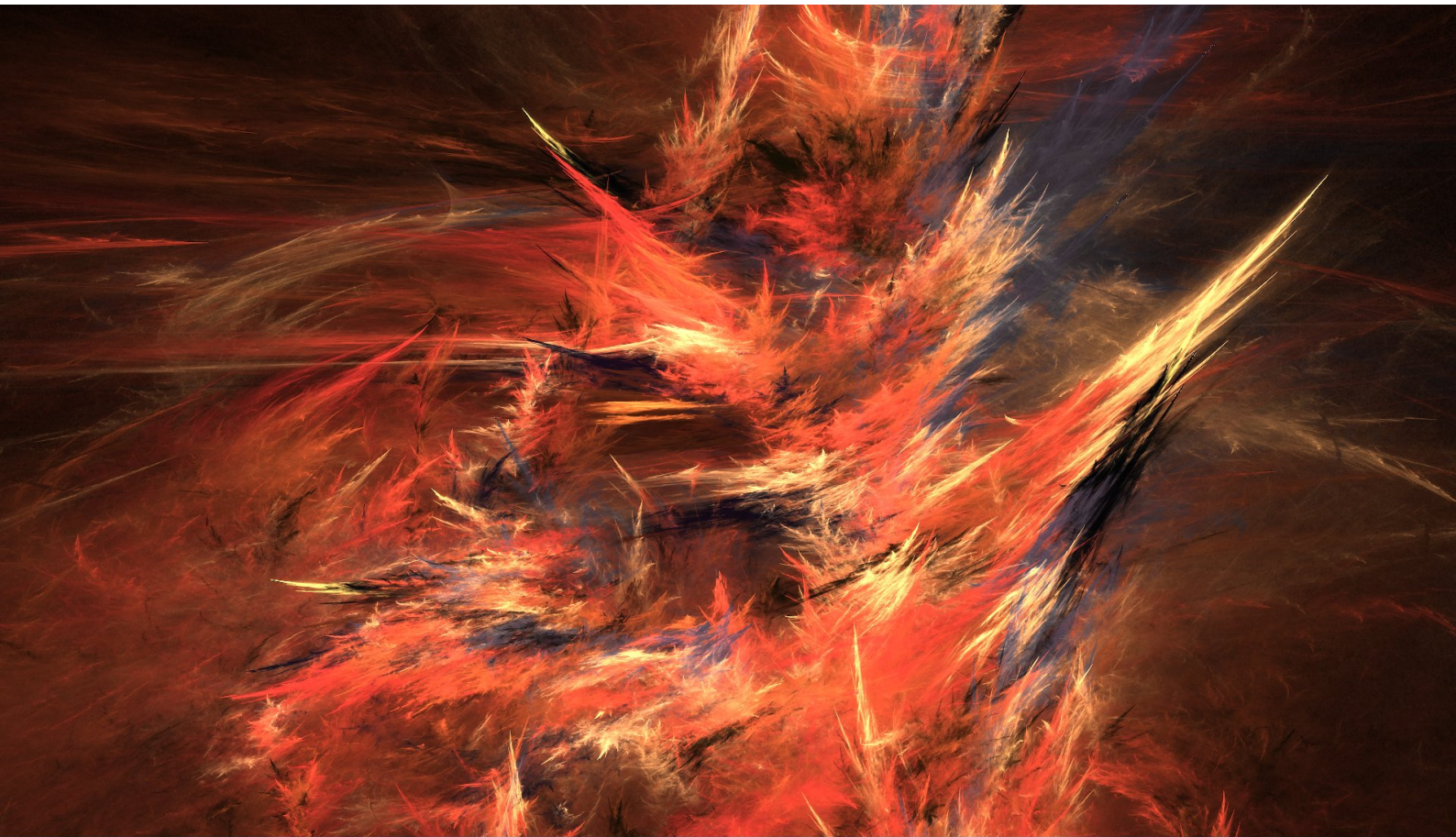




JWildfireSwan users manual

Copyright (C) 2022 - 2025 by Andreas Maschke

Version 1.0.0

Table of Contents

1. Introduction	1
1.1. Why two separate applications? Is <i>JWildfire</i> dead now?	2
1.1.1. Single frames vs. animations	2
1.1.2. GPU computing	2
1.1.3. Steam Deck and music	2
1.1.4. Further development	3
1.2. Key Features of <i>JWildfireSwan</i>	3
2. Launching the application and main navigation	5
3. flame editor	7
3.1. core concepts	7
3.2. flame tree-view	7
3.3. attribute-view	8
3.3.1. various ways for editing attributes	9
3.4. secondary preview	9
3.4.1. affine transforms ("triangles") view	10
3.4.2. variation preview	11
3.5. main preview	11
3.5.1. undo	12
3.6. variation browser	12
3.7. custom variations	13
3.7.1. creating custom variations	13
3.7.1.1. [PARAM_DEFS]-section: define variation parameters	14
3.7.1.2. [VARIATION_CODE]-section: the actual custom shader code	14
3.7.1.3. [INV_VARIATION_CODE]-section: the custom shader code for the inverse transformation of pre-post-variations	14
3.7.1.4. [CUSTOM_FUNCTIONS]-section: define custom functions	15
3.7.1.5. [FUNC_DEPENDENCIES]-section: re-use existing functions	15
3.7.1.6. complete example with parameters (post_rblur variation)	15
3.8. gradient library	20
3.8.1. save the current gradient	20
3.9. gradient editor	20
3.9.1. color models	21
3.9.2. converting legacy gradients into color-curves	21
3.9.3. useful functions	21
3.9.3.1. replicate	21
3.9.3.2. predefined curves	21
3.9.3.3. blur	22
4. mutagen	23

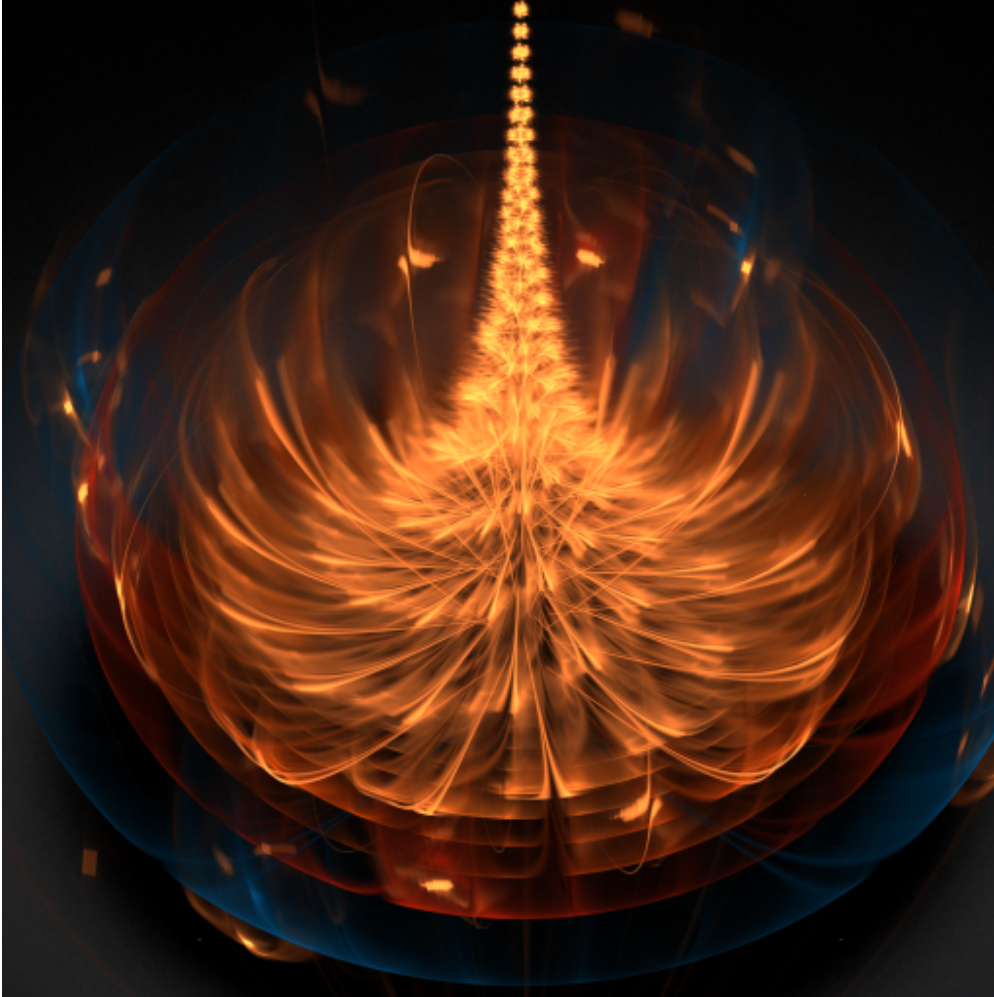
4.1. boost brightness	24
4.2. flame meta information	24
4.3. computation time	25
5. ai img2img	26
5.1. Requirements	27
5.2. Loading flames into ai img2img :	28
5.3. Generating images using AI :	29
5.3.1. Saving generated images.	29
5.3.2. Refine generated images / Re-use settings	29
5.4. ai img2img image generation options:	30
5.5. Installation of <i>AUTOMATIC1111</i> :	35
6. flame library	37
6.1. folder view	37
6.2. flames view	37
6.3. <i>flame</i> meta-data.	38
7. key-frame-based animations	39
8. working with random flames	40
8.1. creating random batches	40
9. jukebox	41
9.1. list of flames.	41
9.2. randomize motion	42
9.3. list of audio files	42
9.4. getting started / picking random flames	42
9.5. motion blur	42
9.6. optimize (render) settings	42
9.7. play single file or all files	43
9.8. fullscreen mode	43
9.9. general tips.	43
10. application settings	45
11. application info.	46
12. integration of <i>ffmpeg</i> to encode videos	47
12.1. Installation of <i>ffmpeg</i>	47
12.1.1. Windows	47
12.1.2. Mac OS	47
12.2. Setting the path to <i>ffmpeg</i> in the application settings	47
12.3. extra encoding settings	47

Chapter 1. Introduction

Welcome to *JWildfireSwan* and this user manual!

Great importance was attached to explaining things as briefly but clearly as possible.

If this is not the case in individual cases, please feel free to contact me and I will endeavour to improve the instructions.



JWildfireSwan (“*Swan*” for short) is the successor to *JWildfire*, a very sophisticated flame fractal generator whose roots go back some 25 years. While *JWildfire* is something like the Swiss army knife of *flame fractals*, *Swan* is a much more modern and streamlined application that fully supports GPU rendering.

Swan not only renders much faster, but also produces images with much better quality and much more fractal detail, as the rendering algorithm has been redefined from the ground up to not compromise on quality.

Also, fractal animation is a first-class citizen in *Swan*, which means that there is now a realtime animation player that allows you to preview the animation in real time. And, all random flames generated by the app are animated by default.

It is powered by the Godot engine and is truly cross-platform. No matter whether you are working under Linux, Windows, ... you will always have the same experience and always have access to

GPU rendering.

The application has a lean design. Therefore, it does not support all the functions you may be familiar with from *JWildfire*. This is to make the user interface simpler, more responsive and less complex. But you can assume that the essential functions are available. And if you are missing something, you are welcome to submit a feature request.

1.1. Why two separate applications? Is *JWildfire* dead now?

1.1.1. Single frames vs. animations

Although both applications deal with flame fractals, they are completely different and do not have a single line of code in common.

In short, *JWildfire* is like "Photoshop for flame fractals", while *JWildfireSwan* is more like "Adobe Premiere for flame fractals".

JWildfireSwan finally brings flame fractals to life, i.e. they are animated by default, which has been a long time dream of mine. And it really is game-changing in the user experience, even if you don't want to create animations. There's so much more to discover when things are in motion.

This requires a completely different approach (GPU computing) and cannot be done in *JWildfire*. Of course, you can also create animations in *JWildfire*, but this is more like batch processing in Photoshop, you have to render frame by frame and then assemble the result, which can be a tedious process.

JWildfire is still very useful if you want to create large artwork for print. This is possible because it uses CPU computing (so you can use more memory to render images at higher resolutions), and it has more options to craft a single image.

So *JWildfire* is better suited for creating single large images, while *JWildfireSwan* is suited for creating a large number of smaller images (animations).

1.1.2. GPU computing

Actually, there is also GPU computing in *JWildfire*, but since it is a command line tool, it is very limited in doing realtime renders (only a frame at a time) and only works for CUDA devices. Again, it's a big change to be able to just "play" a fractal animation instead of creating a series of frames and then encoding a *.mp video and watching it. Hence, the comparison with Adobe Photoshop and Premiere).

1.1.3. Steam Deck and music

I also plan on integrating sound or music into *JWildfireSwan* and good support for *Steam Deck*, which is also a longtime dream of mine. I think it will be super cool to be able to control the fractals with the Steam Deck controller. This is also not possible with the *JWildfire* app, but in *JWildfireSwan* you can do this very well thanks to the *Godot engine*.

The target groups of both apps are therefore also little different.

While *JWildfire* is more for those interested in math or art, *JWildfireSwan* has a much more playful approach and is also aimed at VJs. I know there are already VJ's using *JWildfire*, but it kinda sucks because it can take forever to calculate the animations, and the animations can't respond nicely to music (in realtime or almost realtime).

1.1.4. Further development

So both applications make sense, and I will continue to develop them both. *JWildfire* will receive fewer updates because it already has a large number of functions. But it is not "dead".

JWildfireSwan, on the other hand, will never get all the features of *JWildfire*. Some of them will not work on the GPU, others are very specific. *JWildfireSwan* should remain an app that feels light and playful, but is still powerful, with far fewer bells and whistles on the user interface.

In the long term, the two apps will therefore not converge, but rather diverge. At the current stage of development, there are of course some common features, such as an editor for the fractals. But they are already very different, even if they seem to deliver very similar-looking results.

1.2. Key Features of *JWildfireSwan*

- real-time preview of changes to the flame attributes
- excellent support for fractal animations (all random-flames are animations by default), also for looped animations
- really fast GPU rendering on all platforms (no more differences between CPU and GPU rendering, as there is now only GPU rendering)
- drastically improved render quality/degree of detail
- you will notice a much lower system load when rendering (unlike *JWildfire*, which sometimes feels like it is taking over your system).
- compatibility/interchangeability with *JWildfire* (you can load most of your existing flames)
- around 400 variants are currently supported, with more in the pipeline
- integrated sophisticated random flame and random gradient generators to play effortlessly with endless flames
- powerful flame editor that allows you to edit existing flames or create new flames from scratch and see the changes in near real time
- support for key-framed animation with motion curve editor
- jukebox-module for audio-spectrum-based animations of flame fractals in real time
- variation browser that interactively displays the effect of a variation and helps to understand how it works
- integrated flame-library with hundreds of curated examples created by the author of the software
- powerful curve-based gradient-editor

- integrated gradient-library with numerous of pre-defined gradient examples
- cross-platform (Linux, Windows, macOS)
- **mutagen** module to generate new flames by mutating existing flames
- **ai img2img** module to generate new images based on a flame fractal using **AI**

I hope you enjoy using Swan as much as I enjoyed creating it.

Andreas Maschke, Grambek (Germany) 2025

Chapter 2. Launching the application and main navigation

The application looks the same on all supported platforms.

Most of the space on the screen is taken up by one of the various program modules, which you can switch between using the toolbar on the right-hand side of the window.

To make navigation as easy as possible, there are no menus that hide functions.



You can access the following modules via the navigation toolbar on the right-hand side:

- **flame editor**: the main module of the application. Here you can create and edit *flame fractals*.
- **mutagen**: another sub-module of the **flame editor**, to generate flames no the basis of existing flames by applying mutations to them.

- **ai img2img**: a sub-module of the **flame editor**, to generate **ai-images** using the **Stable Diffusion** family of models, based on both a fractal image and the structure of the fractal itself.
- **flame library**: a large collection of included examples *flames* as well as the space to save your own creations.
- **<jukebox***: a module that allows you to create audio-spectrum-based animations of flame fractals in real time
- **application settings**: Here you can customize various application settings according to your wishes.
- **application info**: Displays some useful information about the application.

You can also access your **flame folder** and **image folder** at any time via the main toolbar on the right.

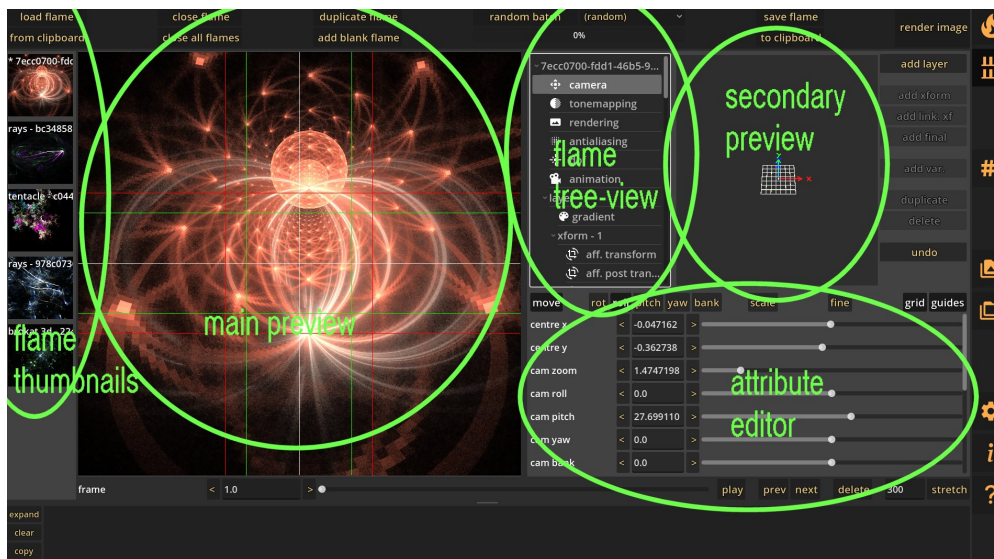
Chapter 3. flame editor

The **flame editor** is the heart of *JWildfireSwan*. Here you can edit existing *flame fractals* (“*flames*” for short) or create new *flames* from scratch. The most common use case is to modify (“tweak”) existing *flames*, as it is often not easy to achieve a specific result due to the nature and complexity of the *flame fractal* generation algorithm.

But if you have mathematical understanding or a lot of experience, you can create exactly the shapes or artwork you can imagine.

However, this is not always desirable. Often it’s just fun to start with something and play around with it to develop something completely different that you couldn’t have imagined before. This is why fractal art can sometimes also be described as the “art of chance”.

3.1. core concepts



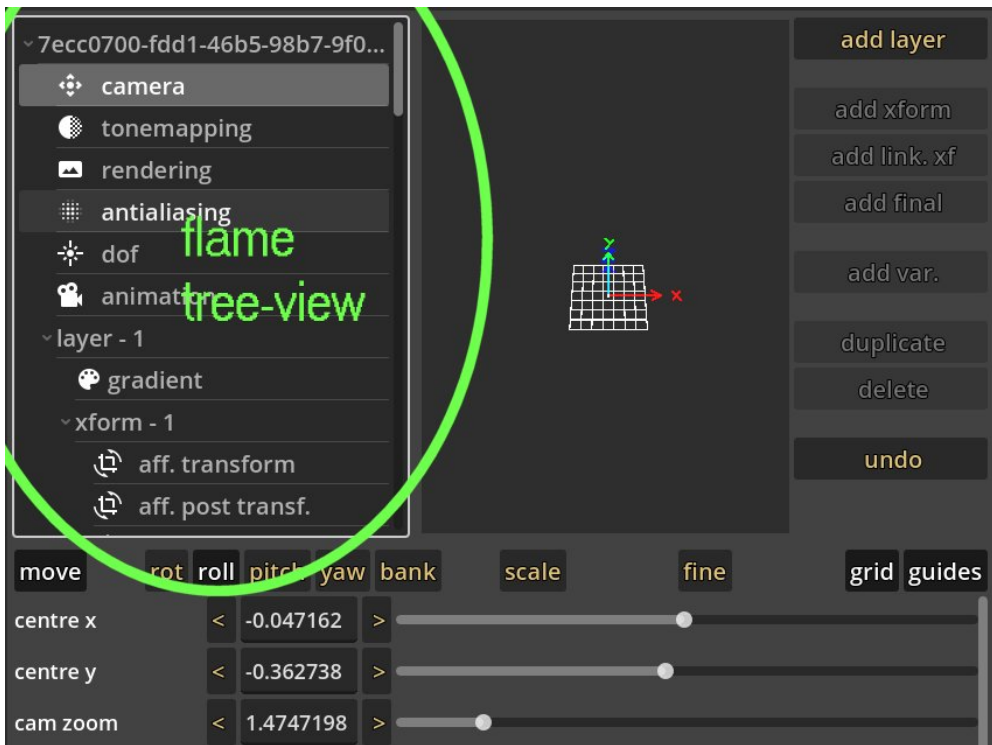
Since *flame fractals* are already complex in themselves, great importance was attached to making the user interface as clear and simple as possible.

Flame fractals have a hierarchical structure (similar to a tree that has a trunk, branches, twigs, etc.). They consist of basic attributes such as the camera viewing angle and can contain any number of *layers*.

Each *layer* in turn consists of basic attributes, a color gradient and any number of *transformations*, often also called “*xforms*”.

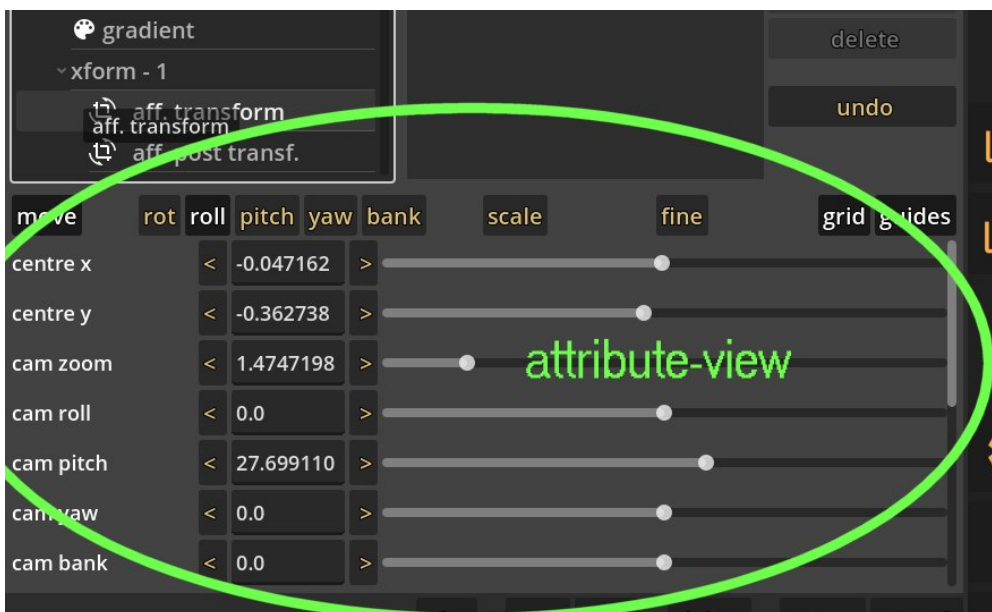
A *transformation* in turn has basic attributes and contains any number of so-called *variations*, which are (usually non-linear) functions that contribute to a *transformation*.

3.2. flame tree-view



Since a flame has a tree-like structure, the main component for displaying the structure of a *flame* is a tree view.

3.3. attribute-view



If you click on a node within the flame tree-view, a detailed view of this node is displayed. Here you can edit the attributes of this node.

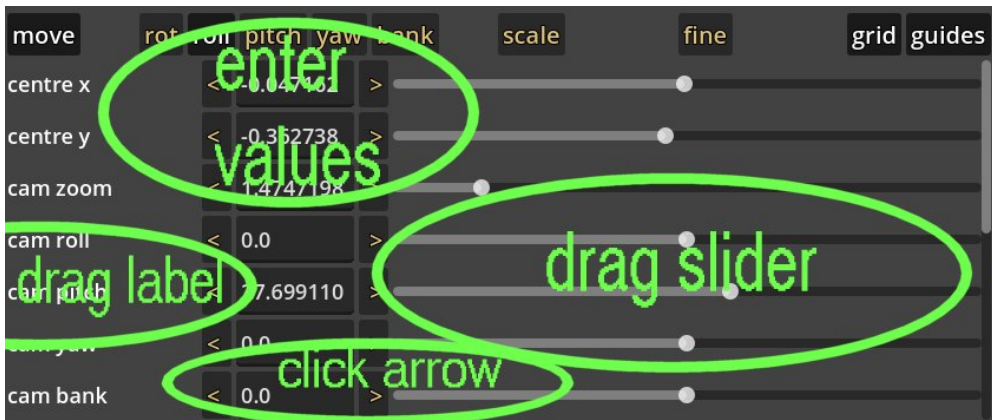
To make editing easier, some attributes are displayed in groups (as separate nodes in the *flame* tree view, even if they are not separate nodes of a *flame*).

For example:

- **camera:** all attributes that change the camera.

- **tonemapping:** tone mapping/color-related attributes.
- **animation:** all attributes relating to the animation.
- etc.

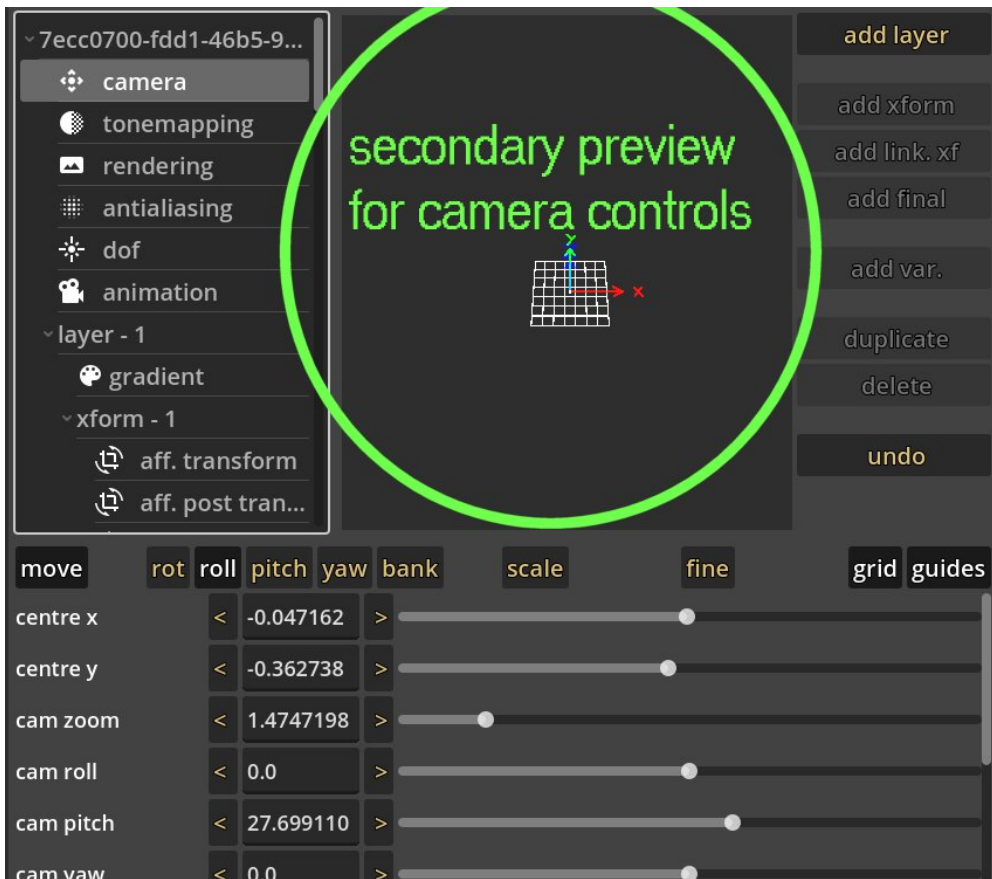
3.3.1. various ways for editing attributes



Because the precise editing of the *flame* attributes is very important, there are several ways to edit them:

- **direct numerical input:** You can enter a value directly in the input field.
- **slider:** You can change the value using the slider.
- **Drag with the mouse over the label of the attribute:** Works in a similar way to the slider, but often gives more direct feedback and works on a finer scale.
- **arrow buttons:** You can use the arrow buttons to increase or decrease the value. You can press and hold the button to change the value continuously.

3.4. secondary preview

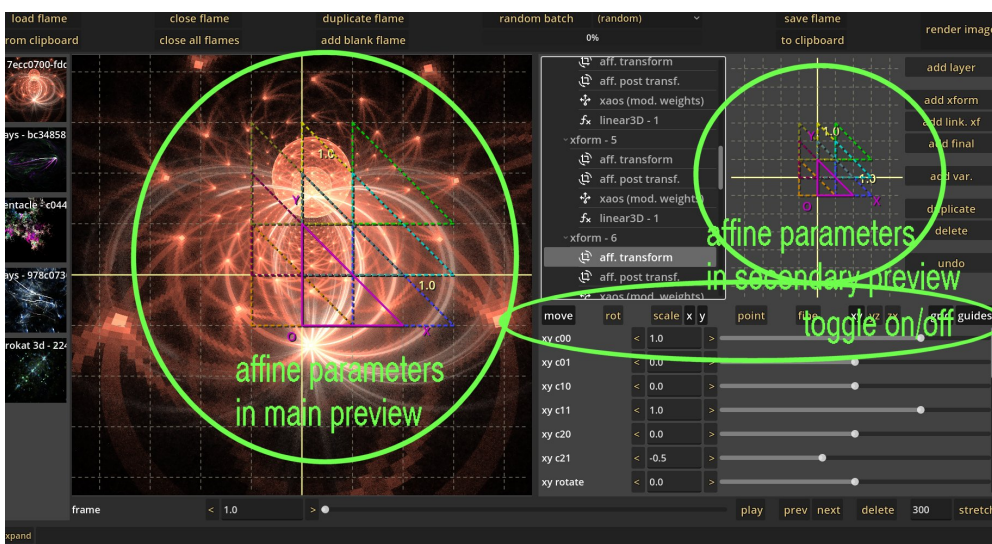


In some cases (depending on the node type), a special preview is also displayed to help you both understand the effect of the node and change its attributes.

There are currently the following types of secondary previews:

- **camera preview:** This preview shows the effects of the camera attributes.
- **affine transforms ("triangles"):** This preview shows the effect of the affine transformations.
- **variation preview:** This preview shows the effect of a variation. Here you can see directly how the different attributes of a particular variation work and play with them. You can also enlarge or reduce the view by dragging the mouse into the view.

3.4.1. affine transforms ("triangles") view

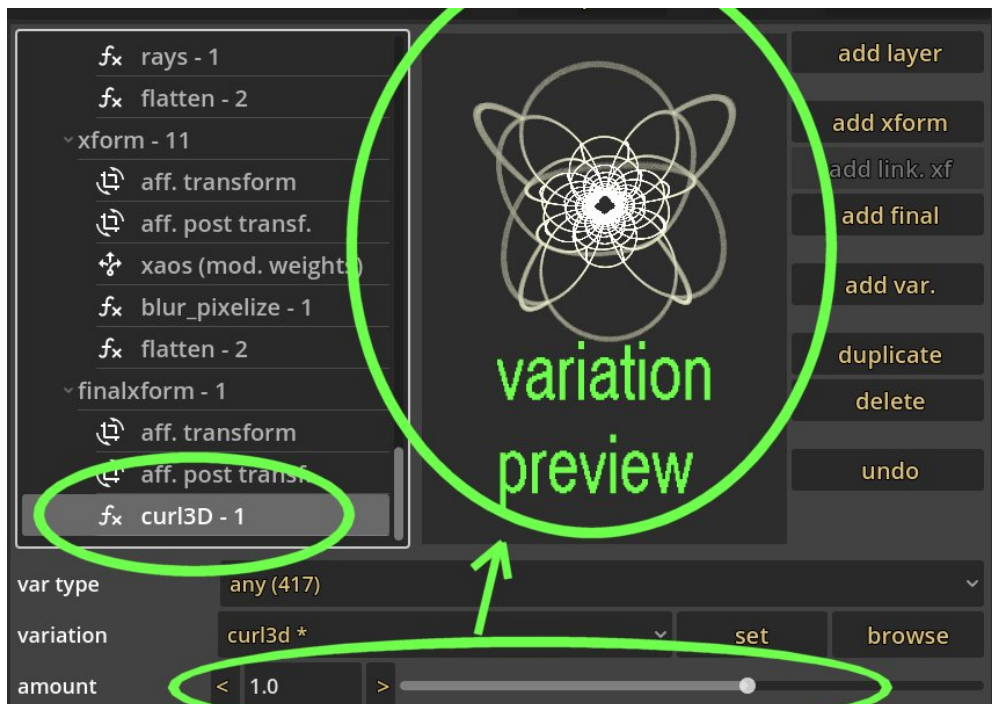


This preview shows the effect of the affine transformations. Depending on the selected node in the *Flame* tree view, either the effect of the regular affine transformation or the post-affine transformation is displayed.

In this view, you can also select a specific transformation by clicking on its triangle representation. You can then click on a triangle to select the next triangle if they overlap.

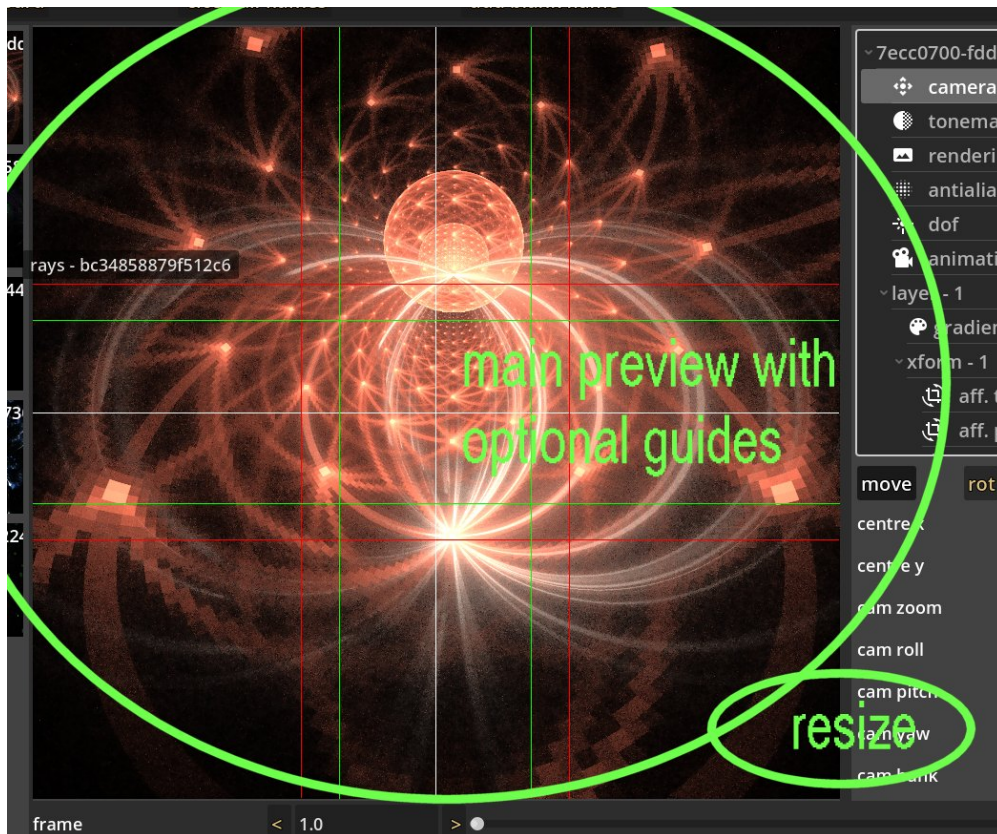
You can move, rotate or scale the triangles to influence the actual affine transformation represented by the selected triangle.

3.4.2. variation preview



This preview shows the effect of a variation. Here you can see directly how the different attributes of a particular variation work and play with them. You can also enlarge or reduce the view by dragging the mouse into the view.

3.5. main preview

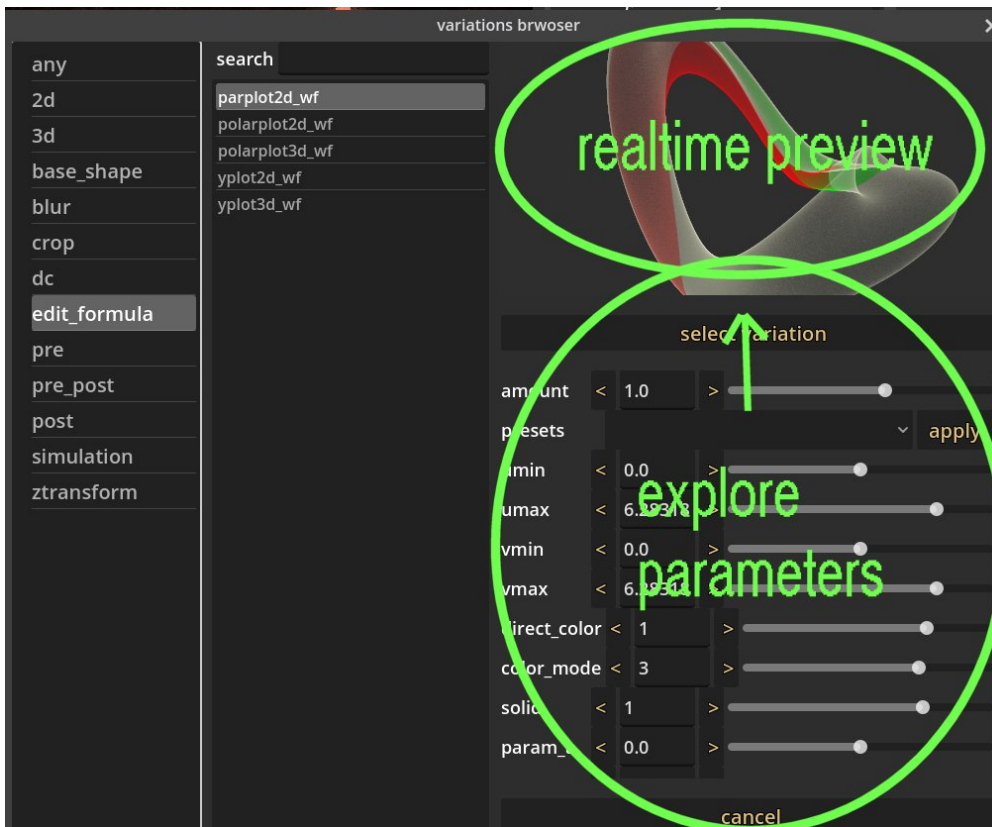


This is the main preview where all the changes you make to the flame attributes are displayed, many of them almost in real time.

3.5.1. undo

You can undo your last changes by pressing 'Ctrl-Z' or 'Cmd-Z' on your keyboard if you are not satisfied with them. If you have changed several attributes at once (e.g. moving the camera position in the x and y directions), all these changes will be undone.

3.6. variation browser



The **variation Browser** helps you to explore and learn the huge number of *variations* and their parameters. It also displays all changes to parameters in real time, but does not apply them to the currently edited flame.

3.7. custom variations

You can create your own custom variations in glsl shader language, which are compiled on GPU. While they are different from JWildfire custom variations (which are written in Java code) it is also very straight-forward to create them (at least for more simple variations). To help with this, several examples are provided (via variation presets) and there is a dedicated editor-window which shows syntax/compile errors on the fly. To make the custom variation code more easy to handle, it contains several sections all into one file:

- parameter definitions
- the actual glsl code
- custom functions
- function dependencies

3.7.1. creating custom variations

To create a custom variation, use one of the variations

- **custom_glsl_code**: for regular variations
- **custom_post_glsl_code**: for post-variations
- **custom_pre_glsl_code**: for pre-variations

- **custom_prepost_gsls_code**: for pre-post-variations

The separation into these four variations is useful because they all work slightly different (e.g. pre-post-variations have two gsls-code-blocks, one for the regular transformation, one for the inverse transformation)

You can either select one of the existing presets or start from your own. In the latter case, it is recommended to start with the "empty"-preset, because this contains the template with all the sections you will need.

3.7.1.1. [PARAM_DEFS]-section: define variation parameters

This section is used to define the parameters of the variation and is optional. Each parameter is defined in a separate line, using the following syntax:

```
[PARAM_DEFS]
name = c1, value = 0.1, type = float, min = -3.0, max = 3.0
name = c2, value = 0.0, type = float, min = -3.0, max = 3.0
```

3.7.1.2. [VARIATION_CODE]-section: the actual custom shader code

This section contains the actual gsls code of the custom variation. Any parameters defined in the [PARAM_DEFS]-section can be used here.

Example:

```
[VARIATION_CODE]
float re = 1.0 + c1 * _tx + c2 * (sqr(_tx) - sqr(_ty));
float im = c1 * _ty + c2 * 2.0 * _tx * _ty;
float r = amount / (sqr(re) + sqr(im));
_vx += (_tx * re + _ty * im) * r;
_vy += (_ty * re - _tx * im) * r;
```

3.7.1.3. [INV_VARIATION_CODE]-section: the custom shader code for the inverse transformation of pre-post-variations

This section is only used for pre-post-variations and contains the inverse transformation of the variation.

```
[INV_VARIATION_CODE]
...
```

To see an actual example, please have a look at the **custom_prepost_gsls_code** - variation and use a preset there.

3.7.1.4. [CUSTOM_FUNCTIONS]-section: define custom functions

In more complex variations, you may want to define custom functions that can be used in the shader code. Also, this code is written in glsl, but must contain complete functions.

Example:

```
[CUSTOM_FUNCTIONS]
float swfunc_log10(float val) {
    return log(val) / 2.30258509299;
}
```

It is recommended to use a prefix for the function name, so that it does not conflict with other existing functions. In this example, the prefix is **swfunc_**.

3.7.1.5. [FUNC_DEPENDENCIES]-section: re-use existing functions

In this section you can specify existing functions that are provided by **JWildfireSwan** and are used by existing built-in variations. They are specified by a name, while the name of the actual function may be different, in order to avoid name conflicts with existing (standard glsl) functions. In some case it is not desirable to use the original function, because it may have limitations, e.g. not working well for certain ranges of arguments.

When you want to use more than one function, you may define them in separate lines or separate them by commas.

Example:

```
[FUNC_DEPENDENCIES]
acosh, cosh
hypot
```

3.7.1.6. complete example with parameters (post_rblur variation)

```

[PARAM_DEFS]
name = strength, value = 1.0, type = float, min = -3.0, max = 3.0
name = offset, value = 1.0, type = float, min = -3.0, max = 3.0
name = center_x, value = 0.0, type = float, min = -3.0, max = 3.0
name = center_y, value = 0.0, type = float, min = -3.0, max = 3.0

[VARIATION_CODE]
float s2 = 2.0 * strength;
float r = sqrt(sqr(_vx - center_x) + sqr(_vy - center_y)) - offset;
r = r < 0.0 ? 0.0 : r;
r *= s2;
_vx = amount * (_vx + (rand8(uv, rngState) - 0.5) * r);
_vy = amount * (_vy + (rand8(uv, rngState) - 0.5) * r);

[INV_VARIATION_CODE]

[CUSTOM_FUNCTIONS]

[FUNC_DEPENDENCIES]

```

3.7.1.6.1. list of available functions

The following functions are available in the **FUNC_DEPENDENCIES** section:

- **acosh**

```
float sw_acosh(float x);
```

- **cosh**

```
float sw_cosh(float x);
```

- **erf**

```
float erf(float x);
```

- **gamma**

```
float lgamma(float x);
```

- **hypot**

```
float hypot (float x, float y);
```

- **j1**

```
float j1(float x);
```

- **jacobi_elliptic**

```
float jacobi_elliptic(float u, float em);
```

- **log10**

```
float sw_log10(float x);
```

- **modulo**

```
int modulo(int a,int b);
```

- **pow**

```
float sw_pow(float x, float b);
```

- **pow_int**

```
float sw_pow_int(float x, int b);
```

- **rint**

```
float sw_rint(float x);
```

- **round**

```
float sw_round(float x);
```

- **safediv**

```
float safediv(float q, float r);
```

- **sgn**

```
float sgn(float x);
```


- **sinh**

```
float sw_sinh(float x);
```

- **single_simplex**

```
float singleSimplex(int seed, float x, float y, float z);
```

- **tanh**

```
float sw_tanh(float x);
```

- **trunc**

```
float sw_trunc(float x);
```

- **lib_complex**

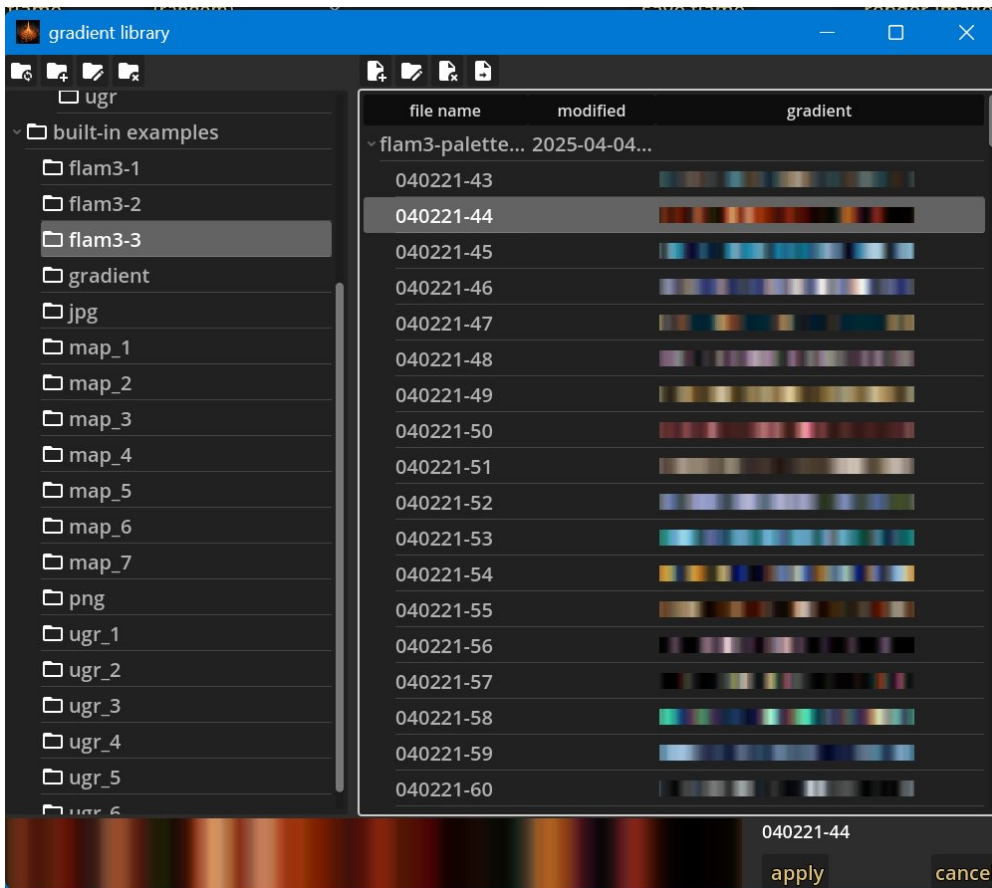
```

struct Complex {
    float per_fix;
    float re;
    float im;
    float save_re;
    float save_im;
};

void Complex_Init(inout Complex c, float Rp, float Ip);
float Complex_Mag2(Complex c);
float Complex_MagInv(Complex c);
void Complex_Recip(inout Complex c);
void Complex_Dec(inout Complex c);
void Complex_Inc(inout Complex c);
void Complex_Neg(inout Complex c);
void Complex_Div(inout Complex c, Complex zz);
void Complex_DivR(inout Complex T, Complex zz);
void Complex_Copy(inout Complex c, Complex zz);
float Complex_Mag2eps(Complex c);
float Complex_Arg(Complex c);
void Complex_Log(inout Complex c);
void Complex_Scale(inout Complex c, float mul);
void Complex_AtanH(inout Complex c);
void Complex_AcotH(inout Complex c);
void Complex_Flip(inout Complex c);
void Complex_Sqr(inout Complex c);
void Complex_Add(inout Complex c, Complex zz);
void Complex_Sub(inout Complex c, Complex zz);
void Complex_Mul(inout Complex c, Complex zz);
void Complex_One(inout Complex c);
void Complex_Conj(inout Complex c);
float Complex_Radius(Complex c);
void Complex_Sqrt(inout Complex c);
void Complex_ToP(Complex c, inout Complex dst);
void Complex_UnP(Complex c, inout Complex dst);
void Complex_Pow(inout Complex c, float exp);
void Complex_AsinH(inout Complex c);
void Complex_AsecH(inout Complex c);
void Complex_Exp(inout Complex c);
void Complex_Acosh(inout Complex c);
void Complex_AcosecH(inout Complex c);
void Complex_Sinh(inout Complex c);
void Complex_Cosh(inout Complex c);
void Complex_Sin(inout Complex c);
void Complex_Cos(inout Complex c);
void Complex_Asin(inout Complex c);
void Complex_Acos(inout Complex c);
void Complex_Atan(inout Complex c);

```

3.8. gradient library



The gradient library is a collection of color gradients that you can use for your flames. It contains a large number of gradients that are included with the application as well as your own gradients. It is therefore very similar to the **flame library**.

You can access the gradient library via the "choose"-button in the gradient section of the flame editor.

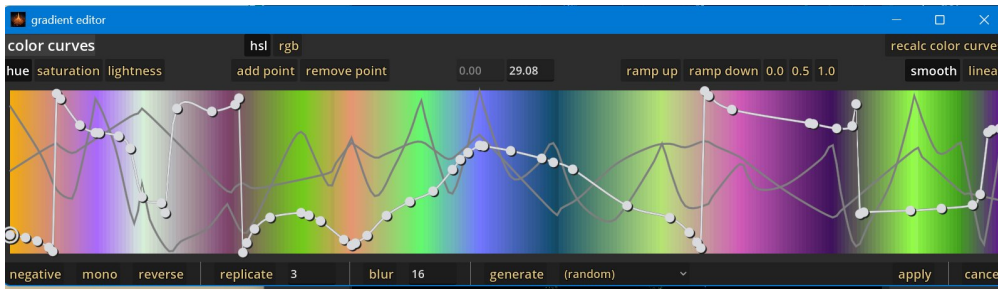
3.8.1. save the current gradient

To save the current gradient, click on the "save gradient" button. Choose a folder inside your gradient library (note your gradient library") and press the "save current gradient" button.



Please make sure that you do not select a different color gradient before you click on the "save current gradient" button. Otherwise, the current color gradient becomes the newly selected color gradient. If this happens by mistake, simply close the color gradient library by pressing the "cancel" button and open it again.

3.9. gradient editor



The **gradient editor** is a powerful tool to create and edit color gradients. It is curve-based and allows you to create very complex gradients with a few mouse clicks.

3.9.1. color models

There are two color models available in the gradient editor: **hsl** and **rgb**. **hsl** is the default color model and is recommended for most use cases. But you can switch to **rgb** (and back to **hsl**) at any time if you prefer to work with it.

3.9.2. converting legacy gradients into color-curves

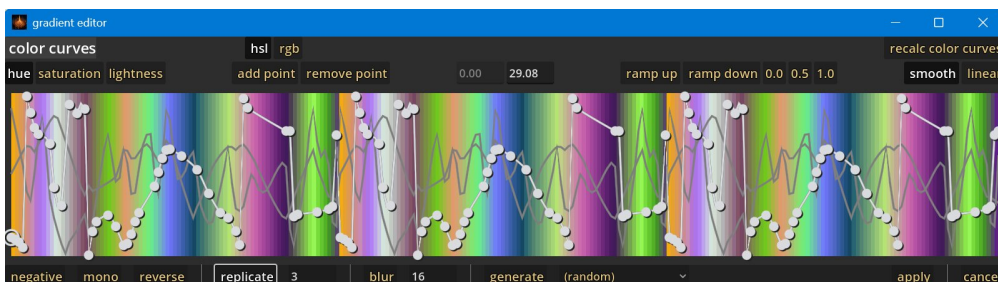
Most legacy gradients (also these in the gradient library) do not contain color curves. So when you edit such a color gradient, the software automatically creates color curves. Depending on the complexity of the gradient, this may take a while. (*Swan* tries to create the simplest possible color curves, which helps later when editing the color curves. the color curves).

3.9.3. useful functions

There are many useful functions in the gradient editor that help you to create very interesting gradients.

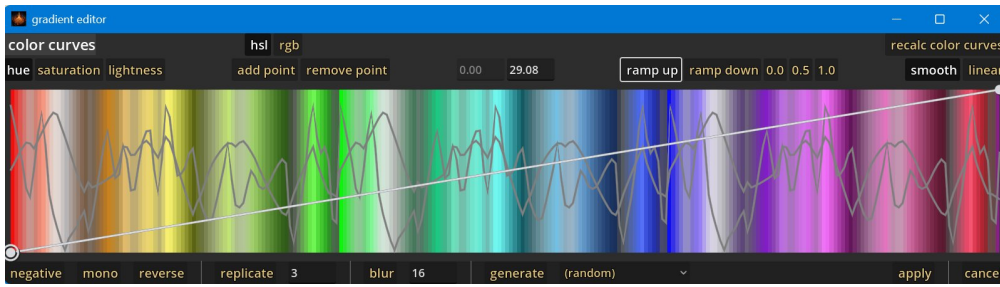
Some to mention:

3.9.3.1. replicate



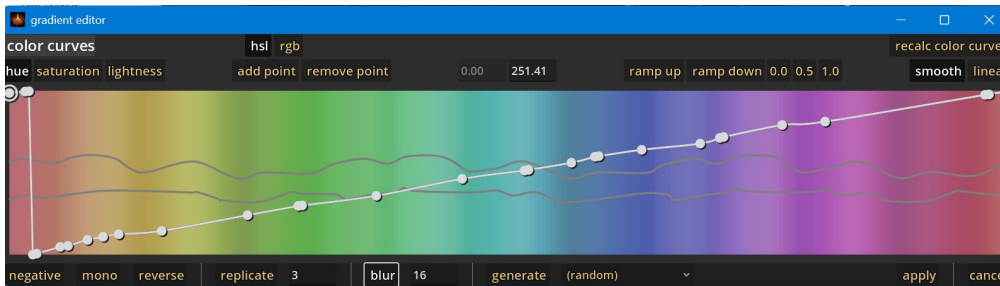
Replicates the current color gradient and creates copies of it at smaller scale.

3.9.3.2. predefined curves



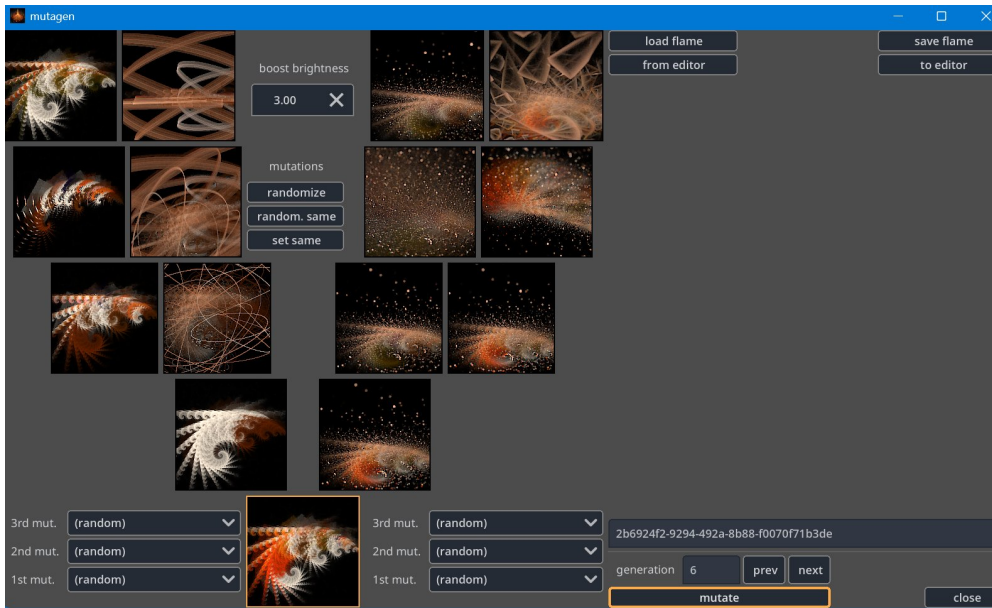
There are some predefined curves (e.g. "ramp up") that you can use to modify the gradient. They are applied only to the currently selected color channel: (hue, saturation, lightness) in the case of the **hsl** color model, or (red, green or blue) in the case of the **rgb** color model.

3.9.3.3. blur



Reduces the contrast of the gradient by applying a blur to it. May be very useful to create smooth gradients in a certain "mood".

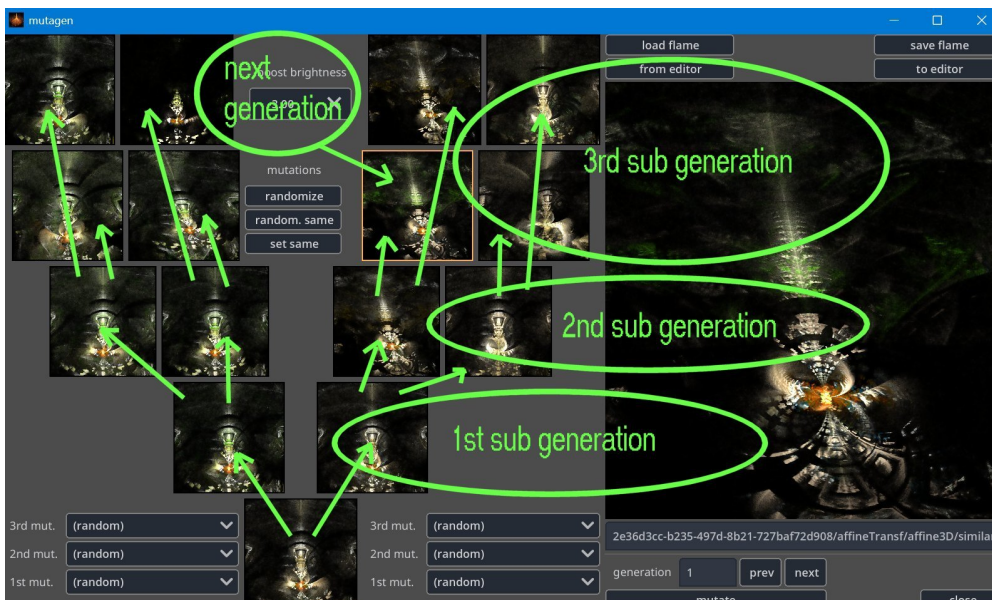
Chapter 4. mutagen



The **mutagen** module creates (endless) variants of existing flames using sophisticated algorithms.

Unlike the similar module in *JWildfire*, **mutagen** in *Swan* has a tree-like structure to the variants created, and you have more control over which mutations are applied:

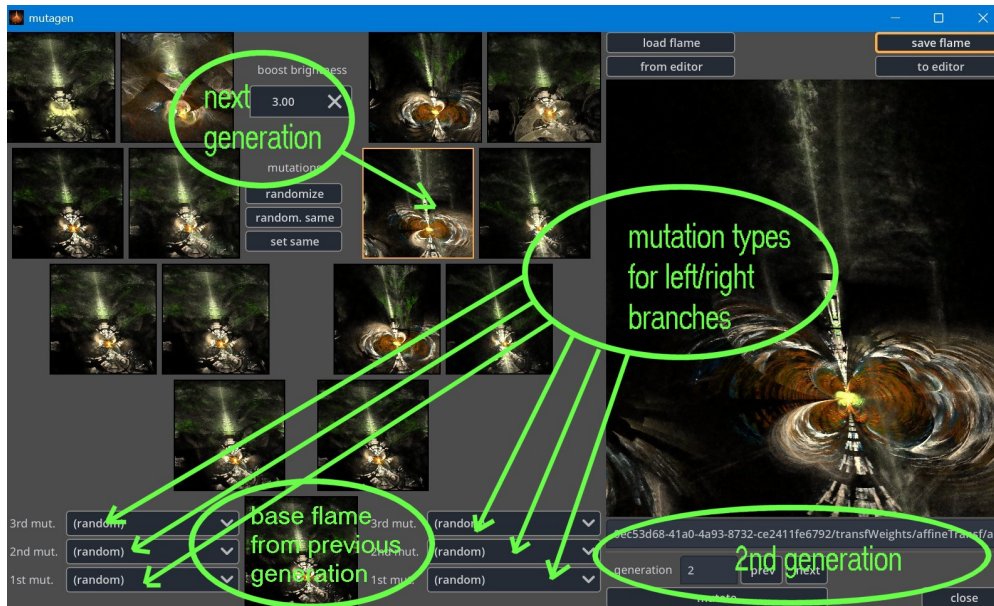
- there are two sets of mutations: left and right. The left set is applied to the left branches of the tree, while the right set is applied to the right branches of the tree.
- each generation (=set of generated flames) consists of three sub-generations. For each of these sub-generations, you can specify the mutation type (for both and the left branch).



If you do not need this level of control, you can keep the default settings or randomize the settings. During generation, you may click at the already generated variations to generate a larger preview or save a flame you like.

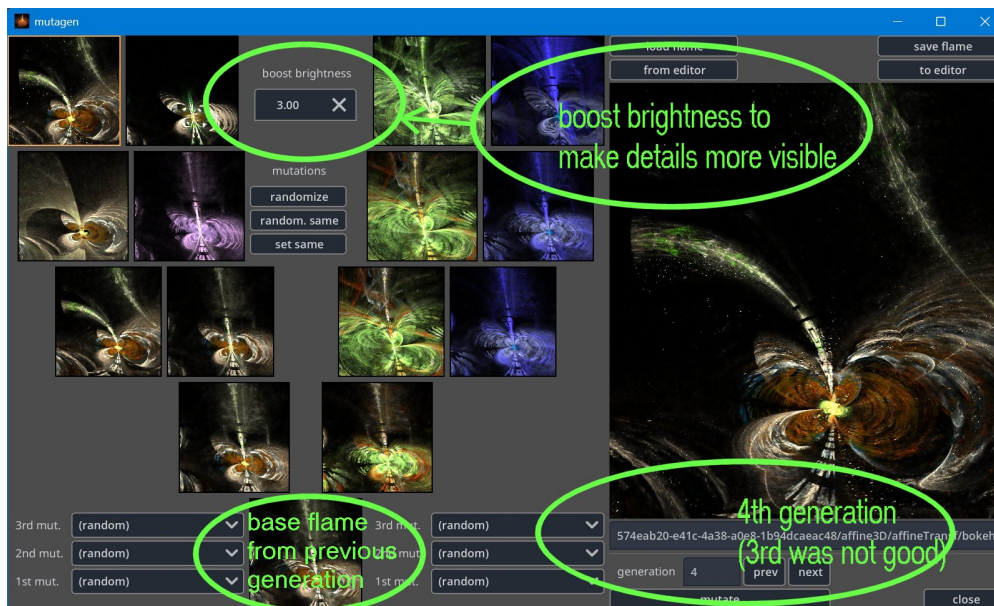
To generate a subsequent generation select any flame you like and press the "mutate"-button again.

You can also switch back to previously generated generations.



4.1. boost brightness

Many flames tend to be rendered using rather dark colors. So, many details are not visible. While this is fine for a final render, fractal details may be interesting in the mutation process. The **boost brightness** option will increase the brightness of the generated previews to make details more visible, but it does not change the actual flame.



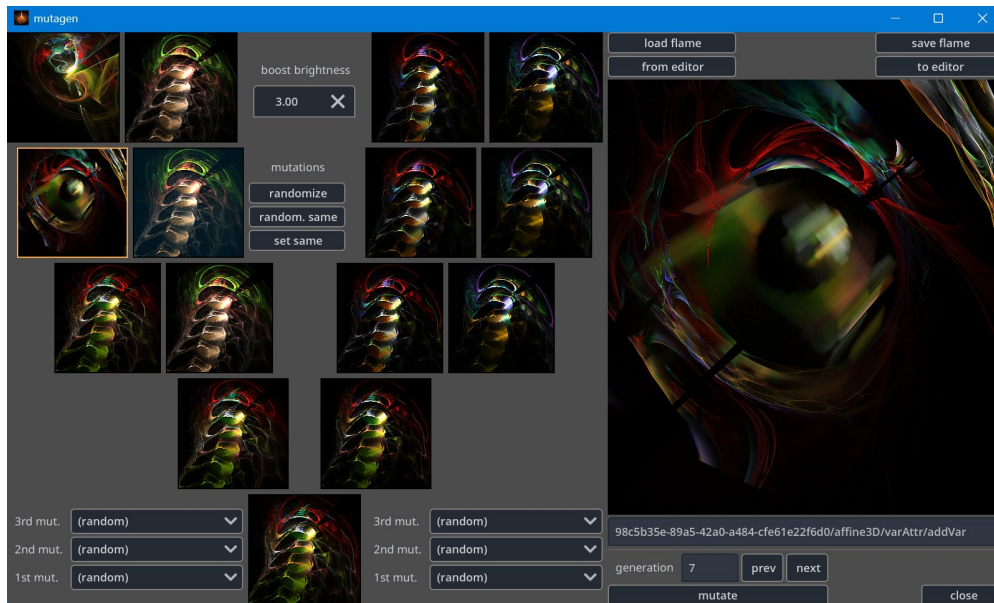
4.2. flame meta information

If you are interested in how a particular flame was generated, you can see the path of the generation in the meta information:

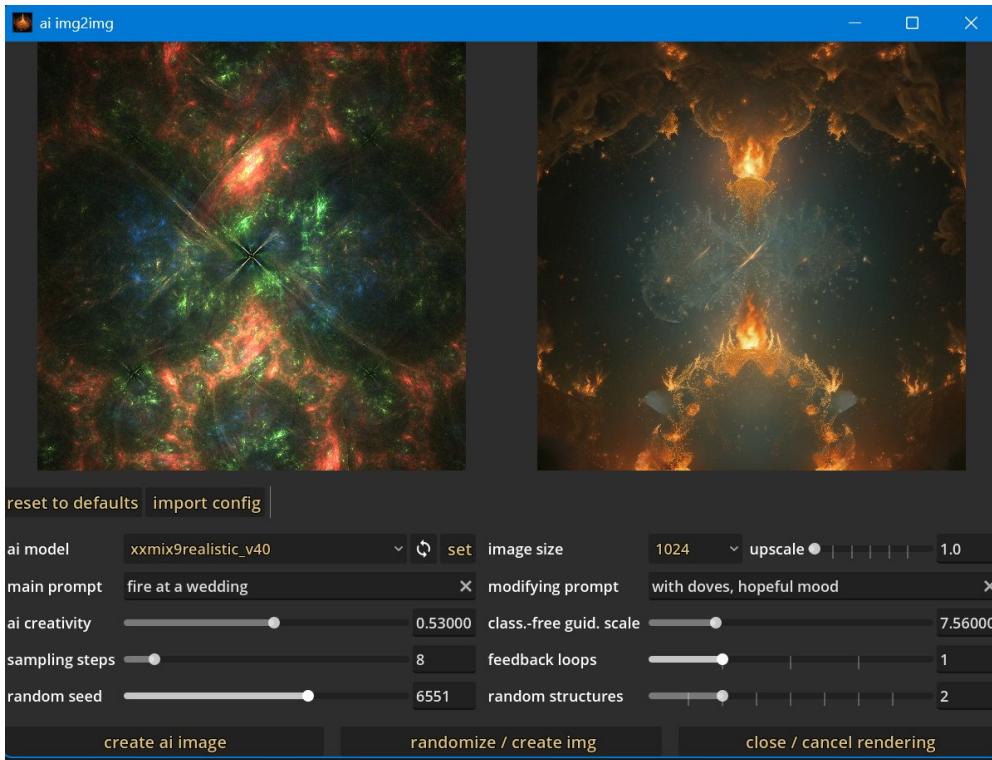
- the unique ID of the base flame is given first, followed by all mutations that have been carried out.

4.3. computation time

Please note that some mutations are very demanding (e.g. “similar gradient”) and require a longer calculation time.



Chapter 5. ai img2img



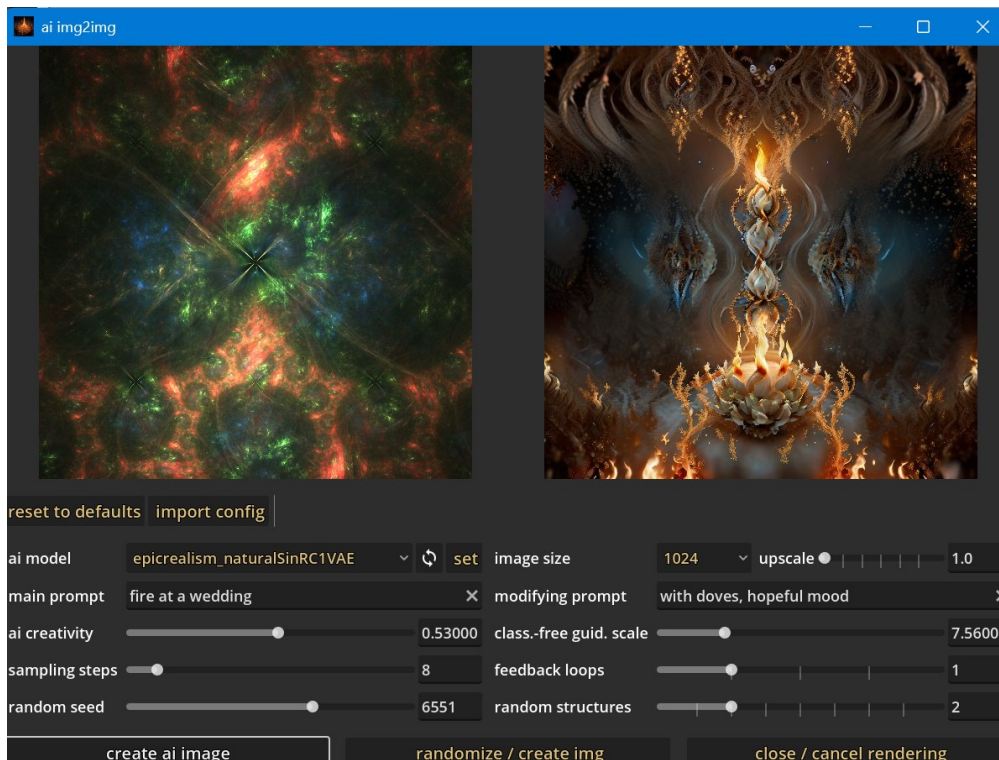
The module **ai img2img** generates new images based on a flame fractal using **AI**. It does this by using the API of the "AUTOMATIC1111 **StableDiffusion WebUI** (<https://github.com/AUTOMATIC1111/stable-diffusion-webui>). The module generates prompts based on several attributes of the current flame and feeds the **AI** both with the generated prompt and a rendered image of the fractal.

There are also some options to control the creative process of the **AI**.

Together with the random flame generators this module is a literally endless image generator. And because the module is non-modal, you may easily try out a lot of flames without closing/re-opening windows.

For example, you can just keep it open while editing you flame and create new **AI** image at any time, taking the current state of the edited flame into account.

5.1. Requirements



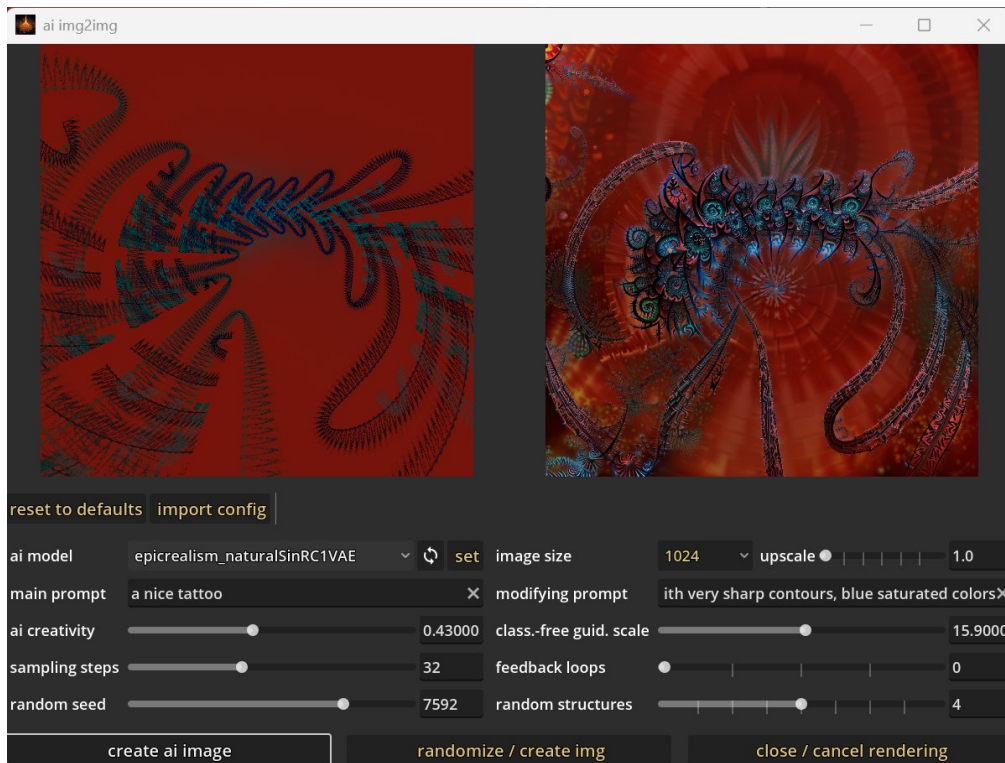
You will have to install and run *AUTOMATIC1111* locally on your computer or somewhere in the cloud.

The **advantage** is that you can use any of the many models available and it's all local (or in your cloud), which means there's no censorship and no extra costs.

The **downside**, of course, is that you need powerful hardware with a powerful graphics processor and lots of dedicated GPU memory (to generate images in higher resolutions).

And you have to take care of the installation of *AUTOMATIC1111*. But for most devices, this should be very easy.

5.2. Loading flames into ai img2img:



The window is not modal (unlike many other windows) and is connected to the main editor. You can therefore leave it open while you change the flame in the editor (or load another flame). When generating an **AI** image, the currently processed flame is always used. This makes it very easy to try out the function for many different flames.

5.3. Generating images using AI:

To generate an image there two buttons:

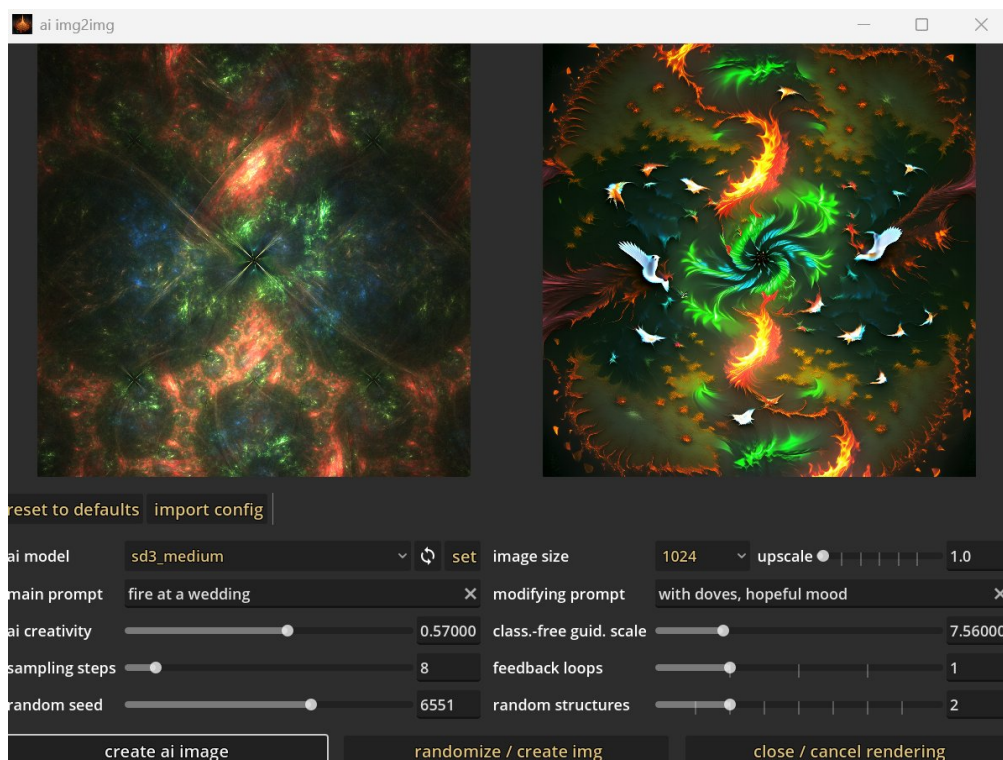
- **create ai image:** This button generates a new image based on the current flame (in the **flame editor**) and the selected options. It is good for refining your image.
- **randomize / create img:** This button generates a new image based on the current flame (in the **flame editor**) and randomized settings (except the prompts). It is good for exploring.

5.3.1. Saving generated images

The generated images are automatically saved together with **AI** settings (as *.txt-file) and the original flame image (So you will find no explicit save option). You can find these images in the "ai_img2img"-subfolder inside your image folder. They are grouped together by the flame name.

5.3.2. Refine generated images / Re-use settings

At each image generation action the corresponding parameters are also saved as a *.txt-file. You may later import this file by copying its contents into the clipboard and press the "import config" button (e.g. for using the same parameters for different flames).



5.4. ai img2img image generation options:

- **ai model:** shows the list of valid models installed in *AUTOMATIC1111*. You can change the model both directly in *AUTOMATIC1111* (via the web-ui) and in *JWildfireSwan*.

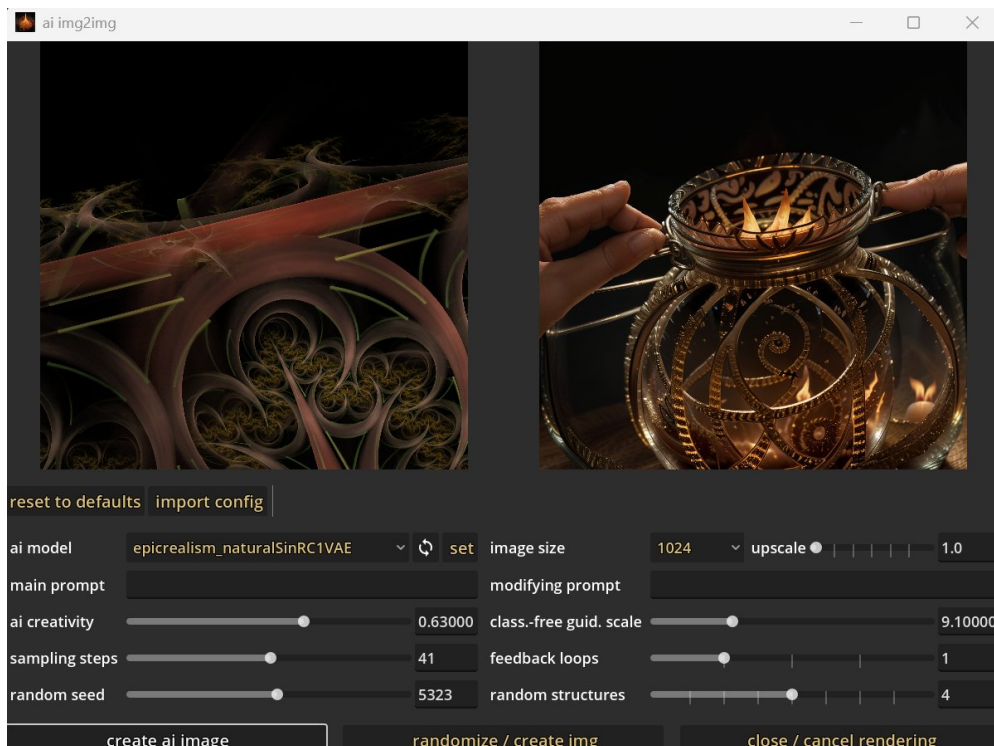


Please note that loading a different model is very time-consuming and can take a long time. You must therefore explicitly click on the “Set” button to load the new model into *AUTOMATIC1111*. Then please be patient, *JWildfireSwan* will notify you when loading is complete.

- **image size:** a list of predefined (square) image sizes. Larger sizes take considerably longer and require more GPU memory, but generally produce more interesting results (and not just larger images).



But, please note, that not all image sizes will work on any device. Especially the sizes above 1024 will require modern hardware.

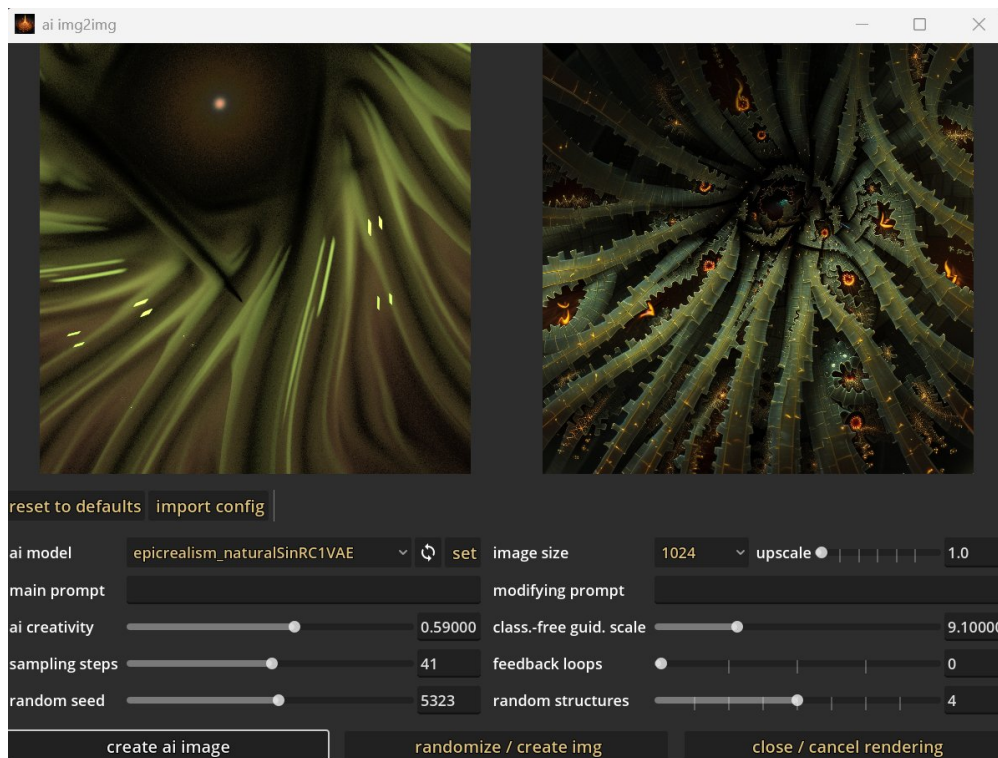


And please note that changing the image size can drastically change the image content. This is due to the sampling algorithm used by the **AI**. An alternative may be to enlarge an image with the **upscale** option. However, this can also drastically change the content, depending on which **ai model** is used.

- **upscale:** Upscale factor to enlarge the image. This works in a more sophisticated way than a classic image enlargement algorithm and can both drastically change the image and create more detail. So the best method is to play around to see which method works best in a particular context: Changing the **image size** or up scaling the image.
- **ai creativity:**

The freedom you give the **AI** to fulfill the request. The lower the value, the more the generated

image resembles the original. The higher the value, the more options the **AI** has to fulfill the request, but the more the generated image differs from the original. The default value is rather low in order to create images that actually look like fractals. However, this is one of the most interesting parameters to play with.

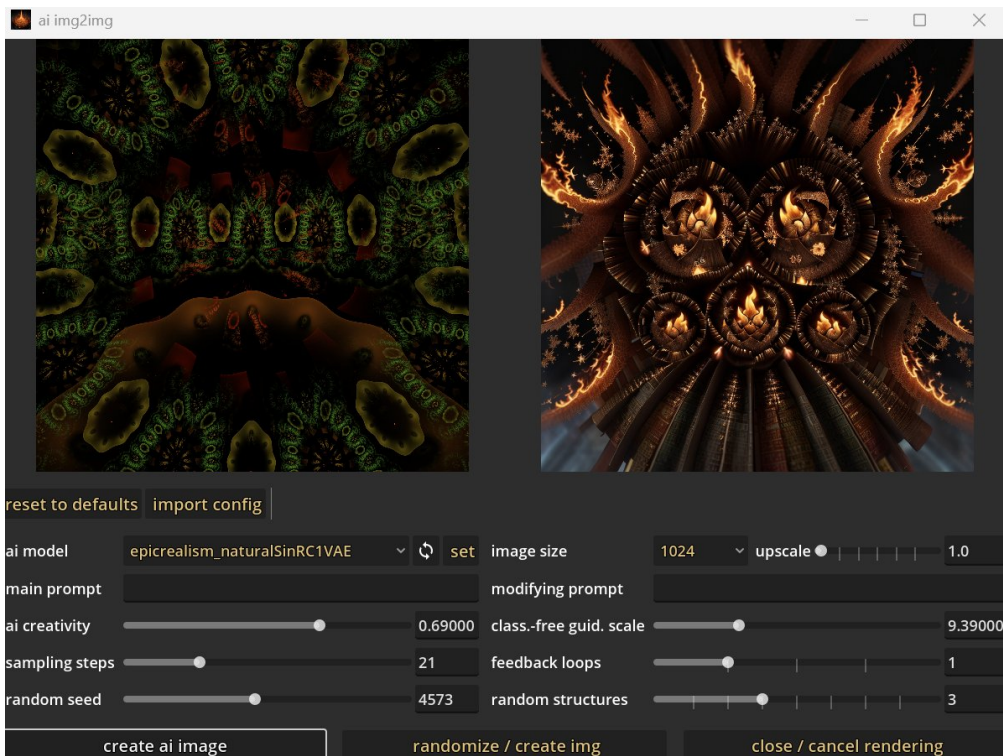


- **main prompt:** This prompt helps to customize the image to your liking. It is included in the generated prompt, which aims to create an image that resembles the original fractal. So it is only a modification, but it has a big impact.

The results depend heavily on the model used. As a rule, the better the “understanding” of the terms used by the model, the better the results. More modern models (such as SD3) are generally better in this respect.



If you find that your prompt is not being honored, try increasing the **ai creativity** value.

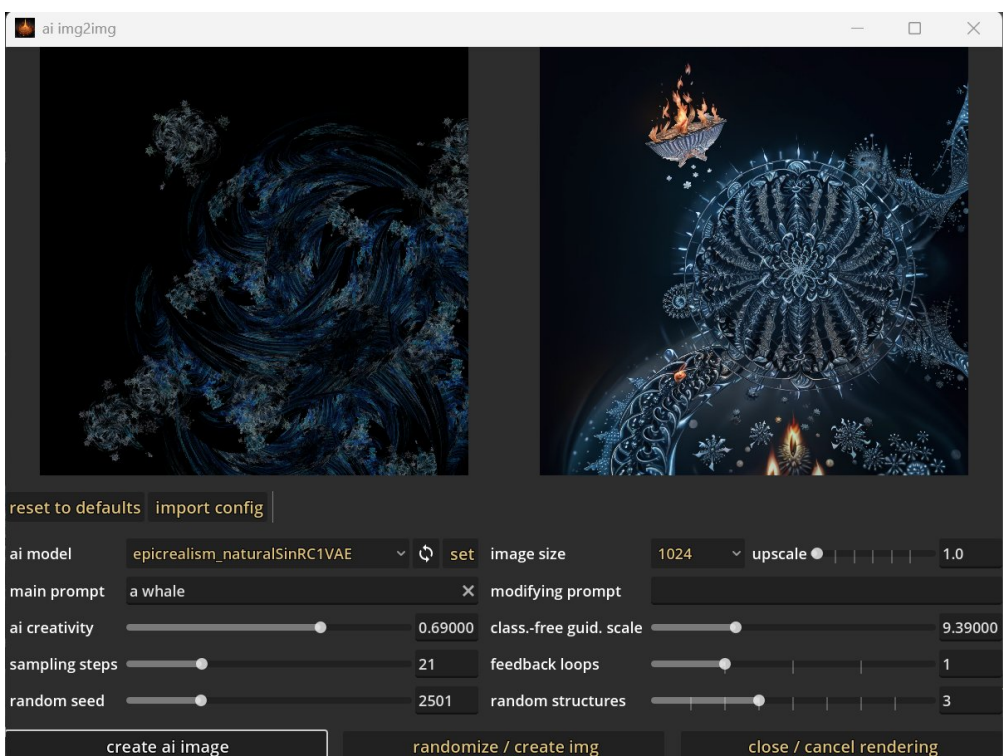


- **modifying prompt:** an additional prompt that is included in the generated prompt with less influence.

The results depend heavily on the model used. As a rule, the better the “understanding” of the terms used by the model, the better the results. More modern models (such as SD3) are generally better in this respect.



If you find that your prompt is not being honored, try increasing the **ai creativity** value.



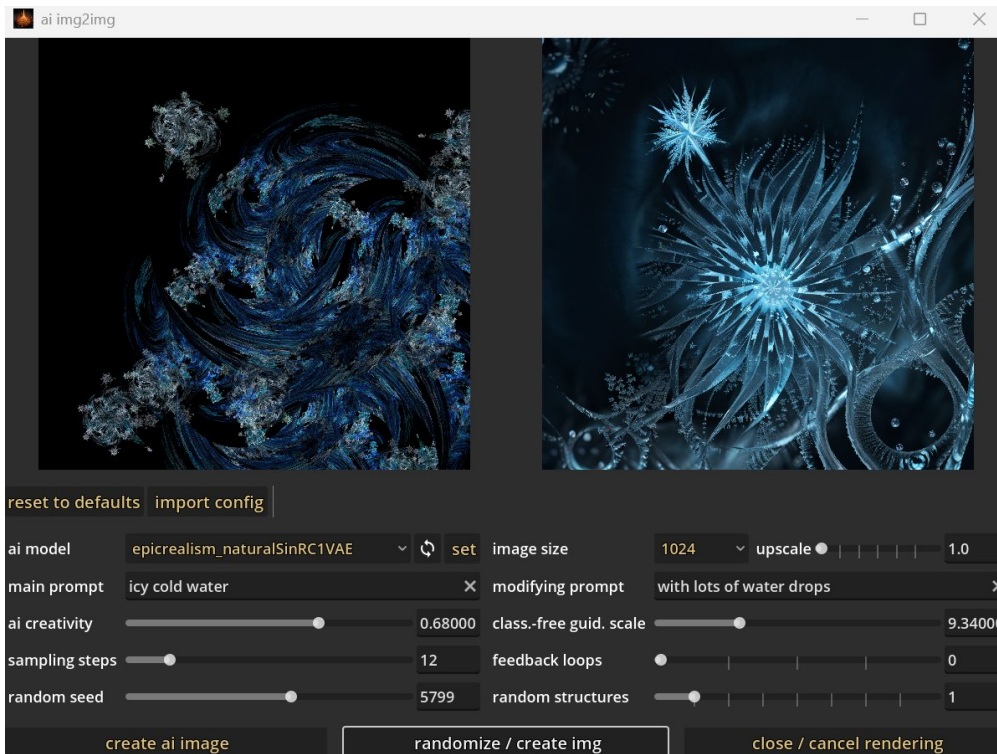
- **class.-free guid. scale** (classifier-free guidance scale): Another parameter that helps the model

to fulfill the prompt. It is usually model specific and not specific to a particular artwork you want to create. See <https://stable-diffusion-art.com/cfg-scale> for more details. The default value of 8.0 should be sufficient in most cases.

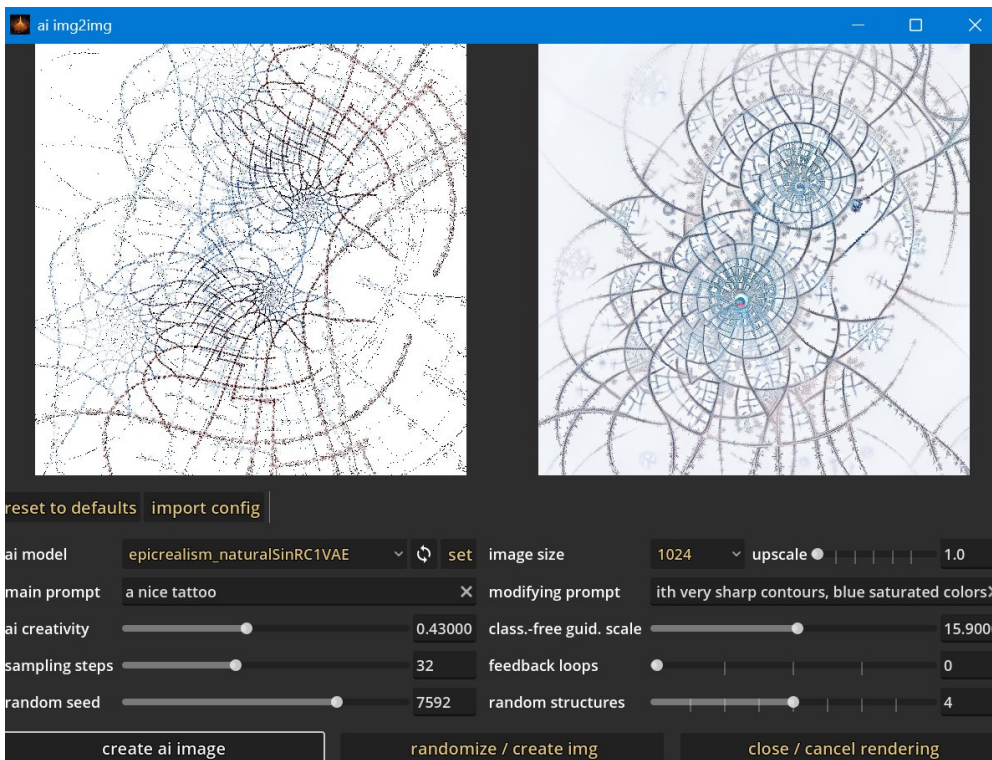
- **random structures:** Creates additional randomized structures within the new images. To do this, random phrases are created and inserted into the generated prompt.



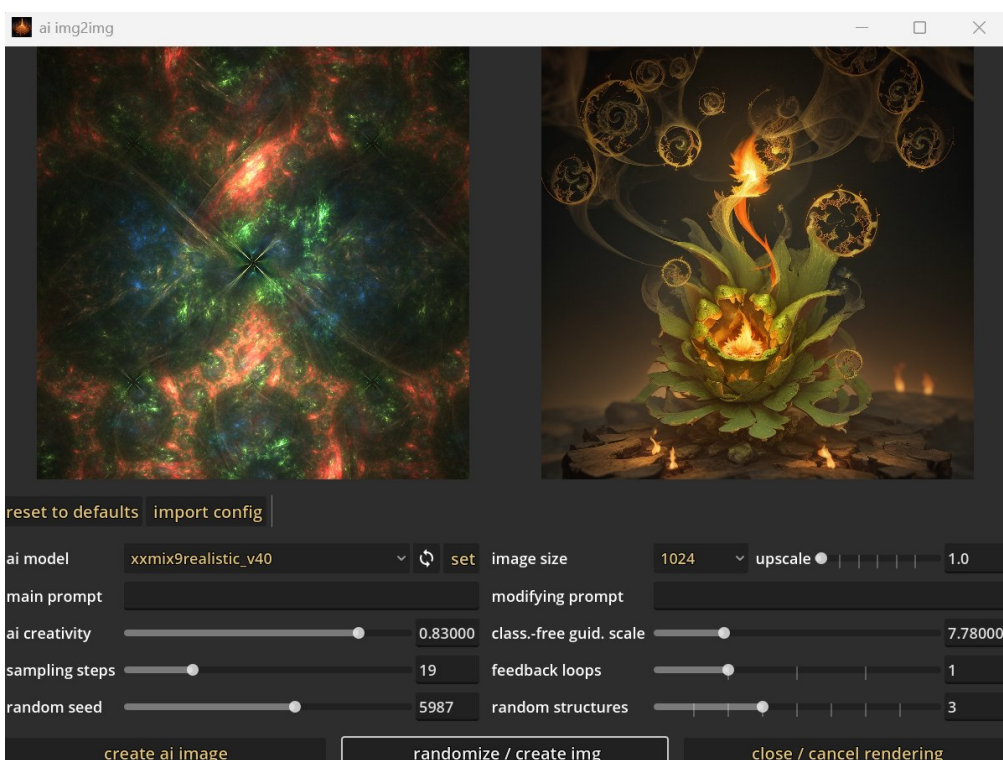
If you find that your prompt is not being honored, try increasing the **ai creativity** value.



- **feedback loops:** An optional function that will help the AI to better fulfill the prompt. This is done by feeding parts of the generated image back into another image generation. Although this can have a strong impact on the final effects, it also has a major influence on the calculation time. However, it is recommended to activate this option at least as a test and to have at least one feedback loop. It can lead to very interesting results.

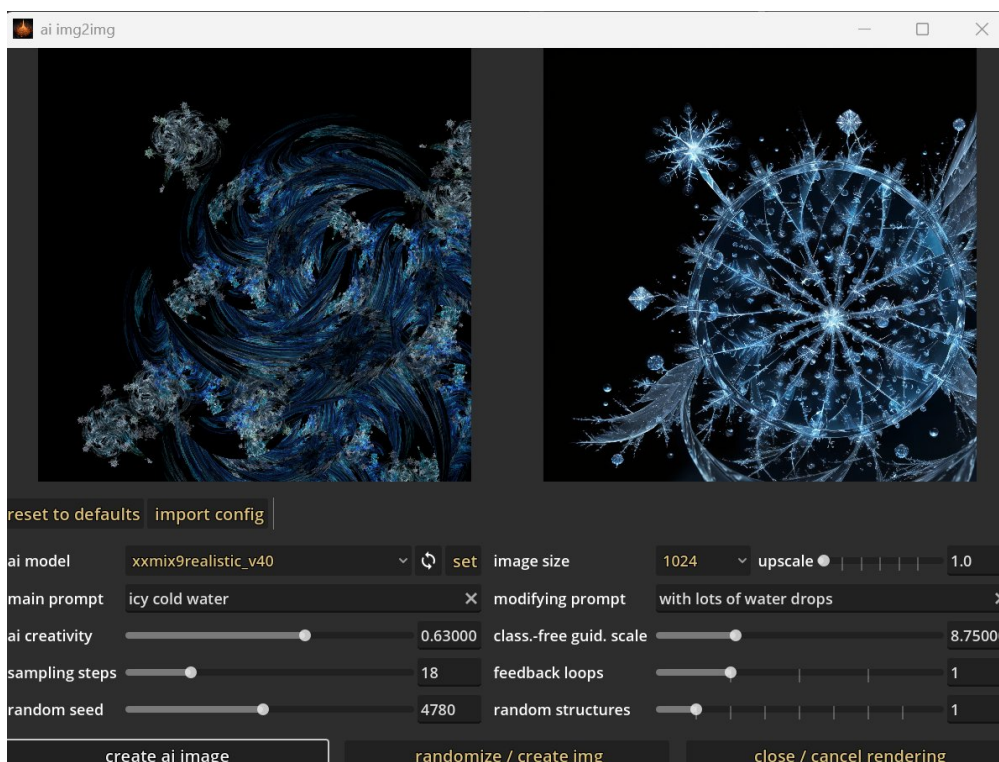


- **sampling steps:** Number of steps the model performs to create the desired image. As a rule, this is only a guideline, as not all steps may be carried out in the sampling method. And a higher value is not always “better”, so there is room for experimentation here. As a rule of thumb, values between 10 and 30 are a good choice. Higher values generally require more calculation time.
- **random seed:** Random seed, which applies to both the **AI** and the prompt generator. So using the same seed (with the same other settings) will produce the same prompt and the same image of that prompt. By varying the seeds, you can try numerous different versions of the same set of options.



5.5. Installation of *AUTOMATIC1111*:

- install *AUTOMATIC1111* from the official web-site: <https://github.com/AUTOMATIC1111/stable-diffusion-webui>.
 - There may be special instructions for your device or operating system (It has been successfully tested on numerous devices, both with Windows and macOS so far).
- install any models you want to use into *AUTOMATIC1111*.
 - This is easily done by downloading it and placing it in the “models/Stable-diffusion” folder in your *AUTOMATIC1111* installation folder.
- there really are a huge number of great models, here are some examples that have been successfully tested:
 - **epiCRealism** (<https://civitai.com/models/25694/epicrealism>)
 - **Juggernaut-X-Hyper** (https://huggingface.co/RunDiffusion/Juggernaut-X-Hyper/blob/main/JuggernautXRundiffusion_Hyper.safetensors)
 - **epiCPhotoGasm** (<https://civitai.com/models/132632/epicphotogasm>)
 - **Stable Diffusion 3 ("SD3")** (<https://huggingface.co/stabilityai/stable-diffusion-3-medium>)
- **important:** enable the API of *AUTOMATIC1111*. *JWildfireSwan* will communicate with *AUTOMATIC1111* via http. For this to work, the API of *AUTOMATIC1111* must be activated. The easiest way is to add the **--api** argument when calling *AUTOMATIC1111* via the command line.



- **optionally** change the port of *AUTOMATIC1111*. It was reported, that under some circumstances the default port (7860) of *AUTOMATIC1111* will not make the API to work. In this case you can easily change the port by specifying the port number using the commandline: **--port <port>** (also in this case you have to add the **--api** option).



It is important to use the same port in JWildfireSwan. You can do this by changing the port in the parameter "a1111 api" in the application settings.

Chapter 6. flame library



The **flame library** serves two main purposes: It contains an extensive collection of prepared and curated sample *flames* that you can customize and learn from. This collection will be expanded from version to version. You cannot change the content of this collection directly, but you can copy from it.

The other purpose of the **flame library** is to store your own *flames*. They are therefore saved on your computer and are not changed when a new version of *JWildfireSwan* is installed.

6.1. folder view

There are two main folders in the folder view on the left-hand side:

- **your flame library:** This refers to a directory on your computer in which your own *flames* are stored. This directory is created automatically when the software is first installed, but is never changed later by *JWildfireSwan*. It can contain any number of subdirectories that you can use to organize your *Flames*. You can use a file manager of your operating system to manage your files as well as the functionality provided by *Swan*.



You can access your flame folder at any time via the right-hand main toolbar. If you click on the folder icon, a file explorer window opens showing the contents of the main folder.



It is recommended that you back up your *Flames* regularly.

- **built-in examples:** This refers to a virtual directory in which the contained example *flames* are provided. You can therefore only access these files from *Swan*.

6.2. flames view

After you have selected a folder in the folder view, the `_flames` are displayed in the flame view.

If you click on a *flame* in the flame view, a preview image of this flame is rendered and **meta-data** about the flame is displayed.

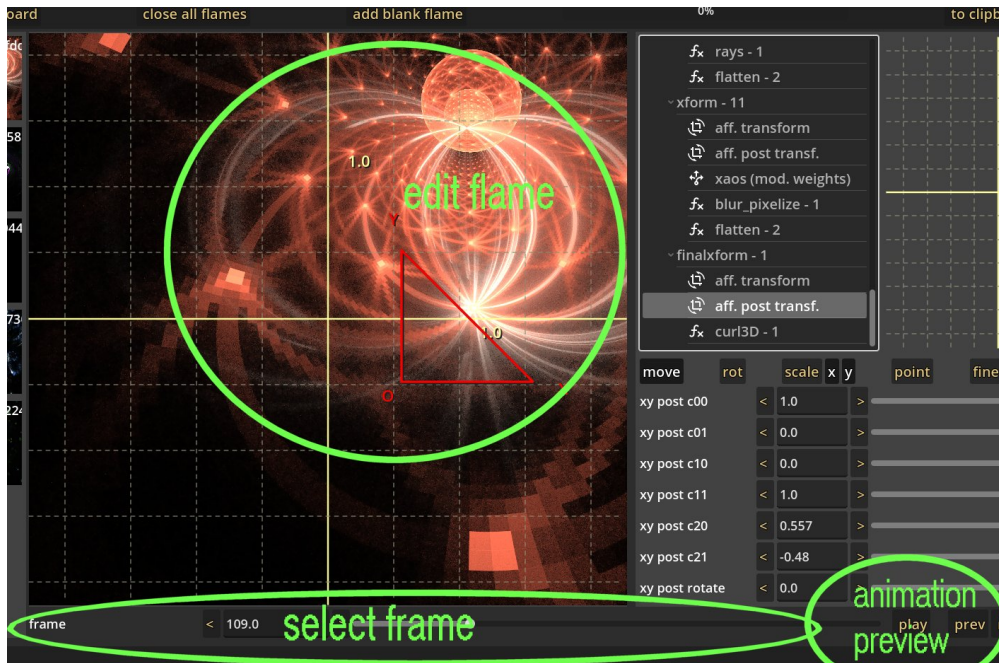
6.3. *flame* meta-data

Each *flame* in the library has a number of attributes that describe it:

- **flame name:** a short name that describes the *flame*.
- **comment:** a longer description of the *flame*.
- **author:** the name of the creator of the *flame*.
- **tags:** any number of tags that describe the *flame*.
- **creation time:** the date and time when the *flame* was created.

You can change these attributes for your own *flames* both in the **flame editor** and in the **flame library**.

Chapter 7. key-frame-based animations

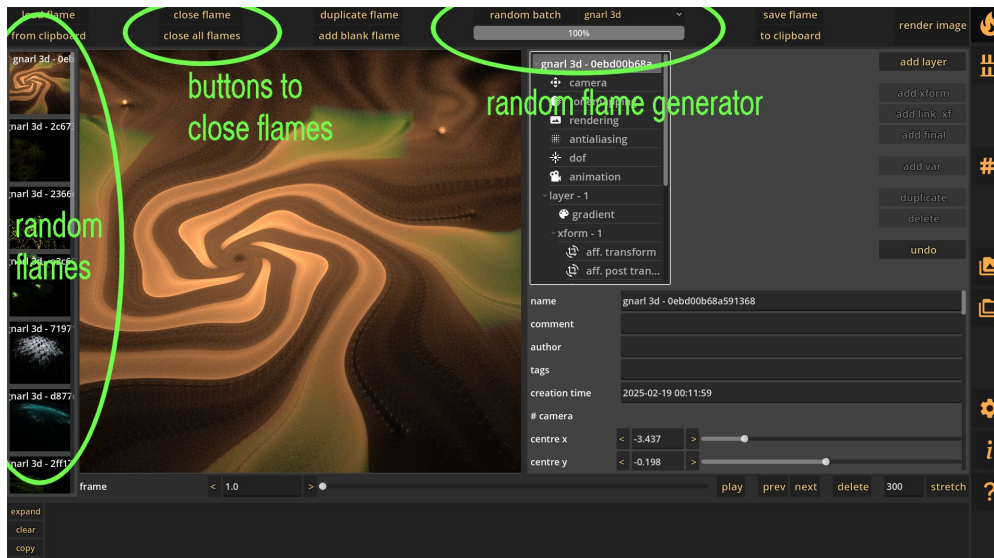


Creating keyframe-based animations is very easy in *Swan*. For a static (non-animated) flame, you normally edit all flame attributes on frame 1.

If you want to animate them, you edit the attributes in additional frames (called key frames), and *Swan* automatically interpolates the values between the key frames to create a smooth animation.

You can also jump to existing keyframes and edit or delete them.

Chapter 8. working with random flames



Creating new *flames* from scratch can be a very difficult task. There are so many concepts and options, most of which can be combined. There are literally endless possibilities. That's why we love *flame fractals*, but the process has a very challenging and long learning curve. In fact, it can take years to master.

A good place to start is to research and tweak existing flames. One option is to work with randomly generated flames. *JWildfire* has always been very strong at this, and *JWildfireSwan* takes this to another level. Most of the random flame generators you know from *JWildfire* can also be found in *Swan*. However, they have been optimized in quality and variety.

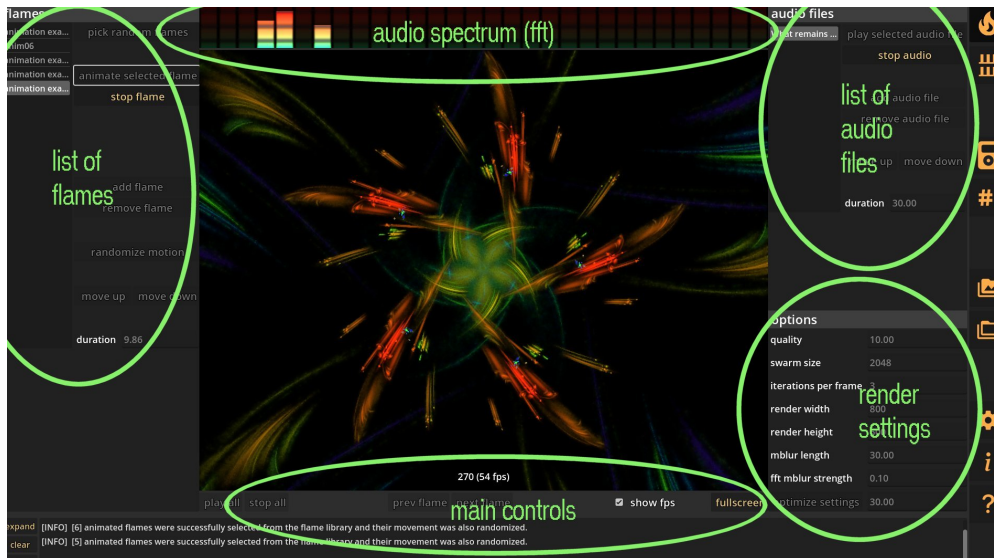
8.1. creating random batches

Random flames are usually created in so-called batches. The basic idea is that *Swan* creates a collection of random flames and you keep the ones you like. You can then edit these *flames* further or use them as they are.



The default batch size is 8. You may change this in the [application settings](#).

Chapter 9. jukebox



This module allows you to create audio-spectrum-based animations of flame fractals in real time. It requires a strong graphics card to work smoothly.

It has been successfully tested with the following devices to produce excellent results: * NVIDIA GeForce RTX 4090 on Windows * Mac M1 Pro on Mac OS

In general, it will work on all devices, but may produce less smooth results on older or weaker devices. There are several settings that you can adjust to optimize the performance of the jukebox, like render resolution, sample quality, etc.

There is also an option to display the fps (frames per second), which should help to play with the various options. In general, you should aim for a fps value of 25 or higher, otherwise the animation will not be smooth.

9.1. list of flames

The list of flames, which are animated one after the other, is on the left-hand side. You can control the duration (in seconds) of each flame by entering a value in the input field.

If you add more than one flame, all flames will be prepared for rendering at the same time to allow a smooth transition during the animation. You can even force the change to the next or previous flame by pressing the corresponding buttons (at the bottom area).



Only flames which are already animated can be animated in this module. Simply put, the *jukebox* adds a music-based animation to the existing animations (e.g. motion curves).

But, most random flames are animated, and there is already a collection of animated flames in the flame library. So, it should be no problem to find some flames to play with.



Currently, there is a limit of 8 flames you may animate at once. This is to limit the memory usage on your graphics card.

9.2. randomize motion

Currently, you cannot have much influence on how Jukebox animates the flames. But you can randomize the movement with the “Randomize motion” button. This changes how the audio spectrum influences the original movement curves of the flame.

9.3. list of audio files

The list of audio files, which are played one after the other, is on the right-hand side. You can control the duration (in seconds) of each audio file by entering a value in the input field.

9.4. getting started / picking random flames

To make it easier for you to get started, there is a function for randomly selecting flames from your flame library. It analyzes the flames and selects only the ones that are animated.

Simply press the “pick random flames” button (it will take a little longer the first time) and then “play all” to start the real-time animation. (For convenience, also a random audio track is selected if no one is present yet.)

9.5. motion blur

jukebox supports beautiful motion blur. It distinguishes between the normal motion blur of the animated fractal and the motion blur caused by the animation based on the audio spectrum. Often, the audio spectrum changes very quickly, so the motion blur can be very strong. You can therefore adjust the level of motion blur by changing the “fft mblur strength” setting.

The “mblur length” setting determines the length of the motion blur. It is the same value as in the *flame editor*, but it is applied uniformly to all flames in the *jukebox*.

The higher the value, the longer the motion blur.

9.6. optimize (render) settings

There are various settings that control the render speed. The render speed is particularly important for the *jukebox*, otherwise the animations will not look smooth.

- **quality:** The render quality. The higher the quality, the better the fractals look, but the slower the rendering.
- **swarm size:** The number of particles used to render the fractals. The higher the number, the faster the rendering is usually. However, much more memory is also required, so this number should be increased with caution.
- **iterations per frame:** The number of fractal iterations per frame. It can be helpful to play with

this value at higher quality levels (>10.0). However, the results depend on the fractals. Some fractals are difficult to calculate, so adding many iterations during a frame will not help to increase the overall rendering speed.

- **render width** and **render size**: The resolution of the rendered fractals. The higher the resolution, the better the fractals look, but the slower the rendering.

The “optimize settings” button attempts to find the best settings for your system in order to achieve a certain fps value (this fps value is entered in the field on the right). In general, you should aim for a fps value of at least 25, otherwise the animation will not be smooth. However, values higher than 50 are not suitable for most systems, as flame fractals are very expensive to render (compared to other structures, e.g. 3D objects).

9.7. play single file or all files

To animate fractal you have to choices:

- interactive:
 - select any flame from the list on the left-hand side and press the “animate selected flame” button.
 - select any audio file from the list on the right-hand side and click on the “play selected audio file” button.
 - You can then pause the animation or audio file and start a different flame or audio file, e.g. to play around with which flame goes well with which soundtrack
- automatic:
 - press the “play all” button to play all flames with all audio files in the list.
 - but even in this mode you have a certain degree of control. You can switch between the flames in real time using the “next flame” and “previous flame” buttons.

9.8. fullscreen mode

You may switch to fullscreen mode by pressing the “fullscreen” button. This will hide all other elements of the user interface and only show the main preview in fullscreen mode.

To exit the fullscreen mode, press the <Escape>-key.

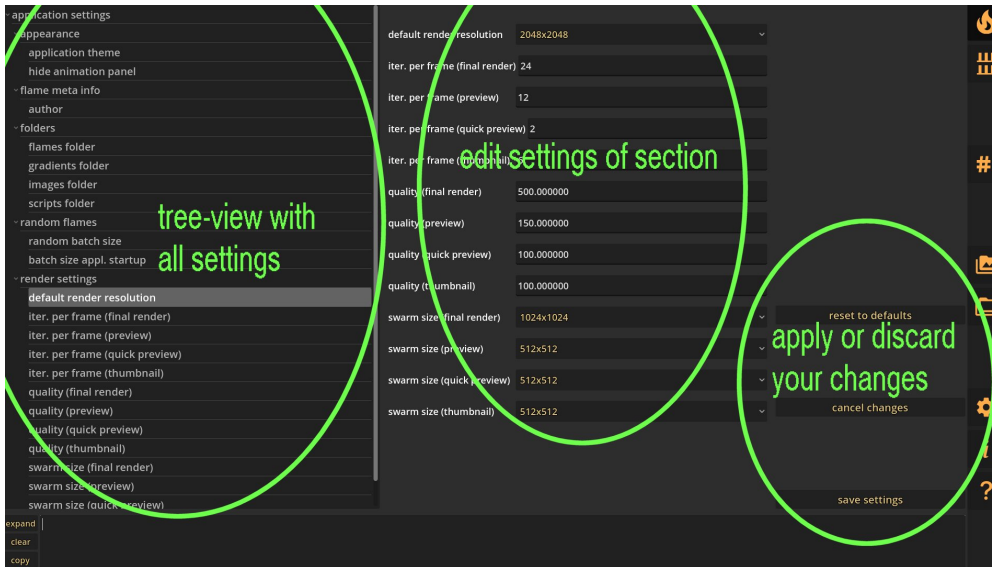
To start/stop the animation, press the <Space>-key. This corresponds to the “play all/stop all” buttons.

9.9. general tips

- “Simple” flames are usually better suited for the jukebox. The more complex a flame is, the more difficult it is to animate it in real time.
 - “Simple” means that the flame has only a few variations. The number of affine transformations plays *no* role.

- Some variants are really heavyweight and can lead to a low fps value. Popular examples are “synth” or “crackle”. Try to avoid these or remove them for *jukebox*. As a rule of thumb, the more parameters a variation has, the more complex it is.
- Avoid final transformations if possible. They are exceptionally expensive to render.
- Use the random flame generators to create endless animated flames that you can use in the *jukebox*. Well suited generators are “simple exp.”, “simple but stunning” or “linear only”. (But many others also work very well, it just takes more attempts). Use the “play button” in the [flame editor](#) to see if the flame is well animated.
- Use flames that look good at low quality settings. Especially the use of a strong blur can be problematic. With low (real-time) rendering quality settings, the visual quality can be too low and interfere with the motion blur, making the end result too blurry.

Chapter 10. application settings

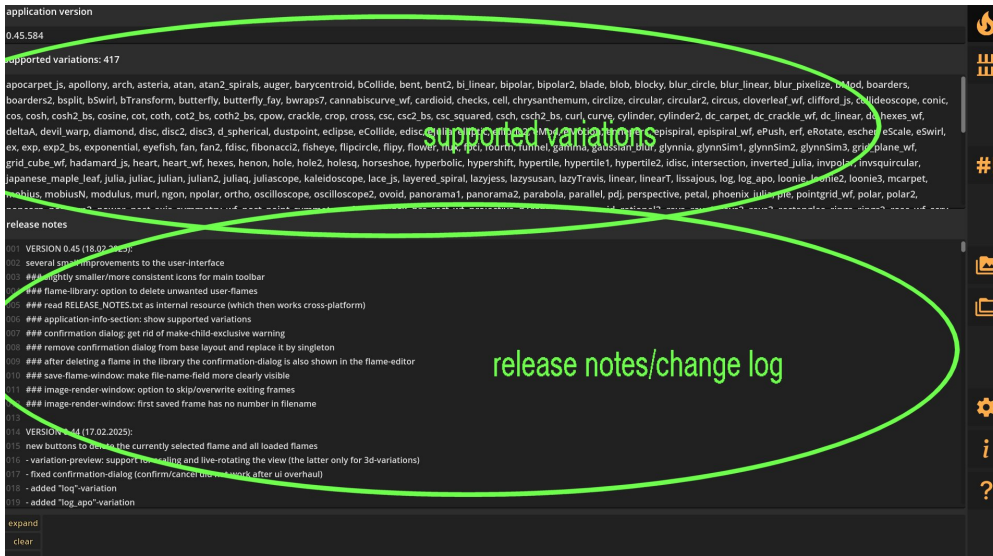


Here you can adjust various application settings according to your wishes. After changing a value, this change is indicated by an asterisk (*) in the label. You must save the changes to apply them.



Some settings require a restart of the application to take the full effect.

Chapter 11. application info



This module show some useful information about the application:

- **version:** the version of the application.
- **supported variations:** number and name of currently supported *variations*.
- **release notes:** a complete log of the application's release notes since the first version.

Chapter 12. integration of ffmpeg to encode videos

To create an encoded mp4 video directly from your animated flames, *JWildfireSwan* supports the use of *ffmpeg*, a free and extremely powerful video encoder.

12.1. Installation of *ffmpeg*

For copyright reasons, *ffmpeg* is not included in the *Swan* distribution. You must therefore download it once and install it yourself. But this is really very simple, basically you need to download the *ffmpeg* executable file from the official website and copy it somewhere on your computer.

Depending on our operating system, you can also use an installer or package manager such as *homebrew* to install *ffmpeg*.

12.1.1. Windows

Under Windows, you can download a fully compiled executable file from the following page: <https://www.gyan.dev/ffmpeg/builds>

12.1.2. Mac OS

On Mac OS, you can use *homebrew* to install *ffmpeg*, or download an executable from this page: <https://www.osxexperts.net>



Make sure that you select the correct build for your processor architecture (Apple Silicon/ARM architecture or Intel architecture).

12.2. Setting the path to *ffmpeg* in the **application settings**

After installing *ffmpeg*, you must set up the path to its executable file once in the **application settings**. This is done in the “*ffmpeg*” section. Under “*ffmpeg* command”, enter the full path to the *ffmpeg* executable file (which is normally called “*ffmpeg*” or “*ffmpeg.exe*” under Windows).

You can then create videos from your animated flames with a single click in the animation render window.

12.3. extra encoding settings

You can also specify additional encoding settings in the same section under **application settings**. However, you should be careful here, as incorrect settings can lead to undesirable results or bring the entire encoding process to a standstill.