GLSL

Kyle Genova

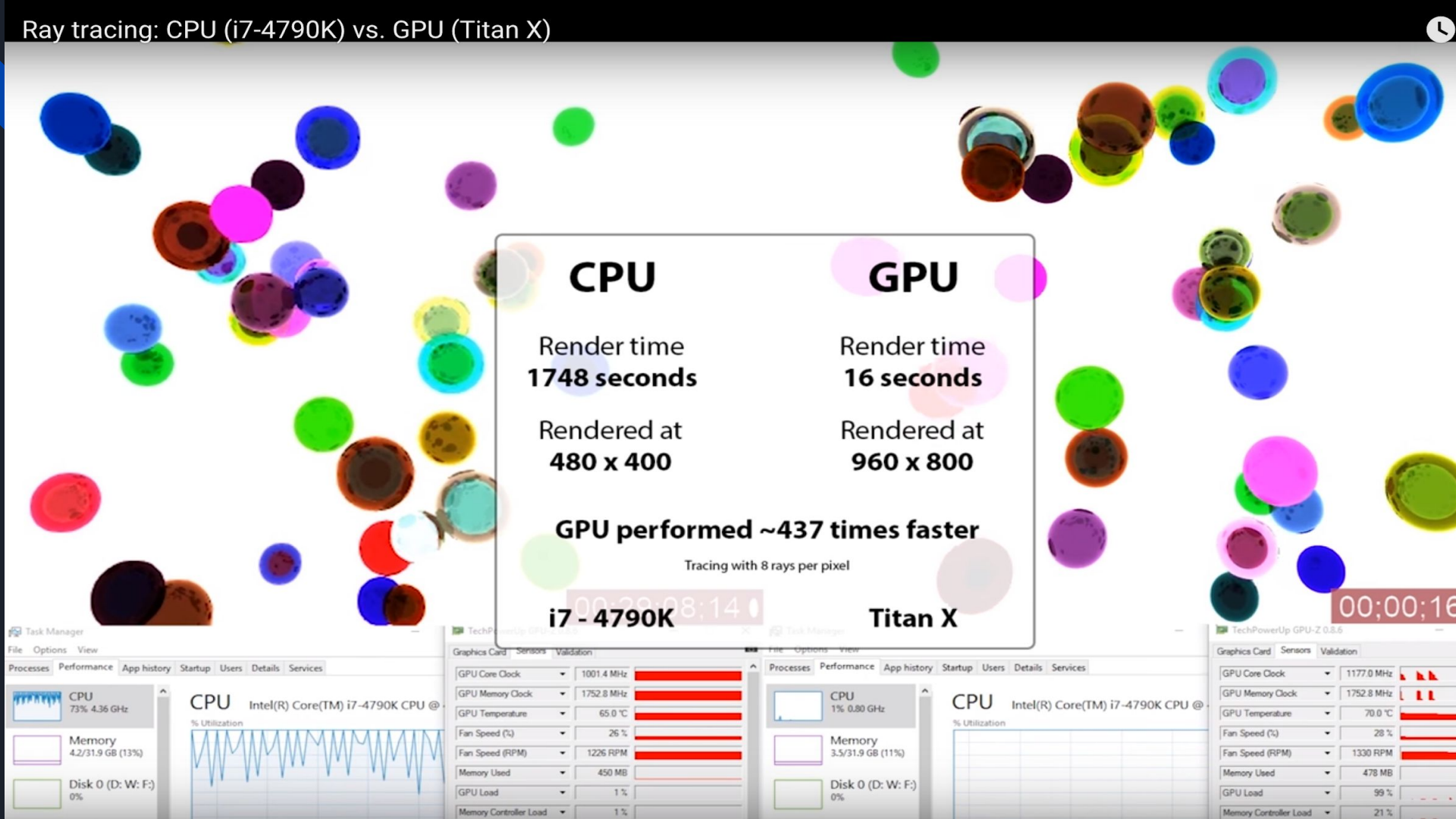
What is GLSL?

It’s the Open **G**raphics **L**ibrary **S**hader **L**anguage

What is GLSL?

A C-like language (syntactically) with more type safety and no recursion that executes code directly on the GPU.

Why do we want it for Ray Tracing?


Ray tracing: CPU (i7-4790K) vs. GPU (Titan X)
CPU
GPU
Render time
1748 seconds
Render time
16 seconds
Rendered at
480 x 400
Rendered at
960 x 800
GPU performed ~437 times faster
Tracing with 8 rays per pixel
i7 - 4790K
Titan X
00;00;16

Two Critical Components: Vertex & Fragment Shaders

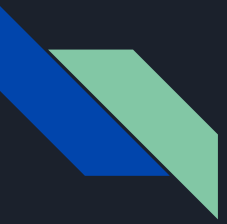
- **Vertex Shader**: Runs automatically once per vertex. Must output the final vertex position and any attributes the fragment shader needs.
- **Fragment Shader**: Runs automatically once per pixel (AKA fragment). Runs after the vertex shader. Must output the final pixel color.

What’s Missing in GLSL Syntax: C \ GLSL

- Recursion
- Implicit Casting
- Libraries
- Dynamic memory allocation
- Pointers
- Objects
- Char, String
- Console I/O ?!

GLSL Syntax Extensions: GLSL \ C

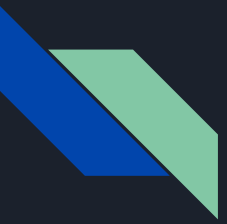
- New Types
  - Varying (AKA in/out)
  - Uniform
  - Attribute
  - vecN
  - Polymorphic builtins- max, min, sqrt, dot, cross, etc.
  - Swizzle: vec3 yxz_comp = some_vec3.yxz;
- Predefined Variables
  - gl_Position
  - gl_FragCoord
  - gl_FragColor, gl_FragData[]
  - Other gl_* variables (out of scope)



Uniform (AKA Dynamically Uniform)

Uniform variables are statically shared between all vertices and pixels.

For example: the lights and objects in the scene.



Varying: The GPU does the heavy lifting

Varying (AKA **in/out**) variables are per-vertex outputs in the vertex shader.

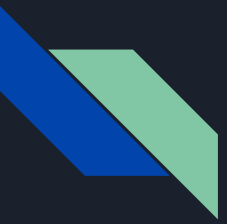
They are **automatically** barycentrically interpolated between triangle vertices by the GPU and passed as per-pixel inputs to the fragment shader.

Attribute: Vertex Shader Only

Attributes are values that are unique per-vertex and are passed into the vertex shader.

vecN and **matN**: Easier vector math

```
vec3 a = vec3(1.0, 2.0, 3.0);

vec4 b = vec4(a, 1.0);

vec3 c = b.xyz;

vec2 d = c.yy;

vec4 e; e.xyz = c; e[3] = b.w;

mat3 m; m[1].xyz = e.yxy;
```



gl_Position and other **gl_*** values: Built-ins

gl_Position The key vertex shader output. The vertex position.

gl_FragColor The key fragment shader output: the pixel color.

gl_FragCoord The pixel location in window space.

A Simple Vertex Shader

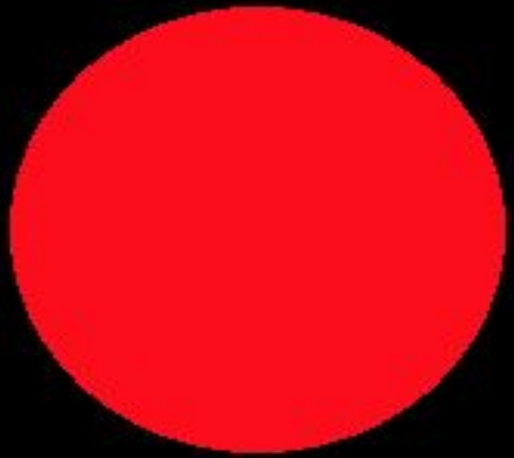
```
<script id="2d-vertex-shader" type="x-shader/x-vertex”>

attribute vec2 a_position;
  void main() {
    gl_Position = vec4(a_position, 0, 1);
  }

</script>
```

A Simple Fragment Shader

```
<script id="2d-fragment-shader" type="x-shader/x-fragment”>

void main() {
  gl_FragColor = vec4(gl_FragCoord.x / canvas_width,
                      gl_FragCoord.y / canvas_height, 0, 1);
  }

</script>
```

A (Less) Simple Fragment Shader

```
<script id="2d-fragment-shader" type="x-shader/x-fragment”>
  #ifdef GL_FRAGMENT_PRECISION_HIGH
    precision highp float;
  #else
    precision mediump float;
  #endif
  void main() {
    float cX = gl_FragCoord.x – width/2.0;
    float cY = gl_FragCoord.y – height/2.0;
    if (sqrt(cX*cX + cY*cY) < 80.0){
            gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
        } else {
            gl_FragColor = vec4(0.0, 0.0, 0.0, 1.0);
        }
  }
```



How to Avoid Recursion in a Recursive Ray Tracer

```
#define MAX_RECURSION 10
  function g() {
    float x = 0.0, weight = 1.0, res = 0.0;
    for (int i=0; i< MAX_RECURSION; i++ ) {
          cur_contrib = f();
          res = res + weight * f();
          weight = weight * 0.8;
    }
   return res;
  }
```

Visual Debugging

No console IO or breakpoints makes traditional debugging techniques ineffective. Instead, you must do “visual debugging,” which is simply creative use of the one fragment shader output you have: the pixel color.

Some simple suggestions:

- Output red for sphere, yellow for triangle, green for cylinder, etc.
- Output the normal vector of the surface directly.
- if (some_condition) then GREEN else normal shading. This can track down which pixels are problematic.
- Move around in the scene! The real-time performance of the raytracer is a huge asset.

Additional Learning Resources

https://www.opengl.org/wiki/Core_Language_%28GLSL%29

We use GLSL ES 1.00!