

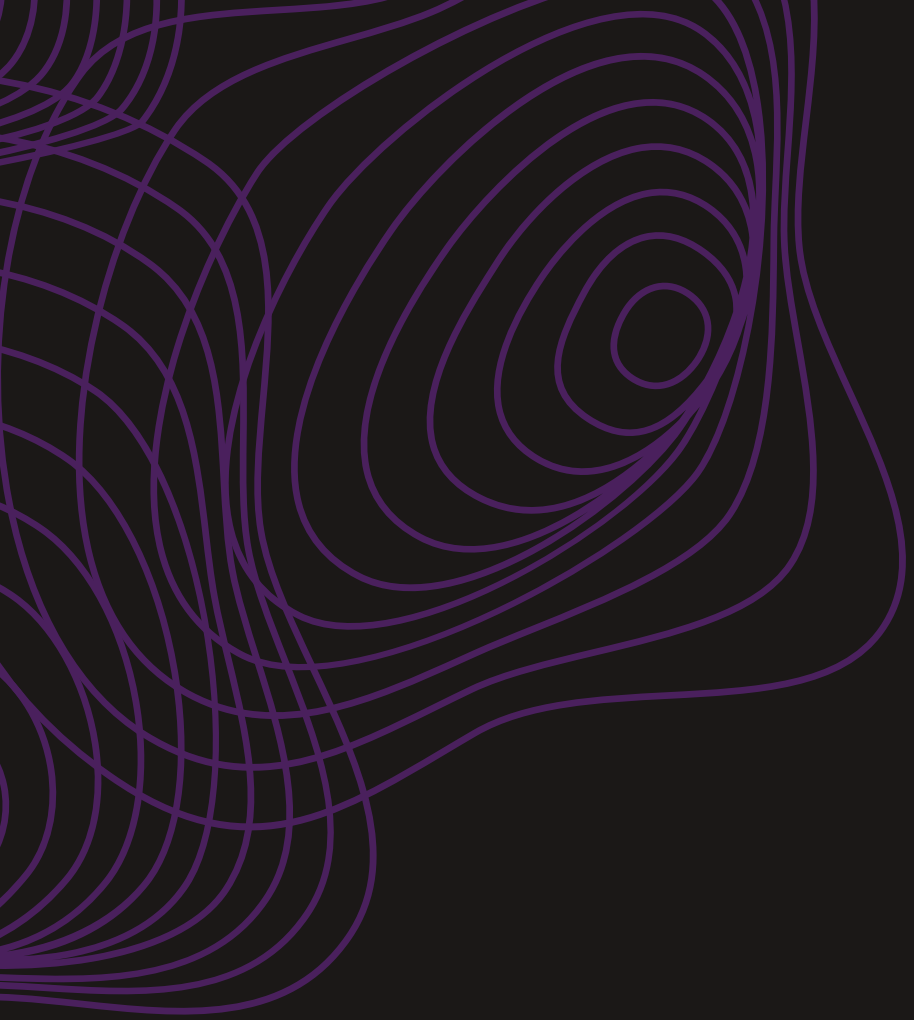
GRAPHICS PROGRAMMING KNIGHTS

Intro to Shader Programming

An introductory exploration of shader concepts, syntax, and examples.

This was
made with a
shader!





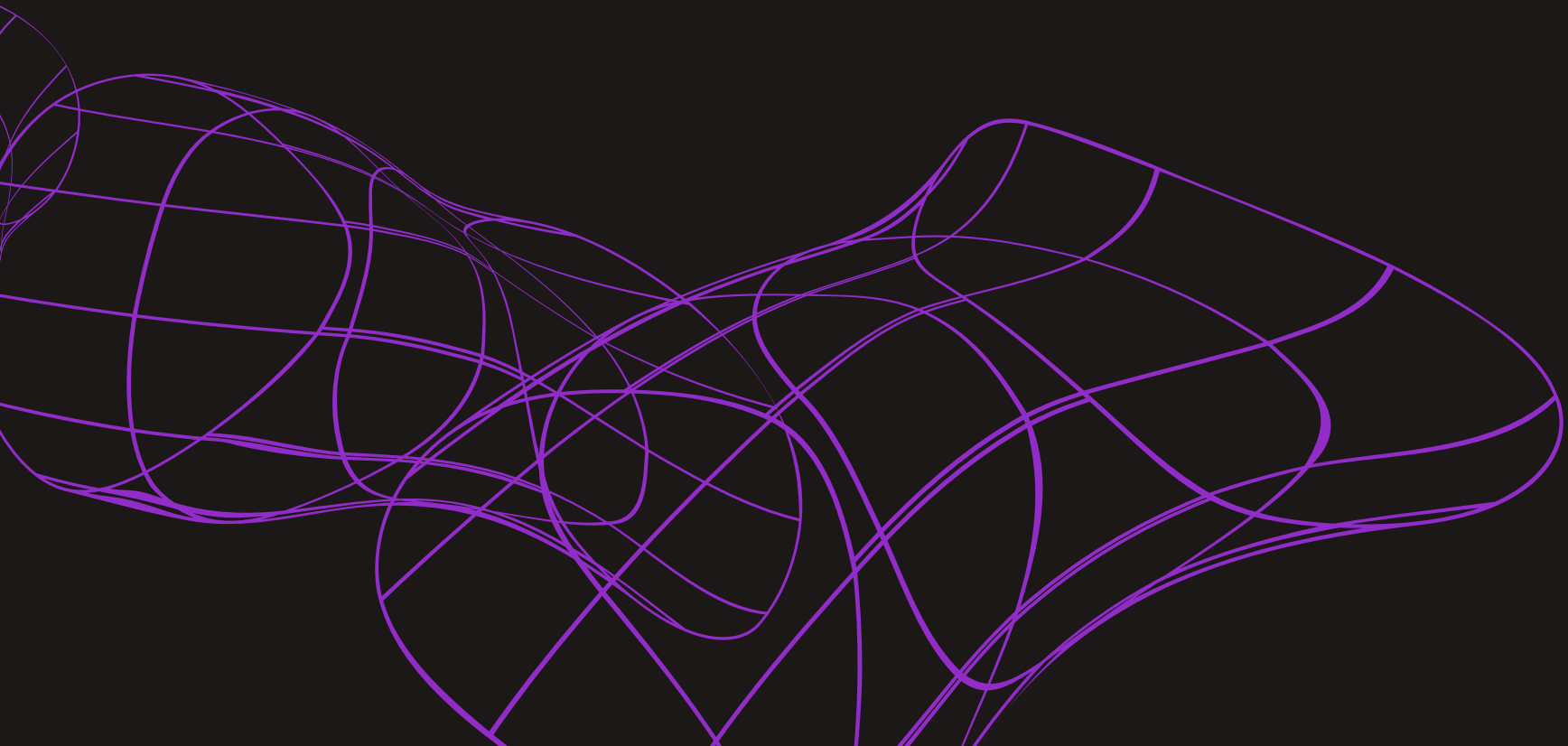
What we will be
using today

shadertoy.com

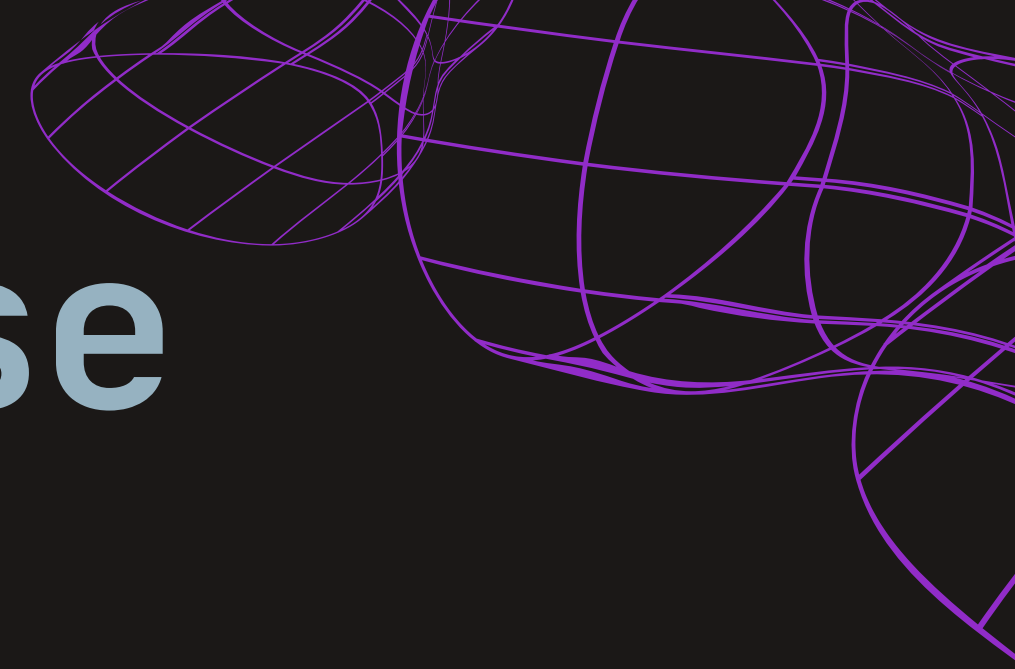


What is a shader?

A shader is a GPU program
that runs on GPU hardware.



What do people use shaders for?



Vertex Shaders

- Perform operations on the vertex and geometry data of a 3D model.

Fragment Shaders

- Perform operations and output a color for every pixel or 'fragment' on a screen.
- Typically the output of the Vertex shader is fed into these shaders.

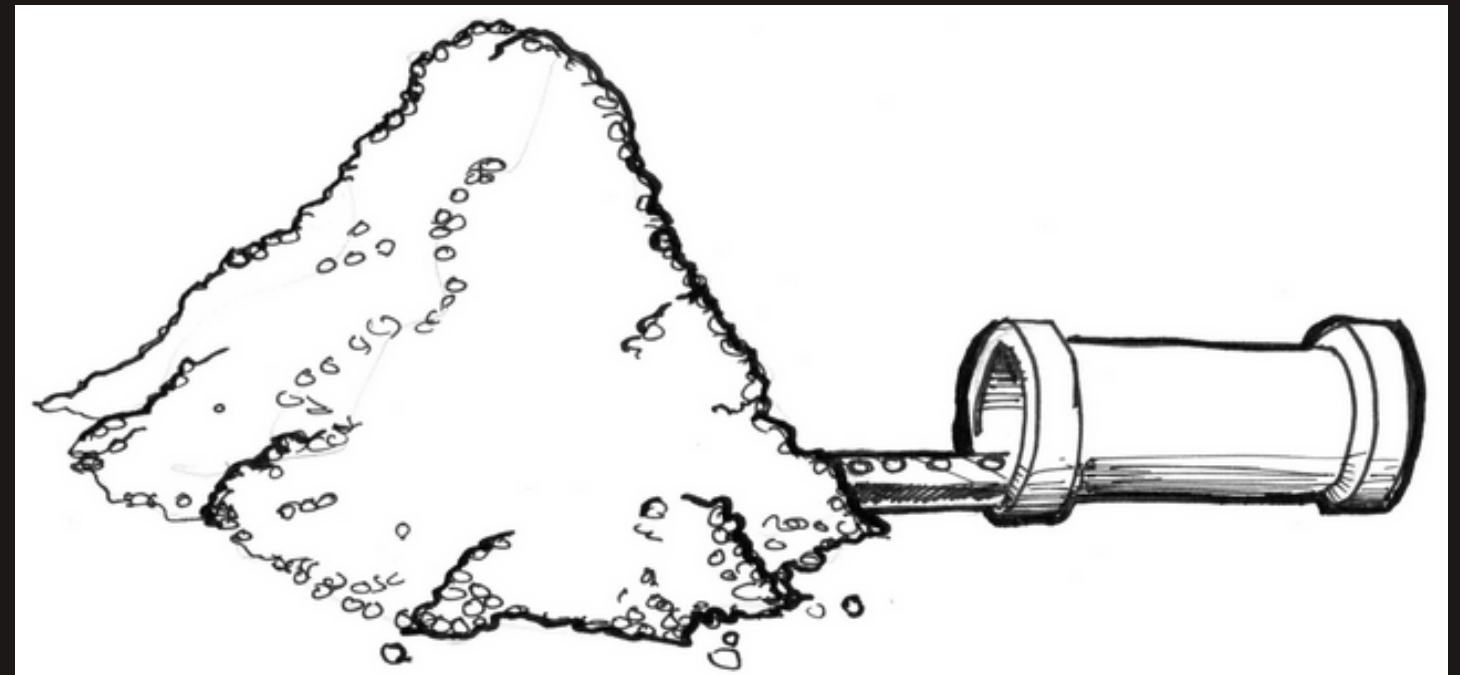
Compute Shaders

- Used to perform any kind of parallel computation.

Why shaders?

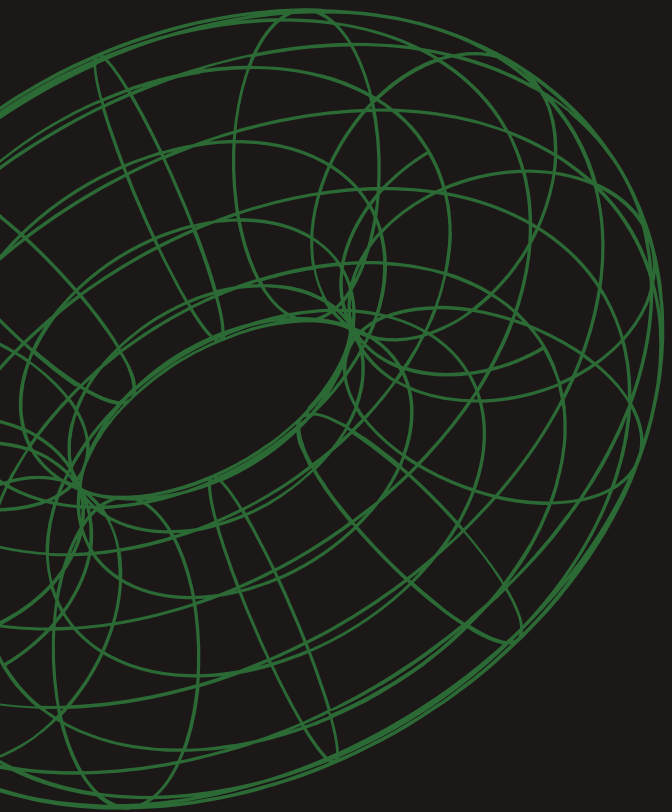
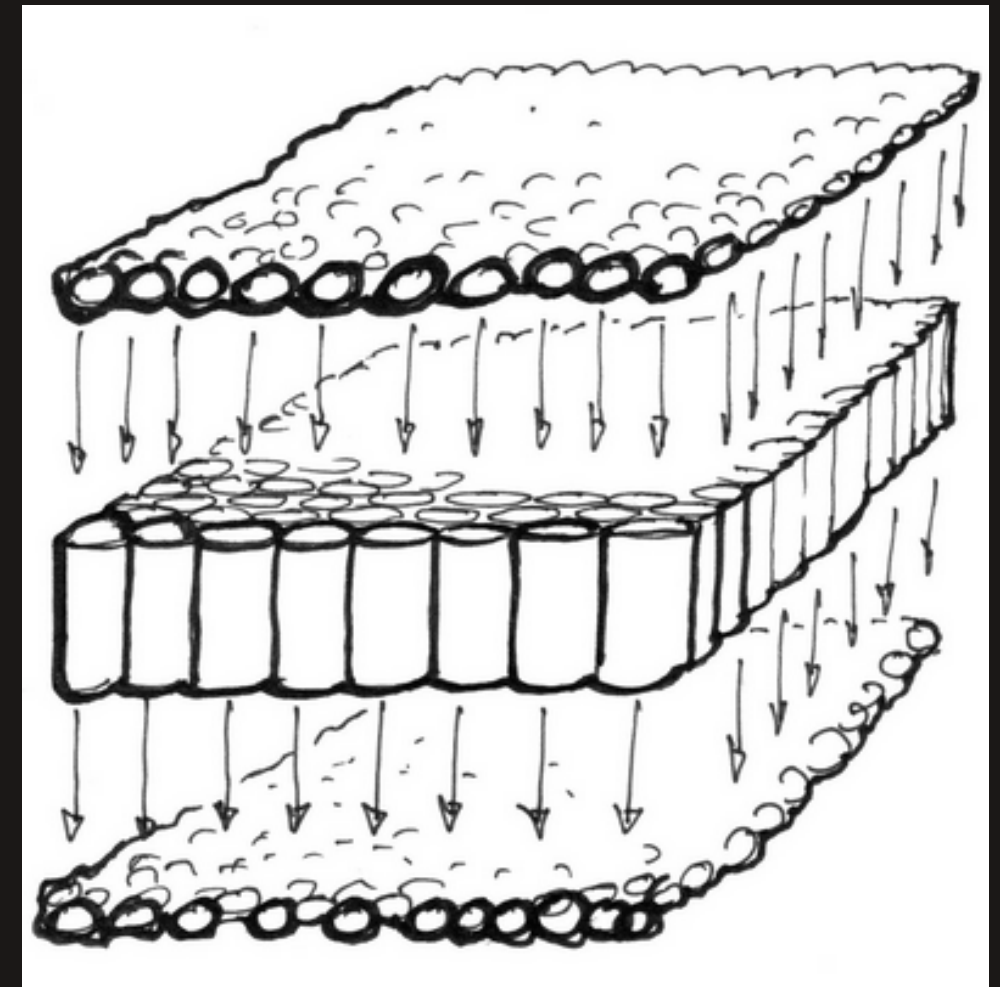
Why write a shader when traditional code gives us the same outputs?

CPU



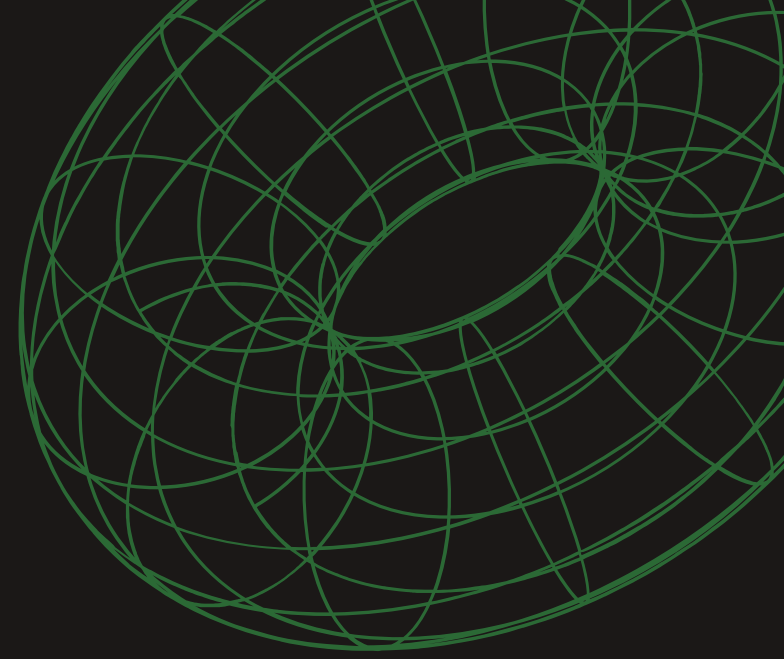
VS.

GPU



Limitations of shader programs

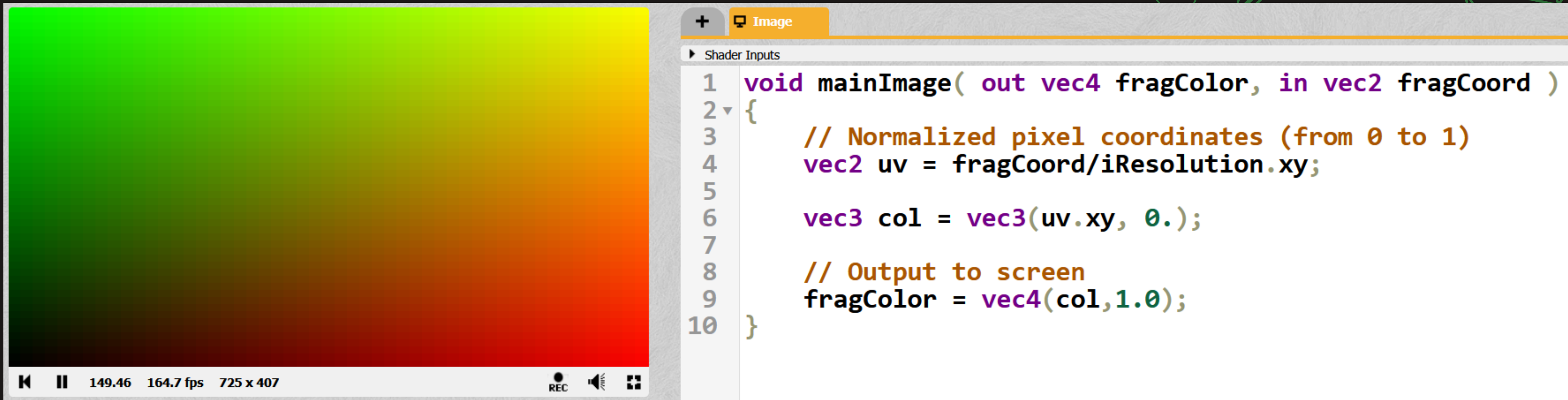
Why is shader programming so weird!



1. Every thread must be independent from every other one.
2. A thread has no 'memory' of its state at a previous moment.

Basically, threads are **BLIND** and **MEMORYLESS**!

Looking at a simple example



- C-like language.
- Single 'main' function that returns a color at the end.
- Built in *variables* (fragColor, fragCoord, iResolution, etc.), *types* (vec2,3,4, etc.), and *functions* (mainImage()).
- Access vectors using
 - .xyzw
 - .rgba
- These accessors can be used in any order, aka, Swizzling. Example: 'uv.yz'

Useful GLSL functions

length(x)

- Returns the scalar length from given n-dimensional vector *x*.

$$\sqrt{x[0]^2 + x[1]^2 + \dots}$$

step(edge, x)

- Compares *edge* to *x*.

if *x* < *edge*, return 0.0
else, return 1.0

mix(x, y, a)

- Linear interpolation between *x* and *y* using *a* to weight between them.

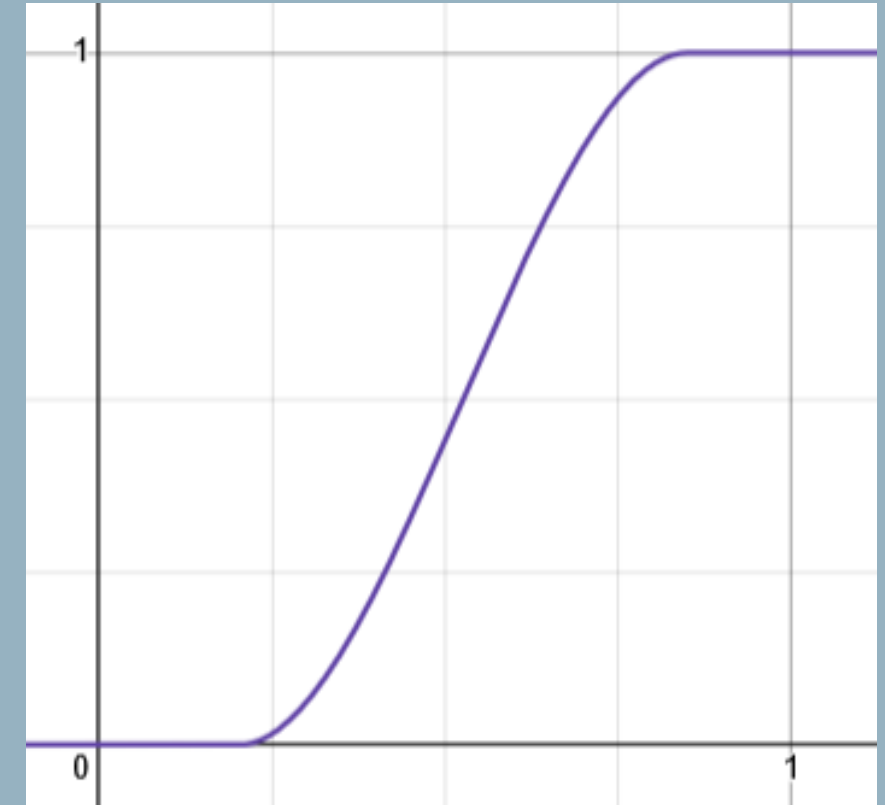
fract(x)

- Returns the fractional part of *x*.
- This is calculated as *x* - *floor*(*x*).

More useful GLSL functions

`smoothstep(edge0, edge1, x)`

- Similar to `step()`, but with a lower and upper bound.
- Smooth interpolation of x between $0-1$ when $edge0 < x < edge1$
- Otherwise, returns 0.0 or 1.0 respectively.



`sin(angle) & cos(angle)`

- Returns the sin/cos of a given angle in radians.

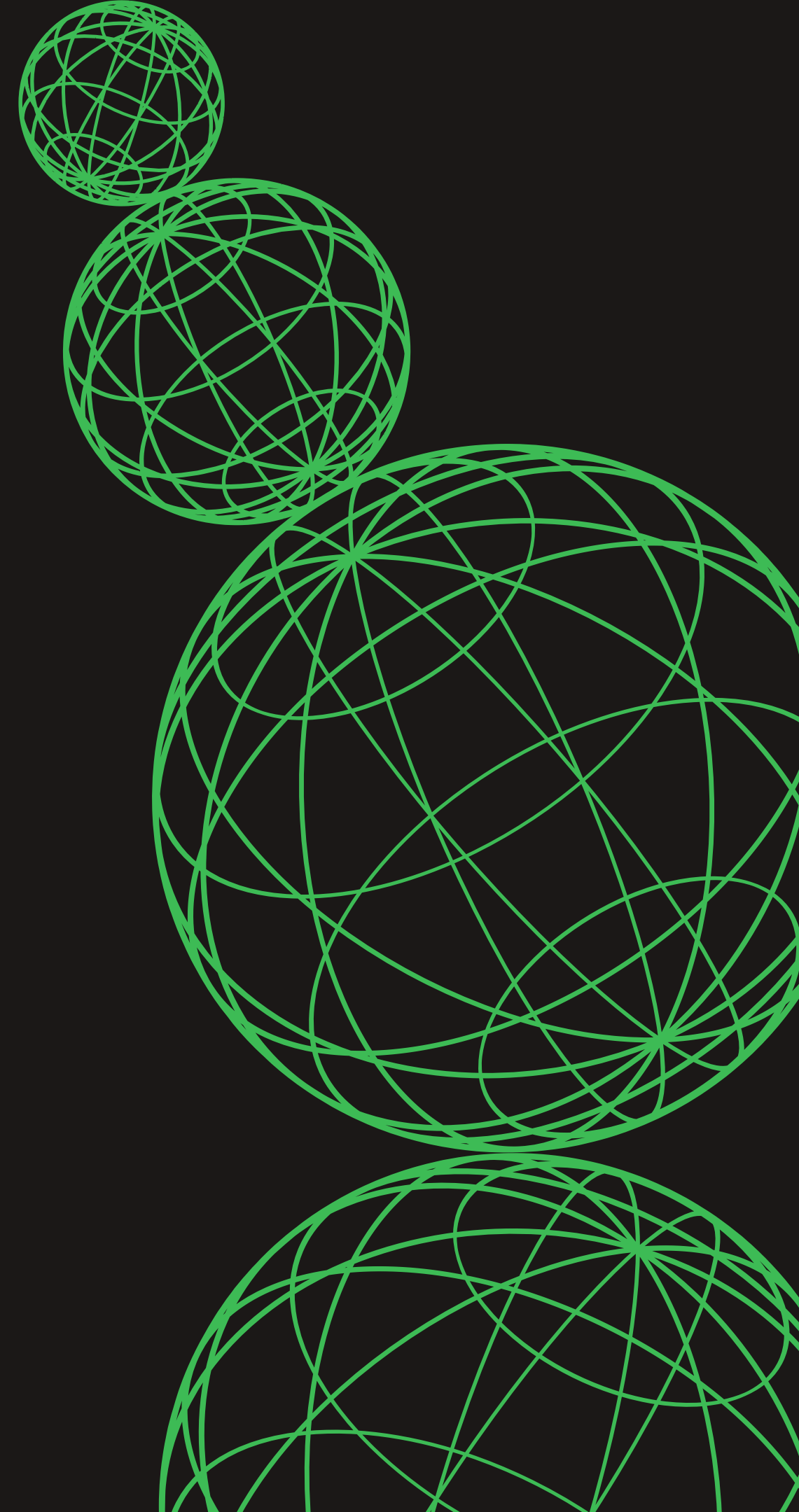
Shaping Functions

To create cool visuals, often times you need to be able to 'shape' your inputs (uv coords, textures, noise, etc.) to get a desired effect.

Shaping functions are just specialized math formulas that help blend, attenuate, clip, or otherwise shape values. Mastering these is the key to making really good shaders.

If you want to dive into cool shaping functions, Inigo Quilez has a great article.

iquilezles.org/articles/functions/



Let's try a
simple shader
together!



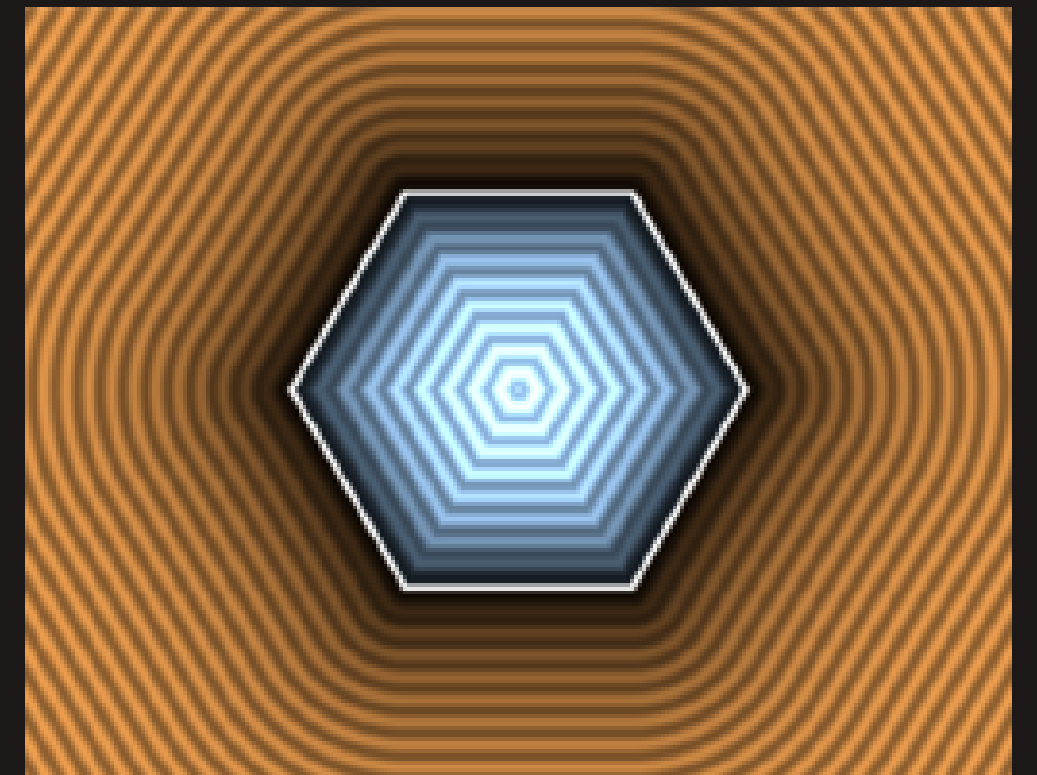
```
1 void mainImage( out vec4 fragColor, in vec2 fragCoord )
2 {
3     vec2 uv = fragCoord.xy; // 0 <> iResolution
4     uv *= 2.; // 0 <> iResolution * 2
5     uv -= iResolution.xy; // -iResolution <> iResolution
6     uv /= iResolution.y; // -1 <> 1 (if our screen is square)
7
8     vec3 col;
9     // col = vec3(uv.xy, 0.);
10
11     float t = abs(sin(iTime));
12     //col = vec3(t);
13
14     vec3 c = mix(vec3(1.,0.,0.), vec3(0.,1.,0.), t);
15     //col = c;
16
17     fragColor = vec4(col, 1.0);
18 }
```

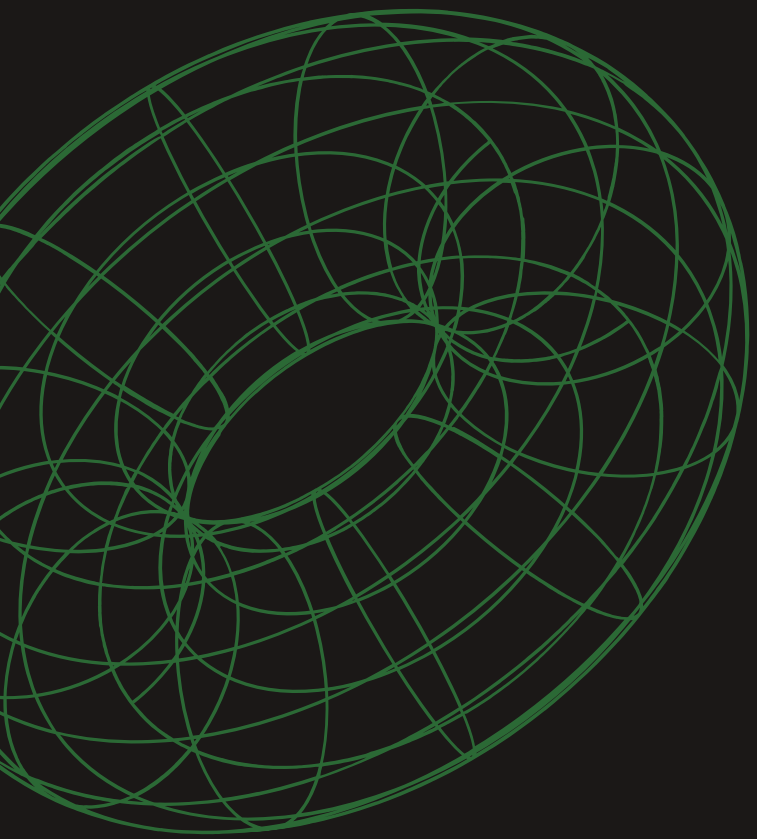
Signed Distance Fields

Signed Distance Fields (SDFs) are used to represent shapes. They take in a position in space as input, and return a value representing if that position is inside of the shape or not.

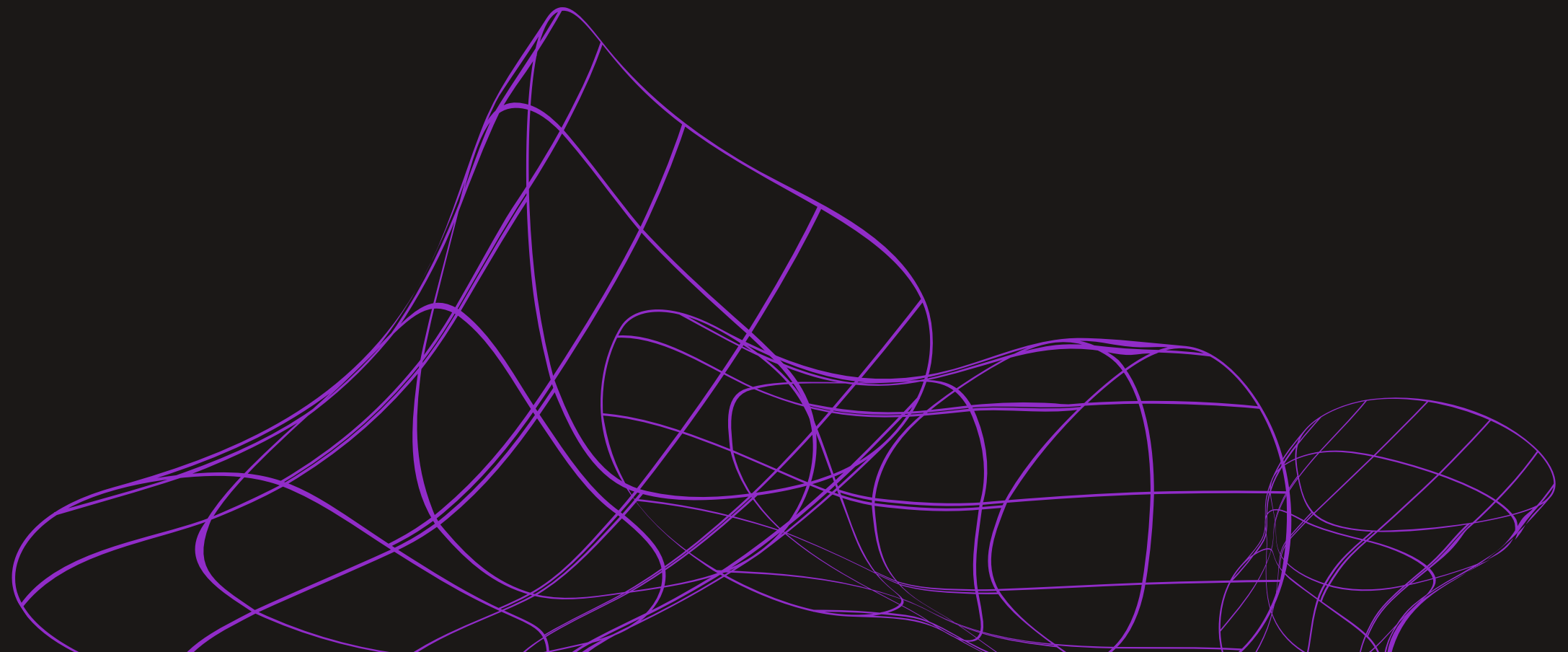
It is called **signed** because the distance is positive outside the shape, negative inside the shape, and zero on the edge.

They are extremely useful for shaders as you can define pretty much any shape you want given some position on your screen.





Let's try another
shader together!



```
1 void mainImage( out vec4 fragColor, in vec2 fragCoord )
2 {
3     // range of -1 <> 1
4     vec2 uv = (fragCoord * 2. - iResolution.xy) / iResolution.y;
5
6     vec3 col;
7
8     float r = 0.7;
9     float d = length(uv) - r; // circle sdf
10    //col = vec3(d);
11
12    float sd = step(0.01, d);
13    //col = vec3(sd);
14
15    float ssd = smoothstep(0.01, 0.1, d);|
16    //col = vec3(ssd);
17
18    fragColor = vec4(col, 1.0);
19 }
```

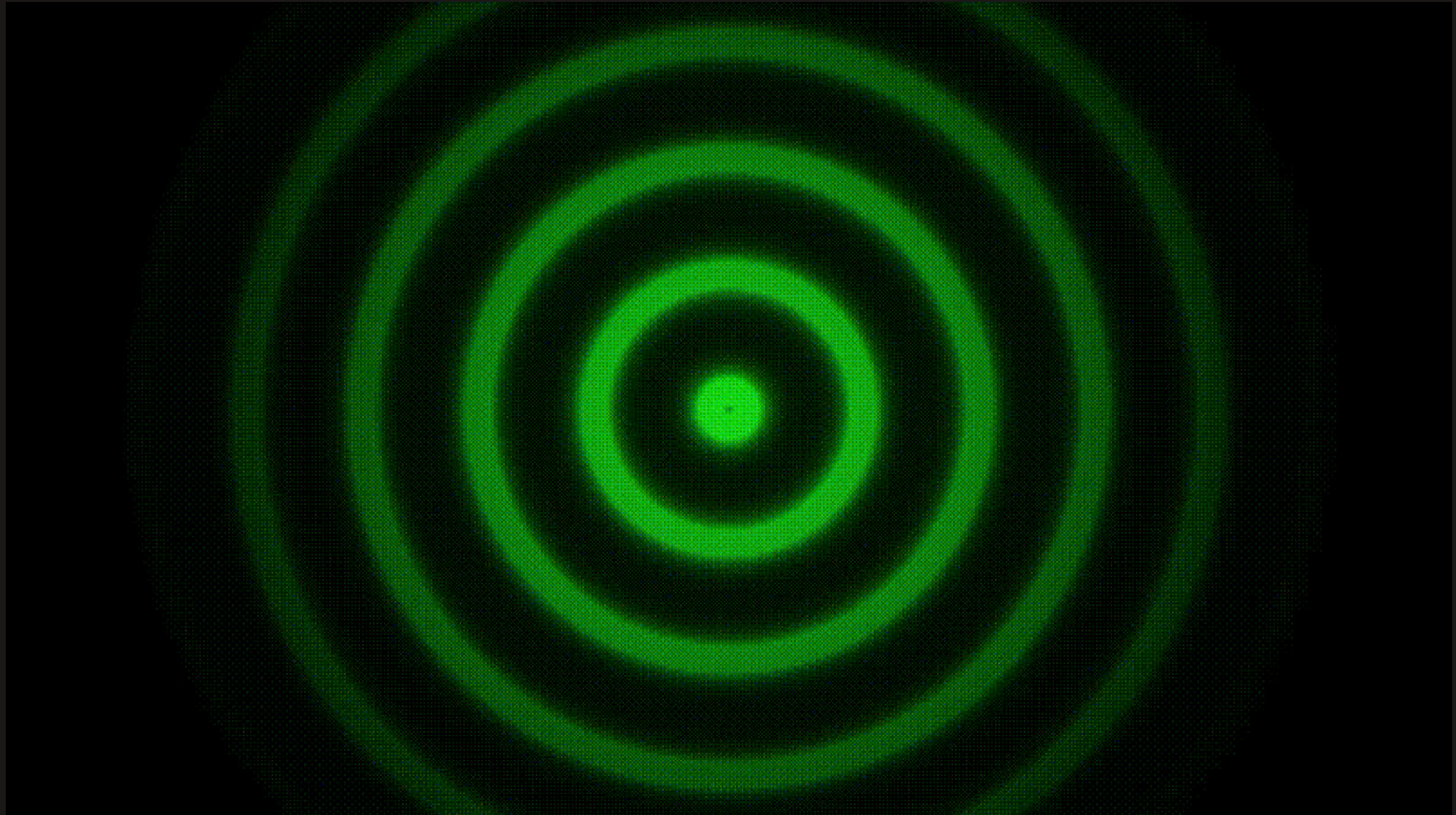
Now you try!

1. Go to shadertoy.com/new
2. Mess with the starter code and notice how things change.

Challenge on the next slide...



Challenge



```
7 void mainImage( out vec4 fragColor, in vec2 fragCoord )
8 {
9     // range of -1 <> 1
10    vec2 uv = (fragCoord * 2. - iResolution.xy) / iResolution.y;
11
12    float d = length(uv); // 0 <> inf.
13
14    float mask = sin(d*11. - iTime); // -1 <> 1
15    mask = abs(mask); // 0 <> 1
16
17    mask = 0.03/mask;
18
19    // smoothstep puts you in 0 <> 1 range
20    mask = smoothstep(0.01, 0.1, mask);
21
22    // interpolation value for color
23    float t = length(uv);
24    t /= 1.5;
25    t = 1.-t;
26
27    vec3 color = vec3(mix(vec3(0.), vec3(0.098, 0.859, 0.086), t));
28
29    color *= mask;
30
31    fragColor = vec4(color,1.0);
32 }
```

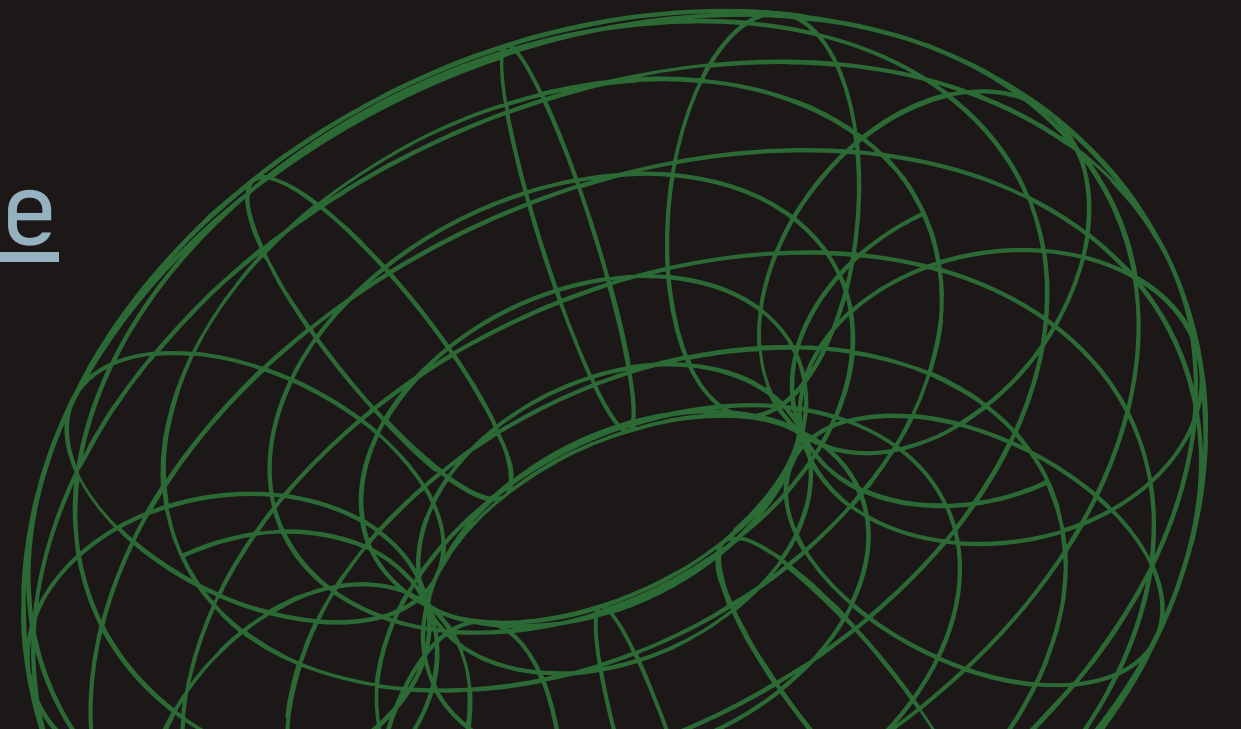
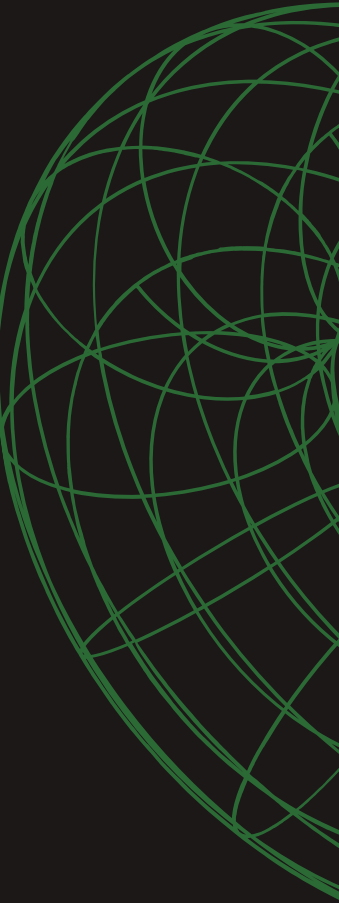

Links for Further Exploration

General Learning for Shaders:

- [An introduction to Shader Art Coding – YouTube](#)
- [thebookofshaders.com](#)
- [iquilezles.org/articles](#)
- [shaderacademy.com](#) (new personal fav)

For my Unity Devs:

- [catlikecoding.com](#)
- [Freya Holmér Shader Tutorials – YouTube](#)





Thank you!