
Cascade

Release 0.2

ttddee

May 18, 2022

CONTENTS

1	Contents	3
1.1	Usage	3
1.2	Quickstart	4
1.3	Contributing	7
1.4	Building from Source	8
1.5	Writing a shader	9
1.6	Using ISF shaders	13

Cascade is a node-based image editor for Windows and Linux.

Check out the [Usage](#) section for information on how to run the application.

To get a quick intro on how to actually use the software, see the [Quickstart](#).

Note: This project is under active development.

CONTENTS

1.1 Usage

Cascade is available as a portable build in the form of Windows binaries or a Linux appimage.

You can find the latest version on the [releases page](#) of the Github repository.

1.1.1 Requirements

In order to run the application you will need a graphics card and driver that is able to run Vulkan.

Vulkan is widely supported nowadays. If you are in doubt, you can check if your GPU is supported [here](#).

1.1.2 Windows

Simply download the zip file from the releases page, extract it, and run *Cascade.exe*.

1.1.3 Linux

Download the AppImage from the releases page and make it executable.

```
chmod a+x Cascade-Image-Editor-x86_64-xxx-release-xxx.AppImage
```

Then you can run the AppImage like so.

```
./Cascade-Image-Editor-x86_64-xxx-release-xxx.AppImage
```

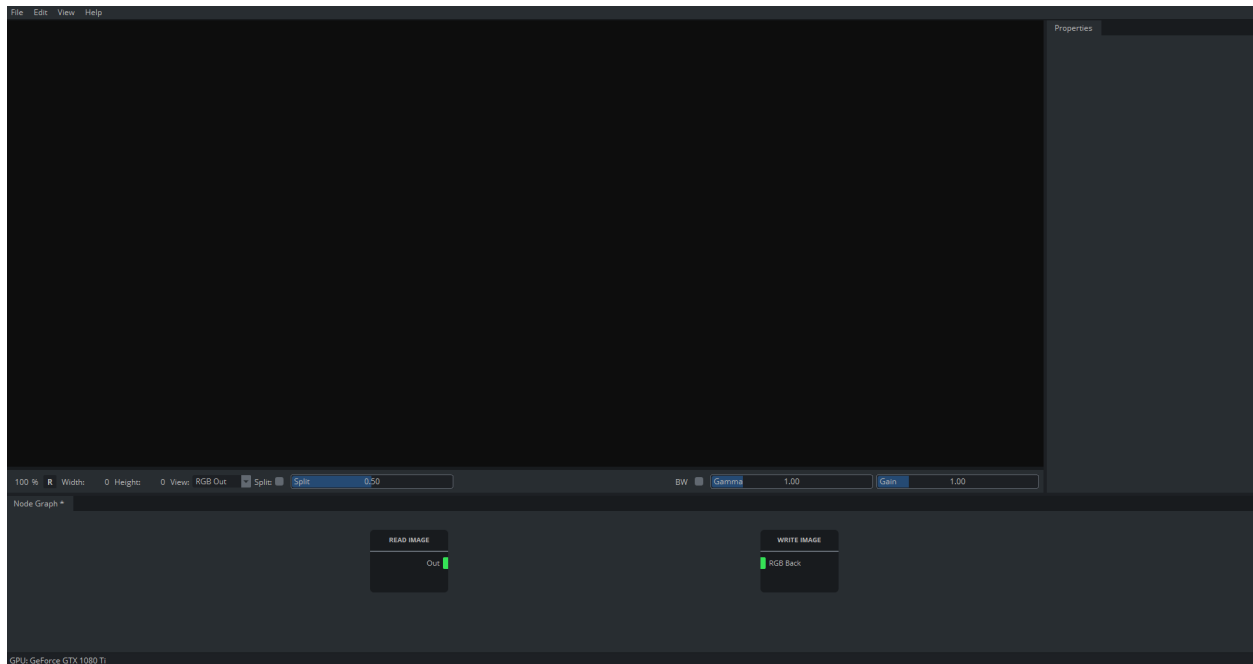
Note: Cascade creates some files for logging and persistence when you first start it. In order to keep things neat, it is best to put the AppImage into its own folder.

1.2 Quickstart

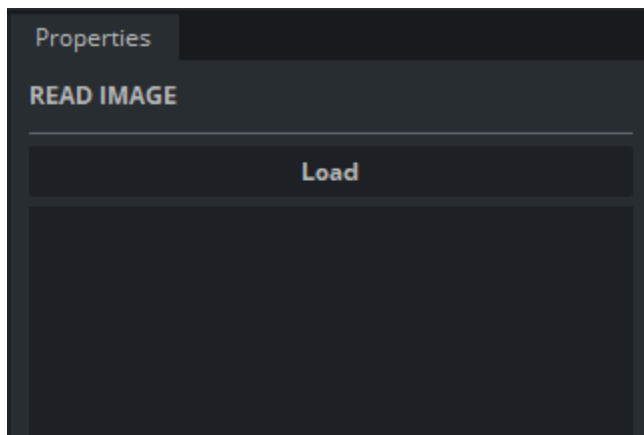
This is a quick introduction on how to use *Cascade*.

1.2.1 Navigating the Node Graph

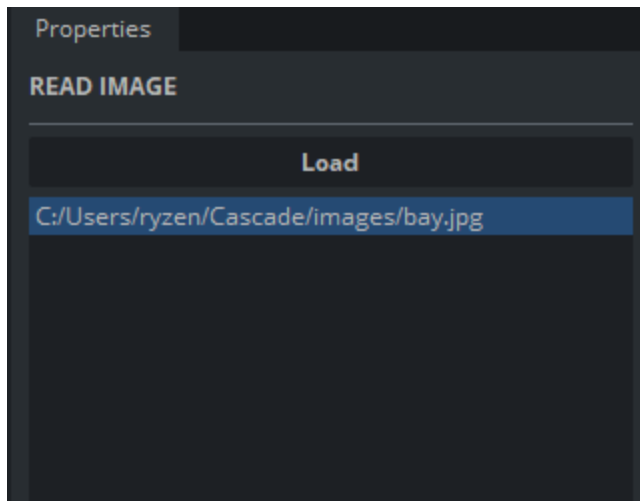
The node graph is the window you see at the bottom. When the application starts there will be two default nodes created for you. A *Read* and a *Write*.



To load an image *double-click* on the *Read Node*. This brings up the *Properties Panel* on the right. Click on the *Load* button and choose an image file in the dialog.



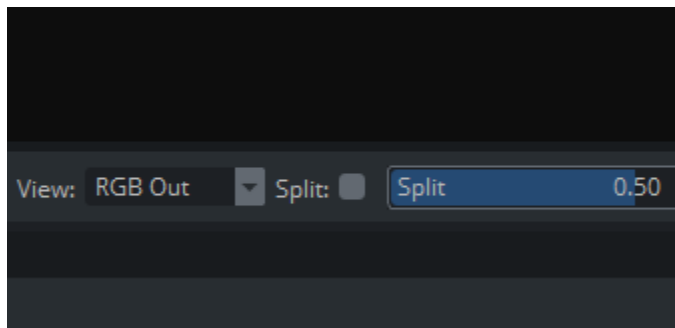
This loads the image into the *Read Node*. You will see the path added to the list in the properties panel.



We still can't see the image though. That is because we are not *viewing* the node. To view the output of a node, click on it and press *F4*.

This shows you the RGB image the node is outputting. If you press *F4* again, you can toggle between viewing the RGB output and the Alpha output.

There is a little box under the viewer that shows you what you are currently looking at.

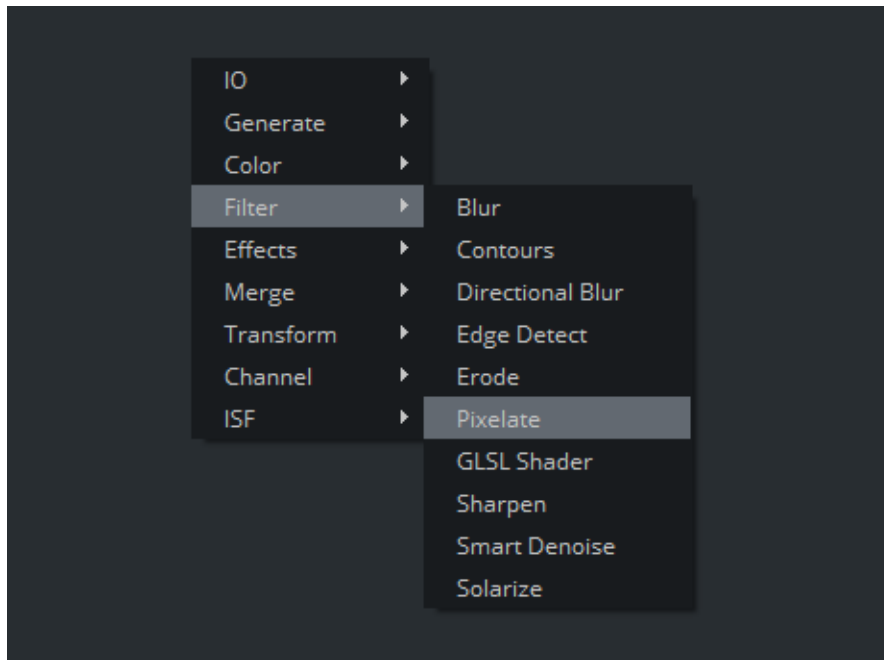


The viewer can be toggled between five states:

- RGB Front input (F1)
- RGB Back input (F2)
- Alpha input (F3)
- RGB/Alpha Output (F4)

Now, let's do something with our image.

Right-click anywhere in the node graph to bring up the context menu and navigate to *Filter > Pixelate*.



This creates a *Pixelate Node*. Now, left click and drag from the green output of the *Read Node* to create a connection and hook it up to the input of the *Pixelate Node*.

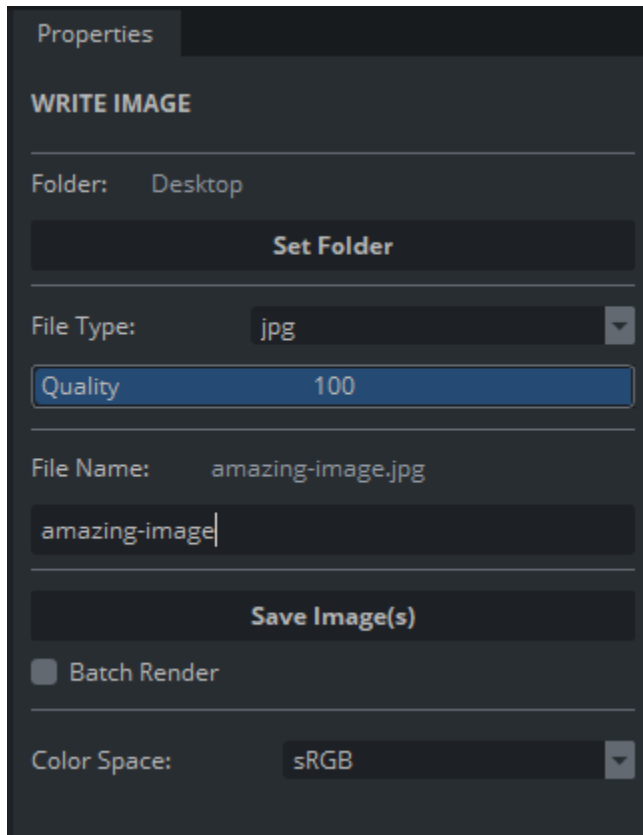
You should see the pixelation effect on your image. If not, make sure to click the *Pixelate Node* and press *F4*.

In the properties panel you can adjust the strength of the effect.

Note: Whenever you want to reset a slider control to its default value you can do so by left-clicking it with the *Ctrl-Key* pressed.

If you want to save the result image to disk, connect the output of the *Pixelate Node* to the input of the *Write Node*.

In the *Write Node* properties set a folder and a file name and choose a file type.



The screenshot shows a dark-themed 'Properties' panel for Cascade. The 'WRITE IMAGE' section is active. It includes a 'Folder' dropdown set to 'Desktop' with a 'Set Folder' button below it. The 'File Type' dropdown is set to 'jpg'. A 'Quality' slider is set to 100. The 'File Name' field contains 'amazing-image.jpg' and a text input below it contains 'amazing-image'. A 'Save Image(s)' button is prominently displayed. Below it is an unchecked 'Batch Render' checkbox. At the bottom, the 'Color Space' dropdown is set to 'sRGB'.

Now, click on *Save Image* and your file will be saved to disk.

If you want to keep this node setup and come back to it later, you can save it by going to the menu at the top and choosing *File > Save Project*.

1.3 Contributing

Thanks for being here!

There are many ways in which you can contribute to the development of Cascade.

1.3.1 Testing

Run Cascade on your system. See if you can find those pesky bugs. Create an issue on Github if you got one.

1.3.2 Features

There is a feature you would really like to see? Or there is an existing feature that you think could be improved? Let us know about it!

1.3.3 UI/UX and design

Give feedback on the UI and how it could be improved. If you are a designer, maybe you want to help create a website? Or maybe you have a Youtube channel and want to make a tutorial video?

1.3.4 Documentation

Is there something that you would like to see in the documentation? From a developer's point of view it is sometimes very hard to tell what's obvious and what's not. Getting an outside view is very much appreciated.

1.3.5 GLSL

If you know how to write *GLSL* you could add new effects to Cascade by writing a shader. Or you could improve an existing one, check out the *shaders* folder.

1.3.6 C++

Look through the open issues and see if you can find something that interests you. If you need further information or clarification, don't hesitate to ask on Github or Discord.

1.4 Building from Source

1.4.1 Windows

To build the project on Windows, you will have to install *Qt* and the *Vulkan SDK* manually. All other dependencies are handled by *vcpkg*.

Windows versions are compiled with *MSVC 2019 64bit*.

Install Qt for open source using the [official installer](#). At the moment we are using version 5.15.2.

Make sure you have an environment variable *QT5_DIR* pointing to the path of your Qt installation. The path might look like this:

```
C:\Qt515\5.15.2\msvc2019_64
```

Get the [Vulkan SDK](#) and install it. We are currently using version *1.2.198.1*. The environment variable *VULKAN_SDK* has to be set to your Vulkan SDK. For example:

```
C:/VulkanSDK/1.2.198.1
```

It's easiest to use Qt Creator as IDE, but feel free to use Visual Studio if you want.

Open a command prompt and clone the Cascade repository:

```
git clone https://github.com/ttddee/Cascade
```

Enter the project directory and install vcpkg:

```
cd Cascade
git clone https://github.com/microsoft/vcpkg
.\vcpkg\bootstrap-vcpkg.bat
```

Now you can install all other dependencies using the command below:

```
.\vcpkg\vcpkg --feature-flags="versions" --triplet=x64-windows install
```

This will take a while to compile but upon completion you should be able to open the project in Qt Creator, configure your environment and build.

1.4.2 GNU/Linux

To download source files and dependencies, you can run:

```
git clone https://github.com/ttddee/Cascade
cd Cascade/scripts
sh install-deps.sh
```

This script might ask you for the administrator password if a package dependency can be installed via the system package manager.

Now, open the file *Cascade.pro* with QtCreator.

1.5 Writing a shader

In **Cascade**, effects are implemented as **GLSL** shaders and processed on the GPU.

This has several advantages:

- The parallel nature of GPU processing is a perfect fit for image processing. Aka it's fast.
- **GLSL** is a simple language that is easy to learn.
- Processing algorithms are contained in small, portable shader files.
- There exists a large amount of reference code for all kinds of image processing applications.

Here we are going to see how we can write our own shader, and run it on the GPU.

1.5.1 What is GLSL

The **OpenGL Shading Language** is the principal shading language for **OpenGL** with a straight-forward syntax inspired by C.

Every **GLSL** shader is a self-contained program, to be run on the GPU.

1.5.2 Getting started

Fire up **Cascade** and load an image in the **Read Node**. You do this by double-clicking the node and then clicking on the **Load** button.

To view the image, with the **Read** node still selected, press **F4**.

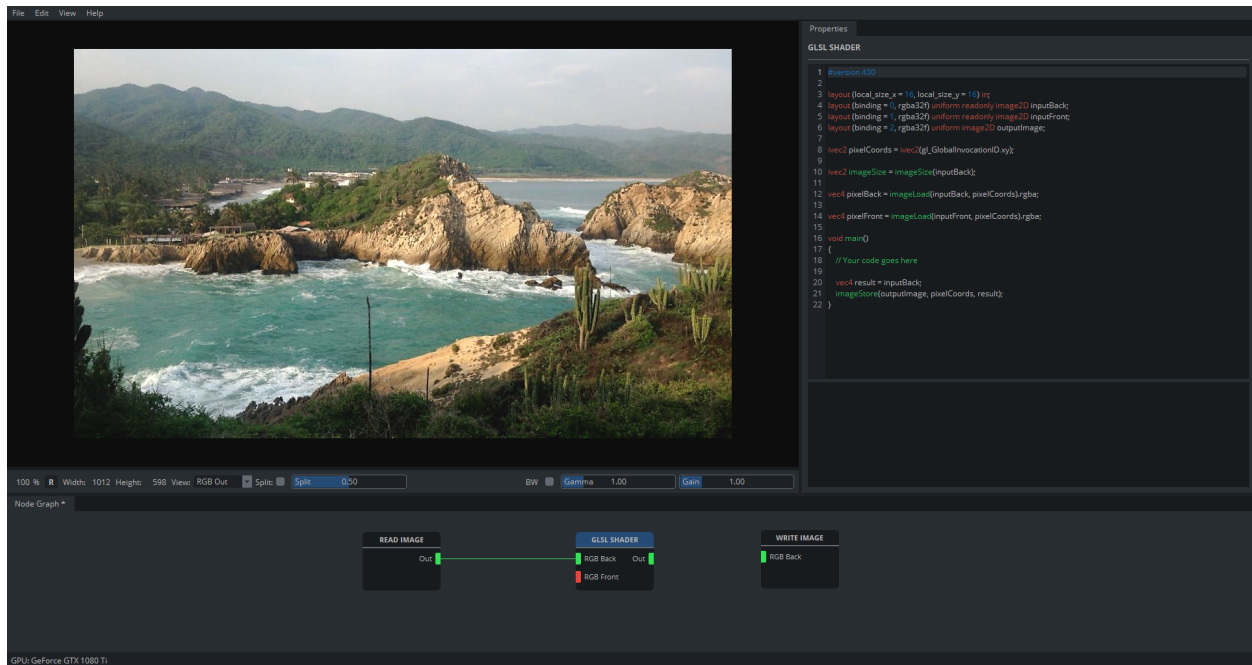
Now, we need to create a **GLSL Shader** node. Right click on the node graph (bottom window) and in the menu choose **Filter > GLSL Shader**.

Connect the output from the **Read** node to the **RGB back** input of the **GLSL Shader** node. Click the **GLSL Shader** node and press **F4** to view it.

The image should be the same as before, since the default shader does not do anything except for passing the image through.

On the right side you will now see the code editor, containing the default shader. You can click and drag to expand the window, so you can see the code a little better.

Your setup should look similar to this:



1.5.3 The default shader

The code editor contains the following:

```
#version 430

layout (local_size_x = 16, local_size_y = 16) in;
layout (binding = 0, rgba32f) uniform readonly image2D inputBack;
layout (binding = 1, rgba32f) uniform readonly image2D inputFront;
layout (binding = 2, rgba32f) uniform image2D outputImage;

ivec2 pixelCoords = ivec2(gl_GlobalInvocationID.xy);
ivec2 imgSize = imageSize(inputBack);
vec2 pixelCoordsNorm = vec2(float(pixelCoords.x) / imgSize.x, float(pixelCoords.y) /
↪imgSize.y);

vec4 pixelBack = imageLoad(inputBack, pixelCoords).rgba;
vec4 pixelFront = imageLoad(inputFront, pixelCoords).rgba;

void main()
{
    // Your code goes here

    vec4 result = inputBack;
```

(continues on next page)

(continued from previous page)

```
imageStore(outputImage, pixelCoords, result);
}
```

Most of the things here you don't have to worry about, let's go through the important parts.

```
layout (local_size_x = 16, local_size_y = 16) in;
layout (binding = 0, rgba32f) uniform readonly image2D inputBack;
layout (binding = 1, rgba32f) uniform readonly image2D inputFront;
layout (binding = 2, rgba32f) uniform image2D outputImage;
```

In the beginning the inputs and outputs are declared. There are two images on the input side and one on the output. Since we only have the back input connected in the node graph, we are only going to use the **inputBack** image.

Following are a couple of convenience functions, to make life easier.

```
ivec2 pixelCoords = ivec2(gl_GlobalInvocationID.xy);
```

This gets the current pixel coordinates and stores them in the integer vector **pixelCoords**.

A GLSL shader gets executed for every pixel in the image. This variable tells us which pixel we are currently working on.

```
ivec2 imgSize = imageSize(inputBack);
```

This gets the image size and stores it in the variable **imgSize** for later use.

```
vec2 pixelCoordsNorm = vec2(float(pixelCoords.x) / imgSize.x, float(pixelCoords.y) /
↪imgSize.y);
```

Calculates the normalized pixel coordinates and stores them in **pixelCoordsNorm**.

```
vec4 pixelBack = imageLoad(inputBack, pixelCoords).rgba;
```

Loads the RGBA values of the **back** image, at the current pixel coordinates, into **pixelBack**.

```
vec4 pixelFront = imageLoad(inputFront, pixelCoords).rgba;
```

Loads the RGBA values of the **front** image, at the current pixel coordinates, into **pixelFront**.

Since there is nothing connected to our front input, we ignore this value for the example.

Now, this is where it gets a little more interesting:

```
void main()
{
    // Your code goes here

    vec4 result = pixelBack;
    imageStore(outputImage, pixelCoords, result);
}
```

This is the main function and the entry point for our shader.

```
vec4 result = pixelBack;
imageStore(outputImage, pixelCoords, result);
```

Here you can see that the **inputBack** value is copied into **result** and then saved to the output image via **imageStore**. That's what this shader does, it copies the input to the output, without doing anything else.

1.5.4 Writing our own shader

Now, let's see how we can do something with our image.

If you change the line

```
vec4 result = pixelBack;
```

to

```
vec4 result = 1.0 - pixelBack;
```

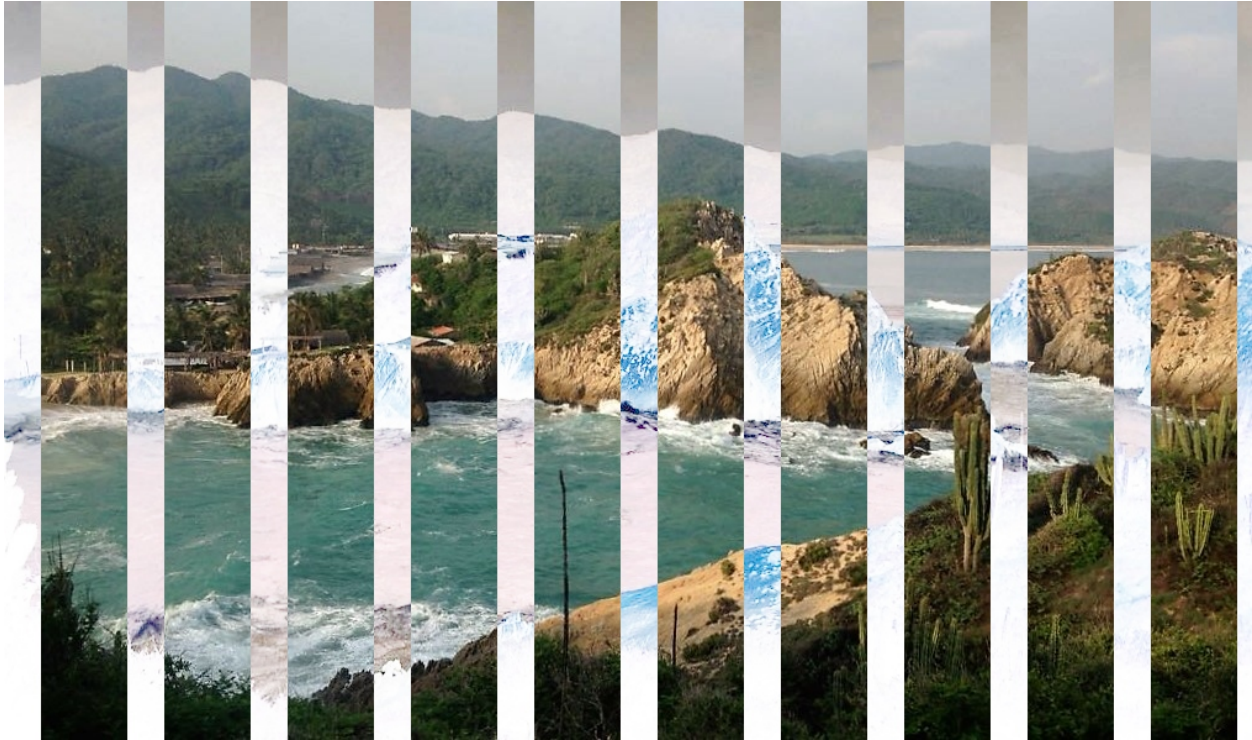
you will see that this inverts our image.



Let's say we want some inverted vertical stripes, we could do something like this:

```
vec4 result = pixelBack;  
  
if (pixelCoords.x % 100 < 30)  
{  
    result = 1.0 - pixelBack;  
}
```

which gives us this:



Of course, this is a very simple example, but I hope it helps as an explanation on how to create your own effects in **Cascade**.

You could now render your image, using a write node. You can also save your node setup, including any shaders you created by going to **File > Save Project**.

If you need inspiration on shaders or you want to figure out how certain effects are implemented, I recommend checking out [Shadertoy](#) and [ISF](#).

1.6 Using ISF shaders

1.6.1 What is ISF?

From the [ISF Spec](#):

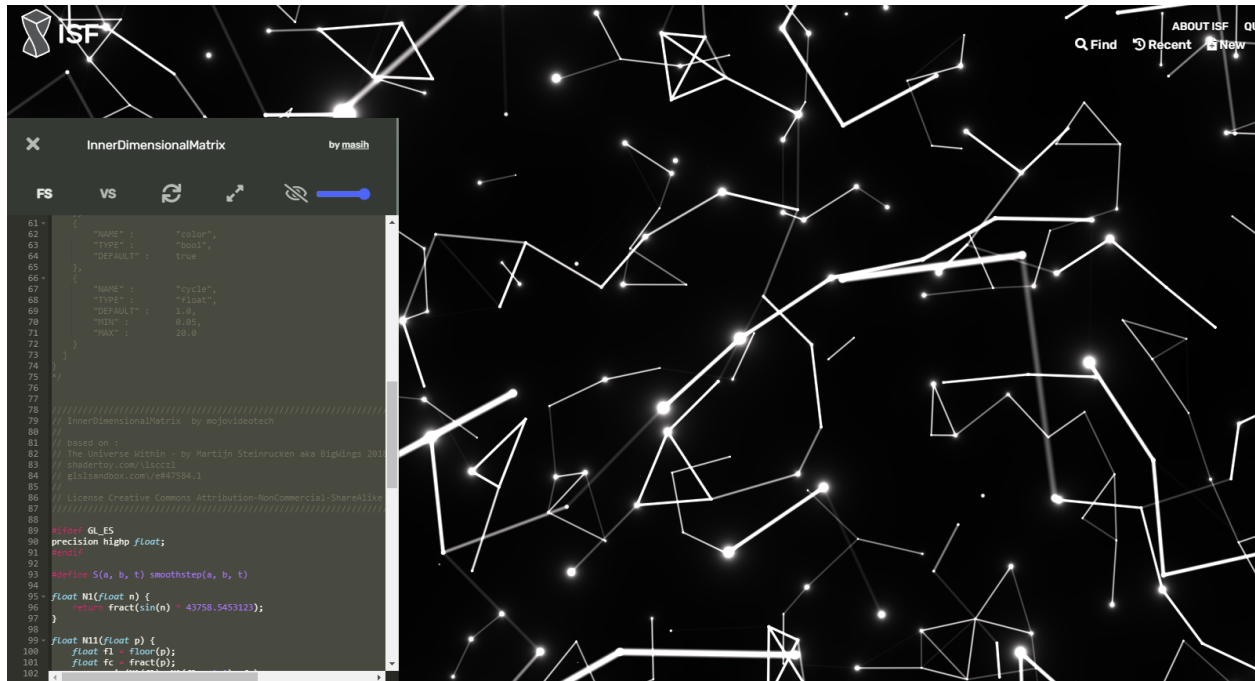
ISF stands for “Interactive Shader Format”, and is a file format that describes a GLSL fragment shader, as well as how to execute and interact with it. The goal of this file format is to provide a simple and minimal interface for image filters and generative video sources that allows them to be interacted with and reused in a generic and modular fashion. ISF is nothing more than a [slightly modified] GLSL fragment shader with a JSON blob at the beginning that describes how to interact with the shader (how many inputs/uniform variables it has, what their names are, what kind of inputs/variables they are, that sort of thing). ISF isn’t some crazy new groundbreaking technology- it’s just a simple and useful combination of two things that have been around for a while to make a minimal- but highly effective- filter format.

1.6.2 How do I use it?

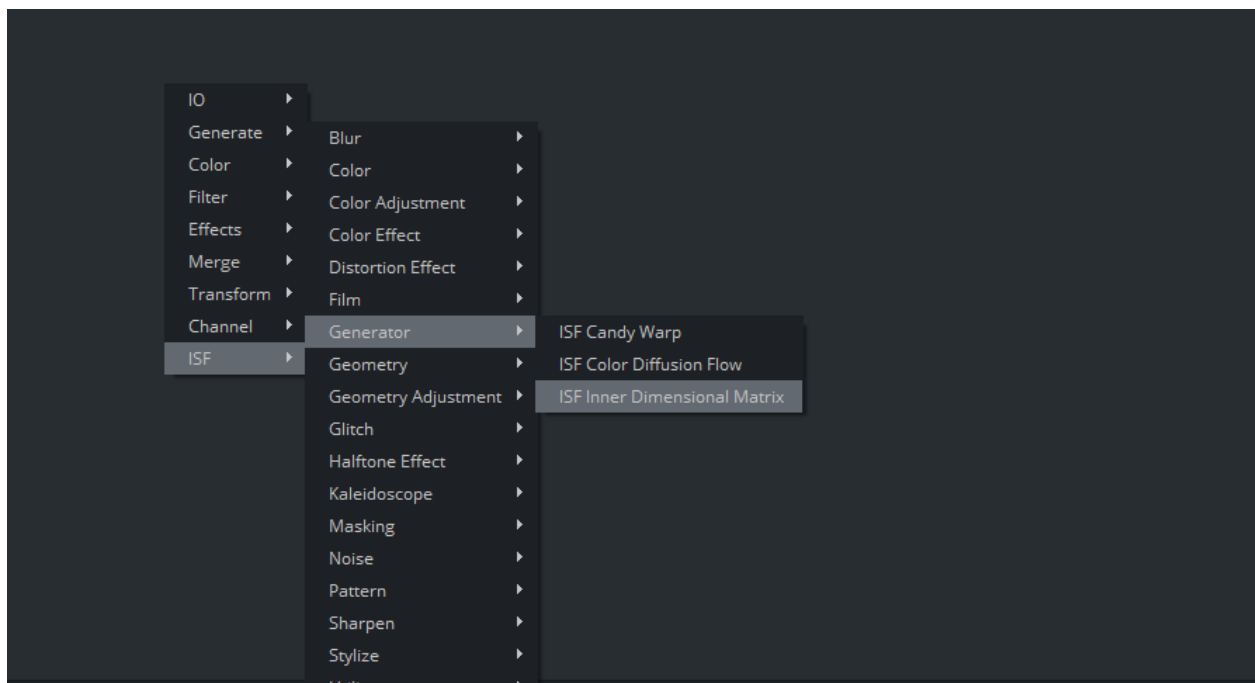
On first start Cascade creates a folder called *isf* which contains all the default shaders that are shipped with the software.

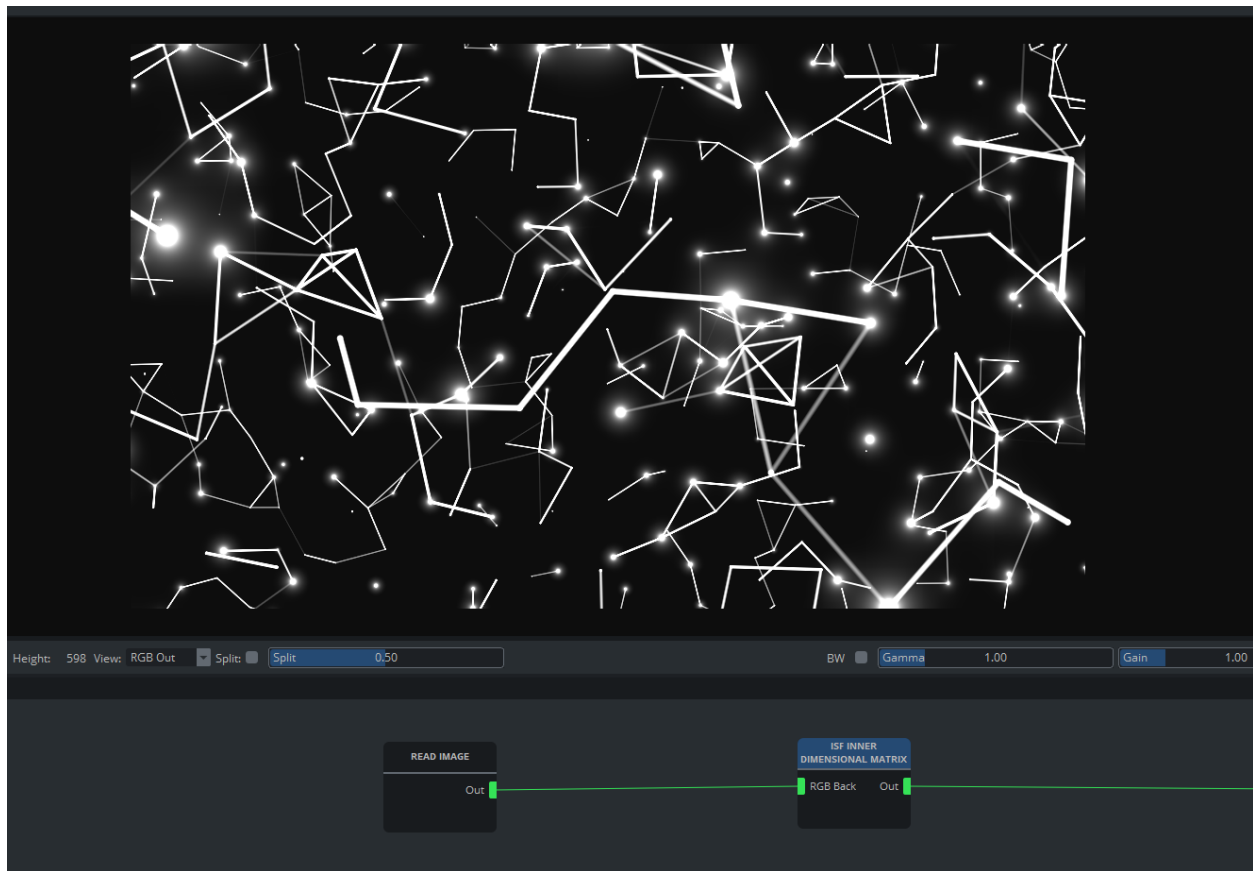
To see the whole shader collection, or write your own, go to editor.isf.video.

If you see an effect that you would like to use, simply copy the code from the editor window on the left to a text file with the ending `.fs`.



Place the text file into the *isf* folder. External shaders are compiled when the application starts, so you will have to restart Cascade and the new shader should show up as a node in the menu.





Note: As of now, the ISF specification has not been fully implemented. What's still to come, in particular, is support for multiple input images and multiple render passes. If you add a shader and it does not show up in the menu, check the log file *Cascade.log* for clues.