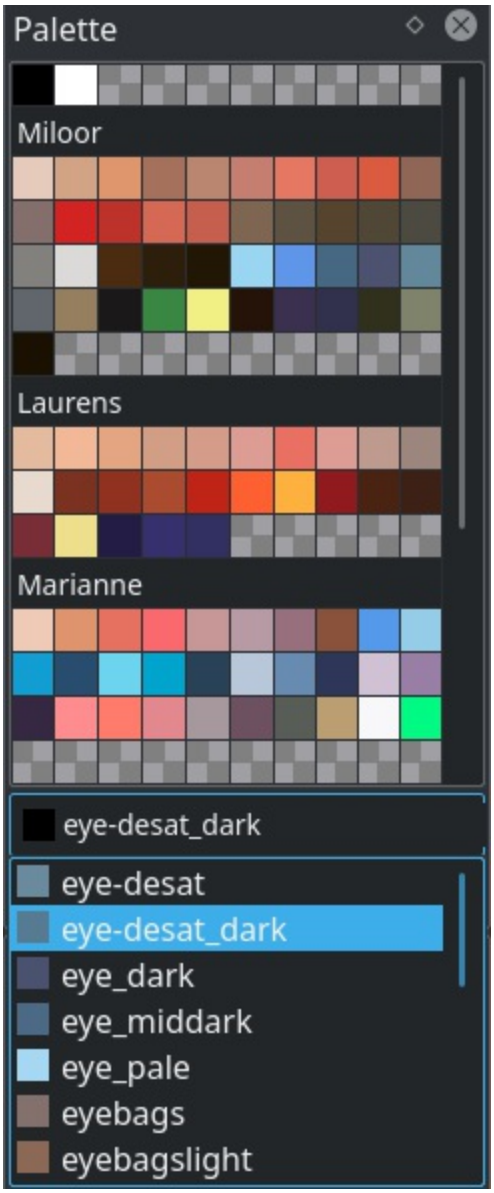
There are furthermore basic navigation functions: Dragging the zoom-slider allows you quickly change the zoom.

New in version 4.2: Toggling the mirror button will allow you to mirror the view of the canvas (but not the full image itself) and dragging the rotate slider allows you to adjust the rotation of the viewport. To reset the rotation,  the slider to edit the number, and type ‘0’.

Palette Docker

The palette docker displays various color swatches for quick use. It also supports editing palettes and organizing colors into groups, as well as arbitrary positioning of swatches.

New in version 4.2: The palette docker was overhauled in 4.2, allowing for grid ordering, storing palette in the document and more.



You can choose from various default palettes or you can add your own colors to the palette.

To choose from the default palettes click on the icon in the bottom left corner of the docker, it will show a list of pre-loaded color palettes. You can click on one and to load it into the docker, or click on import resources to load your own color palette from a file. Creating a new palette can be done by pressing the +. Fill out the *name* input, pressing *Save* and Krita will select your new palette for you.

Since 4.2 Krita's color palettes are not just a list of colors to store, but also a grid to organize them on. That's why you will get a grid with 'transparency checkers', indicating that there is no entry. To add an entry, just click a swatch and a new entry will be added with a default name and the current foreground color.

- Selecting colors is done by  on a swatch.
- Pressing the delete icon will remove the selected swatch or group. When removing a group, Krita will always ask whether you'd like to keep the swatches. If so, the group will be merged with the default group.
- Double  a swatch will call up the edit window where you can change the color, the name, the id and whether it's a spot color. On a group this will allow you to set the group name.
-  drag will allow you to drag and drop swatches and groups to order them.
-  on a swatch will give you a context menu with modify and delete options.
- Pressing the + icon will allow you to add a new swatch.
- The drop down contains all the entries, id numbers and names. When a color is a spot color the thumbnail is circular. You can use the dropdown to search on color name or id.

Pressing the Folder icon will allow you to modify the palette. Here you can add more columns, modify the default group's rows, or add more groups and modify their rows.

Palette Name
: Modify the palette name. This is the proper name for the palette as shown in the palette chooser dropdown.

File name
: This is the file name of the palette, which should be file system friendly. (Avoid quotation marks, for example).

Column Count
: The amount of columns in this palette. This counts for all entries. If you accidentally make it smaller than the amount of entries that take up columns, you can still make it bigger until the next restart of Krita.

Where is the palette stored:
: Whether to store said palette in the document or resource folder.

  Resource Folder
  : The default, the palette will be stored in the resource folder.

  Document
  : The palette will be removed from the resource folder and stored in the document upon save. It will be loaded into the resources upon loading the document.

Add group
: Add a new group. On clicking you will be asked for a name and a set of rows.

Group Settings
: Here you can configure the groups. The dropdown has a selection of groups. The default group is at top.

  Row Count
  : The amount of rows in the group. If you want to add more colors to a group and there’s no empty areas to click on anymore, increase the row count.

Rename Group
: Rename the group.

Delete Group
: Delete the group. It will ask whether you want to keep the colors. If so, it will merge the group's contents with the default group.

The edit and new color dialogs ask for the following:

Color
: The color of the swatch.

Name
: The Name of the color in a human readable format.

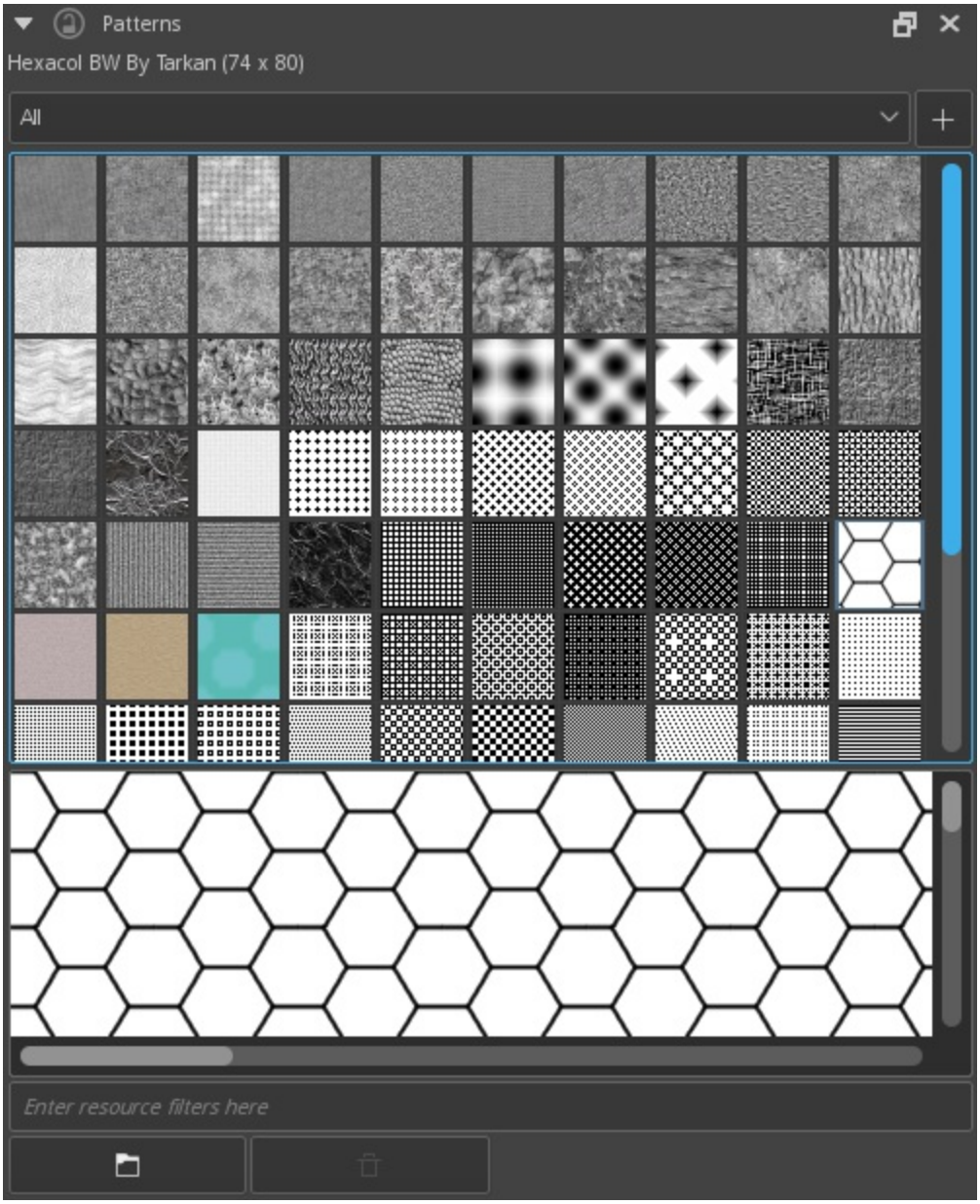
ID
: The ID is a number that can be used to index colors. Where Name can be something like "Pastel Peach", ID will probably be something like "RY75". Both names and ids can be used to search the color in the color entry dropdown at the bottom of the palette.

Spot color
: Currently not used for anything within Krita itself, but spot colors are a toggle to keep track of colors that represent a real world paint that a printer can match. Keeping track of such colors is useful in a printing workflow, and it can also be used with python to recognize spot colors.

Krita's native palette format is since 4.0 *.kpl. It also supports importing…

- Gimp Palettes (.gpl)
- Microsoft RIFF palette (.riff)
- Photoshop Binary Palettes (.act)
- PaintShop Pro palettes (.psp)
- Photoshop Swatches (.aco)
- Scribus XML (.xml)
- Swatchbooker (.sbz).

Patterns Docker



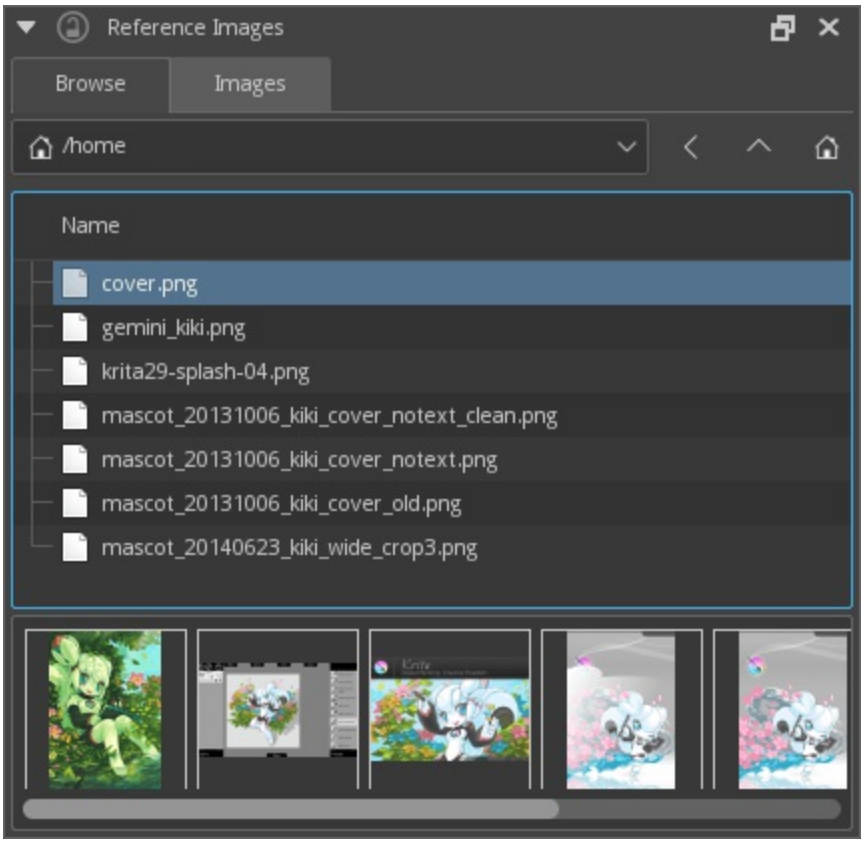
This docker allows you to select the global pattern. Using the open-file button you can import patterns. Some common shortcuts are the following:

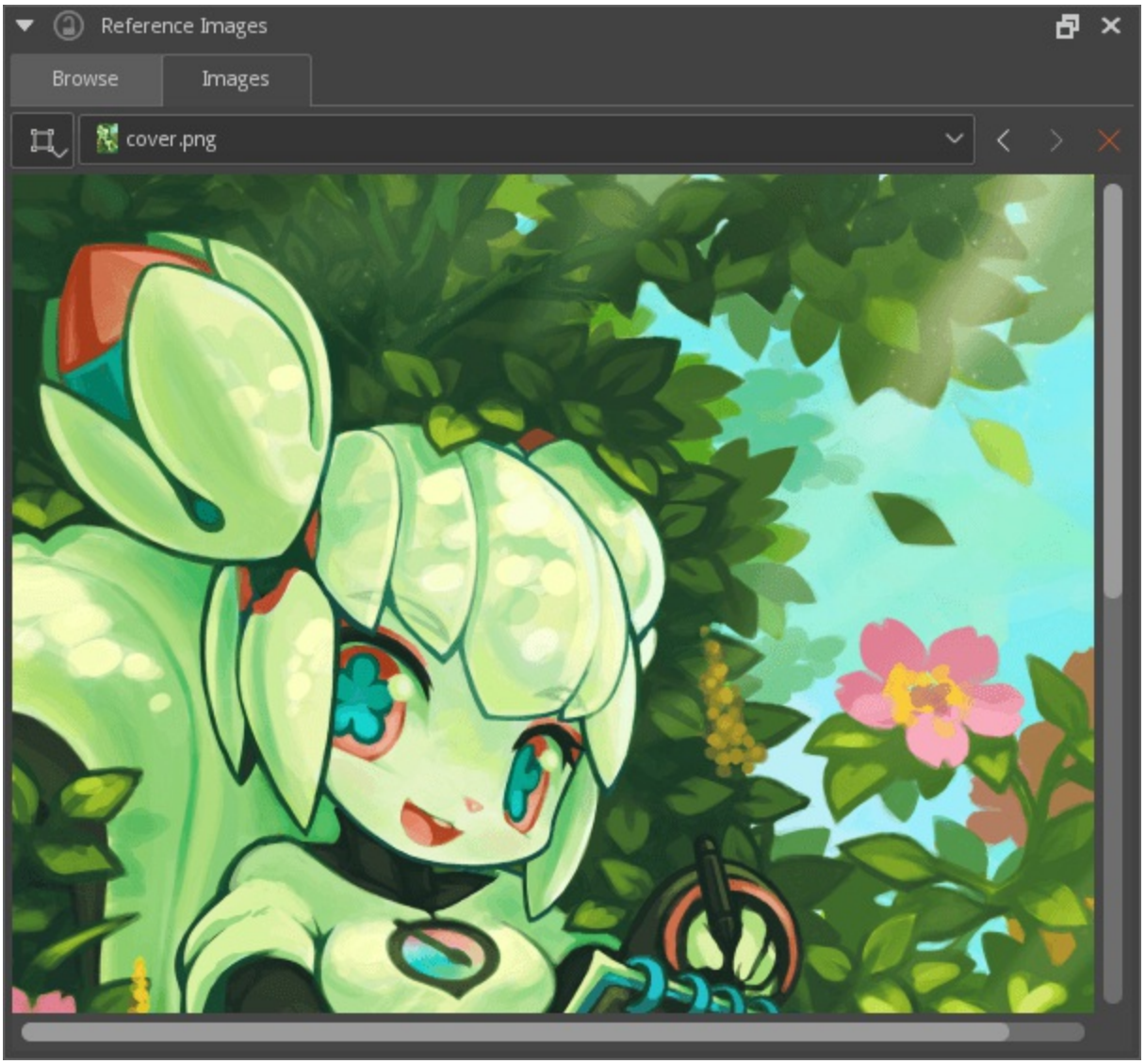
-  a swatch will allow you to set tags.

-  a swatch will allow you to set it as global pattern.
- `Ctrl + scroll` you can resize the swatch sizes.

Reference Images Docker

Deprecated since version 4.0: This docker was removed in Krita 4.0 due to crashes on Windows. The reference images tool in 4.1 replaces it..





This docker allows you to pick an image from outside of Krita and use it as a reference. Even better, you can pick colors from it directly.

The docker consists of two tabs: Browsing and Image.

Browsing

Browsing gives you a small file browser, so you can navigate to the map where the image you want to use as reference is located.

There’s an image strip beneath the browser, allowing you to select the image which you want to use. Double click to load it in the *Image* tab.

Image

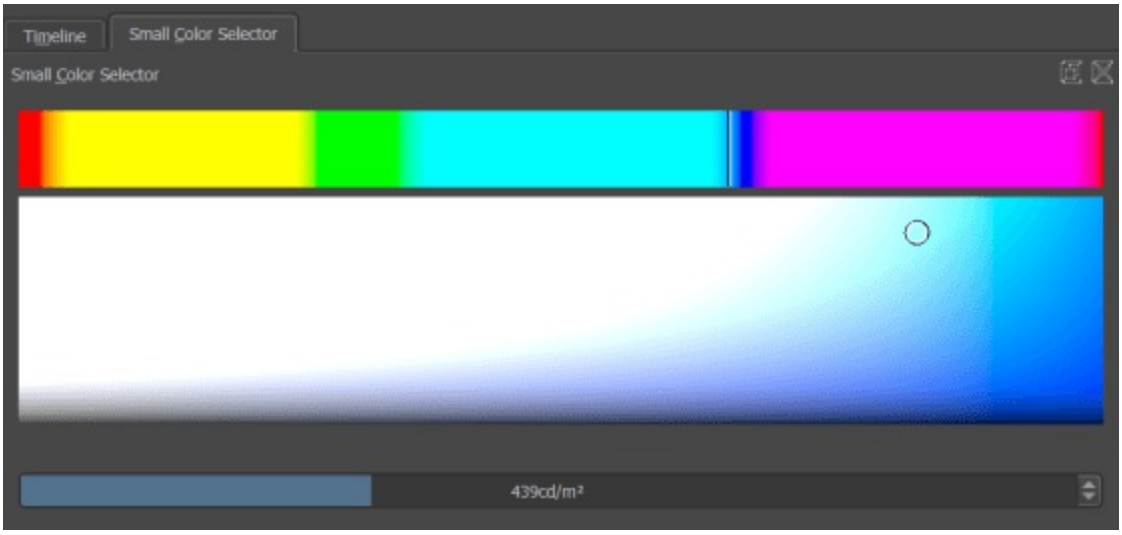
This tab allows you to see the images you selected, and change the zoom level. Clicking anywhere on the image will allow you to pick the merged color from it. Using the cross symbol, you can remove the icon.

Shape Properties Docker

Deprecated since version 4.0: This docker is deprecated, and its functionality is folded into the Shape Edit Tool.

This docker is only functional when selecting a rectangle or circle on a vector layer. It allows you to change minor details, such as the rounding of the corners of a rectangle, or the angle of the formula for the circle-shape.

Small Color Selector

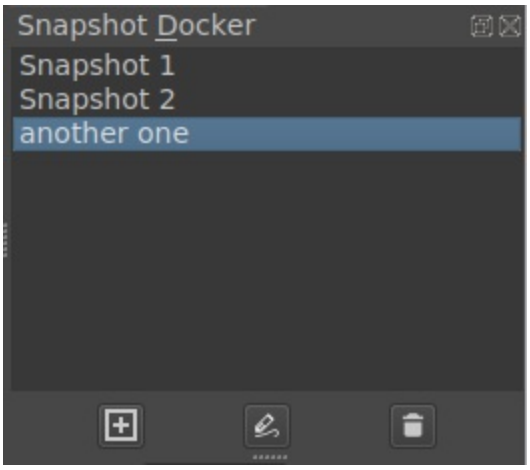


This is Krita's most simple color selector. On the left there's a bar with the hue, and on the right a square where you can pick the value and saturation.

New in version 4.2: The small color selector is the only selector which can show HDR values. When your build of Krita is HDR enabled and you are on Windows, you can drag the slider at the bottom to increase the 'nits' of the colors in the small selector. This is the direct value of the brightness of the colors, and you need a value above 100 (100 being the maximum value used for the brightest value of sRGB colors), to have an HDR color. The small color selector will also select wide gamut values.

Snapshot Docker

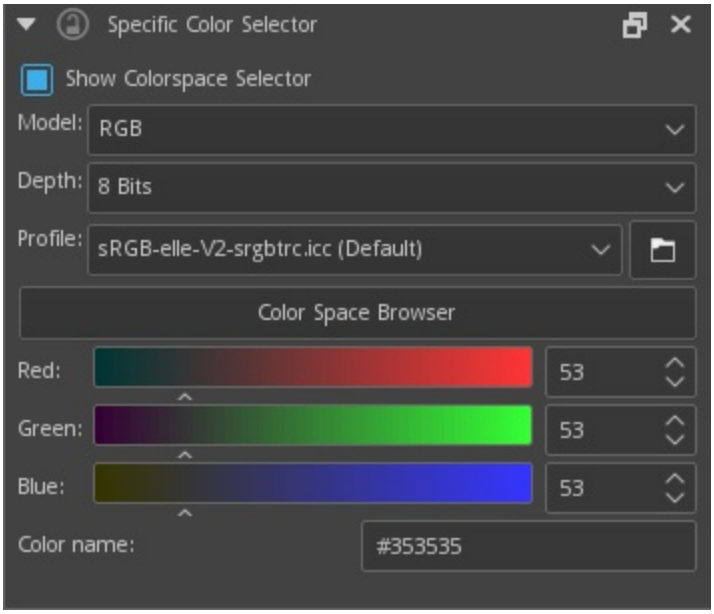
A docker that allows you to create snapshots (copies) of the current document, and to return to these states afterwards.



The main part of the docker is a list of all saved snapshots. At the bottom of the docker, there are three buttons: from left to right, they are *Create snapshot*, *Switch to selected snapshot*, and *Remove selected snapshot*. You can create a snapshot from the current state of the document by clicking *Create snapshot*. Click *Switch to selected snapshot* to switch to the selected snapshot. The undo stack will be discarded after switching. If you would like to save the current state, make another snapshot before switching. Click *Remove selected snapshot* to delete the selected snapshot. You can edit the names of snapshots by double-clicking them.

Please be aware that all snapshots will be gone if you close the document. If you want to keep them, you need to explicitly save or export them.

Specific Color Selector



The specific color selector allows you to choose specific colors within a color space.

Color Space Chooser

Fairly straightforward. This color space chooser allows you to pick the color space, the bit depth and the ICC profile in which you are going to pick your color. Use the checkbox ‘show color space selector’ to hide this feature.

Sliders

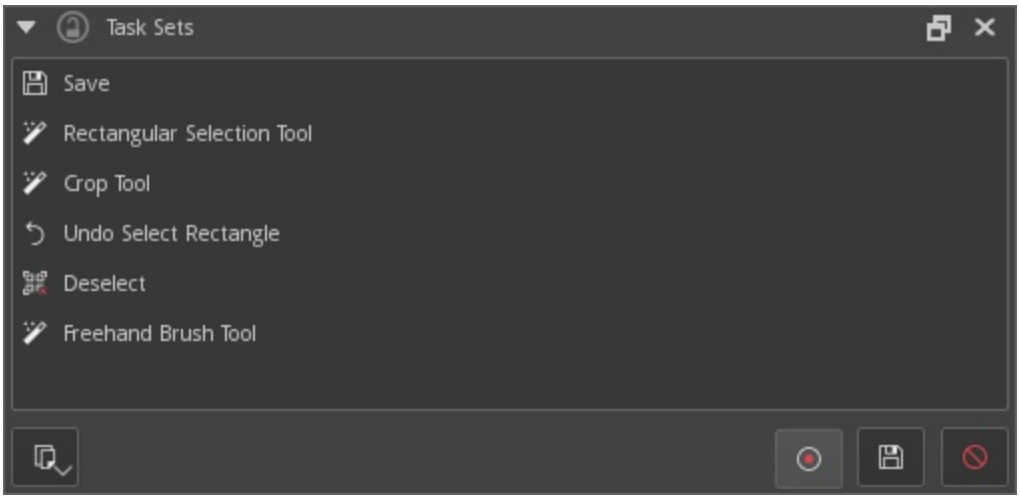
These change per color space. If you chose 16bit float or 32 bit float, these will go from 0 to 1.0, with the decimals deciding the difference between colors.

Hex Color Selector

This is only available for the color spaces with a depth of 8 bit. This allows you to input hex color codes, and receive the RGB, CMYK, LAB, XYZ or YCrCb equivalent, and the other way around!

Task Sets Docker

Task sets are for sharing a set of steps, like a tutorial. You make them with the task-set docker.

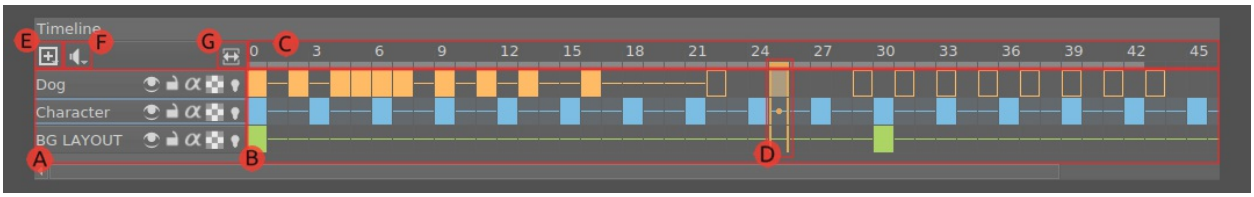


Task sets can record any kind of command also available via the shortcut manager. It can not record strokes, like the macro recorder can. However, you can play macros with the tasksets!

The tasksets docker has a record button, and you can use this to record a certain workflow. Then use this to let items appear in the taskset list. Afterwards, turn off the record. You can then click any action in the list to make them happen. Press the ‘Save’ icon to name and save the taskset.

Timeline Docker

The **Timeline Docker** works in tandem with the Animation Docker at the heart of **Krita**'s animation tools. While the Animation Docker provides access to the fundamental controls for playing back and editing animations, the Timeline Docker contains the layered frames and specific timings that define your animation. In other words, the Timeline Docker is the digital equivalent to a traditional animator's "dope sheet".



Legend:

A. Layer List – This area contains some subset of the layers of your current document. Similar to the Layers Docker, each layer has various properties that can also be toggled here (visibility, locking, onion skins, etc.). While the currently active layer is always shown here, layers can also be "pinned" to the timeline using the *Pin to Timeline* menu action or the *Pin Existing Layer* submenu so they will be visible even when inactive. Layers that are created via the Timeline are pinned automatically and by checking the *Automatically pin new layers to timeline* option in *Settings –> Configure Krita… –> General –> Miscellaneous* all new paint layers can be pinned automatically.

- **Active Layer**
  A highlighted row in the table shows the current active layer. One can change which layer is active by clicking on the layer's name within the left header. It is *not* possible to change the active layer by clicking inside the table in order to not disturb the user when scrubbing and editing frame positions on the timeline.

B. Frame Table – The Frame Table is a large grid of cells which can either hold a single frame or be empty. Each row of the Frame Table represents an *animation layer* and each column represents a *frame timing*. Just like the Layer List, the active layer is highlighted across the entire Frame Table. It's important to understand that frame timings are not based on units of time like seconds, but are based on frames which can then be played back at any speed, depending on the Animation Docker's *frame rate* and *play speed* settings.

Frames can be moved around the timeline by simply left-clicking and dragging from one frame to another slot, even across layers. Furthermore, holding the `Ctrl` key while moving creates a copy. Right-clicking anywhere in the Frame Table will bring up a helpful context menu for adding, removing, copying, and pasting frames or adjusting timing with holds.

- **Current Selection:**

  Frames highlighted in orange represent a selection or multiple selections, which can be created by mouse or keyboard. While multiple frames are selected, right-clicking anywhere in the Frame Table will bring up a context menu that will allow for adding or removing frames or holds within the current selection. Finally, it is also possible to have multiple non-contiguous/separate selections if needed.

  Warning

  Painting always happens only in the *active frame* (represented by a small dot), which is not necessarily part of your current selection!

- **Keys, Blanks, and Holds:**

  The Timeline Docker now shows us even more useful information about both what is there as well as what is not. **Key frames** which contain drawings are still displayed as *filled blocks* within a cell, while **blank** or empty key frames are shown as a *hollow outline*. In Krita, every drawn frame is automatically held until the next frame; these **holds** are now

clearly shown with a *colored line* across all held frames. The color of frames can be set per-frame by the animator using the right-click menu, and is a matter of personal workflow.

C. Frame Timing Header – The Frame Timing Header is a ruler at the top of the Frame Table. This header is divided into small notched sections which are based on the current *frame rate* (set in the Animation Docker). Integer multiples of the frame rate have a subtle double-line mark, while smaller subdivisions have small single-line marks. Each major notch is marked with a helpful *frame number*.

- **Cached Frames:**

  The Frame Timing Header also shows important information about which frames are currently *cached*. When something is said to be "cached", that means that it is stored in your device's working memory (RAM) for extra fast access. Cached frames are shown by the header with a small light-gray rectangle in each column. While this information isn't always critical for us artists, it's helpful to know that Krita is working behind the curtains to cache our animation frames for the smoothest possible experience when scrubbing through or playing back your animation.

D. Current Time Scrubber – A highlighted column in the Frame Table which controls the current frame time and, as such, what is currently displayed in the viewport.

- **Active Frame:**

  A frame of the *active layer* at the *current time* position. The active frame is always marked with a small circle inside. All drawing, painting, and image editing operations happen on this frame only!

  Warning

  Don't mix the active frame up with the current selection!

E. Layer Menu – A small menu for manipulating animation layers. You can create new layers, remove existing ones, as well as pin or unpin the active layer. (This menu also shows up when right-clicking on layer headers inside of the Layer List.)

F. Audio Menu: Another small menu for animating along with audio sources. This is where you can open or close audio sources and control output volume/muting.

G. Zoom Handle: This allows you to zoom in and out on the Frame Table, centered around the current frame time. Click-dragging starting on the zoom handle controls the zoom level.

Usage:

How to use the Timeline Docker is not immediately obvious because **Krita** doesn't automatically create a key frame out of your initial drawing. In fact, *until you make a key frame on a layer*, Krita assumes that there's no animation going on at all on that layer and it will keep the image static over the whole animation.

So, to make our first *animated layer*, we need to make a key frame!

 any square on the timeline docker and select *Create Blank Frame*. A blank frame (one that you haven't yet drawn anything in) appears as a *hollow outline* instead of a solid box, making that frame active and drawing on the canvas will make it appear as a *solid, colored rectangle*. To move a frame around, you can drag and drop it into another empty frame slot.

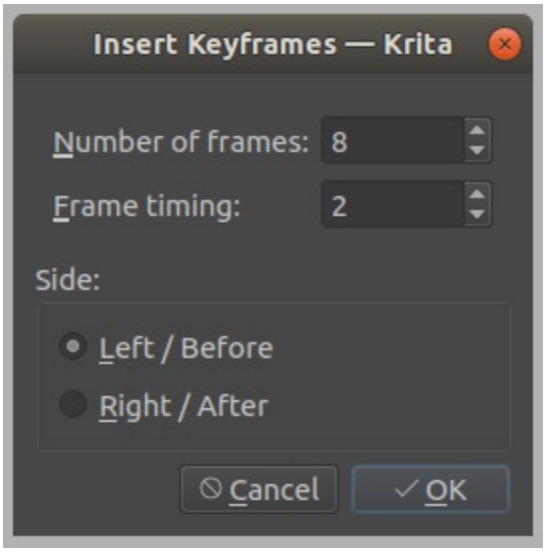
While animating you may find that you want to keep a layer "pinned", making it visible in the Timeline Docker regardless of which layer is selected. There are a few ways to do this in Krita. By  clicking on any layer in the Layers Docker, you can pin it by activating the *Pin to Timeline* menu item. This allows you to decide which layers are important to see in your timeline (i.e. those which contain keyframe drawings) and which layers are not (i.e. a static layout drawing or background painting). Alternatively,

you can open the Timeline Docker's Layer Menu and select a layer from the *Pin Existing Layer* submenu. Finally, you can enable the *Automatically pin new layers to timeline* option from the *Settings –> Configure Krita… –> General –> Miscellaneous* dialog.

To add a new frame, either right-click on an empty frame slot and select *Create Blank Frame* to create a fresh blank frame, or select *Create Duplicate Frame* to create a new copy of the previous frame. It's also possible to add multiple key frames by right-clicking inside the Frame Table and selecting *Keyframes ▸ Insert Multiple Keyframes*. With this option you can specify a number of frames to add with the option of built in timing for quickly creating a series of 1s, 2s, 3s, etc. These settings are saved between uses.

You can also change the color of frames so that you can easily identify important frames or distinguish between different sections of your animation. The current color selection is remembered for new frames so that you can easily make a set of colored frames and then switch to another color.

Clicking with  within the Frame Timing Header instead of the Frame Table gives you access to a few more options which allow you to add or remove entire columns of frames or holds at a time. For example, selecting *Keyframe Columns ▸ Insert Keyframe Column Left* will add new frames to each layer that's currently visible in the Timeline Docker.



Krita only tracks key frame changes. This is unlike **Flash** where you have to manually indicate how long a key frame will hold. Instead, **Krita** just assumes that the space between key frame 1 and key frame 2 is supposed to be filled with key frame 1. Frames that are held in this way (a.k.a. “holds”) are displayed as a continuous line in the Frame Table.

To delete frames,  the frame and press *Remove Keyframe*. This will delete all selected frames. Similarly, selecting *Remove Frame and Pull* will delete the selected frames and pull or shift all subsequent frames back/left as much as possible.

To manually play your animation back and forward using your mouse, an important technique that’s known as *scrubbing*, click-drag within the Frame Timing Header.

GUI Actions:

1. **Layer List**
    - : Select active layer.
    - : Layers Menu (add/remove/show layers, etc.).
2. **Frame Timing Header**
    - : Move to time and select frame of the active layer.
    - `+ drag` : Scrub through time and select frame of the active layer.
    - : Frame Columns Menu (insert/remove/copy/paste columns and hold columns).
3. **Frames Table: all**
    - : Selects a single frame or slot and switches time, but *does*

not switch active layer.

- `Space +`  : Pan.
- `Space +`  : Zoom.

4. **Frames Table (On Empty Slot).**

    - : Frames menu (insert/copy/paste frames and insert/remove holds).
    - `+ drag` : Select multiple frames and switch time to the last selected, but *does not switch active layer*.
    - `Shift +`  : Select all frames between the active and the clicked frame.
    - `Ctrl +`  : Select individual frames together. `click + drag` them into place.

5. **Frames Table (On Existing Frame)**

    - : Frames menu (remove/copy/paste frames and insert/remove holds).
    - `+ drag` : *Move* a frame or multiple frames.
    - `Ctrl +`  `+ drag` : Copy a frame or multiple frames.
    - `Alt + drag` : Move selected frame(s) and *all* the frames to the right of it. (This is useful for when you need to clear up some space in your animation, but don't want to select all the frames to the right of a particular frame!)

Touch Docker

The Touch Docker is a QML docker with several convenient actions on it. Its purpose is to aid those who use Krita on a touch-enabled screen by providing bigger gui elements.

Its actions are…

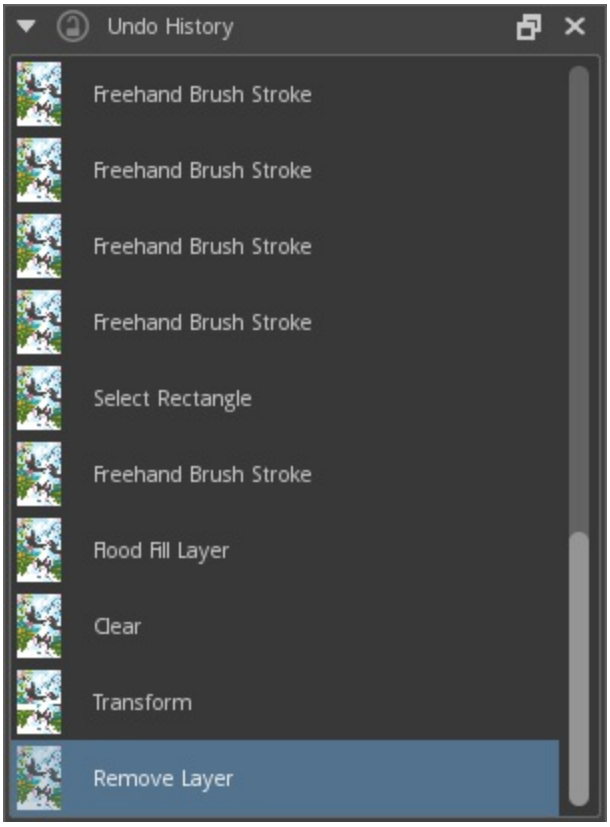
Open File	Save File	Save As…
Undo	Redo	
Decrease Opacity		
Increase Opacity		
Increase Lightness		
Decrease Lightness		
	Zoom in	
Rotate Counter Clockwise 15°	Reset Canvas Rotation	Rotate Clockwise 15°
	Zoom out	

Decrease Brush Size

Increase Brush Size

Delete Layer Contents

Undo History



This docker allows you to quickly shift between undo states, and even go back in time far more quickly than rapidly reusing the `Ctrl + Z` shortcut.

Cumulative Undo

 an item in the undo-history docker to enable cumulative undo.  again to change the parameters:

Start merging time
: The amount of seconds required to consider a group of strokes to be worth one undo step.

Group time

According to this parameter – groups are made. Every stroke is put into the same group until two consecutive strokes have a time gap of more than T seconds. Then a new group is started.

Split strokes.

A user may want to keep the ability of Undoing/Redoing his/her last N strokes. Once N is crossed – the earlier strokes are merged into the group's first stroke.

Vector Library

The Vector Library Docker loads the symbol libraries in SVG files, when those SVG files are put into the “symbols” folder in the resource folder *Settings* ▸ *Manage Resources…* ▸ *Open Resource Folder*.

The vector symbols can then be dragged and dropped onto the canvas, allowing you to quickly use complicated images.

Currently, you cannot make symbol libraries with Krita yet, but you can make them by hand, as well as use Inkscape to make them. Thankfully, there’s quite a few svg symbol libraries out there already!

Dr. MinGW Debugger

Note

The information on this page applies only to the Windows release of Krita. Usually, the %LOCALAPPDATA%kritacrash.log log file will contain enough information for the developers to get an idea of why Krita crashed. Using the debug package is only needed when more precise information is needed.

Using the Debug Package

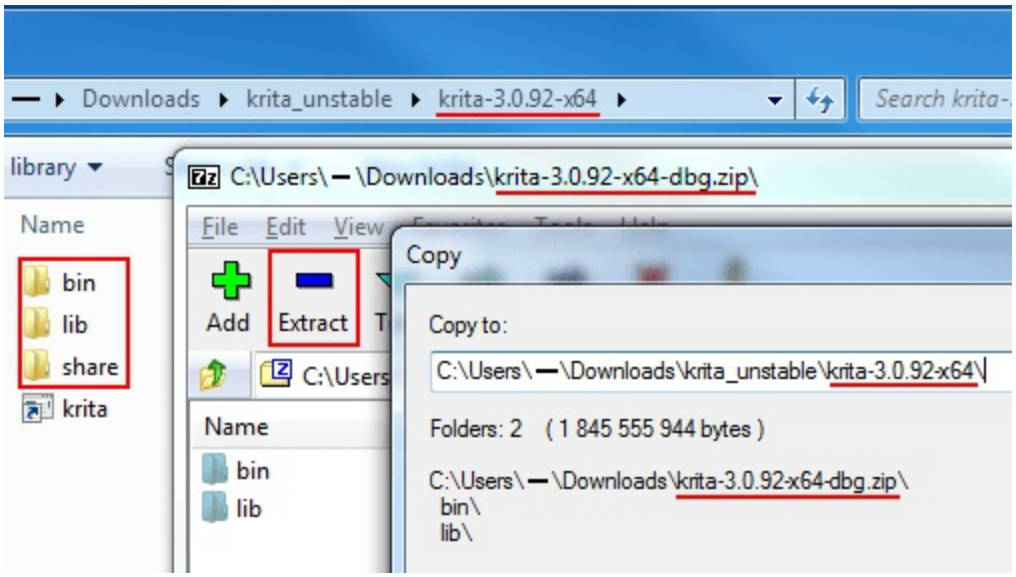
If you have your Krita version installed and you want to get a backtrace, it's best to download a portable version (either the latest release, or the one that someone told you to try, for example Krita Next or Krita Plus package). Alongside downloading the portable version, download the debug symbols package, too. It should be located in the same place you download Krita. You can see which is which by checking the end of the name of the ZIP file - debug symbols package always ends with *-dbg.zip*.

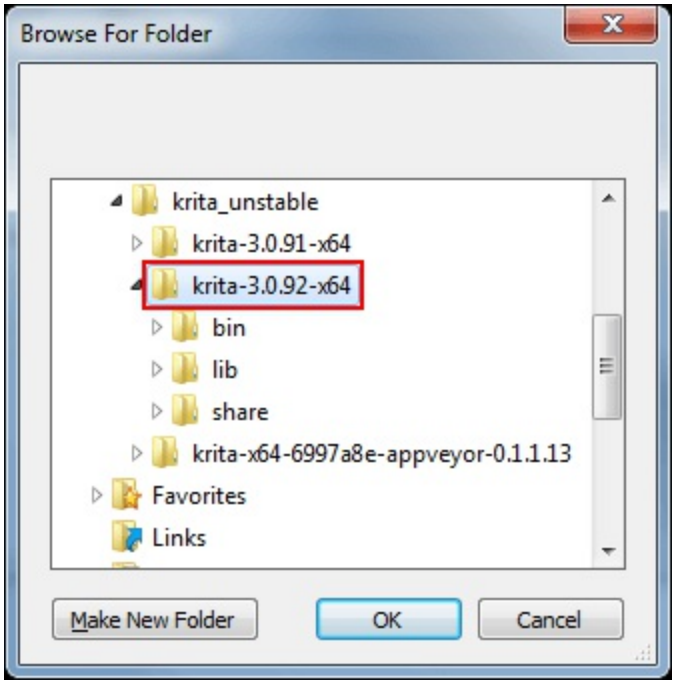
- Links to the debug packages should be available on the release announcement news item on https://krita.org/, along with the release packages. You can find debug packages for any release either in https://download.kde.org/stable/krita for stable releases or in https://download.kde.org/unstable/krita for unstable releases (for example beta versions). Portable ZIP and debug ZIP are found next to each other.

- Make sure you've downloaded the same version of debug package for the portable package you intend to debug / get a better backtrace.

- Extract the portable Krita.

- Extract the files from the debug symbols package inside the portable Krita main directory, where the sub-directories *bin*, *lib* and *share* is located, like in the figures below:

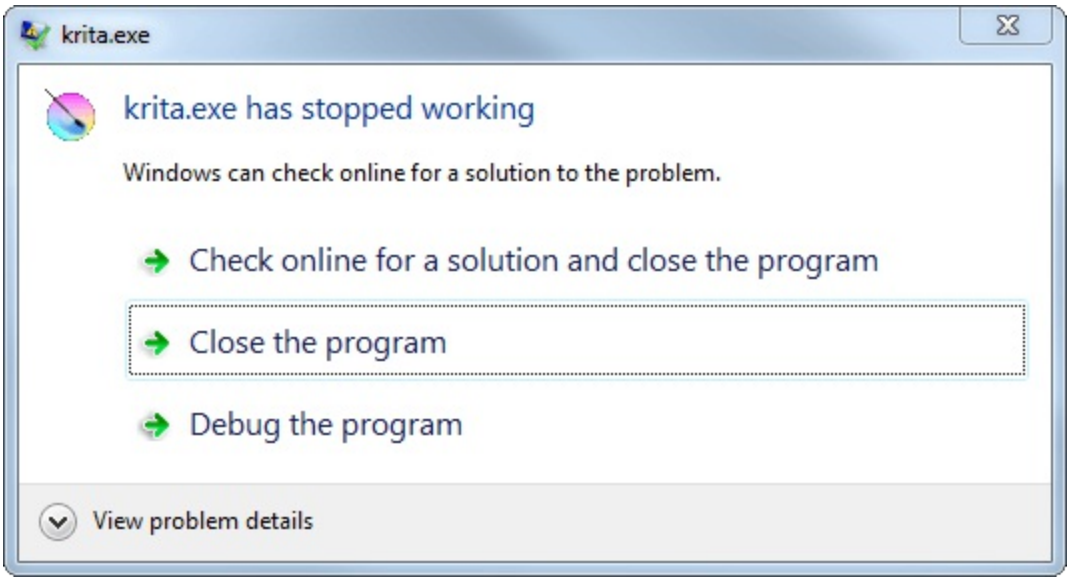




- After extracting the files, check the `bin` dir and make sure you see the `.debug` dir inside. If you don't see it, you probably extracted to the wrong place.

Getting a Backtrace

1. When there is a crash, Krita might appear to be unresponsive for a short time, ranging from a few seconds to a few minutes, before the crash dialog appears.



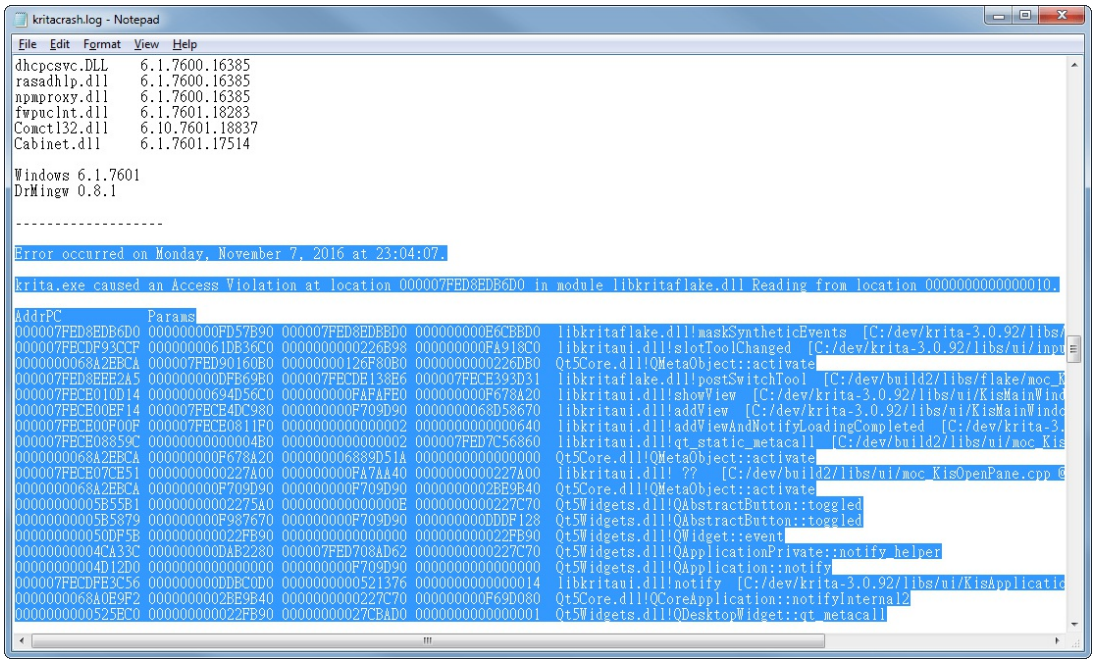
   An example of the crash dialog.

   - If Krita keeps on being unresponsive for more than a few minutes, it might actually be locked up, which may not give a backtrace. In that situation, you have to close Krita manually. Continue to follow the following instructions to check whether it was a crash or not.

2. Open Windows Explorer and type `%LocalAppData%` (without quotes) on the address bar and press the `Enter` key.

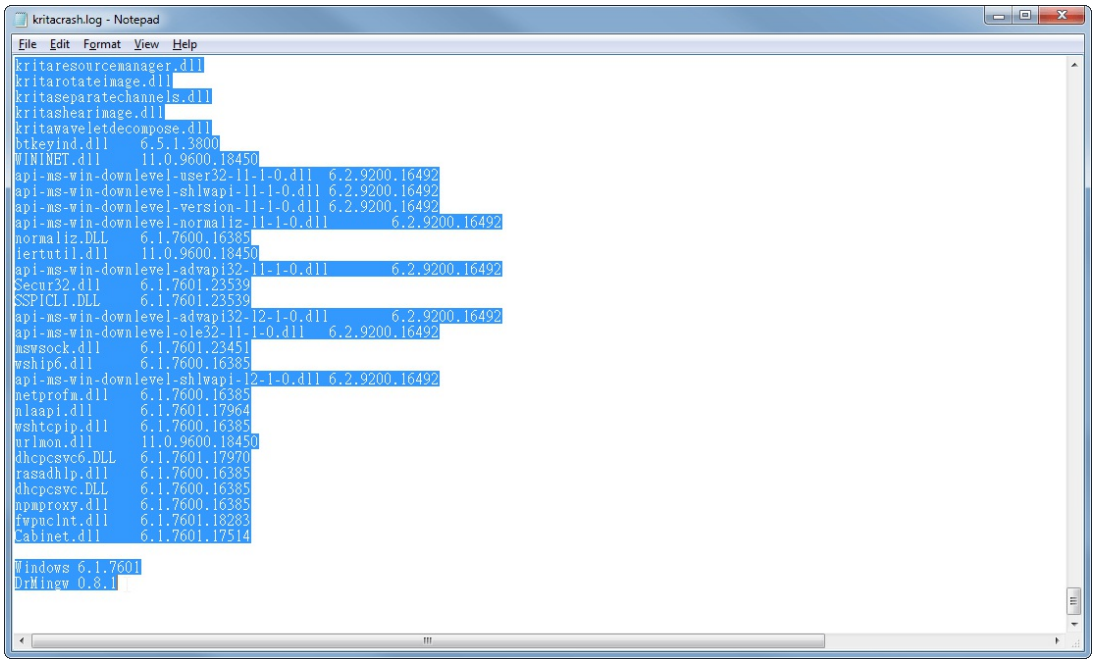
3. Find the file `kritacrash.log` (it might appear as simply `kritacrash` depending on your settings.)

4. Open the file with Notepad and scroll to the bottom, then scroll up to the first occurrence of “Error occurred on <time>” or the dashes.



Start of backtrace.

Check the time and make sure it matches the time of the crash.



End of backtrace.

The text starting from this line to the end of the file is the most

recent backtrace.

- If `kritacrash.log` does not exist, or a backtrace with a matching time does not exist, then you don’t have a backtrace. This means Krita was very likely locked up, and a crash didn’t actually happen. In this case, make a bug report too.
- If the backtrace looks truncated, or there is nothing after the time, it means there was a crash and the crash handler was creating the stack trace before being closed manually. In this case, try to re-trigger the crash and wait longer until the crash dialog appears.

Filters

Filters are little scripts or operations you can run on your drawing. You can visualize them as real-world camera filters that can make a photo darker or blurrier. Or perhaps like a coffee filter, where only water and coffee gets through, and the ground coffee stays behind.

Filters are unique to digital painting in terms of complexity, and their part of the painting pipeline. Some artists only use filters to adjust their colors a little. Others, using Filter Layers and Filter Masks use them to dynamically update a part of an image to be filtered. This way, they can keep the original underneath without changing the original image. This is a part of a technique called ‘non-destructive’ editing.

Filters can be accessed via the *Filters* menu. Krita has two types of filters: Internal and G’MIC filters.

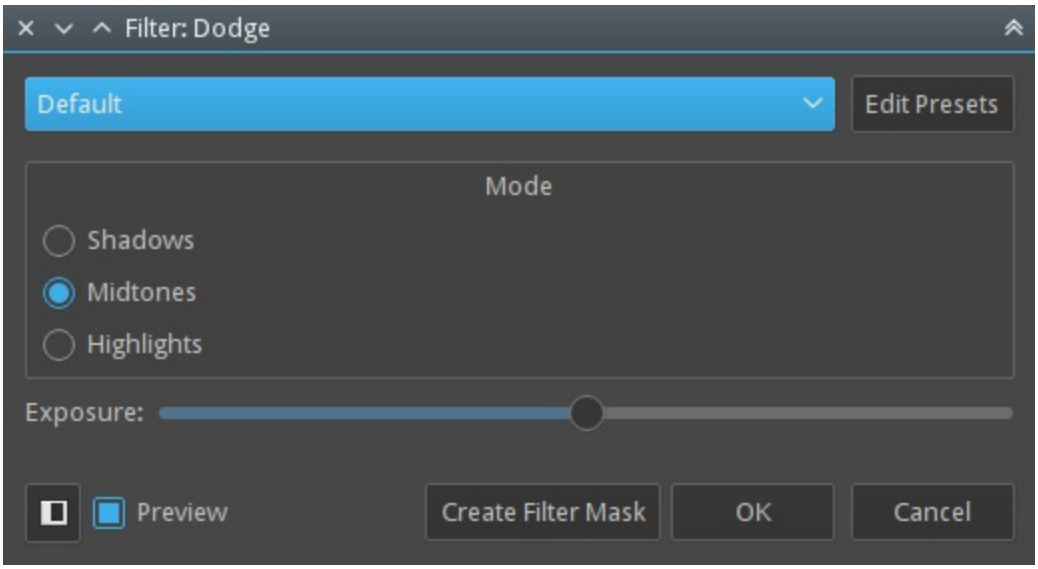
Internal filters are often multithreaded, and can thus be used with the filter brush or the adjustment filters.

Adjust

The Adjustment filters are image-wide and are for manipulating colors and contrast.

Dodge

An image-wide dodge-filter. Dodge is named after a trick in traditional dark-room photography that gave the same results.



Shadows
: The effect will mostly apply to dark tones.

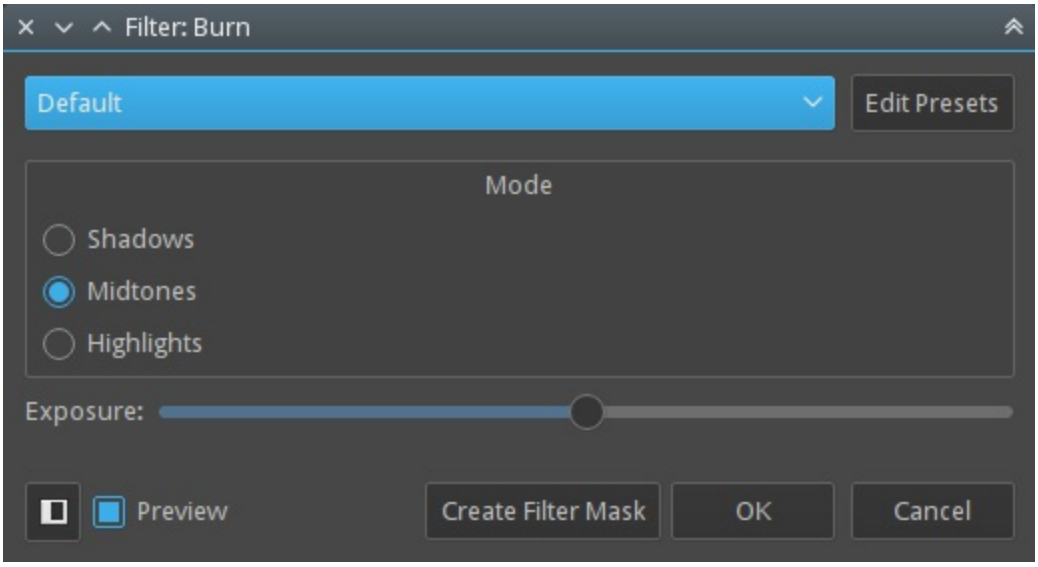
Midtones
: The effect will apply to mostly midtones.

Highlights
: This will apply the effect on the highlights only.

Exposure
: The strength at which this filter is applied.

Burn

An image-wide burn-filter. Burn is named after a trick in traditional dark-room photography that gave similar results.



Shadows
: The effect will mostly apply to dark tones.

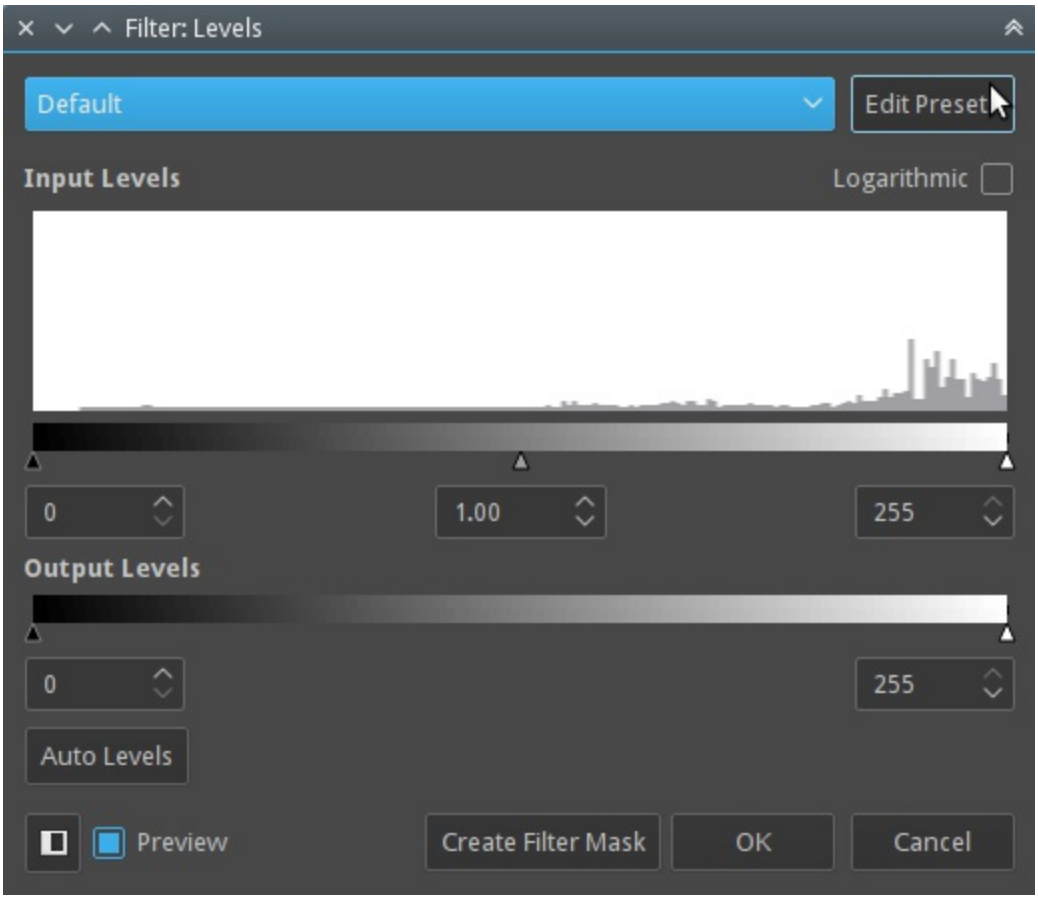
Midtones
: The effect will apply to mostly midtones.

Highlights
: This will apply the effect on the highlights only.

Exposure
: The strength at which this filter is applied.

Levels

This filter allows you to directly modify the levels of the tone-values of an image, by manipulating sliders for highlights, midtones and shadows. You can even set an output and input range of tones for the image. A histogram is displayed to show you the tonal distribution. The default shortcut for levels filter is `Ctrl + L`.



This is very useful to do an initial cleanup of scanned lineart or grayscale images. If the scanned lineart is light you can slide the black triangle to right to make it darker or if you want to remove the gray areas you can slide the white slider to left.

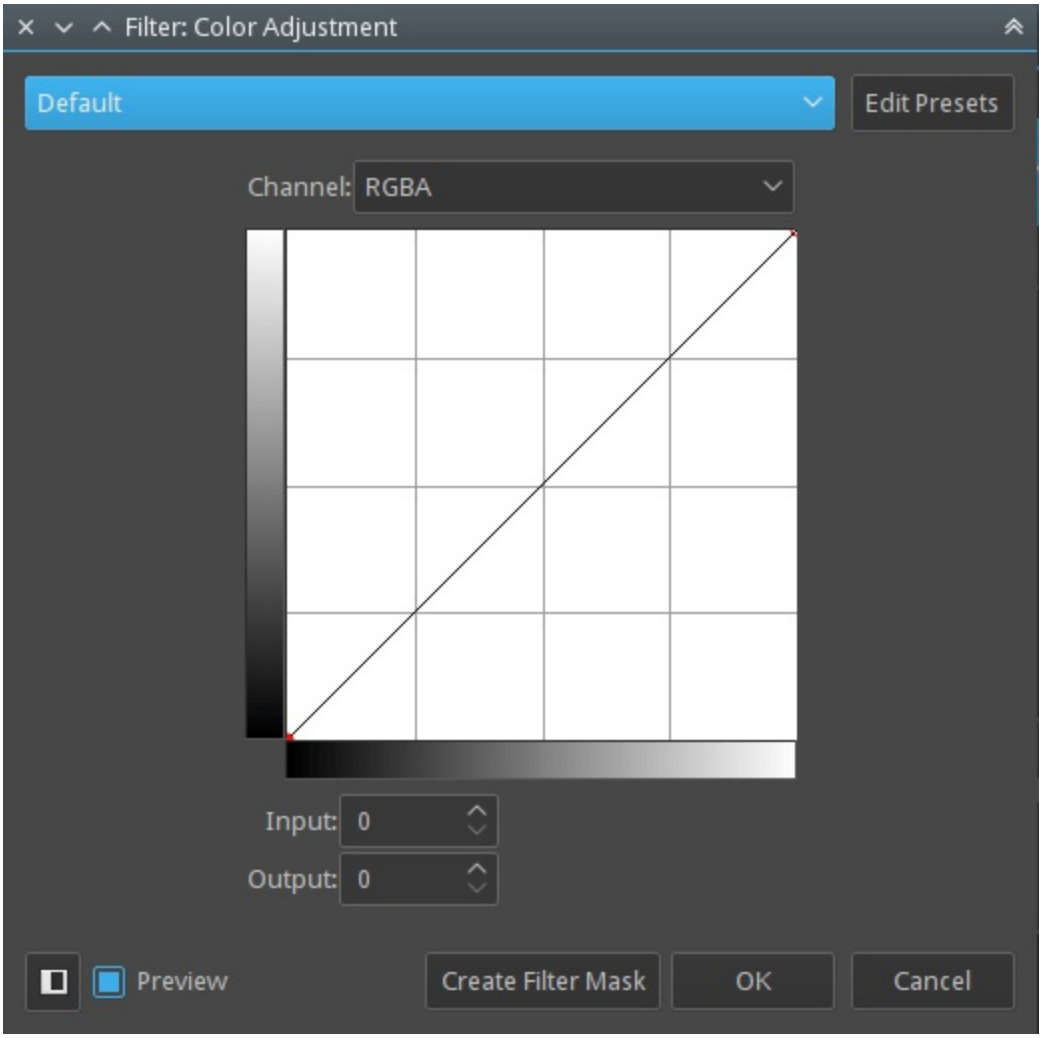
Auto levels is a quick way to adjust tone of an image. If you want to change the settings later you can click on the *Create Filter Mask* button to add the levels as a filter mask.

Color Adjustment Curves

This filter allows you to adjust each channel by manipulating the curves. You can even adjust the alpha channel and the lightness channel through this filter. This is used very often by artists as a post processing filter to slightly heighten the mood of the painting by adjust the overall color. For example a scene with fire breathing dragon may be made more red and yellow by

adjusting the curves to give it more warmer look, similarly a snowy mountain scene can be made to look cooler by adjusting the blues and greens. The default shortcut for this filter is `Ctrl + M`.

Changed in version 4.1: Since 4.1 this filter can also handle Hue and Saturation curves.



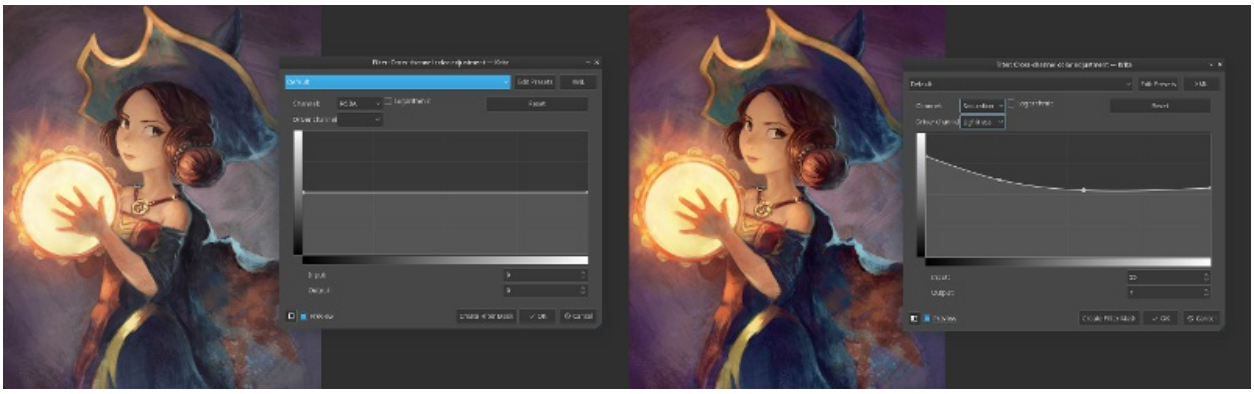
Cross-channel color adjustment

New in version 4.1.

Sometimes, when you are adjusting the colors for an image, you want bright colors to be more saturated, or have a little bit of brightness in the purples.

The Cross-channel color adjustment filter allows you to do this.

At the top, there are two drop-downs. The first one is to choose which *Channel* you wish to modify. The *Driver Channel* drop down is what channel you use to control which parts are modified.



The curve, on the horizontal axis, represents the driver channel, while the vertical axis represent the channel you wish to modify.

So if you wish to increase the saturation in the lighter parts, you pick *Saturation* in the first drop-down, and *Lightness* as the driver channel. Then, pull up the right end to the top.

If you wish to desaturate everything but the teal/blues, you select *Saturation* for the channel and *Hue* for the driver. Then put a dot in the middle and pull down the dots on either sides.

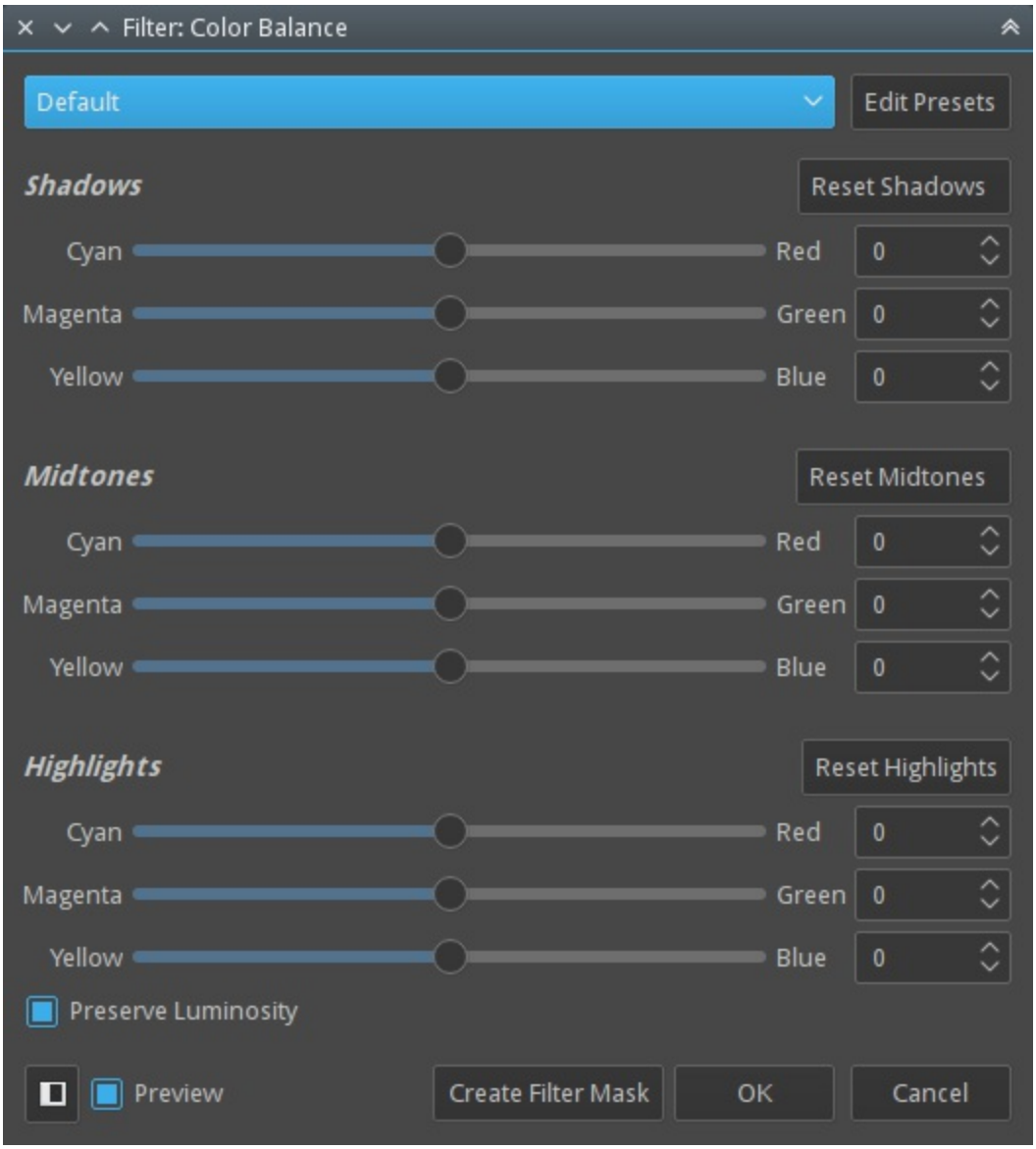
Brightness/Contrast curves

This filter allows you to adjust the brightness and contrast of the image by adjusting the curves.

Deprecated since version 4.0: These have been removed in Krita 4.0, because the Color Adjustment filter can do the same. Old files with brightness/contrast curves will be loaded as Color Adjustment curves.

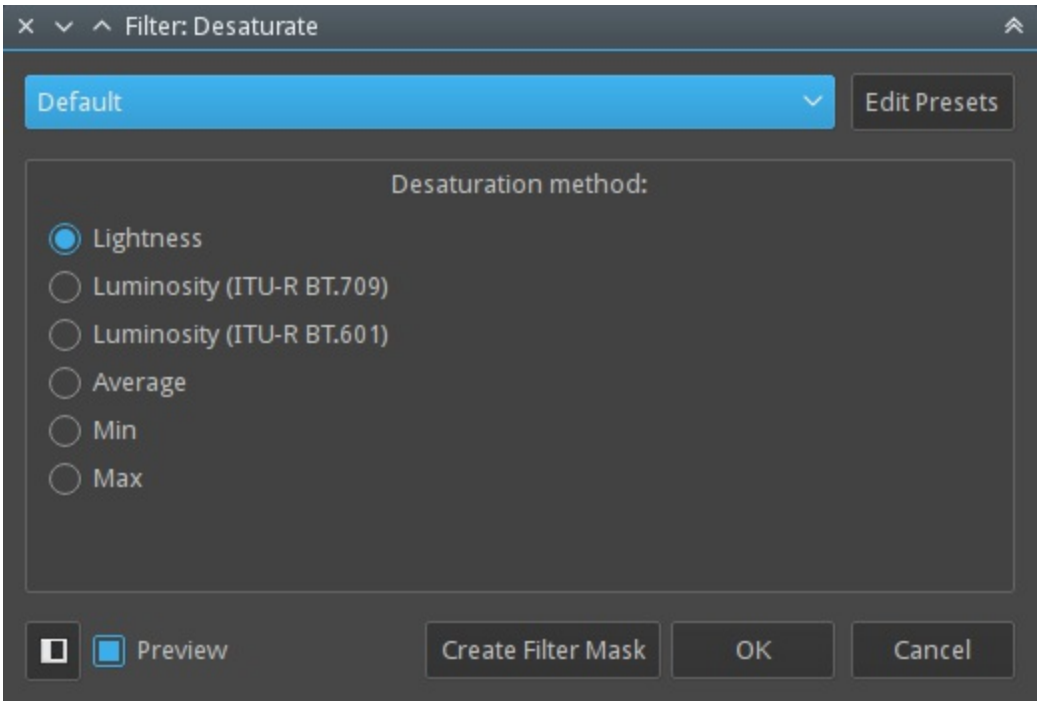
Color Balance

This filter allows you to control the color balance of the image by adjusting the sliders for Shadows, Midtones and Highlights. The default shortcut for this filter is `Ctrl + B`.



Desaturate

Image-wide desaturation filter. Will make any image Grayscale. Has several choices by which logic the colors are turned to gray. The default shortcut for this filter is `Ctrl + Shift + U`.



Lightness
: This will turn colors to gray using the HSL model.

Luminosity (ITU-R BT.709)
: Will turn the color to gray by using the appropriate amount of weighting per channel according to ITU-R BT.709.

Luminosity (ITU-R BT.601)
: Will turn the color to gray by using the appropriate amount of weighting per channel according to ITU-R BT.601.

Average
: Will make an average of all channels.

Min
: Subtracts all from one another to find the gray value.

Max
: Adds all channels together to get a gray value.

Invert

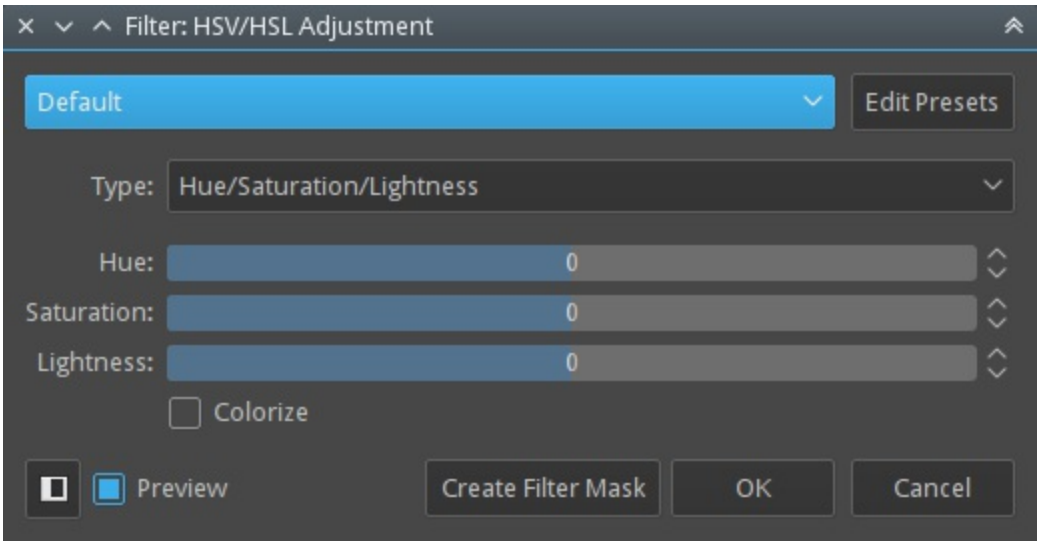
This filter like the name suggests inverts the color values in the image. So white (1,1,1) becomes black (0,0,0), yellow (1,1,0) becomes blue (0,1,1), etc. The default shortcut for this filter is `Ctrl + I`.

Auto Contrast

Tries to adjust the contrast the universally acceptable levels.

HSV/HSL Adjustment

With this filter, you can adjust the Hue, Saturation, Value or Lightness, through sliders. The default shortcut for this filter is `Ctrl + U`.



Colorize
: This is an option to have all the pixels have the same hue. It uses a HSL formula by default.

Legacy Mode
: In the development of Krita 4.3, the HSV algorithm was adjusted to maintain the variation in brightness better. This is important because brightness contrast is the most important contrast, so you want to avoid losing variation in it. This option toggles the old behaviour for files made in previous versions.

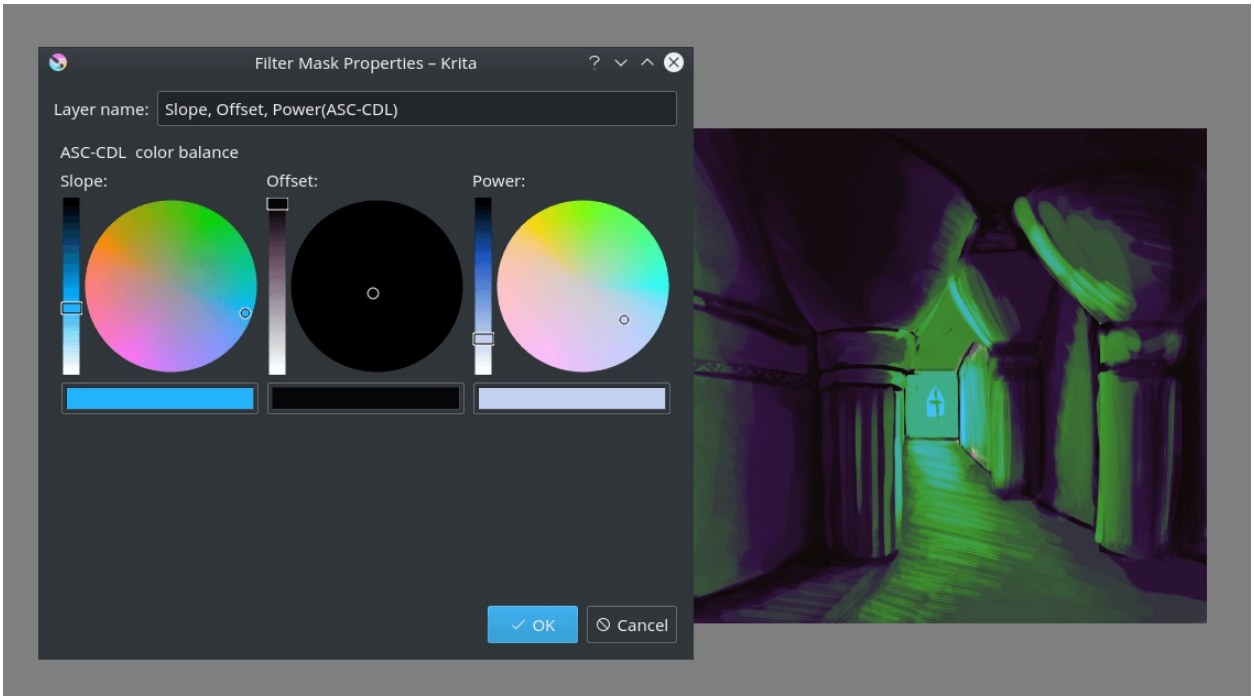
Threshold

A simple black and white threshold filter that uses sRGB luminosity. It'll convert any image to a image with only black and white, with the input number indicating the threshold value at which black becomes white.

Slope, Offset, Power

A different kind of color balance filter, with three color selectors, which will have the same shape as the one used in settings.

This filter is particular useful because it has been defined by the American Society for Cinema as "ASC_CDL", meaning that it is a standard way of describing a color balance method.



Slope
: This represents a multiplication and determine the adjustment of the brighter colors in an image.

Offset
: This determines how much the bottom is offset from the top, and so

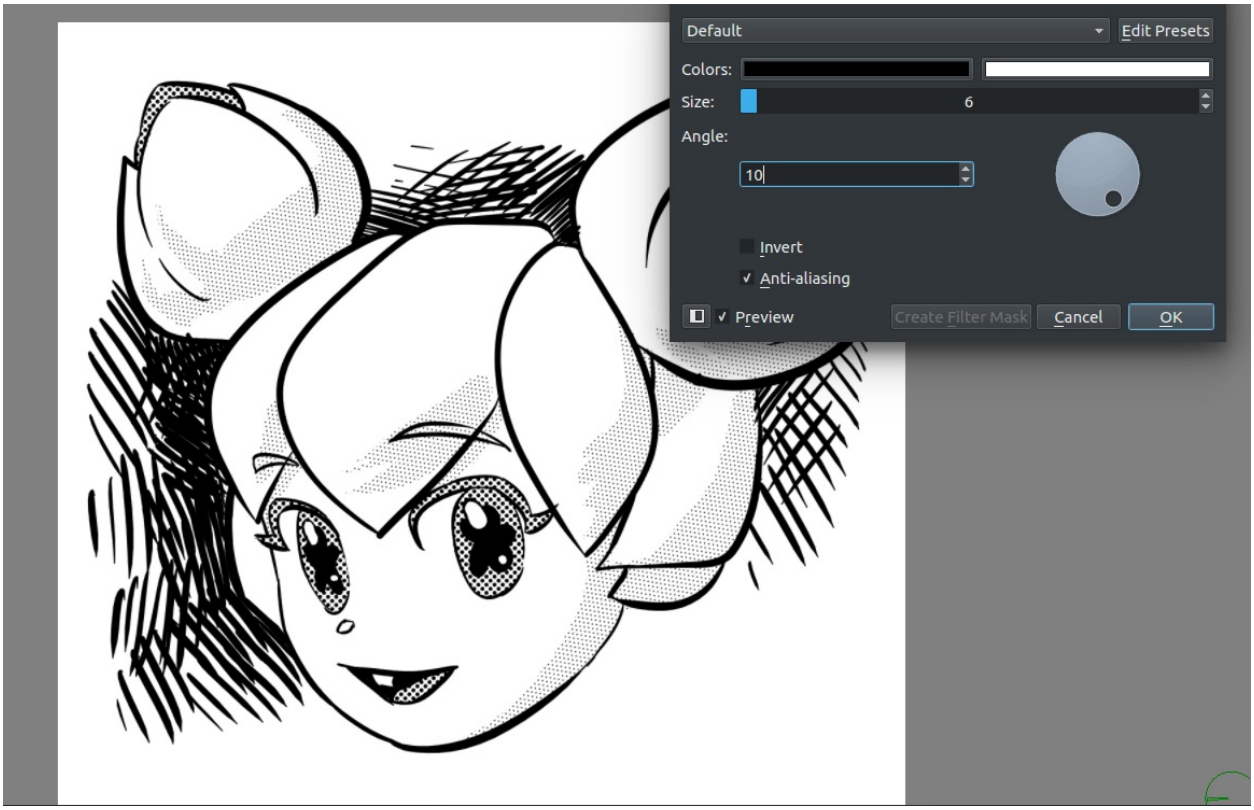
determines the color of the darkest colors.

Power
: This represents a power function, and determines the adjustment of the mid-tone to dark colors of an image.

Artistic

The artistic filter are characterised by taking an input, and doing a deformation on them.

Halftone



The halftone filter is a filter that converts the colors to a halftone dot pattern.

Colors
: The colors used to paint the pattern. The first is the color of the dots, the second the color of the background.

Size
: The size of the cell in pixels. The maximum dot size will be using the diagonal as the cell size to make sure you can have pure black.

Angle
: The angle of the dot pattern.

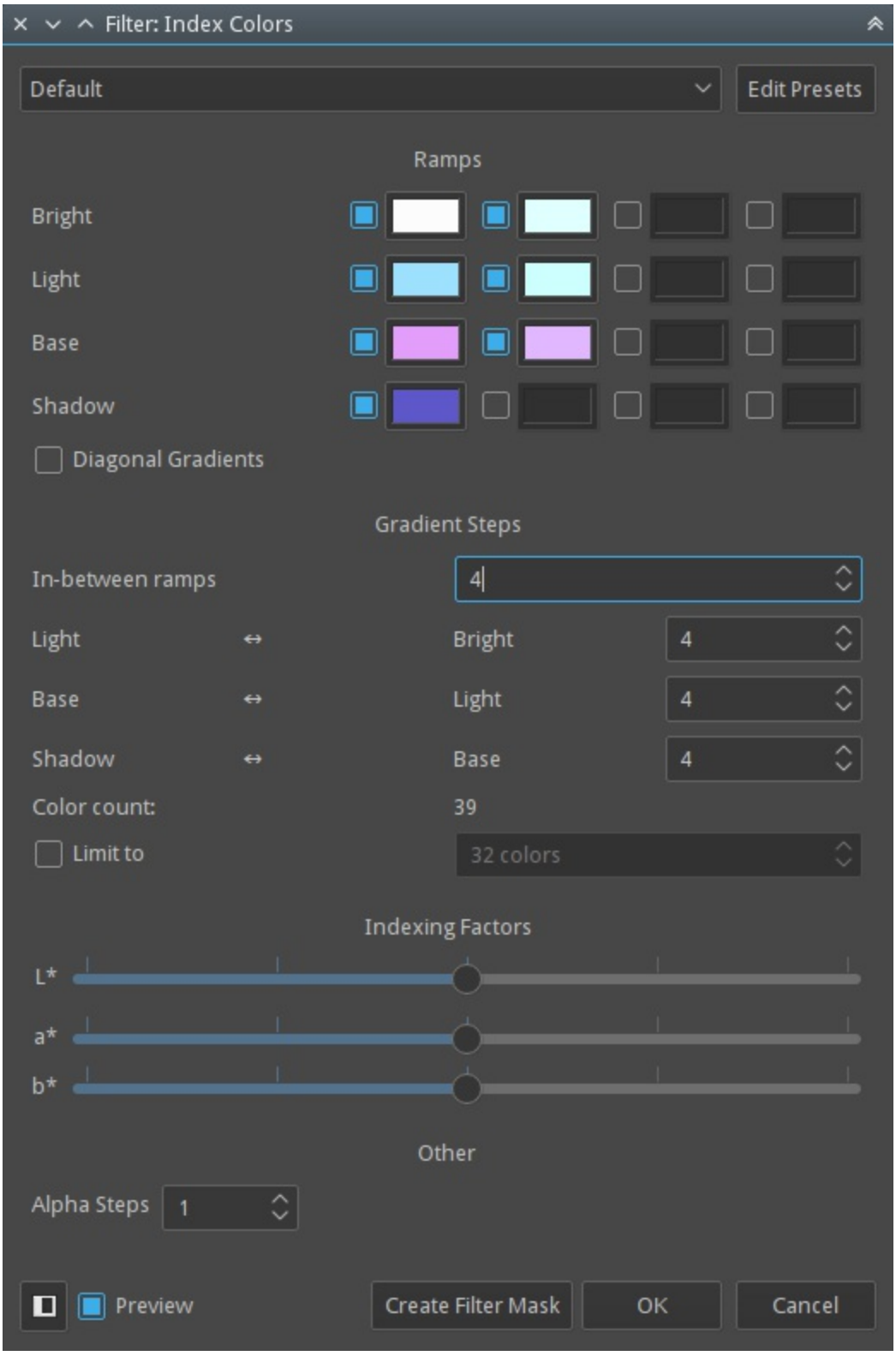
Invert
: This inverts the intensity calculated per dot. Thus, dark colors will give tiny dots, and light colors big dots. This is useful in combination with inverting the colors, and give a better pattern on glowy-effects.

Anti-aliasing
: This makes the dots smooth, which is good for webgraphics. Sometimes, for print graphics, we want there to be no grays, so we turn off the anti-aliasing.

Index Color

The index color filter maps specific user selected colors to the grayscale value of the artwork. You can see the example below, the strip below the black and white gradient has index color applied to it so that the black and white gradient gets the color selected to different values.

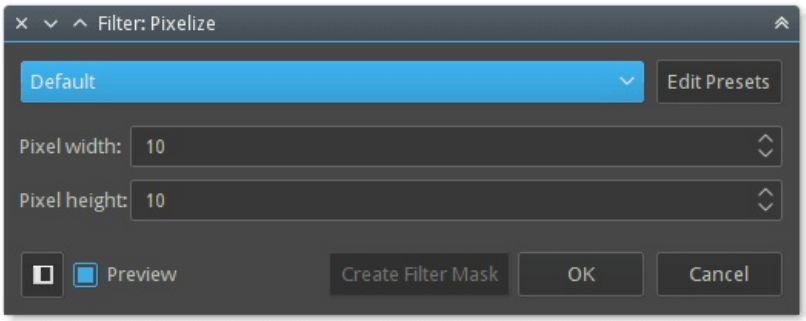
You can choose the required colors and ramps in the index color filter dialog as shown below .

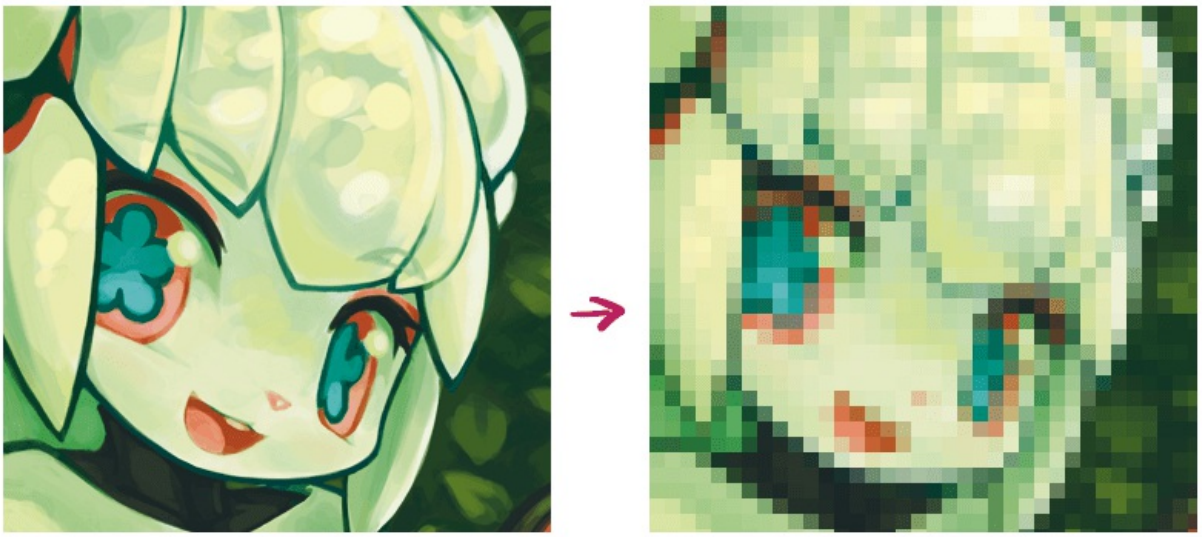


You can create index painting such as one shown below with the help of this filter.

Pixelize

Makes the input-image pixely by creating small cells and inputting an average color.



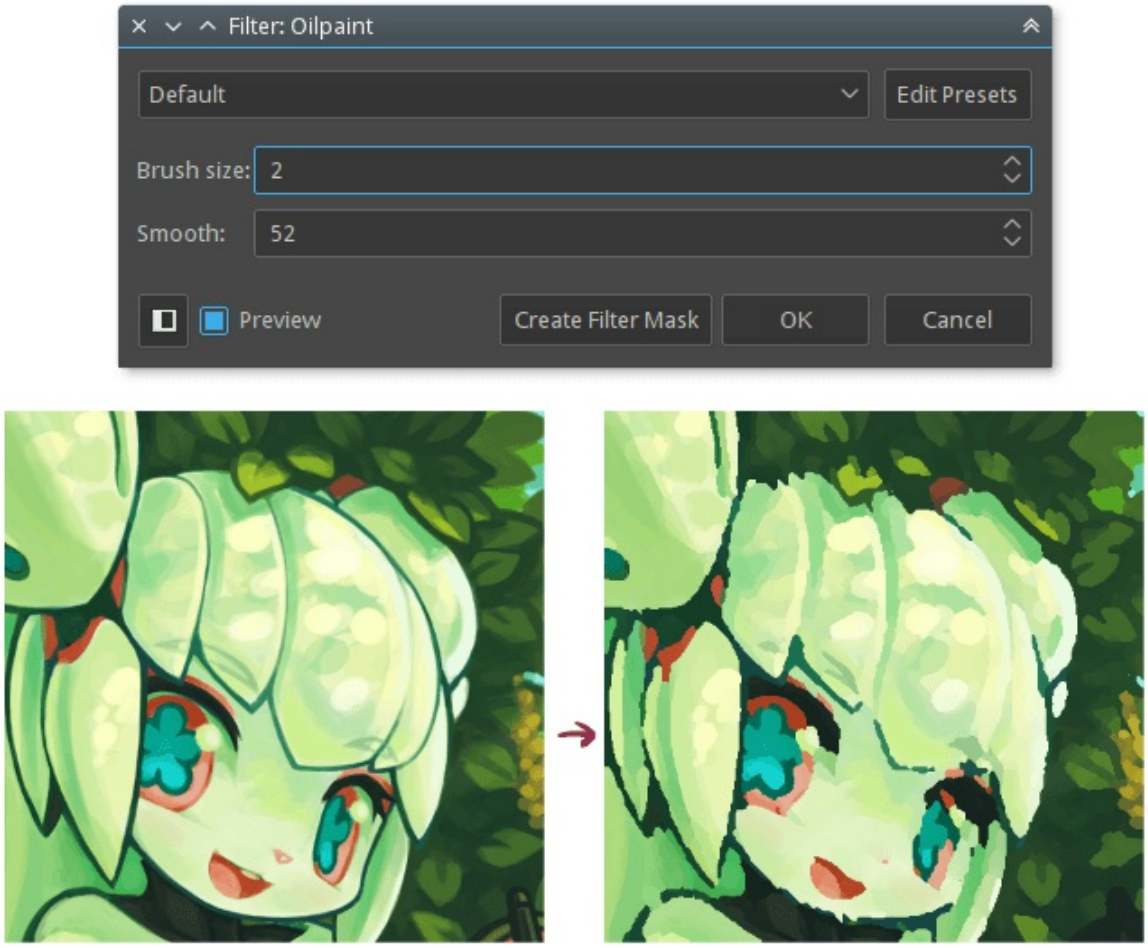


Raindrops

Adds random raindrop-deformations to the input-image.

Oilpaint

Does semi-posterisation to the input-image, with the ‘brush-size’ determining the size of the fields.

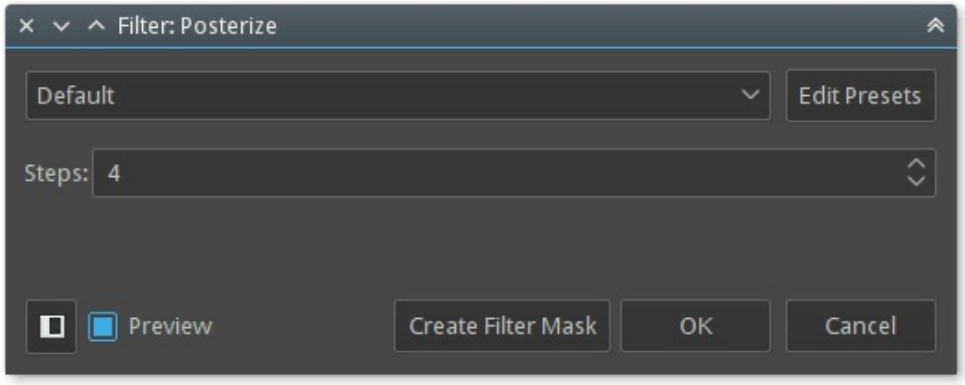


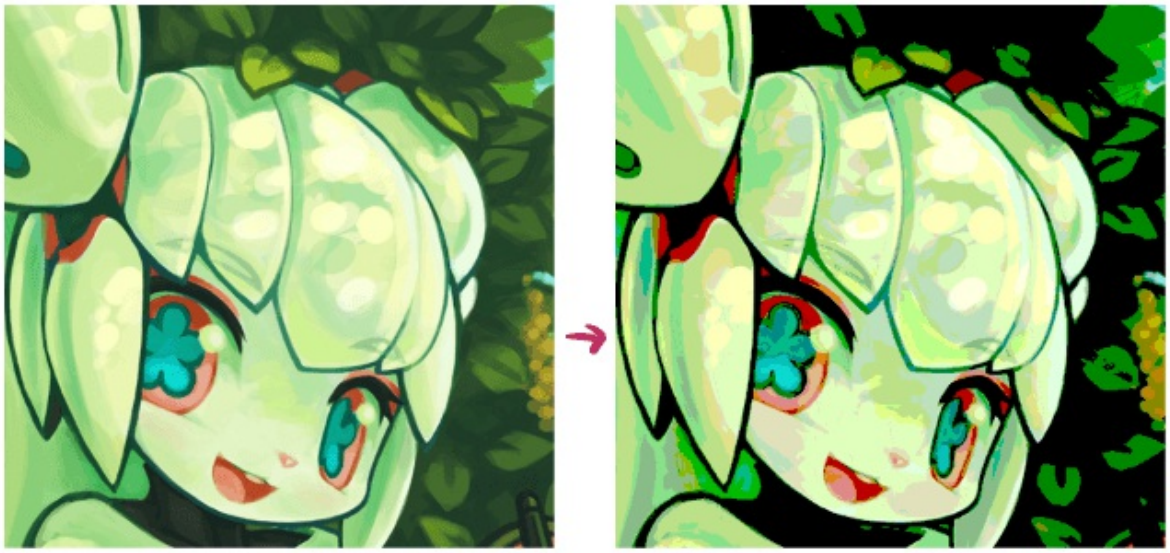
Brush-size
: Determines how large the individual patches are. The lower, the more detailed.

Smoothness
: Determines how much each patch’s outline is smoothed out.

Posterize

This filter decreases the amount of colors in an image. It does this per component (channel).

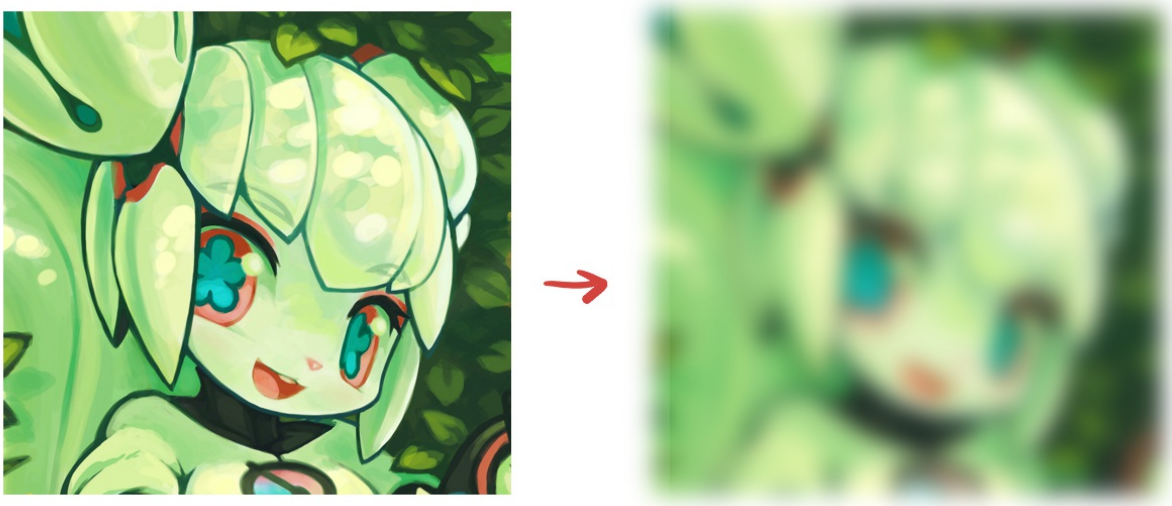




The *Steps* parameter determines how many colors are allowed per component.

Blur

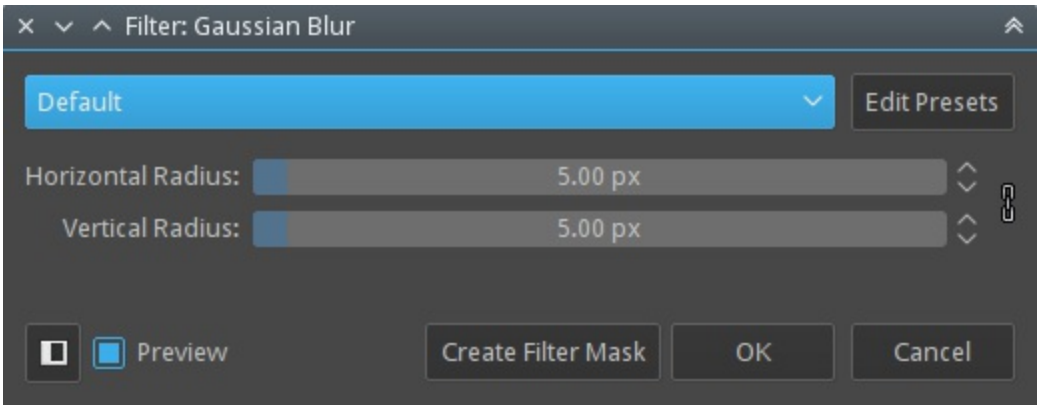
The blur filters are used to smoothen out the hard edges and details in the images. The resulting image is blurry. below is an example of a blurred image. The image of Kiki on right is the result of blur filter applied to the image on left.



There are many different filters for blurring:

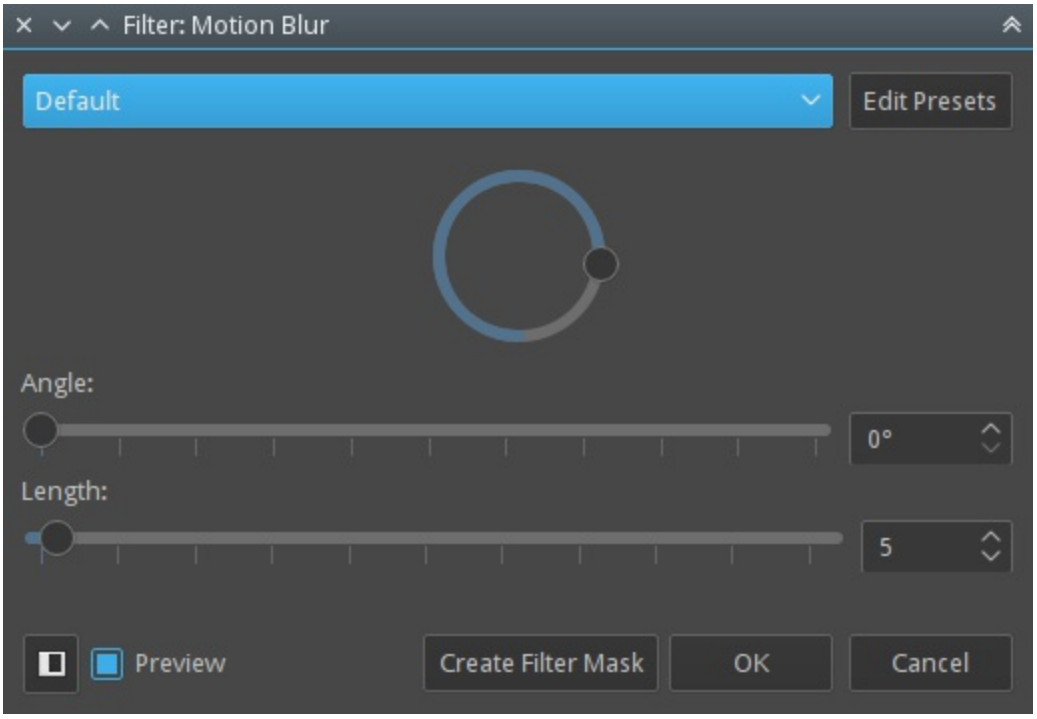
Gaussian Blur

You can input the horizontal and vertical radius for the amount of blurring here.



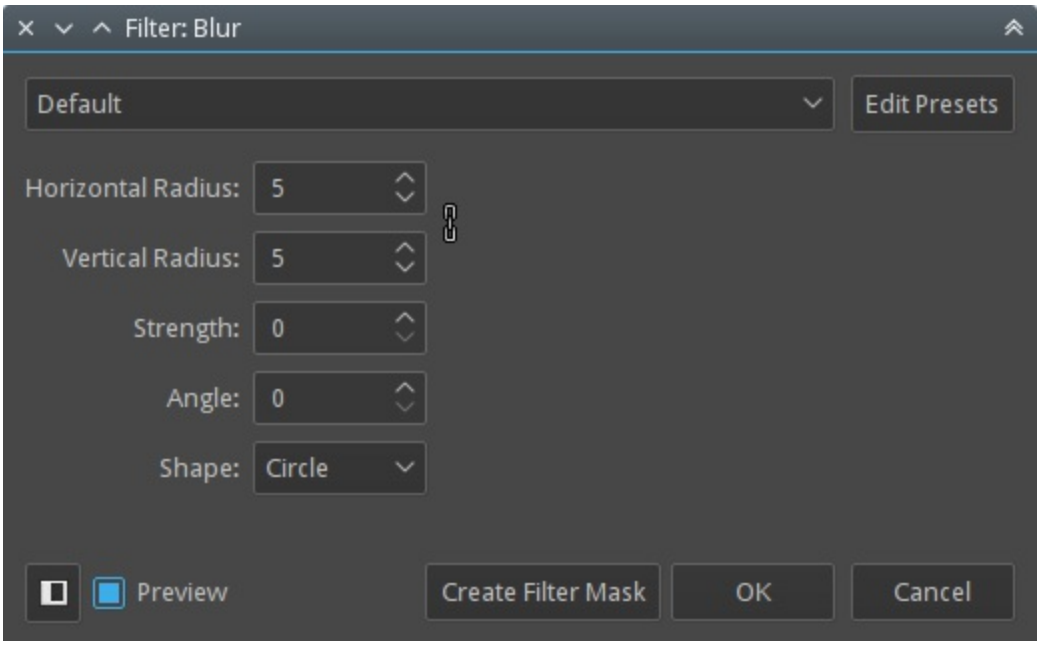
Motion Blur

Doesn’t only blur, but also subtly smudge an image into a direction of the specified angle thus giving a feel of motion to the image. This filter is often used to create effects of fast moving objects.



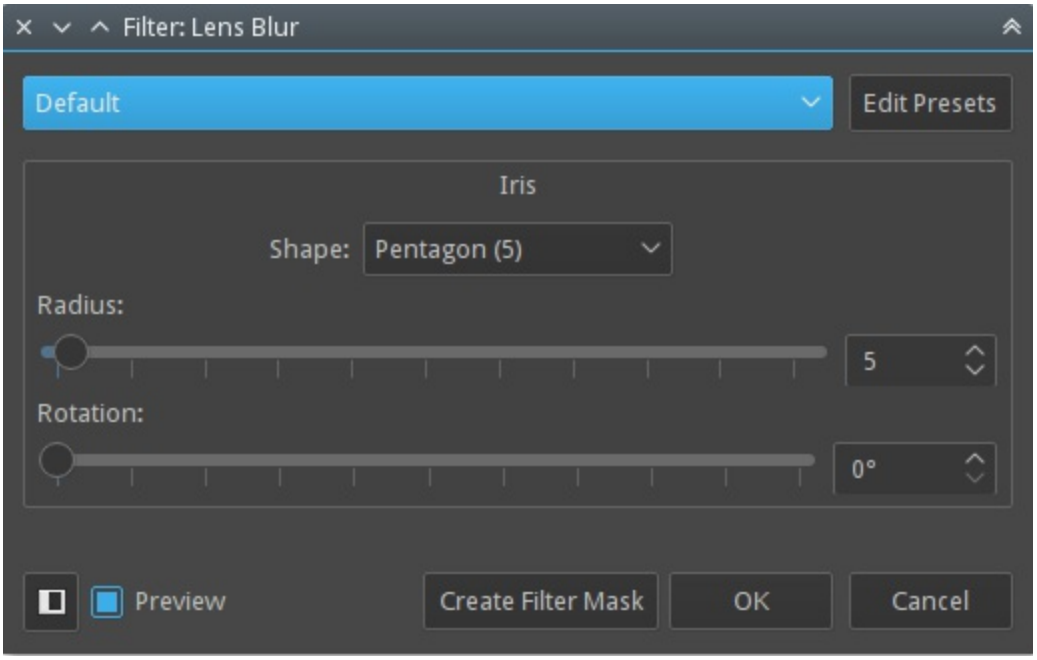
Blur

This filter creates a regular blur.



Lens Blur

Lens Blur Algorithm.

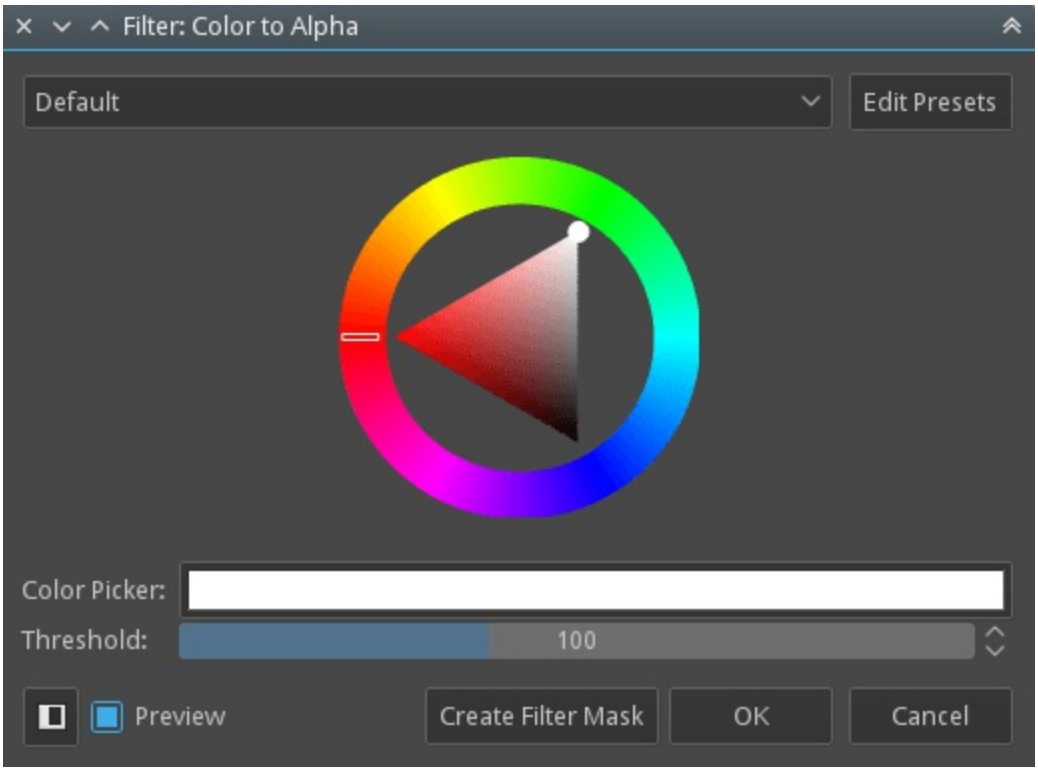


Color

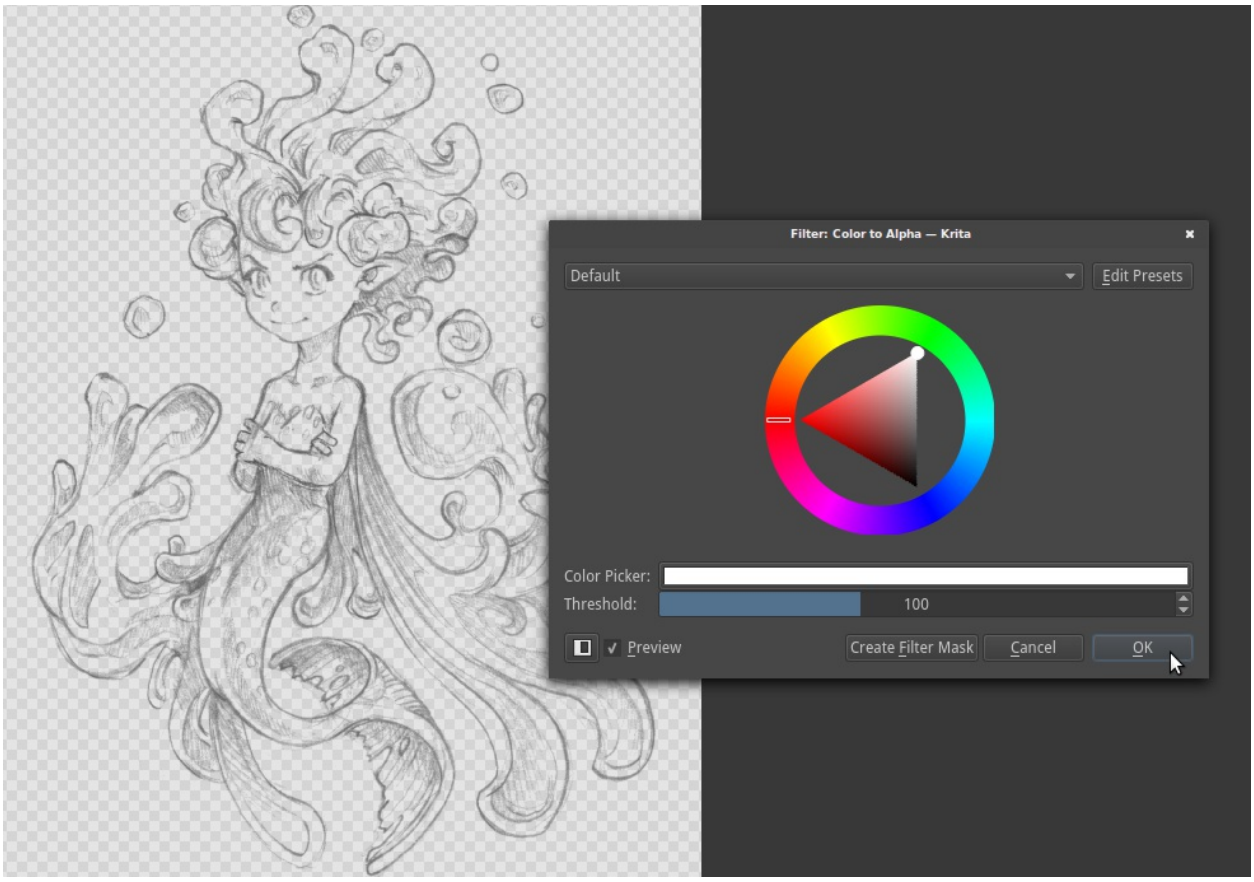
Similar to the Adjust filters, the color filters are image wide color operations.

Color to Alpha

This filter allows you to make one single color transparent (alpha). By default when you run this filter white is selected, you can choose a color that you want to make transparent from the color selector.



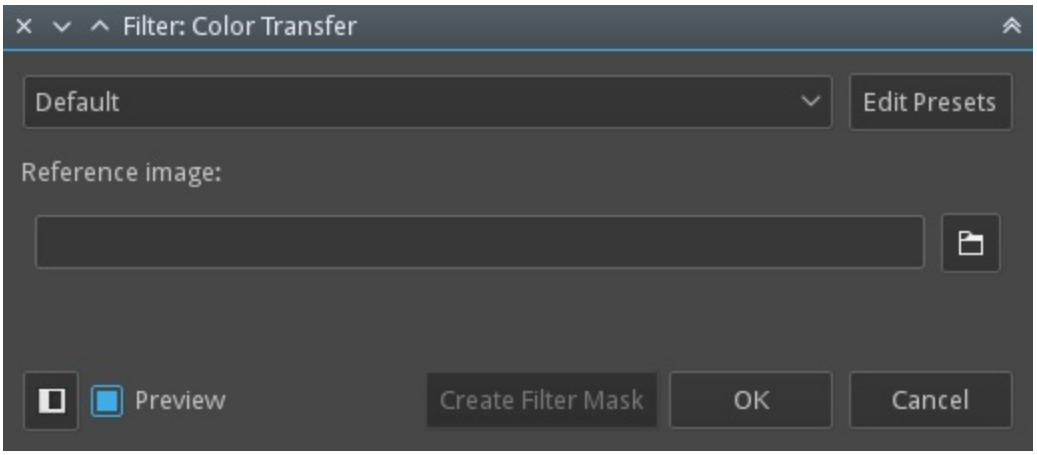
The Threshold indicates how much other colors will be considered mixture of the removed color and non-removed colors. For example, with threshold set to 255, and the removed color set to white, a 50% gray will be considered a mixture of black+white, and thus transformed in a 50% transparent black.



This filter is really useful in separating line art from the white background.

Color Transfer

This filter converts the colors of the image to colors from the reference image. This is a quick way to change a color combination of an artwork to an already saved image or a reference image.



Maximize Channel

This filter checks for all the channels of a each single color and set all but the highest value to 0.

Minimize Channel

This is reverse to Maximize channel, it checks all the channels of a each single color and sets all but the lowest to 0.

Edge Detection

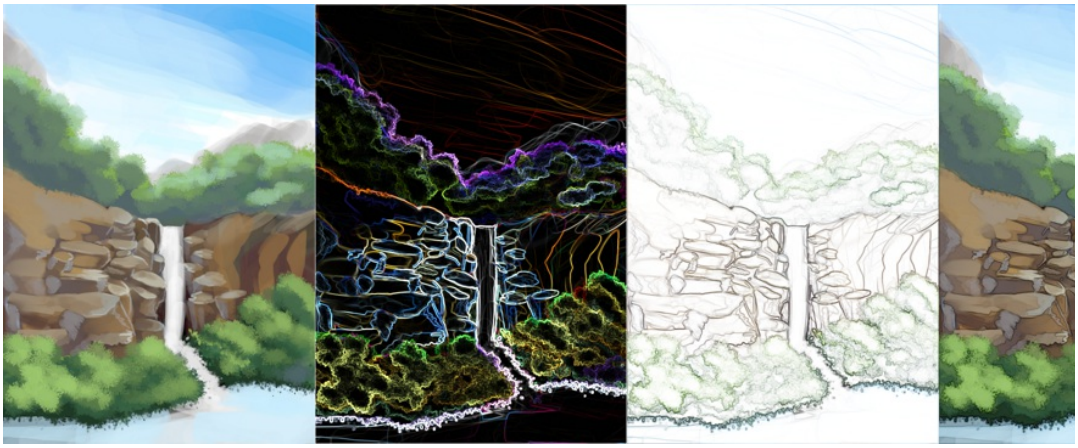
Edge detection filters focus on finding sharp contrast or border between colors in an image to create edges or lines.

Since 4.0 there are only two edge detection filters.

Edge Detection

New in version 4.0.

A general edge detection filter that encapsulates all other filters. Edge detection filters that were separate before 4.0 have been folded into this one. It is also available for filter layers and filter brushes.



From left to right: Original, with Prewitt edge detection applied, with Prev
applied and result applied to alpha channel, and finally the original with an
layer with the same settings as 3, and the filter layer blending mode se

Formula

The convolution kernel formula for the edge detection. The difference between these is subtle, but still worth experimenting with.

Simple

A Kernel that is not square unlike the other two, and while this makes it fast, it doesn’t take diagonal pixels into account.

Prewitt
: A square kernel that includes the diagonal pixels just as strongly as the orthogonal pixels. Gives a very strong effect.

Sobel
: A square kernel that includes the diagonal pixels slightly less strong than the orthogonal pixels. Gives a more subtle effect than Prewitt.

Output
: The output.

All sides
: Convolves the edge detection into all directions and combines the result with the Pythagorean theorem. This will be good for most uses.

Top Edge
: This only detects changes going from top to bottom and thus only has top lines.

Bottom Edge
: This only detects changes going from bottom to top and thus only has bottom lines.

Right Edge
: This only detects changes going from right to left and thus only has right lines.

Left Edge
: This only detects changes going from left to right and thus only has left lines.

Direction in Radians
: This convolves into all directions and then tries to output the direction of the line in radians.

Horizontal/Vertical radius

: The radius of the edge detection. Default is 1 and going higher will increase the thickness of the lines.

Apply result to Alpha Channel.

: The edge detection will be used on a grayscale copy of the image, and the output will be onto the alpha channel of the image, meaning it will output lines only.

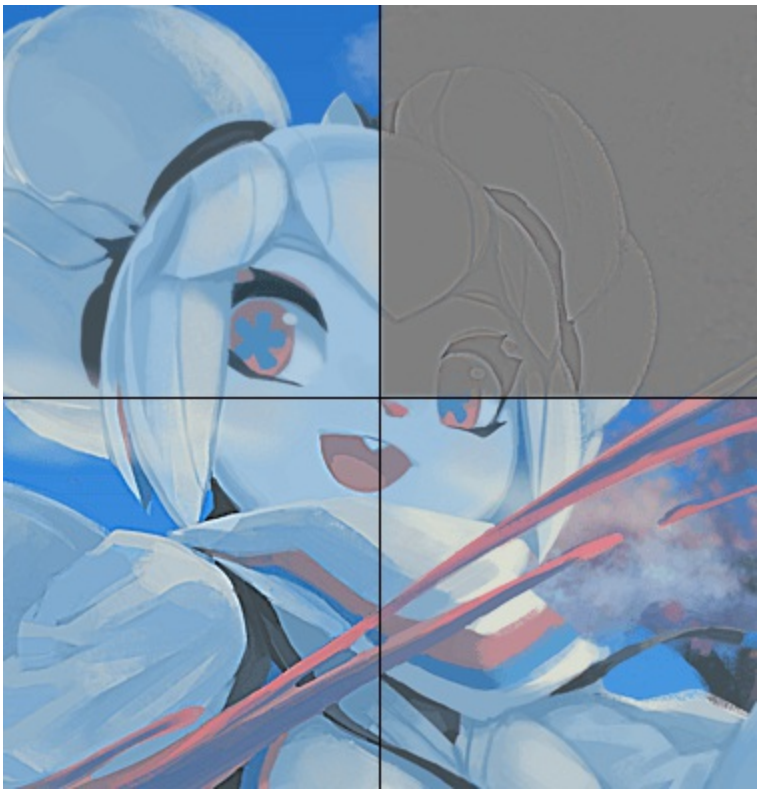
Gaussian High Pass

A High Pass filter is a type of edge detection filter. It is usually used to enhance contrasts, much like a sharpen filter, but within a texture editing workflow it is also used to remove local gradients.

Radius

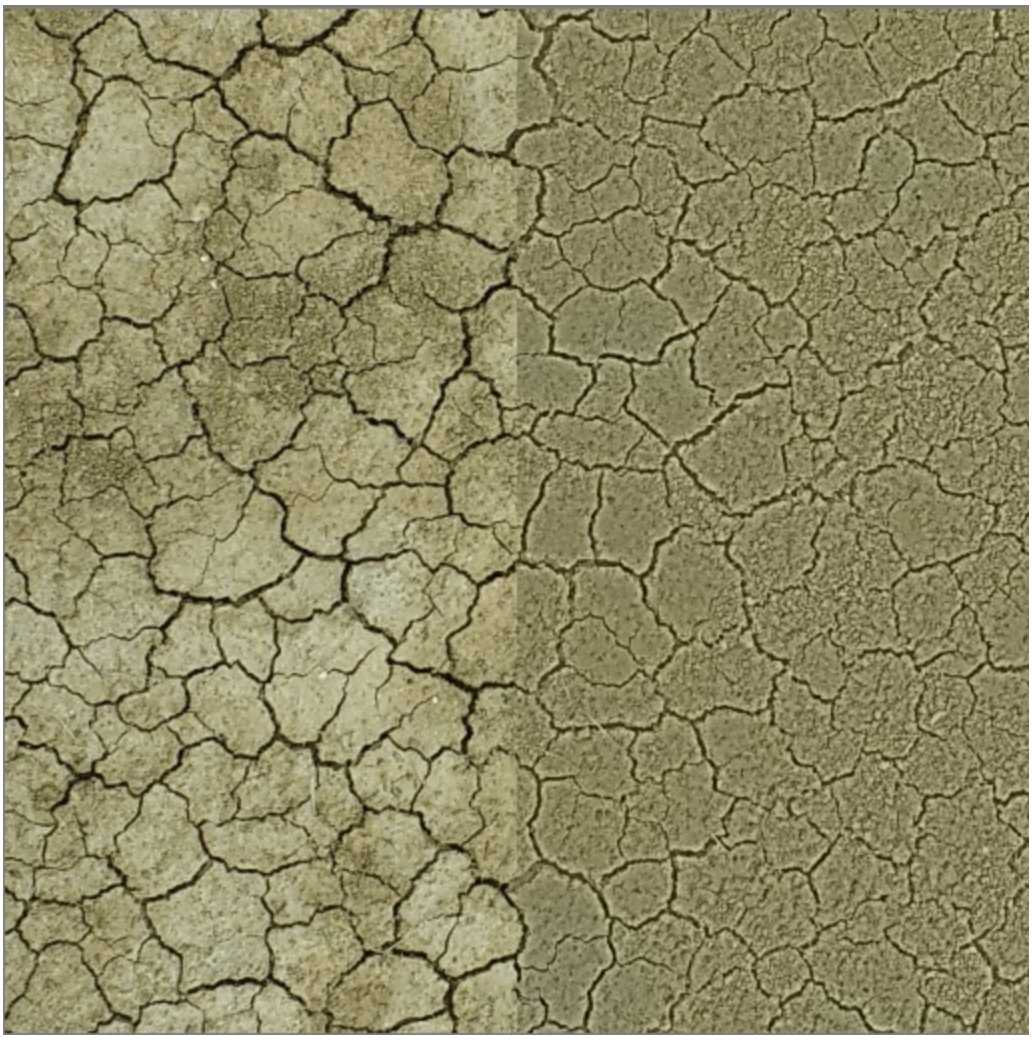
: The radius within the Gaussian High Pass filter is similar to the radius in the Edge Detection filter.

To use this as a sharpen filter, create a filter layer with this filter, and then set the blending mode to modes like ‘soft light’, ‘overlay’, ‘hard light’, ‘linear light’. Different blending modes give different results.



Top left: Original, **top right**: Gaussian Highpass Result with radius 3, **bottom left**: Gaussian High Pass Result with radius 3 blended over the original with to Linear Light, **bottom right**: Gaussian High Pass result with radius 3 blended over the original with Soft Light.

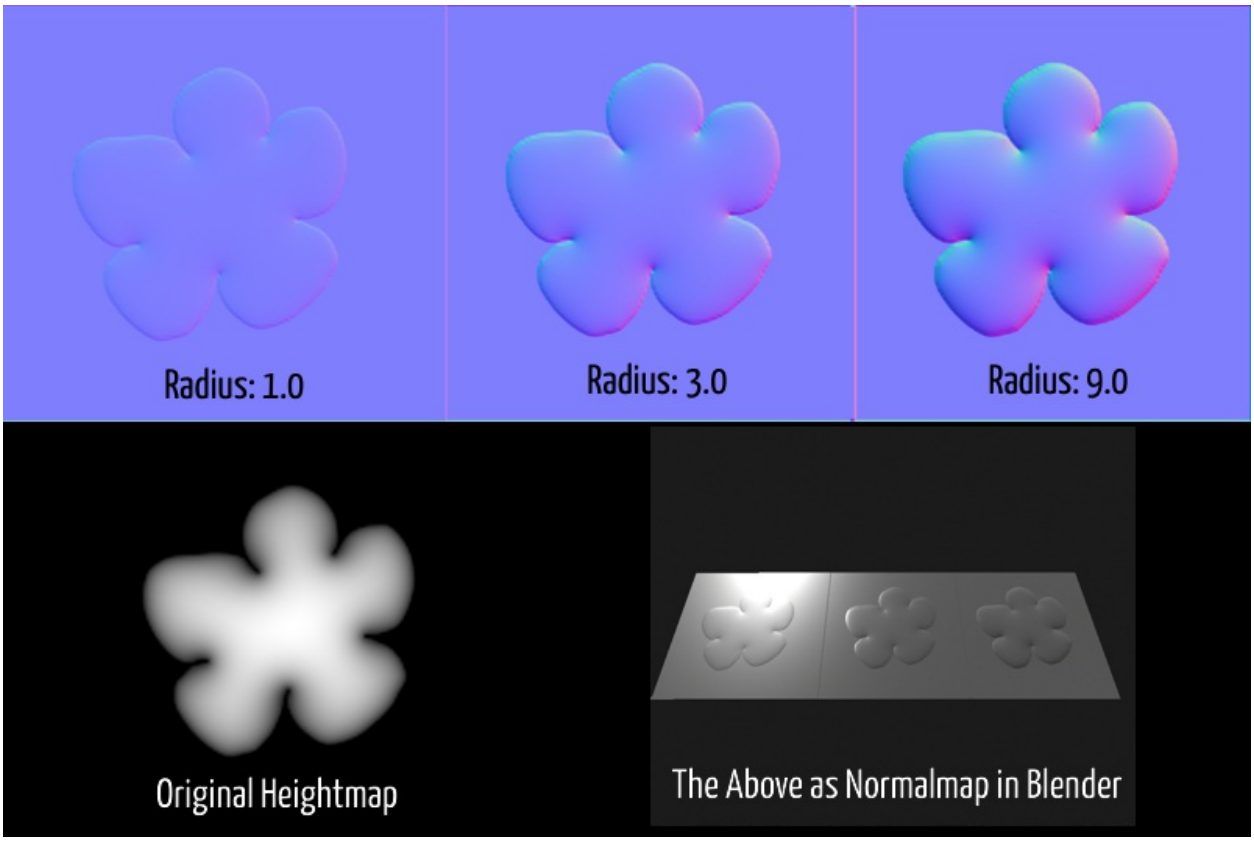
To remove local gradients from a texture, create a clone layer, and apply this filter as a filter mask. Then, put a filter layer with gaussian blur set to the full amount in between the clone layer and the original. Finally, set the clone layer to luminosity or multiply (in this case an extra filter mask needs to be added to reduce the levels so that the multiplication result will not be as strong).



Left: Original, **top right**: Gaussian High Pass Result blended with luminosity to remove the local gradients but keep the sharp details. In this specific example the lack of local gradients removes some character, but the gaussian high pass result could also be used to create a heightmap.

Height to Normal Map

New in version 4.0.



A filter that converts Height maps to Normal maps through the power of edge detection. It is also available for the filter layer or filter brush.

Formula

The convolution kernel formula for the edge detection. The difference between these is subtle, but still worth experimenting with.

Simple

A Kernel that is not square unlike the other two, and while this makes it fast, it doesn't take diagonal pixels into account.

Prewitt

A square kernel that includes the diagonal pixels just as strongly as the orthogonal pixels. Gives a very strong effect.

Sobel

A square kernel that includes the diagonal pixels slightly less strong than the orthogonal pixels. Gives a more subtle effect than Prewitt.

Channel

Which channel of the layer should be interpreted as the grayscale heightmap.

Horizontal/Vertical radius

The radius of the edge detection. Default is 1 and going higher will increase the strength of the normal map. Adjust this if the effect of the resulting normal map is too weak.

XYZ

An XYZ swizzle, that allows you to map Red, Green and Blue to different 3d normal vector coordinates. This is necessary mostly for the difference between MikkT-space normal maps (+X, +Y, +Z) and the OpenGL standard normal map (+X, -Y, +Z).

Emboss

Filters that are named by the traditional embossing technique. This filter generates highlight and shadows to create an effect which makes the image look like embossed. Emboss filters are usually used in the creation of interesting GUI elements, and mostly used in combination with filter-layers and masks.

Emboss Horizontal Only

Only embosses horizontal lines.

Emboss in all Directions

Embosses in all possible directions.

Emboss (Laplacian)

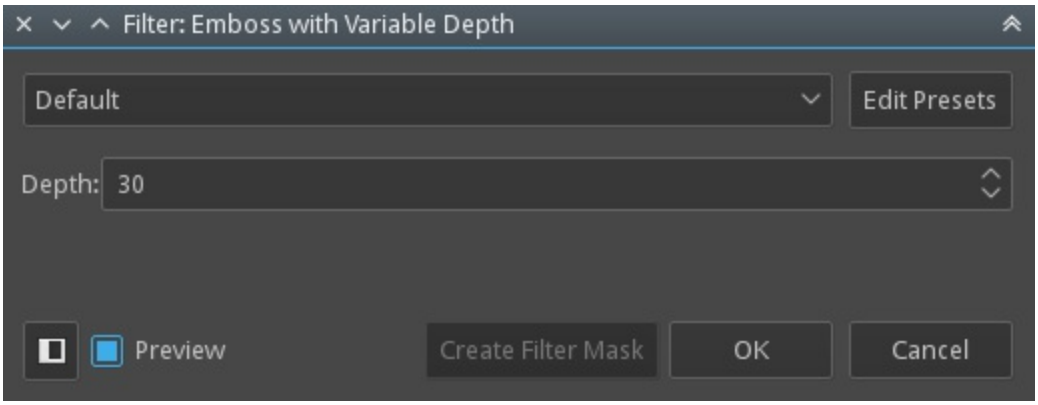
Uses the laplacian algorithm to perform embossing.

Emboss Vertical Only

Only embosses vertical lines.

Emboss with Variable depth

Embosses with a depth that can be set through the dialog box shown below.



Emboss Horizontal and Vertical

Only embosses horizontal and vertical lines.

Enhance

These filters all focus on reducing the blur in the image by sharpening and enhancing details and the edges. Following are various sharpen and enhance filters in provided in Krita.

- Sharpen
- Mean Removal
- Unsharp Mask
- Gaussian Noise reduction
- Wavelet Noise Reducer

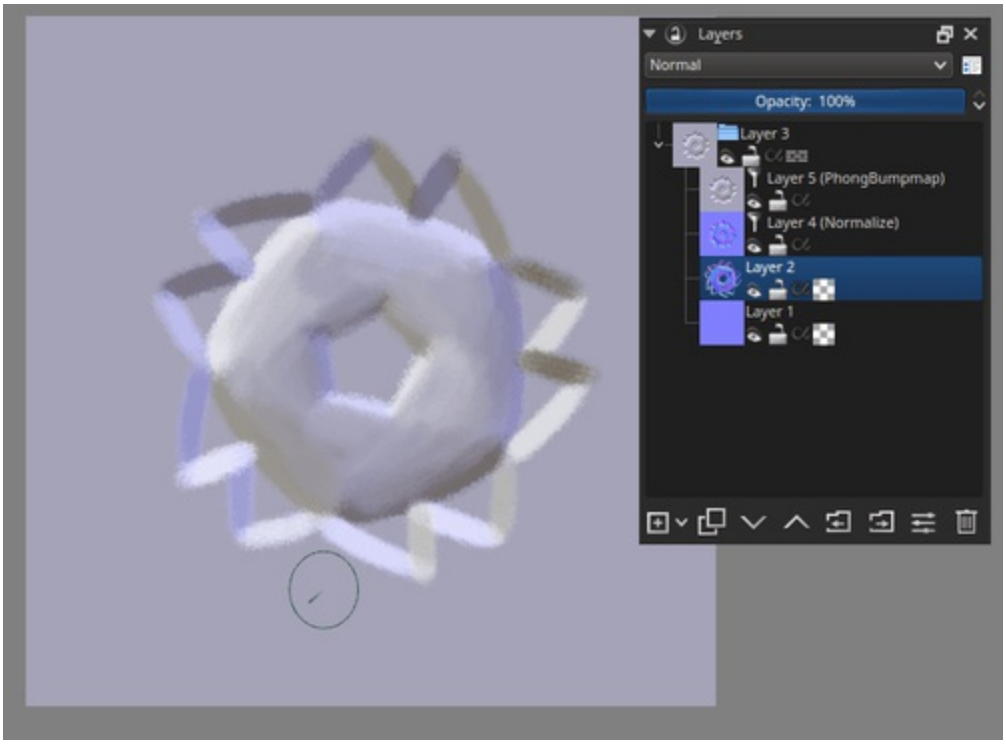
Map

Filters that are signified by them mapping the input image.

Small Tiles

Tiles the input image, using its own layer as output.

Phong Bumpmap



Uses the input image as a height-map to output a 3d something, using the phong-lambert shading model. Useful for checking one's height maps during game texturing. Checking the *Normal Map* box will make it use all channels and interpret them as a normal map.

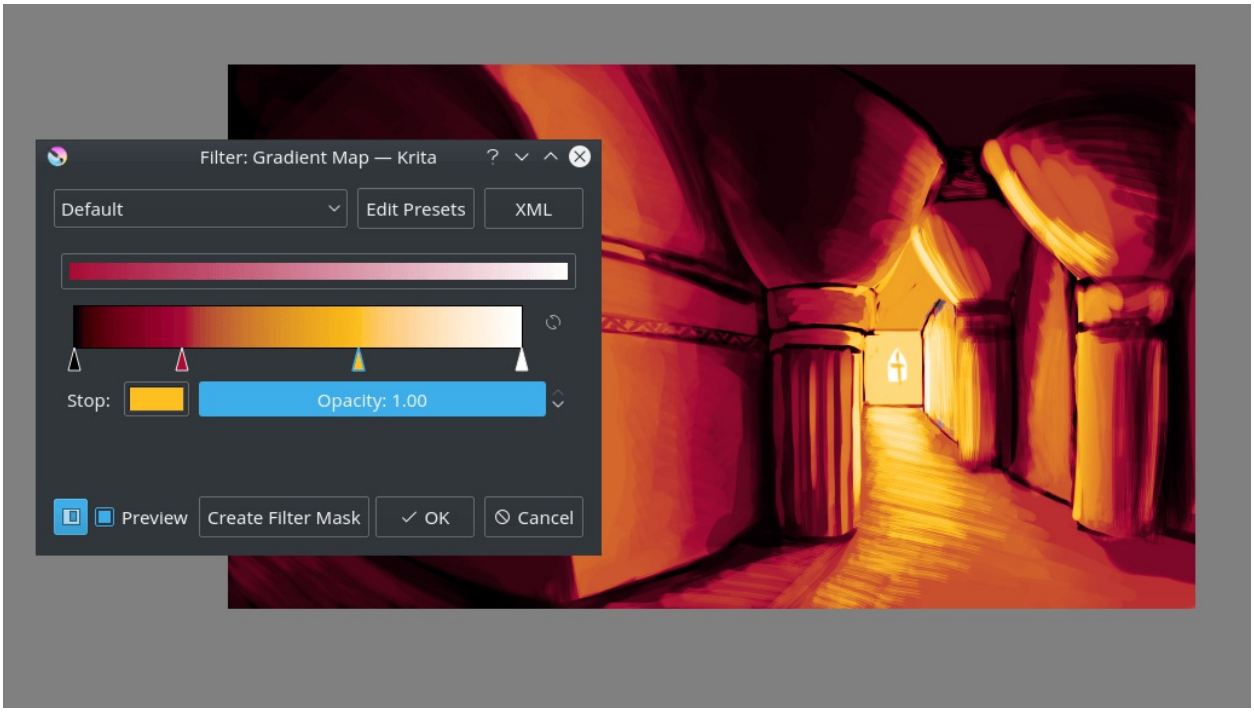
Round Corners

Adds little corners to the input image.

Normalize

This filter takes the input pixels, puts them into a 3d vector, and then normalizes (makes the vector size exactly 1) the values. This is helpful for normal maps and some minor image-editing functions.

Gradient Map



Maps the lightness of the input to the selected gradient. Useful for fancy artistic effects.

In 3.x you could only select predefined gradients. In 4.0, you can select gradients and change them on the fly, as well as use the gradient map filter as a filter layer or filter brush.

Color Modes

- **Blend:** smoothly blend colors between stops

- **Nearest:** selects color from nearest stops
- **Dither:** dithers between stop colors as per Dithering Threshold Modes.

Palettize

Maps the color of the input to the nearest color in the selected palette. Useful for limiting color in pixel art and for artistic effects.

Optional dithering may be applied with the covered value range controlled by the spread value.

Colorspace Modes

- **Lab:** finds nearest colors in Lab colorspace
- **RGB:** finds nearest colors in RGB colorspace

Dithering Threshold Modes

- **Pattern:** uses the lightness or alpha value of the selected pattern to threshold the input color between palette colors
- **Noise:** uses a randomly generated value per pixel to threshold the input color between palette colors

Dithering Color Modes

- **Per-Component Offset:** independently offsets each color channel by the threshold amount, scaled by the offset scale value
- **Nearest Colors:** finds the two nearest colors then applies the threshold amount to the relative distances of the two color to find the resulting color

Dithering Alpha Modes

- **Clip:** thresholds alpha at the clip position
- **Index:** uses the selected palette index as the transparent color
- **Dither:** applies dither to the alpha value as per Dithering Threshold

Modes

Other

Filters signified by them not fitting anywhere else.

Wave

Adds a cute little wave-distortion effect to the input image.

Random Noise

Gives Random Noise to input image.

Random Pick

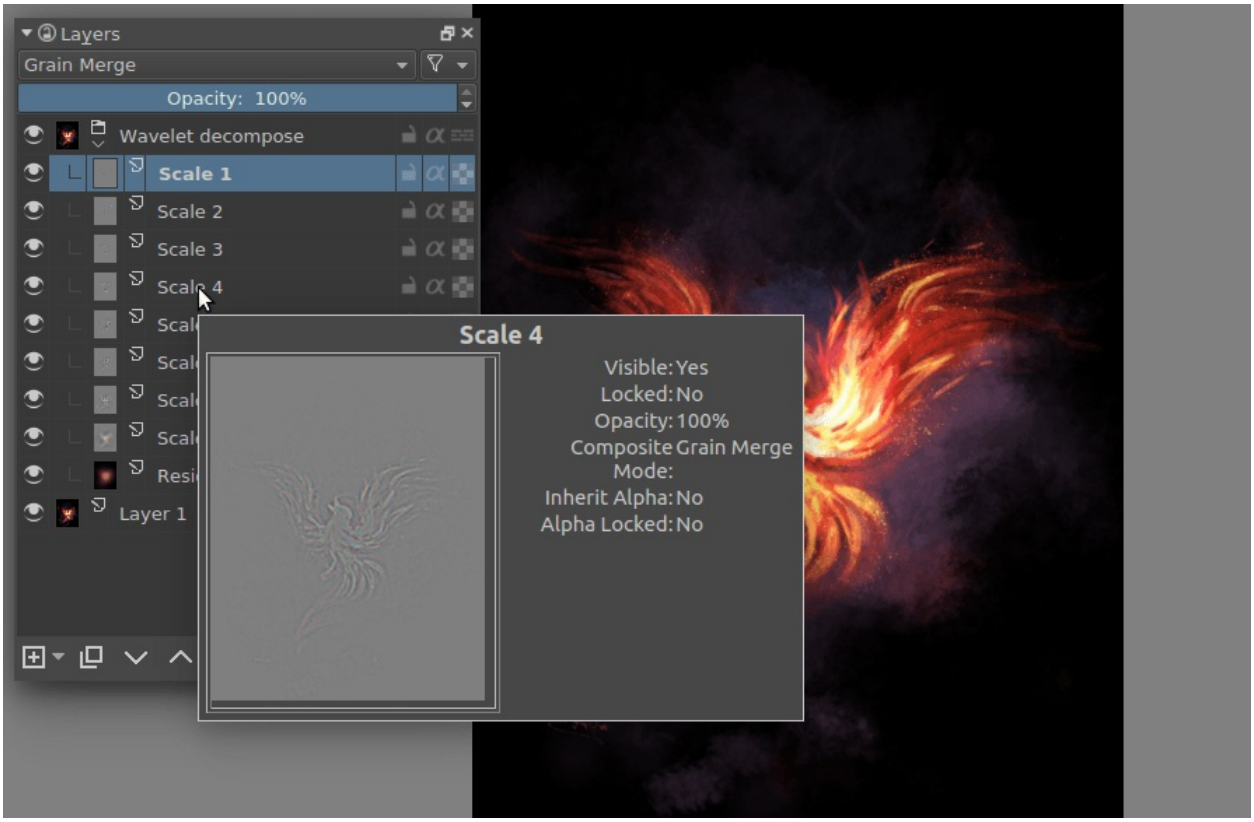
Adds a little pixely-fringe to the input image.

Wavelet Decompose

Wavelet decompose uses wavelet scales to turn the current layer into a set of layers with each holding a different type of pattern that is visible within the image. This is used in texture and pattern making to remove unwanted noise quickly from a texture.

You can find it under *Image*.

When you select it, it will ask for the amount of wavelet scales. More scales, more different layers. Press *OK*, and it will generate a group layer containing the layers with their proper blending modes:



Adjust a given layer with middle gray to neutralize it, and merge everything with the *Grain Merge* blending mode to merge it into the end image properly.

HDR Display

New in version 4.2.

Note

Currently only available on Windows.

Since 4.2 Krita can not just edit high bitdepths images, but also render them on screen in a way that an HDR capable setup can show them as HDR images. HDR images, to put it simply, are images with really bright colors. They do this by having a very large range of colors available, 16 bit and higher, and to understand the upper range of the available colors as brighter than the brightest white most screens can show. HDR screens, in turn, are screens which can show brighter colors than most screens can show, and can thus show the super-bright colors in these HDR images. This allows for images where bright things, like fire, sunsets, magic, look really spectacular! It also shows more subtle shadows and has a better contrast in lower color values, but this requires a sharper eye.

Configuring HDR

Krita cannot show HDR with any given monitor, you will need an HDR capable setup. HDR capable setups are screens which can show more than 100 nits, preferably a value like 1000 and can show the rec 2020 PQ space. You will need to have the appropriate display cable(otherwise the values are just turned into regular SDR) and a graphics card which supports HDR, as well as suitable drivers. You then also need to configure the system settings for HDR.

If you can confirm that the system understands your setup as an HDR setup, you can continue your configuration in Krita, in *Settings* ▸ *Configure Krita…* ▸ *Display*. There, you need to select the preferred surface, which should be as

close to the display format as possible. Then restart Krita.

Painting in HDR

To create a proper HDR image, you will need to make a canvas using a profile with rec 2020 gamut and a linear TRC. *Rec2020-elle-V4-g10.icc* is the one we ship by default.

HDR images are standardized to use the Rec2020 gamut, and the PQ TRC. However, a linear TRC is easier to edit images in, so we don't convert to PQ until we're satisfied with our image.

For painting in this new exciting color space, check the Scene Linear Painting page, which covers things like selecting colors, gotchas, which filters work and cool workflows.

Exporting HDR

Now for saving and loading.

The KRA file format can save the floating point image just fine, and is thus a good working file format.

For sharing with other image editors, *.exr is recommended. For sharing with the web we currently only have HDR PNG export, but there's currently very little support for this standard. In the future we hope to see heif and avif support.

For exporting HDR animations, we support saving HDR to the new codec for mp4 and mkv: H.265. To use these options…

- Get a version of FFmpeg that supports H.265.
- Have an animation open.
- *File* ▸ *Render Animation*.
- Select *Video*.
- Select for *Render as*, 'MPEG-4 video' or 'Matroska'.
- Press the configure button next to the file format dropdown.

- Select at the top ‘H.265, MPEG-H Part 2 (HEVC)’.
- Select for the *Profile*, ‘main10’.
- *HDR Mode* should now enable. Toggle it.
- click *HDR Metadata* to configure the HDR metadata (options described below).
- finally, when done, click ‘render’.

HDR Metadata

This is in the render animation screen. It configures the SMPTE ST.2086 or Master Display Color Volumes metadata and is required for the HDR video to be transferred properly to the screen by video players and the cable.

Master Display
: The colorspace characteristics of the display on for which your image was made, typically also the display that you used to paint the image with. There are two default values for common display color spaces, and a custom value, which will enable the *Display* options.

Display
: The precise colorspace characteristics for the display for which your image was made. If you do not have custom selected for *Master Display*, these are disabled as we can use predetermined values.

  Red/Green/Blue Primary
  : The xyY x and xyY y value of the three chromacities of your screen. These define the gamut.

  White Point
  : The xyY x and xyY y value of the white point of your screen, this defines what is considered ‘neutral grey’.

  Min Luminance
  : The darkest value your screen can show in nits.

  Max Luminance
  : The brightest value your screen can show in nits.

MaxCLL

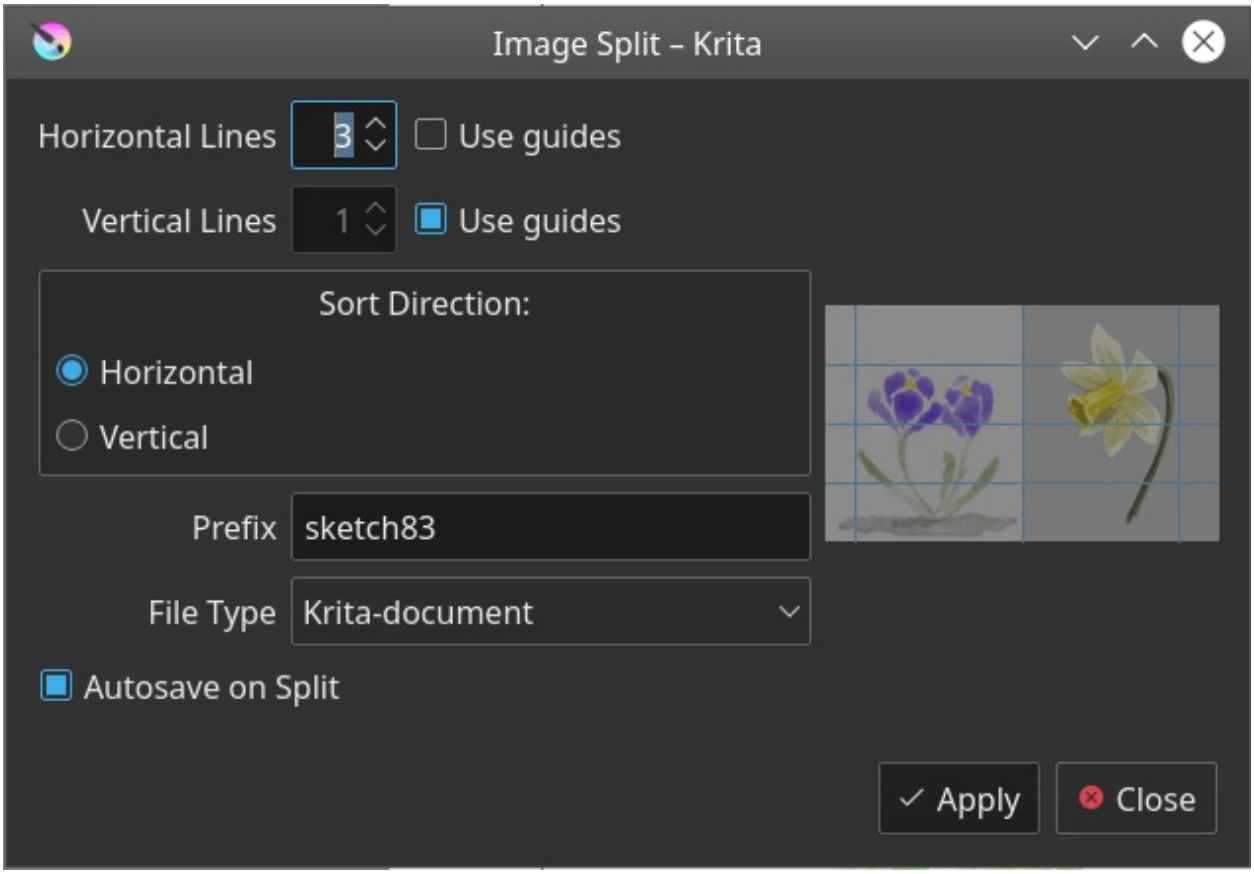
The value of the brightest pixel of your animation in nits.

MaxFALL

The average ‘brightest value’ of the whole animation.

Image Split

Found under *Image* ▸ *Image Split*, the Image Split function allows you to evenly split a document up into several sections. This is useful for splitting up spritesheets for example.



Horizontal Lines
: The amount of horizontal lines to split at. 4 lines will mean that the image is split into 5 horizontal stripes.

Vertical Lines
: The amount of vertical lines to split at. 4 lines will mean that the image is split into 5 vertical stripes.

New in version 4.3:

Use Guides
: Instead of splitting the image up into even parts, you can choose to use the image guides to function as horizontal or vertical lines. This provides a little bit more control on how the image is split.

Sort Direction

New in version 4.2.

Whether to number the files using the following directions:

Horizontal
: Left to right, top to bottom.

Vertical
: Top to bottom, left to right.

Prefix
: The prefix at which the files should be saved at. By default this is the current document name.

File Type
: Which file format to save to.

Autosave on split
: This will result in all slices being saved automatically using the above prefix. Otherwise Krita will ask the name for each slice.

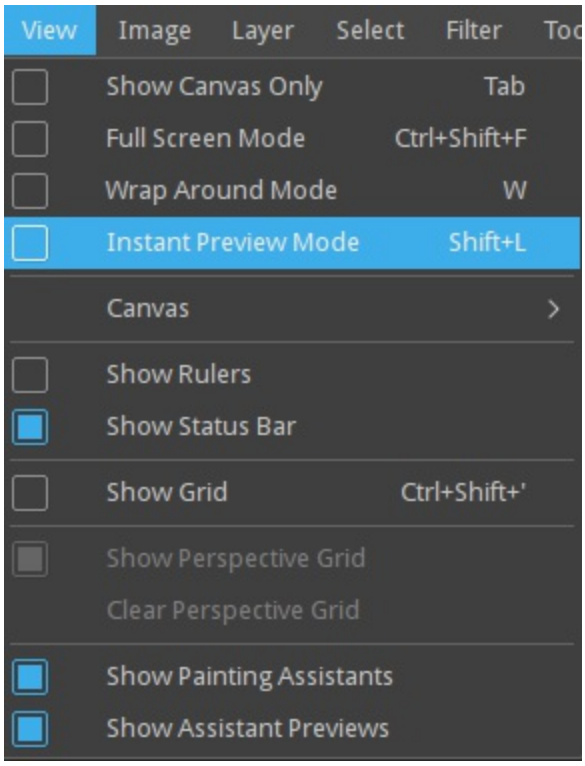
Instant Preview

Instant Preview (previously known under the code name Level Of Detail/LOD strokes) is Krita's special speed-up mechanism that was funded by the 2015 Kickstarter. Krita slows down with really large images due to the large amount of data it's crunching in painting these images. Instant Preview works by taking a smaller version of the canvas, and drawing the feedback on there while Krita calculates the real stroke in the background. This means that if you have a 4k screen and are working on a 4k image at 100% zoom, you won't feel any speed up.

Activating Instant Preview

Warning

Instant Preview requires OpenGL 3.0 support at minimum. So if you don't have *high-quality* scaling available in *Settings* ▸ *Configure Krita…* ▸ *Display* ▸ *Display scaling filter*, then you won't be able to use Instant Preview either.



The Global Instant Preview toggle is under the view menu.

Instant Preview is activated in two places: The view menu (`Shift + L` shortcut), and the settings of the given paintop by default. This is because Instant Preview has different limitations with different paint operations.

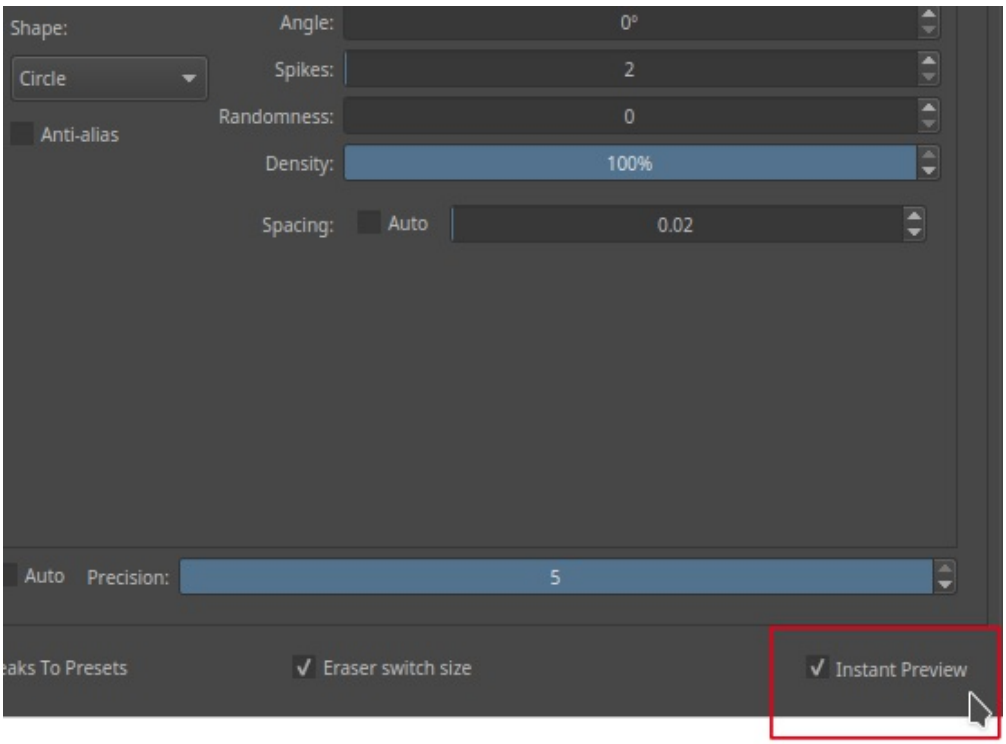
For example, the overlay mode in the color smudge brush will disable the ability to have Instant Preview on the brush, so does using ‘fade’ sensor for size.

Similarly, the auto-spacing, fuzzy sensor in size, use of density in brush-tip and the use of texture paintops will make it more difficult to determine a stroke, and thus will give a feeling of ‘popping’ when the stroke is finished.

When you check the brush settings, the Instant Preview checkbox will have a * behind it. Hovering over it will give you a list of options that are affecting the Instant Preview mode.

New in version 4.0:  this pop-up will give a slider, which can be used to determine the threshold size at which instant preview activates. By default

this 100px. This is useful for brushes that are optimised to work on small sizes.



The Instant Preview checkbox at the bottom of the brush settings editor will give you feedback when there's settings active that can't be previewed right. Hover over it to get more detail. In this case, the issue is that auto-spacing is on.

Tools that benefit from Instant Preview

The following tools benefit from Instant Preview:

- The Freehand brush tool.
- The geometric tools.
- The Move Tool.
- The Filters.
- Animation.

Krita 4 Preset Bundle Overview

Krita comes with a large collection of brush presets. This collection was designed with many considerations:

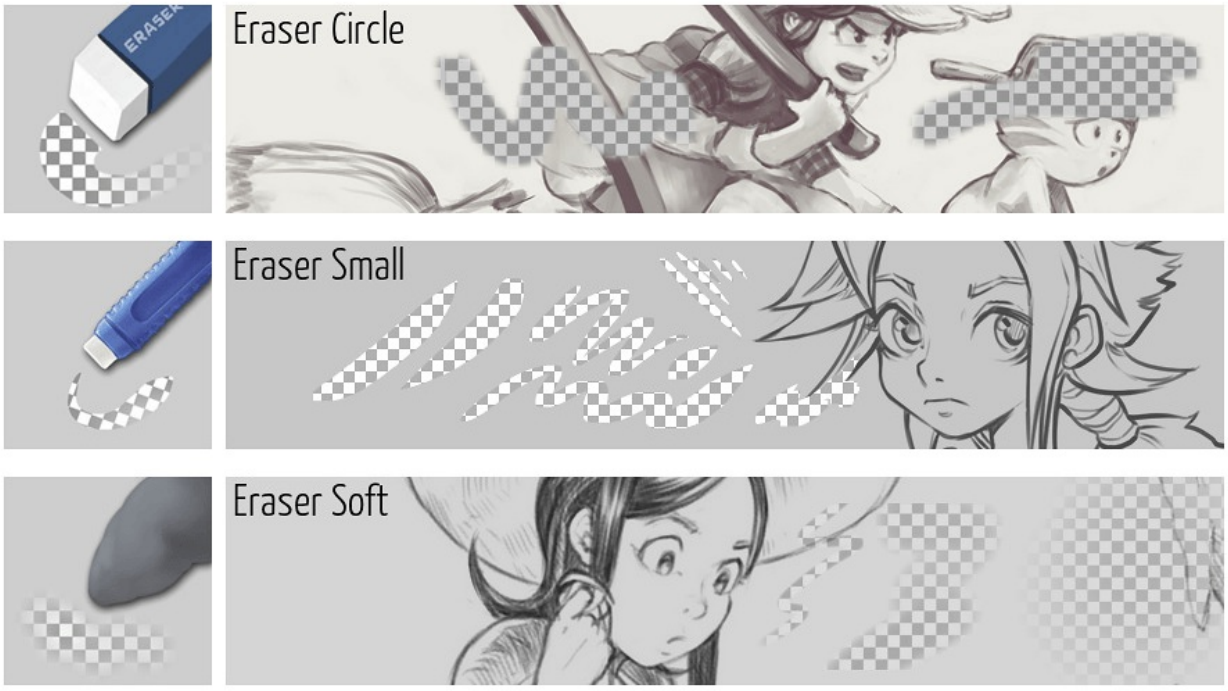
- Help the beginner and the advanced user with brushes that are ready-to-use.
- Propose tools for the various ways Krita is used: Comic inking and coloring, Digital Painting, Mate Painting, Pixel Art, 3D texturing.
- Show a sample of what the brush engines can do.

This page illustrates and describes the included default brush presets in Krita 4.

Erasers

- The large one is for removing large portions of a layer (eg. a full character).
- The small one is designed to use when drawing thin lines or inking. It has a very specific shape so you will notice with the square shape of your cursor you are in eraser-mode.
- The soft one is used to erase or fade out the part of a drawing with

various levels of opacity.

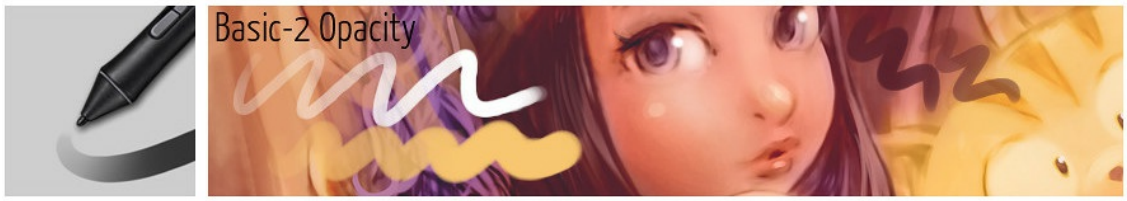


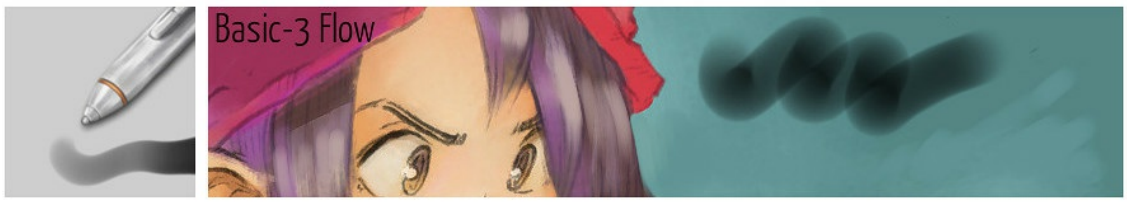
Basics

The basic brush family all use a basic circle for the brush tip with a variation on opacity, flow or size. They are named Basic because brushes of this type are the fundamental stones of every digital painting program. These brushes will work fast since they use simple properties.


Airbrush Soft

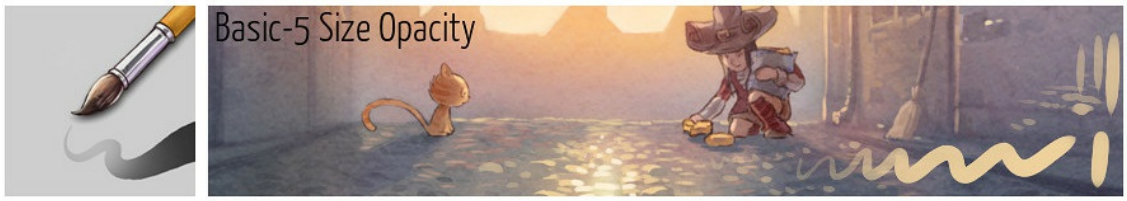

Basic-1


Basic-2 Opacity


Basic-3 Flow


Basic-4 Flow Opacity


Basic-5 Size default


Basic-5 Size Opacity


Basic-6 Details

Pencils

These presets tends to emulate the effect of pencil on paper. They all have a thin brush that uses a paper-texture. Some focus on being realistic to help with correcting a pencil scan. Some focus more on showing the effects on your computer monitor. The two last (Tilted/Quick Shade) assist the artist to obtain specific effects; like quickly shading a large area of the drawing without having to manually crosshatch a lot of lines.

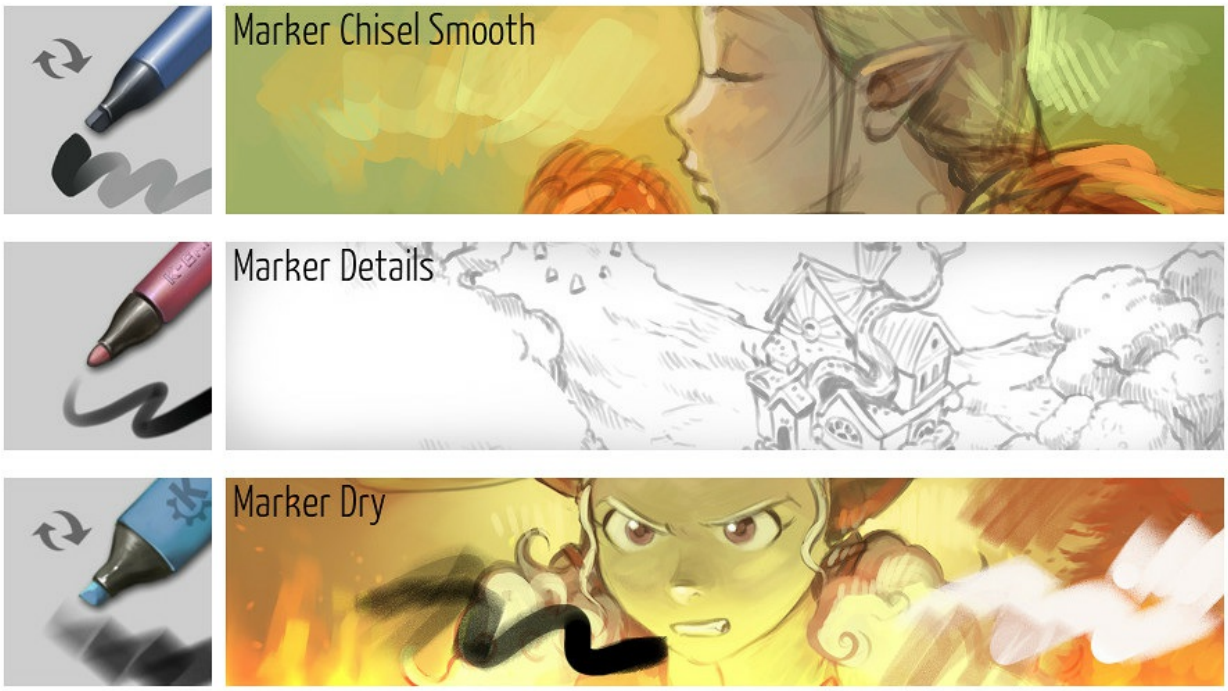
Inking

For the black & white illustrator or the comic artist. The Inking brushes help you produce line art and high contrast illustrations.

- Ink Precision: A thin line designed to take notes or draw tiny lines or details.
- Ink Fineliner: A preset with a regular width to trace panels, technical details, or buildings.
- Ink GPen: A preset with a dynamic on size to ink smoothly.
- Ink Pen Rough: A preset for inking with a focus on having a realistic ink line with irregularities (texture of the paper, fiber of paper absorption).
- Ink Brush Rough: A brush for inking with also a focus on getting the delicate paper texture appearing at low pressure, as if the brush slightly touch paper.
- Ink Sumi-e: A brush with abilities at revealing the thin texture of each bristle, making the line highly expressive.

Markers

A small category with presets simulating a marker with a slight digital feeling to them.

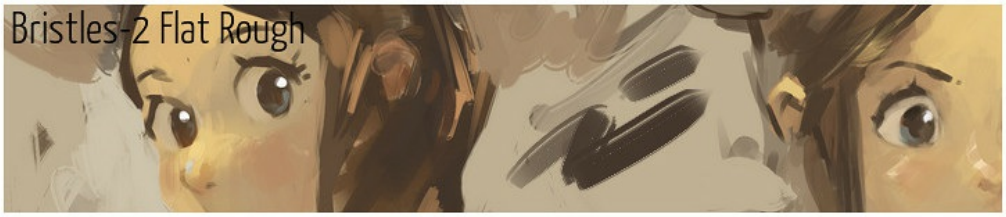


Dry Painting

The Dry Painting category is full set of brushes that appear like bristles. They do not interact with the color already on the canvas; that’s why they are called “dry”. They work as if you were painting on a dry artwork: the color replace, or overlay/glaze over the previous painting stroke. This brush emulates techniques that dry quickly as tempera or acrylics.

Bristles-1 Details

Bristles-2 Flat Rough



Bristles-3 Large Smooth

Bristles-4 Glaze

Bristles-5 Flat

Charcoal Rock Soft

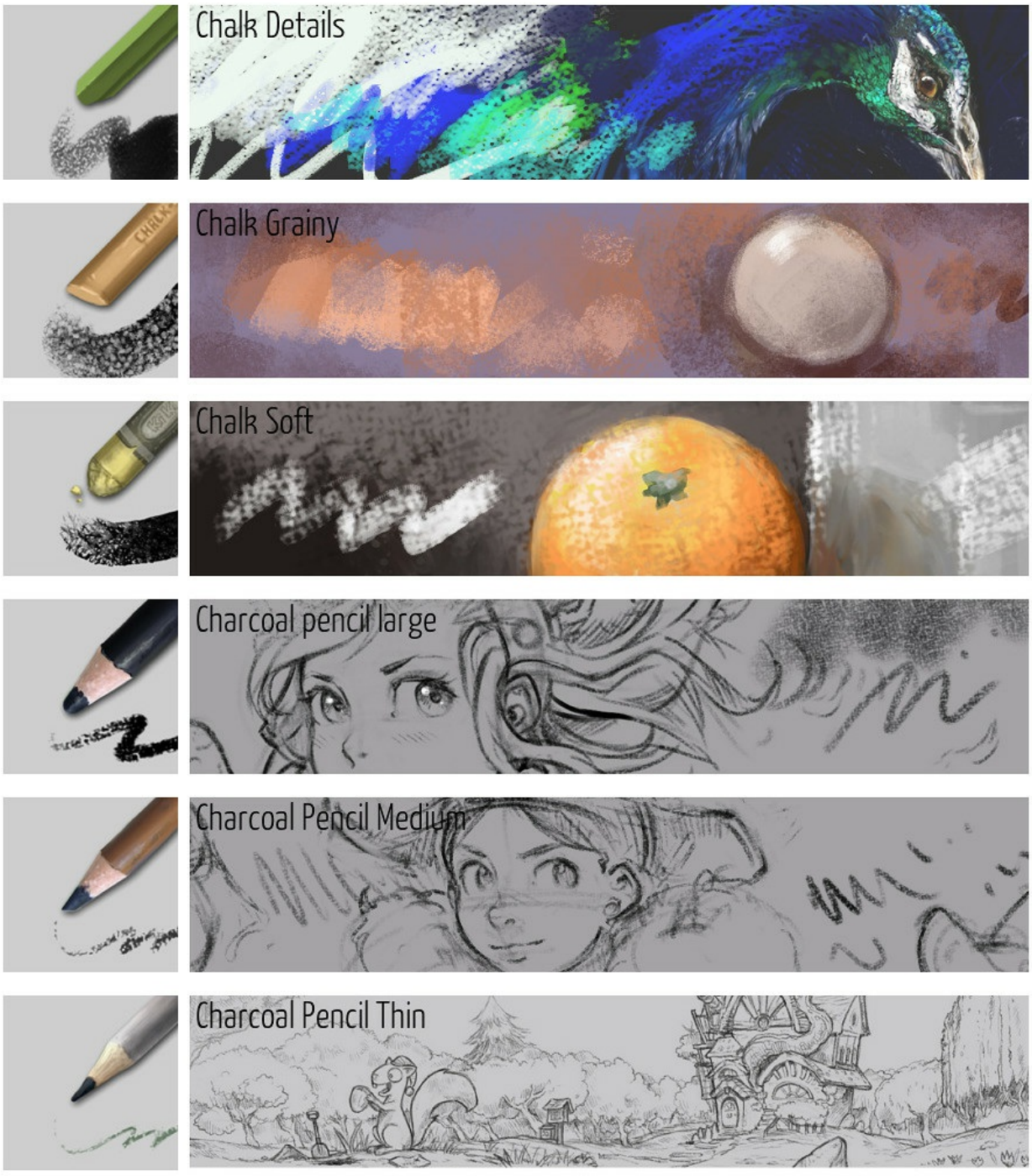
Dry Roller

Dry Painting Textured

Almost the same family as the previous one, except these brush presets lay down a textured effect. They simulate the painting effect you can obtain with very thick painting on a brush caressing a canvas with fabric texture. This helps to build painterly background or add life in the last bright touch of colors.

Chalk, Pastel and Charcoal

Still part of the dry family. These brushes focus on adding texture to the result. The type of texture you would obtain by using a dry tool such as chalk, charcoal or pastel and rubbing a textured paper.



Wet painting

This family of brushes is wet in a sense they all interact with the color on the canvas. It triggers the feeling of having a wet artwork and mixing color at the same time. The category has variations with bristle effects or simple rounded

brushes.


Wet Bristles


Wet Bristles Rough


Wet Circle


Wet Knife


Wet Paint Details

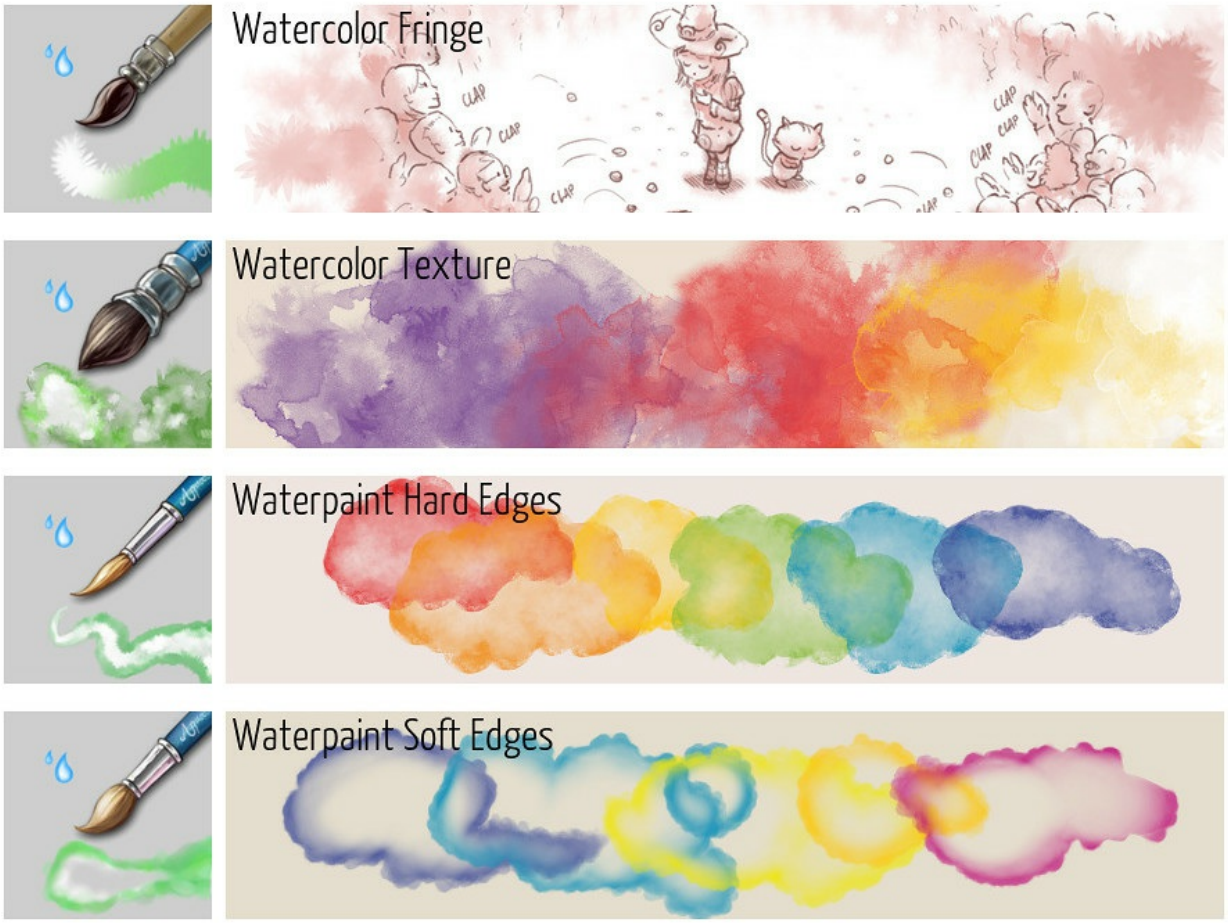

Wet Paint


Wet Smear


Wet Textured Soft

Watercolors

Simulating real watercolors is highly complex. These brushes only partially simulate the watercolor texture. Don’t expect crazy pigment diffusion because these brushes are not able to do that. These brushes are good at simulating a fringe caused by the pigments and various effects.

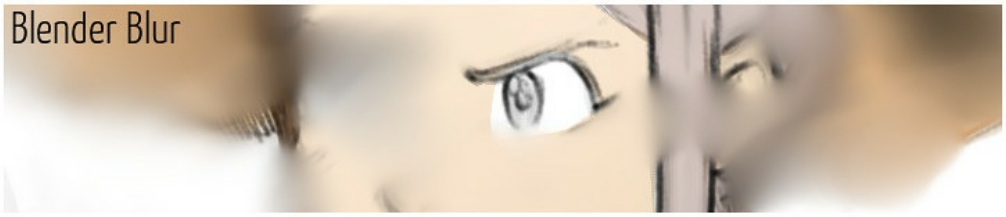


Blender

These brushes don’t paint any colors. They interact with the color you already have on the canvas. Don’t expect them to have any effect on a white page. All these presets give a different result with how they smudge or smear. It helps to blend colors, blur details, or add style on a painting. Smearing pixels can help with creating smoke and many other effects.

Blender Basic

Blender Blur



Blender Knife Edge

Blender Pixelize

Blender Rake

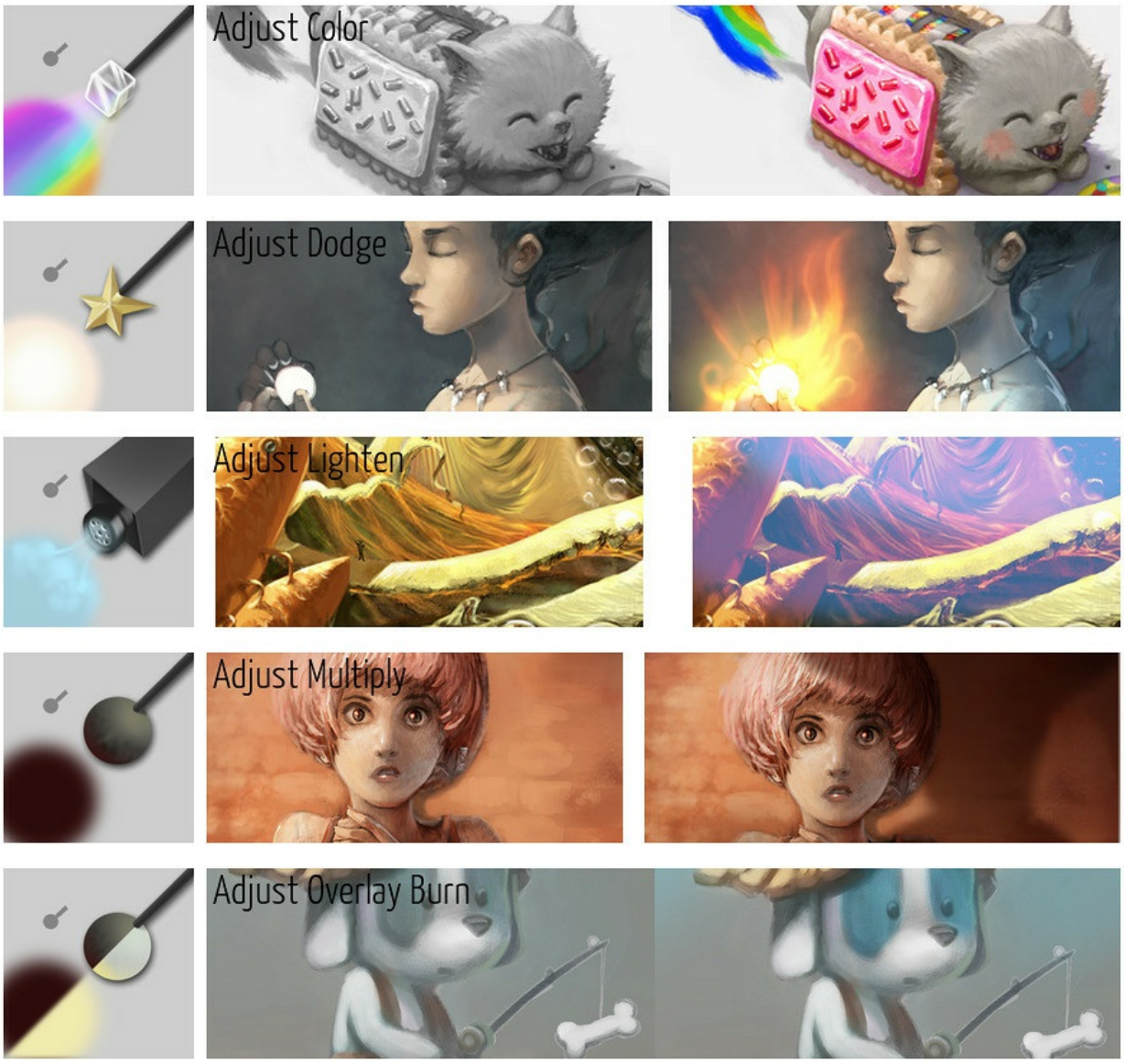
Blender Smear

Blender Textured Soft

Adjustments

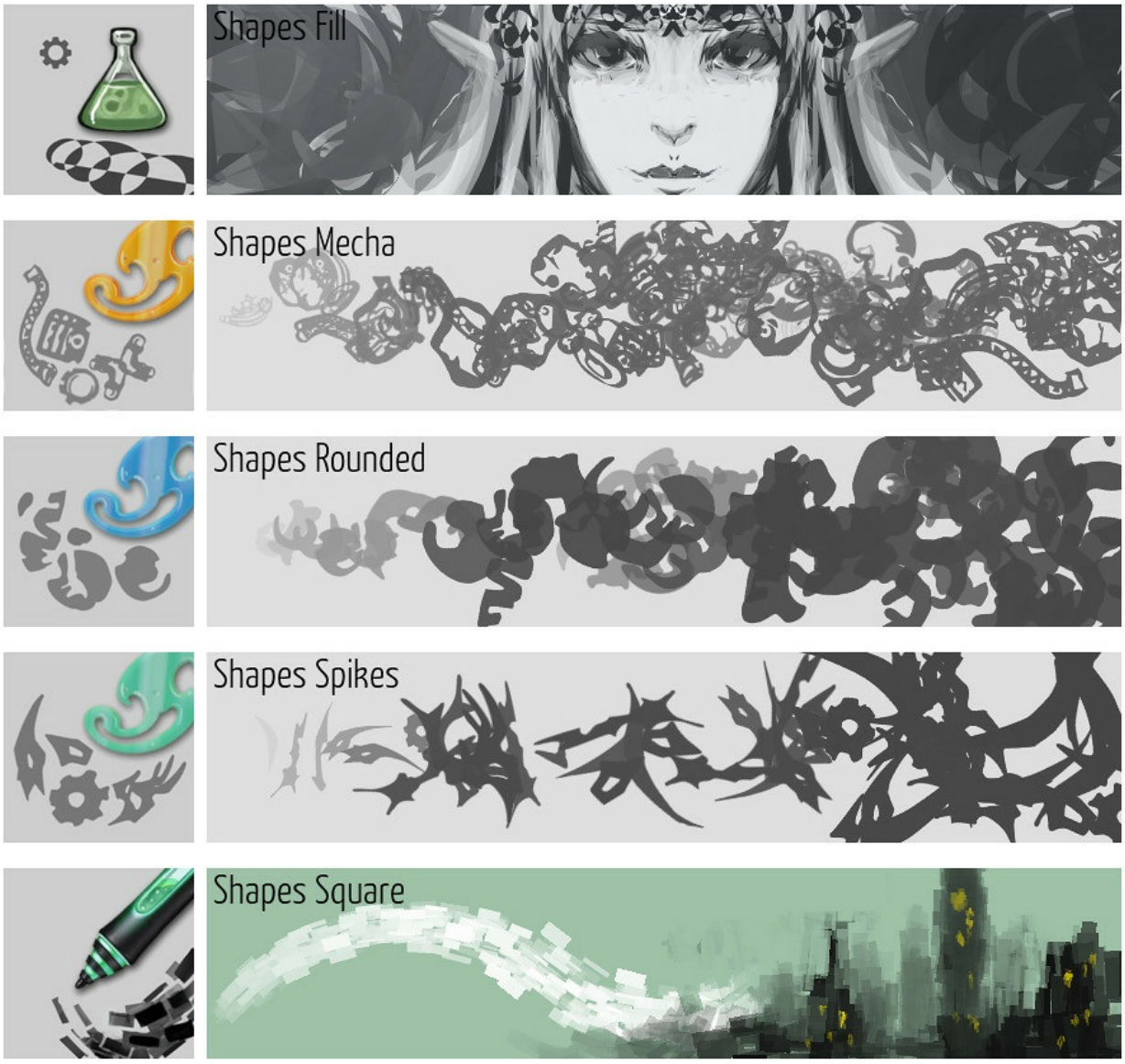
This family of airbrushes has variations on the blending modes. Different blending modes will give different results depending on the effect you are trying to achieve.

- Color - Can help to re-color or desaturate a part of your artwork. It changes only the hue and saturation, not the value, of the pixels.
- Dodge - Will assist you in creating effects such as neon or fire.
- Lighten - Brightens only the area with the selected color: a good brush to paint depth of field (sfumato) and fog.
- Multiply - Darkens all the time. A good brush to create a quick vignette effect around an artwork, or to manage big part in shadow.
- Overlay - Burn helps to boost the contrast and overlay color on some areas.



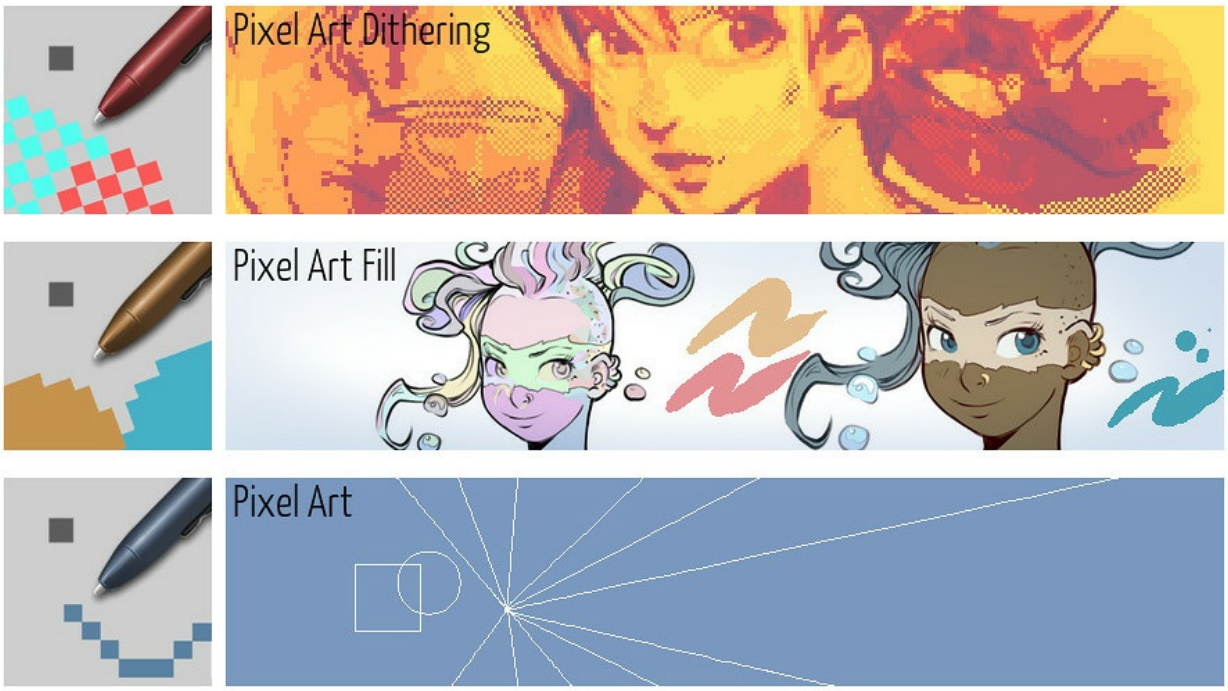
Shapes

Painting with ready-made shapes can help concept artists create happy-accidents and stimulate the imagination. The Shape Fill tool is a bit specific: you can draw a silhouette of shape and Krita fills it in real time. Painting shapes over an area helps fill it with random details. This is useful before painting over with more specific objects.



Pixel

You might believe this section is specific to pixel-artist, but in many situations dealing with specific pixels are needed to make corrections and adjustments even on normal paintings. A thin 1px brush can be used to trace guidelines. A brush with aliasing is also perfect to fix the color island created by the Coloring-mask feature.



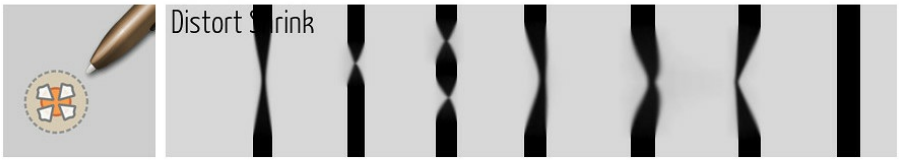
Experimental

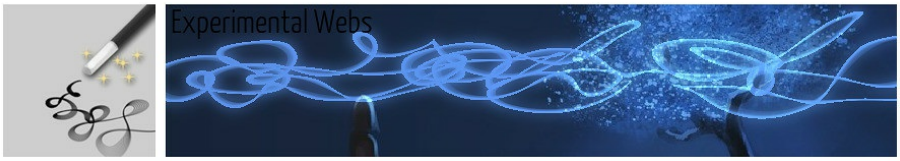
When categorizing brushes, there is always a special or miscellaneous category. In this family of brushes you'll find the clone brush along with brushes to move, grow, or shrink specific areas.

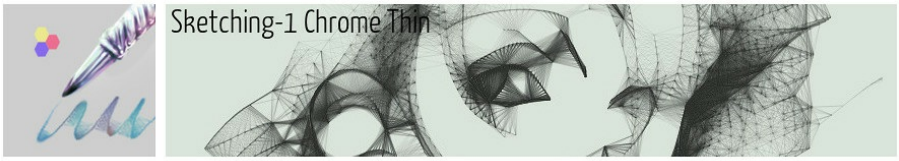

Clone Tool


Distort Grow


Distort Move


Distort Shrink

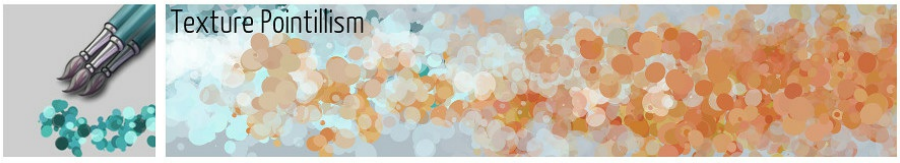

Experimental Webs


Sketching-1 Chrome Thin


Sketching-2 Chrome Large

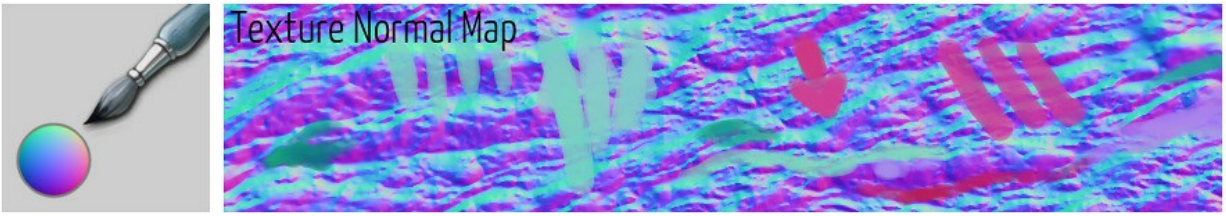

Sketching-3 Leaky


Texture Impressionism


Texture Pointillism

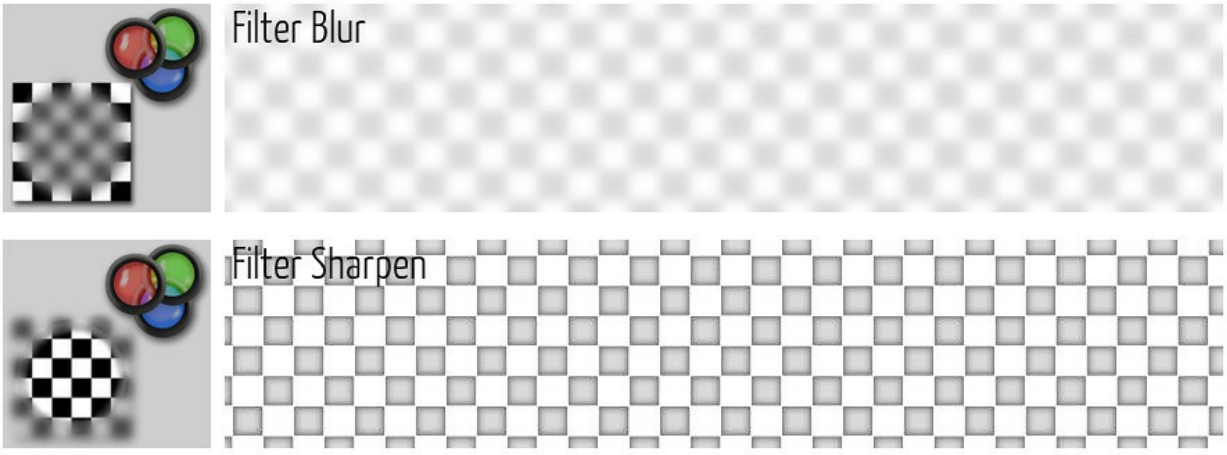
Normal Map

Useful for 3D programs and texture artists. If your tablet supports tilting and rotation this brush will allow you to paint on your normal map using your brush rotation and orientation. You can “sculpt” your details in the texture with the different colors. Each color will map to an angle that is used for 3D lighting. It works well on pen-tablet display (tablet with a screen) as you can better sync the rotation and tilting of your stylus with the part of the normal map you want to paint.



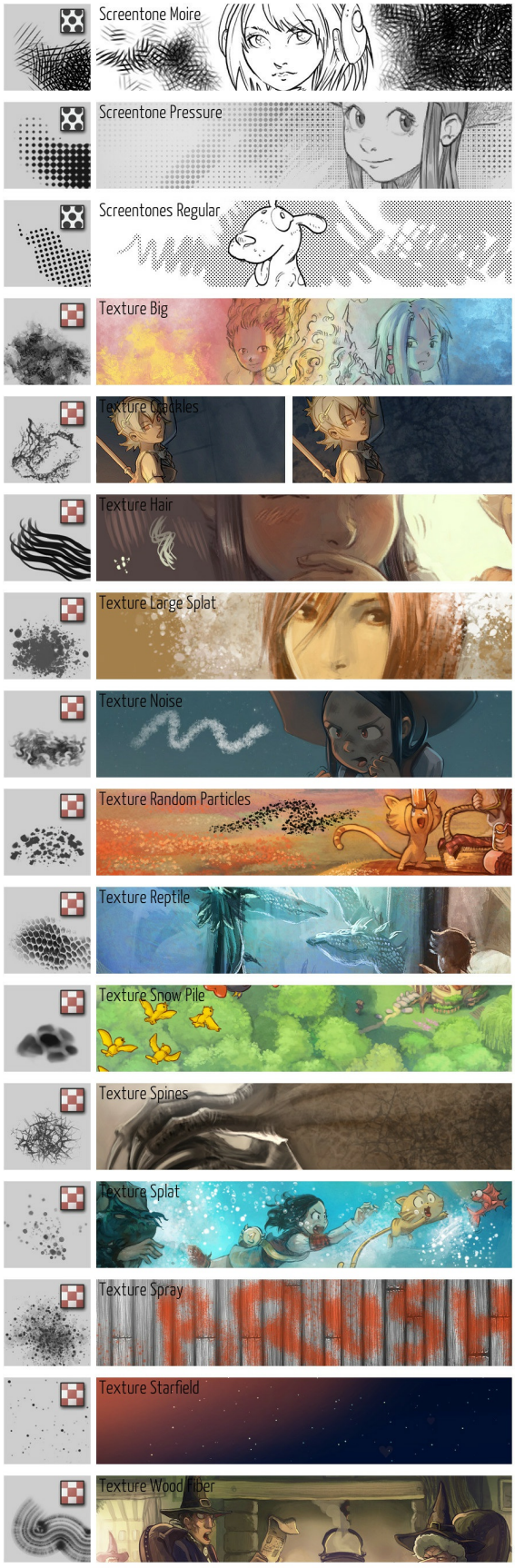
Filters

Krita can apply many of its filters on a brush thanks to the filter brush engine. The result is usually not efficient and slow, but a good demo of the ability of Krita.



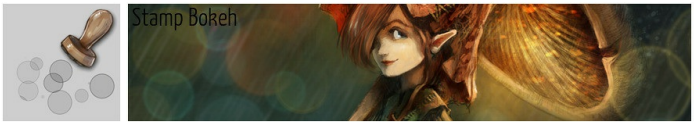
Textures

Adding textures is not only useful for the 3D artist or video-game artist: in many artworks you'll save a lot of time by using brushes with random patterns.

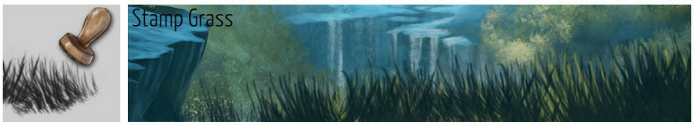

Screentone Moire
Screentone Pressure
Screentones Regular
Texture Big
Texture Crackles
Texture Hair
Texture Large Splat
Texture Noise
Texture Random Particles
Texture Reptile
Texture Snow Pile
Texture Spines
Texture Splat
Texture Spray
Texture Starfield
Texture Wood Fiber

Stamps

The stamps are a bit similar to the texture category. Stamps often paint a pattern that is easier to recognize than if you tried to paint it manually. The results appear more as decorations than for normal painting methods.


Stamp Bokeh


Stamp Floor


Stamp Grass


Stamp Grass Patch

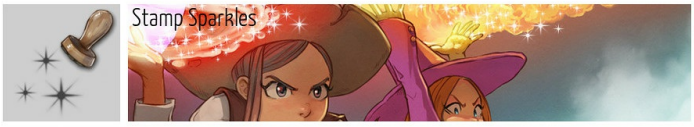

Stamp Hearts


Stamp Herbals


Stamp Leaves


Stamp Mountains Distant


Stamp Shoujo Bubbles


Stamp Sparkles


Stamp Stylised Tree


Stamp Vegetal


Stamp Water

Layers and Masks

Layers are a central concept in digital painting.

With layers you can get better control over your artwork, for example you can color an entire artwork just by working on the separate color layer and thereby not destroying the line art which will reside above this color layer.

Furthermore, layers allow you to change the composition easier, and mass transform certain elements at once.

Masks on the other hand allow you to selectively apply certain effects on a layer, like transparency, transformation and filters.

Check the Introduction to Layers and Masks for more information.

- Clone Layers
- File Layers
- Fill Layers
- Filter Layer
- Filter Masks
- Group Layers
- Layer Styles
- Paint Layers
- Selection Masks
- Split Alpha
- Transformation Masks
- Transparency Masks
- Vector Layers

Clone Layers

A clone layer is a layer that keeps an up-to-date copy of another layer. You cannot draw or paint on it directly, but it can be used to create effects by applying different types of layers and masks (e.g. filter layers or masks).

Example uses of Clone Layers

For example, if you were painting a picture of some magic person and wanted to create a glow around them that was updated as you updated your character, you could:

1. Have a Paint Layer where you draw your character
2. Use the Clone Layer feature to create a clone of the layer that you drew your character on
3. Apply an HSV filter mask to the clone layer to make the shapes on it white (or blue, or green etc.)
4. Apply a blur filter mask to the clone layer so it looks like a “glow”.

As you keep painting and adding details, erasing on the first layer, Krita will automatically update the clone layer, making your “glow” apply to every change you make.

Changing the source of Clone Layers

You can change the source of one or more Clone Layers in the Layers Docker. To do so, select one or more Clone Layers in the docker (hold `Ctrl` or `Shift` and left-click the layers). Then, right-click on any selected layer. In the context menu, there is an action named *Set Copy From*. Click it. A dialog will pop up and there is a drop-down menu with all possible layers to be set as the source of all selected Clone Layers. If the current source of them consists of multiple layers, the default activated selection in the drop-down menu will be blank. Otherwise, it would be the common source of selected Clone Layers.

Possible target layers are determined through the following criteria:

1. Any Clone Layer that is selected is invalid.
2. A parent or clone of any invalid layer is invalid.
3. All other layers are valid.

If you select one layer in the drop-down menu, a preview of the canvas will be shown. Click *OK* to apply the changes. Click *Cancel* to discard the changes. If you make changes to the image outside the dialog, the changes will be applied and the dialog will be automatically closed.

File Layers

File Layers are references to files outside of the document: If the referenced document updates, the file layer will update. Do not remove the original file on your computer once you add it to Krita. Deleting your original image will break the file layer. If Krita cannot find the original file, it'll ask you where to find it. File layers cannot display animations.

File Layers have the following scaling options:

No Scaling
: This'll import the file layer with the full pixel-size.

Scale to Image Size
: Scales the file layer to fit exactly within the canvas boundaries of the image.

Adapt to image resolution
: If the imported layer and the image have a different resolution, it'll scale the filelayer by scaling its resolution. In other words, import a 600dpi A4 image onto a 300dpi A4 image, and the filelayer will be scaled to fit precisely on the 300dpi image. Useful for comics, where the ink-layer is preferred to be at a higher resolution than the colors.

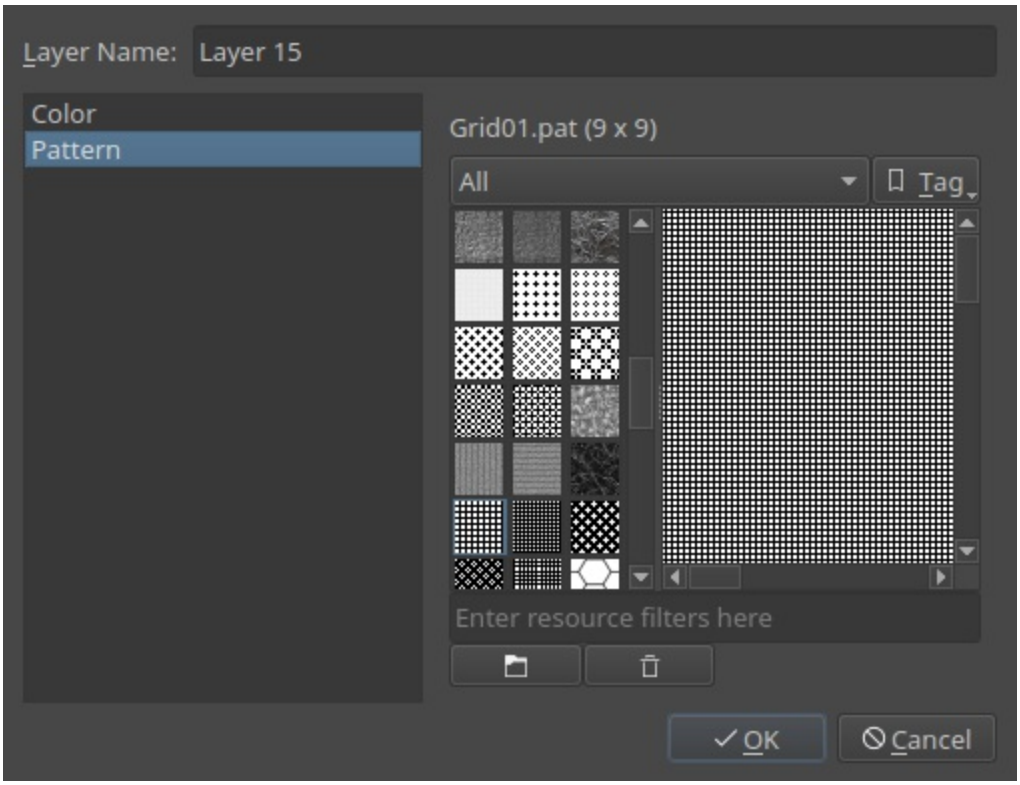
File Layers can currently not be painted on. If you want to transform a file layer, you need to apply a transformation mask to it and use that.

New in version 3.3: In the layerdocker, next to the file layer only, there's a little folder icon. Pressing that will open the file pointed at in Krita if it hadn't yet. Using the properties you can make the file layer point to a different file.

New in version 4.0: You can turn any set of layers into a file layer by right-clicking them and doing *Convert ▸ to File Layer*. It will then open a save prompt for the file location and when done will save the file and replace the layer with a file layer pointing at that file.

Fill Layers

A Fill Layer is a special layer that Krita generates on-the-fly that can contain either a pattern or a solid color.



By default, the dialog selects the flat color fill. This fills the layer with a singular color. Since version 4.2, newly created colored fill layers will be assigned to the currently active foreground color.

However, there are many more options, with more complex features:

- Gradient Fill
- Multigrid
- Pattern Fill
- Screentone
- SeExpr
- Simplex Noise

Painting on a fill layer

A fill-layer is a single-channel layer, meaning it only has transparency. Therefore, you can erase and paint on fill-layers to make them semi-opaque, or for when you want to have a particular color only. Being single channel, fill-layers are also a little bit less memory-consuming than regular 4-channel paint layers.

Gradient Fill

New in version 4.4.2.

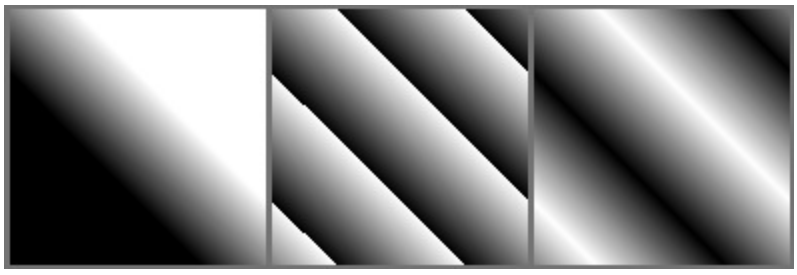
Fills the layer with a gradient in the same way as the gradient tool does.

General Options

Shape:

Linear

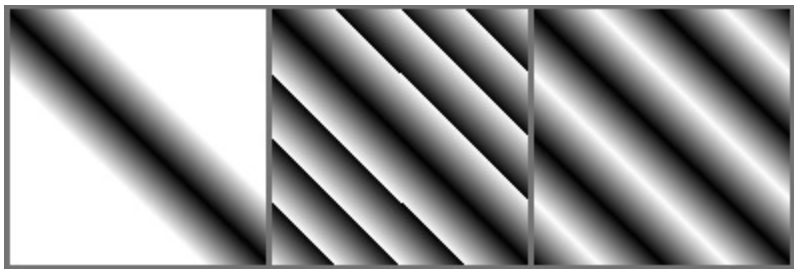
This will draw a straight gradient. The distance and angle between the start and end points will define the size and rotation of the gradient, respectively.



Left: **None**. Middle: **Forwards**. Right: **Alternating**.

Bilinear

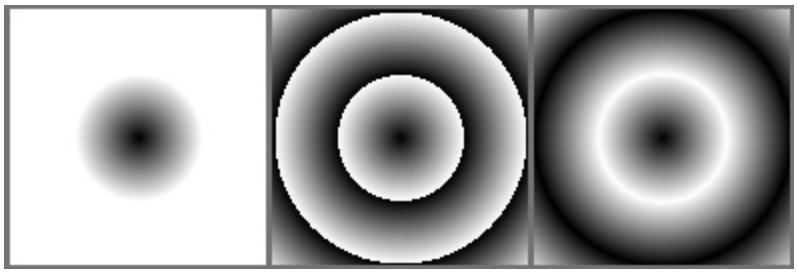
This will draw a straight gradient, mirrored along the axis. The distance and angle between the start and end points will define the size and rotation of the gradient, respectively.



Left: **None**. Middle: **Forwards**. Right: **Alternating**.

Radial

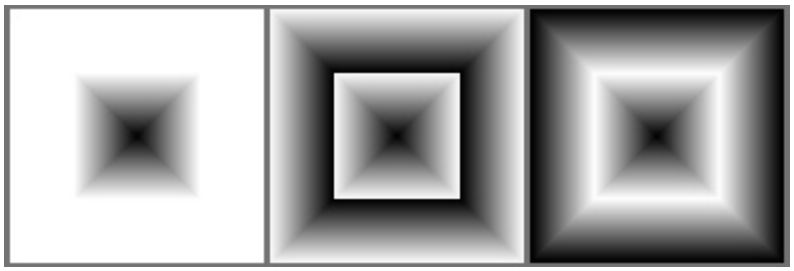
> This will draw the gradient from a center, defined by the start point, with a radius equal to the distance between the start ans end points.



Left: **None**. Middle: **Forwards**. Right: **Alternating**.

Square

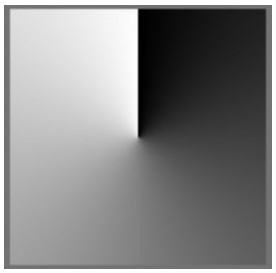
> This will draw the gradient from a center in a square shape, defined by the start point. The distance and angle between the start and end points will define the distance from that center to a side of the square and its rotation, respectively.



Left: **None**. Middle: **Forwards**. Right: **Alternating**.

Conical

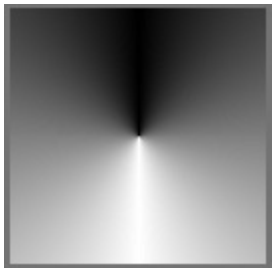
> This will wrap the gradient around a center, defined by the start point. The angle between the start and end points will define where the gradient starts.



Left: **None**. Middle: **Forwards**. Right: **Alternating**.

Conical-symmetric

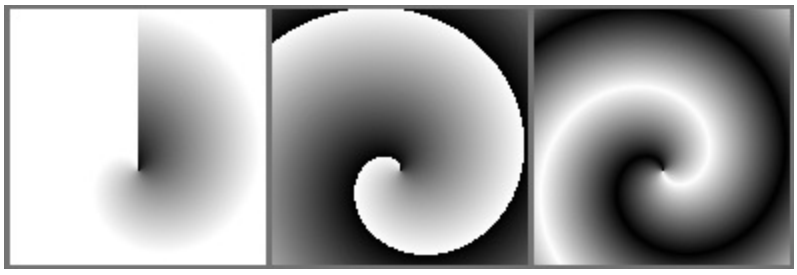
This will wrap the gradient around a center, defined by the start point, but will mirror the wrap once. The angle between the start and end points will define where the gradient starts.



Left: **None**. Middle: **Forwards**. Right: **Alternating**.

Spiral

This will draw the gradient spiral from a center, defined by the start point. The distance and angle between the start and end points will define the distance between loops and the rotation of the spiral, respectively.

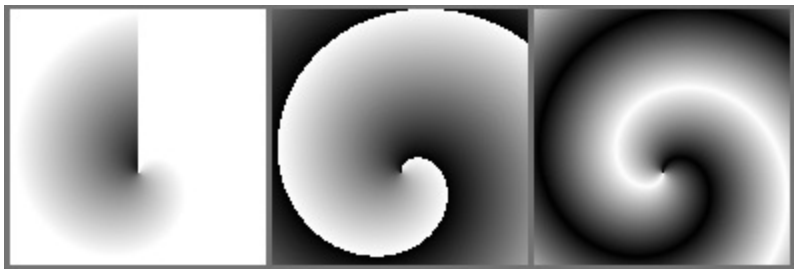


Left: **None**. Middle: **Forwards**. Right: **Alternating**.

Reverse Spiral

This will draw the gradient spiral from a center, defined by the start point, but direction is flipped perpendicular to the line that passes

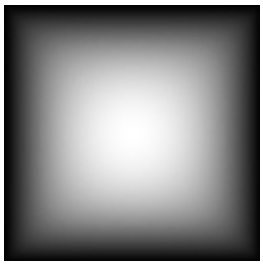
through the star and end points. The distance and angle between the start and end points will define the distance between loops and the rotation of the spiral, respectively.



Left: **None**. Middle: **Forwards**. Right: **Alternating**.

Shaped

This will shape the gradient depending on the image bounds.



Repeat:

None

This will extend the gradient into infinity.

Forward

This will repeat the gradient into one direction.

Alternating

This will repeat the gradient, alternating the normal direction and the reversed.

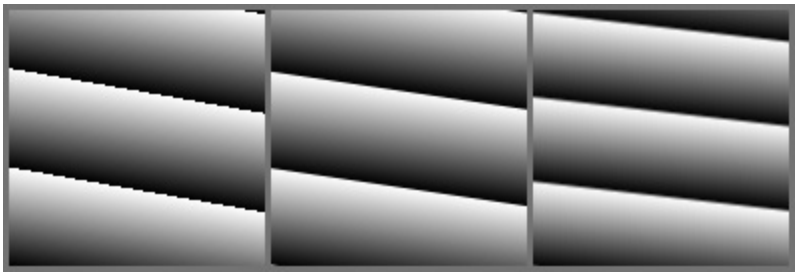
Reverse

Reverses the direction of the gradient.

Antialias threshold

Controls how smooth is the border between repetitions.

- A value equal to 0 means there is no smoothing. The border is aliased.
- A value greater than 0 tells Krita how many pixels to each side of the border should be smoothed.



Left: **0**. Middle: **0.5**. Right: **1**.

Positioning Options

Start
: Allows you to set the start point for the gradient (in the gradient tool this is the point where you first click).

End
: Allows you to set the endpoint for the gradient (in the gradient tool this is the point where you release the mouse button after dragging).

Units
: You can make the values set for the start and end points mean different things by changing the units associated with them:

  Pixels
  : The value indicates a distance in pixels.

  Percent of the width
  : The value indicates a distance as a percentage of the width of the image. So for example, if the image is 1000 pixels wide and 500 pixels high, and the value is 25, then this would be translated to pixels as 250 (25% of the width).

  Percent of the height
  : The value indicates a distance as a percentage of the height of the

image. So for example, if the image is 1000 pixels wide and 500 pixels high, and the value is 25, then this would be translated to pixels as 125 (25% of the height).

Percent of the shortest side

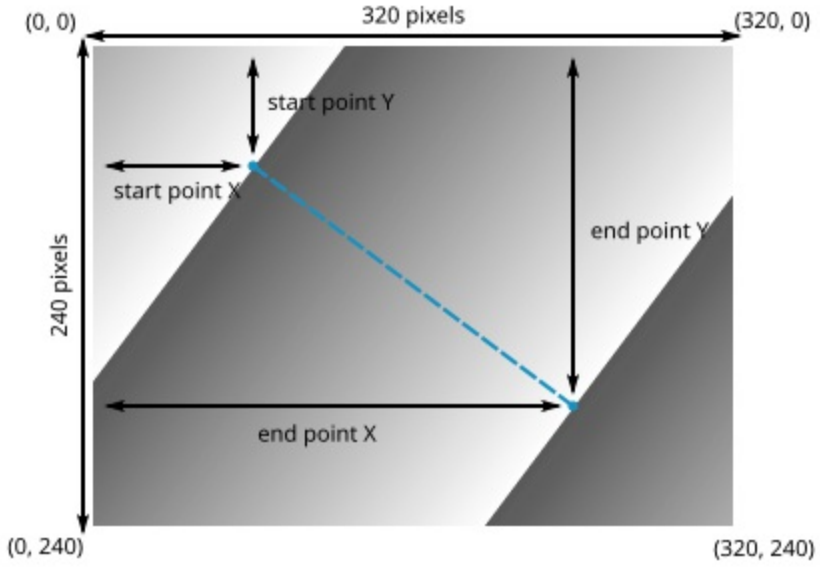
The value indicates a distance as a percentage of the shortest side of the image. So for example, if the image is 1000 pixels wide and 500 pixels high, and the value is 25, then this would be translated to pixels as 125 (25% of the shortest side).

Percent of the longest side

The value indicates a distance as a percentage of the longest side of the image. So for example, if the image is 1000 pixels wide and 500 pixels high, and the value is 25, then this would be translated to pixels as 250 (25% of the longest side).

Values that have percentage units are useful when changing the image size. For example, if you want to have a linear gradient that always goes from the left to the right of the image, you can set the start point to (x = 0 pixels, y = 0 pixels) and the end point to (x = 100% of the width, y = 0 pixels).

On the other hand, if you want a gradient to have the same size regardless of the image size, you should use pixel units.



To make a gradient like the one in the image above, you can set the start position's X coordinate to 80 pixels, 25% of the width, or 25% of the longest side, and its Y coordinate to 60 pixels, 25% of the height, or 25% of the shortest side. Likewise, you could set the end position's X coordinate to 240 pixels, 75% of the width, or 75% of the longest side, and its Y coordinate to 180 pixels, 75% of the height, or 75% of the shortest side.

Keep in mind that if you use percentages the gradient's start and end positions will adapt to the size of the image, while using pixel units will make the gradient's positions be static.

Positioning

You can choose if the end point's coordinates are relative to the start point.

Absolute

The end coordinate indicates a distance from the top-left corner of the image.

Relative

The end coordinate indicates a distance from the start point.

Coordinate System

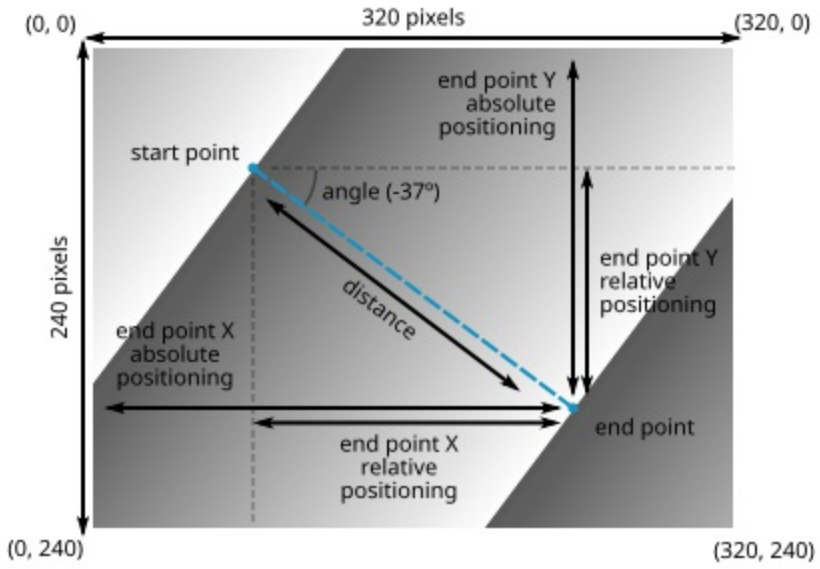
You can set the end point in cartesian or polar coordinates.

Cartesian

The coordinates of the end point are set by establishing horizontal and vertical distances relative to the top-left corner of the image or to the start point.

Polar

The coordinates of the end point are set by establishing an angle and a distance relative to the start point.



To set the end point's position in the gradient on the image above, you could use cartesian coordinates and set the X and Y coordinates relative to the top-left corner of the image (absolute positioning) or relative to the start point (relative positioning).

However, in some cases it is more convenient using polar coordinates and setting the end point's position by establishing an angle and a distance relative to the start point (polar coordinates are always relative to the start point's position).

Gradient Colors

Here you can select the actual colors used by the gradient.

Multigrid

A fill layer based on de Bruijn's 1981 multigrid method to generate Penrose Tilings [https://en.wikipedia.org/wiki/Penrose_tiling] . This generator projects a hyperdimensional grid lattice onto a 2d plane, giving some pretty cool patterns. Besides looking cool, there's a few interesting and potentially useful features the resulting patterns have:

1. It always produces rhombs, that is diamond or rectangle shapes. This is particularly useful for 3d artists.
2. For all dimensions but 3, 4 and 6 the results are aperiodic, this means that it will never repeat itself in the width or height of the image.
3. The results do repeat symmetrically around the center. The amount of symmetric repetitions is the same as the amount of dimensions projected.

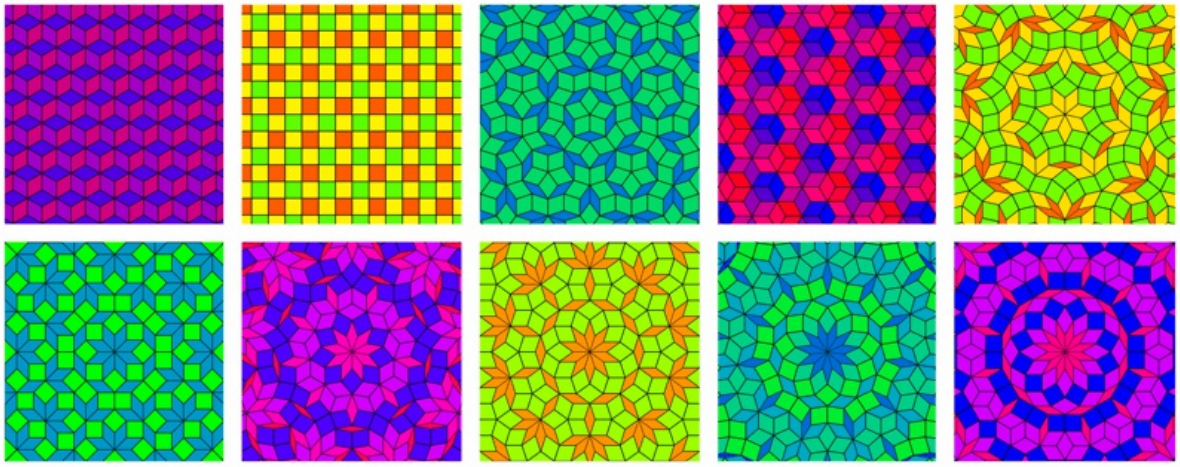
The resulting patterns are also known to show up in nature as quasicrystals.

Shapes

The meat of the algorithm. The default values for this produce the Star Penrose tiling.

Dimensions

: The amount of dimensions the hyperlattice has. 3 is a lattice of cubes, 4 is a lattice of tesseracts, 5 is a lattice of penteracts, and so forth.



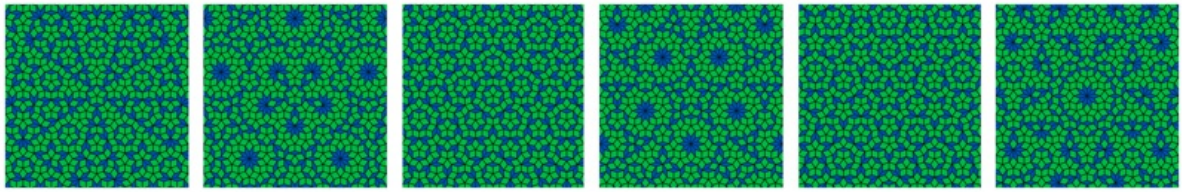
Multigrid with different dimensions, starting at 3 and ending at 12. 3d, 4d and 6d are colored with the intersect color factor while the rest uses ratio exclusively. In 3d, 4d and 6d, all the rhombs have the same ratio.

Divisions

Effectively a zoom-out. This is the subdivisions of the length of the width between the center and the corner of the image. This is then used to determine how many lines are projected for each dimension.

Offset

This controls how much each set of lines is offset from the center of the image. Changing this value changes the pattern within the same dimension significantly.



Multigrid with 5 dimensions and 20 divisions. The offsets from left to right are: 0.3, 0.1, 0.2 (Star Penrose tiling), 0.3, 0.4 (Sun Penrose Tiling), 0.48.

Lines

Line Width

The width of the outlines of the rhombs in image pixels. Due to the way

the rhombs are drawn, there is still a hairfine line visible at 0 px.

Connector Lines
: This optionally draws lines between the different sides of the shape. This is typically used to show that a specific tiling has certain matching rules, but it also gives cool looking results.

Acute Angle
: Draws an arc between the sides that connect to an acute angle.

Obtuse Angle
: Draws an arc between the sides that connect to an obtuse angle.

Cross
: Draws two lines crossed between the sides of each rhomb. Particularly interesting with 0 line width.

Colors

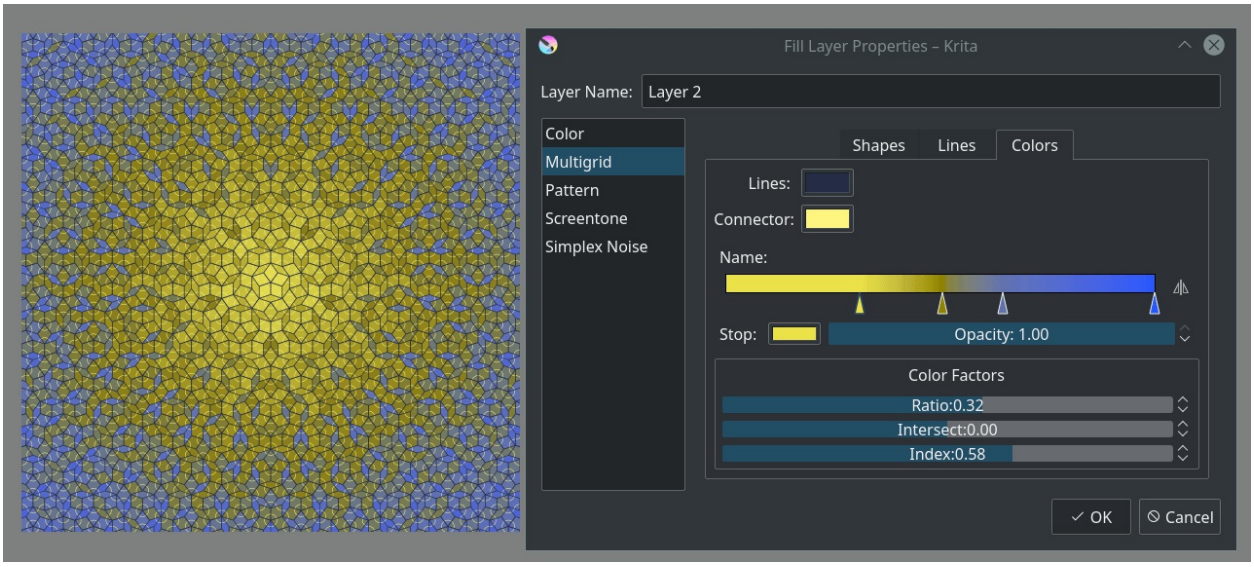


Image showing the Star Penrose Tiling with 29 divisions and connector lines at the acute angles. The complex gradient and the combination of ratio and index to color the image results in some of the more impressive results that can be gotten from this fill layer.

This section controls all the colors, all grouped together because Krita's color buttons allow drag and dropping colors to one another. You can change the color for the outlines and the connector lines, and there is a gradient for coloring the individual rhombs.

The color factors determine which properties of each rhomb is used to determine its coloring. This value is used as a multiplier, to finally result in a value that can be used to get the value from the gradient.

Ratio
: This colors the rhombs based on their ratio. Thin rhombs have a low ratio, thick rhombs have a high ratio, and perfect squares have the largest ratio.

Intersect
: This colors the rhombs based on which intersecting lines resulted in this rhomb. In effect, this colors the rhomb depending on which side of the hyperlattice the rhomb is on, as is especially clear when setting the dimension to 3.

Index
: This colors the rhombs based on the index of the intersecting lines from the center. In effect, rhombs closer to the center will have a lower value, while rhombs further from the center will have a higher value.

Pattern Fill

This fills the layer with a predefined pattern or texture that has been loaded into Krita through the Resource Management interface. Patterns can be a simple and interesting way to add texture to your drawing or painting, helping to recreate the look of watercolor paper, linen, canvas, hardboard, stone or an infinite other number of options. For example if you want to take a digital painting and finish it off with the appearance of it being on canvas you can add a Fill Layer with the Canvas texture from the texture pack below and set the opacity very low so the “threads” of the pattern are just barley visible. The effect is quite convincing.

You can create your own and use those as well. For a great set of well designed and useful patterns check out one of our favorite artists and a great friend of Krita, David Revoy’s free texture pack (https://www.davidrevoy.com/article156/texture-pack-1).

Transform

New in version 4.4.

This allows setting a number of transformation options on the pattern, such as scaling or rotating it.

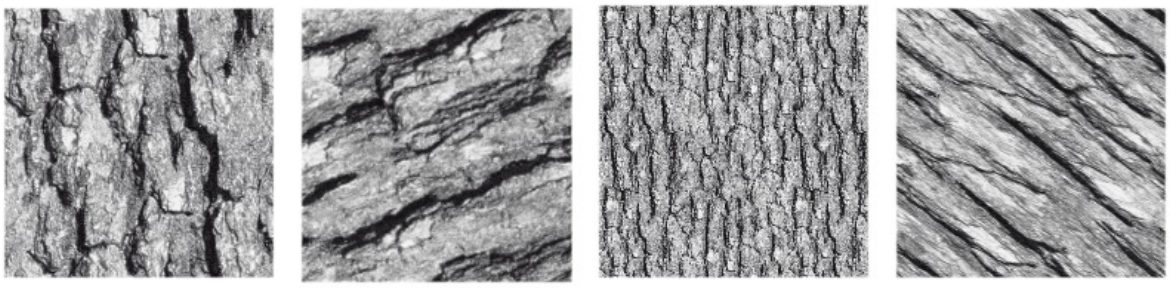


Image showing several transforms applied to a single texture.

Screentone

New in version 4.4.

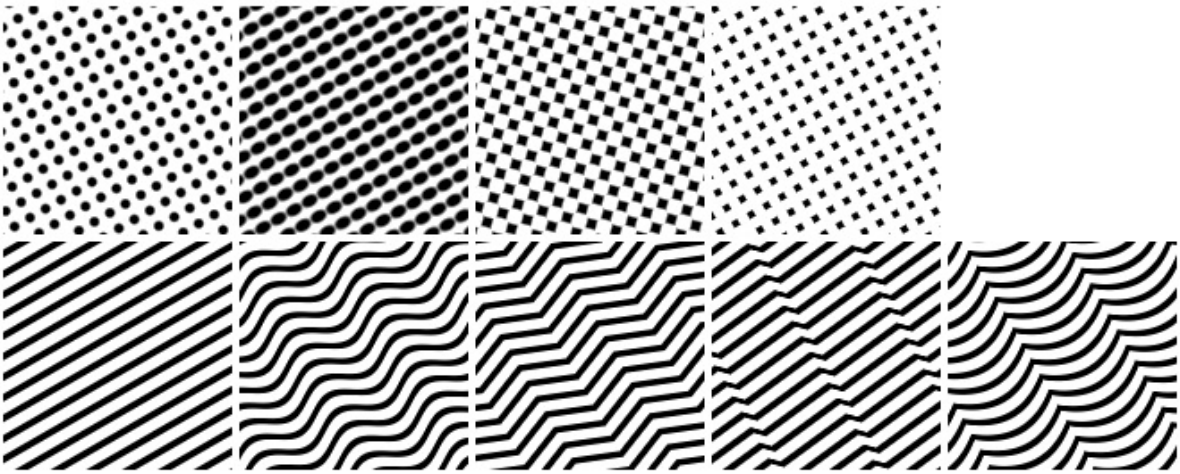
Fills the layer with simple regular patterns like dots and lines like the ones used in traditional screentone [https://en.wikipedia.org/wiki/Screentone] or halftone [https://en.wikipedia.org/wiki/Halftone] techniques.

Screentone Type
: Pattern
  : Select the global appearance (dots, lines).

  Shape
  : Select the specific appearance of the pattern (for example: round dots, diamond dots, straight lines, sine wave lines).

  Interpolation
  : Select how the pattern transitions from the foreground to the background.

    In most cases but the dot pattern with round shape combination, the effect of the interpolation mode is not very noticeable.



From left to right, top to bottom: dot pattern with round shape, dot pattern with ellipse shape, dot pattern with a diamond shape, dot pattern with a

square shape, line pattern with straight shape, line pattern with sine wave shape, line pattern with triangular wave shape, line pattern with sawtooth wave shape and line pattern with a curtain shape.

Transformation

Select how the pattern is arranged geometrically in the image (position, size, rotation, shear). Some patterns benefit from the capability of choosing horizontal and vertical sizes separately. For example, the sin wave lines pattern has a small period by default and by choosing a large horizontal size the period will look also larger.

From left to right: dot pattern with round shape without rotation, dot pattern with round shape and rotated 45 degrees, line pattern with sine wave shape and a size of 20px horizontally and vertically, line pattern with sine wave shape and a size of 50px horizontally and 20px vertically.

Postprocessing

Background & Foreground

Allows you to choose the color and opacity of the foreground (dots, lines) and the background.

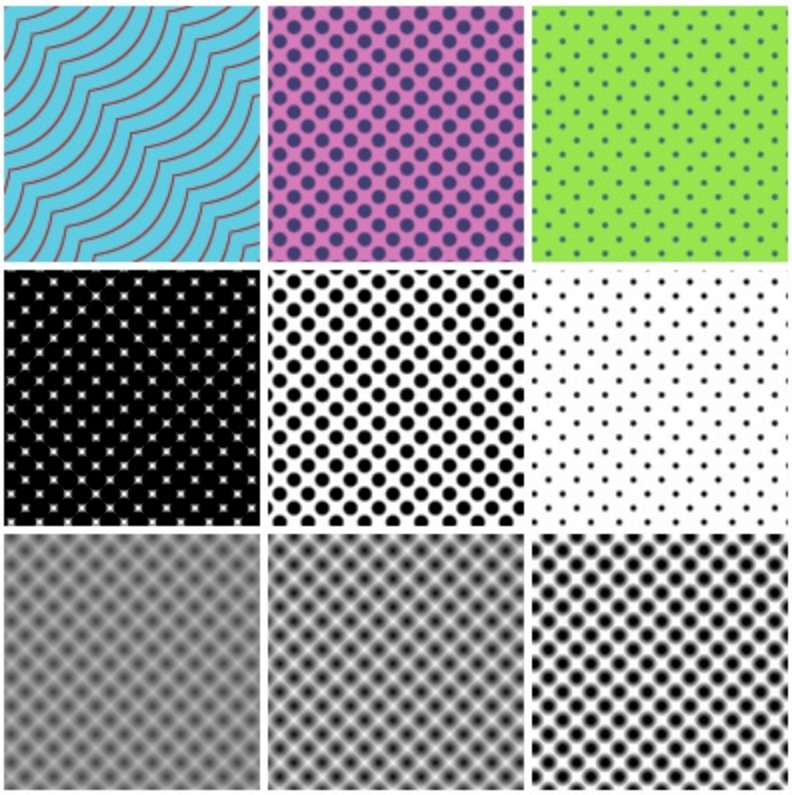
Invert

This flips what is treated as foreground and background.

Brightness & Contrast

The brightness controls how close to the foreground or background color the pattern appears (how *dark* or *light* in the case of black foreground and white background). So if you want to simulate small dots, for example, set the brightness to a high value and to obtain big dots set it to a low value.

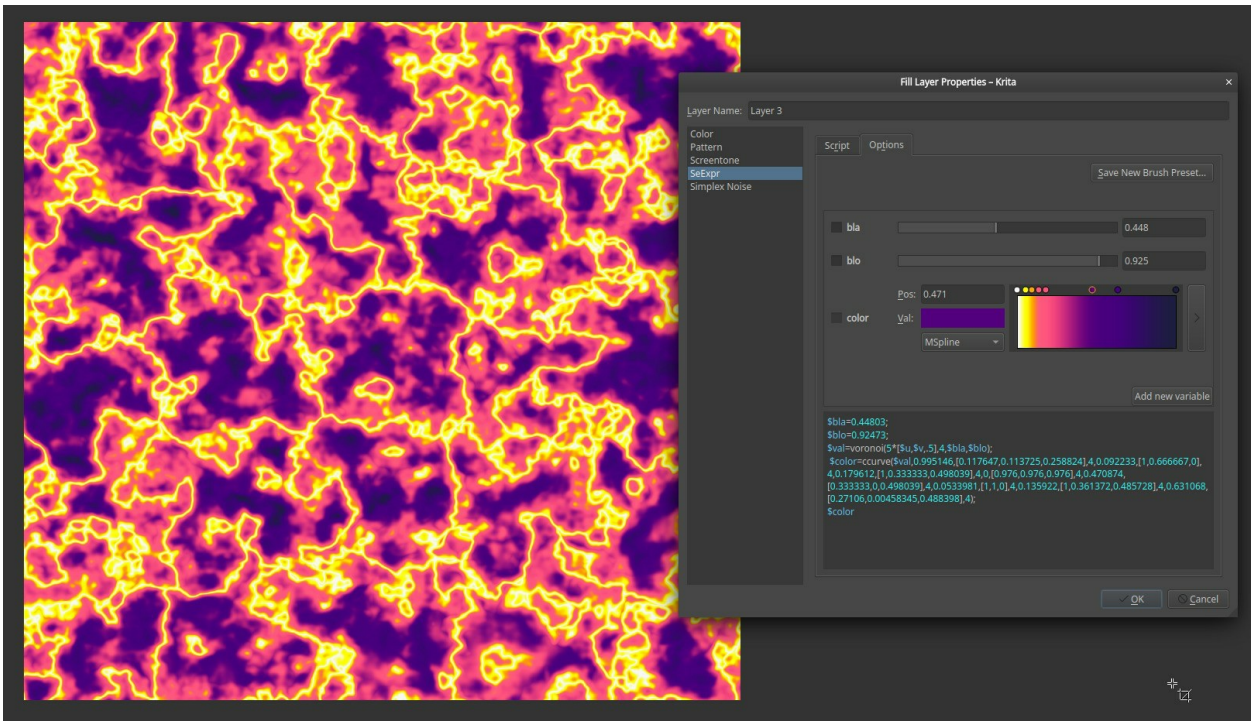
The contrast controls how smooth or sharp is the transition between the foreground and background colors. By default, the contrast is set to 50% (smooth). To achieve the typical sharp borders the contrast must be set to a higher value.



First row: different combinations of foreground and background colors. Second row, from left to right: 25%, 50% and 75% brightness with 90% contrast. Third row, from left to right: 25%, 50% and 75% contrast with 50% brightness.

SeExpr

New in version 4.4.



Fills the layer with a pattern specified through Disney Animation's SeExpr expression language [https://wdas.github.io/SeExpr].

See also

- Introduction to SeExpr
- SeExpr Quick Reference
- SeExpr Scripts
- "Procedural texture generator (example and wishes)" on Krita Artists [https://krita-artists.org/t/procedural-texture-generator-example-and-wishes/7638]
- Inigo Quilez's articles [https://iquilezles.org/www/index.htm]
- The Book of Shaders [https://thebookofshaders.com/]

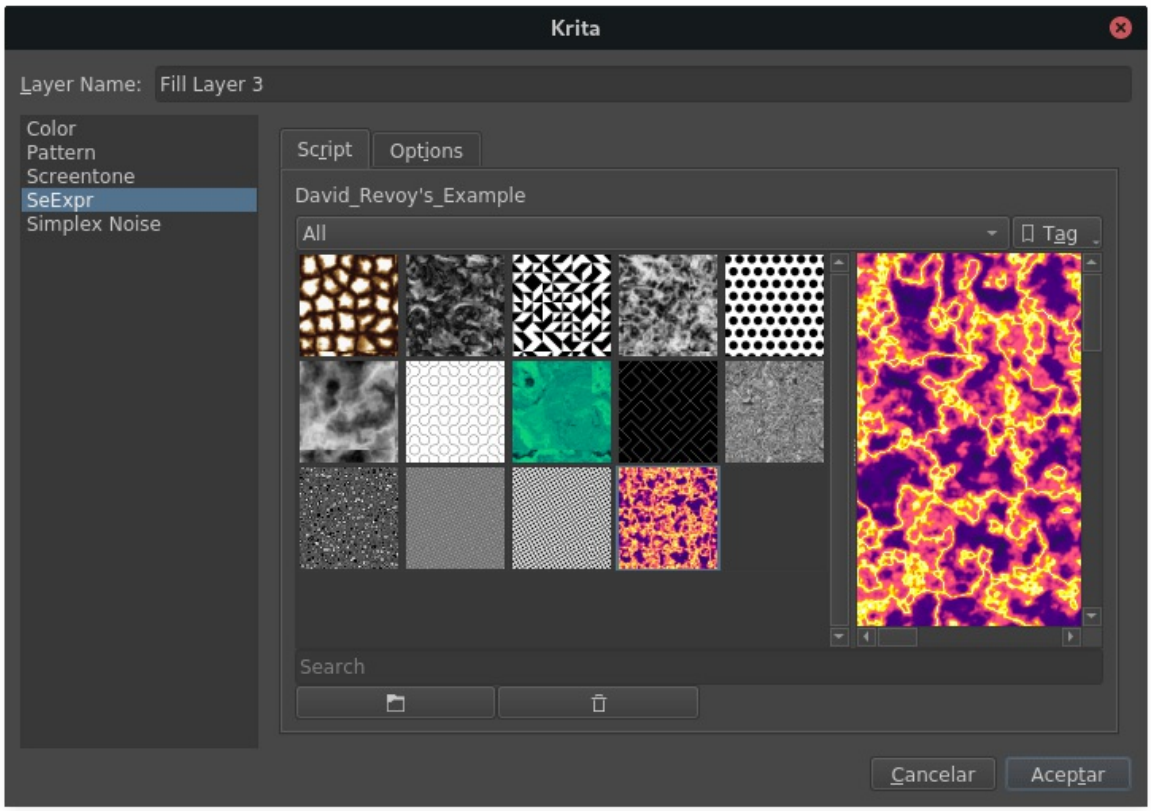
SeExpr is an embeddable, arithmetic expression language that enables you to

write shader-like scripts. Through this language, Krita can add dynamically generated textures like lava (example above), force fields, wood, marble, etc. to your layers.

As with Patterns, you can create your own and use those as well. For some examples, please check out the thread "Procedural texture generator (example and wishes)" on Krita Artists [https://krita-artists.org/t/procedural-texture-generator-example-and-wishes/7638]. You can download them as a bundle through Amyspark's blog [https://www.amyspark.me/blog/posts/2020/07/03/third-alpha-release.html].

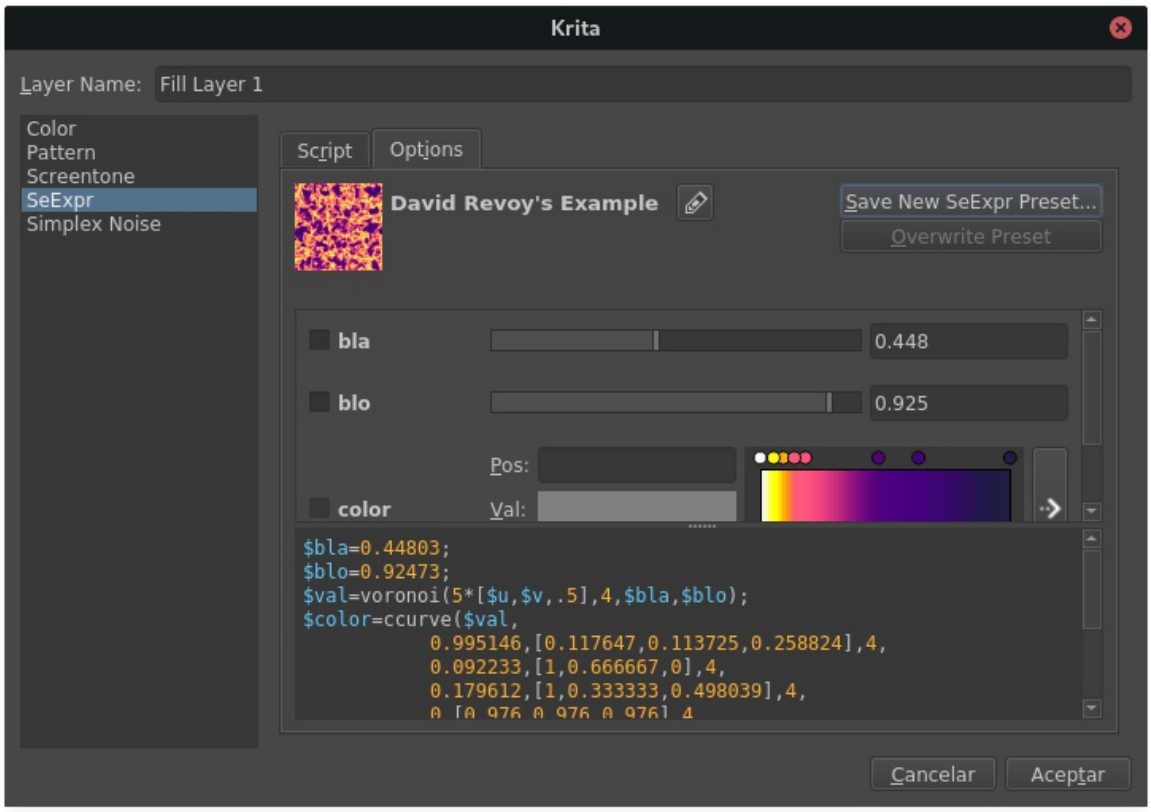
Script

Select the desired preset out of any existing bundled presets. This tab is identical to the Pattern preset selector.

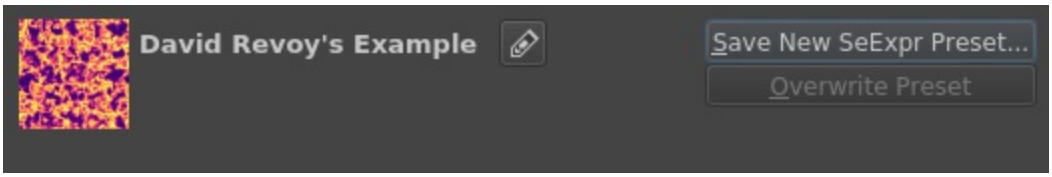


Options

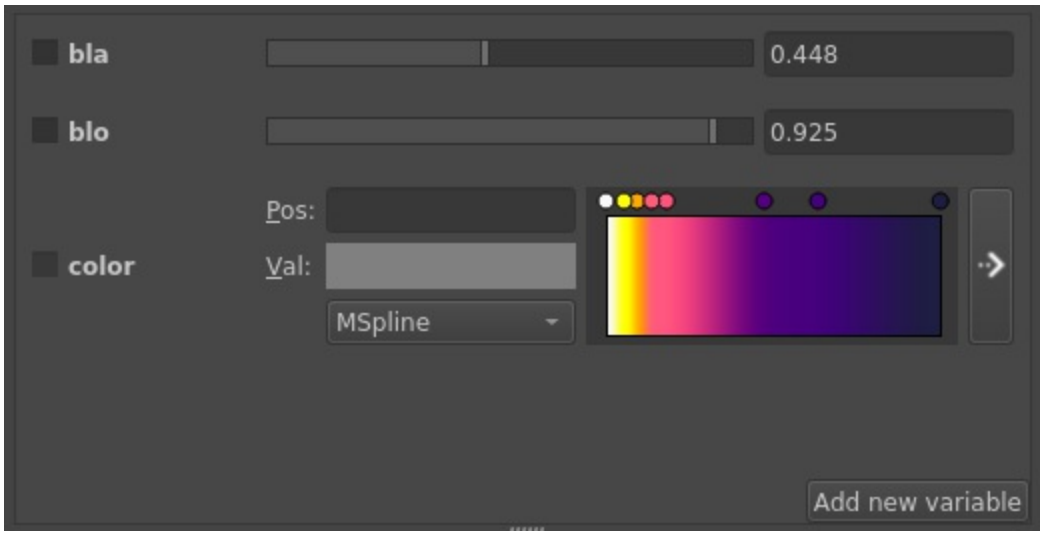
This tab allows you to edit the selected preset, and apply its script to the layer.



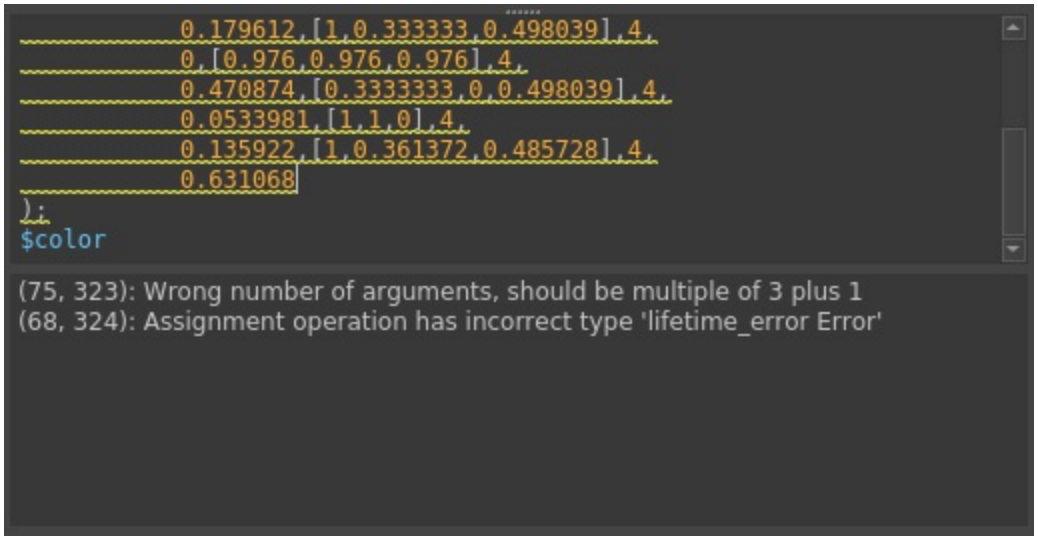
There are three sections. The first bar allows you to edit and save the selected preset:



If your script is syntactically correct, the middle box lets you adjust its variables through widgets.



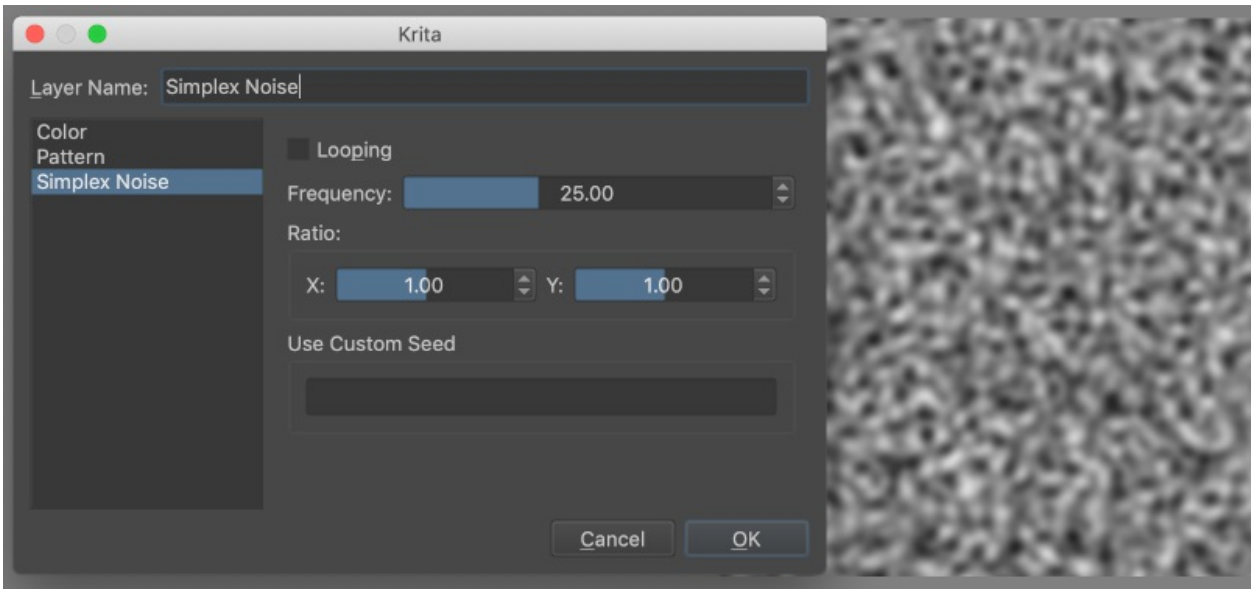
The lower box contains the script text, and shows the detected syntax errors, if any.



You can adjust how much space the latter two boxes have through their splitter.

Simplex Noise

New in version 4.2.



This fills the layer with generated OpenSimplex noise. OpenSimplex is different from the more common Perlin noise (often named ‘clouds’ in other software) and also different from Improved Perlin noise. OpenSimplex has less dimensional artifacts (the subtle “checker” texture often found high frequency Perlin noise) and is a ubiquitous open standard. Since OpenSimplex noise is important to texture generation, this fill layer has the option to loop around the canvas edge. You can read more about OpenSimplex here [https://en.wikipedia.org/wiki/OpenSimplex_noise].

There are a few different use cases for simplex noise. One of these is to create interesting looping patterns, achieved by stacking multiple simplex noise fills with different blending modes. It becomes even more expressive when combined with the levels adjustment layers. For texture artists, this can be a useful utility when combined with a gradient map filter layer to provide color diversity to a looping texture. For traditional artists, simplex noise layers can be converted to selection masks to create brush transparency dynamics and masking effects. Experimenting with different combinations can be fun and

produce interesting results!

Looping
: Whether or not to force the pattern to loop.

Frequency
: The frequency of the waves used to generate the pattern. Higher frequency results in a finer noise pattern.

Ratio
: The ratio of the waves in the x and y dimensions. This makes the noise have a rectangular appearance.

Use Custom Seed
: The seed for the random component. You can input any value or text here, and it will always try to use this value to generate the random values with (which then are always the same for a given seed). Leaving the value empty will use the randomly-assigned seed value on layer creation.

Filter Layer

Filter layers show whatever layers are underneath them, but with a filter such as Layer Styles, Blur, Levels, Brightness / Contrast. For example, if you add a **Filter Layer**, and choose the Blur filter, you will see every layer under your filter layer blurred.

Unlike applying a filter directly on to a section of a Paint Layer, Filter Layers do not actually alter the original image in the Paint Layers below them. Once again, non-destructive editing! You can tweak the filter at any time, and the changes can always be altered or removed.

Unlike Filter Masks though, Filter Layers apply to the entire canvas for the layers beneath. If you wish to apply a filter layer to only *some* layers, then you can utilize the Group Layer feature and add those layers into a group with the filter layer on top of the stack.

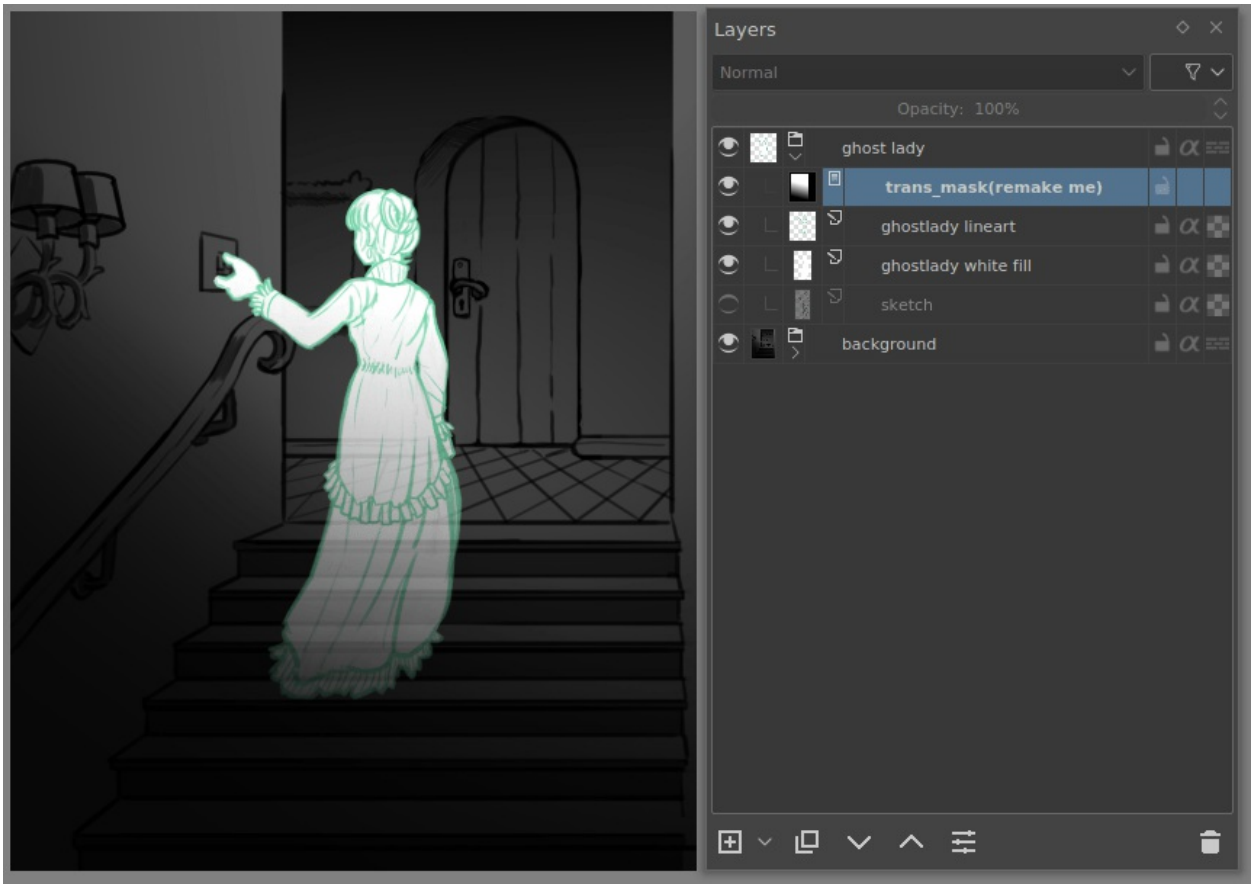
You can edit the settings for a filter layer, by double clicking on it in the Layers docker.

Note

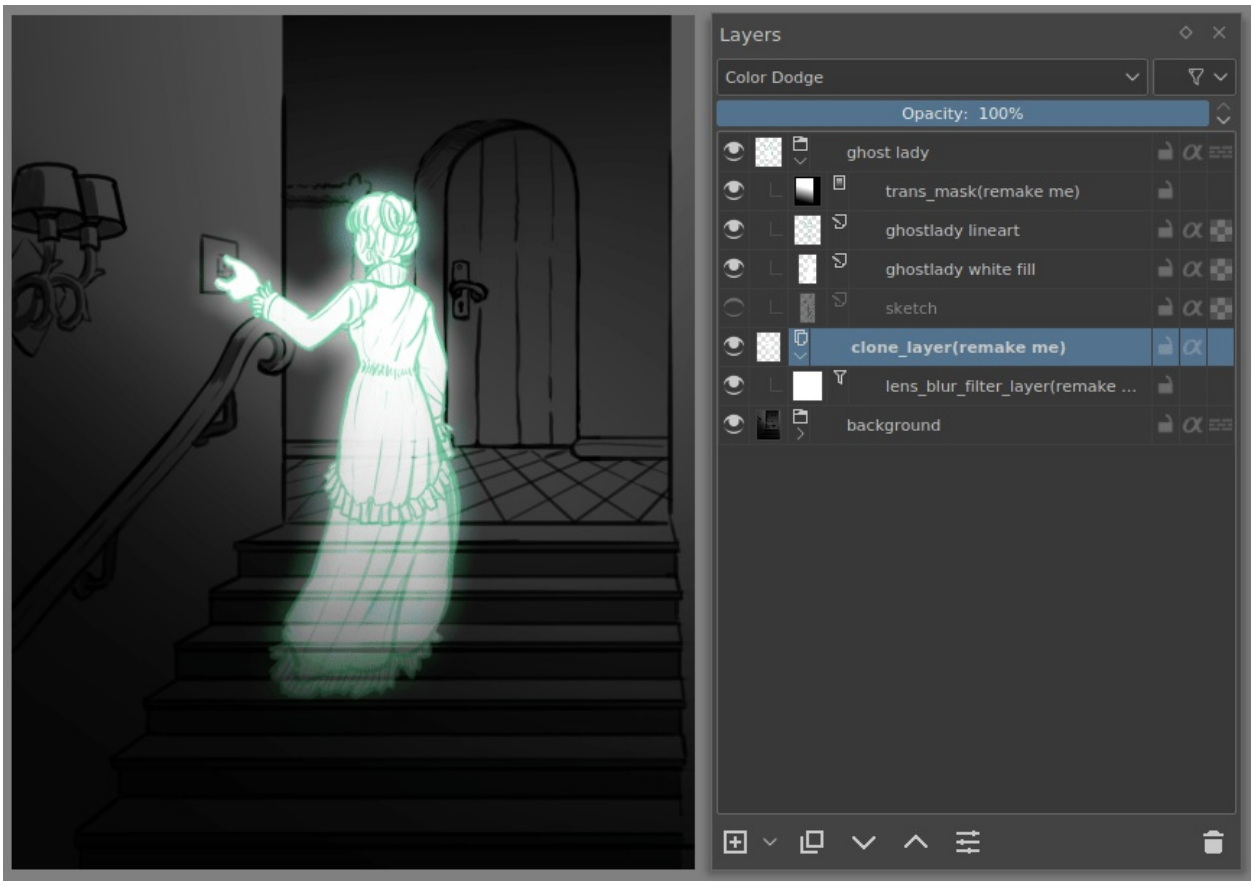
Only Krita native filters (the ones in the *Filters* menu) can be used with Filter Layers. Filter Layers are not supported using the externally integrated G'Mic filters.

Filter Masks

Filter masks show an area of their layer with a filter (such as blur, levels, brightness / contrast etc.). For example, if you select an area of a paint layer and add a Filter Layer, you will be asked to choose a filter. If you choose the blur filter, you will see the area you selected blurred.



With filter masks, we can for example make this ghost-lady more ethereal by putting a clone layer underneath, and setting a lens-blur filter on it.



Set the blending mode of the clone layer to *Color Dodge* and she becomes really spooky!

Unlike applying a filter to a section of a paint layer directly, filter masks do not permanently alter the original image. This means you can tweak the filter (or the area it applies to) at any time. Changes can always be altered or removed.

Unlike filter layers, filter masks apply only to the area you have selected (the mask).

You can edit the settings for a filter mask at any time by double clicking on it in the Layers docker. You can also change the selection that the filter mask affects by selecting the filter mask in the Layers docker and then using the paint tools in the main window. Painting white includes the area, painting black excludes it, and all other colors are turned into a shade of gray which applies proportionally.

Group Layers

While working on complex artwork you'll often find the need to group some layers or portions and elements of the artwork as one unit. Group layers come in handy for this: They allow you to segregate some layers so you can hide these quickly, or so you can recursively transform the content of the group, or so you can apply a mask to all the layers inside this group as if they are one (e.g. by dragging an existing mask to to a group layer), etc.. You can quickly create a group layer by selecting the layers that should be grouped together and then pressing the `Ctrl + G` shortcut.

A thing to note is that the layers inside a group layer are considered separately from the outside when the group layer gets composited: All layers inside are combined first, and then this intermediate result is used on the outside for compositing the rest of the image. In Photoshop on the contrary, groups have something called pass-through mode which makes the layers behave as if they are not in a group and get composited along with other layers of the stack. Recent versions of Krita offer a pass-through mode as well, which can be enabled in order to get similar behavior.

Layer Styles

Layer styles are effects that are added on top of your layer. They are editable and can easily be toggled on and off. To add a layer style to a layer go to *Layer* ▸ *Layer Style*. You can also right-click a layer to access the layer styles.

When you have the layer styles window up, make sure that the *Enable Effects* item is checked.

There are a variety of effects and styles you can apply to a layer. When you add a style, your layer docker will show an extra “Fx” icon. This allows you to toggle the layer style effects on and off.

Note

This feature was added to increase support for **Adobe Photoshop**. The features that are included mirror what that application supports.

Paint Layers

Paint layers are the most commonly used type of layers used in digital paint or image manipulation software like Krita. If you've ever used layers in **Photoshop** or the **Gimp**, you'll be used to how they work. In short, a paint layer, also called a pixel, bitmap or raster layer, is a bitmap image (an image made up of many points of color).

Paint layers let you apply many advanced effects such as smearing, smudging and distorting. This makes them the most flexible type of layer. However, paint layers don't scale well when enlarged (they pixelate), and any effects that have been applied can't be edited.

To deal with these two drawbacks, digital artists will typically work at higher Pixel Per Inch (PPI) counts. It is not unusual to see PPI settings of 400 to 600 PPI for a canvas with a good amount of detail. To combat the issue of applied effects that cannot be edited it is best to take advantage of the non-destructive layer capabilities of filter, transparency and transform masks.

As long as you have enough resolution / size on your canvas though, and as long as you aren't going to need to go back and tweak an effect you created previously, then a paint layer is usually the type of layer you will want. If you click on the *New layer* icon in the layers docker you'll get a paint layer. Of course you can always choose the *New layer* drop-down to get another type.

The hotkey for adding a new paint layer is the `Ins` key.

Selection Masks

Local Selection masks let you remember and recall edit a selection on a layer. They work in a similar way to extra channels in other image editing programs. One difference is **Krita's** ability to assign them to specific layers and activate a selection with a single click on the layer. Just click the round icon with the dotted outline on the local selection layer in the Layers docker.

You can make them by making a selection, and  the layer you want to add it to select *Local Selection*.

When isolating a selection mask with the `Alt` +  shortcut, you can perform transformation, deformation and paint operations on the selection layer, modifying the selection.

A single layer can contain multiple Local Selection Masks. Repeating. A single layer can contain multiple Local Selection Masks (LSM). This is important because it means that you can, for instance, have several different outline parts of an image and save each as its own LSM and then recall it with a single click. Without using LSM you would have to create layer upon layer for each mask. Not only would this be inefficient for you but also for Krita and the program would slow down trying to keep up with it all. LSM’s are one of the most important features in Krita!

The example below shows three LSM items all attached (under) Layer1. Any of these can be activated and used at any time.

Global Selection

You can modify the global selection the same way you can with a local-selection. To do so, you first need to activate the global selection as a layer node. To do so, go into *Select* ▸ *Show Global Selection Mask*. The global selection, if you have anything selected, will now appear on the top of the

layer stack as a selection mask.

Split Alpha

Sometimes especially in the field of game development, artists need to work with the alpha channel of the texture separately. To assist such workflow, Krita has a special functionality called *Split Alpha*. It allows splitting alpha channel of a paint layer into a separate Transparency Masks. The artist can work on the transparency mask in an isolated environment and merge it back when he has finished working.

How to work with alpha channel of the layer

1.  the paint layer in the layers docker.
2. Choose *Split Alpha* ▸ *Alpha into Mask.*
3. Use your preferred paint tool to paint on the Transparency Mask. Black paints transparency (see-through), white paints opacity (visible). Gray values paint semi-transparency.
4. If you would like to isolate alpha channel, enter Isolated Mode by  + *Isolate Layer* (or the `Alt` +  shortcut).
5. When finished editing the Transparency Mask,  on it and select *Split Alpha* ▸ *Write as Alpha*.

How to save a PNG texture and keep color values in fully transparent areas

Normally, when saving an image to a file, all fully transparent areas of the image are filled with black color. It happens because when composing the layers of the image, Krita drop color data of fully transparent pixels for efficiency reason. To avoid this of color data loss you can either avoid compositing of the image i.e. limit image to only one layer without any masks or effects, or use the following method:

1.  the layer in the layers docker.
2. Choose *Split Alpha* ▸ *Alpha into Mask.*
3.  on the created mask and select *Split Alpha* ▸ *Save Merged.…*

Transformation Masks

Rather than working with a brush to affect the mask, transformation masks allow you to transform (move, rotate, shear, scale and perspective) a layer without applying the transform directly to the paint layer and making it permanent.

In the same way that Filter and Transparency Masks can be attached to a Paint layer and are non-destructive, so too can the Transformation Mask.

Adding a Transformation Mask

- First add a transform mask to an existing layer.
- Select the transformation tool.
- Select any of the transform modes in the Tools Options dock and, with the transform mask selected, apply them on the layer.
- Hit apply.
- Toggle the transform visibility to see the difference between the original and the transform applied.

Note

Affine transforms, like Move, Rotate, Shear, Scale and Perspective get updated instantly once the original is updated. Other transforms like Warp, Cage and Liquify take up much more processing power, and to not to waste that, Krita only updates those every three seconds.

To edit a transform, select the transform mask, and try to use the transform tool on the layer. The transform mode will be the same as the stored transform, regardless of what transform you had selected. If you switch transform modes, the transformation will be undone.

Transparency Masks

The Transparency mask allows you to selectively show or hide parts of a layer. By using a mask, you are able to avoid deleting parts of an image that you just might want in the future. This allows you to work non-destructively.

In addition, it allows you to do things like remove a portion of a layer in the layer stack so you can see what's behind it. One example would be if you wanted to replace a sky, but were unsure of how much you wanted to replace.

How to add a transparency mask

1. Click on a paint layer in the layers docker.
2. Click on "+" drop-down in the bottom left corner of the layers docker and choose *Transparency Mask*.
3. Use your preferred paint tool to paint on the canvas. Black paints transparency (see-through), white paints opacity (visible). Gray values paint semi-transparency.

You can always fine-tune and edit what you want visible and any layer. If you discover you've hidden part of your paint layer accidentally, you can always show it again just by painting white on your transparency mask.

This makes for a workflow that is extremely flexible and tolerant of mistakes.

Vector Layers

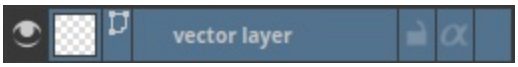
Warning

This page is outdated. Check Vector Graphics for a better overview.

What is a Vector Layer?

A Vector Layers, also known as a shape layer, is a type of layers that contains only vector elements.

This is how vector layers will appear in the **Krita** Layers docker.



It shows the vector contents of the layer on the left side. The icon showing the page with the red bookmark denotes that it is a vector layer. To the right of that is the layer name. Next are the layer visibility and accessibility icons. Clicking the “eye” will toggle visibility. Clicking the lock into a closed position will lock the content and editing will no longer be allowed until it is clicked again and the lock on the layer is released.

Creating a vector layer

You can create a vector layer in two ways. Using the extra options from the “Add Layer” button you can click the “Vector Layer” item and it will create a new vector layer. You can also drag a rectangle or ellipse from the **Add shape** dock onto an active Paint Layer. If the active layer is a Vector Layer then the shape will be added directly to it.

Editing Shapes on a Vector Layer

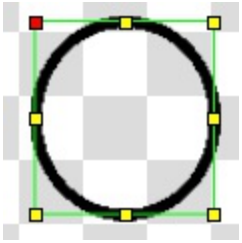
Warning

There's currently a bug with the vector layers that they will always consider themselves to be at 72dpi, regardless of the actual pixel-size. This can make manipulating shapes a little difficult, as the precise input will not allow cm or inch, even though the vector layer coordinate system uses those as a basis.

Basic Shape Manipulation

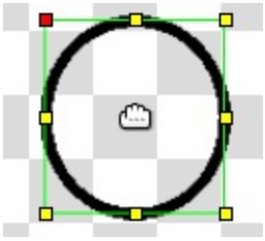
To edit the shape and colors of your vector element, you will need to use the basic shape manipulation tool.

Once you have selected this tool, click on the element you want to manipulate and you will see guides appear around your shape.



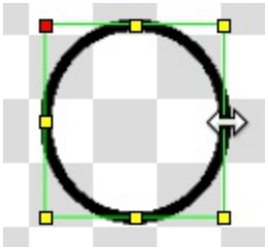
There are four ways to manipulate your image using this tool and the guides on your shape.

Transform/Move



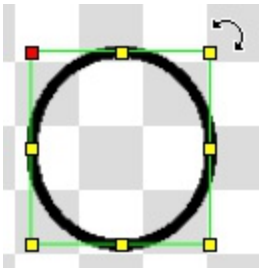
This feature of the tool allows you to move your object by clicking and dragging your shape around the canvas. Holding the `Ctrl` key will lock your moves to one axis.

Size/Stretch



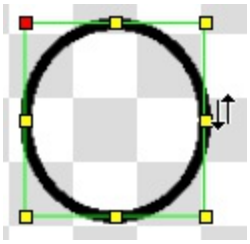
This feature of the tool allows you to stretch your shape. Selecting a midpoint will allow stretching along one axis. Selecting a corner point will allow stretching across both axis. Holding the `Shift` key will allow you to scale your object. Holding the `Ctrl` key will cause your manipulation to be mirrored across your object.

Rotate



This feature of the tool will allow you to rotate your object around its center. Holding the `Ctrl` key will cause your rotation to lock to 45 degree angles.

Skew



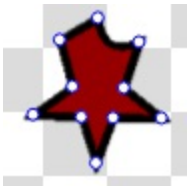
This feature of the tool will allow you to skew your object.

Note

At the moment there is no way to scale only one side of your vector object. The developers are aware that this could be useful and will work on it as manpower allows.

Point and Curve Shape Manipulation

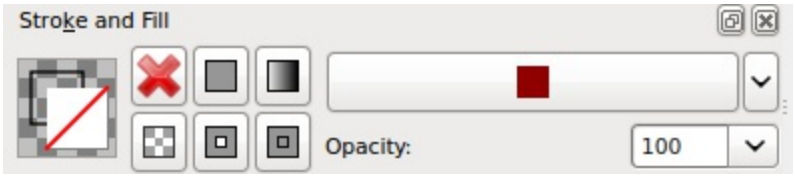
Double-click on a vector object to edit the specific points or curves which make up the shape. Click and drag a point to move it around the canvas. Click and drag along a line to curve it between two points. Holding the `Ctrl` key will lock your moves to one axis.



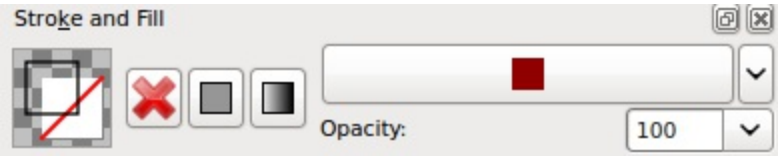
Stroke and Fill

In addition to being defined by points and curves, a shape also has two defining properties: **Fill** and **Stroke**. **Fill** defines the color, gradient, or pattern that fills the space inside of the shape object. '**Stroke**' defines the color, gradient, pattern, and thickness of the border along the edge of the shape. These two can be edited using the **Stroke and Fill** dock. The dock has two modes. One for stroke and one for fill. You can change modes by clicking in the dock on the filled square or the black line. The active mode will be shown by which is on top of the other.

Here is the dock with the fill element active. Notice the red line across the solid white square. This tells us that there is no fill assigned therefore the inside of the shape will be transparent.

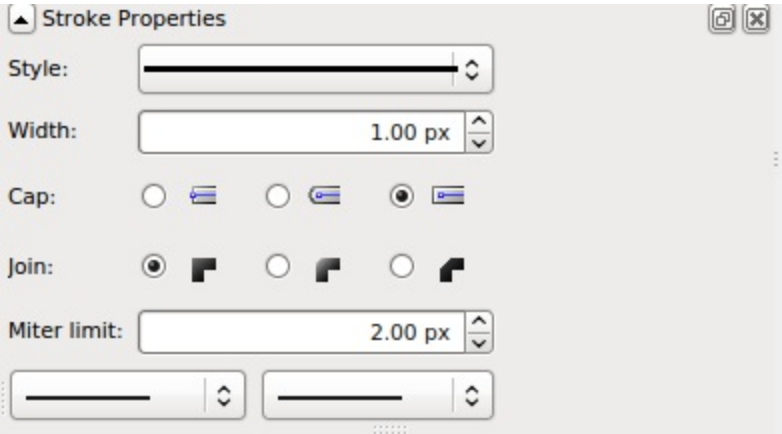


Here is the dock with the stroke element active.



Editing Stroke Properties

The stroke properties dock will allow you to edit a different aspect of how the outline of your vector shape looks.

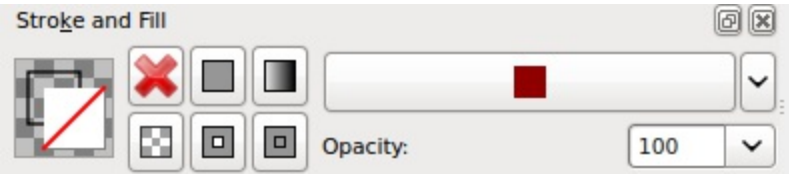


The style selector allows you to choose different patterns and line styles. The width option changes the thickness of the outline on your vector shape. The cap option changes how line endings appear. The join option changes how corners appear.

The Miter limit controls how harsh the corners of your object will display. The higher the number the more the corners will be allowed to stretch out past the points. Lower numbers will restrict the stroke to shorter and less sharp corners.

Editing Fill Properties

All of the fill properties are contained in the **Stroke and Fill** dock.

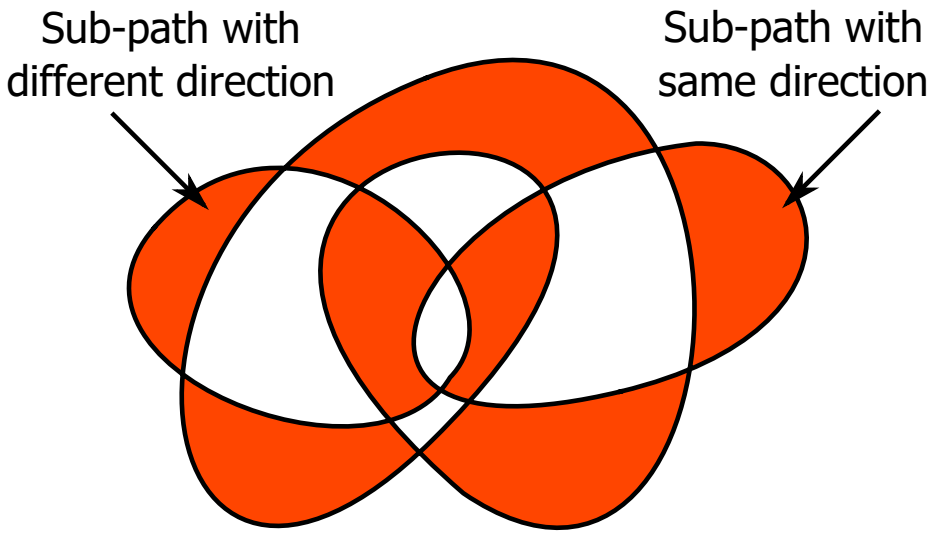


The large red **X** button will set the fill to none causing the area inside of the vector shape to be transparent.

To the right of that is the solid square. This sets the fill to be a solid color which is displayed in the long button and can be selected by pressing the arrow just to the right of the long button. To the right of the solid square is the gradient button. This will set the fill to display as a gradient. A gradient can be selected by pressing the down arrow next to the long button.

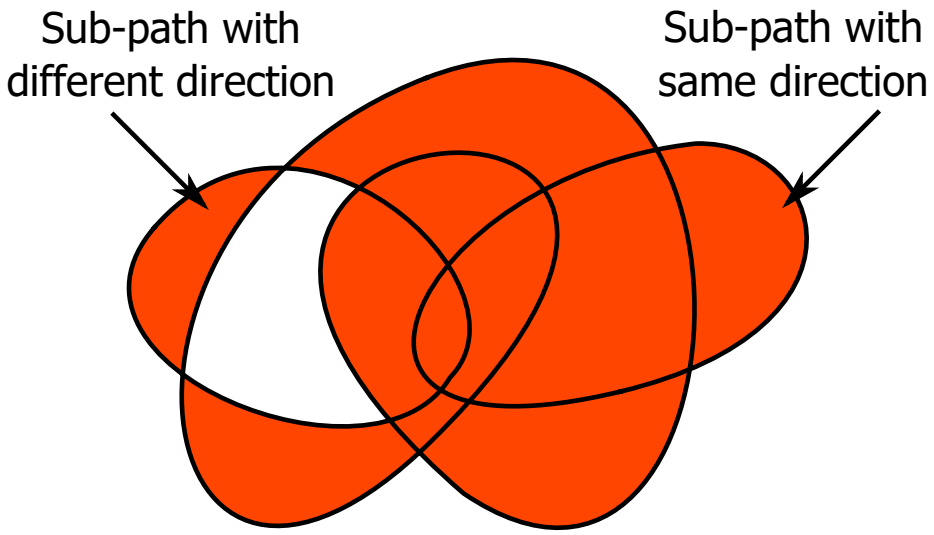
Under the **X** is a button that shows a pattern. This inside area will be filled with a pattern. A pattern can be chosen by pressing the arrows next to the long button. The two other buttons are for **fill rules**: the way a self-overlapping path is filled.

- The button with the inner square blank toggles even-odd mode, where every filled region of the path is next to an unfilled one, like this:



- The button with the inner square filled toggles non zero mode, where most of the time a self overlapping path is entirely filled except when it overlaps with a sub-path of a different direction that ‘decrease the level

of overlapping' so that the region between the two is considered outside the path and remain unfilled, like this:



For more (and better) information about fill rules check the Inkscape manual [http://tavmjong.free.fr/INKSCAPE/MANUAL/html/Attributes-Fill-Stroke.html#Attributes-Fill-Rule].

Linux Command Line

As a native Linux program, Krita allows you to do operations on images without opening the program when using the Terminal. This option was disabled on Windows and OSX, but with 3.3 it is enabled for them!

This is primarily used in bash or shell scripts, for example, to mass convert KRA files into PNGs.

Export

This allows you to quickly convert files via the terminal:

```
krita importfilename --export --export-filename exportfilename
```

importfilename
: Replace this with the filename of the file you want to manipulate.

`--export`
: Export a file selects the export option.

`--export-filename` `<filename>`
: Export filename says that the following word is the filename it should be exported to.

exportfilename
: Replace this with the name of the output file. Use a different extension to change the file format.

Example:

```
krita file.png --export --export-filename final.jpg
```

This line takes the file `file.png` and saves it as `file.jpg`.

`--export-sequence`

New in version 4.2.

Export animation to the given filename and exit.

If a KRA file has no animation, then this command prints “This file has no animation.” error and does nothing.

```
krita --export-sequence --export-filename file.png test.kra
```

This line takes the animation in *test.kra*, and uses the value of –export-filename (*file.png*), to determine the sequence fileformat(‘png’) and the frame prefix (‘file’).

PDF export

PDF export looks a bit different, using the `--export-pdf` option.

```
krita file.png --export-pdf --export-filename final.pdf
```

This option exports the file `file.png` as a PDF file.

Warning

This has been removed from 3.1 because the results were incorrect.

Open with Custom Screen DPI

Open Krita with specified Screen DPI.

`--dpi` `<dpiX,dpiY>`

Open Krita with specified Screen DPI.

For example:

```
krita --dpi <72,72>
```

Open template

Open krita and automatically open the given template(s). This allows you to, for example, create a shortcut to Krita that opens a given template, so you can get to work immediately!

```
krita --template templatename.desktop
```

--template `templatename.desktop`
: Selects the template option.

  All templates are saved with the .desktop extension. You can find templates in the .local/share/krita/template or in the install folder of Krita.

  ```
  krita --template BD-EuroTemplate.desktop
  ```

  This opens the European BD comic template with Krita.

  ```
  krita --template BD-EuroTemplate.desktop BD-
  EuroTemplate.desktop
  ```

  This opens the European BD template twice, in separate documents.

Start up

--nosplash
: Starts krita without showing the splash screen.

--canvasonly
: Starts krita in canvasonly mode.

--fullscreen
: Starts krita in fullscreen mode.

--workspace `Workspace`
: Starts krita with the given workspace. So for example…

  ```
  krita --workspace Animation
  ```

Starts Krita in the Animation workspace.

`--file-layer` `<filename>`

Starts krita with `filename` added as a file-layer. Note that you must either open an existing file or create a new file using the `new-image` argument.

Example:

```
krita file.kra --file-layer image.png
```

```
krita --new-image RGBA,U8,1000,1000 --file-layer image.jpg
```

If an instance of Krita is already running and Multiple instances are disabled, then this option can be used alone to add a file-layer to the running krita document.

Example: `krita --file-layer image.png`

The List of Supported Tablets

This is specifically about support on Windows, not Linux or OSX.

Brand	Model	Supported
Adesso	CyberTablet T12	❓ Unknown
Adesso	CyberTablet Z12	❓ Unknown
Adesso	CyberTablet T10	❓ Unknown
Adesso	CyberTablet T22HD	❓ Unknown
Adesso	CyberTablet M14	❓ Unknown
Adesso	CyberTablet W10	❓ Unknown
Adesso	CyberTablet Z8	✔ Supposed to work
Aiptek	HyperPen Mini	❓ Unknown
Aiptek	MediaTablet 10000u	❓ Unknown

Aiptek	MediaTablet 14000u	❓ Unknown
Aiptek	MediaTablet Ultimate II	❓ Unknown
Aiptek	MyNote Bluetooth	❓ Unknown
Aiptek	MyNote Pen	❓ Unknown
Aiptek	SlimTablet 600u Premium II	❓ Unknown
Artisul	(by UC-Logic)D13	✕ Reported to not work
Artisul	(by UC-Logic)D10	✕ Reported to not work
Artisul	(by UC-Logic)Pencil (S/M)	✕ Reported to not work
Bosto	22HDX	✕ Reported to be broken
Bosto	22UX	✕ Reported to be broken
Bosto	22HD Mini	✕ Reported to be broken

Bosto	22U Mini	❓ Unknown
Bosto	14WX	❓ Unknown
Bosto	13HD	❓ Unknown
CalComp	DrawingBoard VI	❓ Unknown
CalComp	Creation Station	❓ Unknown
CalComp	SummaSketch	❓ Unknown
Dynalink	FreeDraw 4x5	❓ Unknown
Elmo	CRA-1 wireless tablet	❓ Unknown
Gaomon	S56K	❓ Unknown
Gaomon	GM185	❓ Unknown
Gaomon	M10K	❓ Unknown
Gaomon	P1560	✔ Supposed to work
Genius	EasyPen	❓ Unknown

Genius	EasyPen 340	❓ Unknown
Genius	EasyPen F610E	❓ Unknown
Genius	EasyPen i405	❓ Unknown
Genius	EasyPen i405X	❓ Reported to be working on Linux only for 2.9.x versions
Genius	EasyPen i405XE	❓ Unknown
Genius	EasyPen M406	❓ Unknown
Genius	EasyPen M406W	❓ Unknown
Genius	EasyPen M406WE	❓ Unknown
Genius	EasyPen M406XE	❓ Unknown
Genius	EasyPen M506	❓ Unknown
Genius	EasyPen M506A	❓ Unknown
Genius	EasyPen M508W	❓ Unknown

Genius	EasyPen M610	✕ Reported to be broken
Genius	EasyPen M610X	? Unknown
Genius	EasyPen M610XA	? Unknown
Genius	G-Pen 340	? Unknown
Genius	G-Pen 450	? Unknown
Genius	G-Pen 560	? Unknown
Genius	G-Pen F350	? Unknown
Genius	G-Pen F509	? Unknown
Genius	G-Pen F610	? Unknown
Genius	G-Pen M609	? Unknown
Genius	G-Pen M609X	? Unknown
Genius	G-Pen M712	? Unknown
Genius	G-Pen M712X	? Unknown

Genius	MousePen 8x6	? Unknown
Genius	MousePen i608	? Unknown
Genius	MousePen i608X	? Unknown
Genius	MousePen i608XE	? Unknown
Genius	MousePen M508	? Unknown
Genius	MousePen M508W	? Unknown
Genius	MousePen M508X	? Unknown
Genius	MousePen M508XA	? Unknown
Genius	PenSketch 9x12	? Unknown
Genius	PenSketch M912	✕ Reported to be broken
Genius	PenSketch T609A	? Unknown
Genius	WizardPen 5x4	? Unknown

Hanvon	ESP2210	❓ Unknown
Hanvon	HW-S05	❓ Unknown
Hanvon	Sell T&Mouse	❓ Unknown
Hanvon	Sell Writing Tablet (SuperPen 0403)	❓ Unknown
Hanvon	Sell Writing Tablet (SuperPen 0503)	❓ Unknown
Hanvon	Sell Painting Master (0504)	❓ Unknown
Hanvon	Sell Painting Master (0605)	❓ Unknown
Hanvon	Sell Painting Master (0806)	❌ Reported to be broken
Huion / Turcom	H420	✔ Supposed to work
Huion / Turcom	W58	✔ Supposed to work
Huion / Turcom	680TF	✔ Supposed to work
Huion / Turcom	G10T	✔ Supposed to work

Huion / Turcom	H610	✔ Supported
Huion / Turcom	H610PRO	✔ Supported
Huion / Turcom	H690	✔ Supposed to work
Huion / Turcom	WH1409	✔ Supported
Huion / Turcom	1060Plus	✔ Supposed to work
Huion / Turcom	New 1060Plus	✔ Supposed to work
Huion / Turcom	K26	✔ Supposed to work
Huion / Turcom	K58	✔ Supposed to work
Huion / Turcom	W58	✔ Supposed to work
Huion / Turcom	680S	✔ Supposed to work

Huion / Turcom	P608N	✔ Supposed to work
Huion / Turcom	H58L	✔ Supposed to work
Huion	H950P	✔ Supported
Huion / Turcom	DWH96	✔ Supposed to work
Huion / Turcom	G-T156HD (KAMVAS)	✔ Supposed to work
Huion / Turcom	GT-185	✔ Supposed to work
Huion / Turcom	GT-190	✔ Supposed to work
Huion / Turcom	GT-191 (KAMVAS)	? Reported to work with experimental user space driver
Huion / Turcom	GT-220	✔ Supposed to work

Huion / Turcom	PC185HD	✔ Supported
Huion / Turcom	PC2150	✔ Supposed to work
Huion / Turcom	Inspiroy Q11K	✔ Supported
KB Gear	JamStudio	❓ Unknown
KB Gear	Pablo Internet Edition	❓ Unknown
KB Gear	Sketchboard Studio	❓ Unknown
Microsoft	Surface Pro Surface Pro 2	✔ Supported
Microsoft	Surface Pro 3 Surface Pro 4 Surface Studio Surface Pro (2017) Surface Laptop	✔ Supported
Monoprice	8x6”	❓ Unknown
Monoprice	MP1060-HA60 (10x6.25”)	❓ Unknown
Monoprice	10x6.25” (110594)	❓ Unknown
Monoprice	8x6” MP Select Professional	❓ Unknown

Monoprice	“8x6”” MP Select Professional with Quick Select Wheel”	❓ Unknown
Monoprice	12x9” (106815)	❓ Reported to work with some issues
Monoprice	MP 22-inch (114481)	❓ Unknown
Parblo	A610	✕ Reported to be broken
Parblo	Bay B960	❓ Unknown
Parblo	GT19	❓ Unknown
Parblo	GT22HD	❓ Unknown
Parblo	Coast22	❓ Unknown
Parblo	Coast10	❓ Unknown
Parblo	Island A609	❓ Reported to work, but tablet is low-quality and not recommended.

PenPower	TOOYA Master	❓ Unknown
PenPower	TOOYA X	❓ Unknown
PenPower	Monet	❓ Unknown
PenPower	Picasso	❓ Unknown
Perixx	Peritab-502EVO	❓ Unknown
Perixx	Peritab 502	❓ Unknown
Perixx	Peritab 302	❓ Unknown
Samsung	Galaxy Book	✔ Supported
Trust	Flex Design	❓ Unknown
Trust	Slimline Widescreen	✕ Reported to be broken
Trust	Slimline Sketch	❓ Unknown
Trust	Slimline Mini	❓ Unknown
Trust	TB2100	❓ Unknown

Trust	TB3100	❓ Unknown
Turcom / Huion	Interactive Pen Display	✔ Supposed to work
Turcom / Huion	TS-6608	✔ Supposed to work
Turcom / Huion	TS-6580B Pro	✔ Supposed to work
Turcom / Huion	TS-6580W Pro	✔ Supposed to work
Turcom / Huion	TS-6610H Professional Wide	✔ Supposed to work
Turcom / Huion	TS-690	✔ Supposed to work
Turcom / Huion	TS-680	✔ Supposed to work
Turcom / Huion	TS-6540	✔ Supposed to work
UC-Logic / Digipro	DigiPro WP4030	❓ Unknown

UC-Logic / Digipro	WP806U	❓ Unknown
Ugee	HK1060pro	❓ Unknown
Ugee	HK1560	❓ Unknown
Ugee	UG-1910B	❓ Unknown
Ugee	UG-2150	❓ Reported to work with the new drivers released January 2018
Ugee	EX05	❓ Unknown
Ugee	EX07	❓ Unknown
Ugee	G3	❓ Reported to work on windows 7
Ugee	G5	❓ Working with Windows with official drivers installed. No drivers are

		currently available under Linux.
Ugee	M504	? Unknown
Ugee	M708	✕ Reported to be broken, connected strokes
Ugee	M6370	? Unknown
Ugee	M1000L	✕ Reported to be broken
Ugee	Chocolate	? Unknown
Ugee	CV720	? Unknown
Ugee	Rainbow 3	? Unknown
VisTablet	Mini	? Unknown
VisTablet	Mini Plus	? Unknown
VisTablet	VT Original	? Unknown

VisTablet	Realm Pro	❓ Unknown
VisTablet	Realm Graphic	❓ Unknown
VisTablet	VT 12” Touch	❓ Unknown
Wacom	Intuos Draw	✔ Supposed to work
Wacom	Intuos Art	✔ Supposed to work
Wacom	Intuos Photo	✔ Supposed to work
Wacom	Intuos Comic	✔ Supposed to work
Wacom	Intuos 3D	✔ Supposed to work
Wacom	Intuos Pro (S/M/L)	✔ Supposed to work
Wacom	Intuos Pro Paper	✔ Supposed to work
Wacom	Cintiq Pro 13	✔ Supposed to

		work
Wacom	Cintiq Pro 16	✔ Supposed to work
Wacom	Cintiq 13HD	✔ Supposed to work
Wacom	Cintiq 22HD	✔ Supposed to work
Wacom	Cintiq 22HD Touch	✔ Supposed to work
Wacom	Cintiq 27 QHD	✔ Supposed to work
Wacom	Cintiq 27 QHD Touch	✔ Supposed to work
Wacom	Cintiq Companion	✔ Supposed to work
Wacom	Cintiq Companion 2	✔ Supposed to work
Wacom	Cintiq Companion Hybrid	✔ Supported

Wacom	MobileStudio Pro 13	✔ Supported
Wacom	MobileStudio Pro 16	✔ Supported
Wacom	Intuos 5	✔ Supported
Wacom	Intuos 4	✔ Supported
Wacom	Intuos 3	✔ Supported
Wacom	Intuos 2 (XD)	✔ Supposed to work
Wacom	Cintiq 12WX	✔ Supposed to work
Wacom	Cintiq 24HD	✔ Supposed to work
Wacom	Bamboo Create	✔ Supposed to work
Wacom	Bamboo Capture	✔ Supposed to work
Wacom	Bamboo Connect	✔ Supposed to work

Wacom	Bamboo Splash	✔ Supposed to work
Wacom	Bamboo CTL	✔ Supposed to work
Wacom	Bamboo CTH	✔ Supposed to work
Wacom	Bamboo CTE	✔ Supposed to work
Wacom	Bamboo One	✔ Supposed to work
Wacom	Cintiq20 (DTZ)	✔ Supposed to work
Wacom	Cintiq21	✔ Reported to work
Wacom	Intuos (GD)	✔ Supposed to work
Wacom	Graphire2	✔ Supposed to work
Wacom	Graphire (ET)	✔ Supposed to work

Waltop	Venus M	❓ Unknown
Waltop	Media	❓ Unknown
Waltop	Q-Pad	❓ Unknown
XP Pen	Artist 16	✔ Supposed to work
XP Pen	Artist 22	✔ Supposed to work
XP Pen	Artist 22E	✔ Supposed to work
XP Pen	Artist Display 10S	✔ Supposed to work
XP Pen	Star 05 Wireless	✔ Works with the Star 04 driver
XP Pen	Star G540 Game Play	❓ Unknown
XP Pen	Star G430 Game Play	✔ Supposed to work
XP Pen	Star 04 Flash Memory	❓ Unknown

XP Pen	Star 03 Express Keys	✔ Works
XP Pen	Star 02 Touch Hot Keys	? Unknown
XP Pen	Star 01 Pen Tablet	? Unknown
Yiynova	SP 1001 (UC-Logic)	? Unknown
Yiynova	MVP10U	✔ Supported
Yiynova	MVP10U HD	✔ Supposed to work
Yiynova	MVP10U HD+IPS	✔ Supposed to work
Yiynova	DP10U+	✔ Supposed to work
Yiynova	DP10U	✔ Supposed to work
Yiynova	DP10	✔ Supposed to work
Yiynova	DP10S	✔ Supposed to work

Yiynova	DP10HD	✔ Supposed to work
Yiynova	MSP15	✔ Supposed to work
Yiynova	MSP19	✔ Supposed to work
Yiynova	MSP19U	✔ Supposed to work
Yiynova	MSP19U+	✔ Supposed to work
Yiynova	MSP19U+ (V5)	✔ Supposed to work
Yiynova	MVP22U+IPS (V3)	✔ Supposed to work
Yiynova	MVP20U+RH	✔ Supposed to work
Yiynova	MVP22U+DT	✔ Supposed to work

Yiynova	MVP22U+RH	✔ Supposed to work
Yiynova	MJP19	✔ Supposed to work
Yiynova	MKP19	✔ Supposed to work
Yiynova	YA20HD	✔ Supposed to work

See also

Pages you might want to check :

Huion’s krita support topic on deviant art [https://huion.deviantart.com/journal/Problem-with-Krita-Come-On-In-439442607].

List of tablets models and branding [https://digimend.github.io/tablets/].

Main Menu

A list of all of main menu actions and a short description on what they do.

Edit Menu

Undo
Undoes the last action. Shortcut: `Ctrl + Z`

Redo
Redoes the last undone action. Shortcut: `Ctrl + Shift+ Z`

Cut
Cuts the selection or layer. Shortcut: `Ctrl + X`

Copy
Copies the selection or layer. Shortcut: `Ctrl + C`

Cut (Sharp)
This prevents semi-transparent areas from appearing on your cut pixels, making them either fully opaque or fully transparent.

Copy (Sharp)
Same as Cut (Sharp) but then copying instead.

Copy Merged
Copies the selection over all layers. Shortcut: `Ctrl + Shift + C`

Paste
Pastes the copied buffer into the image as a new layer. Shortcut: `Ctrl + V`

Paste at Cursor
Same as paste, but aligns the image to the cursor. Shortcut: `Ctrl + Alt + V`

Paste into New Image
Pastes the copied buffer into a new image. Shortcut: `Ctrl + Shift + N`

Clear

Clear the current layer. Shortcut: `Del`

Fill with Foreground Color

Fills the layer or selection with the foreground color. Shortcut: `Shift + Backspace`

Fill with Background Color

Fills the layer or selection with the background color. Shortcut: `Backspace`

Fill with Pattern

Fills the layer or selection with the active pattern.

Stroke Selected Shapes

Strokes the selected vector shape with the selected brush, will create a new layer.

Stroke Selection

Strokes the active selection using the menu.

File Menu

New
Make a new file. Shortcut: `Ctrl + N`

Open…
Open a previously created file. Shortcut: `Ctrl + O`

Open Recent
Open the recently opened document.

Save
File formats that Krita can save to. These formats can later be opened back up in Krita. Shortcut: `Ctrl + S`

Save As…
Save as a new file. Shortcut: `Ctrl + Shift + S`

Open existing Document as Untitled Document…
Similar to import in other programs.

Export…
Additional formats that can be saved. Some of these formats may not be later imported or opened by Krita.

Import Animation Frames…
Import frames for animation.

Render Animation…
Render an animation with FFmpeg. This is explained on the Render Animation page.

Save Incremental Version
Save as a new version of the same file with a number attached. Shortcut:

`Ctrl + Alt + S`

Save Incremental Backup

Copies and renames the last saved version of your file to a back-up file and saves your document under the original name. Shortcut: `F4`

Create Template from Image…

The `\*.kra` file will be saved into the template folder for future use. All your layers and guides will be saved along!

Create Copy from Current Image

Makes a new document from the current image, so you can easily reiterate on a single image. Useful for areas where the template system is too powerful.

Document Information

Look at the document information. Contains all sorts of interesting information about image, such as technical information or metadata.

Close

Close the view or document. Shortcut: `Ctrl + W`

Close All

Close all views and documents. Shortcut: `Ctrl + Shift + W`

Quit

Close Krita. Shortcut: `Ctrl + Q`

Help Menu

Krita Handbook
: Opens a browser and sends you to the index of this manual. Shortcut: `F1`

Report Bug…
: Sends you to the bugtracker.

Show System Information for Bug Reports
: This is a selection of all the difficult to figure out technical information of your computer. This includes things like, which version of Krita you have, which version your operating system is, and most prudently, what kind of OpenGL functionality your computer is able to provide. The latter varies a lot between computers and due that it is one of the most difficult things to debug. Providing such information can help us figure out what is causing a bug.

About Krita
: Shows you the credits.

About KDE
: Tells you about the KDE community that Krita is part of.

Image Menu

Properties…
: Gives you the image properties.

Image Background Color and Transparency…
: Change the background canvas color.

Convert Current Image Color Space…
: Converts the current image to a new colorspace.

Trim to image size
: Trims all layers to the image size. Useful for reducing filesize at the loss of information.

Trim to Current Layer
: A lazy cropping function. Krita will use the size of the current layer to determine where to crop.

Trim to Selection
: A lazy cropping function. Krita will crop the canvas to the selected area.

Rotate
: Rotate the image.

Shear Image…
: Shear the image.

Mirror Image Horizontally
: Mirror the image on the horizontal axis.

Mirror Image Vertically
: Mirror the image on the vertical axis.

Scale to New Size…
: The resize function in any other program. Shortcut `Ctrl + Alt + I`

Offset Image…
: Offset all layers.

Resize Canvas…
: Change the canvas size. Don’t confuse this with Scale to new size.
Shortcut `Ctrl + Alt + C`

Image Split
: Calls up the Image Split dialog.

Wavelet Decompose…
: Does Wavelet Decompose on the current layer.

Separate Image…
: Separates the image into channels.

Layers Menu

These are the topmenu options are related to Layer Management, check out that page first, if you haven't.

Cut Layer (3.0+)
: Cuts the whole layer rather than just the pixels.

Copy Layer (3.0+)
: Copy the whole layer rather than just the pixels.

Paste Layer (3.0+)
: Pastes the whole layer if any of the top two actions have been taken.

New
: Organizes the following actions:

  Paint Layer
  : Add a new paint layer.

  New layer from visible (3.0.2+)
  : Add a new layer with the visible pixels.

  Duplicate Layer or Mask
  : Duplicates the layer.

  Cut Selection to New Layer
  : Single action for cut+paste.

  Copy Selection to New Layer
  : Single action for copy+paste.

Import/Export
: Organizes the following actions:

Save Layer or Mask
: Saves the Layer or Mask as a separate image.

Save Vector Layer as SVG
: Save the currently selected vector layer as an SVG.

Save Group Layers
: Saves the top-level group layers as single-layer images.

Import Layer
: Import an image as a layer into the current file.

Import as…
: Import an image as a specific layer type. The following layer types are supported:

    - Paint layer
    - Transparency Mask
    - Filter Mask
    - Selection Mask

Convert
: Organizes the following actions:

Convert a layer to…

Convert to Paint Layer
: Convert a mask or vector layer to a paint layer.

Transparency Mask
: Convert a layer to a transparency mask. The image will be converted to grayscale first, and these grayscale values are used to drive the transparency.

Filter Mask
: Convert a layer to a filter mask. The image will be converted to grayscale first, and these grayscale values are used to drive the filter effect area.

Selection Mask
: Convert a layer to a selection mask. The image will be converted to grayscale first, and these grayscale values are used to drive the selected area.

File Layer
: Convert the selected layer in to a file layer. This will open a dialog box, which will ask the user for a location to save the layer as file layer and reference it in place of the original layer. This feature cannot be used if the selected layer is either a clone layer or a file layer.

Convert Group to Animated Layer
: This takes the images in the group layer and makes them into frames of an animated layer.

Convert Layer Color Space
: This only converts the color space of the layer, not the image.

Select (3.0+):
: Organizes the following actions:

All layers
: Select all layers.

Visible Layers
: Select all visible layers.

Invisible Layers
: Select all invisible layers, useful for cleaning up a sketch.

Locked Layers
: Select all locked layers.

Unlocked Layers
: Select all unlocked layers.

Group

Organizes the following actions:

Quick Group (3.0+)
: Adds all selected layers to a group.

Quick Clipping Group (3.0+)
: Adds all selected layers to a group and adds a alpha-inherited layer above it.

Quick Ungroup
: Ungroups the activated layer.

Transform
: Organizes the following actions:

Mirror Layer Horizontally
: Mirror the layer horizontally using the image center.

Mirror Layer Vertically
: Mirror the layer vertically using the image center.

Rotate
: Rotate the layer around the image center.

Scale Layer
: Scale the layer by the given amounts using the given interpolation filter.

Shear Layer
: Shear the layer pixels by the given X and Y angles.

Offset Layer
: Offset the layer pixels by a given amount.

Split
: Organizes the following actions:

Split Alpha

Split the image transparency into a mask. This is useful when you wish to edit the transparency separately.

Split Layer
: Split the layer into given color fields.

Clones Array
: A complex bit of functionality to generate clone-layers for quick sprite making. See Clones Array for more details.

Edit Metadata…
: Each layer can have its own metadata.

Histogram
: Shows a histogram.

  Deprecated since version 4.2: Removed. Use the Histogram Docker instead.

Merge with Layer Below
: Merge a layer down.

Flatten Layer
: Flatten a Group Layer or flatten the masks into any other layer.

Rasterize Layer
: For making vectors into raster layers.

Flatten Image
: Flatten all layers into one. Shortcut `Ctrl + Shift + E`

Layerstyle… (2.9.5+)
: Set the PS-style layerstyle.

Select Menu

Select All

Selects the whole layer. Shortcut `Ctrl + A`

Deselect

Deselects everything (except for active Selection Mask). Shortcut `Ctrl + Shift + A`

Reselect

Reselects the previously deselected selection. Shortcut `Ctrl + Shift + D`

Invert Selection

Inverts the selection. Shortcut `Ctrl + Shift + I`

Convert to Vector Selection

This converts a raster selection to a vector selection. Any layers of transparency there might have been are removed.

Convert to Raster Selection

This converts a vector selection to a raster selection.

Convert Shapes to Vector Selection

Convert vector shape to vector selection.

Convert to Shape

Converts vector selection to vector shape.

Display Selection

Display the selection. If turned off selections will be invisible. Shortcut `Ctrl + H`

Show Global Selection Mask

Shows the global selection as a selection mask in the layers docker. This

is necessary to be able to select it for painting on.

Scale…

Scale the selection.

Select from Color Range…

Select from a certain color range.

Select Opaque

Select all opaque (non-transparent) pixels in the current active layer. If there's already a selection, this will add the new selection to the old one, allowing you to select the opaque pixels of multiple layers into one selection. Semi-transparent (or semi-opaque) pixels will be semi-selected.

Feather Selection…

Feathering in design means to soften sharp borders. So this adds a soft border to the existing selection. Shortcut `Shift + F6`

Grow Selection…

Make the selection a few pixels bigger.

Shrink Selection…

Make the selection a few pixels smaller.

Border Selection…

Take the current selection and remove the insides so you only have a border selected.

Smooth

Make the selection a little smoother. This removes jiggle.

Settings Menu

The Settings Menu houses the configurable options in Krita and where you determine most of the “look and feel” of the application.

Show Dockers

Show Dockers

> Determines whether or not the dockers are visible. This is a nice aid to cleaning up the interface and removing unnecessary “eye-ball clutter” when you are painting. If you’ve got your brush and you know you’re just going to be painting for awhile why not flip the dockers off? You’d be amazed what a difference it makes while you’re working. However, if you know you’re swapping out tools or working with layer or any of the other myriad things Krita lets you do then there’s no point getting caught up in flipping the docks on and off. Use you time wisely!

> **Tip**
>
> This is a great candidate to add to the toolbar so you can just click the dockers on and off and don’t even have to open the menu to do it. See Configure Toolbars… below for more.

Dockers

Krita subdivides the access of many of its features into functional panels called Dockers. Dockers are small windows that can contain, for example, things like the Layer Stack, Color Palette or Brush Presets. Think of them as the painter’s palette, or his water, or his brushkit.

Learning to use dockers effectively is a key concept to optimizing your time

using Krita.

Themes

Krita provides a number of color-themed interfaces or “looks”. The current set of themes are the following:

- Dark (Default)
- Blender
- Bright
- Neutral

There is no easy way to create and share themes. The color themes are defined in the *Share* ▸ *Color Schemes* folder where Krita is downloaded.

Configure Shortcuts

Configuring shortcuts is another way to customize the application to fit you. Whether you are transitioning from another app, like Photoshop or MyPaint, or you think your own shortcut keys make more sense for you then Krita has got you covered. You get to the shortcuts interface through *Settings* ▸ *Configure Krita…* and by choosing the *Keyboard Shortcuts* tab.

To use, just type the *Action* into the Search box you want to assign/reassign the shortcut for. Suppose we wanted to assign the shortcut `Ctrl + G` to the *Action* of Group Layers so that every time we pressed the `Ctrl + G` shortcut a new Layer Group would be created. Use the following steps to do this:

1. Type “Group Layer”.
2. Click on Group Layer and a small inset box will open.
3. Click the Custom radio button.
4. Click on the first button and type the `Ctrl + G` shortcut.
5. Click OK.

From this point on, whenever you press the `Ctrl + G` shortcut you’ll get a new *Group Layer*.

Tip

Smart use of shortcuts can save you significant time and further streamline your workflow.

Manage Resources…

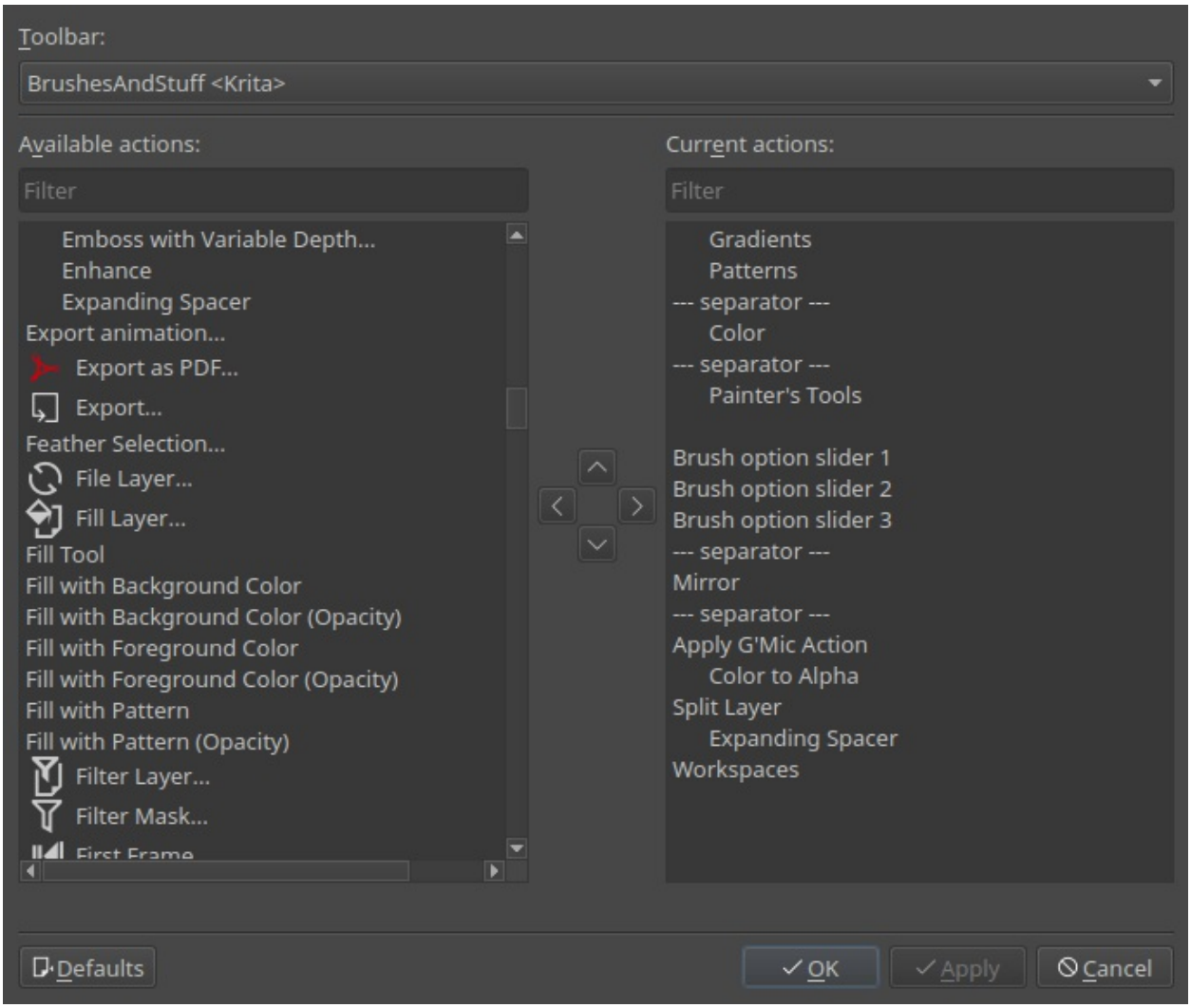
Manage the resources. You can read more about it here.

Switch Application Language…

If you wish to use Krita in a different translation.

Configure Toolbars…

Krita allows you to highly customize the Toolbar interface. You can add, remove and change the order of nearly everything to fit your style of work. To get started, choose *Settings* ▸ *Configure Toolbars…* menu item.



The dialog is broken down into three main sections:

The Toolbar
: Choose to either modify the “Main” or “Brushes and Stuff” toolbars.

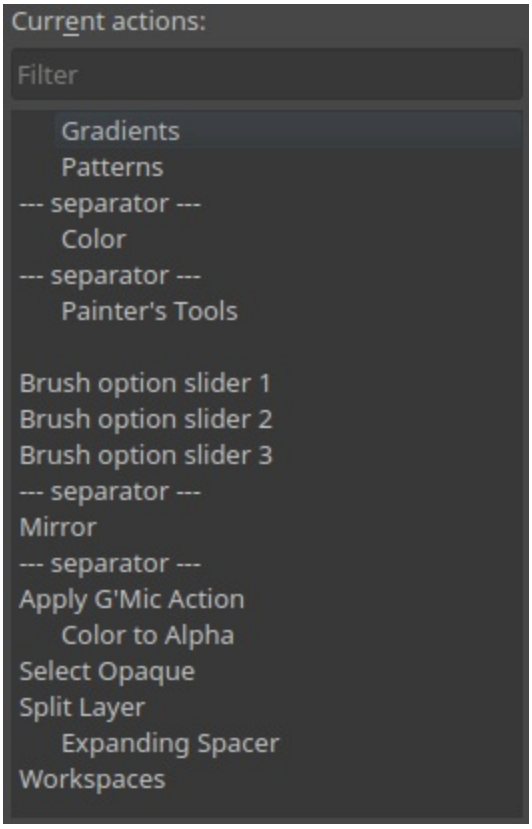
Available Actions:
: All the options that can be added to a toolbar.

Current Actions:
: All the actions currently assigned and the order they are in.

Use the arrows between the *Available* and *Current* actions sections to move items back and forth and up and down in the hierarchy. This type of inclusion/exclusion interface has been around on PCs for decades so we don’t

need to go into great detail regarding its use. What is important though is selecting the correct Toolbar to work on. The Main Toolbar allows you to add items between the *New*, *Open* and *Save* buttons as well as to the right of the *Save* button. The Brushes and Stuff Toolbar, lets you modify anything from the Gradients button over to the right. This is probably where you'll do most of your editing.

Here we've added *Select Opaque*, *Local Selection*, *Transparency Mask*, *Isolate Layer*, *Show Assistant Previews*. This is just an example of a couple of options that are used frequently and might trim your workflow. This is what it looks like in the configuration tool:



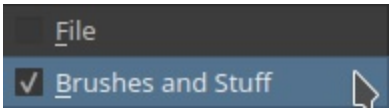
You'll notice that some of the items are text only and some only icons. This is determined by whether the particular item has an associated icon in Krita. You can select anything from the *Available* section and move it to the *Current* one and rearrange to fit your own workflow.

If you add so many that they won't all fit on your screen at once, you will see a small chevron icon appear. Click it and the toolbar expands to show the

remaining items.

Toolbars shown
: Gives a list of toolbars that can be shown.

At this time Krita does not support the ability to create additional toolbars. The ones available are:



Although not really advisable, you can turn them off (but why would you..really?)

New in version 4.2: Finally, Toolbars also can be moved. You can do this by  and dragging the handler at the left side of the toolbar.

Tools Menu

This contains three things.

Scripting

When you have python scripting enabled and have scripts toggled, this is where most scripts are stored by default.

Recording

Deprecated since version 3.0: Macro recording and playing is disabled since 3.0 due to unmaintained code and buggy behaviour leading to crash.

Record a macro. You do this by pressing start, drawing something and then pressing stop. This feature can only record brush strokes. The resulting file is stored as a *.kritarec file.

Macros

Deprecated since version 3.0: Macro recording and playing is disabled since 3.0 due to unmaintained code and buggy behaviour leading to crash.

Play back or edit a krita rec file. The edit can only change the brush preset on strokes or add and remove filters.

View Menu

Show Canvas Only
: Only shows the canvas and what you have configured to show in *Canvas Only* settings. Shortcut `Tab`

Fullscreen mode
: This will hide the system bar. Shortcut `Ctrl + Shift + F`

Wrap Around Mode
: This will show the image as if tiled orthographically. Very useful for tiling 3d textures.

Instant Preview Mode
: Toggle Instant Preview globally. Shortcut `Shift + L`

Soft Proofing
: Activate Soft Proofing. Shortcut `Ctrl + Y`

Out of Gamut Warnings
: See the Soft Proofing page for details. Shortcut `Ctrl + Shift + Y`

Canvas
: Contains view manipulation actions.

Mirror View
: This will mirror the view. Hit the `M` key to quickly activate it. Very useful during painting.

Show Rulers
: This will display a set of rulers.  the rulers after showing them, to change the units.

Rulers Track Pointer
: This adds a little marker to the ruler to show where the mouse is in

relation to them.

Show Guides
: Show or hide the guides.

Lock Guides
: Prevent the guides from being able to be moved by the cursor.

Show Status Bar
: This will show the status bar. The status bar contains a lot of important information, a zoom widget, and the button to switch Selection Display Mode.

Show Grid
: Shows and hides the grid. Shortcut: `Ctrl + Shift + '`

Show Pixel Grid
: Show the pixel grid as configured in the Display Settings.

Snapping
: Toggle the Snapping types.

Show Painting Assistants
: Shows or hides the Assistants.

Show Painting Previews
: Shows or hides the Previews.

Window Menu

A menu completely dedicated to window management in Krita.

New Window
: Creates a new window for Krita. Useful with multiple screens.

New View
: Make a new view of the given document. You can have different zoom or rotation on these.

Workspace
: A convenient access panel to the Workspaces.

Close
: Close the current view.

Close All
: Close all documents.

Tile
: Tiles all open documents into a little sub-window.

Cascade
: Cascades the sub-windows.

Next
: Selects the next view.

Previous
: Selects the previous view.

List of open documents.
: Use this to switch between documents.

Maths Input

Also known as Numerical Input boxes. You can make Krita do simple maths for you in the places where we have number input. Just select the number in a spinbox, or right-click a slider to activate number input. It doesn't do unit conversion yet, but this is planned.

Possible Functions

Addition (Operator: +)
: Just adds the numbers. Usage: `50+100` Output: `150`

Subtraction (Operator: -)
: Just subtracts the last number from the first. Usage: `50-100` Output: `50`

Multiplication (Operator: *)
: Just multiplies the numbers. Usage: `50*100` Output: `5000`

Division (Operator: /)
: Just divides the numbers. Usage: `50/100` Output: `0.5`

Exponent (Operator: ^)
: Makes the last number the exponent of the first and calculates the result. Usage: `2^8` Output: `256`

Sine (Operator: sin())
: Gives you the sine of the given angle. Usage: `sin(50)` Output: `0.76`

Cosine (Operator: cos())
: Gives you the cosine of the given angle. Usage: `cos(50)` Output: `0.64`

Tangent (Operator: tan())
: Gives you the tangent of the given angle. Usage: `tan(50)` Output: `1.19`

Arc Sine (Operator: asin())

Inverse function of the sine, gives you the angle which the sine equals the argument. Usage: `asin(0.76)` Output: `50`

Arc Cosine (Operator: acos())
Inverse function of the cosine, gives you the angle which the cosine equals the argument. Usage: `acos(0.64)` Output: `50`

Arc Tangent (Operator: atan())
Inverse function of the tangent, gives you the angle which the tangent equals the argument. Usage: `atan(1.19)` Output: `50`

Absolute (Operator: abs())
Gives you the value without negatives. Usage: `abs(75-100)` Output: `25`

Exponent (Operator: exp())
Gives you given values using e as the exponent. Usage: `exp(1)` Output: `2.7183`

Natural Logarithm (Operator: ln())
Gives you the natural logarithm, which means it has the inverse functionality to exp(). Usage: `ln(2)` Output: `0.6931`

The following are technically supported but bugged:

Common Logarithm (Operator: log10())
Gives you logarithms of the given value. Usage: `log10(50)` Output: `0.64`

Order of Operations.

The order of operations is a globally agreed upon reading order for interpreting mathematical expressions. It solves how to read an expression like:

```
2+3*4
```

You could read it as 2+3 = 5, and then 5*4 =20. Or you could say 3*4 = 12 and then 2+12 = 14.

The order of operations itself is Exponents, Multiplication, Addition, Subtraction. So we first multiply, and then add, making the answer to the above 14, and this is how Krita will interpret the above.

We can use brackets to specify certain operations go first, so to get 20 from the above expression, we do the following:

```
( 2+3 )*4
```

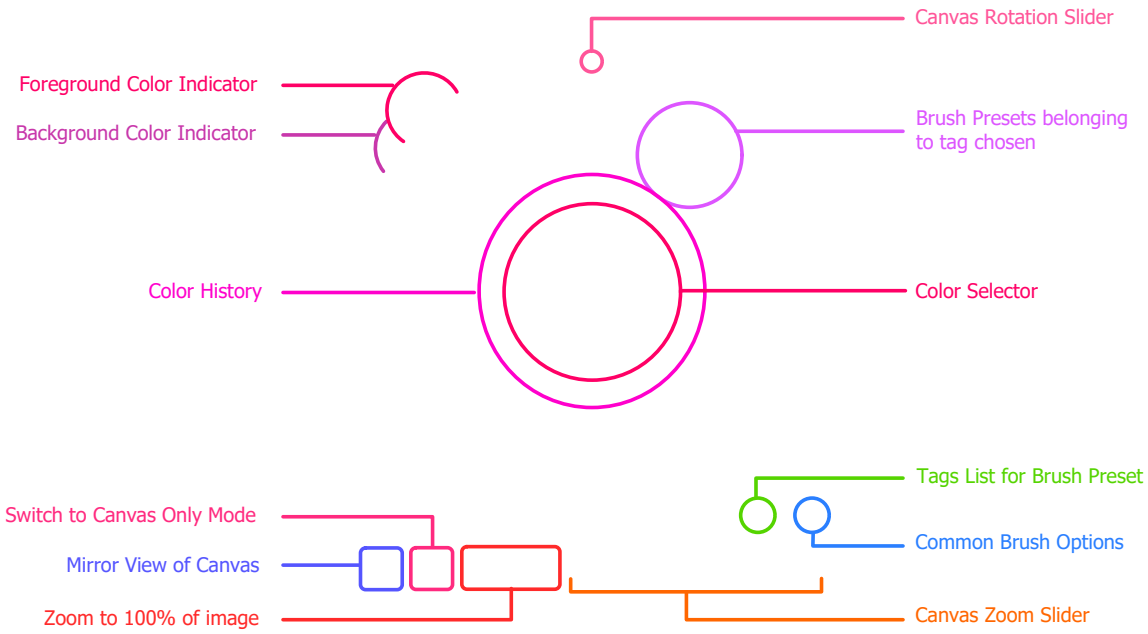
Krita can interpret the brackets accordingly and will give 20 from this.

Errors

Sometimes, you see the result becoming red. This means you made a mistake and Krita cannot parse your maths expression. Simply click the input box and try again.

Pop-up Palette

The Pop-up Palette is a feature unique to Krita amongst the digital painting applications. It is designed to increase productivity and save time of the artists by providing quick access to some of the most frequently used tools and features in Krita. The Pop-up palette can be accessed by  on the canvas. A circular palette similar to what is shown in the image below will spawn at the position your mouse cursor.



As shown in the image above, the pop-up palette has the following tools and quick access shortcuts integrated into it.

- Foreground color and Background color indicators on the top left of the palette.
- A canvas rotation circular slider, which can help the artist quickly rotate the canvas while painting.
- A group of brush presets, based on the tag selected by the artist. By default the **My Favorite** tag is selected. By default only first 10 presets

from the tag are shown, however you can change the number of brush presets shown by changing the value in the Miscellaneous Settings Section of the dialog box.

- Color Selector with which you can select the hue from the circular ring and lightness and saturation from the triangular area in the middle.
- Color history area shows the most recent color swatches that you have used while painting.
- The tag list for brush preset will show you the list of both custom and default tags to choose from, selecting a tag from this list will show the corresponding brush presets in the palette.
- The common brush options such as size, opacity, angle et cetera will be shown when you click the > icon. A dialog box will appear which will have the sliders to adjust the brush options. You can choose which options are shown in this dialog box by clicking on the settings icon.
- The zoom slider allows you to quickly zoom the canvas.
- The 100% button sets the zoom to the 100% of the image size.
- The button with the canvas icon switches to the canvas only mode, where the toolbar and dockers are hidden.
- The button with the mirror icon mirrors the canvas to help you spot the errors in the painting.

Preferences

Krita is highly customizable and makes many settings and options available to customize through the Preferences area. These settings are accessed by going to *Settings* ▸ *Configure Krita…* menu item. On MacOS, the settings are under the topleft menu area, as you would expect of any program under MacOS.

Krita's preferences are saved in the file `kritarc`. This file is located in `%LOCALAPPDATA%\` on Windows, `~/.config` on Linux, and `~/Library/Preferences` on OS X. If you would like to back up your custom settings or synchronize them from one computer to another, you can just copy this file. It even works across platforms!

If you have installed Krita through the Windows store, the kritarc file will be in another location:

`%LOCALAPPDATA%\Packages\49800Krita_`*`RANDOM STRING`*`\LocalCache\Local\kritarc`

Custom shortcuts are saved in a separate file `kritashortcutsrc` which can also be backed up in the same way. This is discussed further in the shortcuts section.

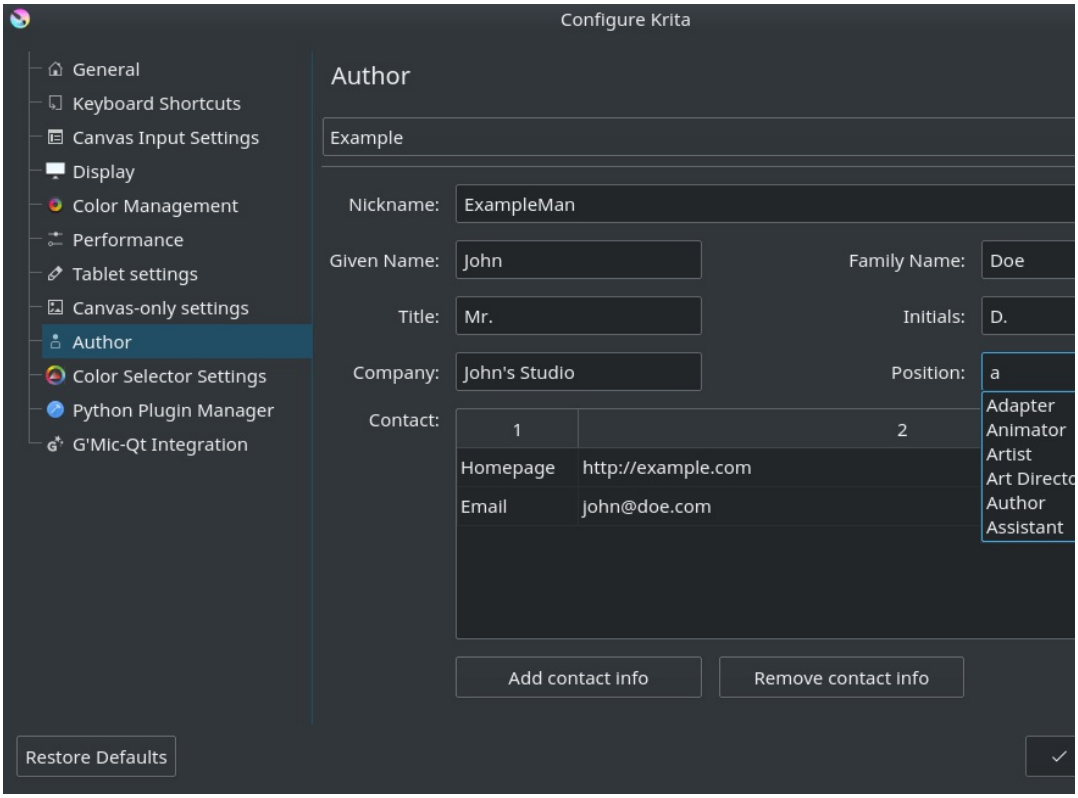
Author Profile Settings

Krita allows creating an author profile that you can use to store contact info into your images.

The main element is the Author page. This page was overhauled massively in 4.0.

By default, it will use the “Anonymous” profile, which contains nothing. To create a new profile, press the “+” button, and write up a name for the author profile.

You can then fill out the fields.



The position field is special in that it has a list of hard coded common artis
suggest.

In older versions of Krita there could only be one of each contact info. In 4.0, you can make as many contact entries as you'd like.

Press *Add Contact Info* to add an entry in the box. By default it will set the type to homepage, because that is the one that causes the least spam. Double  homepage to change the contact type. Double  the "New Contact Info" text to turn it into a line edit to change the value.

Using the new profile

To use a profile for your current drawing, go to *Settings* ▸ *Active Author Profile* and select the name you gave your profile. Then, when pressing *Save* on your current document, you will be able to see your last author profile as the last person who saved it in *File* ▸ *Document Information* ▸ *Author*.

Exporting author metadata to JPEG and PNG

New in version 4.0: The JPEG and PNG export both have *Sign with author data* options. Toggling these will store the Nickname and the *first entry in the contact info* into the metadata of PNG or JPEG.

For the above example in the screenshot, that would result in: `ExampleMan (http://example.com)` being stored in the metadata.

Canvas Input Settings

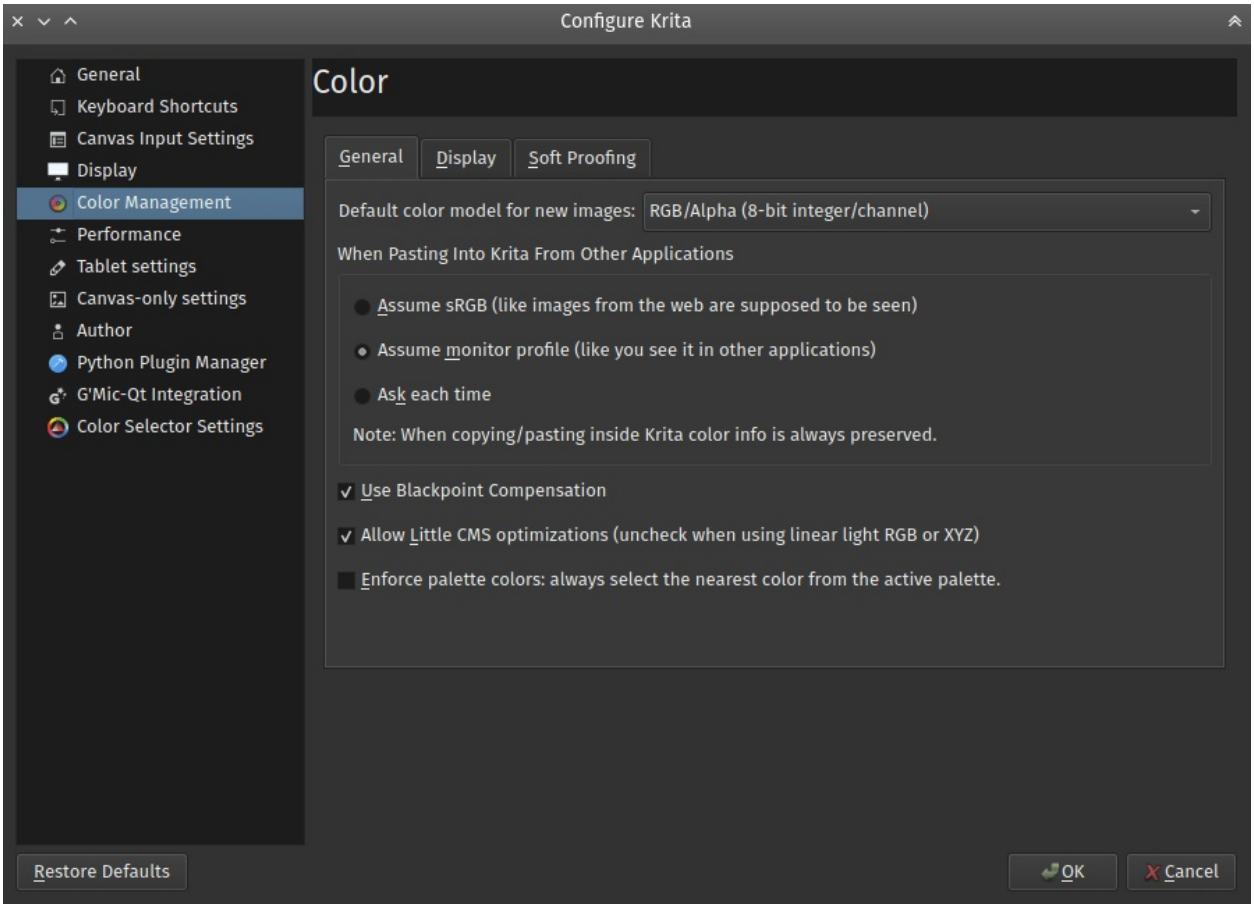
Krita has ways to set mouse and keyboard combinations for different actions. The user can set which combinations to use for a certain Krita command over here. This section is under development and will include more options in future.

Profile
: The user can make different profiles of combinations and save them.

Canvas Only Mode

Canvas Only mode is Krita's version of full screen mode. It is activated by hitting the `Tab` key on the keyboard. Select which parts of Krita will be hidden in canvas-only mode – The user can set which UI items will be hidden in canvas-only mode. Selected items will be hidden.

Color Management Settings



Krita offers extensive functionality for color management, utilising Little CMS [http://www.littlecms.com/] We describe Color Management in a more overall level here: Color Managed Workflow.

General

Default Color Model For New Images

Choose the default model you prefer for all your images.

When Pasting Into Krita From Other Applications

The user can define what kind of conversion, if any, Krita will do to an image that is copied from other applications i.e. Browser, GIMP, etc.

Assume sRGB
: This option will show the pasted image in the default Krita ICC profile of sRGB.

Assume monitor profile
: This option will show the pasted image in the monitor profile selected in system preferences.

Ask each time
: Krita will ask the user each time an image is pasted, what to do with it. This is the default.

Note

When copying and pasting in Krita color information is always preserved.

Use Blackpoint Compensation

This option will turn on Blackpoint Compensation for the conversion. BPC is explained by the maintainer of LCMS as following:

BPC is a sort of “poor man’s” gamut mapping. It basically adjust contrast of images in a way that darkest tone of source device gets mapped to darkest tone of destination device. If you have an image that is adjusted to be displayed on a monitor, and want to print it on a large format printer, you should realize printer can render black significantly darker that the screen. So BPC can do the adjustment for you. It only makes sense on Relative colorimetric intent. Perceptual and Saturation does have an implicit BPC.

Allow LittleCMS optimizations

Uncheck this option when using Linear Light RGB or XYZ.

Enforce palette colors: always select the nearest color from the active palette

By default, palette selection widgets take the current foreground color, compare it to the swatches in its active palette and highlight the swatch that is nearest to the current foreground color.

When *Enforce palette colors* option is checked, it switches the internal color selector into a mode in which, instead of just highlighting the swatch, the current foreground color is replaced with the color of the nearest swatch.

Display

Use System Monitor Profile
: This option when selected will tell Krita to use the ICC profile selected in your system preferences.

Screen Profiles
: There are as many of these as you have screens connected. The user can select an ICC profile which Krita will use independent of the monitor profile set in system preferences. The default is sRGB built-in. On Unix systems, profile stored in $/usr/share/color/icc (system location) or $~/.local/share/color/icc (local location) will be proposed. Profile stored in Krita preference folder, $~/.local/share/krita/profiles will be visible only in Krita.

Rendering Intent
: Your choice of rendering intents is a way of telling Littlecms how you want colors mapped from one color space to another. There are four options available, all are explained on the ICC profiles manual page.

Softproofing options

These allow you to configure the *default* softproofing options. To configure the actual softproofing for the current image, go to *Image* ▸ *Image Properties* ▸ *Softproofing* .

For indepth details about how to use softproofing, check out the page on softproofing.

Color Selector Settings

These settings directly affect Advanced Color Selector Dockers and the same dialog box appears when the user clicks the settings button in that docker as well. They also affect certain hotkey actions.

This settings menu has a drop-down for Advanced Color Selector, and Color Hotkeys.

Advanced Color Selector

These settings are described on the page for the Advanced Color Selector.

Color Hotkeys

These allow you to set the steps for the following actions:

Make Brush Color Darker
: This is defaultly set to K key and uses the *lightness* steps. This uses luminance when possible.

Make Brush Color Lighter
: This is defaultly set to L key and uses the *lightness* steps. This uses luminance when possible.

Make Brush Color More Saturated
: This is defaultly unset and uses the *saturation* steps.

Make Brush Color More Desaturated
: This is defaultly unset and uses the *saturation* steps.

Shift Brushcolor Hue clockwise
: This is defaultly unset and uses the *Hue* steps.

Shift Brushcolor Hue counter-clockwise

This is defaultly unset and uses the *Hue* steps.

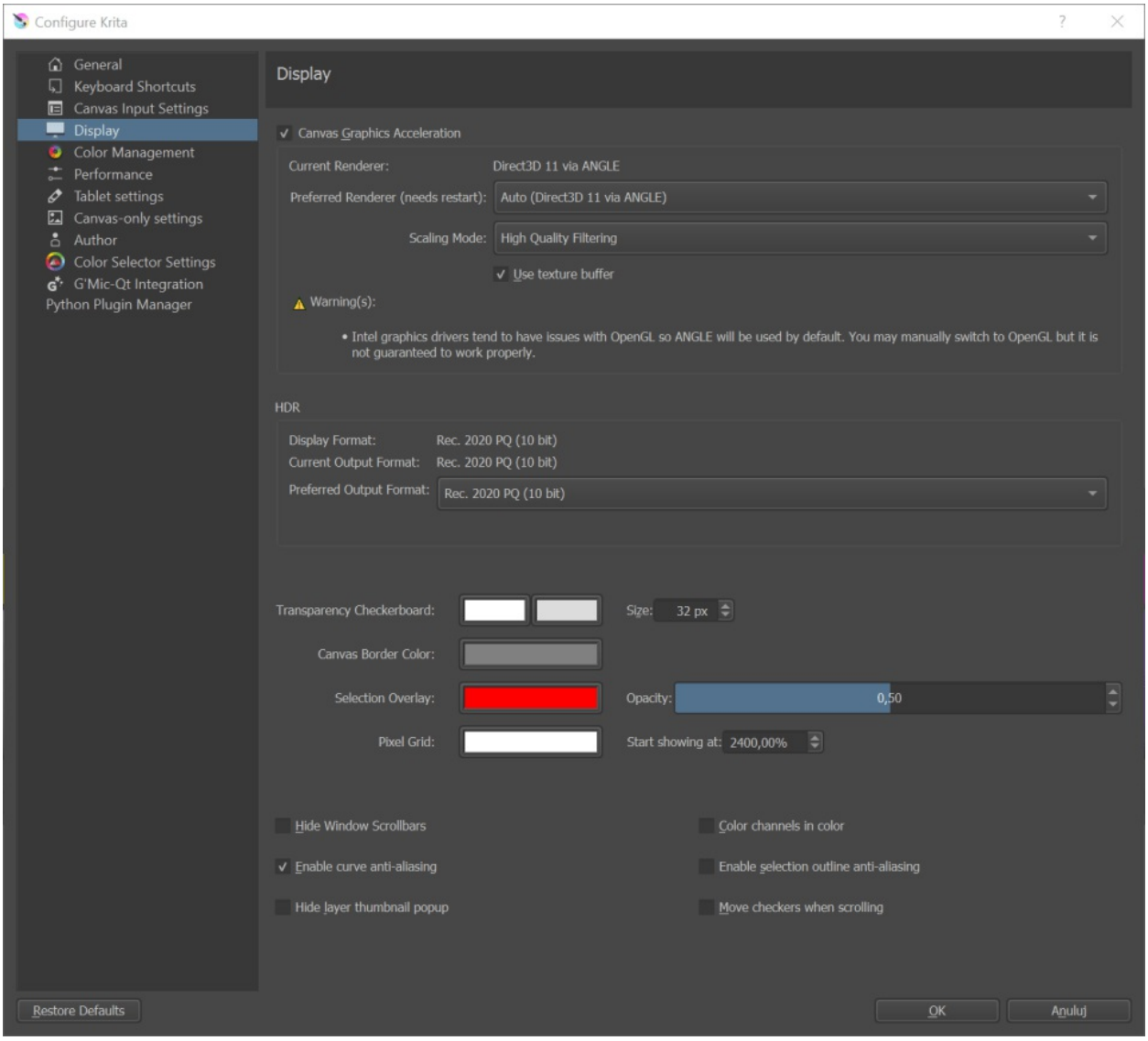
Make Brush Color Redder
This is defaultly unset and uses the *Redder/Greener* steps.

Make Brush Color Greener
This is defaultly unset and uses the *Redder/Greener* steps.

Make Brush Color Yellower
This is defaultly unset and uses the *Bluer/Yellower* steps.

Make Brush Color Bluer
This is defaultly unset and uses the *Bluer/Yellower* steps.

Display Settings



Here various settings for the rendering of Krita can be edited.

OpenGL

For Krita 3.3 or later: Reworded as “*Canvas Graphics Acceleration*”

OpenGL is a bit of code especially for graphics cards. Graphics cards a

dedicate piece of hardware for helping your computer out with graphics calculations, which Krita uses a lot. All modern computer have graphics cards.

For Krita 3.3 or later: On Windows, Krita also supports using Direct3D instead with the help of the ANGLE library. ANGLE works by converting the OpenGL functions that Krita makes use of to the equivalent in Direct3D. It may (or may not) be slower than native OpenGL, but it has better compatibility with typical Windows graphics drivers.

Enable OpenGL **(For Krita 3.3 or later: Reworded as *Canvas Graphics Acceleration*)**

Selecting this checkbox will enable the OpenGL / ANGLE canvas drawing mode. With a decent graphics card this should give faster feedback on brushes and tools. Also the canvas operations like Rotate, Zoom and Pan should be considerably faster.

For Krita 3.3 or later:

Renderer

On Windows: You can switch between native OpenGL or ANGLE Direct3D 11 rendering. The usual recommendation is to leave it as "Auto", which Krita will decide the best to use based on some internal compatibility checking. Changes to this option require a restart of Krita to take effect.

Use Texture Buffer

This setting utilizes the graphics card's buffering capabilities to speed things up a bit. Although for now, this feature may be broken on some AMD/Radeon cards and may work fine on some Intel graphics cards.

Scaling Mode

The user can choose which scaling mode to use while zooming the canvas. The choice here only affects the way the image is displayed during canvas operations and has no effect on how Krita scales an image when a transformation is applied.

Nearest Neighbour

This is the fastest and crudest filtering method. While fast, this results in a large number of artifacts - ‘blockiness’ during magnification, and aliasing and shimmering during minification.

Bilinear Filtering
: This is the next step up. This removes the ‘blockiness’ seen during magnification and gives a smooth looking result. For most purposes this should be a good trade-off between speed and quality.

Trilinear Filtering
: This should give a little better result than Bilinear Filtering.

High Quality Filtering
: Only available when your graphics card supports OpenGL 3.0. As the name suggests, this setting provides the best looking image during canvas operations.

HDR

New in version 4.2: These settings are only available when using Windows.

Since 4.2 Krita can not just edit floating point images, but also render them on screen in a way that an HDR capable setup can show them as HDR images.

The HDR settings will show you the display format that Krita can handle, and the current output format. You will want to set the preferred output format to the one closest to what your display can handle to make full use of it.

Display Format
: The format your display is in by default. If this isn’t higher than 8bit, there’s a good chance your monitor is not an HDR monitor as far as Krita can tell. This can be a hardware issue, but also a graphics driver issue. Check if other HDR applications, or the system HDR settings are configured correctly.

Current Output format
: What Krita is rendering the canvas to currently.

Preferred Output Format
: Which surface type you prefer. This should be ideally the closest to the display format, but perhaps due to driver issues you might want to try other formats. This requires a restart.

Transparency Checkboxes

Krita supports layer transparency. Of course, the nasty thing is that transparency can’t be seen. So to indicate transparency at the lowest layer, we use a checker pattern. This part allows you to configure it.

Size
: This sets the size of the checkers which show up in transparent parts of an image.

Color
: The user can set the colors for the checkers over here.

Move Checkers When Scrolling
: When selected the checkers will move along with opaque elements of an image during canvas Panning, Zooming, etc. Otherwise the checkers remain stationary and only the opaque parts of an image will move.

Canvas Border

Color
: The user can select the color for the canvas i.e. the space beyond a document’s boundaries.

Hide Scrollbars
: Selecting this will hide the scrollbars in all view modes.

Pixel Grid

New in version 4.0.

This allows configuring an automatic pixel-by-pixel grid, which is very useful for doing pixel art.

Color
: The color of the grid.

Start Showing at
: This determines the zoom level at which the pixel grid starts showing, as showing it when the image is zoomed out a lot will make the grid overwhelm the image, and is thus counter productive.

Miscellaneous

Color Channels in Color
: This is supposed to determine what to do when only a single channel is selected in the channels docker, but it doesn’t seem to work.

Enable Curve Anti-Aliasing
: This allows anti-aliasing on previewing curves, like the ones for the circle tool, or the path tool.

Enable Selection Outline Anti-Aliasing
: This allows automatic anti-aliasing on selection. It makes the selection feel less jaggy and more precise.

Hide window scrollbars.
: Hides the scrollbars on the canvas.

Hide Layer thumbnail popup
: This disables the thumbnail that you get when hovering over a layer.

G’Mic Settings

G’Mic or GREYC’s Magic for Image Computing is an opensource filter framework. The G’Mic plugin for Krita exists only on Windows and Linux.

Krita has had G’Mic integration for a long time, but this is its most stable incarnation. On Windows, the plugin is included in the installer and Krita will find it automatically. On Linux, we are making an appimage available for download.

You set it up as following:

1. Download the gmic-krita appimage.
2. Make it executable.
3. Go to *Settings* ▸ *Configure Krita…* ▸ *G’Mic plugin* and set G’MIC to the filepath there.
4. Then restart Krita.

Updates to G’Mic

There is a refresh button at the bottom of the G’Mic window that will update your version. You will need an internet connection to download the latest version.

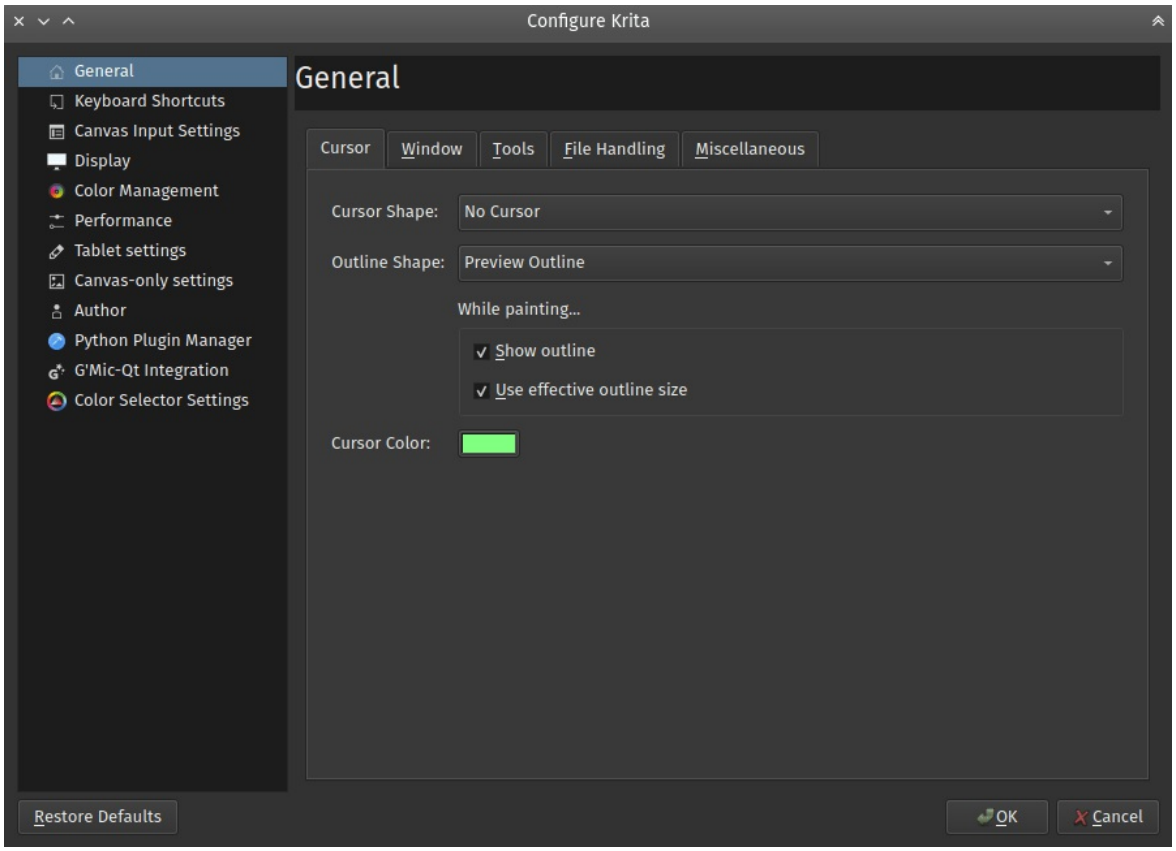
If you have issues downloading the update through the plugin, you can also do it manually. If you are trying to update and get an error, copy the URL that is displayed in the error dialog. It will be to a “.gmic” file. Download it from from your web browser and place the file in one of the following directories.

- Windows : %APPDATA%/gmic/update2XX.gmic
- Linux : $HOME/.config/gmic/update2XX.gmic

Load up the G’Mic plugin and press the refresh button for the version to update.

General Settings

You can access the General Category of the preferences by first going to *Settings* ▸ *Configure Krita…* menu item.



Cursor Settings

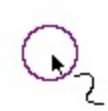
Customize the drawing cursor here:

Cursor Shape

Select a cursor shape to use while the brush tools are used. This cursor will always be visible on the canvas. It is usually set to a type exactly where your pen nib is at. The available cursor types are shown below.

Tool Icon

Shows the currently selected tool icon, even for the freehand brush.



Arrow

Shows a generic cursor.

Crosshair

Shows a precision reticule.

Small circle

Shows a small white dot with a black outline.

No Cursor

Show no cursor, useful for tablet-monitors.

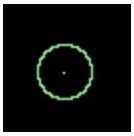
Triangle Right-Handed.

Gives a small white triangle with a black border.

Triangle Left-Handed.
Same as above but mirrored.

Black Pixel
Gives a single black pixel.

White Pixel
Gives a single white pixel.



Outline Shape

Select an outline shape to use while the brush tools are used. This cursor shape will optionally show in the middle of a painting stroke as well. The available outline shape types are shown below. (pictures will come soon)

No Outline
No outline.

Circle Outline
Gives a circular outline approximating the brush size.

Preview Outline
Gives an outline based on the actual shape of the brush.

Tilt Outline
: Gives a circular outline with a tilt-indicator.

While Painting…

Show Outline
: This option when selected will show the brush outline while a stroke is being made. If unchecked the brush outline will not appear during stroke making, it will show up only after the brush stroke is finished. This option works only when Brush Outline is selected as the Cursor Shape.

  Changed in version 4.1: Used to be called “Show Outline When Painting”.

Use effective outline size

  New in version 4.1.

  This makes sure that the outline size will always be the maximum possible brush diameter, and not the current one as affected by sensors such as pressure. This makes the cursor a little less noisy to use.

Cursor Color:
: The default cursor color. This is mixed with the canvas image so that it will usually have a contrasting color, but sometimes this mixing does not work. This is usually due driver problems. When that happens, you can configure a more pleasant color here.

Window Settings

Multiple Document Mode
: This can be either tabbed like **GIMP** or **Painttool Sai**, or subwindows, like **Photoshop**.

Background image
: Allows you to set a picture background for subwindow mode.

Window Background
: Set the color of the subwindow canvas area.

Don’t show contents when moving sub-windows
: This gives an outline when moving windows to work around ugly glitches with certain graphics-cards.

Show on-canvas popup messages
: Whether or not you want to see the on-canvas pop-up messages that tell you whether you are in tabbed mode, rotating the canvas, or mirroring it.

Enable Hi-DPI support
: Attempt to use the Hi-DPI support. It is an option because we are still experiencing bugs on windows.

Allow only one instance of Krita
: An instance is a single entry in your system’s task manager. Turning this option makes sure that Krita will check if there’s an instance of Krita open already when you instruct it to open new documents, and then have your documents opened in that single instance. There’s some obscure uses to allowing multiple instances, but if you can’t think of any, just keep this option on.

Tools Settings

In docker (default)
: Gives you the tool options in a docker.

In toolbar
: Gives you the tool options in the toolbar, next to the brush settings. You can open it with the \ key.

Brush Flow Mode
: In Krita 4.2 the behavior of flow in combination with opacity was changed. This allows you to turn it back to the 4.1 behavior. This will however be removed in future versions.

Switch Control/Alt Selection Modifiers
: This switches the function of the `Ctrl` and `Alt` keys when modifying selections. Useful for those used to Gimp instead of Photoshop, or Lefties without a right `Alt` key on their keyboard.

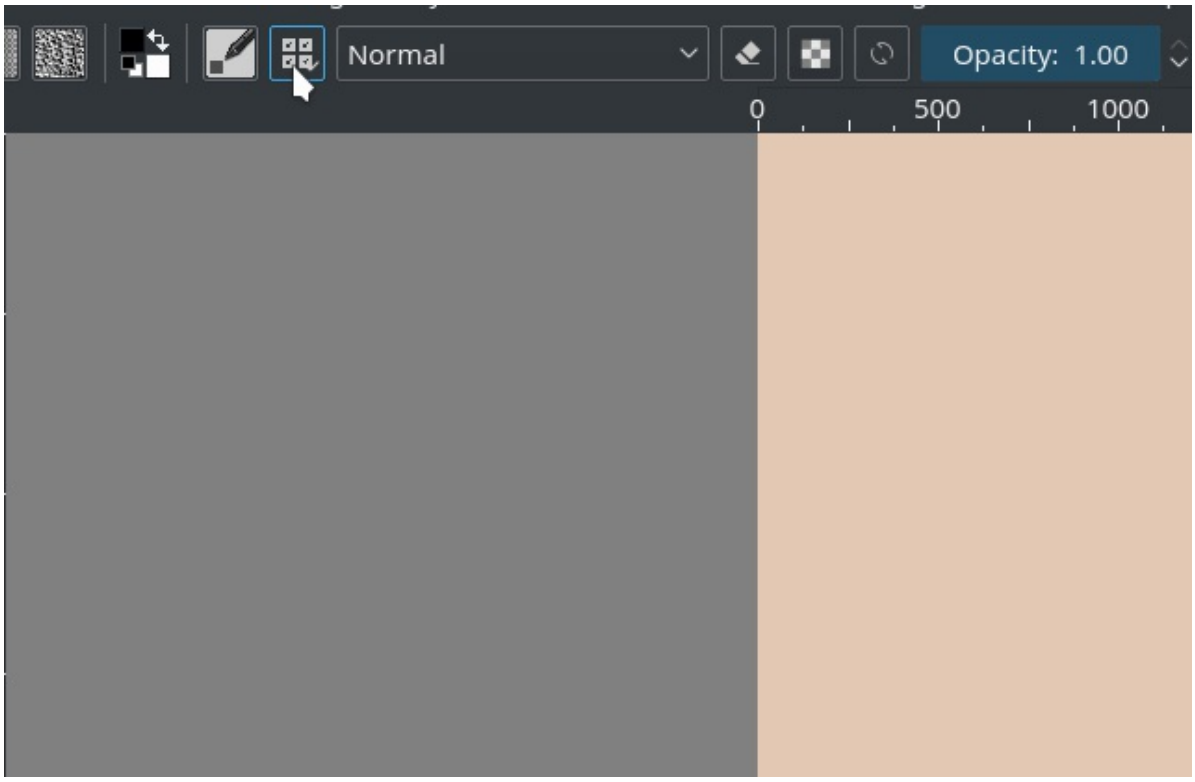
Enable Touchpainting
: This allows finger painting with capacitive screens. Some devices have both capacitive touch and a stylus, and then this can interfere. In that case, just toggle this.

Activate transform tool after pasting

New in version 4.2.

A convenience feature. When enabling this, the transform tool will activate after pasting for quick moving or rotating.

Kinetic Scrolling (Needs Restart)
: This enables kinetic scrolling for scrollable areas.



Kinetic scrolling on the brush chooser drop-down with activation mode

set to *On Click Drag*, with this disabled all of these clicks would lead to a brush being selected regardless of drag motion.

Activation
: How it is activated.

  On Middle-Click Drag
  : Will activate when using the middle mouse button.

  On Touch Drag
  : Will activate if it can recognize a touch event. May not always work.

  On Click Drag
  : Will activate when it can recognize a click event, will always work.

Sensitivity
: How quickly the feature activates, this effective determines the length of the drag.

Hide Scrollbar
: Whether to show scrollbars when doing this.

File Handling

New in version 4.2.

Enable Autosaving
: Determines whether or not Krita should periodically autosave.

Autosave Every
: Here the user can specify how often Krita should autosave the file, you can tick the checkbox to turn it off. For Windows these files are saved in the %TEMP% directory. If you are on Linux it is stored in /home/'username'.

Unnamed autosave files are hidden by default

This determines whether the filename of autosaves has a period prepended to the name. On Linux and Mac OS this is a technique to ensure the file is hidden by default.

Create Backup File

When selected Krita will, upon save, rename the original file as a backup file and save the current image to the original name. The result is that you will have saved the image, and there will be a copy of the image that is saved separately as a backup. This is useful in case of crashes during saves.

Backup File Location

The default location these backups should be stored.

Same Folder as Original File

Store the file in the same folder as the original file was stored.

User Folder

This is the main folder of your computer. On Linux and Mac OS this is the 'Home' folder, on Windows, the 'c:UsersYOUR_USER_NAME' folder (where YOUR_USER_NAME is your windows username).

Temporary File Folder

This stored the file in the temp folder. Temp folders are special folders of which the contents are emptied when you shut down your computer. If you don't particularly care about your backup files and want them to be 'cleaned' automatically, this is the best place. If you want your backup files to be kept indefinitely, this is a wrong choice.

Backup File Suffix

The suffix that will be placed after the full filename. '*filename.kra*' will then be saved as '*filename.kra~*', ensuring the files won't show up in Krita's open file dialog.

Number of Backup Files Kept

Number of backup files Krita keeps, by default this is only one, but this can be up to 99. Krita will then number the backup files.

Compress *.kra files more (slows loading/saving)
: This increases the ZIP compression on the saved Krita files, which makes them lighter on disk, but this takes longer to load.

Use Zip64
: KRA files are ZIP files. Zip64 allows you to use.

Miscellaneous

When Krita starts
: This is the option for handling user sessions. It has the following options:

  Open Default Window
  : This opens the regular empty window with the last used workspace.

  Load Previous Session
  : Load the last opened session. If you have *Save session when Krita closes* toggled, this becomes the last files you had open and the like.

  Show Session Manager
  : Show the session manager directly so you can pick a session.

  New in version 4.1.

Save session when Krita closes
: Save the current open windows, documents and the like into the current session when closing Krita so you can resume where you left off.

  New in version 4.1.

Upon importing Images as Layers, convert to the image color space.
: This makes sure that layers are the same color space as the image, necessary for saving to PSD.

Undo Stack Size
: This is the number of undo commands Krita remembers. You can set the value to 0 for unlimited undos.

Favorite Presets
: This determines the amount of presets that can be used in the pop-up palette.

Hide splash screen on startup.
: This'll hide the splash screen automatically once Krita is fully loaded.

  Deprecated since version 4.1: Deprecated because Krita now has a welcome widget when no canvas is open.

Enable Native File Dialog
: This allows you to use the system file dialog. By default turned off because we cannot seem to get native file dialogues 100% bugfree.

Maximum brush size
: This allows you to set the maximum brush size to a size of up to 10.000 pixels. Do be careful with using this, as a 10.000 pixel size can very quickly be a full gigabyte of data being manipulated, per dab. In other words, this might be slow.

Recalculate animation cache in background.
: Krita will recalculate the cache when you're not doing anything.

  Changed in version 4.1: This is now in the Performance Settings under *Animation Cache*.

Performance Settings

Krita, as a painting program, juggles a lot of data around, like the brushes you use, the colors you picked, but primarily, each pixel in your image. Due to this, how **Krita** organizes where it stores all the data can really speed up **Krita** while painting, just like having an organized artist's workplace can really speed up the painting process in real life.

These preferences allow you to configure **Krita's** organisation, but all do require you to restart **Krita**, so it can do this organisation properly.

RAM

RAM, or Random Access Memory, is the memory your computer is immediately using. The difference between RAM and the hard drive memory can be compared to the difference between having files on your desk and having files safely stored away in an archiving room: The files on your desk as much easier to access than the ones in your archive, and it takes time to pull new files from the archive. This is the same for your computer and RAM. Files need to be loaded into RAM before the computer can really use them, and storing and removing them from RAM takes time.

These settings allow you to choose how much of your virtual desk you dedicate to **Krita**. **Krita** will then reserve them on start-up. This does mean that if you change any of the given options, you need to restart **Krita** so it can make this reservation.

Memory Limit
: This is the maximum space **Krita** will reserve on your RAM on startup. It's both available in percentages and Bytes, so you can specify precisely. **Krita** will not take up more space than this, making it safe for you to run an internet browser or music on the background.

Internal Pool

A feature for advanced computer users. This allows **Krita** to organize the area it takes up on the virtual working desk before putting its data on there. Like how a painter has a standard spot for their canvas, **Krita** also benefits from giving certain data it uses its place (a memory pool), so that it can find them easily, and it doesn't get lost among the other data (memory fragmentation). It will then also not have to spend time finding a spot for this data.

Increasing this, of course, means there's more space for this type of data, but like how filling up your working desk with only one big canvas will make it difficult to find room for your paints and brushes, having a large internal pool will result in **Krita** not knowing where to put the other non-specific data.

On the opposite end, not giving your canvas a spot at all, will result in you spending more time looking for a place where you will put the new layer or that reference you just took out of the storage. This happens for **Krita** as well, making it slower.

This is recommended to be a size of one layer of your image, e.g. if you usually paint on the image of 3000x3000x8bit-ARGB, the pool should be something like 36 MiB.

As **Krita** does this on start-up, you will need to restart **Krita** to have this change affect anything.

Deprecated since version 4.4: This setting was not needed from user side and is deprecated starting from 4.4.

Swap Undo After

Krita also needs to keep all the Undo states on the virtual desk (RAM). Swapping means that parts of the files on the virtual desk get sent to the virtual archive room. This allows **Krita** to dedicate more RAM space to new actions, by sending old Undo states to the archive room once it hits this limit. This will make undoing a little slower, but this can be desirable for the performance of **Krita** overall. This too needs **Krita** to be restarted.

Swapping

File Size Limit
: This determines the limit of the total space **Krita** can take up in the virtual archive room. If **Krita** hits the limit of both the memory limit above, and this Swap File limit, it can't do anything anymore (and will freeze).

Swap File Location
: This determines where the Swap File will be stored on your hard-drive. Location can make a difference, for example, Solid State Drives (SSD) are faster than Hard Disk Drives (HDD). Some people even like to use USB-sticks for the swap file location.

Advanced

Multithreading

Since 4.0, Krita supports multithreading for the animation cache and handling the drawing of brush tips when using the pixel brush.

CPU Limit
: The number of cores you want to allow Krita to use when multithreading.

Frame Rendering Clones Limit
: When rendering animations to frames, Krita multithreads by keeping a few copies of the image, with a maximum determined by the number of cores your processor has. If you have a heavy animation file and lots of cores, the copies can be quite heavy on your machine, so in that case try lowering this value.

Other

Limit frames per second while painting.
: This makes the canvas update less often, which means Krita can spend more time calculating other things. Some people find fewer updates

unnerving to watch however, hence this is configurable.

Debug logging of OpenGL framerate
: Will show the canvas framerate on the canvas when active.

Debug logging for brush rendering speed.
: Will show numbers indicating how fast the last brush stroke was on canvas.

Disable vector optimizations (for AMD CPUs)
: Vector optimizations are a special way of asking the CPU to do maths, these have names such as SIMD and AVX. These optimizations can make Krita a lot faster when painting, except when you have an AMD CPU under Windows. There seems to be something strange going on there, so just deactivate them then.

Enable progress reporting
: This allows you to toggle the progress reporter, which is a little feedback progress bar that shows up in the status bar when you let Krita do heavy operations, such as heavy filters or big strokes. The red icon next to the bar will allow you to cancel your operation. This is on by default, but as progress reporting itself can take up some time, you can switch it off here.

Performance logging
: This enables performance logging, which is then saved to the `Log` folder in your `working directory`. Your working directory is where the autosave is saved at as well.

  So for unnamed files, this is the `$HOME` folder in Linux, and the `%TEMP%` folder in Windows.

Animation Cache

New in version 4.1.

The animation cache is the space taken up by animation frames in the memory of the computer. A cache in this sense is a cache of precalculated

images.

Playing back a video at 25 FPS means that the computer has to precalculate 25 images per second of video. Now, video playing software is able to do this because it really focuses on this one single task. However, Krita as a painting program also allows you to edit the pictures. Because Krita needs to be able to do this, and a dedicated video player doesn't, Krita cannot do the same kind of optimizations as a dedicated video player can.

Still, an animator does need to be able to see what kind of animation they are making. To do this properly, we need to decide how Krita will regenerate the cache after the animator makes a change. There's fortunately a lot of different options how we can do this. However, the best solution really depends on what kind of computer you have and what kind of animation you are making. Therefore in this tab you can customize the way how and when the cache is generated.

Cache Storage Backend

In-memory

Animation frame cache will be stored in RAM, completely without any limitations. This is also the way it was handled before 4.1. This is only recommended for computers with a huge amount of RAM and animations that must show full-canvas full resolution 6k at 25 fps. If you do not have a huge amount (say, 64GiB) of RAM, do *not* use this option (and scale down your projects).

Warning

Please make sure your computer has enough RAM *above* the amount you requested in the *General* tab. Otherwise you might face system freezes.

- For 1 second of FullHD @ 25 FPS you will need 200 extra MiB of Memory.
- For 1 second of 4K UltraHD@ 25 FPS, you will need 800 extra MiB of Memory.

On-disk

Animation frames are stored in the hard disk in the same folder as the swap file. The cache is stored in a compressed way. A little amount of extra RAM is needed.

Since data transfer speed of the hard drive is slow, you might want to limit the *Cached Frame Size* to be able to play your video at 25 fps. A limit of 2500 px is usually a good choice.

Cache Generation Options

Limit Cached Frame Size

Render scaled down version of the frame if the image is bigger than the provided limit. Make sure you enable this option when using On-Disk storage backend, because On-Disk storage is a little slow. Without the limit, there's a good chance that it will not be able to render at full speed. Lower the size to play back faster at the cost of lower resolution.

Use Region Of Interest

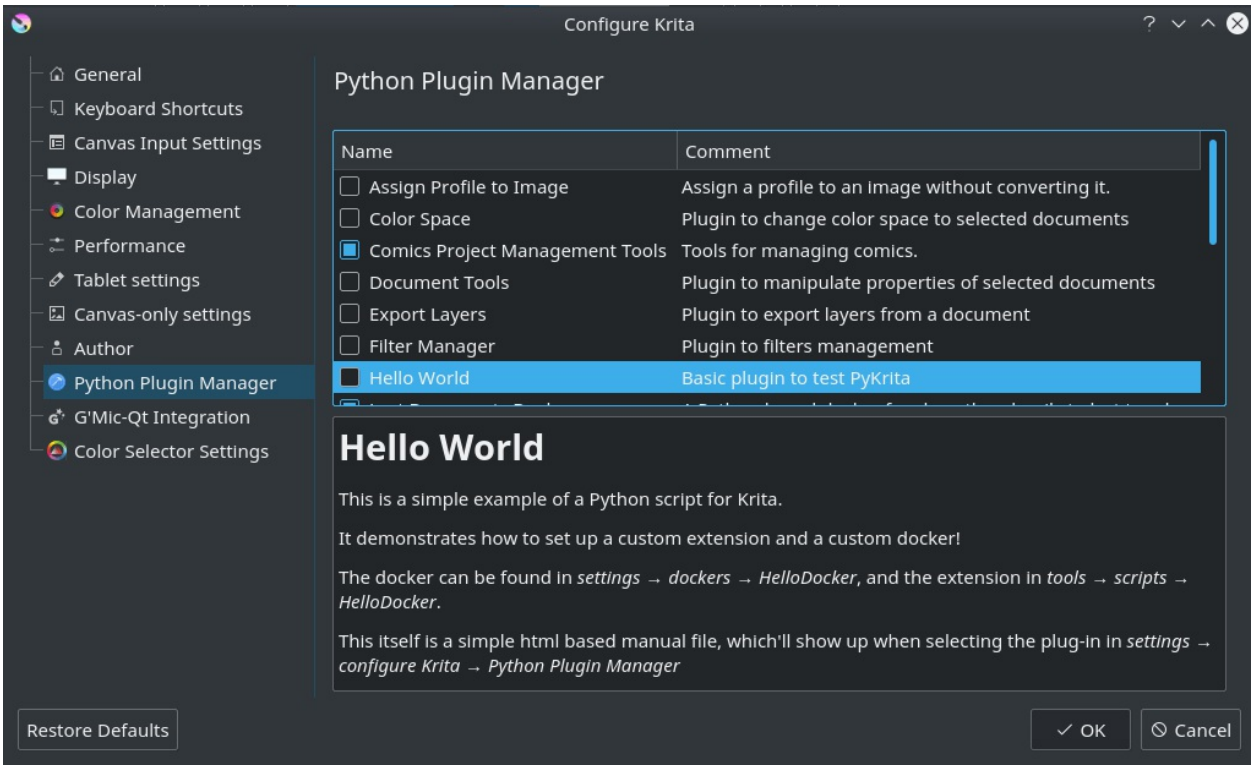
We technically only need to use the section of the image that is in view. Region of interest represents that section. When the image is above the configurable limit, render only the currently visible part of it.

Enable Background Cache Generation

This allows you to set whether the animation is cached for playback in the background (that is, when you're not using the computer). Then, when animation is cached when pressing play, this caching will take less long. However, turning off this automatic caching can save power by having your computer work less.

Python Plugin Manager

This is part of Krita's python support.



The python plugin manager can be accessed from *Settings* ▸ *Configure Krita…* ▸ *Python Plugin Manager*. It allows you decide which of the Python Plugins are active.

It will show you a list of python plugins Krita has found, as well as their description. By default, Python Plugins are disabled, because many python scripts are autostarted, so this ensures only the ones you want to run are being run.

You can use the checkboxes to toggle them. A restart is required to complete switching off or on the python plugin.

If you  a plugin, and the plugin has a manual, Krita will display it in the

box at the bottom.

For more information on python, check the python scripting category.

Shortcut Settings

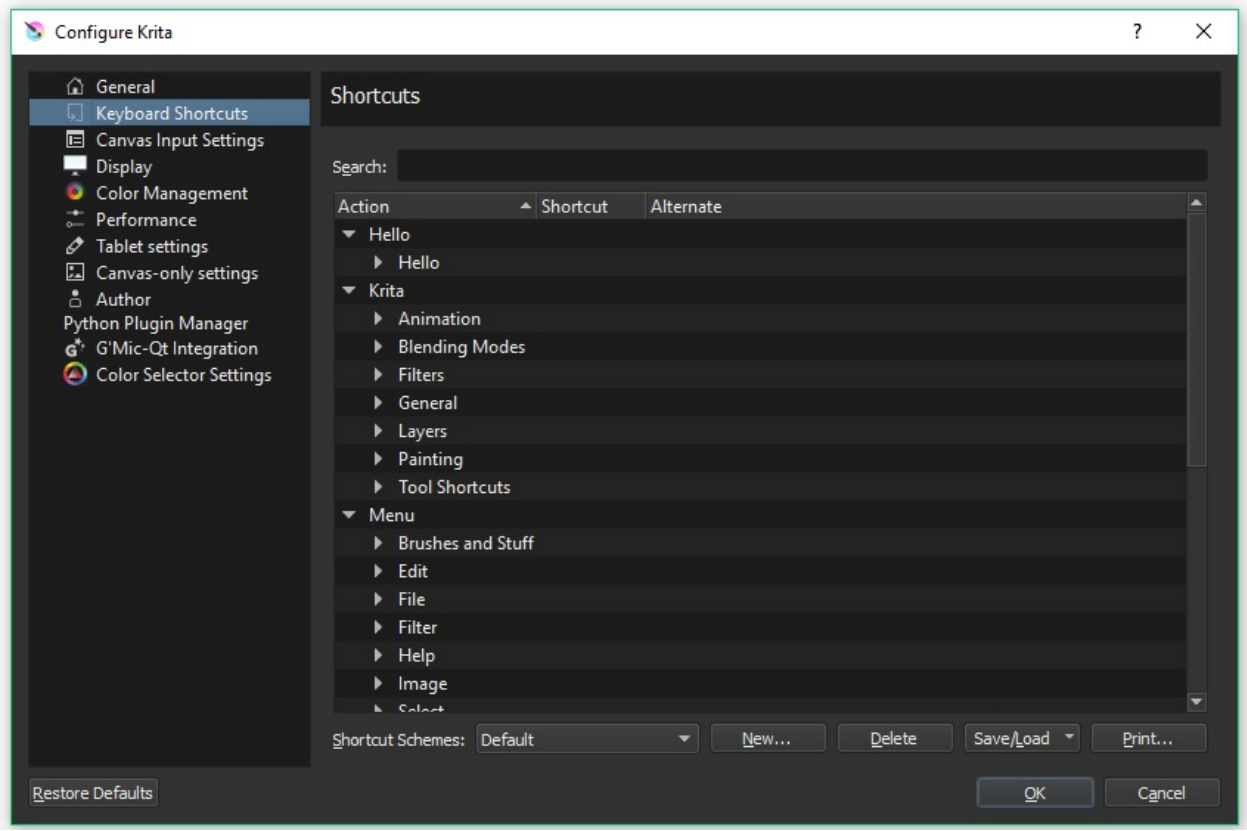
Most of Krita's shortcuts are configured in the menu section *Settings* ▸ *Configure Krita…* ▸ *Shortcuts*. The shortcuts configured here are simple key combinations, for example the `Ctrl + X` shortcut to cut. Shortcuts can also be sequences of key combinations (e.g. `Shift + S` shortcut then the `B` key). Krita also has a special interface for configuring the mouse and stylus events sent to the canvas, found under Canvas Input Settings.

Menu Items

Search bar
: Entering text here will search for matching shortcuts in the shortcut list.

Shortcut List
: Shortcuts are organized into sections. Each shortcut can be given a primary and alternate key combination.

Load/Save Shortcuts Profiles
: The bottom row of buttons contains commands for exporting and import keyboard shortcuts.



Configuration

Primary and alternate shortcuts

Each shortcut is assigned a default, which may be empty. The user can assign up to two custom shortcuts, known as primary and alternate shortcuts. Simply click on a “Custom” button and type the key combination you wish to assign to the shortcut. If the key combination is already in use for another shortcut, the dialog will prompt the user to resolve the conflict.

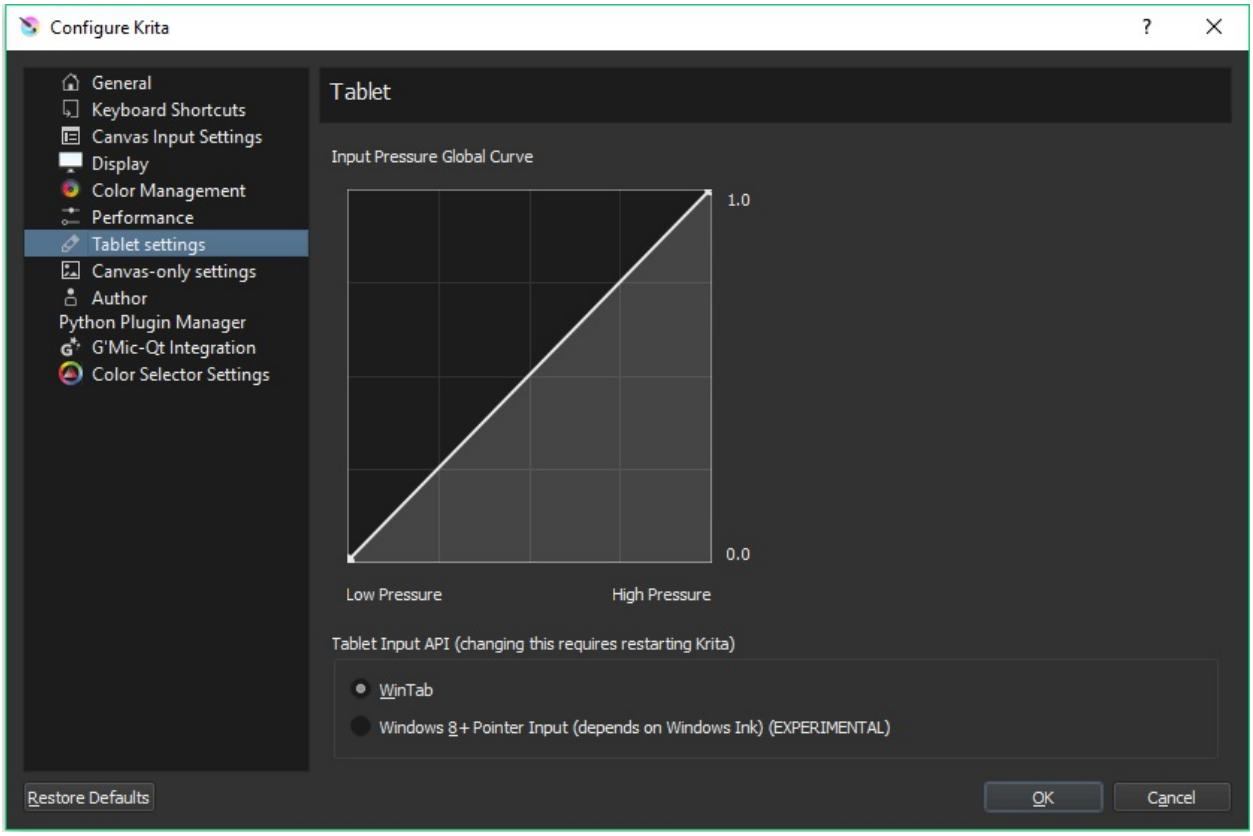
Shortcut schemes

Many users migrate to Krita from other tools with different default shortcuts. Krita users may change the default shortcuts to mimic these other programs. Currently, Krita ships with defaults for Photoshop and Paint Tool Sai. Additional shortcut schemes can be placed in the ~/.config/krita/input/ folder.

Saving, loading and sharing custom shortcuts

Users may wish to export their shortcuts to use across machines, or even share with other users. This can be done with the save/load drop-down. Note: the shortcuts can be saved and overridden manually by backingup the text file kritashortcutsrc located in ~/.config/krita/. Additionally, the user can export a custom shortcut scheme file generated by merging the existing scheme defaults with the current customizations.

Tablet Settings



Tablet

Input Pressure Global Curve : This is the global curve setting that your tablet will use in Krita. The settings here will make your tablet feel soft or hard globally.

Use Mouse Events for Right and Middle clicks.

Some tablet devices don’t tell us whether the side buttons on a stylus. If you have such a device, you can try activate this workaround. Krita will try to read right and middle-button clicks as if they were coming from a mouse instead of a tablet. It may or may not work on your device (depends on the tablet driver implementation). After changing this option Krita should be restarted.

New in version 4.2.

For Krita 3.3 or later:Tablet Input API

On Windows 8 or above only.

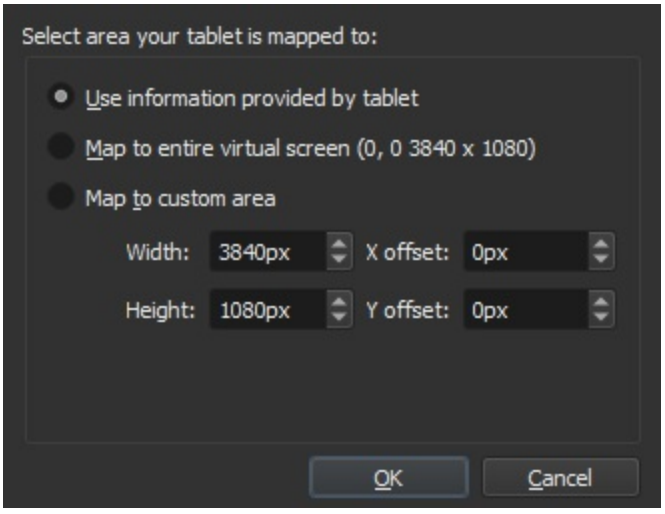
WinTab

Use the WinTab API to receive tablet pen input. This is the API being used before Krita 3.3. This option is recommended for most Wacom tablets.

Windows 8+ Pointer Input

Use the Pointer Input messages to receive tablet pen input. This option depends on Windows Ink support from the tablet driver. This is a relatively new addition so it's still considered to be experimental, but it should work well enough for painting. You should try this if you are using an N-Trig device (e.g. recent Microsoft Surface devices) or if your tablet does not work well with WinTab.

Advanced Tablet Settings for WinTab



When using multiple monitors or using a tablet that is also a screen, Krita will get conflicting information about how big your screen is, and sometimes if it has to choose itself, there will be a tablet offset. This window allows you to select the appropriate screen resolution.

Use Information Provided by Tablet

Use the information as given by the tablet.

Map to entire virtual screen
: Use the information as given by Windows.

Map to Custom Area
: Type in the numbers manually. Use this when you have tried the other options. You might even need to do trial and error if that is the case, but at the least you can configure it.

  If you have a dual monitor setup and only the top half of the screen is reachable, you might have to enter the total width of both screens plus the double height of your monitor in this field.

New in version 4.2: To access this dialog in Krita versions older than 4.2, you had to do the following:

1. Put your stylus away from the tablet.
2. Start Krita without using a stylus, that is using a mouse or a keyboard.
3. Press the `Shift` key and hold it.
4. Touch a tablet with your stylus so Krita would recognize it.

If adjusting this doesn't work, and if you have a Wacom tablet, an offset in the canvas can be caused by a faulty Wacom preference file which is not removed or replaced by reinstalling the drivers.

To fix it, use the "Wacom Tablet Preference File Utility" to clear all the preferences. This should allow Krita to detect the correct settings automatically.

Warning

Clearing all wacom preferences will reset your tablet's configuration, thus you will need to recalibrate/reconfigure it.

Tablet Tester

New in version 4.1.

This is a special feature for debugging tablet input. When you click on it, it will open a window with two sections. The left section is the **Drawing Area** and the right is the **Text Output**.

If you draw over the Drawing Area, you will see a line appear. If your tablet is working it should be both a red and blue line.

The red line represents mouse events. Mouse events are the most basic events that Krita can pick up. However, mouse events have crude coordinates and have no pressure sensitivity.

The blue line represents the tablet events. The tablet events only show up when Krita can access your tablet. These have more precise coordinates and access to sensors like pressure sensitivity.

Important

If you have no blue line when drawing on the lefthand drawing area, Krita cannot access your tablet. Check out the page on drawing tablets for suggestions on what is causing this.

When you draw a line, the output on the right will show all sorts of text output. This text output can be attached to a help request or a bug report to figure out what is going on.

External Links

David Revoy wrote an indepth guide on using this feature to maximum advantage [https://www.davidrevoy.com/article182/calibrating-wacom-stylus-pressure-on-krita].

Render Animation

Render animation allows you to render your animation to an image sequence, `.gif`, `.mp4`, `.mkv`, or `.ogg` file. It replaces *Export Animation*.

For rendering to an animated file format, Krita will first render to a PNG sequence and then use FFmpeg, which is really good at encoding into video files, to render that sequence to an animated file format. The reason for this two-step process is that animation files can be really complex and really big, and this is the best way to allow you to keep control over the export process. For example, if your computer has a hiccup, and one frame saves out weird, first saving the image sequence allows you to only resave that one weird frame before rendering.

This means that you will need to find a good place to stick your frames before you can start rendering. If you only do throwaway animations, you can use a spot on your hard-drive with enough room and select *Delete Sequence After Rendering*.

Image Sequence

Base Name
: The base name of your image sequence. This will get suffixed with a number depending on the frame.

File Format
: The file format to export the sequence to. When rendering we enforce PNG. The usual export options can be modified with ….

Render Location
: Where you render the image sequence to. Some people prefer to use a flash-drive or perhaps a harddrive that is fast.

First Frame
: The first frame of the range of frames you wish to adjust. Automatically

set to the first frame of your current selection in the timeline. This is useful when you only want to re-render a little part.

Last Frame
: As above, the last frame of the range of frames you wish to adjust. Automatically set to the last frame of your current selection in the timeline.

Naming Sequence starts with
: The frames are named by using *Base Name* above and adding a number for the frame. This allows you to set where the frame number starts, so rendering from 8 to 10 with starting point 3 will give you images named 11 and 15. Useful for programs that don't understand sequences starting with 0, or for precision output.

Render Animation

Render As
: The file format to render to. All except GIF have extra options that can be manipulated via ….

File
: Location and name of the rendered animation.

FFmpeg
: The location where your have FFmpeg. If you don't have this, Krita cannot render an animation. For proper GIF support, you will need FFmpeg 2.6, as we use its palettegen functionality.

Delete Sequence After Rendering
: Delete the prerendered image sequence after done rendering. This allows you to choose whether to try and save some space, or to save the sequence for when encoding fails.

Warning

Krita currently does not support rendering video with transparent elements,

and will instead render them as black. To combat this, you can add in a fully colored, opaque layer at the bottom of the file before rendering.

Setting Up Krita for Exporting Animations

You will need to download an extra application and link it in Krita for it to work. The application is pretty big (50MB), so the Krita developers didn't want to bundle it with the normal application. The software that we will use is free and called FFmpeg. The following instructions will explain how to get it and set it up. The setup is a one-time thing so you won't have to do it again.

Step 1 - Downloading FFmpeg

For Windows

Open the FFmpeg download page. [https://www.gyan.dev/ffmpeg/builds/]

Go to *release* section and choose the download link that says `ffmpeg-release-essentials.zip`.

Note

Don't download the file which filename contains the word `shared`. It won't work with Krita.

If the filename ends with `.7z`, you can still use it, but then you need to have a program that can open 7zip archives (for example 7zip itself [https://www.gyan.dev/ffmpeg/builds/]). In case of a `.zip` file, you can open it just using the Windows file browser.

For OSX

Please see the section above. However, FFmpeg is obtained from here [https://evermeet.cx/ffmpeg/] instead. Just pick the big green button on the left under

the FFmpeg heading. You will also need an archiving utility that supports .7z, since FFmpeg provides their OSX builds in .7z format. If you don't have one, try something like Keka [https://www.kekaosx.com].

Alternatively you can find the smaller text under the big green button that says *Download as ZIP*. Then you should be able to extract it just using Finder.

For Linux

FFmpeg can be installed from the repositories on most Linux systems. Version 2.6 is required for proper GIF support, as we use the palettegen functionality.

Step 2 - Unzipping and Linking to Krita

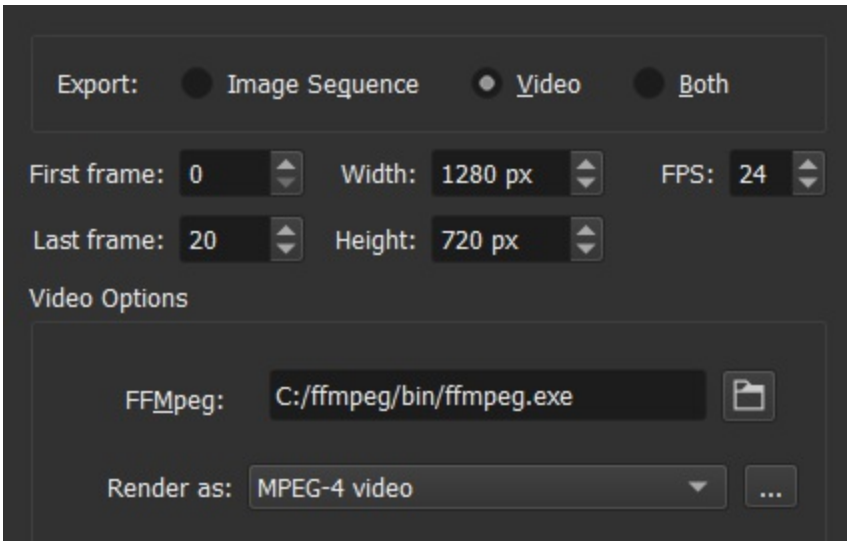
For Windows

Once you've downloaded, go to the file location. Right click on the FFmpeg file, and select `Extract All...`. Select the file destination, and rename the file to 'ffmpeg'.

Hint

It is easiest to save the file under C: drive, but any location is fine.

Open Krita back up and go to *File* ▸ *Render Animation*.... Under `Export > Video`, click the file icon next to FFmpeg. Select this file `C:/ffmpeg/bin/ffmpeg.exe` and click *OK*.



Tip

If you have saved FFmpeg to a different location, choose `<ffmpeg location>/ffmpeg/bin/ffmpeg.exe`.

For OSX

After downloading FFmpeg, you just need to extract it and then simply point to it in the FFmpeg location in Krita like `/Users/user/Downloads/ffmpeg` (assuming you downloaded and extracted the .7z file to /Users/user/Downloads).

For Linux

FFmpeg is, if installed from the repositories, usually found in `/usr/bin/ffmpeg`.

Step 3 - Testing out an animation

ffmpeg.exe is what Krita uses to do all of its animation export magic. Now that it is hooked up, let us test it out.

Let’s make an animated GIF. In the Render Animation dialog, change the

Render As field to “GIF image”. Choose the file location where it will save with the “File” menu below. I just saved it to my desktop and called it “*export.gif*”. When it is done, you should be able to open it up and see the animation.

Warning

By default, FFmpeg will render MP4 files with a too new codec, which means that Windows Media Player won’t be able to play it. So for Windows, select “baseline” for the profile instead of “high422” before rendering.

Note

OSX does not come with any software to play MP4 and MKV files. If you use Chrome for your web browser, you can drag the video file into that and the video should play. Otherwise you will need to get a program like VLC to see the video.

Resource Management

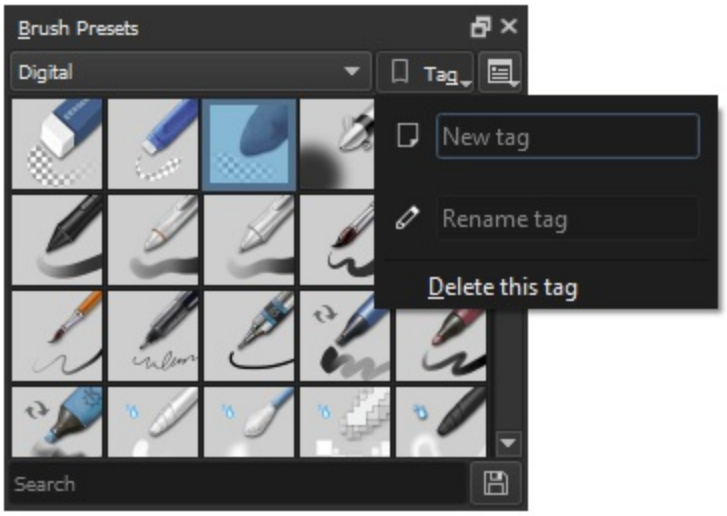
Resources are pluggable bits of data, like brush presets or patterns. Krita has variety of resources and also has a good resource management starting from 2.9, making it really easy for artists to share and collate all the resources together.

Bundles

Starting from 2.9 Krita has a new format to manage resources it is called ‘’Bundles’’, a bundle is just a compressed file containing all the resources together.

Tags

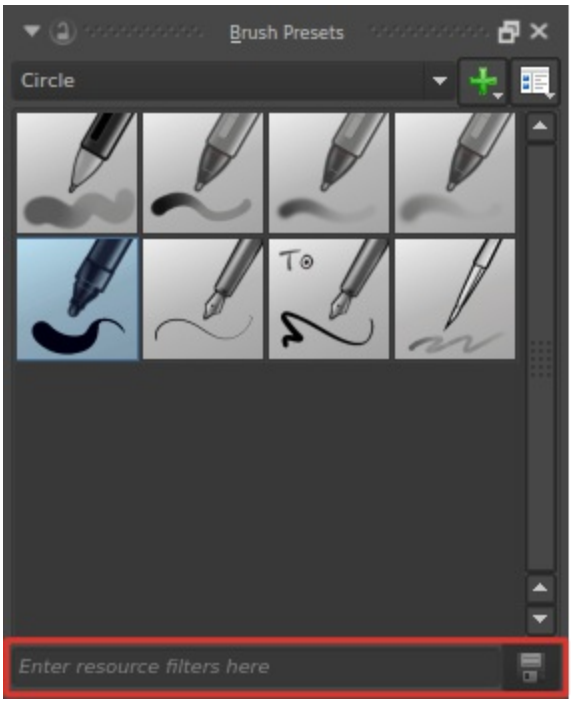
Krita also has a robust tagging system for you to manage the resources on the fly while painting. All Krita resources can be tagged. These tags can be added via the resource manager, but also via the respective dockers such as brush preset docker, pattern docker etc. You can  the plus icon in the docker and add a tag name. In addition to adding you can rename and delete a tag as well.



- Resources can belong to one or more tags. For example, you may have a Brush Preset of a favorite **Ink Pen** variant and have it tagged so it shows in up in your Inking, Painting, Comics and Favorites groups of brushes.
- Brushes in the “Predefined” tab of the Brush Settings Editor can be also tagged and grouped for convenience.

Filtering

Some dockers, for example the brush preset docker as shown below, have a resource filter, which functions like a powerful search bar for the resource in question.



You can enter brush name, tag name to quickly pull up a list of brush presets you want. When you select any tag from the tag drop-down and want to include brush presets from other tags as well then you can add filters the following way:

- To filter based on the partial, case insensitive name of the resources you can add `partialname` or `!partialname`.
- To include other Tags type the respective name of the tag in square brackets like this `[Tagname]` or to exclude a tag type `![Tagname]`.

- For case sensitive matching of preset name type `"Preset name"` or `! "Preset name"` to exclude.

An incredibly quick way to save a group or brushes into a tag is to:

1. Create a new tag by  on the green plus sign. This will empty out the contents of the Brush Preset docker.
2. Use the *Resource Filter* at the bottom of the *Brush Presets* dock or *Brush Settings Editor* to type in what you want to group. For instance: if you type **Pencil** in the filter box you will get all Brush Presets with **Pencil** somewhere in their name. Now you have all the Pencil-related Brush Presets together in one place.
3. To finish, click the *Save* button (small disk icon to the right of the *Resource Filter* box) or press the `Enter` key and all the items will be saved with the new tag you created.

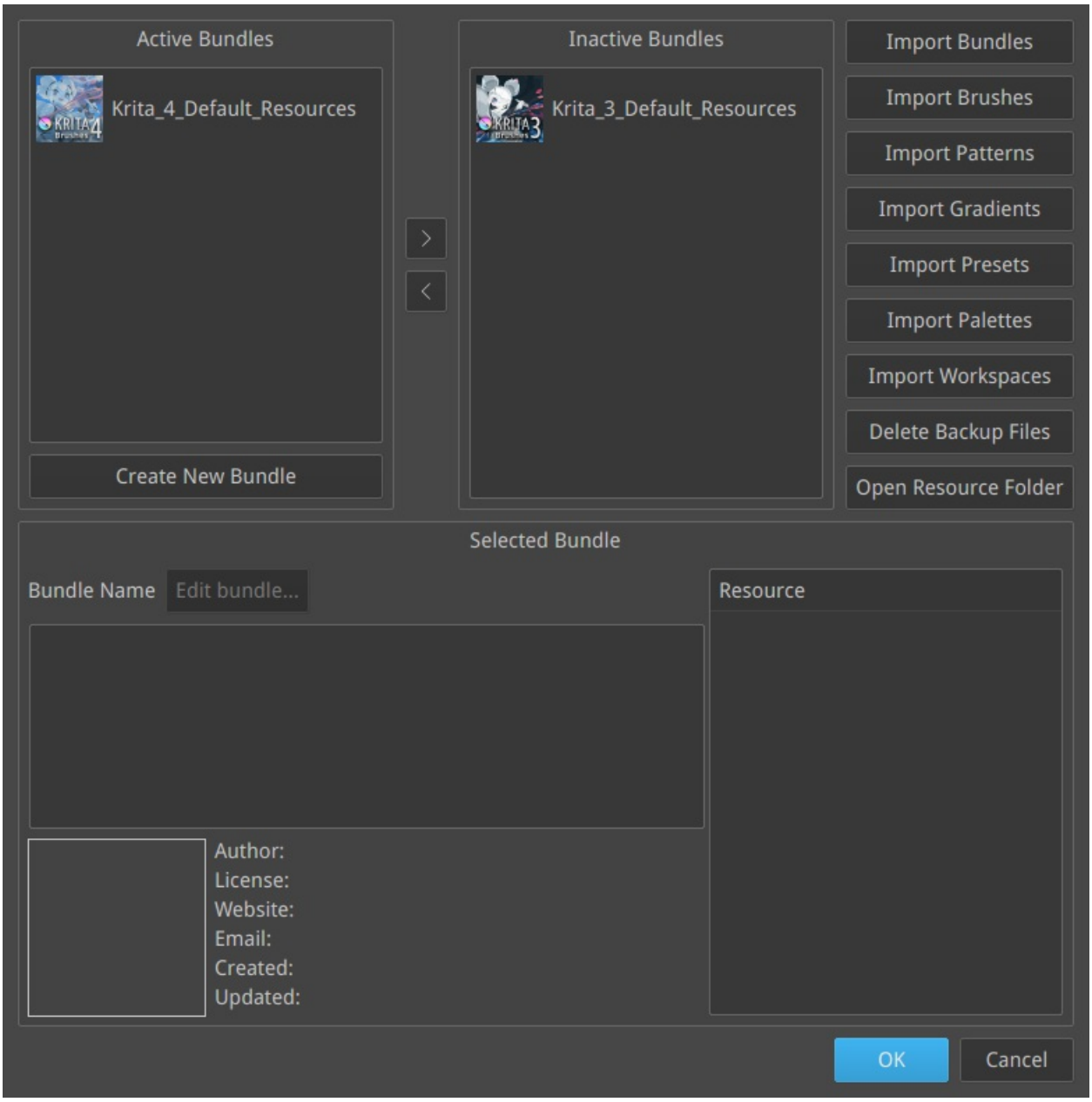
Now, anytime you want to open up your “digital pencil box” and see what you have to work with all you have to do is use the pull-down and select *Pencils*. The Resource Filter works the same way in other parts of Krita so be on the lookout for it!

Resource Zooming
: If you find the thumbnails of the resources such as color swatches brushes and pattern to be small you can make them bigger or *Zoom in*. All resource selectors can be zoomed in and out of, by hovering over the selector and using the `Ctrl +`  shortcut.

Managing Resources

As mentioned earlier Krita has a flexible resource management system. Starting from version 2.9 you can share various resources mentioned above by sharing a single compressed ZIP file created within Krita.

The manage resources section in the settings was also revamped for making it easy for the artists to prepare these bundle files. You can open manage resource section by going to *Settings* ▸ *Manage Resources…* menu item.

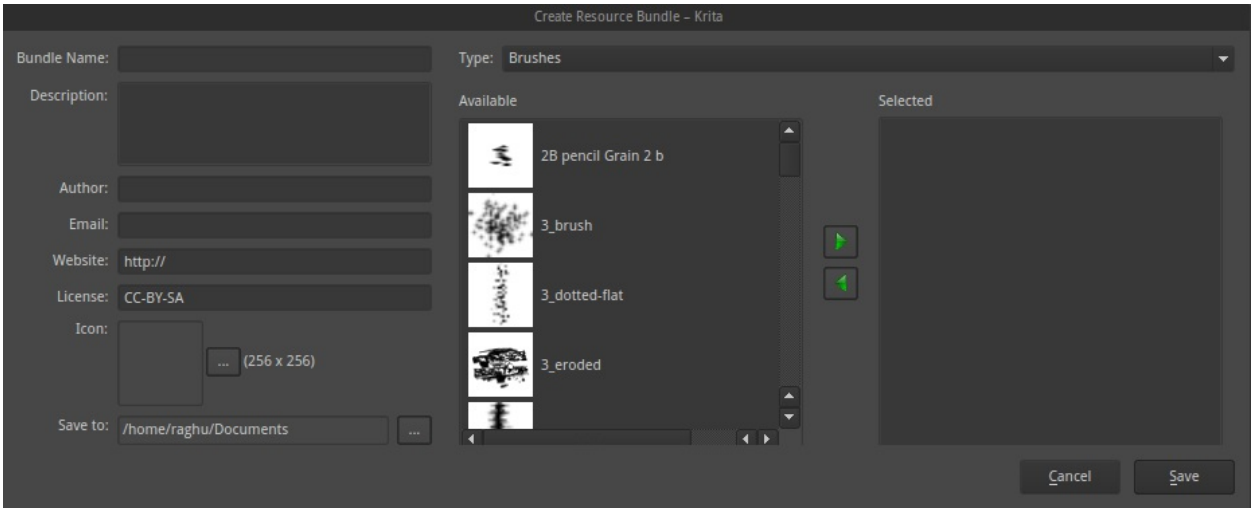


Importing Bundles

To import a bundle click on *Import Bundles/Resources* button on the top right side of the dialog. Select `.bundle` file format from the file type if it is not already selected, browse to the folder where you have downloaded the bundle, select it and click *Open*. Once the bundle is imported it will be listed in the *Active Bundle* section. If you don't need the bundle you can temporarily make it inactive by selecting it and clicking on the arrow button, this will move it to the *Inactive* section.

Creating your own Bundle

You can create your own bundle from the resources of your choice. Click on the *Create bundle* button. This will open a dialog box as shown below.



The left hand section is for filling up information about the bundle like author name, website, email, bundle icon, etc. The right hand side provides a list of available resources. Choose the type of resource you wish to add in to the bundle from the drop-down above and add it to the bundle by selecting a resource and clicking on the arrow button.

Warning

Make sure you add brush tips for used in the respective paintop presets you are adding to the bundle. If you don’t provide the brush tips then the brush presets loaded from this bundle will have a ‘X’ mark on the thumbnail denoting that the texture is missing. And the brush preset won’t be the same.

Once you have added all the resources you can create bundle by clicking on the *Save* button, the bundle will be saved in the location you have specified. You can then share this bundle with other artists or load it on other workstations.

Deleting Backup files

When you delete a preset from Krita, Krita doesn't actually delete the physical copy of the preset. It just adds it to a black list so that the next time when you start Krita the presets from this list are not loaded. To delete the brushes from this list click on *Delete Backup Files*.

Deleting Imported Bundles

In case you wish to delete the bundles you have imported permanently click on the *Open Resource Folder* button in the *Manage Resources* dialog. This will open the resource folder in your file manager / explorer. Go inside the bundles folder and delete the bundle file which you don't need any more. The next time you start Krita the bundle and its associated resources will not be loaded.

Resource Types in Krita

- Brush Preset
- Brushes
- Gradients
- Patterns
- Workspaces
- SeExpr Scripts

Brush Preset

Paint Op presets store the preview thumbnail, brush-engine, the parameters, the brush tip, and, if possible, the texture. They are saved as .kpp files.

For information regarding the brush system, see Brushes.

The Docker

The docker for Paint-op presets is the Preset Docker. Here you can tag, add, remove and search paint op presets.

Editing the preview thumbnail

You can edit the preview thumbnail in the brush-scratchpad, but you can also open the *.kpp file in Krita to get a 200x200 file to edit to your wishes.

Brushes

These are the brush tip or textures used in the brush presets. They can be PNG files or `.abr` file from Photoshop or `.gbr` files from Gimp.

Note

Currently Krita only import a brush texture from abr file, you have to recreate the brushes by adding appropriate values in size, spacing etc.

They can be modified/tagged in the brush preset editor.

See Brush Tips for more info.

Example: Loading a Photoshop Brush (*.ABR)

For some time Photoshop has been using the ABR format to compile brushes into a single file. Krita can read and load `.abr` files, although there are certain features. For this example we will use an example of an `.abr` file that contains numerous images of types of trees and ferns. We have two objectives. The first is to create a series of brushes that we an quickly access from the Brush Presets dock to easily put together a believable forest. The second is to create a single brush that we can change on the fly to use for a variety of flora, without the need to have a dedicated Brush Preset for each type.

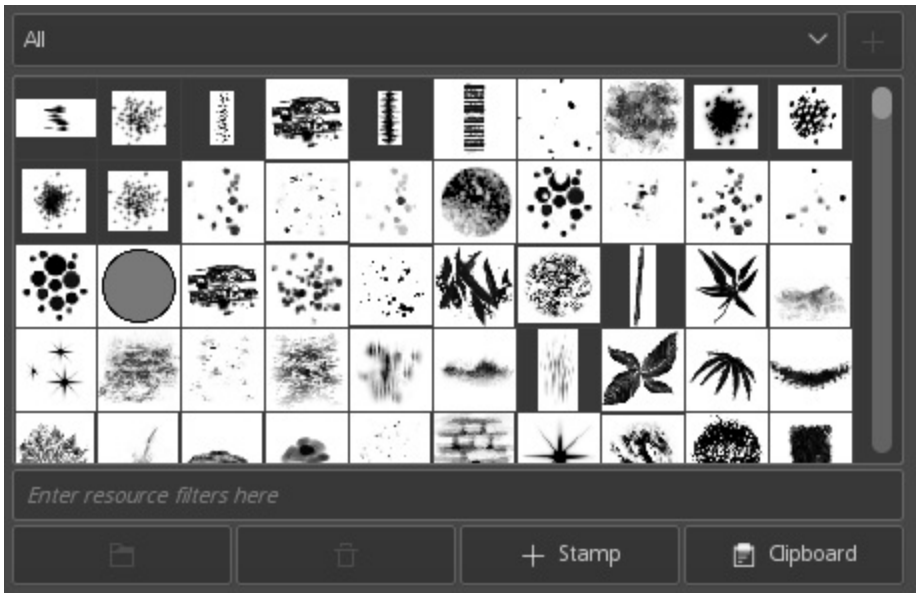
1. First up is download the file (`.zip`, `.rar`,...) that contains the `.abr` file and any licensing or other notes. Be sure to read the license if there is one!

2. Extract the `.abr` file into Krita's home directory for brushes.

3. In your Brush Presets dock, select one of your brushes that uses the Pixel Brush Engine. An Ink Pen or solid fill type should do fine.

4. Open the Brush Settings Editor (`F5` key).

5. Click on the tab “Predefined” next to “Auto”. This will change the editor to show a scrollable screen of thumbnail images, most will be black on a white background. At the bottom of the window are two icons:



6. Click on the blue file folder on the left and then navigate to where you saved your `.abr` file and open it.

7. If everything went fine you will see a number of new thumbnails show up at the bottom of the window. In our case, they would all be thumbnails representing different types of trees. Your job now is to decide which of these you want to have as Brush Preset (Just like your Pencil) or you think you’ll only use sporadically.

8. Let’s say that there is an image of an evergreen tree that we’re pretty sure is going to be a regular feature in some of our paintings and we want to have a dedicated brush for it. To do this we would do the following:

9. Click on the image of the tree we want.

10. Change the name of the brush at the very top of the Brush Editor Settings dialog. Something like “Trees - Tall Evergreen” would be

appropriate.

11. Click the “Save to Presets” button.

12. Now that you have a “Tall Evergreen” brush safely saved you can experiment with the settings to see if there is anything you would like to change, for instance, by altering the size setting and the pressure parameter you could set the brush to change the tree size depending on the pressure you were using with your stylus (assuming you have a stylus!).

13. Once you’re satisfied with your brush and its settings you need to do one last thing (but click *Overwrite Brush* first!).

It’s time now to create the Brush Preview graphic. The simplest and easiest way to do this for a brush of this type is to clear out the ScratchPad using the *Reset* button. Now, center your cursor in the Brush Preview square at the top of the ScratchPad and click once. You should see an image of your texture (in this case it would be the evergreen tree). In order to work correctly though the entire image should fit comfortably within the square. This might mean that you have to tweak the size of the brush. Once you have something you are happy with then click the *Overwrite Brush* button and your brush and its preview image will be saved.

An alternative method that requires a little more work but gives you greater control of the outcome is the following:

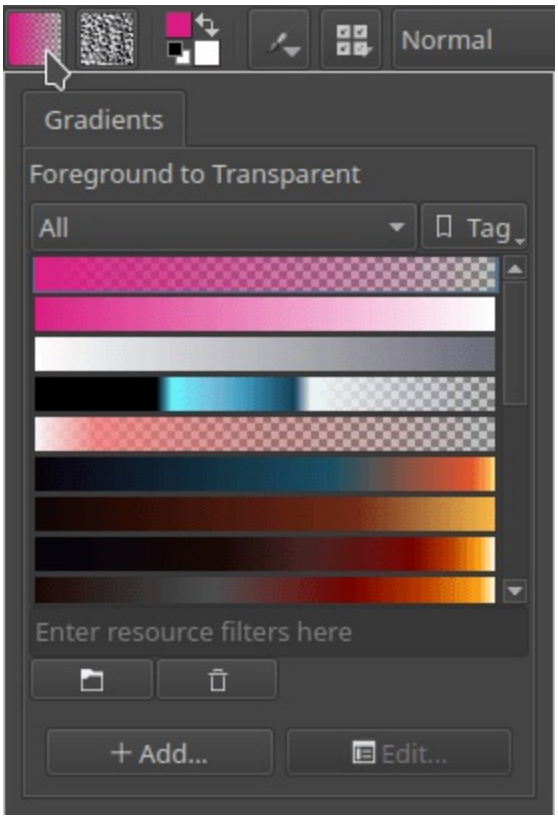
Locate the Brush Preview thumbnail `.kpp` file in Krita and open it to get a 200x200 file that you can edit to your wishes.

You’re ready to add the next texture! From here on it’s just a matter of wash, rinse and repeat for each texture where you want to create a dedicated Brush Preset.

Gradients

Accessing a Gradient

The Gradients configuration panel is accessed by clicking the Gradients icon (usually the icon next to the disk).



Gradients are configurations of blending between colors. Krita provides over a dozen preset dynamic gradients for you to choose from. In addition, you can design and save your own.

Some typical uses of gradients are:

- Fill for vector shapes.
- Gradient tool
- As a source of color for the pixel brush.

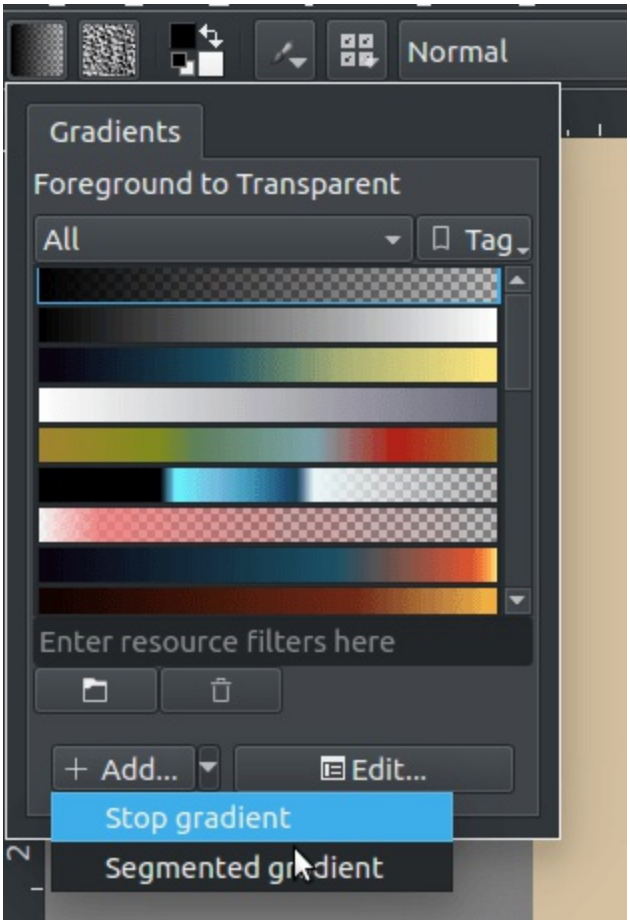
There is no gradients docker. They can only be accessed through the gradient “quick-menu” in the toolbar.

Editing a Gradient

Krita has two gradient types:

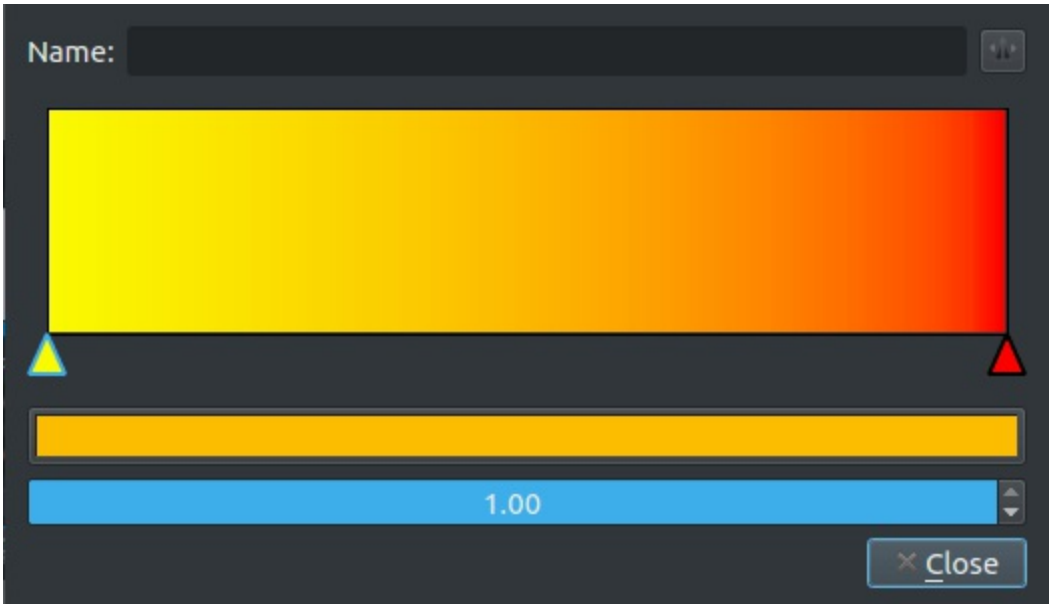
1. Segmented Gradients, which are compatible with GIMP, have many different features but are also a bit complicated to make.
2. Stop Gradients, which are saved as SVG files and similar to how most applications do their gradients, but has less features than the segmented gradient.

Initially we could only make segmented gradients in Krita, but in 3.0.2 we can also make stop gradients.



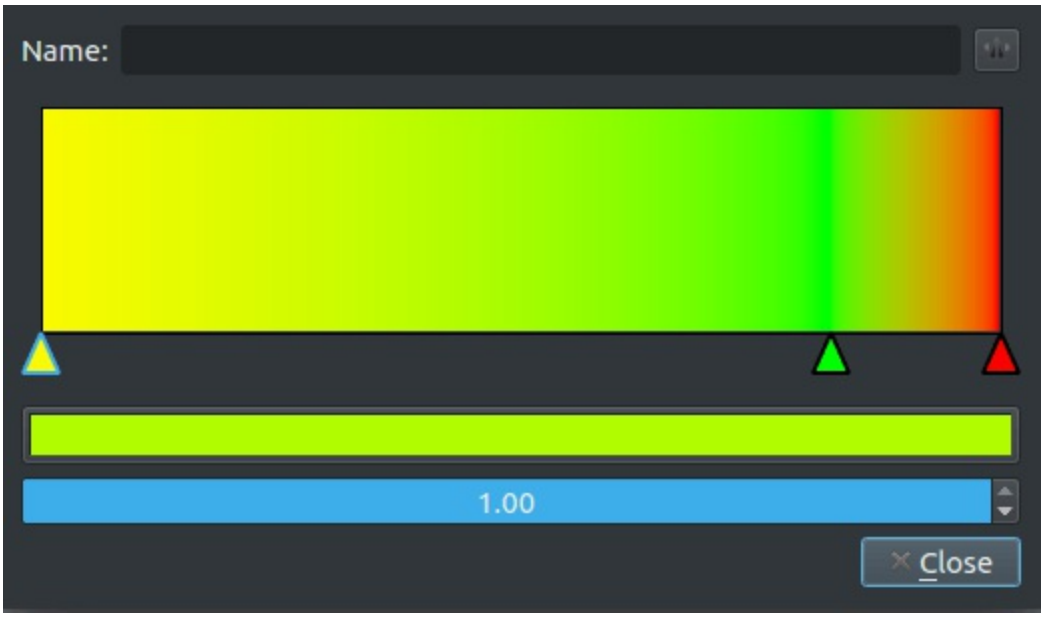
You can make a new gradient by going into the drop-down and selecting the gradient type you wish to have. By default Krita will make a stop-gradient.

Stop Gradients

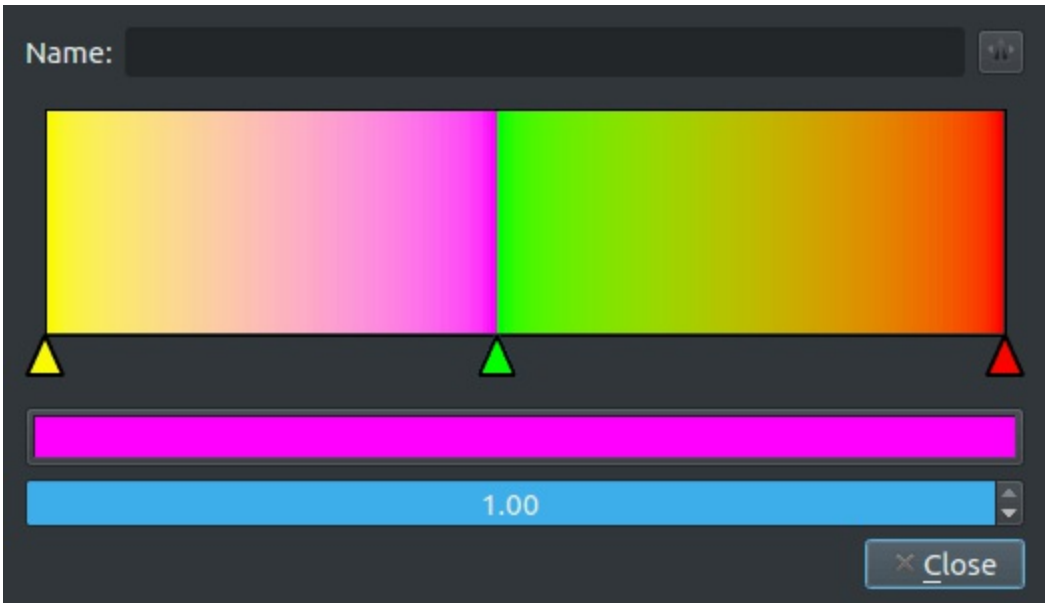


Stop gradients are very straight forward:

-  on the gradient to add a stop.
-  on the stops to select them, and drag to move them.
- Drag stops outside of the bar (further than the way to the left or right) to remove them.

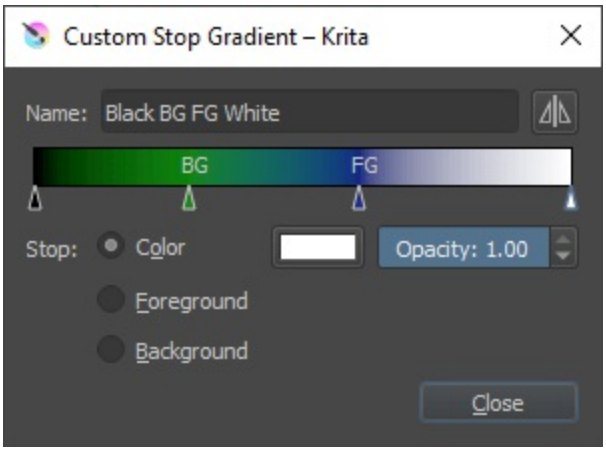


A selected stop can have its color and transparency changed using the color button and the opacity slider below.

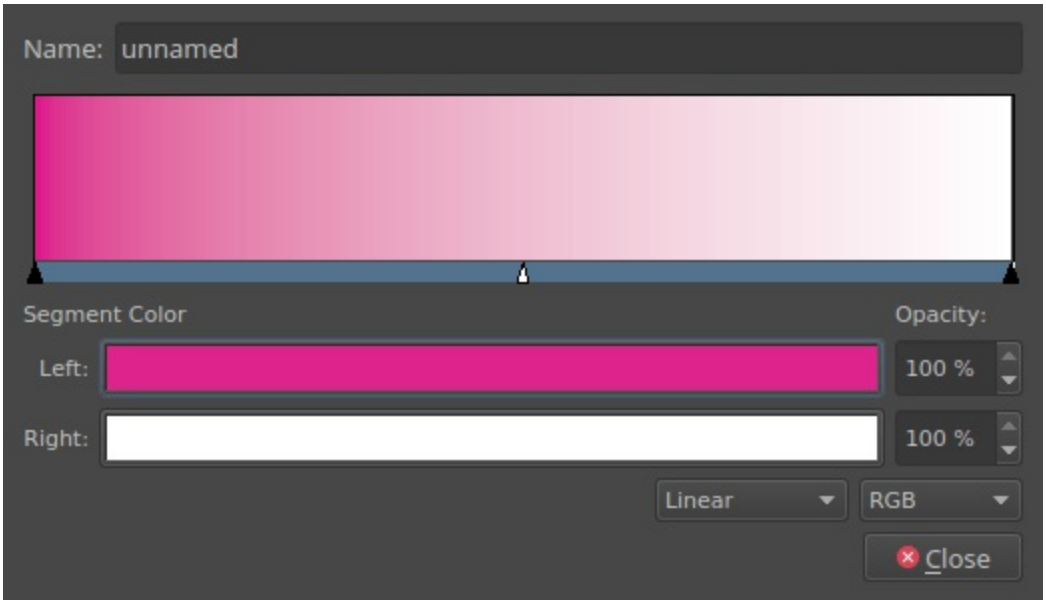


As per SVG spec, you can make a sudden change between stops by moving them close together. The stops will overlap, but you can still drag them around.

New in version 4.4: Gradients can have stops that use the currently selected Foreground or Background colors

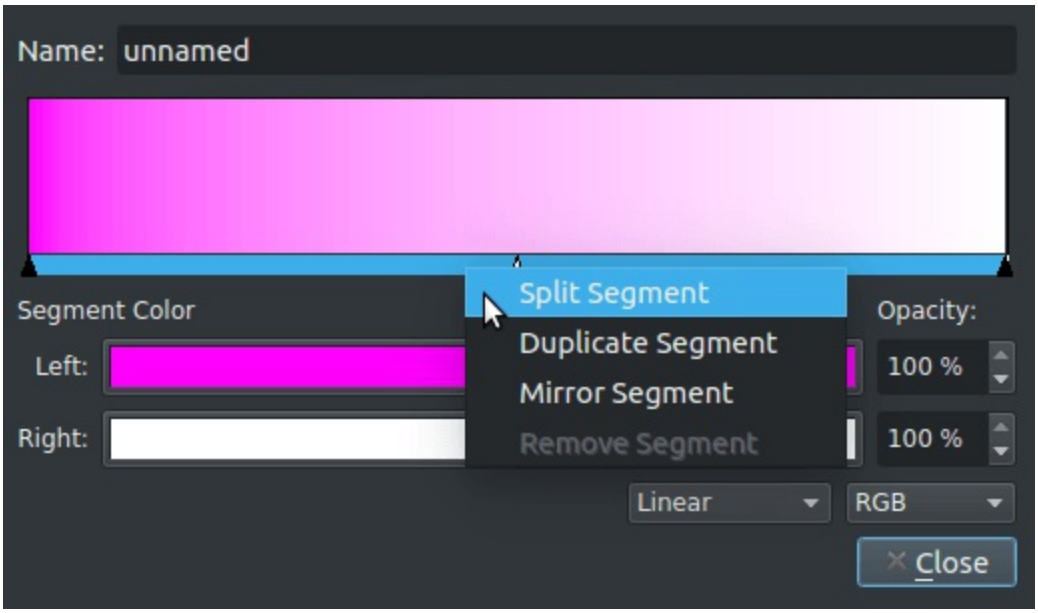


Segmented Gradients



Segmented gradients are a bit more tricky. Instead of going from color to color, it allows you to define segments, which each can have a begin and end color.

 the gradient to call up this menu:



Split Segment
: This splits the current segment in two, using the white arrow, the segment middle as the place to split. It will also use the color at the white arrow to define the new colors in place in the new segments.

Duplicate segment
: Similar to split, but instead the two new segments are copies of the old one.

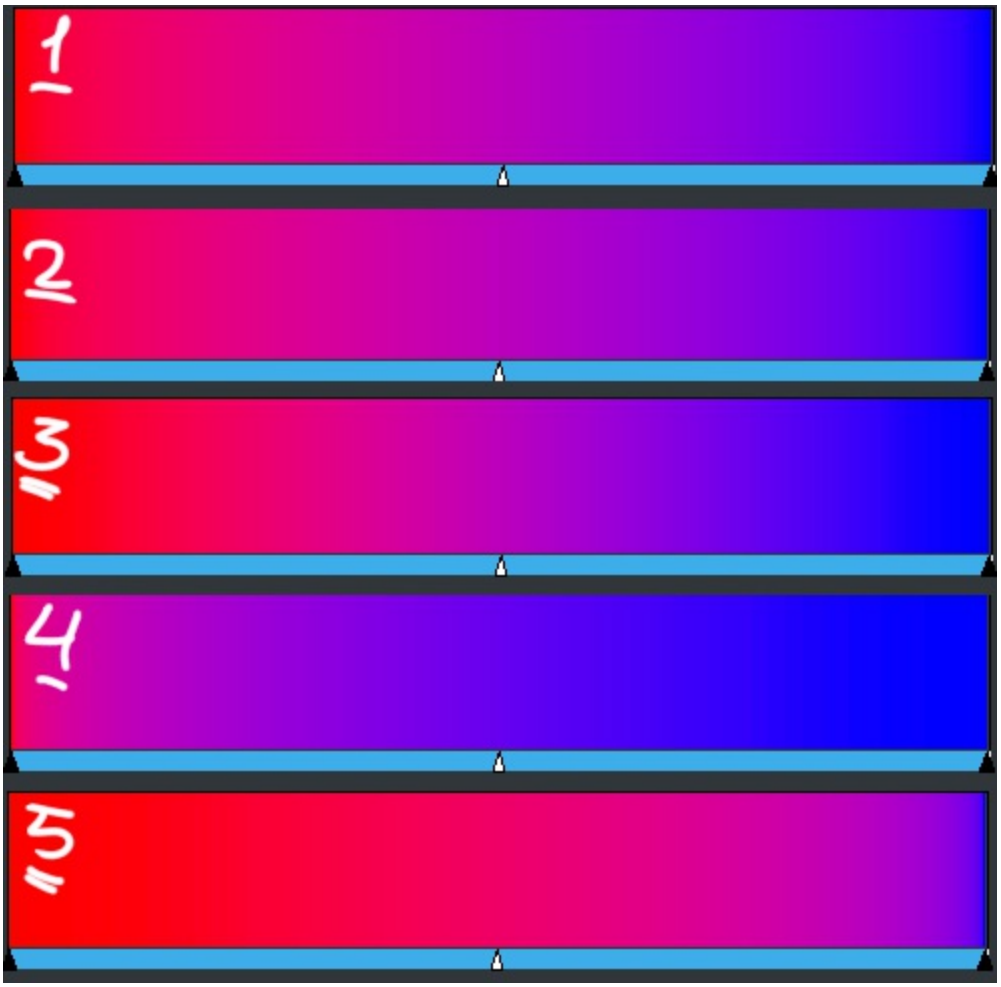
Mirror segment
: Mirrors the segment colors.

Remove segment
: Removes the segment.

 + dragging the black arrows will resize the segments attaching to those arrows.  + dragging the white arrows will change the mid point of that segment, changing the way how the mixture is made.

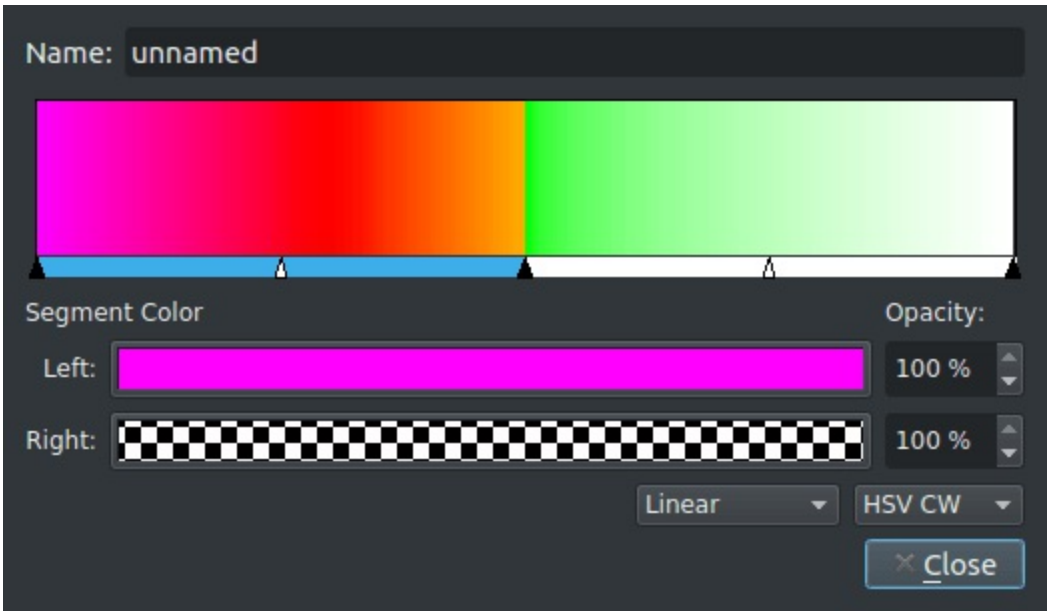
At the bottom, you can set the color and transparency of either part of the segment.

You can also set the blending. The first is the interpolation mode:



1. Linear - Does a linear blending between both segments.
2. Curved - This causes the mix to ease-in and out faster.
3. Sine - Uses a sine function. This causes the mix to ease in and out slower.
4. Sphere, increasing - This puts emphasis on the later color during the mix.
5. Sphere, decreasing - This puts emphasis on the first color during the mix.

Finally, there’s the model:

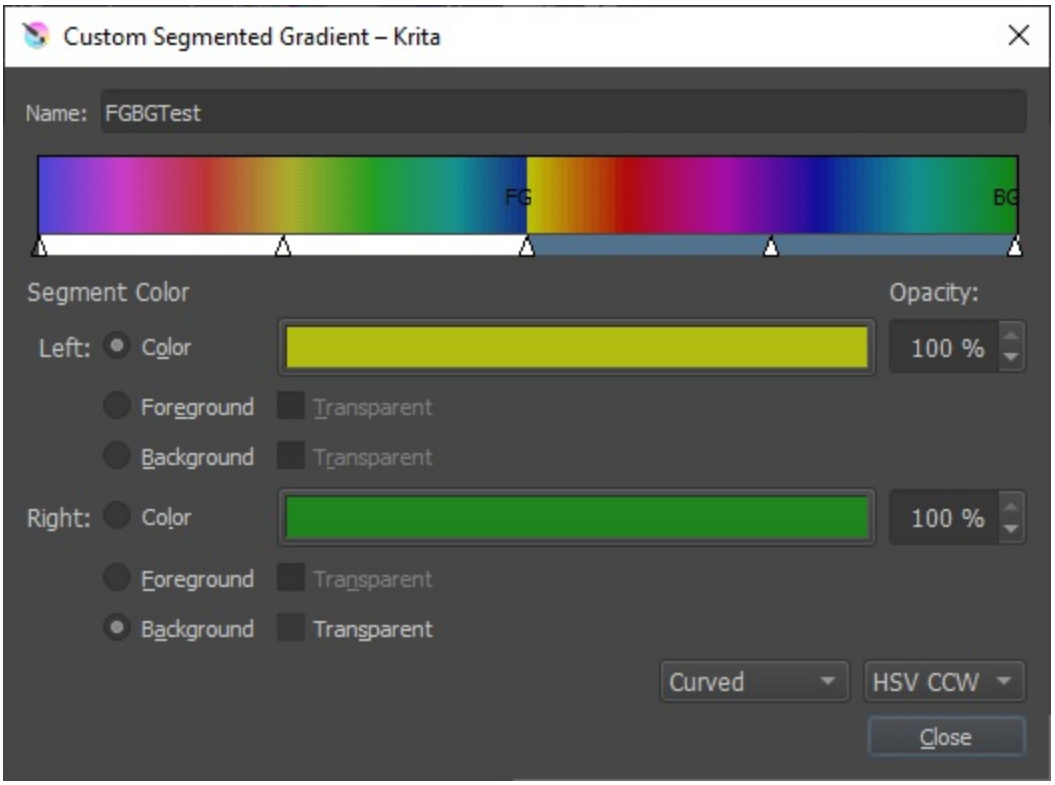


RGB
: Does the blending in RGB model.

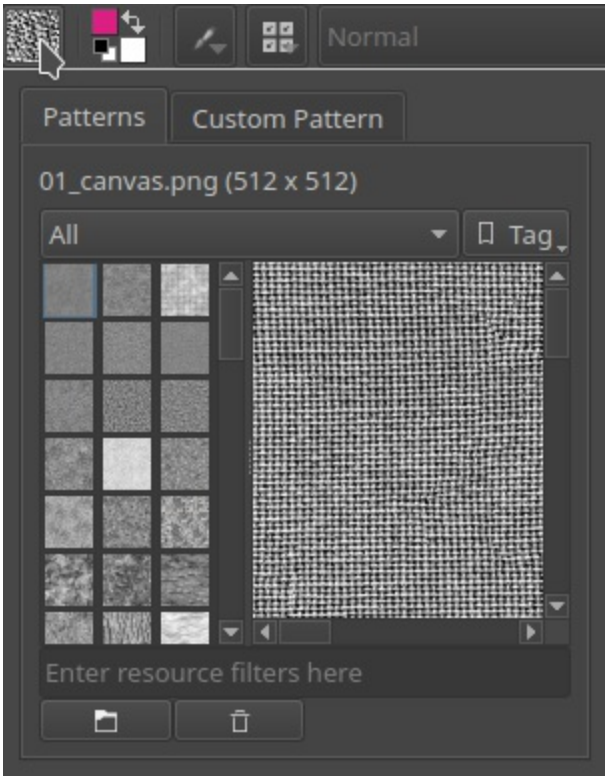
HSV clockwise
: Blends the two colors using the HSV model, and follows the hue clockwise (red-yellow-green-cyan-blue-purple). The above screenshot is an example of this.

HSV counter-clock wise.
: Blends the color as the previous options, but then counter-clockwise.

New in version 4.4: Gradients can have segment endpoints that use the currently selected Foreground or Background colors, and those endpoints can be transparent. These features allow full compatibility with GIMP gradients.


Custom Segmented Gradient – Krita
Name: FGBGTest
FG
BG
Segment Color
Opacity:
Left: Color
100 %
Foreground
Transparent
Background
Transparent
Right: Color
100 %
Foreground
Transparent
Background
Transparent
Curved
HSV CCW
Close

Patterns



Patterns are small raster files that tile. They can be used as following:

- As fill for a vector shape.
- As fill-tool color.
- As height-map for a brush using the ‘texture’ functionality.
- As fill for a generated layer.

Adding new patterns

You can add new patterns via the pattern docker, or the pattern-quick-access menu in the toolbar. At the bottom of the docker, beneath the resource-filter input field, there are the *Import resource* and *Delete resource* buttons. Select the former to add png or JPG files to the pattern list.

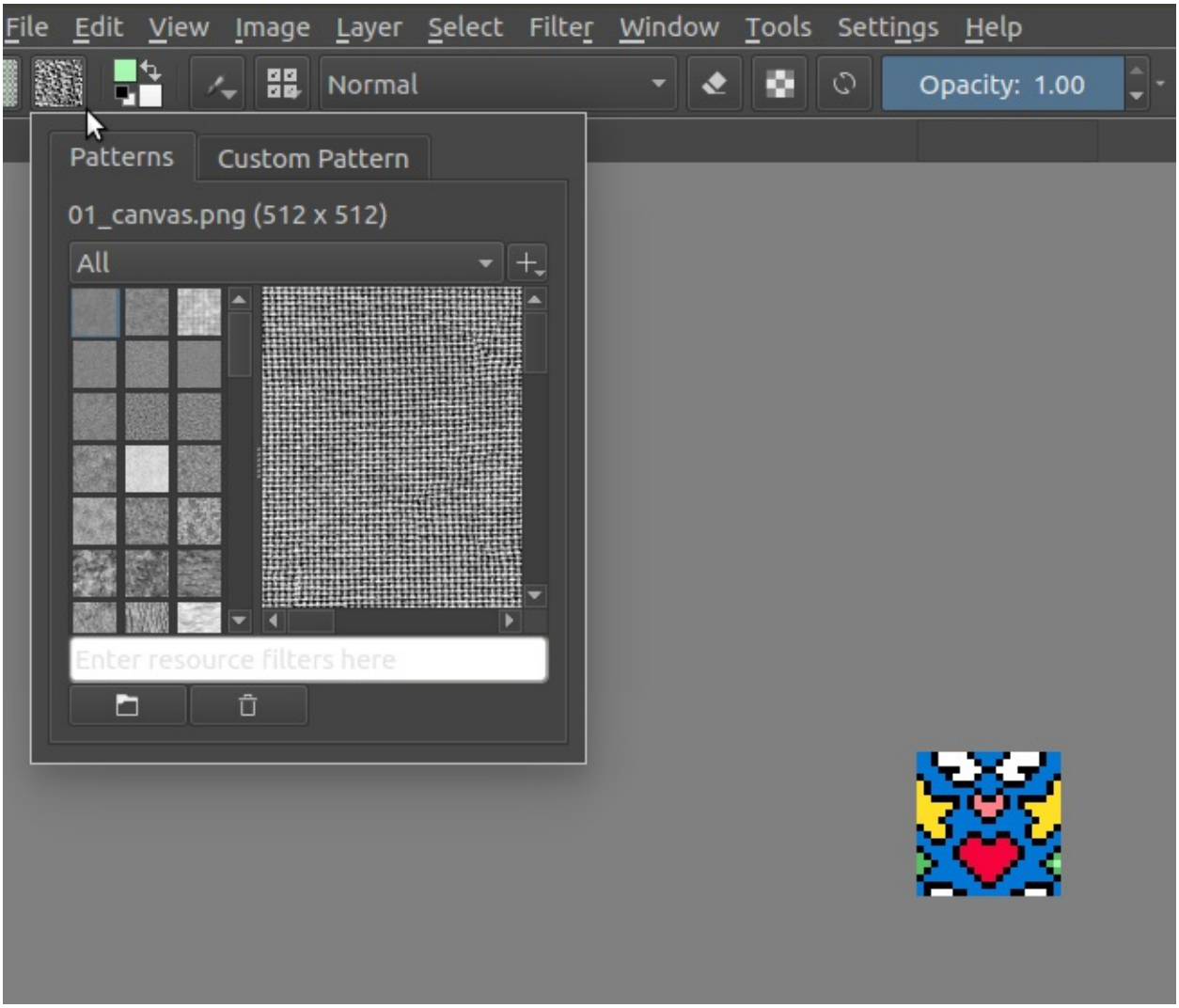
Similarly, removing patterns can be done by pressing the :guilabel::*Delete*

resource button. Krita will not delete the actual file then, but rather black list it, and thus not load it.

Temporary patterns and generating patterns from the canvas

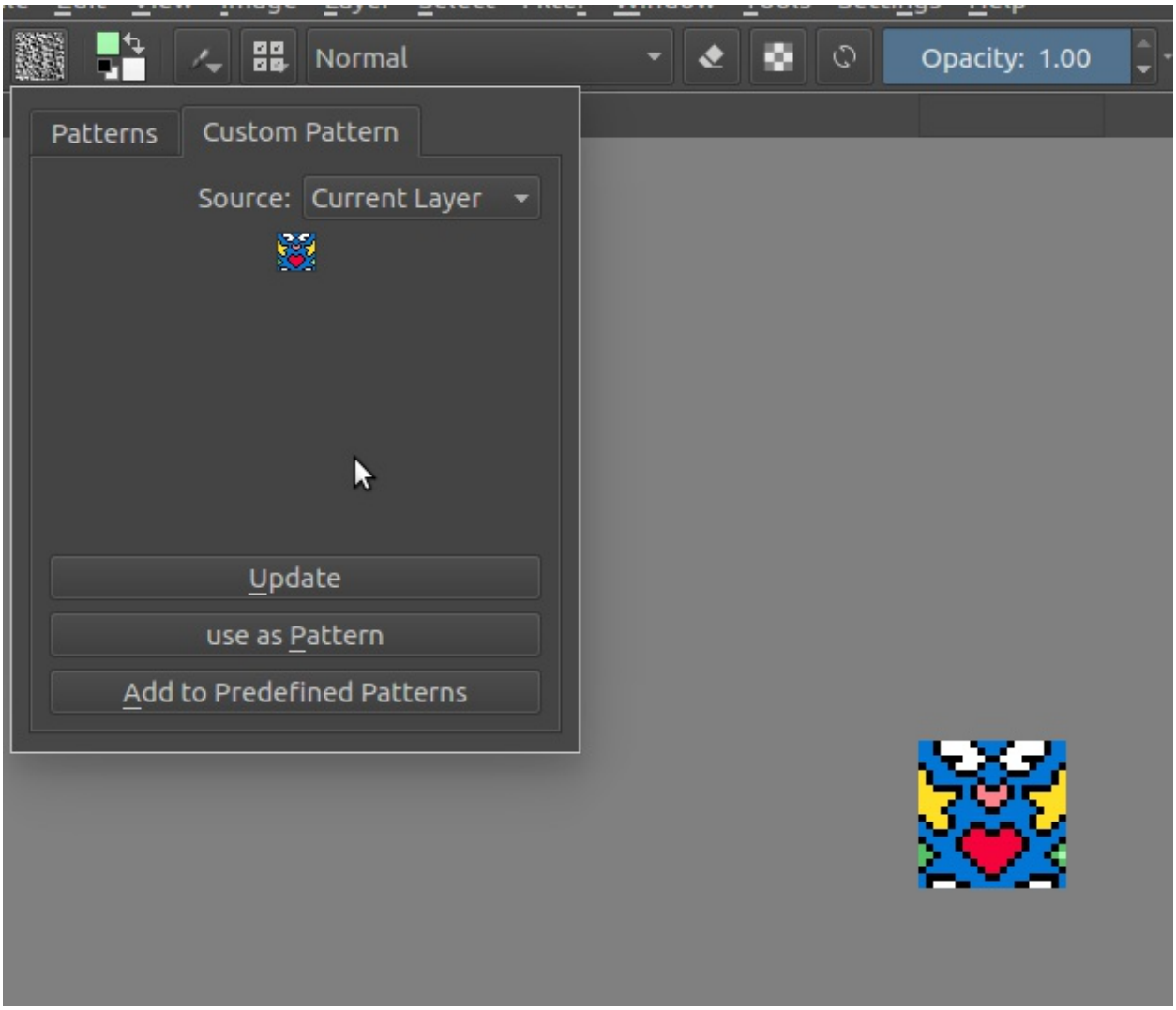
You can use the pattern drop-down to generate patterns from the canvas but also to make temporary ones.

First, draw a pattern and open the pattern drop-down.



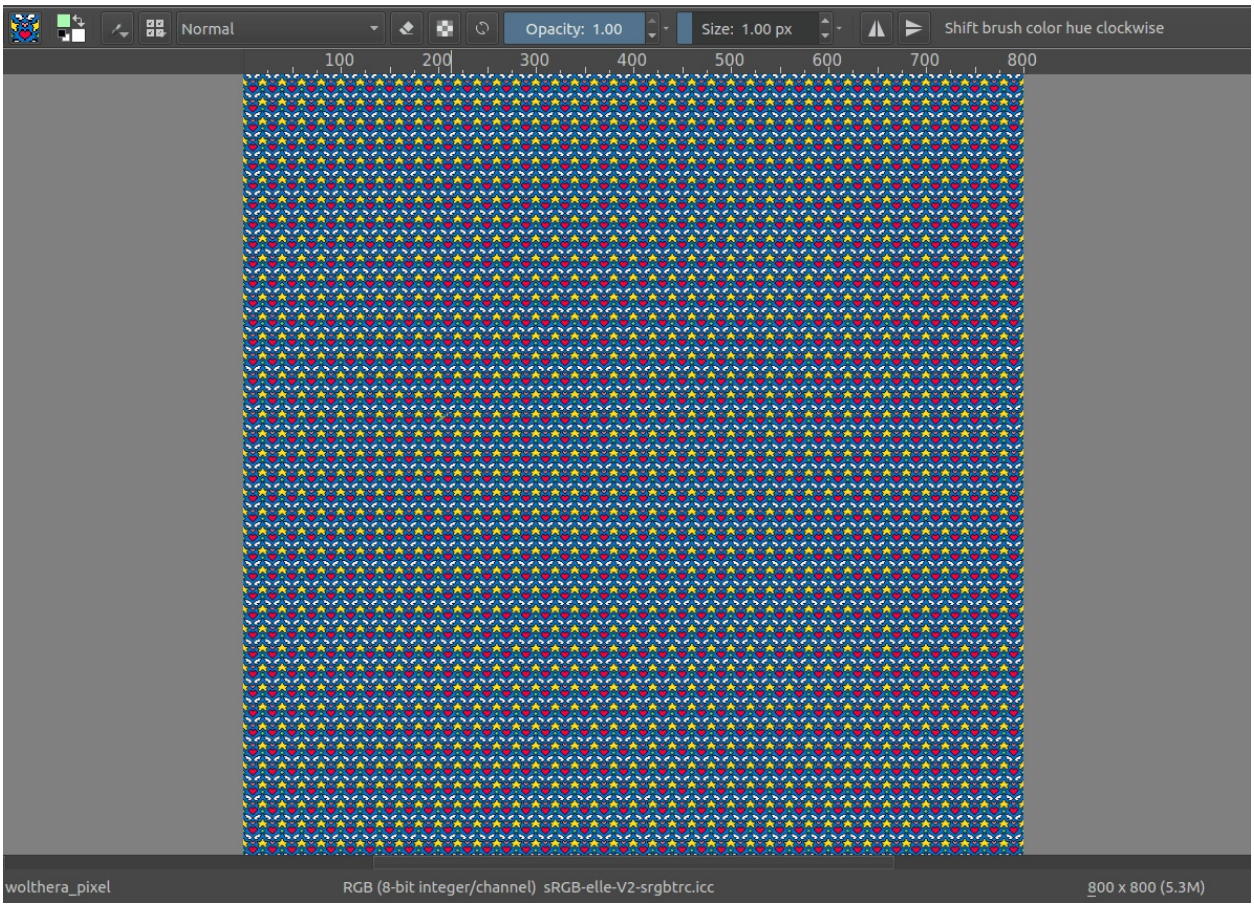
Then go into *custom* and first press *Update* to show the pattern in the docker.

Check if it’s right. Here, you can also choose whether you use this layer only, or the whole image. Since 3.0.2, Krita will take into account the active selection as well when getting the information of the two.



Then, click either *Use as Pattern* to use it as a temporary pattern, or *Add to predefined patterns* to save it into your pattern resources!

You can then start using it in Krita by for example making a canvas and doing *Edit –> Fill with Pattern*.



Patterns Docker

You can tag patterns here, and filter them with the resource filter.

Workspaces

Workspaces are basically saved configurations of dockers. Each workspace saves how the dockers are grouped and where they are placed on the screen. They allow you to easily move between workflows without having to manual reconfigure your setup each time. They can be as simple or as complex as you want.

Workspaces can only be accessed via the toolbar or *Window* ▸ *Workspaces* There's no docker for them. You can save workspaces, in which your current configuration is saved. You can also import them (from a *.kws file), or delete them (which black lists them).

Workspaces can technically be tagged, but outside of the resource manager this is not possible.

Window Layouts

When you work with multiple screens, a single window with a single workspace won't be enough. For multi monitor setups we instead can use sessions. Window layouts allow us to store multiple windows, their positions and the monitor they were on.

You can access Window Layouts from the workspace drop-down in the toolbar.

Primary Workspace Follows Focus
: This treats the workspace in the first window as the 'primary' workspace, and when you switch focus, it will switch the secondary windows to that primary workspace. This is useful when the secondary workspace is a very sparse workspace with few dockers, and the primary is one with a lot of different dockers.

Show Active Image In All Windows
: This will synchronise the currently viewed image in all windows. Without

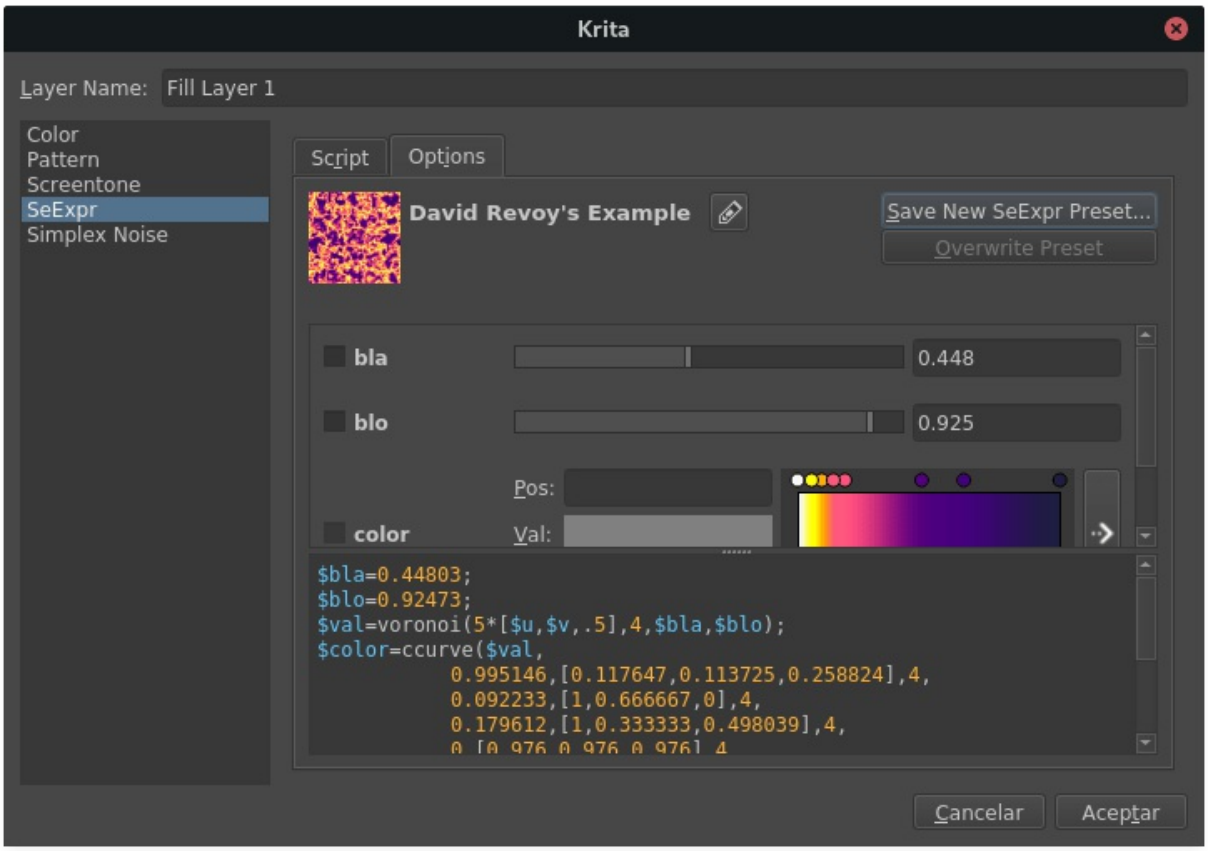
it, different windows can open separate views for an image via *Window* ▸ *New View* ▸ *document.kra*.

Sessions

Sessions allow Krita to store the images and windows opened. You can tell Krita to automatically save current or recover previous sessions if so configured in the Miscellaneous.

You can access sessions from *File* ▸ *Sessions*.

SeExpr Scripts



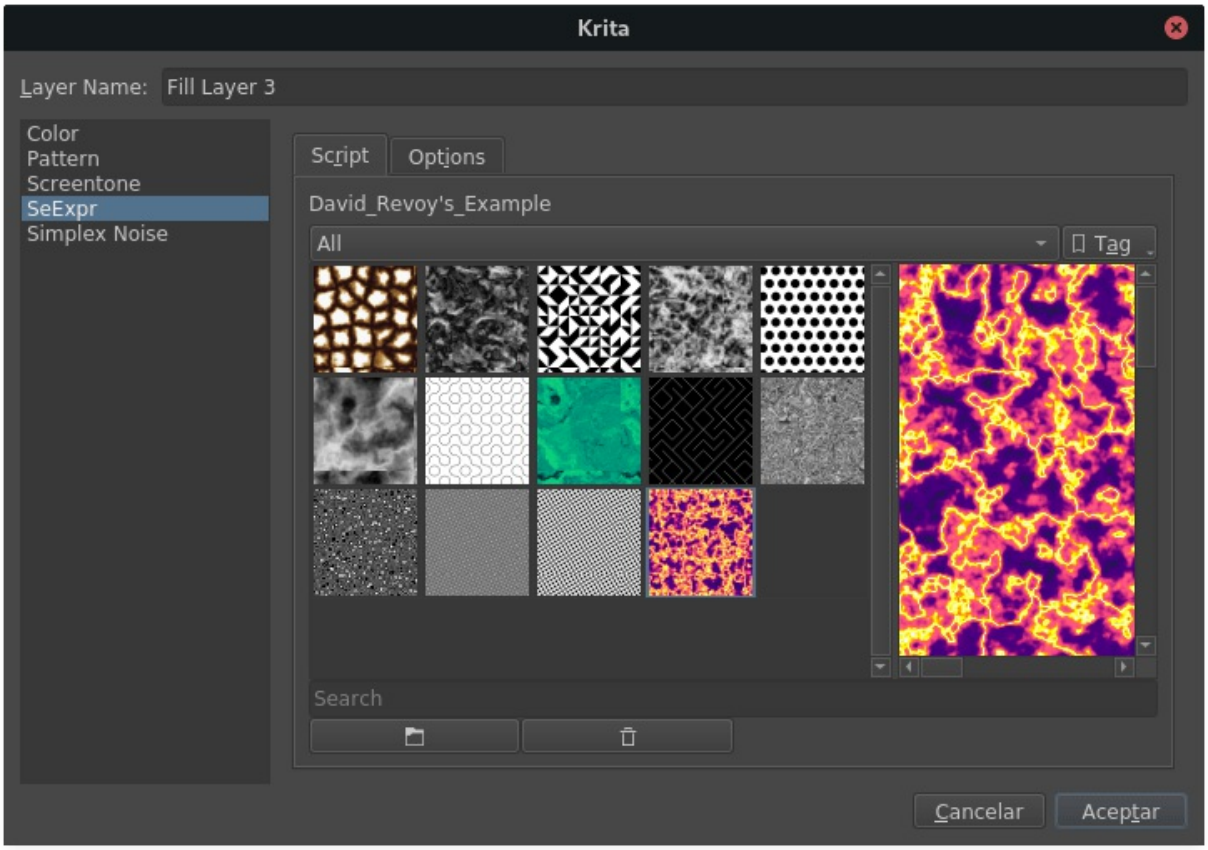
SeExpr scripts allow you to render dynamically generated textures. They are saved as .se files. They can be used as fill for generated layers.

See also

- Introduction to SeExpr
- SeExpr Quick Reference
- SeExpr
- “Procedural texture generator (example and wishes)” on Krita Artists [https://krita-artists.org/t/procedural-texture-generator-example-and-wishes/7638]
- Inigo Quilez’s articles [https://iquilezles.org/www/index.htm]
- The Book of Shaders [https://thebookofshaders.com/]

Adding new scripts

Like with Patterns, there is a similar UI with which you can manage SeExpr presets. This panel is available by opening the Fill Layer dialog, and selecting *SeExpr*:



At the bottom of the tab, beneath the resource-filter input field, there are the *Import resource* and *Delete resource* buttons. Select the former to add presets to the list.

Similarly, removing presets can be done by pressing the *Delete resource* button. Krita will not delete the actual file then, but rather black list it, and thus not load it.

Temporary scripts

Layers keep a copy of the textual script that was used to generate them. You can access it by going to the Options tab and copying the text. You can then paste it into a new SeExpr Fill Layer.

SeExpr Quick Reference

This page details all the available variables, functions, and operators in SeExpr. It is a heavily edited version of the official user documentation [https://wdas.github.io/SeExpr/doxygen/userdoc.html], adapted for usage with Krita.

See also

- Source code at KDE Invent [https://invent.kde.org/lsegovia/seexpr]
- Disney's SeExpr API Documentation [http://wdas.github.io/SeExpr/doxygen/]

See also

- Introduction to SeExpr
- SeExpr
- SeExpr Scripts
- "Procedural texture generator (example and wishes)" on Krita Artists [https://krita-artists.org/t/procedural-texture-generator-example-and-wishes/7638]
- Inigo Quilez's articles [https://iquilezles.org/www/index.htm]
- The Book of Shaders [https://thebookofshaders.com/]

Contents

Variables

External variables

These variables are provided by host applications, in this case Krita. They are registered with SeExpr's autocomplete help, which can be accessed by `Ctrl+Space`.

$u, $v

Pixel position in normalized coordinates.

$w, $h

Image's width and height in pixels.

Local Variables

Local variables can be defined at the start of the expression:

```
$a = noise($P);
$b = noise($a * 1);
pow($a, 0.5) + $b
```

External variables can also be overridden by local assignment. This can be useful to scale the noise frequency for instance:

```
$P = $P * 10; # increase noise frequency
fbm(vnoise($P) + $P/4)
```

You can also define namespaced variables, e.g.:

```
$A::a = $u * 10;
```

Control Structures

SeExpr provides the well-known *if* conditional structure:

```
if ($ u > 0.5) {
    $color = [0, 0, 1];
}
else {
    $color = [1, 0, 0];
}

$color
```

And the *ternary operator*:

```
$u = $i < .5 ? 0.0 : 10.0
```

You can freely nest ternary operators, e.g.:

```
$color = $u < .5 ? ($v < 0.5 ? [0, 0, 1] : [1, 0, 0]) : [0, 1, 0]
$color
```

You can also achieve the same with *if* structures:

```
if ($ u > 0.5) {
    if ($v < 0.5) {
        $color = [0, 0, 1];
    }
    else {
        $color = [1, 0, 0];
    }
}
else {
    $color = [1, 0, 0];
}

$color
```

Operators (listed in decreasing precedence)

[a,b,c]

vector constructor

$P[n]

vector component access

Hint

`n` must be 0, 1, or 2, e.g.:

```
$P[0]
```

^

exponentiation

Note

Same as the `pow` function.

!

logical NOT

~

inversion

Hint

```
~$A
```

gives the same result as:

```
1 - $A
```

*/ %

multiply, divide, modulus

Note

`%` is the same as the `fmod` function.

+-

add, subtract

<> <= >=

comparison: less than, greater than, less or equal than, greater or equal than

Note

Only uses the first component of a vector.

== !=

equality, inequality

&&

logical AND

||

logical OR

?:

ternary `if` operator

Hint

Example:

```
$u < .5 ? 0 : 1
```

->

apply - The function on the right of the arrow is applied to the expression on the left.

Hint

Examples:

```
$Cs->contrast(.7) -> clamp(0.2,0.8)
$u->hsi(20,1.2,1,$Cs->gamma(1.2))
```

Assignment Operators

Besides the basic assignment statement:

```
$foo = $bar
```

you can also do operator assignments such as:

```
$foo += $bar;
```

which is equivalent to:

```
$foo = $foo + $bar;
```

Additionally, there are:

- `+=`
- `-=`
- `/=`
- `%=`
- `*=`
- `^=`

Comments

You can add comments to your script by using the # character. SeExpr will then skip the rest of the line for rendering purposes. However, they are not ignored; comments can still be used to declare the valid value range of

integer, float, and vector variables. This enables you to manage them using widgets that will accept the specified range.

Hint

`$var0` is an integer variable that ranges between 0 and 10 inclusive:

```
$var0 = 0; # 0, 10
```

`$var1` is a floating point variable with the same range:

```
$var1 = 0; # 0.000, 10.000
```

`$var2` is a vector variable:

```
$var2 = [0, 0, 0] # 0.000, 10.000
```

The latter is very helpful; SeExpr considers vectors with range `[0, 1]` as colors:

```
# this is a dark red
$color = [0.5, 0, 0] # 0.000, 1.000
```

In all cases, if not specified, the associated widgets' range will go from 0 to 1.

For a multi-line expression, each line may have its own comment.

Logging Functions

float printf (string format, [param0, param1, …])

Prints a string to stdout that is formatted as given. Formatting parameters possible are `%f` for float (takes the first component of vector argument) or `%v` for vector.

Hint

For example, if you wrote:

```
$u = printf("test %f %v",[1,2,3],[4,5,6]);
```

you would get in your console:

```
test 1 [4,5,6]
```

string sprintf (string format, [double|string, double|string, …])

Returns a string formatted from the given values. See `man sprintf` for format details.

Color, Masking, and Remapping Functions

float bias (float x, float b)

Variation of gamma where control parameter goes from `0` to `1` with values `> 0.5` pulling the curve up and values `< 0.5` pulling the curve down. Defined as `pow(x, log(b)/log(0.5))`.

float boxstep (float x, float a)
float gaussstep (float x, float a, float b)
float linearstep (float x, float a, float b)
float smoothstep (float x, float a, float b)

The step functions are zero for `x < a` and one for `x > b` (or `x > a` in the case of boxstep). Between `a` and `b`, the value changes continuously between zero and one. The `gausstep` function uses the standard Gaussian "bell" curve which is based on an exponential curve. The `smoothstep` function uses a cubic curve. Intuitively, `gausstep` has a sharper transition near one and a softer transition near zero whereas `smoothstep` has a medium softness near both one and zero.

float clamp (float x, float lo, float hi)

Constrain `x` to range `[lo, hi]`.

float compress (float x, float lo, float hi)

Compress the dynamic range from `[0, 1]` to `[lo, hi]`.

float contrast (float x, float c)

Adjust the contrast. For `c` from `0` to `0.5`, the contrast is decreased. For `c > 0.5`, the contrast is increased.

float expand (float x, float lo, float hi)

Expand the dynamic range from `[lo, hi]` to `[0, 1]`.

float fit (float x, float a1, float b1, float a2, float b2)

Linear remapping of `[a1..x..b1]` to `[a2..x..b2]`

float gamma (float x, float g)

`pow(x, 1/g)`

float invert (float x)

Invert the value. Defined as `1 - x`.

color hsi (color x, float h, float s, float i, float map=1)

The `hsi` function shifts the hue by `h` (in degrees) and scales the saturation and intensity by `s` and `i` respectively. A map may be supplied which will control the shift - the full shift will happen when the map is one and no shift will happen when the map is zero. The shift will be scaled back for values between zero and one.

color hsltorgb (color hsl)
color rgbtohsl (color rgb)

RGB to HSL color space conversion. HSL is Hue, Saturation, Lightness (all in the range `[0, 1]`). These functions have also been extended to support RGB and HSL values outside of the range `[0, 1]` in a reasonable way. For any RGB or HSL value (except for negative values), the conversion is well-defined and reversible.

color midhsi (color x, float h, float s, float i, float map, float falloff=1, int interp=0)

The `midhsi` function is just like the `hsi` function except that the control map is centered around the mid point (value of `0.5`) and can scale the

shift in both directions. At the mid point, no shift happens. At *1.0*, the full shift happens, and at `0.0`, the full inverse shift happens. Additional `falloff` and `interp` controls are provided to adjust the map using the `remap` function. The default `falloff` and `interp` values result in no remapping.

float mix (float a, float b, float alpha)

Blend of a and b according to alpha. Defined as `a*(1-alpha) +b*alpha`.

float remap (float x, float source, float range, float falloff, int interp)

General remapping function. When `x` is within `± range` of source, the result is one. The result falls to zero beyond that range over `falloff` distance. The falloff shape is controlled by `interp`.

Note

Numeric values or named constants may be used:

- int **linear** = 0
- int **smooth** = 1
- int **gaussian** = 2

Noise Functions

float cellnoise (vector v)
float cellnoise1 (float x)
float cellnoise2 (float x, float y)
float cellnoise3 (float x, float y, float z)
color ccellnoise (vector v)

`cellnoise` generates a field of constant colored cubes based on the integer location. This is the same as the PRMan cellnoise function [https://renderman.pixar.com/resources/RenderMan_20/cellnoise.html].

Note

`ccellnoise` outputs color cellnoise.

float fbm (vector v, int octaves=6, float lacunarity=2, float gain=0.5)
color cfbm (vector v, int octaves=6, float lacunarity=2, float gain=0.5)
vector vfbm (vector v, int octaves=6, float lacunarity=2, float gain=0.5)
float fbm4 (vector v, float time, int octaves=6, float lacunarity=2, float gain=0.5)
color cfbm4 (vector v, float time, int octaves=6, float lacunarity=2, float gain=0.5)
vector vfbm4 (vector v, float time, int octaves=6, float lacunarity=2, float gain=0.5)

`fbm` (Fractal Brownian Motion) is a multi-frequency noise function. The base frequency is the same as the `noise` function. The total number of frequencies is controlled by `octaves`. The `lacunarity` is the spacing between the frequencies - a value of 2 means each octave is twice the previous frequency. The `gain` controls how much each frequency is scaled relative to the previous frequency.

Note

`cfbm` and `cfbm4` outputs color noise.

`vfbm` and `vfbm4` outputs vector noise.

float hash (float seed1, [float seed2, …])

Like `rand`, but with no internal seeds. Any number of seeds may be given and the result will be a random function based on all the seeds.

float noise (vector v)
float noise (float x, float y)
float noise (float x, float y, float z)

float noise (float x, float y, float z, float w)
color cnoise (vector v)
color cnoise4 (vector v, float t)
float pnoise (vector v, vector period)
float snoise (vector v)
float snoise4 (vector v, float t)
vector vnoise (vector v)
vector vnoise4 (vector v, float t)

`noise` is a random function that smoothly blends between samples at integer locations. This is Ken Perlin's original noise function.

Note

`cnoise` and `cnoise4` output color noise.

`noise4` outputs signed vector noise.

`pnoise` outputs periodic noise.

`snoise` and `snoise4` output signed noise with range `[-1, 1]`.

`vnoise` outputs signed vector noise.

float rand ([float min, float max], [float seed])

Random number between `[min, max]` (or `[0, 1]` if unspecified). If a seed is supplied, it will be used in addition to the internal seeds and may be used to create multiple distinct generators.

float turbulence (vector v, int octaves=6, float lacunarity=2, float gain=0.5)
color cturbulence (vector v, int octaves=6, float lacunarity=2, float gain=0.5)
vector vturbulence (vector v, int octaves=6, float lacunarity=2, float gain=0.5)

`turbulence` is a variant of `fbm` where the absolute value of each noise

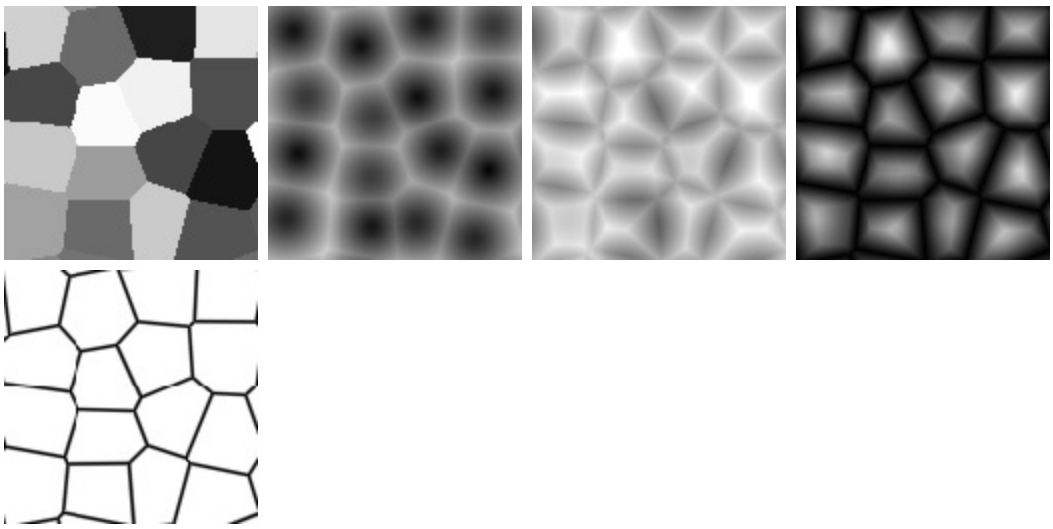
term is taken. This gives a more billowy appearance.

float voronoi (vector v, int type=1, float jitter=0.5, float fbmScale=0, int fbmOctaves=4, float fbmLacunarity=2, float fbmGain=0.5)
color cvoronoi (vector v, int type=1, float jitter=0.5, float fbmScale=0, int fbmOctaves=4, float fbmLacunarity=2, float fbmGain=0.5)
vector pvoronoi (vector v, float jitter=0.5, float fbmScale=0, int fbmOctaves=4, float fbmLacunarity=2, float fbmGain=0.5)

`voronoi` is a cellular noise pattern. It is a jittered variant of `cellnoise`. The type parameter describes different variants of the noise function. The `jitter` param controls how irregular the pattern is (0 is like ordinary cellnoise). The `fbm...` params can be used to distort the noise field. When `fbmScale` is zero (the default), there is no distortion. The remaining params are the same as for the `fbm` function.

Hint

Voronoi types 1 through 5:



Note

`cvoronoi` returns a random color for each cell and `pvoronoi` returns the

point location of the center of the cell.

Selection Functions

float choose (float index, float choice1, float choice2, […])

Chooses one of the supplied choices based on the index (assumed to be in the range `[0, 1]`).

int cycle (int index, int loRange, int hiRange)

Cycles through values between loRange and hiRange based on supplied index. This is an offset `mod` function. The result is computed as `loRange + value % (hiRange-loRange+1)`.

int pick (float index, int loRange, int hiRange, [float weights, …])

Picks values randomly between loRange and hiRange based on supplied index (which is automatically hashed). The values will be distributed according to the supplied weights. Any weights not supplied are assumed to be 1.0.

float wchoose (float index, float choice1, float weight1, float choice2, float weight2, […])

Chooses one of the supplied choices based on the index (assumed to be in range `[0, 1]`). The values will be distributed according to the supplied weights.

Hint

This example returns integer values between 1 and 10:

```
pick(value, 1, 10)
```

This example returns the values 1 and 2 twice and 2.5 times as often respectively as compared to the other values (3-10):

```
pick(value, 1, 10, 2, 2.5)
```

This example returns 10, 11, and 13 through 20 (12 is skipped due to zero weight):

```
pick(value, 10, 20, 1, 1, 0)
```

General Mathematical Constants and Functions

float PI

```
float PI = 3.14159...
```

float E

```
float E = 2.71828...
```

float abs (float x)

Absolute value of `x`.

float cbrt (float x)

Cube root.

float ceil (float x)

Next higher integer.

float exp (float x)

`E` raised to the `x` power.

float floor (float x)

Next lower integer.

float fmod (float x, float y)

Remainder of `x / y`.

Note

Also available as the `%` operator.

float log (float x)

Natural logarithm.

float log10 (float x)

Base 10 logarithm.

float max (float a, float b)

Greater of a and b.

float min (float a, float b)

Lesser of a and b.

float pow (float x, float y)

x to the y power.

Note

Also available as the ^ operator.

float round (float x)

Nearest integer.

float sqrt (float x)

Square root.

float trunc (float x)

Nearest integer towards zero.

Trigonometry Functions

float acos (float x)

Arc cosine.

float acosd (float x)

Arc cosine in degrees.

float acosh (float x)
Hyperbolic arc cosine.

float asin (float x)
Arc sine.

float asind (float x)
Arc sine in degrees.

float asinh (float x)
Hyperbolic arc sine.

float atan (float x)
Arc tangent.

float atand (float x)
Arc tangent in degrees.

float atan2 (float y, float x)
Arc tangent of `y/x` between `-PI` and `PI`.

float atan2d (float y, float x)
Arc tangent in degrees of `y/x` between `-180°` and `180°`.

float atanh (float x)
Hyperbolic arc tangent.

float cos (float x)
Cosine.

float cosd (float x)
Cosine in degrees.

float cosh (float x)
Hyperbolic cosine.

float deg (float x)
Radians to degrees.

float hypot (float x, float y)
Length of 2D vector `[x, y]`.

float rad (float x)
Degrees to radians.

float sin (float x)
Sine.

float sind (float x)
Sine in degrees.

float sinh (float x)
Hyperbolic sine.

float tan (float x)
Tangent.

float tand (float x)
Tangent in degrees.

float tanh (float x)
Hyperbolic tangent.

Vector Functions

float angle (vector a, vector b)
Angle between two vectors (in radians).

vector cross (vector a, vector b)
Vector cross product.

float dist (vector a, vector b)

Distance between two points.

float dot (vector a, vector b)
Vector dot product.

float length (vector v)
Length of vector.

vector norm (vector v)
Vector scaled to unit length.

vector ortho (vector a, vector b)
Vector orthographic to two vectors.

vector rotate (vector v, vector axis, float angle)
Rotates `v` around axis by the given `angle` (in radians).

vector up (vector v, vector up)
Rotates `v` such that the Y axis points in the given `up` direction.

Vector Support

Vectors (points, colors, or 3D vectors) may be intermixed with *scalars* (simple floating point values). If a scalar is used in a vector context, it is replicated into the three components, e.g. `0.5` becomes `[0.5, 0.5, 0.5]`.

If a vector is used in a scalar context, only the first component is used. One of the benefits of this is that all the functions that are defined to work with scalars automatically extend to vectors. For instance, `pick`, `choose`, `cycle`, `spline`, etc., will work just fine with vectors.

Arithmetic operators such as `+`, `*`, etc., and scalar functions are applied component-wise to vectors. For example, applying the `gamma` function to a map adjusts the gamma of all three color channels.

Curve Functions

Interpolation of parameter values to a set of control points is governed by the following functions.

color ccurve (float param, float pos0, color val0, int interp0, float pos1, color val1, int interp1, […])

Interpolates color ramp given by control points at `param`. Control points are specified by triples of parameters `pos_i`, `val_i`, and `interp_i`.

Hint

Interpolation codes are:

- 0 - none
- 1 - linear
- 2 - smooth
- 3 - spline
- 4 - monotone (non-oscillating) spline

float curve (float param, float pos0, float val0, int interp0, float pos1, float val1, int interp1, […])

Interpolates a 1D ramp defined by control points at `param`. Control points are specified by triples of parameters `pos_i`, `val_i`, and `interp_i`.

Hint

Interpolation codes are:

- 0 - none
- 1 - linear
- 2 - smooth
- 3 - spline
- 4 - monotone (non-oscillating) spline

float spline (float param, float y1, float y2, float y3, float y4, […])

Interpolates a set of values to the parameter specified where `y1`, …, `yn` are distributed evenly from `[0, 1]`.

Custom Plugins

Custom functions may be written in C++ and loaded as one or more dynamic plugins. See Writing Custom Expression Plugins [https://wdas.github.io/SeExpr/doxygen/plugins.html] for more details.

Warning

This functionality is not supported in Krita.

Separate Image

The separate image dialog allows you to separate the channels of the image into layers. This is useful in game texturing workflows, as well as more advanced CMYK workflows. When confirming, *Separate Image* creates a series of layers for each channel. So for RGB, it will create Red, Green and Blue layers, each representing the pixels of that channel, and for CMYK, Cyan, Magenta, Yellow and Key layers.

Source
: Which part of the image should be separated.

  Current Layer
  : The active layer.

  Flatten all layers before separation
  : The total image.

Alpha Options
: What to do with the alpha channel. All Krita images have an alpha channel, which is the transparency. How should it be handled when separating?

  Copy Alpha Channel to Each separate channel as an alpha channel.
  : This adds the transparency of the original source to all the separation layers.

  Discard Alpha Channel
  : Just get rid of the alpha altogether.

  Create separate separation from alpha channel
  : Create a separate layer with the alpha channel as a grayscale layer.

Downscale to 8bit before separating.

Convert the image to 8bit before doing the separation.

Output to color, not grayscale.

This results in the Red separation using black to red instead of black to white. This can make it easier to recombine using certain methods.

Activate the current channel

If the separated layers are in color, activate only the channel that was the origin of this separation layer.

Getting Krita logs

There are three different kinds of logs that Krita can produce. Depending on the issue, you might be asked for a specific one or for all of them. This page will tell you how to gather the necessary information to give to Krita developers or user supporters.

1. Krita Usage Log – this log contains your last 10 Krita sessions (one session means opening Krita). It shows times when you opened it, basic information about your system and Krita, and all files you created, opened and saved, including all auto-saves.
2. System information – this is not exactly a log, but a file that contains detailed system information related to Krita.
3. Crash log/backtrace – this log is created when Krita closes incorrectly because of an internal issue. This log is often necessary to get the issue fixed if developers cannot reproduce issue (repeat steps to get the crash).
4. Krita console output/Log Viewer output/DebugView output – this log contains anything random that Krita felt the need to report. It often contains some useful additional information that can help solving the issue.

Quick access

- Windows
    - Krita Usage Log
    - System information
    - Backtrace
    - Krita text output from Log Viewer (in GUI) or DebugView (external application)
- Linux
    - Krita Usage Log
    - System information

    - Backtrace
    - Krita text output from Log Viewer (in GUI) or console

- Mac

    - Krita Usage Log
    - System information
    - Backtrace
    - Krita text output from Log Viewer (in GUI) or console

Krita Usage Log

Through GUI

The easiest way to get Krita Usage Log is through Krita's GUI. Go to *Help* ▸ *Show Krita Log for bug reports*. A new dialog will open, showing the content of the log.

> *Changed in version 4.2.9:* In versions 4.2.0-4.2.9, Krita Usage Log is accessible through *Help* ▸ *Show system information for bug reports*. Note that for Krita versions 4.1.7 and below, no usage log is available and the menu entry only contains system information.

From the file system

Sometimes however it is not possible to use Krita's GUI, for example when it doesn't even open. Since logs are regular text files, you can get them from your file system by yourself.

The file is called `krita.log`. Location of the file:

Linux
: `$HOME/.local/share/krita.log`

Windows
: `%LOCALAPPDATA%\krita.log`

MacOS X

```
$HOME/Library/Application Support/krita.log
```

Note

In Windows you can simply paste this path into the Windows Explorer's search box, on the top bar, and it will find the file for you.

System information related to Krita

Through GUI

The easiest way to get system information related to Krita is through Krita's GUI. Go to *Help* ▸ *Show system information for bug reports*. A new dialog will open, showing the content.

> *Changed in version 4.2.9:* In versions 4.2.0-4.2.8, this menu entry contains both system information and Krita Usage Log.

From the file system

Sometimes however it is not possible to use Krita's GUI, for example when it doesn't even open. Since logs are regular text files, you can get them from your file system by yourself.

The file is called `krita-sysinfo.log`. Location of the file:

Linux
```
$HOME/.local/share/krita-sysinfo.log
```

Windows
```
%LOCALAPPDATA%\krita-sysinfo.log
```

MacOS X
```
$HOME/Library/Application Support/krita-sysinfo.log
```

Note

In Windows you can simply paste this path into the file browser textbox on the top bar and it will find you the file.

Crash log and backtrace

Location and the way to get a backtrace is different on all systems.

Windows

Usually, it is sufficient to share the content of *Help ▸ Show Krita Log for bug reports* as it contains the backtrace.

If you cannot open Krita because it crashes on startup, please provide the `%LOCALAPPDATA%\kritacrash.log`. Sometimes more detailed information is needed, then you will be asked to closely follow Dr. Mingw debugger guide.

Linux

On Linux, there are five ways of installing Krita.

- Using distribution packages
- Building Krita yourself from source
- Using a snap package
- Using a flatpak package
- Using the official appimage

Only distribution packages and built-from-source can produce usable crash logs/backtraces. For distribution packages, you will need to install the corresponding debug or dbg packages; the method for that is different from distribution to distribution. If you use distribution packages and the KDE Plasma Desktop, a crash dialog will be shown that has the backtrace in the “Developer” tab.

Otherwise, you have to use `gdb` in a terminal window.

1. Open Krita in `gdb`:

```
# if you have Krita installed from repositories, you may
# if not, write the path to the executable file
gdb path/to/krita
```

2. Disable pagination:

```
set pagination off
```

3. Run Krita:

```
run
```

4. Make it crash.

5. Get the short backtrace:

```
thread apply all bt
```

6. Get the long backtrace:

```
thread apply all bt full
```

7. Short and long backtraces save to separate text files.

8. From the short backtrace, it’s recommended to cut out all threads that are identical to some others or don’t seem to hold any additional information.

   If you feel like you know which part of the backtrace is the most important (it’s usually the longest thread), then cut it out and put this fragment in the bug report in a comment. Both backtraces still will be needed: attach them to the bug report as well.

   If you prefer not to make this decision, just attach both files with backtraces to the bug report.

Mac

On Mac it’s recommended to use `lldb`.

1. Open Terminal.app
2. Open Krita in `lldb`:

   ```
   lldb /Applications/krita.app/Contents/MacOS/krita
   ```

3. Run Krita:

   ```
   run
   ```

4. Make it crash.
5. Get the backtrace:

   ```
   thread backtrace all
   ```

6. Save the backtrace to a text file.
7. From the backtrace it's recommended to cut out all threads that are identical to some others or don't seem to hold any additional information to put into the comment (so it's easily accessible for the developer).

   If you feel like you know which part of the backtrace is the most important (it's usually the longest thread), then cut it out and put this fragment in the bug report in a comment. The full backtrace still will be needed: attach it to the bug report as well.

   If you prefer not to make this decision, just attach the file with the backtrace to the bug report.

Krita's text output

Most of Krita's text output can be gathered using *Log Viewer*. The only exception are messages from when Krita is starting up, so there is no GUI yet, or when it closes or crashes so no user interaction is possible after the event.

Through GUI

1. Go to *Settings* ▸ *Dockers* ▸ *Log Viewer*.
2. The first button from the left enables and disables logging, so make sure it is pressed.
3. Do the thing you want to get the output of.
4. Use the third button (tooltip says: *Save the log*) to save the log to a file.
5. Attach the file to the bug report.

From the console

Using the console is the most reliable way to get Krita's text output. This way is similar on Mac and Linux. Unfortunately, it's not possible on Windows. To check Windows' equivalent, see DebugView guide.

1. On Mac open `Terminal.app`, on Linux open your favourite terminal or console application.

2. Write the path to the Krita executable.

```
# On Linux, if installed from repositories:
krita
# On Linux, in all other cases:
#  (remember that if you want to reference a file from th
#   you're currently in, you need to write: './krita_file
#   and remember that this file need to have execution ri
path/to/krita
# On Mac:
/Applications/krita.app/Contents/MacOS/krita
```

3. Do the thing you want to get the output of.

4. Copy the content, save to a file and attach to the bug report.

From the DebugView

To get the text output of Krita on Windows, you need an external program called `DebugView`.

1. Download DebugView [https://docs.microsoft.com/en-us/sysinternals/downloads/debugview] if you haven't already. Click on the blue

bold *Download DebugView* text with underline, downloading should start immediately.
2. The file you download is a .zip archive. Windows 10 has a zip archive opener already included. Just extract all of the files somewhere. You can learn more about extracing on Windows extracting manual page [https://support.microsoft.com/en-us/help/4028088/windows-zip-and-unzip-files].
3. There is a file inside the archive that is called `DbgView.exe` (which you can see as `DbgView`, depending on your system settings). Double-click on it.
4. Let the program run and open Krita.
5. Do things you want to get output of.
6. Switch to DebugView and copy the content. Save to a file and attach to the bug report.

Split Layer

Splits a layer according to color. This is useful in combination with the Colorize Mask and the R +  shortcut to select layers at the cursor.

At the top, of the dialog there is a dropdown, here you can choose between…

Split Into Layers
: The image's colors is split into paint layers. Fantastic for artwork that works more with flats, such as the cel-shaded look.

Split Into Local Selection Masks
: *New in version 4.2.9.*

  The image's colors are outlined as a selection, and a Selection Masks is made. This is useful for artwork that has a more painterly look, with the selection masks making it easy to select several areas at once. Because selection masks are not paint layers, several of the options below are unavailable.

The other options are:

Put all new layers in a group layer
: Put all the result layers into a new group.

Put every layer in its own, separate group layer
: Put each layer into its own group.

Alpha-lock every new layer
: Enable the alpha-lock for each layer so it is easier to color.

Hide the original layer
: Turns off the visibility on the original layer so you won't get confused.

Sort layers by amount of non-transparent pixels

This ensures that the layers with large color swathes will end up at the top.

Disregard opacity
: Whether to take into account transparency of a color.

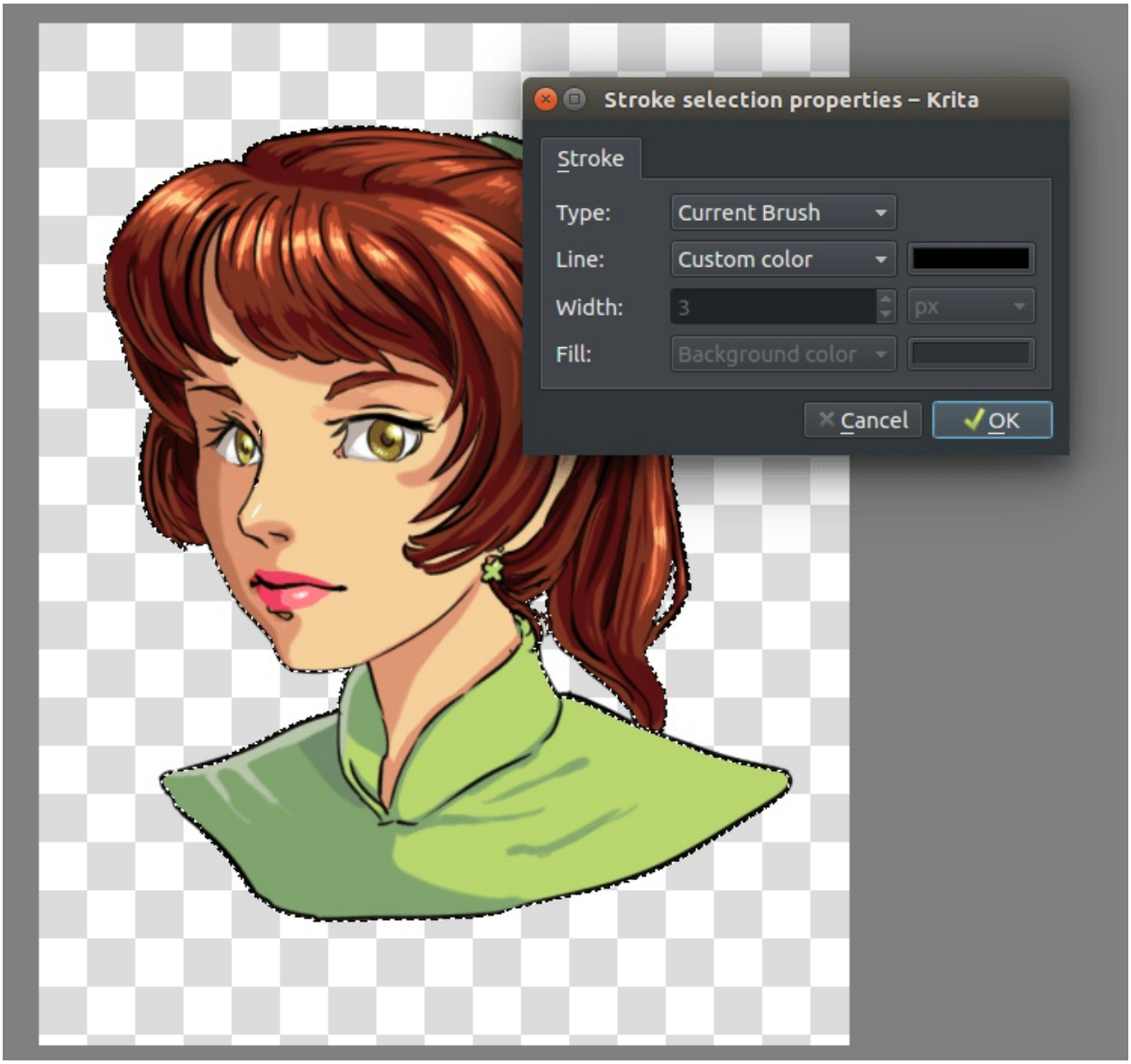
Fuzziness
: How precise the algorithm should be. The larger the fuzziness, the less precise the algorithm will be. This is necessary for splitting layers with anti-aliasing, because otherwise you would end up with a separate layer for each tiny pixel.

Palette to use for naming the layers
: Select the palette to use for naming. Krita will compare the main color of a layer to each color in the palette to get the most appropriate name for it.

Stroke Selection

Sometimes, you want to add an even border around a selection. Stroke Selection allows you to do this. It's under *Edit* ▸ *Stroke Selection*.

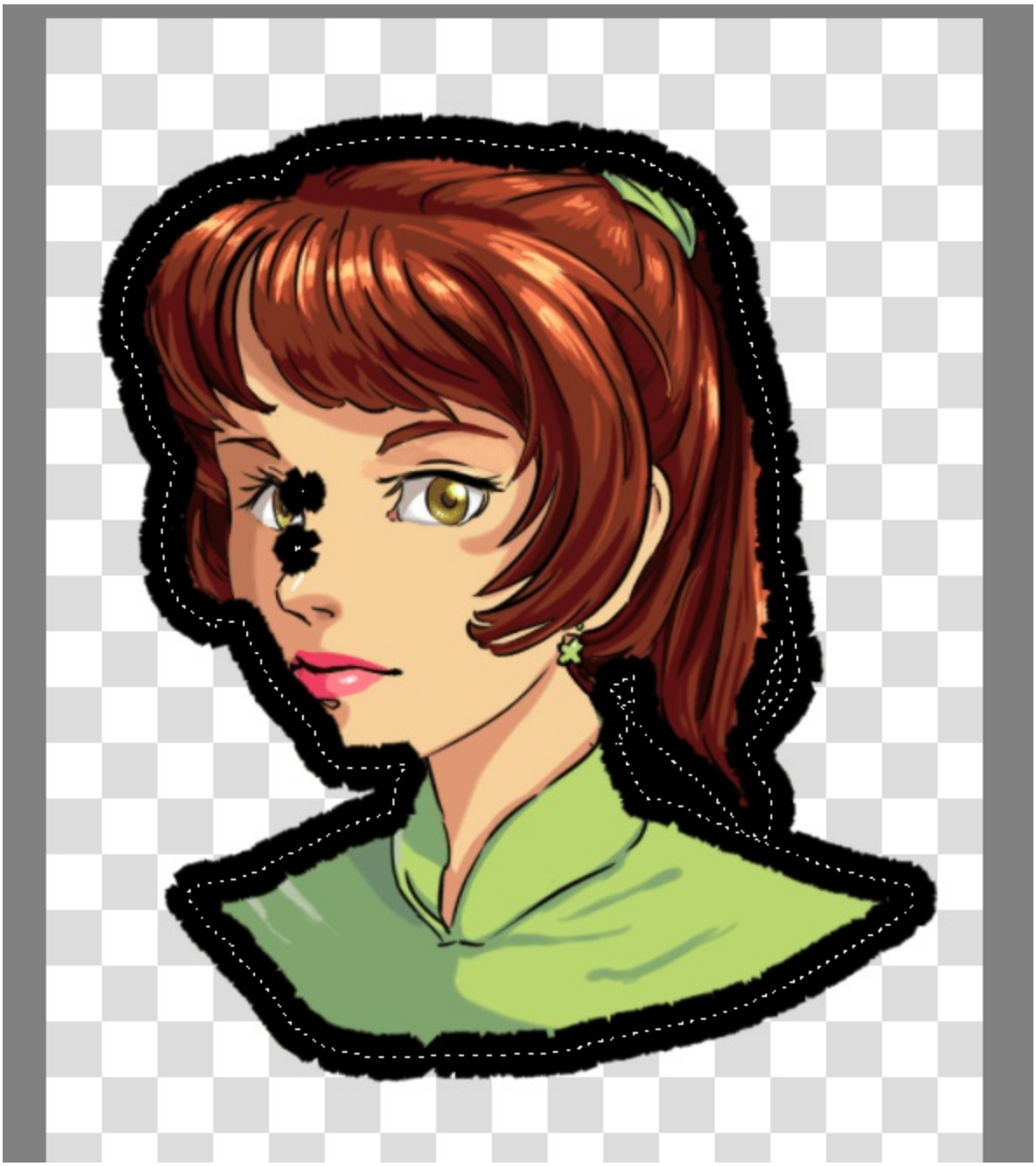
First make a selection and call up the menu:



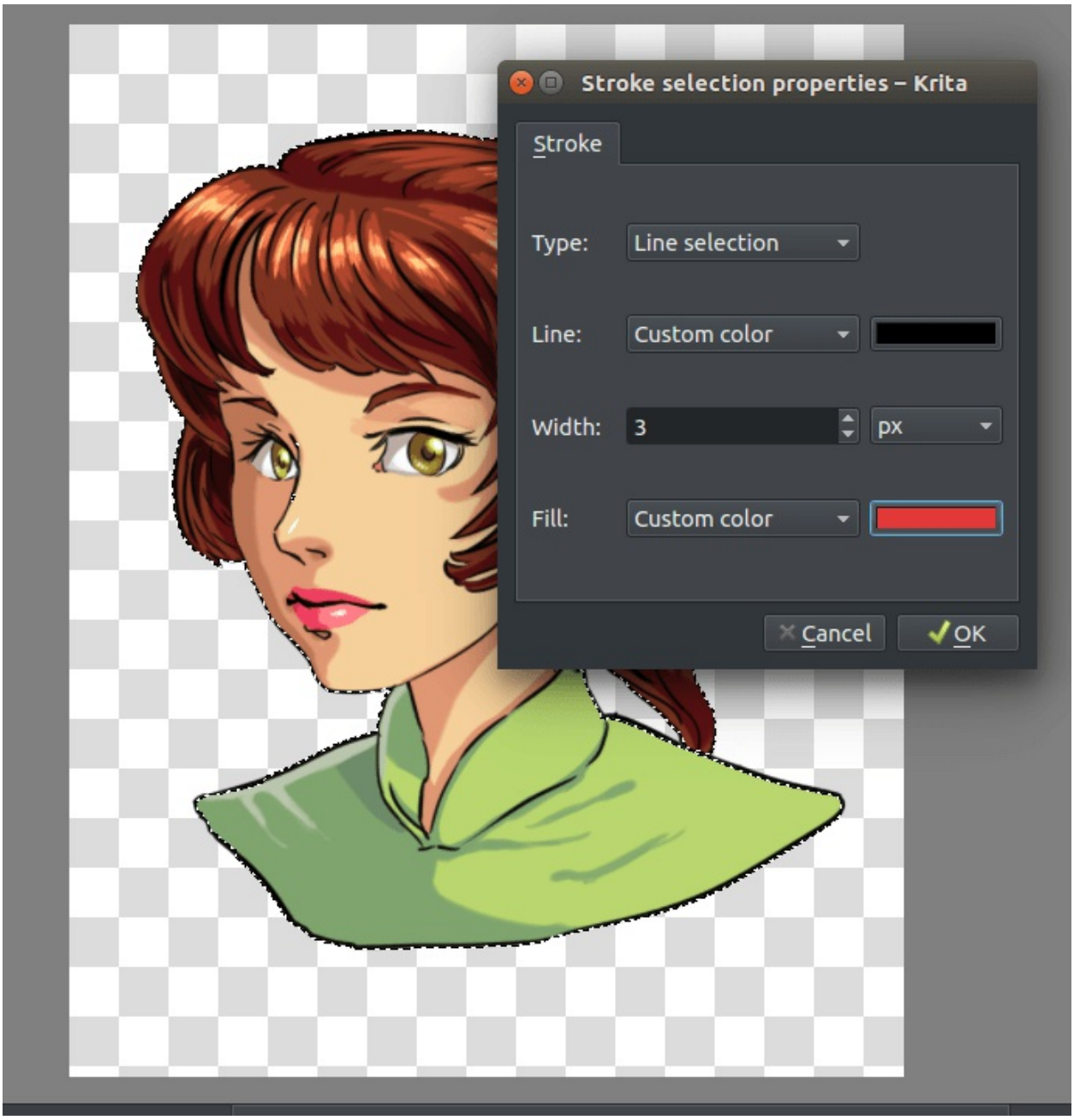
The main options are about using the current brush, or lining the selection with an even line. You can use the current foreground color, the background

color or a custom color.

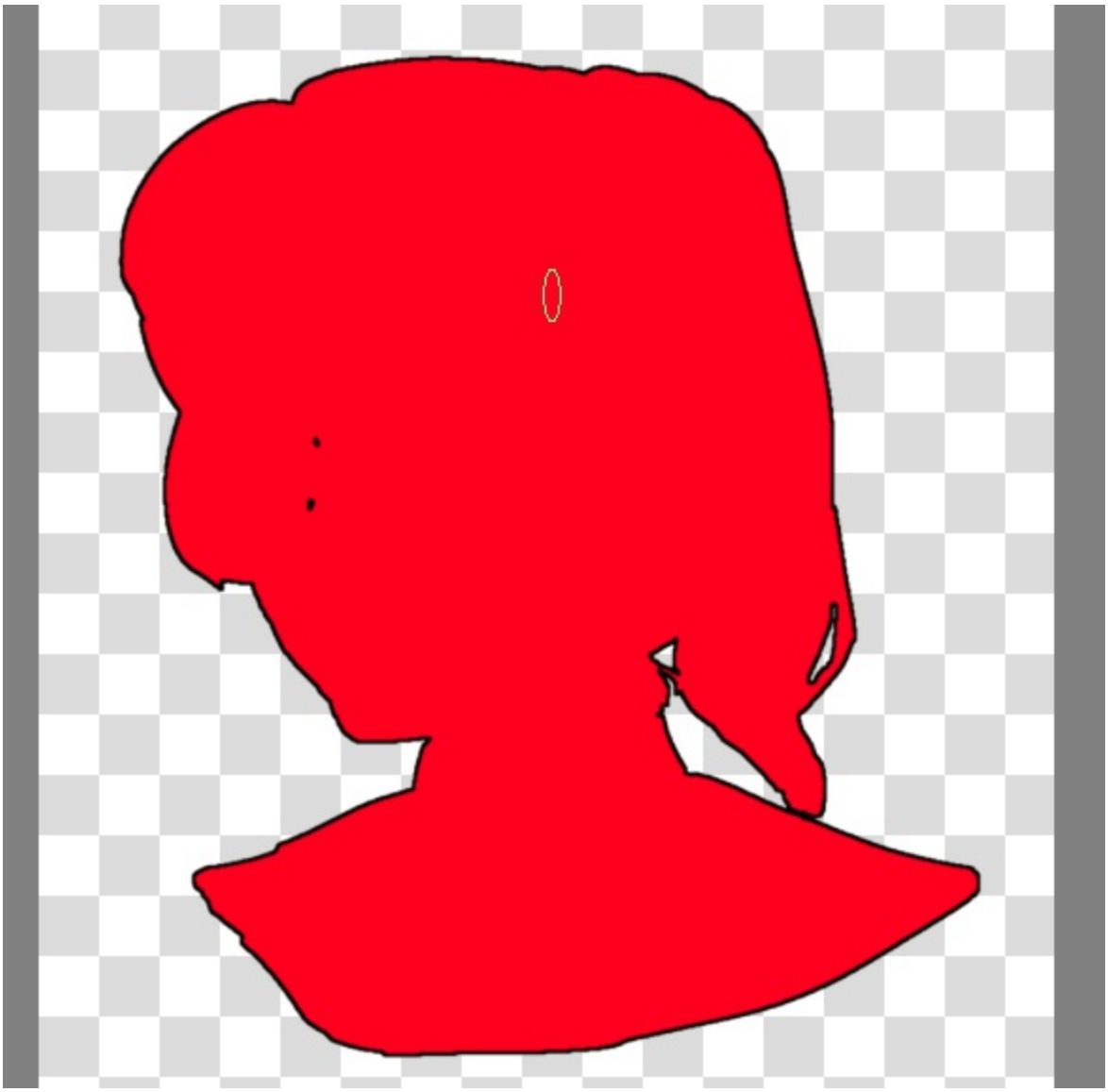
Using the current brush allows you to use textured brushes:



Lining the selection also allows you to set the background color, on top of the line width in pixels or inches:



This creates nice silhouettes:



Tools

The contents of the toolbox docker.

- Shape Selection Tool
- Shape Edit Tool
- Text Tool
- Gradient Editing Tool
- Pattern Editing Tool
- Calligraphy Tool
- Freehand Brush Tool
- Straight Line Tool
- Rectangle Tool
- Ellipse Tool
- Polygon Tool
- Polyline Tool
- Bezier Curve Tool
- Freehand Path Tool
- Dynamic Brush Tool
- Multibrush Tool
- Crop Tool
- Move Tool
- Transform Tool
- Fill Tool
- Gradient Tool
- Color Selector Tool
- Colorize Mask
- Smart Patch Tool
- Assistant Tool
- Reference Images Tool
- Measure Tool
- Rectangular Selection Tool
- Elliptical Selection Tool
- Outline Selection Tool
- Polygonal Selection Tool

- Contiguous Selection Tool
- Path Selection Tool
- Similar Color Selection Tool
- Magnetic Selection Tool
- Zoom Tool
- Pan Tool

Shape Selection Tool

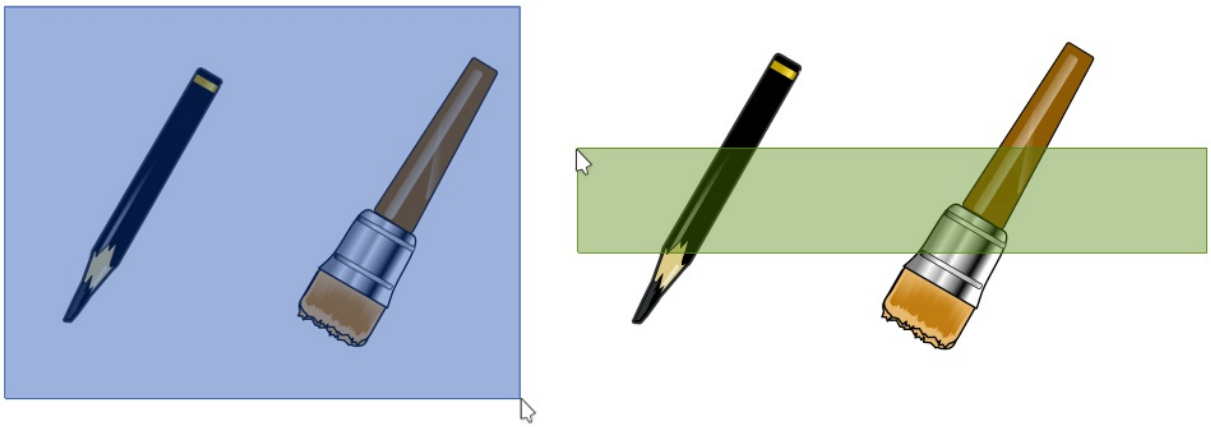
The shape selection tool used to be called the “default” tool. This had to do with Krita being part of an office suite once upon a time. But this is no longer the case, so we renamed it to its purpose in Krita: Selecting shapes! This tool only works on vector layers, so trying to use it on a paint layer will give a notification.

After you create vector shapes, you can use this tool to select, transform, and access the shape’s options in the tool options docker. There are a lot of different properties and things you can do with each vector shape.

Selection

Selecting shapes can be done by two types of actions:

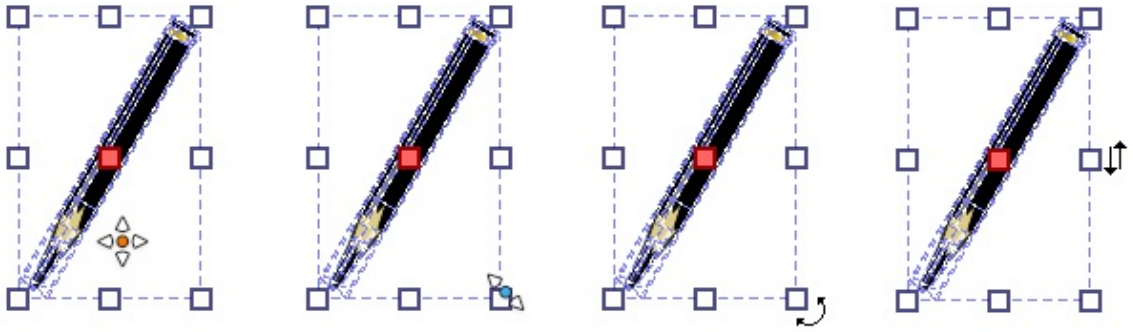
-  on a single shape to select it.
-  and drag to select multiple shapes.
    - *Blue selection* (drag left to right): selects only shapes fully covered.
    - *Green selection* (drag right to left): selects all the touched shapes.



Blue selection: left-to-right, selects fully covered images. – Green selection: right-to-left, selects touched shapes.

Placement, Scale, Angle and Distortion

Once an object is selected, a dashed bounding box will appear around it. The box will also have square handles. You can use this bounding box to do adjust: placement, scale, angle and distortion of the selected object.



Left to right: Placement, Scale, Angle and Distortion.

Placement

 and hold inside the bounding box, while holding move the shape to the desired position.

Scale

 and hold inside any of the square handles, move to adjust the dimensions of the object.

Angle

Place the cursor slightly outside any of the corner handles.  and drag to adjust the angle of the shape.

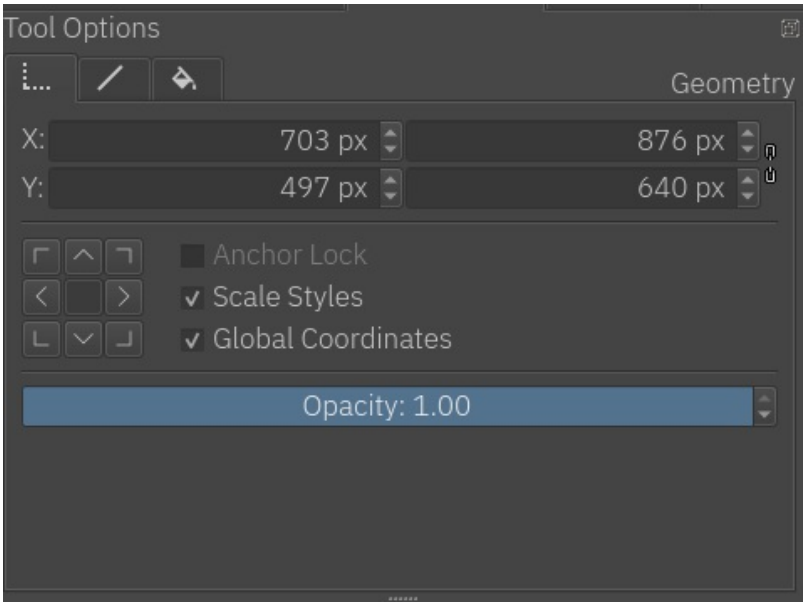
Distortion

Place the cursor slightly outside any of the middle handles.  and drag to skew the shape.

Tool Options

The tool options of this menu are quite involved, and separated over 3 tabs.

Geometry



Geometry is the first section in the tool options. This section allows you to set precisely the ‘x’ and ‘y’ coordinates, and also the width and height of the shape.

Uniform scaling

Enabled: when scaling, it will scale the stroke width with the shape.

Not enabled: when scaling, the stroke width will stay the same.

Global coordinates

Determines whether the width and height bars use the width and height of the object, while taking transforms into account.

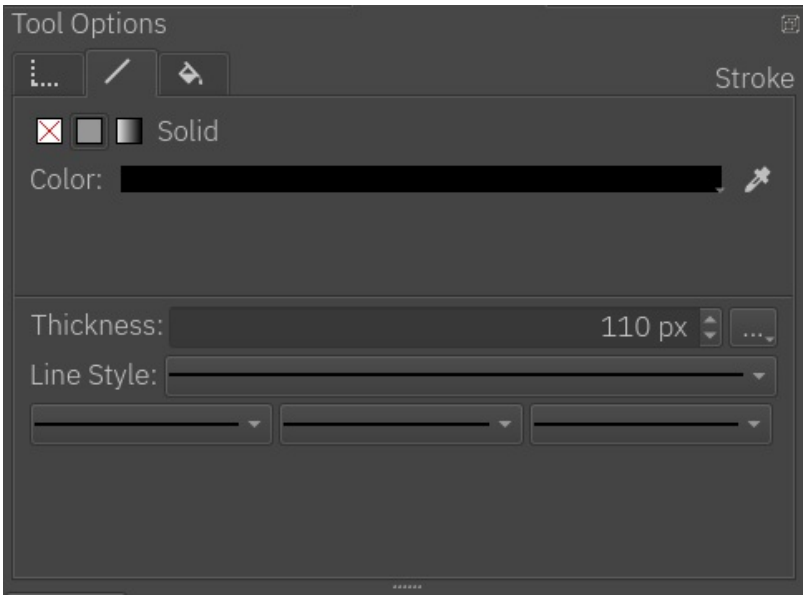
Opacity

The general opacity, or transparency, of the object. Opacity for stroke and fill are explained in the next two sections.

Warning

Anchor Lock is not implemented at the moment.

Stroke



The stroke tab determines how the stroke around the object should look.

The first set of buttons allows us to set the fill of the stroke: *None*, *Color* and

Gradient; the same options exist for the fill of the shape, please refer to the following "**Fill**" section for more details on how to use both of them.

Then, there are the settings for the stroke style:

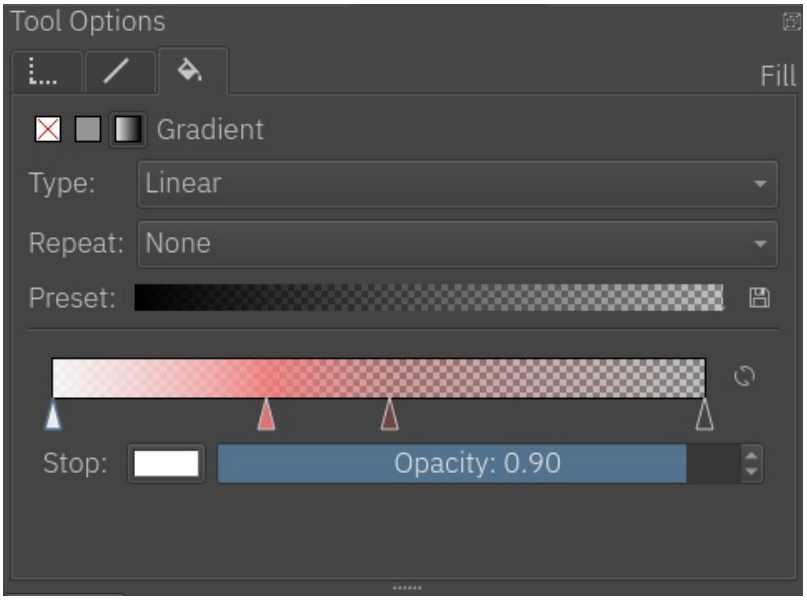
Thickness
: Sets the width of the stroke. When creating a shape, Krita will use the current brush size to determine the width of the stroke.

Cap and corner style
: Sets the stroke cap and stroke corner style, this can be accessed by pressing the three dots button next to the thickness entry.

Line-style
: Sets the line style of the stroke: *solid*, *dashes*, *dots*, or mixes of *dashes and dots*.

Markers
: Adds markers to the stroke. Markers are little figures that will appear at the start, end or all the nodes in between, depending on your configuration.

Fill



This section is about the color that fills the shape. As mentioned above in the **Stroke** section, the features are the same for both the fill of the stroke and the fill of the shape. Here is the explanation for both:

A fill can be: *solid color*, *gradient*, or *none* (transparent)

None
: No fill. It's transparent.

Color
: A flat color, you can select a new one by pressing the color button.

Gradient
: As the name implies this type fills the shape with a gradient. It has the following options:

  Type
  : A linear or radial gradient.

  Repeat
  : How the gradient repeats itself.

  Preset
  : A menu for selecting a base gradient from a set of predefined gradient presets, which can be edited as desired.

  Save Gradient
  : A quick way for saving the current gradient as a preset.

  Stops Options Line
  : A representation of how the gradient colors should look. The stops are represented by triangles. There are two stops by default one at the beginning and one at the end. You can create more stops just by clicking anywhere on the line. To select a stop  inside the triangle. To delete the stops,  drag them to left or right until the end of the line.

Flip Gradient
: A quick way to invert the order of the gradient.

Stop
: Choose a color for the current selected stop.

Opacity
: Choose the opacity for the current selected stop.

Hint

When a stop triangle is selected, it is highlighted with a slight blue outline. The selected stop triangle will change its color and opacity accordingly when these options are changed.

Hint

You can edit the gradient in two ways. The first one is the actual gradient in the docker that you can manipulate. Vectors always use stop-gradients. The other way to edit gradients is editing their position on the canvas.

Mesh Gradient
: Fills the shape with a Mesh Gradient. It has following options:

Stop
: Change the color of the selected stop. Only the color of a Corner can be changed (rectangle), changing the color of Bezier handle (circle) is not possible.

Rows
: Change the number of rows in a Mesh Gradient.

Columns
: Change the number of columns in a Mesh Gradient.

Smoothing

There are two possible values, *Bilinear* (default) and *Bicubic*. *Bilinear* is the linear interpolation of the color of stops, however it may create Mach Banding effect [https://en.wikipedia.org/wiki/Mach_bands]. *Bicubic* is the bicubic interpolation of the color stops, this should produce smoother gradient.

Note

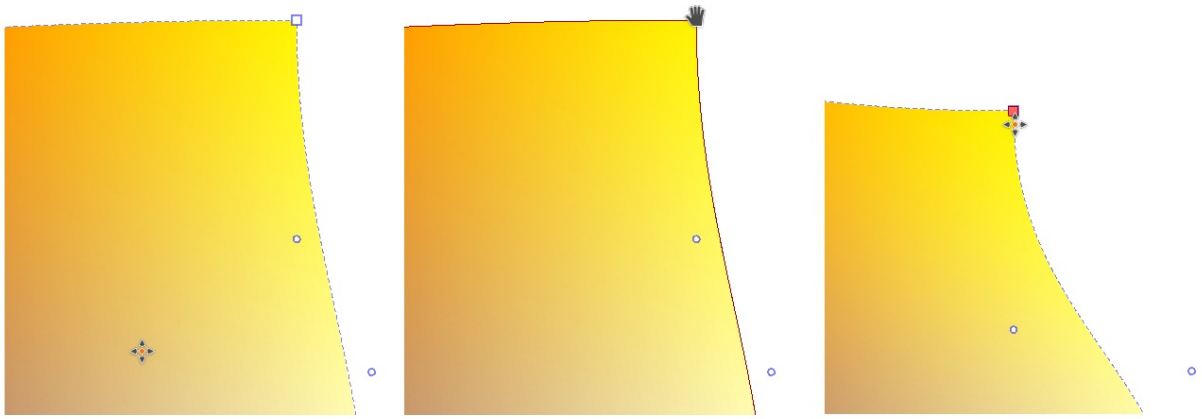
Mesh Gradients follow SVG draft 2 [https://svgwg.org/svg-next/pservers.html#MeshGradients] for the rendering.

Creating Mesh Gradients

When Mesh Gradients option is selected for a shape, Krita fills it with default mesh gradient, which is an alternating color (it alternates between background color and white). Changing number of rows and columns from Tool Options, will add more patches to the Mesh Gradient and corners can be changed individually.

Editing Mesh Gradients

The way to edit Mesh Gradients is pretty straight forward. Each curve is an individual Bezier Curve. There are two ways to change the structure, one is by dragging the corner (rectangle) and other is by dragging the handle (circle).



Left to right: Normal, Corner Hovered, Corner Moved and Selected.

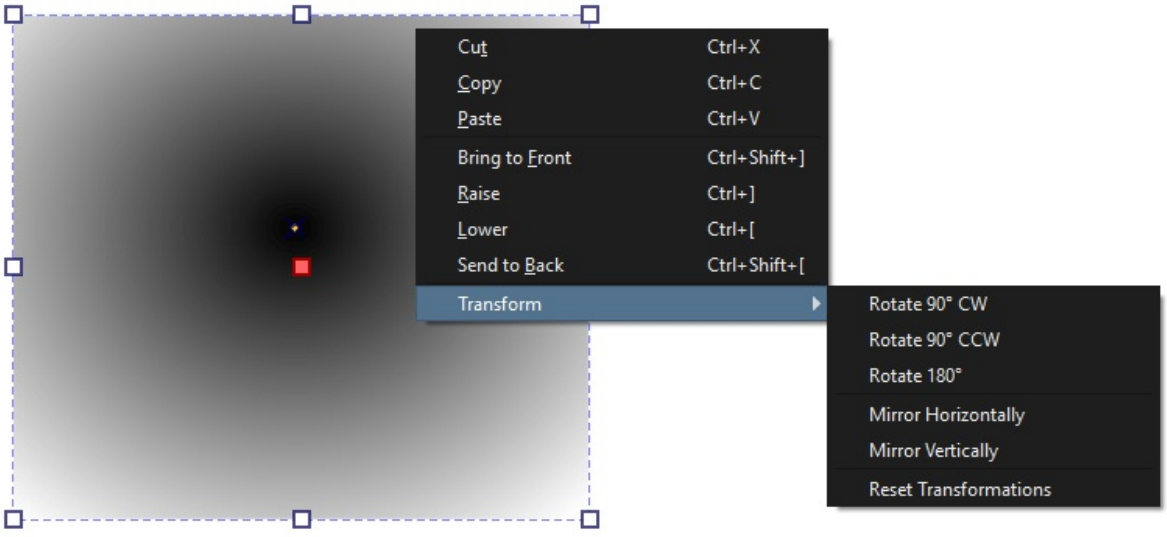
To change the color, a corner has to be selected and then Tool Options can be used to change the color of the selected corner.

Note

When importing from Inkscape, shapes may be grouped, which may not allow Krita to edit Mesh Gradients. To fix this, first ungroup (via ) them.

Right-click menu

The shape selection tool has a nice right click menu that gives you several features. If you have an object selected, you can perform various functions like cutting, copying, or moving the object to the front or back.

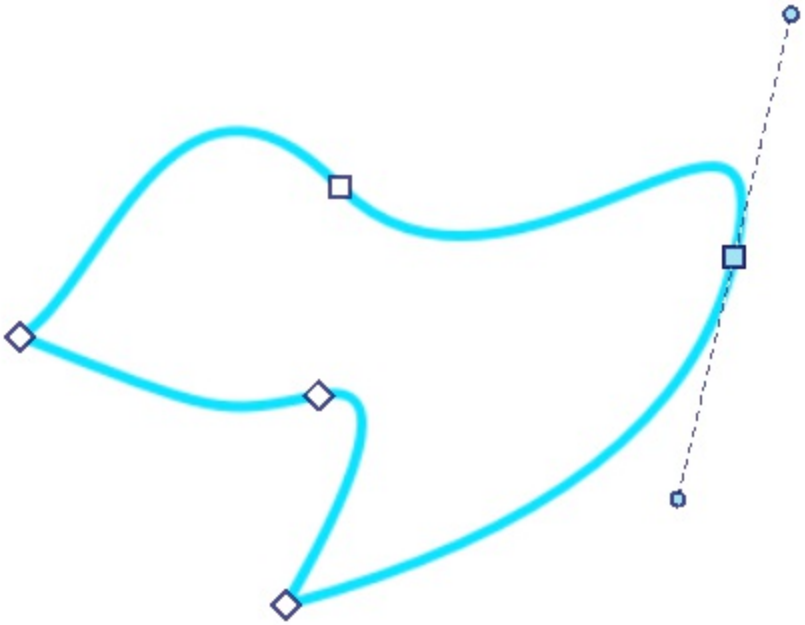


If you have multiple objects selected you can perform “Logical Operators” on them, or boolean operations as they are commonly called. It will be the last item on the right-click menu. You can unite, intersect, subtract, or split the

selected objects.

Shape Edit Tool

The shape editing tool is for editing vector shapes. In Krita versions before 4.0 it would only show up in the docker when you had a vector shape selected. In Krita 4.0, this tool is always visible and has the Shape Properties docker as a part of it.



You can access the Edit Shapes tool by clicking on the icon in the toolbox, but you can also access it by pressing the `Enter` key when in the Shape Selection tool and having a shape selected that can be most efficiently edited with the edit shapes tool (right now, that's all shapes but text).

On Canvas Editing of Shapes

As detailed further in the Tool Options, there's a difference between path shapes and specialized vector shapes that make it easy to have perfect

ellipses, rectangles and more.

Path Shapes

Path shapes can be recognized by the different nodes they have.

Paths in Krita are mostly bezier curves, and are made up of nodes. For straight lines, the nodes are connected by a line-segment and that's it. For curved lines, each node has a side handle to allow curving of that segment using the cubic bezier curve algorithm [https://en.wikipedia.org/wiki/B%C3%A9zier_curve#/media/File:B%C3%A9zier_3_big.gif].

What that means, in short, is that moving the side handles into a given direction will make the segment curve in that direction, and the longer the line of the node to the side handle, the stronger the curving.

Selecting Nodes for Editing

You can select a single node with  , they will turn bright green if selected.

 + `Shift` on unselected nodes will add them to a selection.

 + drag will make a selection rectangle. All nodes whose handles are touched by the rectangle will be selected. This combines with the  + `Shift` shortcut above.

Selected Nodes

You can add and remove side handles from a selected node with the  + `Shift` shortcut.

Krita has several node-types that allow you control the side handles more efficiently. These are the corner, smooth and symmetric modes.

Corner
: Represented by a circle, the corner type allows you to have handles that can point in different directions and have different lengths.

Smooth
: Represented by a square, the smooth type will ensure a smooth transition by always pointing the handles into opposite directions, but they can still have different lengths.

Symmetric
: Represented by a diamond, the symmetric node will force handles to always point in opposite directions and have the same length.

 `+ Ctrl` on a selected node will cycle between the node-types.

`Del` will remove the selected node.

Selected Segments

Segments are the lines between nodes. Hovering over a segment will show a dotted line, indicating it can be selected.

You can  and drag on a segment to curve it to the mouse point. Clicking on different parts of the segment and dragging will curve it differently.

Double  on a segment will add a node on the segment under the mouse cursor. The new node will be selected.

Other Shapes

Shapes that aren’t path shapes only have a single type of node: A small diamond like, that changes the specific parameters of that shape on-canvas. For example, you can change the corner radius on rectangles by dragging the nodes, or make the ellipse into a pie-segment.

Tool Options

Path shapes have options. The top left options are for converting to different anchor point types. The bottom left options are for adding or removing points. The top right options are for converting the line to different types. The bottom right options are for breaking and joining line segments.

The tool options of the Edit Shapes Tool change depending on the type of shape you have selected. With the exception of the path shape, all shapes have a *Convert to Path* action, which converts said shape to a path shape.

Path Shapes

Path shapes are the most common shape and can be made with the following tools:

- Bezier Curve Tool
- Straight Line Tool
- Polygon Tool
- Polyline Tool
- Freehand Path Tool

Node Editing
: Edit the nodes.

  Corner Point
  : Make the selected node a corner or cusp. This means that the side handles can point in different directions and be different lengths.

Smooth Point
: Make the selected node smooth. The two side handles will always point in opposite directions, but their length can be different.

Symmetric Point
: Make the selected node smooth. The two side handles will always point in opposite directions, and their length will stay the same.

Insert Point
: Insert a new node into the middle of the selected segment.

Remove Point
: Remove the selected node.

Line Segment Editing
: Edit line segments between nodes.

Segment To Line
: Make the current segment a straight line.

Segment To Curve
: Make the current segment a curve: It’ll add side handles for this segment to the nodes attached to it.

Make Line Point
: Turn the selected node into a sharp corner: This will remove the side handles.

Make Curve Point
: Turn the selected node into one that can curve: This will add side handles to the node.

Break at Point
: Break the path at this point.

Break Segment
: Break the path at the selected segment.

Join with Segment
: Join two nodes that are only attached on one side with a segment.

Merge Points
: Merge two nodes into one, if the nodes are adjacent or if both nodes are only attached on one side with a segment.

Rectangle Shapes

Rectangle shapes are the ones made with the Rectangle Tool. It has extra options to make rounded corners easy.

Corner radius x
: The radius of the x-axis of the corner curve.

Corner radius y
: The radius of the y-axis of the corner curve.

Ellipse Shapes

Ellipse shapes are the ones made with the Ellipse Tool.

Type
: The type of ellipse shape it is.

  Arc
  : An arc shape will keep the path open when it isn’t fully circular.

  Pie
  : A pie shape will add two extra lines to the center when the shape isn’t fully circular, like how one cuts out a piece from a pie.

  Cord

A cord shape will add a straight line between the two ends if the path isn’t fully circular, as if a cord is being strung between the two points.

Start Angle

The angle at which the shape starts.

End Angle

The angle at which the shape ends.

Close Ellipse

An action to quickly make the ellipse fully circular.

Text Tool

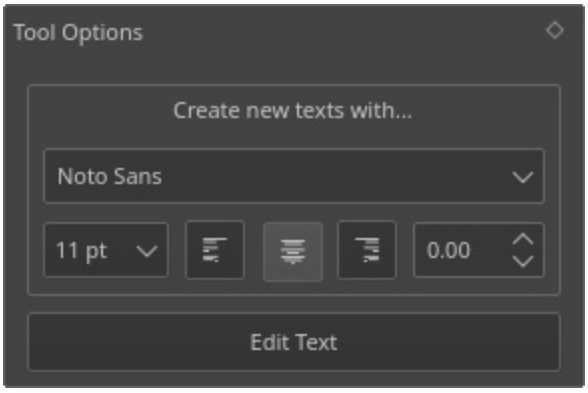
T

This tool allows you to add text to your artwork.

You use it by doing + drag to create a rectangular selection. When releasing a default text will be created and the text editor window will pop-up.

Hovering over other text shapes will highlight their bounding box. on a highlighted text will select it as the active text.

Tool Options



Create new texts with…

This contains features with which to create new texts, the following items are available:

Font

The letter type used by newly created texts.

Size in pt

The letter-size used by newly created texts. It is in pts (points), which is a common standard for fonts that is measured 72 points per inch. It therefore will stay proportionally the same size if you increase or decrease canvas dpi.

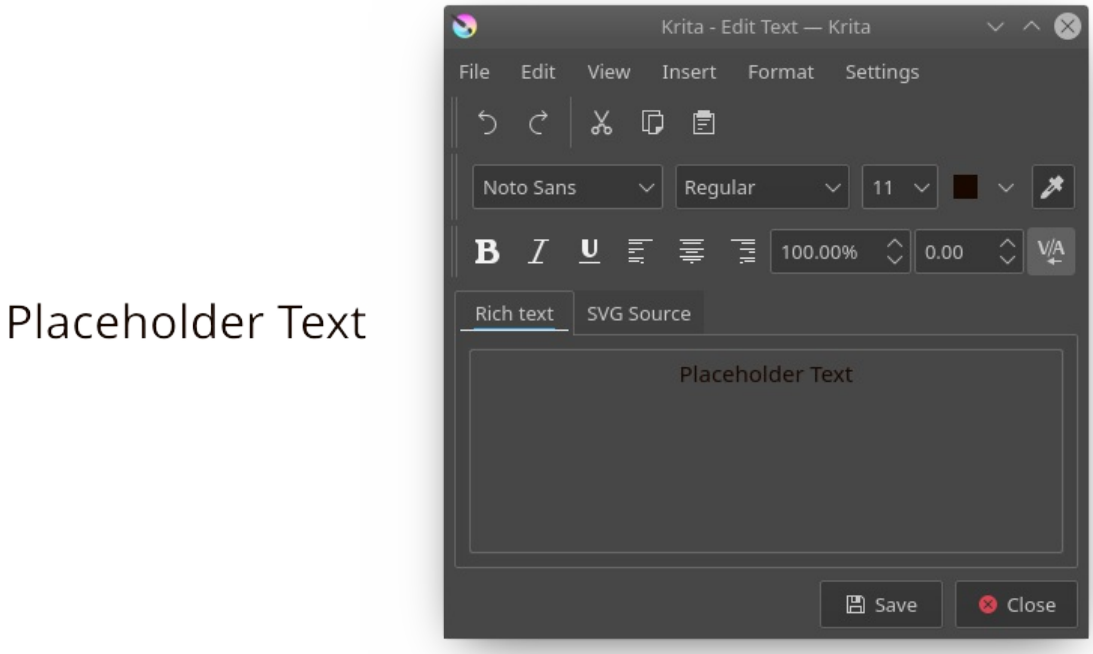
Anchor/Align text to the left/middle/right
: Text alignment. This allows you to align text to the left, center it, or to the right. This is called text-anchor because SVG 1.1's multiline text only uses text-anchor, and this is a slight bit different than text-align (and also the reason justify isn't available at the moment).

Letter Spacing
: The letter spacing used by newly created texts.

Edit Text
: This will summon the text editor for the currently selected shape. This can be quickly invoked with either pressing the `Enter` key or `double-click +`  shortcut on the shape.

Text Editor

A small window for all your text editing needs. The Text Editor has two tabs: Rich text and SVG source.



Activating

You can use the Text tool to first create a text box. There are a few options in the tool options if you want to customize how the text will be adding. You will need to drag a rectangle on the canvas to create the text area. Once your text is created, you can edit the text from two ways:

1. Select the text with the shape selection tool (first tool). Press the `Enter` key. The text editor will appear.
2. Select the text with the shape selection tool (first tool). Then click the Text tool. In the tool options there is an *Edit Text* button. When you click that the text editor window will appear.

Editing

If you are unfamiliar with the way SVG text works, use the rich text tab, it will allow you to edit the text as you see it, at the cost of not having all functionality.

If you are a little bit more familiar with SVG text, you can directly edit the SVG source. Do note that not all attributes and properties are

converted back to the rich text editor, so do be careful when switching back.

Press *Save* as you're done with your edits to preview them on canvas.

File

Save `Ctrl + S`
: Save current edits to the text on canvas.

Close `Ctrl + W`
: Close the editor.

Edit

Basic editing functions:

Undo `Ctrl + Z`
: Undo the last action in the text editor.

Redo `Ctrl + Shift + Z`
: Redo the last undone action in the text editor.

Cut `Ctrl + X`
: Cut selected text to the clipboard.

Copy `Ctrl + C`
: Copy selected text to the clipboard.

Paste `Ctrl + V`
: Paste text from the clipboard.

Select all `Ctrl + A`
: Select all text in the active editor.

Deselect `Ctrl + Shift + A`
: Deselect any selected text.

Find `Ctrl + F`
: Pops up a window with an input to find the given text in the active

editor.

Find Next `F3`
: Searches for the next text using the last search key.

Find Previous `Shift + F3`
: Searches for the last text using the last search key.

Replace… `Ctrl + R`
: Pops up a dialog with two inputs: The string you wish to find, and the string you wish to replace it with. Will always replace ALL found instances.

View

Zoom Out `Ctrl + -`
: Zoom out the text.

Zoom In `Ctrl + +`
: Zoom in the text.

Insert

Special Character… `Alt + Shift + C`
: Pops up a dialog that allows you to search for special characters that are difficult to type in with your keyboard.

Format

Bold `Ctrl + B`
: Set the font-weight to **bold**.

Italic `Ctrl + I`
: Sets the selected text *italic*.

Underline `Ctrl + U`
: Underline the selected text.

Strike-Through
: Adds a strike-through text decoration.

Superscript `Ctrl + Shift + P`
: Sets the text to super-script baseline.

Subscript `Ctrl + Shift + B`
: Sets the text to subscript baseline.

Weight
: Sets the font weight a little more specifically. Possibilities are… Light, Normal, Bold, and Black.

Align Left
: Align the selected paragraph to the left.

Align Center `Ctrl + Alt + C`
: Center the selected paragraph.

Align Right `Ctrl + Alt + R`
: Align the selected paragraph to the right.

Kerning
: Toggles kerning for selected text.

Settings

Settings…
: Calls up the text-editor settings dialog.

Text Editor Settings

The settings that can be configured for the text editor.

Editor Mode
: Whether you want both the Rich Text Editor and the SVG Source Editor, or only one of either.

Colors
: Here you can configure the syntax highlighting for the SVG source.

  Keyword

These highlights important characters like <, /, and >.

Element
: The format for highlighting the element tag name. **text** and **tspan** are examples of element names.

Attribute
: The format for highlighting the attributes of the tag. For example, `font-family`, when it isn't in the `style` tag is usually written as an attribute.

Value
: The format for highlighting value of attributes.

Comment
: This highlights XML comments, which are written as following: `<!-- This is an XML comment -->`. Comments are pieces of text that never get interpreted.

Editor Text Color
: The main color of the editor.

Editor background color
: The main background color of the editor.

Fonts
: This allows you to filter the selection of fonts in the editor by writing system. Some systems have a lot of fonts specifically for showing foreign languages properly, but these might not be useful for you. You just tick the writing systems which you use yourself, and the font drop-down will only show fonts that have full support for that language.

Fine typographic control with the SVG Source tab

So, the rich text editor cannot control all functionality that SVG text allows for. For that, you will need to touch the SVG source directly. But to do that, you will first need to go to the text editor settings and enable either *SVG*

Source or *Both* editor mode. The Rich Text editor will lose some information, so if you go all out, use *SVG Source*.

Word-spacing, Letter-spacing and Kerning

These three are written and read from the rich text tab, but only two of them can be controlled from Rich Text tab.

Kerning

Kerning, in SVG 1.1 [https://www.w3.org/TR/SVG/text.html#KerningProperty] behaves slightly differently than `font-kerning` in CSS. Krita by default uses the `auto` property for this, which means it is on. To turn it off use `kerning: 0;` in the `style` section of the text. Any other numeric value will be added to current `letter-spacing`.

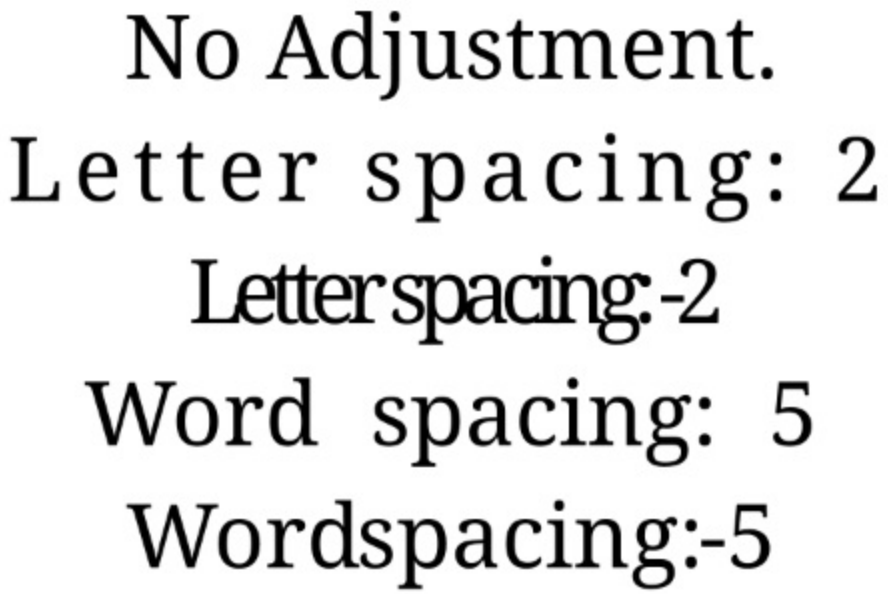

```
<text style="kerning:0; font-family:Dancing Script; font-size:
    <tspan>No Kerning on Valhalla Tower.</tspan>
</text>
```

Letter-spacing

This is the distance between letters in pts, usually. Just write `letter-spacing` in the `style` and add a distance in pts behind it. A negative value will decrease the value between letters.

Word-spacing

This is the extra distance between words, defaulting to pts. By default, `word-spacing: 0;` will have it use only the width of the space character for that font. A negative value will decrease the amount of space between words:




```
<text style="font-family:Noto Serif; font-size:12pt; font-size
    <tspan>No Adjustment.</tspan>
    <tspan style="letter-spacing:2" x="0" dy="22pt">Letter spa
    <tspan style="letter-spacing:-2" x="0" dy="22pt">Letter sp
    <tspan style="word-spacing:5" x="0" dy="22pt">Word spacing
    <tspan style="word-spacing:-5" x="0" dy="22pt">Word spacin
</text>
```

x, y, dx, dy

These are even finer-grained controls that can be used to position text. However, they CANNOT be reliably converted to the format of the rich text editor, because the rich text editor uses these to figure out if a line is a new-line and thus writes to these.

X and Y
: X and Y are absolute coordinates. But because you cannot change the absolute coordinates of the text from the editor, these get added to the position when they show up in a tspan.

dx and dy
: These are relative coordinates to the position of the previous letter.

Font-stretch and Small-caps

These can also be stored and written to the rich text tab’s internal format, but they don’t get used in the on screen text object.

Dominant Baseline, Alignment baseline, Font-size-adjust, Writing mode, Glyph-orientation, rotate

These are not stored in the rich text right now, and while they can be written into the SVG text, the SVG text-shape doesn’t do anything with them.

Krita generates `font-size-adjust` for the font when coming from rich text, as this can help designers when they want to use the SVG source as a basis for later adjustments.

Gradient Editing Tool

Deprecated since version 4.0: This tool has been removed in Krita 4.0, and its functionality has been folded into the Shape Selection Tool.

This tool allows you to edit the gradient on canvas, but it only works for vector layers. If you have a vector shape selected, and draw a line over the canvas, you will be able to see the nodes, and the stops in the gradient. Move around the nodes to move the gradient itself. Select the stops to change their color in the tool options docker, or to move their position in the on canvas gradient. You can select preset gradient in the tool docker to change the active shape’s gradient to use those stops.

Pattern Editing Tool

Deprecated since version 4.0: The pattern editing tool has been removed in 4.0, currently there's no way to edit pattern fills for vectors.

The Pattern editing tool works on Vector Shapes that use a Pattern fill. On these shapes, the Pattern Editing Tool allows you to change the size, ratio and origin of a pattern.

On Canvas-editing

You can change the origin by click dragging the upper node, this is only possible in Tiled mode.

You can change the size and ratio by click-dragging the lower node. There's no way to constrain the ratio in on-canvas editing, this is only possible in Original and Tiled mode.

Tool Options

There are several tool options with this tool, for fine-tuning:

First there are the Pattern options.

Repeat:
: This can be set to:

Original:
: This will only show one, unstretched, copy of the pattern.

Tiled (Default):
: This will let the pattern appear tiled in the x and y direction.

Stretch:
: This will stretch the Pattern image to the shape.

Reference point:
: Pattern origin. This can be set to:

- Top-left
- Top
- Top-right
- Left
- Center
- Right
- Bottom-left
- Bottom
- Bottom-right.

Reference Point Offset:
: For extra tweaking, set in percentages.

X:
: Offset in the X coordinate, so horizontally.

Y:
: Offset in the Y coordinate, so vertically.

Tile Offset:
: The tile offset if the pattern is tiled.

Pattern Size:
: Fine Tune the resizing of the pattern.

W:
: The width, in pixels.

H:
: The height, in pixels.

And then there's *Patterns*, which is a mini pattern docker, and where you can

pick the pattern used for the fill.

Calligraphy Tool

The Calligraphy tool allows for variable width lines, with input managed by the tablet. Press down with the stylus/left mouse button on the canvas to make a line, lifting the stylus/mouse button ends the stroke.

Tool Options

Fill

Doesn’t actually do anything.

Calligraphy

The drop-down menu holds your saved presets, the *Save* button next to it allows you to save presets.

Follow Selected Path
: If a stroke has been selected with the default tool, the calligraphy tool will follow this path.

Use Tablet Pressure
: Uses tablet pressure to control the stroke width.

Thinning
: This allows you to set how much thinner a line becomes when speeding up the stroke. Using a negative value makes it thicker.

Width
: Base width for the stroke.

Use Tablet Angle
: Allows you to use the tablet angle to control the stroke, only works for

tablets supporting it.

Angle
: The angle of the dab.

Fixation
: The ratio of the dab. 1 is thin, 0 is round.

Caps
: Whether or not an stroke will end with a rounding or flat.

Mass
: How much weight the stroke has. With drag set to 0, high mass increases the ‘orbit’.

Drag
: How much the stroke follows the cursor, when set to 0 the stroke will orbit around the cursor path.

Note

The calligraphy tool can be edited by the edit-line tool, but currently you can’t add or remove nodes without converting it to a normal path.

Freehand Brush Tool

The default tool you have selected on Krita start-up, and likely the tool that you will use the most.

The freehand brush tool allows you to paint on paint layers without constraints like the straight line tool. It makes optimal use of your tablet's input settings to control the brush-appearance. To switch the brush, make use of the brush-preset docker.

Hotkeys and Sticky keys

The freehand brush tool's hotkey is `B`.

- The alternate invocation is the ‘’color picker’’ (standardly invoked by the `Ctrl` key). Press the `Ctrl` key to switch the tool to “color picker”, use left or right click to pick fore and background color respectively. Release the `Ctrl` key to return to the freehand brush tool.
- The Primary setting is “size” (standardly invoked by the `Shift` key). Press the `Shift` key and drag outward to increase brush size. Drag inward to decrease it.
- You can also press the `V` key as a stickykey for the straight-line tool.

The hotkey can be edited in *Settings* ▸ *Configure Krita…* ▸ *Shortcuts*. The sticky-keys can be edited in *Settings* ▸ *Configure Krita…* ▸ *Canvas Input Settings*.

Tool Options

Smoothing

Smoothing, also known as stabilising in some programs, allows the program

to correct the stroke. Useful for people with shaky hands, or particularly difficult long lines.

The following options can be selected:

No Smoothing.
: The input from the tablet translates directly to the screen. This is the fastest option, and good for fine details.

Basic Smoothing.
: This option will smooth the input of older tablets like the Wacom Graphire 3. If you experience slightly jagged lines without any smoothing on, this option will apply a very little bit of smoothing to get rid of those lines.

Weighted smoothing:
: This option allows you to use the following parameters to make the smoothing stronger or weaker:

  Distance
  : The distance the brush needs to move before the first dab is drawn. (Literally the amount of events received by the tablet before the first dab is drawn.)

  Stroke Ending
  : This controls how much the line will attempt to reach the last known position of the cursor after the left-mouse button/or stylus is lifted. Will currently always result in a straight line, so use with caution.

  Smooth Pressure
  : This will apply the smoothing on the pressure input as well, resulting in more averaged size for example.

  Scalable Distance
  : This makes it so that the numbers involved will be scaled along the zoom level.

Stabilizer

This option averages all inputs from the tablet. It is different from weighted smoothing in that it allows for always completing the line. It will draw a circle around your cursor and the line will be a bit behind your cursor while painting.

Distance
: This is the strength of the smoothing.

Delay
: This toggles and determines the size of the dead zone around the cursor. This can be used to create sharp corners with more control.

Finish Line
: This ensures that the line will be finished.

Stabilize sensors
: Similar to *Smooth Pressure*, this allows the input (pressure, speed, tilt) to be smoother.

Scalable Distance
: This makes it so that the numbers involved will be scaled along the zoom level.

Assistants

Ticking this will allow snapping to Assistant Tool, and the hotkey to toggle it is `Ctrl + Shift + L`. See Painting with Assistants for more information.

The slider will determine the amount of snapping, with 1000 being perfect snapping, and 0 being no snapping at all. For situations where there is more than one assistant on the canvas, the defaultly ticked *Snap Single* means that Krita will only snap to a single assistant at a time, preventing noise. Unticking it allows you to chain assistants together and snap along them.

Straight Line Tool

This tool is used to draw lines. Click the  to indicate the first endpoint, keep the button pressed, drag to the second endpoint and release the button.

Hotkeys and Sticky Keys

To activate the Line tool from freehand brush mode, use the `V` key. Use other keys afterwards to constraint the line.

Use the `Shift` key while holding the mouse button to constrain the angle to multiples of 15º. You can press the `Alt` key while still keeping the  down to move the line to a different location.

Note

Using the `Shift` keys BEFORE pushing the holding the left mouse button/stylus down will, in default Krita, activate the quick brush-resize. Be sure to use the `Shift` key after.

Tool Options

The following options allow you to modify the end-look of the straight-line stroke with tablet-values. Of course, this only work for tablets, and currently only on Paint Layers.

Use sensors
: This will draw the line while taking sensors into account. To use this effectively, start the line and trace the path like you would when drawing

a straight line before releasing. If you make a mistake, make the line shorter and start over.

Preview

This will show the old-fashioned preview line so you know where your line will end up.

Rectangle Tool

This tool can be used to paint rectangles, or create rectangle shapes on a vector layer. Click and hold  to indicate one corner of the rectangle, drag to the opposite corner, and release the button.

Hotkeys and Sticky-keys

There’s no default hotkey for switching to rectangle.

If you hold the `Shift` key while drawing, a square will be drawn instead of a rectangle. Holding the `Ctrl` key will change the way the rectangle is constructed. Normally, the first mouse click indicates one corner and the second click the opposite. With the `Ctrl` key, the initial mouse position indicates the center of the rectangle, and the final mouse position indicates a corner. You can press the `Alt` key while still keeping  down to move the rectangle to a different location.

You can change between the corner/corner and center/corner drawing methods as often as you want by pressing or releasing the `Ctrl` key, provided that you keep  pressed. With the `Ctrl` key pressed, mouse movements will affect all four corners of the rectangle (relative to the center), without the `Ctrl` key, one of the corners is unaffected.

Tool Options

Fill

Not filled
: The rectangle will be transparent from the inside.

Foreground color
: The rectangle will use the foreground color as fill.

Background color
: The rectangle will use the background color as fill.

Pattern
: The rectangle will use the active pattern as fill.

Pattern Transform
: *New in version 4.4.*

  This enables upon using pattern as the fill, and has options for changing the pattern transformation a little.

  Rotation
  : This allows you to rotate the pattern used in the fill.

  Scale
  : This allows you to scale the pattern used in the fill.

Outline

No Outline
: The Rectangle will render without outline.

Brush
: The Rectangle will use the current selected brush to outline.

Note

On vector layers, the rectangle will not render with a brush outline, but rather a vector outline.

Anti-aliasing
: This toggles whether or not to give selections feathered edges. Some

people prefer hard-jagged edges for their selections.

Width
: Gives the current width. Use the lock to force the next selection made to this width.

Height
: Gives the current height. Use the lock to force the next selection made to this height.

Ratio
: *New in version 4.2.*

  Gives the current ratio. Use the lock to force the next selection made to this ratio.

  Round X

  The horizontal radius of the rectangle corners.

  Round Y

  The vertical radius of the rectangle corners.

Ellipse Tool

Use this tool to paint an ellipse. The currently selected brush is used for drawing the ellipse outline. Click and hold the left mouse button to indicate one corner of the ‘bounding rectangle’ of the ellipse, then move your mouse to the opposite corner. **Krita** will show a preview of the ellipse using a thin line. Release the button to draw the ellipse.

While dragging the ellipse, you can use different modifiers to control the size and position of your ellipse:

In order to make a circle instead of an ellipse, hold the `Shift` key while dragging. After releasing the `Shift` key any movement of the mouse will give you an ellipse again:

In order to keep the center of the ellipse fixed and only growing and shrinking the ellipse around it, hold the `Ctrl` key while dragging:

In order to move the ellipse around, hold the `Alt` key:

You can change between the corner/corner and center/corner dragging methods as often as you want by holding down or releasing the `Ctrl` key, provided you keep the left mouse button pressed. With the `Ctrl` key pressed, mouse movements will affect all four corners of the bounding rectangle (relative to the center), without the `Ctrl` key, the corner opposite to the one you are moving remains still. With the `Alt` key pressed, all four corners will

be affected, but the size stays the same.

Tool Options

Polygon Tool

With this tool you can draw polygons. Click the  to indicate the starting point and successive vertices, then double-click or press the `Enter` key to connect the last vertex to the starting point.

`Shift + Z` undoes the last clicked point.

Tool Options

Polyline Tool

Polylines are drawn like Polygon Tool, with the difference that the double-click indicating the end of the polyline does not connect the last vertex to the first one.

Bezier Curve Tool

You can draw curves by using this tool. Click the  to indicate the starting point of the curve, then click again for consecutive control points of the curve.

Krita will show a blue line with two handles when you add a control point. You can drag these handles to change the direction of the curve in that point.

On a vector layer, you can click on a previously inserted control point to modify it. With an intermediate control point (i.e. a point that is not the starting point and not the ending point), you can move the direction handles separately to have the curve enter and leave the point in different directions. After editing a point, you can just click on the canvas to continue adding points to the curve.

Pressing the `Del` key will remove the currently selected control point from the curve. Double-click the  on any point of the curve or press the `Enter` key to finish drawing, or press the `Esc` key to cancel the entire curve. You can use the `Ctrl` key while keeping the  pressed to move the entire curve to a different position.

While drawing the `Ctrl` key while dragging will push the handles both ways. The `Alt` key will create a sharp corner, and the `Shift` key will allow you to make a handle while at the end of the curve.  will undo the last added point.

Tool Options

New in version 4.1.3:

Autosmooth Curve

Toggling this will have nodes initialize with smooth curves instead of angles. Untoggle this if you want to create sharp angles for a node. This will not affect curve sharpness from dragging after clicking.

Angle Snapping Delta

The angle to snap to.

Activate Angle Snap

Angle snap will make it easier to have the next line be at a specific angle of the current. The angle is determined by the *Angle Snapping Delta*.

Freehand Path Tool

With the Freehand Path Tool you can draw a path (much like the Shape Brush Engine) the shape will then be filled with the selected color or pattern and outlined with a brush if so chosen. While drawing a preview line is shown that can be modified in pattern, width and color.

This tool can be particularly good for laying in large swaths of color quickly.

Dynamic Brush Tool

Add custom smoothing dynamics to your brush. This will give you similar smoothing results as the normal freehand brush. There are a couple options that you can change.

Mass
: Average your movement to make it appear smoother. Higher values will make your brush move slower.

Drag
: A rubberband effect that will help your lines come back to your cursor. Lower values will make the effect more extreme.

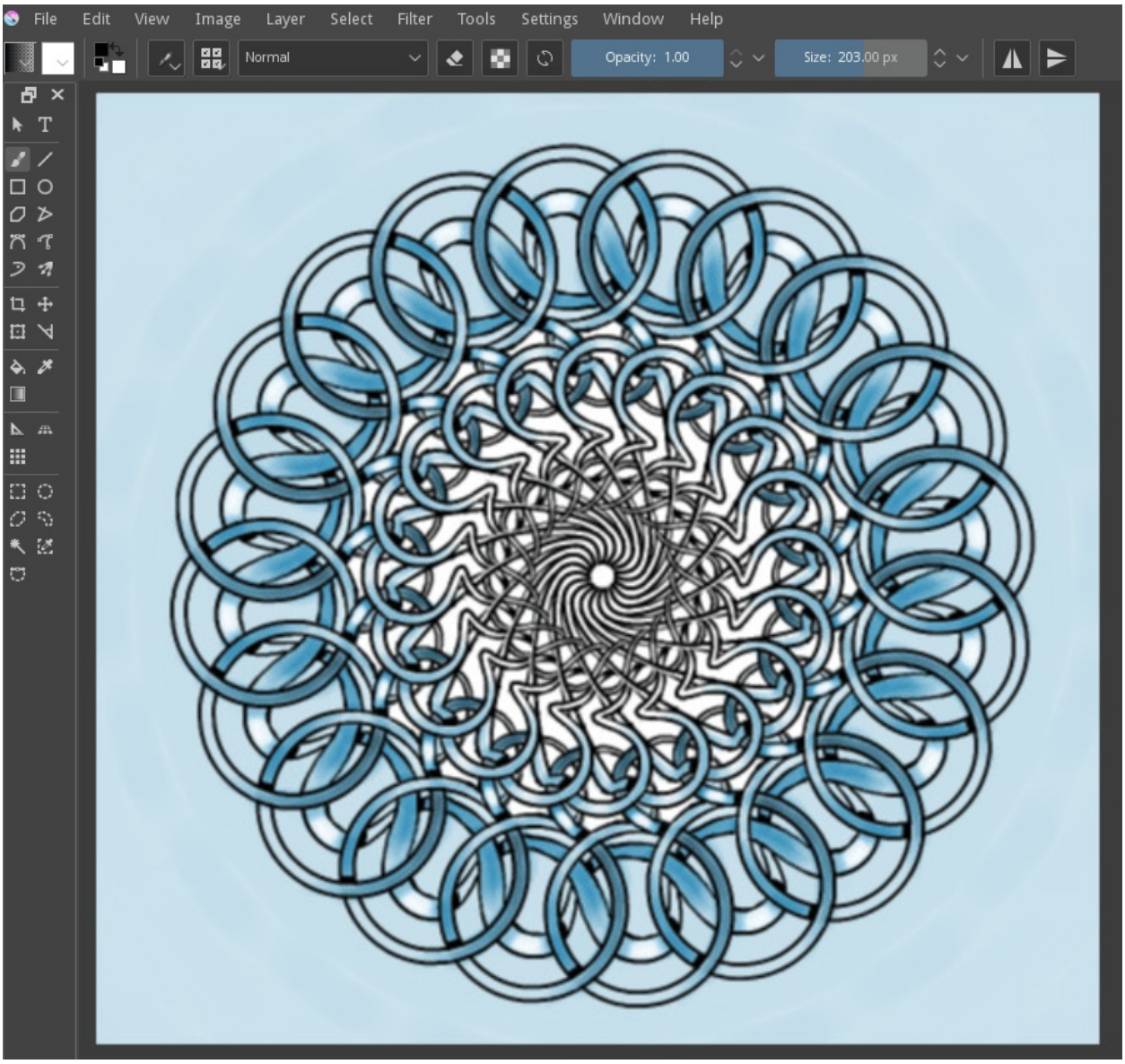
Recommended values are around 0.02 Mass and 0.92 Drag.

Multibrush Tool

The Multibrush tool allows you to draw using multiple instances of a freehand brush stroke at once, it can be accessed from the Toolbox docker or with the default shortcut `Q`. Using the Multibrush is similar to toggling the Mirror Tools, but the Multibrush is more sophisticated, for example it can mirror freehand brush strokes along a rotated axis.

The settings for the tool will be found in the tool options dock.

The multibrush tool has three modes and the settings for each can be found in the tool options dock. Symmetry and mirror reflect over an axis which can be set in the tool options dock. The default axis is the center of the canvas.



The three modes are:

Symmetry

Symmetry will reflect your brush around the axis at even intervals. The slider determines the number of instances which will be drawn on the canvas.

Mirror

Mirror will reflect the brush across the X axis, the Y axis, or both.

Translate

Translate will paint the set number of instances around the cursor at the radius distance.

Snowflake
: This works as a mirrored symmetry, but is a bit slower than symmetry+toolbar mirror mode.

Copy Translate
: This allows you to set the position of the copies relative to your own cursor. To set the position of the copies, first toggle *Add*, and then  the canvas to place copies relative to the multibrush origin. Finally, press *Add* again, and start drawing to see the copy translate in action.

The assistant and smoothing options work the same as in the Freehand Brush Tool, though only on the real brush and not its copies.

Crop Tool

The crop tool can be used to crop an image or layer. To get started, choose the *Crop* tool and then click once to select the entire canvas. Using this method you ensure that you don't inadvertently grab outside of the visible canvas as part of the crop. You can then use the options below to refine your crop. Press the `Enter` key to finalize the crop action, or use the *Crop* button in the tool options docker.

At its most basic, the crop tool allows you to size a rectangle around an area and reduce your image or layer to only that content which is contained within that area. There are several options which give a bit more flexibility and precision.

The two numbers on the left are the exact horizontal position and vertical position of the left and top of the cropping frame respectively. The numbers are the right are from top to bottom: width, height, and aspect ratio. Selecting the check boxes will keep any one of these can be locked to allow you to manipulate the other two without losing the position or ratio of the locked property.

Center
: Keeps the crop area centered.

Grow
: Allows the crop area to expand beyond the image boundaries.

Applies to
: Lets you apply the crop to the entire image or only to the active layer. When you are ready, hit the *Crop* button and the crop will apply to your image.

Decoration
: Help you make a composition by showing you lines that divide up the screen. You can for example show thirds here, so you can crop your

image according to the Rule of Thirds [https://en.wikipedia.org/wiki/Rule_of_thirds].

Continuous Crop

If you crop an image, and try to start a new one directly afterwards, Krita will attempt to recall the previous crop, so you can continue it. This is the *continuous crop*. You can press the `Esc` key to cancel this and crop anew.

Move Tool

With this tool, you can move the current layer or selection by dragging the mouse.

Move current layer
: Anything that is on the selected layer will be moved.

Move layer with content
: Any content contained on the layer that is resting under the four-headed Move cursor will be moved.

Move the whole group
: All content on all layers will move. Depending on the number of layers this might result in slow and, sometimes, jerky movements. Use this option sparingly or only when necessary.

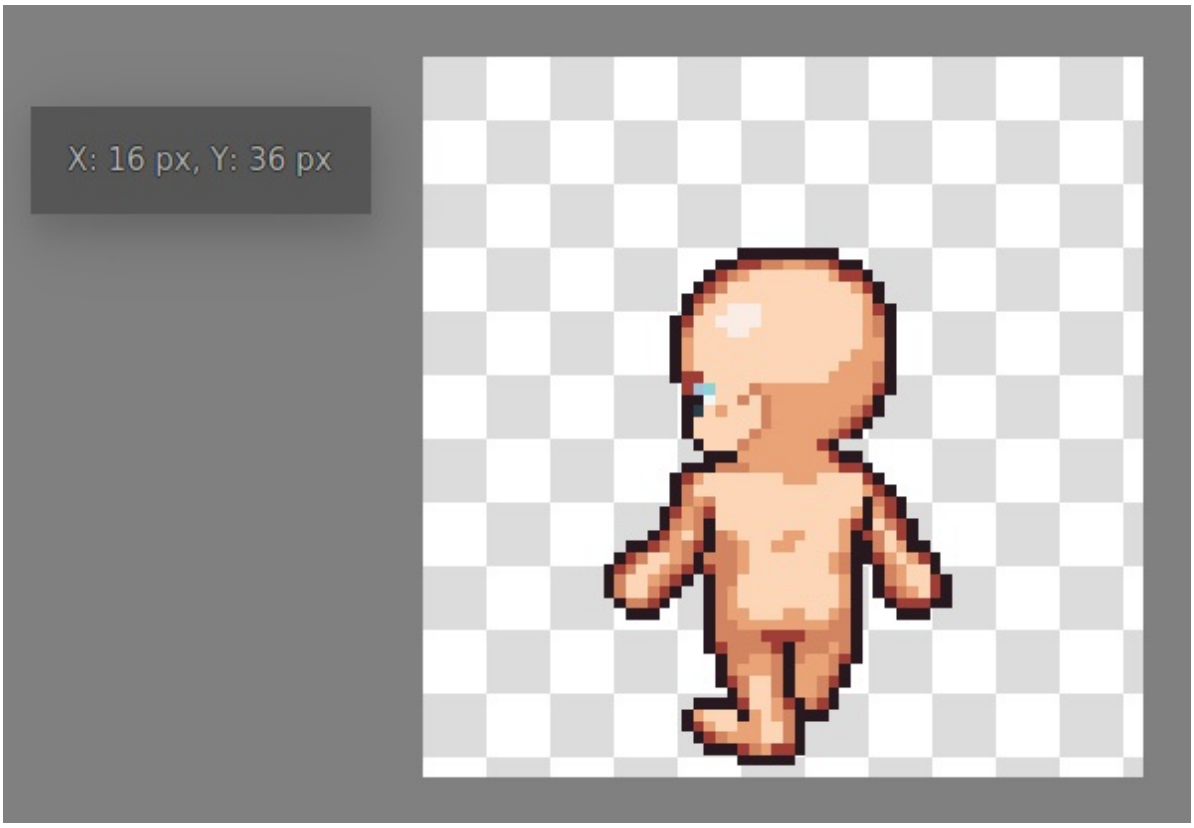
Shortcut move distance (3.0+)
: This allows you to set how much, and in which units, the `Left Arrow`, `Up Arrow`, `Right Arrow` and `Down Arrow` cursor key actions will move the layer.

Large Move Scale (3.0+)
: Allows you to multiply the movement of the Shortcut Move Distance when pressing the `Shift` key before pressing a direction key.

Show coordinates
: When toggled will show the coordinates of the top-left pixel of the moved layer in a floating window.

Constrained movement
: If you click, then press the `Shift` key, then move the layer, movement is

constrained to the horizontal and vertical directions. If you press the `Shift` key, then click, then move, all layers will be moved, with the movement constrained to the horizontal and vertical directions.



Position

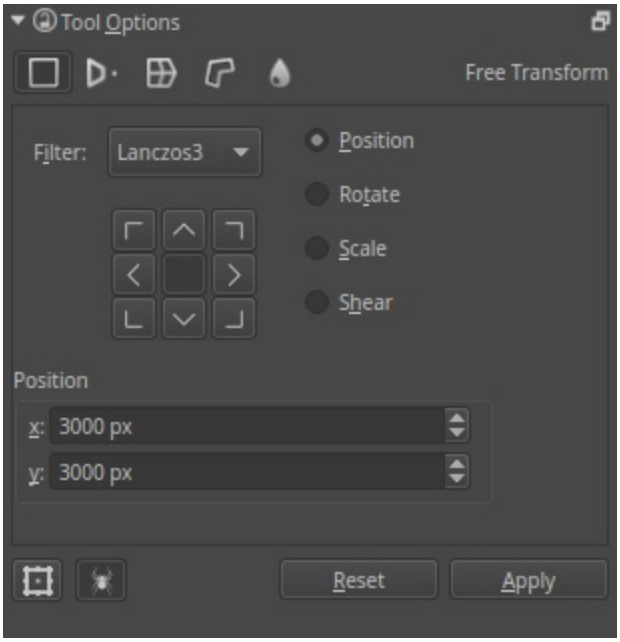
Gives the top-left coordinate of the layer, can also be manually edited.

Transform Tool

The Transform tool lets you quickly transform the current selection or layer. Basic transformation options include resize, rotate and skew. In addition, you have the option to apply advanced transforms such as Perspective, Warp, Cage and Liquid. These are all powerful options and will give you complete control over your selections/layers.

When you first invoke the tool, handles will appear at the corners and sides, which you can use to resize your selection or layer. You can perform rotations by moving the mouse above or to the left of the handles and dragging it. You can also click anywhere inside the selection or layer and move it by dragging the mouse.

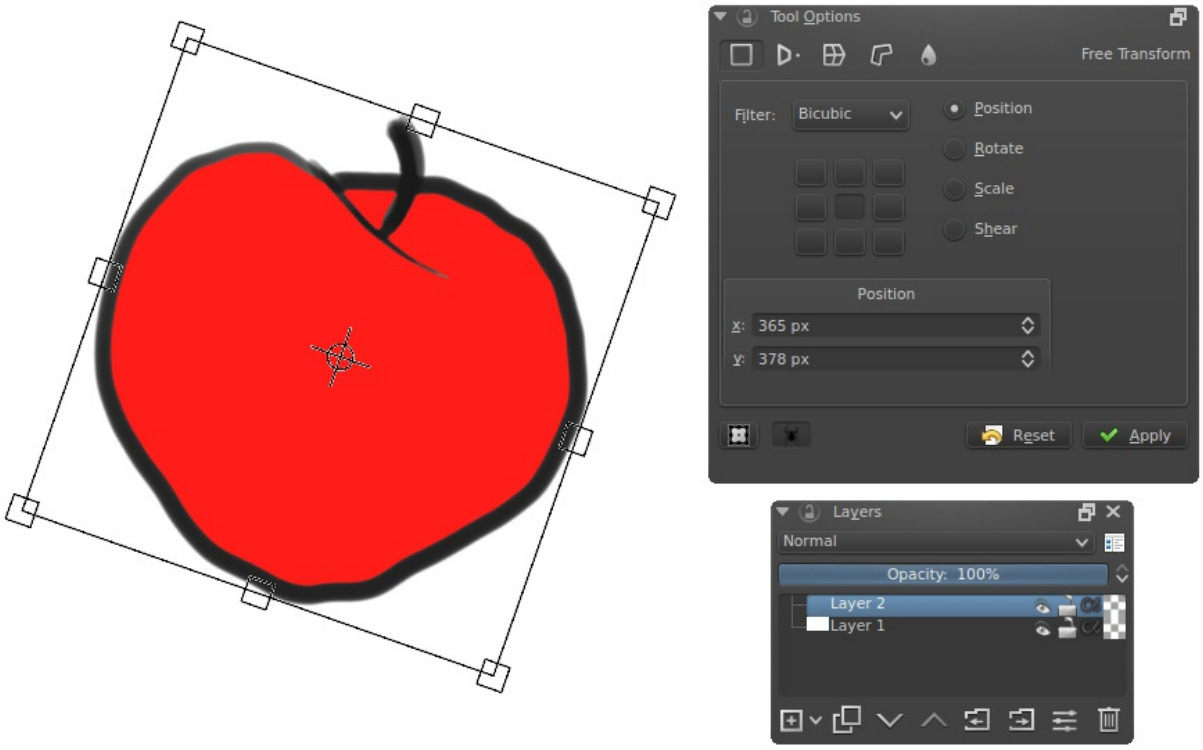
You can fine-tune the transform tool parameters using tool options docker. The parameters are split between five tabs: Free Transform, Warp, Perspective, Cage and Liquify.



Free Transform docker.

Free transform

This allows you to do basic rotation, resizing, flipping, and even perspective skewing if you hold the `Ctrl` key. Holding the `Shift` key will maintain your aspect ratio throughout the transform.



Free transform in action.

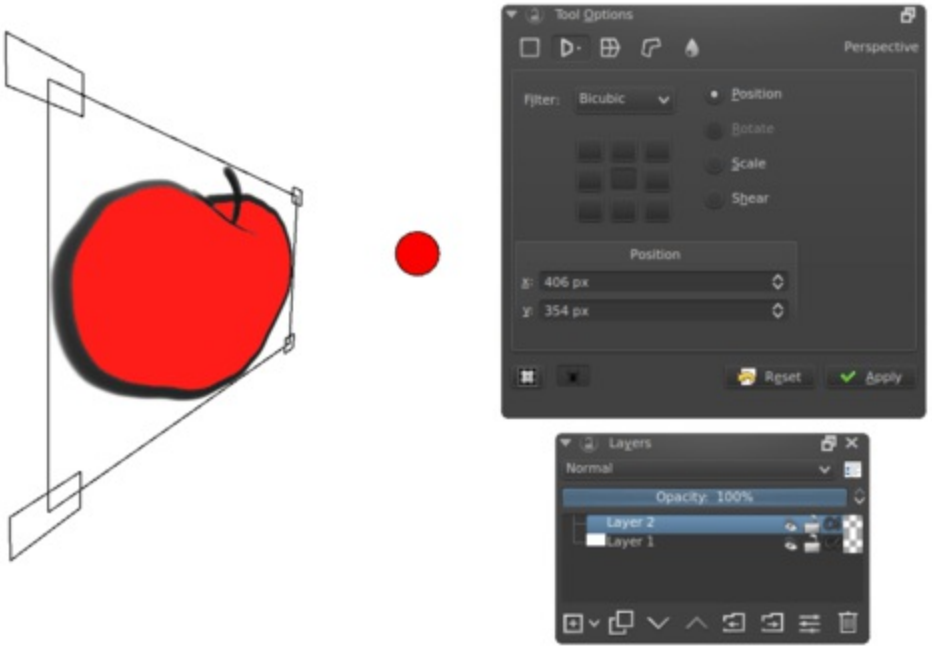
If you look at the bottom, there are quick buttons for flipping horizontally, vertically and rotating 90 degrees left and right. Furthermore, the button to the left of the anchor point widget allows you to choose whether to always transform using the anchor point, or not.

Video of how to use the anchor point for resizing. [https://www.youtube.com/watch?v=grzccBVd0O8]

Perspective

While free transform has some perspective options, the perspective transform allows for maximum control. You can drag the corner points, or even the designated vanishing point.

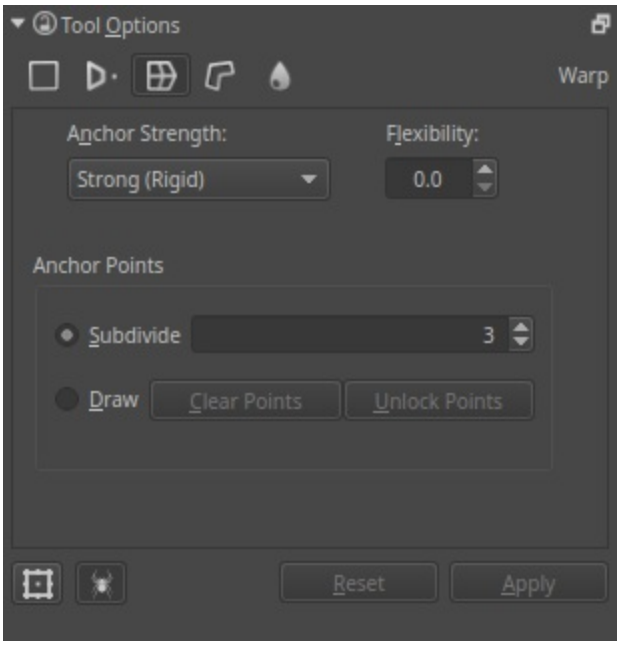
You can also change the size, shear and position transform while remaining in perspective with the tool-options.



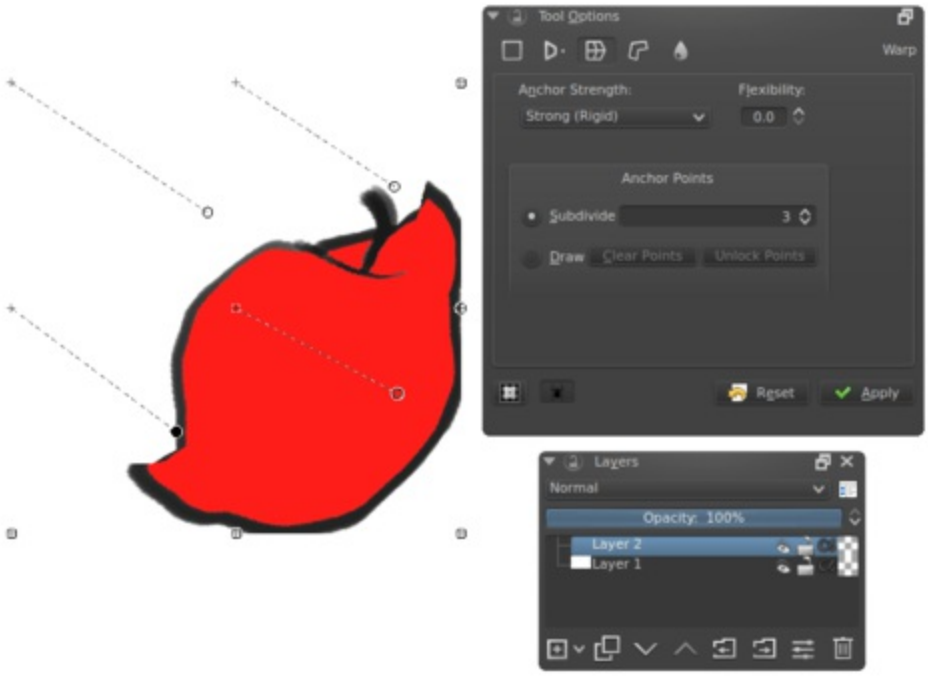
Perspective transform.

Warp

Warp allows you to deform the image by dragging from a grid or choosing the dragging points yourself.



Warp Option.



Free transform in action.

There are warp options: Rigid, Affine and Similitude. These change the algorithm used to determine the strength of the deformation. The flexibility determines, how strong the effect of moving these points are.

Anchor Points

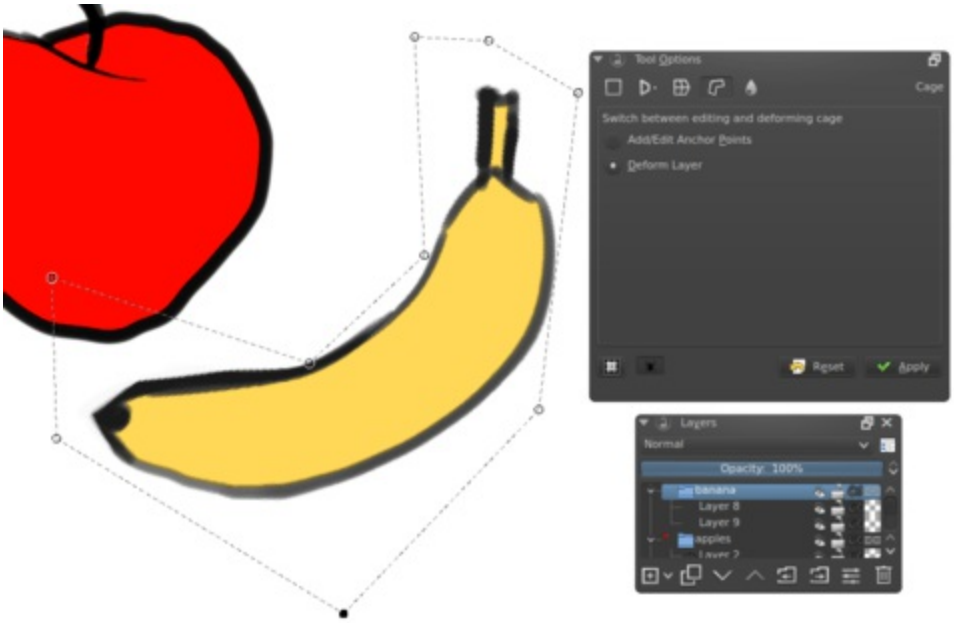
You can divide these either by subdivision or drawing custom points.

Subdivision
: This allows you to subdivide the selected area into a grid.

Draw
: Draw the anchor points yourself. Locking the points will put you in transform mode. Unlocking the points back into edit mode.

Cage

Create a cage around an image, and when it’s closed, you can use it to deform the image. If you have at least 3 points on the canvas, you can choose to switch between deforming and editing the existing points.



Transforming a straight banana to be curved with the cage tool.

Adjust Granularity

New in version 4.2.

This adjusts the precision of the cage transform grid. Lower precision gives more speed but also gives less precise results.

Preview
: Adjusts the granularity of the preview. It is recommended to have this lower than the *Real* value, as it speeds up adjusting.

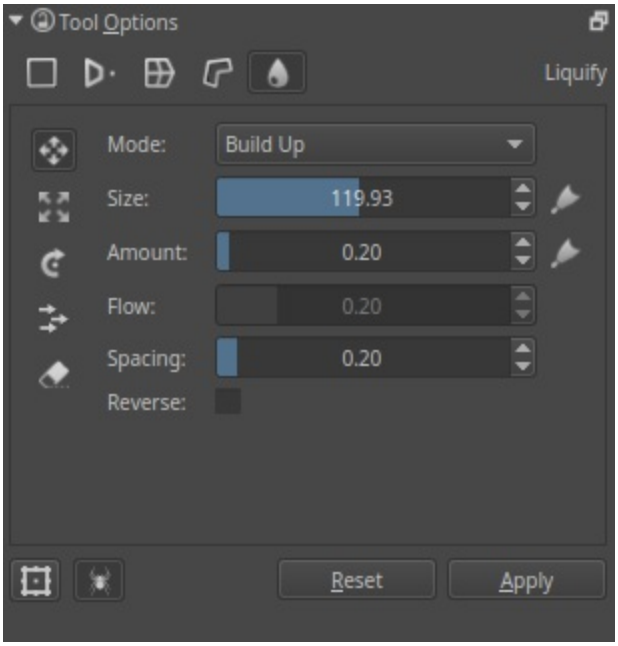
Real
: Adjusts the granularity of the final result.

Hotkeys

Both Cage and Warp use little nodes. These nodes can be selected and deselected together by pressing the `Ctrl` key before clicking nodes.

Then you can move them by pressing the cursor inside the bounding box. Rotating is done by pressing and dragging the cursor outside the bounding box and scaling the same, only one presses the `Ctrl` key before doing the motion.

Liquify



Like our deform brush, the liquify brush allows you to draw the deformations straight on the canvas.

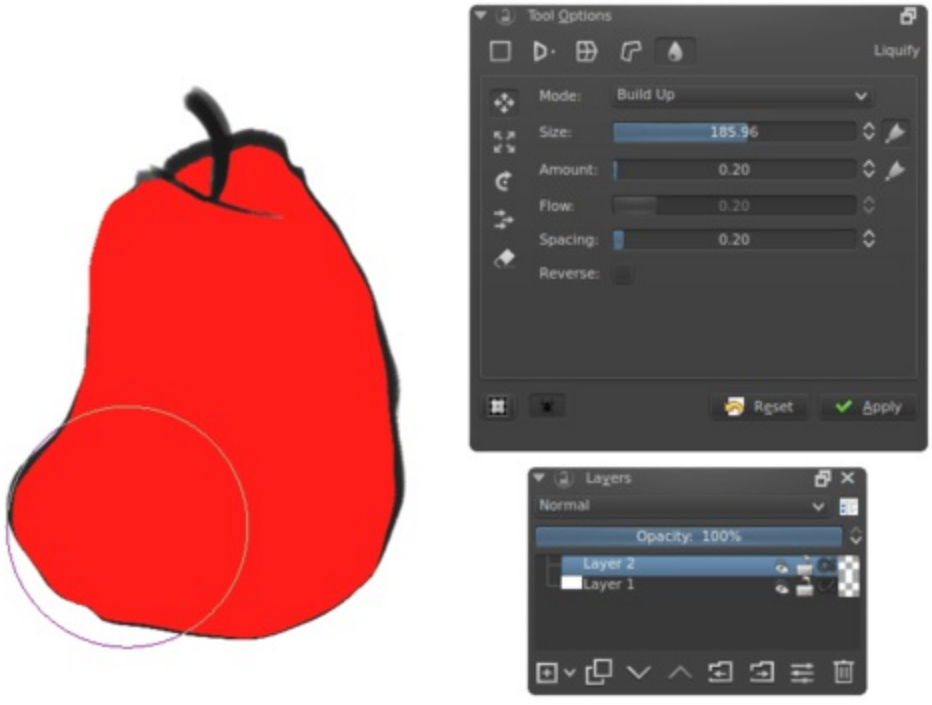
Move
: Drag the image along the brush stroke.

Scale
: Grow/Shrink the image under the cursor.

Rotate
: Twirl the image under the cursor.

Offset
: Shift the image under the cursor.

Undo
: Erases the actions of other tools.



Liquify used to turn an apple into a pear.

In the options for each brush there are:

Mode
: This is either *Wash* or *Build up*. *Wash* will normalize the effect to be between none, and the amount parameter as maximum. *Build up* will just add on until it’s impossible.

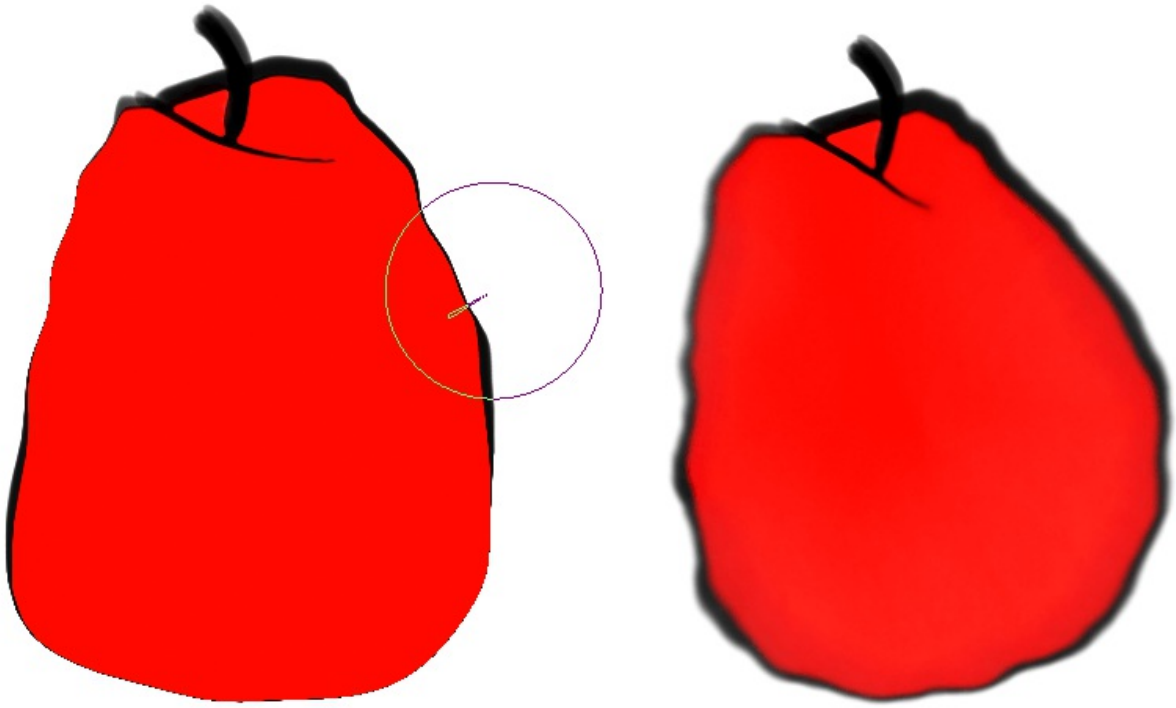
Size
: The brush size. The button to the right allows you to let it scale with pressure.

Amount
: The strength of the brush. The button to the right lets it scale with tablet pressure.

Flow
: Only applicable with *Build up*.

Spacing
: The spacing of the liquify dabs.

Reverse
: Reverses the action, so grow becomes shrink, rotate results in clockwise becoming counter-clockwise.



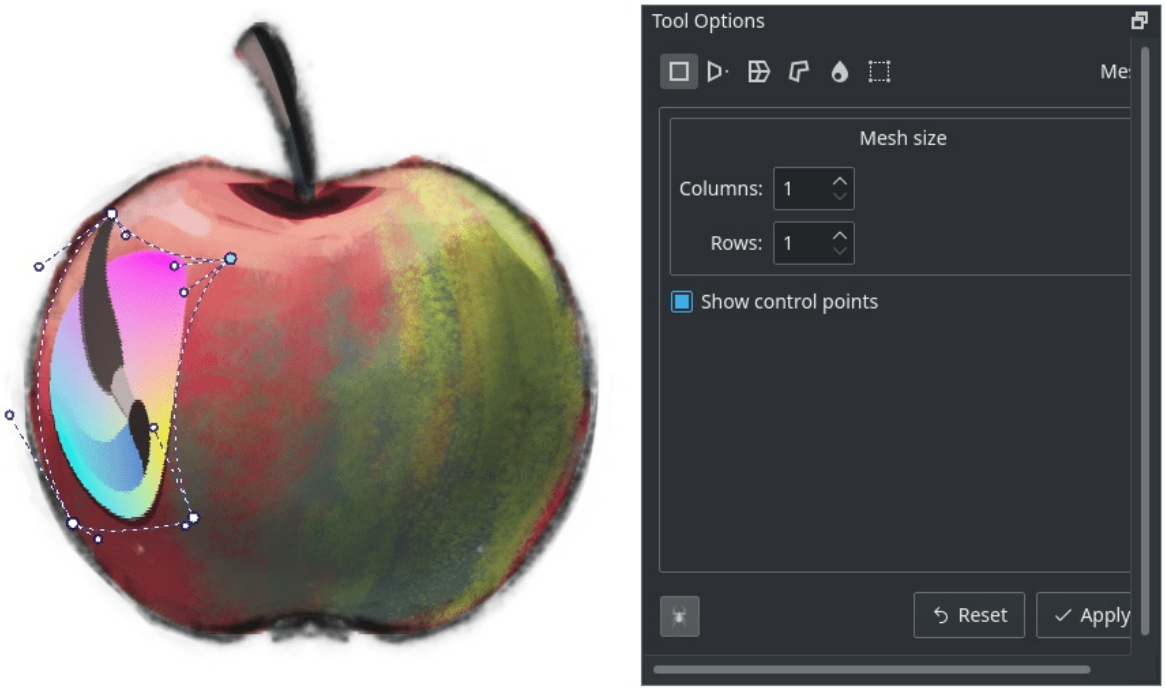
Liquify on the left and deform brush on the right.

Krita also has a Deform Brush Engine which is much faster than liquify, but has less quality. If you are attempting to make liquify a little faster, note that it speeds up with the less information it needs to process, so working with liquify within a selection or using liquify on a separate layer with little on it will greatly enhance the speed.

Mesh

New in version 4.4.2.

The mesh transform is similar to the warp and the cage transform, except that its interface uses patches comprised of Bezier curve segments. This transform mode is particularly useful for placing images and textures on curved surfaces:



Curving a logo to an apple with the mesh transform, with the control points shown.

This is a very keyboard shortcut heavy transform mode. When you start the transform you will see the overlay, which consists of several nodes that can be dragged around. You can drag on the segments between the nodes to curve them precisely, or drag on patches themselves to freely transform them. `Ctrl` + `Alt` +  + drag on nodes and segments will allow you to subdivide the mesh. For more precision, enable the control points in the tool options, so each Bezier segment can be fine-tuned to your content.

To assist in maintaining the curvature of a mesh, this tool has a concept of ‘locked’ transform. This mode is signalled by the lock icon in the cursor, and on by default. When this is enabled, adjusting one segment will also adjust its neighbouring segment in another patch. You can press `Shift` while dragging a segment or control point to turn this feature off, allowing for sharp angles in the mesh. After a sharp angle has been created, the locked mode will try to keep this as well.

Shortcuts

Node or control point move

 + drag any of the round points. The big ones are the ‘nodes’ which determine the corners of a patch, and the small ones are the ‘control points’, which determine the curvature for their associated segment.

Unlocked node or control point move

`Shift` +  + drag on a node or control point.

Locked segment move:

 + drag on a segment. As explained above, this will adjust neighbouring segments as well, to keep the curvature of the node intact.

Segment move

`Shift` +  + drag on a segment of the mesh.

Free patch deform:

 + drag on an empty area inside the mesh. This will allow you to intuitively adjust a segment by just clicking anywhere and dragging. The whole segment will then adjust all its control points around the point of the cursor.

Split mesh or Move/Delete split:

- `Ctrl` + `Alt` +  + drag on a border segment to split the mesh
- `Ctrl` + `Alt` +  + drag on a node to change the split
- `Ctrl` + `Alt` +  + drag away a node to remove the split

Select multiple nodes

`Ctrl` +  on a node or control or segment, these can then be moved

Move selection or mesh

`Shift` +  + drag on empty area outside the mesh.

Rotate selection or mesh

 + drag on an empty area outside the mesh, if there is a selection of nodes, it will rotate only them, otherwise the whole mesh will be rotated.

Scale selection or mesh

`Ctrl` +  + drag on empty area outside the mesh, if there is a selection of nodes, it will scale only them, otherwise the whole mesh will be scaled.

Tool options

Mesh Size

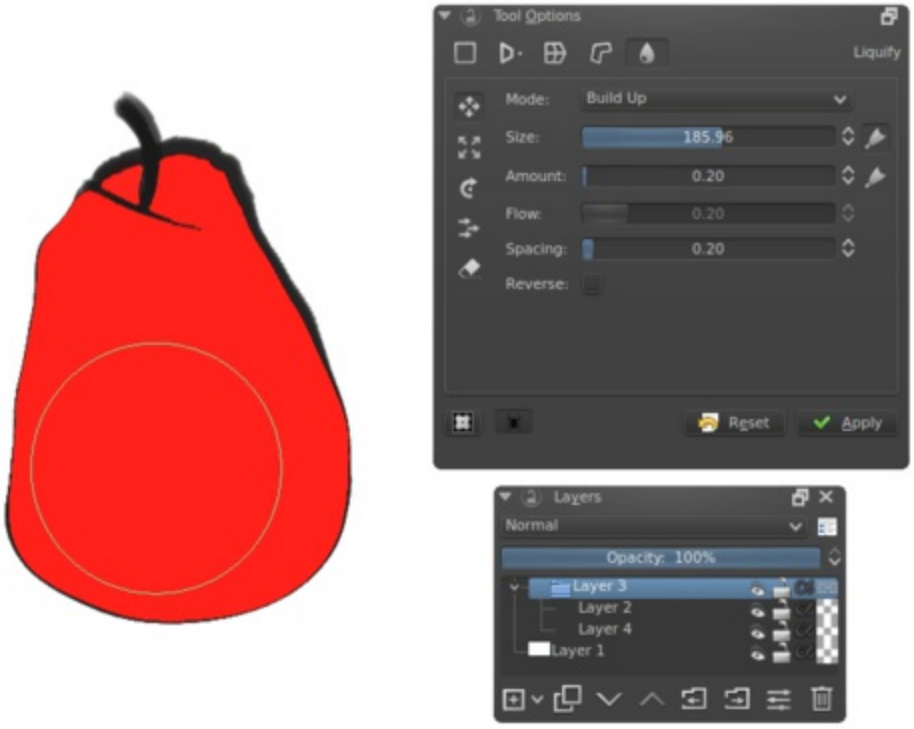
Gives precise controls to change the amount of patches vertically and horizontally. When increasing or decreasing the amount of patches, Krita will try to keep the curvature the same, which can be used to your advantage.

Show control points

This will toggle the control points.

Recursive Transform

The little spider icon on the lower-left of the transform tool options is the *Recursive Transform*.



Recursive transform transforms all the layers in the group, so with this apple, both the lineart as the fill will be transformed.

Recursive transform, when toggled, allows you to mass-transform all the layers in a group when only transforming the group.

Continuous Transform

If you apply a transformation, and try to start a new one directly afterwards, Krita will attempt to recall the previous transform, so you can continue it. This is a *continuous transform*. You can press the `Esc` key to cancel this and start a new transform, or press *Reset* in the tool options while no transform is active.

Transformation Masks

These allow you to make non-destructive transforms, check here for more info.

Fill Tool

Krita has one of the most powerful and capable Fill functions available. The options found in the Tool Options docker and outlined below will give you a great deal of flexibility working with layers and selections.

To get started, clicking anywhere on screen with the fill-tool will allow that area to be filed with the foreground color.

Tool Options

Fast Mode
: This is a special mode for really fast filling. However, many functions don’t work with this mode.

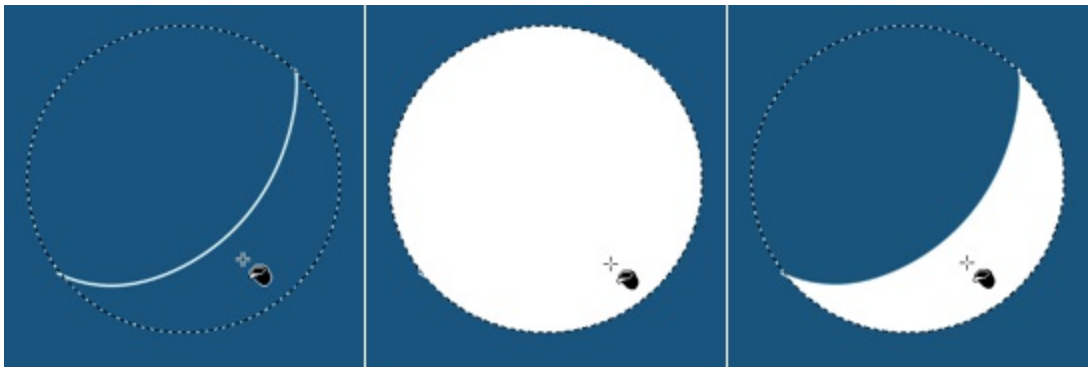
Threshold
: Determines when the fill-tool sees another color as a border.

Grow Selection
: This value extends the shape beyond its initial size.

Feathering Radius
: This value will add a soft border to the filled-shape.

Fill Entire Selection
: Activating this will result in the shape filling the whole of the active selection, regardless of threshold.

  New in version 4.4:
  Use selection as boundary
  : When checked, this will count the borders of the selection as an extra boundary on top of the pixel difference.



Left: Original selection with a line. Middle: Filled with ‘use selection as boundary’ off. Right: Filled with ‘use selection as boundary’ on.

Use Pattern
: Ticking this will result in the active pattern being used.

Rotation
: *New in version 4.4.*

  This allows you to rotate the pattern used in the fill.

Scale
: *New in version 4.4.*

  This allows you to scale the pattern used in the fill.

Sample
: *New in version 4.3.*

  Select which layers to use as a reference for the fill tool. The options are:

  Current Layer
  : Only use the currently selected layer.

  All layers
  : Use all visible layers.

  Color Labeled Layers
  : Use only the layers specified with a certain color label. This is useful for complex images, where you might have multiple lineart layers.

Label them with the appropriate color label and use these labels to mark which layers to use as a reference.

Labels Used

New in version 4.3.

Used with the ‘Color Labeled Layers’ option above.

Gradient Tool

The Gradient tool is found in the Tools Panel. Left-Click dragging this tool over the active portion of the canvas will draw out the current gradient. If there is an active selection then, similar to the Fill Tool, the paint action will be confined to the selection's borders.

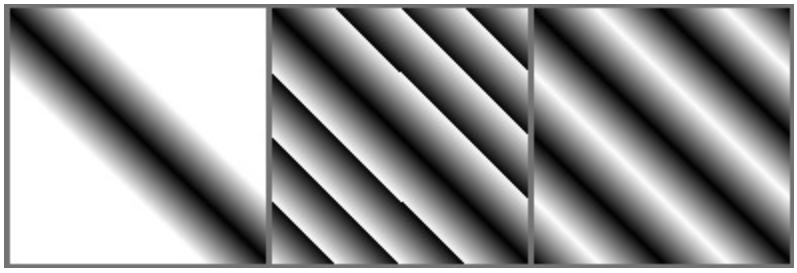
Tool Options

Shape:

- Linear
  - This will draw a straight gradient.

Left: **None**. Middle: **Forwards**. Right: **Alternating**.

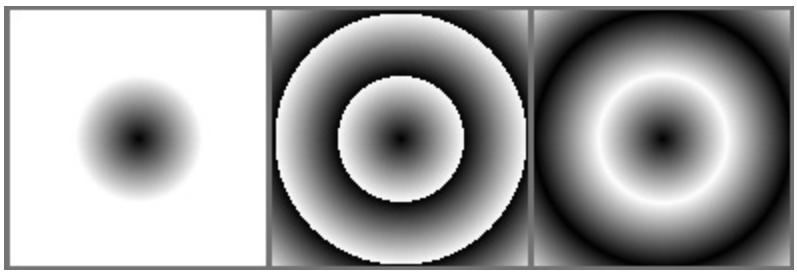
- Bilinear
  - This will draw a straight gradient, mirrored along the axis.



Left: **None**. Middle: **Forwards**. Right: **Alternating**.

- Radial

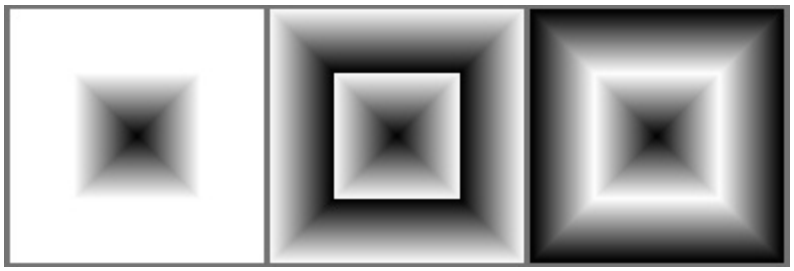
This will draw the gradient from a center, defined by where you start the stroke.



Left: **None**. Middle: **Forwards**. Right: **Alternating**.

Square

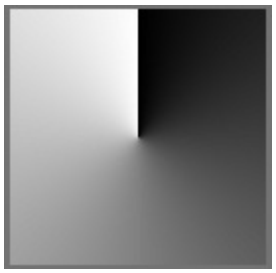
This will draw the gradient from a center in a square shape, defined by where you start the stroke.



Left: **None**. Middle: **Forwards**. Right: **Alternating**.

Conical

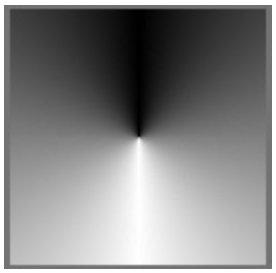
This will wrap the gradient around a center, defined by where you start the stroke.



Left: **None**. Middle: **Forwards**. Right: **Alternating**.

Conical-symmetric

This will wrap the gradient around a center, defined by where you start the stroke, but will mirror the wrap once.



Left: **None**. Middle: **Forwards**. Right: **Alternating**.

Spiral

This will draw the gradient spiral from a center, defined by where you start the stroke.

Left: **None**. Middle: **Forwards**. Right: **Alternating**.

Reverse Spiral

This will draw the gradient spiral from a center, defined by where you start the stroke, but direction is flipped perpendicular to the direction of stroke.

Left: **None**. Middle: **Forwards**. Right: **Alternating**.

Shaped

This will shape the gradient depending on the selection or layer.

Repeat:

None

This will extend the gradient into infinity.

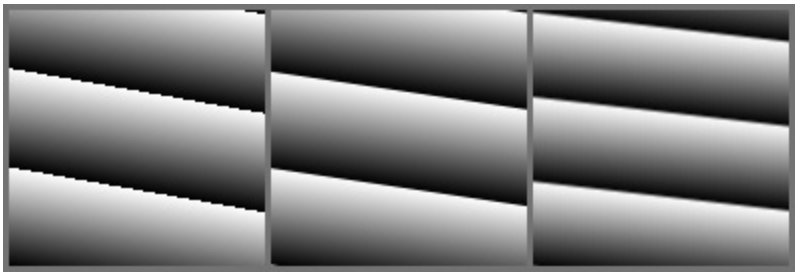
Forward
: This will repeat the gradient into one direction.

Alternating
: This will repeat the gradient, alternating the normal direction and the reversed.

Reverse
: Reverses the direction of the gradient.

Antialias threshold
: Controls how smooth is the border between repetitions.

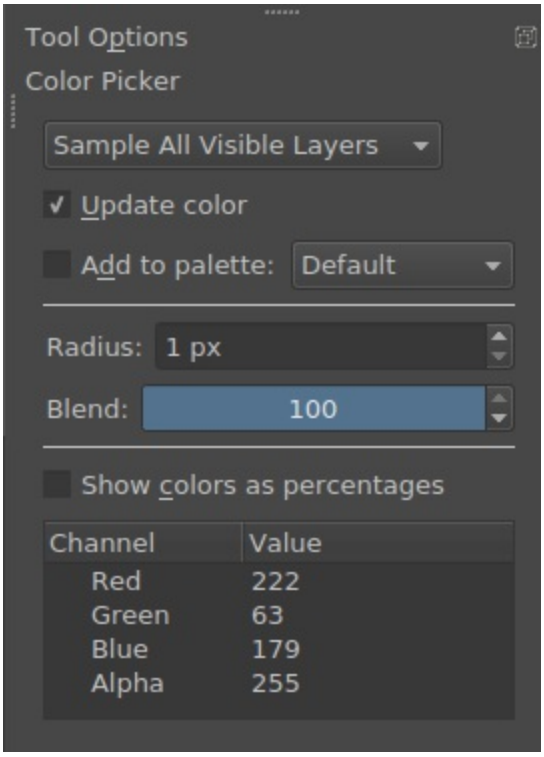
- A value equal to 0 means there is no smoothing. The border is aliased.
- A value greater than 0 tells Krita how many pixels to each side of the border should be smoothed.



Left: **0**. Middle: **0.5**. Right: **1**.

Color Selector Tool

This tool allows you to choose a point from the canvas and make the color of that point the active foreground color. When a painting or drawing tool is selected the Color Picker tool can also be quickly accessed by pressing the `Ctrl` key.



There are several options shown in the *Tool Options* docker when the *Color Picker* tool is active:

The first drop-down box allows you to select whether you want to sample from all visible layers or only the active layer. You can choose to have your selection update the current foreground color, to be added into a color palette, or to do both.

New in version 4.1: The middle section contains a few properties that change how the Color Picker picks up color; you can set a *Radius*, which will average the colors in the area around the cursor, and you can now also set a

Blend percentage, which controls how much color is “soaked up” and mixed in with your current color. Read Mixing Colors for information about how the Color Picker’s blend option can be used as a tool for off-canvas color mixing.

At the very bottom is the Info Box, which displays per-channel data about your most recently picked color. Color data can be shown as 8-bit numbers or percentages.

Colorize Mask

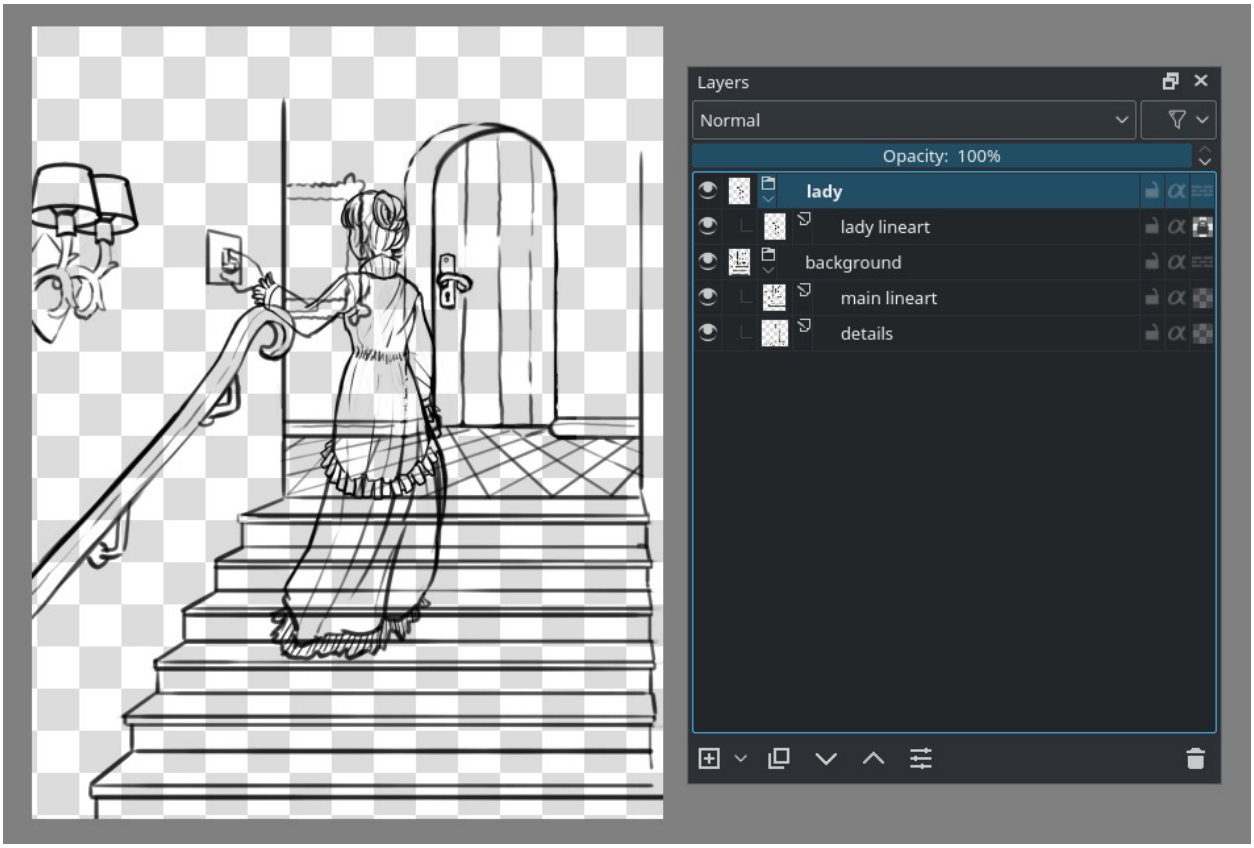
A tool for quickly coloring line art, the Colorize Mask Editing tool can be found next to the gradient tool on your toolbox.

This feature is technically already in 3.1, but disabled by default because we had not optimized the filling algorithm for production use yet. To enable it, find your krita configuration file, open it in notepad, and add “disableColorizeMaskFeature=false” to the top. Then restart Krita. Its official incarnation is in 4.0.

Usage

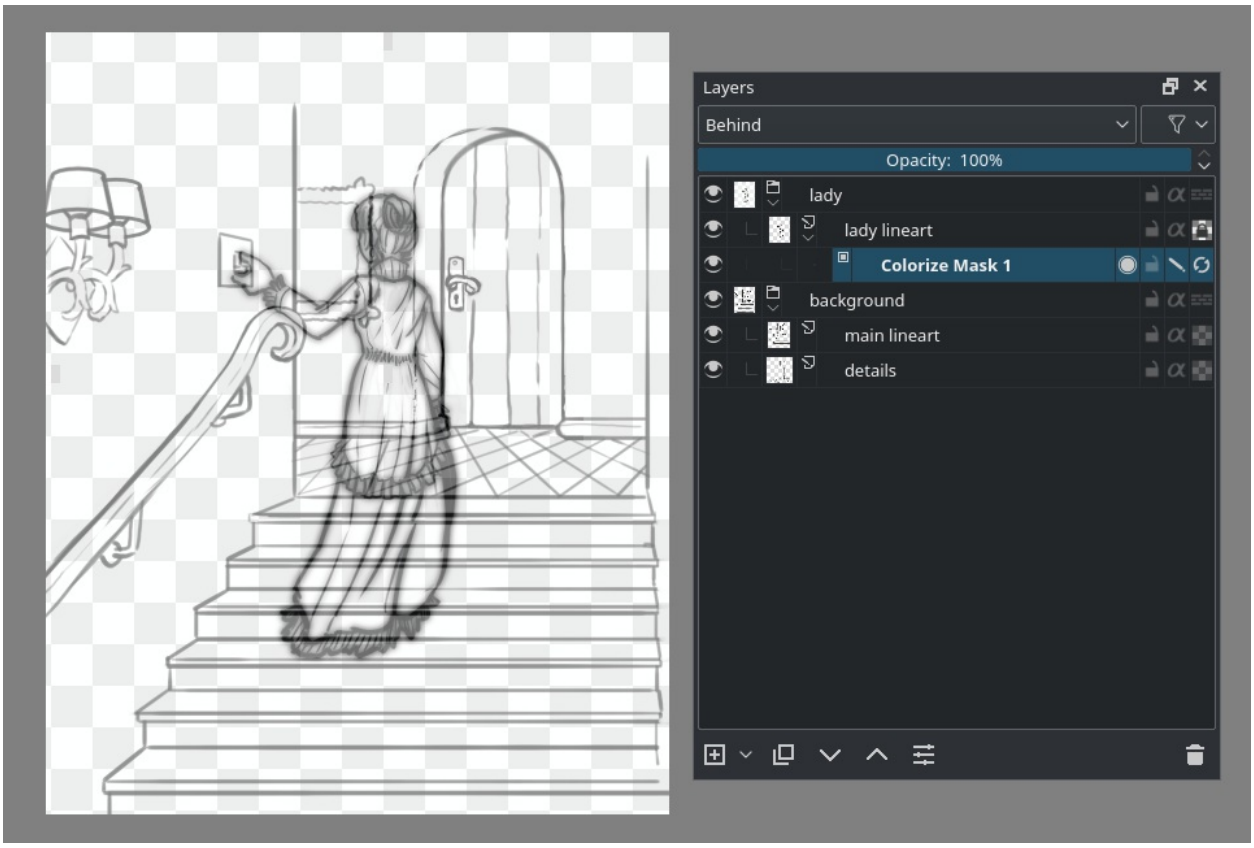
This tool works in conjunction with the colorize mask, and the usage is as follows:

For this example, we’ll be using the ghost lady also used to explain masks on the basic concepts page.

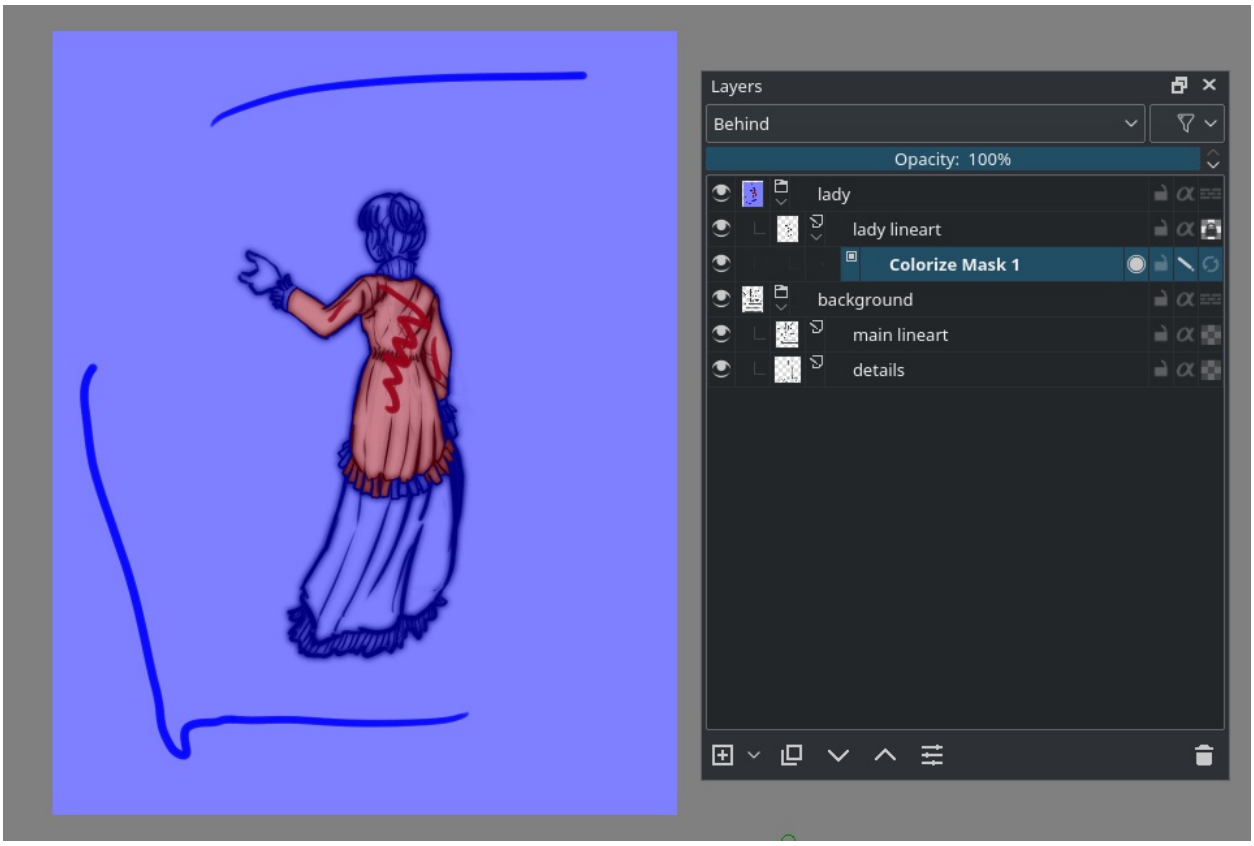


This image has the line art for the lady separated from the background, and what's more, the background is made up of two layers: one main and one for the details.

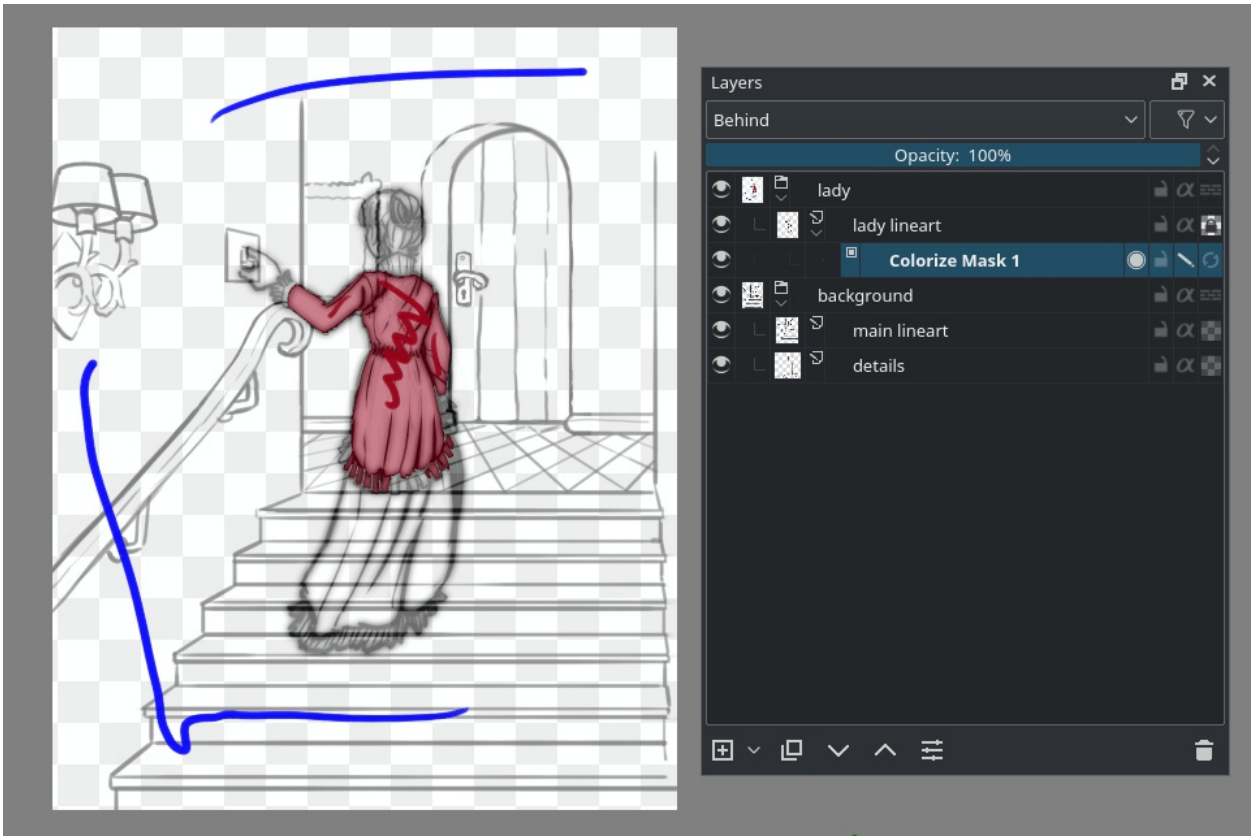
First, select the colorize mask editing tool while having the line art layer selected.  the canvas will add a colorize mask to the layer. You can also  the line art layer, and then *Add* ▸ *Colorize Mask*. The line art will suddenly become really weird, this is the prefiltering which are filters through which we put the line art to make the algorithm easier to use. The tool options overview below shows which options control that.



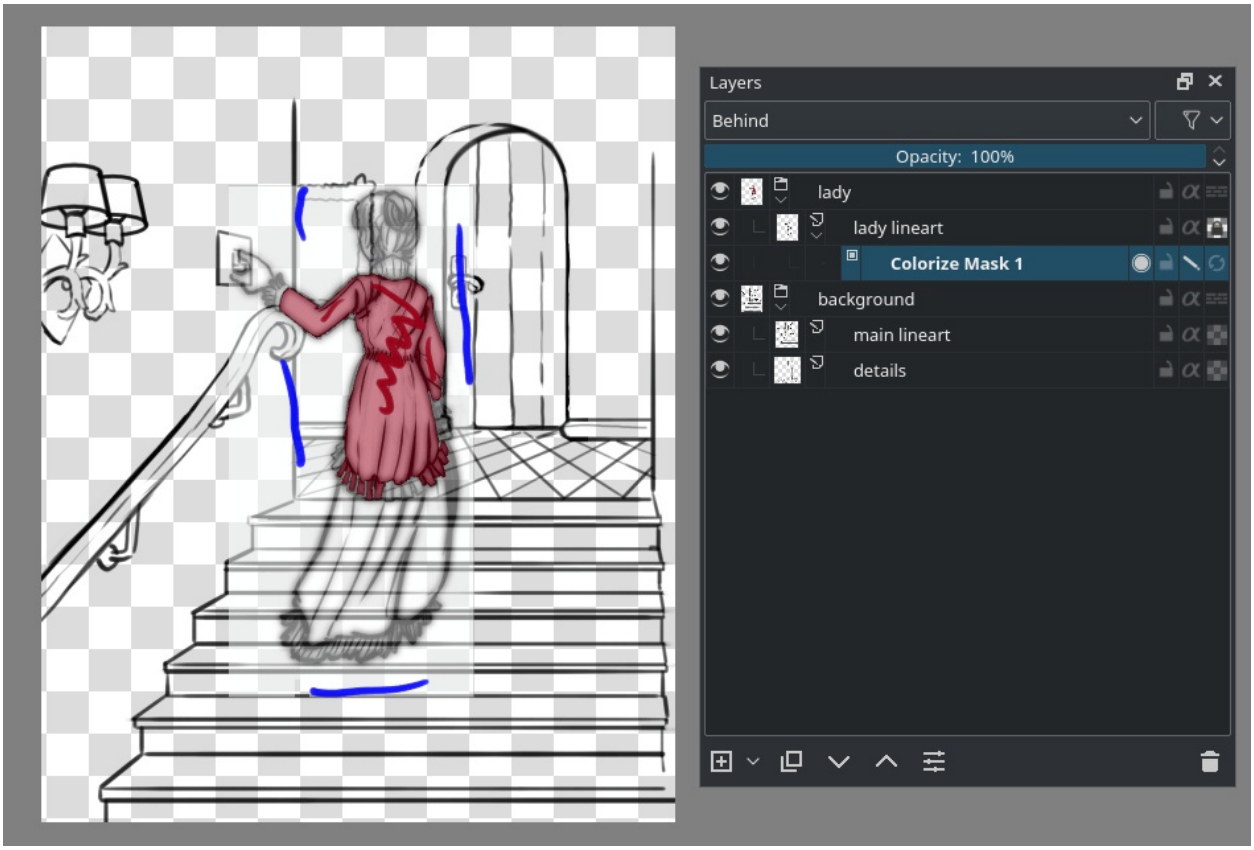
Now, you make strokes with brush colors, press *Update* in the tool options, or tick the last icon of the colorize mask properties. In the layer docker, you will be able to see a little progress bar appear on the colorize mask indicating how long it takes. The bigger your file, the longer it will take.



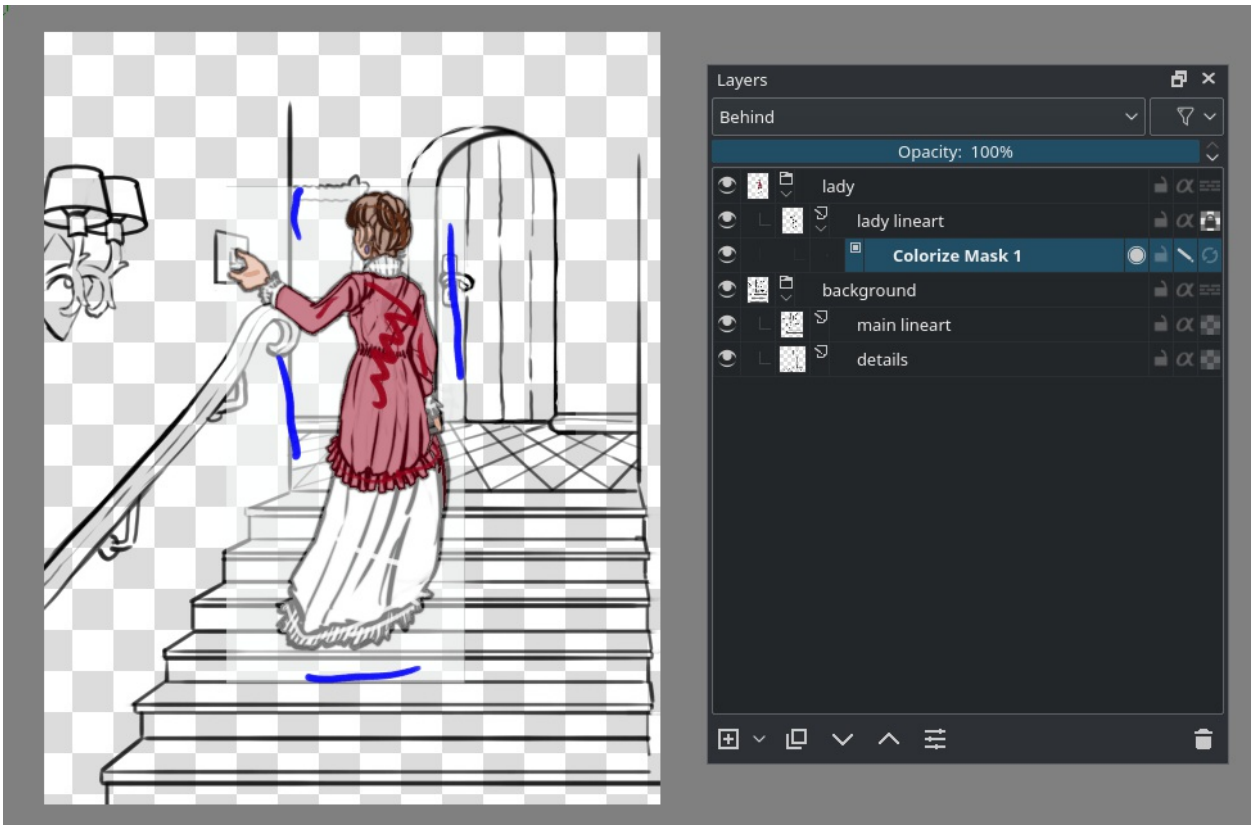
We want to have the blue transparent. In the tool options of the colorize editing tool you will see a small palette. These are the colors already used. You can remove colors here, or mark a single color as standing for transparent, by selecting it and pressing “transparent”. Updating the mask will still show the blue stroke, but the result will be transparent:



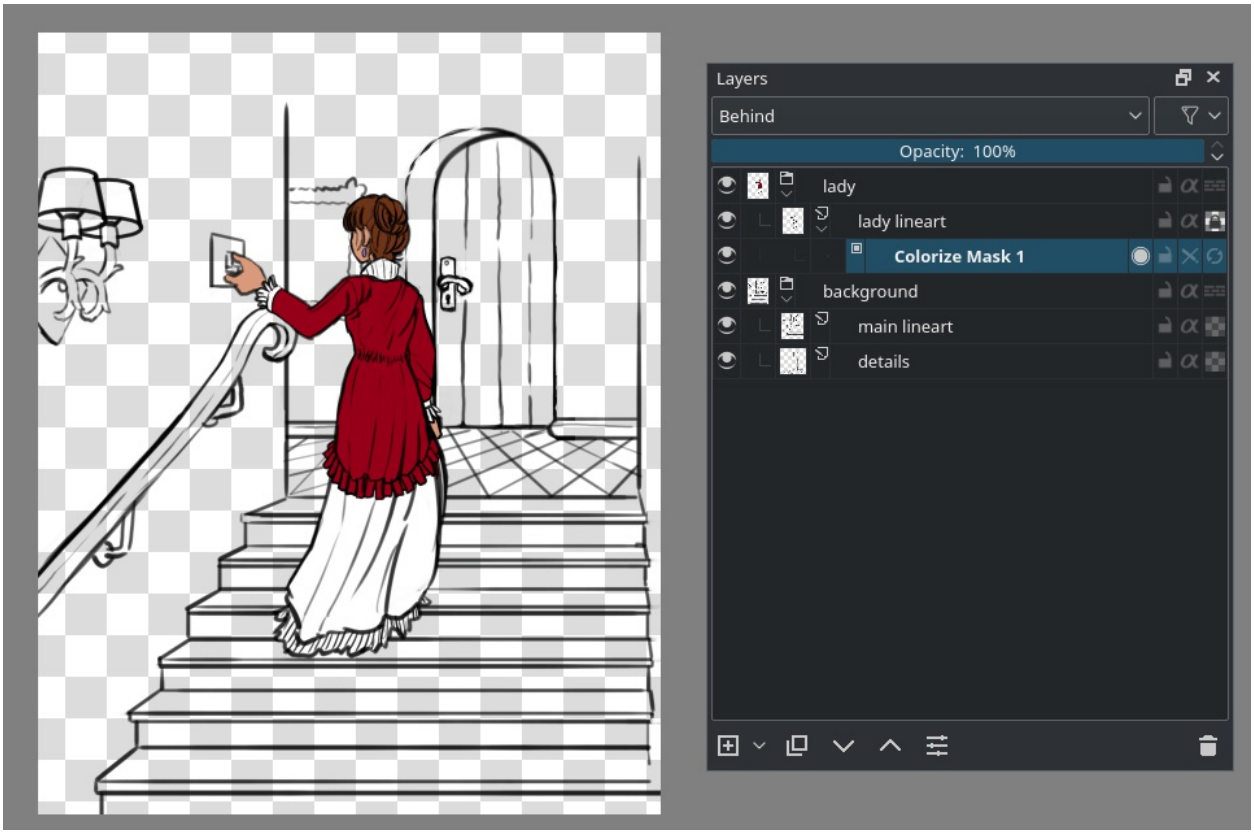
Because the colorize mask algorithm is slow, and we only need a part of our layer to be filled to fill the whole ghost lady figure, we can make use of *Limit to layer bounds*. This will limit Colorize Mask to use the combined size of the line art and the coloring key strokes. Therefore, make sure that the colorizing keystrokes only take up as much as they really need.



Now the algorithm will be possibly a lot faster, allowing us to add strokes and press *Update* in rapid succession:



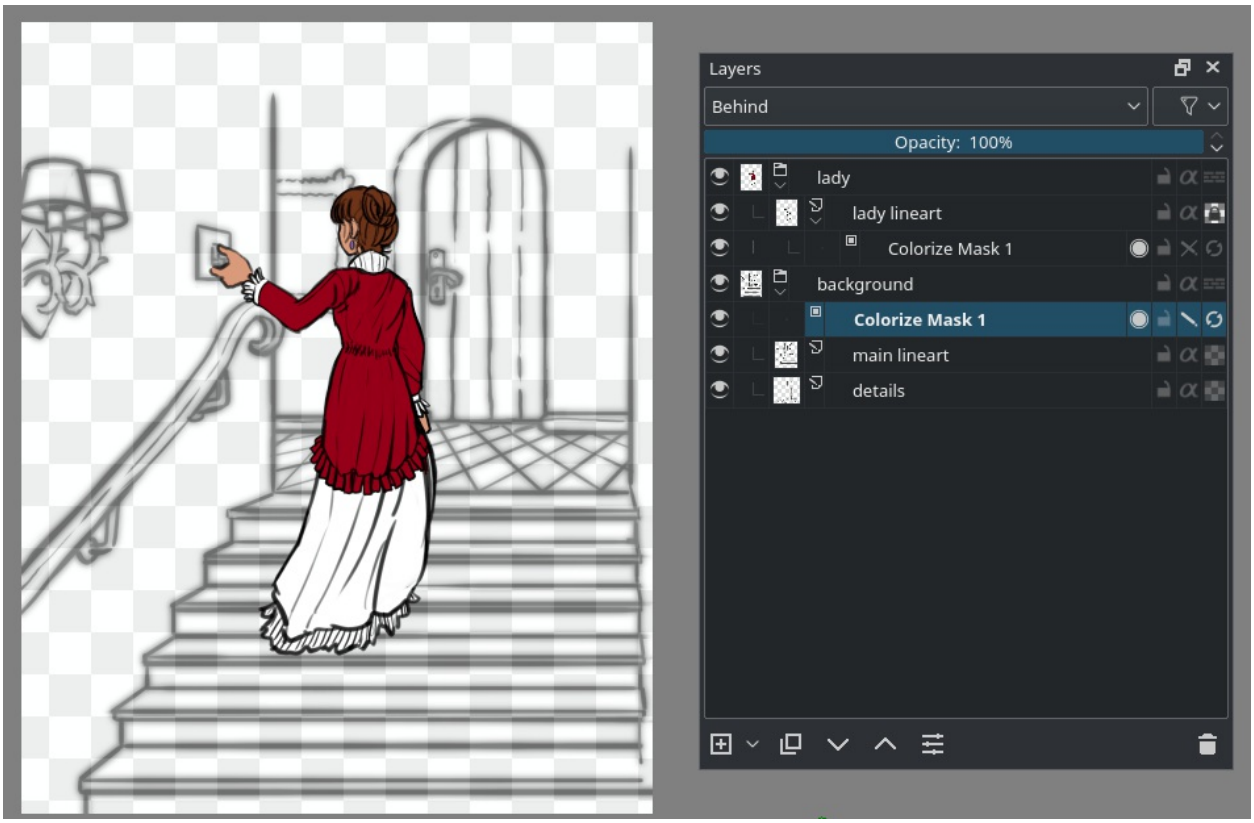
To see the final result, disable *Edit Key Strokes* or toggle the second to last icon on the colorize mask.



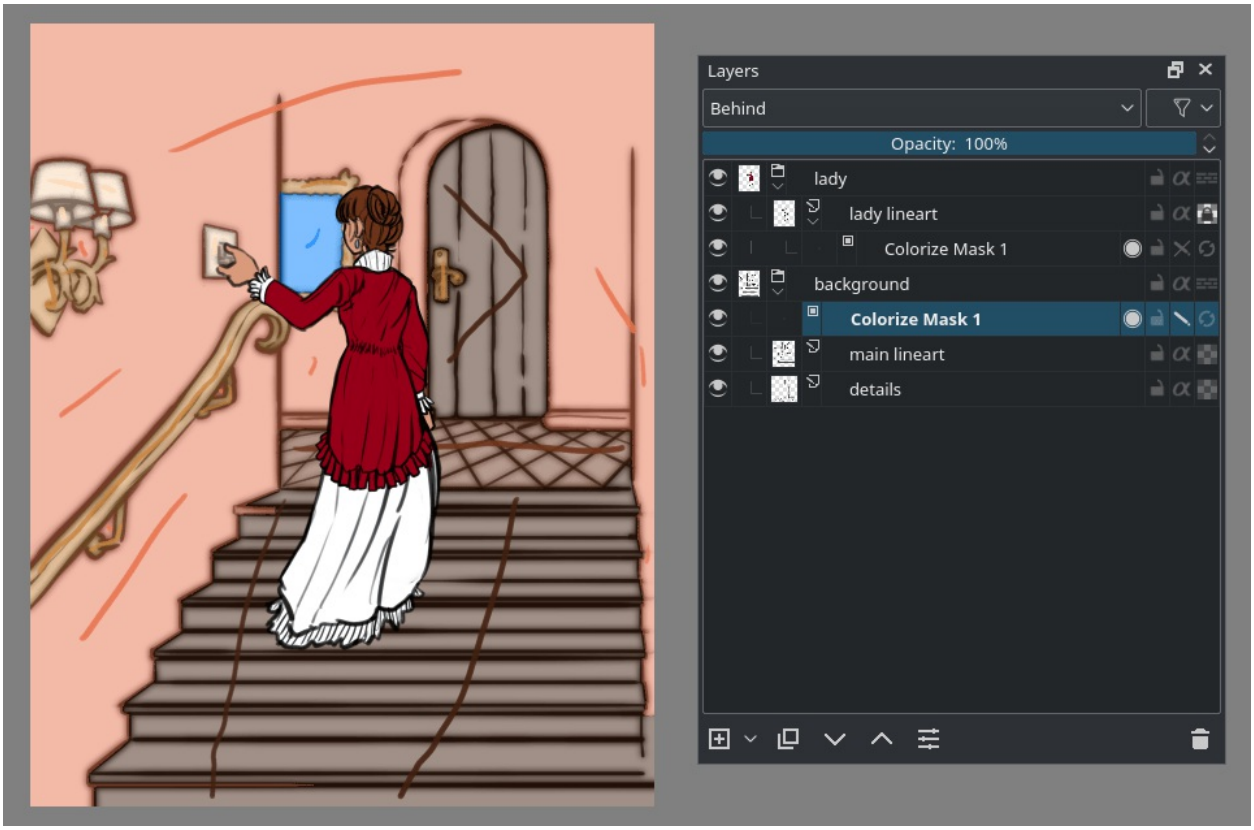
If you want to edit the strokes again, re-enable *Edit Key Strokes*.

Now, the colorize mask, being a mask, can also be added to a group of line art layers. It will then use the composition of the whole group as the line art. This is perfect for our background which has two separate line art layers. It also means that the colorize mask will be disabled when added to a group with pass-through enabled, because those have no final composition. You can recognize a disabled colorize mask because its name is stricken through.

To add a colorize mask to a group, select the group and  the canvas with the Colorize Mask editing tool, or  the layer to *Add* ▸ *Colorize Mask*.



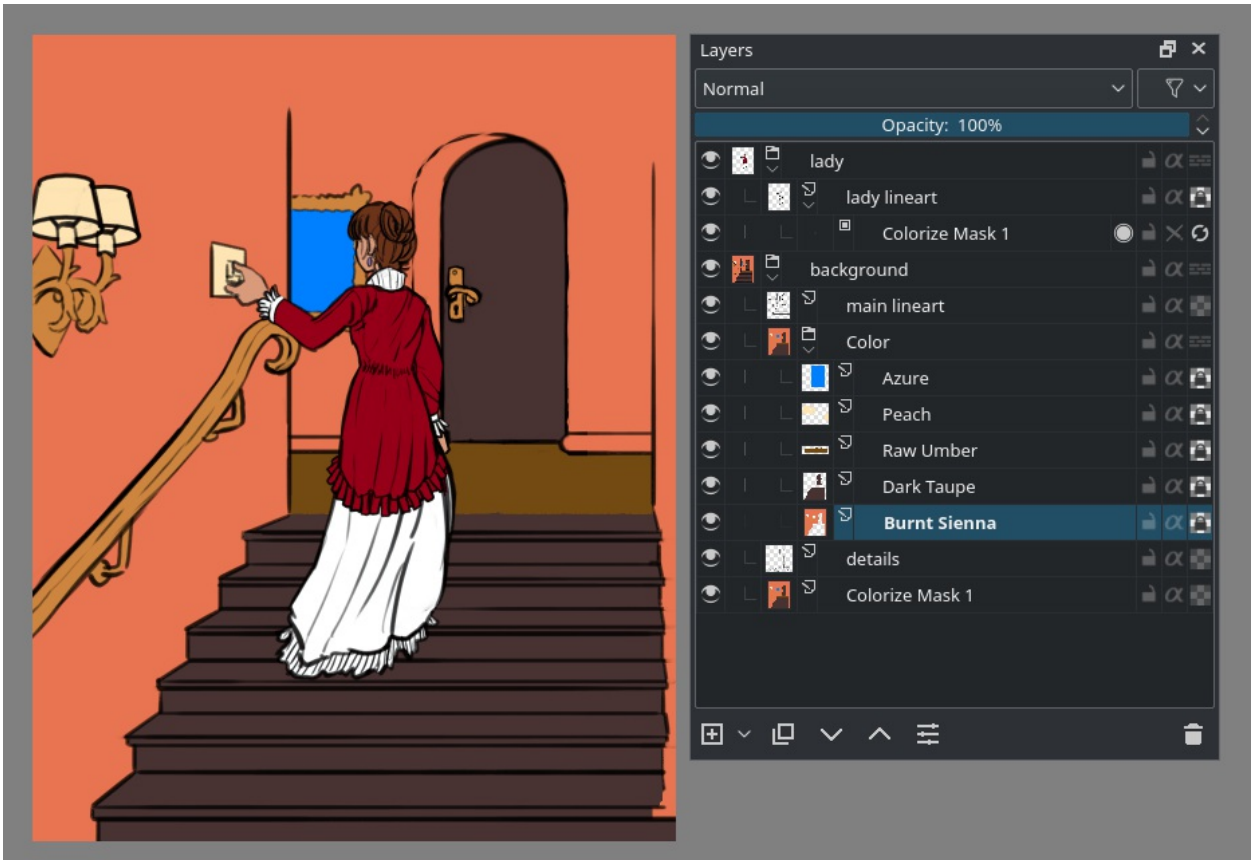
Now, we add strokes to the background quickly. We do not need to use the *Limit to Layer Bounds* because the background covers the whole image.



For the record, you can use other brushes and tools also work on the colorize mask as long as they can draw. The Colorize Mask Editing tool is just the most convenient because you can get to the algorithm options.

Out final result looks like this:

Now we are done,  the colorize mask and *Convert ▸ to Paint Layer*. Then, *Layer ▸ Split ▸ Split Layer*. This will give separate color islands that you can easily edit:



This way we can very quickly paint the image. Due to the colorize mask going from the first image to the following took only 30 minutes, and would’ve taken quite a bit longer.

The colorize masks are saved to the `.kra` file, so as long as you don't save and open to a different file format, nor convert the colorize mask to a paintlayer, you can keep working adjust the results.

Tool Options

Update
: Run the colorize mask algorithm. The progress bar for updates on a colorize mask shows only in the layer docker.

Edit key strokes
: Put the mask into edit mode. In edit mode, it will also show the 'prefiltering' on the line art, which is for example a blur filter for gap closing.

Show output

Show the output of the colorize mask. If *Edit key strokes* is active, this will be shown semi-transparently, so it will be easy to recognize the difference between the strokes and the output.

On the **Left**: *Show Output* is on, *Edit Key Strokes* is off. In the **Middle**: *Show Output* and *Edit Key Strokes* are on. On the **Right**: *Show Output* is off and *Edit Key Strokes* is on.

Limit to layer bounds

Limit the colorize mask to the combined layer bounds of the strokes and the line art it is filling. This can speed up the use of the mask on complicated compositions, such as comic pages.

Edge detection

Activate this for line art with large solid areas, for example shadows on an object. For the best use, set the value to the thinnest lines on the image. In the image below, note how edge detection affects the big black areas:

From left to right: an example with big black shadows on an object but no edge detection, the same example without the edit key strokes enabled.

Then the same example with edge detection enabled and set to 2px, and that same example without edit key strokes enabled.

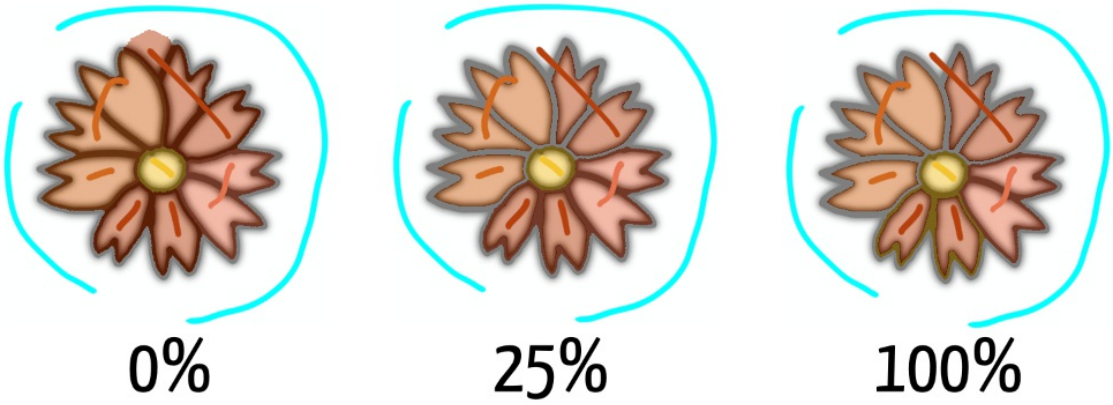
Gap close hint

While the algorithm is pretty good against gaps in contours, this will improve the gap recognition. The higher this value is, the bigger the gaps it will try to close, but a too high value can lead to other errors. Note how the prefiltered line art (that's the blurry haze) affects the color patches.

On the **Left**: *Gap close hint* is 0px. In the **Middle**: *Gap close hint* is 15px (the lines are 10px). On the **Right**: *Gap close hint* is 275px.

Clean up

This will attempt to handle messy strokes that overlap the line art where they shouldn't. At 0 no clean up is done, at 100% the clean-up is most aggressive.



Key strokes

This palette keeps track of the colors used by the strokes. This is useful so you can switch back to colors easily. You can increase the swatch size by hovering over it with the mouse, and doing `Ctrl +`  .

Transparent
: This button is under the keystrokes palette, you can mark the selected color to be interpreted a ‘transparent’ with this. In the clean-up screenshot above, cyan had been marked as transparent.

Layer properties

The colorize mask layer has four properties. They are all the buttons on the right side of the colorize mask layer:

Show output
:  The show output icon allows you to toggle whether you’ll see the output from the colorize algorithm.

Lock
:  This icon stops the mask from being edited.

Edit key strokes
:  This icon shows whether the colorize mask is in edit mode. In edit mode it’ll show the strokes, and the output will be semi-transparent.

Update
:  This icon will force the colorize mask to update, even when you’re in a different tool.

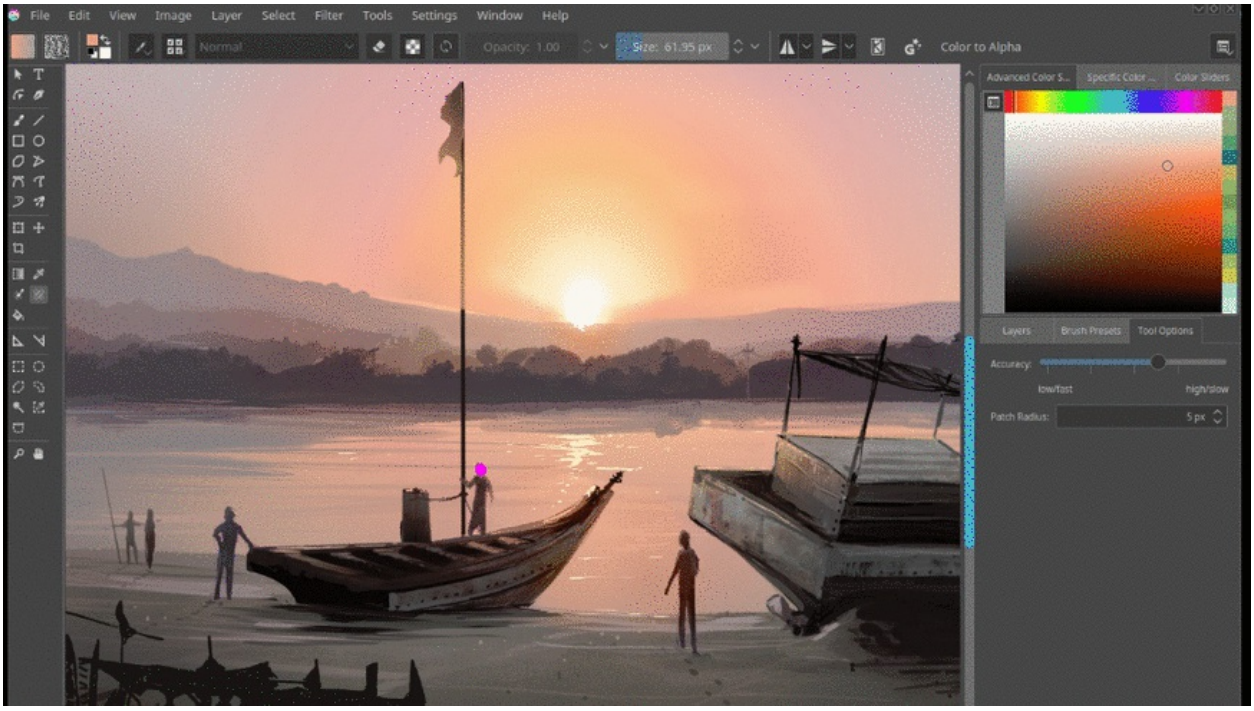
Note

Colorize masks cannot be animated.

Smart Patch Tool

The smart patch tool allows you to seamlessly remove elements from the image. It does this by letting you draw the area which has the element you wish to remove, and then it will attempt to use patterns already existing in the image to fill the blank.

You can see it as a smarter version of the clone brush.



The smart patch tool has the following tool options:

Accuracy

Accuracy indicates how many samples, and thus how often the algorithm is run. A low accuracy will do few samples, but will also run the algorithm fewer times, making it faster. Higher accuracy will do many samples, making the algorithm run more often and give more precise results, but because it has

to do more work, it is slower.

Patch size

Patch size determines how big the size of the pattern to choose is. This will be best explained with some testing, but if the surrounding image has mostly small elements, like branches, a small patch size will give better results, while a big patch size will be better for images with big elements, so they get reused as a whole.

Assistant Tool

Create, edit, and remove drawing assistants on the canvas. There are a number of different assistants that can be used from this tool. The tool options allow you to add new assistants, and to save/load assistants. To add a new assistant, select a type from the tool options and begin clicking on the canvas. Each assistant is created a bit differently. There are also additional controls on existing assistants that allow you to move and delete them.

The set of assistants on the current canvas can be saved to a "*.paintingassistant" file using the *Save* button in the tool options. These assistants can then be loaded onto a different canvas using the Open button. This functionality is also useful for creating copies of the same drawing assistant(s) on the current canvas.

Check Painting with Assistants for more information.

Tool Options

New in version 4.0.

Global Color:
: Global color allows you to set the color and opacity of all assistants at once.

New in version 4.1.

Custom Color:
: Custom color allows you to set a color and opacity per assistant, allowing for different colors on an assistant. To use this functionality, first 'select' an assistant by tapping its move widget. Then go to the tool options docker to see the *Custom Color* check box. Check that, and then use the opacity and color buttons to pick either for this particular assistant.

Reference Images Tool

New in version 4.1.

The reference images tool is a replacement for the reference images docker. You can use it to load images from your disk as reference, which can then be moved around freely on the canvas and placed wherever.

Tool Options

Add Reference Image
: Load a single image to display on the canvas.

Paste as Reference Image
: Load an image from the system clipboard and add it as a reference image.

Load Set
: Load a set of reference images.

Save Set
: Save a set of reference images.

Delete all reference images
: Delete all the reference images.

Keep aspect ratio
: When toggled this will force the image to not get distorted.

Opacity
: Lower the opacity.

Saturation

Desaturate the image. This is useful if you only want to focus on the light/shadow instead of getting distracted by the colors.

Storage mode
: How is the reference image stored.

  Embed to *.kra
  : Store this reference image into the KRA file. This is recommended for small vital files you'd easily lose track of otherwise.

  Link to external file.
  : Only link to the reference image, krita will open it from the disk everytime it loads this file. This is recommended for big files, or files that change a lot. This option is only available when reference images are loaded from a local path.

You can move around reference images by selecting them with  , and dragging them. You can rotate reference images by holding the cursor close to the outside of the corners till the rotate cursor appears, while tilting is done by holding the cursor close to the outside of the middle nodes. Resizing can be done by dragging the nodes. You can delete a single reference image by clicking it and pressing `Del`. You can select multiple reference images with `Shift` and perform all of these actions.

To hide all reference images temporarily use *View* ▸ *Show Reference Images*.

Measure Tool

This tool is used to measure distances and angles. Click the  to indicate the first endpoint or vertex of the angle, keep the button pressed, drag to the second endpoint and release the button. The results will be shown on the Tool Options docker. You can choose the length units from the drop-down list.

Tool Options

The measure tool-options allow you to change between the units used. Unit conversion varies depending on the DPI setting of a document.

Rectangular Selection Tool

This tool, represented by a rectangle with a dashed border, allows you to make Selections in a rectangular shape. To create a rectangular selection simply  and drag on the area of the canvas that you wish to select.

Important

Most of the behavior of the Rectangular Selection Tool is common to all other selection tools, please make sure to read Selections to learn more about this tool.

Hotkeys and Stickykeys

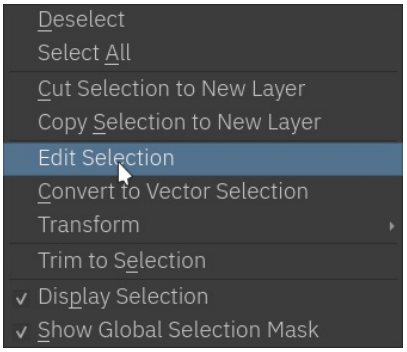
- `Ctrl + R` selects this tool.
- `R` sets the selection to ‘replace’ in the tool options, this is the default mode.
- `A` sets the selection to ‘add’ in the tool options.
- `S` sets the selection to ‘subtract’ in the tool options.
-  + `Shift` constrains the selection to a perfect square. (Make sure to press  before `Shift`)
-  + `Ctrl` makes the selection resize from the center. (Make sure to press  before `Ctrl`)
-  + `Alt` allows you to move the selection. (Make sure to press  before `Alt`)
- `Shift` +  sets the subsequent selection to ‘add’. You can release the

`Shift` key while dragging, but it will still be set to ‘add’. Same for the others.

- `Alt` +  sets the subsequent selection to ‘subtract’.
- `Ctrl` +  sets the subsequent selection to ‘replace’.
- `Shift + Alt` +  sets the subsequent selection to ‘intersect’.

New in version 4.2:

- Hovering your cursor over the dashed line of the selection, or marching ants as it is commonly called, turns the cursor into the move tool icon, which you  and drag to move the selection.
-  will open up a selection quick menu with amongst others the ability to edit the selection.



Hint

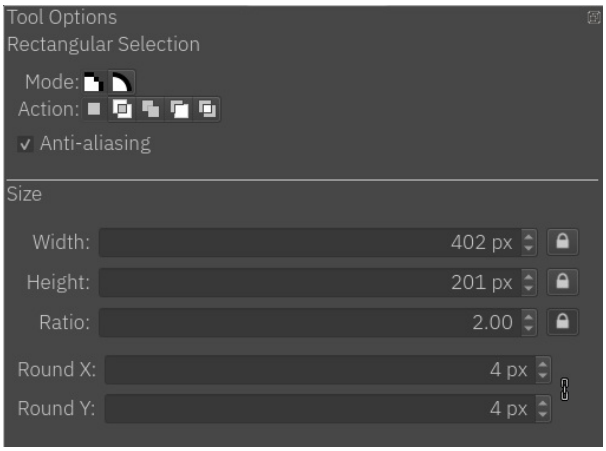
To subtract a perfect square, you can follow two different methods:

1. Press `S` to subtract then  to select and press `Shift` while dragging to constrain to a perfect square.
2. Press `Alt` +  , then release the `Alt` key while dragging and press `Shift` to constrain.

Tip

You can switch the behavior of the `Alt` key to use `Ctrl` instead by toggling the switch in the General Settings

Tool Options



Mode
: This option is explained in the Pixel and Vector Selection Types section.

Action
: This option is explained in the Pixel and Vector Selection Types section.

Anti-aliasing
: This toggles whether or not to give selections feathered edges. Some people prefer hard-jagged edges for their selections.

Note

Anti-aliasing is only available on Pixel Selection Mode.

Width
: Shows you the current width while you are creating the selection. You can manually type the value and use the ‘Lock Width’ for your next

selection to have the selected value.

Lock Width
Forces the next selection to have the current width.

Height
Shows you the current height while you are creating the selection. You can manually type the value and use the ‘Lock Height’ for your next selection to have the selected value.

Lock Height
Forces the next selection to have the current height.

Ratio
Shows the ratio between height and width of the selection. Similar to Height, and Width, you can manually type the value and use the ‘Lock Ratio’ for your next selection to have the selected value.

Lock Ratio
Forces the next selection to have the current ratio.

Hint

If you want your selection to be of specific size:

1. Type the width and height.
2. Press the Lock Width and Lock Height buttons.
3.  where you want your selection to be.

New in version 4.1.3:
Round X
The horizontal radius of the rectangle corners.

Round Y
The vertical radius of the rectangle corners.

Chain Link

When linked the aspect ratio between the roundness of X and Y coordinates will be locked. To disconnect the chain just click in the links and it will separate in two parts.

Elliptical Selection Tool

This tool, represented by an ellipse with a dashed border, allows you to make Selections of a elliptical area. Simply  and drag around the section you wish to select.

Important

Most of the behavior of the Elliptical Selection Tool is common to all other selection tools, please make sure to read Selections to learn more about this tool.

Hotkeys and Stickykeys

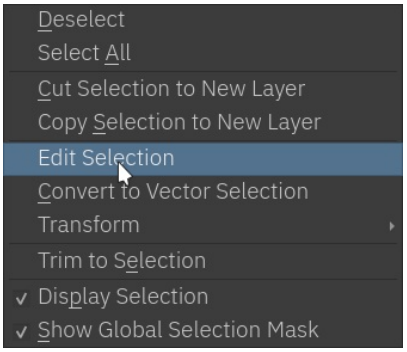
- `J` selects this tool.
- `R` sets the selection to 'replace' in the tool options, this is the default mode.
- `A` sets the selection to 'add' in the tool options.
- `S` sets the selection to 'subtract' in the tool options.
-  + `Shift` constrains the selection to a perfect circle. (Make sure to press  before `Shift`)
-  + `Ctrl` makes the selection resize from the center. (Make sure to press  before `Ctrl`)
-  + `Alt` allows you to move the selection. (Make sure to press  before `Alt`)
- `Shift` +  sets the subsequent selection to 'add'. You can release the

`Shift` key while dragging, but it will still be set to ‘add’. Same for the others.

- `Alt` + sets the subsequent selection to ‘subtract’.
- `Ctrl` + sets the subsequent selection to ‘replace’.
- `Shift + Alt` + sets the subsequent selection to ‘intersect’.

New in version 4.2:

- Hovering your cursor over the dashed line of the selection, or marching ants as it is commonly called, turns the cursor into the move tool icon, which you and drag to move the selection.
- will open up a selection quick menu with amongst others the ability to edit the selection.



Hint

To subtract a perfect circle, you can follow two different methods:

1. Press `S` to subtract then to select and press `Shift` while dragging to constrain to a perfect circle.
2. Press `Alt` + , then release the `Alt` key while dragging and press `Shift` to constrain.

Tip

You can switch the behavior of the `Alt` key to use the `Ctrl` key instead by toggling the switch in the General Settings.

Tool Options

Mode
: This option is explained in the Pixel and Vector Selection Types section.

Action
: This option is explained in the Pixel and Vector Selection Types section.

Anti-aliasing
: This toggles whether or not to give selections feathered edges. Some people prefer hard-jagged edges for their selections.

Note

Anti-aliasing is only available on Pixel Selection Mode.

Width
: Shows you the current width while you are creating the selection. You can manually type the value and use the ‘Lock Width’ for your next selection to have the selected value.

Lock Width

Forces the next selection to have the current width.

Height

Shows you the current height while you are creating the selection. You can manually type the value and use the ‘Lock Height’ for your next selection to have the selected value.

Lock Height

Forces the next selection to have the current height.

Ratio

Shows the ratio between height and width of the selection. Similar to Height, and Width, you can manually type the value and use the ‘Lock Ratio’ for your next selection to have the selected value.

Lock Ratio

Forces the next selection to have the current ratio.

Hint

If you want your selection to be of specific size:

1. Type the width and height.
2. Press the Lock Width and Lock Height buttons.
3.  where you want your selection to be.

Outline Selection Tool

Make Selections by drawing freehand around the canvas. Click and drag to draw a border around the section you wish to select.

Important

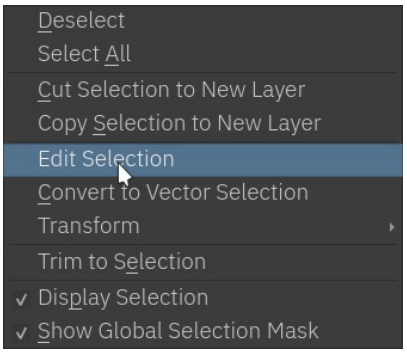
Most of the behavior of the Outline Selection Tool is common to all other selection tools, please make sure to read Selections to learn more about this tool.

Hotkeys and Sticky keys

- `R` sets the selection to ‘replace’ in the tool options, this is the default mode.
- `A` sets the selection to ‘add’ in the tool options.
- `S` sets the selection to ‘subtract’ in the tool options.
- `Shift +`  sets the subsequent selection to ‘add’. You can release the `Shift` key while dragging, but it will still be set to ‘add’. Same for the others.
- `Alt +`  sets the subsequent selection to ‘subtract’.
- `Ctrl +`  sets the subsequent selection to ‘replace’.
- `Shift + Alt +`  sets the subsequent selection to ‘intersect’.
- Holding `Ctrl` key while drawing the selection temporarily makes this tool to behave like polygon selection tool and you can then draw straight line selections by just clicking on canvas.

  New in version 4.2:

- Hovering your cursor over the dashed line of the selection, or marching ants as it is commonly called, turns the cursor into the move tool icon, which you  and drag to move the selection.
-  will open up a selection quick menu with amongst others the ability to edit the selection.



Tip

You can switch the behavior of the `Alt` key to use `Ctrl` instead by toggling the switch in Tool Settings in the General Settings

Tip

This tool is not bound to any Hotkey, if you want to define one, go to *Settings* ▸ *Configure Krita* ▸ *Keyboard Shortcuts* and search for 'Outline Selection Tool', there you can select the shortcut you want. Check Shortcut Settings for more info.

Tool Options

Mode
: This option is explained in the Pixel and Vector Selection Types section.

Action
: This option is explained in the Pixel and Vector Selection Types section.

Anti-aliasing
: This toggles whether or not to give selections feathered edges. Some people prefer hard-jagged edges for their selections.

Note

Anti-aliasing is only available on Pixel Selection Mode.

Polygonal Selection Tool

This tool, represented by a polygon with a dashed border, allows you to make Selections in a polygonal shape. To make a polygonal selection  and place points or nodes of the polygon. To finalize your selection area you can do either  on the first created point, or double  click to end the polygon.

Important

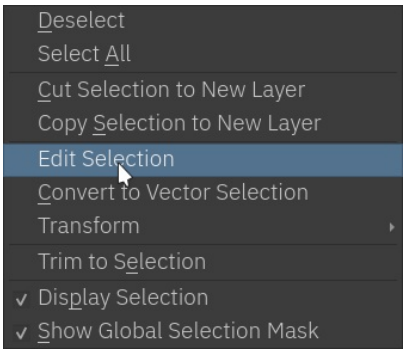
Most of the behavior of the Polygonal Selection Tool is common to all other selection tools, please make sure to read Selections to learn more about this tool.

Hotkeys and Sticky keys

- `R` sets the selection to ‘replace’ in the tool options, this is the default mode.
- `A` sets the selection to ‘add’ in the tool options.
- `S` sets the selection to ‘subtract’ in the tool options.
- `Shift` +  sets the subsequent selection to ‘add’. You can release the `Shift` key while dragging, but it will still be set to ‘add’. Same for the others.* `Alt` +  sets the subsequent selection to ‘subtract’.
- `Ctrl` +  sets the subsequent selection to ‘replace’.
- `Shift + Alt` +  sets the subsequent selection to ‘intersect’.

  New in version 4.2:

- Hovering your cursor over the dashed line of the selection, or marching ants as it is commonly called, turns the cursor into the move tool icon, which you  and drag to move the selection.
-  will open up a selection quick menu with amongst others the ability to edit the selection.



Tip

You can switch the behavior of the `Alt` key to use `Ctrl` instead by toggling the switch in Tool Settings in the General Settings.

Tip

This tool is not bound to any Hotkey, if you want to define one, go to *Settings* ▸ *Configure Krita* ▸ *Keyboard Shortcuts* and search for 'Polygonal Selection Tool', there you can select the shortcut you want. Check Shortcut Settings for more info.

Tool Options

Mode
: This option is explained in the Pixel and Vector Selection Types section.

Action
: This option is explained in the Pixel and Vector Selection Types section.

Anti-aliasing
: This toggles whether or not to give selections feathered edges. Some people prefer hard-jagged edges for their selections.

Note

Anti-aliasing is only available on Pixel Selection Mode.

Contiguous Selection Tool

This tool, represented by a magic wand, allows you to make Selections by selecting a point of color. It will select any contiguous areas of a similar color to the one you selected. You can adjust the “fuzziness” of the tool in the tool options dock. A lower number will select colors closer to the color that you chose in the first place.

Hotkeys and Sticky keys

- `R` sets the selection to ‘replace’ in the tool options, this is the default mode.
- `A` sets the selection to ‘add’ in the tool options.
- `S` sets the selection to ‘subtract’ in the tool options.
- `Shift +`  sets the subsequent selection to ‘add’. You can release the `Shift` key while dragging, but it will still be set to ‘add’. Same for the others.
- `Alt +`  sets the subsequent selection to ‘subtract’.
- `Ctrl +`  sets the subsequent selection to ‘replace’.
- `Shift + Alt +`  sets the subsequent selection to ‘intersect’.

  New in version 4.2:

- Hovering over a selection allows you to move it.
-  will open up a selection quick menu with amongst others the ability to edit the selection.

Note

You can switch the behavior of the `Alt` key to use the `Ctrl` key instead by toggling the switch in the General Settings.

Tool Options

Anti-aliasing
: This toggles whether or not to give selections feathered edges. Some people prefer hard-jagged edges for their selections.

Fuzziness
: This controls whether or not the contiguous selection sees another color as a border.

Grow/Shrink selection.
: This value extends/contracts the shape beyond its initial size.

Feathering
: This value will add a soft border to the selection-shape.

Sample
: *New in version 4.3.*

  Select which layers to use as a reference for the contiguous select tool. The options are:

  Current Layer
  : Only use the currently selected layer.

  All layers
  : Use all visible layers.

  Color Labeled Layers
  : Use only the layers specified with a certain color label. This is useful for complex images, where you might have multiple lineart layers. Label them with the appropriate color label and use these labels to mark which layers to use as a reference.

Labels Used

New in version 4.3.

Used with the ‘Color Labeled Layers’ option above.

Path Selection Tool

This tool, represented by an ellipse with a dashed border and a curve control, allows you to make a Selections of an area by drawing a path around it. Click where you want each point of the path to be. Click and drag to curve the line between points. Finally click on the first point you created to close your path.

Hotkeys and Sticky keys

- `R` sets the selection to ‘replace’ in the tool options, this is the default mode.
- `A` sets the selection to ‘add’ in the tool options.
- `S` sets the selection to ‘subtract’ in the tool options.
- `Shift +`  sets the subsequent selection to ‘add’. You can release the `Shift` key while dragging, but it will still be set to ‘add’. Same for the others.
- `Alt +`  sets the subsequent selection to ‘subtract’.
- `Ctrl +`  sets the subsequent selection to ‘replace’.
- `Shift + Alt +`  sets the subsequent selection to ‘intersect’.

  New in version 4.2:

- Hovering over a selection allows you to move it.
-  will open up a selection quick menu with amongst others the ability to edit the selection.

Note

You can switch the behavior of the `Alt` key to use the `Ctrl` key instead by toggling the switch in the General Settings.

Tool Options

New in version 4.1.3:
Autosmooth Curve
: Toggling this will have nodes initialize with smooth curves instead of angles. Untoggle this if you want to create sharp angles for a node. This will not affect curve sharpness from dragging after clicking.

Anti-aliasing
: This toggles whether or not to give selections feathered edges. Some people prefer hard-jagged edges for their selections.

New in version 4.2:
Autosmooth Curve
: Toggling this will have nodes initialize with smooth curves instead of angles. Untoggle this if you want to create sharp angles for a node. This will not affect curve sharpness from dragging after clicking.

Angle Snapping Delta
: The angle to snap to.

Activate Angle Snap
: Angle snap will make it easier to have the next line be at a specific angle of the current. The angle is determined by the *Angle Snapping Delta*.

Similar Color Selection Tool

This tool, represented by a dropper over an area with a dashed border, allows you to make Selections by selecting a point of color. It will select any areas of a similar color to the one you selected. You can adjust the “fuzziness” of the tool in the tool options dock. A lower number will select colors closer to the color that you chose in the first place.

Important

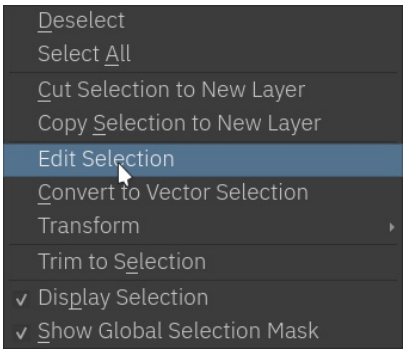
Most of the behavior of the Similar Color Selection Tool is common to all other selection tools, please make sure to read Selections to learn more about this tool.

Hotkeys and Sticky keys

- `R` sets the selection to ‘replace’ in the tool options, this is the default mode.
- `A` sets the selection to ‘add’ in the tool options.
- `S` sets the selection to ‘subtract’ in the tool options.
- `Shift +`  sets the subsequent selection to ‘add’. You can release the `Shift` key while dragging, but it will still be set to ‘add’. Same for the others.
- `Alt +`  sets the subsequent selection to ‘subtract’.
- `Ctrl +`  sets the subsequent selection to ‘replace’.
- `Shift + Alt +`  sets the subsequent selection to ‘intersect’.

  New in version 4.2:

- Hovering your cursor over the dashed line of the selection, or marching ants as it is commonly called, turns the cursor into the move tool icon, which you  and drag to move the selection.
-  will open up a selection quick menu with amongst others the ability to edit the selection.



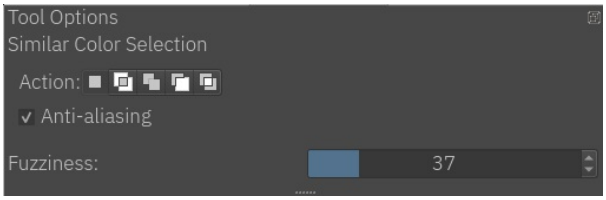
Tip

You can switch the behavior of the `Alt` key to use `Ctrl` instead by toggling the switch in Tool Settings in the General Settings

Tip

This tool is not bound to any Hotkey, if you want to define one, go to *Settings* ▸ *Configure Krita* ▸ *Keyboard Shortcuts* and search for 'Similar Color Selection Tool', there you can select the shortcut you want. Check Shortcut Settings for more info.

Tool Options



Action

This option is explained in the Pixel and Vector Selection Types section.

Anti-aliasing

This toggles whether or not to give selections feathered edges. Some people prefer hard-jagged edges for their selections.

Note

Anti-aliasing is only available on Pixel Selection Mode.

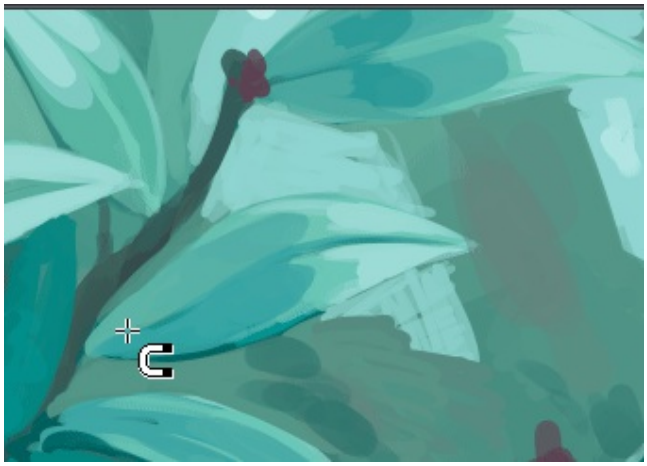
Fuzziness

This controls the range of the color hue used to create the selection. A lower number will select colors closer to the color that you chose in the first place. And a higher number will expand the hue range and select colors even if they are not so similar to the original color.

Magnetic Selection Tool

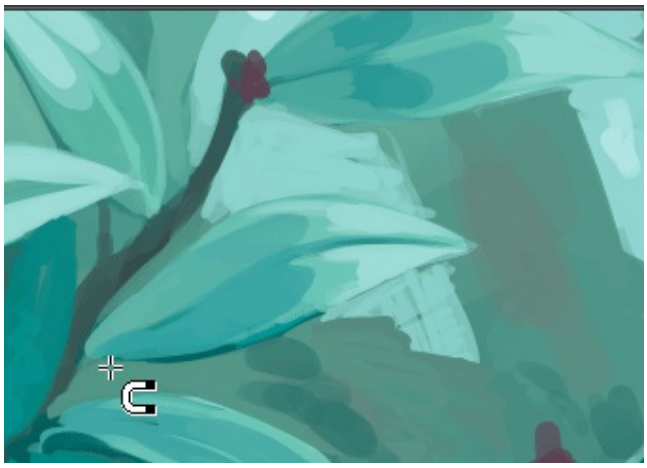
This tool, represented by a magnet over a selection border, allows you to make freeform Selections, but unlike the Polygonal Selection Tool or the Outline Selection Tool, it will try to magnetically snap to sharp contrasts in your image, simplifying the creation of selection drastically.

There are two ways to make a magnetic selection:



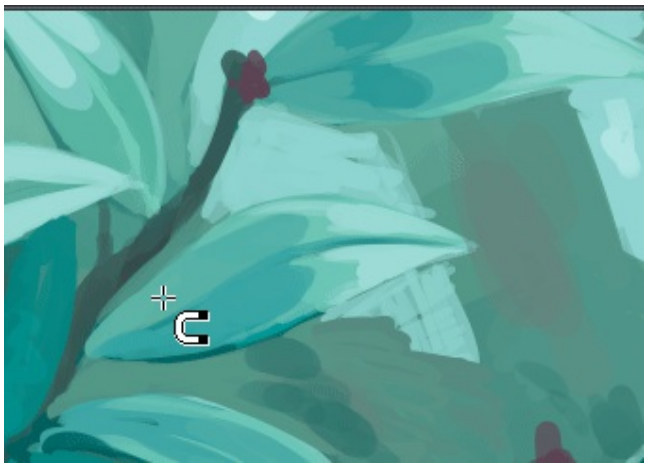
Animation showing the first mode of creating a magnetic selection.

The first is to use and place points or nodes of the magnetic selection. To finalize your selection area you can do either on the first created point to complete the loop and click on it again to create a selection, or press `Enter` to end the magnetic selection.



Animation showing the second (interactive) mode of creating a magnetic selection.

The second, interactive mode, is to  `+ drag` over a portion of an image.



The first and second mode can be mixed.

You can edit previous points by  dragging them. You can remove points by dragging it out of the canvas area. After a path is closed. Points can be undone with `Shift + Z`. A halfway done magnetic selection can be canceled with `Esc`.

Important

Most of the behavior of the Magnetic Selection Tool is common to all other

selection tools, please make sure to read Selections to learn more about this tool.

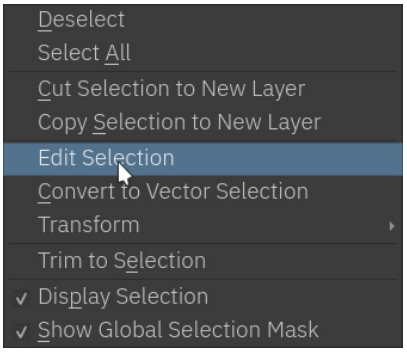
Hotkeys and Sticky keys

- `R` sets the selection to ‘replace’ in the tool options, this is the default mode.
- `A` sets the selection to ‘add’ in the tool options.
- `S` sets the selection to ‘subtract’ in the tool options.
- `Shift +`  sets the subsequent selection to ‘add’. You can release the `Shift` key while dragging, but it will still be set to ‘add’. Same for the others.
- `Alt +`  sets the subsequent selection to ‘subtract’.
- `Ctrl +`  sets the subsequent selection to ‘replace’.
- `Shift + Alt +`  sets the subsequent selection to ‘intersect’.

  New in version 4.2:

- Hovering your cursor over the dashed line of the selection, or marching ants as it is commonly called, turns the cursor into the move tool icon, which you  and drag to move the selection.

-  will open up a selection quick menu with amongst others the ability to edit the selection.



Tip

You can switch the behavior of the `Alt` key to use `Ctrl` instead by toggling the switch in Tool Settings in the General Settings.

Tip

This tool is not bound to any Hotkey, if you want to define one, go to *Settings* ▸ *Configure Krita* ▸ *Keyboard Shortcuts* and search for 'Magnetic Selection Tool', there you can select the shortcut you want. Check Shortcut Settings for more info.

Tool Options

Mode
: This option is explained in the Pixel and Vector Selection Types section.

Action
: This option is explained in the Pixel and Vector Selection Types section.

Anti-aliasing
: This toggles whether or not to give selections feathered edges. Some people prefer hard-jagged edges for their selections.

Filter Radius:
: Determine the radius of the edge detection kernel. This determines how aggressively the tool will interpret contrasts. Low values mean only the sharpest of contrast will be a seen as an edge. High values will pick up on subtle contrasts. The range of which is from 2.5 to 100.

Threshold:
: From 0 to 255, how sharp your edge is, 0 is least while 255 is the most. Used in the interactive mode only.

Search Radius:

The area in which the tool will search for a sharp contrast within an image. More pixels means less precision is needed when placing the points, but this will require Krita to do more work, and thus slows down the tool.

Anchor Gap:

When using  `+ drag` to place points automatically, this value determines the average gap between 2 anchors. Low values give high precision by placing many nodes, but this is also harder to edit afterwards. The pixels are in screen dimensions and not image dimensions, meaning it is affected by zoom.

To the **left**: 20 px anchor gap, to the **right**: 40px anchor gap.

Note

Anti-aliasing is only available on Pixel Selection Mode.

Zoom Tool

The zoom tool allows you to zoom your canvas in and out discretely. It can be found at the bottom of the toolbox, and you just activate it by selecting the tool, and doing  on the canvas will zoom in, while  will zoom out.

You can reverse this behavior in the *Tool Options*.

There's a number of hotkeys associated with this tool, which makes it easier to access from the other tools:

- `Ctrl + Space +`  `+ drag` on the canvas will zoom in or out fluently.
- `Ctrl +`  `+ drag` on the canvas will zoom in or out fluently.
- `Ctrl + Alt + Space +`  `+ drag` on the canvas will zoom in or out with discrete steps.
- `Ctrl + Alt +`  `+ drag` on the canvas will zoom in or out with discrete steps.
- `+` will zoom in with discrete steps.
- `-` will zoom out with discrete steps.
- `1` will set the zoom to 100%.
- `2` will set the zoom so that the document fits fully into the canvas area.
- `3` will set the zoom so that the document width fits fully into the canvas area.

For more information on such hotkeys, check Navigation.

Pan Tool

The pan tool allows you to pan your canvas around freely. It can be found at the bottom of the toolbox, and you just it by selecting the tool, and doing  `+ drag` over the canvas.

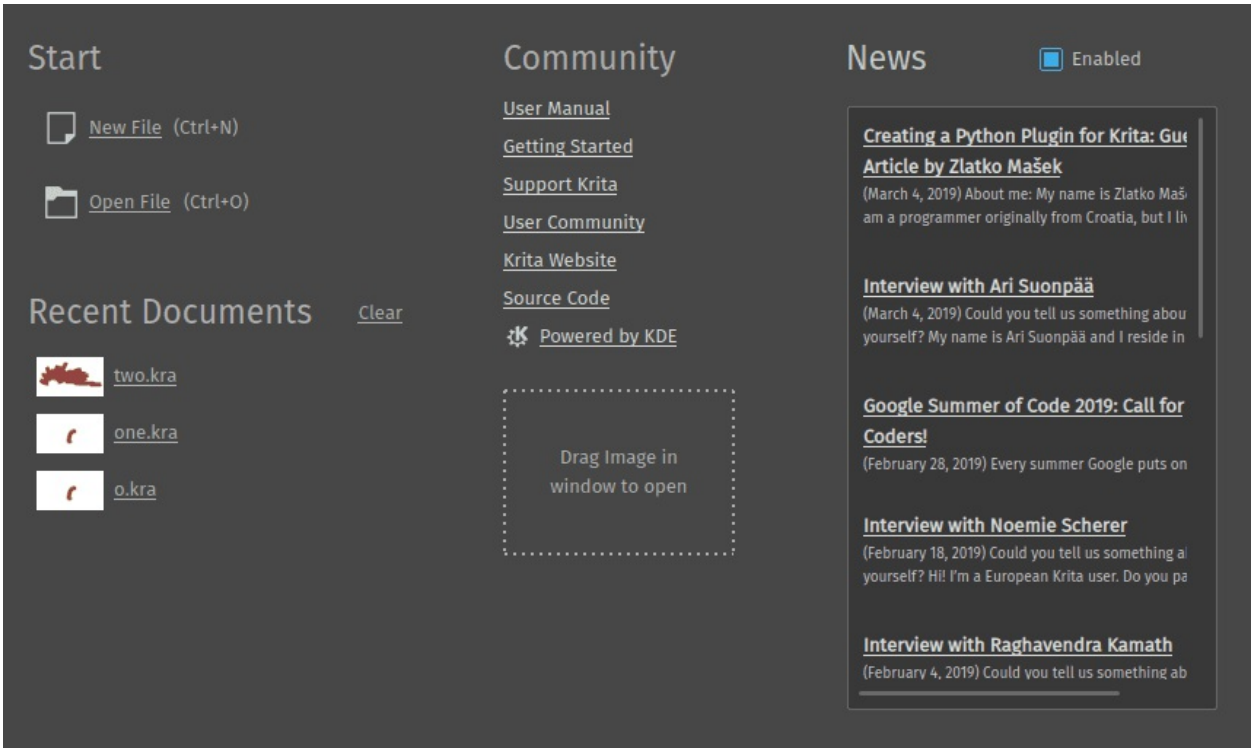
There are two hotkeys associated with this tool, which makes it easier to access from the other tools:

- `Space +`  `+ drag` over the canvas.
-  `+ drag` over the canvas.

For more information on such hotkeys, check Navigation.

Welcome Screen

When you open Krita, starting from version 4.1.3, you will be greeted by a welcome screen. This screen makes it easy for you to get started with Krita, as it provides a collection of shortcuts for the most common tasks that you will probably be doing when you open Krita.



The screen is divided into 4 sections:

- The *Start* section there are links to create new document as well to open an existing document.
- The *Recent Documents* section has a list of recently opened documents from your previous sessions in Krita.
- The *Community* section provides some links to get help, Supporting development of Krita, Source code of Krita and to links to interact with our user community.
- The *News* section, which is disabled by default, when enabled provides you with latest news feeds fetched from Krita website, this will help you

stay up to date with the release news and other events happening in our community.

Other than the above sections the welcome screen also acts as a drop area for opening any document. You just have to drag and drop a Krita document or any supported image files on the empty area around the sections to open it in Krita.

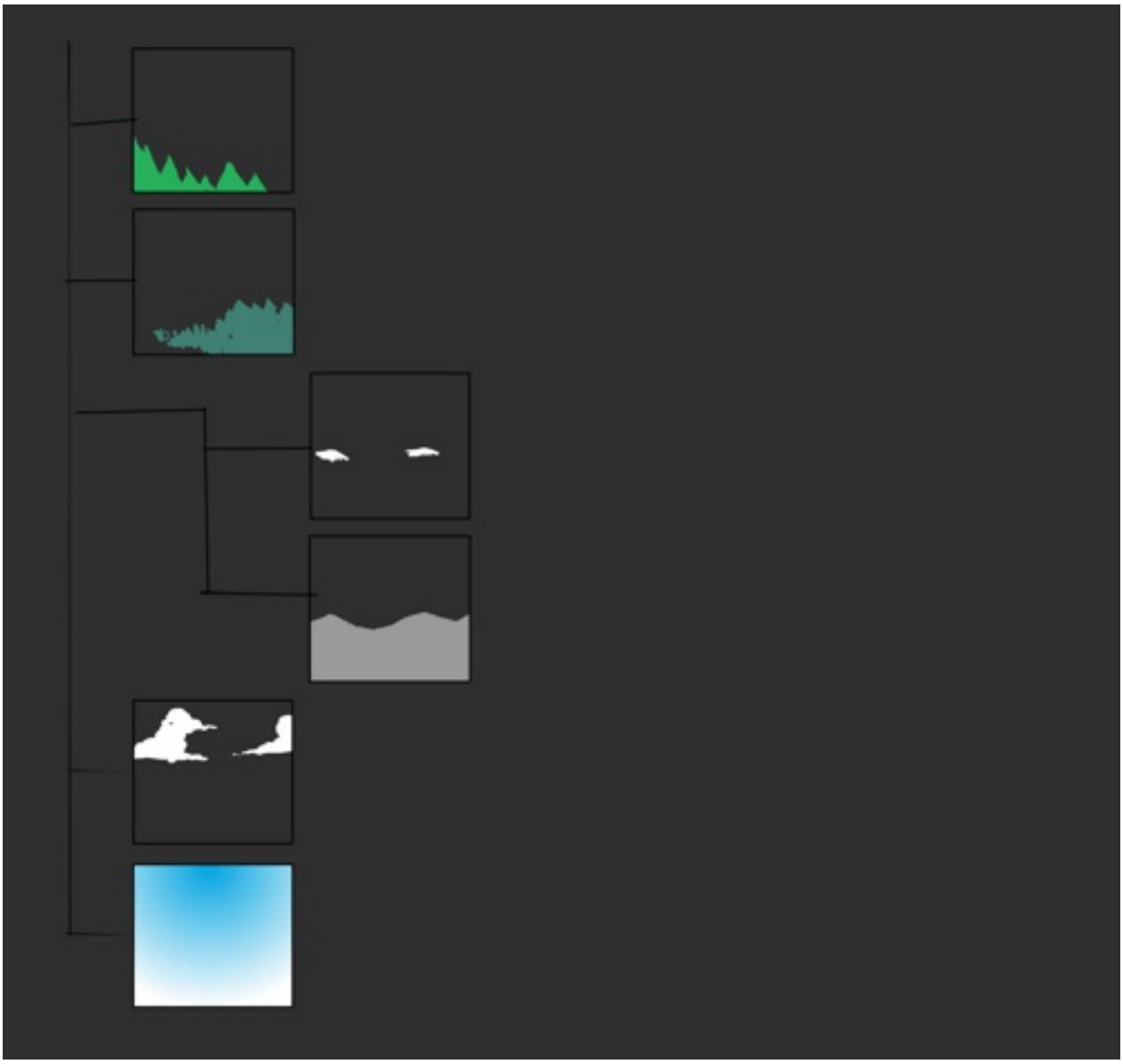
Tutorials and Howto’s

Learn through developer and user generated tutorials to see Krita in action.

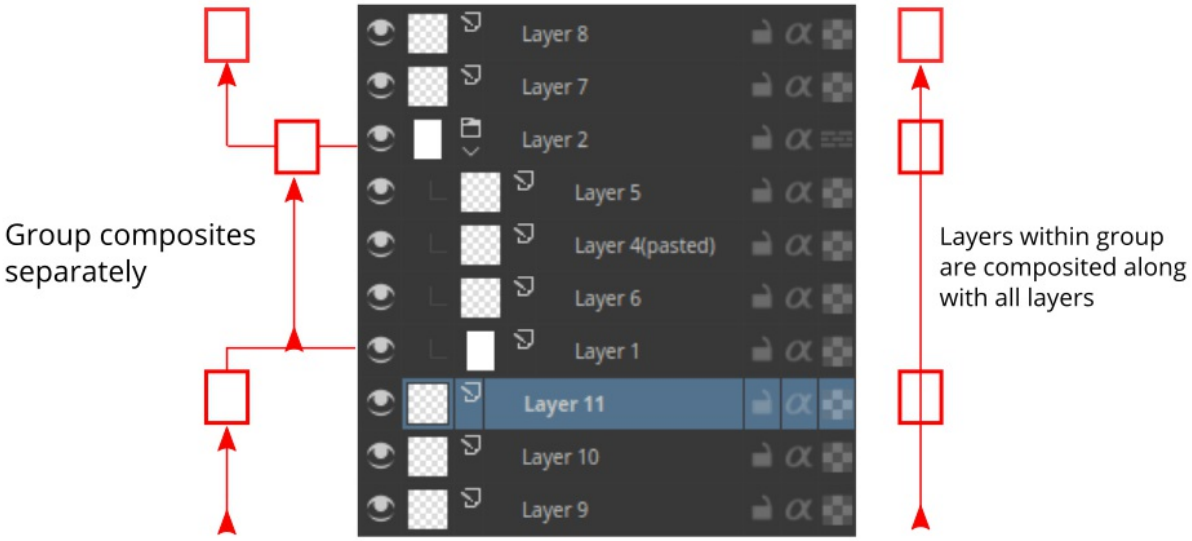
Contents:

Clipping Masks and Alpha Inheritance

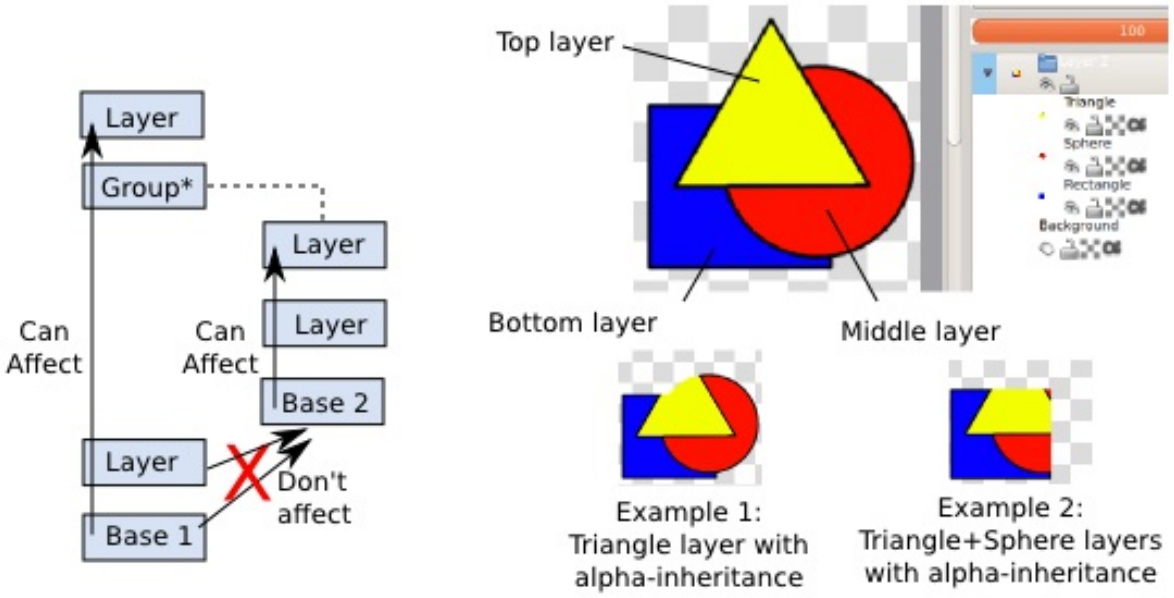
Krita doesn’t have clipping mask functionality in the manner that Photoshop and programs that mimic Photoshop’s functionality have. That’s because in Krita, unlike such software, a group layer is not an arbitrary collection of layers. Rather, in Krita, group layers are composited separately from the rest of the stack, and then the result is added into the stack. In other words, in Krita group layers are in effect distinct images inside your image.



The exception is when using pass-through mode, meaning that alpha inheritance won’t work right when turning on pass-through on the layer.

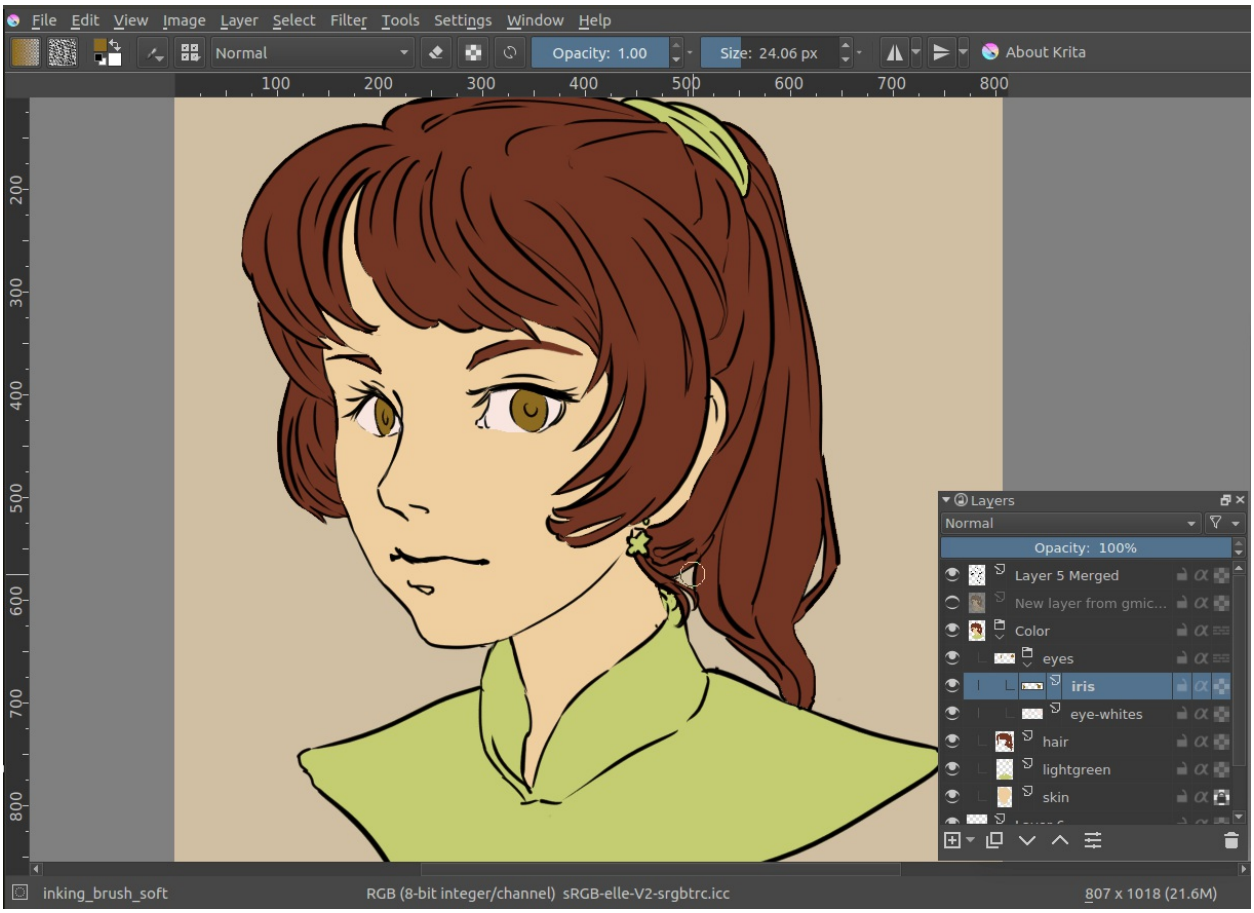


When we turn on alpha inheritance, the alpha-inherited layer keeps the same transparency as the layers below.



Combined with group layers this can be quite powerful. A situation where this is particularly useful is the following:

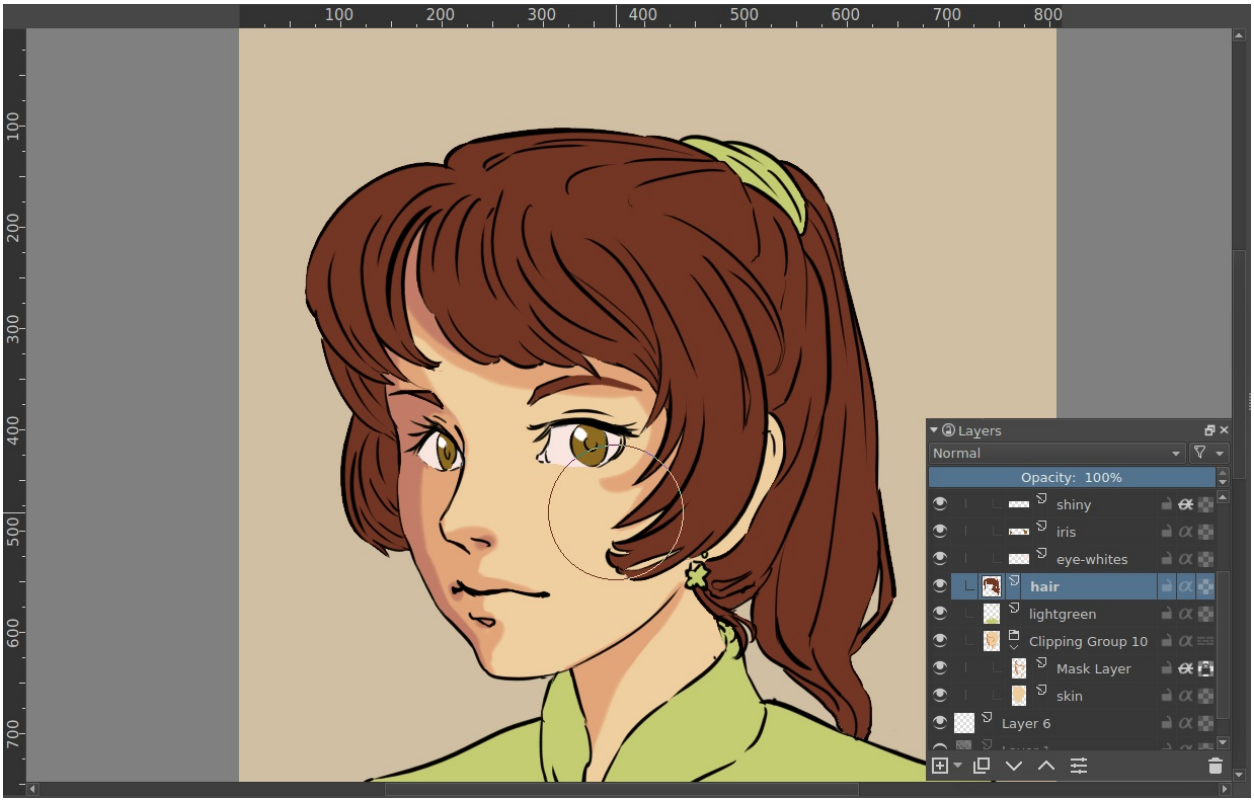
Here we have an image with line art and a layer for each flat of colors. We want to add complicated multi-layered shading to this, while keeping the neatness of the existing color flats. To get a clipping mask working, you first need to put layers into a group. You can do this by making a group layer and drag-and-dropping the layers into it, or by selecting the layers you want grouped and pressing the `Ctrl + G` shortcut. Here we do that with the iris and the eye-white layers.



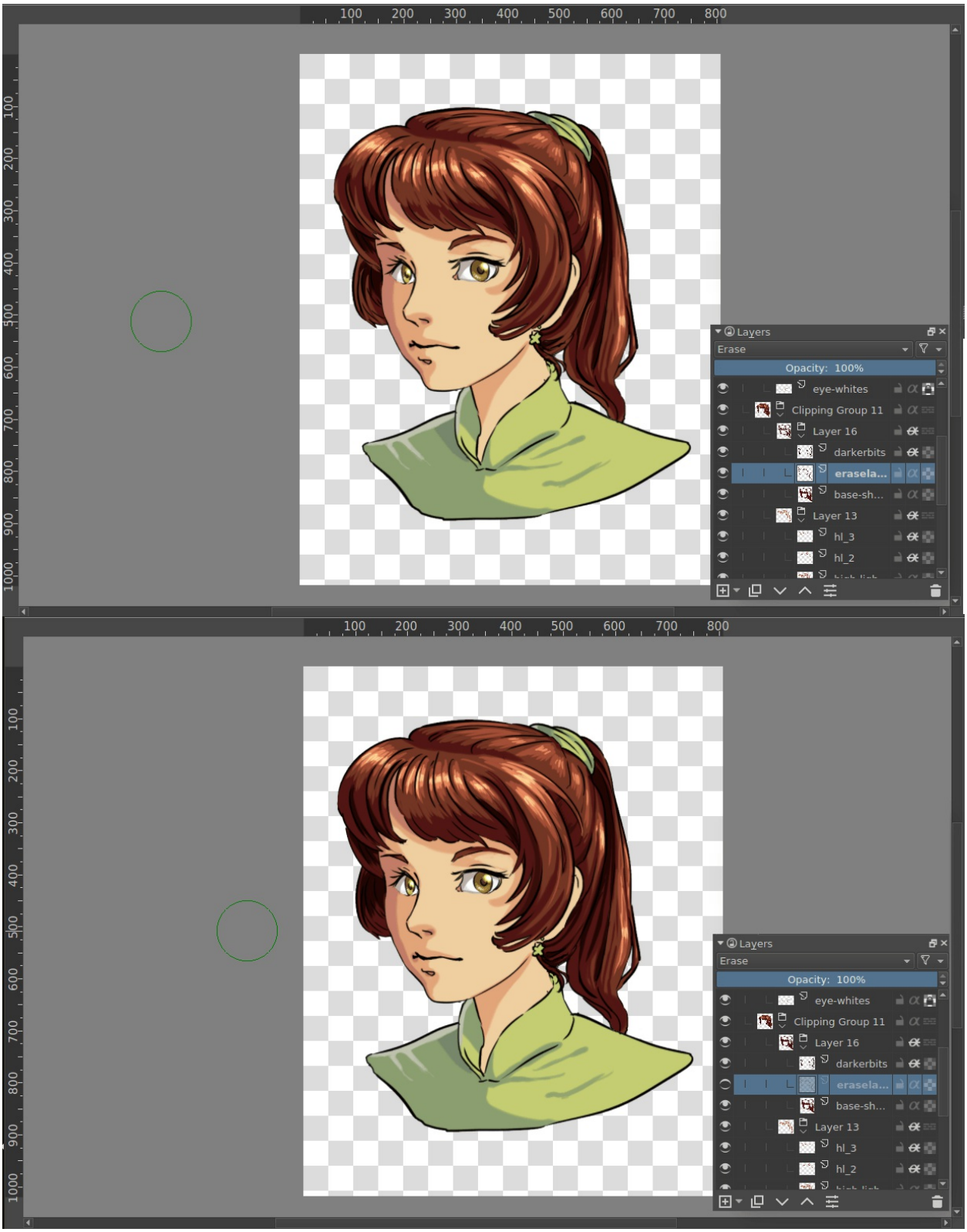
We add a layer for the highlight above the other two layers, and add some white scribbles.

In the above, we have our layer with a white scribble on the left, and on the right, the same layer, but with alpha inheritance active, limiting it to the

combined area of the iris and eye-white layers.

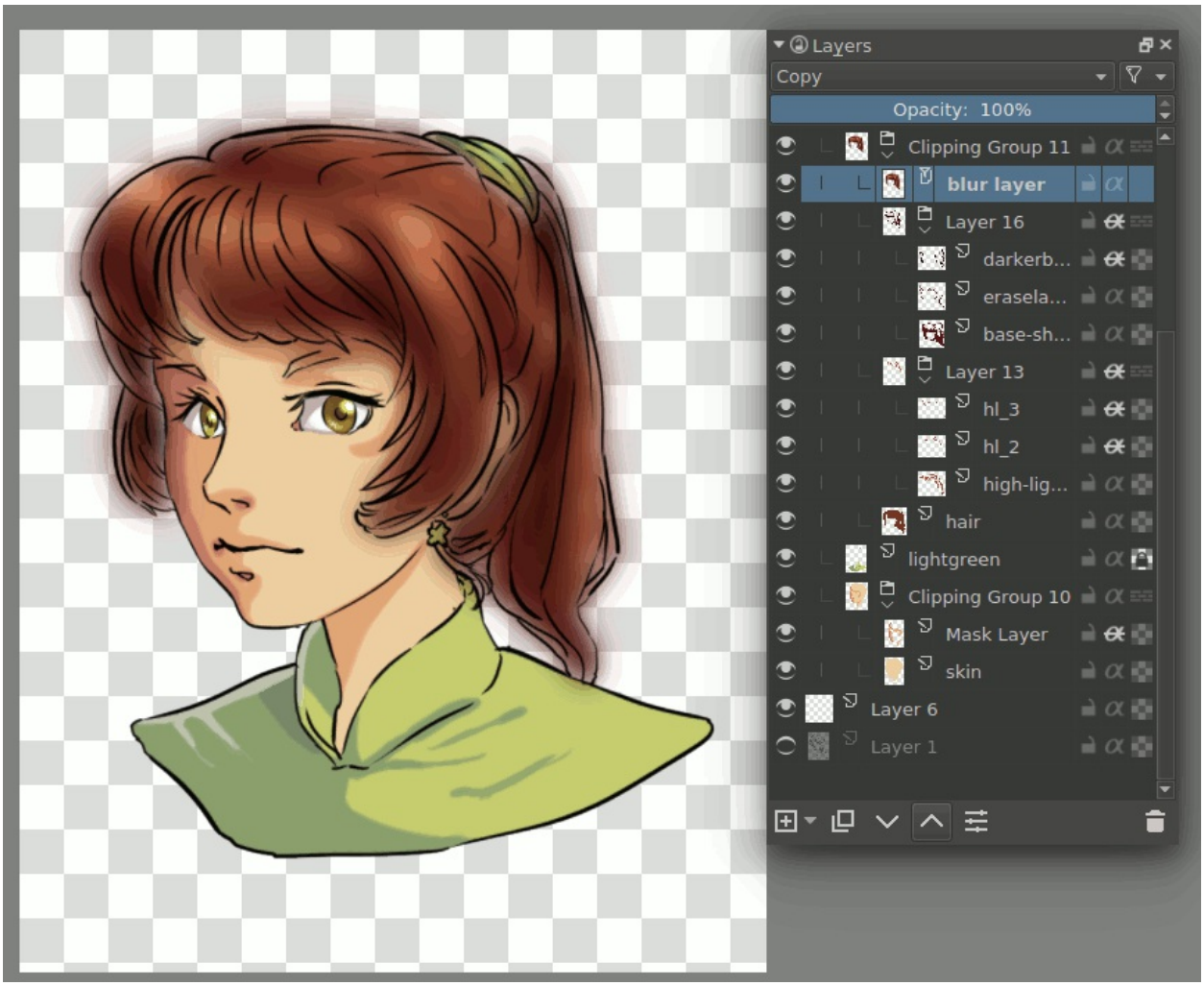


Now there’s an easier way to set up alpha inheritance. If you select a layer or set of layers and press the `Ctrl + Shift + G` shortcut, you create a quick clipping group. That is, you group the layers, and a ‘mask layer’ set with alpha inheritance is added on top.



The fact that alpha inheritance can use the composited transparency from a combination of layers means that you can have a layer with the erase-

blending mode in between, and have that affect the area that the layer above is clipped to. Above, the lower image is exactly the same as the upper one, except with the erase-layer hidden. Filters can also affect the alpha inheritance:



Above, the blur filter layer gives different results when in different places, due to different parts being blurred.

Common Workflows

Krita's main goal is to help artists create a digital painting from scratch. Krita is used by comic artists, matte painters, texture artists, and illustrators around the world. This section explains some common workflow that artists use in Krita. When you open a new document in Krita for the first time, you can start painting instantly. The brush tool is selected by default and you just have to paint on the canvas. However, let us look at what artists do in Krita. Below are some of the common workflows used in Krita:

Speed Painting and Conceptualizing

Some artists work only on the digital medium, sketching and visualizing concepts in Krita from scratch. As the name suggests a technique of painting done within a matter of hours to quickly visualize the basic scene, character, look and feel of the environment or to denote the general mood and overall concept is called a **speed painting**. Finishing and finer details are not the main goals of this type of painting, but the representation of form value and layout is the main goal.

Some artists set a time limit to complete the painting while some paint casually. Speed painting then can be taken forward by adding finer details and polish to create a final piece. Generally, artists first block in the composition by adding patches and blobs of flat colors, defining the silhouette, etc. Krita has some efficient brushes for this situation, for example, the brushes under **Block Tag** like Block fuzzy, Block basic, layout_block, etc.

After the composition and a basic layout has been laid out the artists add as many details as possible in the given limited time, this requires a decent knowledge of forms, value perspective and proportions of the objects. Below is an example of speed paint done by David Revoy [https://www.davidrevoy.com/] in an hours time.

Artwork by David Revoy, license : CC-BY
[https://creativecommons.org/licenses/by/3.0/]

You can view the recorded speed painting demo for the above image on Youtube [https://www.youtube.com/watch?v=93lMLEuxSLk].

Colorizing Line Art

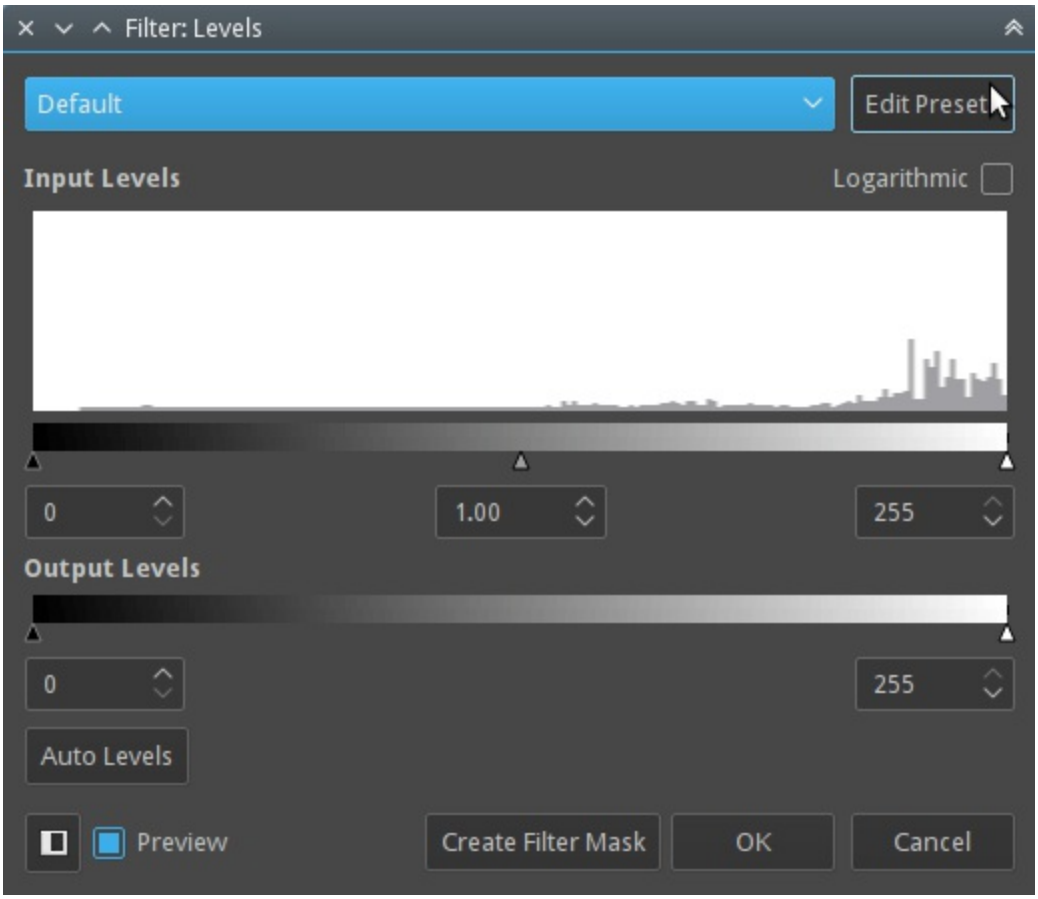
Often an artist, for example a comic book colorist, will need to take a pencil sketch or other line art of some sort and use Krita to paint underneath it. This can be either an image created digitally or something that was done outside the computer and has been scanned.

Preparing the line art

If your images have a white or other single-tone background, you can use either of the following methods to prepare the art for coloring:

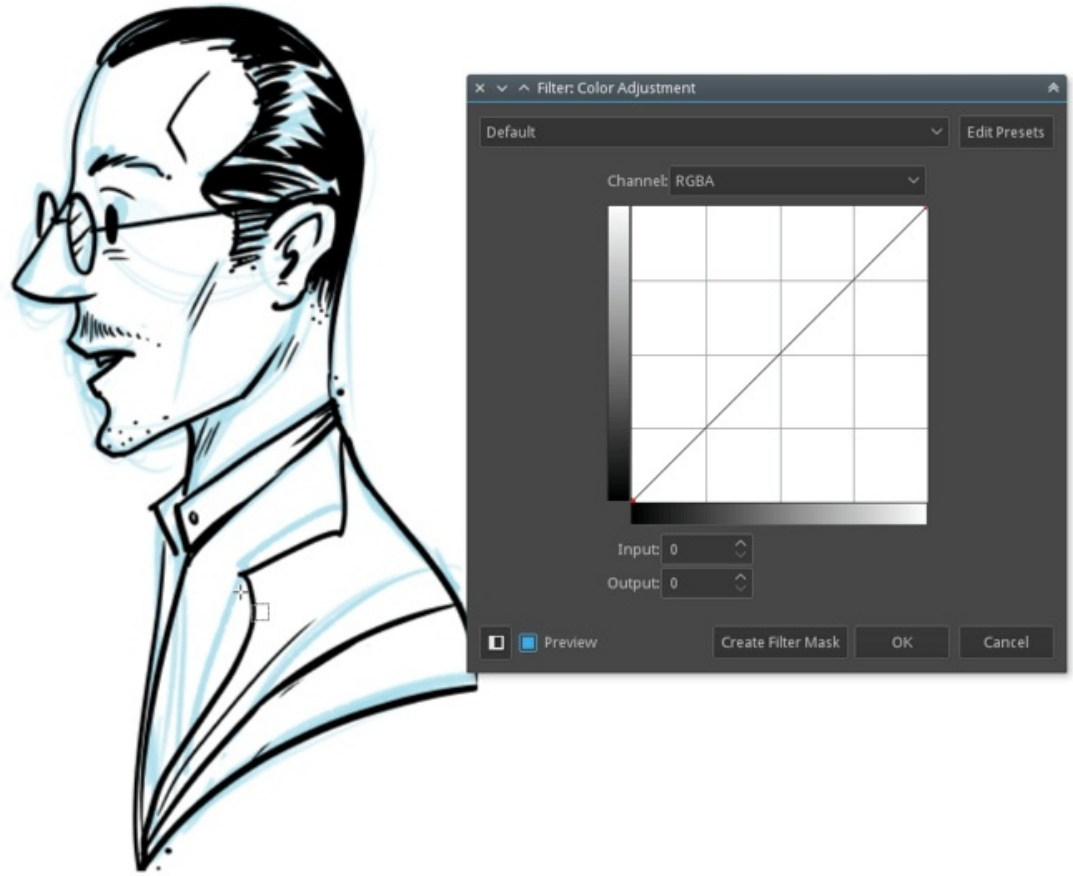
Place the line art at the top of the layer stack and set its layer blending mode to *Multiply*.

If you want to clean the line art a bit you can press the `Ctrl + L` shortcut or go to *Filters* ▸ *Adjust* ▸ *Levels*.

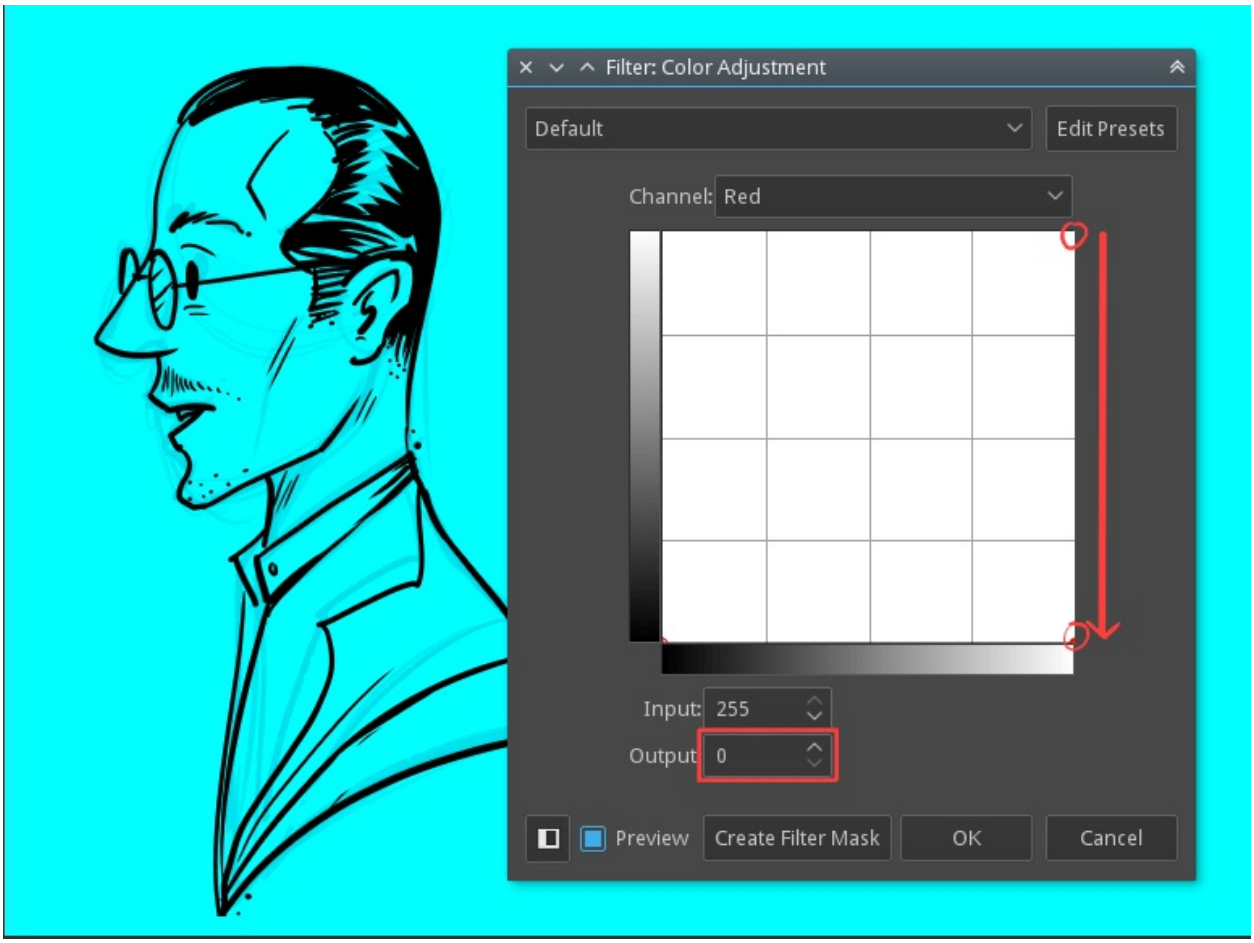


You can clean the unwanted grays by moving the white triangle in the input levels section to left and darken the black by moving the black triangle to right.

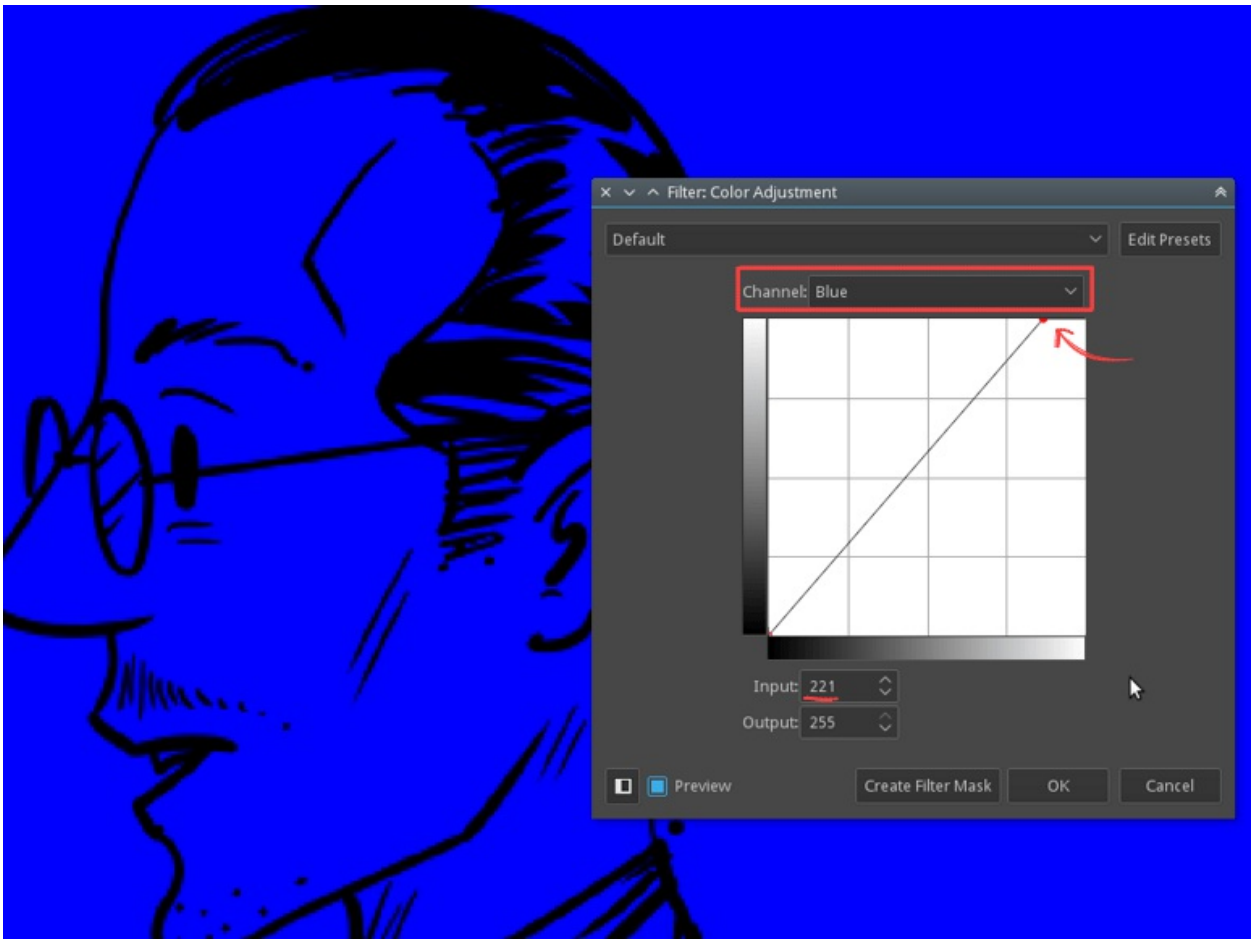
If you draw in blue pencils and then ink your line art you may need to remove the blue lines first to do that go to *Filters* ▸ *Adjust* ▸ *Color adjustment curves* or press the `Ctrl + M` shortcut.



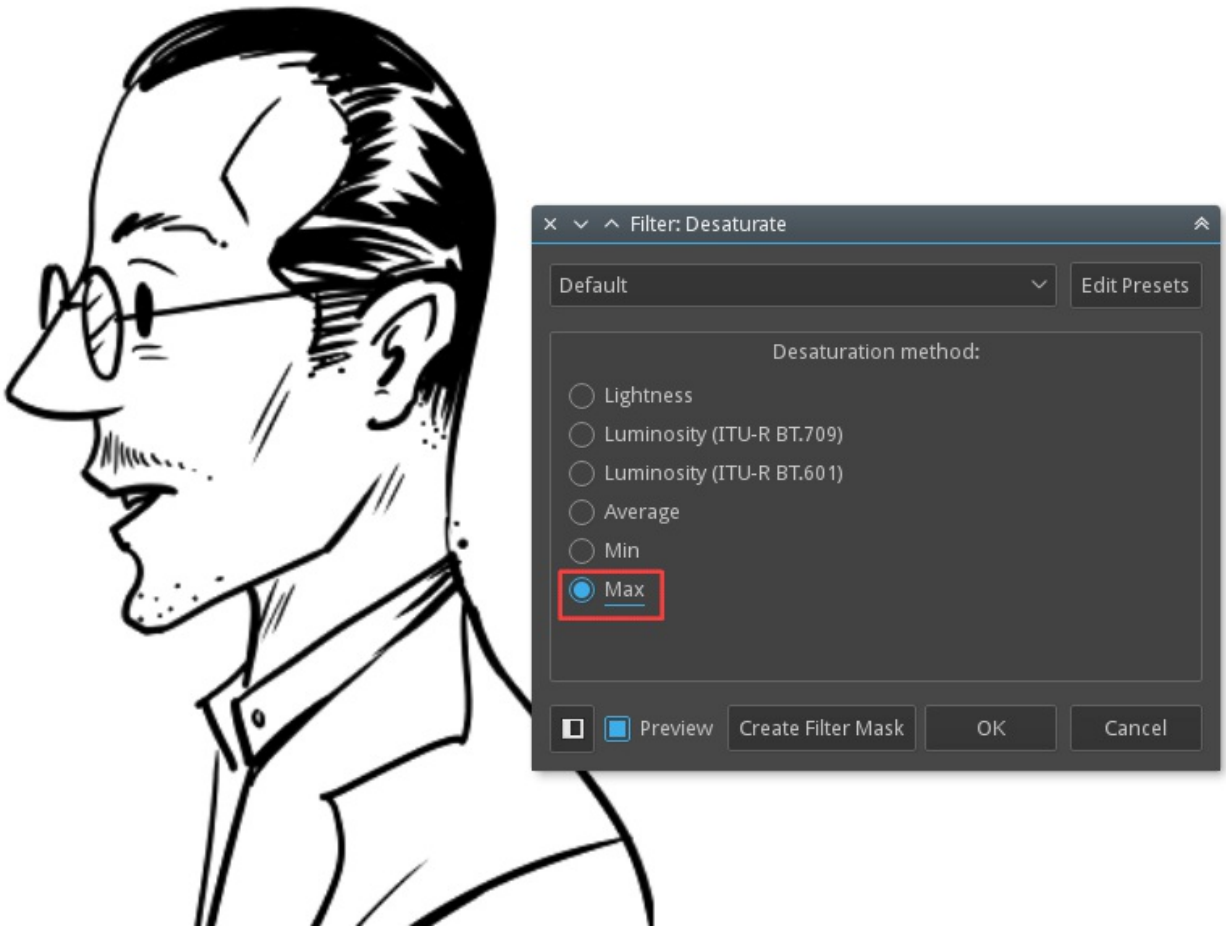
Now select **Red** from the drop-down, click on the top right node on the graph and slide it all the way down. Or you can click on the top right node and enter **0** in the input field. Repeat this step for **Green** too.



Now the whole drawing will have a blue overlay, zoom in and check if the blue pencil lines are still visible slightly. If you still see them, then go to **Blue** Channel in the color adjustment and shift the top right node towards left a bit, Or enter a value around 190 (one that removes the remaining rough lines) in the input box.



Now apply the color adjustment filter, yes we still have lots of blue on the artwork. Be patient and move on to the next step. Go to *Filters* ▸ *Adjust* ▸ *Desaturate* or press the `Ctrl + Shift + U` shortcut. Now select *Max* from the list.



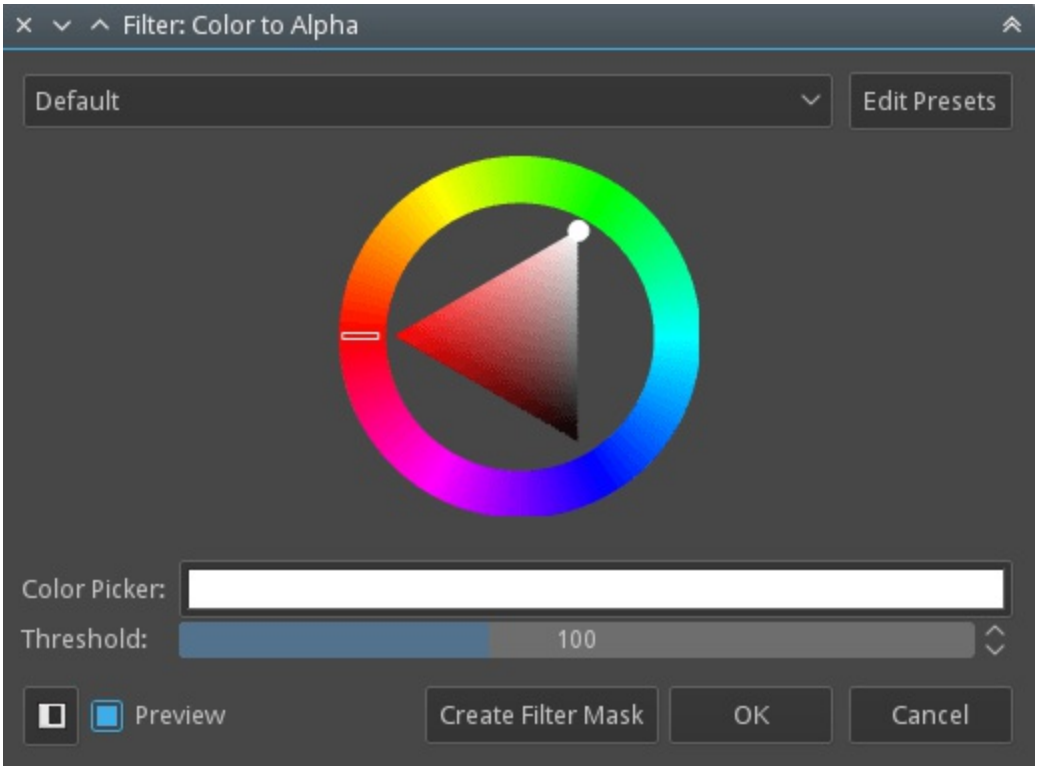
Hint

It is good to use non-photo-blue pencils to create the blue lines as those are easy to remove. If you are drawing digitally in blue lines use #A4DDED color as this is closer to non-photo-blue color.

You can learn more about doing a sketch from blue sketch to digital painting here in a tutorial by David Revoy [https://www.davidrevoy.com/article239/clean-blue-sketch-traditional-line-art-to-color-it-digital-with-in-krita].

After you have a clean black and white line art you may need to erase the white color and keep only black line art, to achieve that go to *Filters* ▸ *Colors* ▸ *Color to Alpha…* menu item. Use the dialog box to turn all the white areas of the image transparent. The Color Picker is set to White by default. If you have imported scanned art and need to select another color for the paper color

then you would do it here.



This will convert the white color in your line art to alpha i.e. it will make the white transparent leaving only the line art. Your line art can be in grayscale color space, this is a unique feature in Krita which allows you to keep a layer in a color-space independent from the image.

Laying in Flat Colors

There are many ways to color a line art in Krita, but generally, these three are common among the artists.

1. Paint blocks of color directly with block brushes.
2. Fill with Flood Fill Tool.
3. Use a Colorize Mask.

Blocking with brush

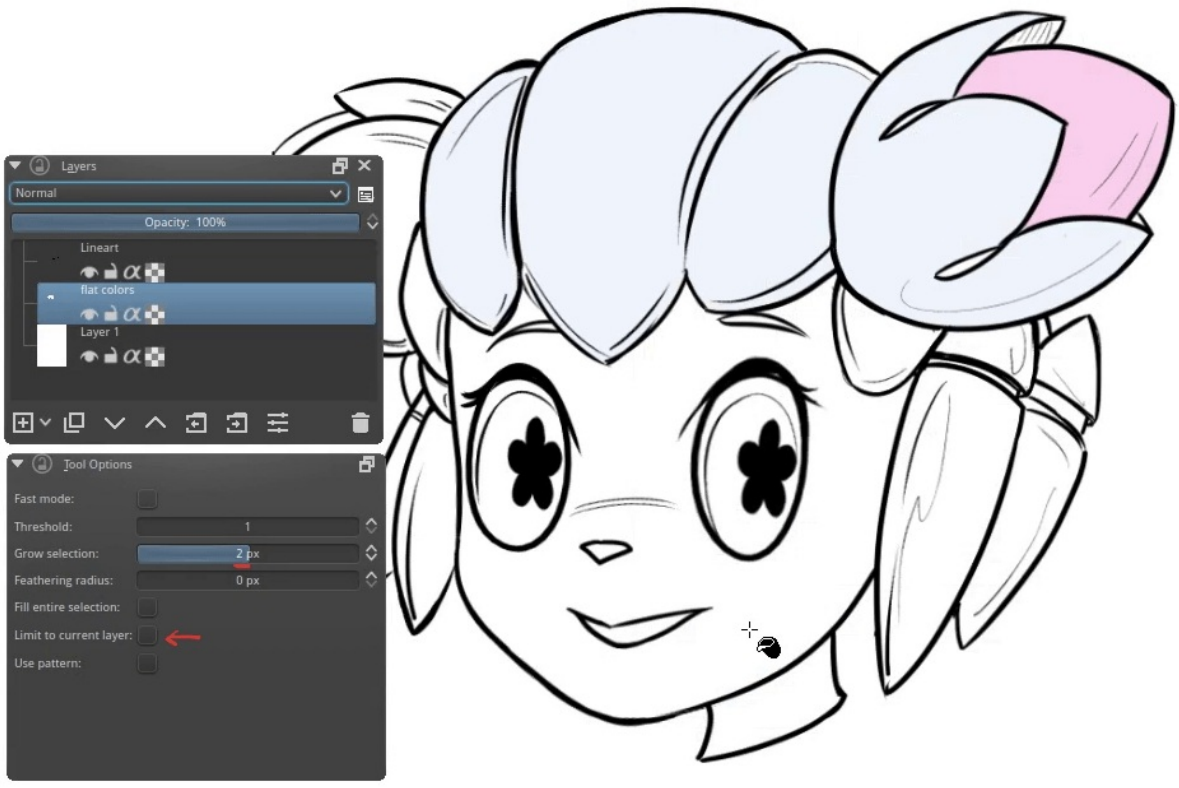
The first is the more traditional method of taking a shape brush or using the

geometric tools to lay in color. This would be similar to using an analog marker or brush on paper. There are various block brushes in Krita, you can select **Block** Tag from the drop-down in the brush presets docker and use the brushes listed there.

Add a layer underneath your line art layer and start painting with the brush. If you want to correct any area you can press the E key and convert the same brush into an eraser. You can also use a layer each for different colors for more flexibility.

Filling with Flood Fill tool

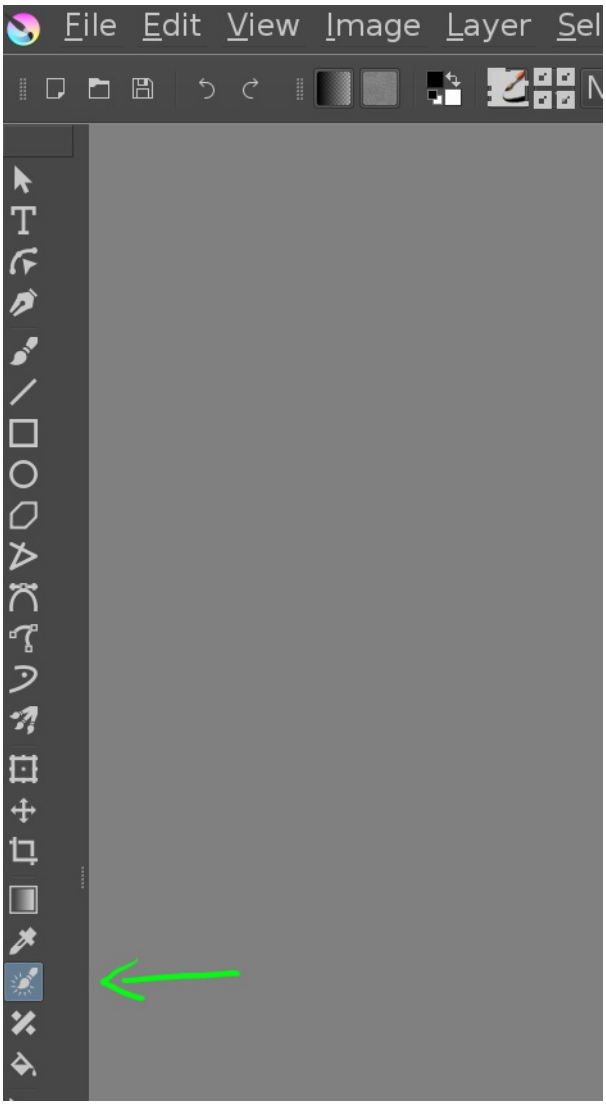
The second method is to use the Flood fill tool to fill large parts of your line art quickly. This method generally requires closed gaps in the line art. To begin with this method place your line art on a separate layer. Then activate the flood fill tool and set the *Grow selection* to 2px, uncheck *Limit to current layer* if previously checked.



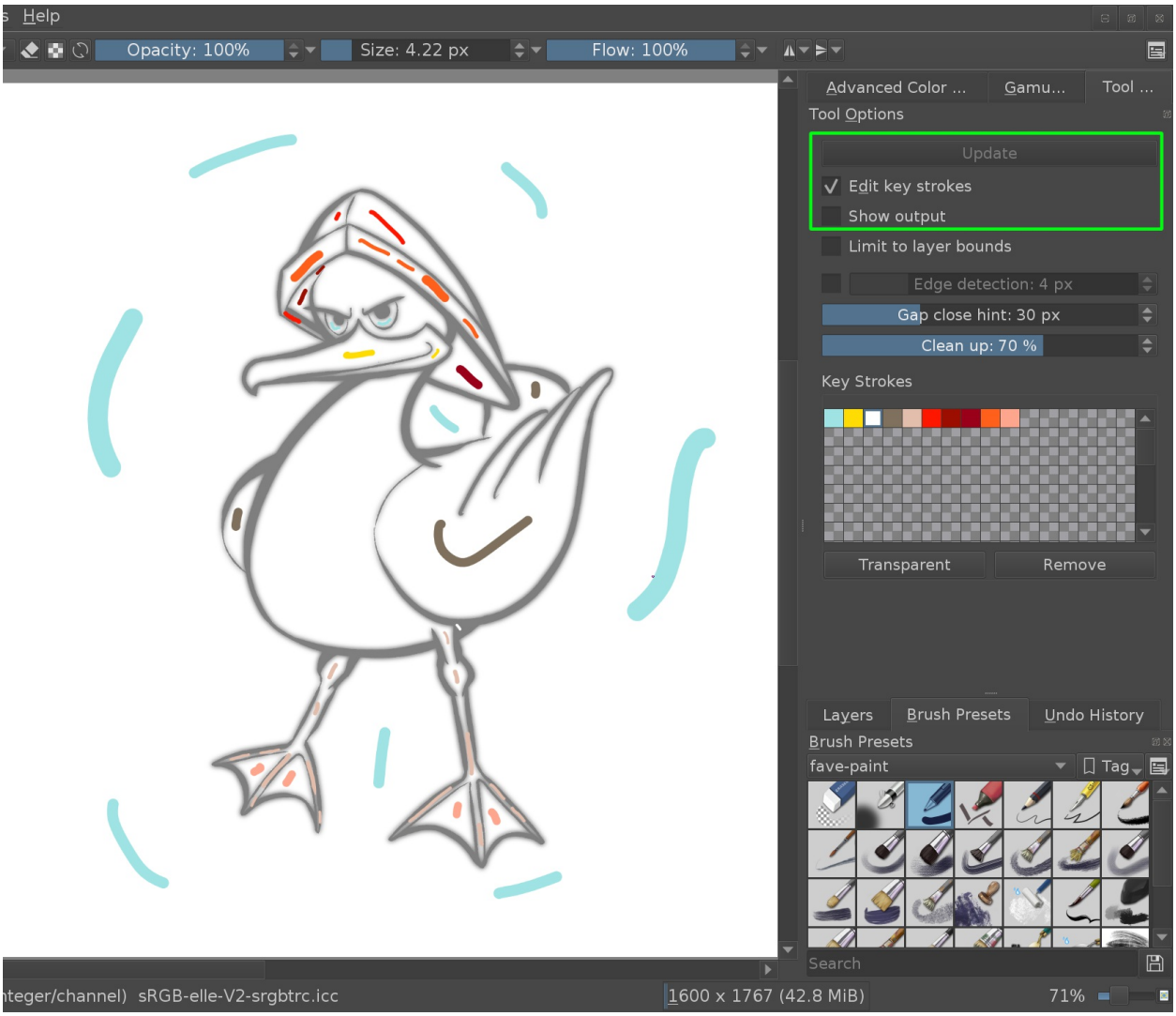
Choose a color from color selector and just click on the area you want to fill the color. As we have expanded the fill with grow selection the color will be filled slightly underneath the line art thus giving us a clean fill.

Colorize Mask

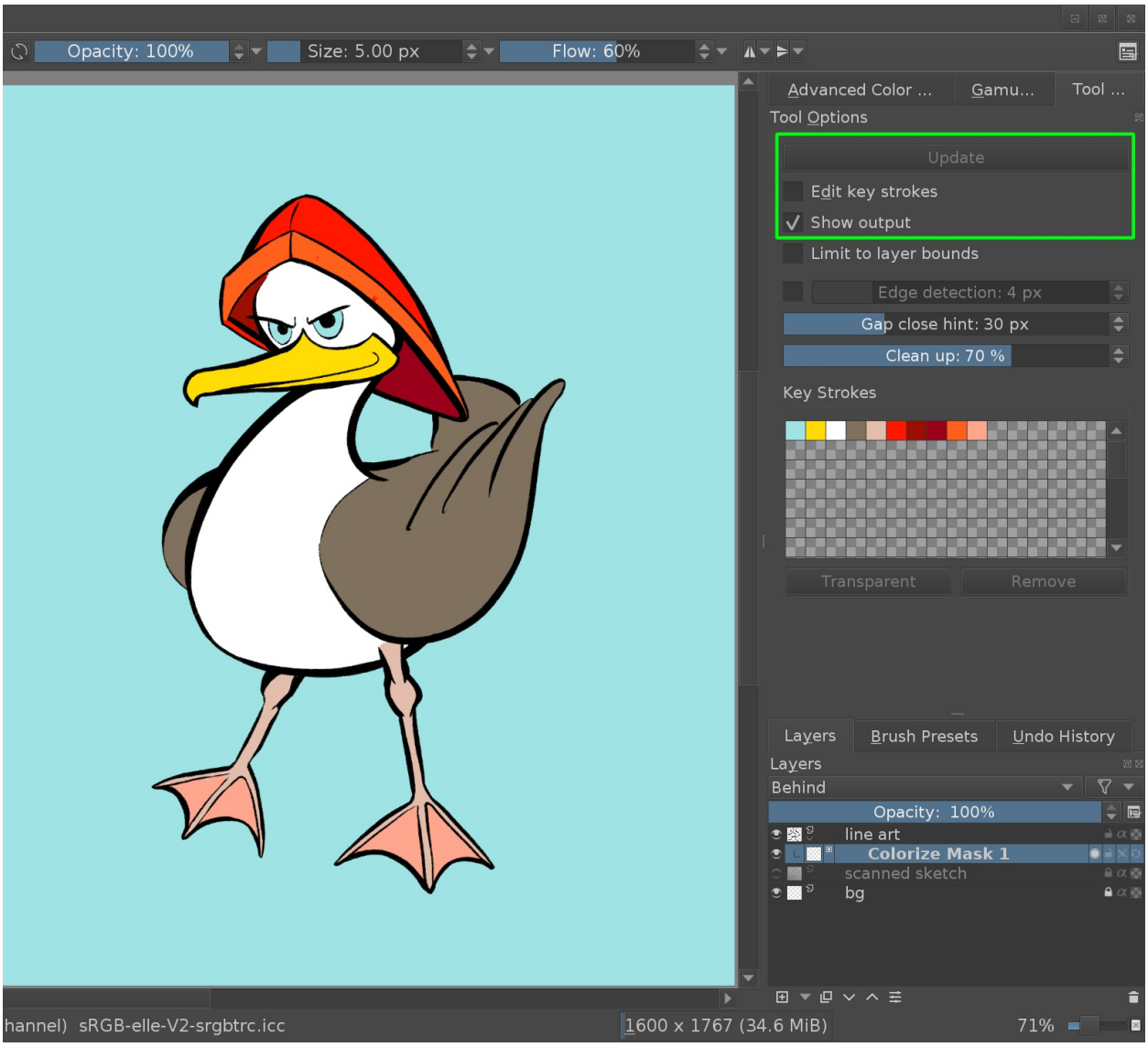
The third method is to take advantage of the built-in Colorize Mask. This is a powerful tool that can dramatically improve your workflow and cut you down on your production time. To begin coloring with the Colorize Mask, select your line art layer and click the *Colorize Mask Editing Tool* icon in the toolbar.



With the Colorize Mask Editing Tool enabled, click on the canvas—this will add a Colorize Mask layer to your document and make your lineart look a little blurry. You can now lay down solid brush strokes to indicate which areas should be colored in what colors:



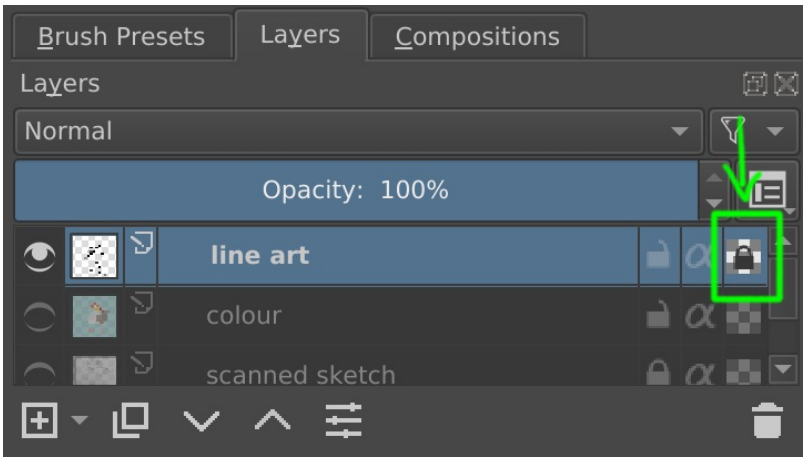
Whenever you press the *Update* button in the Tool Options, you will see which colors will fill which areas. You can continue to edit your brush strokes until you are happy with the result. To get a clean look of your painting, disable the “Edit key strokes” checkbox:



Once you are done, you can convert the Colorize Mask layer into a paint layer in the Layers docker. Have a look at the Colorize Mask manual to learn more about this tool.

Changing Line Art Color

To change the color of your line art, you can use the Alpha Lock feature. In the layer docker, click on the rightmost icon of your line art layer. It’s the icon that looks like a little checker board:



When Alpha Lock is enabled, you can only change the *color* of the pixels, not their opacity—meaning that everything you paint will only change the colors of your existing lines, not add new lines.

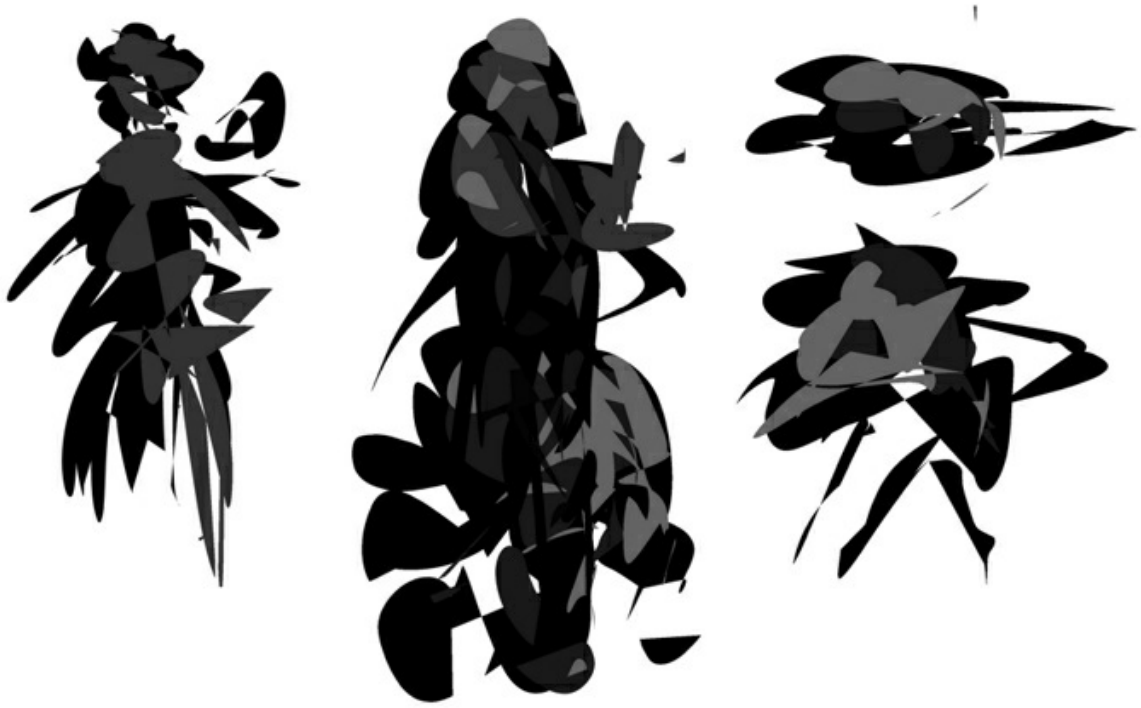
If you want to change the color of your line art to one solid color, you can now use the bucket fill tool and it will only apply to your existing lines. Or if you want to apply several different colors to specific areas of your line art, you can quickly paint over your line art with a broad brush:

Painting

Starting from chaos

Here, you start by making a mess through random shapes and texture, then taking inspirations from the resulting chaos you can form various concepts. It is kind of like making things from clouds or finding recognizable shapes of things in abstract and random textures. Many concept artists work with this technique.

You can use brushes like the shape brush, or the spray brush to paint a lot of different shapes, and from the resulting noise, you let your brain pick out shapes and compositions.



You then refine these shapes to look more like shapes you think they look, and paint them over with a normal paintbrush. This method is best done in a painting environment.

Starting from a value based underground

This method finds its origins in old oil-painting practice: You first make an

under-painting and then paint over it with color, having the dark underground shine through.

With Krita you can use blending modes for this purpose. Choosing the color blending mode on a layer on top allows you to change the colors of the image without changing the relative luminosity. This is useful, because humans are much more sensitive to tonal differences than the difference in saturation and hue. This'll allow you to work in grayscale before going into color for the polishing phase.

You can find more about this technique here [https://www.davidrevoy.com/article185/tutorial-getting-started-with-krita-1-3-bw-portrait].

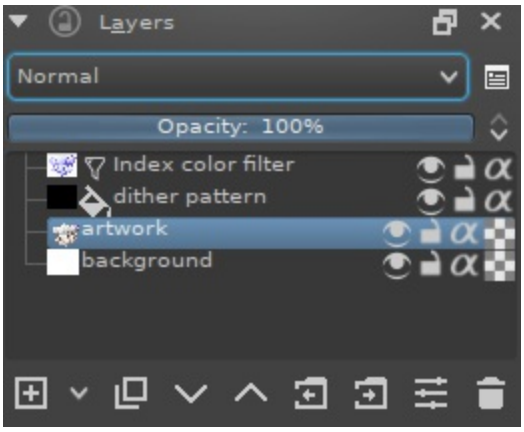
Preparing Tiles and Textures

Many artists use Krita to create textures for 3d assets used for games animation, etc. Krita has many texture templates for you to choose and get started with creating textures. These templates have common sizes, bit depth and color profiles that are used for texturing workflow.

Krita also has a real-time seamless tile mode to help texture artist prepare tiles and texture easily and check if it is seamless on the fly. The tiled mode is called wrap-around mode, to activate this mode got to *View* ▸ *Wrap Around Mode*. Now when you paint the canvas is tiled in real-time allowing you to create seamless pattern and texture, it is also easy to prepare interlocking patterns and motifs in this mode.

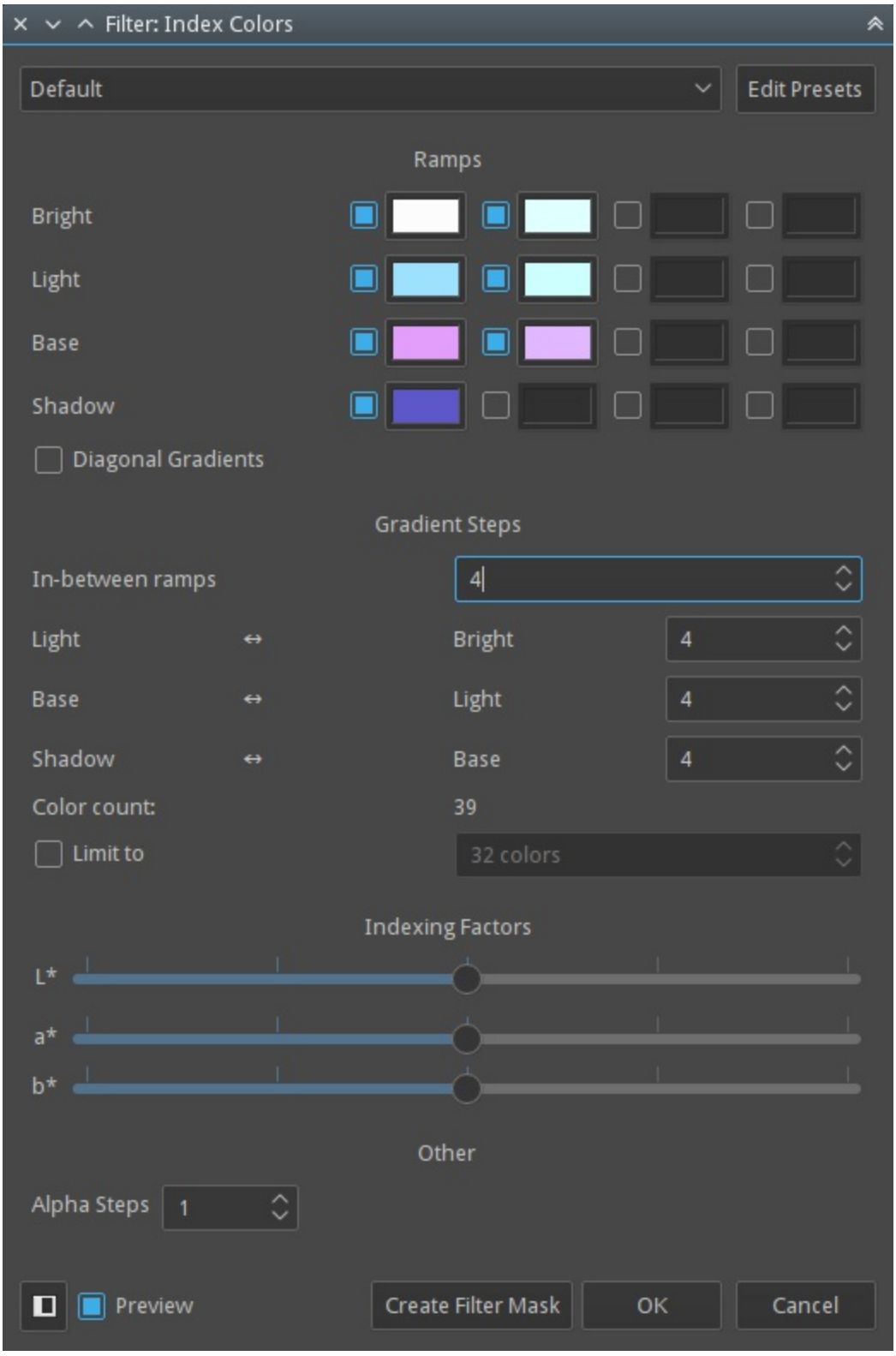
Creating Pixel Art

Krita can also be used to create a high definition pixel painting. The pixel art look can be achieved by using Index color filter layer and overlaying dithering patterns. The general layer stack arrangement is as shown below.



The index color filter maps specific user-selected colors to the grayscale value of the artwork. You can see the example below, the strip below the black and white gradient has an index color applied to it so that the black and white gradient gets the color selected to different values.

You can choose the required colors and ramps in the index color filter dialog as shown below.



Dithering can be used to enhance the look of the art and to ease the banding occurred by the index color filter. Krita has a variety of dithering patterns by

default, these can be found in pattern docker. You can use these patterns as fill layer, then set the blend mode to **overlay** and adjust the opacity according to your liking. Generally, an opacity range of 10% - 25% is ideal.

Paint the artwork in grayscale and add an index color filter layer at the top then add the dithering pattern fill layer below the index color filter but above the artwork layer, as shown in the layer stack arrangement above. You can paint or adjust the artwork at any stage as we have added the index color filter as a filter layer.

You can add different groups for different colors and add different dithering patterns for each group.

Below is an example painted with this layer arrangement.

An Example Setup for Using Krita with an Eye Tracker

Attention

This is not a reference document. It is based on the experiences of only one user. The information might not be as applicable when using different eye tracker devices or different control software.

Eye tracker devices are becoming more affordable and they are finding their way into more computer setups. Although these devices are used by various types of users, we will mainly focus on users who have physical disabilities and can only use their eyes to interact with the computer.

If you don’t already have experience with such a case, here are a few things you’ll need to know before you start:

- The eye tracker needs to be properly calibrated such that the pointer will be very close to the point where the user is looking at. This might be difficult to achieve, especially if the positioning of the eye tracker with respect to the user can not be fixed between different sessions.
- The lack of accuracy in control makes it nearly impossible to hit small areas on the screen such as small buttons or menu items. Corners and edges of the screen might be difficult to reach too. You also don’t want to put interface elements close to one another since it increases the chances of selecting the wrong element accidentally.
- Mouse operations like single click, double click, right click, drag and drop, etc. all demand extra effort in the form of switching modes in the program that controls the device. You will want to keep these switches to a minimum so that the work will not be interrupted frequently.
- Switching the mode doesn’t automatically start the operation. You need an extra action for that. In our case, this action is “dwelling”. For example, to start a program, you switch to the left double click mode

and then dwell on the icon for the application to activate the double click. Adjusting the dwell time is an important tradeoff: shorter dwell times allow for faster work but are also more error-prone.

Requirements

Besides the *obvious* requirement of having an eye tracker device, you will also need a control program that will let you interact with the device. When you obtain the device, such a program will most probably be provided to you but that program might not be sufficient for using the device with Krita.

One of the basic functionalities of these programs is to emulate mouse clicks. In our case, the program provides a hovering menu which includes large buttons for switching modes between left/right mouse buttons and single/double clicks. After selecting the mode, the hovering menu can be collapsed so that it will leave more screen space for the application.

In order to make them easier to configure and use, some programs include only basic modes like single clicks. This is sufficient for many popular applications like e-mail agents and browsers, but for Krita you need the drag and drop mode to be able to draw. If the provided control software doesn’t support this mode (usually called “mouse emulation”), you can contact the manufacturer of the device for assistance, or look for open source options.

Starting Krita

Basically, setting the control program to left double click mode and dwelling on the Krita icon on the desktop would be enough to start up Krita but there are some issues with this:

- On startup, Krita asks you to choose a template. It’s likely that you don’t want to go through this setting every time and just want to start with a blank template.
- Later, saving the document will require interacting with the file save dialog which is not very friendly for this type of use.

A workaround for these issues could be creating and saving a blank template

and running a script that will copy this template under a new name and send it to Krita. Here's an example script for Windows which uses a timestamp suffix to make sure that each file will have a different name (replace USERNAME with the actual user name):

```
@echo off
for /f "tokens=2 delims==" %%a in ('wmic OS Get localdatetime /va
set "YY=%dt:~2,2%" & set "YYYY=%dt:~0,4%" & set "MM=%dt:~4,2%" &
set "HH=%dt:~8,2%" & set "Min=%dt:~10,2%" & set "Sec=%dt:~12,2%"
set "datestamp=%YYYY%%MM%%DD%" & set "timestamp=%HH%%Min%%Sec%"
set "fullstamp=%YYYY%-%MM%-%DD%_%HH%-%Min%-%Sec%"
set filename=USERNAME_%fullstamp%.kra
copy "C:\Users\USERNAME\Pictures\blank.kra" "%filename%"
start "C:\Program Files\Krita (x64)\bin\krita.exe" "%filename%"
```

Double clicking on this script will create a new Krita file in the same folder as the script file. Since the file already has a name, the file save dialog will be avoided. Combined with autosaving, this can be an efficient way to save your work.

Tip

Storing these files directly on a cloud storage service will be even safer.

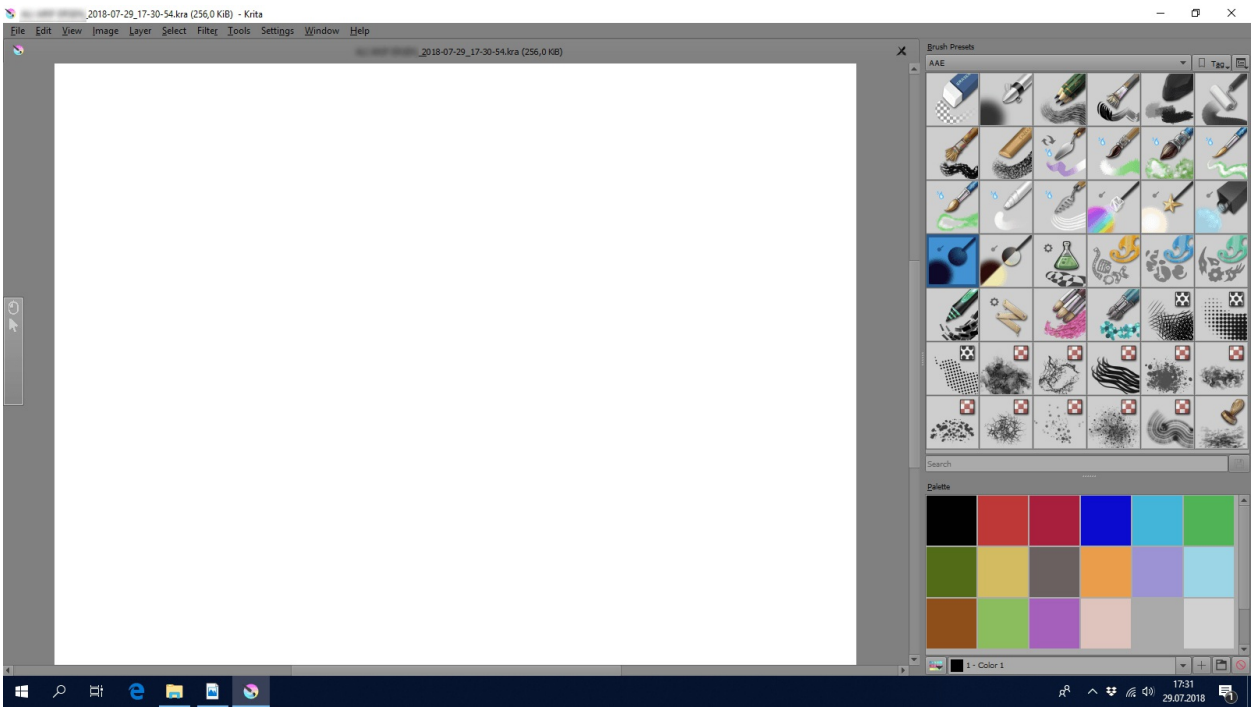
You might also deal with some timing issues when starting Krita:

- After the icon for Krita or for the script is double clicked and Krita starts loading, lingering on the icon will start a second instance.
- Similarly, after double clicking, if another window is accidentally brought to the foreground, Krita might start up partially visible behind that window.

To prevent these problems, it will help if the users train themselves to look at some harmless spot (like an empty space on the desktop) until Krita is loaded.

Layout

Since the interface elements need to be large, you have to use the screen area economically. Running in full-screen mode and getting rid of unused menus and toolbars are the first steps that you can take. Here's the screenshot of our layout:



You will want to put everything you need somewhere you can easily access. For our drawings, the essential items are brushes and colors. So we've decided to place permanent dockers for these.

Krita features many brushes but the docker has to contain a limited number of those so that the brush icons can be large enough. We recommend that you create a custom brush preset to your own liking.

There are various tools for selecting color but most of them are not easily usable since they require quite a high level of mouse control. The Python Palette Docker is the simplest to use where you select from a set of predefined colors, similar to brush presets. Again, similarly to brush selection, it will help to create a custom set of favorite colors.

Once you are happy with your layout, another feature that will help you is to lock the dockers. It's possible to accidentally close or move dockers. For

example, in drag and drop mode you can accidentally grab a docker and drag it across the screen. To prevent this, put the following setting in the `kritarc` file:

```
LockAllDockerPanels=true
```

(Check the Krita FAQ for how to find the configuration kritarc file on your system.)

If you're using a hovering mouse control menu like we do, you also have to figure out where to place it when it's collapsed. Put it somewhere where it will be easily accessible but where it will not interfere with Krita. On the screenshot you can see it at the left edge of the screen.

Summary

In summary, we work as explained below.

To start Krita:

1. On the desktop, pull up the hovering mouse menu and select left double click mode.
2. Double click on the new drawing creation script. Look away at some harmless spot until Krita loads.

Drawing with Krita:

1. Switch to left single click mode.
2. Select a brush and/or color using the dockers.
3. Switch to drag and drop mode. You're ready to draw.
4. Go to the point where you want to start a stroke and dwell until dragging starts (this emulates pressing and holding your finger on the mouse button).
5. Draw.
6. When you want to finish the current stroke, dwell at the ending point until you get out of dragging (this emulates lifting your finger from the mouse button).
7. Repeat the whole process.

Finishing:

1. Switch to left single click mode.
2. Click on the button for closing the window.
3. When warned about unsaved changes, click the button for saving the file.

Flat Coloring

So you've got a cool black on white drawing, and now you want to color it! The thing we'll aim for in this tutorial is to get your line art colored in with flat colors. So no shading just yet. We'll be going through some techniques for preparing the line art, and we'll be using the layer docker to put each color on a separate layer, so we can easily access each color when we add shading.

Note

This tutorial is adapted from this tutorial [http://theratutorial.tumblr.com/post/66584924501/flat-colouring-in-the-kingdom-of-2d-layers-are] by the original author.

Understanding Layers

To fill line art comfortably, it's best to take advantage of the layerstack. The layer stack is pretty awesome, and it's one of those features that make digital art super-convenient.

In traditional art, it is not uncommon to first draw the full background before drawing the subject. Or to first draw a line art and then color it in. Computers have a similar way of working.

In programming, if you tell a computer to draw a red circle, and then afterwards tell it to draw a smaller yellow circle, you will see the small yellow circle overlap the red circle. Switch the commands around, and you will not see the yellow circle at all: it was drawn before the red circle and thus 'behind' it.

This is referred to as the "drawing order". So like the traditional artist, the computer will first draw the images that are behind everything, and layer the subject and foreground on top of it. The layer docker is a way for you to

control the drawing order of multiple images, so for example, you can have your line art drawn later than your colors, meaning that the lines will be drawn over the colors, making it easier to make it neat!

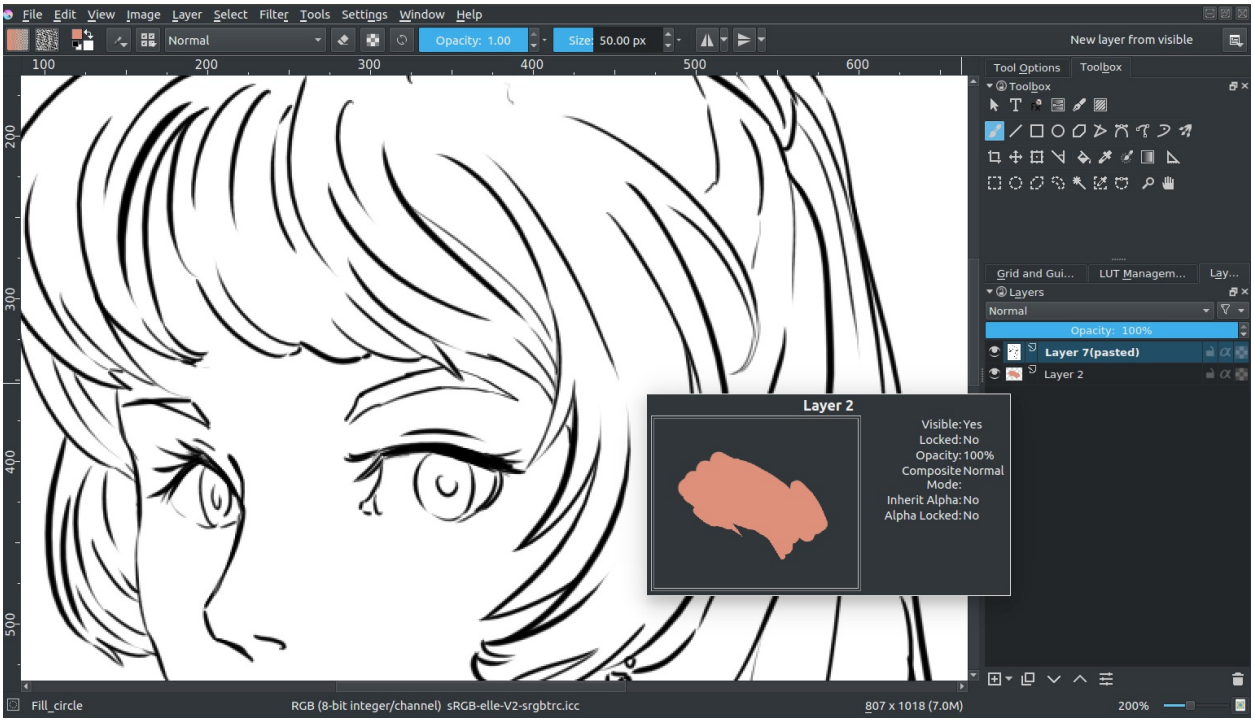
Other things that a layer stack can do are blending the colors of different layers differently with blending modes, using a filter in the layer stack, or using a mask that allows you to make parts transparent.

Tip

Programmers talk about transparency as ‘’Alpha’’, which is because the ‘a’ symbol is used to present transparency in the algorithms for painting one color on top of another. Usually when you see the word ‘’Alpha’’ in a graphics program, just think of it as affecting the transparency.

Preparing your line art

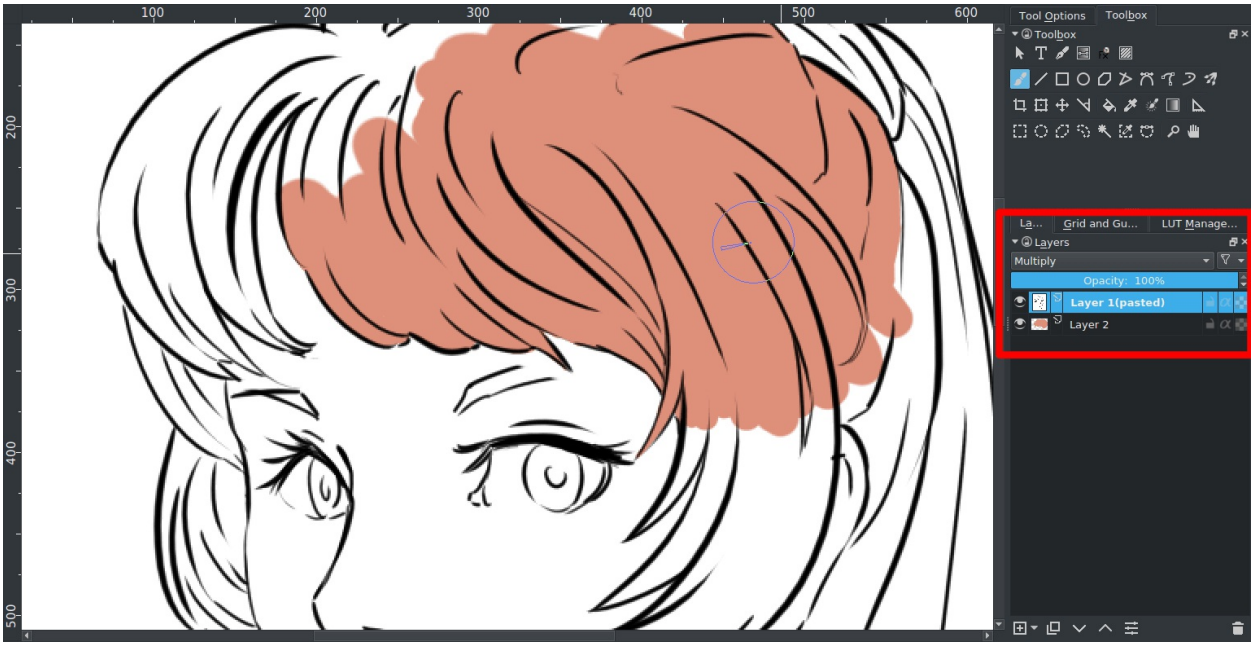
Put the new layer underneath the layer containing the line art (drag and drop or use the up/down arrows for that), and draw on it.



…And notice nothing happening. This is because the white isn't transparent. You wouldn't really want it to either, how else would you make convincing highlights? So what we first need to do to color in our drawing is prepare our line art. There's several methods of doing so, each with varying qualities.

The Multiply Blending Mode

So, typically, to get a black and white line art usable for coloring, you can set the blending mode of the line art layer to Multiply. You do this by selecting the layer and going to the drop-down that says **Normal** and setting that to **Multiply**.



And then you should be able to see your colors!

Multiply is not a perfect solution however. For example, if through some image editing magic I make the line art blue, it results into this:

This is because multiply literally multiplies the colors. So it uses maths!

What it first does is take the values of the RGB channels, then divides them by the max (because we're in 8bit, this is 255), a process we call normalising. Then it multiplies the normalized values. Finally, it takes the result and multiplies it with 255 again to get the result values.

	Pink	**Pink (normalized)**	**Blue**	**Blue (normalized)**	**Normalized, multiplied**	**Result**
Red	222	0.8705	92	0.3607	0.3139	80
Green	144	0.5647	176	0.6902	0.3897	99
Blue	123	0.4823	215	0.8431	0.4066	103

This isn’t completely undesirable, and a lot of artists use this effect to add a little richness to their colors.

Advantages

Easy, can work to your benefit even with colored lines by softening the look of the lines while keeping nice contrast.

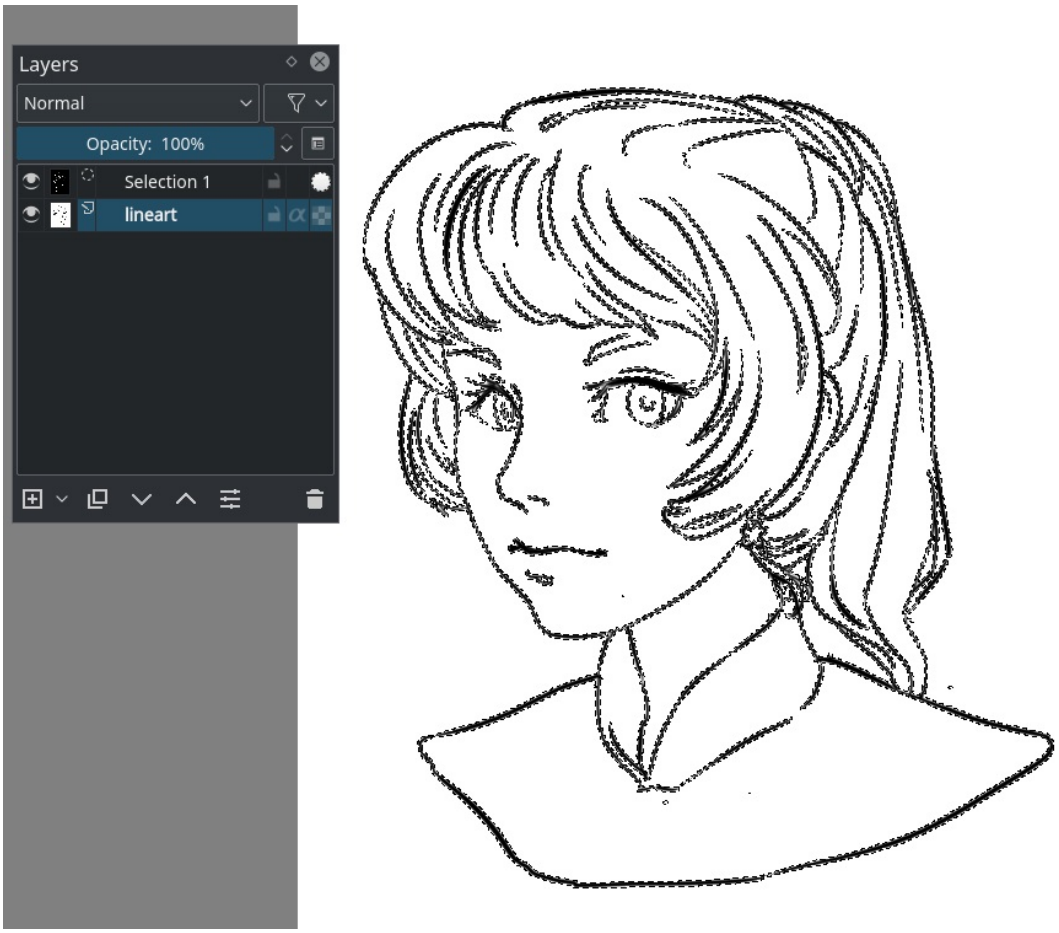
Disadvantages

Not actually transparent. Is a little funny with colored lines.

Using Selections

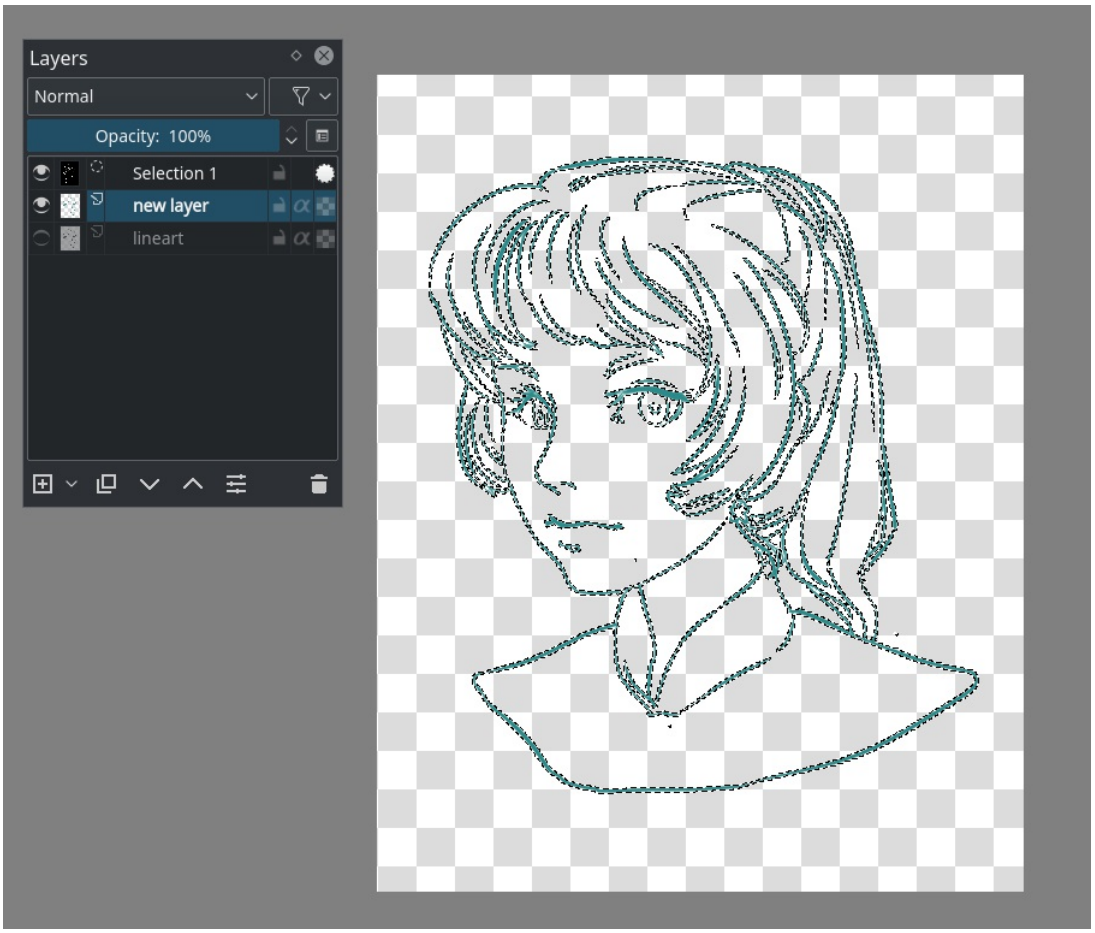
The second method is one where we’ll make it actually transparent. In other programs this would be done via the channel docker, but Krita doesn’t do custom channels, instead it uses Selection Masks to store custom selections.

1. Duplicate your line art layer.
2. Convert the duplicate to a selection mask.  the layer, then *Convert ▸ to Selection Mask.*



3. Invert the selection mask. *Select* ▸ *Invert Selection*.

4. Make a new layer, and do *Edit* ▸ *Fill with Foreground Color*.



And you should now have the line art on a separate layer.

Advantages

Actual transparency.

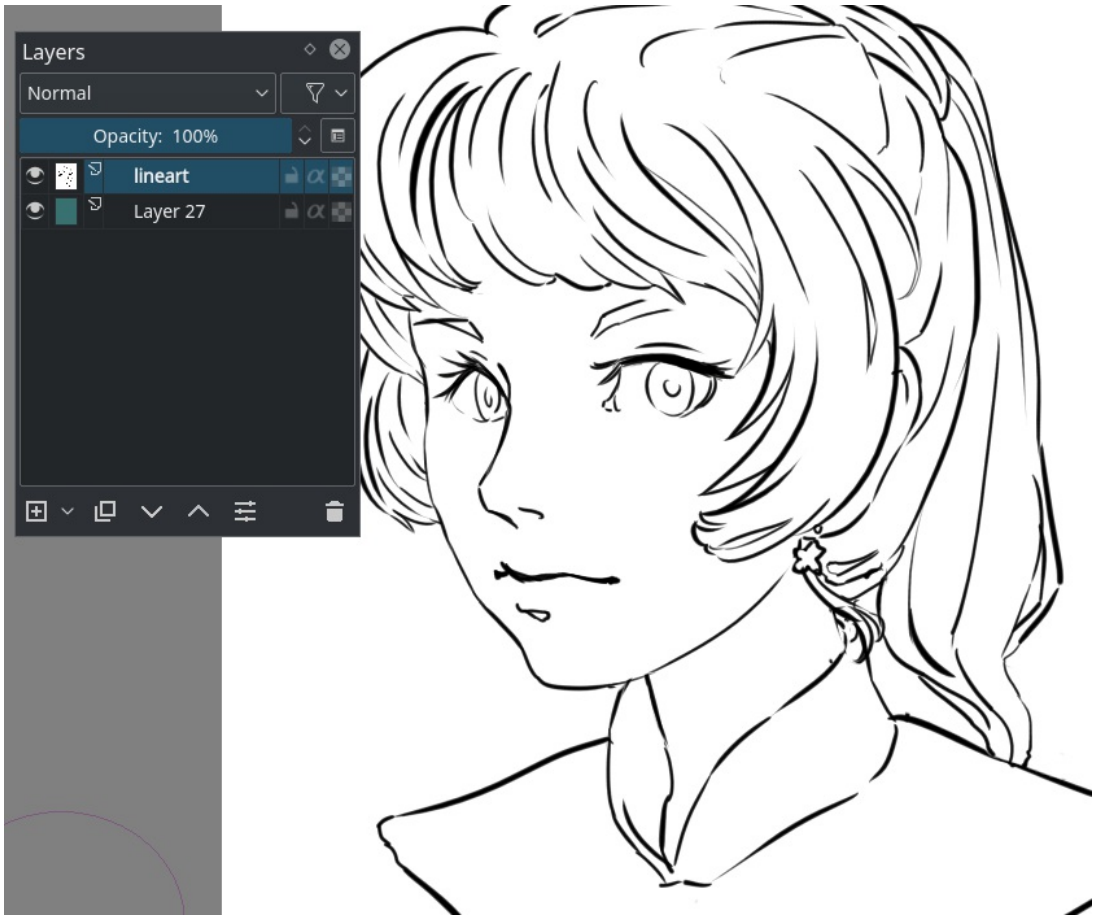
Disadvantages

Doesn’t work when the line art is colored.

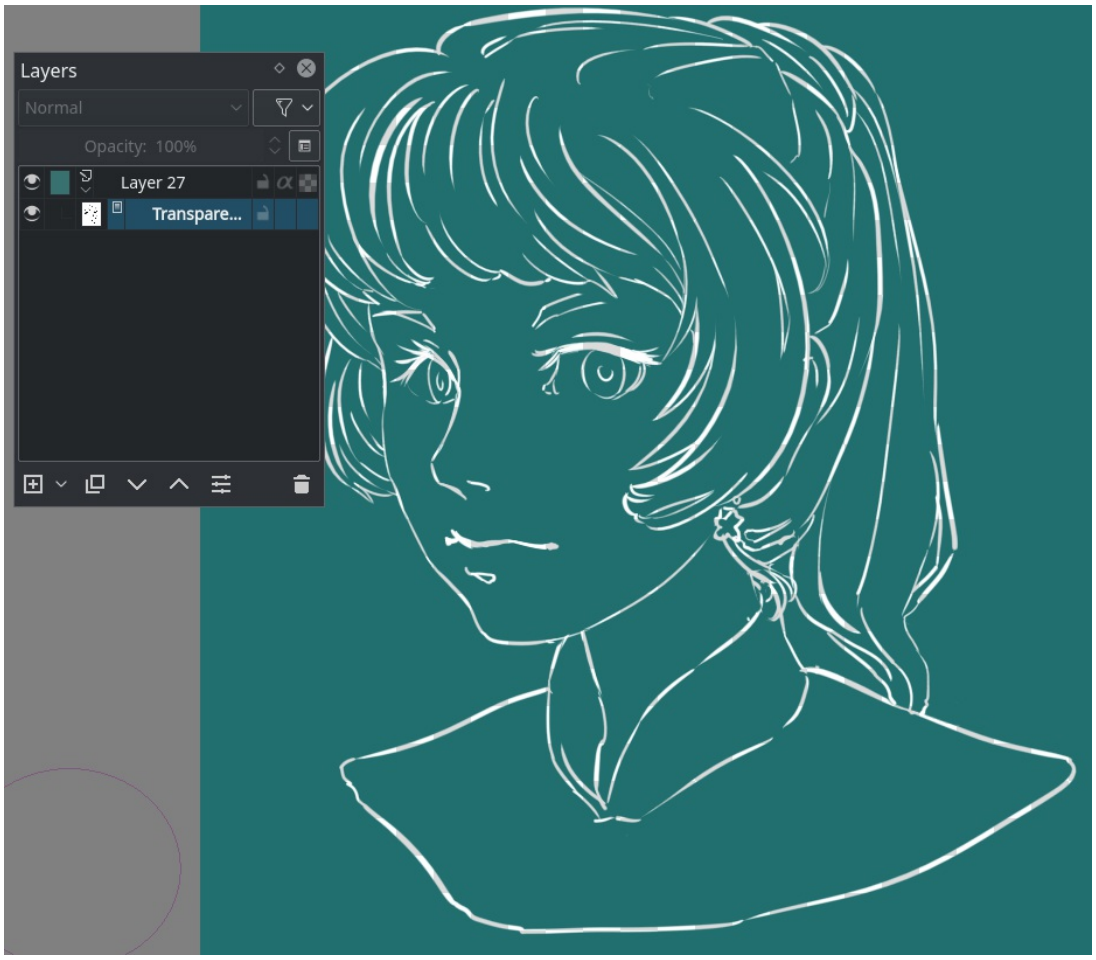
Using Masks

This is a simpler variation of the above.

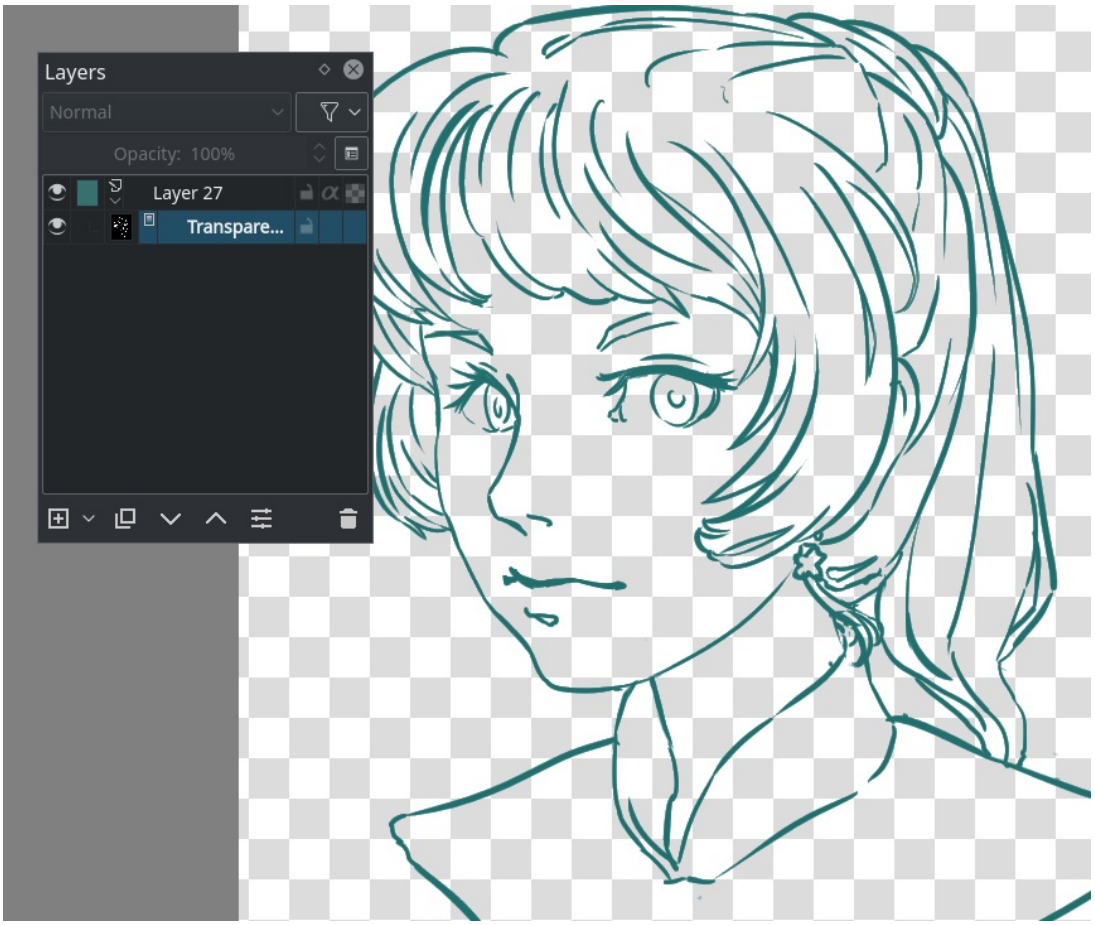
1. Make a filled layer underneath the line art layer.



2. Convert the line art layer to a transparency mask  the layer, then *Convert ▸ to Transparency Mask.*



3. Invert the transparency mask by going to *Filter* ▸ *Adjust* ▸ *Invert*.



Advantages

Actual transparency. You can also very easily doodle a pattern on the filled layer where the mask is on without affecting the transparency.

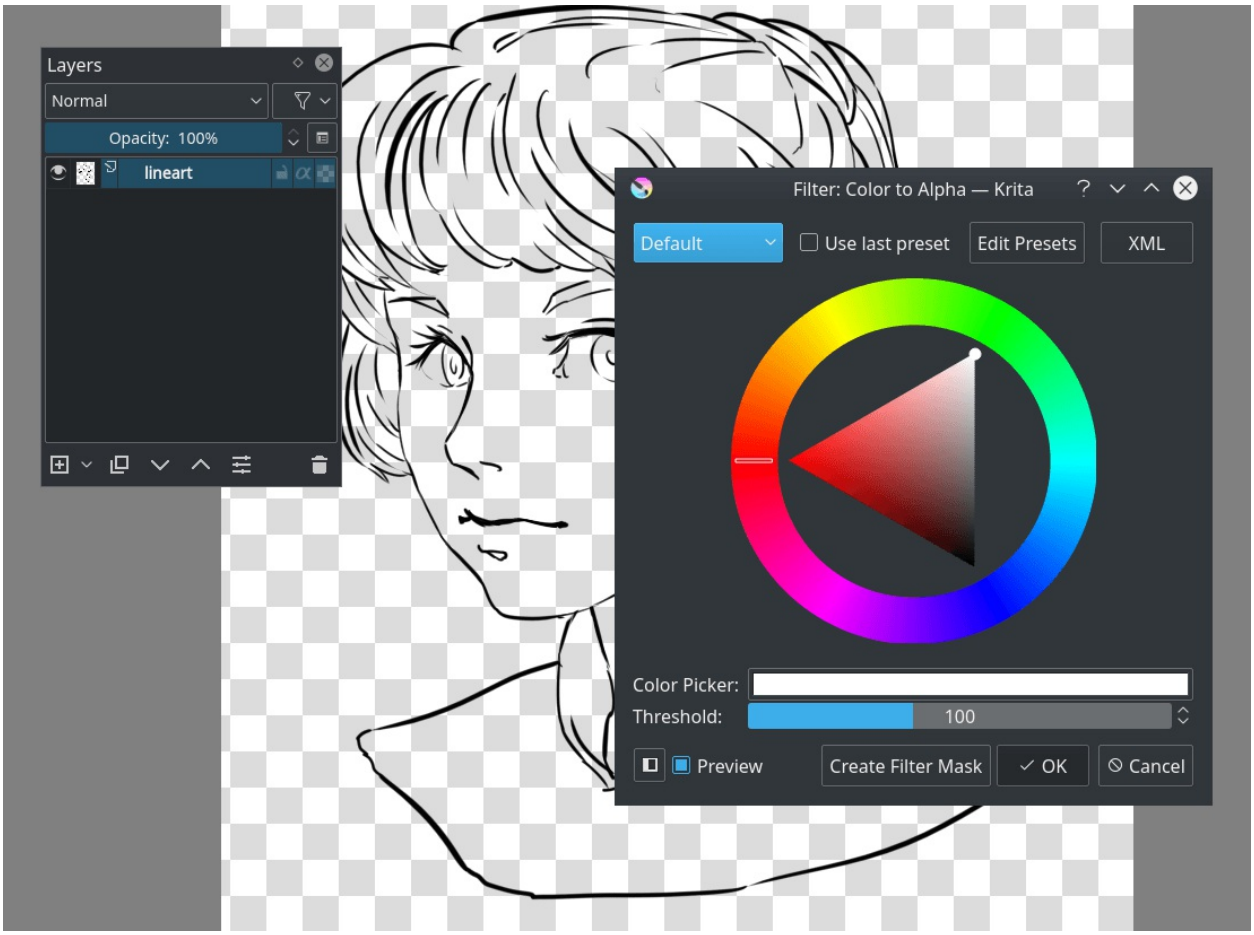
Disadvantages

Doesn’t work when the line art is colored already. We can still get faster.

Using Color to Alpha

By far the fastest way to get transparent line art.

1. Select the line art layer and apply the *Filter: Color to Alpha* dialog under *Filters* ▸ *Colors* ▸ *Color to Alpha…* menu item. The default values should be sufficient for line art.



Advantages

Actual transparency. Works with colored line art as well, because it removes the white specifically.

Disadvantages

You'll have to lock the layer transparency or separate out the alpha via the right-click menu if you want to easily color it.

Coloring the image

Much like preparing the line art, there are many different ways of coloring a layer.

You could for example fill in everything by hand, but while that is very precise it also takes a lot of work. Let’s take a look at the other options, shall we?

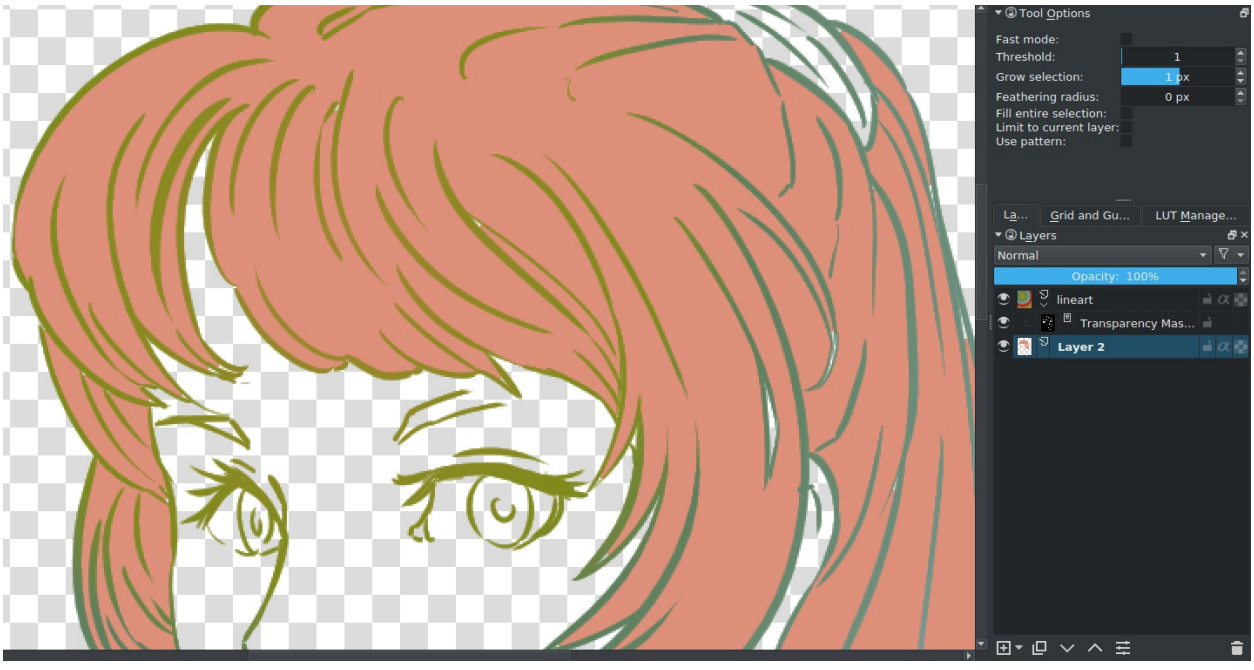
Fill Tool

In most cases the fill-tool can’t deal with the anti-aliasing (the soft edge in your line art to make it more smooth when zoomed out) In Krita you have the grow-shrink option. Setting that to say… 2 expands the color two pixels.

Threshold decides when the fill-tool should consider a different color pixel to be a border. And the feathering adds an extra soft border to the fill.

Now, if you click on a gapless-part of the image with your preferred color… (Remember to set the opacity to 1.0!)

Depending on your line art, you can do flats pretty quickly. But setting the threshold low can result in little artifacts around where lines meet:



However, setting the threshold high can end with the fill not recognizing

some of the lighter lines. Besides these little artifacts can be removed with the brush easily.

Advantages

Pretty darn quick depending on the available settings.

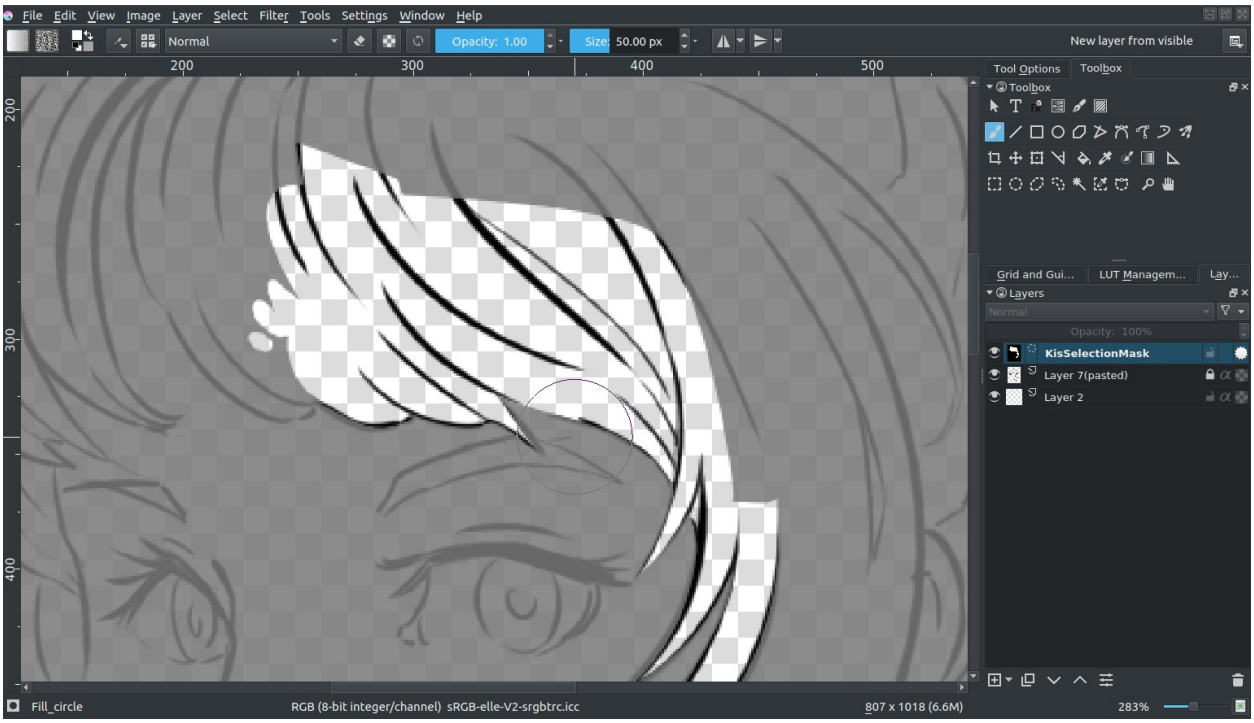
Disadvantages

Again, not great with gaps or details. And it works best with aliased line art.

Selections

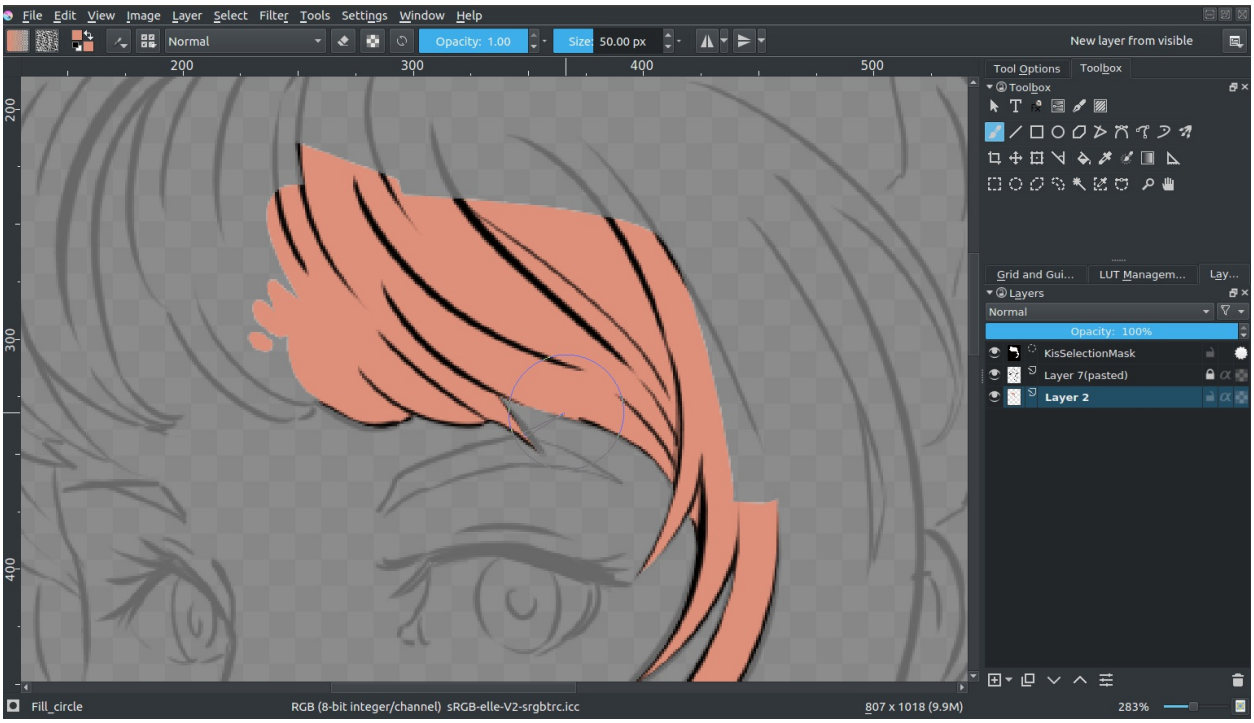
Selections work using the selection tools.

For example with the Path Selection Tool you can easily select a curved area, and the with `Shift` +  (not  + `Shift`, there's a difference!) you can easily add to an existing selection.



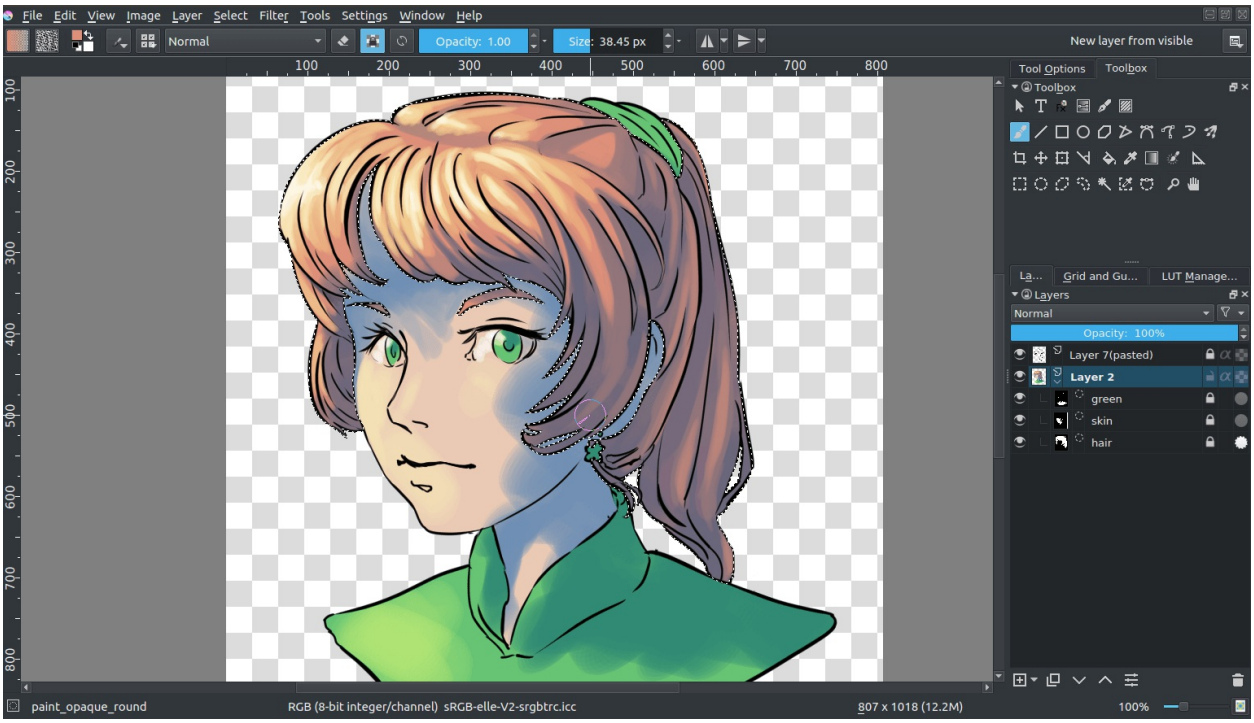
You can also edit the selection if you have *Select* ▸ *Show Global Selection Mask* turned on. Then you can select the global selection mask, and paint on it. (Above with the alternative selection mode, activated in the lower-left corner of the stats bar)

When done, select the color you want to fill it with and press the `Shift + Backspace` shortcut.



You can save selections in selection masks by  a layer, and then going to *Add* ▸ *Local Selection*. You first need to deactivate a selection by pressing the circle before adding a new selection.

This can serve as an alternative way to split out different parts of the image, which is good for more painterly pieces:



Advantages

A bit more precise than filling.

Disadvantages

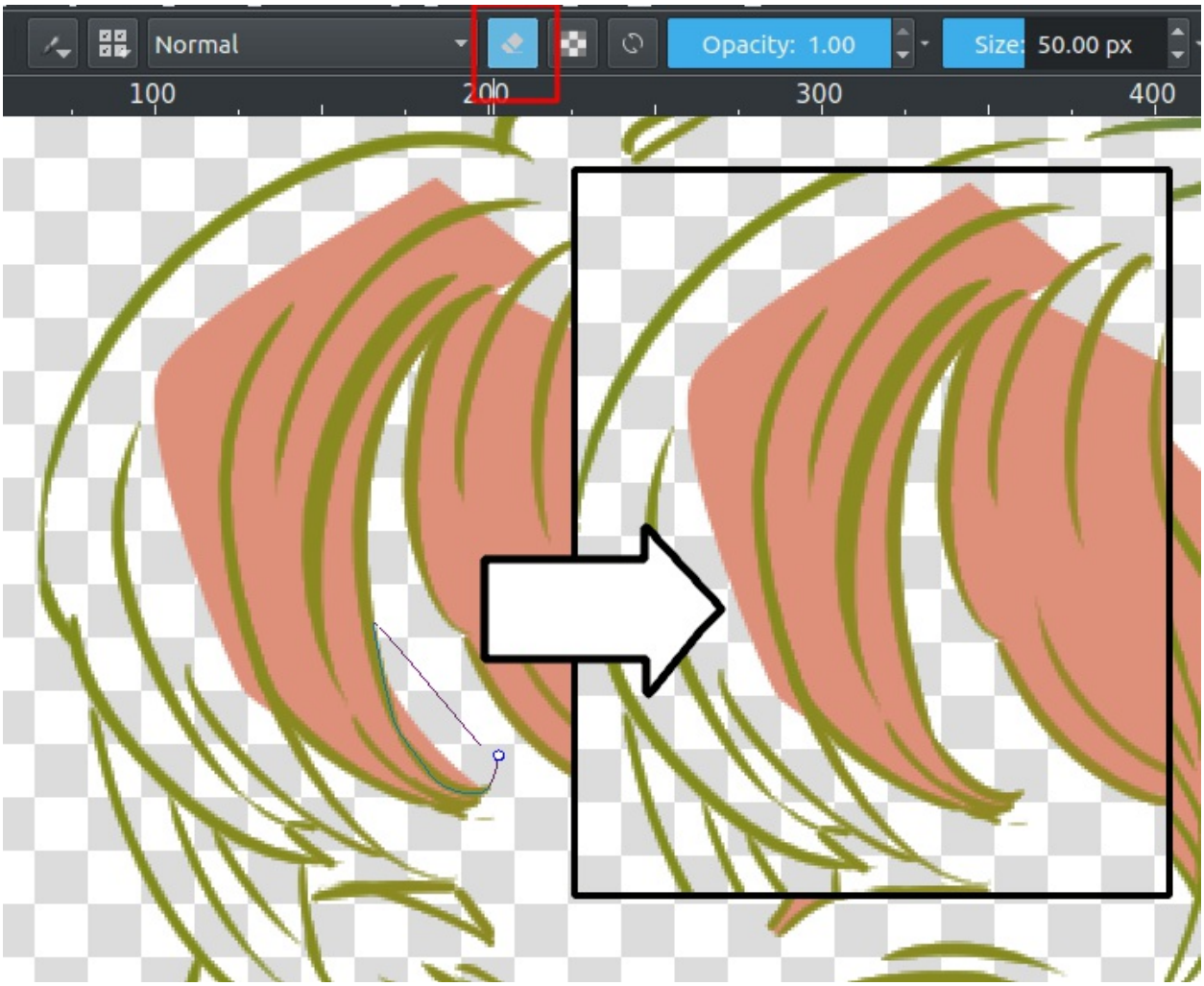
Previewing your color isn’t as easy.

Geometric tools

So you have a tool for making rectangles or circles. And in the case of Krita, a tool for bezier curves. Select the path tool (), and set the tool options to fill=foreground and outline=none. Make sure that your opacity is set to 1.00 (fully opaque).

By clicking and holding, you can influence how curvy a line draw with the path tool is going to be. Letting go of the mouse button confirms the action, and then you’re free to draw the next point.

You can also erase with a geometric tool. Just press the E key or the eraser button.



Advantages

Quicker than using the brush or selections. Also decent with line art that contains gaps.

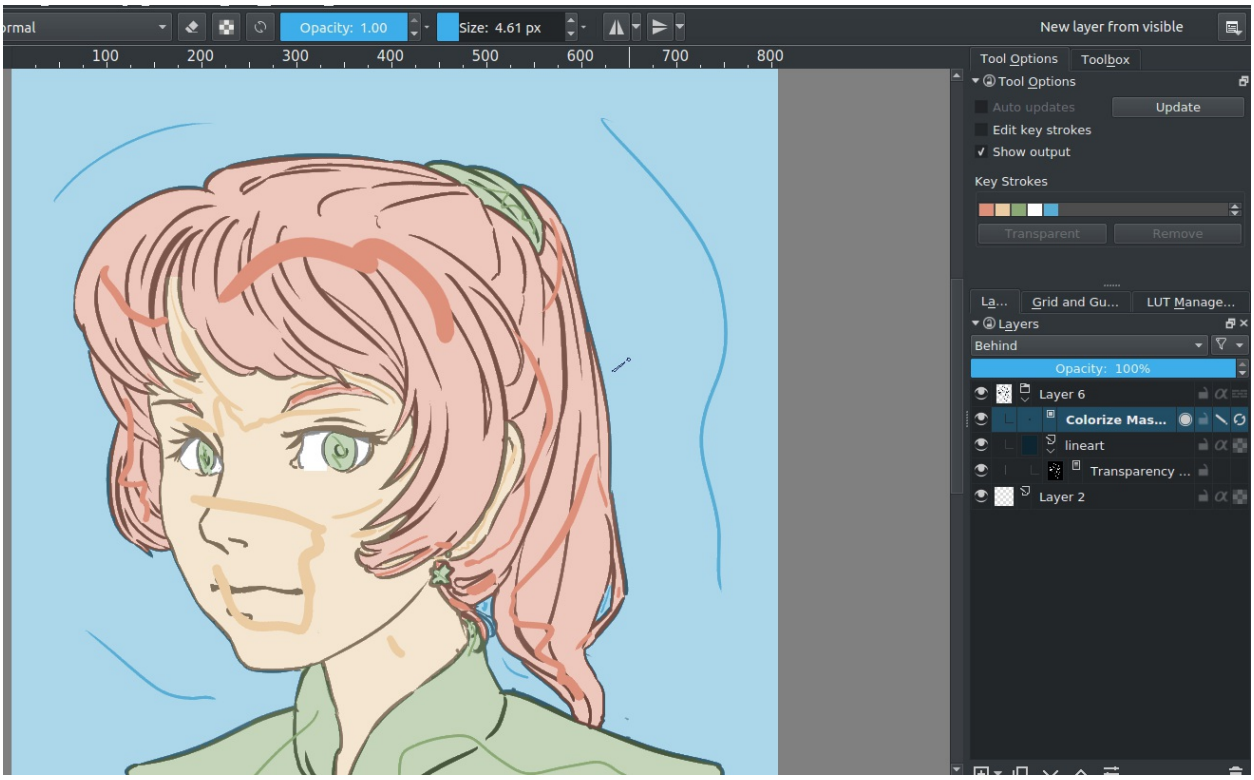
Disadvantages

Fiddly details aren’t easy to fill in with this. So I recommend skipping those and filling them in later with a brush.

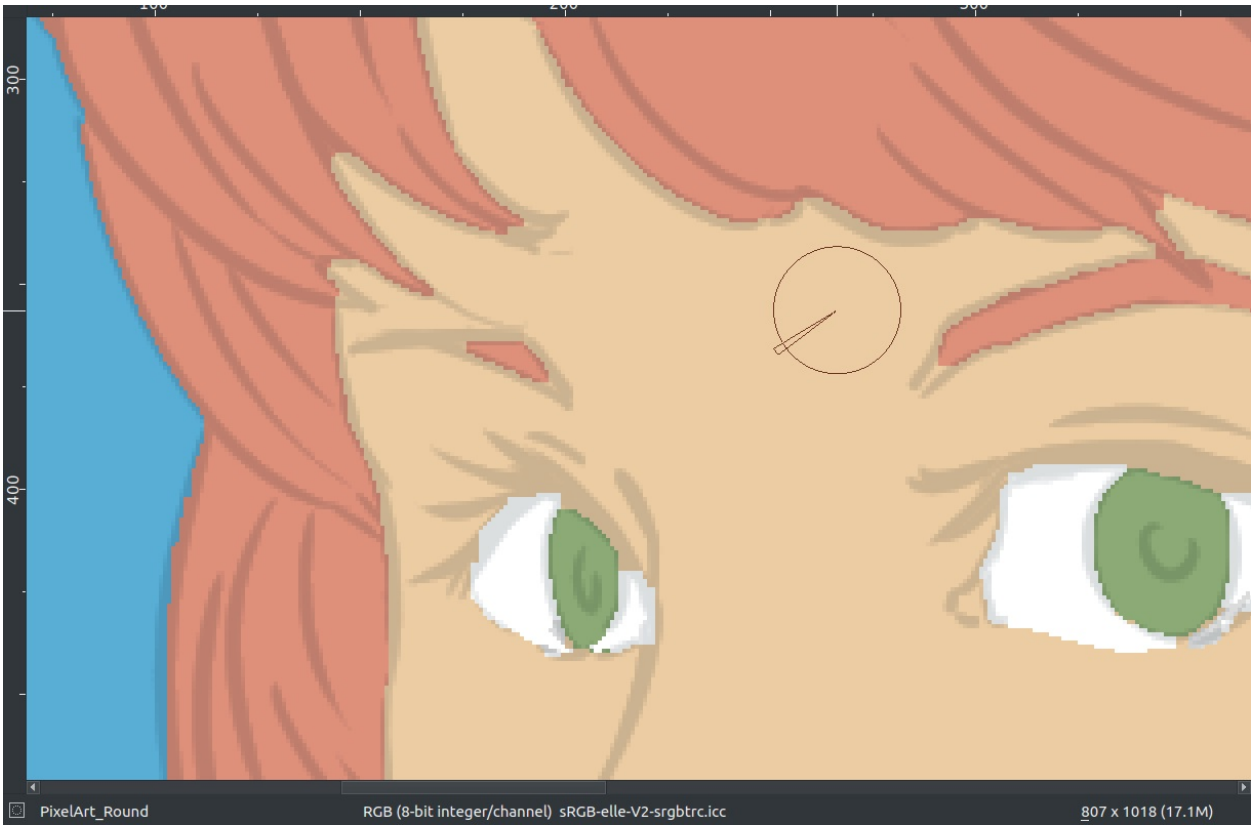
Colorize Mask

So it works like this:

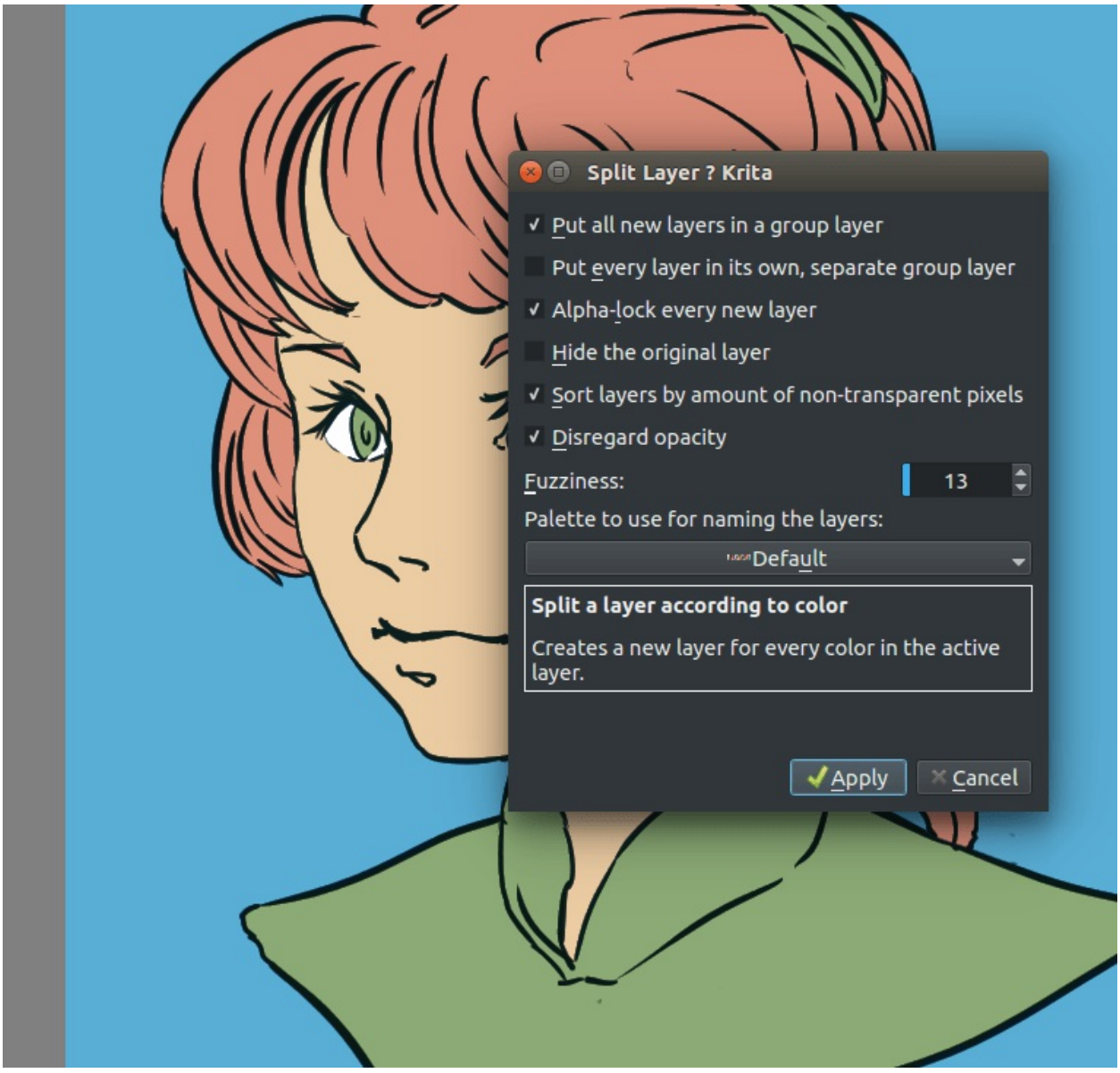
1. Select the colorize mask tool.
2. Tick the layer you're using.
3. Paint the colors you want to use on the colorize mask.
4. Click update to see the results:



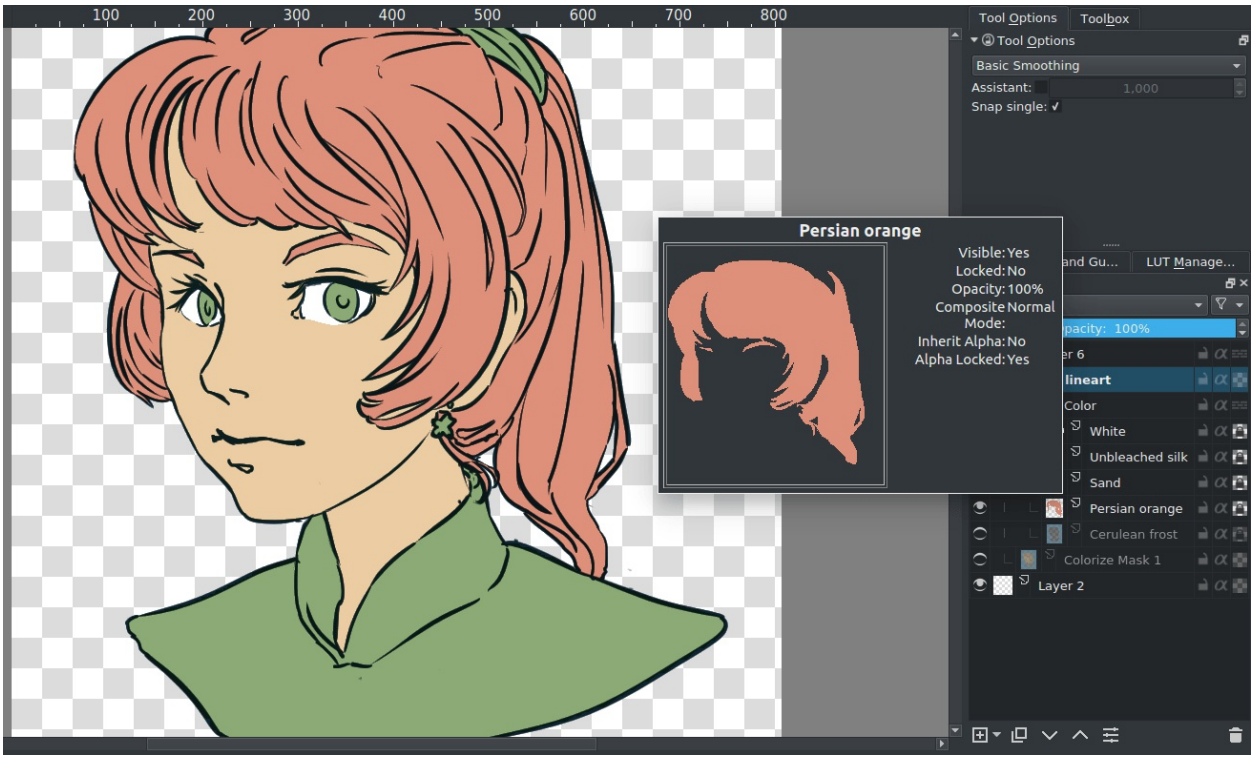
When you are satisfied,  the colorize mask, and go to *Convert* ▸ *Paint Layer*. This will turn the colorize mask to a generic paint layer. Then, you can fix the last issues by making the line art semi-transparent and painting the flaws away with a pixel art brush.



Then, when you are done, split the layers via *Layer* ▸ *Split* ▸ *Split Layer*. There are a few options you can choose, but the following should be fine:



Finally, press **Ok** and you should get the following. Each color patch it on a different layer, named by the palette in the menu and alpha locked, so you can start painting right away!



Advantages

Works with anti-aliased line art. Really quick to get the base work done. Can auto-close gaps.

Disadvantages

No anti-aliasing of its own. You have to choose between getting details right or the gaps auto-closed.

Conclusion

I hope this has given you a good idea of how to fill in flats using the various techniques, as well as getting a hand of different Krita features. Remember that a good flat filled line art is better than a badly shaded one, so keep practicing to get the best out of these techniques!

Inking

The first thing to realize about inking is that unlike anatomy, perspective, composition or color theory, you cannot compensate for lack of practice with study or reasoning. This is because all the magic in drawing lines happens from your shoulder to your fingers, very little of it happens in your head, and your lines improve with practice.

On the other hand, this can be a blessing. You don’t need to worry about whether you are smart enough, or are creative enough to be a good inker. Just dedicated. Doubtlessly, inking is the Hufflepuff of drawing disciplines.

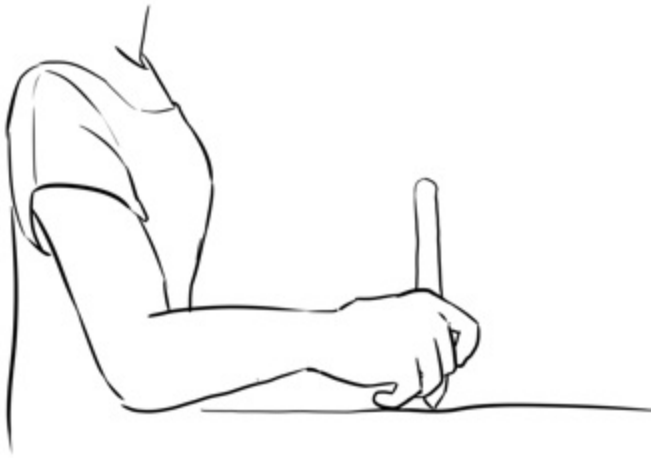
That said, there are a few tips to make life easy:

Pose

Notice how I mentioned up there that the magic happens between your shoulders and fingers? A bit weird, not? But perhaps, you have heard of people talking about adopting a different pose for drawing.

You can in fact, make different strokes depending on which muscles and joints you use to make the movement: The Fingers, the wrist and lower-arm muscles, the elbow and upper-arm muscles or the shoulder and back muscles.

Generally, the lower down the arm the easier it is to make precise strokes, but also the less durable the joints are for long term use. We tend to start off using our fingers and wrist a lot during drawing, because it’s easier to be precise this way. But it’s difficult to make long strokes, and furthermore, your fingers and wrist get tired far quicker.



Your shoulders and elbows on the other hand are actually quite good at handling stress, and if you use your whole hand you will be able to make long strokes far more easily. People who do calligraphy need shoulder based strokes to make those lovely flourishes (personally, I can recommend improving your handwriting as a way to improve inking), and train their arms so they can do both big and small strokes with the full arm.

To control pressure in this state effectively, you should press your pinky against the tablet surface as you make your stroke. This will allow you to precisely judge how far the pen is removed from the tablet surface while

leaving the position up to your shoulders. The pressure should then be put by your elbow.

So, there are not any secret rules to inking, but if there is one, it would be the following: *The longer your stroke, the more of your arms you need to use to make the stroke*.

Stroke smoothing

So, if the above is the secret to drawing long strokes, that would be why people having been inking lovely drawings for years without any smoothing? Then, surely, it is decadence to use something like stroke smoothing, a short-cut for the lazy?

Example of how a rigger brush can smooth the original movement (here in red)

Not really. To both, actually. Inkers have had a real-life tool that made it easier to ink, it’s called a rigger-brush, which is a brush with very long hairs. Due to this length it sorta smooths out shakiness, and thus a favoured brush when inking at three in the morning.

With some tablet brands, the position events being sent aren't very precise, which is why we having basic smoothing to apply the tiniest bit of smoothing on tablet strokes.

On the other hand, doing too much smoothing during the whole drawing can make your strokes very mechanical in the worst way. Having no jitter or tiny bumps removes certain humanity from your drawings, and it can make it impossible to represent fabric properly.

Therefore, it's wise to train your inking hand, yet not to be too hard on yourself and refuse to use smoothing at all, as we all get tired, cold or have a bad day once in a while. Stabilizer set to 50 or so should provide a little comfort while keeping the little irregularities.

Bezier curves and other tools

So, you may have heard of a French curve. If not, it's a piece of plastic representing a stencil. These curves are used to make perfectly smooth curves on the basis of a sketch.

In digital painting, we don't have the luxury of being able to use two hands, so you can't hold a ruler with one hand and adjust it while inking with the other. For this purpose, we have instead Bezier curves, which can be made with the Path Selection Tool.

You can even make these on a vector layer, so they can be modified on the fly.

The downside of these is that they cannot have line-variation, making them a bit robotic.

You can also make small bezier curves with the Assistant Tool, amongst the other tools there.

Then, in the freehand brush tool options, you can tick *Snap to Assistants* and start a line that snaps to this assistant.

Presets

So here are some things to consider with the brush-presets that you use:

Anti-aliasing versus jagged pixels

A starting inker might be inclined to always want to use anti-aliased brushes, after all, they look so smooth on the screen. However, while these look good on screen, they might become fuzzy when printing them. Therefore, Krita comes with two default types. Anti-aliased brushes like ink_brush_25 and slightly aliased brushes like ink_tilt, with the latter giving better print results. If you are trying to prepare for both, it might be an idea to consider making the inking page 600dpi and the color page 300dpi, so that the inking page has a higher resolution and the ‘jaggies’ aren’t as visible. You can turn any pixel brush into an aliased brush, by going the `F5` key and ticking **Sharpness**.

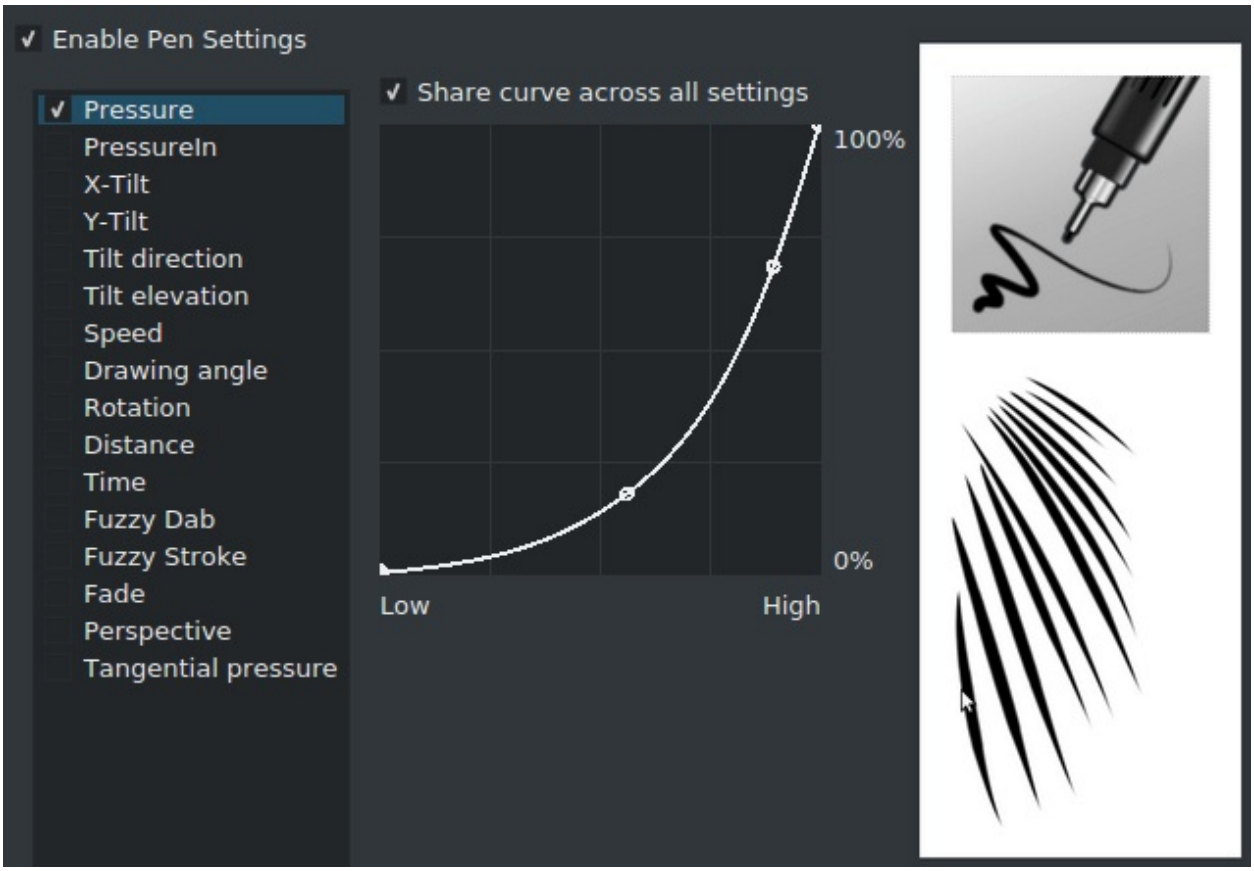
Texture

Do you make smooth ‘wet’ strokes? Or do you make textured ones? For the longest time, smooth strokes were preferred, as that would be less of a headache when entering the coloring phase. Within Krita there are several methods to color these easily, the colorize mask being the prime example, so textured becomes a viable option even for the lazy amongst us.

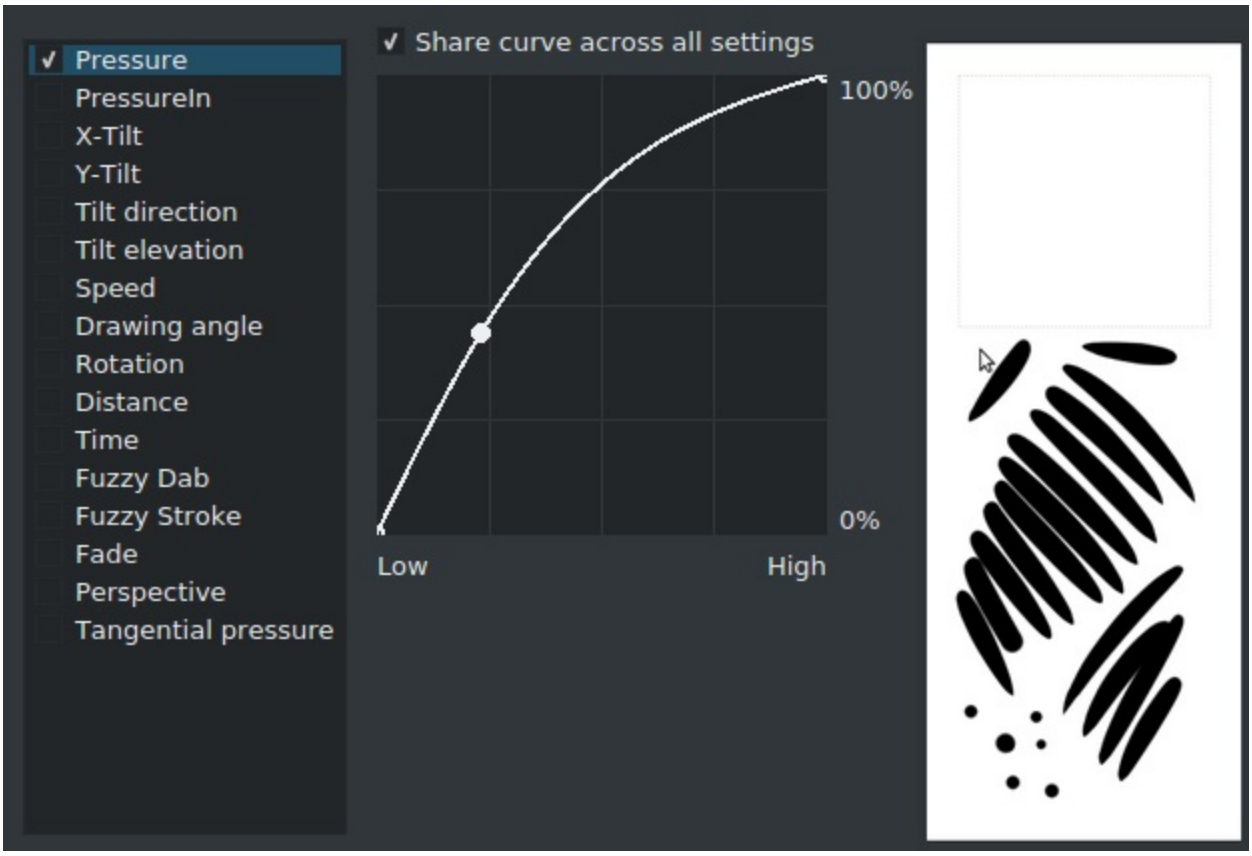
Left: No texture, Center: Textured, Right: Predefined Brush tip.

Pressure curve

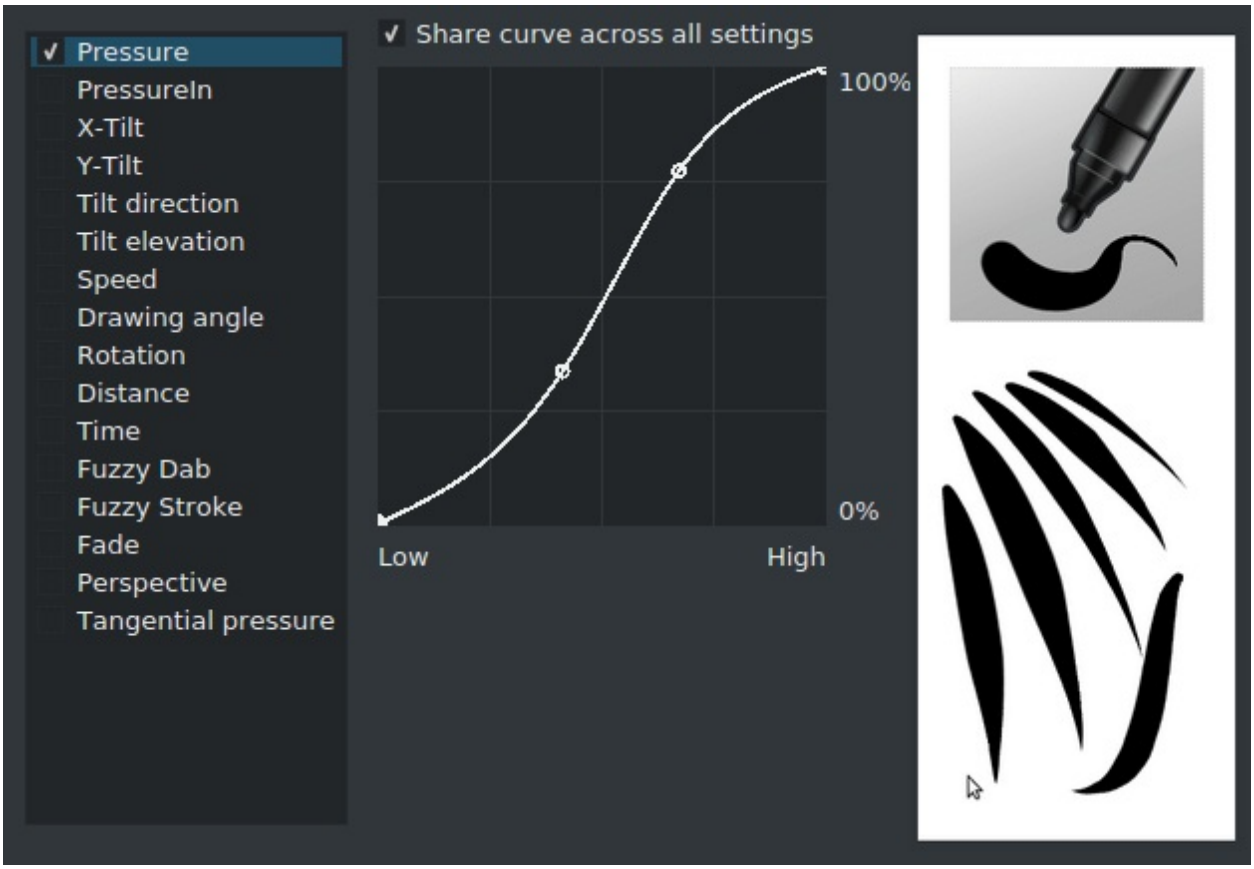
Of course, the nicest lines are made with pressure sensitivity, so they dynamically change from thick to thin. However, different types of curves on the pressure give different results. The typical example is a slightly concave line to create a brush that more easily makes thin lines.



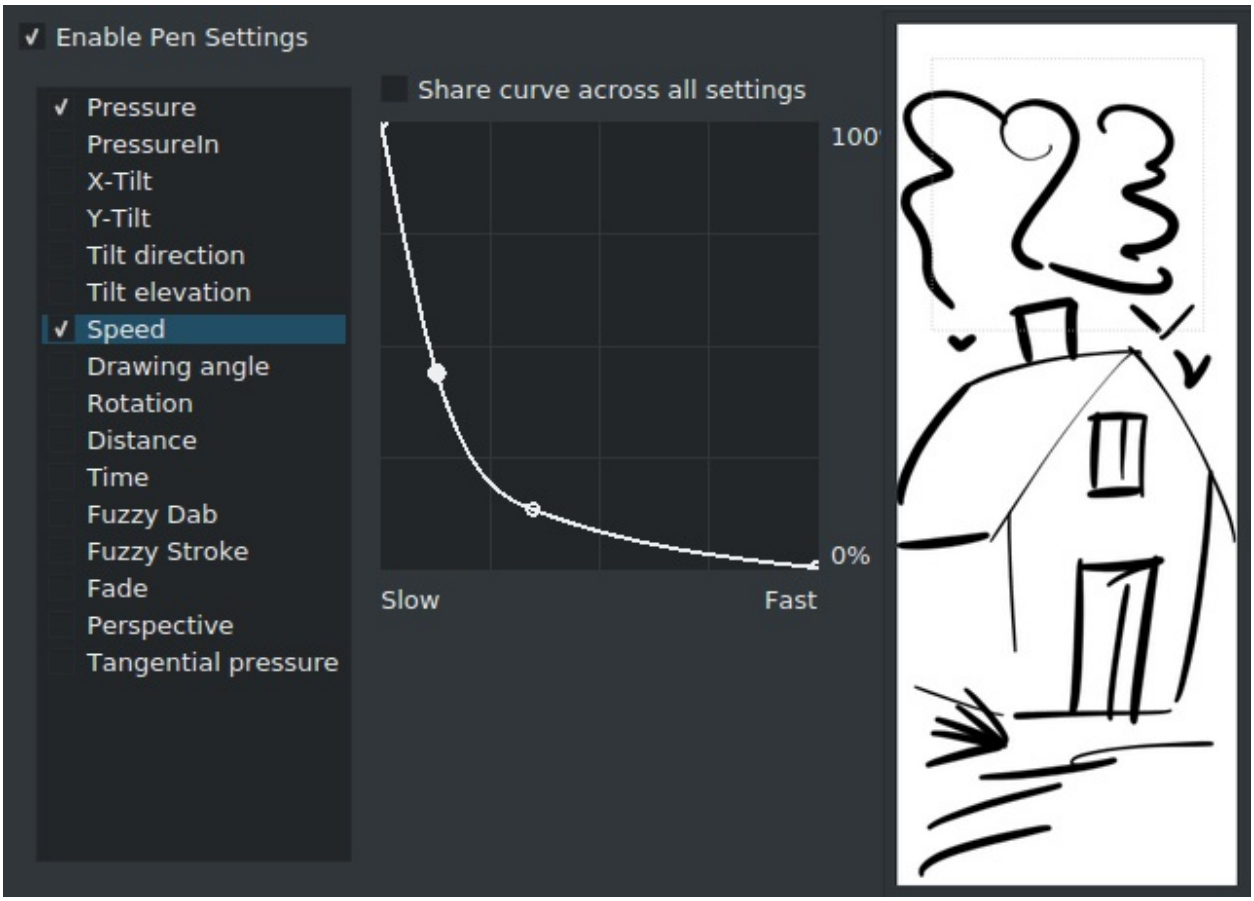
Ink_Gpen_25 is a good example of a brush with a concave pressure curve. This curve makes it easier to make thin lines.



Conversely, here’s a convex brush. The strokes are much rounder.



Fill_circle combines both into an s-curve, this allows for very dynamic brush strokes.



Pressure isn't the only thing you can do interesting things with, adding an inverse convex curve to speed can add a nice touch to your strokes.

Preparing sketches for inking

So, you have a sketch and you wish to start inking it. Assuming you've scanned it in, or drew it, you can try the following things to make it easier to ink.

Opacity down to 10%

Put a white (just press the `Backspace` key) layer underneath the sketch. Turn down the opacity of the sketch to a really low number and put a layer above it for inking.

Make the sketch colored

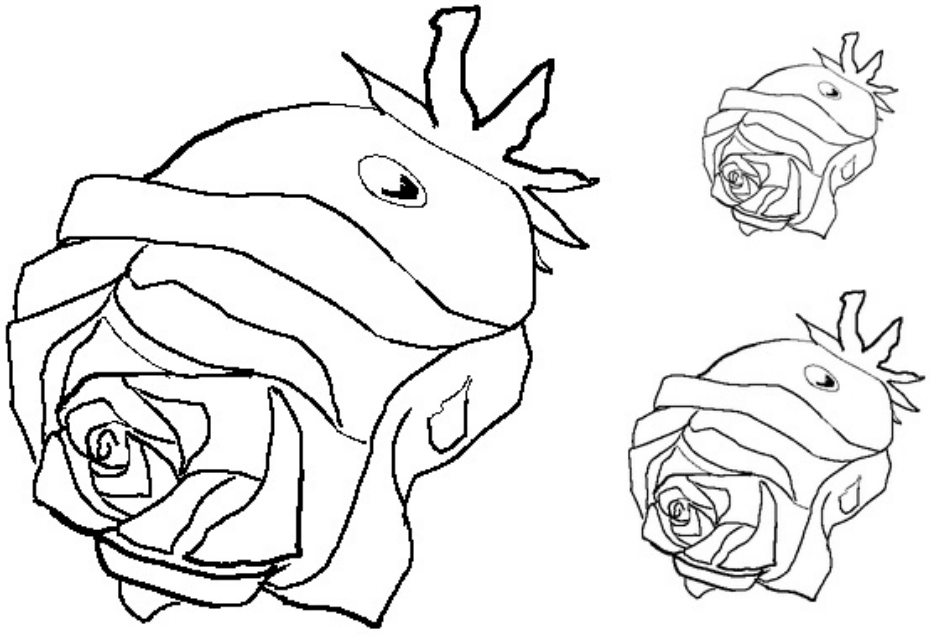
Put a layer filled with a color you like between the inking and sketch layer. Then set that layer to ‘screen’ or ‘addition’, this will turn all the black lines into the color! If you have a transparent background, or put this layer into a group, be sure to tick the alpha-inherit symbol!

Make the sketch colored, alternative version

Or,  on the layer, go to layer properties, and untick ‘blue’. This works easier with a single layer sketch, while the above works best with multi-layer sketches.

Super-thin lines

If you are interested in super-thin lines, it might be better to make your ink at double or even triple the size you usually work at, and, only use an aliased pixel brush. Then, when the ink is finished, use the fill tool to fill in flats on a separate layer, split the layer via *Layer* ▸ *Split* ▸ *Layer Split*, and then resize to the original size.



This might be a little of an odd way of working, but it does make drawing thin lines trivial, and it’s cheaper to buy RAM so you can make HUGE images than to spent hours on trying to color the thin lines precisely, especially as colorize mask will not be able to deal with thin anti-aliased lines very well.

Tip

David Revoy made a set of his own inking tips for Krita and explains them in this youtube video [https://www.youtube.com/watch?v=xvQ5l0edsq4].

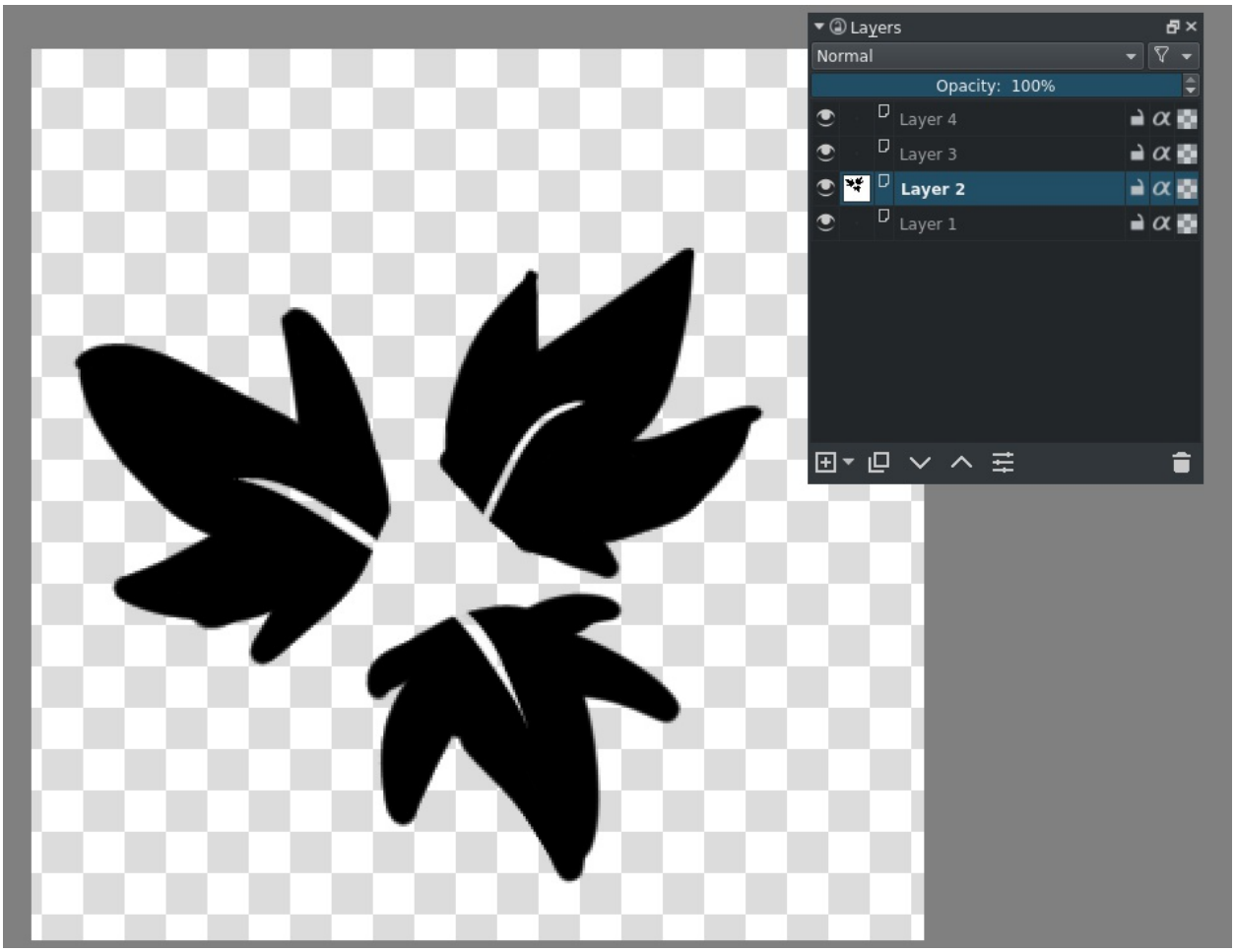
Brush-tips:Animated Brushes

Question

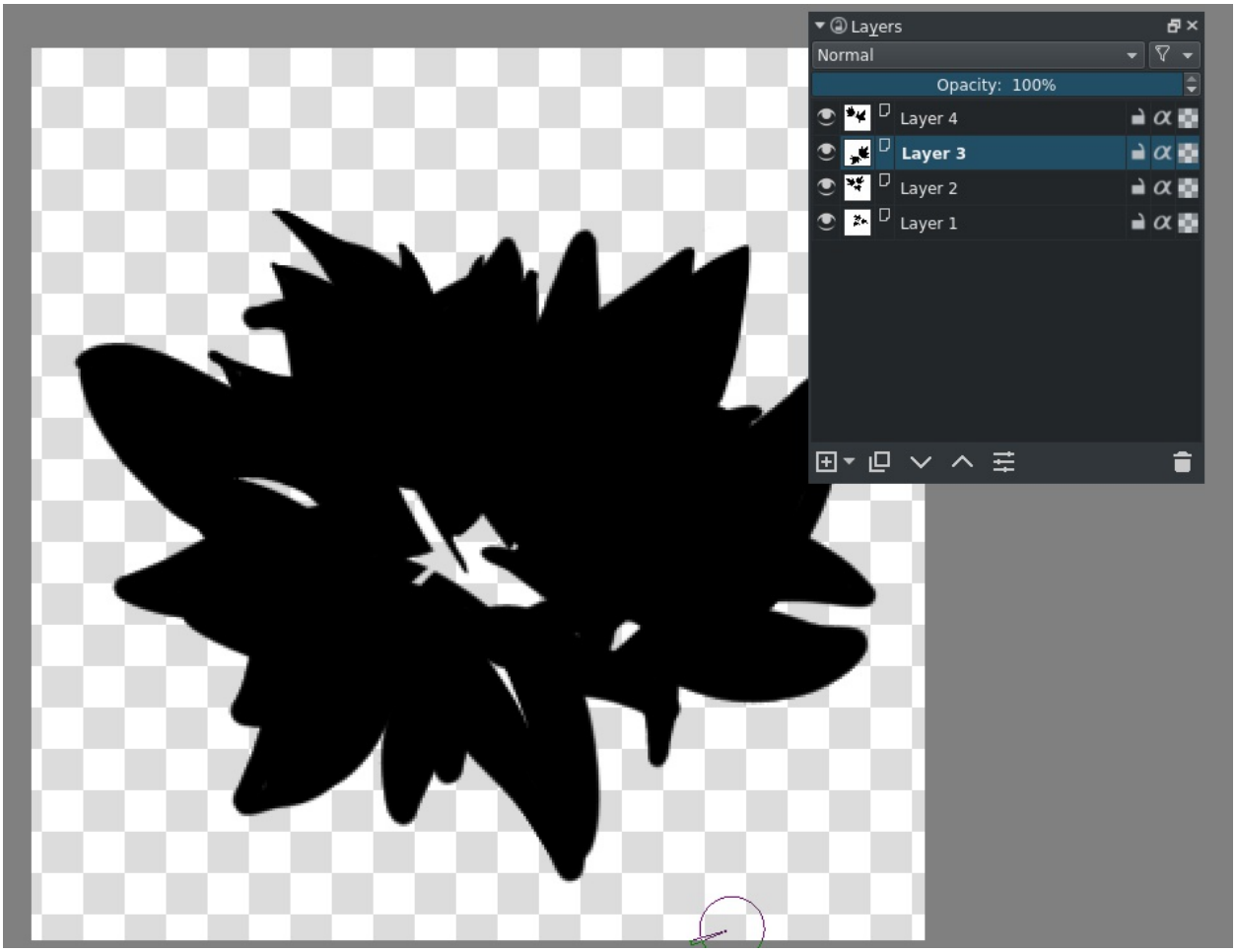
I was messing with the brushes and noticed there is an option for them to be “animated”. What does it mean and how do I use it?

Basically, they’re what are officially called ‘image hoses’, and they’re quite fun. They are basically brush-tips with multiple image files.

The typical way to make them is to first draw the ‘frames’ on a small canvas, per layer:

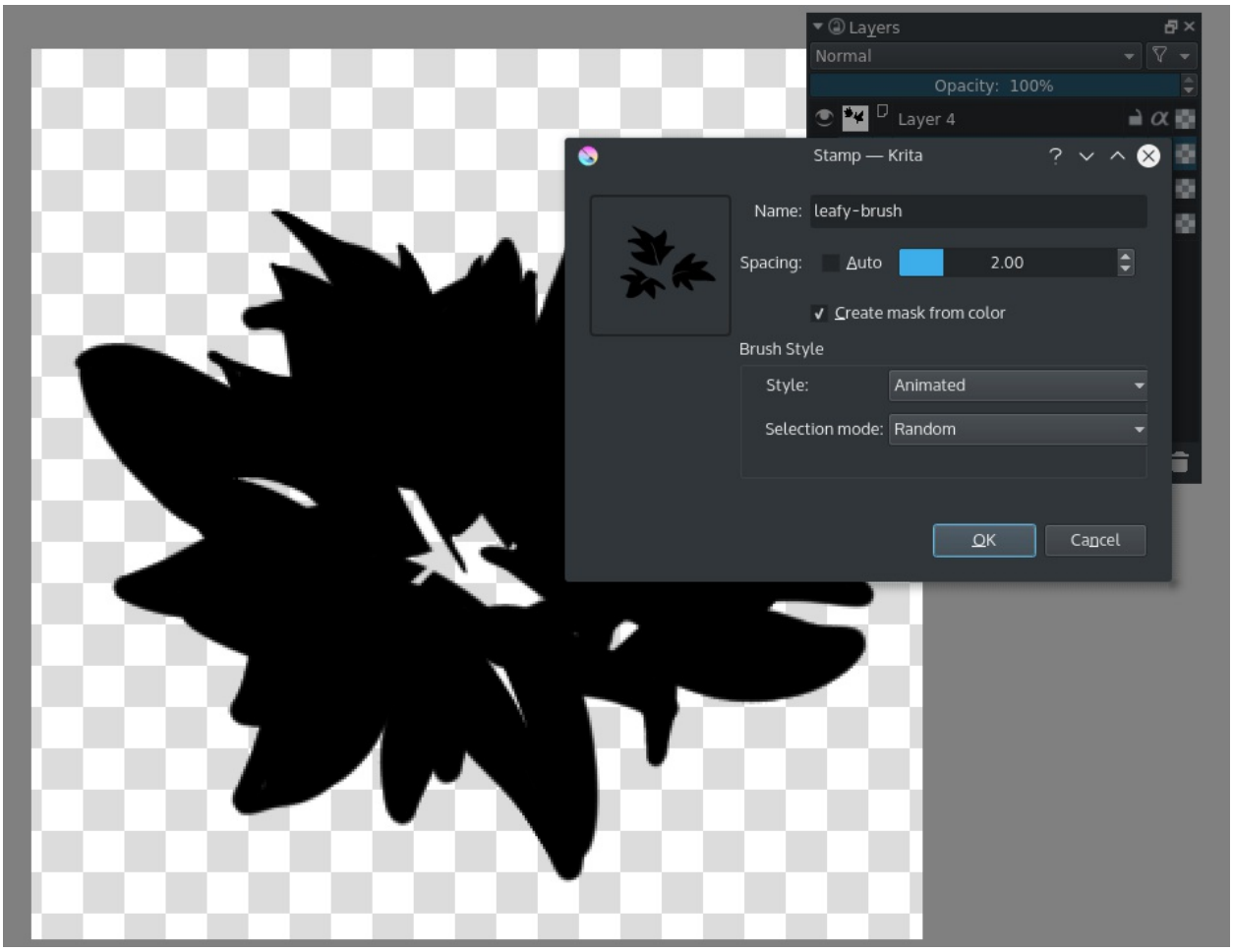


You can use the `Alt` +  shortcut on the layer thumbnails to isolate layers without hiding them.



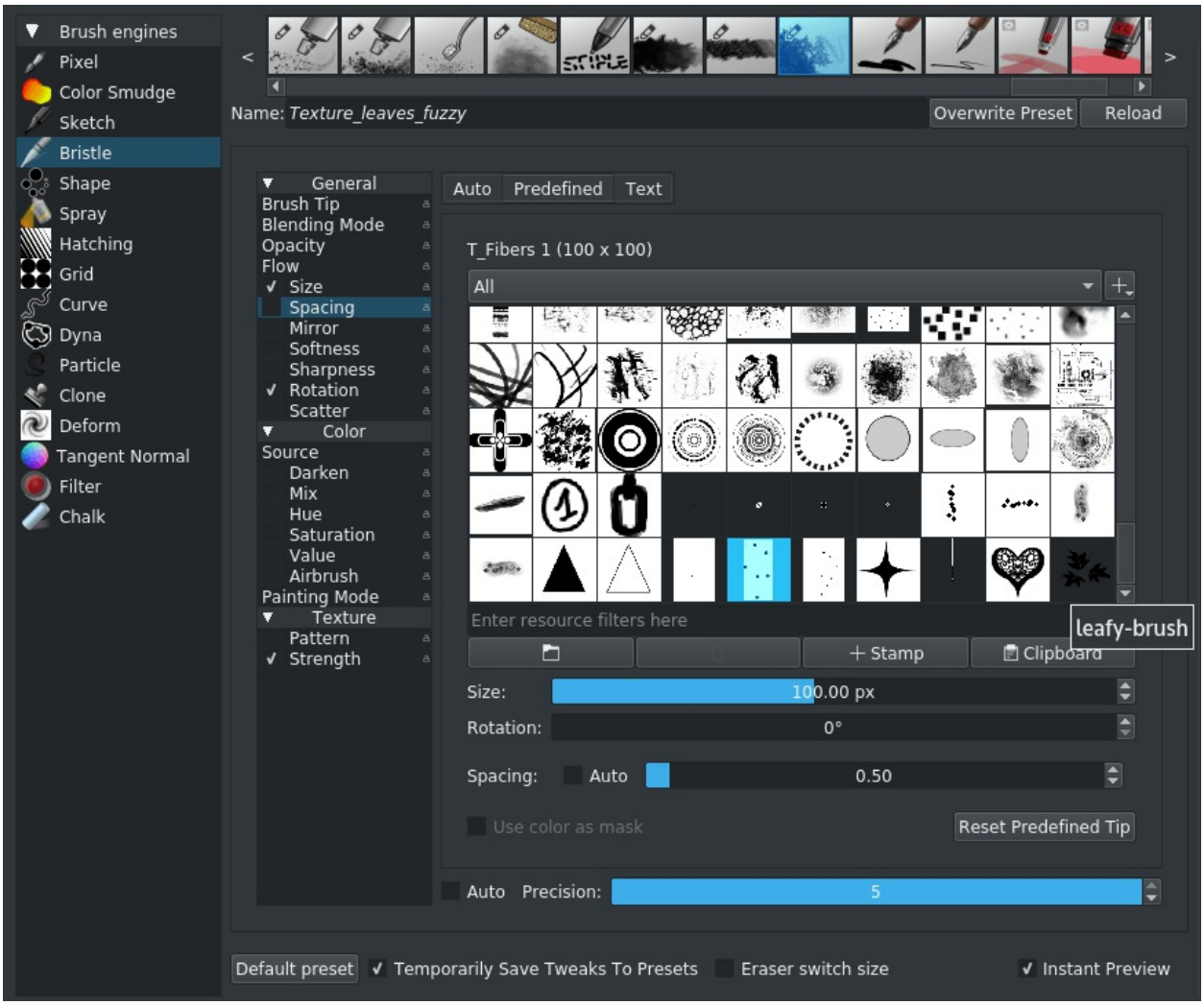
When done you should have a mess like this.

Go into the brush settings (`F5` key), and go to predefined brush-tips, and click stamp. You will get this window.

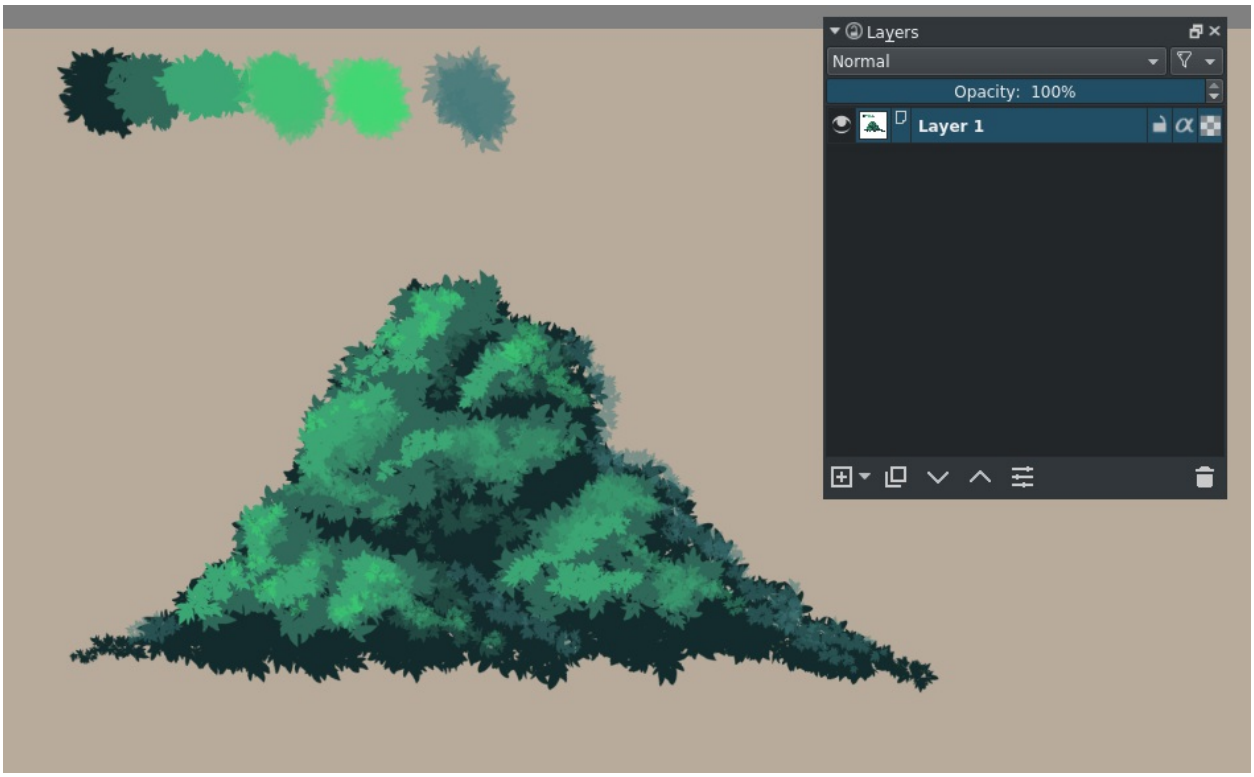


And then use style **animated** and selection mode set to **random**.

Krita uses Gimp’s image hose format which allows for random selection of the images, angle based selection, pressure based selection, and incremental selection (I have no idea what constant does).



Then select the above brush and your new leafy-brush tip.



And use it to paint trees! (for example)

You can also use animated brush tips to emulate, for example, bristle brush tips that go from very fine bristles to a fully opaque stamp based on pressure, like a dry paintbrush might do. To do this, you would follow the above instructions, but for each layer, create a different cross-section of the brush to correspond with the amount of pressure applied.

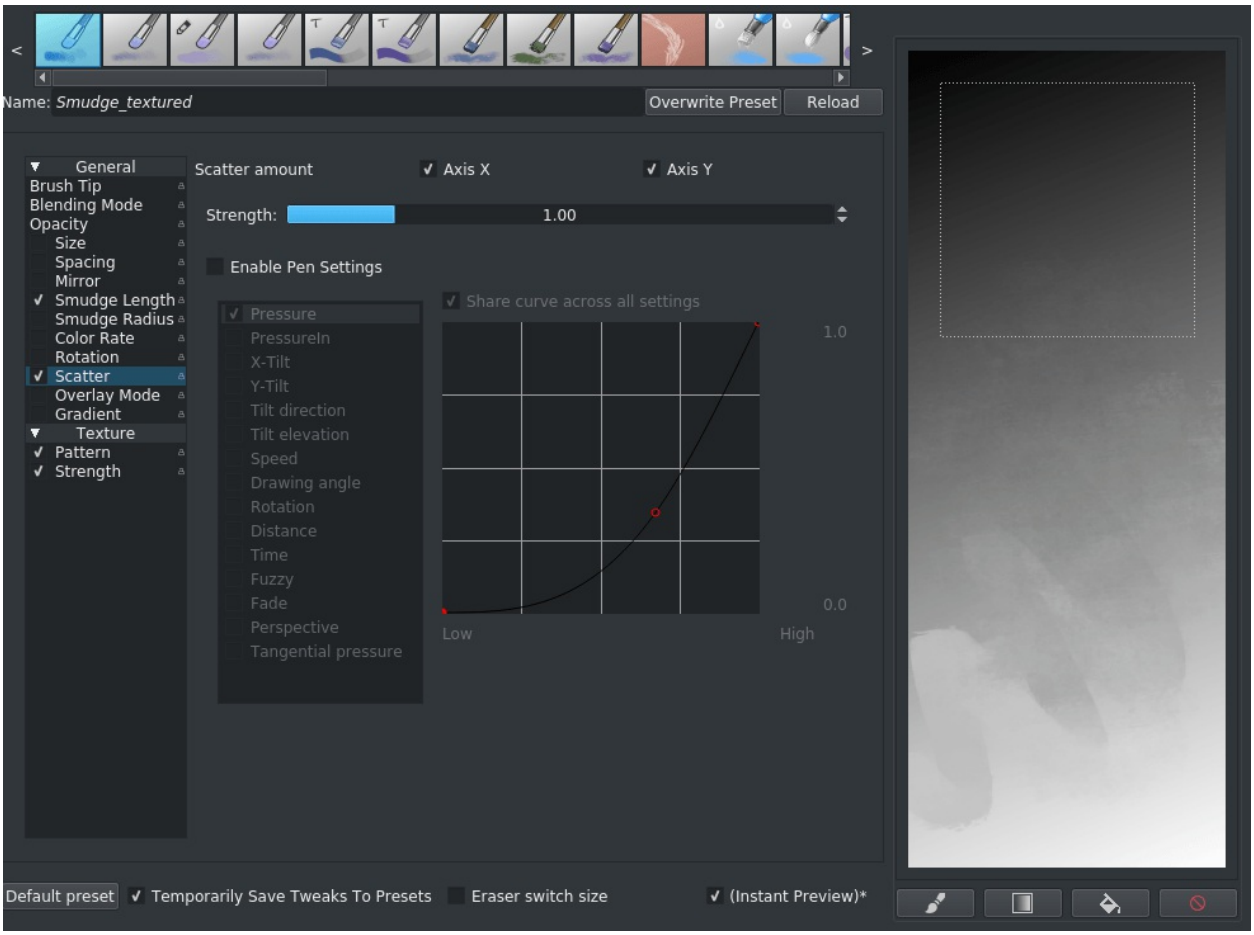
Brush Tips: Bokeh

Question

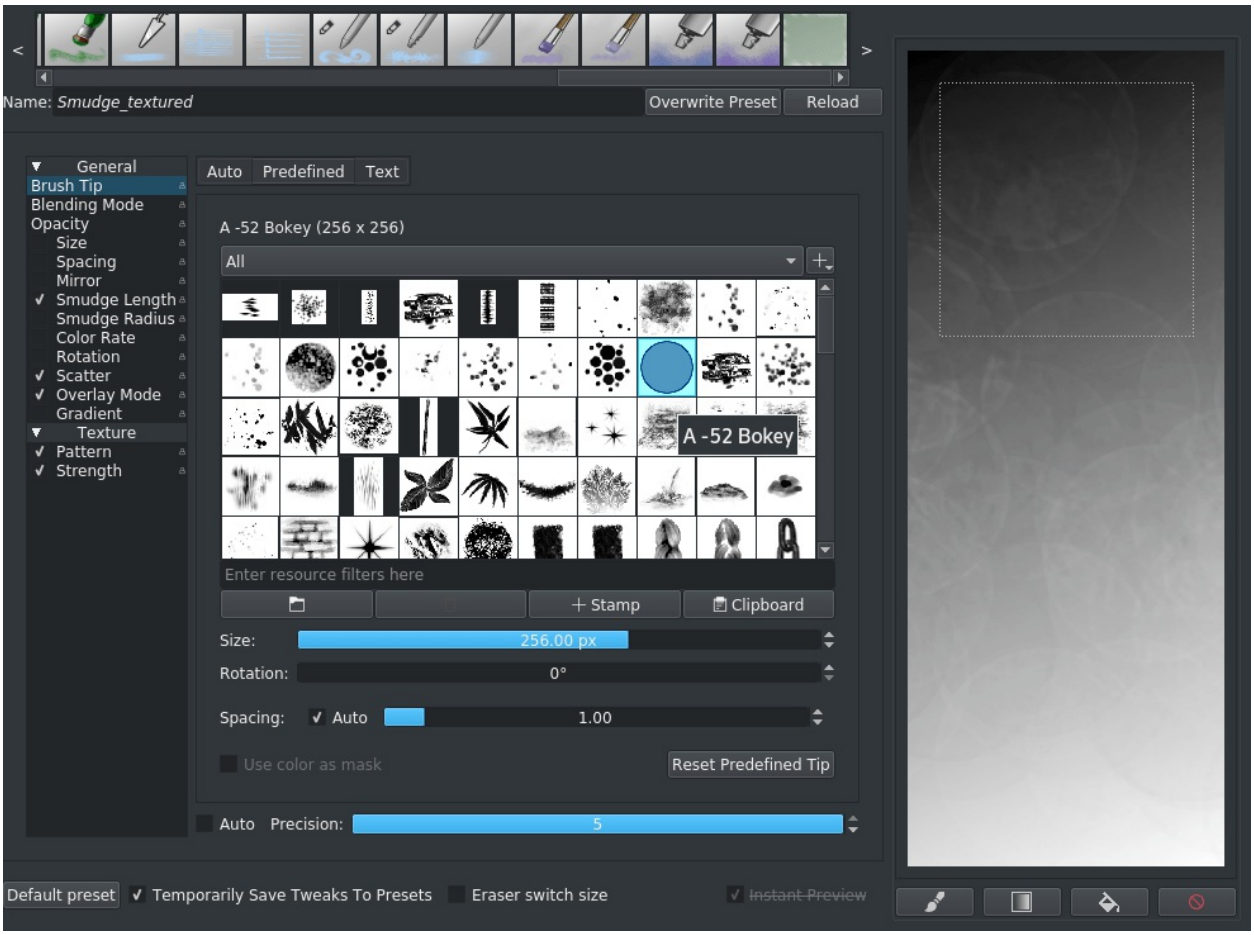
How do you do bokeh effects?

First, blur your image with the Lens Blur to roughly 50 pixels.

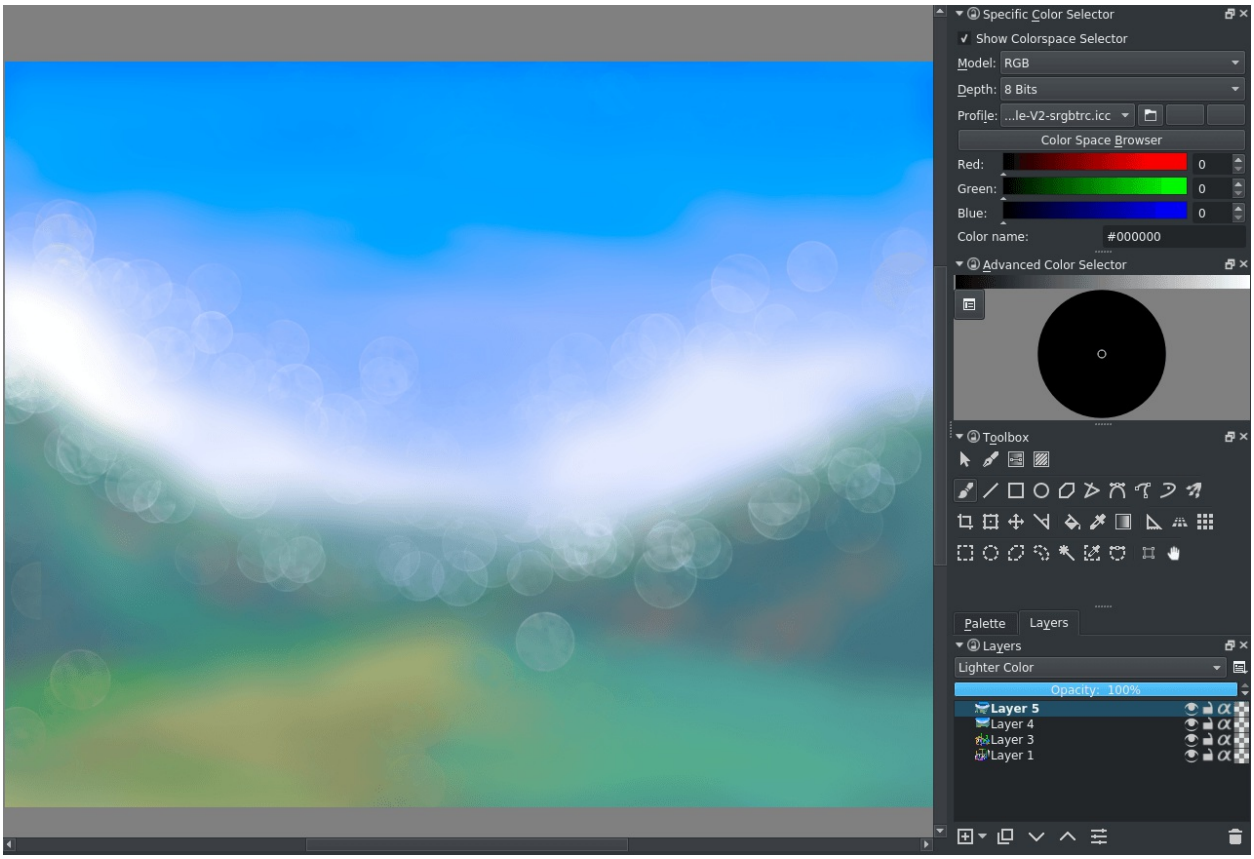
Take smudge_textured, add scattering, turn off tablet input.



Change the brush-tip to ‘Bokeh’ and check ‘overlay’ (you will want to play with the spacing as well).



Then make a new layer over your drawing, set that to ‘lighter color’ (it’s under lighter category) and painter over it with you brush.



Overlay mode on the smudge brush allows you to sample all the layers, and the ‘lighter color’ blending mode makes sure that the Bokeh circles only show up when they are a lighter color than the original pixels underneath. You can further modify this brush by adding a ‘fuzzy’ sensor to the spacing and size options, changing the brush blending mode to ‘addition’, or by choosing a different brush-tip.

Brush Tips: Caustics

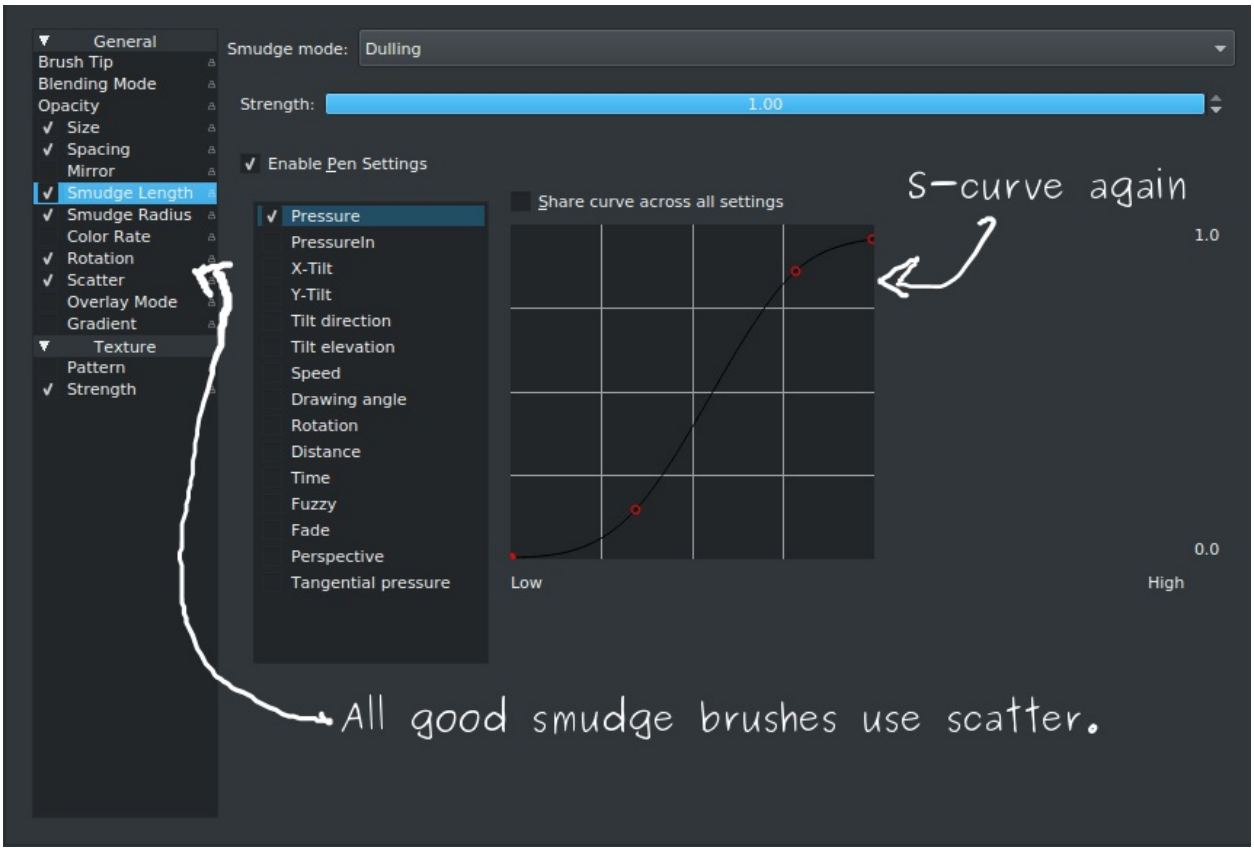
Question

Could you do a tutorial on how to recreate the look of light refracting in water?

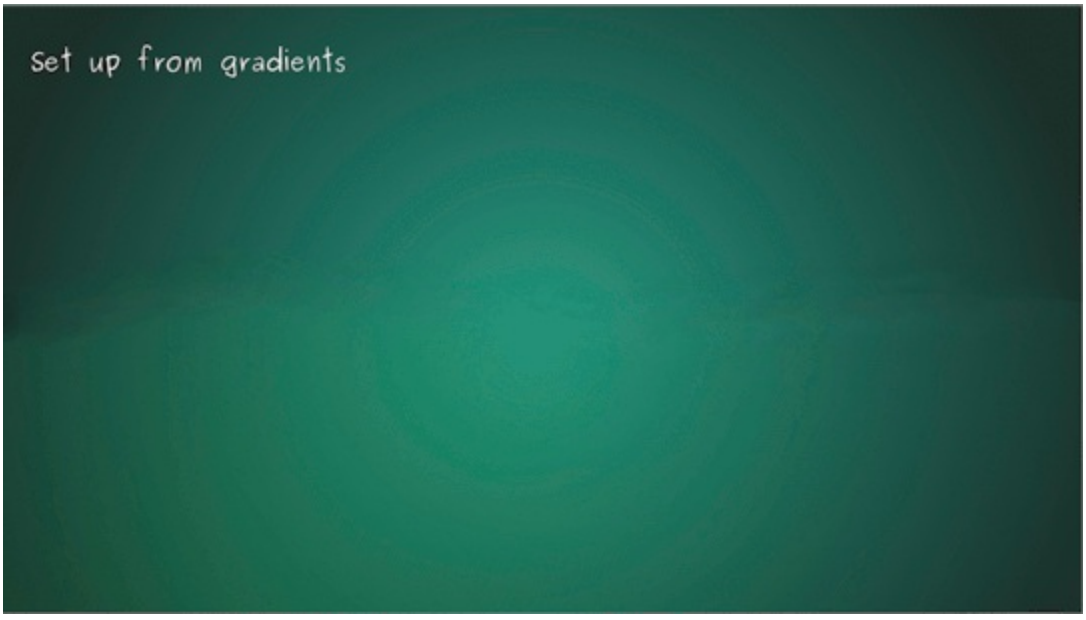
Sure, caustics, it’s not like it’s the most complicated effect known to CG graphics… Okay, so the first thing is that light effects never work in isolation: you need to be spot on with colors and other effects to make it work. So we first need to recreate the surroundings a bit.

We set up something simple with gradients. Some radial, some linear. The eraser mode works with gradients as well, so use that to your advantage!

We create a simple smudge brush by taking smudge_soft and adding scattering to it, as well as an s-curve on the smudge length.



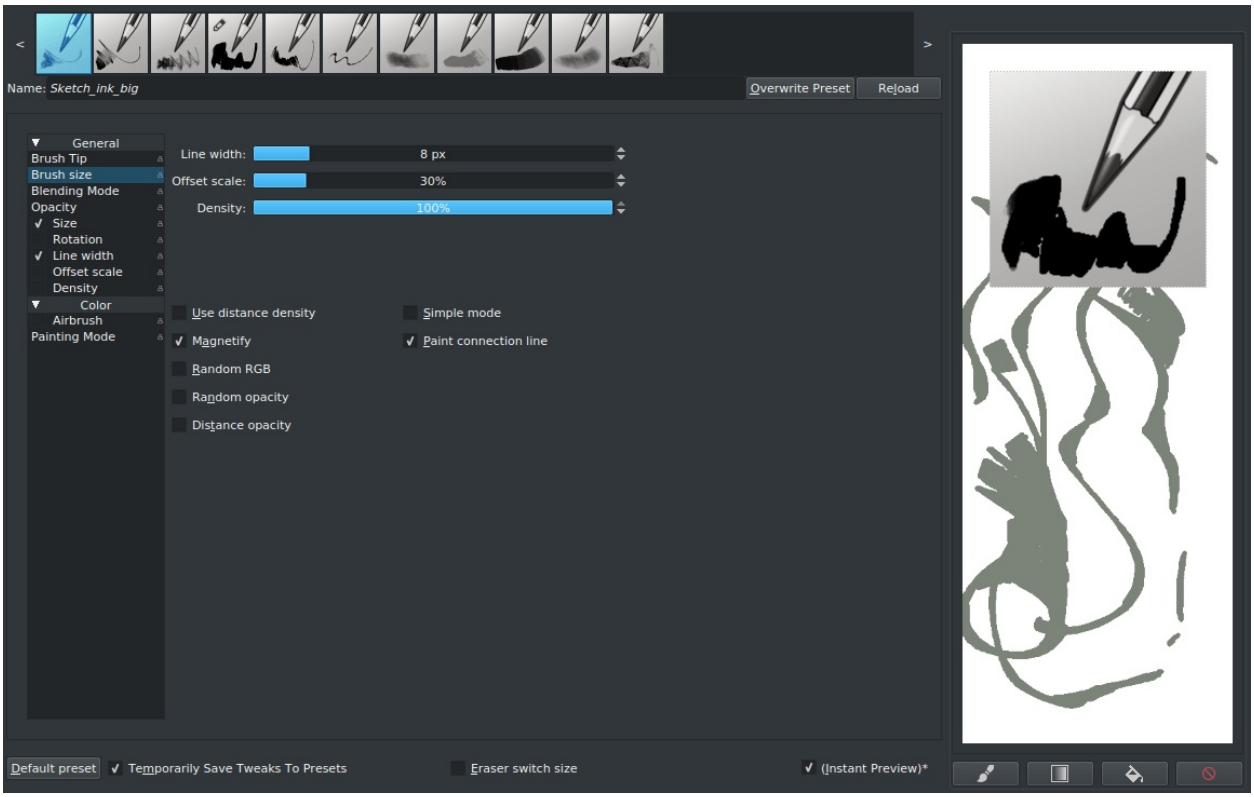
And then we build up a quick base:



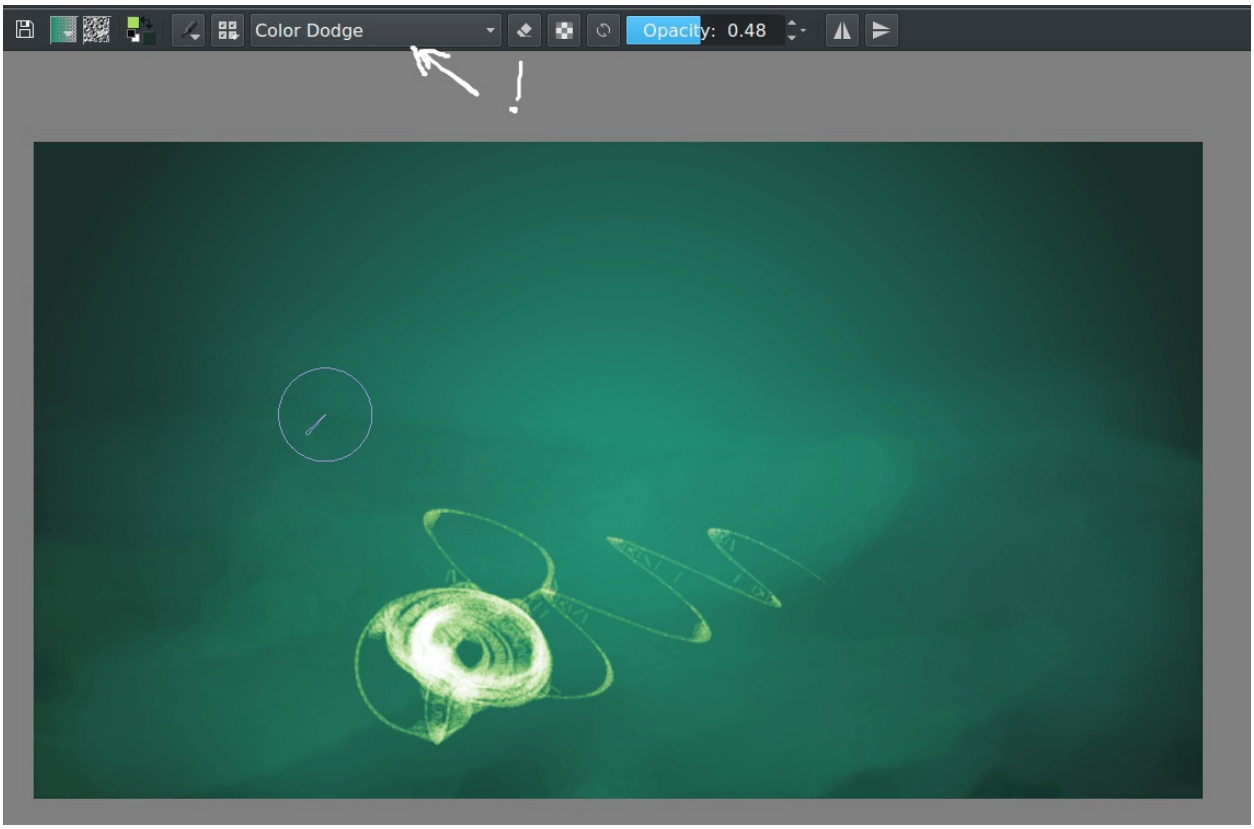
Note how the smudge brush here is used not just to mix areas, but also to create definition of borders by lowering the scatter. (If you reverse the

pressure curve on the scatter, this'll be easily done by increasing the pressure on the stylus)

Now for the real magic. Caustics are a bit hairy, which means it's a good candidate for the sketch brush engine.

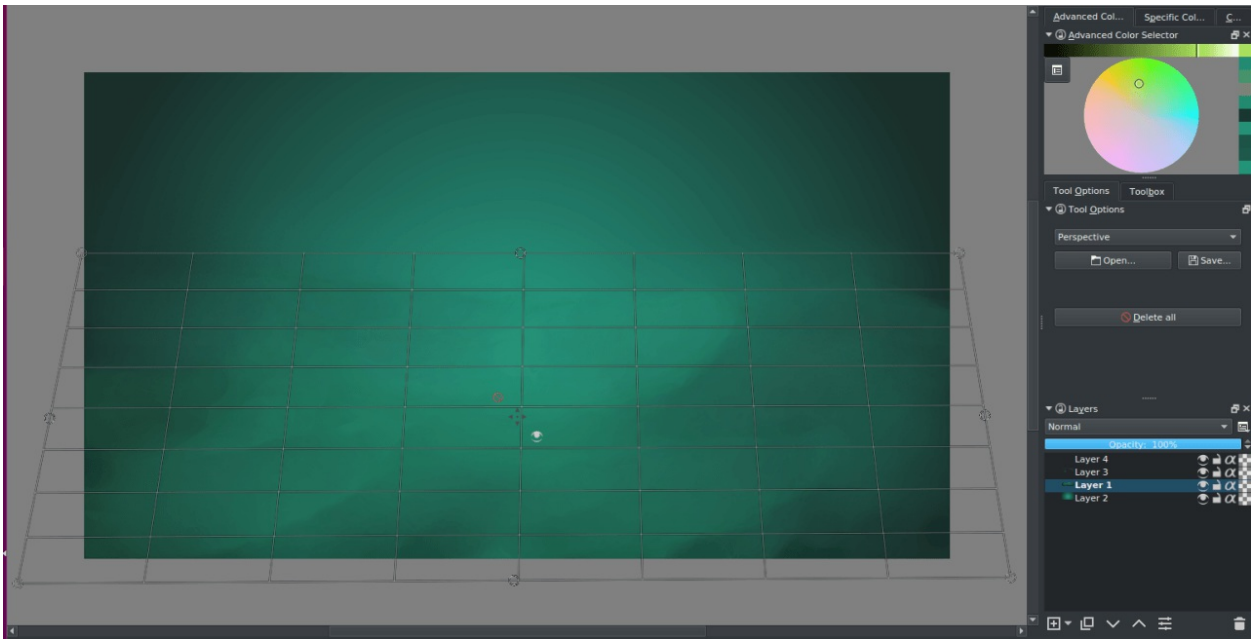


Take *sketch_ink_big*, and add pressure to the *Line width* while setting *Density* under the *Brush size* to 100%. This makes it extra hairy.

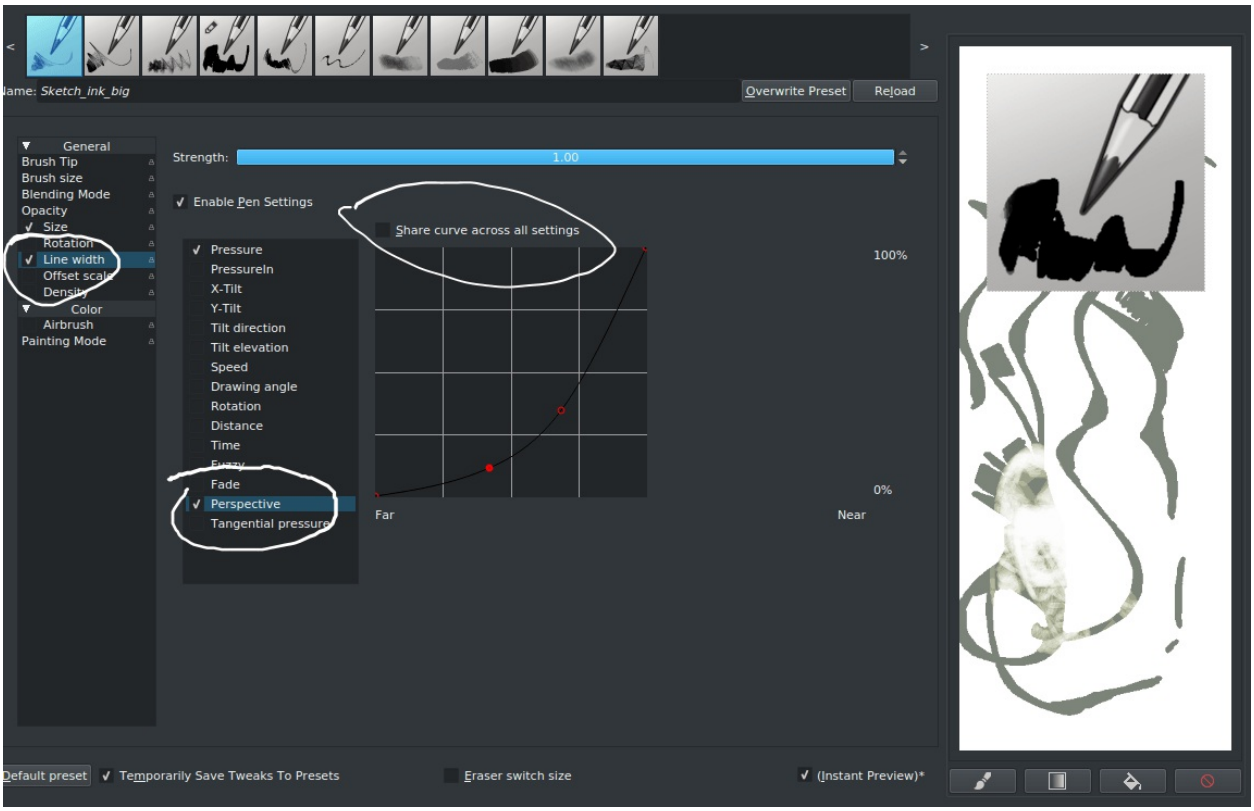


Set the brush blending mode to *Color Dodge*, and select the color of our caustics. Color dodge will cause a move towards white by applying special dodge color maths to our brush dabs instead of the *Normal* averaging color maths.

Outside of pressure for making varying strokes, glowiness for the light and extra density, we also want to have the size of the line decrease the further away it is…

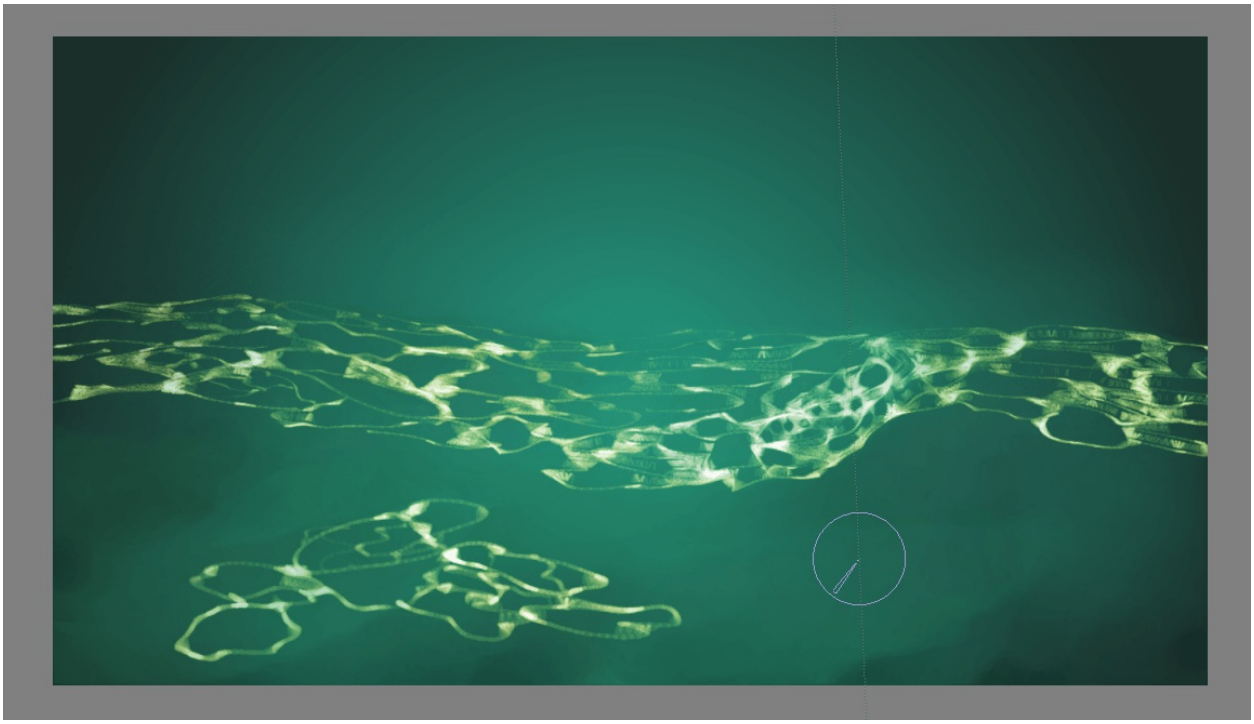


Then, use the assistant editor tool to add a perspective grid. It doesn't need to be perfectly in perspective, because we'll only use it for the perspective sensor.

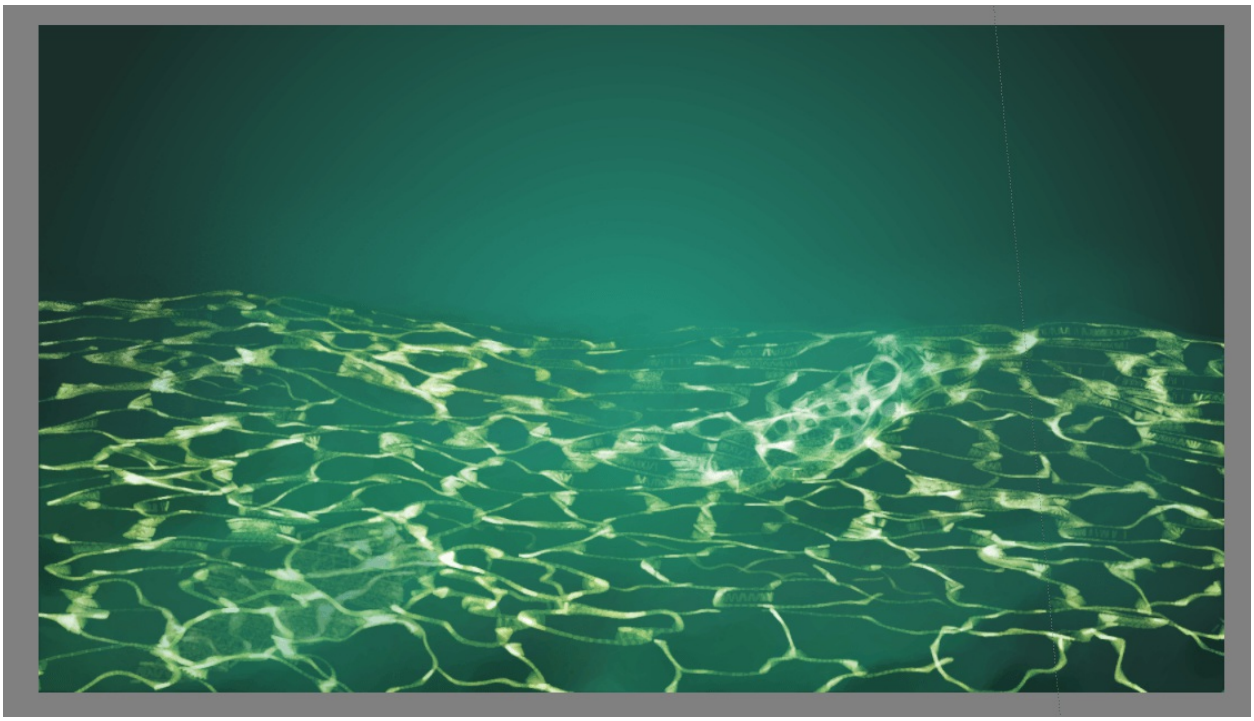


This will cause the brush to give smaller lines the further it registers on the perspective assistant. (It only works per single perspective assistant, making it not very good for chaining, but for our purpose this is good.)

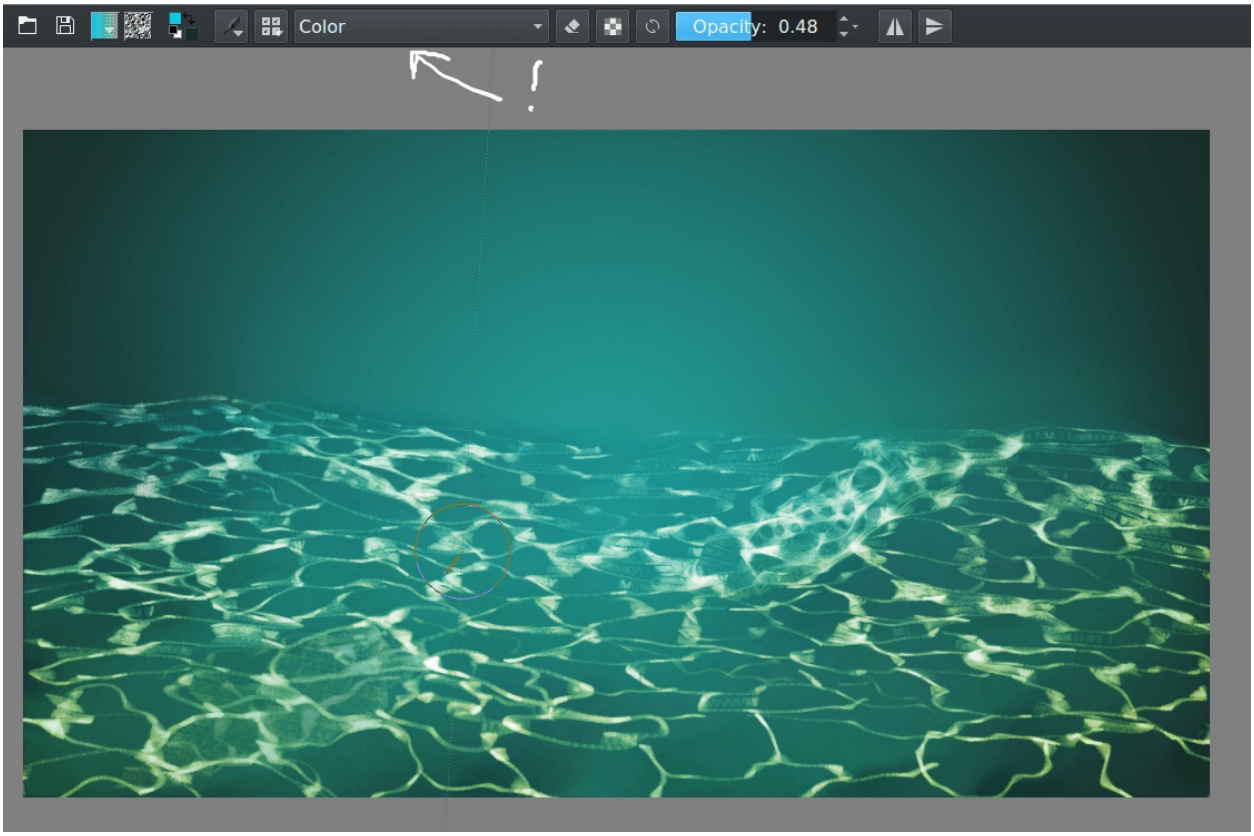
Then you start slowly building up your lines. (Make sure to make a copy of the layer. The color dodge blending doesn't work well on a separate layer, so do it on one that also has the ground on it.)



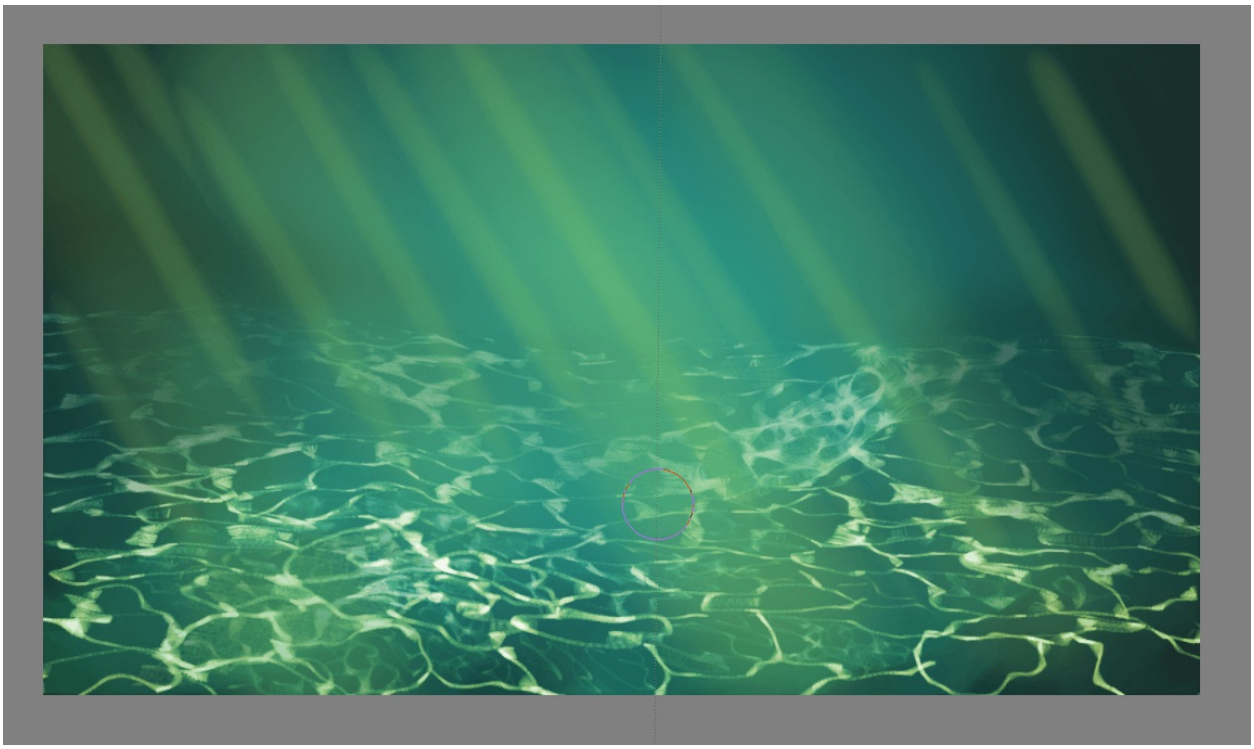
Make sure to try and follow the shapes you made. (*I failed at this*) The great thing about the sketch brush is that it causes those little 'melt-togethers' where two lines cross. This is only per stroke, so make a lot of long ongoing strokes with this brush to make use of it.



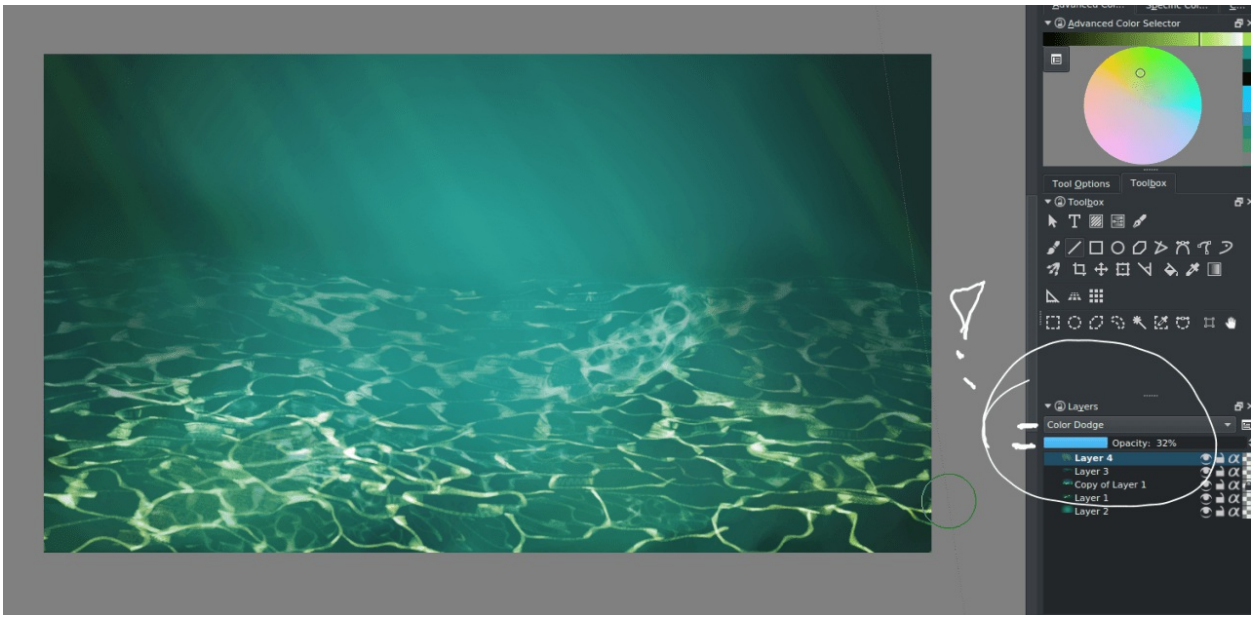
Then take the gradient tool, and set the blending mode to color and the paint tool to a light blue, so we can get in the bluish atmospheric effect.



Then use the airbrush_pressure with the line tool to make some light-shafts of different sizes on a separate layer. (Don’t forget you can use the eraser mode for subtle erasing with the line tool as well)



Set the blending mode to color dodge and lower the opacity.



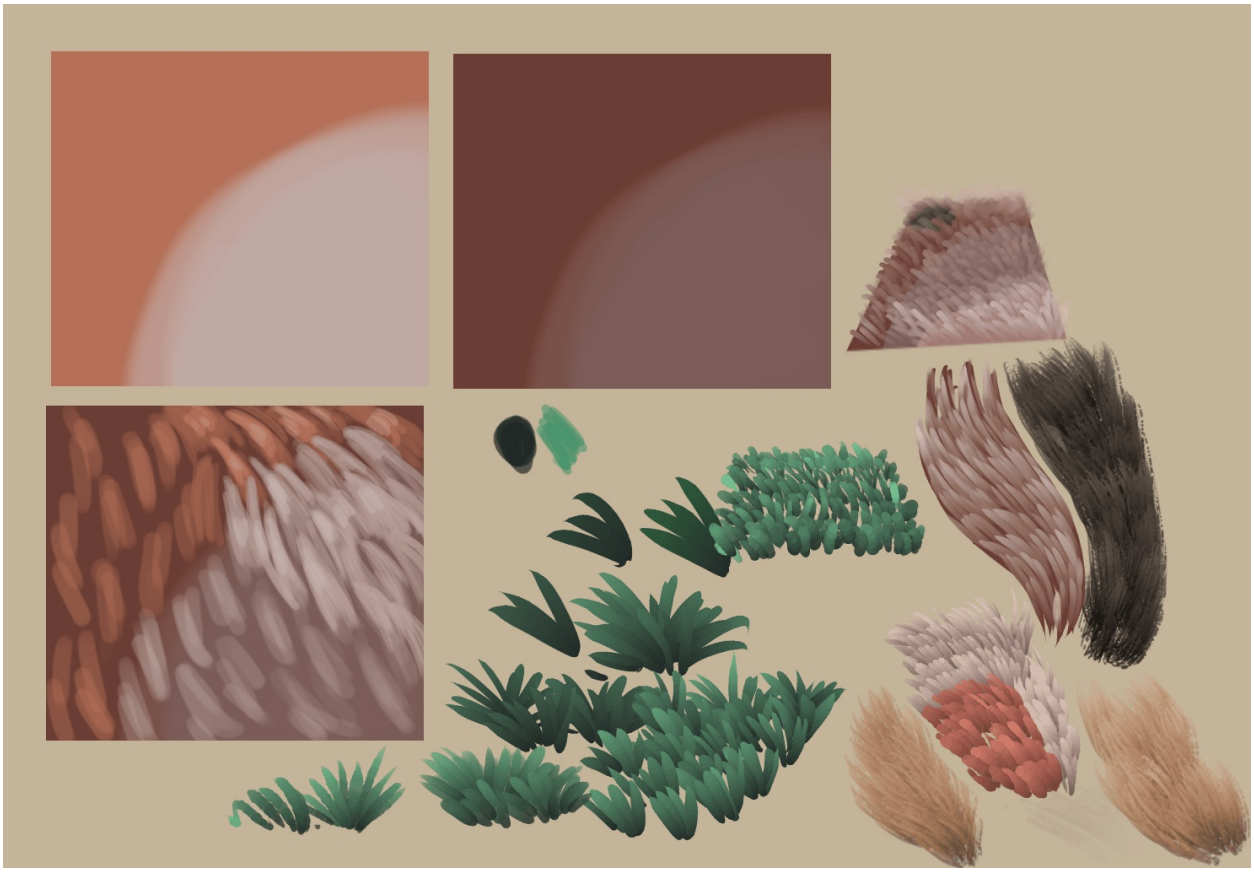
Finally, polish the piece with the airbrush tool and some local color picking.

Final Result

Painting fur

Question

What brushes are best for fur textures?

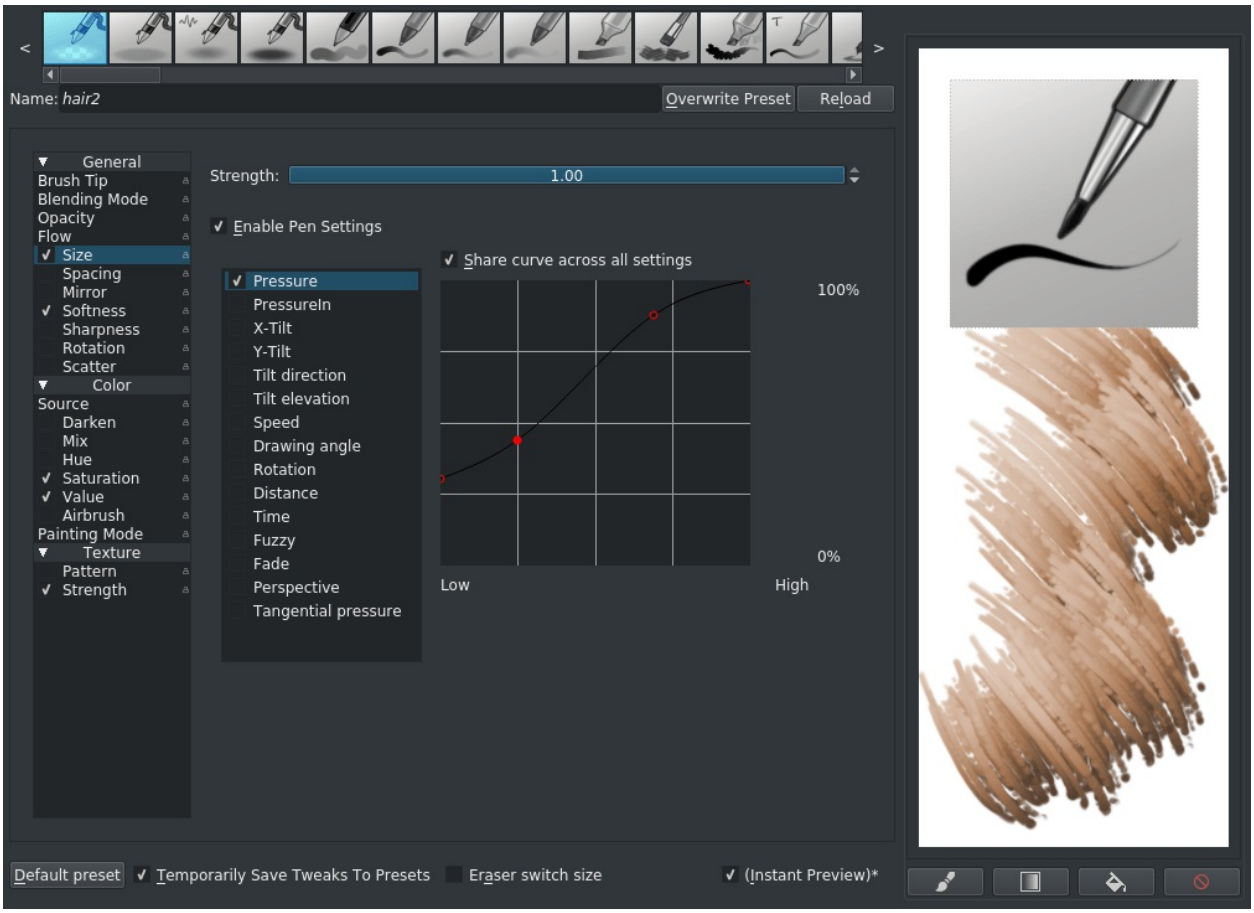


So typically, you see the same logic applied on fur as on regular Brush-tips:Hair.

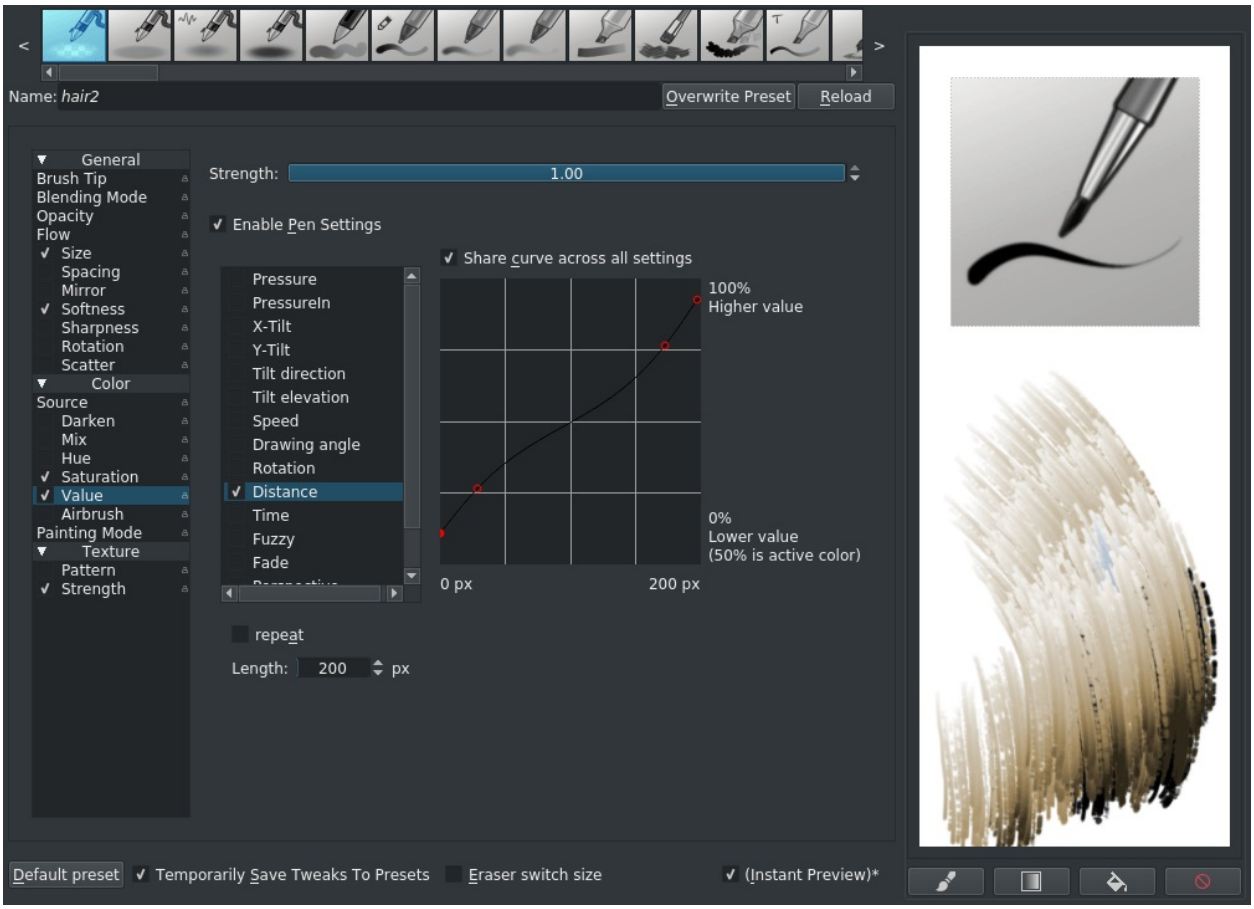
However, you can make a brush a little easier by using the Gradient, Mix and HSV options in the pixel and color smudge brushes. Basically, what we want to do is have a stroke start dark and then become lighter as we draw with it, to simulate how hair-tips catch more light and look lighter due to being thinner at the ends, while at the base they are frequently more dark.

Note

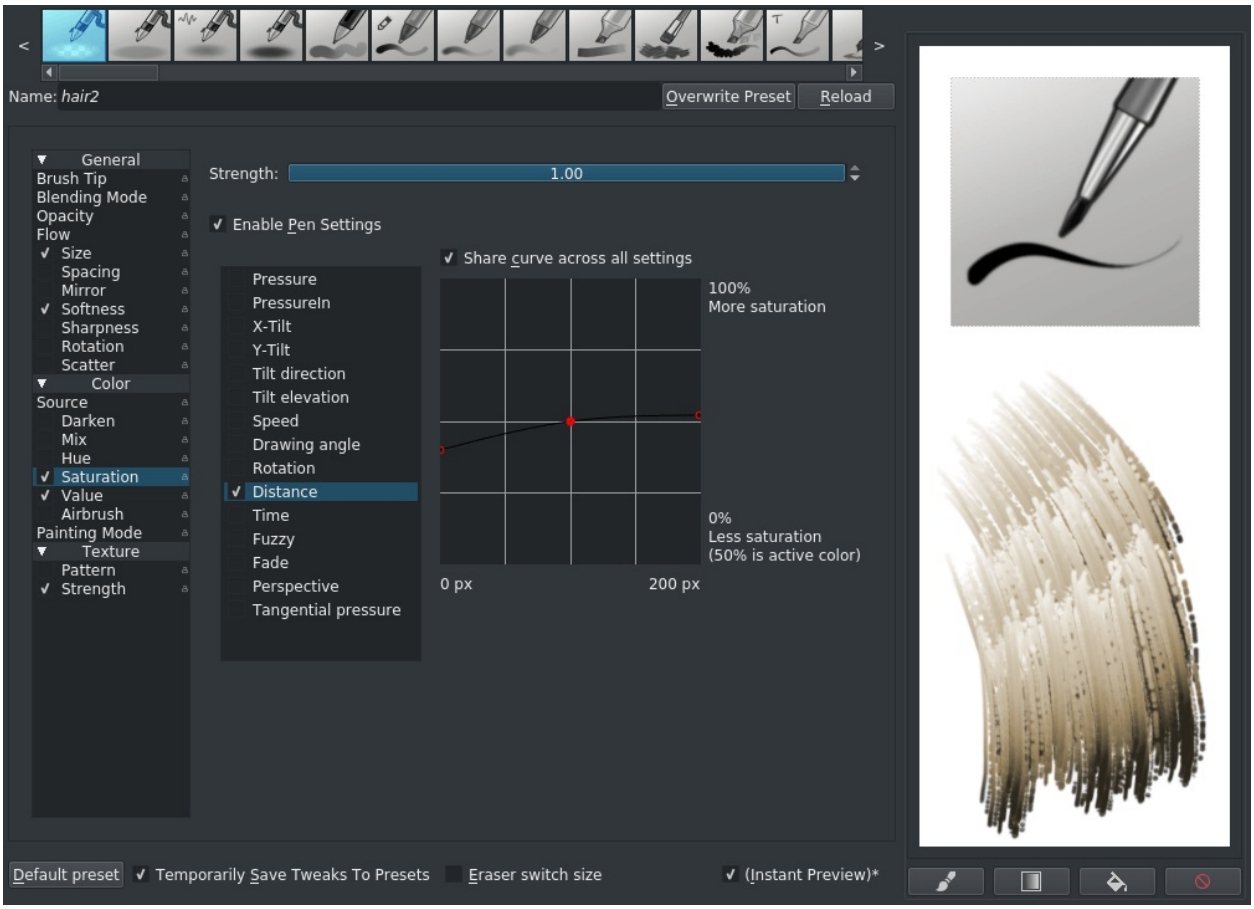
This tutorial contains outdated screenshots, but you should be able to follow along if you ignore a bit different look of the brush editor. Originally it also used brush preset and brush tip from the older resources set (*ink_brush_25* and *A - 2 Dirty Brush*), but you should be able to get similar results with brush preset *b) Basic-5 Size* and brush tip *sparkle* or other similar brush preset (based on Pixel Engine) and similar brush tip (multiple dots).



Take the *b) Basic-5 Size* and choose under *Brush Tip* ▸ *Predefined “sparkle”*. Set the spacing to *Auto* and right-click the spacing bar to type in a value between 0.25 and 0.4. Also turn on the *Enable Pen Settings* on flow. Replicate the pressure curve above on the size option. We don’t want the hairs to collapse to a point, hence why the curve starts so high.

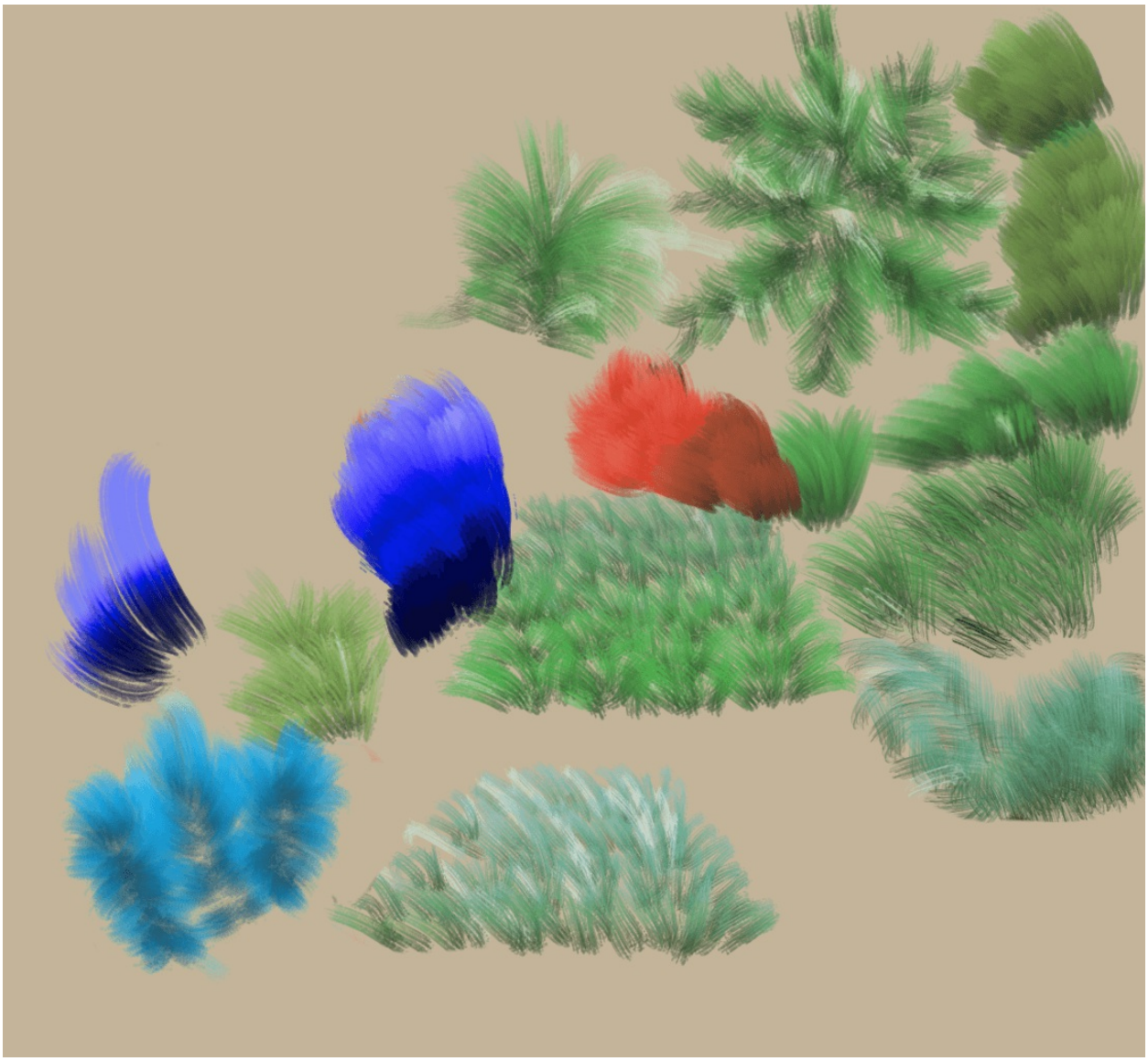


Then activate value and reproduce this curve with the *Distance* or *Fade* sensor. Like how the pressure sensor changes a value (like size) with the amount of pressure you put on the stylus, the distance sensor measures how many pixels your stroke is, and can change an option depending on that. For the HSV sensors: If the curve goes beneath the middle, it'll become remove from that adjustment, and above the vertical middle it'll add to that adjustment. So in this case, for the first 100px the brush dab will go from a darkened version of the active paint color, to the active paint color, and then for 100px+ it'll go from the active color to a lightened version. The curve is an inverse S-curve, because we want to give a lot of room to the mid-tones.



We do the same thing for saturation, so that the darkened color is also slightly desaturated. Notice how the curve is close to the middle: This means its effect is much less strong than the value adjustment. The result should look somewhat like the fifth one from the left on the first row of this:

The others are done with the smudge brush engine, but a similar setup, though using color rate on distance instead. Do note that it’s very hard to shade realistic fur, so keep a good eye on your form shadow. You can also use this with grass, feathers and other vegetation:



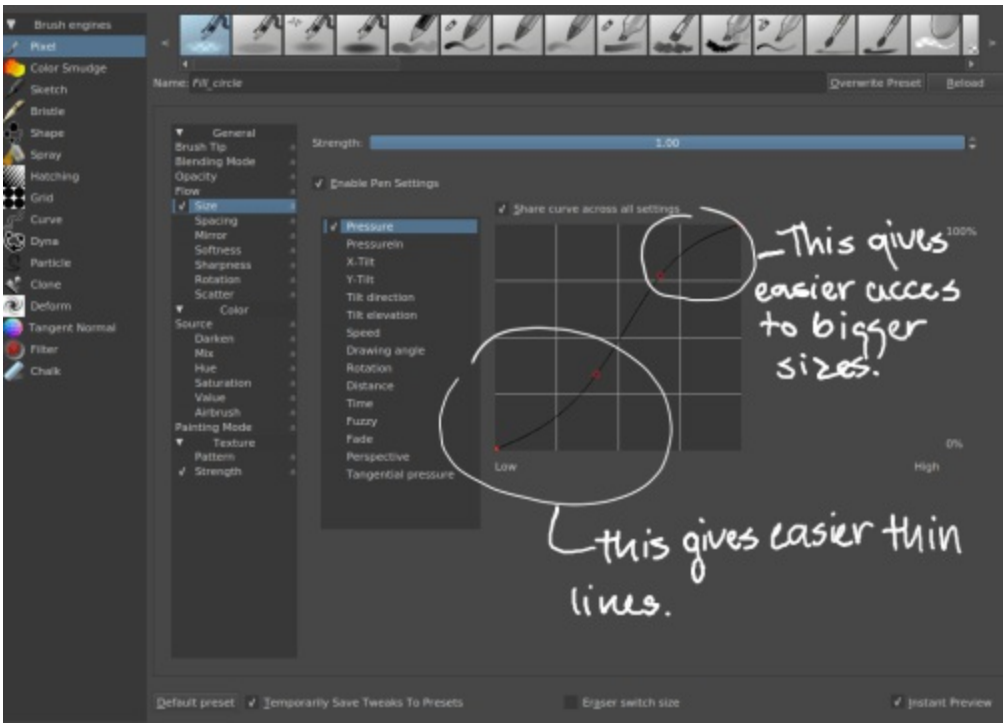
For example, if you use the mix option in the pixel brush, it’ll mix between the fore and background color. You can even attach a gradient to the color smudge brush and the pixel brush. For color smudge, this is just the *Gradient* option, and it’ll use the active gradient. For the pixel brush, set the color-source to *Gradient* and use the mix option.

You can also combine this with the lighter color blending mode and wraparound mode to make making grass-textures really easy!

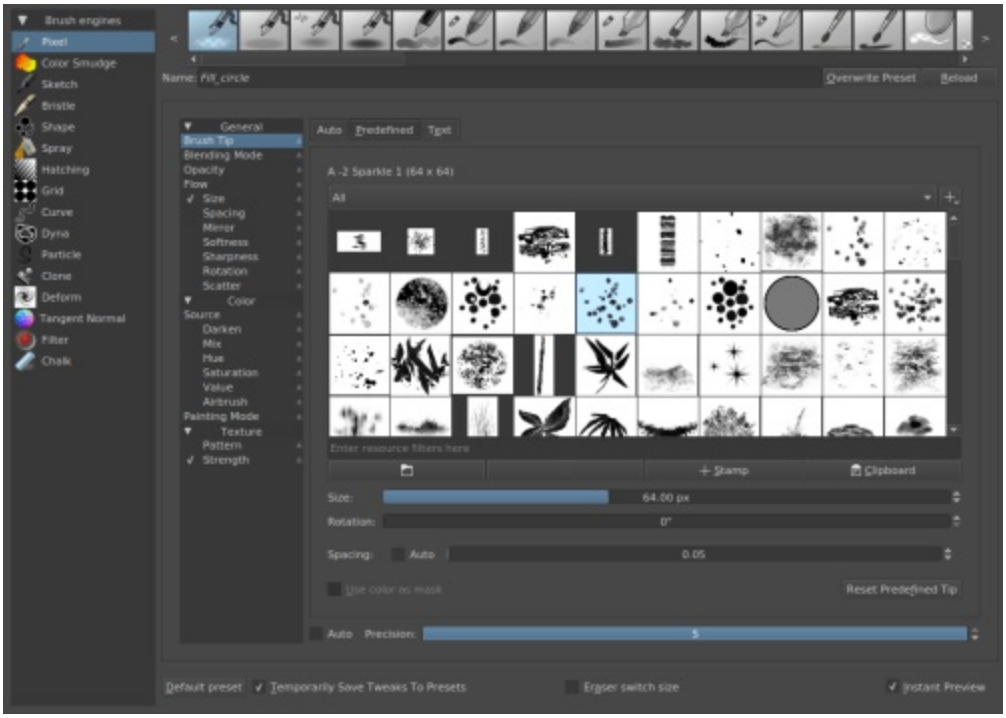
Brush-tips:Hair

Usually, most digital styles tend to focus on simple brushes, like the round brushes, and their usage in hair is no different. So, the typical example would be the one on the left, where we use *fill_round* to draw a silhouette and build up to lighter values.

The reason I use *fill_round* here is because the pressure curve on the size is s-shaped. My tablet has a spring-loaded nib which also causes and s-shaped curve hard-ware wise. This means that it becomes really easy to draw thin lines and big lines. Having a trained inking hand helps a lot with this as well, and it’s something you build up over time.



We then gloss the shadow parties with the *basic_tip_default*. So you can get really far with basic brushes and basic painting skills and indeed I am almost convinced tysontan, who draws our mascot, doesn't use anything but the *basic_tip_default* sometimes.



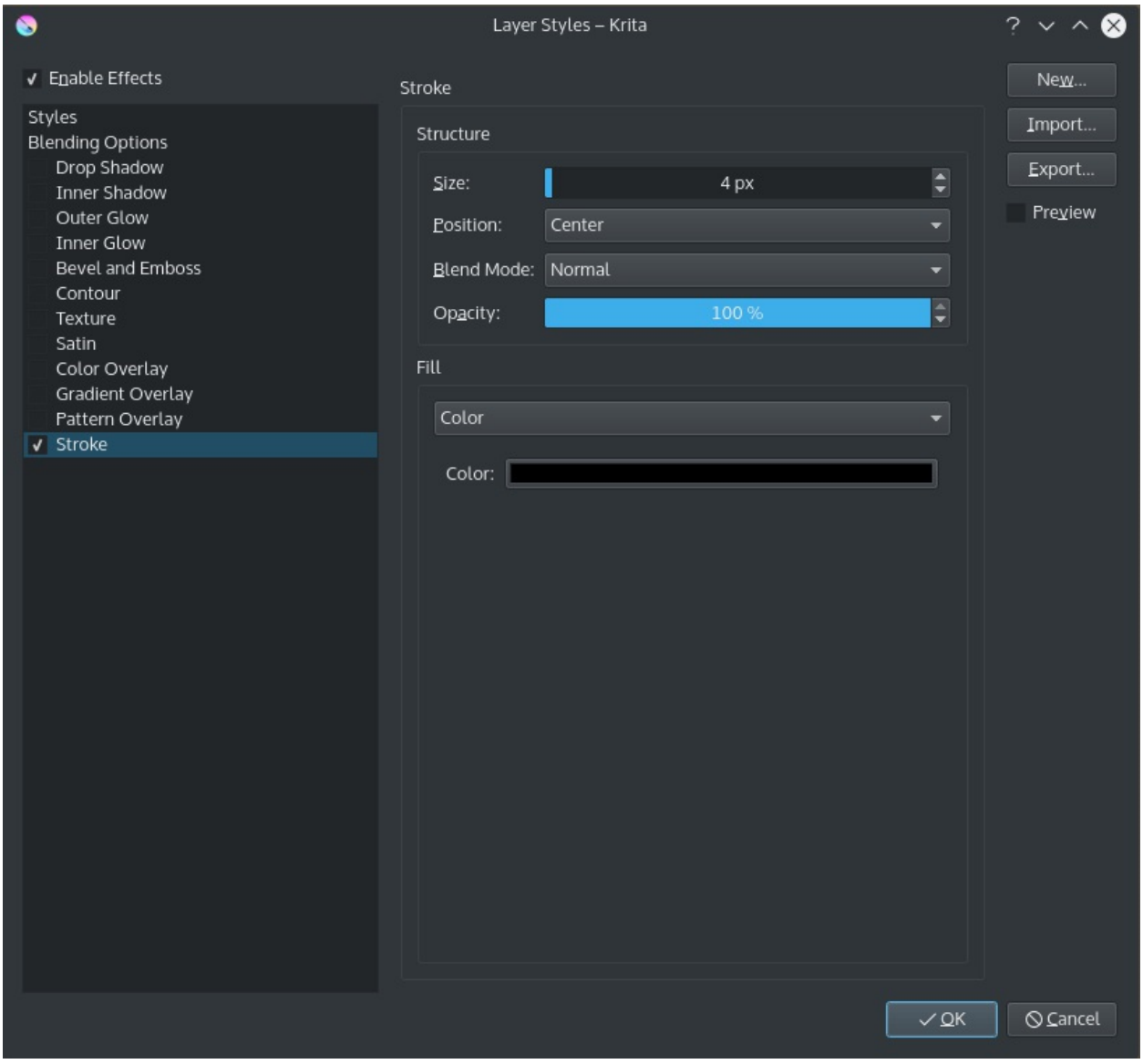
However, if you want an easy hair brush, just take the *fill_round*, go to the brush-tip, pick predefined and select *A2-sparkle-1* as the brush tip. You can fiddle with the spacing below the selection of predefined brushtip to space the brush, but I believe the default should be fine enough to get result.

Brush-tips:Outline

Question

How to make an outline for a single brush stroke using Krita?

Not really a brush, but what you can do is add a layer style to a layer, by a layer and selecting layer style. Then input the following settings:



Then, set the main layer to multiply (or add a Color to Alpha filter mask), and paint with white:

(The white thing is the pop-up that you see as you hover over the layer.)

Merge into a empty clear layer after ward to fix all the effects.

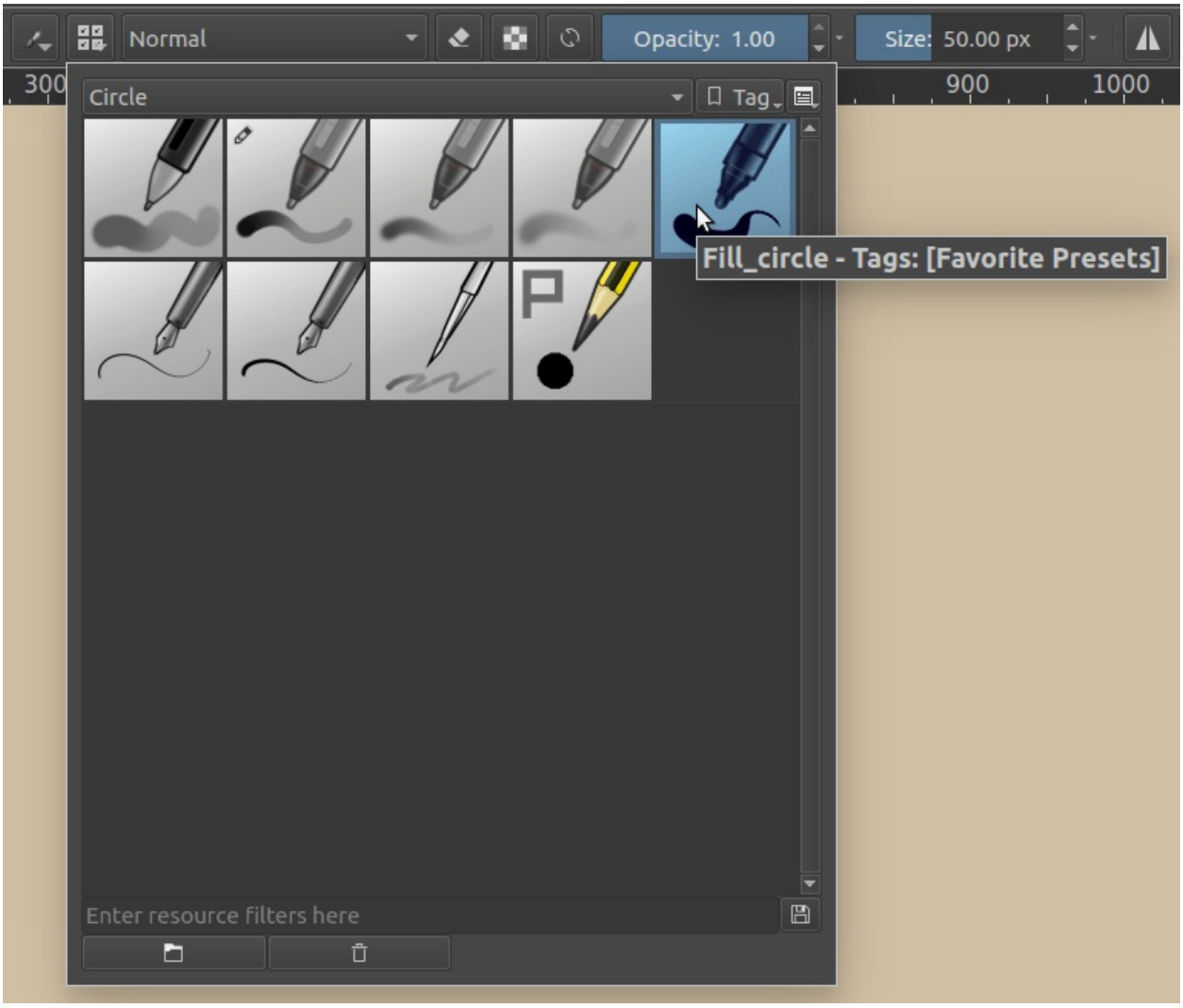
Brush-tips:Rainbow Brush

Question

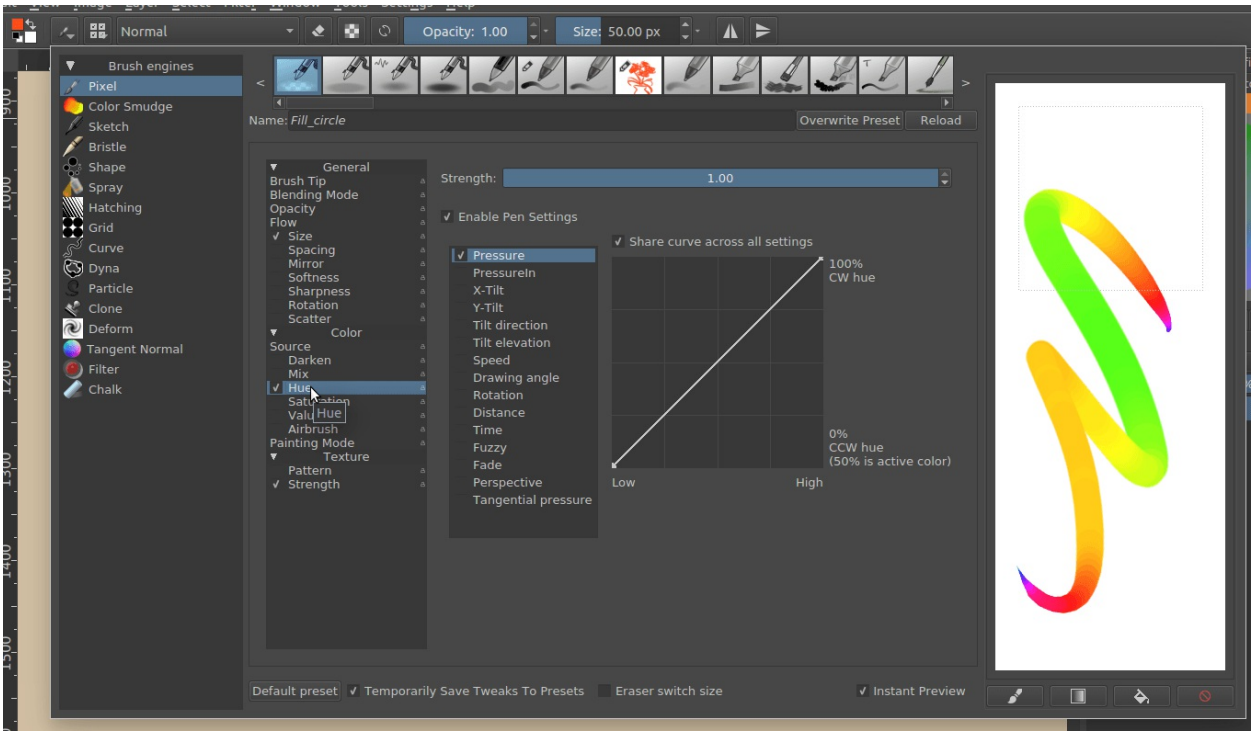
Hello, there is a way to paint with rainbow on Krita?

Yes there is.

First, select the fill_circle:



Then, press the `F5` key to open the brush editor, and toggle **Hue**.

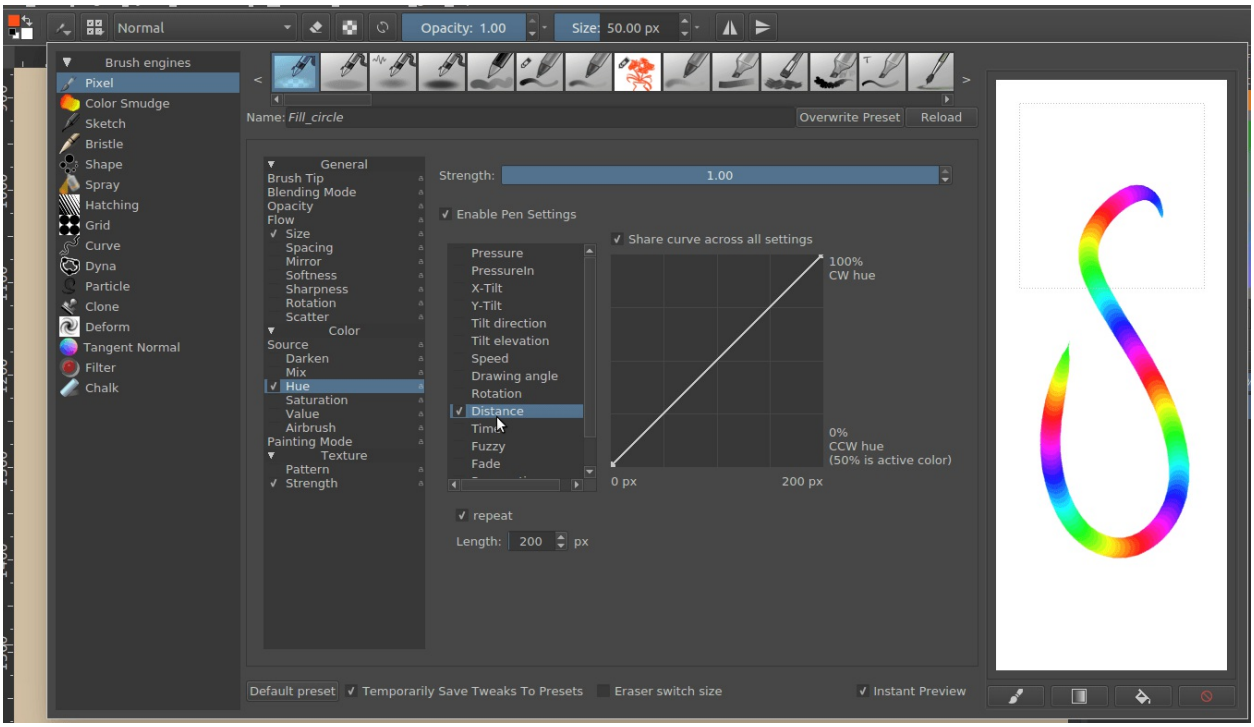


This should allow you to change the color depending on the pressure.

Caution

The brightness of the rainbow is relative to the color of the currently selected color, so make sure to select bright saturated colors for a bright rainbow!

Uncheck **Pressure** and check **Distance** to make the rainbow paint itself over distance. The slider below can be  to change the value with keyboard input.



When you are satisfied, give the brush a new name and save it.

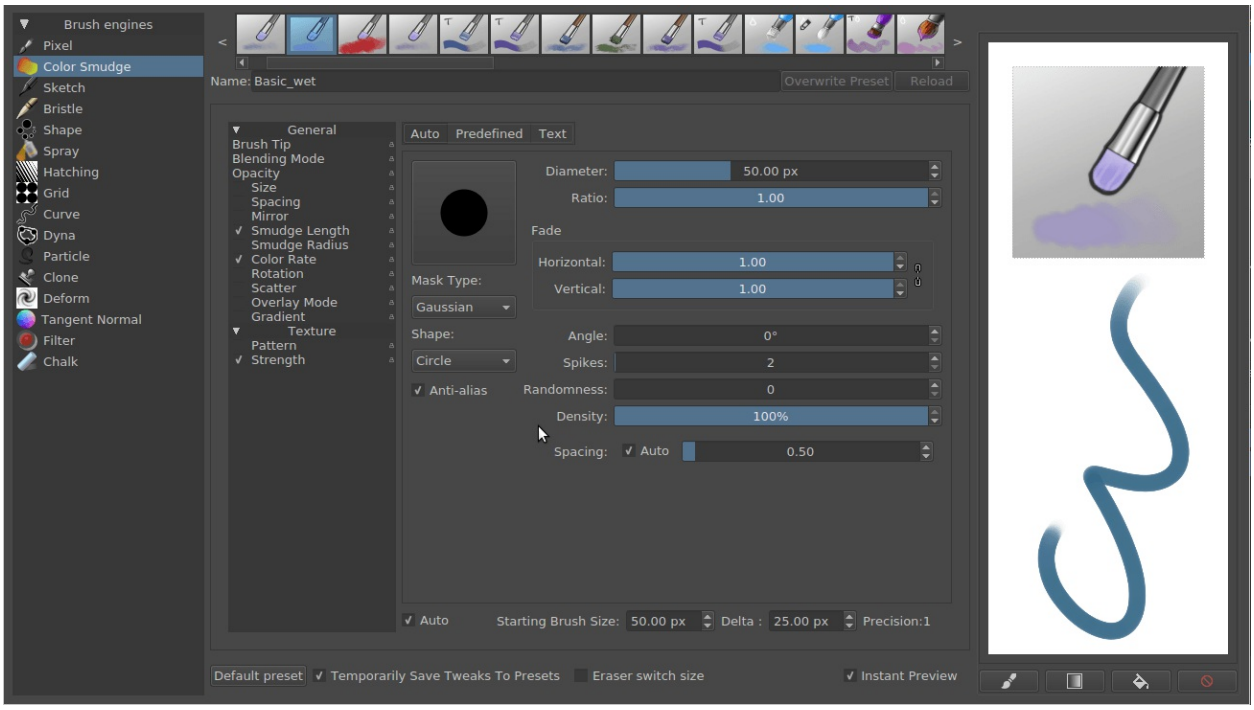
Brush-tips:Sculpt-paint-brush

Question

How do I make a brush like the one in Sinix's paint-like-a-sculptor video?

It's actually quite easy, but most easy to do since Krita 3.0 due a few bugfixes.

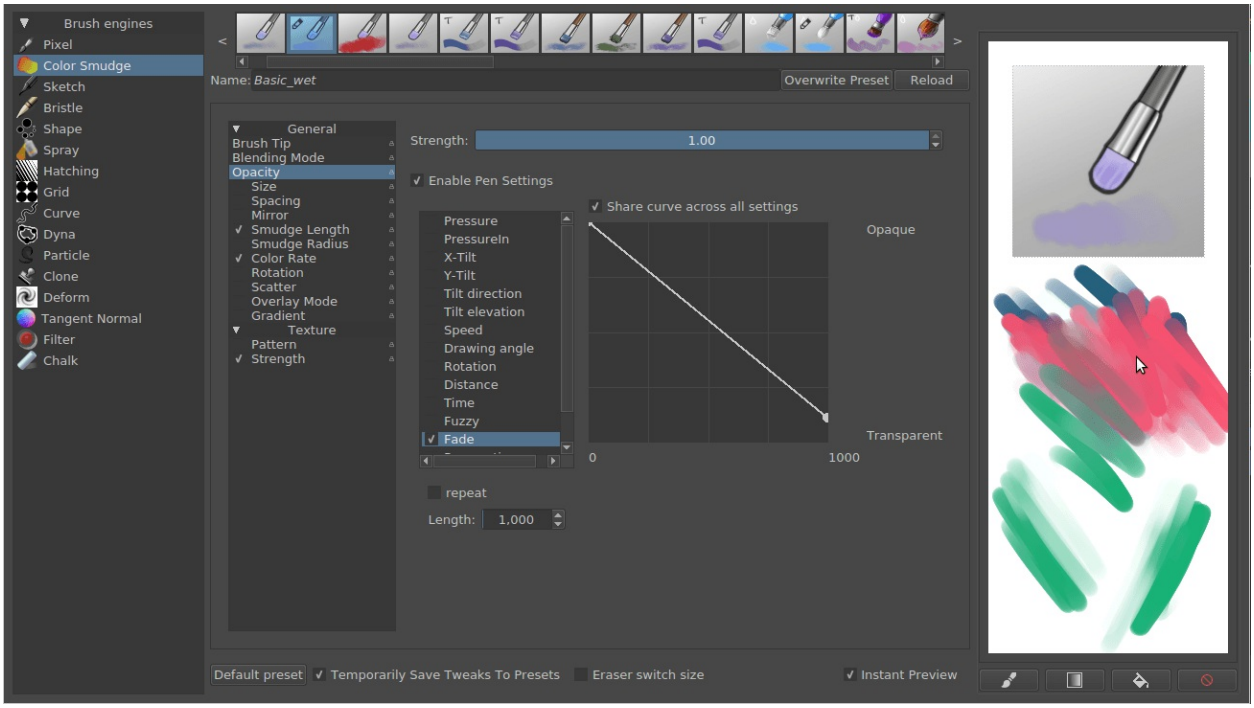
First, select *Basic_Wet* from the default presets, and go into the brush editor with the `F5` key.



Then, the trick is to go into **Opacity**, untoggle **Pressure** from the sensors, toggle **Fade** and then reverse the curve as shown above. Make sure that the curve ends a little above the bottom-right, so that you are always painting something. Otherwise, the smudge won't work.

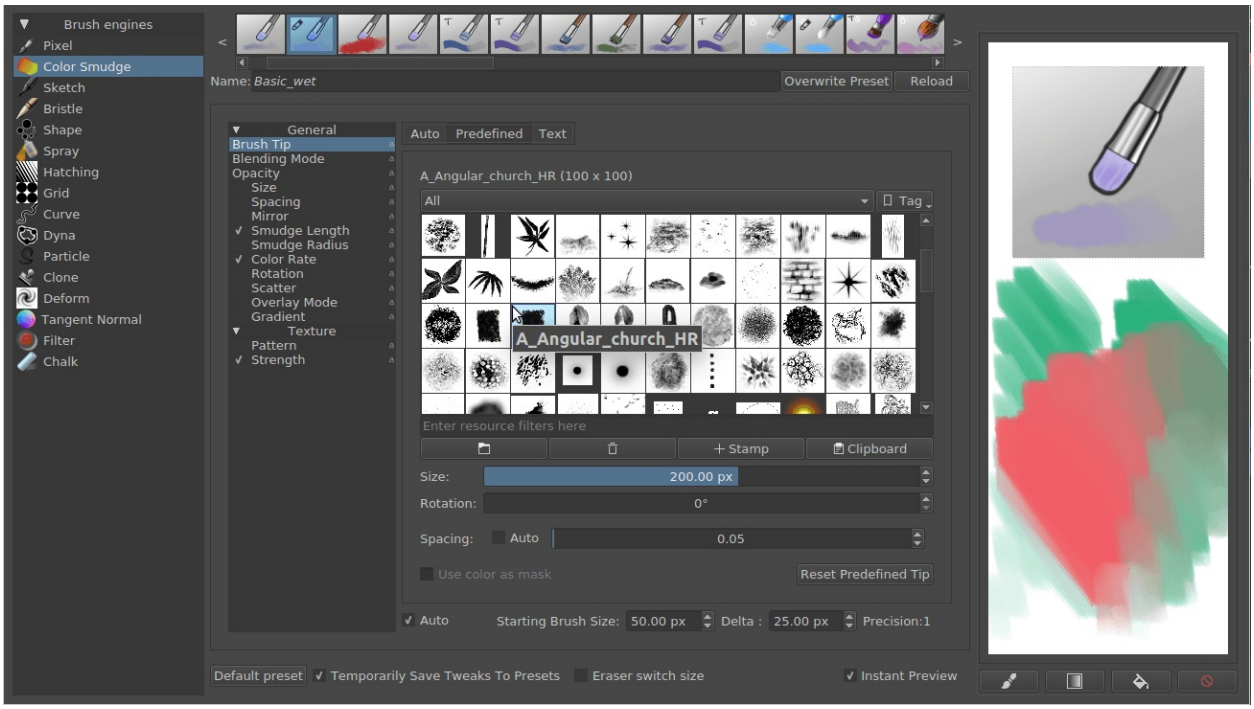
This'll make the color rate decrease and turn it into a smudge brush as the

stroke continues:



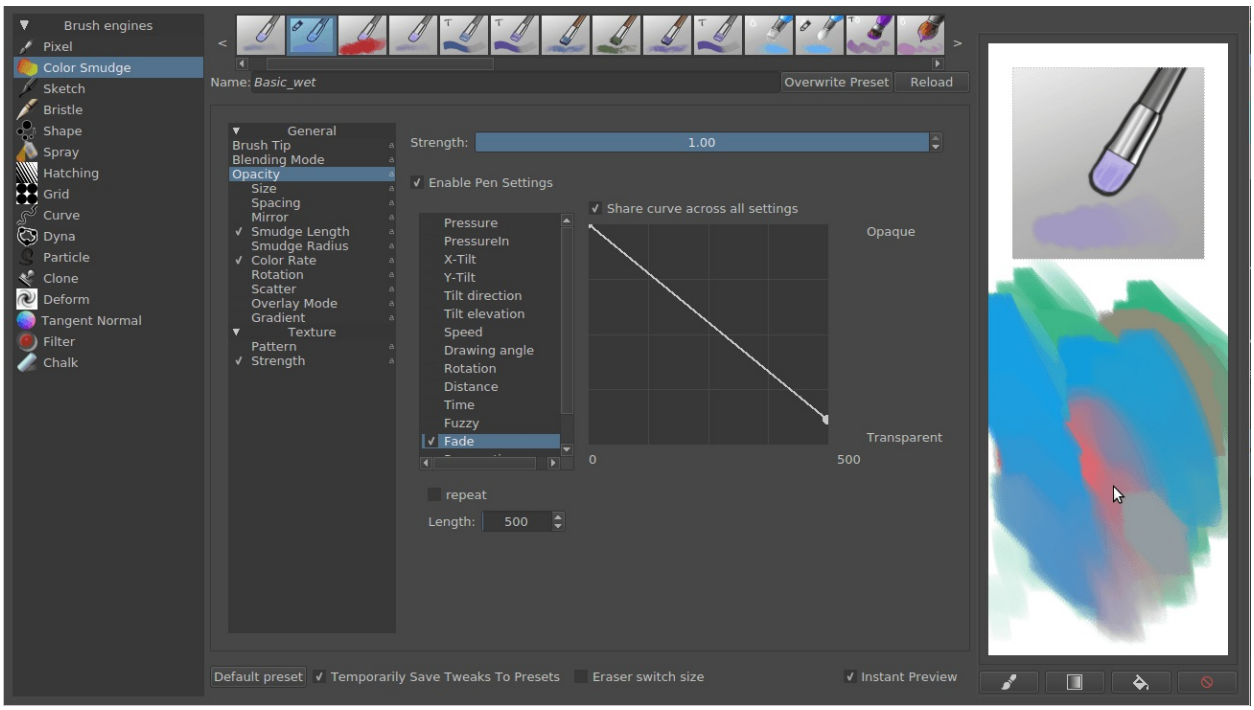
The **Fade** sensor will base the stroke length on brush size. The **Distance** sensor will base it on actual pixels, and the **Time** on actual seconds.

Then, select *Brushtip* ▸ *Predefined* and select the default *A_Angular_Church_HR* brushtip.



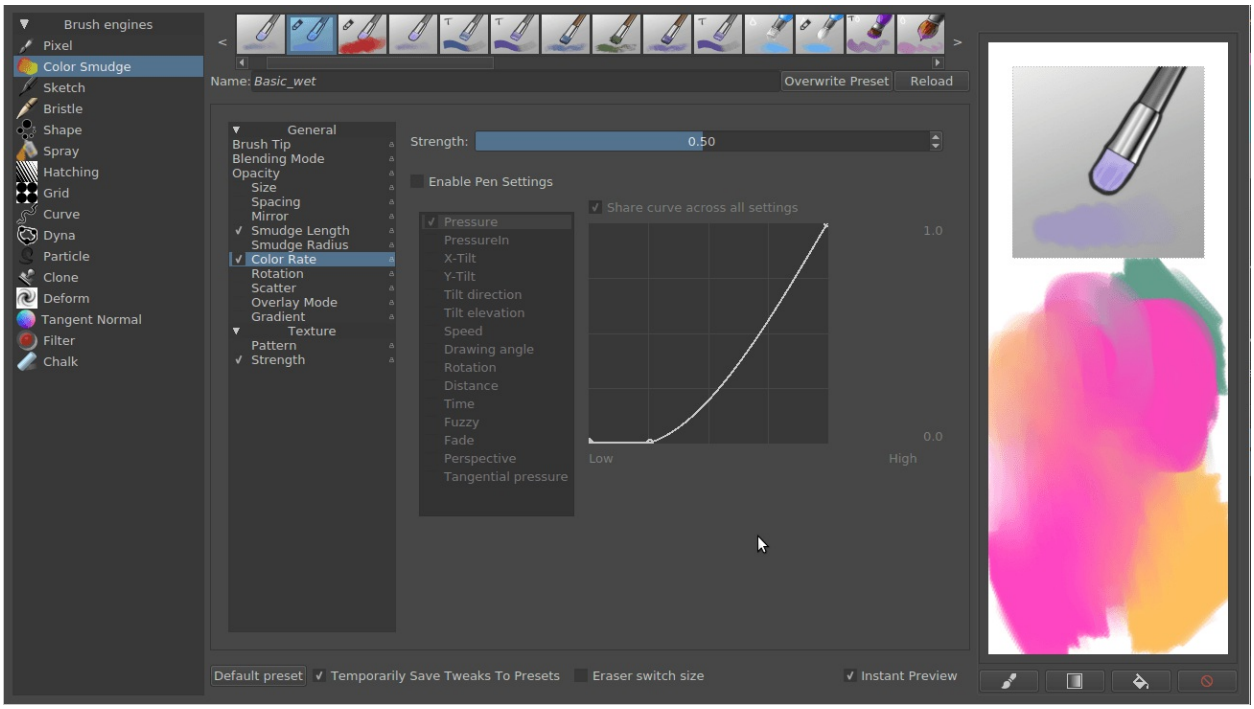
This makes for a nice textured square brush.

Of course, this'll make the stroke distance longer to get to smudging, so we go back to the *Opacity*.

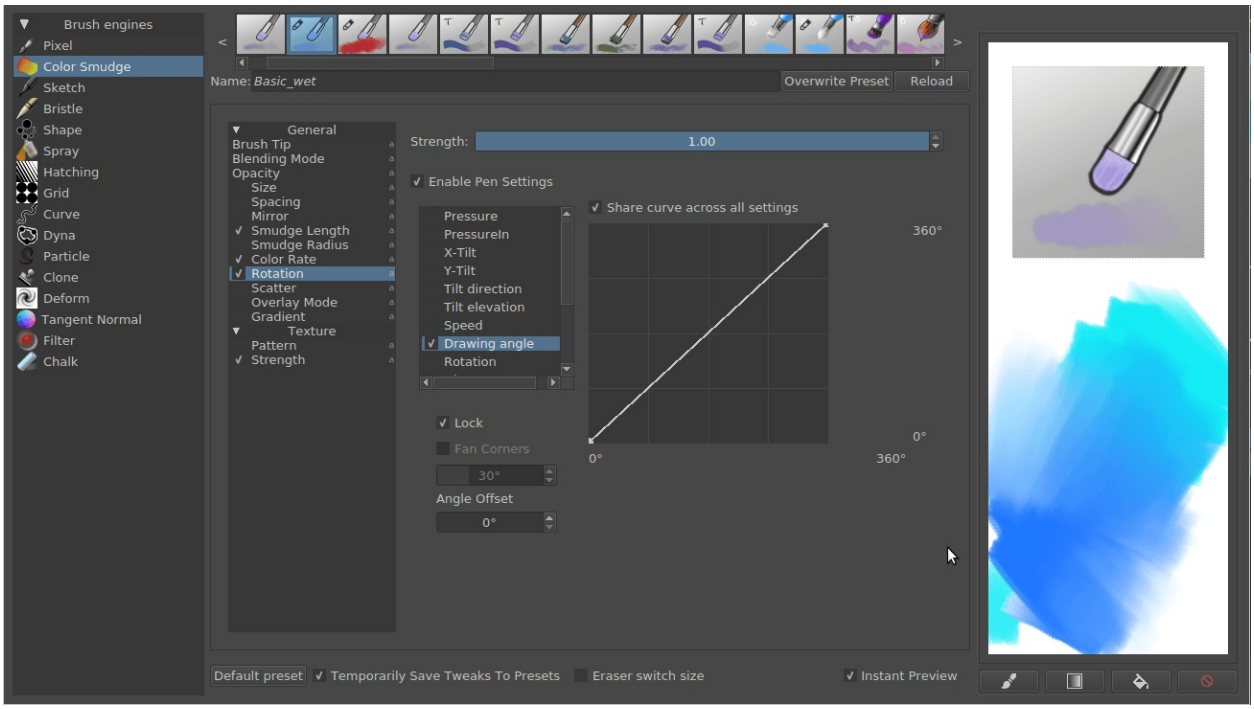


Just adjust the fade-length by  on the slider bar. You can then input a number. In the screenshot, I have 500, but the sweet spot seems to be somewhere between 150 and 200.

Now, you'll notice that on the start of a stroke, it might be a little faded, so go into **Color Rate** and turn off the **Enable Pen Settings** there.



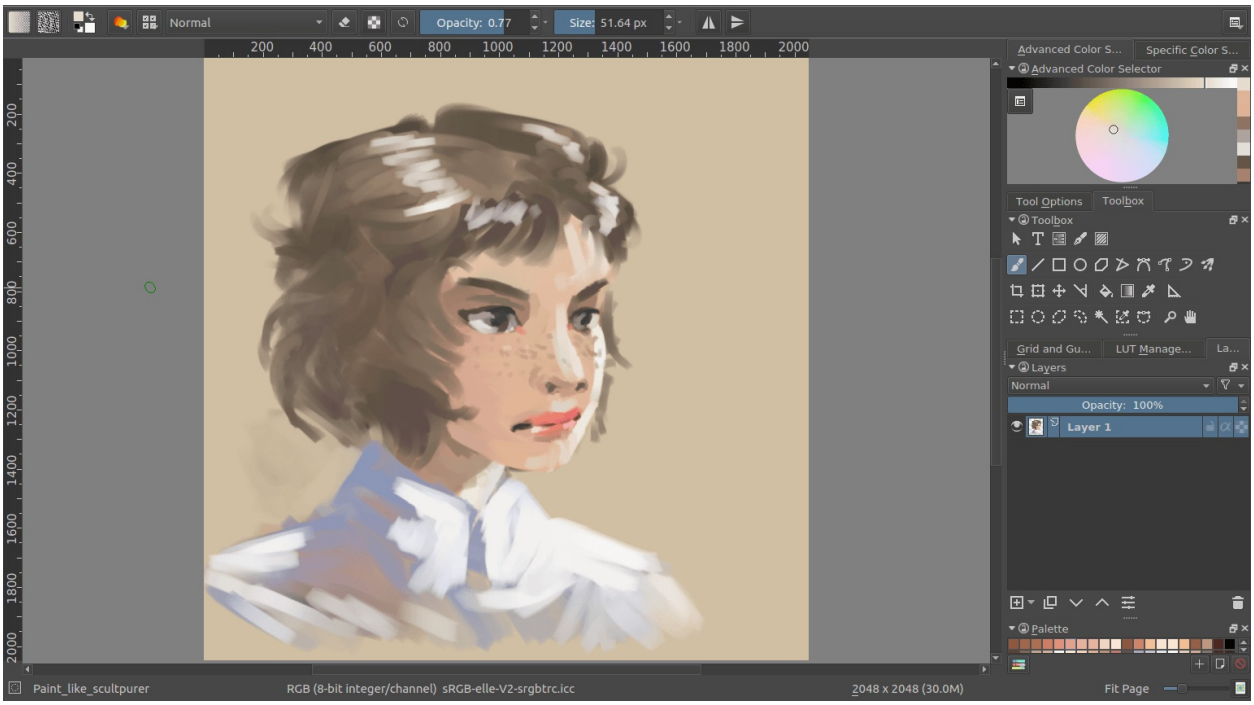
Then, finally, we'll make the brush rotate.



Tick the **Rotation** parameter, and select it. There, untick **Pressure** and tick **Drawing Angle**.

Then, for better angling, tick **Lock** and set the **Angle Offset** to 90 degrees by  the slider bar and typing in 90.

Now, give your brush a new name, doodle on the brush-square, **Save to presets** and paint!


Normal
Opacity: 0.77
Size: 51.64 px
Advanced Color S...
Specific Color S...
Advanced Color Selector
Tool Options
Toolbox
Toolbox
Grid and Gu...
LUT Manage...
La...
Layers
Normal
Opacity: 100%
Layer 1
Palette
Paint_like_scultpurer
RGB (8-bit integer/channel) sRGB-elle-V2-srgbtrc.icc
2048 x 2048 (30.0M)
Fit Page

Making An Azalea With The Transformation Masks

Note

This page was ported from the original post on the main page to KDE

UserBase wiki.

Okay, so I've wanted to do a tutorial for transform masks for a while now, and this is sorta ending up to be a flower-drawing tutorial. Do note that this tutorial requires you to use **Krita 2.9.4 at MINIMUM**. It has a certain speed-up that allows you to work with transform masks reliably!

I like drawing flowers because they are a bit of an unappreciated subject, yet allow for a lot of practice in terms of rendering. Also, you can explore cool tricks in Krita with them.

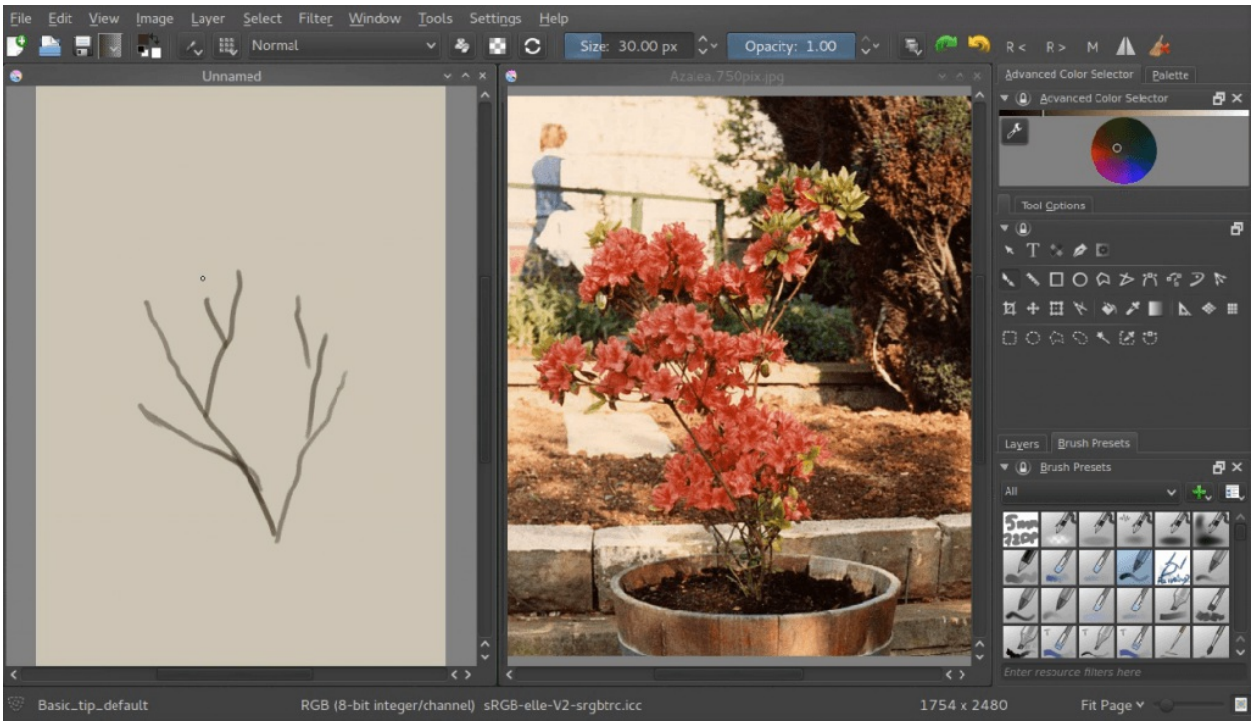
Today's flower is the Azalea flower. These flowers are usually pink to red and appear in clusters, the clusters allow me to exercise with transform masks!

I got an image from Wikipedia for reference, mostly because it's public domain, and as an artist I find it important to respect other artists. You can copy it and, if you already have a canvas, *Edit* ▸ *Paste into New Image* or *New* ▸ *Create from Clipboard*.

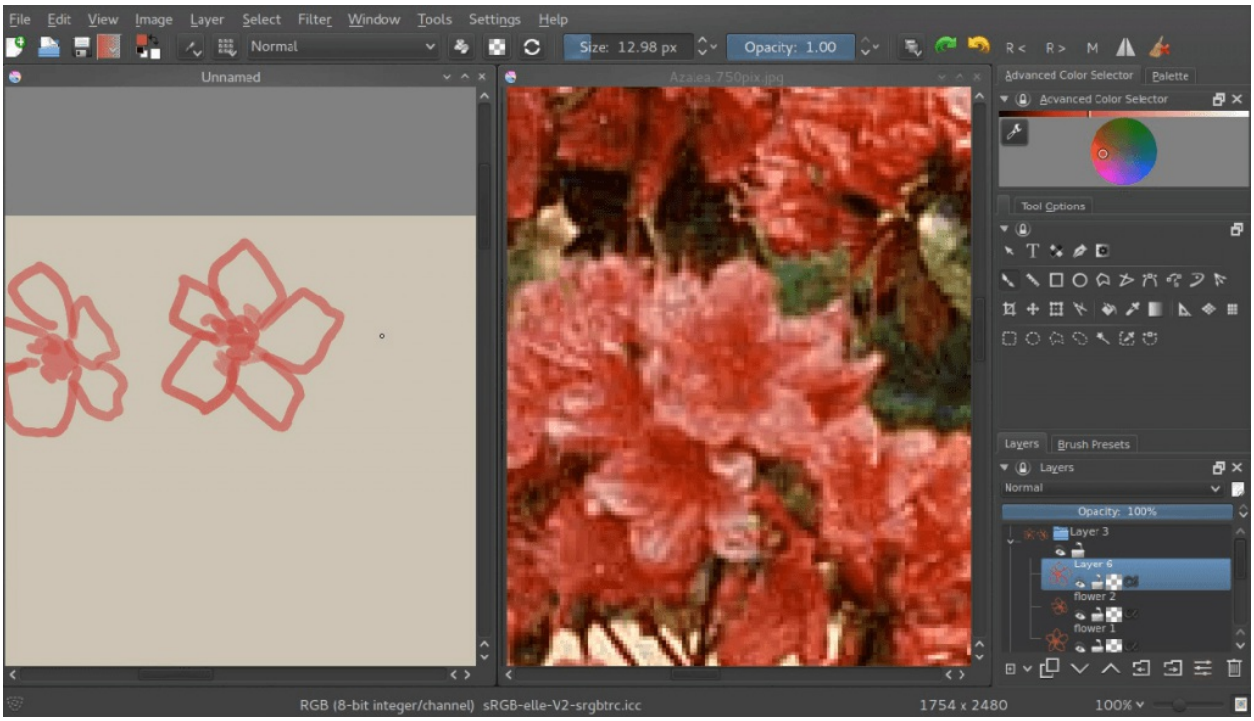
Then, if you didn't have a new canvas make one. I made an A5 300dpi canvas. This is not very big, but we're only practicing. I also have the background color set to a yellow-grayish color (#CAC5B3), partly because it reminds me of paper, and partly because bright screen white can strain the eyes and make it difficult to focus on values and colors while painting. Also, due to the lack of strain on the eyes, you'll find yourself soothed a bit. Other artists use #c0c0c0, or even more different values.

So, if you go to *Window* ▸ *Tile*, you will find that now your reference image and your working canvas are side by side. The reason I am using this instead of the docker is because I am lazy and don't feel like saving the wikipedia image. We're not going to touch the image much.

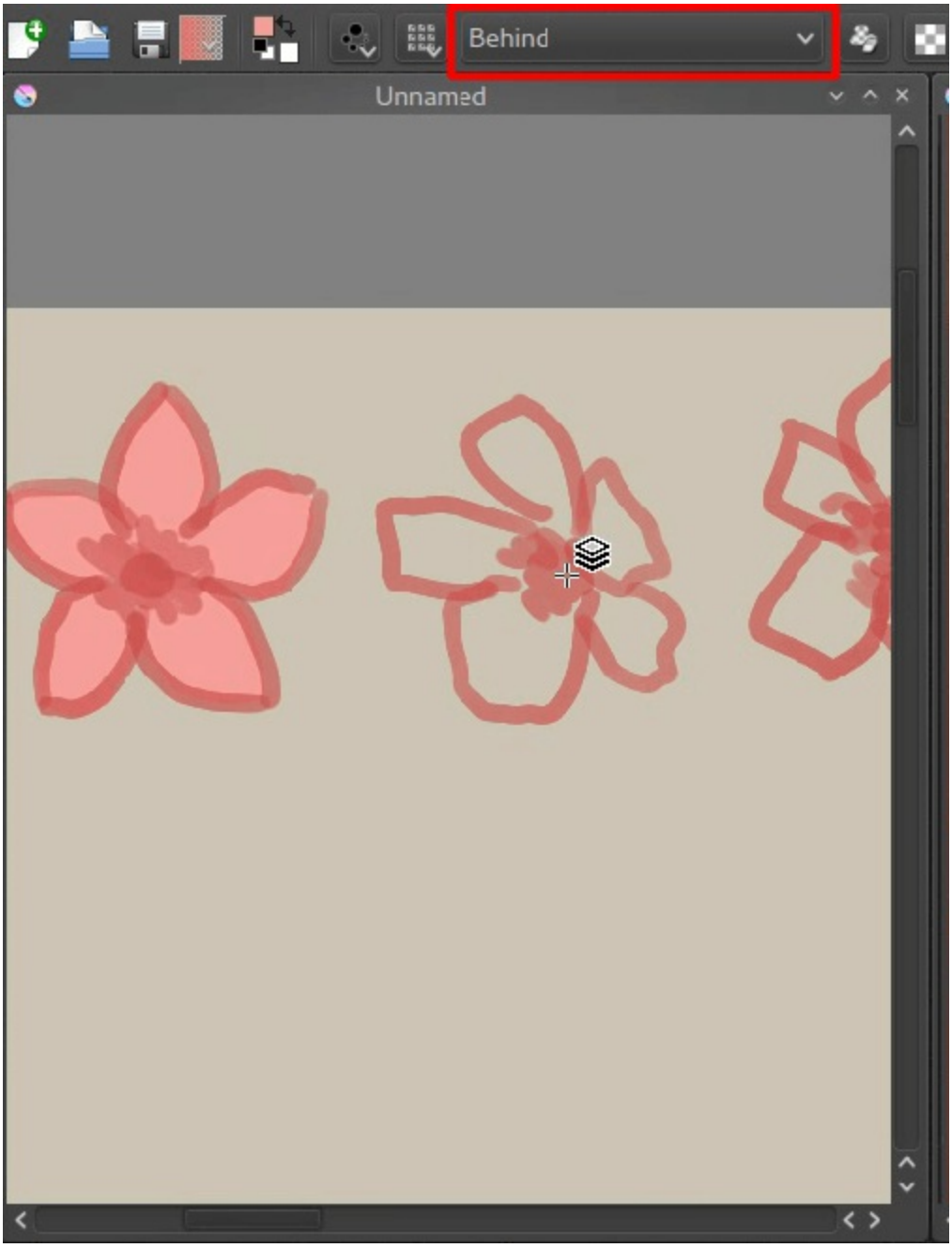
Let's get to drawing!



First we make a bunch of branches. I picked a slightly darker color here than usual, because I know that I'll be painting over these branches with the lighter colors later on. Look at the reference how branches are formed.

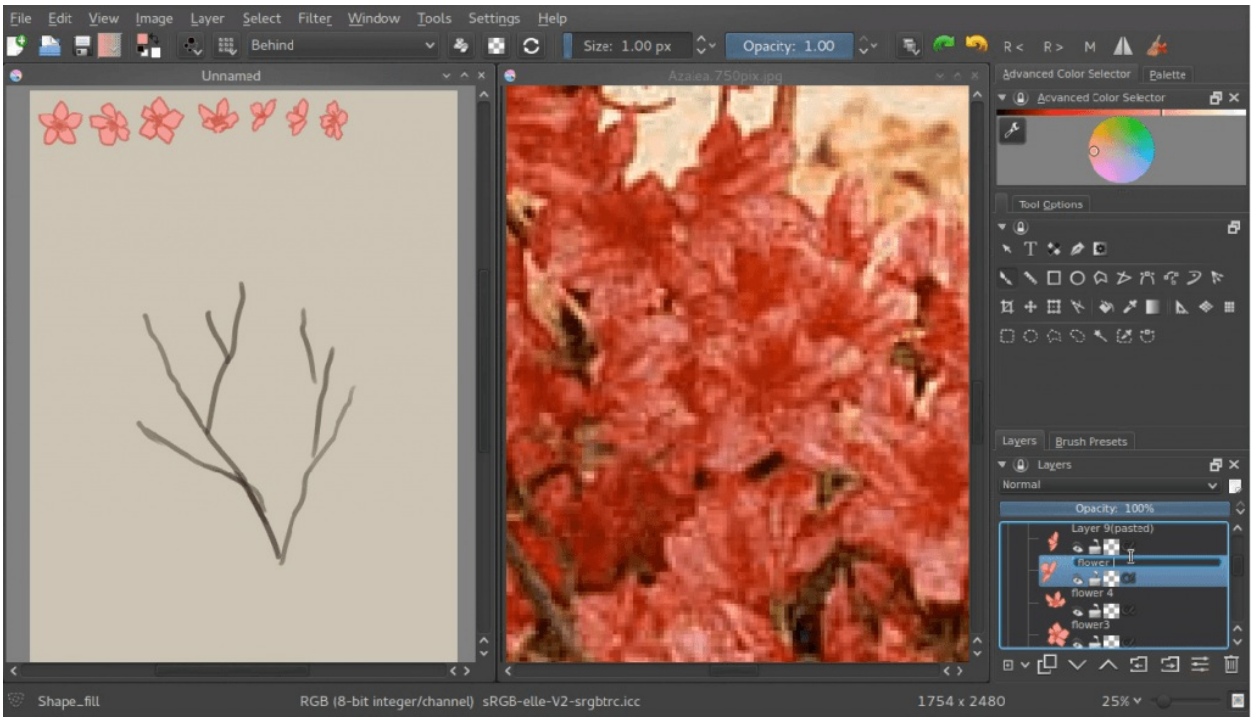


Then we make an approximation of a single flower on a layer. We make a few of these, all on separate layers. We also do not color pick the red, but we guess at it. This is good practice, so we can learn to analyze a color as well as how to use our color selector. If we'd only pick colors, it would be difficult to understand the relationship between them, so it's best to attempt matching them by eye.



I chose to make the flower shape opaque quickly by using the *behind*

blending mode. This'll mean Krita is painting the new pixels behind the old ones. Very useful for quickly filling up shapes, just don't forget to go back to *normal* once you're done.

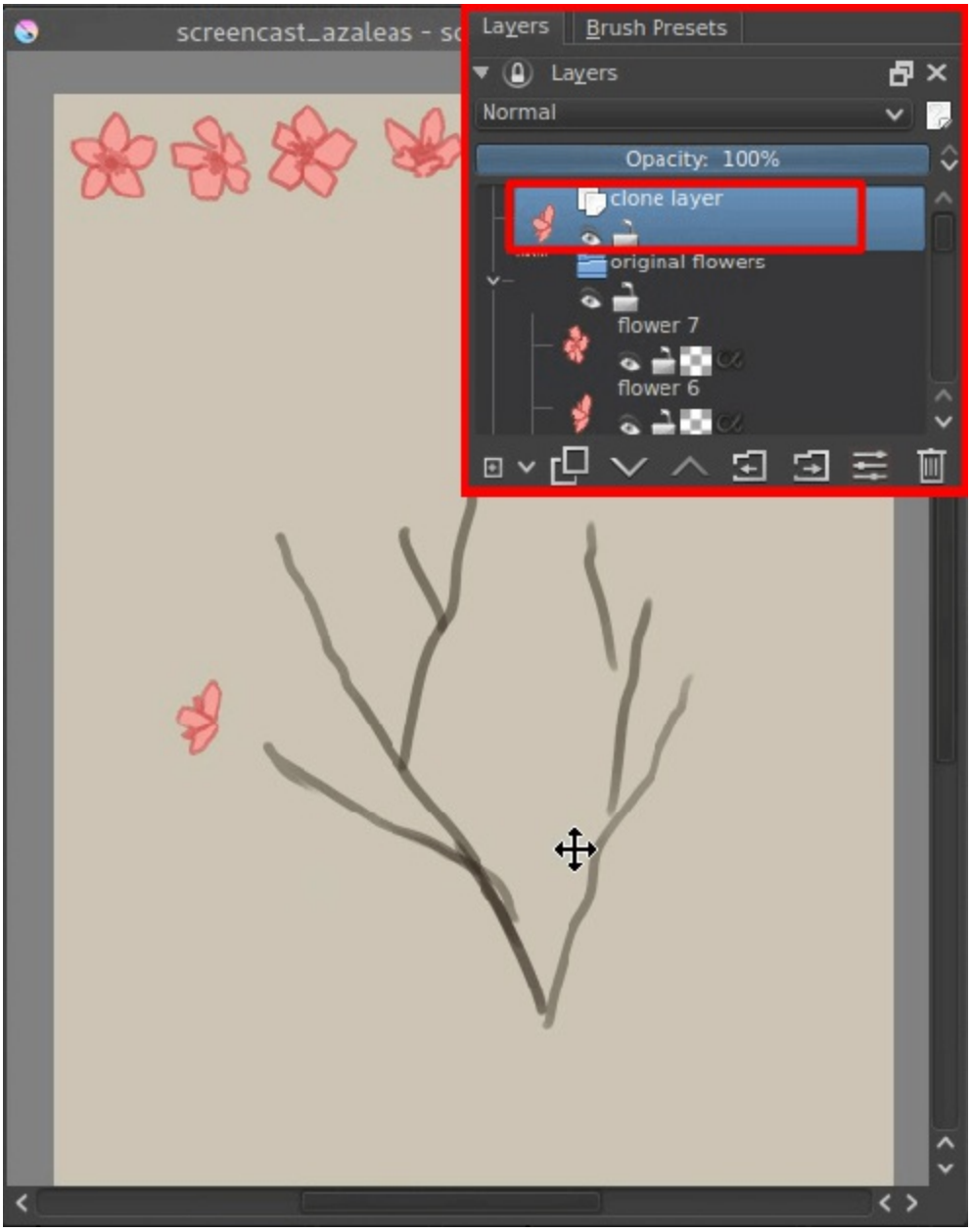


Now, we'll put the flowers in the upper left corner, and group them. You can group by making a group layer, and selecting the flower layers in your docker with the `Ctrl` +  shortcut and dragging them into the group. The reason why we're putting them in the upper left corner is because we'll be selecting them a lot, and Krita allows you to select layers with the `R` +  shortcut on the canvas quickly. Just hold the `R` key and  the pixels belonging to the layer you want, and Krita will select the layer in the Layer docker.

Clone Layers

Now, we will make clusters. What we'll be doing is that we select a given flower and then make a new clone layer. A clone layer is a layer that is literally a clone of the original. They can't be edited themselves, but edit the original and the clone layer will follow suit. Clone Layers, and File layers, are our greatest friends when it comes to transform masks, and you'll see

why in a moment.



You'll quickly notice that our flowers are not good enough for a cluster: we need far more angles on the profile for example. If only there was a way to transform them… but we can't do that with clone layers. Or can we?

Enter Transform Masks!

Transform Masks are a really powerful feature introduced in 2.9. They are in

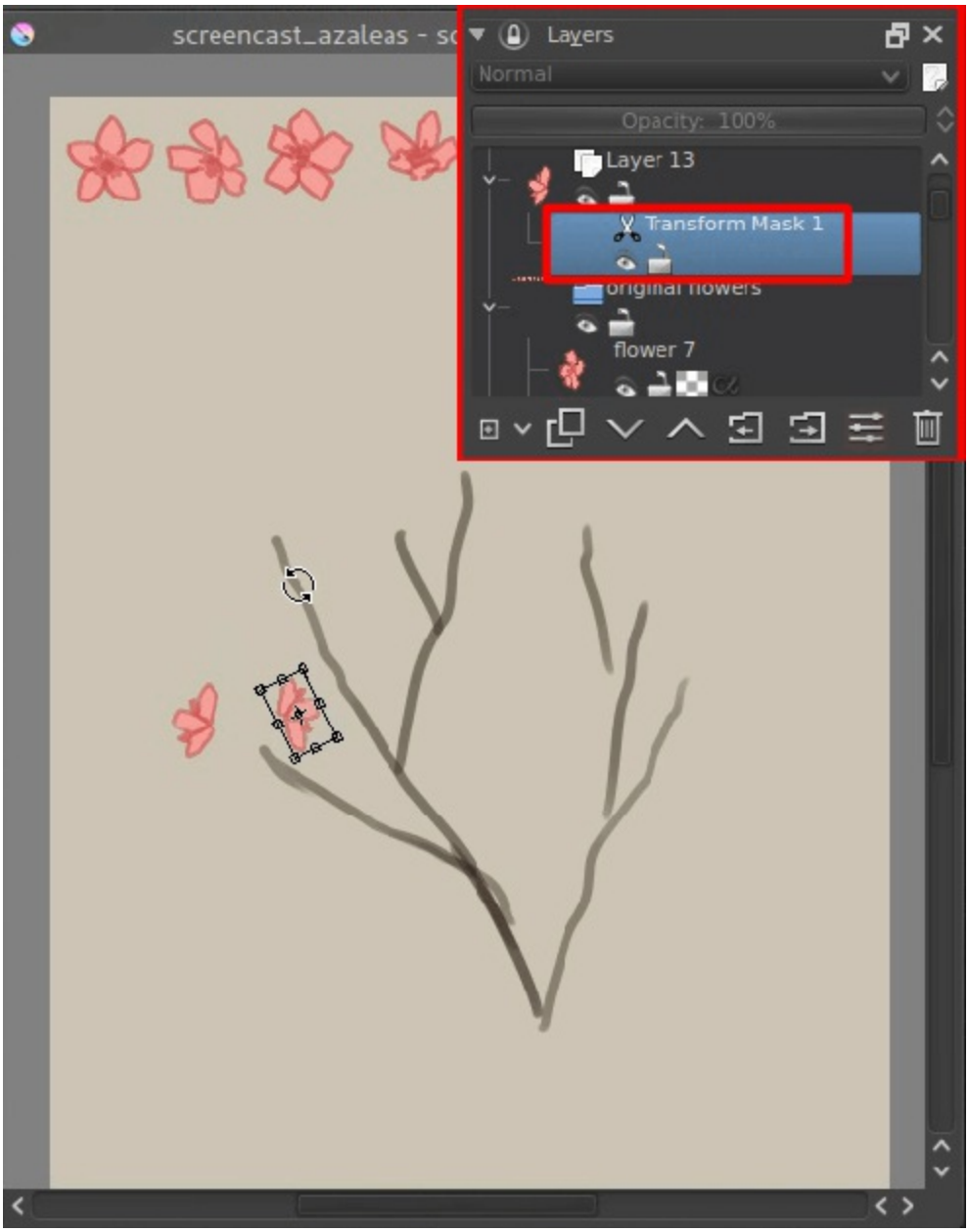
fact so powerful, that when you first use them, you can’t even begin to grasp where to use them.

Transform masks allow us to do a transform operation onto a layer, any given layer, and have it be completely dynamic! This includes our clone layer flowers!

How to use them:

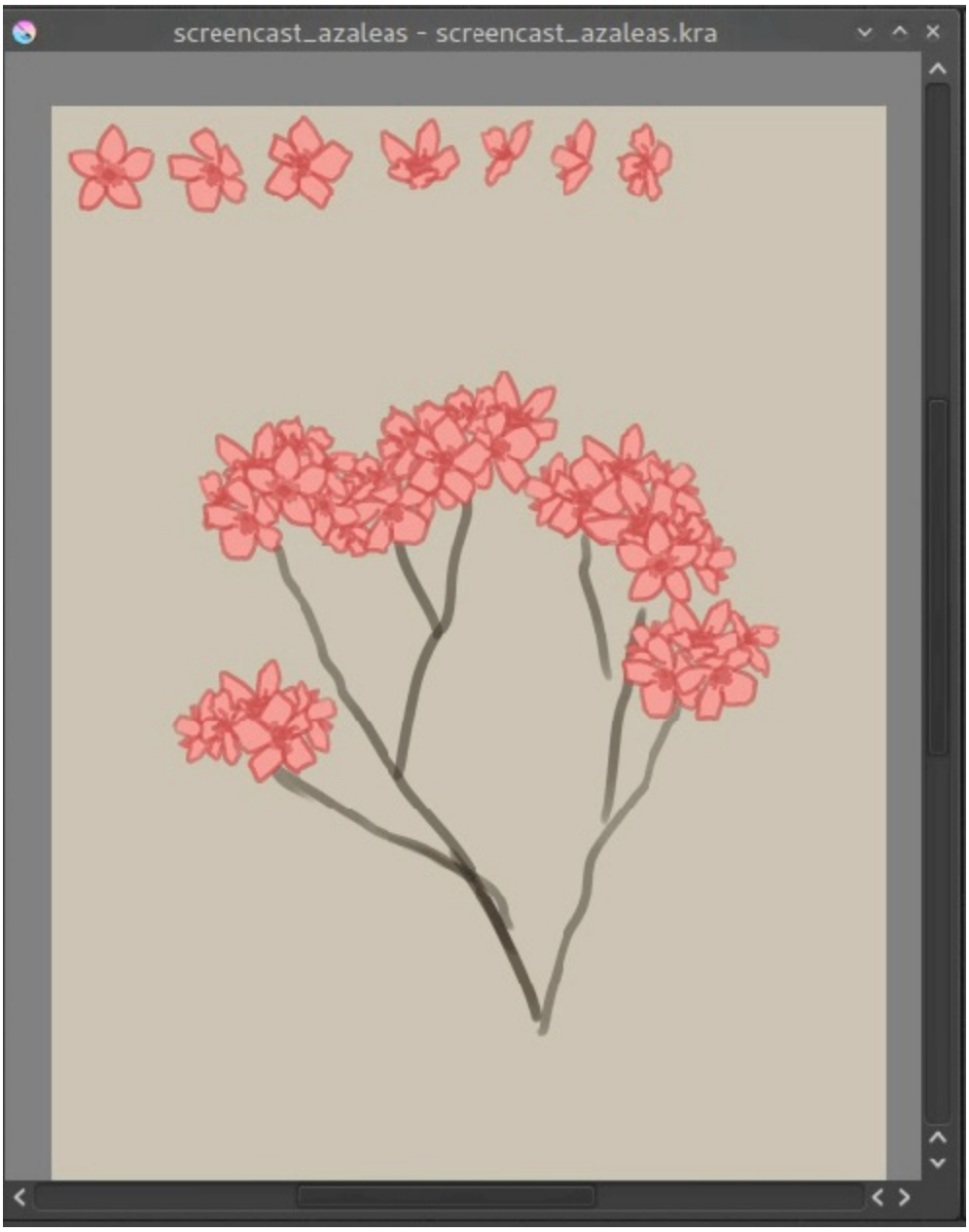
 the layer you want to do the transform on, and add a **Transform mask**.

A transform mask should now have been added. You can recognize them by the little ‘scissor’ icon.



Now, with the transform mask selected, select the  , and rotate our clone layer. Apply the transform. You know you're successful when you can hide the transform mask, and the layer goes back to its original state!

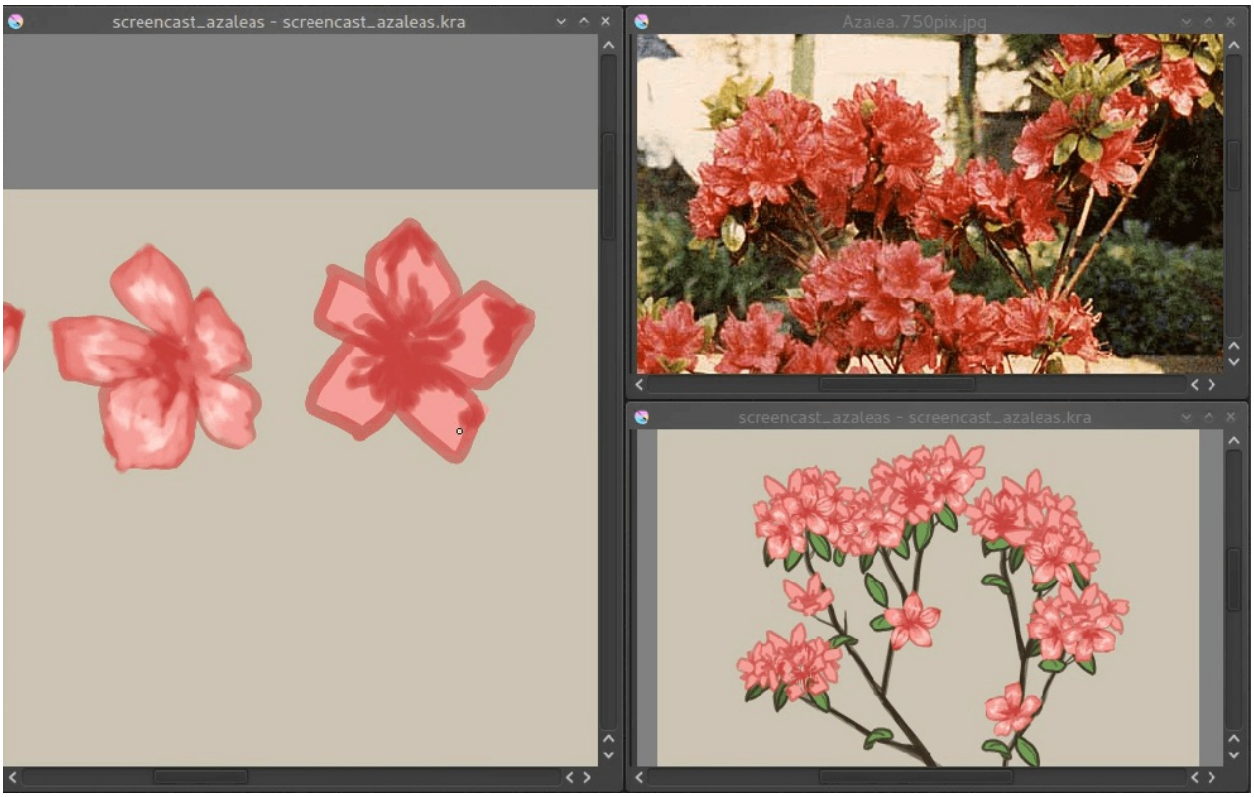
You can even go and edit your transform! Just activate the  again while on a transform mask, and you will see the original transform so you can edit it. If you go to a different transform operation however, you will reset the transform completely, so watch out.



We’ll be only using affine transformations in this tutorial (which are the regular and perspective transform), but this can also be done with warp, cage and liquify, which’ll have a bit of a delay (3 seconds to be precise). This is to prevent your computer from being over-occupied with these more complex transforms, so you can keep on painting.

We continue on making our clusters till we have a nice arrangement.

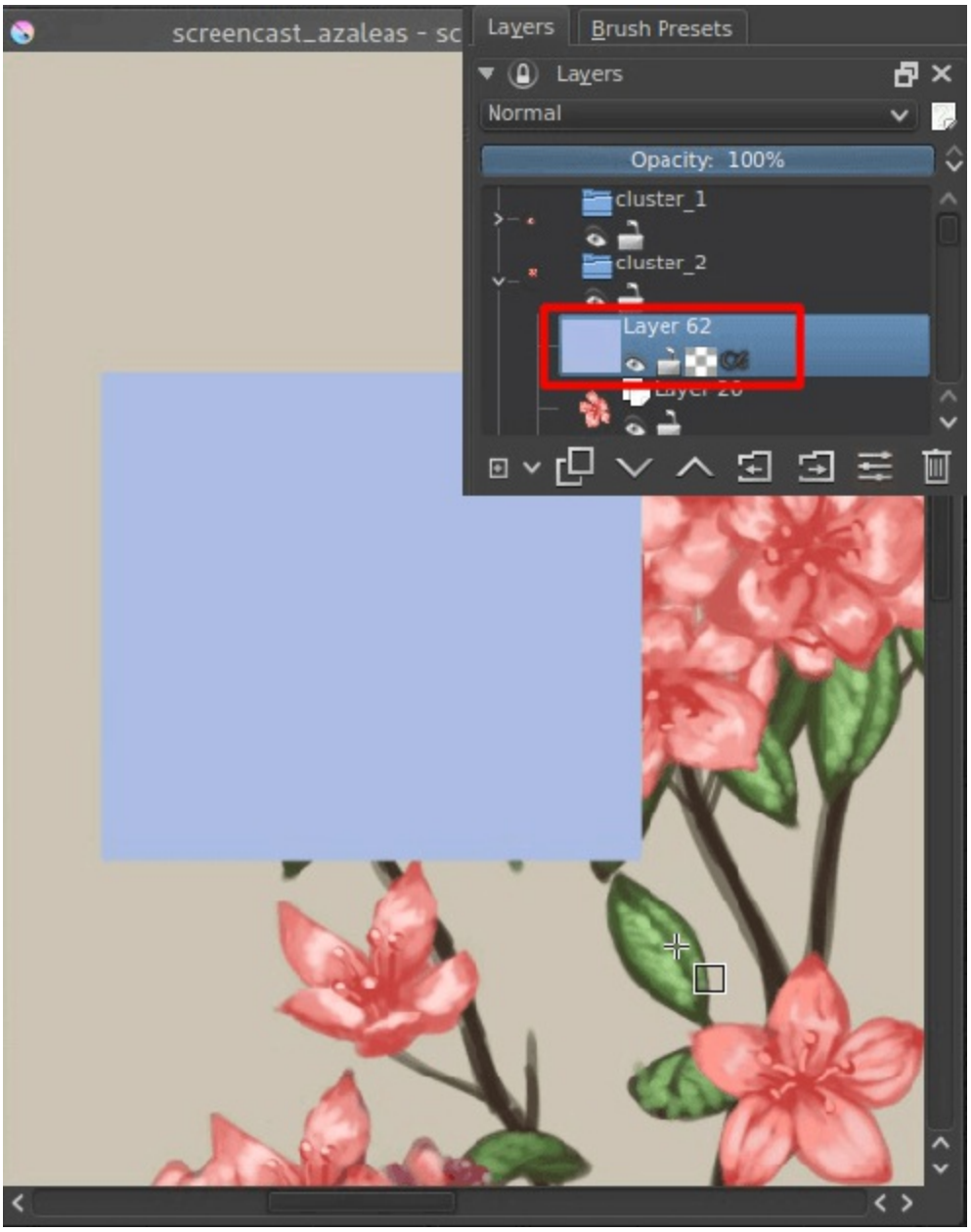
Now do the same thing for the leaves.



Now, if you select the original paint layers and draw on them, you can see that all clone masks are immediately updated!

Above you can see there's been a new view added so we can focus on painting the flower and at the same time see how it'll look. You can make a new view by going *Window* ▸ *New View* and selecting the name of your current canvas (save first!). Views can be rotated and mirrored differently.

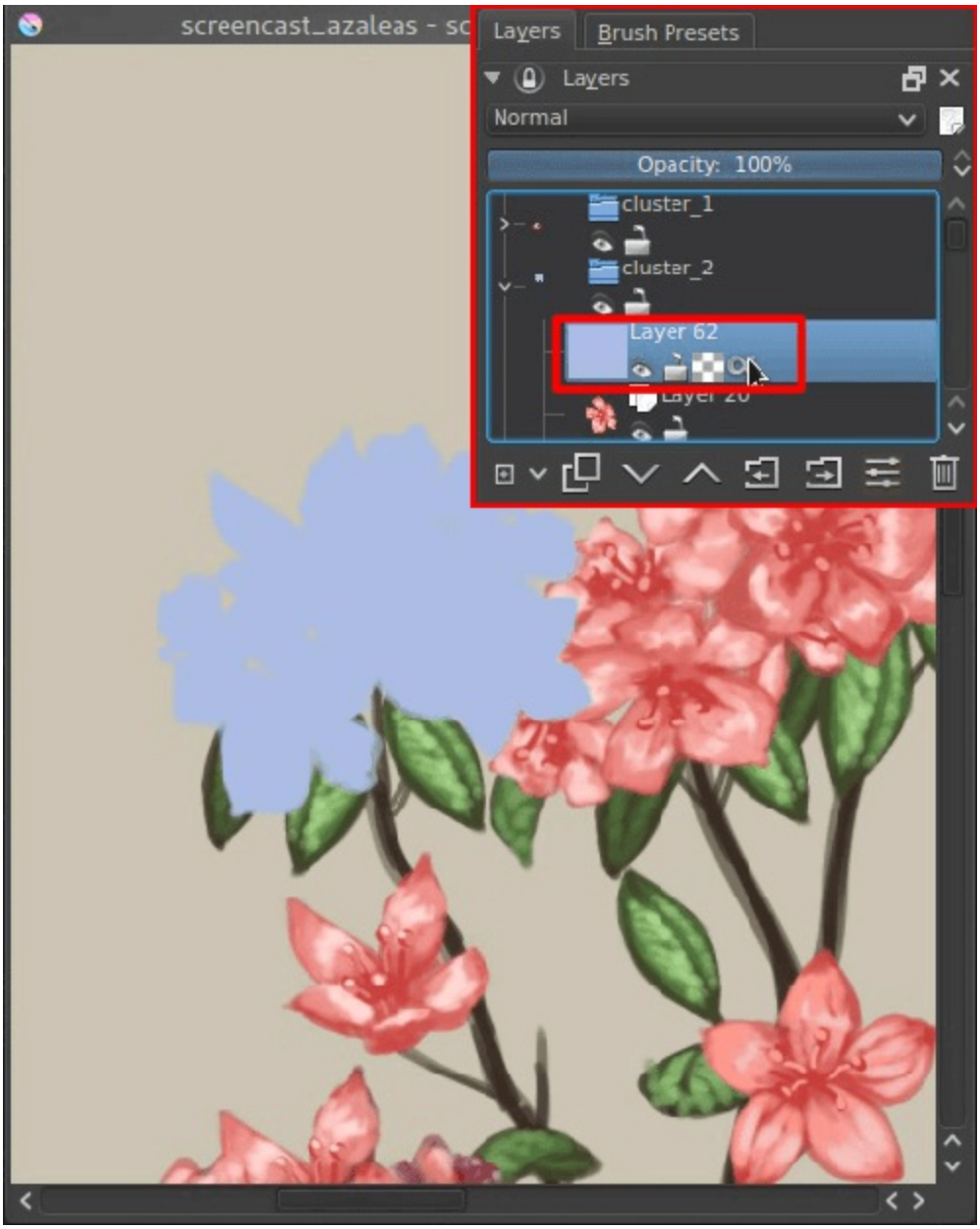
Now continue painting the original flowers and leaves, and we'll move over to adding extra shadow to make it seem more lifelike!



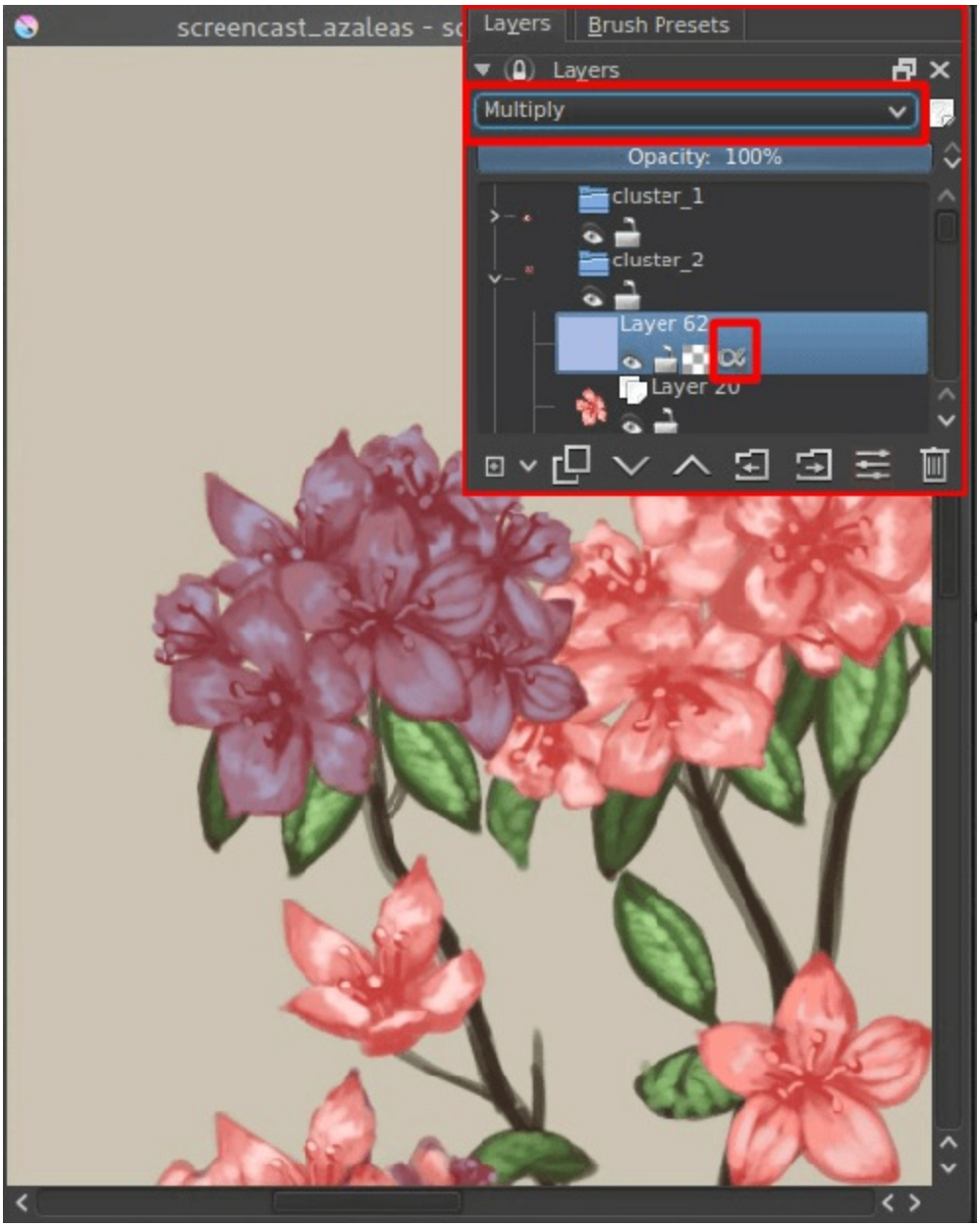
We’re now going to use *Alpha Inheritance*. Alpha inheritance is an ill-understood concept, because a lot of programs use *clipping masks* instead, which clip the layer’s alpha using only the alpha of the first next layer.

Alpha inheritance, however, uses all layers in a stack, so all the layers in the group that haven’t got alpha inheritance active themselves, or all the layers in the stack when the layer isn’t in a group. Because most people have an opaque layer at the bottom of their layer stack, alpha inheritance doesn’t seem to do much.

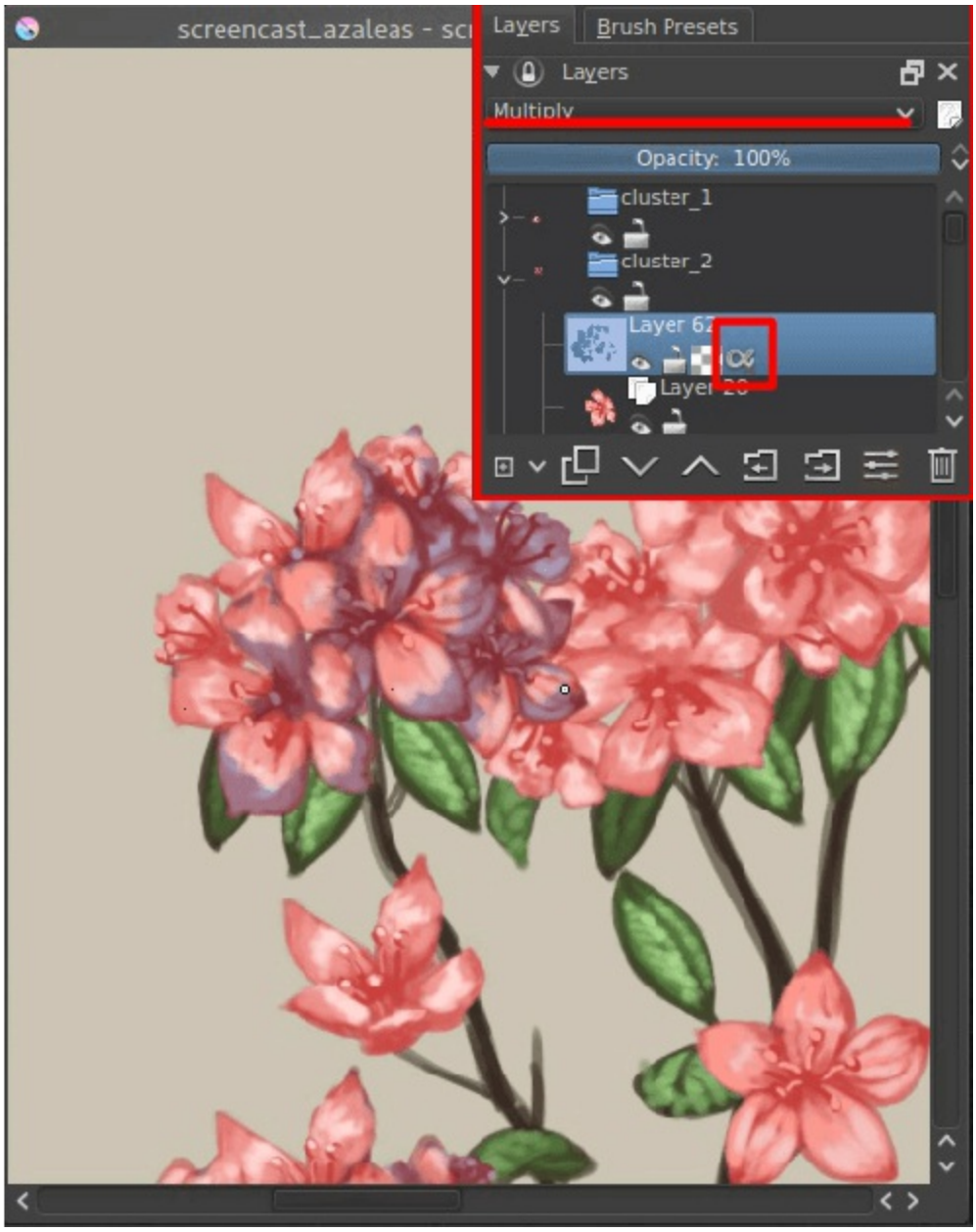
But for us, alpha inheritance is useful, because we can use all clone-layers in a cluster (if you grouped them), transformed or not, for clipping. Just draw a light blue square over all the flowers in a given cluster.



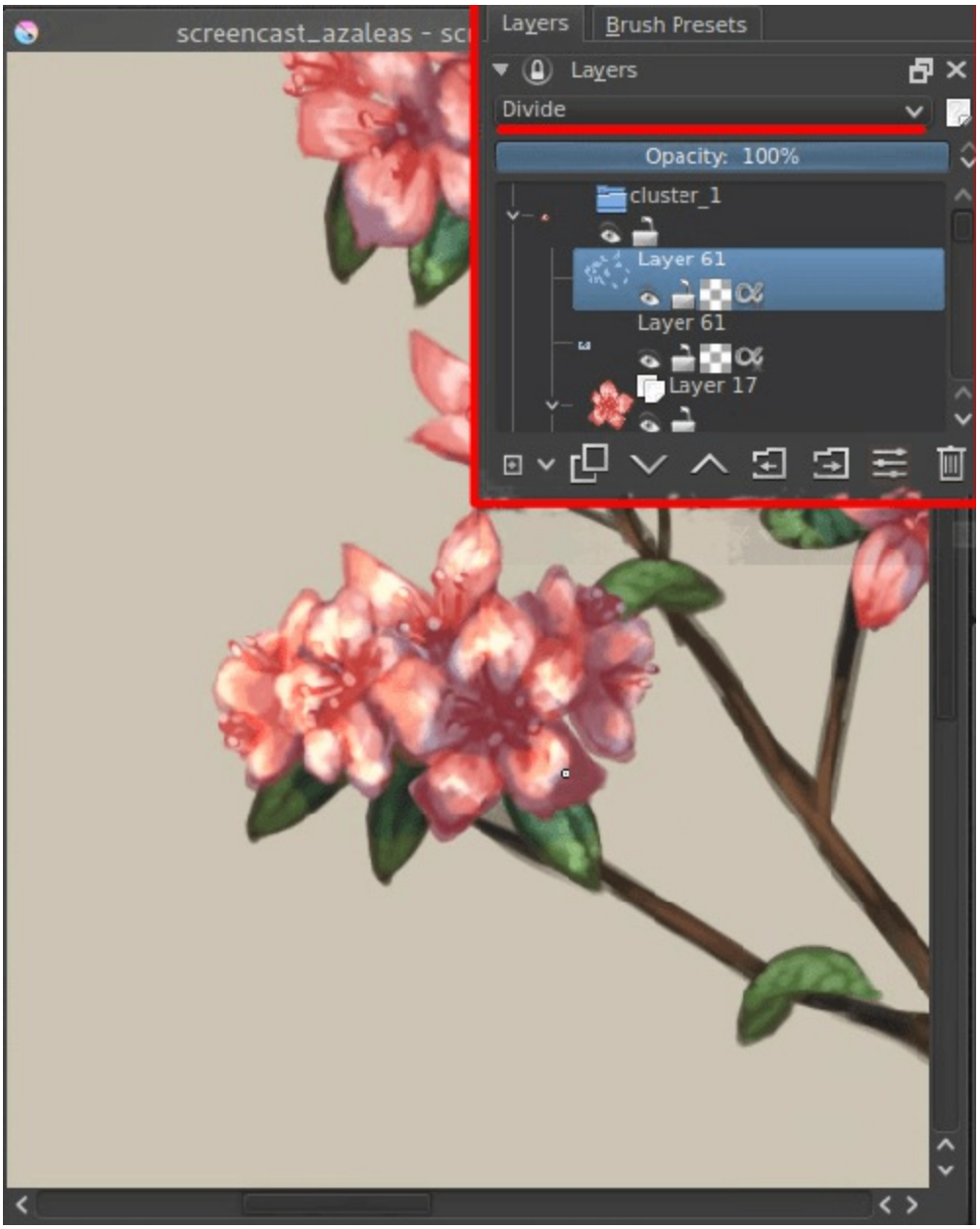
Then press the last icon in the layer stack, the alpha-inherit button, to activate alpha-inheritance.



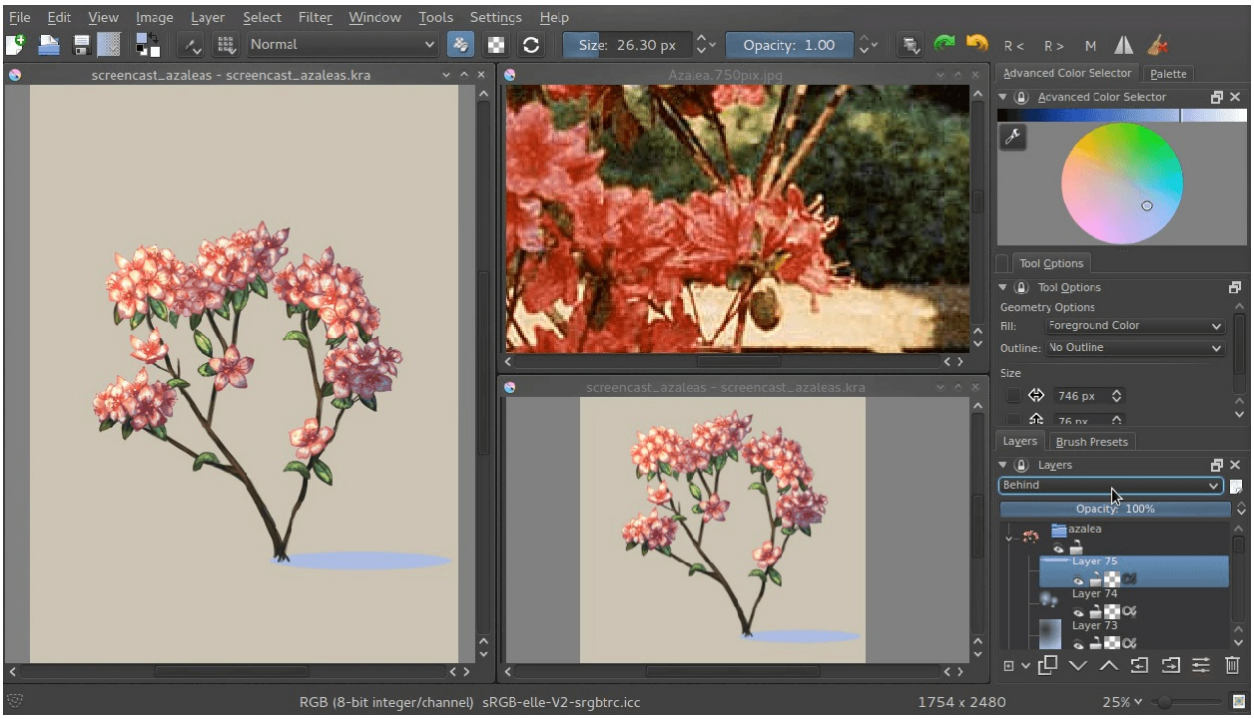
Set the layer to *multiply* then, so it’ll look like everything’s darker blue.



Then, with multiply and alpha inheritance on, use an eraser to remove the areas where there should be no shadow.



For the highlights use exactly the same method, AND exactly the same color, but instead set the layer to Divide (you can find this amongst the Arithmetic blending modes). Using Divide has exactly the opposite effect as using multiply with the same color. The benefit of this is that you can easily set up a complementary harmony in your shadows and highlights using these two.



Do this with all clusters and leaves, and maybe on the whole plant (you will first need to stick it into a group layer given the background is opaque) and you're done!

Transform masks can be used on paint layers, vector layers, group layers, clone layers and even file layers. I hope this tutorial has given you a nice idea on how to use them, and hope to see much more use of the transform masks in the future!

You can get the file I made here [https://share.kde.org/public.php?service=files&t=48c601aaf17271d7ca516c44cbe8590e] to examine it further! (Caution: It will freeze up Krita if your version is below 2.9.4. The speed-ups in 2.9.4 are due to this file.)

Saving For The Web

Krita's default saving format is the *.kra format. This format saves everything Krita can manipulate about an image: Layers, Filters, Assistants, Masks, Color spaces, etc. However, that's a lot of data, so `*.kra` files are pretty big. This doesn't make them very good for uploading to the internet. Imagine how many people's data-plans hit the limit if they only could look at `*.kra` files! So instead, we optimise our images for the web.

There are a few steps involved:

1. Save as a `.kra`. This is your working file and serves as a backup if you make any mistakes.
2. Flatten all layers. This turns all your layers into a single one. Just go to *Layer* ▸ *Flatten Image* or press the `Ctrl + Shift + E` shortcut. Flattening can take a while, so if you have a big image, don't be scared if Krita freezes for a few seconds. It'll become responsive soon enough.
3. Convert the color space to 8bit sRGB (if it isn't yet). This is important to lower the filesize, and PNG for example can't take higher than 16bit. *Image* ▸ *Convert Image Color Space…* and set the options to **RGB**, **8bit** and **sRGB-elle-v2-srgbtrc.icc** respectively. If you are coming from a linear space, uncheck **little CMS** optimisations
4. Resize! Go to *Image* ▸ *Scale Image To New Size…* or use the `Ctrl + Alt + I` shortcut. This calls up the resize menu. A good rule of thumb for resizing is that you try to get both sizes to be less than 1200 pixels. (This being the size of HD formats). You can easily get there by setting the **Resolution** under **Print Size** to **72** dots per inch. Then press **OK** to have everything resized.
5. Sharpen the image a little. This is especially necessary for social media. Social media websites often scale and convert your image in such a way that it gets a little blurry, because they optimize towards photos and not paintings. To have your images stay sharp, it is worth it to run a sharpen filter beforehand. Because the sharpen filter is quite powerful, you are best off adding a sharpen filter mask on top of the stack and lowering its opacity till you feel the sharpness is appropriate.

6. Save as a web-safe image format. There's three that are especially recommended:

JPG

Use this for images with a lot of different colors, like paintings.

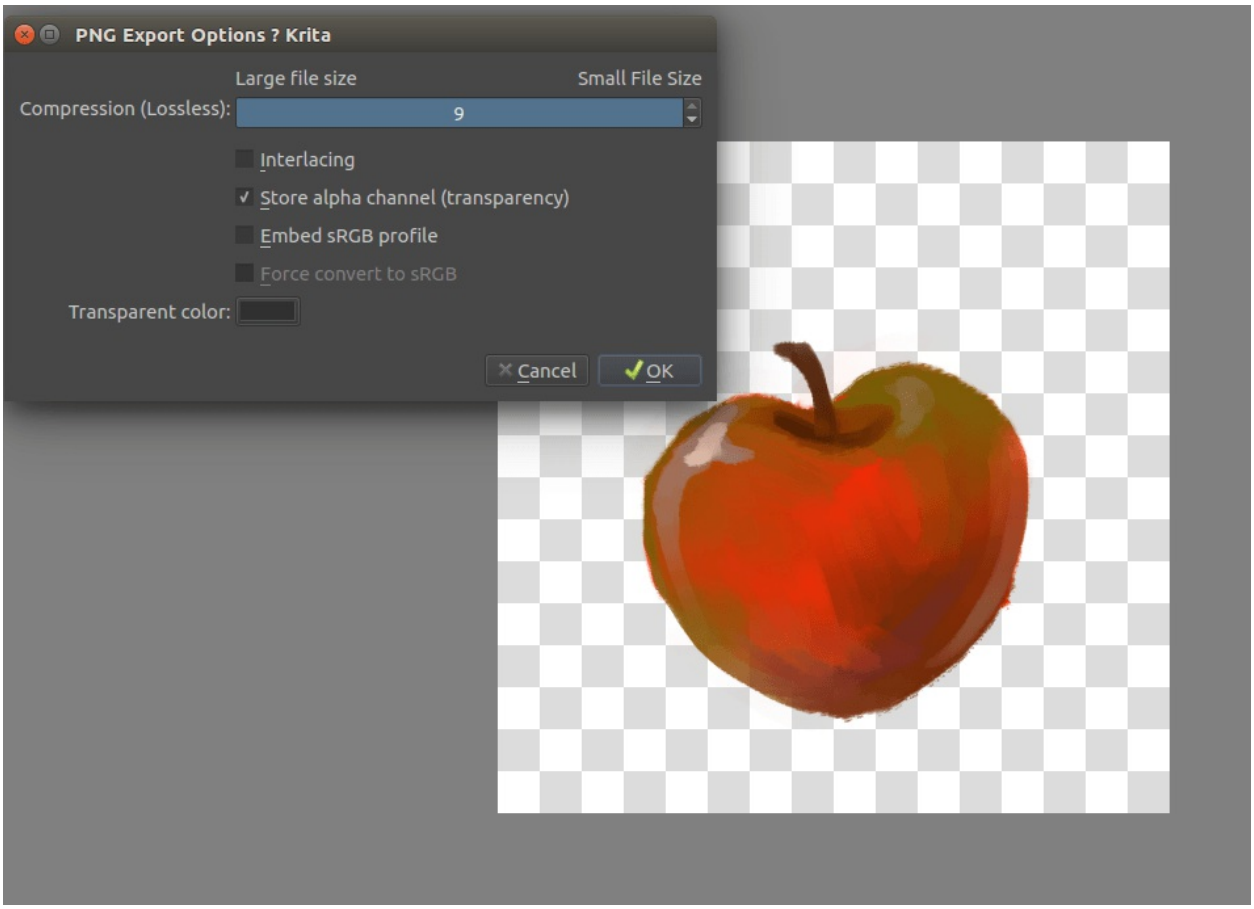
PNG

Use this for images with few colors or which are black and white, like comics and pixel-art. Select *Save as indexed PNG, if possible* to optimise even more.

GIF

Only use this for animation (will be supported this year) or images with a super low color count, because they will get indexed.

Saving with Transparency



Saving with transparency is only possible with GIF and PNG. First, make sure you see the transparency checkers (this can be done by simply hiding the bottom layers, changing the projection color in *Image* ▸ *Image Background Color and Transparency…*, or by using *Filters* ▸ *Colors* ▸ *Color to Alpha…*). Then, save as PNG and tick **Store alpha channel (transparency)**

Save your image, upload, and show it off!

Introduction to SeExpr

New in version 4.4: This document will introduce you to the SeExpr expression language.

What is SeExpr?

SeExpr is an embeddable expression language, designed by Disney Animation, that allows host applications to render dynamically generated content. Pixar calls it in its documentation [https://renderman.pixar.com/resources/RenderMan_20/PxrSeExpr.html] a “scriptable pattern generator and combiner”.

SeExpr is available within Krita as a Fill Layer.

See also

- SeExpr Quick Reference
- SeExpr
- SeExpr Scripts
- “Procedural texture generator (example and wishes)” on Krita Artists [https://krita-artists.org/t/procedural-texture-generator-example-and-wishes/7638]
- Inigo Quilez’s articles [https://iquilezles.org/www/index.htm]
- The Book of Shaders [https://thebookofshaders.com/]

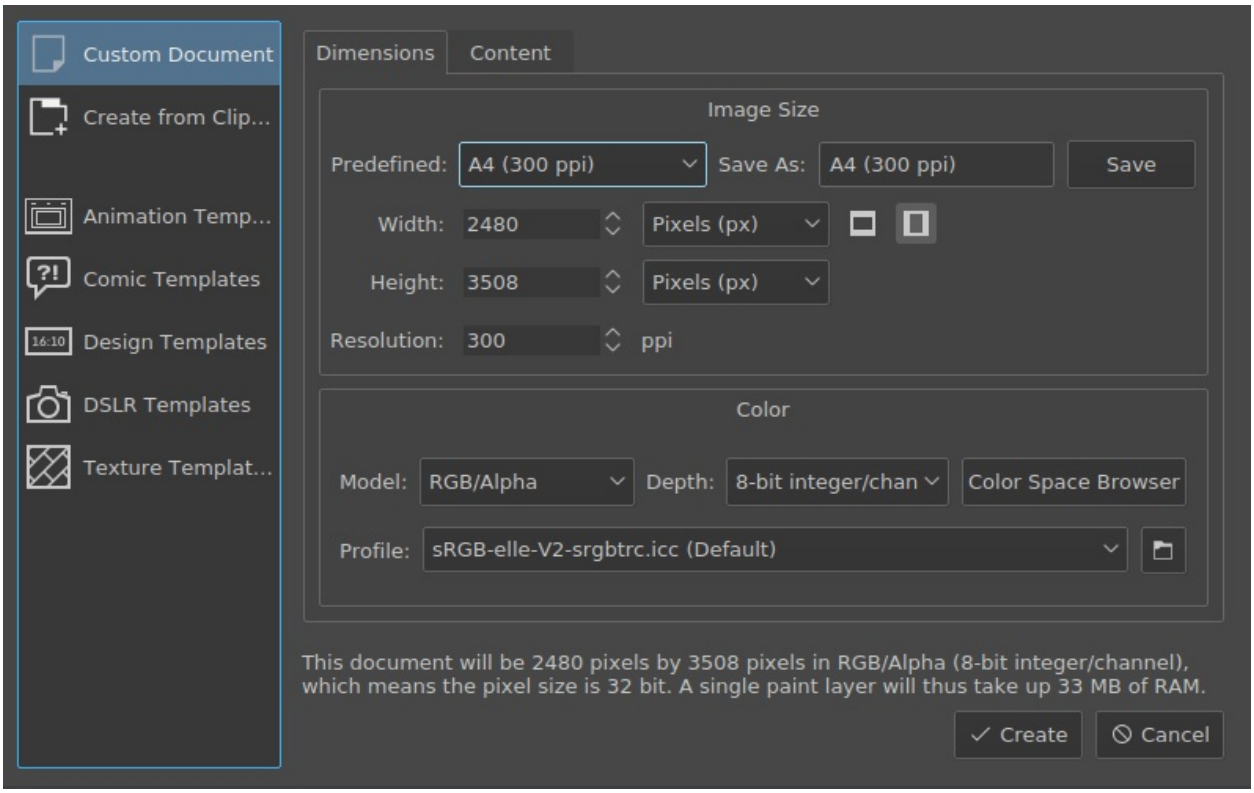
Background

To understand what SeExpr is about, we need to differentiate between two types of graphics, *raster* and *procedural*.

The vast majority of the computer-generated stuff you see every day belong to the first type– images like photos, your favorite anime screenshots, memes, are all a multitude of tiny little dots of color, or *pixels*, arranged into a grid.

Raster graphics have two drawbacks. First, once you create them, their resolution is **fixed**. You cannot zoom in and magically get any more detail. And if you need to change them, either you go back to the source and sample it again (which is sometimes impossible), or edit it with a raster graphics program, like Krita.

One of the biggest problems, however, is that we are always limited by the **space** our programs can use; either secondary storage, like SD cards, or RAM. Unless compressed, image memory needs are quadratic in the size of the image [https://blender.stackexchange.com/questions/112505/why-is-my-half-resolution-render-taking-a-quarter-of-the-time-of-the-full-one]. For a quick example, the Create New Document dialog of Krita tells you three bits of information: its size in pixels, the size of the pixel itself, and *the total memory needed*.



Here's a summary for square textures. Note that the memory needed is for *one layer only*:

Size	Memory needed

256	256 KB
512	1 MB
1024	4 MB
2048	16 MB
4096	64 MB

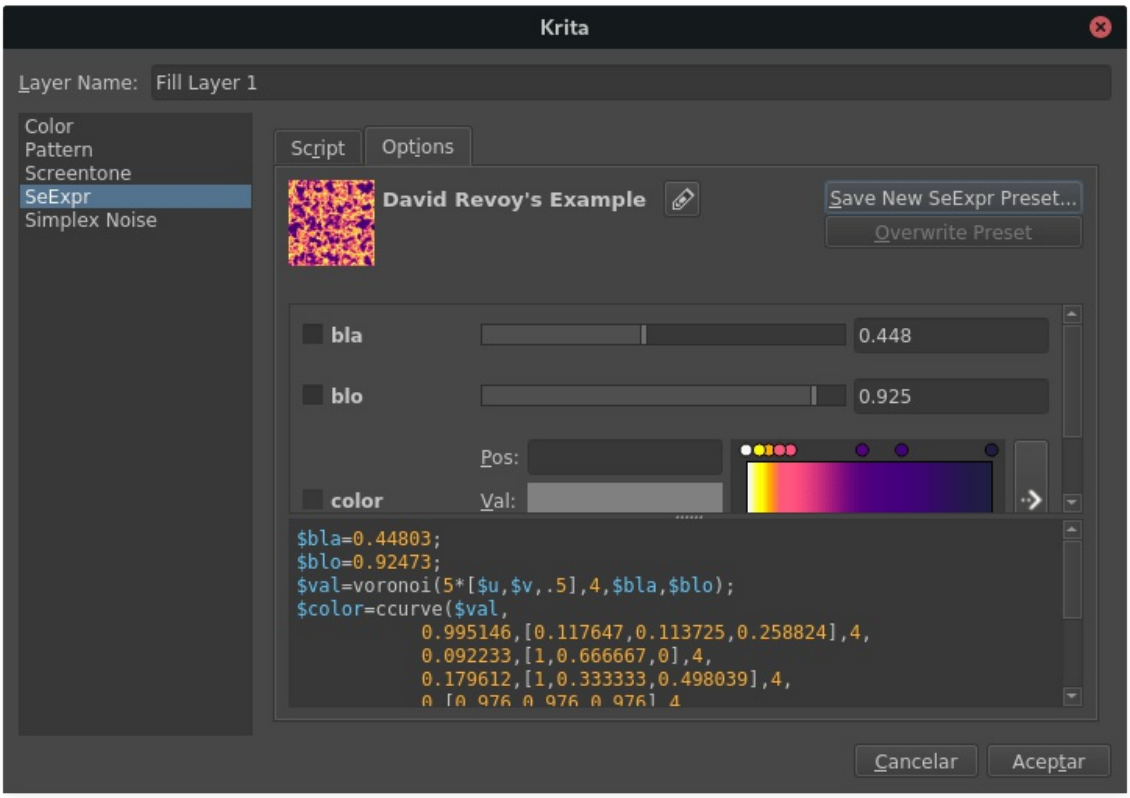
An alternative is to use Vector Graphics. Vector graphics, for instance SVGs, employ mathematic formulae like splines and Bézier curves to describe a shape. As they are mathematically defined, they can be resized to suit your needs without losing resolution.

SeExpr belongs to a different class, *procedural graphics*. Similar to vector graphics, procedural graphics only need a few KBs of secondary storage for their definition. But they are not defined by mathematical formulae; you actually *code* how the color is calculated at each point of the texture. And, because it is not limited in its precision, you can render complex patterns in your layers at completely arbitrary resolution.

Writing a script

In this tutorial, we'll show you how to write a script in SeExpr, render it to a layer, and then save it as a preset.

We'll start by going to the Layers, and adding a new Fill Layer. Then select the SeExpr generator from the list. You'll be greeted by this window:



The SeExpr generator dialog is divided into two tabs. For now, we'll stay on *Options*.

Note

Fill Layers describes these tabs in more detail.

Let's start by painting a layer in light blue.

First, SeExpr scripts must define an output variable, let's call it `$color`. As SeExpr thinks of colors in the RGB color space, color variables are defined by a triplet of numbers known as a *vector*. We'll start by defining the `$color` variable and giving it a value.

Go to the text box, and clear it if it has any text. Then, define and set `$color` to something like `[0.5, 0.5, 1]` (half lit red, half lit green, fully lit blue):

```
$color = [0.5, 0.5, 1];
```

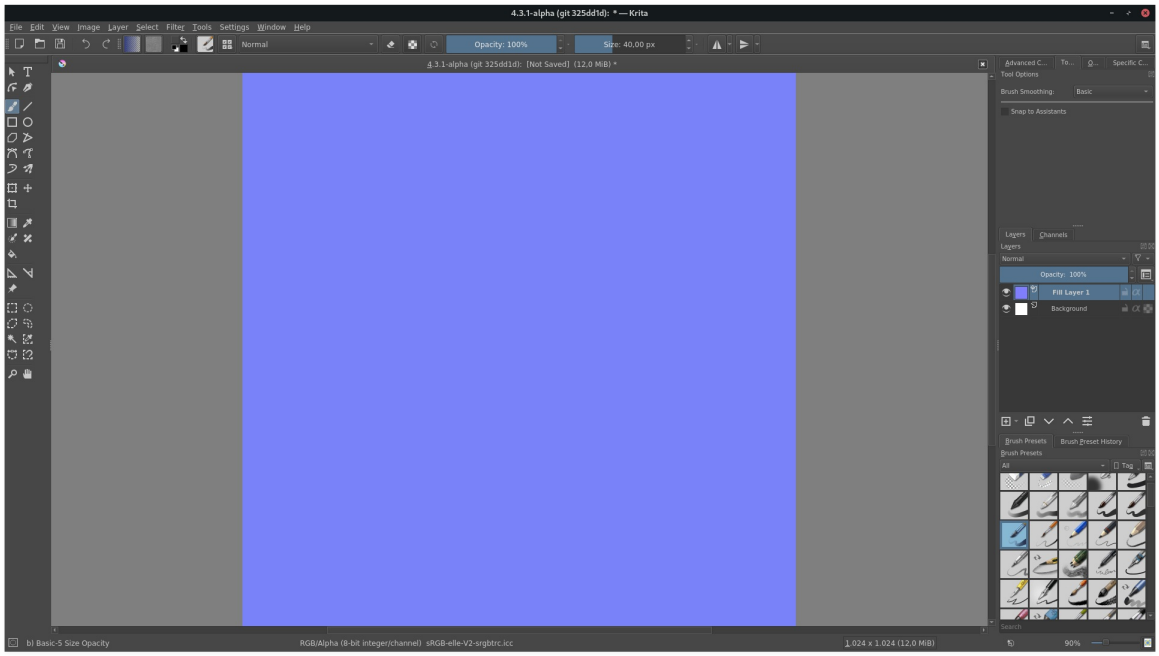
SeExpr needs to know which variable holds the final color value. This is done by writing at the end, on its own line, the name of the variable:

```
$color
```

The script should now look like this:

```
$color = [0.5, 0.5, 1];
$color
```

Click *OK*, and you'll render your first script!



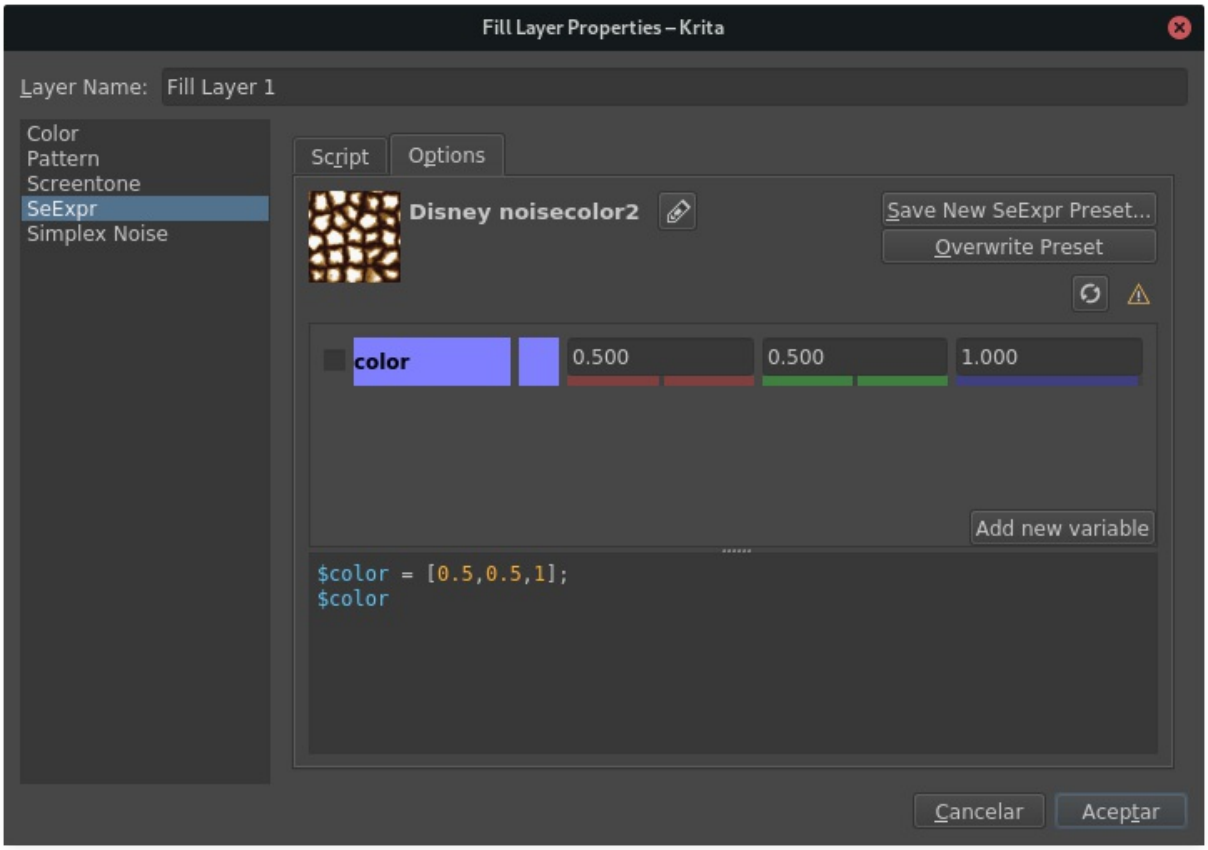
Warning

To be absolutely precise, SeExpr **has no color management**. It always renders textures as 32-bit float, gamma corrected, sRGB images. Krita transforms them into your document's color space using the sRGB-elle-V2-srgbtrc.icc profile.

See Color Managed Workflow for what this means.

Managing your script using widgets

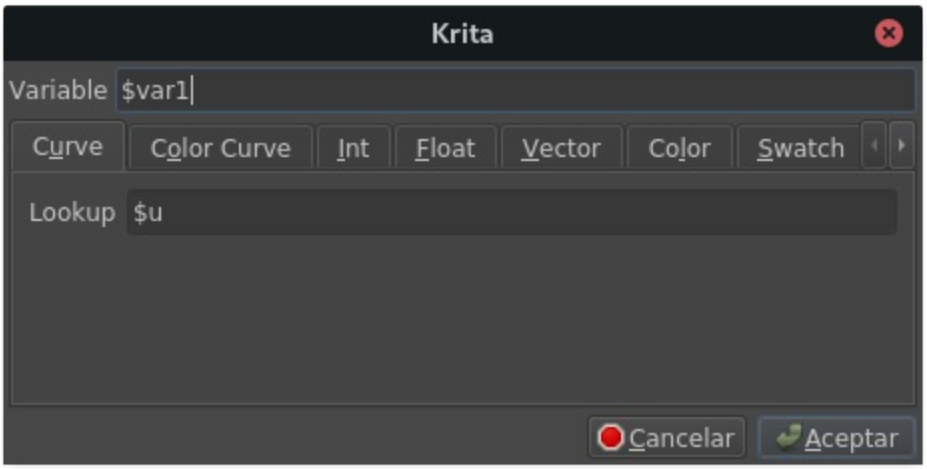
There is also another way to define and edit your variables. Open the fill layer's properties by right-clicking on *Fill Layer 1*, and selecting *Layer Properties…*.



Notice the middle box? Once it detects a syntactically correct script, SeExpr enables a whole chunk of knobs to manage individual variables. In our example above, you can change `$color`'s in three ways:

- enter the red, green, or blue channel's value in the input fields
- move the little colored sliders to change the respective channel
- click on the preview square to the left of the boxes, to select a completely new color.

The last button on the middle box is always *Add new variable*. Click it and this dialog will open:



This dialog shows you all the types of variables that SeExpr accepts:

Curve and Color curve

They are the SeExpr version of Stop Gradients: they interpolate a ramp given by a set of values.

Curves represent 1D gradients, returning a single float at each evaluation point.

Color curves represent RGB gradients, returning a Color at each point.

Integers and Floats

Numbers.

Vector

A triplet of floats.

Color

A vector representing an RGB color.

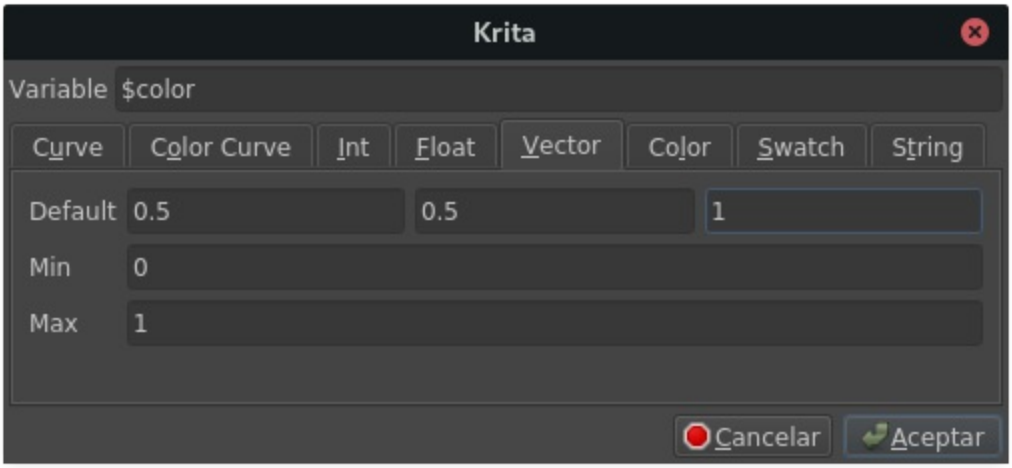
Swatch

A list of Colors.

String

Usually single words.

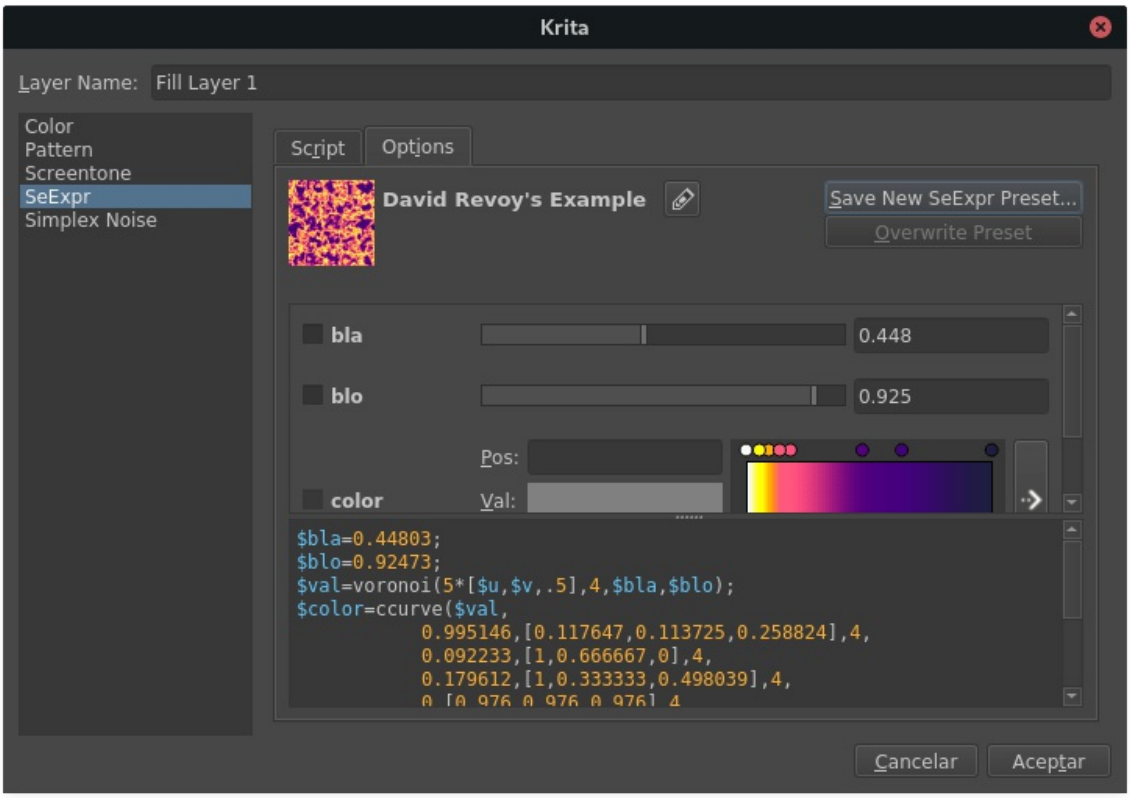
For instance, you could replicate `$color` in the *Vector* tab:



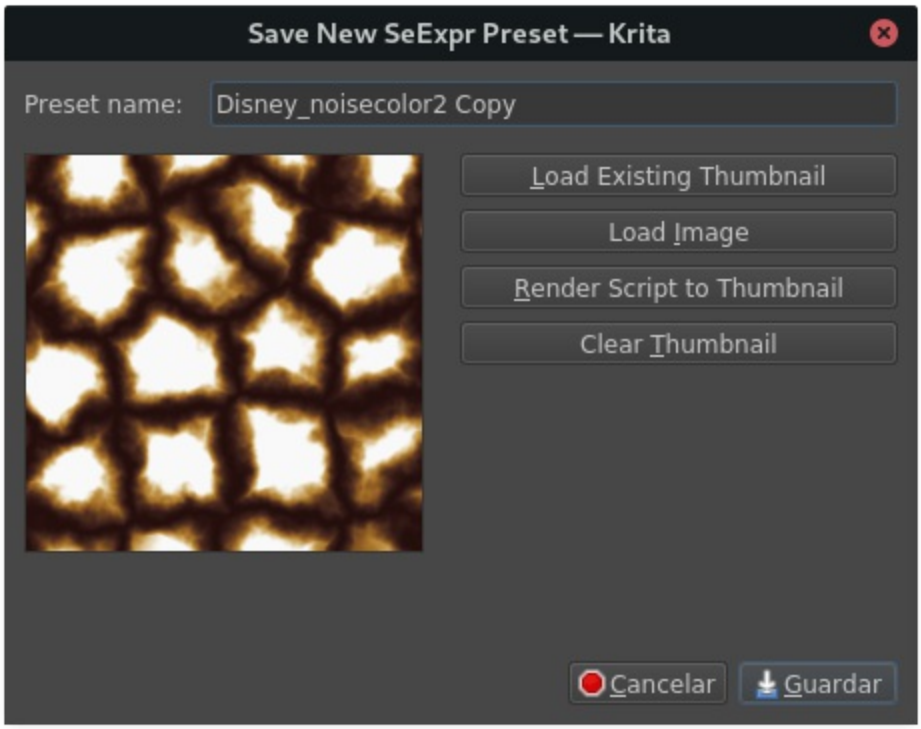
Creating your first preset

Once your script is ready, you can reuse it by making a preset.

You can create one through the top bar of the *Options* tab:



Select *Save New SeExpr Preset.…* and the following dialog will open:



You can edit the name of the preset in the top line edit box, and set a thumbnail for easy identification.

Hint

The dialog will append “Copy” to the preset’s name if it is a copy of an existing one. You can change it at will.

The dialog provides the following choices for setting a thumbnail:

Load Existing Thumbnail

If the preset already has a thumbnail (for instance, if you created it from an existing preset), this button will load and apply it.

Load Image

Applies an image from the filesystem as a thumbnail.

Render Script to Thumbnail

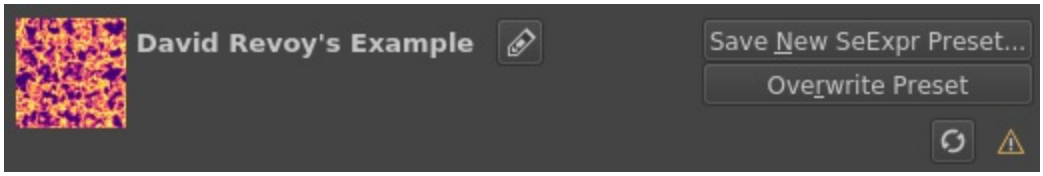
Renders your script to a 256x256 texture, and applies the latter as a thumbnail.

Clear Thumbnail

Deletes the thumbnail. Note that, if the preset is a copy of an existing one, this can be reverted by clicking *Load Existing Thumbnail*.

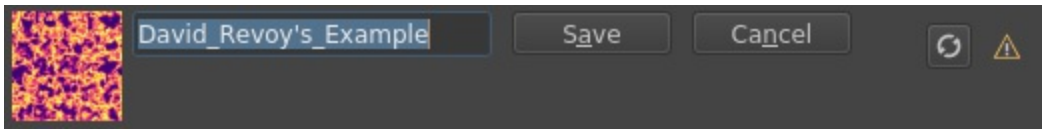
Changing existing presets

If you change a preset's script, you will notice two new buttons in the top bar of the *Options* tab:



The reload button will restore the preset to its original properties, while clicking on *Overwrite Preset* will save your changes.

Additionally, you can edit the preset's name by clicking on the rename button, entering the new name, and clicking on *Save*:



Bundling your presets

Sharing your scripts is easy! SeExpr script presets are just like any other resource in Krita. Follow the instructions in Resource Management to create your own bundles.

Krita FAQ

This page contains common problems people have with Krita. Note that we assume that you are using the latest version of Krita. Please verify that to make sure.

Contents

General

General questions

What is Krita?

This is our vision for the development of Krita:

> Krita is a free and open source cross-platform application that offers an end-to-end solution for creating digital art files from scratch. Krita is optimized for frequent, prolonged and focused use. Explicitly supported fields of painting are illustrations, concept art, matte painting, textures, comics and animations. Developed together with users, Krita is an application that supports their actual needs and workflow. Krita supports open standards and interoperates with other applications.

Is it possible to use Krita in my own language, not English?

Krita should automatically use the system language. If that is not the case, please follow these steps:

1. With *Settings* ▸ *Switch Application Language…* menu item will appear a small window.
2. Click *Primary language* and select your language.
3. Click *OK* to close the window.
4. Restart krita and it will be displayed in your selected language!

If this doesn't work, you might have to add a fall-back language as well. This is a bug, but we haven't found the solution yet.

I have a problem, how to get support for Krita?

1. Ask your question on one of the user support forums. It can be Krita Artists forum [https://krita-artists.org/] (recommended), Krita KDE forum [https://forum.kde.org/viewforum.php?f=136] or alternatively Krita subreddit [https://www.reddit.com/r/krita/] .
2. Remember – most probably the person you're interacting with is a volunteer, and even if not, it's not someone actually paid for doing user support. Treat them nicely and respect their time! They will for sure reciprocate that.
3. Create a new thread or post for your issue and state the issue in the title. Even if you don't know how to describe it exactly, for example you title the post "Krita's text tool is acting weird", it is much better than simple "Help!".
4. Describe your issue. You can add screenshots and videos, but even if you do that, write a few words what the video shows and what the issue is.
5. State all relevant details: most importantly your operating system (Windows, MacOS, Linux…), which version of Krita you use (go to *Help* ▸ *About Krita* to find out) and what kind of tablet you have (if your issue is related to a tablet driver).
6. Answer all questions your supporter asks, even if you don't think they're relevant. They probably have a reason to ask about that.
7. If you don't know the answer or you don't know how to get the information your supporter asks for, don't hesitate to ask for clarification.
8. If you mentioned an issue, please help with testing if you're asked to do that – that will speed up the process of finding the cause and preparing a fix.

Does Krita have layer clip or clipping mask?

Krita has no clipping mask, but it has a clipping feature called inherit alpha. Let's see this page and learn how to do clipping in Krita!

Where are the configuration files stored?

These are stored at the following places for the following operating systems:

Linux
`$HOME/.config/kritarc`

Windows
`%APPDATA%\Local\kritarc`

MacOS X
`$HOME/Library/Preferences/kritarc`

The kritarc file is the configuration file. Krita does not store settings in the Windows registry.

Resetting Krita configuration

You can reset the Krita configuration in one of the following ways:

- Press and hold `Shift + Alt + Ctrl` while starting Krita. This should show a pop-up asking if you want to reset the configuration. Press *yes* to reset it.

- For Krita 3.0 and later: Delete/rename the kritarc file, found here:

    Linux
    `$HOME/.config/kritarc`

    Windows
    `%LOCALAPPDATA%\kritarc`

    MacOS X
    `$HOME/Library/Preferences/kritarc`

There can be two other files you might want to remove: kritaopenglrc and kritadisplayrc.

If the configuration was causing a crash, don’t delete the mentioned file, but instead rename and send it to us in order for us to figure what caused the

crash.

If you have installed Krita through the Windows store, the kritarc file will be in another location

```
%LOCALAPPDATA%\Packages\49800Krita_RANDOM
STRING\LocalCache\Local\kritarc
```

The random string depends on your installation.

Windows users have a habit of uninstalling and reinstalling applications to solve problems. Unless the problem is that the installation was corrupted by a virus scanner or drive failure, that will NOT work. Uninstalling Krita then reinstalling replaces the bytes on your drive with exactly the same bytes that were there before. It doesn't reset anything, least of all Krita's settings.

Why my configuration resets on its own?

There are two possible reasons:

- You don't save your settings.

This is most probable if you are on Windows and you have either a display with a small resolution (below fullHD) or if you have fullHD resolution with UI scaling in Windows settings (which is 150% by default). In those cases it might happen that you don't see the *OK* button in the *Configure Krita* dialog. You can use `Alt + O` instead. (You can go to *Configure Krita…* ▸ *General* ▸ *Window* and make sure that *Enable HiDPI* checkbox is unchecked to disable scaling for Krita and get a smaller UI).

- You close your computer using the power button.

If you are on Windows and you use power button instead of a standard procedure to close or restart your computer, it might happen that Krita's configuration file gets corrupted. To solve this, just use the correct way of closing your system: either *Start* ▸ *Restart* or *Start* ▸ *Shutdown*.

Where are my resources stored?

Linux

```
$HOME/.local/share/krita/
```

Windows

```
%APPDATA%\krita\
```

Mac OS X

```
~/Library/Application Support/Krita/
```

If you installed Krita in the Windows Store, your custom resources will be in a location like:

```
%LOCALAPPDATA%\Packages\49800Krita_RANDOM
STRING\LocalCacheRoamingkrita
```

Krita tells me it can't find some files and then closes, what should I do?

Causes for this could be the following:

- It might be that your download got corrupted and is missing files (common with bad wifi and bad internet connection in general), in that case, try to find a better internet connection before trying to download again. Krita should be around 80 to 100 MB in size when downloading.
- It might be that something went wrong during installation. Check whether your harddrive is full and reinstall Krita with at least 120 MB of empty space. If not, and the problem still occurs, there might be something odd going on with your device and it's recommended to find a computer expert to diagnose what is the problem.
- Some unzippers don't unpack our ZIP files correctly. The native ones on Windows, OSX and most Linux distributions should be just fine, and we recommend using them.
- You manually, using a file manager deleted or moved resources around, and thus Krita cannot find them anymore.

What Graphics Cards does Krita support?

Krita can use OpenGL to accelerate painting and canvas zooming, rotation and panning. Nvidia and recent Intel GPUs give the best results. Make sure

your OpenGL drivers support OpenGL 3.2 as the minimum. AMD/ATI GPU's are known to be troublesome, especially with the proprietary drivers on Linux. However, it works perfectly with the Radeon free driver on Linux for supported AMD GPU. Try to get a graphics card that can support OpenGL 3.2 or above for the best results, some examples:

Intel
: Intel 3rd Generation HD Graphics, IvyBridge or Bay-Trail microarchitecture, released in 2012. Commonly available products: Celeron J1x00, N2x00, Celeron (G)1xx0, Pentium J2x00, N3500, Pentium (G)2xx0, Core i3/5/7-3xx0.

AMD/ATI
: Radeon HD 2000 family, TeraScale 1 microarchitecture, Released in 2007. Commonly available products: Radeon HD 2400 PRO, Radeon HD 2600 PRO, etc.

Nvidia
: GeForce 8 family, Tesla microarchitecture, released in 2006. Commonly available products: GeForce 8400 GS, GeForce 8800 GTS, 9800 GTX, GTS 250, etc.

For Krita 3.3 or later: Krita on Windows can use Direct3D 11 for graphics acceleration (through ANGLE). This is enabled automatically on systems with an Intel GPU.

I can't edit text from PSD files created by Photoshop

There is no text support for PSD file yet. The text will appear rasterized and converted into a paint layer.

How much memory does my image take?

For simple images, its easy to calculate: you multiply width * height * channels * size of the channels (so, for a 1000×1000 16 bit integer rgba image: 1000 x 1000 x 4 x 2). You multiply this by the number of layers plus two (one for the image, one for the display). If you add masks, filter layers or

clone layers, it gets more complicated.

Why do I get a checkerboard pattern when I use the eraser?

You're probably used to Gimp or Photoshop. The default background or first layer in these applications doesn't have an alpha channel by default. Thus, on their background layer, the eraser paints in the background color.

In Krita, all layers have an alpha channel, if you want to paint in the background color, you should simply do it in a layer above the first one (Layer 1), that would prevent you from erasing the white background color, making the checkerboard visible. You get the same effect in, say, Gimp, if you create new image, add an alpha channel and then use the eraser tool. Most Krita users will actually start a sketch in Krita by adding a new blank layer first before doing anything else. (The `Ins` key is a useful shortcut here). That doesn't use extra memory, since a blank layer or a layer with a default color just takes one pixel worth of memory.

Can krita work with 8 bit (indexed) images?

No. Krita has been designed from the ground up to use real colors, not indexed palettes. There are no plans to support indexed color images, although Krita can export to some indexed color image formats, such as GIF. However, it does not offer detailed control over pixel values.

Where can I find older versions of Krita?

All the older versions of Krita that are still available can be found here:

- Very old builds [https://download.kde.org/Attic/krita/].

On Windows, the Krita User Interface is too big on my screen

If you're using Windows, you can set the display scaling to 150% or 200%. Krita comes with HiDPI enabled by default, so if you do that, the Krita UI might be too big for your screen. You can turn it off using the following steps:

- On the menu, select *Settings* ▸ *Configure Krita…*
- On *General* page, switch to *Window* tab.
- Uncheck *Enable Hi-DPI support* (or check if you wish to enable it)
- Press *OK*, if the settings screen is too big, `Alt + O` will trigger the OK button too.
- Restart Krita

You can also change the toolbox icon size by right-clicking on the toolbox and selecting a size.

Windows: In fullscreen mode, why is there a thin gap at the bottom of the window?

When Canvas Graphics Acceleration is set to OpenGL, you may see a thin gap at the bottom of the window which you can see through. This is done deliberately to work around a bug causing menus and dropdowns to be unusable. If you find it distracting, you can consider changing the Renderer to Direct3D 11 which doesn't require this workaround.

Windows: OBS can't record the Krita OpenGL canvas

The possible workarounds for this is to do either of the following:

1. Turn off OpenGL in *Settings* ▸ *Configure Krita…* ▸ *Display*.
2. Or don't use the hardware accelerated mode (game recording mode) in OBS, thus capturing the whole desktop instead of attempting to capture only Krita.

You might also be able to work around the problem by using the ANGLE renderer instead of native OpenGL.

Windows: Can I use Krita with Sandboxie?

No, this is not recommended. Sandboxie causes stuttering and freezes due to the way it intercepts calls for resources on disk.

Windows: Krita cannot save

If the message is “File not found. Check the file name and try again.”, you probably have Controlled Folder Access enabled.

- Select *Start* ▸ *Settings*.
- Choose *Update & security* ▸ *Windows Defender*.
- Select *Open Windows Defender Security Center*.
- Select *Virus & threat protection*, and then choose *Virus & threat protection settings*.
- Under *Controlled folder access*, turn it on or off.

You can also whitelist Krita, following these instructions [https://docs.microsoft.com/en-us/windows/security/threat-protection/microsoft-defender-atp/customize-controlled-folders#allow-specific-apps-to-make-changes-to-controlled-folders].

Windows: Krita cannot open my file anymore

Your file got corrupted. There are several things that might cause this:

1. Windows was shutdown improperly, like by holding the power button. This prevents your harddrive from finishing up the things it is doing and file away your files incorrectly. Please always try to shutdown your computer via the proper shutdown procedure, and if you are in a situation where this is not possible (like frequent blackouts), make daily backups! This may lead to the file being filled with zeroes, so it cannot be recovered from.

   > *Changed in version 4.2.8:* Krita version 4.2.8 introduced special safety measure for Windows that should help avoiding this situation. But in any case, unless something makes it impossible, always make sure to shutdown your system using the standard approach. On Windows that means going to Start menu and selecting “Shutdown”.

2. Badly programmed security software may attempt to rewrite KRA files, or prevent Krita from writing to the folder you wish to save to. These cases can be checked by trying to save in that location, and then, without shutting down Krita, checking in the folder to see if the file saved. Files lost due this cannot be recovered.

3. Cloud services like dropbox and onedrive have been known to prevent Krita from saving. We've implemented fixes for this, but much like the above point it is worth checking that this isn't the cause of the issue. Files lost due this cannot be recovered.

4. Occasionally the ZIPs that KRA files comprise of will have the last few bytes missing. We're doing everything in our power to prevent this kind of corruption, but it might be a file system issue. This particular bug can be fixed by renaming the extension (in windows you will need to enable the file extensions, which this FAQ will not cover) to ZIP, and then using a ZIP repairing utility to fix the ZIP file. Then rename it back to KRA.

5. If Krita doesn't give an error message, but rather crashes, your file is too big, and Krita is not so much crashing as that the operating system is shutting it down. Try shutting down some other programs like webbrowsers or streaming services to free up working memory. You should be able to open the file in question. At this point the recommended course of action is to try and reduce the file size in some manner, such as merging layers, splitting up an animation or scaling the image down.

How to recover my files?

1. Check whether you have any backup file or autosave left: Saving, AutoSave and Backup Files.

2. Check whether you can open the file as ZIP archive.
    1. Rename the extension of the file from `.kra` to `.zip`.
    2. Try to open (your system should automatically select an archive opener tool).
    3. There is file called mergedimage.png inside that represents all layers merged that you can use for reference in case you can't restore anything else.

3. Check whether ZIP repairer tool helps.
    1. Copy the file so you have a backup just in case.

2. Rename the extension of the file from `.kra` to `.zip`.

3. Use ZIP repairer tool on the `.zip` file.

```
# On Linux:
mv file.kra file_copy.zip
zip -F file_copy.zip --out file_new1.zip
unzip file_new1.zip
# if it still doesn't work:
zip -FF file_copy.zip --out file_new2.zip
unzip file_new2.zip
# if it still doesn't work, try to run it again or

# On Windows:
Copy the file, rename the extension.
Use any graphical ZIP repairer on the new file. (F
```

4. Try to open in Krita.

5. If it cannot be opened in Krita, try the trick from 2.: open the archive and find mergedimage.png file.

4. Open your file in Notepad or any other text editor. If the the content of the file is only a repeated *NUL* symbol, it means the file is most probably unrecoverable using the standard method. If it's of a very high importance for you, you can try to recover the previous save using methods that checks the hard drive directly.

Krita crashes on Windows 7 on start-up

Starting with Krita 4.2.0, Krita uses version 5.12 of the Qt toolkit. This needs to have access to Direct3D 11 or OpenGL ES 2.0 or higher. You might need to install drivers appropriate to your GPU (Nvidia, AMD/ATI, Intel). This also makes it hard to run Krita in a virtual environment: in Virtual Box you need to install the guest addition in safe mode, and enable the experimental Direct3D support.

Krita freezes randomly on my Windows system

Are you using a dictionary app (e.g. Youdao Dictionary for Chinese users)? Some dictionary apps can read words from other app's windows and show popup translations in real time. However, it has been reported that such apps tend to cause Krita to freeze randomly. If you are using one of those, make sure to QUIT them (no notification icon) when using Krita. Some of those apps keep running in the background even after being closed. In such case, you will have to uninstall them.

Windows: How can I produce a backtrace?

See also

Dr. Mingw debugger

If you experience a crash on Windows, and can reproduce the crash, the bug report will be much more valuable if you can create a backtrace. A backtrace is somewhat akin to an airplane's blackbox, in that they tell what set of instructions your computer was running when it was crashing (where the crash happened), making it very useful to figure out why the crash happened.

The Dr. Mingw debugger is bundled with Krita. Please visit the page Dr. Mingw debugger for instructions on getting a backtrace with it.

Windows: Krita's window is semi-transparent

Chances are you are using an NVidia GPU. Due to a bug in Nvidia's driver that we haven't been able to workaround yet, sometimes Krita's window will be transparent or semi-transparent. The solution is to enable the Angle renderer in Krita's Settings dialog. Open the *Settings* menu (Press Alt-N if the menubar is not visible and your system is in English), then open the *Configure Krita* dialog. In the dialog window select the *Display* page and select the Angle renderer in the *Preferred Renderer* combobox. Restart Krita.

Why are there ampersand (&) characters in some docker titles?

This is a bug in one of the third party libraries Krita uses (and consequently has no direct influence over), where letters that should actually be underlined (they point out keyboard shortcuts that can be used when the *Settings* ▸ *Docker* menu is open) are instead prepended with an ampersand (&).

This bug only occurs with specific system configurations (it's related to the "Fusion" style) and/or in Krita packages obtained from third parties (e.g. in some Linux distributions).

If you are on Linux the best way to resolve this is to use an official package from krita.org [https://krita.org], such as the AppImage, Snap/Flatpak or PPA releases that are officially provided on the download page [https://krita.org/download/krita-desktop/].

Tablets

What tablets does Krita support?

Krita isn't much fun without a pressure sensitive tablet. If the tablet has been properly configured, Krita should work out of the box.

On Windows, you need to either install the Wintab drivers for your tablet, or enable the *Windows 8+ Pointer Input* option in Krita's settings.

You can find a community curated list of tablets supported by krita here.

If you're looking for information about tablets like the iPad or Android tablets, look here.

What if your tablet is not recognized by Krita?

First, check if you have installed drivers and the like. The Drawing Tablets page has some explanations and descriptions of common issues. If none of those work, we would like to have a bug report at bugs.kde.org, with a tablet log. Here's how you make a tablet log:

1. You need to have something to output the log to. On 4.2 you can use the

Log Viewer docker for this. Just open the log viewer, and enable logging.

Changed in version 4.2: The log viewer got added to Krita in 4.2, so for older versions of Krita, you will need to either run Krita in the terminal if you have Linux or MacOS, or for Windows install DebugView [https://docs.microsoft.com/en-us/sysinternals/downloads/debugview] from the official Microsoft site, start DebugView and then start Krita.

When using a terminal, make sure to enable 'unlimited scrollback'.

2. Press the `Ctrl + Shift + T` shortcut, you will see a message box telling the logging has started.

3. Try to reproduce your problem, you will be able to see the log being created in the log viewer as you draw.

4. Save the output from the log viewer into a txt file, and attach it to the bugreport.

On Linux, it is also useful to have the following information:

1. `lsmod`
2. `xinput`
3. `xinput list-props` (id can be fetched from the item 2)

However, in 100% of the cases where Windows users have reported that their tablet didn't work over the past five years, the problem has been either a buggy driver or a broken driver installation, but not a bug in Krita.

How to fix a tablet offset on multiple screen setup on Windows

If you see that your tablet pointer has an offset when working with Krita canvas, it might be highly likely that Krita got an incorrect screen resolution from the system. That problem happens mostly when an external monitor is present and when either a monitor or a tablet was connected after the system booted.

You can configure this by going to the Tablet Settings.

Microsoft Surface Pro and N-Trig

Krita 3.3.0 and later supports the Windows Pointer API (Windows Ink) natively. Your Surface Pro or other N-Trig enabled pen tablet should work out of the box with Krita after you enable Windows Ink in *Settings* ▸ *Configure Krita…* ▸ *Tablet*.

Tablet Pro and the Surface Pro

Unlike Wacom's Companion, the Surface line of tablets doesn't have working hardware buttons. Tablet Pro is a (non-free) utility that puts virtual buttons on screen. Krita 3.1 and above will have predefined shortcut profiles to work with Tablet Pro.

https://tabletpro.com/

See https://www.youtube.com/watch?v=WKXZgYqC3tI for instructions.

Weird stuff happens on Windows, like ripples, rings, squiggles or poltergeists

Windows comes with a lot of settings to make it work with a pen. All these settings can be annoying. This tool can help you set the settings correctly when you're using a tablet:

https://github.com/saveenr/Fix_My_Pen/releases

Touch doesn't seem to work on Windows

You might have to disable and enable the touch driver: go to the device manager. (Click the *Start* button and type device manager). Choose HID (User interface devices or something like that). Choose Intel(R) Precise Touch Device. Right click, Disable it. Right click, Enable it.

Toolbox

Toolbox missing

You can reset the Workspace by pressing the right most button on the toolbar, the Workspace switcher, and click on a desired Workspace from the list.

Or you can right-click on any docker title bar or open space in any toolbar, and select Toolbox. It's the first option.

Also, you can check the *Settings* menu, it has got a lot of interesting stuff, then go to the Dockers menu and select *Toolbox*.

Tool icons size is too big

Right click the toolbox to set the size.

Krita can't get maximized

This happens when your dockers are placed in such a way that the window cannot be made less high. Rearrange your Workspace.

Resources

Is there a way to restore a default brush that I have mistakenly overwritten with new settings to default?

Yes. First go to the resource folder, which is in

Linux

```
$HOME/.local/share/krita/
```

Windows

`user\Appdata\Roaming\krita\` or `%APPDATA%\Roaming\krita\`

OSX
   `~/Library/Application Support/Krita/`

You can easily do this by going into *Settings* ▸ *Manage Resources…* ▸ *Open Resource Folder*.

Then go into the *paintoppresets* folder and remove the latest created file that you made of your preset.

After that go back to the resources folder and edit the blacklist file to remove the previous paintoppreset so Krita will load it. (Yes, it is a bit of a convoluted system, but at the least you don't lose your brushes)

How do I set favorite presets?

Right-click a brush in the brush docker and assign it a tag. Then right-click on canvas to call popup palette, click the second right-most icon on the bottom-right of the palette, now you can pick the tag which contains the brush you assigned to it.

Can Krita load Photoshop Brushes?

Yes, but there are limitations. You can load ABR files by using the *Import* button in the *Predefined brush* tab in the brush editor. Since Adobe hasn't disclosed the file format specification, we depend on reverse-engineering to figure out what to load, and currently that's limited to basic features.

Krita is slow

There is a myriad of reasons why this might be. Below is a short checklist.

- Something else is hogging the CPU or the memory: spotify and other electron apps have been known to do this.
- You are running Windows, and have 3rdparty security software like Sandboxie or Total Defender installed
- You are working on images that are too big for your hardware (dimensions, channel depth or number of layers)

- You do not have canvas acceleration enabled
- You have (NVidia) Vertical Sync enabled
- On macOS, with some macs, you might need to disable canvas acceleration in Krita's settings.

Please also check this page [https://phabricator.kde.org/T7199].

Slow start-up

You probably have too many resources installed. Deactivate some bundles under the *Settings* ▸ *Manage Resources…* menu item.

If you're using Windows with the portable ZIP file, Windows will scan all files every time you start Krita. That takes ages. Either use the installer or tell Microsoft Security Essentials to make an exception for Krita.

Slow Brushes

- Check if you accidentally turned on the stabilizer in the tool options docker.
- Try another scaling mode like trilinear. *Settings* ▸ *Configure Krita…* ▸ *Display*.
- Try a lower channel depth than 16-bit.
- For NVidia, try a 16-bit floating point color space.
- For older AMD CPU's (Krita 2.9.10 and above), turn off the vector optimizations that are broken on AMD CPUs. *Settings* ▸ *Configure Krita…* ▸ *Performance*. This isn't needed if you've got an AMD Threadripper™ CPU.
- It's a fairly memory hungry program, so 2GB of RAM is the minimum, and 4GB is the preferable minimum.
- Check that nothing else is hogging your CPU.
- Check that Instant Preview is enabled if you're using bigger brushes (but for very small brushes, make sure is disabled).
- Set brush precision to 3 or auto.
- Use a larger value for brush spacing.
- If all of this fails, record a video and post a link and description on the Krita forum.

- Check whether OpenGL is enabled, and if it isn't, enable it. If it is enabled, and you are on Windows, try the Angle renderer. Or disable it.

Slowdown after a been working for a while

Once you have the slowdown, click on the image-dimensions in the status bar. It will tell you how much RAM Krita is using, if it has hit the limit, or whether it has started swapping. Swapping can slow down a program a lot, so either work on smaller images or turn up the maximum amount of RAM in *Settings* ▸ *Configure Krita…* ▸ *Performance* ▸ *Advanced Tab*.

Animation

Why is my animation black in my video player

You did not render the animation using the "baseline" option and you are using the default Windows media player. Re-render using the baseline option or use a better video player application, like VLC. Check this useful diagram [https://www.deviantart.com/tiarevlyn/art/T-Krita-4-1-7-rendering-issues-manual-783473428].

Tools

Why does the Transform Tool give a good result and then get blurry upon finalizing?

The transform tool makes a preview that you edit before computing the finalized version. As this preview is using the screen resolution rather than the image resolution, it may feel that the result is blurry compared to the preview. See this page [https://forum.kde.org/viewtopic.php?f=139&t=127269] for more info.

Shortcuts

Some shortcuts become useless after drawing for a while

Have you loaded any Keyboard or Canvas Shortcut Schemes (e.g. Photoshop/SAI compatible schemes) other than the Default one? If that's the case, make sure you have loaded the same scheme for both Keyboard and Canvas Shortcuts (e.g. Photoshop Compatible for both Keyboard and Canvas shortcuts). If the schemes on both sides are not matching, you might run into shortcut conflicts, like: middle-click zooming/panning/rotation of canvas become unresponsive, even the brush becomes unable to paint sometimes.

License, rights and the Krita Foundation

Who owns Krita?

The Stichting Krita Foundation owns the Krita trademark. The copyright on the source code is owned by everyone who has worked on the source code.

Who and what is Kiki?

Kiki is a cybersquirrel. She's our mascot and has been designed by Tyson Tan. We choose a squirrel when we discovered that 'krita' is the Albanian word for Squirrel.

Why is Krita Free?

Krita is developed as free software [https://www.gnu.org/] within the KDE community. We believe that good tools should be available for all artists. You can also buy Krita on the Windows Store if you want to support Krita's development or want to have automatic updates to newer versions.

Why isn't Krita on Steam and in the Windows Store Free?

Krita on Steam and in the Windows Store is still Free and Open Source software; the binaries are exactly the ones you can also download from krita.org. We've put a price tag on downloading Krita from either store to support Krita's development. Nobody is getting rich out of it, but the income from Steam and the Windows Store currently pays for the full-time involvement with Krita of four developers. See Krita Available from the

Windows Store [https://krita.org/en/item/krita-available-from-the-windows-store/] for more information.

Can I use Krita commercially?

Yes. What you create with Krita is your sole property. You own your work and can license your art however you want. Krita's GPL license applies to Krita's source code. Krita can be used commercially by artists for any purpose, by studios to make concept art, textures, or vfx, by game artists to work on commercial games, by scientists for research, and by students in educational institutions.

You can also make videos or stream your desktop with Krita's interface visible (which can be used to make art tutorials or timelapses).

If you modify Krita itself, and distribute the result, you have to share your modifications with us. Krita's GNU GPL license guarantees you this freedom. Nobody is ever permitted to take it away.

Can I get Krita for iPad? for Android?

> Not for iOS or iPadOS at this point in time: there are problems in any case with putting an application licensed under the GNU Public License V3 in the iOS App Store [https://www.fsf.org/news/2010-05-app-store-compliance]. Krita for Android is currently in beta in the Google Play Store [https://play.google.com/store/apps/details?id=org.krita] F-Droid is coming.

Who translates Krita

Krita is a KDE application [https://www.kde.org/] — and proud of it! That means that Krita's translations are done by KDE localization teams [https://l10n.kde.org/]. If you want to help out, join the team for your language! There is another way you can help out making Krita look good in any language, and that is join the development team and fix issues within the code that make Krita harder to translate.

Reference

https://answers.launchpad.net/krita-ru/+faqs

Contributors Manual

Everything you need to know to help out with Krita!

Contents:

The Krita Community

Get in touch! Apart from the website at https://www.krita.org, the Krita project has three main communication channels:

- Internet Relay Chat (IRC)
- The mailing list
- Phabricator

While Krita developers and users are present on social media such as Twitter, Mastodon, Reddit, Google+, Tumblr or Facebook, those are not the place where we discuss new features, bugs, development or where we make plans for the future.

There are also the:

- bug tracker
- development sprints

You'll find that there are a number of people are almost always around: the core team.

- Boudewijn (irc: boud): project maintainer, lead developer. Works full-time on Krita. Manages the Krita Foundation, triages bugs, does social media and admin stuff. Boudewijn is also on Reddit as boudewijnrempt.
- Dmitry (irc: dmitryK|log): lead developer. Works full-time on Krita.
- Wolthera (irc: Wolthera_laptop): developer, writes the manual and tutorials, triages bugs, helps people out. Works full-time on Krita.
- Ivan Yossi (irc: ivanyossi|log): developer. Works full-time on Krita.
- Agata Cacko (irc: tiar): developer, user supporter. Works full-time on Krita. Also on reddit as u/-tiar- .
- Scott Petrovic (irc: scottyp): UX designer, developer, webmaster.
- David Revoy (irc: deevad): expert user, creates Pepper & Carrot, maintains the preset bundle.
- Alvin Wong (irc: windragon): windows guru.

- Ben Cooksley (irc: bcooksley): KDE system administrator.

Krita's team spans the globe, but most development happens in Europe and Russia.

Krita is part of the larger KDE community. The KDE® Community is a free software community dedicated to creating an open and user-friendly computing experience, offering an advanced graphical desktop, a wide variety of applications for communication, work, education and entertainment and a platform to easily build new applications upon. The KDE contributors guide is relevant for Krita contributors, too, and can be found here [https://archive.flossmanuals.net/kde-guide/].

The Krita Foundation was created to support development of Krita. The Krita Foundation has sponsored Dmitry's work on Krita since 2013.

Internet Relay Chat

IRC is the main communication channel. There are IRC clients for every operating system out there, as well as a web client on the krita website.

- Joining IRC: connect to https://webchat.freenode.net, select a unique nickname and join the #krita and ##krita-chat channels. #krita is for on-topic talk, ##krita-chat for off-topic chat.
- Don't ask to ask: if you've got a question, just ask it.
- Don't panic if several discussions happen at the same time. That's normal in a busy channel.
- Talk to an individual by typing their nick and a colon.
- Almost every Monday, at 16:00 CET or CEST, we have a meeting where we discuss what happened in the past week, what we're doing, and everything that's relevant for the project. The meeting notes are kept in google docs.
- Activity is highest in CET or CEST daytime and evenings. US daytime and evenings are most quiet.
- **IRC is not logged. If you close the channel, you will be gone, and you will not be able to read what happened when you join the channel again. If you ask a question, you have to stay around!**

- It is really irritating for other users and disrupting to conversations if you keep connecting and disconnecting.

Mailing List

The mailing list is used for announcements and sparingly for discussions. Everyone who wants to work on Krita one way or another should be subscribed to the mailing list.

Mailing List Archives [https://mail.kde.org/mailman/listinfo/kimageshop]

The mailing list is called “kimageshop”, because that is the name under which the Krita project was started. Legal issues (surprise!) led to two renames, once to Krayon, then to Krita.

Gitlab (KDE Invent)

Gitlab serves the following purposes for the Krita team:

- Reviewing volunteers’ submissions through Merge Requests (MR) on Graphics/Krita [https://invent.kde.org/graphics/krita/merge_requests] for the code and Documentation/Krita.org Documentation Website [https://invent.kde.org/documentation/docs-krita-org/merge_requests] for the content of the Krita Manual.
- Host the code git repository: https://invent.kde.org/graphics/krita.git . Note that while there are mirrors of our git repository on Github and Phabricator, we do not use them for Krita development.
- Host the Krita Manual content repository: https://invent.kde.org/documentation/docs-krita-org

Do not make new issues on Gitlab or use it to make bug reports.

Do put all your code submissions (merge requests) on Gitlab. **Do not** attach patches to bugs in the bug tracker.

Phabricator

Phabricator serves the following purposes for the Krita team:

- Track what we are working on: https://phabricator.kde.org/maniphest/ This includes development tasks, designing new features and UX design, as well as tasks related to the website.

Do not report bugs as tasks on Phabricator. Phabricator is where we organize our work.

Bugzilla: the Bug Tracker

Krita shares the bug tracker with the rest of the KDE community. Krita bugs are found under the Krita product. There are two kinds of reports in the bug tracker: bugs and wishes. See the chapters on Bug Reporting and Bug Triaging on how to handle bugs. Wishes are feature requests. Do not report feature requests in bugzilla unless a developer has asked you to. See the chapter on Feature Requests for what is needed to create a good feature request.

Sprints

Sometimes, core Krita developers and users come together, most often in Deventer, the Netherlands, to work together on our code design, UX design, the website or whatever needs real, face-to-face contact. Travel to sprints is usually funded by KDE e.V., while accommodation is funded by the Krita Foundation.

Mark-up conventions for the Krita Manual

This details the style conventions for using restructured text for the Krita Manual.

It's recommended to look over the official specification [http://docutils.sourceforge.net/rst.html] for reStructuredText, and given it lives on sourceforge, to save a copy to your harddrive (sourceforge has, at this time of writing, some issues with server uptime):

User Manual:

- Primer [http://docutils.sourceforge.net/docs/user/rst/quickstart.html]
- Quick Ref [http://docutils.sourceforge.net/docs/user/rst/quickref.html]
- Text Cheatsheet [http://docutils.sourceforge.net/docs/user/rst/cheatsheet.txt]

Reference Documentation:

- Introduction [http://docutils.sourceforge.net/docs/ref/rst/introduction.html]
- Markup [http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html]
- Directives [http://docutils.sourceforge.net/docs/ref/rst/directives.html]
- Roles [http://docutils.sourceforge.net/docs/ref/rst/roles.html]

Sphinx specific docs:

- Sphinx' page on restructured text [https://www.sphinx-doc.org/en/master/usage/restructuredtext/index.html] – This is useful for the specific sphinx directives and roles it uses to generate for example table of contents.

There are differences between the official reStructuredText and the sphinx docs multiple ways to do things. This document specifies the suggested conventions to go with.

Contents

Meta data

Each page should start with the following three things:

1. A meta description

   This is a general description of the page. It will be converted to an html meta tag which will be used by search engines:

   ```
   .. meta::
      :description:
          Description.
   ```

2. A list of authors and a license.

   This is just to keep track of who edited the page and to give credit. It should be in a comment so that it will not end up being easily readable by machines. The license of the whole manual is GDL 1.3 and should also be mentioned here:

   ```
   .. metadata-placeholder
   ```

```
   :authors: - Author 1
             - Author 2
   :license: GNU free documentation license 1.3 or later.
```

3. Indexing terms.

   These are comma-separated terms under which the page will be indexed in Index. The generated index is quite useful for both PDF as well as people who are not sure what the exact name is of the term they are looking for. They are defined as follows:

   ```
   .. index:: Keyword, Keyword with Spaces, ! Main Definition
   ```

4. A label.

   This is so we can easily link to the page using `:ref:`label_name``. Try to make this a nice variable name:

   ```
   .. _label_name:
   ```

   After the label you will need to add a heading, as `:ref:`label_name`` will refer to the heading to fill out its link-text.

Headings

Headings will be done in the following order:

```
############
Part/Section
############

For pages that have a lot of subpages.

=========
Heading 1
=========

Start most manual pages with this.

Heading 2
---------

Heading 3
```

```
~~~~~~~~~~

Heading 4
^^^^^^^^^^

Heading 5
''''''''''

Heading 6
""""""""""
```

These conventions were more or less decided by Pandoc [https://pandoc.org/]'s mediawiki to reStructuredText conversion. If you need more than 4 headings, ask yourself first if the page hasn't gotten too complicated and needs splitting up.

Sometimes you need to link to a subsection of a page, add a label above the heading in that case.

Headers should not end with punctuation, as the header will be used as the link name when linking to a label.

Linking

Linking is done with `:ref:`label_name``. When you need an alternative link text, you use `:ref:`actual text shown <label_name>``.

Linking to external pages is done with `` `url`_ `` and `` `link name <url>`_ ``, which'll become link name.

Pandoc [https://pandoc.org/] likes to turn these into `` `link name`__ `` and then add `.. __ :url` at the end of the document. This is a so-called 'anonymous hyperlink', meaning that depending on the order of the links appearing in the text the order of the links at the end of the text are associated with one another. If this sounds confusing and difficult, it is because it is. That is also the exact reason why we'd like to avoid links like these.

Footnotes and further reading

Footnotes can be made in 3 ways, the most common one is with autonumbering, as per reference:

[1] is a reference to footnote 1, and [2] is a reference to footnote 2.

[1] This is footnote 1.

[2] This is footnote 2.

[3] This is footnote 3.

[3] is a reference to footnote 3.

Here is a citation reference: [CIT2002] .

[CIT2002] This is the citation. It's just like a footnote, except the label is textual.

Citation can also be referenced with `` `citation <CIT2002>`_ ``.

We don't actually use footnotes in the manual due to the fact that it is a little bit too academical for our readers. However, we do collect documents and links that give a little bit more information on a topic at the end of a page. Sphinx has the `.. seealso::` directive for linking to external links, while reStructuredText suggests to use `.. rubic:: Footnotes` for specifically collecting footnotes as that plays nice with LaTeX.

Images

Use the image directive for images without captions:

```
.. image:: /images/sample.png
   :width: 800
   :align: center
   :alt: an image.
```

And figure directives for images with captions:

```
.. figure:: /images/sample.png
```

```
   :figwidth: 800
   :align: center
   :alt: an image.

   A caption --  notice how the first letter is aligned with the
```

The latter gives:

A caption – notice how the first letter of the caption in the directive is alig
option.

Images should go into the `/images` folder. By using `/images` instead of `images`, sphinx will know the filepath isn't relative.

In-text Markup

You can make text *emphasized* and **strong** with a single asterisk and double respectively:

```
*emphasize*
**strong**
```

You cannot do both ***emphasized and strong***, so take a pick.

You can subscript text and superscript text by using `:sub:`text`` and `:sup:`text``.

However, use these super-sparingly! It is preferred to use the existing semantic markup in sphinx in any case, because that makes it easier for translators to make decisions about the nature of the text:

```
:menuselection:`Settings --> Configure Krita...`
:guilabel:`File`
:kbd:`Ctrl + Z`
:program:`Krita`
```

Avoid randomly bolding words. It does *not* make the text easier or friendlier to read.

Substitution References

You can create a sort of shorthand for a piece of text or an image by doing:

```
.. |shorthand| replace:: something or the other.
```

which means that if you use `|shorthand|`, in the text, it'll be replaced with 'something or the other'. This is useful for images and text that needs to be formatted in a complicated way, like in the case of "LaTeX".

The krita documentation has `|mouseleft|`, `|mousemiddle|`, `|mousescroll|`

and `|mouseright|`, which'll turn into  ,  ,  and  respectively. These are defined in the sphinx conf.py, and are appended to each rst file.

For links, if you reuse the same link over and over, you can write something like the following at the end of the file:

```
.. _bugzilla: https://bugs.kde.org/
.. _Krita Manual: https://docs.krita.org/
```

Then, when typing a link, you can just use `` `bugzilla`_ `` to link to bugzilla with "bugzilla" used as the text of the link. `` `Krita Manual`_ `` will in turn link to docs.krita.org with the text "Krita Manual".

Lists

Ordinated lists

1. Apple
2. Pear
3. Banana

Or…

A. Table
B. Chair
C. Wardrobe.

I. Augustus
II. Nero
III. Caligula
IV. Trajan

They can be defined as follows:

```
1. Apple
2. Pear
3. Banana
```

```
#. Apple
#. Pear
#. Banana

A. Table
B. Chair
C. Wardrobe

A. Table
#. Chair
#. Wardrobe

I. Augustus
#. Nero
#. Caligula
#. Trajan
```

Unordered lists

- red
- yellow
- green
    - seagreen
    - verdigris
    - teal
    - viridian
    - emerald
        - dark emerald
        - light emerald
            - very light emerald.
- blue

Defined as such:

```
- red
- yellow
- green
    - seagreen
    - verdigris
    - teal
    - viridian
    - emerald
```

```
        - dark emerald
        - light emerald
            - very light emerald.
- blue
```

Definition Lists

A favourite! Definition lists are especially useful when dealing with enumerating all the options in a docker and trying to add a simple explanation behind them.

Definition
: Explanation.

Another option
: Explanation.

To make them.
: You can make them like this:

```
Definition
    Explanation.
Another option
    Explanation.
```

Tables

Purpose	Table type
listing shortcuts	Simple table
lots of colspans	Grid table
Simple but long	List Table

Done as follows:

```
================== ============
Purpose            Table type
================== ============
listing shortcuts  Simple table
lots of colspans   Grid table
Simple but long    List Table
================== ============

+------------------+------------+
|Purpose           |Table Type  |
+==================+============+
|listing shortcuts|Simple table|
+------------------+------------+
|lots of colspans |Grid table  |
+------------------+------------+
|Simple but long  |List table  |
+------------------+------------+

.. list-table::
   :header-rows: 1

   - * Purpose
     * Table Type
   - * listing shortcuts
     * simple table
   - * lots of colspans
     * grid table
   - * simple but long
     * list table
```

Full grid tables are best for when you need all features like complex column and row spans, but they're tricky to make. For that reason, small tables are best off being done with the simple syntax, while really long tables are best done with a list directive because that is just much easier to write and maintain.

Admonishments and asides

Note

Admonishments are sort of like a separate section that the reader needs to

pay attention to.

Admonishments that can be used are the following (in order of seriousness):

Hint

Hints are useful to give a little bit more information on a topic than is useful in the main text. Like, “These packages are named differently in openSuse versus Debian”.

Tip

Extra information on how to do something, like, “You can make a template of your favourite document setup”, or “Use m to mirror the canvas and see errors more easily in your drawing”.

Important

Something that is important to note, but is not necessarily negative.

Warning

This is in general when something is negative.

Attention

General attention grabber. Use this when the subject is more important than warning, but not as important that is could get a dataloss.

Caution

This is for things that could cause dataloss, like forgetting to save, or that

Python currently has no undo functionality.

Danger

This should be for things that are dangerous for the computer in general, this includes things that can cause out of memory style freezes.

Error

This one is probably not relevant for a manual. Sphinx can create these manually given some situations, but our configuration does not do so by default.

Generic admonition that can have any text

Text.

You can make it like this:

```
.. admonition:: Generic admonition that can have any text

    Text.
```

Sphinx also adds:

```
.. seealso::

    Which is useful to collect external links and references.
```

Horizontal Rulers

Horizontal rulers are usually used when the topic switches rather directly. This is very common in more narrative based writing, such as history or fiction. The Krita manual is more instruction and reference style writing, that is to say, we don’t usually tell a long story to indicate how different

elements come together, but rather long stories are there to motivate why certain steps are taken in a certain manner. Topic changes then usually happen because we go into a new section, rather than switching to a related section. It is therefore better to use headings or the `.. Topic::` directive. Headings also make it easier to read.

That said, horizontal rulers can be made with `----`.

The rubric directive

The rubric directive is a heading directive that at first glance looks like “topic”, but where the topic is over several paragraphs, rubric itself only deals with the header, like so:

```
.. rubric:: The rubric directive
```

So, when to use these?

Only use them when you think the subject is too minor to have a proper heading.

Topic
: When the text is separated from the flow, so it goes into a different subject than the text itself is naturally going to.

Rubric
: When the text isn’t separated from the flow, but it does not need a header either.

Admonishments
: Only when they fit semantically. This is especially necessary for the danger and warning admonishments, as seeing them too often can make users blind to them.

Code Snippets

`Inline code snippets` are done with ``backticks``.

Multi-line code snippets are done by ending the previous section with `::`, which'll look like this:

```
This is a paragraph, and we define a preformated snippet like so:

    Be sure to add a white space and a tab afterwards before star
```

You can also use the `.. code::` directive. If you add the language name after it, it'll do the appropriate syntax highlighting:

```
.. code:: python

    # Python comment
    def my_function():
        alist = []
        alist.append(1)
        string = "hello world"
```

Becomes

```python
# Python comment
def my_function():
    alist = []
    alist.append(1)
    string = "hello world"
```

some more…

```cpp
// C++ comment
int myFunction(int i) {
    i += 1;

    // Check if more than 12
    if (i>12) {
        i = 0;
    }
    return i;
}
```

```css
/* CSS comment */
body {
    margin: 0 auto;
```

```
    /* is 800 still sensible? */
    max-width:800px;
    font-size:16px;
    color:#333;
    background-color: #eee;
    padding:1em;
    font-family:serif;
    line-height: 1.4;
}
```

```
<p>this <span style="font-style:italic">is</span> <!-- a HTML com
```

Other preformatted text

One can
preformat
text by
prepending
each line
with a pipe
symbol

Like so:

```
| One can
| preformat
| text by
| prepending
| each line
| with a pipe
| symbol
```

This is generally not used in the manual, and should only be used where it is absolutely required to represent content that needs exact formatting, but never merely for aesthetic reasons.

Glossaries, Terms and Index

These are sphinx features.

Index is used in the top section, right now only single index entries are used.

Glossaries are used for some of the menu entry sections, but not all of them.

Quotes

Quotes are done like this:

```
I am not sure why you'd need quotes in a user manual...

-- Wolthera
```

This becomes a blockquote.

> I am not sure why you'd need quotes in a user manual…
>
> —Wolthera

We do actually use quotes in some places. Try to add a link to the name to define where it came from.

Krita Manual Contribution Guide

Welcome to our new documentation!

We've moved from userbase.kde.org to docs.krita.org, then we moved from Mediawiki to Sphinx. This latter change is because Sphinx allows us to handle translations much better than mediawiki can.

The manual will include:

A reference manual for Krita
: This one is probably what everyone is expecting when they type in docs.krita.org. Dry, basic, 'what does this button do' type of information.

General concept tutorials.
: We've found over the past two years that for certain types of users, a reference manual, even with some examples, just isn't enough. The manual should also provide fast and concise explanations for things, and provide a basic workflow for preparing an image for the web.

  We also have found that certain concepts, such as color management and layer handling are far more advanced in Krita than the average artist is used to. Krita is free and many of its users will not have formal training in digital artwork. So there is no pre-existing artist-focused knowledge on how to use color management or filter layers.

  In addition there are systems that are unique to Krita, for example the brush system, the transform masks, the alpha inheritance and the perspective assistants. Finally, there are users who aren't familiar with even standard painting workflows, and are not flexible enough to understand how to port a tutorial for Sai or Photoshop to Krita.

A list of known tutorials and video tutorials
: Apparently, one of the great things about Krita's team is how we connect with artists and acknowledge that they're doing cool stuff. The same

should count for tutorials, especially because there are ways of using Krita and ways of approaching painting that are unique and we should encourage people to share their knowledge.

Contributor's Manual

Krita is (free) open source software, which makes us effectively a community project, with dozens of volunteers pitching in to make it better. This, of course, requires we keep track of manuals and howto's for new volunteers to come in and help us. The various places we've done this have been rather spread out, and are often under maintained. The contributor's manual is an attempt to solidify all the information. It is therefore very technical in places.

krita.org tutorials

There have been a bunch of tutorials on the krita.org and the krita-foundation.tumblr.com, the former focusing on explaining how to use a new feature and the later stimulated by user request.

FAQ

This one is already online and a merger of the different FAQs that we had. It's currently being translated and we hope to keep this one the primary one to update.

For first timers

Unlike Mediawiki, Sphinx works more like how we write code for Krita.

First things first, you will want to talk to us! For this you can either go to the IRC on krita.org (#krita on freenode.org) [https://krita.org/en/irc/], or, more importantly, make an account at identity.kde.org [https://identity.kde.org/]. The account you make at identity can be used to both access the forum as well as the phabricator [https://phabricator.kde.org], where we organise Krita development.

If you have no idea where to begin, make a Kde identity account and make a post on the forum [https://forum.kde.org/viewforum.php?f=136].

Sphinx works by writing simple text files with reStructuredText mark up, and

then it takes those text files and turns them into the manual. We keep track of changes in the manual by putting them into a version control system called **Git**.

Making changes

Because we use Git, there's only a few people who can put things into the version control system, so if you want to make changes you will need to put it up for review.

Creating merge requests using Edit mode

Note

This method is only suitable if you have no push access to KDE repositories. Otherwise it would commit your changes directly to the repository, which is against the current guidelines.

Recommended for users without a technical knowledge.

Not recommended when you want to change more than one file at once. (See Creating merge requests using WebIDE or Creating merge requests using command line if you want to change more files, or simply edit only one per merge request).

If you have a lot of changes you want to contribute, we recommend trying to follow these instructions.

1. Get a KDE identity.
2. Login to KDE_gitlab [https://invent.kde.org/].
3. Go to the repository [https://invent.kde.org/documentation/docs-krita-org/tree/master] and press *fork*.
4. You should be redirected to the fork of your repository now. Typically it's located at `invent.kde.org/YOUR_KDE_LOGIN_NAME/docs-krita-org`.
5. Come back to the official repository. Make sure you're browsing

`Documentation > Krita.org Documentation`, not your own fork. Otherwise this method won't work correctly.

1. Gitlab has an option to Edit files in the gitlab itself. To access this, go to *Repository* ▸ *Files*.

2. At the top of the page you should see a dropbox with `master` as a chosen item.

3. Find the file you want to edit, open it and then click *Edit*.

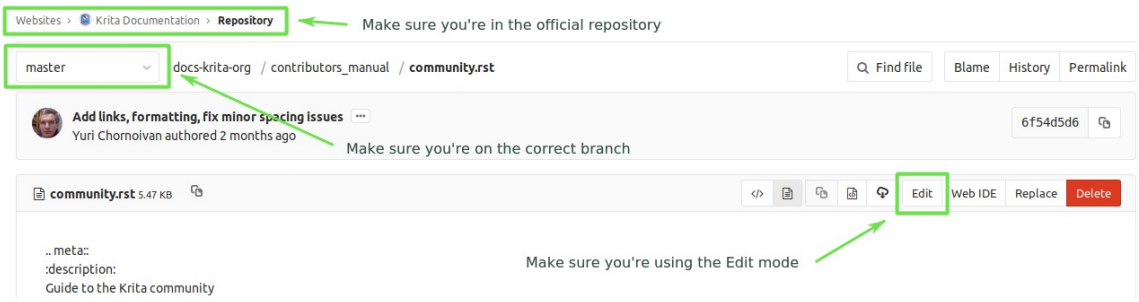
4. Make your changes. (Note: in this mode you can edit only one file at a time).

5. Go to the smaller textbox below and write a nice message in the commit message section with the changes you've made. When done, press *Commit changes*. This will make a merge request for you, just fill in all of the fields as explained here: Guidelines for new merge requests.

   The downside is that right now there's no way to tell if you made errors with the mark up using this method. Please check your changes with the Online Sphinx Editor [https://livesphinx.herokuapp.com/] (just copy and paste the entire file you're editing).

Attention

Edit and *WebIDE* are two different things! Make sure you select *Edit*.



Creating merge requests using WebIDE

Recommended for users with a bit of knowledge about Git that want to edit multiple files at once.

Not recommended when you don't know what a branch is (see Creating merge requests using Edit mode instead).

1. Follow the instructions above to login to KDE_gitlab [https://invent.kde.org/] and create your fork.
2. Go to your fork (make sure the url contains your username).
3. Make sure you're on the `master` branch.
4. Click *WebIDE*. This should take you to a page that has a list of files on the left side and a big empty space for file contents on the right side.
5. Open the files you want to edit and make the changes.
6. Click *Commit…*. Double-click on all files in the *Unstaged changes* category to move them to *Staged changes*.
7. Click *Commit…* again - it will expand a commit message textbox. Write commit message that explains what changes have you made and why.
8. Make sure the settings are correct: you need to select *Create a new branch* (the name of the branch should be: `[username]/[very short description of your changes]`). If you finished your changes, make sure that *Start a new merge request* is checked. Otherwise you'll need to make a new merge request manually later.
9. Click *Stage & Commit*.
10. Fill all of the fields correctly: see Guidelines for new merge requests.
11. To create a new merge request manually, go to Krita Manual official repository (make sure the url *doesn't* contain your username now) and click *Create a new merge request* (bright green button at the left). Select your fork and select the branch that you've created in WebIDE.

Note

If you don't have a push access to the official repository, gitlab won't allow you to save your changes if you were editing the official repository by mistake (and *Create a merge request* won't help with that: you still need to commit your changes to your branch, but if you don't have push access,

you can't do it). It will just show the message: *An error occurred whilst committing your changes. Please try again.*

In this case, simply copy contents of all of the files you changed, go to your fork and paste them in the fork WebIDE.

Creating merge requests using command line

Recommended for users that know how Git works and how to use command line.

Not recommended when you don't know what a branch is (see Creating merge requests using Edit mode instead).

1. Follow the instructions above to login to KDE_gitlab [https://invent.kde.org/] and create your fork.

2. Clone the repository locally with *git clone*. The repository page has the urls you can perform git clone from, and you can then push to your fork. The advantage of this is that you can use all the tools on your computer to edit these text files as well as build the manual locally to check for errors. (You need to do this step only once).

```
# for ssh access
git clone git@invent.kde.org:<username>/docs-krita-org.gi
git remote add upstream git@invent.kde.org:documentation/

# for https access
git clone https://invent.kde.org/<username>/docs-krita-or
git remote add upstream https://invent.kde.org/documentat
```

3. Remember to always pull changes from the official repository before making new changes:

```
git pull upstream master
```

4. Make sure you create a new branch for your changes, since september 2019, all changes should be branched from `master`.

```
git checkout master

# and then:
git checkout -b "<username>/<description of the new featu
```

5. After you make your changes, commit them and push to your fork. For a detailed description of how to use Git in terminal in case of this workflow, go to Forking on Gitlab.

```
# install the python3-sphinx package for your system. For
sudo apt install python3-sphinx
# build the manual (reports potential errors, allows to i
make html
# make sure everything is correct
git status
git diff
# add all of the files
git add .
# commit your changes
git commit
# submit your changes to your fork
git push
```

6. Finally, go to the website of the original repository, and then to Merge Requests. Select your fork and the correct branch and create a new merge request. For instruction on how to fill the fields, see Guidelines for new merge requests.

Guidelines for new merge requests

1. Your commit messages should conform to standards explained here: How to Write a Git Commit Message [https://chris.beams.io/posts/git-commit/]

2. *Title* and *Description* should explain what changes did you make and why did you make them, just like a commit message, so follow the guidelines from the link above in this case, too.

3. *Target* should point to `master`.

4. If you're sure the merge request will demand some changes later, start

the title of your merge request with `[WIP]`.

5. Make sure you checked *Allow commits from members who can merge to the target branch.* – it is often needed for technical reasons that merge request is rebased on master, which technically changes the merge request, but it doesn't change the actual content of it. Rebase can be done by you or by the reviewer – if you don't want to be bothered later too much, better check this checkbox so the reviewer can do it themselves with only a few clicks.

6. You can safely check *Delete source branch when merge request is accepted* in most cases.

7. Unless your reviewers tell you otherwise, check *Squash commits when merge request is accepted*. The first line of the commit message will come from the *Title* of your merge request and the rest of it will be taken from the *Description* of the merge request.

8. When you finish creating your merge request, go to IRC (see Internet Relay Chat) and ask someone with push access to add the `Needs Review` label on your merge request.

9. You might get feedback on your merge request if it has mistakes. Just fix the mistakes in your branch in one of the following ways.

    - If you want to use *Edit* mode, just go to *Changes* section of the merge request and click on the pencil icon (with a tooltip that says *Edit*) to use the Edit mode again.
    - If you want to use *WebIDE* mode, go to your fork, select the branch your changes are on and go to the WebIDE.
    - If you edit files on your computer and work with terminal, make sure you're on the correct branch and push your changes - gitlab will update your merge request automatically.

    After making changes, make sure you ask someone to change the label to `Needs Review` again.

For more detailed information, check out Forking on Gitlab in the technical

section.

Note

At the time of writing this guide setting labels on merge requests is only possible by contributors with write access to the official repository. (If you don't know what that means, you're most probably not one of them). Because of that, when you create or change your merge request you need to get on IRC (see The Krita Community) and ask someone to label it for you.

Building the manual in the command line

For those that first want to try out some changes before embarking on a merge request right away (and already know how to use git and the command line) this is described as part of step 5. in Creating merge requests using command line.

General philosophy

This is for determining what is an appropriate writing style. A writing style, whether we consider its practical or aesthetic qualities, is usually underpinned by a goal or general philosophy. What do we want to achieve with the manual, and for whom is the manual meant?

Demographics and target audience(s)

We cannot talk about a demographic in the sense that we know all Krita users are 55 year old men. Krita is used by a hugely different amount of people, and we are actually kind of proud that we have such a varied userbase.

Despite that, we know a couple of things about our users:

- They are artists. This is explicitly the type of users that we target.
    - Therefore, we know they prefer pretty pictures.

    - They are visual.
    - They are trying to achieve pretty pictures.

Therefore, the implicit goal of each page would be to get the feature used for pretty pictures.

Other than that, we've observed the following groups:

- High-school and college students trying out drawing software for illustrations. These usually have some previous experience with drawing software, like Painttool Sai or Photoshop, but need to be introduced to possibilities in **Krita**. This group's strength is that they share a lot of information with each other like tips and tricks and tutorials.

- Professionals, people who earn their money with digital drawing software. The strength of this group is that they have a lot of know-how and are willing to donate to improve the program. These come in two types:

    - Non technical professionals. These are people who do not really grasp the more mathematical bits of a piece of software, but have developed solid workflows over the years and work with software using their finely honed instincts. These tend to be illustrators, painters and people working with print.
    - Technical professionals. These are people who use **Krita** as part of a pipeline, and care about the precise maths and pixel pushing. These tend to be people working in the games and VFX industry, but occasionally there's a scientist in there as well.

- Adult and elderly hobbyists. This group doesn't know much about computers, and they always seem to get snagged on that one little step missing from a tutorial. Their strength as a group is that they adapt unconventional workflows from real life that the student wouldn't know about and the professional has no time for and create cool stuff with that, as well as that they have a tempering effect on the first group in the larger community.

From these four groups…

- there's only one that is technical. Which is why we need the concept pages, so that we can create a solid base to write our manual texts on top of.
- three of them likely have previous experience with software and may need migration guides and be told how.
- two of them need to know how to get Krita to cooperate with other software.
- two of them have no clue what they are doing and may need to be guided through the most basic of steps.

From that we can get the following rules:

General Writing

Use American English if possible.
: We use American English in the manual, in accordance to Krita's UI being American English by default.

Keep the language polite, but do not use academic language.
: As a community, we want to be welcoming to the users, so we try to avoid language that is unwelcoming. Swearing is already not condoned by KDE, but going to the far other end, an academic style where neither writer nor reader is acknowledged might give the idea that the text is far more complex than necessary, and thus scare away users.

Avoid using GIFs (open for debate)
: The reason is that people with epilepsy may be affected by fast moving images. Similarly, GIFs can sometimes carry too much of the burden of explanation. If you can't help but use GIFs, at the least notify the reader of this in the introduction of the page.

Keep it translation compatible
: This consists of using SVG for infographics, and using the appropriate markup for a given text.

Regarding photos and paintings

- I would like to discourage photos and traditional paintings in the manual if they are not illustrating a concept. The reason is that it is very silly and a little dishonest to show Rembrandt's work inside the Krita GUI, when we have so many modern works that were made in Krita. All of the pepper&carrot artwork was made in Krita and the original files are available, so when you do not have an image handy, start there. Photos should be avoided because Krita is a painting program. Too many photos can give the impression Krita is trying to be a solution for photo retouching, which really isn't the focus.
- Of course, we still want to show certain concepts in play in photos and master paintings, such as glossing or indirect light. In this case, add a caption that mentions the name of the painting or the painter, or mentions it's a photograph.
- Photos can still be used for photobashing and the like, but only if it's obviously used in the context of photobashing.

Regarding images in general

- Avoid text in the images and use the caption instead. You can do this with the figure directive.
- If you do need to use text, make either an SVG, so the text inside can be manipulated easier, or try to minimize the amount of text.
- Try to make your images high quality/cute. Let's give people the idea that they are using a program for drawing!
- Remember that the manual is licensed under GDPL 1.3, so images submitted will be licensed under that. In the case of CC-By-Sa/CC-By ensure that the file gets attributed appropriately through a figure caption. Needless to say, don't submit images that cannot be licensed under either license.

Protocol

So here we line out all the boring workflows.

Tagging and Branches

Adding and removing text will be done in the `draft` branch.

Proofreading results for old pages will be considered as bugfixes and thus will go into the `master` branch and merged into the `draft` branch as necessary.

Before the `draft` branch is merged for a given release:

- The master branch will be tagged with the old version.
- The draft branch is first double checked that it has updated version number and updated epub cover.

The `draft` branch will not be merged until the day before a release to keep the pages intact for long enough.

Each release will have a version of the epub uploaded as part of the release process. .. Where do we get the POT files from? Even the translated versions?

Removing Pages

If a feature is removed in a certain version, the corresponding pages:

1. Will first be marked deprecated.

   This can be done as so:

   ```
   .. deprecated:: version number

      Text to indicate what the user should do without this
   ```

2. Will be linked on a page called ‘deprecated’

3. If the next version rolls around, all the pages linked in the deprecated section will be removed.

Adding Pages

1. Ensure that it is located in the right place.
2. Follow the Mark-up conventions for the Krita Manual to ensure the page is formatted correctly.
3. Add the page to the TOC.
4. If the feature is new, add in versionadded:

```
.. versionadded:: version number

    optional something or the other.
```

As with images, don't add text that you do not have permission to add. This means that text is either written by you, or you have permission to port it from the original author. The manual is GDPL 1.3+ so the text will be relicensed under that.

Changing Pages

If you fully rewrite a page, as opposed to proofreading it, the resulting page should be reviewed.

If you change a page because a feature has changed, and you have commit access, the change can be pushed without review (unless you feel more comfortable with a review), but you should add:

```
.. versionchanged:: version number

    This and that changed.
```

In all cases, check if you want to add yourself to the author field in the metadata section on top.

Using deprecated, versionadded and versionchanged with the version number allows us to easily search the manual for these terms with grep:

```
grep -d recurse versionadded * --exclude-dir={_build,locale}
```

Faulty pages

If a page slips through the cracks, either…

- Make a merge request per the Making changes section.
- Make a task at the Manual Project Workboard [https://phabricator.kde.org/project/view/135/].
- Make a bug at bugzilla [https://bugs.kde.org/] under the project Krita in the section 'documentation'.

Proofreading

There are two types of proofreading that needs to be done.

The most important one is **reviewing changes people make**. You can do this on KDE_gitlab [https://invent.kde.org/] in two ways:

1. Reviewing merge requests

   You can help review merge requests. Request reviewing is usually done by programmers to find mistakes in each other's code, but because programming code is text based just like regular text, we can use this to check against typos as well!

   > A merge request, is an amount of changes done in a document (added, removed) put into a machine readable file. When someone submits a review request (on system like gitlab or github this is a merge or pull request), people who maintain the original files will have to look them over and can make comments about things needing to change. This allows them to comment on things like typos, over-complicated writing but also things that are incorrect. After a patch has been accepted it can be pushed into the version control system.

2. Commenting on changes in the manual.

   > Commenting changes happens after the fact. You can comment on a change by going to the commit message (from the repository

page, go to history and then click on an entry), where you will be able to make comments on the changes made.

In both cases, the interface consists of the difference being shown, with on the left the old version, and on the right the new version. Lines that have been added will be marked in green while lines that have been removed will be marked with red. You can click a speech bubble icon to add an ‘inline’ comment.

The second major way the manual needs to be proofread is **over the whole file**. Many of the pages have only been checked for correctness but not for style and grammar.

For this you will need to follow the Making changes section, so that you can have full access to the pages and edit them.

Translating

Translation of the manual is handled by the KDE localization community [https://l10n.kde.org/]. To join the translation effort, go to the localization site, select the list of translation teams [https://l10n.kde.org/teams-list.php], select the language you wish to translate for, and follow the instructions on the team page to get in contact with fellow translators.

The localization team has access to the PO files for this manual, which is a file type used by translation programs like POEdit and Lokalize. A translation team is able to work together on translating these files and uploading them to the translations SVN. A special script will then take the translations from the SVN and bring them to the manual section to be incorporated on a daily basis.

Images can be translated if a translation team wants to provide their own images. All images in the image folder are by default for ‘en’. When you want to translate a specific image, go into that folder and add another folder with your language code to add in the translated versions of images. So Sphinx will search for a dutch version of `/images/Pixels-brushstroke.png` at `/images/nl/Pixels-brushstroke.png` and for a dutch version of `/images/dockers/Krita-tutorial2-I.1-2.png` in

`/images/dockers/nl/Krita-tutorial2-I.1-2.png`.

Finished translations also need to be added to the build script to show up online. Translator teams which are confident in the state of their translation should contact the main Krita team via the kimageshop mailinglist(kimageshop@kde.org), or foundation@krita.org, to accomplish this.

Other

For restructured text conventions, check Mark-up conventions for the Krita Manual.

Images for the Manual

This one is a little bit an extension to Saving For The Web. In particular it deals with making images for the manual, and how to optimise images.

Contents

Tools for making screenshots

Now, if you wish to make an image of the screen with all the dockers and tools, then Saving For The Web won't be very helpful: It only saves out the canvas contents, after all!

So, instead, we'll make a screenshot. Depending on your operating system, there are several screenshot utilities available.

Windows

Windows has a build-in screenshot tool. It is by default on the `Print Screen` key. On laptops you will sometimes need to use the `Fn` key.

Linux

Both Gnome and KDE have decent screenshot tools showing up by default when using the `Print Screen` key, as well do other popular desktop environments. If, for whatever reason, you have no

ImageMagick
: With imagemagick, you can use the following command:

```
import -depth 8 -dither <filename.png>
```

While we should minimize the amount of GIFs in the manual for a variety of accessibility reasons, you sometimes still need to make GIFs and short videos. Furthermore, GIFs are quite nice to show off features with release notes.

For making short GIFs, you can use the following programs:

- Peek [https://github.com/phw/peek] – This one has an appimage and a very easy user-interface. Like many screenrecording programs it does show trouble on Wayland.

OS X

The Screenshot hotkey on OS X is `Shift + Command + 3`, according to the official apple documentation [https://support.apple.com/en-us/HT201361].

The appropriate file format for the job

Different file formats are better for certain types of images. In the end, we want to have images that look nice and have a low filesize, because that makes the manual easier to download or browse on the internet.

GUI screenshots
    This should use PNG, and if possible, in GIF.

Images that have a lot of flat colors.
    This should use PNG.

Grayscale images
    These should be GIF or PNG.

Images with a lot of gradients
    These should be JPG.

Images with a lot of transparency.
    These should use PNG.

The logic is the way how each of these saves colors. JPEG is ideal for photos and images with a lot of gradients because it compresses differently. However, contrasts don't do well in JPEG. PNG does a lot better with images with sharp contrasts, while in some cases we can even have less than 256 colors, so GIF might be better.

Grayscale images, even when they have a lot of gradients variation, should be PNG. The reason is that when we use full color images, we are, depending on the image, using 3 to 5 numbers to describe those values, with each of those values having a possibility to contain any of 256 values. JPEG and other 'lossy' file formats use clever psychological tricks to cut back on the amount of values an image needs to show its contents. However, when we make grayscale images, we only keep track of the lightness. The lightness is only one number, that can have 256 values, making it much easier to just use GIF or PNG, instead of JPEG which could have nasty artifacts. (And, it is also a bit smaller)

When in doubt, use PNG.

Optimising Images in quality and size

Now, while most image editors try to give good defaults on image sizes, we can often make them even smaller by using certain tools.

Windows

The most commonly recommended tool for this on Windows is IrfranView [https://www.irfanview.com/], but the dear writer of this document has no idea how to use it exactly.

The other option is to use PNGCrush as mentioned in the linux section.

Linux

Optimising PNG

There is a whole laundry list of PNG optimisation tools [https://css-ig.net/png-tools-overview] available on Linux. They come in two categories: Lossy (Using psychological tricks), and Lossless (trying to compress the data more conventionally). The following are however the most recommended:

PNGQuant [https://pngquant.org/]

> A PNG compressor using lossy techniques to reduce the amount of colors used in a smart way.
>
> To use PNGquant, go to the folder of choice, and type:
>
> ```
> pngquant --quality=80-100 image.png
> ```
>
> Where *image* is replaced with the image file name. When you press the `Enter` key, a new image will appear in the folder with the compressed results. PNGQuant works for most images, but some images, like the color selectors don't do well with it, so always double check that the resulting image looks good, otherwise try one of the following options:

PNGCrush [https://pmt.sourceforge.io/pngcrush/]

A lossless PNG compressor. Usage:

```
pngcrush image.png imageout.png
```

This will try the most common methods. Add `-brute` to try out all methods.

Optipng [http://optipng.sourceforge.net/]

Another lossless PNG compressor which can be run after using PNGQuant, it is apparently originally a fork of png crush. Usage:

```
optipng image.png
```

where image is the filename. OptiPNG will then proceed to test several compression algorithms and **overwrite** the *image.png* file with the optimised version. You can avoid overwriting with the `--out imageout.png` command.

Optimising GIF

- FFmpeg [http://blog.pkh.me/p/21-high-quality-gif-with-ffmpeg.html]
- Gifski [https://gif.ski/]
- LossyGif [https://kornel.ski/lossygif]

Optimising JPEG

Now, JPEG is really tricky to optimize properly. This is because it is a lossy file format, and that means that it uses psychological tricks to store its data.

However, tricks like these become very obvious when your image has a lot of contrast, like text. Furthermore, JPEGs don’t do well when they are resaved over and over. Therefore, make sure that there’s a lossless version of the image somewhere that you can edit, and that only the final result is in JPEG and gets compressed further.

MacOS/ OS X

- ImageOptim [https://imageoptim.com/mac] – A Graphical User Interface wrapper around commandline tools like PNGquant and gifski.

Editing the metadata of a file

Sometimes, personal information gets embedded into an image file. Othertimes, we want to embed information into a file to document it better.

There are no less than 3 to 4 different ways of handling metadata, and metadata has different ways of handling certain files.

The most commonly used tool to edit metadata is **ExifTool**, another is to use **ImageMagick**.

Windows and OS X

To get exiftool, just get it from the website [https://www.sno.phy.queensu.ca/~phil/exiftool/].

Linux

On Linux, you can also install exiftool.

Debian/Ubuntu

```
sudo apt-get install libimage-exiftool-perl
```

Viewing Metadata

Change the directory to the folder where the image is located and type:

```
exiftool image
```

where image is the file you'd like to examine. If you just type `exiftool` in any given folder it will output all the information it can give about any file it comes across. If you take a good look at some images, you'll see they contain author or location metadata. This can be a bit of a problem sometimes when it comes to privacy, and also the primary reason all metadata gets stripped.

You can also use ImageMagick's identify [https://www.imagemagick.org/script/identify.php]:

```
identify -verbose image
```

Stripping Metadata

Stripping metadata from the example `image.png` can be done as follows:

ExifTool [http://www.linux-magazine.com/Online/Blogs/Productivity-Sauce/Remove-EXIF-Metadata-from-Photos-with-exiftool]

exiftool -all= image.png

This empties all tags exiftool can get to. You can also be specific and only remove a single tag: *exiftool -author= image.png*

OptiPNG

optipng -strip image.png This will strip and compress the png file.

ImageMagick [https://www.imagemagick.org/script/command-line-options.php#strip]

convert image.png –strip

Extracting metadata

Sometimes we want to extract metadata, like an ICC profile, before stripping everything. This is done by converting the image to the profile type:

ImageMagick's Convert [https://imagemagick.org/script/command-line-options.php#profile]

First extract the metadata to a profile by converting:

```
convert image.png image_profile.icc
```

Then strip the file and readd the profile information:

```
convert -profile image_profile.icc image.png
```

Embedding description metadata

Description metadata is really useful for the purpose of helping people with screenreaders. Webbrowsers will often try to use the description metadata if there's no alt text to generate the alt-text. Another thing that you might want to embed is stuff like color space data.

ExifTool

ImageMagick

Setting an exif value:

```
convert -set exif:ImageDescription "An image description" imag
```

Setting the PNG chunk for description:

```
convert -set Description "An image description" image.png imag
```

Embedding license metadata

In a certain way, embedding license metadata is really nice because it allows you to permanently mark the image as such. However, if someone then uploads it to another website, it is very likely the metadata is stripped with imagemagick.

Using Properties

You can use dcterms:license for defining the document where the license is defined.

ImageMagick

For the GDPL:

```
convert -set dcterms:license "GDPL 1.3+ https://www.gnu.org/li
```

This defines a shorthand name and then license text.

For Creative Commons BY-SA 4.0:

```
convert -set dcterms:license "CC-BY-SA-4.0 https://creativecom
```

The problem with using properties is that they are a non-standard way to define a license, meaning that machines cannot do much with them.

Using XMP

The creative commons website suggest we use XMP for this [https://wiki.creativecommons.org/wiki/XMP]. You can ask the Creative Commons License choose to generate an appropriate XMP file for you when picking a license.

We'll need to use the XMP tags for exiftool [https://www.sno.phy.queensu.ca/~phil/exiftool/TagNames/XMP.html].

So that would look something like this:

```
exiftool -Marked=true -License="https://creativecommons.org/licen
```

Another way of doing the marking is:

```
exiftool -Marked=true -License="https://creativecommons.org/licen
```

With imagemagick you can use the profile option again.

First extract the data (if there is any):

```
convert image.png image_meta.xmp
```

Then modify the resulting file, and embed the image data:

```
convert -profile image_meta.xmp image.png
```

The XMP definitions per license. You can generate an XMP file for the metadata on the creative commons website.

Introduction to User Support

Contents

Tablet Support

The majority of help requests are about pen pressure and tablet support in general.

Quick solutions

Note

With every change mentioned below you might need to restart your PC to see the effect.

For Windows, all devices:

1. Change API in *Settings* ▸ *Configure Krita…* ▸ *Tablet Settings*.

1. Wintab: older standard; it supports multiple buttons and high number of pressure levels. If it works fine for you, don't change to Windows Ink. 2-in-1 devices by default use Windows Ink, you can get a Wintab driver but you need to install it separately.
2. Windows 8+ Pointer (Windows Ink): newer standard; it cuts the pressure levels to 1024. It is more suitable for 2-in-1 devices like Surface Pro and Yoga. Some less known brands might not have this standard implemented.

For Windows, tablet/digitizer devices (not convertible/2-in-1 ones):

1. Reinstall your driver (Windows Update often breaks tablet driver settings, reinstallation helps).
2. *Wacom tablets:* if you get straight lines at the beginnings of the strokes, first try to update your driver: it should be fixed in 6.3.34-3. If it doesn't work, disable/minimize "double-click distance" in Wacom settings.
3. *XP-Pen tablets, pressure being uneven:* either switch to Windows 8+ Pointer (Windows Ink) in *Settings* ▸ *Configure Krita…* ▸ *Tablet Settings*, or disable Windows Ink in XP-Pen settings.

Gathering information

1. Which OS do you use?

2. Which tablet do you have?

3. What is the version of the tablet driver?

4. Please collect Tablet Tester (*Settings* ▸ *Configure Krita…* ▸ *Tablet Settings*) text output and share it: How to share a file.

5. More detailed Tablet Events log:

    1. Go to *Settings* ▸ *Dockers* ▸ *Log Viewer* docker, make sure it's checked.
    2. In the Log Viewer docker, make sure the first button is pressed (which means the logging is turned on).

3. Press `Ctrl + Shift + T` to turn on tablet events logging.
4. Make a few strokes (depending on the situation, the user supporter or developer can ask you for specific series of strokes).
5. Press `Ctrl + Shift + T` to turn off the logging of the tablet events.
6. Press the third button in the Log Viewer to save the output into a file.
7. Share the file or share the content of the file: How to share a file.

On Linux, you can just use a console instead of Log Viewer – then you'd only need to enable tablet events logging, not logging in general.

Additional information for supporters

1. Except for the issue with beginnings of the strokes, Wacom tablets usually work no matter the OS.
2. Huion tablets should work on Windows and on Linux, on Mac there might be issues.
3. XP-Pen tablets and the rest of brands can have issues everywhere (on all systems).
4. If someone asks about a tablet to buy, generally a cheaper Wacom or a Huion are the best options as of 2019, if they want to work with Krita. The List of Supported Tablets.
5. Possibly useful instruction in case of XP-Pen tablet issues [https://www.reddit.com/r/krita/comments/btzh72/xppen_artist_12s_issue_with_krita_how_to_fi

Animation

Issues with rendering animation can be of various shapes and colors. First thing to find out is whether the issue happens on Krita's or FFmpeg's side (Krita saves all the frames, then FFmpeg is used to render a video using this sequence of images). To learn that, instruct the user to render as "Image Sequence". If the image sequence is correct, FFmpeg (or more often: render options) are at fault. If the image sequence is incorrect, either the options are

wrong (if for example not every frame got rendered), or it's a bug in Krita.

Note

If the user opens the Log Viewer docker, turns on logging and then tries to render a video, Krita will print out the whole ffmpeg command to Log Viewer so it can be easily investigated.

There is a log file called *log_encode.log* in the directory that user tries to render to. It can contain information useful to investigation of the issue (sharing files: How to share a file).

Onion skin issues

The great majority of issues with onion skin are just user errors, not bugs. Nonetheless, you need to find out why it happens and direct the user how to use onion skin properly.

Crash

In case of crash try to determine if the problem is known, if not, instruct user to create a bug report (or create it yourself) with following information:

1. What happened, what was being done just before the crash.
2. Is it possible to reproduce (repeat)? If yes, provide a step-by-step instruction to get the crash.
3. Backtrace (crashlog) – the instruction for Windows is here: Dr. MinGW Debugger, and the debug symbols can be found in the annoucement of the version of Krita that the user has. But it could be easier to just point the user to https://download.kde.org/stable/krita.

Other possible questions with quick solutions

1. When the user has any weird issue, something you've never heard about, ask them to reset the configuration: Resetting Krita configuration.

2. When the user has trouble with anything related to preview or display, ask them to change *Canvas Graphics Acceleration* in *Settings* ▸ *Configure Krita…* ▸ *Display*.

   Note

   Telling people to disable canvas acceleration to get better performance is something we shouldn't do, ever.

Advices for supporters

1. If you don't understand the question, ask for clarification – asking for a screen recording or a screenshot is perfectly fine.
2. If you don't know the solution but you know what information will be needed to investigate the issue further, don't hesitate to ask. Other supporters may know the answer, but have too little time to move the user through the whole process, so you're helping a lot just by asking for additional information. This is very much true in case of tablet issues, for example.
3. If you don't know the answer/solution and the question looks abandoned by other supporters, you can always ask for help on Krita IRC channel. It's #krita on freenode.net: The Krita Community.
4. Explain steps the user needs to make clearly, for example if you need them to change something in settings, clearly state the whole path of buttons and tabs to get there.
5. Instead of *Settings* ▸ *Configure Krita…* use just *Configure Krita* – it's easy enough to find and Mac users (where you need to select *Krita* ▸ *Preferences…*) won't get confused.
6. If you ask for an image, mention usage of Imgur [https://imgur.com] or Pasteboard [https://pasteboard.co] (How to share a file), otherwise Reddit users might create a new post with this image instead of including it to the old conversation.
7. If you want to quickly answer someone, just link to the appropriate place in this manual page – you can click on the little link icon next to the section or subsection title and give the link to the user so they for

example know what information about their tablet issue you need.

How to share a file

- Images (e.g. screenshots): Imgur [https://imgur.com] [*], Pasteboard [https://pasteboard.co]
- Text only: Pastebin [https://pastebin.com] [*], BPaste [https://bpaste.net], paste.ubuntu.org.cn [https://paste.ubuntu.org.cn], CentOS's Pastebin Service [https://paste.centos.org/] or KDE Snippets (needs KDE Identity) [https://invent.kde.org/dashboard/snippets].
- `.kra` and other formats: by mail? Or encode the file using *base64* command on Linux, send by mail or on Pastebin, then decode using the same command.

Attention

If you ask user to store their log or other data on a website, make sure it stays there long enough for you to get it – for example bpaste.net stores files by default only for a day! And you can extend it only to one week.

Blocked websites

If the user is behind a firewall of some sorts (for example lives in China), websites with [*] will probably be blocked; please use the alternatives.

Technical Pages

> "In my 20+ year experience managing projects I learned that tools or systems don't manage anything, people do, and if people need tools they'll get them or make them"
>
> —Ton Roosendaal, on #blendercoders

Some parts of the contributor's manual are for people who wish to help with the more technical parts of contributing to an open source project.

Because technical computer terms are very hard to translate, people who wish to do technical contributions must know English. This is not just because these pages would be hard to translate, but also because the main developers who work on the program will have a hard time figuring out the names of technical terms in languages other than English. Therefore, these pages will not be translated.

Outside of these pages we also recommend taking a look at…

- The KDE API Documentation Guidelines
  [https://community.kde.org/Guidelines_and_HOWTOs/API_Documentation]
- KDE wide Guidelines and HOWTOs
  [https://community.kde.org/Guidelines_and_HOWTOs]

Contents:

- Python Developer Tools
- Introduction to Quality Assurance
- Making a release
- Reporting Bugs
- Running Krita from Source
- Strokes queue
- Strokes public API
- Internals of the freehand tool
- Scheduled Undo/Redo
- Processings framework
- Testing Strategy
- Triaging Bugs
- Unittests in Krita

Building Krita from Source

If you want to help developing Krita, you need to know how to build Krita yourself. If you merely want to run the latest version of Krita, to test a bug or play with, you can use the nightly build for Windows [https://binary-factory.kde.org/job/Krita_Nightly_Windows_Build/] the nightly build for Linux [https://binary-factory.kde.org/job/Krita_Nightly_Appimage_Build/], or the nightly build for MacOS [https://binary-factory.kde.org/job/Krita_Nightly_MacOS_Build/].

Contents

You can build Krita on Linux, Windows, MacOS and on Linux for Android. The libraries Krita needs (for instance to load and save various image types) are called dependencies.

Linux is the easiest operating system to build Krita on because all the libraries that Krita needs are available on most recent Linux distributions. For an easy guide to building Krita see Building Krita on Linux for Cats [https://www.davidrevoy.com/article193/compil-krita-from-source-code-on-linux-for-cats].

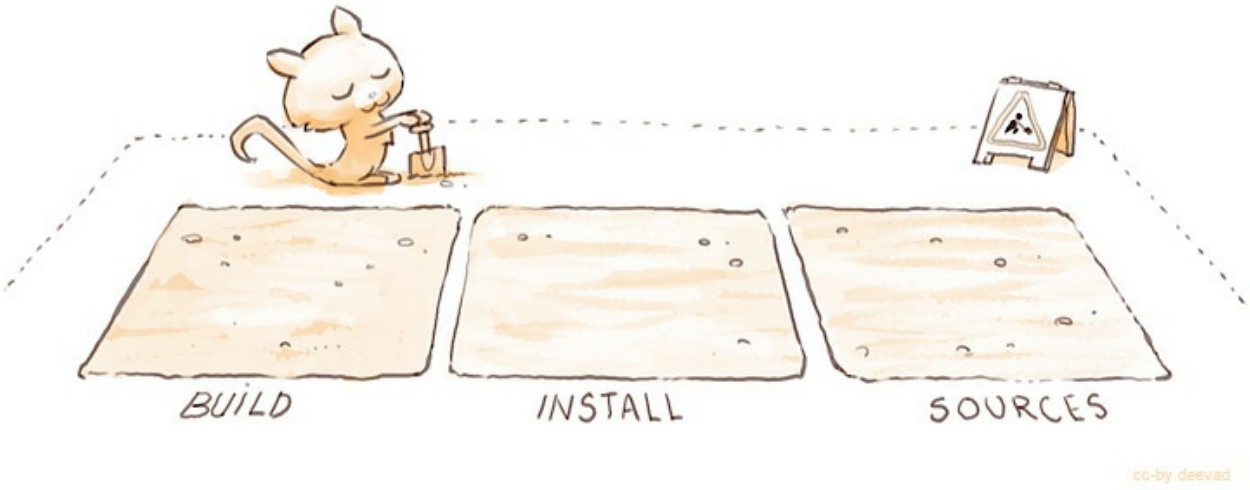
On MacOS you can use tools like homebrew to get the dependencies, or build the dependencies manually. Building the dependencies manually is recommended because we have a number of changes to the source for libraries to make them function better with Krita.

On Windows you will have to build the dependencies yourself.

On all operating systems, you need to be familiar with using a terminal. Building Krita is a technical task and demands accuracy in following instructions and intelligence in understanding what happens.

Building on Linux

Preparing your development environment



The most convenient layout is as follows:

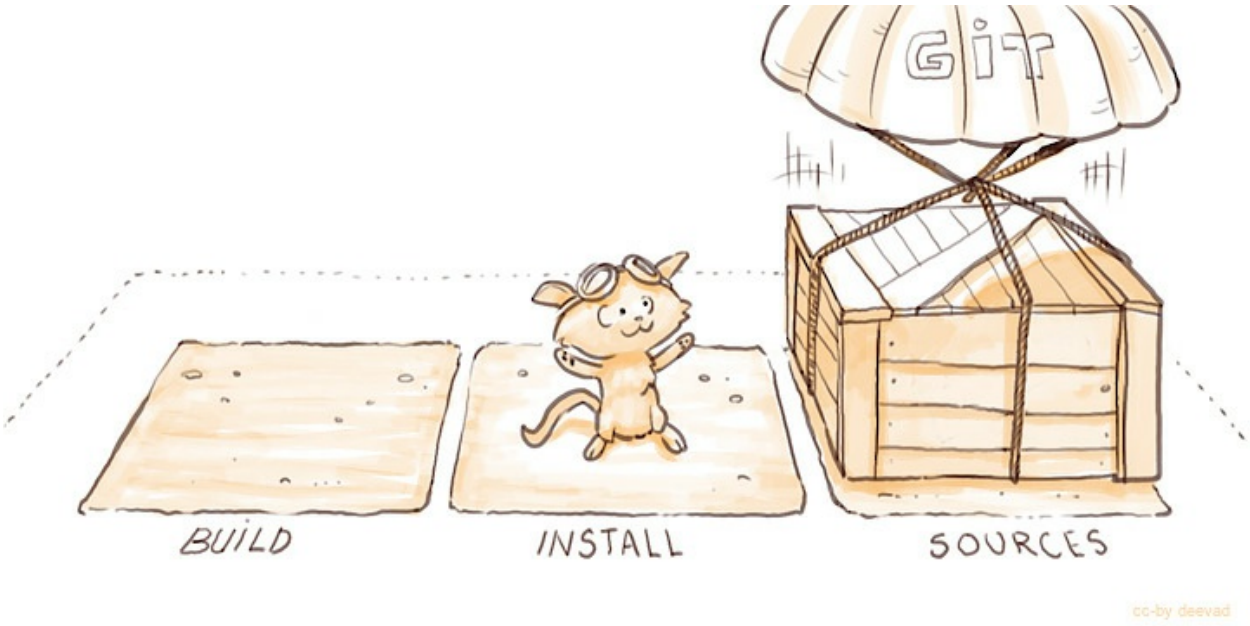
- $HOME/kritadev/krita – the source code
- $HOME/kritadev/build – the location where you compile krita
- $HOME/kritadev/install – the location where you install krita to and run krita from

we will call the “kritadev” folder your build root.

Note: type in what’s shown after ‘>’ in the following commands

```
you@yourcomputer:~>cd
you@yourcomputer:~>mkdir kritadev
you@yourcomputer:~/>cd kritadev
you@yourcomputer:~/kritadev> mkdir build
you@yourcomputer:~/kritadev> mkdir install
```

Getting the Source Code



Open a terminal and enter the build root. Clone Krita from kde's git infrastructure (not github):

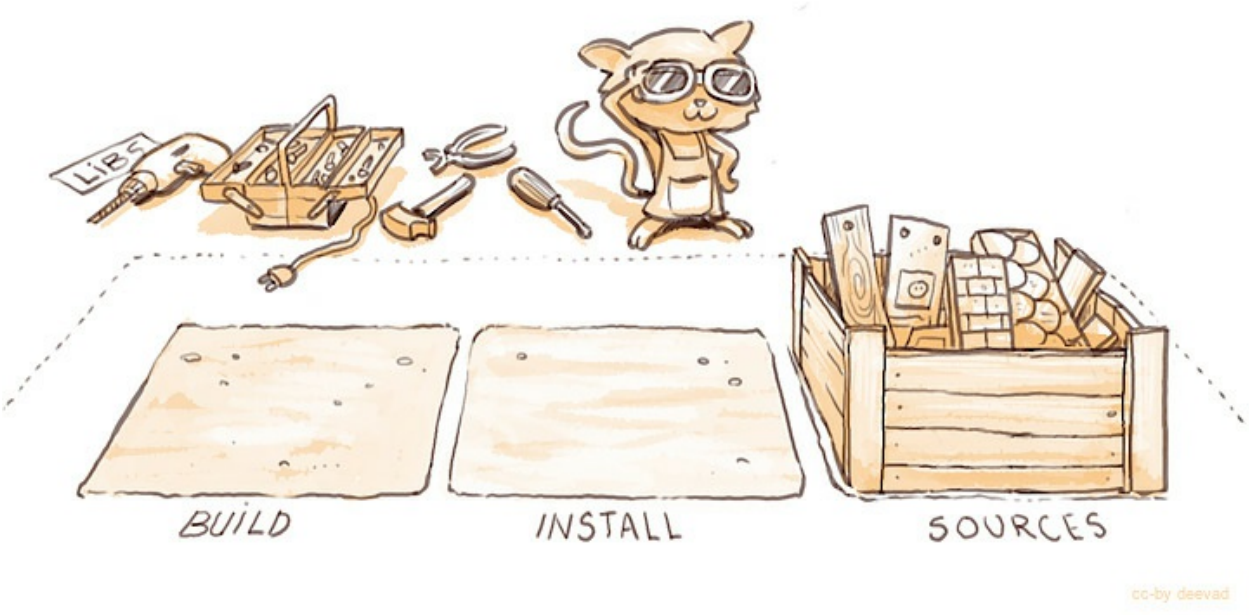
```
you@yourcomputer:~/kritadev> git clone https://invent.kde.org/gra
```

Configuring the Build


```
you@yourcomputer:~/kritadev> cd build
```

Krita uses cmake (https://cmake.org) to define how Krita is built on various platforms. You first need to run cmake to generate the build system, in the `kritadevs/build` directory, then run make to make Krita, then run make install to install krita.

```
you@yourcomputer:~/kritadev/build>cmake ../krita \
        -DCMAKE_INSTALL_PREFIX=$HOME/kritadev/install  \
        -DCMAKE_BUILD_TYPE=Debug \
        -DKRITA_DEVS=ON
```



Unless you have installed all the dependencies Krita needs, on first running cmake, cmake will complain about missing dependencies. For instance:

```
-- The following RECOMMENDED packages have not been found:

* GSL, <https://www.gnu.org/software/gsl/>
Required by Krita's Transform tool.
```

This is not an error, and you can fix this by installing the missing package using your distribution's package manager. Do not download these packages manually from the source website and build them manually. Do use your distribution's package manager to find the right packages.

For example, for Ubuntu, you can start with:

```
you@yourcomputer:~/kritadev/build>apt-get build-dep krita
```

Which will install all the dependencies of the version of Krita in the repositories. You might need to enable the deb-src repositories by editing /etc/apt/sources.list (see https://help.ubuntu.com/community/Repositories/CommandLine) or, if you're using the KDE Plasma desktop, enabling them in the the Settings of the Discover application.

However, the development version might use different dependencies, to find these, you can use `apt-cache search`:

```
you@yourcomputer:~/kritadev/build>apt-cache search quazip
libquazip-dev - C++ wrapper for ZIP/UNZIP (development files, Qt4
libquazip-doc - C++ wrapper for ZIP/UNZIP (documentation)
libquazip-headers - C++ wrapper for ZIP/UNZIP (development header
libquazip1 - C++ wrapper for ZIP/UNZIP (Qt4 build)
libquazip5-1 - C++ wrapper for ZIP/UNZIP (Qt5 build)
libquazip5-dev - C++ wrapper for ZIP/UNZIP (development files, Qt
libquazip5-headers - C++ wrapper for ZIP/UNZIP (development heade
```

You will want to get the 'dev' library here, because you're doing dev, and then Krita is using Qt5, so select that one. If this doesn't help, check the Ubuntu packages search [https://packages.ubuntu.com/].

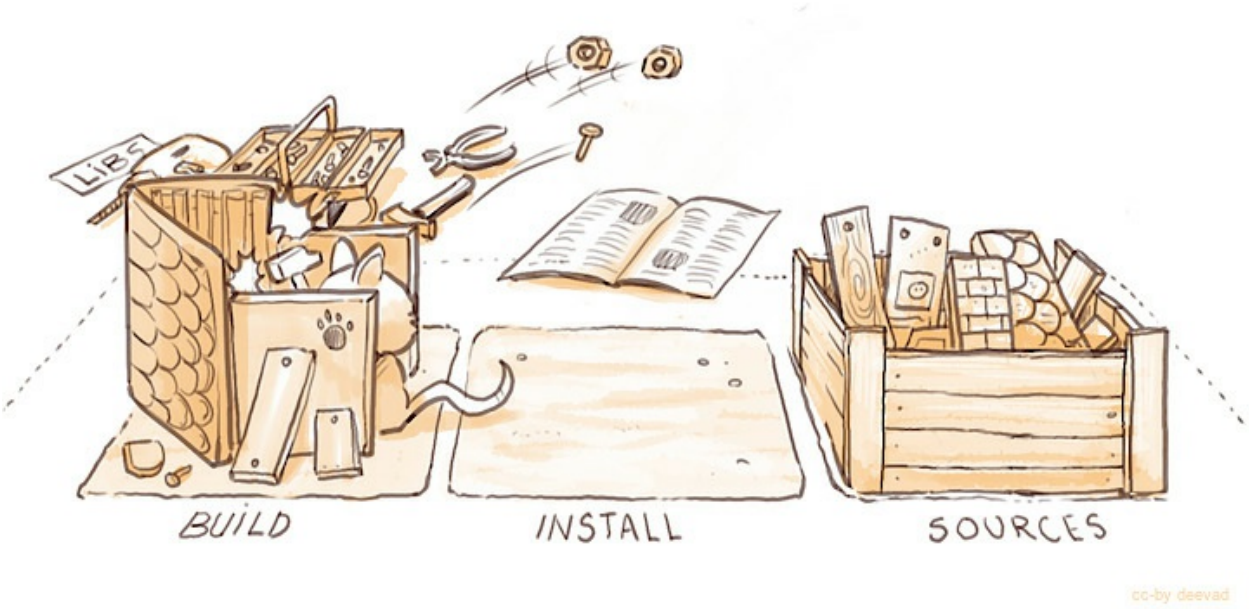
If all dependencies have been installed, cmake will output something like this:

```
-- Configuring done
-- Generating done
-- Build files have been written to: /home/boud/dev/b-krita
```

Warning

There is one run-time package that you need to install. CMake will not warn about it missing. That is the Qt5 SQLite database driver package. On Ubuntu this is named libqt5sql5-sqlite, the name might be different on other distributions. You need this to be able to start Krita after you have built and installed Krita! This is only needed if you build the master (5.0) branch of Krita.

Until that is shown, cmake has not succeeded and you cannot build Krita. When this is shown, you can build Krita:




```
you@yourcomputer:~/kritadev/build> make
```

You can speed this up by enabling multithreading. To do so, you first figure out how many threads your processor can handle:

```
cat /proc/cpuinfo | grep processor | wc -l
```

Then, add the resulting number with -j (for 'Jobs') at the end, so for example:

```
you@yourcomputer:~/kritadev/build> make -j4
```

Installing

When the build has fully succeeded, you can install:

```
you@yourcomputer:~/kritadev/build> make install
```

And when that is complete, you can run Krita:

```
you@yourcomputer:~/kritadev/build>../install/bin/krita
```

Running Krita

You do not have to set environment variables in order to run Krita.

```
you@yourcomputer:~> cd ~/kritadev/
you@yourcomputer:~> ./i/bin/krita
```

Updating

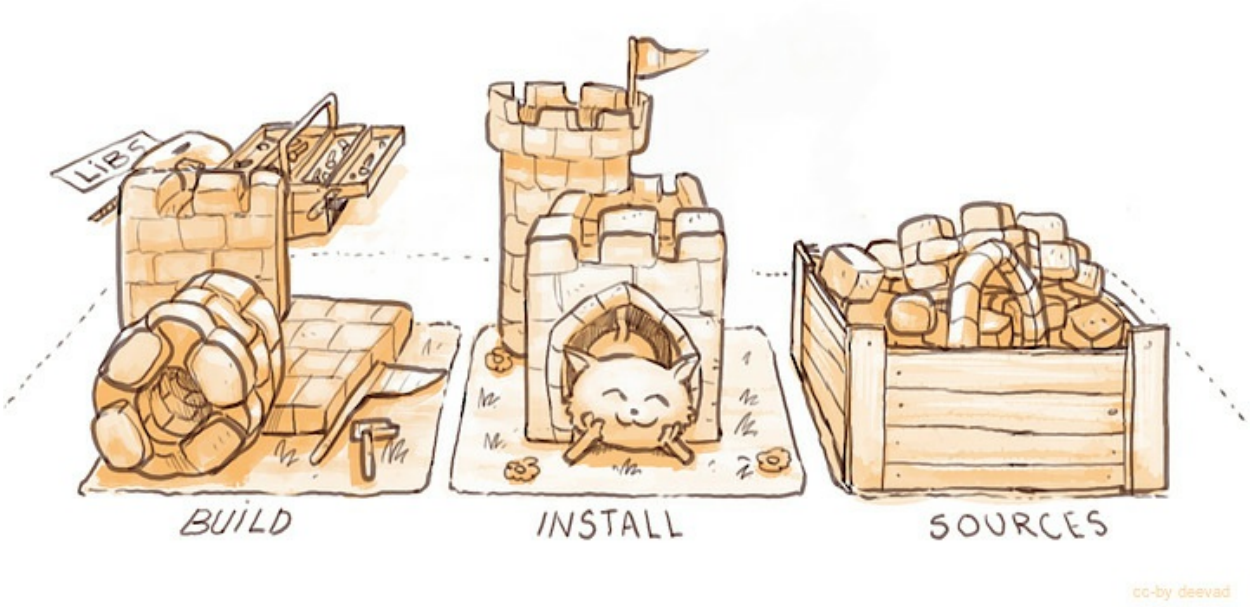
Now, Krita is in constant development, so you will want to update your build from time to time. Maybe a cool feature got in, or a bug was fixed, or you just want the latest source.

First, we get the new source from the git repository:

```
you@yourcomputer:~> cd ~/kritadev/krita/
you@yourcomputer:~/kritadev/krita> git pull
```

If you want to get the code from a specific branch, you will need to `checkout` that branch first:

```
you@yourcomputer:~/kritadev/krita> git checkout <name of the bran
you@yourcomputer:~/kritadev/krita> git pull
```



Then, we build again:

```
you@yourcomputer:~/kritadev/krita> cd ~/kritadev/build/
you@yourcomputer:~/kritadev/build> make install
```

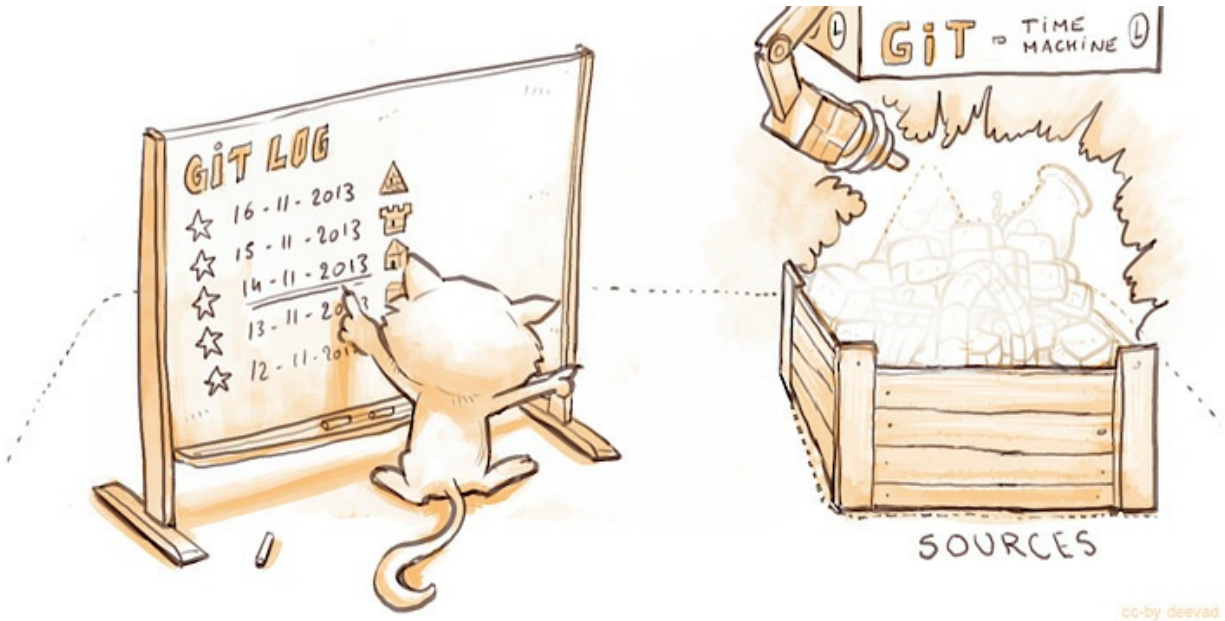
If you update daily, you might want to automate these command by making your own minimal bash script.

Trouble Shooting

The recent development version might break, or sometime be just unusable. Experimental changes are made daily.

This will affect your productivity if you don't know how to 'go back in time' (for example, your favorite brush doesn't work anymore).

But if you know how to do it, *no issue can really affect you*, because you know how to come back to a previous state.



To travel the source in time we need to read the timeline history. The terminal tool for it is `git log`.

```
you@yourcomputer:~> cd ~/kritadev/krita/
you@yourcomputer:~/kritadev/krita> git log
```

With git log, you can consult all the last changes to the code, the ‘commit’. What we’re interested in is the long identification number, the ‘git hash’ (such as `cca5819b19e0da3434192c5b352285b987a48796`). You can scroll the `git log`, copy the ID number then quit(letter `Q` on keyboard). Then time-travel in your source directory:

```
you@yourcomputer:~/kritadev/krita> git checkout cca5819b19e0da343
you@yourcomputer:~/kritadev/krita> git pull
```

And, we build again:

```
you@yourcomputer:~/kritadev/krita> cd ~/kritadev/build/
you@yourcomputer:~/kritadev/build> make install
```

To update again to the actual and ‘fresh from a minute ago’ source-code named `master`, simply ask git to come back to it with `git checkout` then `pull` to update :

```
you@yourcomputer:~/kritadev/krita> git checkout master
you@yourcomputer:~/kritadev/krita> git pull
```

Common problems

Outside of the source being unstable, there's the following common problems:

- The most common problem is a missing dependency. Install it. A missing dependency is not an "error" that you need to report to the other Krita developers.
- A dependency can also be too old. CMake will report when the version of a dependency is too old. That is also not an "error". You might need to update your Linux installation to a newer version.
- You can also have a successful build, then update your linux installation, and then find that Krita no longer builds. A library got updated, and you need to remove the `CMakeCache.txt` file in your build dir and run cmake again.
- Sometimes, changes in Krita's source code from git revision to git revision make it necessary to make your installation and/or build dir empty and build from scratch. One example is where a plugin is removed from Krita; the plugin will be in your install dir, and won't get updated when Krita's internals change.

Building on Windows

On Windows, you will have to build all the dependencies yourself. This will take a long time. Note that you will do all your work in a CMD command window.

This is also more difficult than building Krita on Linux, so you need to pay attention to details. If you follow the guide closely, install correct dependencies and make sure your PATH doesn't contain anything unwanted, there should be no issues.

Prerequisites

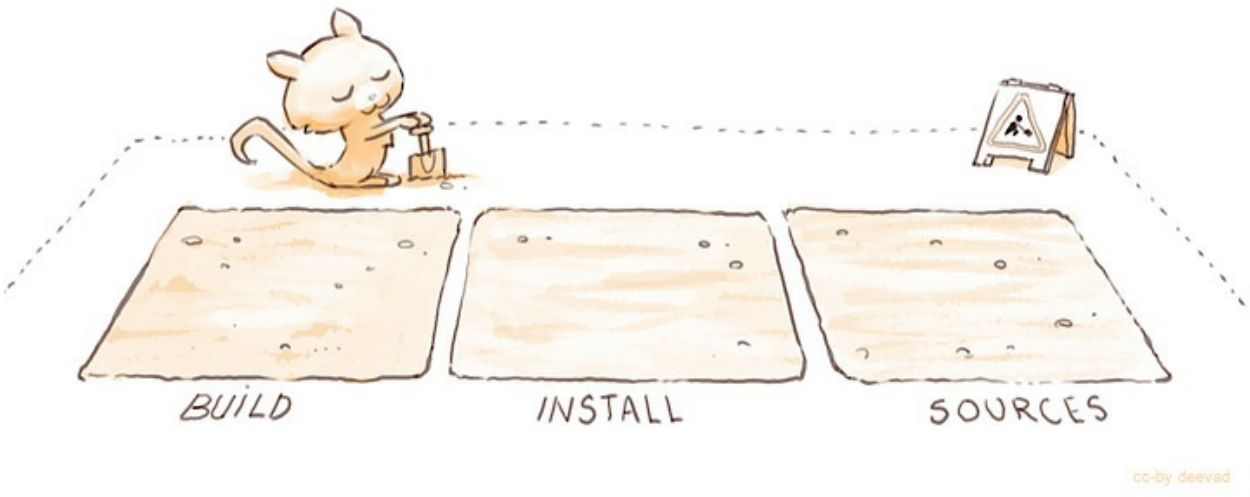
1. Git - https://git-scm.com/downloads
2. CMake 3.3.2 or later - https://cmake.org/download/
    - CMake 3.9 does not build Krita properly at the moment, please use 3.8 or 3.10 instead.
3. Make sure you have a compiler - Only mingw-w64 7.3 (by mingw-builds) - https://files.kde.org/krita/build/x86_64-7.3.0-release-posix-seh-rt_v5-rev0.7z
    - For threading, select posix.
    - For exceptions, select seh (64-bit) or dwarf (32-bit).
    - Unzip mingw with 7zip [https://www.7-zip.org/] into a folder like C:mingw-w64; the full path must not contain any spaces.
    - MSVC is *not* supported at the moment.
    - CLANG is *not* supported at the moment.
    - MSYS is *not* supported at the moment.
4. You will also need a release of Python 3.8 (not 3.7, not 3.9) - https://www.python.org.
    - Make sure to have that version of python.exe in your path. This version of Python will be used for two things to configure Qt and to build the Python scripting module. Do not set PYTHONHOME or PYTHONPATH.
    - Make sure that your Python will have the correct architecture for the version you are trying to build. If building for 32-bit

target, you need the 32-bit release of Python.

5. Install the Windows 10 SDK - https://developer.microsoft.com/en-us/windows/downloads/windows-10-sdk/
6. It is useful to install Qt Creator - https://download.qt.io/official_releases/qtcreator/

MAKE DOUBLE PLUS SURE YOU DO NOT HAVE ANY OTHER COMPILER OR DEVELOPMENT ENVIRONMENT OR PYTHON INSTALLATION IN YOUR PATH

Preparation



After installing the Prerequisites, prepare your working directory. Keep this as short as possible.

```
cd  c:\
mkdir c:\dev
mkdir c:\d
mkdir c:\i
```

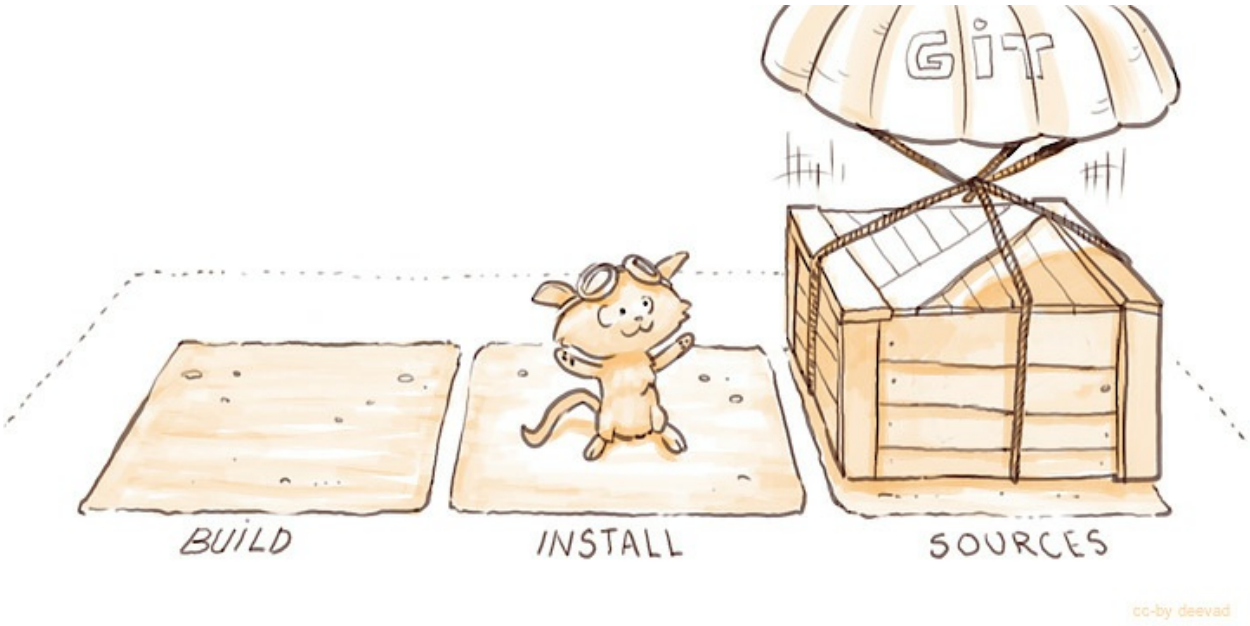
Then prepare a batch file to set the environment. Every time you want to build or run your home-grown Krita, open the CMD windows, go to the c:dev folder and run the env.bat file. Read this example and ADJUST THE

VERSION NUMBERS where necessary so the PATH is correct.

```
set DLLTOOL_EXE=C:\mingw-w64\x86_64-7.3.0-posix-seh-rt_v5-rev0\mi
set MINGW_GCC_BIN=C:\mingw-w64\x86_64-7.3.0-posix-seh-rt_v5-rev0\
set MINGW_BIN_DIR=C:\mingw-w64\x86_64-7.3.0-posix-seh-rt_v5-rev0\
set BUILDROOT=c:\dev
set BUILDDIR_INSTALL=%BUILDROOT%\i
set PATH=%BUILDROOT%\i\bin;%BUILDROOT%\i\lib;c:\python38;c:\pytho
set WindowsSdkDir=C:\Program Files (x86)\Windows Kits\10
```

```
cd c:\dev
env.bat
```

Then get krita:




```
cd c:\dev
git clone https://invent.kde.org/graphics/krita.git
```

Building the dependencies

We will build everything on Windows with the same script that is used to make the nightly builds and the releases:

```
cd c:\dev
krita\build-tools\windows\build.cmd --no-interactive --jobs 8 --s
```

This will take several hours, but you only need to do it once. When it's ready, make a zip archive of the c:devi folder. That's a backup because we will install krita into the same folder as the dependencies, and if you need to nuke your krita build (because you're switching between branches or for some other reason, you'll also nuke your built dependencies. You can also build the depedencies into another folder, like c:devi_deps, BUT in that case you're going to have trouble running Krita without first packaging it.

Building Krita

Again, on the command line, with the same script that is used to make the nightly builds and the releases:

```
cd c:\dev
krita\build-tools\windows\build.cmd --no-interactive --jobs 8 --s
```

If you are hacking on Krita, you can can rebuild Krita without running this script by entering the build directory and running mingw3-make install.

```
cd c:\dev\b_krita
mingw32-make install
```

Running Krita

You must start Krita from the command prompt, after having run env.bat:

```
cd c:\dev\b_krita
env.bat
c:\dev\i\bin\krita.exe
```

Building on MacOS

We will build Krita on MacOS with the same scripts that are used to build the nightly builds and the releases. We will *NOT* be building krita from within XCode, but from within the terminal.

Prequisites

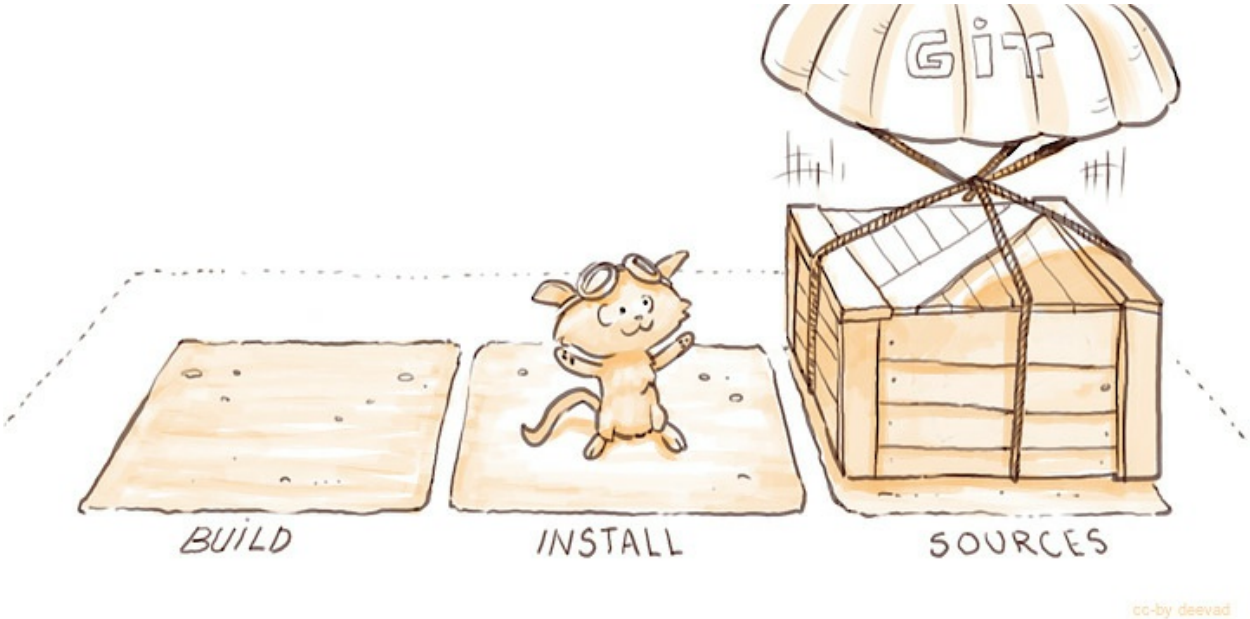
You will need to install:

- CMake: https://cmake.org
- XCode: get it from the app store
- Qt Creator: https://download.qt.io/official_releases/qtcreator/

Preparation

Open Terminal.app

```
cd
mkdir dev
cd dev
git clone https://invent.kde.org/graphics/krita.git
```



Create an env.sh file that should contain the following lines:

```
export BUILDROOT=$HOME/dev
export PATH=/Applications/CMake.app/Contents/bin:$BUILDROOT/i/bin
```

Building the dependencies

It is possible to build Krita against dependencies installed through MacPorts or some similar packaging service. If you do that, you're on your own though.

Open Terminal.app and source the env.sh file you just created:

```
cd ~/dev
. env.sh
./krita/packaging/macos/osxbuild.sh builddeps
```

This will complain several time that it cannot find the Java SDK: just click that away, and don't worry. Building the dependencies will take several hours.

Building Krita

In the same terminal window (if you open a new one, you will have to *source* the env.sh script again by running ". env.sh" – that's a dot.

```
./krita/packaging/macos/osxbuild.sh buildinstall
```

This will build and install Krita to $HOME/dev/i/krita.app

Running Krita

You can run krita in the same terminal window:

```
~/dev/i/krita.app/Contents/MacOS/krita
```

If you want to debug krita with lldb:

```
lldb ~/dev/i/krita.app/Contents/MacOS/krita
(lldb) target create "./i/bin/krita.app/Contents/MacOS/krita"
Current executable set to './i/bin/krita.app/Contents/MacOS/krita
(lldb) r
```

Building on Android

Use Linux to build Krita for Android. Building Krita for Android on another system is *NOT* supported.

Setting up Android SDK and NDK

We right now use Android NDK version `r18b` to do our builds. So, I would recommend to use that. Download it from google's website [https://developer.android.com/ndk/downloads/older_releases.html] then extract it.

Next, Android SDK. You can either download Android Studio or just the `sdk-tools`. Both could be downloaded from google's website [https://developer.android.com/studio].

If you downloaded Android Studio then open SDK manager and download `Android SDK Build-Tools`. (more info: https://developer.android.com/studio/intro/update#sdk-manager)

If you download just `sdk-tools`, then, extract it and run:

```
cd <extracted-android-sdk-tools>/tools/bin
./sdkmanager --licenses
```

```
./sdkmanager platform-tools
./sdkmanager "platforms;android-21"
./sdkmanager "platforms;android-28"    # for androiddeployqt
./sdkmanager "build-tools;28.0.2"
```

If you get some `ClasNotFoundException` it might be because `java` version is set to `11`. For `sdkmanager` to work, set it to `8` and then run it again.

That's the only dependency we have to manage manually!

Building Krita

Now, to build krita, run `<krita-source>/packaging/android/androidbuild.sh --help` and pass the required arguments.

Example:

```
./androidbuild.sh -p=all --src=/home/sh_zam/workspace/krita --bui
```

Installing Krita APK

To install run `adb install -d -r <build-root>/krita_build_apk/build/outputs/apk/debug/krita_build_apk-debug.apk`.

`adb` should be in `<sdk-root>/platform-tools/`

Crash

If Krita crashes you can look up the logs using `adb logcat`

Specialized Ways of Building Krita

These are specialized ways of building Krita on Windows and Linux while re-using the dependencies built on KDE's binary factory. You only need to

try this if you don’t want to build Krita’s dependencies yourself on Windows or use distribution dependencies on Linux.

Contents:

Half-automatic building script on Windows

This guide is aimed at those who want to develop on Windows but are not willing to build the dependencies themselves. It is, admittedly, a pain to prepare all these dependencies all by yourself. Therefore we will use the pre-built dependencies instead.

Note

For simplicity, this guide assumes that you are on a 64-bit Windows system. If you are on a 32-bit system, you may need to adapt the packages to the corresponding 32-bit versions.

The script

The building script lies at https://github.com/tusooa/scripts/blob/servant/bin/krita-build.perl. To run this script you will need a Perl interpreter. If you do not have one yet on your computer, one simple way is to install it through MSys [https://www.msys2.org/]. Msys is not needed for the building process. If you are using the Msys installer, it is advised to install MinGW with it. If you do not want to install Msys, you can just install MinGW [http://www.mingw.org/].

In this article we assume that you have installed MSys.

Dependencies

Most of the dependencies you will need to build Krita is available at the KDE binary factory [https://binary-factory.kde.org/job/Krita_Nightly_Windows_Dependency_Build/]. You will get a zip archive from that link. After downloading, unpack the archive to a place you like.

Besides the dependencies you get from the binary factory, you will also need CMake, Python 3.8, and Boost (if your GCC version is not 7.3 – it is highly probably the case if you have just downloaded installed MinGW).

To install CMake and Boost, open Msys shell and enter `pacman -S mingw-w64-x86_64-cmake mingw-w64-x86_64-boost`.

The Python installer can be downloaded from its official website [https://www.python.org/downloads/release/python-383/]. It is important to use Python 3.8 and not 3.5 or 3.7 or 3.9 or other versions.

Fetch the sources

It is recommended to fetch the source code using Git. Under Msys, you may install Git through the command `pacman -S git`.

You need to choose a directory to store Krita's source code, and switch to that directory. For example, if we choose `C:/Home/Code` as the storing directory, we do:

```
mkdir -pv C:/Home/Code && cd C:/Home/Code
```

Then clone the git repository:

```
git clone https://invent.kde.org/graphics/krita.git.git
```

Or:

```
git clone git://anongit.kde.org/krita.git
```

From now on you will have a `krita` directory under `C:/Home/Code`. We shall then call `C:/Home/Code/krita` the *source code directory*.

It is suggested to create a separate build directory for CMake projects. For example, we chose `C:/Home/Code/krita-build`, so we create and switch to it using:

```
mkdir -pv C:/Home/Code/krita-build && cd C:/Home/Code/krita-build
```

We call `C:/Home/Code/krita-build` the *build directory*.

Invoke the script

Before you invoke the script, it is necessary to edit the configuration part and change the variables there to fit your needs. It is marked in the script by `Config Part -- change as needed`. The following needs to be set up in this manner:

- depsDir – The absolute path to `deps-install` directory you have extracted from `krita-deps.zip`.
- mingwDir – The path to MinGW installation. It is usually the `mingw64` directory under your Msys installation path.
- pythonDir – The path to your Python 3.8.
- kritaInstallDir – Where you want to install Krita.
- kritaSrcDir – The source code directory we have set up before.
- kritaBuildDir – The build directory we have set up before.
- jobs – The maximum number of parallel jobs running through `make`. It is suggested to use (number of processors) + 1 if you want the maximum compiling speed, or (number of processors) - 1 if you want to do other things when building Krita.
- tests – Whether you want to build tests. On Windows many of them are broken, so you can disable tests by setting this variable to 0.

It is suggested to run the script outside the Msys environment. For example, you can use the Command Prompt. The Perl interpreter is located at `<msysDir>/usr/bin/perl.exe`, where `<msysDir>` is the directory where Msys is installed.

Invoke the script under the command prompt using:

```
<msysDir>/usr/bin/perl.exe <absolute path of your krita-build.per
```

For simplicity, we shall now call the line above `<krita-build>`.

Prepare the dependencies for building

After extracting, the dependencies cannot be used directly for the build because it contains hard-coded paths. You will need to run the following command once:

```
<krita-build> prepare
```

How this works will not be covered here for readability reasons. For more information on how this works, please refer to the comments in the script.

Run CMake on the source

Switch to the build directory under the Command Prompt, then run cmake:

```
cd C:\Home\Code\krita-build
<krita-build> cmake
```

Compile and install Krita

Use an IDE to assist in compilation

You may want to use an IDE for development purposes. KDevelop and QtCreator are suggested for developing Krita.

If you use KDevelop, just open the source directory through *Project -> Open/Import Project…* and then choose the build directory that we have set up before. Then, go to *Project -> Open Configuration… -> Make* and choose the `mingw32-make.exe` executable as *Make executable*. It is located in `<mingwDir>/bin/mingw32-make.exe`. Then click *Build* on the toolbar.

Compile on the command line

Alternatively, you can manually invoke the script from the command line to build and install Krita.

```
<krita-build> install
```

Run Krita

Before running, you need to link the dependencies to Krita's installation directory. You may need to start a Command Prompt as Administrator to do so:

```
<krita-build> link-deps
```

This is needed only once, after you have firstly installed Krita. Then you can invoke it using (this does not need Administrator):

```
<krita-build> run
```

Unless you delete the installation directory and perform a `<krita-build> install` again, you will not need to `link-deps` again.

Building krita with Docker on Linux

This guide is useful when you are an advanced developer and want to build krita with the same patched dependencies that are used for the appimages. If you just want to hack on Krita, read the Build Krita from Source guide.

The *Dockerfile* is based on the official KDE build environment that is used on KDE CI for building official AppImage packages. This guide is valid for Ubuntu and Ubuntu-based Linux distributions.

Contents

Prerequisites

First make sure you have Docker installed

```
sudo apt install docker docker.io
```

Decide where you want to store your Docker images. All the docker images and containers are by default stored in a special docker-daemon controlled folder under */var* directory. You might not have enough space there for building Krita (it needs about 10 GiB). In such a case it is recommended to move the docker images folder into another location, where there is enough space.

1. Stop docker service

   ```
   sudo systemctl stop docker
   ```

2. Edit the config file:

   On newer systems, like Ubuntu 18.04 and higher you need to open file */etc/docker/daemon.json* and add the following json config options:

   ```
   {
       "data-root" : "/path/where/you/want/to/store/docker/i
   }
   ```

   If you have older version of Ubuntu, e.g. Ubuntu 16.04, then you need to do the following:

   ```
   echo 'DOCKER_OPTS="-g /path/where/you/want/to/store/docke
   ```

3. Restart the docker service

   ```
   sudo systemctl start docker
   ```

Then you need to download deps and Krita source tree. These steps are not included into the *Dockerfile* to save internal bandwidth

```
# create directory structure for container control directory
git clone https://invent.kde.org/dkazakov/krita-docker-env krita-

cd krita-auto-1
mkdir persistent
```

```
# copy/checkout Krita sources to 'persistent/krita'
cp -r /path/to/sources/krita ./persistent/krita

## or ...
# git clone kde:krita persistent/krita

# download the deps archive
./bin/bootstrap-deps.sh
```

Build the docker image and run the container

```
./bin/build_image krita-deps
./bin/run_container krita-deps krita-auto-1
```

Enter the container and build Krita

```
# enter the docker container (the name will be
# fetched automatically from '.container_name' file)

./bin/enter
```

… now you are inside the container with all the deps prepared …

```
# build Krita as usual
# you should be in ~/appimage-workspace/krita-build/
~/bin/run_cmake.sh ~/persistent/krita
make -j8 install

# start Krita
../appimage-workspace/krita.appdir/usr/bin/krita
```

Building AppImage package for your version of Krita

If you want to build a portable package for your version of Krita, just enter the container and type:

```
~/bin/build_krita_appimage.sh
```

The built package will be copied to *./persistent/* folder.

By default, the package will be built in release mode. If you want to add debugging information, add *–debug* option to the command line:

```
~/bin/build_krita_appimage.sh --debug
```

Creating a full clone of the container

It is possible to copy the container with the entire environment, sources, build directory and QtCreator installation and configuration. After cloning, no rebuild of Krita is needed!

To copy container to *../krita-auto-2*, just type in the host system

```
./bin/spawn-clone -d ../krita-auto-2
```

spawn-clone will make an image from the current container and create a new one out of it. This image will be cached for further usages. If you need to flush the cache, pass *-f* option to *spawn-clone*:

```
./bin/spawn-clone -f -d ../krita-auto-2
```

You can start several instances of *spawn-clone* on the same container concurrently (e.g. for building multiple merge requests). It has internal locking mechanism for resolving concurrency problems

Testing merge requests using container clones

To quickly build a merge request ‘123’ basing on the current state of the container type in the host system

```
./bin/spawn-clone -m 123 -be
```

The script will clone the container, checkout the merge request branch, build it and provide you a terminal for running Krita. The container will be created at *./clones/clone-mr-123* subfolder of the current container.

If you also want to build an AppImage, add *–release-appimage* option:

```
./bin/spawn-clone -m 123 --release-appimage -be
```

AppImage will be places at *./persistent* subfolder of the clone. When finished with testing the merge request, you can remove the clone completely by running

```
./bin/discard-clone /clones/clone-mr-123
```

You can build multiple merge requests at once!

Updating dependencies in the docker

Sometimes dependencies in Krita change and building Krita or making the appimage fails. To fix that, you need to update the dependencies.

Note

This method is slow, because you need to rebuild the whole docker, which includes rebuilding whole Krita.

Run those commands in the console in the host system. If you want to update the dependencies in a clone docker, just go to the clone directory where you see `bin` and `persistent` directories and run those commands there.

```
# remove old dependencies
rm ./persistent/krita-appimage-deps.tar

# download new deps
./bin/bootstrap-deps.sh

# build image
./bin/build_image krita-deps

# remove the current container
./bin/remove_container krita-auto-1

# run the container (it will create one)
./bin/run_container krita-deps krita-auto-1
```

After that you need to build Krita in the docker as usual.

Extra developer tools

To install QtCreator, enter the container and start the installer, downloaded while fetching dependencies. Make sure you install it into '~/qtcreator' directory without any version suffixes, then you will be able to use the script below:

```
# inside the container
./persistent/qt-creator-opensource-linux-x86_64.run
```

To start QtCreator:

```
# from the host
./bin/qtcreator
```

Stopping the container and cleaning up

When not in use you can stop the container. All your filesystem state is saved, but all the currently running processes are killed (just ensure you logout from all the terminals before stopping).

```
# stop the container
./bin/stop

# start the container
./bin/start
```

If you don't need your container/image anymore, you can delete them from the docker

```
# remove the container
sudo docker rm krita-auto-1

# remove the image
sudo docker rmi krita-deps
```

Troubleshooting

Krita binary is not found after the first build

Either relogin to the container or just execute *source ~/.devenv.inc*

OpenGL doesn’t work on NVidia GPU with proprietary drivers

The docker run script automatically forwards the GPU devices into the container, but it doesn’t install the drivers for the GPU. You should install exactly the same version of the driver that is installed on your host system. Just run the following script when you are on host:

```
./bin/install_nvidia_drivers.sh
```

CMake Settings for Developers

The CMake [https://www.cmake.org] build system generators used by Krita is one of the most used build system generators in the C++ world. A build system is a system that describes how an application should be built from source code. CMake generates a build system from the information given in the CMakeLists.txt and *.*cmake* files. It is a complete but rather unusual language.

If you start working on Krita, you will need knowledge of two things: how to run the cmake generator, and which variables are important there, and how to edit the CMakeLists.txt files. This page tells you how to run the cmake generator.

The cmake generator is run like this:

```
cmake -DSOME_CMAKE_VARIABLE=SOME_VALUE ../path/to/source
```

That is, every option is prefixed with -D, followed by a usually uppercase variable name, the equal sign and the value. The following variables are important for Krita.

You cannot build Krita inside the source directory, so you need to give the path to the source directory, where the top-level CMakeLists.txt file is found.

Contents

BUILD_TESTING

If set to ON, the unittests will be built. *All* developers should have this enabled! You run the unittests with `` `make test` ``, or you can run them on their own from their location in the build tree.

CMAKE_INSTALL_PREFIX

This determines where Krita will be installed to. By default this is `` `/usr/local` `` on Linux, which is not what you want.

CMAKE_BUILD_TYPE

This has three options: Debug, RelWithDebInfo and Release. Developers should *always* use Debug, because otherwise ASSERTS will not fire, and developers should pay attention to asserts. Packagers should use RelWithDebInfo.

CMAKE_PREFIX_PATH

This can be set to make the build system look for dependencies in other places than the default one.

HIDE_SAFE_ASSERTS

If set to ON, Krita will not show popups whenever the code encounters a problem that developers need to know about, but users not. If set to OFF,

Krita will popup a little message window telling you about the error, of OFF, it will print the information to the terminal. For developers, either is fine, at least, if you start Krita and pay attention to the terminal output. For packagers, it should be ON.

KRITA_DEVS

This is to be used with the Debug CMAKE_BUILD_TYPE, to re-enable optimizations that make it possible to actually work with Krita. By default, Debug disables all compiler optimizations, and Krita needs those.

PYQT_SIP_DIR_OVERRIDE

If you have built your own PyQt and SIP, use this to make sure Krita can find them.

USE_LOCK_FREE_HASH_TABLE

This option enables the experimental lock free hash table. This is ON by default at the moment.

FOUNDATION_BUILD

This option is for packaging Krita on systems that do not have the default color themes shipped by KDE Plasma.

KRITA_ENABLE_BROKEN_TESTS

A number of unittests are known to be broken. They should be fixed, but in the meantime, having dozens of failing unittests hides regressions. Set this to ON to run the broken tests. These tests are always built.

LIMIT_LONG_TESTS

When set to ON, the default, some unittests will be cut short. Set to OFF to test for stress conditions.

ENABLE_PYTHON_2

Use Python 2 instead of Python 3. Only to be used when integrating Krita in a python2-based VFX pipeline.

BUILD_KRITA_QT_DESIGNER_PLUGINS

OFF by default, enable this to build plugins for Qt Designer/Qt Creator so you can add Krita specific widgets to .ui files.

Introduction to Hacking Krita

Contents

Getting started with KDE Software

Krita is a great place to start even if you are brand new to KDE development. We'd love to have you join! You'll be able to work on one of the coolest and fastest-growing open source painting programs out there. Krita also benefits from a modular architecture and the use of the KDE Frameworks and Qt libraries, which makes it easier to focus on new features instead of reinventing the wheel. And it makes coding fun! To work on Krita, you have to use C++ and Qt. It's a good way to learn both, actually!

KDE has undergone big changes since a major 2014 reorganization

[https://www.kde.org/announcements/kde-frameworks-5.0.php]. As a result, working with KDE software has never been easier. Unfortunately, since the changes were so widespread, the documentation has not caught up at all. If you are embarking on this journey, it would be very generous to share your discoveries with others and update pages. (=

Getting Started

Here's some links to get your started.

1. Most important, the repository [https://invent.kde.org/graphics/krita.git]. There is a mirror on Github [https://github.com/KDE/krita], however note that we do not use Github for development, do not create pull requests or file issues on github.

2. KDE Developer wiki - The KDE Techbase Wiki has instructions for new developers. On top of basic tools like C++, git, and general notions such as building software libraries, some special tools that are particular to Krita are Qt, CMake, and KDE Frameworks. It can be very helpful to get started by finding some of the articles discussing these tools and reading up. Here are some of the more useful pages to get you started:

    - https://techbase.kde.org/Development
    - https://techbase.kde.org/Contribute
    - https://techbase.kde.org/Development/Git/Configuration
    - https://techbase.kde.org/Development/Tutorials
    - https://booki.flossmanuals.net/kde-guide/
    - https://doc.qt.io/ Qt has some of the best documentation of any software library.

3. Set up your development environment and build Krita!

4. Find a few bugs to fix in KDE's Bugtracking system [https://bugs.kde.org/]. It's often a good idea to get some experience with the code through fixing bugs, to get familiar with the development process without being overwhelmed. Though there's nothing against working on that cool feature that scratches your itch!

5. If you intend to be a regular contributor to Krita, even just for bugreports and feature discussion, the first thing you will want to do is register for a KDE Identity account [https://identity.kde.org/]. This serves as your mostly-universal login to KDE code repositories and websites.

Building Krita

To get started, all you need to do is get a copy of Krita and build it! This is not all that much much different from building something off GitHub… except that Krita is very large compared to most software. There are build guides to get you going on various platforms, but of course Linux is easiest.

Working with the Krita codebase

Here's some pointers for working with our codebase.

Architecture

The code base changes all the time with Krita, we're not afraid of big refactorings, so there is no up to date documentation on the code architecture. There have been some written in the past, but they quickly became outdated and of little use. There is a fairly up to date API guide

[https://api.kde.org/extragear-api/graphics-apidocs/krita/html/index.html] if you want to look at how the code is structured.

Integrated Development Environment (IDE)

The most popular IDEs that we use are Qt Creator, Emacs, KDevelop, or vim. Qt Creator has the advantage of the ctrl-k menu, which lets you leap to classes, lines, everywhere. You don't have to build with Qt Creator though! It can be easier to jump to the terminal, do a 'make', check what's up, and then jump back to the IDE.

Resources

The most important step to learning the code is to really understand memory management: pointers, smart pointers and pointer arithmetic. This is something that Java and C# developers will need to spend a little more time understanding. Here are a couple resources to get you more familiar with C++ and Qt:

- Qt Concepts [https://doc.qt.io/archives/qt-4.8/how-to-learn-qt.html]
- Design Patterns with Qt [https://www.ics.com/designpatterns/book/index.html]
- C++ in a Nutshell by O'Reilly (book)

Debugging

There are large and small problems. For small problems the debugger in Qt Creator (run external application) or adding qDebug messages to the code is fine. If the problem is difficult, the first step should always be to write a unit test. A small bit of code that follows a set pattern and exercises the faulty code and shows the problem. That helps so much figuring out a fix and keeping it fixed.

When you run a debug build of Krita, you may be surprised how little debug output you see. This is because most of Krita's debugging information is turned off by default. The debug statements are grouped into categories such as `dbgUI`, `dbgKrita` and so on. The output categories are controlled by an environment variable `QT_LOGGING_RULES`.

The list of Krita's debug categories is contained in `kis_debug.h` and

`main.cc`, and the rules for the environment variable are described in the Qt reference for QLoggingCategory [https://doc.qt.io/qt-5/qloggingcategory.html].

As an example, to enable most of Krita's debug output, you can run the following:

```
export QT_LOGGING_RULES="krita*=true"; krita
```

Using the rule `*=true` will produce a firehose, if you want it.

Tips when Tackling Issues

Features and Refactorings
: Sometimes you just know that a lot of work is going to be needed to reach a particular goal. These will go in separate feature branches off 'master'.

Performance Improvements
: Sometimes you don't feel like working on a feature – or someone mentioned something being particularly slow. The first thing to do then is carry out that scenario when Krita runs under callgrind [http://c.learncodethehardway.org/book/ex41.html] and vtune [https://en.wikipedia.org/wiki/VTune]. These tools show bottlenecks at the end of a run. It's important to use both, since both give different insights!

Bugs
: Sometimes you rummage around the bugs on b.k.o to see what looks like a nice Saturday morning fix. Sometimes a bug is really urgent (like all data loss bugs). Sometimes someone on IRC or the forum mentions a bug. The first thing to do is reproduce it. The second thing is to look in the code to see what is going on. If it's a crash bug, especially one that seems mysterious, it might help to google for a few of the key lines in the backtrace. Sometimes it's a distribution issue!

Blockers
: If you are helping with Krita and your progress is being blocked by something - let us know! Talk with us on the Krita developer IRC [https://krita.org/irc/] and we will see what we can do to help!

Calligra and Krita

In October 2015, the Krita project separated from the rest of the Calligra office suite. The new repository still clearly contains this history. Most source code files will have one of two prefixes. “Ko” stands for KOffice, the original name of Calligra office suite. These files mostly comprise basic, lower-level libraries. “Kis” stands for KImageShop, the original name of Krita. These files are where most of the painting-specific functionality is maintained.

Krita 2.9 stable is built from the Calligra repo. Krita 3.x and above is built from the Krita repo.

Style guidelines

See `HACKING` in the codebase.

Development Philosophy

Krita is nearly ten years old, consists of something like a million lines of code, and has had many individual contributors throughout the years. If you run into something in the code that doesn’t make sense to you, it may very well not make sense to anyone. Developing a codebase this large is an art form, you should feel confident in making risky changes even if you’re not sure they’ll work, you can always go back with `git checkout -- *` if you mess it up!

Getting in Touch

If you’re working on a bug fix, or maybe a bit of GUI polish, you might get stumped. The best thing to do then is to get in touch with the rest of the Krita team. Part of the fun of working on an open source application is the community, after all! Join us on `#krita on webchat.freenode.net` (keep in mind that most people are in Europe or India) and just ask your question. Stay around, especially if you don’t get an answer immediately. Some of the developers have their irc client open permanently and will often answer questions hours later!

You can also send mail to the mailinglist: `kimageshop@kde.org`. It's better not to send mail to individual developers directly, you might accidentally pick someone who hasn't got the answer, and miss the chance of getting your question answered by another Krita developer.

Contributing Patches

Patch review and development tracking happens on gitlab [https://invent.kde.org]. To log in, enter your KDE Identity in the LDAP login field. You can join the Krita: Next [https://phabricator.kde.org/project/profile/8/]. If you are used to Github, the transition to gitlab is not difficult [https://invent.kde.org/help/#new-to-git-and-gitlab], but it is slightly different.

To push to invent.kde.org, you will not need to have SSH access setup, but you do KDE identity account. If several of your merge requests are accepted, you can get a commiter's account, which will allow you to push directly to the repositories. You can read more about that here: Getting a developer account [https://community.kde.org/Infrastructure/Get_a_Developer_Account]

Attention

Since moving to the gitlab instance, we don't use `git@git.kde.org:krita` but rather `git@invent.kde.org:graphics/krita`. Gitlab will not be able to see your commits if you push to the former. You can use `git remote set-url origin git@invent.kde.org:graphics/krita` to get everything pointing correctly.

So then, how does an aspiring contributor submit patches?

Forking on Gitlab

1. Forking on gitlab is done by going to the repository [https://invent.kde.org/graphics/krita.git] and pressing *fork*. You will then make a personal fork of the repository.

Your fork will probably be located here:

```
https://invent.kde.org/<username>/krita
```

1. In your fork, you press *clone* to get the git urls to do the `git clone` from. You can then pull and push your commits from these.

   ```
   # for ssh access
   git clone git@invent.kde.org:<username>/krita.git
   # for https access
   git clone https://invent.kde.org/<username>/krita.git
   ```

   You can also use the *Web IDE* to make your changes directly on invent.kde.org, but because Krita is a c++ program, we don't recommend this outside of typo fixes and doxygen documentation strings. You wouldn't be able to see the effect of your changes, after all!

2. Set up a new remote which points to the official repository, so you'll be able to update your local master branch.

   ```
   # for ssh access
   git remote add upstream git@invent.kde.org:graphics/krita
   # for https access
   git remote add upstream https://invent.kde.org/graphics/k
   ```

   After that, you can see all of your urls using:

   ```
   git remote --verbose
   ```

   As you can see, *origin* points to your fork, while *upstream* points to the official repository.

3. Create a new branch and checkout to it.

   ```
   git checkout -b "<username>/<description of the new featu
   ```

4. Make your first fix, push everything to your branch in your fork.

   ```
   # make sure you didn't leave any unnecessary debug or unf
   git diff
   # stage all changes
   ```

```
git add .
# make sure that all added files are the ones you want to
git status
# commit changes (here, write a commit messages that foll
git commit
# push to your branch
git push
```

Attention

Make sure all of your commits go to your own branch, not onto master.

5. Once you're done, login to the KDE gitlab instance, go to *merge requests* and press *new merge request*

6. Make sure your merge request is between the branch from your fork and the official master branch.

7. Write a detailed description about the changes that you are proposing with your merge request. If it is a change in the user interface, it would be good if you can provide screenshots through attachments.

Tip

The Krita repository has a merge request template that labels your request appropriately and gives a checklist of common formalities that all patches should adhere to. You can select it from the *Template* drop down.

The Krita developers will be notified of new merge requests, and they will try to review your request as soon as possible. If you suspect your patch slipped through the cracks, don't hesitate to contact us through the means described above.

Update the master branch in your fork

After working for some time, you may want to update the master branch of your fork to be in sync with the master branch of the official repository.

1. Checkout the master branch in your working environment.

   ```
   git checkout master
   ```

2. Pull changes from the official repository.

   ```
   git pull --ff-only upstream master
   ```

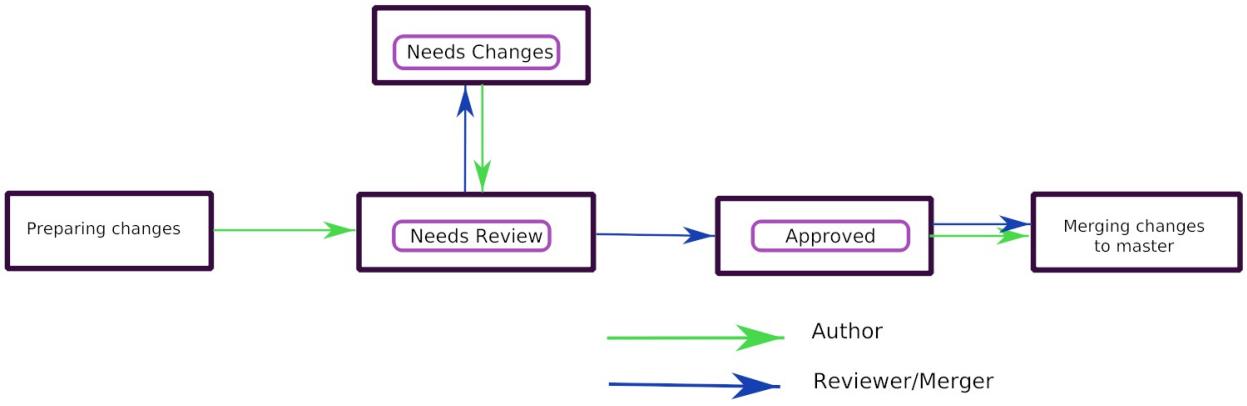
3. Push it to your fork.

   ```
   git push
   ```

   The other possibility is to just delete the fork you worked previously on and create another one – it will be up-to-date with official repository's master branch already.

Label workflow

Make sure the state of your merge request is labeled correctly. The picture below shows the basic label workflow that your merge request should go through:



1. When you create a merge request, mark it with WIP to make sure no one will accidentally merge your request prematurely.

2. When you finish your work, label it with `Needs Review`. That will let

developers know your merge request is ready.

3. A Krita developer will read and test your merge request. After that they will write comments and label the merge request accordingly:

    - If the merge request is ready to be merged, with `Approved` label.
    - If it requires changes to proceed, with `Needs Changes` label.

4. If your merge request is in `Needs Changes` state, please address the concerns of the reviewer and submit the code to your branch. Gitlab will update your merge request accordingly. Add `Needs Review` label to your MR again.

5. When your merge request is in the `Approved` state, you can either merge the code yourself to master if you have developer access, or wait for KDE developer to do it for you.

    Attention

    If you have developer access and merge someone's merge request to the repository, you are partially responsible for the code.

    - Don't merge MRs that weren't approved!
    - Read and test extensively all MRs before you approve or merge!

Note

In time of writing, setting labels on merge requests were only possible by contributors with write access to the official repository. (If you don't know what that means, that probably means you're not one of them). Because of that, when you create or change your merge request you need you get on IRC (see The Krita Community) and ask someone to label it for you.

How to prepare your commits for a merge request

After merging to master, your commits should fit nicely in the Krita git history.

- Commit messages should clearly and concisely state what changes you made with that particular commit and why – see How to Write a Git Commit Message [https://chris.beams.io/posts/git-commit/].
- Every commit should be compilable and follow the KDE commit guidelines – see KDE Commit Policy [https://community.kde.org/Policies/Commit_Policy].
- Commits should be self-contained: if you code a bigger feature, it's better if you divide the code into bits that can possibly exist independently.
- When you add new features during the development, it's fine to add new commits.
- If you only need to fix previous commits, don't add new ones – instead, amend the ones that you made before and force-push your new commits to the branch in your fork.

```
# if you already committed your changes...
git commit
# ...add all changed files the "staged" state
git add .
# and amend the previous commit
git commit --amend
```

Note

You can only force-push to your own branch on your own fork. If you need to remove changes from one of the commits that are already in the official repository, please use `git revert`.

- When you want to reduce the number of commits:
    - you can squash them before making a merge request.

    - if you have developer access, you can squash the commits just before merging with master.
    - See the Beginner's guide to rebasing and squashing [https://github.com/servo/servo/wiki/Beginner's-guide-to-rebasing-and-squashing#squashing] for further guidance.

- Your work should go to a new branch, instead of master.

- Your commits will be rebased and put in master using fast-forward merge. If you need a manual merge (if, for example, you're working on a big feature) and you don't have the commit access, please contact a Krita developer.

Krita SVG Extensions

Krita has a few extensions over SVG format to ensure correct saving and loading of Krita custom elements.

Attribute: `krita:marker-fill-method`

Possible values:

- `default` – markers are filled according to SVG standard rules, that is each marker has its own fill, which is filled in the marker's local coordinates.
- `auto` – markers are considered to be a part of the path. The outline of the path is combined with the outline of the markers and filled with a single pass of the object's fill strategy.

Default value: `default`

[DEPRECATED] Attribute: `krita:arc='arc'` and `krita:arcType='chord'`

That is a temporary namespace that was used before introduction of sodipodi:arc-type='chord' option. Krita never saves files with this option and the support of it will be removed soon.

bg.. meta::

description: Guide to using features from C++11, C++14 and beyond in Krita's codebase.

Modern C++ usage guidelines for the Krita codebase

Contents

General links about using Modern C++ in Qt

There have been a few links discussing mixing C++11 with Qt, and starting with Qt 5.6 C++11 support will be default. *Note:* there is a lot of hype about C++11, and although many of its new features are quite welcome, often the trade-offs from these changes get neglected.

- ICS.com [https://www.ics.com/blog/qt-and-c11]
- qt.io [https://blog.qt.io/blog/2011/05/26/cpp0x-in-qt/]
- woboq.com: c++11 in Qt5 [https://woboq.com/blog/cpp11-in-qt5.html].
- woboq.com: c++14 in Qt5 [https://woboq.com/blog/cpp14-in-qt.html].
- FOSDEM 2013 presentation slides [https://archive.fosdem.org/2013/schedule/event/introcplusplus11/attachments/slides/203/export/

Here are some more general purpose guides to C++11 features.

- C++11 FAQ Bjarne Stroustrup's [http://www.stroustrup.com/C++11FAQ.html] - the grand daddy.
- Older, more thorough introductions to several topics [https://www.informit.com/authors/bio/e19aded6-574c-4c46-8511-101f9f0ed8f8].

Qt's API design principles do not always overlap with the C++ Standards Committee design principles. (Range-based for demonstrates the design clash pretty clearly.)

- https://wiki.qt.io/API_Design_Principles

Particular Features

Under "drawbacks," every item should list: "Programmers will face another feature they must learn about."

Type Inference (auto)

Motivation:

> If a function `f` has a return type Type, it is redundant to write a local variable `Type x = f(y)`. Using auto declarations is a simplification in two ways scenarios. First, it allows the programmer to write code without worrying about doing the manual type deduction, for example:

```
for( KoXmlReader::const_iterator x = iter.begin(),... ) { }
```

versus:

```
for (auto x = iter.begin(), ...) { }
```

This is particularly useful with nested template types and C++11 lambdas, and other complex types which have an obvious role, but a lengthy type definition.

A second important benefit of auto is that it allows the programmer to more easily refactor. Suppose we have a function `gimmeSomeStrings()` which returns a `QList<QString>`, and we access it somewhere else like this

```
auto someStrings = gimmeSomeStrings();
```

If we later decide that we want to store a hash of strings and that `gimmeSomeStrings` should return a `QMap<int, QString>`, we probably won't need to make any changes inside the client snippet if we are doing tasks like iterating.

Drawbacks:

the use of auto is be obfuscating. For example, `auto x = 2` is not obviously an integer, and `auto x = {"a", "b", "c"}` returns `std::initializer_list`, and sometimes it is not clear what some function returns by the name of the function.

Recommendation:

Do not use auto, except, maybe, in loops, where there can be no confusion about the type of what is looped. But even there, hesitate.

Range-based for loop

Motivation:

This is something a long time coming in C++. It is a standardized replacement for Qt's foreach() construct, which works not only with Qt objects but all iterable C++ types.

```
for (T x : list ) { ... }
```

It will work with standard tooling and static analysis, and can be faster by defaulting to in-place access. For this reason range-based iterators should always be used for STL containers, if those are ever needed in Krita.

Drawbacks:

By default, Qt's foreach rewites the code to make a shallow copy and then use const accessors, while c++11 does the opposite, avoiding copying when possible. When using const accessors, this is faster, but if you try to make changes to the data, this will slow your loop down instead [https://www.dvratil.cz/2015/06/qt-containers-and-c11-range-based-loops/].

Recommendation:

> Sometimes, the range-based for is faster. Sometimes the Qt iterator is faster. Personally I like the range-based for in principle, since it works better with static analysis, it has a faster best-case speed, and it is always possible to write it in a way that replicates the `foreach()` behavior, though the reverse is not true.

On the other hand, there is a bad, dangerous worst case performance hit when a detach/copy is triggered, and this is not easy to catch with standard syntax. In the blog post linked above, the discussion explains that is possible to get around this limitation by defining a macro `const_()`, which will gives a new syntax to request the compiler use constant iterators:

```
for (T x : _const(list) ) { ... }
```

Qt's recommendation on the other hand is to use foreach() for Qt iterators, and range-based for on STL containers, because you always know what you're getting, and you always keep your syntax easy to read. In my opinion is the most meaningful new feature without any sort of clear answer, and quite interesting to think about.

General Initializer Lists

Motivation:

Initializer lists are intended to work in many different places to simplify the syntax for complicated initialization. For example, a list of strings could be initialized `const QStringList x = {"abc", "def", "xyz" };` and if you later changed the type to `QVector<QString>`, or even `std::list<std::string>`, you wouldn't have to make any change to the right hand side.

A second place initializer lists are used is in creating standard initial values for class members. This takes the place of writing a lengthy constructor list like:

```
Type::Type()
 : MemberString1("a")
 , Subclass1(0)
 , Subclass2(1)
 , ...
```

In addition to being more concise, it saves you from repeating yourself, if you have several constructors which all start with the same defaults.

Mixed uniform initialization is a separate new feature of initializer lists when constructing classes. It is possible to specify some defaults when you declare member variables, but then override them with delegating constructors. [This MSDN page is a good reference](https://msdn.microsoft.com/en-us/library/dn387583.aspx) [https://msdn.microsoft.com/en-us/library/dn387583.aspx].

Drawbacks:

None I can think of. This is super simple, completely obvious to read and write, and shortens code by removing long unnecessary lists of defaults.

Recommendation:

Yes!

Lambdas, and new-style signals/slots

Motivation:

Lambda expressions are a big new addition for C++11. Many programmers claim they start to feel like an essential part of the

language very quickly. One of the biggest uses for lambdas is in the standard algorithm library <algorithm>, which is described below. In Qt5, this, along with std::function and std::bind, allow for One of the most useful C++11 integrations, a new signal/slot syntax which replaces the moc macros SIGNAL() and SLOT() with standard C++.

Old style:

```
connect(sender, SIGNAL (valueChanged(QString,QString)),  recei
```

New style:

```
connect(sender, &Sender::valueChanged, receiver, &Receiver::up
```

New style signals and slots provide a great benefit from the tooling perspective: now, all types for functions and function arguments can be checked statically, and you don't have to catch typos by monitoring debug messages saying "no such slot."

Another possibility is to use lambdas directly inside connect(), instead of defining a class member function which is only used once. The greatest benefit is that the function can be defined right where it is used; it also aids readability to get rid of a list of tiny helper functions from the header.

- "Qt5: C++11 lambdas are your friend" [https://artandlogic.com/2013/09/qt-5-and-c11-lambdas-are-your-friend/]
- C++ language reference [https://en.cppreference.com/w/cpp/language/lambda]
- Qt.io New Signal/Slot Syntax [https://wiki.qt.io/New_Signal_Slot_Syntax] Also gives detailed pros/cons.

Drawbacks:

The new-style syntax makes it somewhat harder to use default arguments, which requires the use of lambdas. It is also perhaps a little less pretty.

Lambdas in general are have become one of the most clunky pieces of C++11 notation. Since they allow a great deal of options for example, capturing by reference with `[&]` and capturing by value with `[=]`, they are a significant new addition to the C++ learning curve. Using small local

functions with uninformative names like `auto F = [&] ( x ) { whatever }` is confusing for everyone.

Although it is possible to use lambdas are tricky inside signals and slots, there are gotchas. Lambdas will not disconnect automatically, although there is a special syntax to make that happen.

Recommendation:

Lambdas will feel strange to many C++ programmers. At a minimum, any time you use them you should add a comment explaining what you're doing. (Krita codebase could use more comments anyway.) New style signals and slots should be used with caution, especially while the 2.9 branch is being maintained.

Overall, the Qt wiki gives a good overview, and I agree with its suggestions, which is to permit a small amount of mixing of the different syntax. Their recommendation is to use new-style signals and slots when possible, which is the vast majority of the time, to fall back on the old macros when one needs to use a default argument, and to use lambdas very rarely, only in cases when one needs to create a signal that is not bound to a particular object. The latter sort of case is not something that C++ newcomers would want to be touching anyway.

constexpr

Motivation:

Performing calculations at compile time can speed things up at runtime. KDAB: speed up your Qt 5 programs using C++11 [https://www.kdab.com/wp-content/uploads/stories/slides/DD12/mutz-dd-speed-up-your-qt-5-programs-using-c++11.pdf]

Drawbacks:

Not easy to use these features.

Recommendation:

This could be useful in specific places, like KoCompositeOpRegistry. Overall it is not something most programmers will run into.

<algorithm>

Motivation:

A handwritten loop that looks for occurences of the number 20 and replaces it with 99 is routine, and will take several lines to write, including defining local variables. Instead, something like

```
std::replace (myvector.cbegin(), myvector.cend(), 20, 99);
```

is more concise, safer is even self-documenting, since the name of the function itself explains what it is doing. <u>If you make sure to use Qt's const iterators</u>, there should never see a performance penalty compared to a hand-written loop, there can sometimes even see a gain. A list of standard algorithms can be found here. [http://www.cplusplus.com/reference/algorithm/] Historically Qt provided its own algorithm library, but now encourages programmers to use the STL versions instead, and Qt's own algorithm library will mostly become deprecated. https://doc.qt.io/qt-5/qtalgorithms.html Unlike range-based for, where it is difficult to specify a const iterator instead of a standard iterator, with `<algorithm>` we are easily able to specify the const iterator.

Drawbacks:

Some of the standard algorithms are not completely obvious from observing the name. For example, I could not personally list what are the five arguments of `std::replace_copy` off the top of my head, and you shouldn't expect anyone to. When values inside the container need to be modified, non-const iterators may be slower than a Qt foreach() loop.

Recommendation:

Encourage the use of <algorithm> when it improves code clarity. Speed not a big problem most of the time, don't make changes which are hard to understand just for a tiny hypothetical speed boost. However, moving to <algorithm> and away from Qt foreach() inside hot paths could prove useful in the future.

enum class

Motivation:

These are a type-safe version of enums, and allows the programmer to

associate several different types of data with an enum, such as a character. This gives stricter type safety, for example, when it might be possible to accidentally convert a variable into a numeric type. For example:

```
enum class Color : char {Red = 'R', Green = 'G', Blue = 'B'};
```

Other benefits of enum classes are that they can be forward-declared, and that the data can be any sort of constexpr. For example, if one had a constexpr function `color_symbol()` that returned the symbol given some color data, the enum class members could be defined like:

```
enum class Color: char {Red = color_symbol({255, 0, 0}) ...};
```

The standard C++ reference does a nice job explaining these features. https://en.cppreference.com/w/cpp/language/enum

Drawbacks:

Virtually none. Very small changes to the code, more type safety, removes the need for some tables of values. The only problem is sometimes this requires fixing code that was unsafe to begin with.

Recommendation:

Use freely.

Local type definitions (i.e. using)

Motivation:

An easier and localized way to use typedefs. Can be at the namespace, class, or function level. Allows you to rewrite a typedef so that the new name occurs on the left hand side of the equals sign, which is easier to read. They allow you to place typedefs closer to where they're used. They are particularly nice inside templates.

Drawbacks:

Very few. These are quite intuitive

Recommendation:

Go for it.

nullptr

Motivation:

The use of nullptr as a default pointer initializer is a very small change in C++11, and mostly an aesthetic one. Technically, there are only a few things it prevents : it cannot be converted to a numeric type like `int x = nullptr;`, and it cannot be used as a class type in a template, so the following is a compiler error:

```
meta_type<class A, class B>;
meta_type<C, nullptr> x;
```

The most important to nullptr is simply that you are tagging your code - ‘’hey: there is a null pointer lurking around here, be careful!’’

Drawbacks:

It takes longer to type nullptr than it takes to type 0, and it’s not so visually pleasing. Converting the existing code base would be very laborious and mess up git history. Tiny benefits.

Recommendation:

We do not use nullptr in Krita. Not in new code, and we don’t refactor old code to use it. Also not Q_NULLPTR.

Deleted, default, override, final

Motivation:

These are keywords used for designing inheritance patterns. They are useful for preventing accidental copy construction.

Drawbacks:

Since Krita does not export libraries, most of the time we won’t need to worry about these. They are limited to solving some pretty specialized problems.

Recommendation:

No reason to hold back from these features if they seem useful. They are well named and fairly self-explanatory, especially for developers with a

Java or C# background. If you apply them correctly, you can prevent other coders from making mistakes when they use your classes. For others, these definitions can be ignored until they cause a compile error, which tell you that you're doing something the wrong way.

unique_ptr/QScopedPointer

Motivation:

Here is a glowing review of unique_ptr [https://www.drdobbs.com/cpp/c11-uniqueptr/240002708]. This is really about a philosophy of C++ memory management, not just a particular smart pointer type. The idea is that whenever you create an object on the heap, you should *always* house it inside a smart pointer. The reason this philosophy is considered new to C++11 is that unique_ptr is the first time they 'got it right' designing a very nice smart pointer class. Most importantly, the class uses negligible overhead. In particular: `sizeof(unique_ptr<T*>) = size_t`, it can be passed as a function argument without copying, and dereferencing is inline.

QScopedPointer is essentially the same thing as unique_ptr, and perhaps it is more idiomatic to use QScopedPointer instead.

Note

It is a useful idiom to store a d-ptr using *QScopedPointer<Private>*, but if you do this you must also declare a destructor in the header file, even if it has an empty implementation in the source file.

"Rule of Zero": more about the C++ design philosophy behind unique_ptr. [https://rmf.io/cxx11/rule-of-zero/]

Drawbacks:

The philosophy mentioned above can be summarized like this: we should state up front what we are going to prohibit programmers from doing. Like the const keyword, unique_ptr puts restrictions on what can be done with the pointer, the main one being, it cannot be copied. Like enforcing

const correctness, this can be annoying to get right throughout a codebase.

One particular limitation is that Qt container classes. For example `QVector<std::unique_ptr>` is invalid, because QVector requires its members can be copied. This makes converting to unique_ptr a bit slow, since `QVector<T *>` will have to be converted to `std_array<unique_ptr<T*>>`. If the owner was being copied before, it will become uncopiable. This can be a good thing, but it can also be extra work.

Moving a unique_ptr requires additional semantics.
[http://www.cplusplus.com/reference/memory/unique_ptr/operator=/]

Recommendation:

Smart pointers are already prevalent in the codebase with the KisSP family, but more use of them should be encouraged. d_ptrs should be wrapped in a QScopedPointer. The rule is: first Krita's shared pointers, then Qt's, do not use the std smart pointers.

Performance-related (rvalues)

Using move constructors and rvalues are very subtle and advanced features, but widely celebrated as successes of C++11. The point of these features is to save on costs of copying memory when passing function arguments. The idea is that if one is OK allowing a function to steal, alter or destroy its argument, then that function can be called slightly faster if the argument is not copied at all, but instead simply performing an ownership transfer. C++ programmers should already be aware that writing performant code where data gets shuffled around sometimes requires opening a can of ampersands. These features will naturally stay confined to the corners of the codebase behind the scenes where they belong, and should be introduced when they are useful.

- Tutorial for rvalue references
  [http://thbecker.net/articles/rvalue_references/section_01.html]
- KDAB: speed up your Qt 5 programs using C++11 [https://www.kdab.com/wp-content/uploads/stories/slides/DD12/mutz-dd-speed-up-your-qt-5-programs-using-c++11.pdf]
- Slide 37 describes lvalue/rvalue types in exact detail

[http://wiki.hsr.ch/PeterSommerlad/files/MeetingCPP2013_SimpleC++.pdf] Also explains the terms “xvalue” and “prvalue” sometimes seen as well.

Move Constructors

Recommendation:

Use whenever it aids performance.

Reference Qualifiers (rvalue references)

Recommendation:

Use whenever it aids performance.

C++11 features mostly for template programming

Krita makes very light use of templates. These features are useful, coming across them in the code base will add complexity for new learners, and have not been necessary so far.

- decltype : this is the most likely of these features to be useful, for example, in KisInputManager.
- static_assert
- variadic templates

Other C++11 features that will not be useful

- Threading support (Relies on C++ threading model; use Qt threading instead)
- shared_ptr and weak_ptr (Relies on C++ threading model; use KisSharedPointer instead)
- New literal types (already have QString/ki18n)
- Extended Unions (already have QVariant)

Developing Features

There’s several stages to making a feature request become reality. The first section of this page goes over a set of common issues with making feature requests and gives hints on how to make a simple feature request into a proper proposal. The rest documents how a feature goes from a proposal to an actual reality.

Step 1: Making a proposal

“vOpenBlackCanvasMischiefPhotoPaintStormToolKaikai has a frobdinger tool! Krita will never amount to a row of beans unless Krita gets a frobdinger tool, too!”

The cool thing about open source is that you can add features yourself, and even if you cannot code, you talk directly with the developers about the features you need in your workflow. Try that with closed-source proprietary software! But, often, the communication goes awry, leaving both users with bright ideas and developers with itchy coding fingers unsatisfied.

This post is all about how to work, first together with other artists, then with developers to create good feature requests, feature requests that are good enough that they can end up being implemented.

For us as developers it’s sometimes really difficult to read feature requests, and we have a really big to-do list (600+ items at the time of writing, excluding our own dreams and desires). So, having a well written feature proposal is very helpful for us and will lodge the idea better into our conscious. Conversely, a demand for a frobdinger tool because another application has it, so Krita must have it, too, is likely not to get far.

Writing proposals is a bit of an art form in itself, and pretty difficult to do right! Asking for a copy of a feature in another application is almost always wrong because it doesn’t tell us the most important thing:

What we primarily need like to know is HOW you intend to use the feature. This is the most important part. All Krita features are carefully considered in terms of the workflow they affect, and we will not start working on any feature unless we know why it is useful and how it is exactly used. Even better, once we know how it's used, we as developers can start thinking about what else we can do to make the workflow even more fluid!

Good examples of this approach can be found in the pop-up palette using tags, the layer docker redesign of 3.0, the onion skin docker, the time-line dockers and the assistants.

Feature requests should start on the forum, so other artists can chime in. What we want is that a consensus about workflow, about use-cases emerges, something our UX people can then try to grok and create a design for. Once the design emerges, we'll try an implementation, and that needs testing.

For your forum post about the feature you have in mind, check this list:

- It is worth investigating first if the feature in question has similar functionality in Krita that might need to be extended to solve the problem. We in fact kind of expect that you have used Krita for a while before making feature requests. Check the manual first!
- If your English is not very good or you have difficulty finding the right words, make pictures. If you need a drawing program, I heard Krita is pretty good.
- In fact, mock-ups are super useful! And why wouldn't you make them? Krita is a drawing program made for artists, and a lot of us developers are artists ourselves. Furthermore, this gets past that nasty problem called 'communication problems'.
- Focus on the workflow. You need to prepare a certain illustration, comic, matte painting, you would be (much) more productive if you could just do — whatever. Tell us about your problem and be open to suggestions about alternatives. A feature request should be an exploration of possibilities, not a final demand!
- The longer your request, the more formatting is appreciated. Some of us are pretty good at reading long incomprehensible texts, but not all of us. Keep to the ABC of clarity, brevity, accuracy. If you format and organize your request we'll read it much faster and will be able to spent

more time on giving feedback on the exact content. This also helps other users to understand you and give detailed feedback! The final proposal can even be a multi-page PDF.

- We prefer it if you read and reply to other people's requests than to start from scratch. For animation we've had the request for importing frames, instancing frames, adding audio support, from tons of different people, sometimes even in the same thread. We'd rather you reply to someone else's post (you can even reply to old posts) than to list it amongst other newer requests, as it makes it very difficult to tell those other requests apart, and it turns us into bookkeepers when we could have been programming.

Keep in mind that the Krita development team is insanely overloaded. We're not a big company, we're a small group of mostly volunteers who are spending way too much of our spare time on Krita already. You want time from us: it's your job to make life as easy as possible for us!

So we come to: **Things That Will Not Help.**

There's certain things that people do to make their feature request sound important but are, in fact, really unhelpful and even somewhat rude:

"Application X has this, so Krita must have it, too".

See above. Extra malus points for using the words "industry standard", double so if it refers to an Adobe file format.

We honestly don't care if application X has feature Y, especially as long as you do not specify how it's used.

Now, instead of thinking 'what can we do to make the best solution for this problem', it gets replaced with 'oh god, now I have to find a copy of application X, and then test it for a whole night to figure out every single feature… I have no time for this'.

We do realize that for many people it's hard to think in terms of workflow instead of "I used to use this in ImagePainterDoublePlusPro with the humdinger tool, so I need a humdinger tool in krita" — but it's your responsibility when you are thinking about a feature request to go

beyond that level and make a good case: we cannot play guessing games!

“Professionals in the industry use this”.

Which professionals? What industry? We cater to digital illustrators, matte painters, comic book artists, texture artists, animators… These guys don’t share an industry. This one is peculiar because it is often applied to features that professionals never actually use. There might be hundreds of tutorials for a certain feature, and it still isn’t actually used in people’s daily work.

“People need this.”

For the exact same reason as above. Why do they need it, and who are these ‘people’? And what is it, exactly, what they need?

“Krita will never be taken seriously if it doesn’t have a glingangiation filter.”

Weeell, Krita is quite a serious thing, used by hundreds of thousands of people, so whenever this sentence shows up in a feature request, we feel it might be a bit of emotional blackmail: it tries to to get us upset enough to work on it. Think about how that must feel.

“This should be easy to implement.”

Well, the code is open and we have excellent build guides, why doesn’t the feature request come with a patch then? The issue with this is very likely it is not actually all that easy. Telling us how to implement a feature based on a guess about Krita’s architecture, instead of telling us the problem the feature is meant to solve makes life really hard!

A good example of this is the idea that because Krita has an OpenGL accelerated canvas, it is easy to have the filters be done on the GPU. It isn’t: The GPU accelerated canvas is currently pretty one-way, and filters would be a two-way process. Getting that two way process right is very difficult and also the difference between GPU filters being faster than regular filters or them being unusable. And that problem is only the tip of the iceberg.

Some other things to keep in mind:

- It is actually possible to get your needed features into Krita outside of the Kickstarter sprints by funding it directly via the Krita foundation, you can mail the official email linked on krita.org for that.
- It’s also actually possible to start hacking on Krita and make patches. You don’t need permission or anything!
- Sometimes developers have already had the feature in question on their radar for a very long time. Their thinking might already be quite advanced on the topic and then they might say things like ‘we first need to get this done’, or an incomprehensible technical paragraph. This is a developer being in deep thought while they write. You can just ask for clarification if the feedback contains too much technobabble…
- Did we mention we’re overloaded already? It can easily be a year or two, three before we can get down to a feature. But that’s sort of fine, because the process from idea to design should take months to a year as well!

To summarize: a good feature request:

- starts with the need to streamline a certain workflow, not with the need for a copy of a feature in another application
- has been discussed on the forums with other artists
- is illustrated with mock-ups and example
- gets discussed with UX people
- and is finally prepared as a proposal
- and then it’s time to find time to implement it!
- and then you need to test the result.

Step 2: Triaging the proposal

This is strictly a developer task. What is done is that we identify how much work a proposal would need to be implemented. Since 2016 we use these groups to categorize wishbugs so we can plan them into a current release or select them for a fundraiser.

To fulfill this step, we need to have a full list which consolidated the ideas and requirements. A good feature request from step one will have these lined out.

WISHGROUP: Pie-in-the-sky
    not going to happen, but it would be really cool.

WISHGROUP: Big Projects
    needs more definition, maybe two, three months of work.

WISHGROUP: Stretchgoal
    up to a couple of weeks or a month of work.

WISHGROUP: Larger Usability Fixes
    maybe a week or two weeks of work.

WISHGROUP: Small Usability Fixes
    half a day or a day of work.

WISHGROUP: Out of scope
    too far from our current core goals to implement.

WISHGROUP: Needs proposal and design
    needs discussion among artists to define scope first. A good proposal doesn’t need this.

Step 3: Discussing in irc/phab

Again, strictly a developer task. While nothing stops an adventurous programmer from just going in and implementing something, it helps to go to the #krita irc on freenode and tell us you’re working on it. Not because you need permission(Krita is open source after all), but we do want to be able to help you in your endeavours. Such help can include technical help, like where things are in the code, but also interface design help.

Some features, such as new frame types for animation, or multithreading on some part or the other also needs careful discussion so we know what is going to need changes.

Eventually, a phabricator task will be made to track the issue as well as including mockups. Branch progress is also discussed during the weekly meeting in the irc.

Step 4: Work in a feature branch

New feature branches are called ‘name/number-shortdescription’. Examples: “rempt/T379-resource-management”, “kazakov/hdr-support”, “wolthera/edgedetectionfilter”, “jounip/T8764-clone-frames”.

Originally this was lastname only, but some users have an endlessly long last name while others prefer using their kde identity name. The main purpose is to identify who is responsible for the work in the branch.

Work in a feature branch continues till all major elements are done. A review request is done over the whole branch. Sometimes, for UI purposes, people check out the branch to test it.

When the review is accepted, the branch is merged into master for further testing. When such a branch is merged, a mail needs to be sent to kimageshop@kde.org to notify everyone about this, you can do this automatically by adding ‘CCMAIL:kimageshop@kde.org’ to your merge commit.

As Krita’s nightlies are based on master that means a binary will be compiled for the master branch with the new feature in at most 24 hours.

Step 5: Documentation and demonstration

When a feature hits the master branch, an entry will be written for the draft branch of this very manual. In particular a reference manual entry will be written to ensure some documentation, some bigger features that interact with one another might also receive a tutorial.

The people who programmed or designed the feature are encouraged to help with this documentation process(as they know it best), but it is not mandatory. What is appreciated is that the issue or task is assigned to the manual team.

Similarly, demonstration videos or images are welcome, as they will be used for the release notes. The release notes for the next big version are available

here [https://krita.org/en/krita-4-2-release-notes/], come help us with making the page look good!

Finally, upon release a stable branch is created for the master branch (often named Krita/versionnumber), and a release is made with the new feature.

Optimizing tips and tools for Krita

Hot Spots

- thumbnails are recalculated a lot
- the histogram docker calculates even when hidden
- brush outline seems slow
- the calculation of the mask for the autobrush is very slow and doesn't cache anything
- caching a whole row or column of tiles in the h/v line iterators should speed up things a lot
- tile engine 1 has the BKL; tile engine 2 cannot swap yet and isn't optimized yet
- projection recomposition doesn't take the visible area into account
- pigment preloads all profiles (startup hit)
- gradients are calculated on load, instead of being associated with a PNG preview image that is cheap to load

Tools

Valgrind

Tips

You can tell callgrind to focus only on the part of the code you want to optimize. This results in cleaner data. For example, you may want to only monitor the performance when drawing a stroke. Unless the thing you're trying to optimize is the program startup, you can tell valgrind to run with the logging, or instrumentation, turned off at start:

```
valgrind  --tool=callgrind --instr-atstart=no krita
```

Instrumentation can then be activated and deactivated with callgrind_control. To begin performance monitoring:

```
callgrind_control -i on
```

And then to end it:

```
callgrind_control -i off
```

I usually write a few aliases in my .bashrc (or .zshrc):

```
alias callgrind="valgrind --tool=callgrind --instr-atstart=no"
alias callgrind-on="callgrind_control --instr=on"
alias callgrind-off="callgrind_control --instr=off"
```

Sysprof

mutrace

mutrace [http://0pointer.de/blog/projects/mutrace.html] is a tool that count how much time is spend waiting for a mutex to unlock.

Easy optimization

As soon as you see slow code, try to have a look at the code to see if we aren't creating a lot of unnecesserary objects, 90% of the time slow code is caused by this (the remain 10% are often caused by a lot of access to the tilesmanager, like with random accessor)

For instance:

Avoid:

```
for(whatever)
{
    QColor c;
    ...
}
```

Do:

```
QColor c;
for(whatever)
```

```
{

}
```

It might seems insignificant, but really it's not, on a loop of a milion of iterations, this is expensive as hell.

An other example:

Avoid:

```
for(y = 0 to height)
{
    KisHLineIterator it = dev->createHLineIterator(0, y, widt
    for(whatever)
    {
        ...
    }
}
```

Do:

```
KisHLineIterator it = dev->createHLineIterator(0, 0, width);
for(y = 0 to height)
{
    for(whatever)
    {
        ...
    }
    it.nextRow(); // or nextCol() if you are using a VLine it
}
```

Vector instructions

Krita takes heavy advantage of the Vc [https://github.com/VcDevel/Vc] library to speed up its brush strokes with CPU vector instructions. If you are planning to work with that library, it is worth reading through its documentation.

There are more general introductions to what vector instructions are for, and how they work here.

- Reference about MMX on Intel's website [http://developer.intel.com/design/archives/processors/mmx/].
- Fundamentals of Media Processor Designs [http://www.cise.ufl.edu/~peir/cda6159/media12.pdf]: introduction to the use of MMX/SSE instructions.
- Software Optimization Guide for AMD64 [http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/25112.PDF].
- STL like programming but using MMX/SSE{1,2,3} when available [http://www.pixelglow.com/macstl/].

Profile guided optimization

Profile guided optimization is something else though. It is a special way of compiling and linking, that the compiler and linker use profiling information to know how best to optimize the code. So code that is used a lot is compiled with -O3 (the most optimizations), while code that is not used a lot gets -Os (to take less space), and so forth. This is a very useful technique that was not available on Linux until last year, and the news today is that Firefox now builds properly with it and there is a nice noticeable speed improvement for Linux users.

source:
https://linux.slashdot.org/comments.pl?sid=2117150&cid=35987784

wikipedia:
https://en.wikipedia.org/wiki/Profile-guided_optimization

```
g++ -O3 -march=native -pg -fprofile-generate ...
// run my program's benchmark
g++ -O3 -march=native -fprofile-use ...
```

Links

- Design for Performance [https://es.scribd.com/document/53483851/Design-for-Performance]: great read about performance optimization (aimed at game developers, but many tricks apply for Krita).
- TCMalloc [http://goog-perftools.sourceforge.net/doc/tcmalloc.html]: a malloc

replacement which make faster allocation of objects by caching some reserved part of the memory.

- Optmizing CPP [http://www.agner.org/optimize/optimizing_cpp.pdf]: extensive manual on writing optimized code.

Google Summer of Code

Every summer Google puts on a program that helps university developers get involved with the open source community. This is known as Google Summer of Code (GSoC). Krita has always participated in GSoC through the KDE community. For more information you can take a look at the gsoc website [https://summerofcode.withgoogle.com/].

How to participate as a student?

Submitting a resumé or CV isn't how this program works. For you to be picked, you need to be involved with the Krita community early and show you have some capacity to do programming. The summer program involves focusing on one project. You will have a mentor assigned to help learn the ropes. Here [https://community.kde.org/GSoC/2020/Ideas#Krita] are some potential project ideas. If there is another project that you want to see, you can also propose your own. Use these guidelines [https://community.kde.org/GSoC#Student_proposal_guidelines] to help formulate ideas. We've mentored around half a dozen students every year year since GSoC started. Many students enjoyed their work and continue to be involved; perhaps your mentor will be a past GSoC student.

What is expected from you before participating

- A basic understanding of *git*, which would include pulling and pushing code, create branches and rebase commits.
- A fair understanding of *c++* and its ecosystem.
- Ecosystem here would mean tools like cmake, make, gcc, gdb and valgrind.
- Knowing how to work with Qt is not mandatory but would be helpful.
- You should be able to navigate the codebase, using an IDE like Qt Creator is preferred.

Before starting to work on a Proposal

- Build Krita from source.
- Try fixing a bug or implement a wish. If you are unable to find something to work on, don't hesitate to ask us. Someone would surely help you.
- If you are picking something from the list ask whether someone has already picked that idea or not.
- If you are proposing an idea of your own, please do discuss about that with us. We need to see whether the project is viable or not before starting out.
- Whatever you are onto, please communicate before proceeding.

How to create a proper proposal?

- Divide your proposal into separate sections as directed by the KDE student proposal guidelines [https://community.kde.org/GSoC#Student_proposal_guidelines].
- The most important parts are the Goals, Implementation and the Timeline, pay attention to them.
- Goals are the requirements of the project, the features introduced and the bug fixed from the perspective of an user.
- Implementation, as the name says should tell us how are you going to implement the requirements. Put the classes or methods you are going to use, mockups of the UIs here.
- TimeLine would indicate how much time would you devote behind each feature you would be working on. Beware this would later become the yardstick for evaluations.

Tips:

- Start as early as possible, that way you could get most feedback.
- Don't have more than you can chew, it is far better to put what you think is achievable inside 3 months.
- Allocate a bit buffer time, things could go wrong, better to be prepared.
- Don't forget to write documentation, the features should be well

documented in the manual.
- Wherever you see, you could add tests, please do add that, most of the times it is better to write the tests first.

How do I ensure that I get selected?

- Communicate, GSoC is half communication.
- Show that you can code independently by fixing bugs or implementing wishes.
- Know whom to ask for help and when to ask, neither everyone knows everything nor everyone is available all the time.
- Plus points if you can propose an unique idea of your own.
- Even if you do all of them we can't exactly ensure that you will be selected, it all depends on how many slots Google allocates for KDE.

Done with the proposal, what should I do now?

- Try fixing some more bugs or implement a wishlist item.
- If anything is missing from the manual, do make a Merge Request to it.
- Help other students with their proposal, GSoC is not a competetion.

I am selected what now?

- Create a Phabricator Task with the requirements and implementation details of your project.
- If you don't have a developer account already, request for one.
- Add your blog to KDE Planet.
- Create a new branch which would refer to the Phabricator Task with a name like, TXXX-<task_name>.
- Create a status report page at *https://community.kde.org/GSoC/<year>/StatusReports* where you would be listing all the commits and blog posts. This would be sent to Google as the work product.
- Start hacking.

Where to ask for help?

- The best place would be our IRC channel which would be #krita on freenode.
- The second best place would be our mailing list [https://mail.kde.org/mailman/listinfo/kimageshop].
- Keep in mind that the team is spread over 5 continents and most of the time, weekends are awkwardly quiet.
- Any kind of private communication is discouraged, whatever you need to ask, ask in the public channels, unless required.

Advanced Merge Request Guide

Since April 2019, we're using Gitlab to review merge requests and patches to the code. Check Forking on Gitlab on how to start with making a merge request.

When to make a merge request

There's three times you need to make merge requests.

1. When you do not have commit access.
2. When the change you are making is huge, like with feature development, large refactors, complex bugfixes. For these we do not fork, but instead make a branch in the main repository in the following format: `name/number-shortdescription`. Check Developing Features for more information.
3. When you are not sure about whether what you did is correct. It is common within the Krita community that even developers with commit access will have a weaning period in which they still make merge requests for each change as they're getting comfortable with the codebase. It is not mandatory to do so at this point, but requests are the best way for us to help one another with writing good code.

Checklist for review

Here's a quick checklist that is gone over when reviewing patches. While some requests require specialist knowledge, these items are so universal that anyone who knows how to check them can do so and comment on them. Feel free to do this yourself on open requests [https://invent.kde.org/graphics/krita.git/merge_requests?scope=all&utf8=%E2%9C%93&state=opened], we welcome it!

Also check out the manual for reviewing merge requests in Gitlab [https://invent.kde.org/help/user/project/merge_requests/index.md].

Does the code build

Most important check. Apply the patch locally and build it. If it doesn't build, the patch will not be accepted at all.

Does it not crash?

Basically, build the patch, run Krita, and check if the functions associated doesn't crash. If it does, make a backtrace and post it in the review.

Does it leak memory?

Does the patch break tests?

CPP features used.

Is the usage of CPP in accordance with HACKING and the Modern C++ usage guidelines for the Krita codebase guidelines? So for example, is auto not used outside of the single case we accept it?

Is the code in conformance with KDE/Krita style?

Check the HACKING file for directions.

Are the commit messages sensible?

There's several guides for this, but in general, try to make sure that the commit messages actually explain what you did.

- https://github.com/RomuloOliveira/commit-messages-guide

Does the patch make sense.

This is in the category of specialist knowledge, but you can apply some common sense here. If a patch for a filter also adjusts the resource management, you can ask yourself why this would be necessary. Furthermore, does the patch actually fix the thing it says it is fixing? These are not easy checks to make, but important things to consider when looking at the patch.

Requests that need changes to them need to be labeled with `needs-changes`. Requests that are accepted should be labeled `accepted`. This is to ensure we can figure out which requests are in need of review. Requests that need to be reviewed need to lack both labels.

Python Developer Tools

For working with Krita's Python code, there are a couple of tools for running unit tests and code checks.

Contents

Setup

To set up a Python environment for running code checks, unit tests etc, it is recommended to use a Python virtual environment for installing the needed Python packages. For this, install *virtualenwrapper* from your package manager or follow the installation instructions [https://virtualenvwrapper.readthedocs.io/en/latest/install.html/]. It is also possible to install the needed Python packages directly into your system.

To create a virtual environment for Python 3 with *virtualenwrapper*, run:

```
mkvirtualenv krita -p /usr/bin/python3
```

This will create a virtual environment called *krita* and activate it. You will see that your command line now starts with *(krita)* to indicate the active virtual environment.

Now change into your Krita git repository install the needed dependencies into the environment:

```
pip install --upgrade -r dev-tools/python/dev-requirements.txt
```

You can rerun this command to update the packages should the version numbers in the requirement file get updated.

To get out of the virtual environment, type:

```
deactivate
```

And to get back into the virtual environment, type:

```
workon krita
```

Code Checks

The code checker follows Python's official style guide, PEP8 [https://www.python.org/dev/peps/pep-0008/].

To run codechecks on all Python files, type:

```
flake8 .
```

Or limit to a specific directory or file:

```
flake8 plugins/python/plugin_importer/
```

Unit Tests

To run all Python unit tests, type:

```
pytest .
```

Or limit to a specific directory, file, or test:

```
pytest plugins/python/plugin_importer/tests/test_plugin_importer.
```

See Pytest's Getting Started [https://docs.pytest.org/en/latest/getting-started.html] documentation for more information about *pytest* in general.

Unit tests for Python plugins should reside in a *tests* folder inside the plugin's

folder. See the *plugin_importer* plugin for example unit tests. Note that in order to be able to import Krita's Python plugins outside of Krita in the unit test setup, there is a mock *krita* module that will return mock objects for any attribute names so that importing *krita* and registering plugins etc. become no-ops. Thus, it's only possible to test code units that are independent of Krita-related functions. Another caveat is that the mock krita module doesn't work with wildcard imports (*from krita import* *), but those should be avoided anyway.

Introduction to Quality Assurance

We are users and developers systematically working on increasing quality of Krita and the process of it's development. We help sustain the self-auditing culture of Krita's community.

We

- Methodically assess functionality, usability and security.
- Hunt for bugs and cater for bugs already captured.
- Aid in quality management. Create tools to make developer's life easier.

How To Help?

The quality assurance field is really broad and diverse and we are always looking for people of all skills and talents. Below you will find a list of opportunities to help, so you can dive right into it. Also, don't forget to visit us on the IRC, we will be happy to meet you.

Bug Triaging

There is a great amount of incoming bug reports, more than the core team can handle. We are looking for volunteers who would go through the bug tracker and handle the reported bugs. This includes:

- Determining if a bug is really a bug or a new feature request
- Confirming bugs by reproducing
- Guiding reporters to provide all the information needed to fix the bug
- (Optional) Providing logs, backtraces, core dumps

Get Started

- Reporting Bugs provides general information about bug reports and guidance for their creation

- Krita-specific guide to Triaging Bugs

See also

- Guide to Developing Features
- Hints for user support also apply here: Introduction to User Support
- Docs for gathering logs, backtraces, etc.
    - https://docs.krita.org/en/KritaFAQ.html?highlight=mingw#how-can-i-produce-a-backtrace-on-windows
    - https://docs.krita.org/en/reference_manual/dockers/log_viewer.htm

Beta Testing

To validate an upcoming stable version will work as expected, there is the beta version. You can help by dowloading the beta, trying it out and sharing your feedback. Every beta comes with a survey, which will ask for some basic information about your setup (all anonymized, of course) and guide you through testing latest features and bug fixes. You can find link to the survey on Krita's welcome page.

To know when there is a new beta, watch out for the news on the welcome page, or in the News section on Krita website.

For more information about the process refer to the Testing Strategy.

Test Engineering

The test suite is the safety net enabling the community to fearlessly move forward. We have a comprehensive testing strategy to help us find bugs early in the process and deliver the best user experience possible. But without people, the strategy is just a bunch of words. There are many ways you can help, for both technical and less technical people.

- If you like to experiment and try new things, consider exploratory testing. No coding skill required.
- Hone your analytical skills by designing end-to-end tests.

- Try your hand at unit testing. Design and implement the low level tests for both backend and UI code.

Check out Testing Strategy for more information.

Enhancement Projects

There is plenty of projects from small to big, some include writing and organizing, some require coding. We currently register following projects: https://phabricator.kde.org/T11218. Does something catch your eye?

Do you have something else in mind?

This list is not definite. We are always open to new ideas and approaches. Please, join us on the IRC (The Krita Community) to discuss the possibilities.

Making a release

Contents

Before the release

1. Coordinate with #kde-promo

2. Notify translators of string freeze!

3. Verify that the release notes page is done, like https://krita.org/en/krita-4-2-release-notes/

4. Verify that all 8 (eight!) dependency builds are up to date. Remember that these builds are built from **master**, not from the stable branch.

    - https://binary-factory.kde.org/job/Krita_Android_arm64-v8a_Dependency_Build/

- https://binary-factory.kde.org/job/Krita_Android_armeabi-v7a_Dependency_Build/
- https://binary-factory.kde.org/job/Krita_Android_x86_64_Dependency_Build/
- https://binary-factory.kde.org/job/Krita_Android_x86_Dependency_Build/
- https://binary-factory.kde.org/job/Krita_Nightly_Appimage_Dependency_Build
- https://binary-factory.kde.org/job/Krita_Nightly_MacOS_Dependency_Build/
- https://binary-factory.kde.org/job/Krita_Nightly_Windows_Dependency_Build/
- https://binary-factory.kde.org/job/Krita_Release_Windows32_Dependency_Bui

Compare the build date and included commits to the commit in 3rdparty directory in master:

```
git fetch origin && git log origin/master 3rdparty
```

Update version in source code

1. !! REMOVE THE SURVEY LINK !! (or, if this is a beta, make a survey and update the survey link)
2. update the version of krita.xmlgui
3. update the CMakeLists.txt version
4. update the snapcraft.yaml file
5. update the appstream screenshots
6. update org.kde.krita.appdata.xml ‘s release tag
7. update create_tarball’s config.ini
8. update download_release_artifacts.sh

9. update Android version (keep in mind that *all* Krita releases on Android are marked as Beta at the moment):
    - packaging/android/apk/AndroidManifest.xml
    - packaging/android/apk/build.gradle
10. When releasing beta-version double-check that you updated to “beta1”, not just plain “beta”. Only “alpha” versions can be made without a number, because they are built nightly.

Update versions in the stable branch to avoid XMLGUI conflicts

1. stable branch is always marked as “alpha” (without a number in the end)
2. update the version of krita.xmlgui; it should be *$(($VERSION_IN_RELEASE_BRANCH + 1))*
3. update the CMakeLists.txt version
4. update org.kde.krita.appdata.xml ‘s release tag
5. packaging/android/apk/AndroidManifest.xml

Create the tarball

Create and push the tag

1. Set the tag:

```
git tag -a v4.2.9-beta1 -m "Krita 4.2.9 Beta1"
```

2. Push the tag:

```
git push origin refs/tags/v4.2.9-beta1:refs/tags/v4.2.9-b
```

3. If you need to change the tag position (not recommended):

```
# remove the previous tag

git push origin :refs/tags/v4.2.9-beta1
```

```
# make a new tag locally
git tag -a v4.2.9-beta1 -m "Krita 4.2.9 Beta1"

# push the new tag
git push origin refs/tags/v4.2.9-beta1:refs/tags/v4.2.9-b

# all Krita developers now have to refetch tags to
# get the updated tag position
git fetch origin --tags
```

Create the tarball

1. Go into the *./packaging/* folder, check that the version in 'config.ini' file reflects the right tag and version number. It should look like that:

```
[krita]
gitModule   = yes
gitTag      = v4.2.9
category    = graphics
mainmodule  = branches/stable
l10nmodule  = krita
version     = 4.2.9
translations= yes
docs        = no
kde_release = no
```

2. Create the tarball:

```
./create_tarball_kf5.rb -n -a krita
```

3. Check that created archive has 'po' folder and it actually has translations

4. Sign both tarballs:

```
gpg --output krita-4.2.9-beta1.tar.gz.sig --detach-sign k
gpg --output krita-4.2.9-beta1.tar.xz.sig --detach-sign k
```

5. Upload tarballs to files.kde.org, where builders can fpick them up:

    - https://files.kde.org/krita/.release/$version/krita-$version.tar.gz

- https://files.kde.org/krita/.release/$version/krita-$version.tar.xz
- https://files.kde.org/krita/.release/$version/krita-$version.tar.gz.sig
- https://files.kde.org/krita/.release/$version/krita-$version.tar.xz.sig

Make Windows, Linux, OSX and Android packages

8. Request four release builds on binary-factory.kde.org, after starting each build,go to “Console Output” section, click on “Input Requested” and choose a tarball version to build.

    - https://binary-factory.kde.org/job/Krita_Release_Windows32_Build/
    - https://binary-factory.kde.org/job/Krita_Release_Windows64_Build/
    - https://binary-factory.kde.org/job/Krita_Release_Appimage_Build/
    - https://binary-factory.kde.org/job/Krita_Release_MacOS_Build/
    - https://binary-factory.kde.org/job/Krita_Release_Android_arm64-v8a_Build/
    - https://binary-factory.kde.org/job/Krita_Release_Android_armeabi-v7a_Build/
    - https://binary-factory.kde.org/job/Krita_Release_Android_x86_64_Build/
    - https://binary-factory.kde.org/job/Krita_Release_Android_x86_Build/

9. Download all built artifacts using *download_release_artifacts.sh* script. Open the script and modify *KRITA_VERSION* variable to correspond to the version string.

10. For each build check:

- Krita starts
- Localization works
- Python plugins are available
- Basic painting and most recently fixed bugs are fixed

11. Sign both AppImages:

```
gpg --detach-sign --output krita-4.2.9-beta-x86_64.appimage.s
gpg --detach-sign --output gmic_krita_qt-x86_64.appimage.sig
```

12. Sign four Android packages (or send them to Boud for signing)

    - krita-arm64-4.2.9-beta1-unsigned.apk
    - krita-arm32-4.2.9-beta1-unsigned.apk
    - krita-x86-4.2.9-beta1-unsigned.apk
    - krita-x86_64-4.2.9-beta1-unsigned.apk

    After signing, remove “-unsigned” suffix, so the signed packages would look like that:

    - krita-arm64-4.2.9-beta1.apk
    - krita-arm32-4.2.9-beta1.apk
    - krita-x86-4.2.9-beta1.apk
    - krita-x86_64-4.2.9-beta1.apk

13. Now you should have 19(!) files in your release folder

14. Generate an md5sum.txt file for all of them:

```
md5sum ./* > md5sum.txt
```

15. Upload 20(!) files to download.kde.org (or ask sysadmins to do that using this manual ftp://upload.kde.org/README):

    - krita-4.2.9-beta1.tar.gz
    - krita-4.2.9-beta1.tar.gz.sig
    - krita-4.2.9-beta1.tar.xz
    - krita-4.2.9-beta1.tar.xz.sig

- gmic_krita_qt-x86_64.appimage
- gmic_krita_qt-x86_64.appimage.sig
- krita-4.2.9-beta1-x86_64.appimage
- krita-4.2.9-beta1-x86_64.appimage.sig
- krita-x64-4.2.9-beta1-dbg.zip
- krita-x64-4.2.9-beta1-setup.exe
- krita-x64-4.2.9-beta1.zip
- krita-x86-4.2.9-beta1-dbg.zip
- krita-x86-4.2.9-beta1-setup.exe
- krita-x86-4.2.9-beta1.zip
- krita-4.2.9-beta1.dmg
- krita-arm64-4.2.9-beta1.apk
- krita-arm32-4.2.9-beta1.apk
- krita-x86-4.2.9-beta1.apk
- krita-x86_64-4.2.9-beta1.apk
- md5sum.txt

16. Template ticket for sysadmins:

```
Hi, sysadmins!

Could you please do the final steps for publishing Krita rele

There are two tasks:

1) Upload release artifacts (20 files) to download.kde.org:

    * Source link: https://files.kde.org/krita/release-4.2.9-
    * Destination link: https://download.kde.org/unstable/kri
    * There should be 16 files including `md5sum.txt`

2) Add `Krita 4.2.9 Beta1` bugzilla version
```

17. Now the folder on download.kde.org should have 20(!) files. Check if you missed something (and you surely did! :)).

Release coordination

1. Mail KDE release coordination <release-team@kde.org>

2. Send release notes for future Krita versions to news@publisher.ch
3. Create bugzilla version: https://bugs.kde.org/editversions.cgi?product=krita Or file a sysadmin ticket for that.
4. [only for a major release] Warn kde sysadmins that we're going to release and that krita.org is going to take load. Just file a ticket on phabricator.

PR and Communications

Pre-release

1. Update Kiki page
2. Update press pack and page
3. Verify if manual pages are updated, if not annoy @woltherav and add undocumented features to Krita: Manual
4. Notify people that they can start making release demonstrations.

Release

1. Update download page
2. Publish the announcement and release notes
3. Add release links to Release History section of the site: https://krita.org/en/about/krita-releases-overview/

Post-release

- tumblr (wolthera)
- BlenderArtists (wolthera)
- deviantart (wolthera)
- VK (dmitry)
- blendernation (boud)
- twitter (boud)
- facebook (boud)
- 3dpro (boud)
- reddit (raghukamath)

Notes

Additional info can be found here: https://phabricator.kde.org/T10762

Reporting Bugs

Krita is, together with many other projects, part of the KDE community. Therefore, bugs for Krita are tracked in KDE's bug tracker: KDE's bug tracker [https://bugs.kde.org]. The bug tracker is a tool for Krita's developers to help them manage bugs in the software, prioritize them and plan fixes. It is not a place to get user support!

The bug tracker contains two kinds of reports: bugs and wishes. Bugs are errors in Krita's code that interrupt using Krita. Wishes are feature requests: the reporter thinks some functionality is missing or would be cool to have.

Do not just create a feature request in the bug tracker: follow Feature Requests [https://krita.org/en/item/ways-to-help-krita-work-on-feature-requests/] to learn how to create a good feature request.

This guide will help you create a good bug report. If you take the time to create a good bug report, you have a much better chance of getting a developer to work on the issue. If there is not enough information to work with, or if the bug report is unreadable, a developer will not be able to understand and fix the issue.

Contents

Only Report Bugs

A bug is a problem in Krita's code.

- If you have problems with your drawing tablet, for instance no support for pressure, then that is unlikely to be a problem in Krita's code: it is almost certain to be a problem with your setup or the driver for your tablet.
- If you've lost the toolbox, that's not a bug.
- If you have deleted your work, that is not a bug.
- If Krita works differently from another application, that's not a bug.
- If Krita works differently than you expected, that's not a bug.
- If Krita is slower than you expected, that's not a bug.
- If Krita crashes, that's a bug.
- If a file you save comes out garbled, that's a bug.
- If Krita stops working, that's a bug.
- If Krita stops displaying correctly, that's a bug.

Check the FAQ

If you've got a problem with Krita, first check the FAQ [https://docs.krita.org/en/KritaFAQ.html]. See whether your problem is mentioned there. If it is, apply the solution. If that doesn't work for you, do not report a bug. Ask for help on Krita's Forum [https://forums.kde.org].

Ask on Krita's Forum or IRC Chat Channel

If uncertain, ask on Krita's IRC chat channel [https://krita.org/en/irc/] or the Krita Forum [https://forum.kde.org/krita].

Krita's chat channel is maintained on webchat.freenode.net. Developers and users hang out to discuss Krita's development and help people who have questions.

Important

Most Krita developers live in Europe, and the channel is very quiet when it's night in Europe. You also have to be patient: it may take some time for people to notice your question even if they are awake.

Also …

Krita does not have a paid support staff. You will chat with volunteers, users and developers. It is not a help desk.

But you can still ask your question, and the people in the channel are a friendly and helpful lot.

Use the Latest Version of Krita

Check Krita's website to see whether you are using the latest version of Krita. There are two "latest" versions:

- Latest stable: check the Download page [https://krita.org/download/]. Always try to reproduce your bug with this version.
- Stable and Unstable Nightly builds. The stable nightly build is built from the last release plus all bug fixes done since the last release. This is called **Krita Plus*** The unstable nightly build contains new features and is straight from the development branch of Krita. This is called **Krita Next**. You can download these builds from the Download page [https://krita.org/download/].

Check The Bug Tracker for Duplicates

This can be tricky: many bug reports are very unclear, have misleading subjects or are assigned to the wrong component. The Krita team tries to triage incoming bugs, fixing the subject, the component and asking for more information in case the bug is not clear.

But please do try to check whether a problem has already been reported. If it is, please add your report as a comment to that bug ticket.

Be Complete and Be Completely Clear

Give all information. That means that you should give information about your operating system, hardware, the version of Krita you're using and, of course about the problem.

- Operating system: fill in the requisite field in the bug tracker's form
- Version: fill in the requisite field in the bug tracker's form
- Hardware information: copy the information from the *Help ▸ Show System Information for Bug Reports* or Bug Reports window into your report. Note how many displays you have.
- If you are using a drawing tablet, tell us the brand and type.
- Tell us what kind of image you were working on: the size, the resolution, the color model and channel depth.
- If you are reporting a crash, attach a crash log. Follow this link [https://docs.krita.org/en/reference_manual/dr_minw_debugger.html#dr-minw] to learn how to get a crash log on Windows. On Linux, follow your distribution's instructions to install debug symbols if you have installed Krita from a distribution package. It is not possible to create a useful crash log with Linux appimages.
- Attach the contents of the *Help ▸ Show System Information for Bug Reports* dialog to the bug report.

The problem needs to be clearly stated: - what happened, - what had you expected to happen instead, - how the problem can be reproduced.

Give a concise and short description, then enumerate the steps needed to reproduce the problem. If you cannot reproduce the problem, and it isn't a crash, think twice before making the report: the developers likely cannot reproduce it either.

If at all possible, attach your original Krita file (the one that ends in `.kra`) to the bug report, or if it's too big, add a link for download. If you do that, make sure the file will be there for **years** to come: do not remove it.

If you think it would be useful, you can also attach or link to a video. Note that the Krita developers and bug triagers are extremely busy, and that it takes

less time to read a good description and a set of steps to reproduce than it takes to watch a video for clues for what is going on.

When making a video or a screenshot, include the whole Krita window, including the titlebar and the statusbar.

You're Not Done After You Have Filed the Report

After you have filed your bug, mail will be sent out to all Krita developers and bug triagers. You do not have to go to the chat channel and tell us you created a bug.

When a developer decides to investigate your report, they will start adding comments to the bug. There might be additional questions: please answer them as soon as possible.

When the developer has come to a conclusion, they will **resolve** the bug. That is done by changing the resolution status in the bug tracker. These statuses are phrased in developer speak, that is to say, they might sound quite rude to you. There's nothing that we can do about that, so do not take it personally. The bug reporter should *never* change the status after a developer changed it.

These are the most used statuses:

- Unconfirmed: your bug has not been investigated yet, or nobody can reproduce your bug.
- Confirmed: your bug is a bug, but there is no solution yet.
- Assigned: your bug is a bug, someone is going to work on it. There probably will be a corresponding task on the https://phabricator.kde.org/project/view/8/ developer workboard.
- Resolved/Fixed: your bug was a genuine problem in Krita's code. The developer has fixed the issue and the solution will be in the next release.
- Duplicate: your bug has been reported before.
- Needinfo/WaitingForInfo. You need to provide more information. If you do not reply within a reasonable amount of time the bug will be closed.
- Resolved/Invalid: your report was not about a bug.
- Resolved/Upstream: the issue you observed is because of a bug in a

library Krita uses, or a hardware driver, or your operating system. We cannot do anything about it.
- Resolved/Downstream: Only on Linux. The issue you observed happens because your Linux distribution packages Krita in a way that causes problems.

See also our chapter on Bug Triaging
[https://docs.krita.org/en/untranslatable_pages/triaging_bugs.html]

Running Krita from Source

Strokes queue

Strokes, jobs… What it is all about? (theory)

Structure of a stroke

An abstraction of a *stroke* represents a complete action performed by a user. This action can be canceled when it has not been finished yet, or can be undone after it's undo data has been added to the undo stack. Every stroke consists of a set of *stroke jobs*. Every job sits in a queue and does a part of work that the stroke as a whole must perform on an image. A stroke job cannot be canceled while execution and you cannot undo a single job of the stroke without canceling the whole stroke.

Example: Lets look at how the Freehand Tool works. Every time the user paints a single line on a canvas it creates a *stroke*. This stroke consists of several *stroke jobs*: one job initializes indirect painting device and starts a transaction, several jobs paint dabs of a canvas and the last job merges indirect painting device into the canvas and commit the undo information.

The jobs of the stroke can demand special order of their execution. That is the way how they will be executed on a multi-core machine. Every job can be either of the type:

`CONCURRENT`

concurrent job may be executed in parallel with any other concurrent job of the stroke as well as with any update job executed by the scheduler

Example: in Scale Image action each job scales its own layer. All the jobs are executed in parallel.

`SEQUENTIAL`

if the job is *sequential*, no other job may interleave with this one. It means that when the scheduler encounters a sequential job, it waits until all the other stroke jobs are done, starts the sequential job and will not

start any other job until this job is finished. Note that a sequential job can be executed in parallel with update jobs those merge layers and masks.

Example: All the jobs of the Freehand Tool are sequential because you cannot rearrange the painting of dabs. And more than that, you cannot mix the creation of the transaction with painting of anything on a canvas.

`BARRIER`

barrier jobs are special. They created to allow stroke jobs to synchronize with updates when needed. A barrier job works like a sequential one: it does not allow two stroke jobs to be executed simultaneously, but it has one significant addition. A barrier job will not start its execution until *all* the updates (those were requested with `setDirty()` calls before) has finished their execution. Such behavior is really useful for the case when you need to perform some action after the changes you requested in previous jobs are done and the projection of the image does now correspond the changes you've just done.

Example: in Scale Image action the signals of the image like `sigSizeChanged` should be emitted after all the work is done and all the updates are finished. So it runs as a barrier job. See `KisProcessingApplicator` class for details.

Besides one of the types above a job may be defined as `EXCLUSIVE`. Exclusive property makes the job to be executed on the scheduler exclusively. It means that there will be no other jobs (strokes or updates) executed in parallel to this one.

The queue of strokes

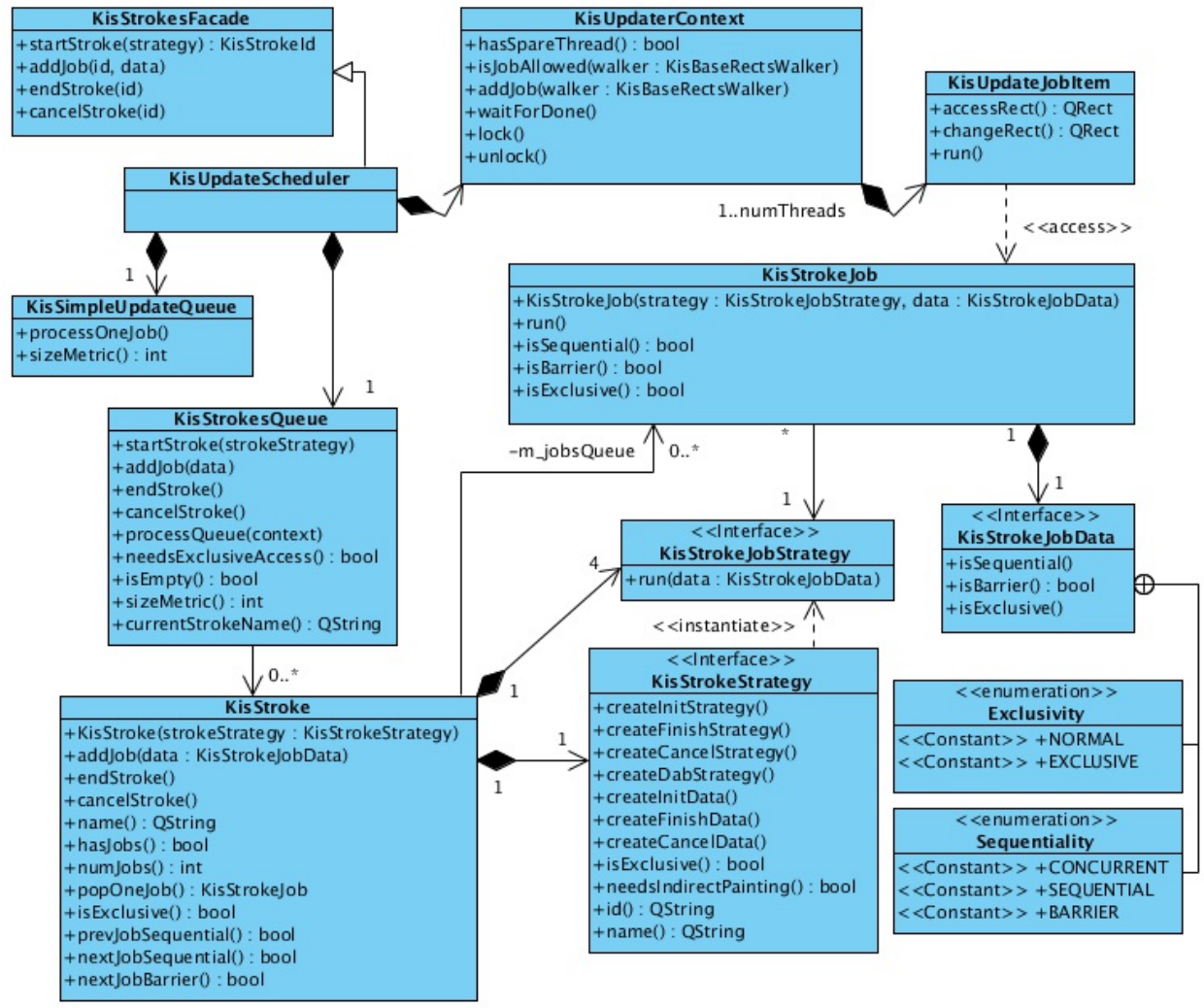
The strokes themselves are stored in a queue and executed one by one. This is important to know that any two jobs owned by different strokes cannot be executed simultaneously. That is the first job of a stroke starts its execution only *after* the last job of the previous stroke has finished.

The stroke is just a container for jobs. It stores some information about the work done, like `id()` and `name()`. Alongside storing this information it can affect the order of execution of jobs as well. The stroke can be defined

exclusive. The meaning of this resembles the behavior of stroke job's exclusive property. *Exclusive stroke* is a stroke that executes its jobs with all the updates blocked. The execution of updates will start only after the stroke is finished.

Implementation (practice)

Implementation of a stroke



Overview of stroke classes

Each stroke is represented by a `KisStroke` object. It has all the basic manipulating methods like: `addJob()`, `endStroke()` and `cancelStroke()`.

The behavior of a stroke is defined by a *stroke strategy* (KisStrokeStrategy class). This strategy is passed to the KisStroke object during construction and owned by the stroke.

Each stroke job is represented by `KisStrokeJob` object. The queue of `KisStrokeJob` objects is stored in every stroke object. This very object is used for actual running the job (`KisUpdateJobItem` calls `KisStrokeJob::run()` method while running). The behavior of the stroke job is defined by a strategy (`KisStrokeStrategy`) and a data (`KisStrokeJobData`). Those two objects are passed during the construction of the KisStrokeJob object.

A stroke can have four types of jobs:

- initialization
- canceling
- finishing
- actual painting (named as ‘dab’ in the code)

During construction the stroke asks its strategy to create strategies for all the four types of job. Then it uses these strategies on creation of jobs on corresponding events: initialization, canceling, finishing and when the user calls `addJob()` method.

The strategies define all the properties of strokes and stroke jobs we were talking above. The data class is used for passing information to the stroke by high-level code.

Example: `FreehandStrokeStrategy::Data` accepts such information as: `node`, `painter`, `paintInformation`, `dragDistance`

Other information that is common to the whole stroke like names of the paintOp, compositeOp are passed directly to the constructor of the stroke strategy.

Execution of strokes by `KisStrokesQueue`

The key class of the strokes’ execution is `KisStrokesQueue`. The most

important method that is responsible for applying all the rules about interleaving of jobs mentioned above is `KisStrokesQueue::processOneJob`. This method is called by the update scheduler each time a free thread appears. First it gets the number of merge and stroke jobs currently executing in the updater context. Then it checks all the rules one by one.

Canceling and undo information trick

It was stated above that a stroke can be canceled in each moment of time. That happens when a user calls `KisStroke::cancelStroke()` method. When it is requested the stroke drops all the jobs those are present in its queue and has not been started yet. Then it enqueues a special kind of job named *cancel job* that reverts all the work done by the stroke. This is used for interactive canceling of tools' strokes.

Taking into account that the strokes can be reverted, we cannot use `QUndoStack` capabilities directly. We should add commands to the stack *after* they have been executed. This resembles the way how `KisTransactionData` works: its first redo() method doesn't do anything because everything has already been painted on a device. Here in strokes this "after-effect-addition" is implemented in general way. Strokes work with a special kind of undo adapter: `KisPostExecutionUndoAdapter`. This adapter wraps the commands in a special wrapper that puts them into the stack without calling `redo()` and controls their threaded `undo()` and `redo()` operations. See information about `KisPostExecutionUndoAdapter` in a separate document.

Queues balancing

So we ended up with a solution where our scheduler has two queues that it should spread between limited amount of threads. Of course there should be some algorithm that balances the queues. Ideally, we should balance them by the total area of image the queue should process. But we cannot achieve that currently. So the formula for size metrics is quite simple:

```
updatesMetric = <number of update jobs in the queue>

strokesMetric = <number of strokes> * <jobs in the first stroke>
```

Balancing formula:

```
balancingRatio = <updatesMetric> / <strokesMetric>
```

Starting a stroke

The main entry point to strokes for the user is `KisStrokesFacade` interface. This interfaces provides four methods: `startStroke()`, `addJob()`, `endStroke()` and `cancelStroke()`. So every time you work with strokes you should work using this interface.

Note: KisImage and KisUpdateScheduler both implement this interface, so you can use them as a strokes facade. But please try not to store pointers to the whole image. Try store a link to interface only, if possible.

So if you want to start a stroke you should do the following:

1. Create a stroke strategy
2. Start a stroke with:

   ```
   KisStrokeId strokeId = strokesFacade-
   >startStroke(myStrategy);
   ```

   Note: you'll get a KisStrokeId handle for the stroke you created. This handle will be used in all the other methods for controlling the stroke. This handle is introduced, because several users can access the strokes facade simultaneously, so there may be several strokes opened simultaneously. It's important to understand that even when several strokes are opened simultaneously, only one of them executes on the cpu. All the other strokes will be delayed until it is finished.
3. Create a data for your stroke job
4. Add a job to the execution queue:

   ```
   strokesFacade->addJob(strokeId, myData);
   ```
5. You may add as many jobs as you wish

6. End or cancel the stroke:

   ```
   strokesFacade->endStroke(strokeId);
   ```

   or

   ```
   strokesFacade->cancelStroke(strokeId);
   ```

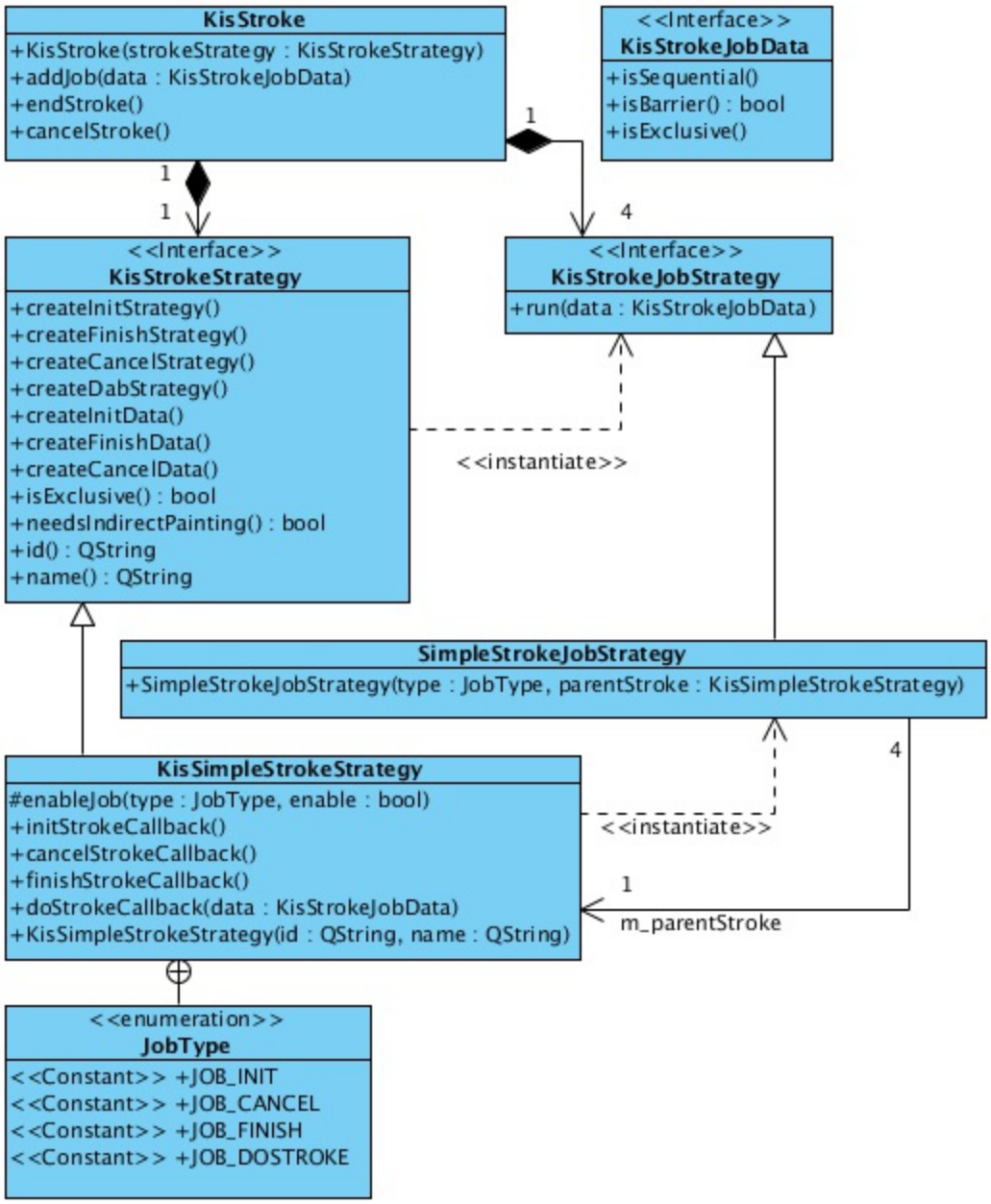
Strokes public API

Simplified stroke classes

As you might noticed the internal strokes API is quite complex. If you decide to create your own stroke you need to create at least six new classes:

- stroke strategy class
- four stroke jobs strategies (init, finish, cancel, dab)
- data that will be passes to a dab-strategy-based job

That is not really a good solution for a public API, so we introduced an adapter that simplifies all these stuff. The class is called `KisSimpleStrokeStrategy`. It allows you to define all the jobs you need in a single class.



Simple stroke classes

This class has four virtual methods those you can use as callbacks. When you need to use one of them just override it in your own class and add activation of the corresponding callback to the constructor of your class:

```
class MyOwnStroke : public KisSimpleStrokeStrategy {
    MyOwnStroke() {
        enableJob(KisSimpleStrokeStrategy::JOB_INIT);
        enableJob(KisSimpleStrokeStrategy::JOB_FINISH);
```

```
        enableJob(KisSimpleStrokeStrategy::JOB_CANCEL);
        enableJob(KisSimpleStrokeStrategy::JOB_DAB);
    }

    void initStrokeCallback()
    {
    }

    void finishStrokeCallback()
    {
    }

    void cancelStrokeCallback()
    {
    }

    void doStrokeCallback(KisStrokeJobData *data)
    {
        Q_UNUSED(data);
    }
};
```

Internally, `KisSimpleStrokeStrategy` creates all the job strategies needed for the lowlevel API. And these internal job strategies call the callbacks of the parental class.

Important: Notice that the job data passed to *init*, *finish* and *cancel* jobs is always null. It means that these jobs will always be *sequential* and *non-exclusive*. That is done intentionally to simplify the API. At the same time that is a limitation of the API. But currently, this is perfectly enough for us.

Unit-testing of the strokes

One of the benefits of using the strokes is that you are able to test them separately from the UI using a common infrastructure.

`utils::StrokeTester` class

That is a really simple class that you can use to test your own stroke. It test the following aspects of your stroke:

- canceling of the stroke
- working with indirect painting activated
- testing updates of the image projection after your stroke
- working with a layer that is not connected to any image

The result of the execution is compared against the reference png files those you create manually while writing your test.

How to write your own test

You can check examples in `MoveStrokeTest` and `FreehandStrokeTest` tests.

1. You need to inherit your tester class from `utils::StrokeTester`. The constructor of that class accepts the name of your stroke (it'll be used for generating filenames), size of the image and a filename of the preset for the paintOp.

```
StrokeTester(const QString &name, const QSize &imageSize,
             const QString &presetFileName = "autobrush_300px
```

2. Then you need to override at least two methods:

```
KisStrokeStrategy* createStroke(bool indirectPainting,
                                KisResourcesSnapshotSP resour
                                KisPainter *painter,
                                KisImageWSP image);
void addPaintingJobs(KisImageWSP image,
                     KisResourcesSnapshotSP resources,
                     KisPainter *painter);
```

   If you thing you need it you may do some corrections for the image and active node in the following method:

```
void initImage(KisImageWSP image, KisNodeSP activeNode);
```

3. Run your test in a testing slot:

```
void MyStrokeTest::testStroke()
{
```

```
    MyTester tester();
    tester.test();
}
```

4. During the first run the test will report you many fails and will generate you several files with actual result of the test. You need to check these files, then move them into the tests' data folder: `tests/data/<your_stroke_name>/`

5. After you copied the files the tester will compare the actual result against these very files. That means it'll catch all the changes in the work of your stroke, so you'll be able to catch all the regressions automatically.

Predefined classes for usage as base classes

`KisPainterBasedStrokeStrategy`

This class can be used for the strokes those work with the node using a painter (or painters like in `KisToolMultihand`). This class accepts resources snapshot (`KisResourcesSnapshot`) and a painter (painters). Initialization, finishing and canceling callbacks of this class do all the work for dealing with indirect painting support, creation of transaction, reverting the stroke on canceling. This base class is used for `FreehandStroke` mostly.

`KisStrokeStrategyUndoCommandBased`

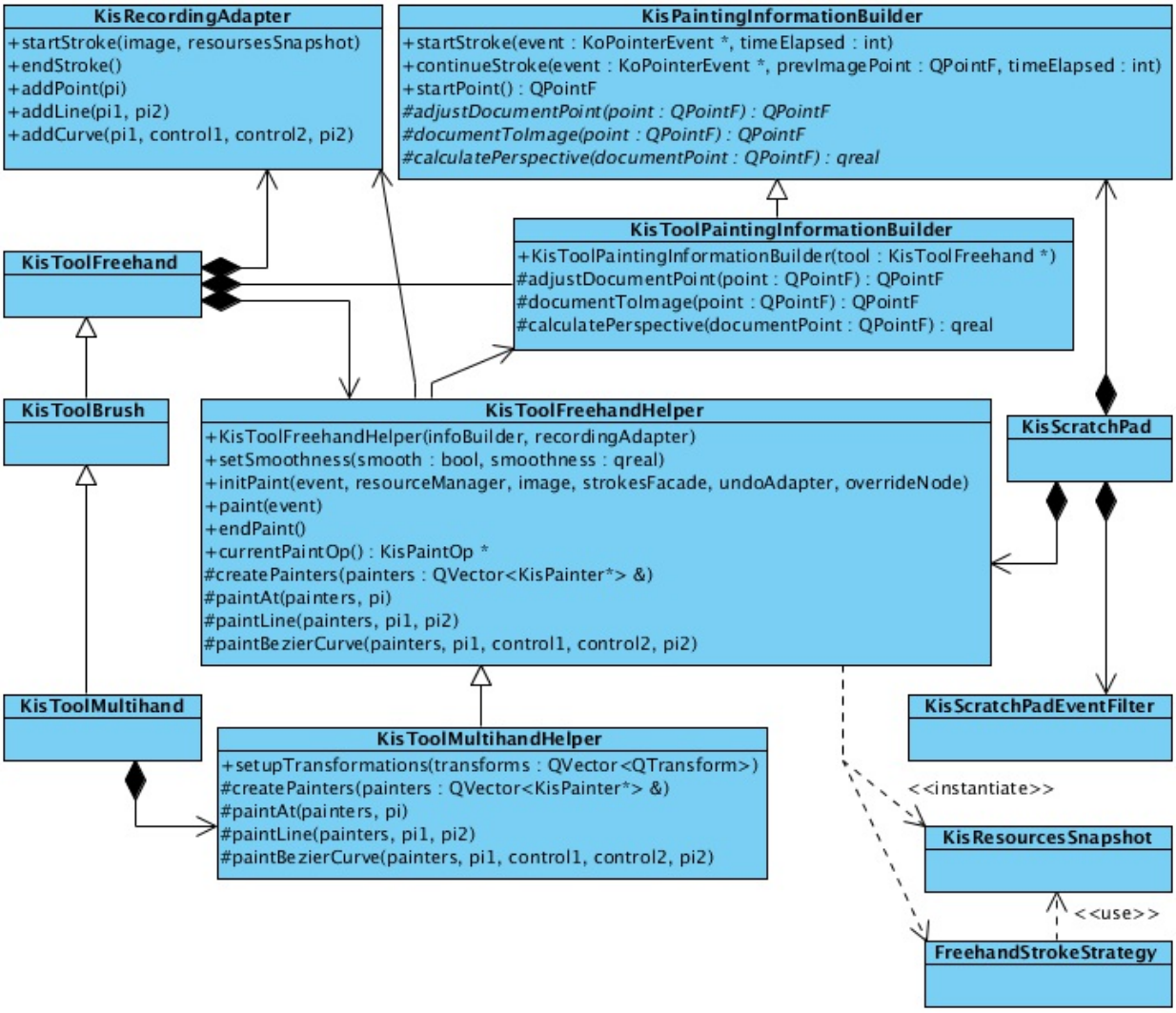
It is obvious from the name of the class that it works with undo commands. In constructor you define which method of undo command should be used undo() or redo(). Afterwards, you just add commands to the stroke and they are executed with any the sequentiality constraints. This stroke strategy does all the work for adding the commands to the undo adapter and for canceling them if needed.

Example classes

- `KisPainterBasedStrokeStrategy`

- `FreehandStrokeStrategy`
- `KisStrokeStrategyUndoCommandBased`
- `MoveStrokeStrategy`

Internals of the freehand tool



Freehand tool classes

Motivation for so many classes

We need to share the codebase between at least four classes:

`KisToolFreehand`, `KisToolMultihand`, `KisScratchPad`. All these classes paint on a canvas with `KisPainter`, so they share quite much common code.

KisResourcesSnapshot

After we introduced the strokes, the moments of time when user paints with mouse and when the line is actually painted on the canvas do not coincide. It means that by the time a thread starts actual changing the device, the contents of `KoCanvasResourceProvider` might have already changed. So before we start a stroke we should create a snapshot of all the resources we have and pass this snapshot to the stroke.

For this purpose we introduced `KisResourcesSnapshot` class. It solves two problems at the same time: first it stores all the resources we might have and second it encapsulates the algorithm of loading these resources into a `KisPainter` object. So this class is really easy to use. You just create the snapshot and then just load all the resources to the painter when needed.

```
KisResourcesSnapshotSP resources =
    new KisResourcesSnapshot(image,
                             undoAdapter,
                             resourceManager);
KisPainter painter;
painter.begin(device, selection);
resources->setupPainter(&painter);

// paint something

painter.end();
```

In our implementation this class is usually created by `KisToolFreehandHelper` and passed to the `KisPainterBasedStrokeStrategy` class. The latter one creates painters and initializes them using `setupPainter()`.

`KisToolFreehand` and `KisScratchPad`

The freehand tool is split into four classes:

`KisToolFreehand`
: highlevel tool class that get the mouse events form the Ko-classes and distributes events among internal classes.

`KisToolPaintingInformationBuilder`
: converts mouse events represented by `KoPointerEvent` objects into `KisPaintInformation` objects.

`KisRecordingAdapter`
: stays in charge of adding recording information into the image's action recorder. This class has two purposes: first we need to be able to disable recording for the scratch pad (then we just pass NULL instead of a recording adapter), second when the strokes are able to do their own recording, it'll be easier to port the freehand tool to it.

`KisToolFreehandHelper`
: this is the main class that combines all the classes we were talking above. It accepts a mouse event, converts it using a painting information builder into the paint information, notifies recording adapter, takes the snapshot of resources and finally starts a stroke. Then it populates the stroke with stroke jobs, when the user moves the mouse (`paint(event)` method) and finishes the stroke in the end.

Such splitting allows us to use the same classes in both `KisToolFreehand` and `KisScratchPad`. The only difference between them is that the scratch pad doesn't have a recording adapter at all, and uses base class `KisPaintingInformationBuilder` instead of `KisToolPaintingInformationBuilder`. The latter differs from the former one in a way that it supports painting assistants (`adjustDocumentPoint()` method), complex coordinate transformations with `KisCoordinatesConverter` (`documentToImage()` method) and perspective painting (`calculatePerspective()` method). The rest of the code is shared.

`KisToolMultihand`

Multihand tool uses the same classes. The only difference, it has a couple of modifications in its helper (`KisToolMultihandHelper`), those allow it to have several painters at the same time. The tool's class inherits the freehand tool's

class and just substitutes the helper with its own (with `resetHelper()` method).

Scheduled Undo/Redo

Two ways of working with undo commands

The key problem of designing the undo system for strokes was that there are two ways of working with undo commands. That is we have two types of commands actually:

- *Qt-like command* - command's redo() method is executed while the command is added into the undo stack
- *Transaction-like command* - the command is added to the stack *after* its action has already been performed. It means that the first redo() of this command (the one that is called by undo stack) does nothing. That is a transaction-like command just saves undo data for the future and does not perform anything on addition.

You already know that our strokes can be reverted on the go, it means that the stroke's undo command should be added to the undo stack only *after* all the actions of the stroke have been performed. So it looks like the stroke's commands are *transaction-like*.

But there is another problem: the stroke should be able to execute regular undo commands those are not transaction-like (like is it done in `KisStrokeStrategyUndoCommand`). More than that, undo and redo of for such strokes should be performed with the same sequentiality properties (read "undo/redo operations should be threaded as well").

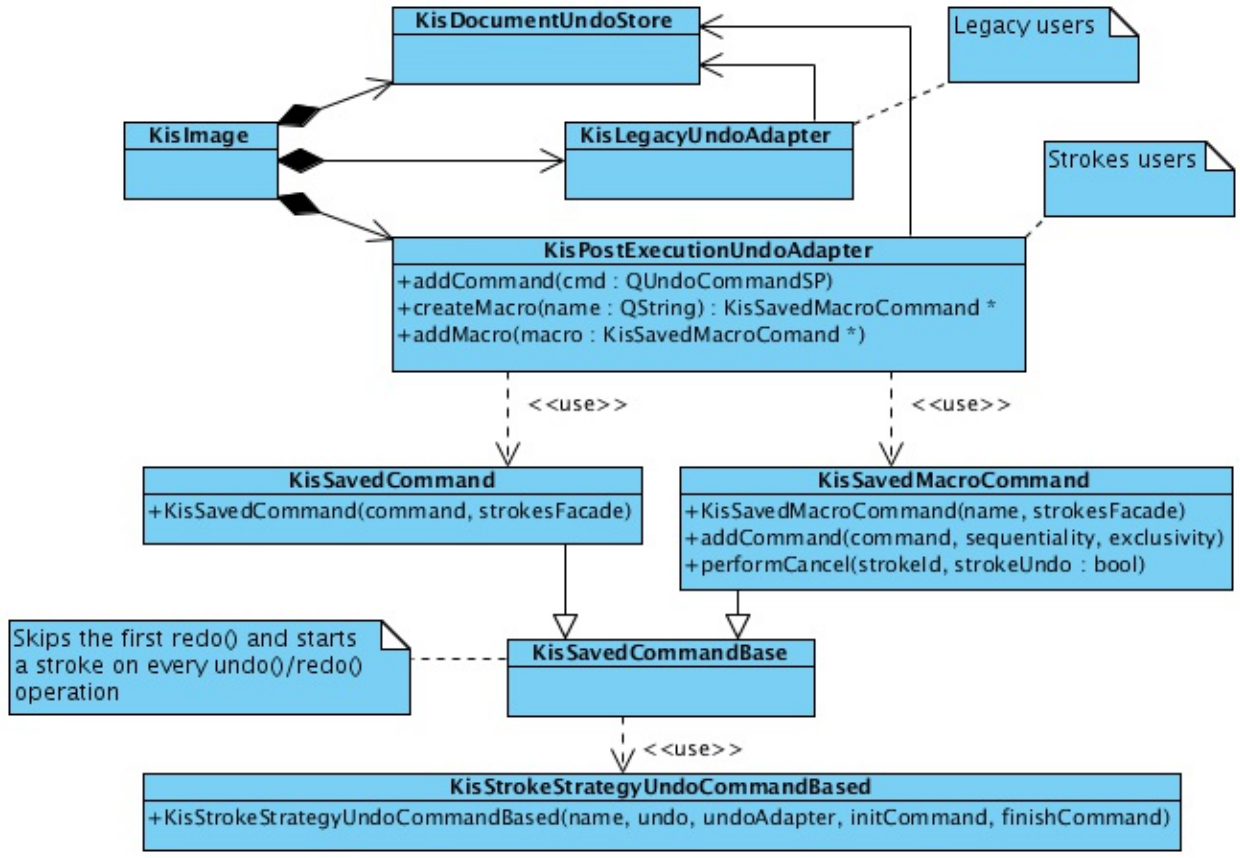
It follows that the undo commands generated by the stroke should be wrapped in a special *wrapper command*, lets call it `KisSavedCommand`, that hold the following properties:

- the wrapper skips the first redo(). It means the wrapped command's redo() method will not be called on its addition to the stack. Obviously,

it is not needed, because the action has already been performed by the stroke itself.

- when undo stack calls to undo/redo methods of the wrapper-command, the command creates a stroke (`KisStrokeStrategyUndoCommandBased`) and runs the wrapped command in a context of this stroke.
- a special *macro wrapper command*, lets call is `KisSavedMacroCommand`, should be able to save all the commands executed by a stroke and undo/redo all of them in the original order with original sequentiality properties (concurrent, sequential, barrier, exclusive).

That is exactly what we have: `KisSavedUndoCommand` skips the first redo and runs undo()/redo() of an internal command in a separate stroke. We have `KisSavedMacroCommand` as well to save the contents of the whole stroke.



Scheduled commands

New Undo Adapters

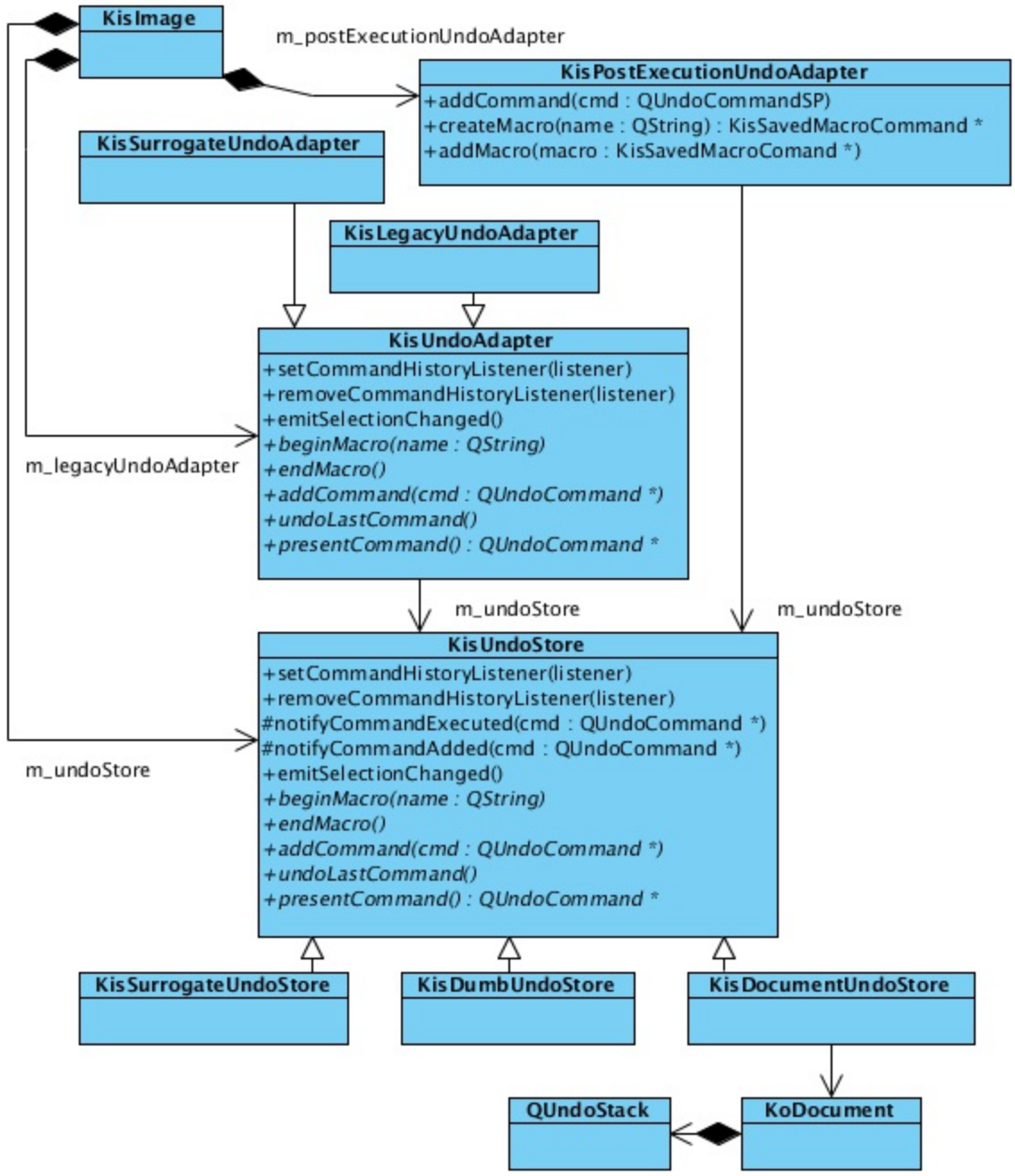
Well, it would be quite insane to ask all the users of strokes to wrap their commands into wrapper, so we introduced a separate undo adapter for strokes: `KisPostExecutionUndoAdapter`. This adapter wraps your command and puts it into the undo stack automatically. This is the only adapter we can use inside strokes, that is why all the strokes accept the pointer to it.

For the legacy code we still have `KisUndoAdapter`, but now we call it “legacy undo adapter”. It works as usual: it adds a command to undo stack directly, so it gets executed right in the moment of addition. But there still is one trick. Stroke’s commands come to the undo stack asynchronously, so if we try to simply add a command to the stack, we can catch a race condition easily. That’s why the legacy undo adapter must guard itself from strokes with locking the strokes system. That is done with a special kind of lock `barrierLock()`. This barrier lock differs from a regular lock in a way that it ways for all the running *strokes* are finished, while a regular lock waits for all the running *stroke jobs* are done. That’s the only difference.

The same race conditions problem applies to the undo()/redo() signals from the UI. The user may request the undo operation while the stroke is adding its commands. This will surely lead to a crash. We solved this problem in a bit hacky way: we hacked `QUndoStack` and made it’s undo()/redo() slots virtual. After that we overridden the stack with our own, and changed these methods to block the strokes while undo()/redo() is happening. We use `tryBarrierLock()` there, because it is easier to cancel the undo than to wait until all the strokes are finished.

Undo Adapters and Undo Stores

Well, we have two types of undo adapters now (not counting `KisSurrrogateUndoAdapter`). It’s obvious that they should share some code. That is why we split the work with the actual undo stack into a separate class `KisUndoStore`. So now the undo store defines “where to store the undo data”, and undo adapter defines “how to adapt krita’s commands to qt’s stack”. There are additional types of store classes for using in tests and for special purposes.



Undo Adapter vs Undo Store

Processings framework

Motivation

In Krita we have many actions which have common structure of execution. Take a look at actions like Scale Image, Rotate Image, Change Color Space - all of them have common phases:

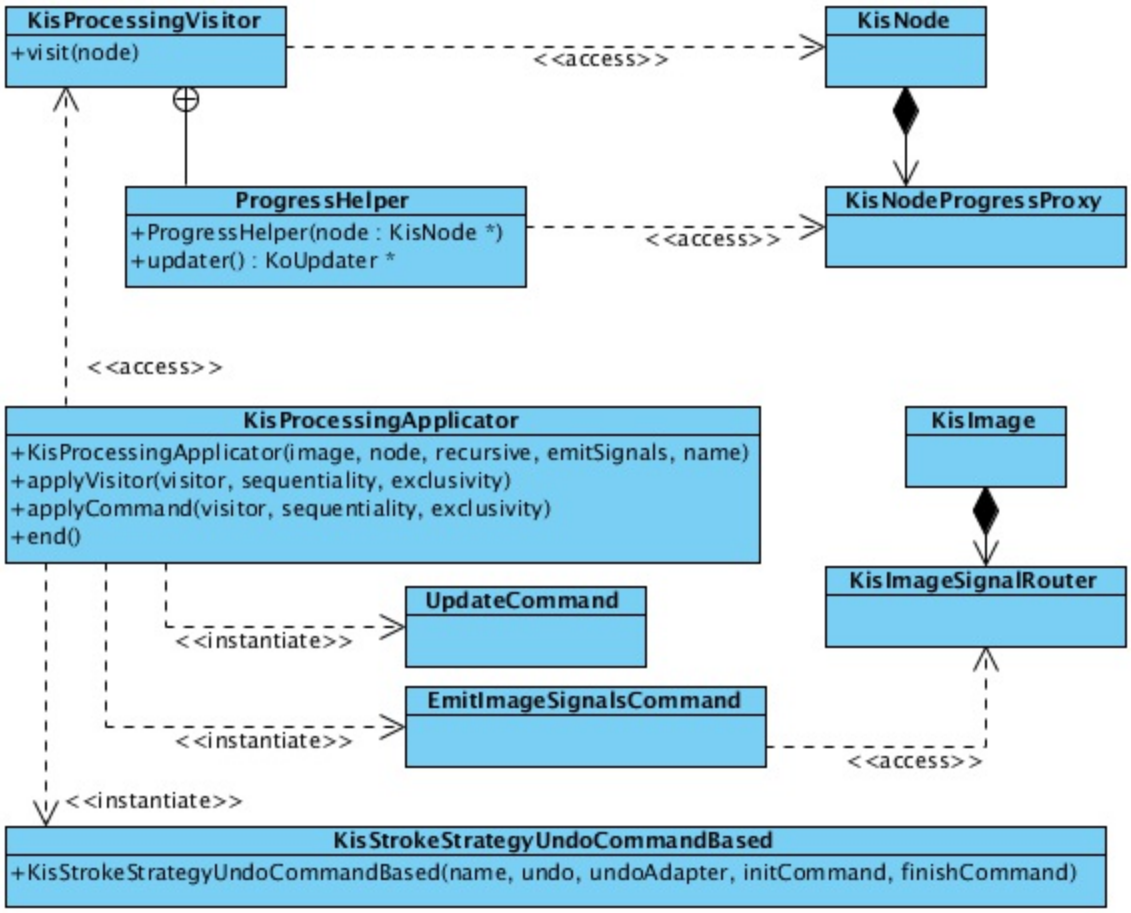
1. Lock the image
2. Do the processing of nodes
3. Unlock the image
4. Emit setDirty() calls and update the projection of the nodes
5. Wait until all the setDirty()'es are finished
6. Emit image's signals like sigImageSizeChanged

More than that, you should pay attention to the fact that all these actions should support undo/redo operations. And the last two phases cannot be implemented as usual qt-commands inside a usual macro, because they should always be executed *in the end* of the action (in qt commands are executed in reverse order during undo operations, that is not what we want).

And, btw, it would be really good idea to have multithreading support for such actions, because some of them (like Scale Image) may be quite slow.

`KisNodeVisitor` cannot fit all these requirements, because it has important design limitations: first, walking through nodes is implemented inside the visitor itself and, second, emitting signals is put into visitors as well. These two limitations prevent the code to be shared between actions. That is why we introduced new shiny `KisProcessingVisitor` and a separate framework for them.

Processing visitors



Processing framework

The key class of the processing framework is `KisProcessingVisitor`. Its main difference from the old visitor is that it is extremely simple. It performs one task only, it processes one node. And that is all. It does no locking, performs no updates, emits no signals. It just processes (that is, changes the content) a single node. You can look at the reference implementation of it in `KisCropProcessingVisitor` and `KisTransformProcessingVisitor`. The key idea of this framework is to keep the processings as simple as possible. So the rest of the work is done by external classes, those are shared between all the processings.

We have one such class. Its name is `KisProcessingApplicator`. This class performs several tasks:

- creates a stroke. So all the actions executed with this applicator will be undo/redo'able.

- applies a visitor to a requested node.
- applies a visitor recursively to a node and all its children. Note, that you can choose any sequentiality property for the execution of your visitor. It means that the visitors can be applied to nodes concurrently in multithreaded way.
- applies a usual qt-command to the image. Sequentiality properties may vary as well.
- emits setDirty() calls for all the nodes which need it. It is done in efficient way, so no nodes are updated twice.
- emits image signals *after* all the actions and updates are finished.

Lets look at an example:

```
void KisImage::resizeImageImpl(const QRect& newRect, bool cropLay
{
    if(newRect == bounds()) return;

    QString actionName = cropLayers ? i18n("Crop Image") : i18n("

(1) KisImageSignalVector emitSignals;
(2) emitSignals << SizeChangedSignal << ModifiedSignal;

(3) KisProcessingApplicator applicator(this, m_d->rootLayer,
                                       KisProcessingApplicator::R
                                       emitSignals, actionName);

    if(cropLayers || !newRect.topLeft().isNull()) {
(4)     KisProcessingVisitorSP visitor =
            new KisCropProcessingVisitor(newRect, cropLayers, tru
(5)     applicator.applyVisitor(visitor, KisStrokeJobData::CONCUR
    }
(6) applicator.applyCommand(new KisImageResizeCommand(this, newRe
(7) applicator.end();
}
```

In lines (1) and (2) we create a list of signals we should emit after the execution of the applicator. This list should be passed to the *constructor* of the applicator (3) (the list is passed to the constructor instead of end() function, because we face a limitation connected with the internals of the implementation of undo for processings, I doubt it can create any troubles). In the line (3) we create a recursive applicator. In lines (4) and (5) we create a

visitor and apply it to nodes recursively in a multithreaded way. **Warning:** the visitor is shared between all the threads so it should be written in a *thread-safe* way. In line (6) we apply a command sequentially, it means that it'll be executed right after *all* the threads with visitors has finished. Line (7) closes the stroke an tells it to perform all the updates and emit all the signals.

Implementation of `KisProcessingApplicator`

The applicator is based on the "undo command"-based stroke (`KisStrokeStrategyUndoCommandBased`). It starts the stroke in the constructor and adds undo commands to it on every user request. The processings are inernally wrapped into a special command (`KisProcessingCommand`). This command has its own undo stack that collects the transactions executed by the processing. This can be easily achieved with our undo adapters interface. The command just defines its own `KisSurrogateUndoAdapter` and passes it to the processing. Processing adds its transactions to the fake adapter. And later, the command just uses the undo stack to undo/redo actions executed by the transaction.

The applicator defines several internal commands as well: `UpdateCommand` and `EmitSignalsCommand`. These commands are added to the beginning and to the end of every stroke, so that they can be executed in the end of both undo and redo operations. The parameter `finalUpdate` controls whether the command is executed during its redo() or undo() operation.

Emission of signals trick

After actions have been moved to separate threads, problems with image signals appeared. When everything was executed in a single thread the connection of signals like `sigAboutToAddNode` and `sigNodeHasBeenAdded` worked as *Qt::DirectConnection*. So these signals were effectively function calls. After we moved the actions to a separate thread, all of them became *Qt::QueuedConnection*. I guess you know what it means. They simply lost all their sense. So we had to start to use *Qt::BlockingQueuedConnection*. But there is another problem with it. Some of the (old) code is still executed in a context of the UI thread and they emit signals as well. So all that code causes

deadlocks when using `Qt::BlockingQueuedConnection`. That is why we had to introduce `KisImageSignalRouter`. This class checks which thread emits the signal and emits it either using `Qt::DirectConnection` or `Qt::BlockingQueuedConnection`. So no deadlocks are possible.

Progress reporting

The fact that a processing visitor does a really simple task (processes a single node) that is very easy to report progress using progress bars in the layer box. We just need to use progress proxy of the node we process (`KisNodeProgressProxy`). Our processings framework provides an even easier way of doing this. You just need to instantiate a `ProgressHelper` object and ask it to create a `KoUpdater` object for you. And all is done. You can see an example in `KisTransformProcessingVisitor` class.

Testing

Usage of a common framework makes testing really simple. There is a separate unittest in image's tests folder: `KisProcessingsTest`. To test a processing you need to write just a couple of lines. Everything is done by `BaseProcessingTest` helper class. This class will run your processing and compare results against reference png files those are stored in data folder. If there are some problems found, it'll dump result files to the current directory.

Testing Strategy

Overview

We're always working on the next version of Krita. We fix bugs and implement new features. Every change to any software comes with a risk of introducing other issues. That's where testing comes in. The tester's job is to uncover defects as early as possible in the development process: a bug caught early enough means easier fixing, better user experience and less load on user support.

The Functional Test Suite

When testing functionality we employ multiple strategies that translate into several layers of the test suite:

- Unit tests are our safeguard against breakage during development.
- End to end tests check that the basic high level workflows function properly.
- Exploratory testing experiments. Unexpected combinations, uncharted workflows.

Test Suite Layers

Unit Tests

Unit tests are the base of our test suite. They are designed to ensure that every individual unit of source code (both backend and UI components) functions as expected. They are fully automated and fast to execute. They are run by developers during development. Also part of nightly testing suite.

In-depth unit testing doc:
https://docs.krita.org/en/untranslatable_pages/unit_tests_in_krita.html

End-to-end UI Tests

Formalized high level tests performed on the running application; carried out either by a computer or by a human.

End to end tests cover both the GUI and the command line interface.

Exploratory Testing

While the other layers of the test suite are composed of carefully curated scripted tests, balancing between coverage and efficiency, exploratory testing approaches testing quite differently. It’s purpose is to allow humans to apply their unique tools: learning, creativity, intuition. There are no suites, scenarios, defined steps. Just you and Krita. Explore and experiment. Try basic workflows. Try unexpected combinations. Try to break things. Then report bugs.

When Do We Test

Continuous testing as part of continuous integration

Automatic test suite is run nightly against the last nightly build.

Beta Testing

Beta testing is a type of user acceptance testing, where a subgroup of target users validates the upcoming release.

As a part of the release process, we collect features and bugs (mainly high impact bugs and those that benefit from testing in multiple different conditions) to test in a Phabricator task connected to the release. From that collection we create a survey on survey.kde.org and publicly release the beta version. Link to the survey is available on the welcome page of the beta release.

Triaging Bugs

There are over 1000 bugs and 350 wishes reported against Krita per year, and that number is rising. The Krita developers cannot handle that stream on their own! Please consider helping out by triaging bugs. This document gives some simple guidelines to get started, and some common cases that can often be answered with a standard text.

For more details, see also https://community.kde.org/Guidelines_and_HOWTOs/Bug_triaging

Contents

Status flow

A bug begins as `UNCONFIRMED`. When triaging, only `UNCONFIRMED` bugs are

still relevant.

Platform

If the user has not entered the Platform correctly (i.e., it is “unspecified/Linux”), then ask which platform they are using. Mark the bug as `NEEDDINFO/WAITINFORINFO`.

> **Tell the user:**
>
> Please indicate your operating system correctly. For Linux, select the distribution, appimage or compiled from sources and Linux, for Windows, select MS Windows/MS Windows, for OS X or macOS, select macports, disk images or homebrew and OS X.

If the user has selected Windows CE for platform, set it to MS Windows without asking them.

Version

If the user has not entered the version (i.e., the version is unspecified), ask them for the version and mark the bug as `NEEDDINFO/WAITINFORINFO`.

> **Tell the user:**
>
> Please select the version of Krita you are using. You can find the version in Help/About Krita.

Can Reproduce

- If you can reproduce the bug, add a comment indicating you can reproduce it, maybe with clearer steps to reproduce and anything pertinent that you observed. If you have a backtrace, also add it. Set the bug status to `CONFIRMED` and add the `triaged` keyword to the keywords.

- If you can reproduce the bug, and want to go the extra mile, use an older version of Krita to see whether you could reproduce it there as well. If you couldn't, it's a *regression*, so add the regression keyword to the keywords and mark which version of Krita the latest was that did not have the bug.

Cannot Reproduce

- If you cannot reproduce, the user either has not given enough information or the bug is specific to their system.

- If there is not enough information, ask for more information. Depending on the report, the steps to reproduce might need to be described more clearly and/or a screenshot, a screen recording or the original files might be necessary. Set text (ask for what you think is needed):

  Ask the user:

  I am sorry, I cannot reproduce your issue. Could you specify the steps to reproduce more clearly, and maybe add a screen recording/screenshot/original file

  - Mark the bug as `NEEDINFO/WAITINGFORINFO`.

- If the issue seems to be specific to the user's system, ask for the output of help/System information for bug reports as well. Set text:

  Tell the user:

  I am sorry, but I cannot reproduce the bug on my system. Please add the output of help/System Information for Bug reports as well.

  - Mark the bug as `NEEDINFO/WAITINGFORINFO`.

Importance

Importance is a tool for developers, not for bug reports. It's developers and triagers who decide what the importance is. If a bug reporter complains about a change in importance, use this text:

Tell the user:

I am sorry, but the importance field is a tool for the developers to work with. Please do not change the importance back.

There are the following Importances:

Critical:
: the bug leads to immediate dataloss. Example: a saved file cannot be opened in Krita

Grave:
: shouldn't be used, it doesn't mean a thing

Major:
: it's a bug, but it's kinda important.

Crash:
: the bug is about a crash or an assert [1]

Normal:
: it's a bug

Minor:
: it's a bug, but it's kinda unimportant

Wish:
: it's a feature request

Task:
: not used.

The main difference is between Wish and the rest: Wishes are feature requests, and don't need immediate triaging. A wish bug is a bug that asks whether some functionality can be added to Krita, or complains that some functionality is missing.

The rest are bugs, that is, problems in Krita that can be fixed by changing Krita's code.

However, we also get many reports that are not bugs and not wishes: reports that are basically users asking for help because they do not understand Krita or their computer, or what a file is, or that Krita isn't the same application as Photoshop. Those reports need to be weeded out, and the status set to `INVALID`.

Guidance for using Importance

- If you encounter a bug that reports dataloss when loading a saved file, set it to critical.
- If you encounter a bug that reports a crash or an assert but is not set to crash, set it to crash.
- If you encounter a report that asks for functionality that is not currently present, set it to wish.
- If you encounter a report that is a user request, check whether you can reply with a link to the faq (https://docs.krita.org/en/KritaFAQ.html), and maybe a canned answer, and change the status of the bug to `INVALID`.

Asserts and Crashes

[1] **Crash or assert.**

These are different things. A crash happens when Krita spontaneously stops working *or* hangs. An assert happens when Krita stops working because we, developers, have added some code to detect an invalid state.

Asserts are printed to the terminal or shown in a popup window. You can identify an assert by asking for terminal output, debugview output or

by checking the backtrace, if there is one.

If the backtrace contains a line like:

```
> SAFE ASSERT (krita): "!sanityCheckPointer.isValid()" in fil
```

Or another mention of assert, Q_ASSERT or similar, it's an assert, not a crash.

Canned Answers and Recognizing Common Reports

We get a lot of duplicate bug reports. Sometimes it's clear that it's a duplicate, and you can mark it a such. In all cases, we want to give the reporter useful information so they can solve their problems. Of course, (almost) all solutions are also in the FAQ, but just pointing people to the FAQ is often considered impolite.

So, do never reply to a bug report with:

> "Just read the FAQ."

It takes a bit of experience to recognize a bug from an often incomplete description. Here are a couple of common cases:

Cannot Save

For instance: "I cannot save/my file doesn't get saved/it says it cannot copy the file"

This happens most often on Windows, if the user has got any security software installed that doesn't come with Windows. Examples are Sandboxie, Totaldefender, or others. Mark the bug as `NEEDSINFO/WAITINGFORINFO` and add this text:

Ask the user:

Are you using Windows? If so, do you have any non-standard security software installed such as Total Defender, Sandboxie or XXX? Please make an exception for Krita in the settings, or uninstall this software. Since Windows 10, it is no longer necessary to have any security software installed other than what comes with Windows.

If the user replies that they are using extra security software, close the bug as `RESOLVED/INVALID`.

Broken Canvas

This happens on Windows. Symptoms will be: the canvas is black, the canvas stays blank, the canvas only updates when the user clicks outside the canvas. Mark the bug as a duplicate of https://bugs.kde.org/show_bug.cgi?id=360601, and add the following text:

Tell the user:

You probably are using a Windows system with an Intel display chip. Please update to Krita 3.3.3, which enables the Direct3D (Angle) renderer by default. If you do not want to update, check https://docs.krita.org/en/KritaFAQ.html#krita-starts-with-an-empty-canvas-and-nothing-changes-when-you-try-to-draw-or-krita-shows-a-black-or-blank-screen-or-krita-crashes-when-creating-a-document-or-krita-s-menubar-is-hidden-on-a-windows-system-with-an-intel-gpu

My stylus has an offset

This happens on Windows. Symptoms will be: the user reports that the stylus cursor has an offset or moves the cursor on another screen. Usually, the user will have a misconfigured multi-monitor system. Mark the bug as `NEEDSINFO/WAITINGFORINFO` and ask the user:

Ask the user:

Do you have a multi-monitor setup? If so, this is a configuration issue. Please reset your tablet driver's configuration and Krita's configuration (https://docs.krita.org/en/KritaFAQ.html#resetting-krita-configuration). If you have a single-monitor setup, then please calibrate your tablet.

If the user checks back and tells us the problems are solved, mark the bug as `RESOLVED/UPSTREAM`.

Other tablet issues

Often, the user will tell you that their tablet will work perfectly with another application. This is not relevant.

Tell the user:

Windows tablet drivers often have a special code for different applications. Whether an application works or not depends on whether the programmers have tested their driver with an application or not. Tablet issues are almost always caused by the drivers being broken.

Krita lags

The word "lag" is meaningless. Complaints about "lag" are not bug reports. However, we should help the complainer.

Mark the bug as `NEEDSINFO/WAITINGFORINFO` and ask the user:

Ask the user:

Have you enabled the stabilizer? Check the tool options panel for the freehand tool. Also check the other possibilities mentioned here: https://docs.krita.org/en/KritaFAQ.html#krita-is-slow

I cannot paint at all, in a particular document

The user probably created, accidentally, a tiny selection, and saved that with the document. Mark as `NEEDSINFO/WAITINGFORINFO` and ask them:

Ask the user:

Do you have a selection saved with that document? Use select/deselect on your image and check whether you can paint again. If not, please attach the `.kra` document to this bug report or make it available.

Unittests in Krita

Contents

What is a unit test is and why is it needed?

Wiki:

> A unit test is a piece of code that automatically checks if your class or subsystem works correctly. The goal of unit testing is to isolate each part of the program and show that the individual parts are correct. A unit test provides a strict, written contract that the piece of code must satisfy. As a result, it affords several benefits [1].

Comment:

> In other words unit testing allows the developer to verify if his initial design decisions has been implemented correctly and all the corner-cases are handled correctly.

In Krita Project we use unit tests for several purposes. Not all of them work equally good, but all together they help developing a lot.

Debugging of new code

Wiki:

> Unit testing finds problems early in the development cycle. This includes both bugs in the programmer's implementation and flaws or missing parts of the specification for the unit. The process of writing a thorough set of tests forces the author to think through inputs, outputs, and error conditions, and thus more crisply define the unit's desired behavior. The cost of finding a bug before coding begins or when the code is first written is considerably lower than the cost of detecting, identifying, and correcting the bug later; bugs may also cause problems for the end-users of the software [1].

Comment:

> Krita is a big project and has numerous subsystems that communicate with each other in complicated ways. It makes debugging and testing new code in the application itself difficult. What is more, just compiling and running the entire application to check a one-line change in a small class is very time-consuming. So when writing a new subsystem we usually split it into smaller parts (classes) and test each of them individually. Testing a single class in isolation helps to catch all the corner-cases in the class public interface, e.g. "what happens if we pass 0 here instead of a valid pointer?" or "what if the index we just gave to this method is invalid?"

Changing/refactoring existing code

Wiki:

> Unit testing allows the programmer to refactor code or upgrade system

libraries at a later date, and make sure the module still works correctly (e.g., in regression testing). The procedure is to write test cases for all functions and methods so that whenever a change causes a fault, it can be quickly identified. Unit tests detect changes which may break a design contract [1].

Comment:

Imagine someone decides to refactor the code you wrote a year ago. How would he know whether his changes didn't break anything in the internal class structure? Even if he/she asks you, how would you know if the changes to a year-old class, whose details are already forgotten, are valid?

Automated regression testing

Most of our unit tests are run nightly on the CI. You can see the results and coverage reports at https://build.kde.org/job/Extragear/job/krita/.

However, some of the unit tests are not stable enough to be run automatically and therefore are disabled. (They do straightforward QImage comparisons, so the test results can depend no only on version of the libraries installed, but also on build options and even type of CPU the tests are run on.) While the overall coverage is decent, this issue limits the ability of the unit test suite to catch regressions in several parts of the codebase. (More information on that in the respective Phabricator task: https://phabricator.kde.org/T11904.)

How to build and run tests?

Building the tests

To enable unit tests, build Krita with an additional cmake flag: `-DBUILD_TESTING=ON`.

```
# example build command
you@yourcomputer:~/kritadev/build>cmake ../krita \
  -DCMAKE_INSTALL_PREFIX=$HOME/kritadev/install \
  -DCMAKE_BUILD_TYPE=Debug \
  -DKRITA_DEVS=ON \
  -DBUILD_TESTING=ON
```

See also

- If you need help with building from source, see Building Krita from Source
- For more information about cmake options, please refer to CMake Settings for Developers

Once built, the tests are run from the the **build** directory. There you can either run the whole suite at once or you can run a single test (or even a single test with a single data row for data-driven tests).

Run all the tests

```
# change to the build directory
you@yourcomputer:~/> cd kritadev/build
# run the whole suite
you@yourcomputer:~/kritadev/build> make test
```

Run a single test

Every test class is built into a separate executable file. This executable file resides in the build directory tree. The relative path is the same as the path in source directory.

To run all tests in a single test class, run the executable:

```
# running all tests in a test class
you@yourcomputer:~/kritadev/build>./libs/ui/tests/KisSpinBoxSplin
```

You can also run a single test method from the class or invoke the test method with a single test data row, if you have a data-driven test. Add the test method name (and optionally the test data row name) as an argument to the test class executable:

```
# the syntax for running single tests:
# you@yourcomputer:~/kritadev/build>./test-class-executable "test

# run a single method in a test class
```

```
you@yourcomputer:~/kritadev/build>./libs/ui/tests/KisSpinBoxSplin

# run a single method in a test class with the selected test data
you@yourcomputer:~/kritadev/build>./libs/ui/tests/KisSpinBoxSplin
```

Environment variables for running tests

Prior to running the tests, you can set several environment variables to change the behavior of the tests.

- Suppress safe assert dialogs:

  ```
  you@yourcomputer:~/kritadev/build> export KRITA_NO_ASSERT_MSG
  ```

- Set source directory for QImage-based test data

  ```
  you@yourcomputer:~/kritadev/build> export KRITA_UNITTESTS_DAT
  ```

- Create reference images for QImage-based tests

  ```
  you@yourcomputer:~/kritadev/build> export KRITA_WRITE_UNITTES
  ```

When to write a unit test?

Ideally a unit test should be written for any new class that implements some logic and provides any kind of public interface. It is especially true if this public interface is going to be used more that one client-class.

What should unit test include?

- corner cases. E.g. what happens if we request merging of two layers, one of which has Inherit Alpha option enabled? What properties and composition mode the final layer should have? Answers to these questions should be given and tested in the unit test.
- non-obvious design decisions. E.g. if a paint device has a non-

transparent default pixel, then its `exactBounds()` returns the rect, not smaller that the size of the image, even though technically the device might be empty.

How to write a unittest?

Suppose you want to write a unittest for kritaimage library. You need to perform just a few steps:

1. Add files for the test class into `./image/tests/` directory:

   kis_some_class_test.h

   ```
   #ifndef __KIS_SOME_CLASS_TEST_H
   #define __KIS_SOME_CLASS_TEST_H

   #include <QtTest/QtTest>

   class KisSomeClassTest : public QObject
   {
       Q_OBJECT
   private Q_SLOTS:
       void test();
   };

   #endif /* __KIS_SOME_CLASS_TEST_H */</syntaxhighlight>
   ```

   kis_some_class_test.cpp

   ```
   #include "kis_some_class_test.h"

   #include <QTest>

   void KisSomeClassTest::test()
   {
   }

   QTEST_MAIN(KisSomeClassTest, GUI)</syntaxhighlight>
   ```

2. Modify ./image/tests/CMakeLists.txt to include your new test class:

   ```
   # ...
   ########### next target ###############
   ```

```
set(kis_some_class_test_SRCS kis_some_class_test.cpp )
ecm_add_tests(${kis_some_class_test_SRCS}
NAME_PREFIX "libs-somelib-"
LINK_LIBRARIES kritaimage Qt5::Test)
# ...
```

3. Write your test. You can use any macro commands provided by Qt (QVERIFY, QCOMPARE or QBENCHMARK).

```
void KisSomeClassTest::test()
{
    QString cat("cat");
    QString dog("dog");

    QVERIFY(cat != dog);
    QCOMPARE(cat, "cat");
}
```

4. Run your test by running an executable in ./image/test/ folder

Krita-specific testing utils

Fetching reference images

All the testing files/images are usually stored in the test's data folder (e.g. ./krita/image/tests/data/). But there are some files which are used throughout all the unit tests. These files are stored in the global folder ./krita/sdk/tests/data/. If you want to access any file, just use `TestUtil::fetchDataFileLazy`. It first searches the file in the local test's folder and if nothing is found checks the global folder.

Example:

```
QImage refImage(TestUtil::fetchDataFileLazy("lena.png"));
QVERIFY(!refImage.isNull());
```

Compare test result against a reference QImage

There are two helper functions to compare a given QImage against an image saved in the data folder.

```
bool TestUtil::checkQImage(const QImage &image, const QString &te
                           const QString &prefix, const QString &
                           int fuzzy = 0, int fuzzyAlpha = -1, in
bool TestUtil::checkQImageExternal(const QImage &image, const QSt
                                   const QString &prefix, const Q
                                   int fuzzy = 0, int fuzzyAlpha
```

The functions search for a PNG file with path

```
./tests/data/<testName>/<prefix>/<prefix>_<name>.png
# or without a subfolder
./tests/data/<testName>/<prefix>_<name>.png
```

The supplied QImage is compared against the saved PNG, and the result is returned to the caller. If the images do not coincide, two images are dumped into the current directory: one with actual result and another with what is expected.

The second version of the function is different. It searches the image in “an external repository”. The point is that PNG images occupy quite a lot of space and bloat the repository size. So we decided to put all the images that are big enough (>10KiB) into an external SVN repository. To configure an external test files repository on your computer, please do the following:

1. Checkout the data repository:

   ```
   # create the tests data folder and enter it
   mkdir ~/testsdata
   cd ~/testsdata

   # checkout the extra repository
   svn checkout svn+ssh://svn@svn.kde.org/home/kde/trunk/tes
   ```

2. Add environment variable pointing to your repository to your ~/.bashrc

   ```
   export KRITA_UNITTESTS_DATA_DIR=
   ~/testsdata/kritatests/unittests
   ```

3. Use `TestUtil::checkQImageExternal` in your unittest and it will fetch data from the external source. If an external repository is not found then

the test is considered “passed”.

QImageBasedTest for complex actions

Sometimes you need to test some complex actions like cropping or transforming the whole image. The main problem of such action is that it should work correctly with any kind of layer or mask, e.g. KisCloneLayer, KisGroupLayer or even KisSelectionMask. To facilitate such complex testing conditions, Krita provides a special class `QImageBasedTest`. It helps you to create a really complex image and check the contents of its layers. You can find the best example of its usage in `KisProcessingsTest`. Basically, to use this class, one should derive it’s own testing class from it, and call a set of callbacks, which do all the work. Let’s consider the code from KisProcessingsTest:

```
// override QImageBasedTest class
class BaseProcessingTest : public TestUtil::QImageBasedTest
{
public:
    BaseProcessingTest()
        : QImageBasedTest("processings")
    {
    }

    // The method is called by test cases. If the test fails, a s
    // is saved into working directory
    void test(const QString &testname, KisProcessingVisitorSP vis

        // create an image and regenerate its projection
        KisSurrogateUndoStore *undoStore = new KisSurrogateUndoSt
        KisImageSP image = createImage(undoStore);
        image->initialRefreshGraph();

        // check if the image is correct before testing anything
        QVERIFY(checkLayersInitial(image));

        // do the action we are trying to test
        KisProcessingApplicator applicator(image, image->root(),
                                           KisProcessingApplicator::

        applicator.applyVisitor(visitor);
        applicator.end();
        image->waitForDone();
```

```
        // check the result, and dump images if something went wr
        QVERIFY(checkLayers(image, testname));

        // Check if undo(!) works correctly
        undoStore->undo();
        image->waitForDone();

        if (!checkLayersInitial(image)) {
            qWarning() << "NOTE: undo is not completely identical
                    << "to the original image. Falling back to "
                    <<"projection comparison";
            QVERIFY(checkLayersInitialRootOnly(image));
        }
    }
};
```

MaskParent object

`TestUtil::MaskParent` is a simple class that, in its constructor, creates an RGB8 image with a single paint layer, which you can use for further testing. The image and the layer can be accessed as simple member variables.

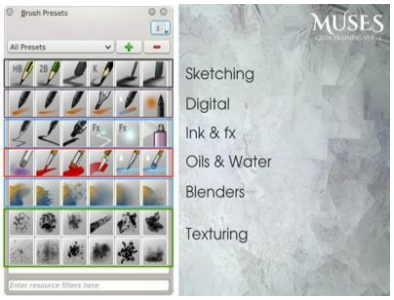
Example:

```
void KisMaskTest::testCreation()
{
    // create an image and a simple layer
    TestUtil::MaskParent p;

    // create a mask and attach its selection to the created laye
    TestMaskSP mask = new TestMask;
    mask->initSelection(p.layer);

    QCOMPARE(mask->extent(), QRect(0,0,512,512));
    QCOMPARE(mask->exactBounds(), QRect(0,0,512,512));
}
```

[1] (1,2,3) https://en.wikipedia.org/wiki/Unit_testing

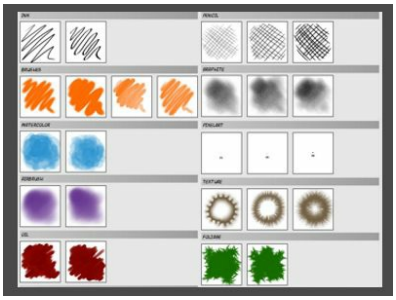
Resources

Brush Packs

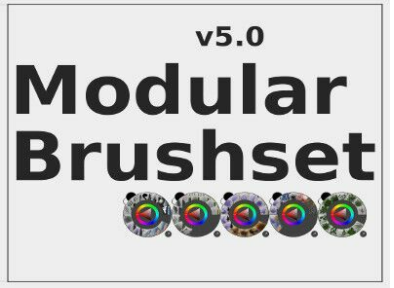


Ramon Miranda

Concept art & Illustration Pack



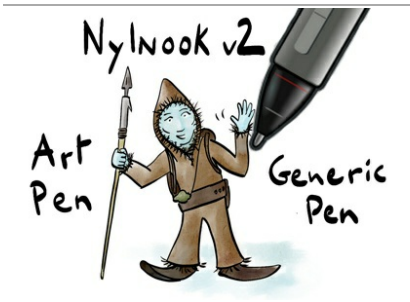
Al-dy



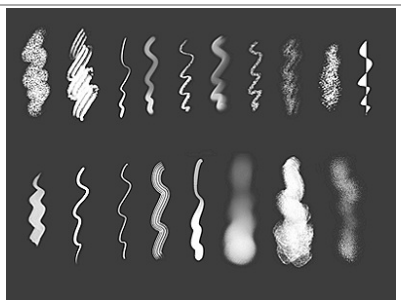
Vasco Basqué

Stalcry

Wolthera



Nylnook



Raghukamath

GDQuest

IForce73

wojtryb

Rakurri

Texture Packs

David Revoy

Vector libraries

Composition templates

Inside the zip archive you'll find all composition templates separate and in a form of a vector library, so all the shapes are easy to access after the import.

The composition templates include: rule of thirds, golden ratio, golden spiral, golden triangle 1, golden triangle 2, harmonious triangle, film safe area template, baroque diagonal and centre.

Link: https://gumroad.com/l/CHhlx

Feather icons

A set of open source icons from https://feathericons.com but in a Vector Library form, so it can be easily accessed from inside Krita after the import. Each icon is designed on a 24x24 grid with an emphasis on simplicity, consistency, and flexibility.

Link: https://github.com/MiAlmeida/krita-feather

Templates

Templates are .kra files that you can base your new documents on. To learn more see Templates:.

Storyboard template

The layout and layer setup is inspired by traditional Studio Ghibli storyboards, and Tony Gaddis' storyboarding process.

Link: https://gumroad.com/l/PtMtm

User-made Python Plugins

This list describes only plugins that are **not** available in Krita, so you need to download and install the ones you'd like to use.

See also

If you want to check descriptions of a plugin available in Krita by default (without a need to download), see Pre-installed Python plugins.

To learn how to install and manage your plugins, see Managing Python plugins.

If you want to know more about an individual plugin, you can access the plugin's manual by going to *Settings* ▸ *Configure Krita…* menu, and then choosing the `Python Plugin Manager` tab. Then you can click on a specific plugin and the manual will appear in the bottom text area.

Caution

Custom Python plugins are made by users of Krita and the Krita team does not guarantee that they work, that they are useful or that they are *safe*. Note that a Python plugin can do everything that Krita can do, which means for example access to your files. Krita team isn't responsible for any damage you might suffer from a custom plugin, this list is informational purposes only and you install any of the custom plugins on your own risk.

If you have information that any of the plugins below is dangerous for the user, please contact Krita team on *kimageshop@kde.org*.

Usability

Direct Eraser

Plugin to switch to an eraser preset and back using one shortcut.

https://www.mediafire.com/file/sotzc2keogz0bor/Krita+Direct+Eraser+Plu

ThreeSlots

This plugin creates three brushtool shortcuts that remembers last used brush preset for each slot independently from each other. It also remembers the size of the brush. One of the slots is for the eraser and it has the eraser mode permanently turned on, while the other two slots have it turned off.

https://github.com/DarkDefender/threeslots

QuickColor

Plugin that adds actions to switch the foreground color to a desired color from a specified palette. The number of actions, which means colors as well, is limited.

https://github.com/JonasLW/QuickColor

BrushColorSwitch

This plugin adds an action/shortcut to switch both a brush and foreground/background color at once.

https://github.com/rkspsm/BrushColorSwitch

Tablet Controls Docker (TabUI)

https://github.com/tokyogeometry/tabui

On-screen Canvas Shortcuts

Plugin that adds an onscreen button bar with shortcuts for Krita.

https://github.com/qeshi/henriks-onscreen-krita-shortcut-buttons/tree/master/henriks_krita_buttons

Workflow improvements

AnimLayers (Animate with Layers)

With this plugin you can animate a specific range of layers by prefixing the layer name with the same letters. Then in the AnimLayers dialog you can enter the prefix in the *Key* field.

https://github.com/thomaslynge/krita-plugins

Reference Image Docker (old style)

Docker for reference images, modeled after the old Reference Images Docker in Krita. Alternative to Reference Images Tool.

https://github.com/antoine-roux/krita-plugin-reference

Mirror Fix

This plugin allows you to correct symmetry errors for example after a transformation of a part of the image.

https://github.com/EyeOdin/mirror_fix

ToggleRefLayer

This plugin lets you assign a keyboard shortcut to toggle the visibility of a reference layer named “reference”.

https://drive.google.com/file/d/11O8FiejleajsT_uHd4Q4VBrCrYX9Rh5v/view?usp=sharing

Shotgun Toolkit Engine for Krita

This plugin allows working in a managed way, loading/saving/publishing artwork, keeping it up to date and publishing your projects and layers into Shotgun Toolkit Engine.

https://github.com/diegogarciahuerta/tk-krita

Photobash Images Docker

Simple Krita Plugin that lists the images you have on a folder you specify, with the ability to filter by words in the path. After setting the references directory in *Photobash Images* docker you can:

- Filter images by words. Using multiple words like “rock marble” will show all the images that have rock OR marble in the name.
- Scroll the pages to access more results.
- Click on an image to create a layer, with the scale that you specify.

https://github.com/veryprofessionaldodo/Krita-Photobash-Images-Plugin

File management

Art Revision Control (using GIT)

> This plugin helps managing multiple versions of the artwork.
>
> https://github.com/abeimler/krita-plugin-durra

Spine File Format Export

> This plugin exports the document in a format compatible with Spine. The README describes what kind of structure the document needs to have to be exported properly. Besides exported images, the plugin creates `spine.json` file.
>
> https://github.com/chartinger/krita-unofficial-spine-export

Color selectors

Pigment.O - Color Picker

> Universal advanced color picker.
>
> https://github.com/EyeOdin/Pigment.O

Interface

UI Redesign

> Plugin that modifies the overall look and feel of the Krita UI Interface. Enables the Toolbox and Tool Options to be toggled, similarly to Blender's UI.
>
> Features a flat theme that can be seen in the repository's README.md section. To give feedback, either create an issue, or join the discussion the thread on Krita Artists.
>
> Krita Artists Thread: https://krita-artists.org/t/call-for-krita-ui-redesign-

plugin-testers/9604

Repository Link: https://github.com/veryprofessionaldodo/Krita-UI-Redesign

Subwindow Organizer

Helps with handling multiple documents in subwindow mode. Introduces responsive fullscreen with other subwindows opened, dynamic snapping of subwindows to canvas borders, drag and drop switching between subwindows, and more.

https://github.com/wojtryb/kritaSubwindowOrganizer

KanvasBuddy

This is a small dialog that floats on top of the canvas packed with enough features to let you spend as much time in Canvas-Only mode as possible. The idea behind KB was to provide the 20% of tools used 80% of the time in the most out-of-the-way GUI possible.

Source, main page and download link: https://github.com/Kapyia/KanvasBuddy

Krita-artists thread: https://krita-artists.org/t/kanvasbuddy-a-minimalist-toolbar/549

Miscellaneous

Timer Watch - Time Management Tool

This plugin adds a timer. You can start it, stop, pause when you want to take a break and restart afterwards. You can set up an alarm to remind you to take a break.

https://github.com/EyeOdin/timer_watch

Post images on Mastodon

With this plugin you can post images on Mastodon from inside of Krita.

https://github.com/spaceottercode/kritatoot

Bash Action (works with OSX and Linux)

Plugin that allows you execute Bash commands and programs as actions on your current Krita images.

https://github.com/juancarlospaco/krita-plugin-bashactions#krita-plugin-bashactions

Other resources

Krita Plugin Generator

An extension to VSCode that generates a Plugin Template for Krita (like *Krita Script Starter*, but directly in VSCode).

Available here: https://github.com/cg-cnu/vscode-krita-plugin-generator

Python auto-complete for text editors

If you have the Krita source code, you can use this Python script to generate the auto-complete file for Python. Many Python editors need a `.py` file to read for auto-complete information. This script reads the C++ header files from Krita's source code and creates a Python file that can be used for auto-completion.

Available here: https://github.com/scottpetrovic/krita-python-auto-complete

External tutorials

Simón Sanchez' "Learn to Create Pixel Art from Zero" course on Udemy

---

See Something We Missed?

Have a resource you made and want to share it with other artists? Let us know in the forum or visit our chat room to discuss getting the resource added to here.

Note

We have curated a list of community created resources for Krita. These resources will be hosted on external website, which is not under the control of Krita or KDE. Please report any error or corrections in the content to the Krita developers.

Index

Symbols

~

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

Krita Brush-tips is an archive of brush-modification tutorials done by the krita-foundation.tumblr.com account based on user requests.

Topics: