# MadMapper Materials

MadMapper "Materials" are generative animated visuals. They are directly rendered on the screen by the GPU (graphics card). They are composed of a fragment shader file and optionally a vertex shader file.

MadMapper Materials are based on ISF specification: http://interactiveshaderformat.com/

Because they are not rendered in a texture (or optionally), but directly on the output, the material shader code is mixed with the surface FX (for different surface types: Quad, Surface 3D, Lines or Fixtures). So the "Materials" for MadMapper are not standard ISF shaders. But they use the same jSon header format, same builtin variables (TIME etc.), same macros (IMG_NORM_PIXEL etc.), they can use texture images the same way etc.
It's almost an ISF shader except that you don't implement the function "main", but a function "materialColorForPixel" that will return a color depending on a 2D coordinate.

The Material consist of a folder with a thumbnail (file name must be "thumbnail.jpg" or "thumbnail.png"), a fragment shader (.fs) and optionally a vertex shader (.vs).
The vertex and fragment shaders are the one mixed with a MadMapper internal shader that is specific to the surface type (Quad, Surface 3D, Lines, Fixture etc.) so they must implement a function with a defined name that will be called by the surface FX. Using the vertex shader is sometimes good for performances, if you have some calculations to do that will be the same for all pixels, you'd better doing it once per vertex (6 times for a simple quad) than once per pixel (2 million times in HD resolution).

Shaders are compiled using GLSL version 150 core to be sure they work on a wide range of computers.

Simple example of fragment shader:

```
// This is a Json header following the ISF specifications for defining parameters
// that will appear as a slider, a checkbox, a combox... in MadMapper Material settings
/*{
    "CREDIT": "MM Team",
    "TAGS": "Testing,Useless",
    "INPUTS": [
        {
            "Label": "Red",
            "NAME": "mat_red",
            "TYPE": "float",
            "MIN": 0.0,
            "MAX": 1.0,
            "DEFAULT": 0.5
        },
        {
            "Label": "Green",
            "NAME": "mat_green",
            "TYPE": "float",
            "MIN": 0.0,
            "MAX": 1.0,
            "DEFAULT": 0.5
        },
        {
            "Label": "Blue",
            "NAME": "mat_blue",
            "TYPE": "float",
            "MIN": 0.0,
            "MAX": 1.0,
            "DEFAULT": 0.5
        }
    ]
}*/

// This is the function MadMapper surface FX calls
// texCoord is the texture coordinate (position in the input media for this pixel)
vec4 materialColorForPixel( vec2 texCoord )
{
    return vec4(mat_red,mat_green,mat_blue,1);
}
```

Simple example of vertex shader:

```
// Declare an output variable we can use as input in the fragment shader
// to avoid doing this computation for each pixel!
out float myPrecomputedData;

// This function makes a pseudo random value
float makeRandomValue() {
    return fract(sin(dot(co.xy ,vec2(12.9898,78.233))) * TIME * 43758.5453);
}

// This is the function MadMapper surface FX calls
// uv is the texture coordinate for this vertex (position in the input media)
void materialVsFunc(vec2 uv) {
    myPrecomputedData = makeRandomValue()
}
```

MadMapper provides extensions to the spec. And some restrictions.
We also provide libraries that were written by Simon Geilfus for MadMapper, they are open source, check file headers for more info.
You will find here all details about writing Materials and using MadNoise.glsl and MadSDF.glsl libraries.
Since MadMapper 4.2, Materials can be optionally rendered into a texture. It will give better performances for a complex shader that you might use on high resolution display or on multiple displays. Since version 5.1, it also offers the option to get the last generated frame as input for complex programming. See section "Rasterzation Settings"

## CONVERTING A STANDARD ISF FILE TO A MADMAPPER MATERIAL

It is very easy to convert a standard ISF to a Material. In general renaming the function "void main()" with "vec4 materialColorForPixel(vec2 texCoord)" and use "texCoord" instead of "isf_FragNormCoord"
Example:

```
void main() {
    vec2 uv = isf_FragNormCoord.xy;
    vec4 color;
    ...
    gl_FragColor = color;
}
```

Is replaced with:

```
vec4 materialColorForPixel(vec2 texCoord)
{
    vec2 uv = texCoord;
    vec4 color;
    ...
    return color;
}
```

## RESTRICTIONS AND CONVENTIONS:

- We advise using a convention for INPUT or IMPORTED names: start with "mat_". For instance "mat_frontColor", "mat_backColor"… Some names are reserved. If you use one, MadMapper will show an error message specifying which name you're using is reserved.

Also using "mat_" avoids conflicts if in MadMapper you use a "Surface FX" and a "Material" on the same surface (in which case both contains an ISF header and we should avoid the same name to be used in both cases)

- MadMapper Materials don't support multi-pass ISF (because Materials are generally directly rendered on screen and we might have dozens of video surface in a mapping, it would cause huge performance issues)

- MadMapper supports those uniform variables defined in ISF specification: FRAMEINDEX, TIME, TIMEDELTA, DATE, FRAMEINDEX, RENDERSIZE. Note that RENDERSIZE is only valid if rendering to a texture (it makes no send for a Quad perspective transformed or mesh warped)

## Macros

ISF defines some macros to sample in the media (ie IMG_NORM_PIXEL(vec2 fragCoord)).
MadMapper adds some other ones:

- vec4 IMG_NORM_PIXEL_LOD(sampler2D sampler, vec2 uv, float lod): will only work if input image "GL_TEXTURE_MIN_FILTER" has been set to "LINEAR_MIPMAP_LINEAR" or other mipmapping mode (see MEDIA section below). Cf https://www.khronos.org/registry/OpenGL-Refpages/gl4/html/textureLod.xhtml
- vec4 IMG_TEXEL_FETCH(sampler2D sampler, ivec2 uv, float lod). This will just call glsl texelFetch function with the same parameters. Cf https://www.khronos.org/registry/OpenGL-Refpages/gl4/html/texelFetch.xhtml

## Includes

MadMapper accepts including glsl files. Ie

```
#include "MyCommonFunctions.glsl"
```

The path in the include statement must be valid relatively to the file containing the include statement, except for the few libraries we bundle with application (for Noises, Signed Distance Field & MadCommon):

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

## INPUTS:

MadMapper extends ISF with a few input types:

### Float Range

A "floatRange" INPUT is used to provide a value range, for instance 0.2 - 0.8. Such input will be transmitted to the shader as a vec2 uniform (ie myInput.x for minimum value & myInput.y) like a point2D. MadMapper will make a "Value Range Widget" in the user interface. The default value is provided as a array of two numeric values, but min & max (optional) are a single numeric value.

```
{
  "NAME": "mat_value_range",
  "LABEL": "Value Range",
  "TYPE": "floatRange",
  "DEFAULT": [0.2,0.8],
  "MIN": 0.0,
  "MAX": 1.0
}
```

## IMPORTED TEXTURES:

The specification of ISF only allows declaring 2D texture. We added a TYPE field to allow loading cube map and 3D textures.
Also we added texture filtering and texture wrapping settings.
Here is a complete example.

```
/*{
    "CREDIT": "mm team",
    "CATEGORIES": [ "Image Control" ],
    "INPUTS": [
        ...
    ],
    "IMPORTED": [
        {   "NAME": "mat_myImage",
            "TYPE": "2D",
            "PATH": "image.png",
            "GL_TEXTURE_MIN_FILTER": "LINEAR",
            "GL_TEXTURE_MAG_FILTER": "LINEAR",
            "GL_TEXTURE_WRAP": "REPEAT"
        },
        {   "NAME": "mat_myCubeMapWithFiles",
            "TYPE": "cube",
            "PATH": ["posx.png","negx.png","posy.png","negy.png","posz.png","negz.png"]
        },
        {   "NAME": "mat_myCubeMapWithFolder",
            "TYPE": "cube",
            "PATH": "cubeMadFolder"
        },
        {   "NAME": "mat_myTexture3d",
            "TYPE": "3D",
            "PATH": "colorGrading.png",
            "DEPTH": 32
        }
    ]
}*/
```

"mat_myImage" is the name of the uniform sample ("uniform sampler2D mat_myImage;" is inserted in the final shader - or sampler3D / sampleCube)
Parameters:

- "TYPE": "2D, "CUBE", "3D"
- "PATH": path of the file, with an exception for type CUBE, where PATH can be either an array of 6 files (ordered posx, negx, posy, negy, posz & negz) or the relative path to a folder that contains 6 images named osx, negx, posy, negy, posz & negz (with extension ".jpg" or ".png")
- "GL_TEXTURE_MIN_FILTER": refer to https://www.khronos.org/opengles/sdk/docs/man/xhtml/glTexParameter.xml
  possible values: "LINEAR", "NEAREST", "LINEAR_MIPMAP_LINEAR", "NEAREST_MIPMAP_NEAREST", "LINEAR_MIPMAP_NEAREST", "NEAREST_MIPMAP_LINEAR"
- "GL_TEXTURE_MAG_FILTER": refer to https://www.khronos.org/opengles/sdk/docs/man/xhtml/glTexParameter.xml
  possible values: "LINEAR", "NEAREST"
- "GL_TEXTURE_WRAP": refer to https://www.khronos.org/opengles/sdk/docs/man/xhtml/glTexParameter.xml
  possible values: "REPEAT", "CLAMP_TO_EDGE", "MIRRORED_REPEAT"
- "DEPTH": depth of the 3D texture. This parameter is optional.
  If not defined, we except a texture with Width = Height * Height, ie 256x32 pixels composed of 32 squares of 32x32 pixels next to each other.
  If specifying a depth, image width must be a multiple of Depth. If DEPTH is 64 and the full image is 256x32, it will make a 16x32x64 texture (width=16, height=32, depth=64)

ISF spec defines IMG_PIXEL, IMG_NORM_PIXEL and IMG_THIS_PIXEL macros to sample a color in the media. In case you specify the GL_TEXTURE_MIN_FILTER to one of the mipmapping mode (the ones containing ...*MIPMAP*...), MadMapper will add some more:

- IMG_NORM_PIXEL_LOD(sampler2D imageName, vec2 uv, float lod): image name is the name of your inported image, uv is the coordinate in the media in range (0,0)-(1,1), lod is the level of detail requested, check: https://www.khronos.org/registry/OpenGL-Refpages/gl4/html/textureLod.xhtml
- IMG_TEXEL_FETCH(sampler2D imageName, ivec2 uv, float lod): image name is the name of your inported image, uv is the coordinate in the media in range (0,0)-(width,height), lod is the level of detail requested, check: https://www.khronos.org/registry/OpenGL-Refpages/gl4/html/texelFetch.xhtml

## GENERATORS

With shaders, we can't store a value that we'll reuse when processing next frame (except using multiple render passes and storing a values in pixels of a persistent buffer, but it makes things a bit complicated to solve a simple problem and MadMapper doesn't support render passes for materials since if you have 400 surfaces using the same material, you wouldn't want to do 400 more render passes just to handle a proper animation…)

This problem is inherent to the fact that the shader code is executed in parallel for each vertex/pixel.

To animate something in a shader, you might write something like:

```
float linePosition = sin(mat_speed * TIME);
```

Where mat_speed is an "INPUT" uniform controlled by user & TIME is the time in seconds (as defined in the ISF spec).
The problem with that is that when changing the speed, we got jumps in the animation, ie:

```
frame N: TIME=120.0, speed=1 => linePosition=sin(120.0)=0.866
frame N+1: TIME=120.16, speed=1 => linePosition=sin(120.166)=0.864
```

That's ok, now user sets speed to 0.2

```
frame N+2: TIME=120.32, speed=0.2 => linePosition=sin(24.064)=0.407
```

The guy just wanted to reduce animation speed and the animation moved to a very different place, not good !

To solve that we added "GENERATORS". It doesn't only solve this case, it allows adding stuffs to filter an INPUT uniform (ie Damper).

GENERATORS are objects with parameters. Their output is a floating point value. For instance:

```
"GENERATORS": [
    {
        "NAME": "mat_anim_time_base",
        "TYPE": "time_base",
        "PARAMS": {"speed": "mat_noise_speed", "speed_curve": 2}
    }
]
```

- "NAME": name of the generated uniform float available in the shader code
- "TYPE": type of generator: "time_base", "damper", "adr", "shape_generator", "animator", "pass_thru" (details below)
- "PARAMS": list parameters of the GENERATOR object. All parameters are optional. Parameter values can be either a value, an INPUT (defined above in the header), or another GENERATOR (so it would take its output and process it)

Let's see each GENERATOR type, why they are useful & their parameters:

## Time Base generator "time_base"

"time_base" is a generator that increments its output value (the uniform "move_position" in following example) depending on its "speed" parameter and elapsed time. Between each rendered frame it does something like: "output_value = output_value + speed * elapsedTime"
But there are more parameters, here is a complete example:

```
{
  "NAME": "mat_move_position",
  "TYPE": "time_base",
  "FILTERS": {"speed": "mat_automovespeed",
              "reverse": false,
              "speed_curve":3,
              "strob": 0.0,
              "bpm_sync": false,
              "link_speed_to_global_bpm": false,
            }
}
```

Parameters:

- "speed": speed of the time base (floating point value)
- "reverse": invert speed value (boolean)
- "speed_curve": the input value will be interpreted as pow(INPUT_VALUE,speed_curve). Using a high speed curve allow to define precisely a low values and to be able to access high values at the same time. If you set a curve of 2 ot other even number, we'll take care to keep the sign (a negative number will stay negative)
- "strob": the output value will not be updated on each frame, but each "strob" seconds, ie if strob=1, each 1 second, if strob=0.1, 10 times per second.
- "bpm_sync": that's a handy one. The application has a global BPM, so if you set this parameter to "true", the speed and strob values will be interpreted as a divider or multiplier of the beat. For instance:
  - If your speed is 1, the output value should be increased by 1 each second
  - If BPM sync is on:
    - If speed is 1 and BPM is 60, the output value will also be increased by 1 each second
    - If speed is 0.5, speed will by quantised on a "power of 2" or "power of two divider" of one beat: 1/64, 1/32, 1/16, 1/8, 1/4, 1/2, 1, 2, 4, 8, 16, 32 … and output value will be increased by "quantised_speed * elapsed_time * BPM/60"
- "link_speed_to_global_bpm": links the speed to MadMapper Global BPM (in Master settings tab). If this is set to "true", speed will be multiplied by global BPM / 120. It has no effect if "bpm_sync" is activated.

## Animator

```
"animator" is a shape generator (Smooth, In, Out, Linear, Cut, Random) with many parameters. We made this cas
```

Parameters:

- Parameters of time_base define how the animation goes
- "shape": possible values:
  - "Smooth": sinusoidal shape each cycle
  - "In": value goes from 0 to 1 in a cycle, then starts again
  - "Out": value goes from 1 to 0 in a cycle, then starts again
  - "Linear": value goes from 0 to 1 in half a cycle, then back to 0 in half a cycle
  - "Cut": value stays half a cycle at 0 then half a cycle at 1
  - "Noise": value will be animated by a Perlin noise

## Pass Thru

Useful to get a channel value (ie BPM value, BPM position, a MIDI/OSC/DMX channel ?) as a uniform, use pass_thru filter with the right channel URL as input.

```
"GENERATORS": [
    {
        "NAME": "mat_bpm_position",
        "TYPE": "pass_thru",
        "PARAMS": {"input_value":"/custom/BPM/bpmPos"}
    }
    {
        "NAME": "mat_bpm_value",
        "TYPE": "pass_thru",
        "PARAMS": {"input_value": "/custom/BPM/bpm"}
    }
]
```

## Damper filter

```
"INPUTS": [
    {
        "Label": "Width X",
        "NAME": "mat_width_x",
        "TYPE": "float",
        "MIN": 0.0,
        "MAX": 1.0,
        "DEFAULT": 0.5
    }
],
"GENERATORS": [
    {
        "NAME": "mat_width_x_damped",
        "TYPE": "damper",
        "PARAMS": {"input_value": "mat_width_x", "hardness": 0.1, "damping": 0.5}
    }
]
```

Parameters:

- "hardness": floating point value, default: 0.1
- "damping": floating point value, default: 0.8

## Attack - Decay - Release filter

```
"GENERATORS": [
  {
    "NAME": "mat_adsr_switch",
        "TYPE": "adsr",
        "PARAMS": {"input_value": "fx_mask", "attack": 0.4, "release": 0.4, "decay": 0.0}
    }
]
```

Useful one, like in audio application for keyboard->audio synth response, but without sustain cause that would have no sense in our case (no one is holding a key ;-).
Parameters:

- "attack": defines how fast output value can increase. If set to 0, when input value is higher than output value, output value will directly take the input value. If set to 1, it will need 10 seconds to go from 0 to 1. The maximum value increase at each

computation is: maxIncrease = 10*pow(1-(0.01+attack*0.99),3) * elapsedSeconds
- "decay": when the input value in decreasing, time in seconds before we start decreasing output value
- "release": defines how fast output value can decrease. If set to 0, when input value is lower than output value (and decay time elapsed), output value will directly take the input value. If set to 1, it will need 10 seconds to go from 1 to 0. The maximum value increase at each computation is: maxDecrease = 10*pow(1-(0.01+release*0.99),3) * elapsedSeconds

## Linear Filter

```
"GENERATORS": [
    {
        "NAME": "mat_linear",
        "TYPE": "linear_filter",
        "PARAMS": {"input_value": "mat_value", "duration": 0.5}
    }
]
```

The "linear_filter" is used to make a linear interpolation from current value to new value in a specified duration. Each time the input value changes, the filter stored the current value and current time, output value will reach new value when the specified duration has elapsed.
Parameters:

- "duration": time in seconds to reach target position

## Ease Filter

```
"GENERATORS": [
    {
        "NAME": "mat_linear",
        "TYPE": "linear_filter",
        "PARAMS": {"input_value": "mat_value", "duration": 0.5}
    }
]
```

The "linear_filter" is used to make a linear interpolation from current value to new value in a specified duration. Each time the input value changes, the filter stored the current value and current time, output value will reach new value when the specified duration has elapsed.
Parameters:

- "type": "EaseInOut", "EaseIn" or "EaseOut" (default: "EaseInOut")
- "curve": defines the ease filter curve from 1 to 10 (default: 2)

## Multiplier

Multiplies up to 4 input values.

```
"GENERATORS": [
    {
        "NAME": "mat_multiplier",
        "TYPE": "multiplier",
        "PARAMS": {"value1": "mat_value", "value2": 2, "value3": 1, "value4": 1}
    }
]
```

Parameters:

- "value1", "value2", "value3", "value4", each input default is 1, the output is the multiplication of those 4 inputs.

### Incrementer

Increments or decrements a value.

```
 "GENERATORS": [
    {
        "NAME": "mat_incrementer",
        "TYPE": "incrementer",
        "PARAMS": {"increment": "mat_plus_one", "decrement": "mat_less_one"}
    }
]
```

Parameters:

- "increment": should an "event" INPUT, each time it's triggered it will increase the output by one
- "decrement": should an "event" INPUT, each time it's triggered it will decrease the output by one

## Rasterzation Settings

Since MadMapper 4.2, user can activate the "Render To Texture" option on a Material, and define render width/height (in pixels). The default values can be set in the ISF header.
Since MadMapper 5.1, a Material can request to get the last generated frame as an input to do recursive programming.
Example:

```
"RASTERISATION_SETTINGS": {
    "DEFAULT_RENDER_TO_TEXTURE": true,
    "DEFAULT_WIDTH": 512,
    "DEFAULT_HEIGHT": 512,
    "DEFAULT_PIXEL_FORMAT": "PF_U8_BGRA",
    "REQUIRES_LAST_FRAME": true
},
```

Note that if "REQUIRES_LAST_FRAME" is set, MadMapper will automatically render to texture and prevent user from disabling "Render To Texture". You can get last frame info using sampler2D mm_LastFrame, for instance if you put some data in the bottom left pixel:

```
ivec2 targetPixelPos = ivec2(texCoord * RENDERSIZE);
vec4 lastFrameInfo = texelFetch(mm_LastFrame,ivec2(0,RENDERSIZE.y-1),0);
```

## Waveform and spectrum textures

We added some parameters compared to ISF spec. In the ISF spec, an "INPUT" of type "audioFFT" will be a texture with the audio spectrum. An "INPUT" with type "audio" will contain the waveform.

```
{
    "NAME": "mat_waveform",
    "TYPE": "audio",
    "SIZE": 512,
},
{
    "NAME": "mat_spectrum_16_decay",
    "TYPE": "audioFFT",
    "SIZE": 16,
    "ATTACK": 0.0,
    "DECAY": 0.4,
    "RELEASE": 0.0
},
{
    "NAME": "mat_spectrum_16",
    "TYPE": "audioFFT",
    "SIZE": 16,
    "ATTACK": 0.0,
    "DECAY": 0.0,
    "RELEASE": 0.2
}
```

Additional settings compared to the spec:

- "SIZE": defines the size of the texture (number of values it will contain). 512 is the maximum. The texture resolution is SIZEx1 (height is always 1 pixel)
- "ATTACK" / "DECAY" / "RELEASE": on spectrum (audioFFT) you can set those parameters and we'll apply an ADR filter for each value of the spectrum texture

## Grouping input using LABEL with "/"

When you add an INPUT, it creates a widget with a label. You can organise those widgets into "Group Boxes" by setting your label to "GroupName/Parameter Label". The group order will be define by their appearance order in ISF header. Example:

"LABEL": "Global/Line Width",
"LABEL": "Global/Smooth",
"LABEL": "Global/BPM Sync",
"LABEL": "Noise/Noise",
"LABEL": "Noise/Amount",
"LABEL": "Noise/Speed",
"LABEL": "Move/Auto Move",
"LABEL": "Move/Size",
"LABEL": "Move/Speed",
"LABEL": "Move/Strob",
"LABEL": "Scale/Auto Scale",
"LABEL": "Scale/Size",
"LABEL": "Scale/Speed",
"LABEL": "Scale/Strob",
"LABEL": "Scale/Shape",

Here we'll have 4 groupboxes: Global, Noise, Move, Scale, in this order. And widgets inside each are also organised by the order of appearance in the ISF header. You can also add more depth, ie:

```
"LABEL": "Scale/Animation/Active"
```

## Inputs flags

- Generate as define: for optimisation purpose, some "INPUT" uniforms can be handled differently by MadMapper. If you set the flag "generate_as_define" on an INPUT, we'll compile the shader with a #define added to specify the actual value of the INPUT. It's especially usefull with INPUTS of type "long" (enumeration). Then in the shader code, instead of writing "if (myValue == 0) {…} else …" you write "#ifdef myValue_IS_FirstPossibleValue … #endif". Check the LinePattern material:

  */{ "INPUTS": [ { "NAME": "mat_animation", "LABEL": "Animation", "TYPE": "long", "VALUES": ["Cross", "Centered Filled","Single","Filled"], "DEFAULT": "Cross", "FLAGS": "generate_as_define" } ] }/*

vec4 materialColorForPixel(vec2 texCoord)
{
#if defined(mat_animation_IS_Cross)
…
#elif defined(mat_animation_IS_Centered_Filled)
…
#elif defined(mat_animation_IS_Single)
…
#else // defined(mat_animation_IS_Filled)
…
#endif
…
}

You can also do it on a bool attribute:

```
{
    "Label": "BassFX/Bass FX1",
    "NAME": "mat_bass_fx1",
    "TYPE": "bool",
    "DEFAULT": false,
    "FLAGS": "generate_as_define,button"
}
...
#if defined(bass_fx1_IS_true)
    ...
#endif
```

- Button: if you declare a "bool" INPUT, it will make a check box widget. Setting the "button" flag will make a Push Button.

  {
  "NAME": "mat_autorotateactive",
  "LABEL": "Rotate/Auto Rotate",
  "TYPE": "bool",
  "DEFAULT": false,
  "FLAGS": "button"
  },

- SpinBox: if you declare an "int" or "float" INPUT, it will make a slider. Setting the "spinbox" flag, MadMapper will make a SpinBox.

  {
  "NAME": "mat_int_value",
  "LABEL": "Int Parameter",
  "TYPE": "int",
  "DEFAULT": 0,
  "MIN": 0,
  "MAX": 10,
  "FLAGS": "spinbox"
  },

- No Alpha: if you declare a "color" INPUT, it will make Color Widget. Setting the "no_alpha" flag, the Colro widget will not handle alpha of the color, it will always be one, which is desired in some cases, ie in Chroma Key shader:

  ```
  {
      "LABEL": "Key Color",
      "NAME": "mat_color",
      "TYPE": "color",
  ```

```
        "DEFAULT": [ 0.0, 1.0, 0.0, 1.0 ],
        "FLAGS": "no_alpha"
    },
```

## Libraries

MadMapper is delivered with 2 major libraries developed by Simon Geilfus: MadNoise and MadSDF. Here is a quick explanation. MadNoise.glsl is a library to generate 2D and 3D noises in different ways. MadSDF.glsl is a library to make 2D shapes easily. SDF stands for Signed Distance Field.

### MadCommon

MadCommon.glsl provides a few common constants & functions: PI, rgb2hsv conversion etc. To use them, just add

```
#include "MadCommon.glsl"
```

Full Header:

```
//! Constants
#define PI 3.1415926535897932384626433832795

//! Returns the luminance of a RGB color
float luma( vec3 color );

//! Returns the luminance of a RGB color
float luminance( vec3 color );

//! Luminance based rgb8/linear conversion
vec3 linearToRgb( vec3 color, vec3 range );

//! Luminance based rgb8/linear conversion
vec3 rgbToLinear( vec3 color, vec3 range )

//! Contrast, saturation, brightness, For all settings: 1.0 = 100% 0.5=50% 1.5 = 150%
vec3 applyContrastSaturationBrightness( vec3 color, float contrast, float saturation, float brightness )

//! Returns HSV values for RGB input color
vec3 rgb2hsv( vec3 color );

//! Returns RGB values for HSV input color
vec3 hsv2rgb( vec3 color );

//! Returns the Hue value for RGB input color
float rgb2hue( vec3 color );

//! Usefull shortcuts
#define saturate(x) clamp( x, 0.0, 1.0 )
#define lerp(x,y,t) mix( x, y, t )
```

### MadNoise

MadNoise.glsl provides many noise functions. For instance, vnoise(xyPos) return a float depending on a vec2. To use them, just add

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

Before your shader code, after ISF header. Most of them can be optimized using a Noise Lookup texture. To use this optimization, just write

```
#define NOISE_TEXTURE_BASED
#include "MadNoise.glsl"
```

And add the Noise lookup texture in the imported textures of your material:

```
/*{
    "CREDIT": "mm team",
    "CATEGORIES": [ "Image Control" ],
    "INPUTS": [
        ...
    ],
    "IMPORTED": [
        {
                    "NAME": "noiseLUT",
                    "PATH": "noiseLUT.png",
                    "GL_TEXTURE_MIN_FILTER": "LINEAR",
                    "GL_TEXTURE_MAG_FILTER": "LINEAR",
                    "GL_TEXTURE_WRAP": "REPEAT"
                }
    ]
}/*
```

Full Header:

```
//! Returns a 2D value noise in the range [0,1]
float vnoise( vec2 p );
//! Returns a 3D value noise in the range [0,1]
float vnoise( vec3 p );

//! Returns a 2D value noise with derivatives in the range [0,1]
vec3 dvnoise( vec2 p );
//! Returns a 3D value noise with derivatives in the range [0,1]
vec4 dvnoise( vec3 p );

//! Returns a 2D simplex noise in the range [-1,1]
float noise( vec2 p );
//! Returns a 3D simplex noise in the range [-1,1]
float noise( vec3 p );

//! Returns a 2D simplex noise with derivatives in the range [-1,1]
vec3 dnoise( vec2 p );
//! Returns a 3D simplex noise with derivatives in the range [-1,1]
vec4 dnoise( vec3 p );

#if ! defined( NOISE_WORLEY_ASHIMA_IMPL )
//! Returns a 2D worley/cellular noise in the range [0,1]
float worleyNoise( vec2 p );
//! Returns a 3D worley/cellular noise in the range [0,1]
float worleyNoise( vec3 p );
#else
//! Returns a 2D worley/cellular noise F1 and F2 in the range [0,1]
vec2 worleyNoise( vec2 p );
//! Returns a 3D worley/cellular noise F1 and F2 in the range [0,1]
vec2 worleyNoise( vec3 p );
#endif

//! Returns a 2D worley/cellular noise with derivatives in the range [0,1]
vec3 dWorleyNoise( vec2 p );
//! Returns a 3D worley/cellular noise with derivatives in the range [0,1]
vec4 dWorleyNoise( vec3 p );

//! Returns a 2D simplex noise with rotating gradients in the range [-1,-1]
float flowNoise( vec2 pos, float rot );
```

```
//! Returns a 2D simplex noise with rotating gradients and derivatives in the range [-1,-1]
vec3 dFlowNoise( vec2 pos, float rot );

//! Returns a 2D Billowed Noise in the range [0,1]
float billowedNoise( vec2 p );
//! Returns a 3D Billowed Noise in the range [0,1]
float billowedNoise( vec3 p );

//! Returns a 2D Billowed Noise with Derivatives in the range [0,1]
vec3 dBillowedNoise( vec2 p );
//! Returns a 3D Billowed Noise with Derivatives in the range [0,1]
vec4 dBillowedNoise( vec3 p );

//! Returns a 2D Ridged Noise in the range [0,1]
float ridgedNoise( vec2 p );
//! Returns a 2D Ridged Noise in the range [0,1]
float ridgedNoise( vec2 p, float ridge );
//! Returns a 3D Ridged Noise in the range [0,1]
float ridgedNoise( vec3 p );
//! Returns a 3D Ridged Noise in the range [0,1]
float ridgedNoise( vec3 p, float ridge );

//! Returns a 2D Ridged Noise with Derivatives in the range [0,1]
vec3 dRidgedNoise( vec2 p );
//! Returns a 2D Ridged Noise with Derivatives in the range [0,1]
vec3 dRidgedNoise( vec2 p, float ridge );
//! Returns a 3D Ridged Noise with Derivatives in the range [0,1]
vec4 dRidgedNoise( vec3 p );
//! Returns a 3D Ridged Noise with Derivatives in the range [0,1]
vec4 dRidgedNoise( vec3 p, float ridge );

//! Returns a 2D Curl of a Simplex Noise in the range [-1,1]
vec2 curlNoise( vec2 v );
//! Returns a 2D Curl of a Simplex Flow Noise in the range [-1,1]
vec2 curlNoise( vec2 v, float t );

//! Returns a 2D Fractal Brownian Motion sum of Simplex Noise in the range [-1,1]
float fBm( vec2 p );
//! Returns a 2D Fractal Brownian Motion sum of Simplex Noise in the range [-1,1]
float fBm( vec2 p, int octaves, float lacunarity, float gain );
//! Returns a 3D Fractal Brownian Motion sum of Simplex Noise in the range [-1,1]
float fBm( vec3 p );
//! Returns a 3D Fractal Brownian Motion sum of Simplex Noise in the range [-1,1]
float fBm( vec3 p, int octaves, float lacunarity, float gain );

//! Returns a 2D Fractal Brownian Motion sum of Simplex Noise in the range [-1,1]
vec3 dfBm( vec2 p );
//! Returns a 2D Fractal Brownian Motion sum of Simplex Noise with Derivatives in the range [-1,1]
vec3 dfBm( vec2 p, int octaves, float lacunarity, float gain );
//! Returns a 3D Fractal Brownian Motion sum of Simplex Noise with Derivatives in the range [-1,1]
vec4 dfBm( vec3 p );
//! Returns a 3D Fractal Brownian Motion sum of Simplex Noise with Derivatives in the range [-1,1]
vec4 dfBm( vec3 p, int octaves, float lacunarity, float gain );

//! Returns a 2D Fractal Brownian Motion sum of Billowed Noise in the range [-1,1]
float billowyTurbulence( vec2 p );
//! Returns a 2D Fractal Brownian Motion sum of Billowed Noise in the range [-1,1]
float billowyTurbulence( vec2 p, int octaves, float lacunarity, float gain );
//! Returns a 3D Fractal Brownian Motion sum of Billowed Noise in the range [-1,1]
float billowyTurbulence( vec3 p );
//! Returns a 3D Fractal Brownian Motion sum of Billowed Noise in the range [-1,1]
float billowyTurbulence( vec3 p, int octaves, float lacunarity, float gain );

//! Returns a 2D Fractal Brownian Motion sum of Billowed Noise with Derivatives in the range [-1,1]
vec3 dBillowyTurbulence( vec2 p );
//! Returns a 2D Fractal Brownian Motion sum of Billowed Noise with Derivatives in the range [-1,1]
vec3 dBillowyTurbulence( vec2 p, int octaves, float lacunarity, float gain );
//! Returns a 3D Fractal Brownian Motion sum of Billowed Noise with Derivatives in the range [-1,1]
vec4 dBillowyTurbulence( vec3 p );
//! Returns a 3D Fractal Brownian Motion sum of Billowed Noise with Derivatives in the range [-1,1]
vec4 dBillowyTurbulence( vec3 p, int octaves, float lacunarity, float gain );

//! Returns a 2D Ridged Multi-Fractal sum of Simplex Noise in the range [-1,1]
float ridgedMF( vec2 p );
```

```
//! Returns a 2D Ridged Multi-Fractal sum of Simplex Noise in the range [-1,1]
float ridgedMF( vec2 p, float ridgeOffset );
//! Returns a 2D Ridged Multi-Fractal sum of Simplex Noise in the range [-1,1]
float ridgedMF( vec2 p, float ridgeOffset, int octaves, float lacunarity, float gain );
//! Returns a 3D Ridged Multi-Fractal sum of Simplex Noise in the range [-1,1]
float ridgedMF( vec3 p );
//! Returns a 3D Ridged Multi-Fractal sum of Simplex Noise in the range [-1,1]
float ridgedMF( vec3 p, float ridgeOffset );
//! Returns a 3D Ridged Multi-Fractal sum of Simplex Noise in the range [-1,1]
float ridgedMF( vec3 p, float ridgeOffset, int octaves, float lacunarity, float gain );

//! Returns a 2D Ridged Multi-Fractal sum of Simplex Noise in the range [-1,1]
vec3 dRidgedMF( vec2 p );
//! Returns a 2D Ridged Multi-Fractal sum of Simplex Noise with Derivatives in the range [-1,1]
vec3 dRidgedMF( vec2 p, float ridgeOffset );
//! Returns a 2D Ridged Multi-Fractal sum of Simplex Noise with Derivatives in the range [-1,1]
vec3 dRidgedMF( vec2 p, float ridgeOffset, int octaves, float lacunarity, float gain );
//! Returns a 3D Ridged Multi-Fractal sum of Simplex Noise with Derivatives in the range [-1,1]
vec4 dRidgedMF( vec3 p );
//! Returns a 3D Ridged Multi-Fractal sum of Simplex Noise with Derivatives in the range [-1,1]
vec4 dRidgedMF( vec3 p, float ridgeOffset );
//! Returns a 3D Ridged Multi-Fractal sum of Simplex Noise with Derivatives in the range [-1,1]
vec4 dRidgedMF( vec3 p, float ridgeOffset, int octaves, float lacunarity, float gain );
```

### MadSDF

Check Libraries/MadSDF.md for more info

## Unofficial features

Caution: you shouldn't use those features except if you have very specific needs and you're doing it only for you - NOT FOR SHARING with other people

When writing a Material in MadMapper, you can test what kind of surface is actually using it using those definitions:

- Quads: #ifdef SURFACE_IS_quad
- Circle: #ifdef SURFACE_IS_circle
- Lines: #ifdef SURFACE_IS_lines
- Triangle: #ifdef SURFACE_IS_triangle
- Surface3D: #ifdef SURFACE_IS_surface3D
- DMX Fixture: #ifdef SURFACE_IS_fixture
- DMX Fixture Line: #ifdef SURFACE_IS_fixture_line

You can specify GLSL version in ISF header. By default we use version "150 core" (june 2017, version 3.0.0).

```
/*{
    "CREDIT": "mm team",
    "CATEGORIES": [ "Image Control" ],
    "GLSL_VERSION": "150 core",
    ...
}/*
```