CS 476: Computer Graphics

GLSL And Shader Basics

GLSL??

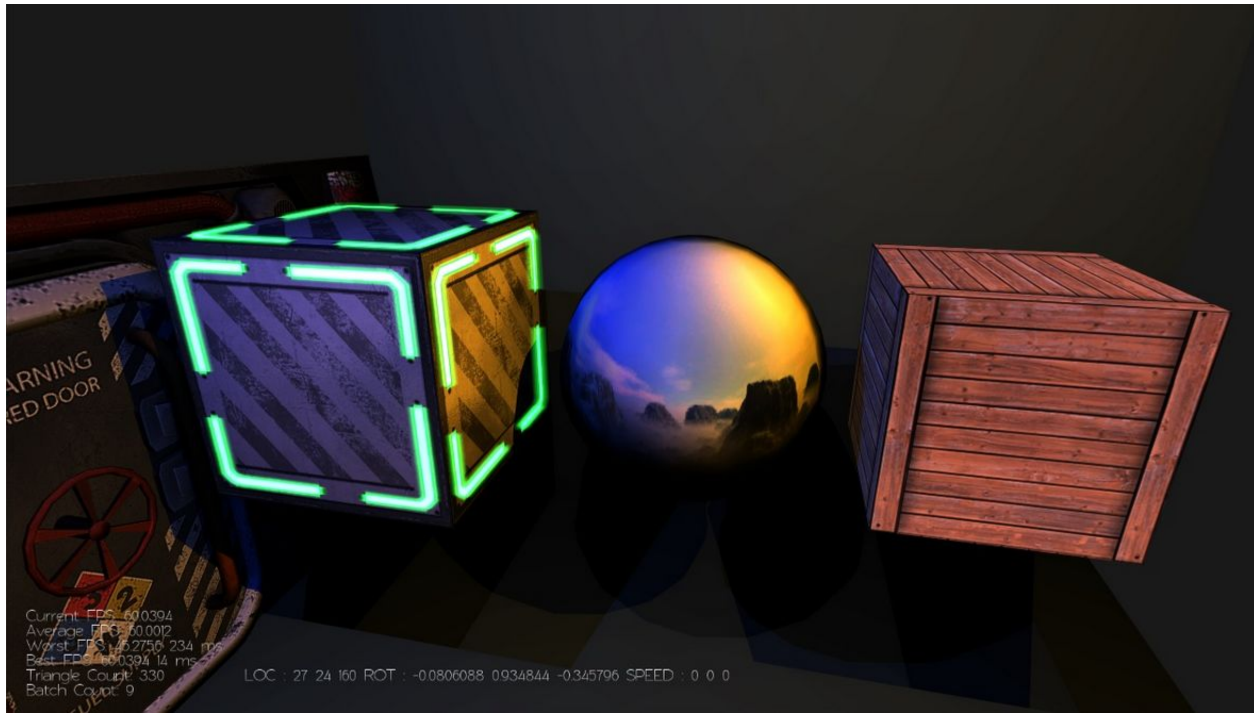
It’s the Open

Graphics

Library

Shader

Language



What's GLSL Like??

- C-style syntax
- Executes on GPU
- Even more type safety (no casting)

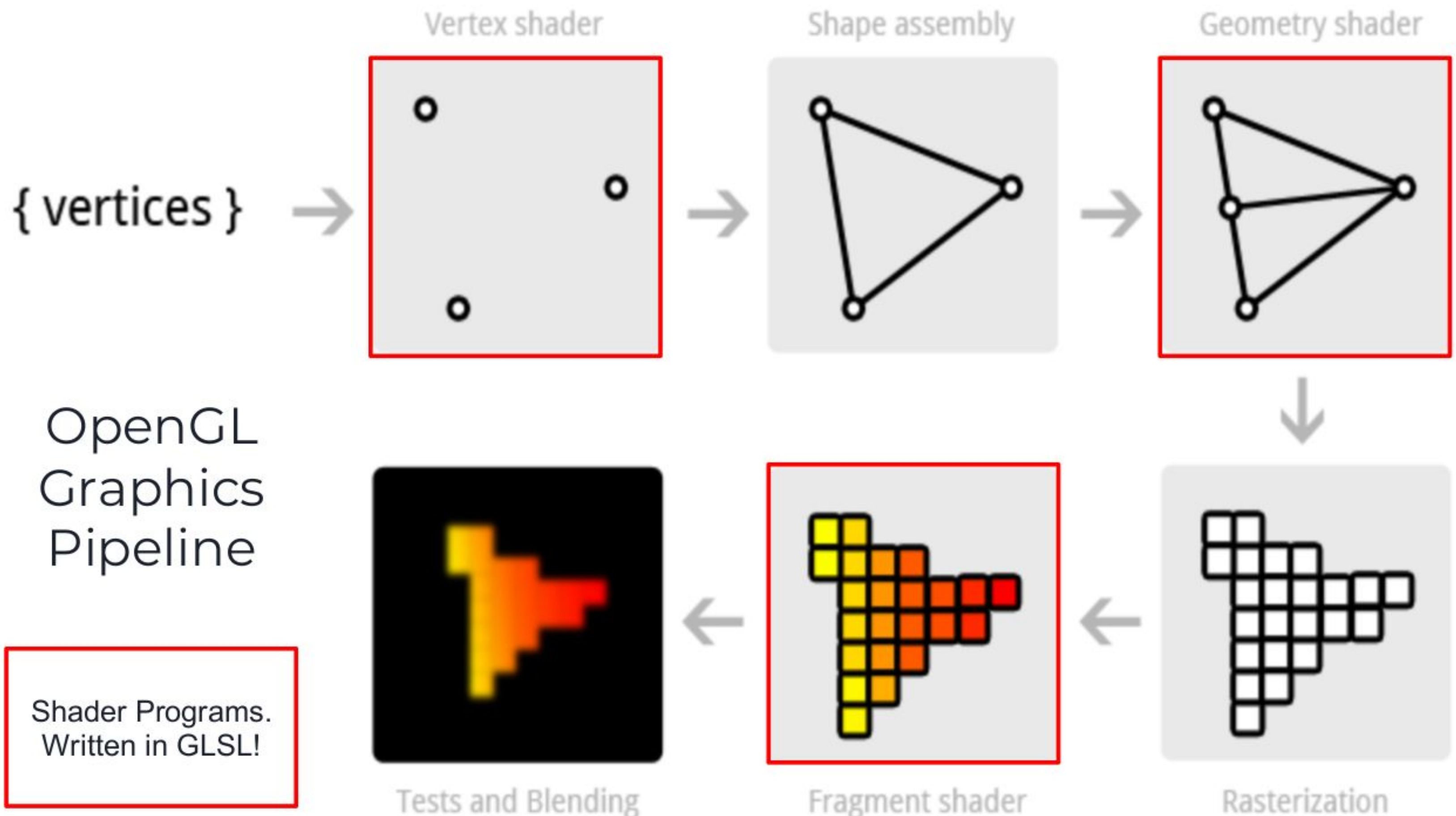
GLSL Challenges

- No recursion
- No dynamic memory allocation
- No pointers/objects
- No libraries
- No console I/O (must debug using colors!)

GLSL COde Behind the scenes in Mini Assignment 2

```
attribute vec3 vPos;
attribute vec3 vNormal;
attribute vec3 vColor;

uniform mat4 uMVMatrix;
uniform mat4 uPMatrix;
uniform mat3 uNMatrix;
uniform vec3 uAmbientColor;
uniform vec3 uLight1Pos;
uniform vec3 uLight2Pos;
uniform vec3 uLightColor;
uniform vec3 uColor;

varying vec3 vLightCoeff;
varying vec3 vColorInterp;

void main(void) {
    vec4 mvPosition = uMVMatrix*vec4(vPos, 1.0);
    gl_Position = uPMatrix * mvPosition;
    vec3 lightingDir = normalize(uLight1Pos - mvPosition.xyz);

    vec3 transformedNormal = uNMatrix*vNormal;
    //vec3 dPos = vec3(vec4(uLight1Pos, 1.0) - uMVMatrix*vec4(vPos, 1.0));

    float dirLightWeight = dot(transformedNormal, lightingDir);
    if (dirLightWeight < 0.0) { //Stupid fix for double sides for now
        dirLightWeight *= -1.0;
    }
    vLightCoeff = uAmbientColor + dirLightWeight*uLightColor;
    // The default value of the uniform color is (2, 2, 2)
    // So ignore and use the vColor from the buffer in this case.
    // Otherwise, override the buffer with the specified uniform color
    if (uColor[0] == 2.0 && uColor[1] == 2.0 && uColor[2] == 2.0) {
        vColorInterp = vColor;

    }
    else {
        vColorInterp = uColor;
    }
}
```

The Rendering Pipeline



OpenGL
Graphics
Pipeline

Shader Programs.
Written in GLSL!

GLSL Rendering Pipeline

- **Vertex Shader:** Runs automatically once per vertex. Must output the final vertex position to **gl_Position**, a 4D vector in homogenous coordinates. It can also output "varying" parameters which are sent to the fragment shader and interpolated

- **Fragment Shader:** Runs automatically once per pixel (aka fragment). Runs after the vertex shader, and must output the final pixel color to **gl_FragColor**, a 4D vector holding (red, green, blue, alpha), all of which are between 0.0 and 1.0

GLSL Type specifiers

- **attribute**: Variables that are sent over via buffers to the vertex shader. One per vertex.

  E.g. To send over positions, colors, and normals at each vertex

- **uniform**: A variable which is constant across all shaders. E.g. to store info about light positions and objects in scene
- **varying**: A variable that is shared between the vertex shader and fragment shader. Its value is interpolated via barycentric coordinates in the fragment shader.

GLSL Built IN vector types

- **vecN**

```
vec3 yxz_comp = other_vec.yxz;

vec4 myvec4 = vec4(yxz_comp, 1.0);

vec4 urvec3 = vec3(1.0, 0.0, 0.0);

float projMag = dot(yxz_comp, urvec3);
```

Basic Triangle Example

1. **triangle.vert (vertex shader)**

```
attribute vec2 a_position;
attribute vec3 a_color;
varying vec3 v_color;

void main() {
  gl_Position = vec4(a_position, 0, 1);
  v_color = a_color;
}
```

- **Position of vertex is always set with gl_Position**
- **Positions are xyz in homogenous coordinates, so a vec4 is needed**

Basic Triangle Example

1. **triangle.frag (fragment shader)**

```
precision mediump float;

// The color of the pixel in this fragment,
// interpolated via barycentric coordinates
// from positions of triangle vertices
varying vec3 v_color;

void main() {
    // Every pixel has this output
    gl_FragColor = vec4(v_color, 1.0);
}
```

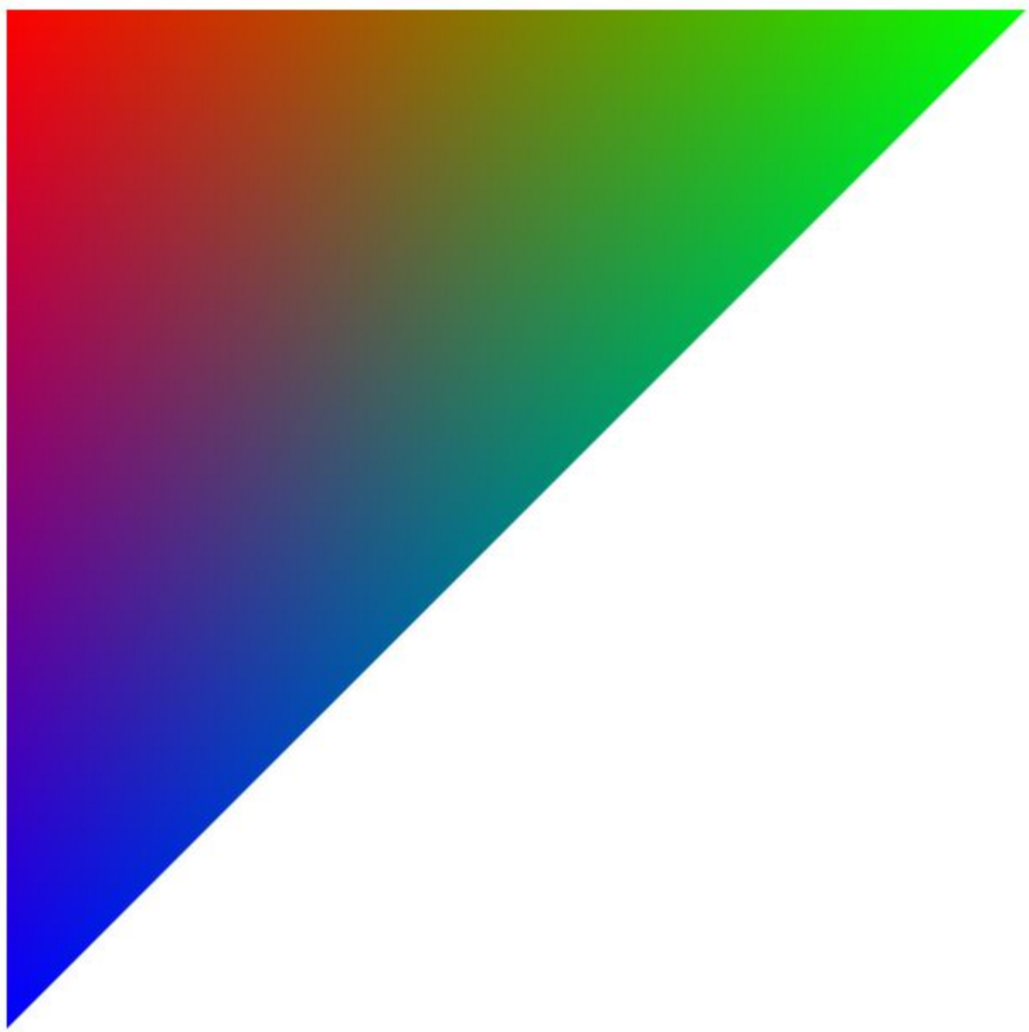
- **v_color is shared between vertex and fragment shaders**
- **Colors are in RGBA format, so a vec4 is needed**

Basic Triangle Example

1. **triangle.frag (fragment shader)**

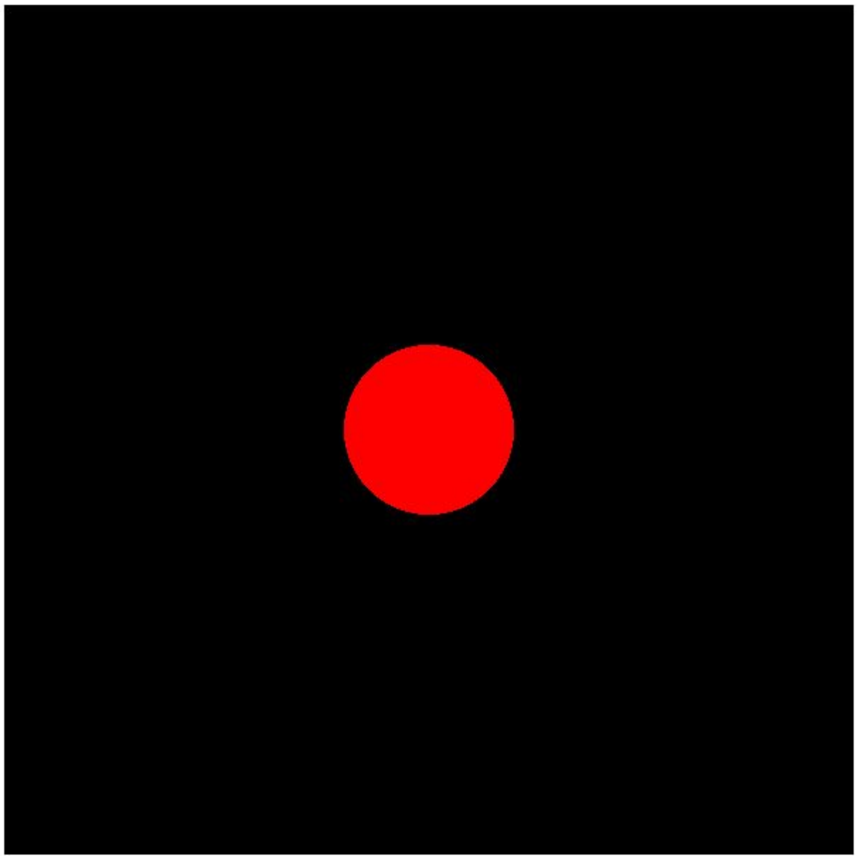
```
precision mediump float;

// The color of the pixel in this fragment,
// interpolated via barycentric coordinates
// from positions of triangle vertices
varying vec3 v_color;

void main() {
    gl_FragColor = vec4(v_color, 1.0);
}
```

Basic Triangle Example: Result



Circle Example: Task 1

Draw a red circle at the center of the viewing window whose radius is determined by the uniform set in circle.html



Code Hints:

- Use the - operator to do vector subtraction, and subtract off the center from the position of each pixel
- Use the dot operator to do a dot product of two vectors (how does the dot product help us determine the radius?)

Circle Example: Task 2

Get the circle to move from the left to the right in an animation loop

Circle Example: Task 3

Get the circle to look like a yellow blob on a red background

1. Make the red component 1.0
2. Make the green component **exp(-dist^2/radius^2)**