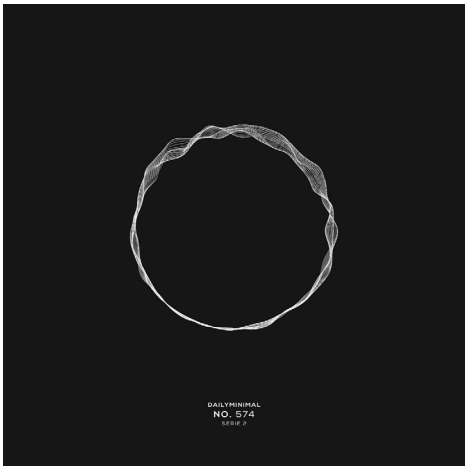
REALTIME REACTIVE VISUALS

MadMapper delivers an intuitive code editor which allows to make realtime visuals

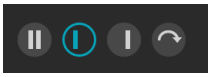
1. Introduction

In this tutorial we will see how to make realtime visuals using MadMapper, like the ones on #1024_Experiments. We will re-create the visual inspired by Daily Minimal, n°574.

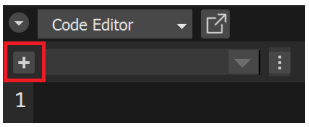
To achieve this, we will need to create a shader with parameters and control them with the MIDI protocol. The music can control and animate the visuals !



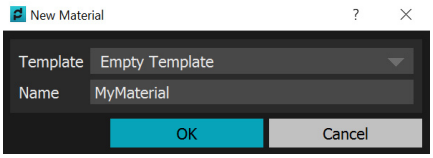
2. Set up



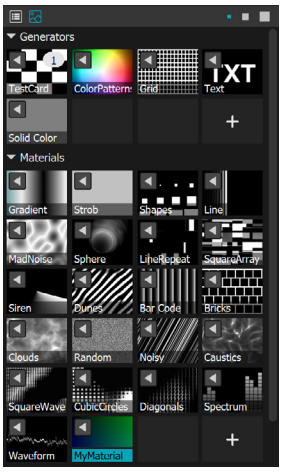
Select the input view by clicking the left bar icon, just below the toolbar. This will allow us to focus only on our visual.



In the code editor, create a new material by click the + icon. A material is a generative animated visual composed of a fragment shader and a vertex shader. In this tutorial we will only use the fragment shader.



Select the Empty Template and name your material whatever you want.



Here the default code should appear in the editor and your material as been added to your materials in the right pannel.

Now create a new quad by clicking the square icon on the top left side, the material should be applied to the surface and appear in the view.
If not, select the quad and right click on your material, then click Apply media to selected surfaces.

Now you are ready to code your reactive shader !

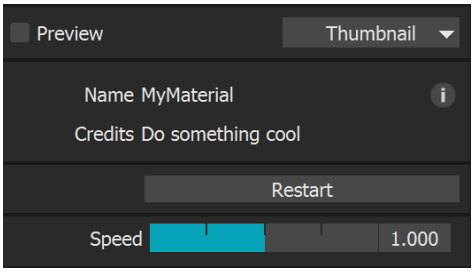
3. Shader creation

A shader is a little program executed by the GPU (Graphic Card).

Here written in GLSL (a programmation language, as part of the OpenGL specifications), it will allow to determine the final color of each rendered pixel on the screen.

3.1. Understanding header

At the start of the code, we can see that part of the text is grayed out and commented (between the characters /* and /*). This part is a JSON header and can directly influence the MadMapper interface (adding sliders, checkbox, description, etc.).



The interface of your shader is shown just bellow the Input Media pannel, we can see the name, credits and a default speed slider. On the info icon we have additionnal informations like the description, tags, path, etc.

Try to modify the credits, descriptions, tags of your shader, the interface is updated in realtime.

Inputs
The INPUTS fields are the uniforms of your shader. Theses are datas passed from the CPU to the GPU, they are only read by the shader. They are called uniforms because they have the same value for every pixel. However they can vary uniformly each frame.

Inputs are composed of a label : the name displayed in the MadMapper interface, a name : the variable name used in the code, a type, which can be float, bool, int, color, etc.
For more informations on data types, go to *https://en.wikipedia.org/wiki/Data_type*

Generators
The GENERATOR are objects with parameters. They can store value and their output is a floating point value. For example the time is a value that we want to get each frame to animate our shader.

Note that you can access MadMapper Materials Documentation from the Help Menu.

Coding the visual
The final color of each pixels is returned by the materialColorForPixel function. It returns a vector4 (red, green, blue and alpha channels). Our code will be contained in this function. These channels are normalized between 0 and 1.

For example, try to put vec4(1, 0, 0, 1) in the final line of your shader, this will display all pixels red (255 for the red channel, 0 for the others and 100% of opacity).

To reproduce the Daily Minimal visual we need to draw multiple circles and apply them a noise which is less and less powerful depending on the Y position. Let's do it !

3.2. Initialisation

First, we need to include the MadSDF library to create basics shapes and the MadNoise library to easily generate a simplex noise.
For more informations on the library, go to *https://www.madmapper.com/doc/materials*

```
#include "MadSDF.glsl"
#include "MadNoise.glsl"
```

These lines will automatically include the Signed Distance Field and Noise functions to your shader codebase.

```
vec3 color = vec3(0.08, 0.08, 0.08);
return vec4(color, 1);
```

Then in the materialColorForPixel function add the background color by creating a vector3 and apply it to the final color.

```
vec2 center = vec2(0.5);
vec2 uv = texCoord - center;
```

After this we need to center the UV.
UV are the coordinates of the current pixel.
In fact, each frame the GPU will execute your shader on all pixels of the screen and determine their colors.
We will use the position of the pixel in UV coordinates to draw our shapes.

By default UV are normalized between 0 and 1 but we will remap their positions between -0.5 and 0.5.

3.3. First circle

```
float circleRadius = 0.25;
```

Time to draw our first circle !
First we will define the radius of our circle, here we will take 0.25.

```
float circle = circle(uv, circleRadius);
```

Then we create the shape of the circle with the circle() function provided by the MadSDF.glsl library. It takes in arguments the position of the pixel and the radius of the circle, things we already have !

```
color = stroke(color, vec3(1), circle, 0.05);
```

After that we need to display our circle to the screen, good news, the MadSDF.glsl library provide a fill() and stroke() function. The stroke function takes 4 arguments in this order : the outside color, the border color, the shape and the stroke thickness.

Now your code should look like this and a white stroked circle should be displayed on a grey background :

```
/*{
   "CREDIT": "1024 architecture",
   "DESCRIPTION": "Inspired by Daily Minimal - NO 574",
   "TAGS": "painting",
   "VSN": "1.0",
   "INPUTS": [
          {"LABEL":"Speed", "NAME": "mat_speed", "TYPE": "float", "MIN": 0.0, "MAX": 2.0, "DEFAULT": 1.0 },
   ],
   "GENERATORS": [
          {"NAME": "mat_time", "TYPE": "time_base", "PARAMS": {"speed": "mat_speed"} },
    ],
}*/

#include "MadSDF.glsl"
#include "MadNoise.glsl"

vec4 materialColorForPixel( vec2 texCoord )
{
   vec3 color = vec3(0.08, 0.08, 0.08);
   vec2 center = vec2(0.5);
   vec2 uv = texCoord - center;

   float circleRadius = 0.25;
   float circle = circle(uv, circleRadius);

   color = stroke(color, vec3(1), circle, 0.05);

    return vec4(color, 1);
}
```

3.4. Adding noise

Let's add a noise to our circle.
First we need to add some inputs to change the frequency and the position of our noise, these sliders will help to refine the values.

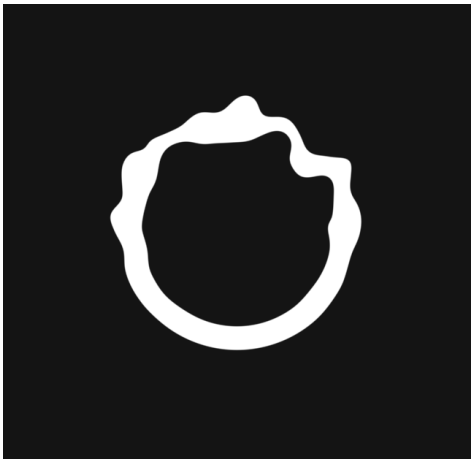
```
{ "LABEL": "Noise/Noise Freq", "NAME": "mat_noiseFrequency", "TYPE": "float", "MIN": 0, "MAX": 10, "DEFAULT": 10 },
{ "LABEL": "Noise/Noise Position", "NAME": "mat_noisePosition", "TYPE": "float", "MIN": 0, "MAX": 1, "DEFAULT": 0.6 },
```

Then we need to make a noise gradient, in fact there is less noise at the bottom of the visual than the top.
To do this we use the smoothstep() function which will ease the noise between 0.3 and 0.7 relative to the Y position of the pixel and the noise position we want.

```
float noisePosition = smoothstep(0.3, 0.7, -uv.y + mat_noisePosition);
```

And then create our noise with the noise() function given by the MadNoise.glsl library which take a vector3. We multiply this noise by our previous noisePosition variable to get a noise less and less powerful from top to bottom.

```
float noiseCircle = noise(vec3(uv * mat_noiseFrequency, mat_time)) * 0.05 * noisePosition;
```



Finally we apply the noise to the circle shape.

```
float circle = circle(uv, circleRadius) + noiseCircle;
```

3.5. Adding more circles

To generate more circles, we will use a for loop.
More informations on the for loop here : *https://en.wikipedia.org/wiki/For_loop*
We define another input to change the number of circles drawn.

```
{ "LABEL": "Circles/Count", "NAME": "mat_circlesCount", "TYPE": "int", "MIN": 1, "MAX": 20, "DEFAULT": 20 },
```

We put our code into a for loop, this will create the number of circles defined by mat_circlesCount :

```
for (int i = 0; i < mat_circlesCount; i++) {
   float noisePosition = smoothstep(0.3, 0.7, -uv.y + mat_noisePosition);
   float noiseCircle = noise(vec3(uv * mat_noiseFrequency, mat_time)) * 0.05 * noisePosition;

   float circleRadius = 0.25;
   float circle = circle(uv, circleRadius) + noiseCircle;

   color = stroke(color, vec3(1), circle, 0.0006);
}
```

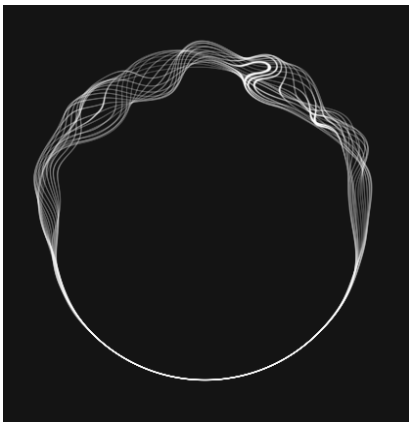
To finish we just have to offset the noise of each circles by adding the iteration to the time.

```
float noiseCircle = noise(vec3(uv * mat_noiseFrequency, i * 0.1 + mat_time)) * 0.05 * noisePosition;
```

And tweak our values like the stroke thickness to 0.0006 for example and play with our Inputs values. At the end you should have a code similar to this :

```
/*{
   "CREDIT": "1024 architecture",
   "DESCRIPTION": "Inspired by Daily Minimal - NO 574",
   "TAGS": "painting",
   "VSN": "1.0",
   "INPUTS": [
           {"LABEL":"Speed", "NAME": "mat_speed", "TYPE": "float", "MIN": 0.0, "MAX": 2.0, "DEFAULT": 1.0 },
           { "LABEL": "Circles/Count", "NAME": "mat_circlesCount", "TYPE": "int", "MIN": 1, "MAX": 20, "DEFAULT": 20 },
           { "LABEL": "Noise/Noise Freq", "NAME": "mat_noiseFrequency", "TYPE": "float", "MIN": 0, "MAX": 10, "DEFAULT": 10 },
           { "LABEL": "Noise/Noise Position", "NAME": "mat_noisePosition", "TYPE": "float", "MIN": 0, "MAX": 1, "DEFAULT": 0.6 },
   ],
   "GENERATORS": [
           {"NAME": "mat_time", "TYPE": "time_base", "PARAMS": {"speed": "mat_speed"} },
    ],
}*/

#include "MadSDF.glsl"
#include "MadNoise.glsl"

vec4 materialColorForPixel( vec2 texCoord )
{
   vec3 color = vec3(0.08, 0.08, 0.08);
   vec2 center = vec2(0.5);
   vec2 uv = texCoord - center;

    for (int i = 0; i < mat_circlesCount; i++) {
           float noisePosition = smoothstep(0.3, 0.7, -uv.y + mat_noisePosition);
           float noiseCircle = noise(vec3(uv * mat_noiseFrequency, i * 0.1 + mat_time)) * 0.05 * noisePosition;

           float circleRadius = 0.25;
           float circle = circle(uv, circleRadius) + noiseCircle;

            color = stroke(color, vec3(1), circle, 0.0006);
    }

    return vec4(color, 1);
}
```

And a result which look like this :



Now you're ready to add MIDI controls to your shader by linking the sliders to Ableton Live or any other MIDI generator application or device.

Thanks for reading, have a good shade !