Computer Graphics:

Introduction to Shadertoy

By: Nick | ACM at UCSD

What is Computer Graphics?

Sorry to those who have taken 167...

Here's the generic Wikipedia definition:

> *"Computer graphics is the discipline of generating images with the aid of computers"*

Computer graphics is used in a variety of applications, mainly:

- Film (either for animations or for visual effects)
- Video Games

UNREAL
ENGINE

unity

Animation

Visual Effects

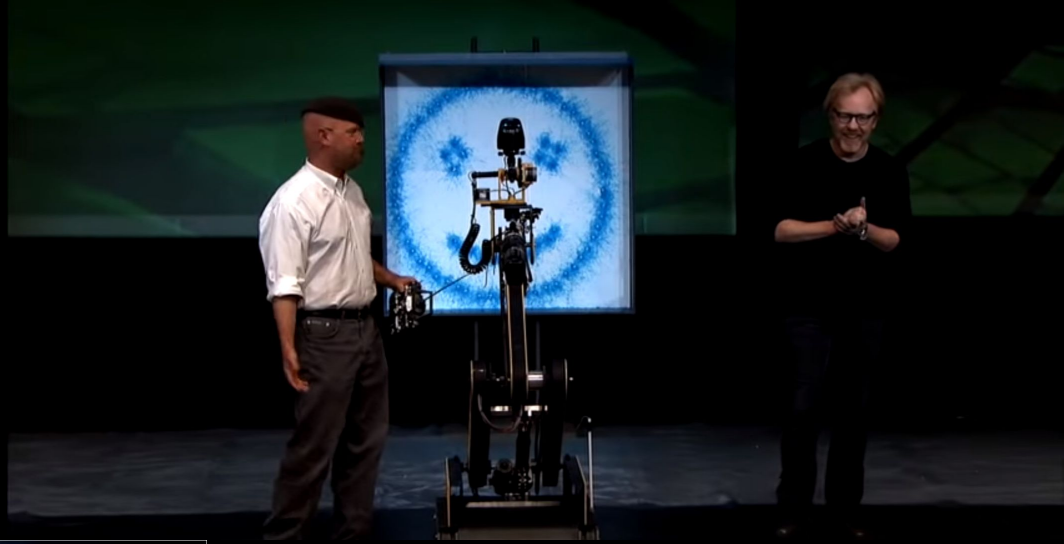
Shaders

If you have experience drawing things with computers, you may know that things are drawn *sequentially*

For example, first draw a circle here, a line here, etc until you finish drawing your object

Shaders are also a set of instructions, except the instructions are executed *all at once* (but not necessarily for every pixel).

One dot at a time



All at once

ShaderToy

ShaderToy sets up the shader so that it runs once for each pixel

Acts like a function that receives the pixel position and a few other inputs and returns the pixel color as an output

For those who have taken 167, it basically rasterizes a full-screen quad and the shader you write is the fragment shader. It also uses WebGL (as opposed to OpenGL, though it is a similar standard) to allow it to run in a browser

GLSL

OpenGL shaders are written in a language called GLSL

*open**GL S**hading L**anguage*

Has many built-in features as opposed to regular programming languages, mainly support for basic Linear Algebra

Oh yes we will be doing some math

PLEASE DON’T FREAK OUT JUST YET PLEASE IT’S JUST BASIC LINEAR ALGEBRA

Image

Shader Inputs

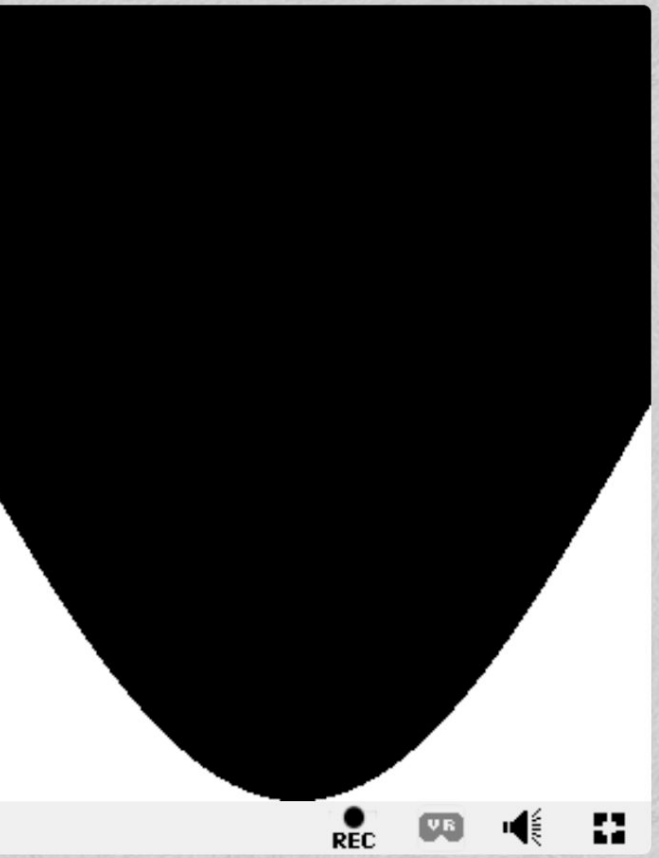
```glsl
const float PI = 3.14159265;

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

    float y = 0.5 + 0.5 * sin(uv.x * 2.0 * PI);

    // calculate color here

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

My First ShaderToy

Now calculate the color!

Simple sin graph (answer):

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



REC VR

0

Created by **nickwn** in 2020-04-03

Image

Shader Inputs

```glsl
const float PI = 3.14159265;

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

    float y = 0.5 + 0.5 * sin(uv.x * 2.0 * PI);

    vec3 col = vec3(0.0);
    if(uv.y < y)
    {
        col = vec3(1.0);
    }

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

Nice, but one problem

Some aliasing issues

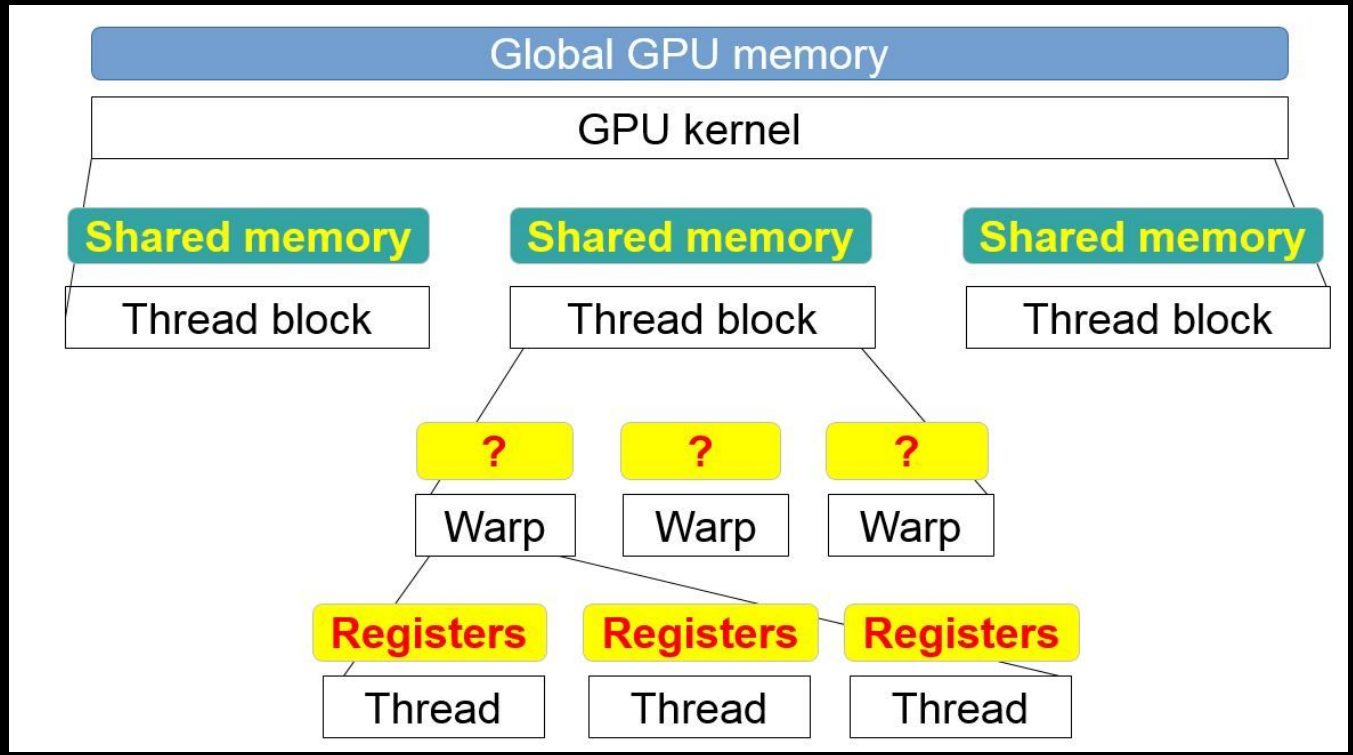
There’s a branch (an if statement), which isn’t that bad but it gives us an opportunity to learn how the GPU works

Why is this a problem?

To better understand, we need to look at the basic architecture of a GPU (terminology may differ):

- GPUs are filled with hundreds and hundreds of processing “threads” which run your code in parallel
- These processing threads are separated into small(ish) groups, called *warps*, which are given work by schedulers

A Fun Diagram



Branching in Warps

The problem is that a warps are issued a single instruction at a time

This means that all threads in a warp have to run the same instruction together

This is inherently against how branches work; ideally threads should be executing different instructions at the same time based on a condition

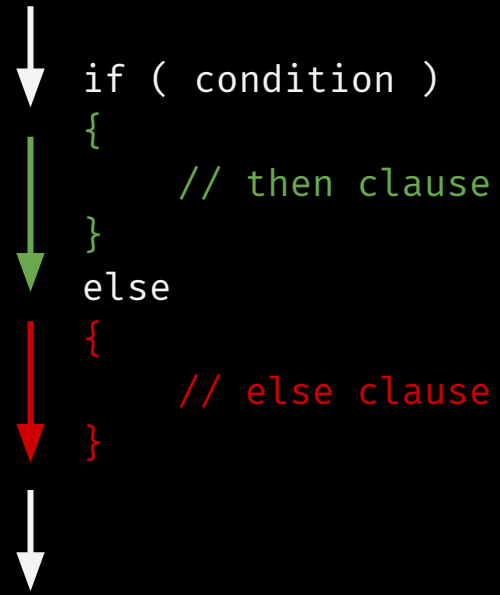
How does the GPU solve this?

More Fun Diagrams

Here's an example of your typical CPU control flow:

If condition is true, the green path is executed.

If condition is false, the red path is executed

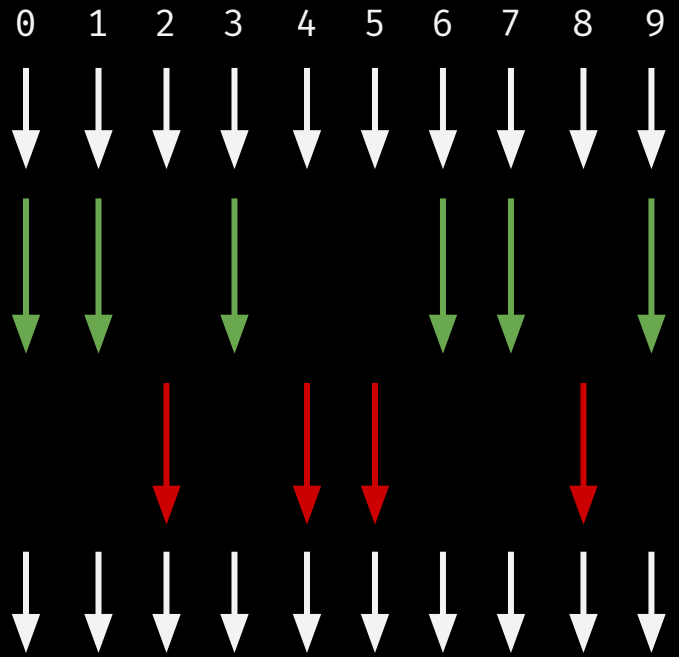


More Fun Diagrams

Here's how it looks on the GPU (where each arrow represents a thread):

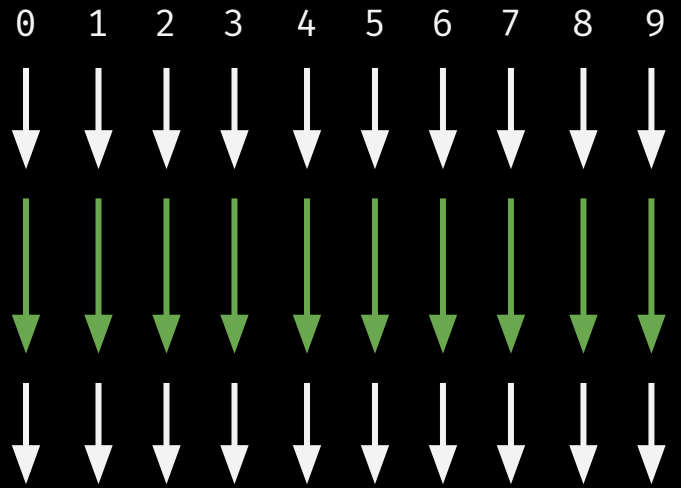
The GPU basically creates a mask for all the threads based on the condition

The ALUs execute every instruction, but their output is masked out if they are unneeded



More Fun Diagrams

Execution within warps also removes one branch if it can, so if only one side of the branch needs to be executed for all threads, then the other side will not be executed



Divergence

The idea that not all the work threads do is productive is important to optimization

If it is very likely for a thread to not be doing productive work, your program has *high divergence*

Applies to branching, but also applies to things like loops. An entire warp could be blocked if it needs to wait for a single thread to complete its loop.

How do we remove the if?

We can remove the `if` using the `smoothstep` function in glsl

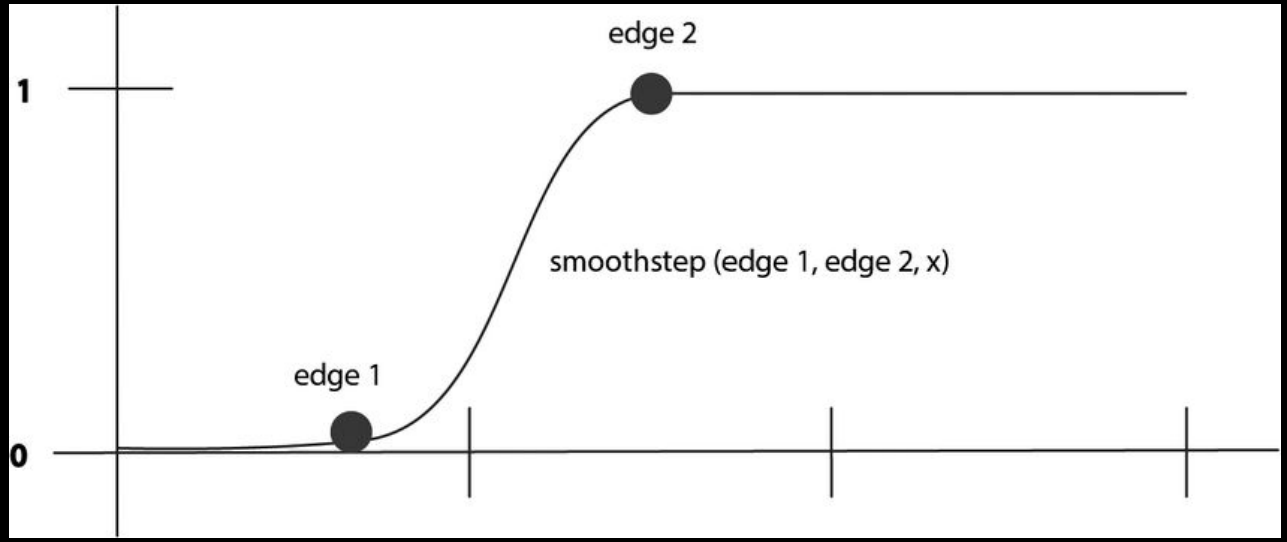
`smoothstep` is basically a fancy interpolation function that can be used to smoothly blend between two values, in our case black and white:

Key thing to note is that it doesn't actually do the interpolation; it just returns the t-value for the interpolation equation (to interpolate between a and b):

$$interp = a * (t-1) + b * t$$

smoothstep

https://thebookofshaders.com/glossary/?search=smoothstep

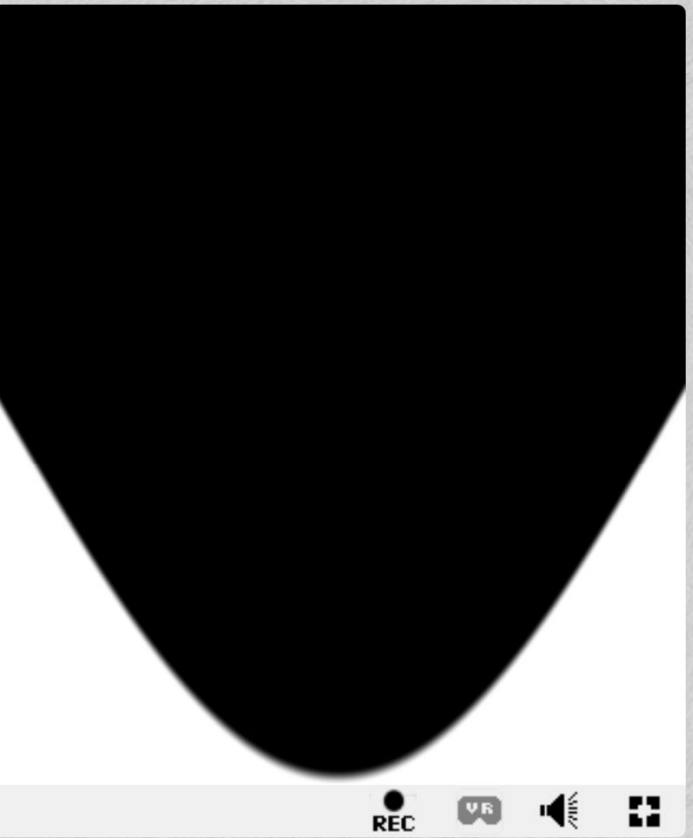


My First ShaderToy (take 2)

Modify your previous code to remove the if statement!

Better sin graph (answer):

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



REC

0

Created by **nickwn** in 2020-04-03

Image

Shader Inputs

```glsl
const float PI = 3.14159265;

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

    float y = 0.5 + 0.5 * sin(uv.x * 2.0 * PI);

    float percent = smoothstep(y + 0.02, y, uv.y);

    vec3 col = vec3(percent);

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

Raymarching

Bonus section

aboutta go 0-100 real quick

Signed Distance Functions

A function that, when passed the coordinates of a point in space, returns the shortest distance between that point and some surface

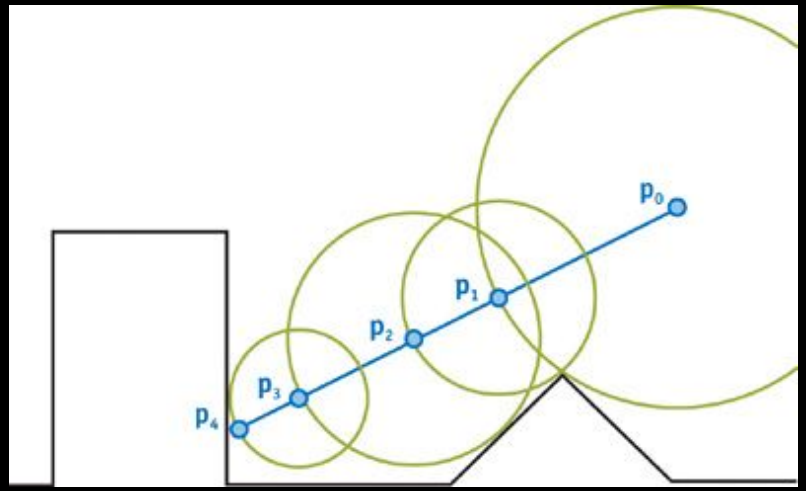
The sign of the return value indicates whether the point is inside that surface or outside

For example, the SDF for a sphere looks like this:

$$f(x, y, z) = sqrt(x^2 + y^2 + z^2) - r$$

Raymarching

We can “step” along a camera view ray at different distances, evaluating the SDF at each step to see if we are close to a surface and to determine the distance to the next step.



Resources

http://iquilezles.org/www/index.htm

- SDFs: http://iquilezles.org/www/articles/distfunctions/distfunctions.htm
- http://iquilezles.org/www/articles/distfunctions2d/distfunctions2d.htm
- Examples: http://iquilezles.org/www/articles/raymarchingdf/raymarchingdf.htm

Final Project

Color in the surfboard/add a beach ball:

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

Final Project

For the Future

Raymarching as it is done in shadertoy is kinda niche

As a general concept however, it is used extensively in volume rendering and a little for global illumination

Download Blender for the workshop next week, where we will be creating a new material using Blender's material graph and OSL that can approximate sub-surface scattering effects

Sub-org???: acmurl.com/graphics-interest

Slide References

UPenn SIGGRAPH's Shadertoy presentation: https://drive.google.com/file/d/1XX4OMwRxrMcgU5NTBPz2qjRKOhRAcPhp/view?usp=sharing

Mythbusters: https://www.youtube.com/watch?v=ZrJeYFxpUyQ

The Book of Shaders: https://thebookofshaders.com/

Old Nvidia presentation: https://www.nvidia.com/docs/IO/116711/sc11-perf-optimization.pdf

New Nvidia Post: https://devblogs.nvidia.com/register-cache-warp-cuda/