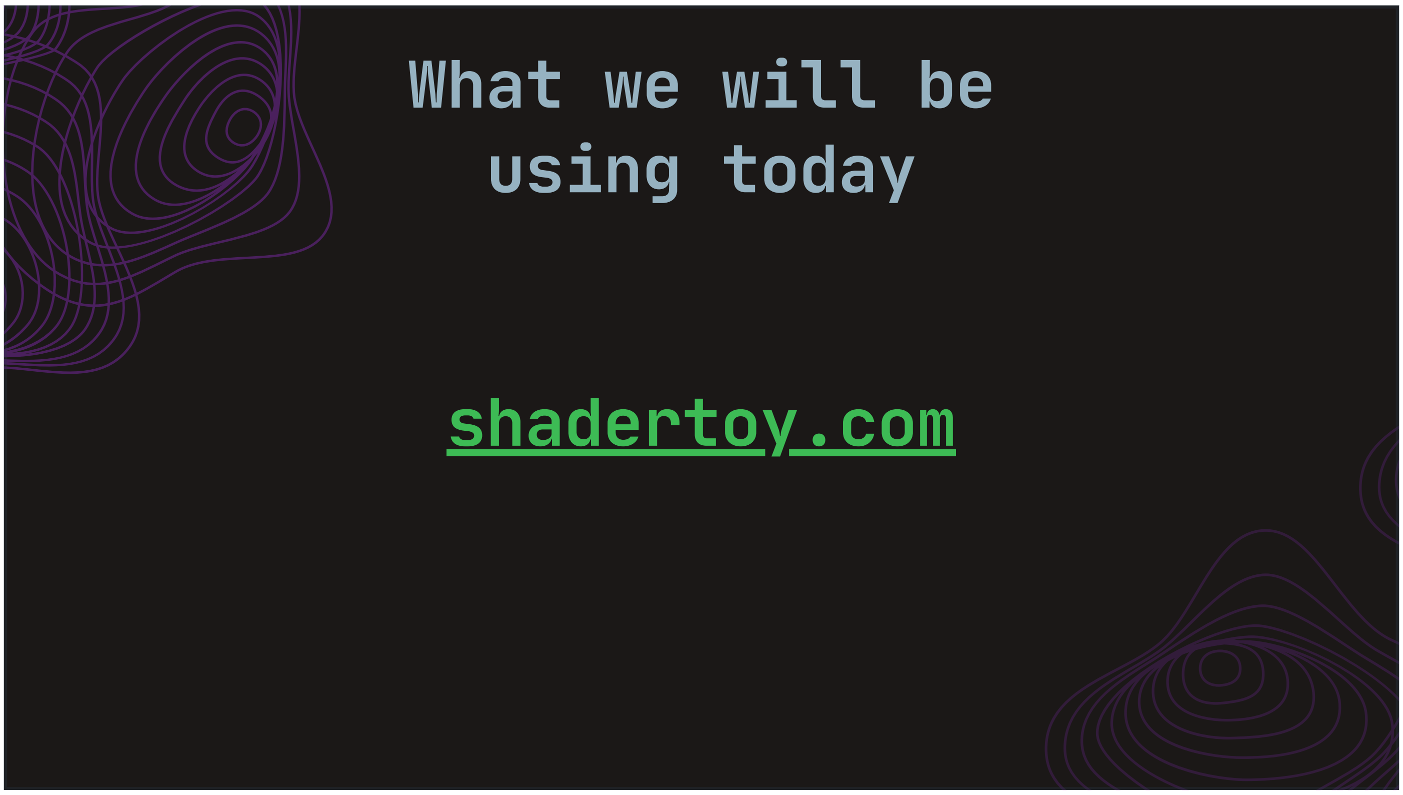
GRAPHICS PROGRAMMING KNIGHTS
X
KNIGHT HACKS

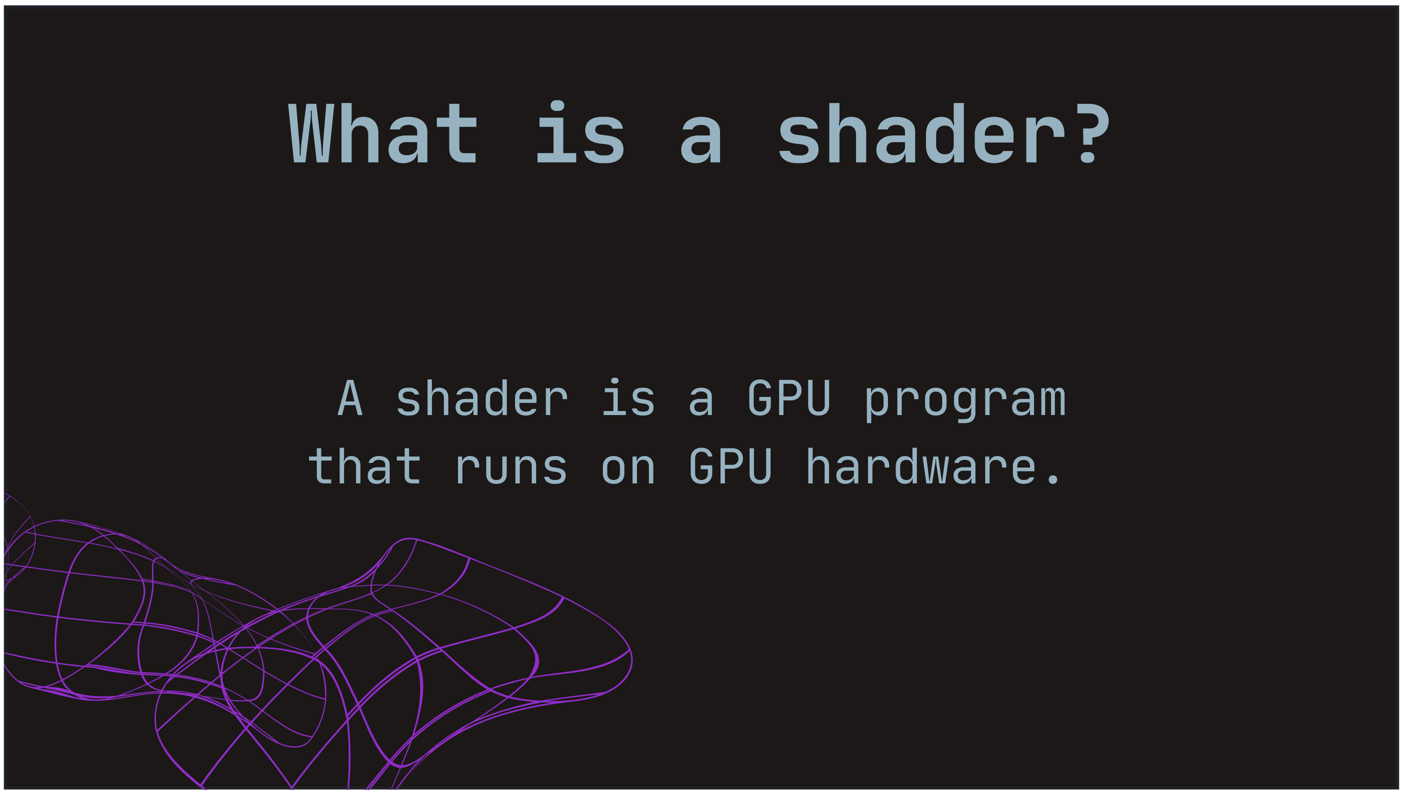
Intro to Shader Programming

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



What we will be using today

shadertoy.com



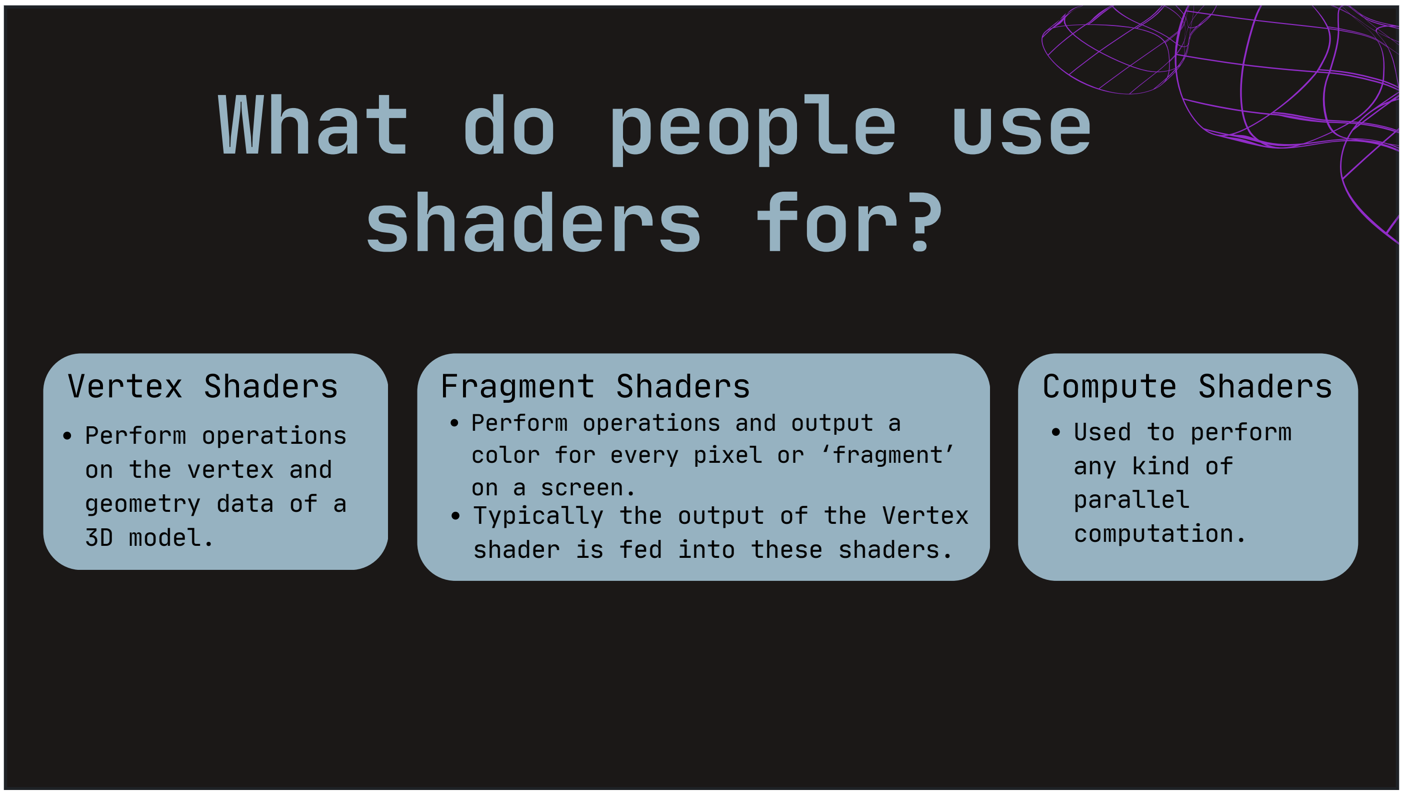
What is a shader?

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

Basically, a shader is a GPU program that runs on GPU hardware.

Why is that different than the regular programs that we write in our favorite languages?

Unlike traditional code written for the CPU, shaders have special constraints that make programming them much different as we will see soon



What do people use shaders for?

Vertex Shaders

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

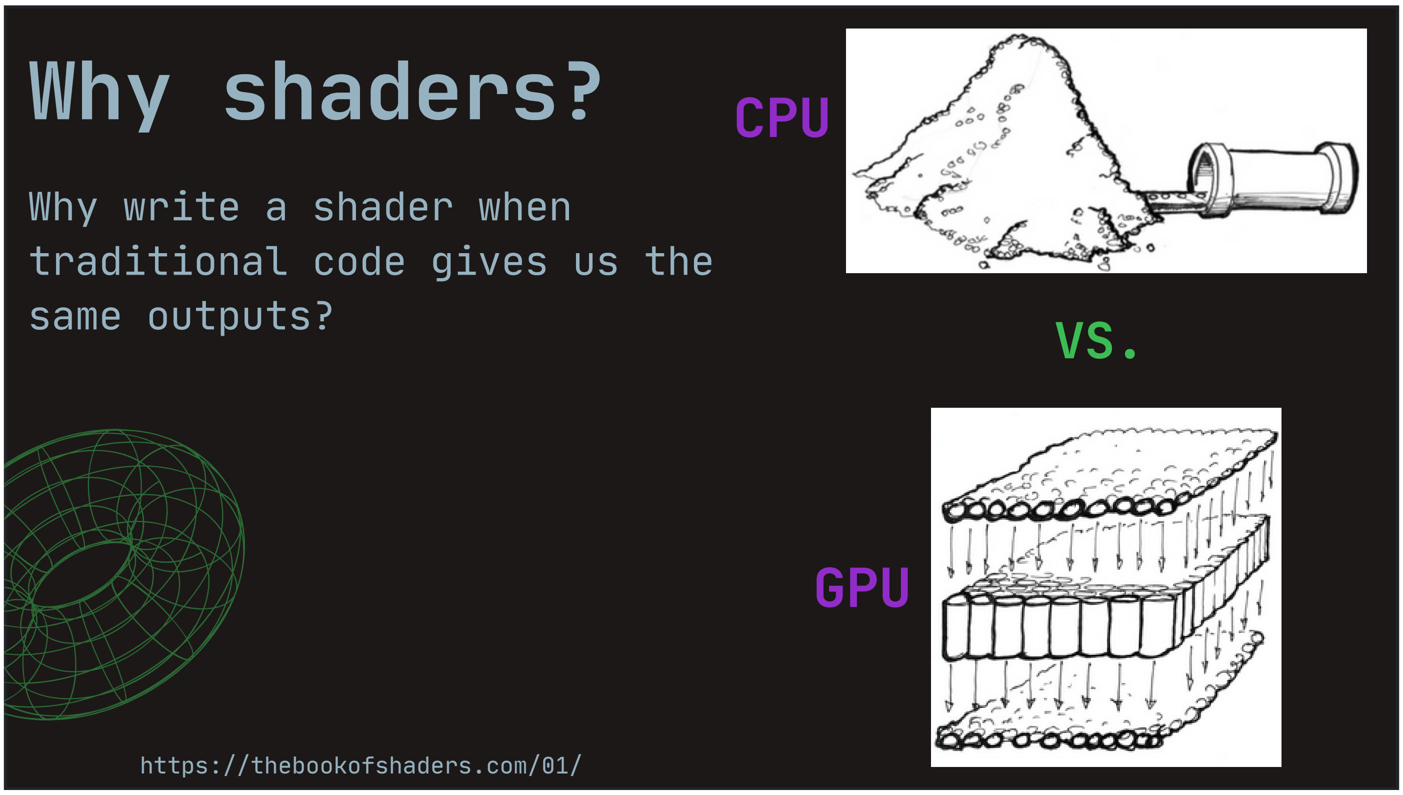
Fragment Shaders

- 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.

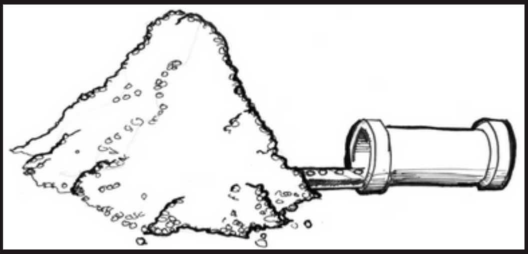
When most people think of shaders, they are usually referring to either Vertex shaders or Fragment/Pixel shaders. But, you can write a shader to solve many kinds of problems as we will discuss soon.

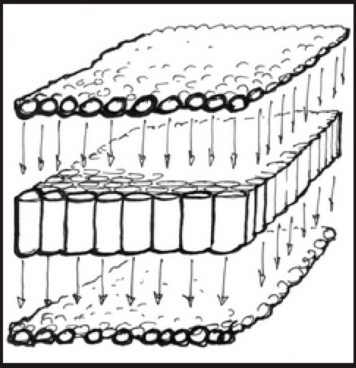


Why shaders?

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

https://thebookofshaders.com/01/





I will explain the benefit of shaders (or GPU code), and why it makes sense to use them for problems like the ones on the previous slide.

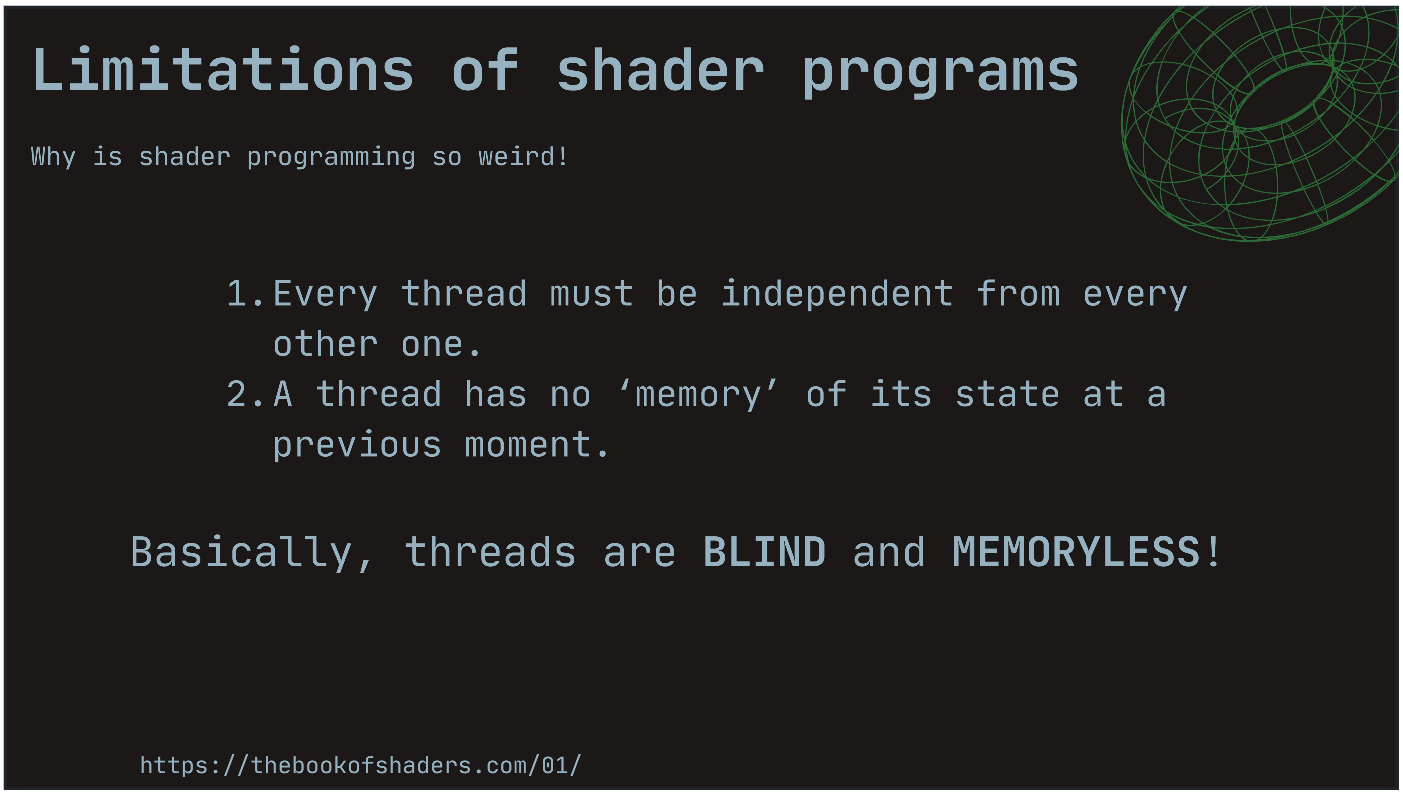
This analogy comes from a great resource called "The Book of Shaders", and I recommend you guys check it out later.

Imagine that your CPU is a big industrial pipe and the tasks that the CPU handles are the things that pass through the pipe. In this analogy, a single pipe is what we call a 'thread'.

Well, this large pipe is great for making complex products one at a time, but what if we want to make a bunch simple products all at once?

This is where the GPU comes in! Its basically a bunch of threads that can compute tiny tasks in parallel. We won't go into much depth about how GPUs work under the hood, but just know that this is the strength of GPUs.

Now, looking from a rendering point of view, your 1920x1080 screen has 2,073,600 pixels that we need to calculate! We don't want to have to wait for each pixel to be calculated one after the other while using the CPU, so we use the GPU instead to operate on all pixels at the same time.



Limitations of shader programs

Why is shader programming so weird!

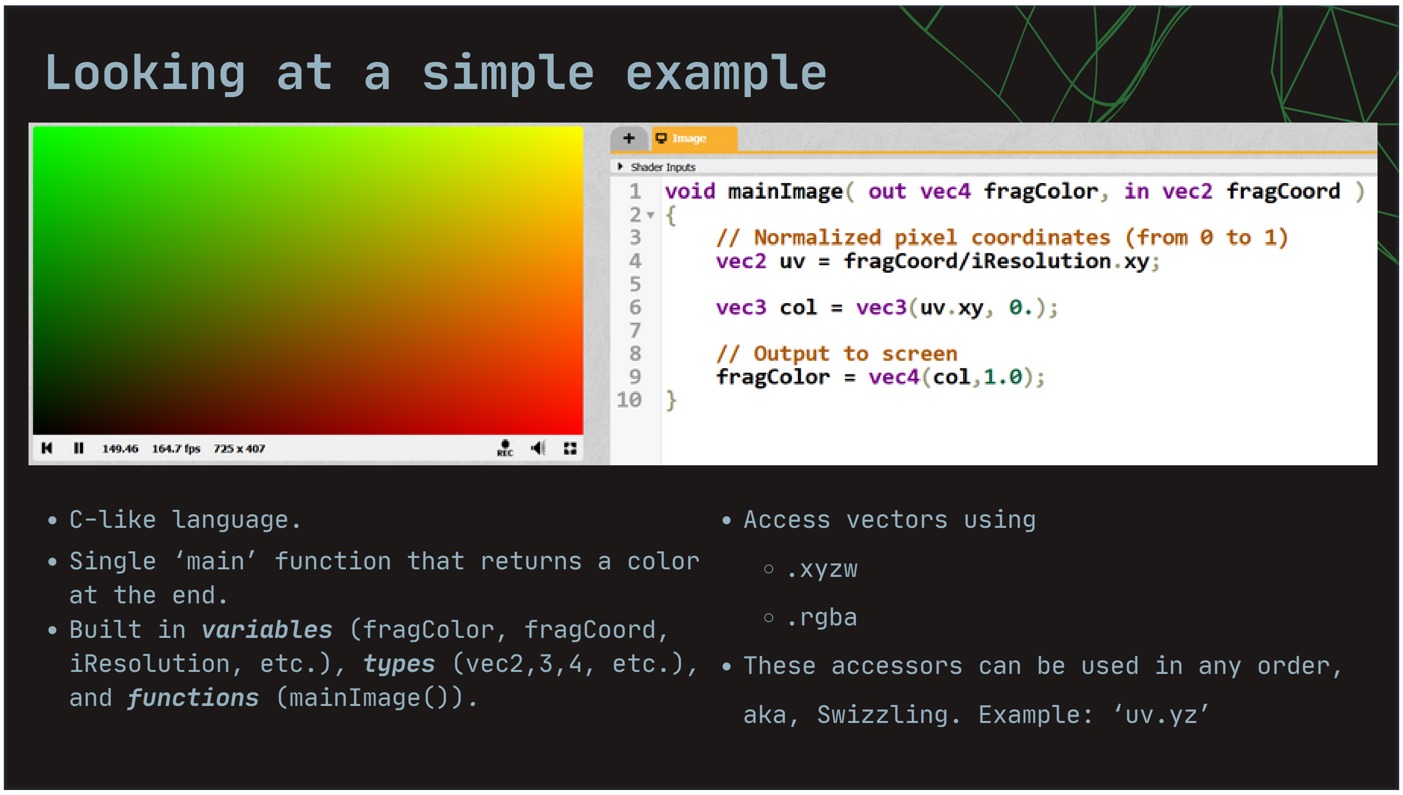
1. Every thread must be independent from every other one.
2. A thread has no 'memory' of its state at a previous moment.

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

https://thebookofshaders.com/01/

In order to run in parallel every thread (or pipe in our analogy) has to be independent from every other thread. So it's impossible to check the result of another thread, modify the input data, or pass the outcome of a thread into another thread.

Also, the GPU keeps its threads constantly busy. As soon as a thread is done, it immediately gets new data to process. This makes knowing what a thread was doing in a previous moment impossible.



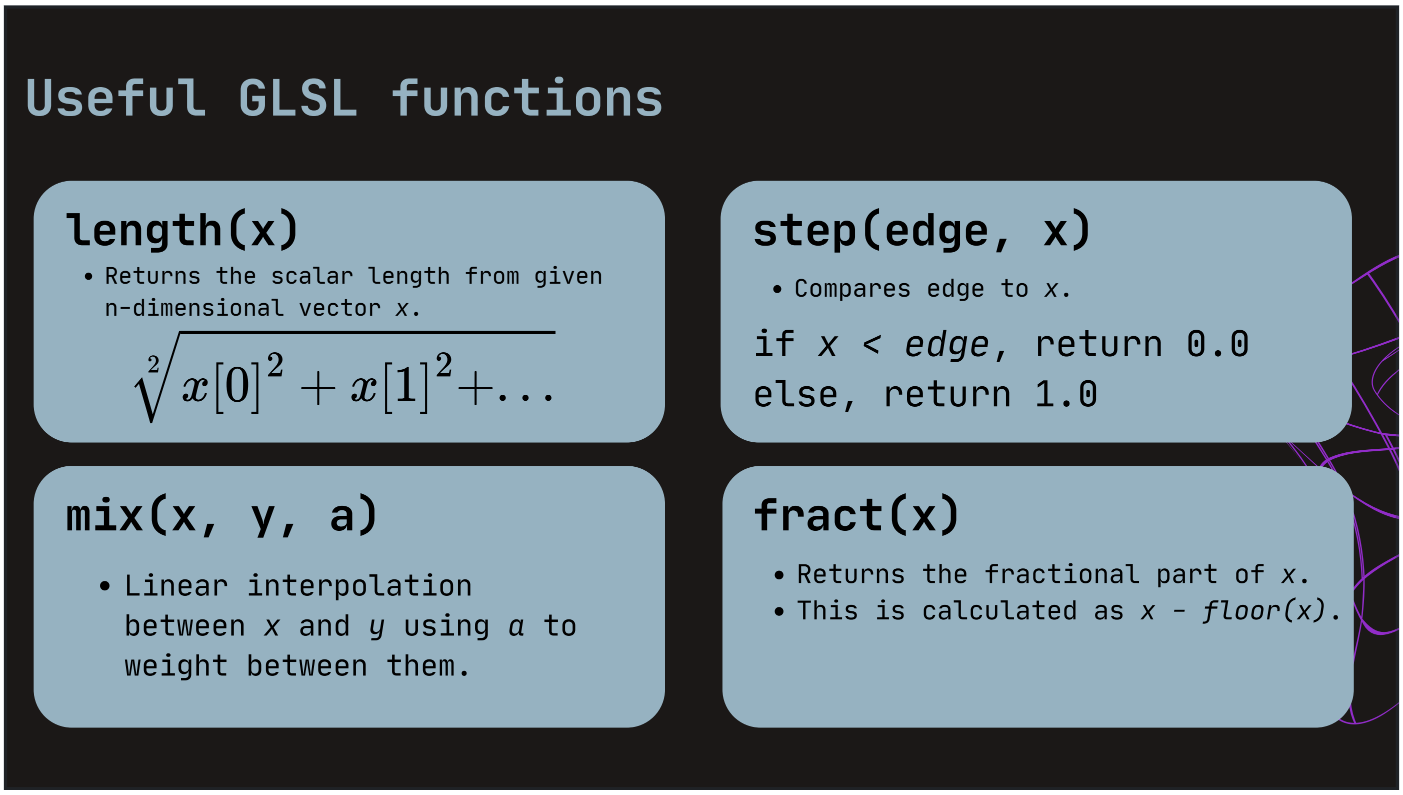
Looking at a simple example

```glsl
void mainImage( out vec4 fragColor, in vec2 fragCoord )
{
    // Normalized pixel coordinates (from 0 to 1)
    vec2 uv = fragCoord/iResolution.xy;

    vec3 col = vec3(uv.xy, 0.);

    // Output to screen
    fragColor = vec4(col,1.0);
}
```

- 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'

Talk about IO (in and out qualifiers assign variables for input and output for our shader program).

Predefined global variables (in ShaderToy, iResolution and things under 'Shader Inputs').



Useful GLSL functions

length(x)

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

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

step(edge, x)

- Compares edge to *x*.

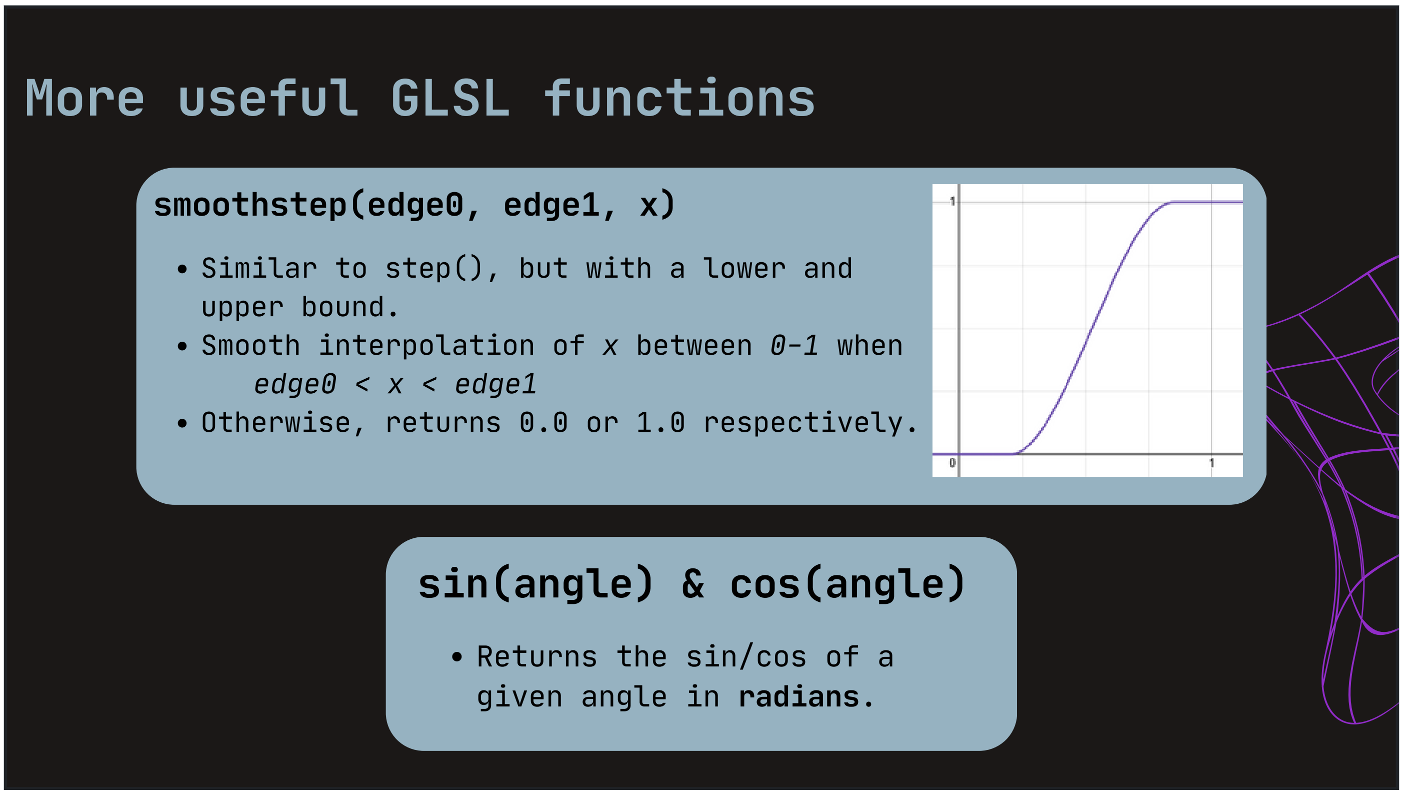
if *x* < *edge*, return 0.0
else, return 1.0

mix(x, y, a)

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

fract(x)

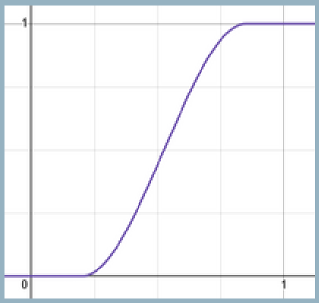
- Returns the fractional part of *x*.
- This is calculated as *x* - *floor(x)*.



More useful GLSL functions

smoothstep(edge0, edge1, x)

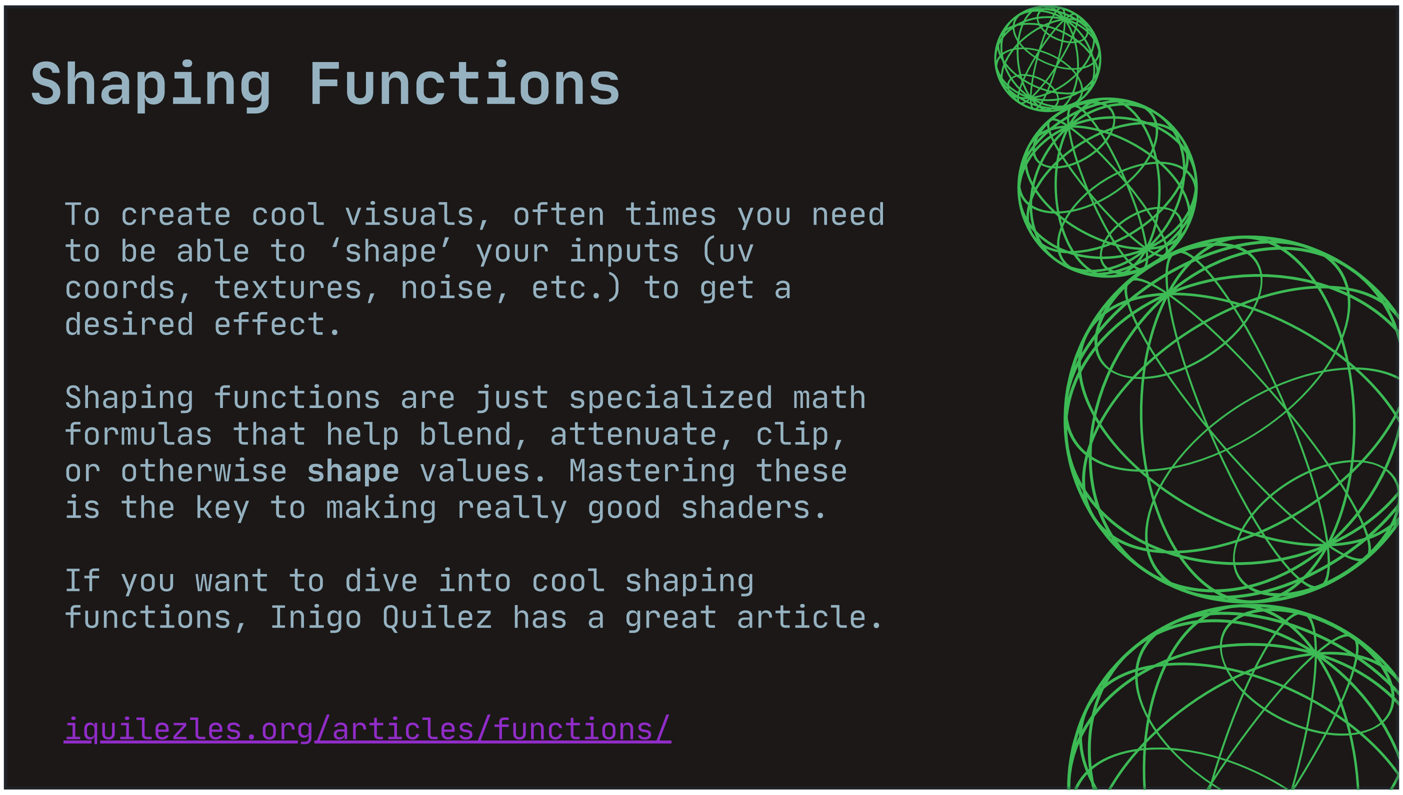
- Similar to step(), but with a lower and upper bound.
- Smooth interpolation of *x* between *0-1* when
  edge0 < x < edge1
- Otherwise, returns 0.0 or 1.0 respectively.



sin(angle) & cos(angle)

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

smoothstep on Desmos:
https://www.desmos.com/calculator/t5nfk17ihx



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.

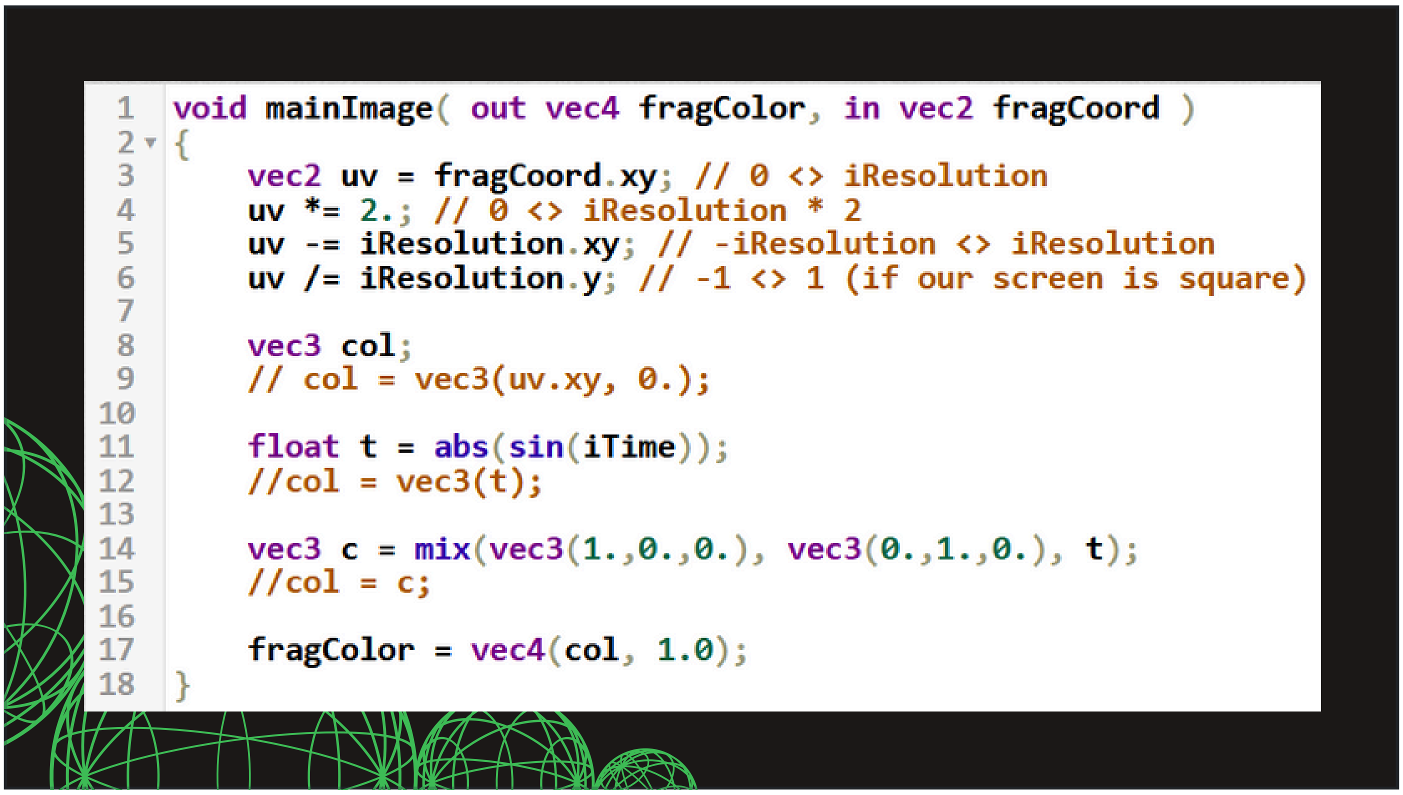
Shaping functions are just specialized math formulas that help blend, attenuate, clip, or otherwise **shape** values. Mastering these is the key to making really good shaders.

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

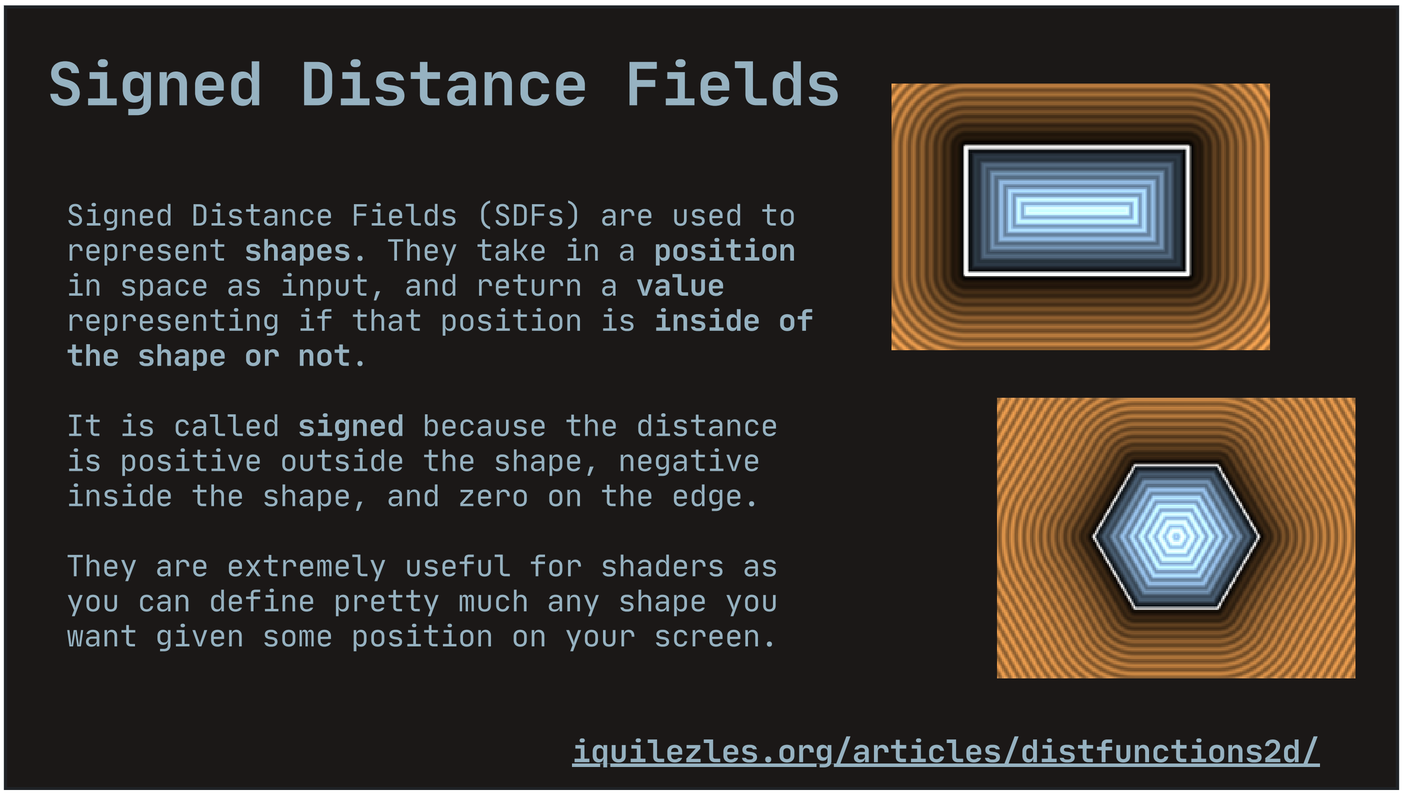
iquilezles.org/articles/functions/

Let's try a simple shader together!

https://www.shadertoy.com/view/Wc2yzy



```glsl
void mainImage( out vec4 fragColor, in vec2 fragCoord )
{
    vec2 uv = fragCoord.xy; // 0 <> iResolution
    uv *= 2.; // 0 <> iResolution * 2
    uv -= iResolution.xy; // -iResolution <> iResolution
    uv /= iResolution.y; // -1 <> 1 (if our screen is square)

    vec3 col;
    // col = vec3(uv.xy, 0.);

    float t = abs(sin(iTime));
    //col = vec3(t);

    vec3 c = mix(vec3(1.,0.,0.), vec3(0.,1.,0.), t);
    //col = c;

    fragColor = vec4(col, 1.0);
}
```

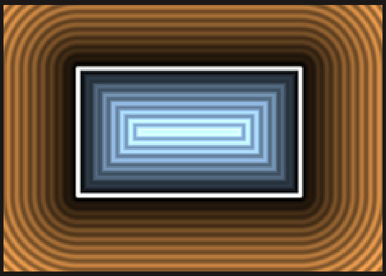


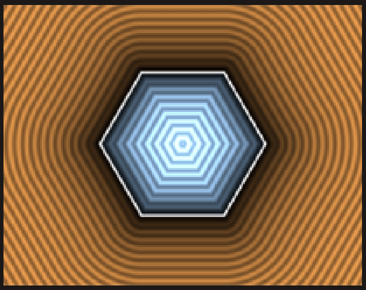
Signed Distance Fields

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.**

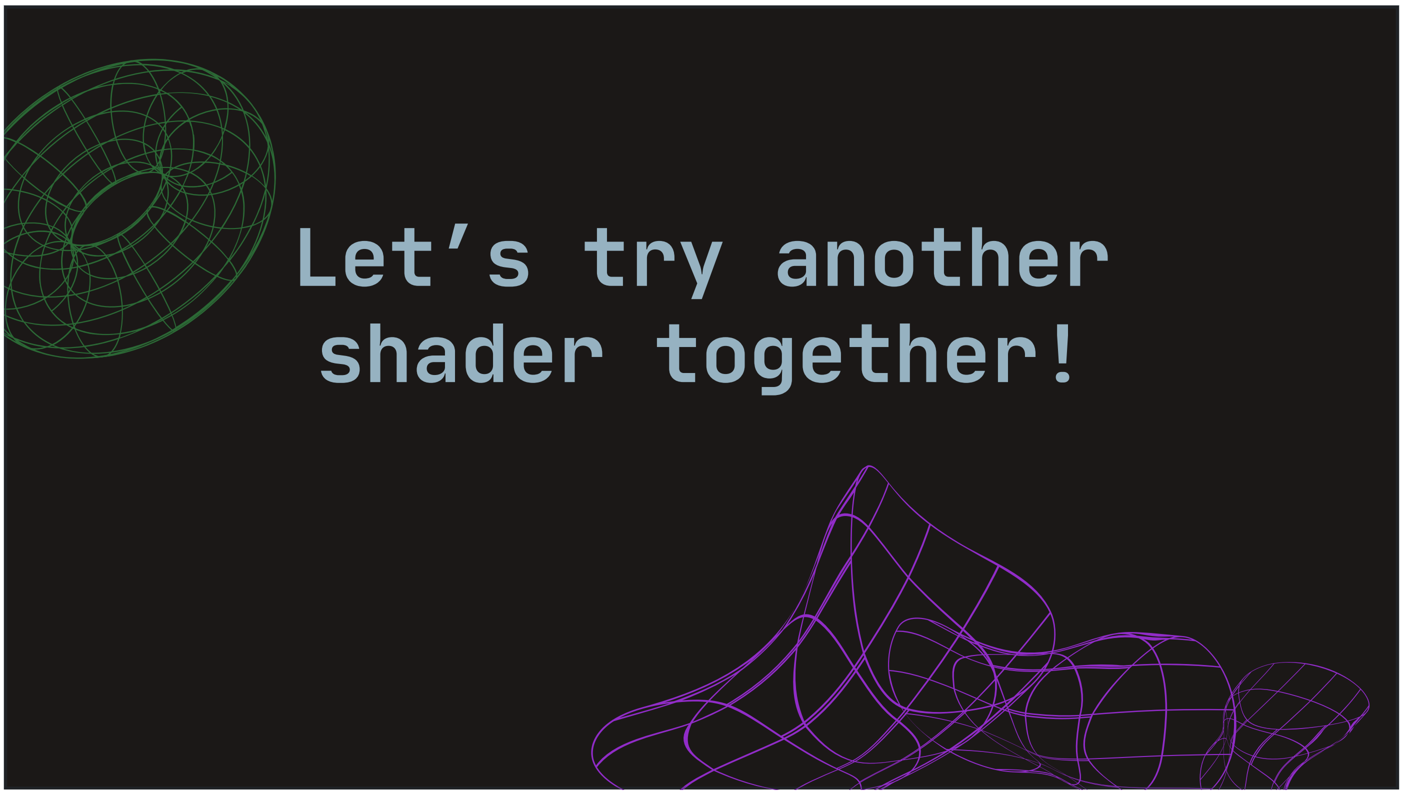
It is called **signed** because the distance is positive outside the shape, negative inside the shape, and zero on the edge.

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



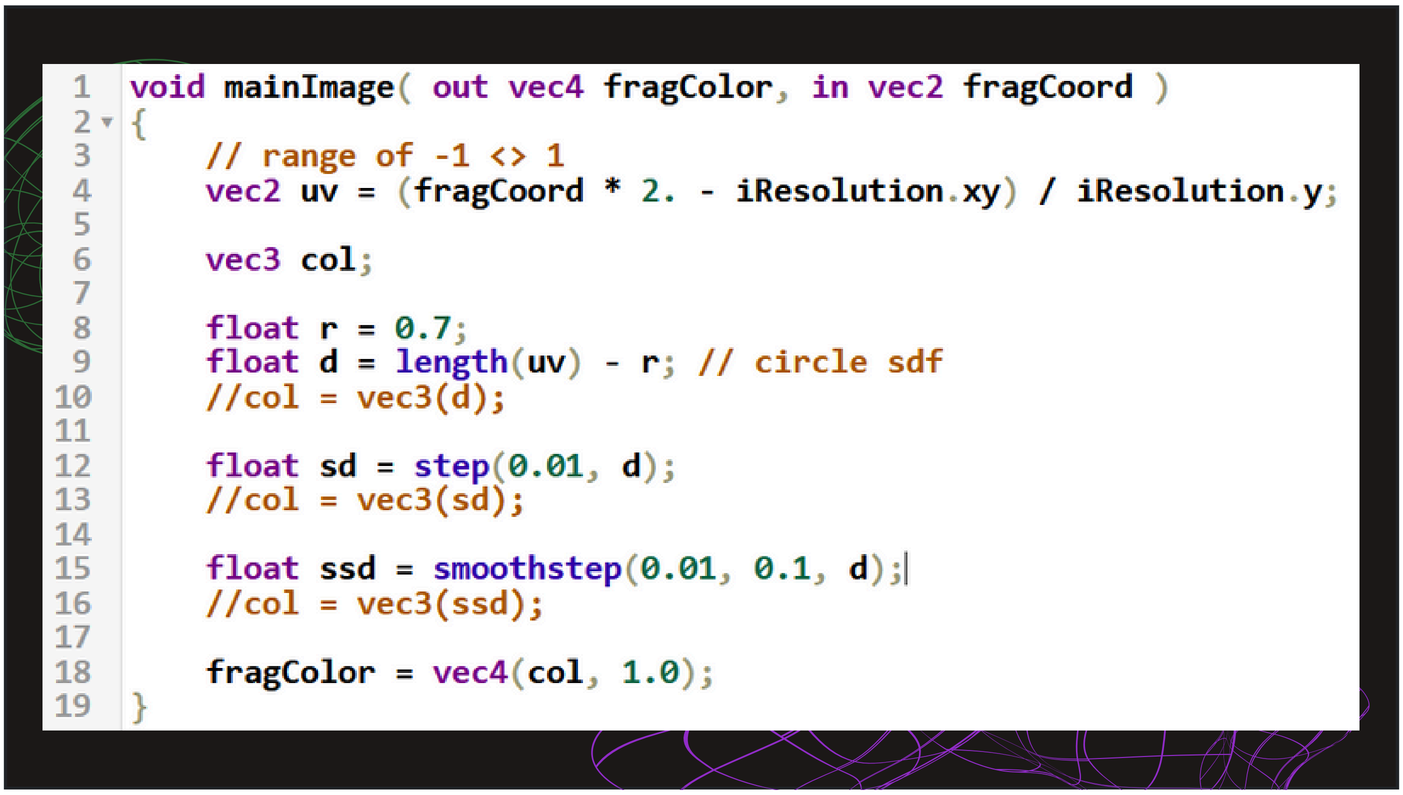


iquilezles.org/articles/distfunctions2d/

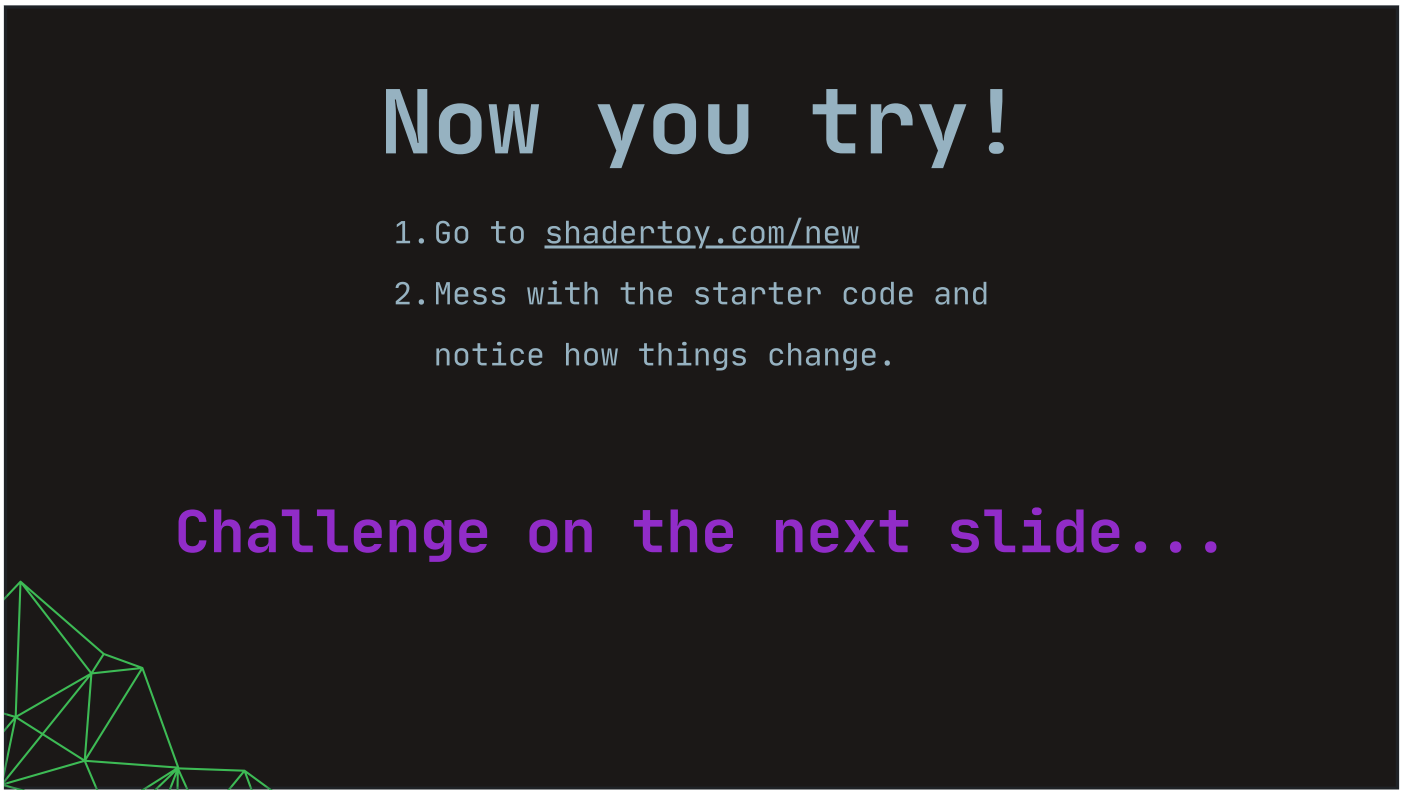


Let's try another shader together!

https://www.shadertoy.com/view/3fSyz3



```
void mainImage( out vec4 fragColor, in vec2 fragCoord )
{
    // range of -1 <> 1
    vec2 uv = (fragCoord * 2. - iResolution.xy) / iResolution.y;

    vec3 col;

    float r = 0.7;
    float d = length(uv) - r; // circle sdf
    //col = vec3(d);

    float sd = step(0.01, d);
    //col = vec3(sd);

    float ssd = smoothstep(0.01, 0.1, d);
    //col = vec3(ssd);

    fragColor = vec4(col, 1.0);
}
```

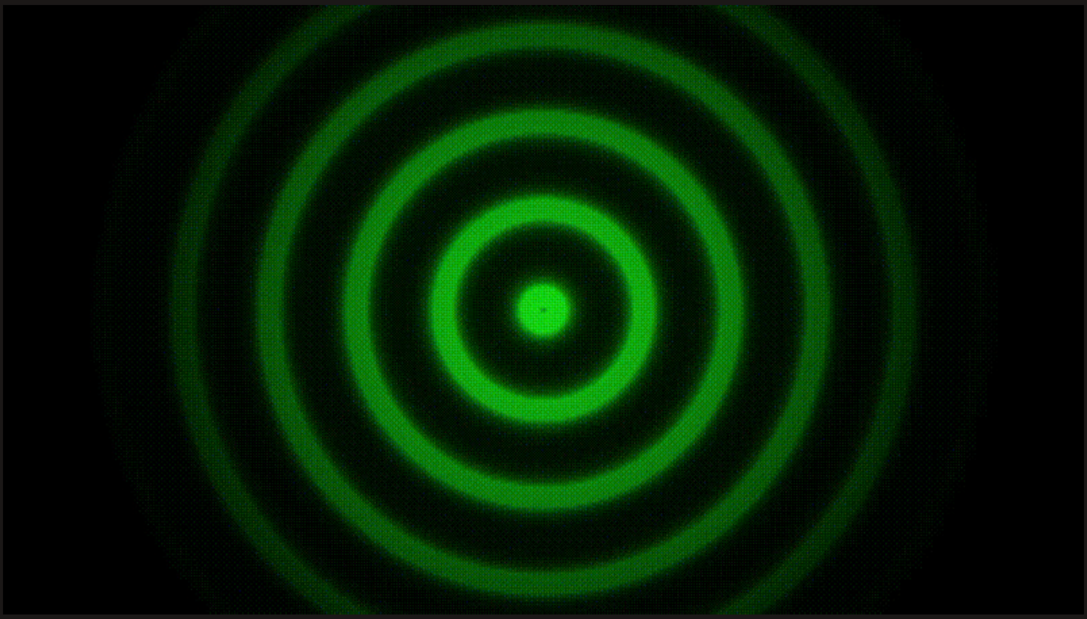


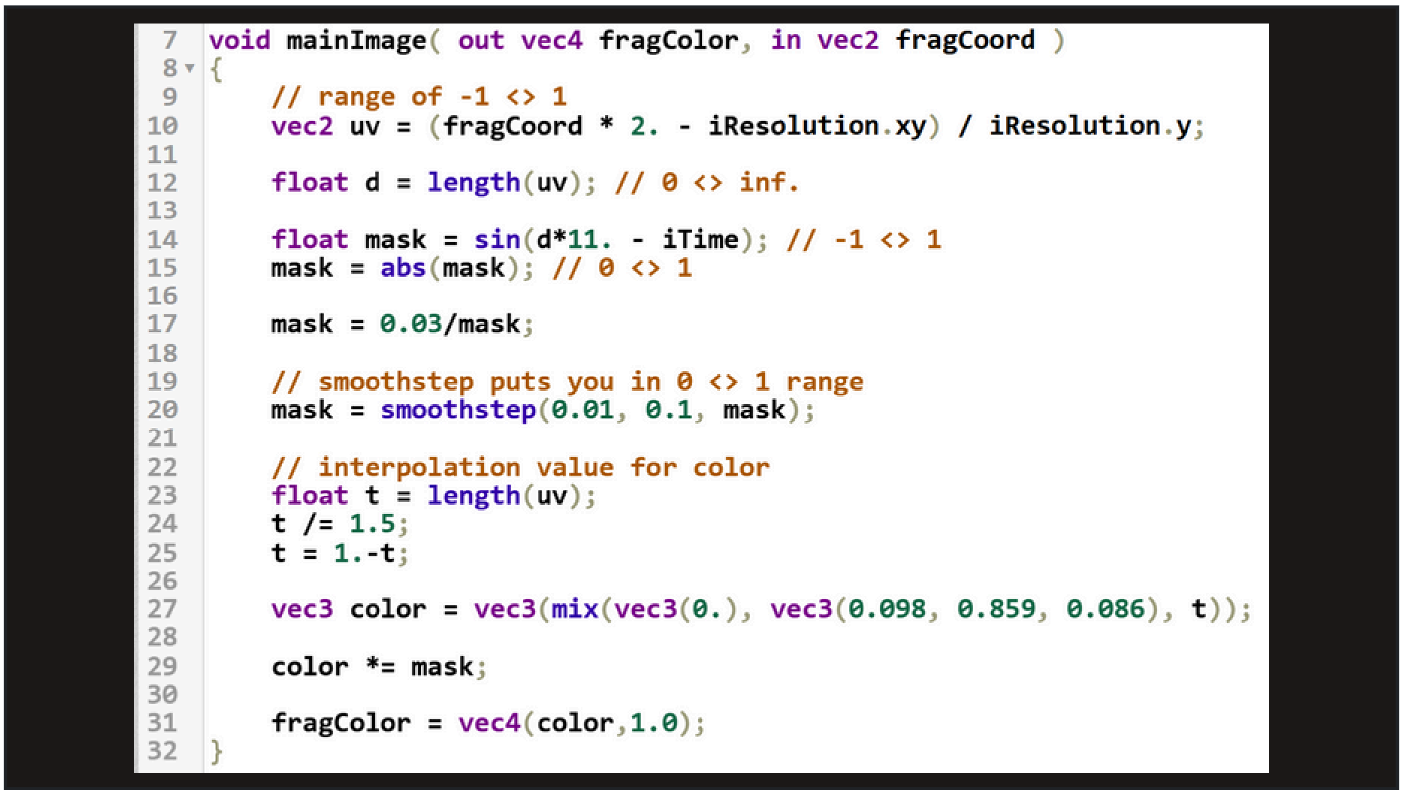
Now you try!

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

Challenge on the next slide...

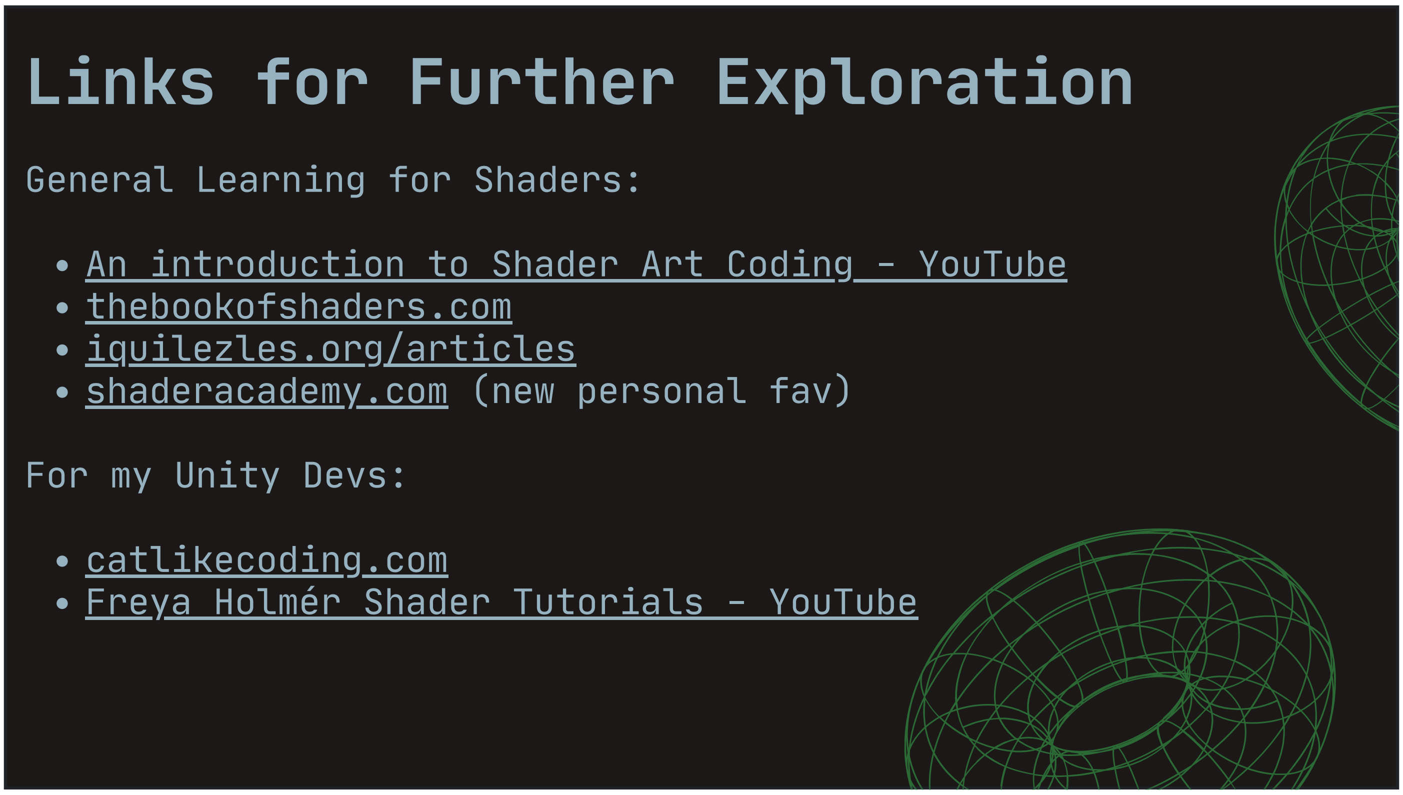
Challenge





```
void mainImage( out vec4 fragColor, in vec2 fragCoord )
{
    // range of -1 <> 1
    vec2 uv = (fragCoord * 2. - iResolution.xy) / iResolution.y;

    float d = length(uv); // 0 <> inf.

    float mask = sin(d*11. - iTime); // -1 <> 1
    mask = abs(mask); // 0 <> 1

    mask = 0.03/mask;

    // smoothstep puts you in 0 <> 1 range
    mask = smoothstep(0.01, 0.1, mask);

    // interpolation value for color
    float t = length(uv);
    t /= 1.5;
    t = 1.-t;

    vec3 color = vec3(mix(vec3(0.), vec3(0.098, 0.859, 0.086), t));

    color *= mask;

    fragColor = vec4(color,1.0);
}
```

Graphs for walkthrough: https://www.desmos.com/calculator/wzc25y5ozt



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

Follow our Socials!

- Stay up to date on our events by following our Instagram!
- Rewatch GPK meetings at your own pace when published on YouTube!

Thank you!