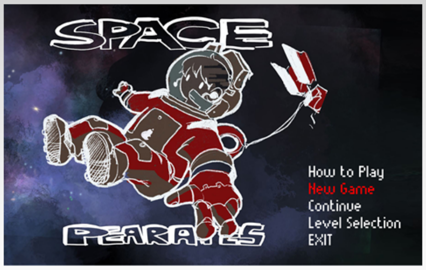
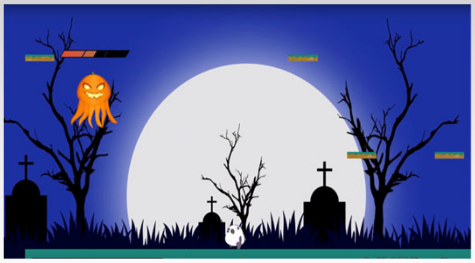
CPSC 427
Video Game Programming

OpenGL/Shaders





Rendering Pipeline

Abstract model of

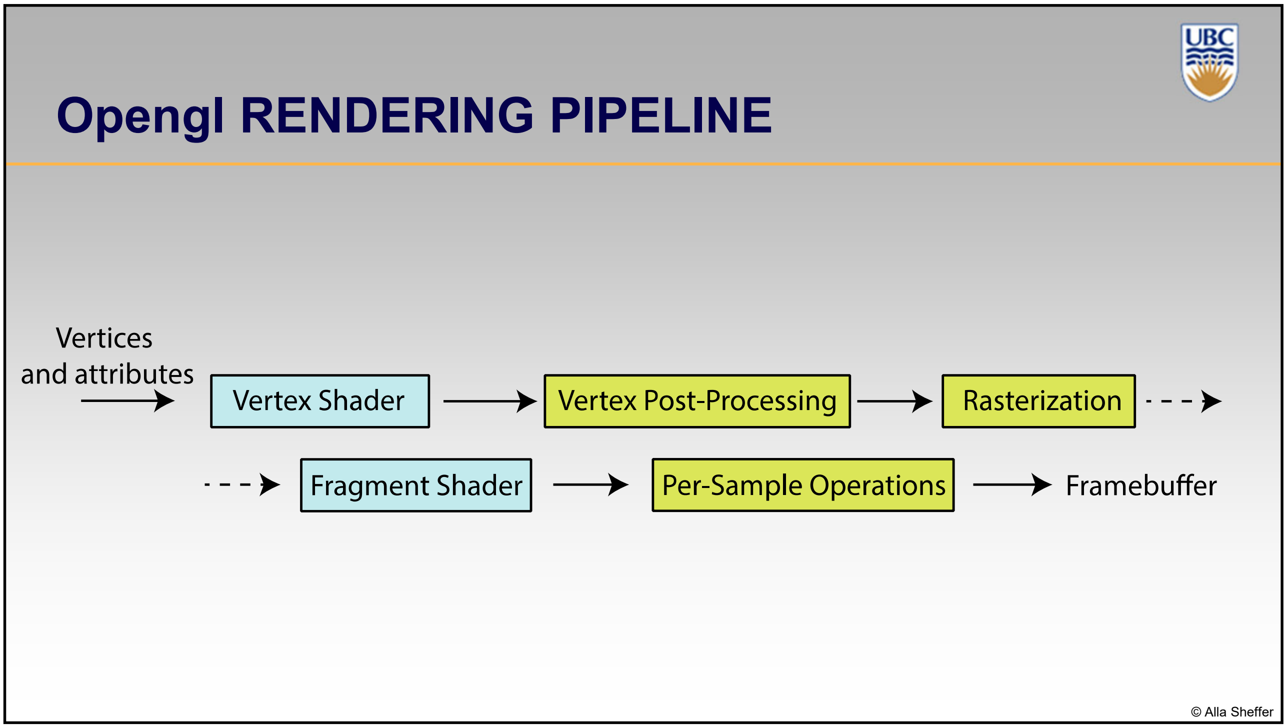
- sequence of operations to transform geometric model into digital image
- graphics hardware workflow

Underlying API (application programming interface) model for programming graphics hardware

- OpenGL
- Direct 3D

Actual implementations vary

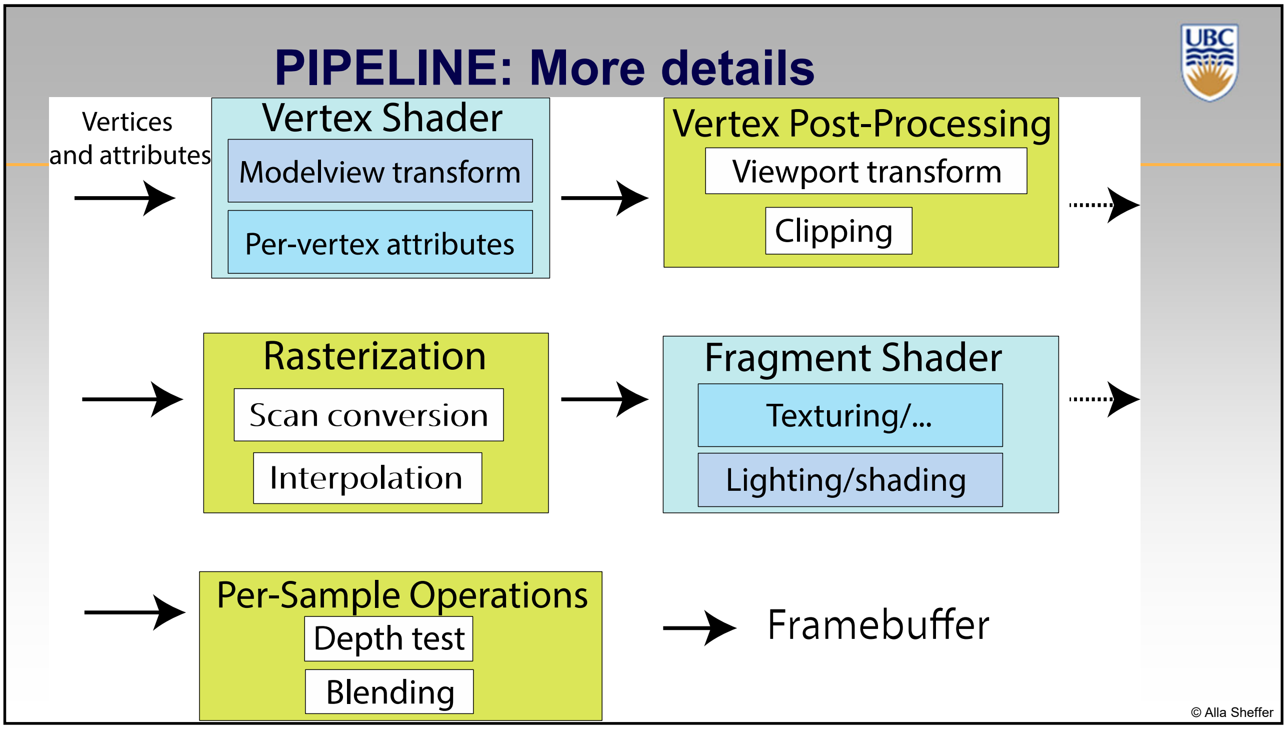
Optimized for GPU => parallel computation



Opengl RENDERING PIPELINE

Vertices
and attributes → Vertex Shader → Vertex Post-Processing → Rasterization - - →

- - → Fragment Shader → Per-Sample Operations → Framebuffer



PIPELINE: More details

Vertices
and attributes →

Vertex Shader
- Modelview transform
- Per-vertex attributes

→

Vertex Post-Processing
- Viewport transform
- Clipping

→

→

Rasterization
- Scan conversion
- Interpolation

→

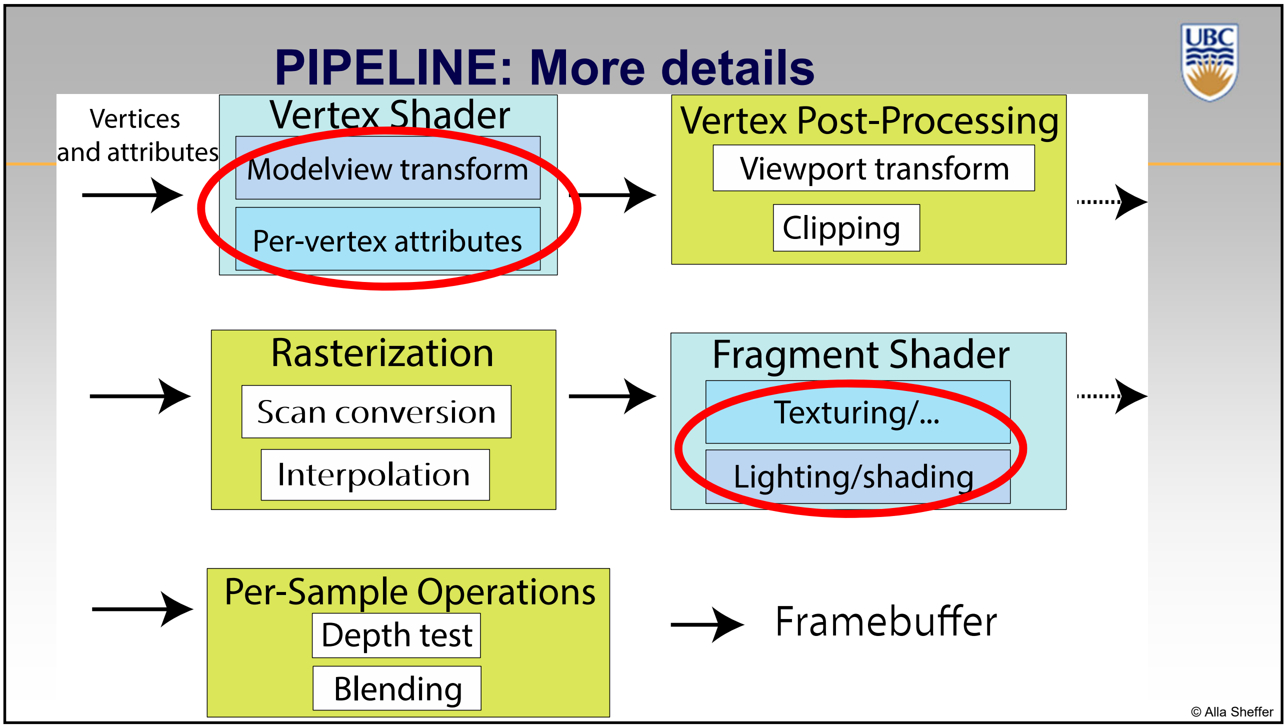
Fragment Shader
- Texturing/...
- Lighting/shading

→

→

Per-Sample Operations
- Depth test
- Blending

→ Framebuffer



PIPELINE: More details

Vertices and attributes →

Vertex Shader
- Modelview transform
- Per-vertex attributes

→

Vertex Post-Processing
- Viewport transform
- Clipping

→

Rasterization
- Scan conversion
- Interpolation

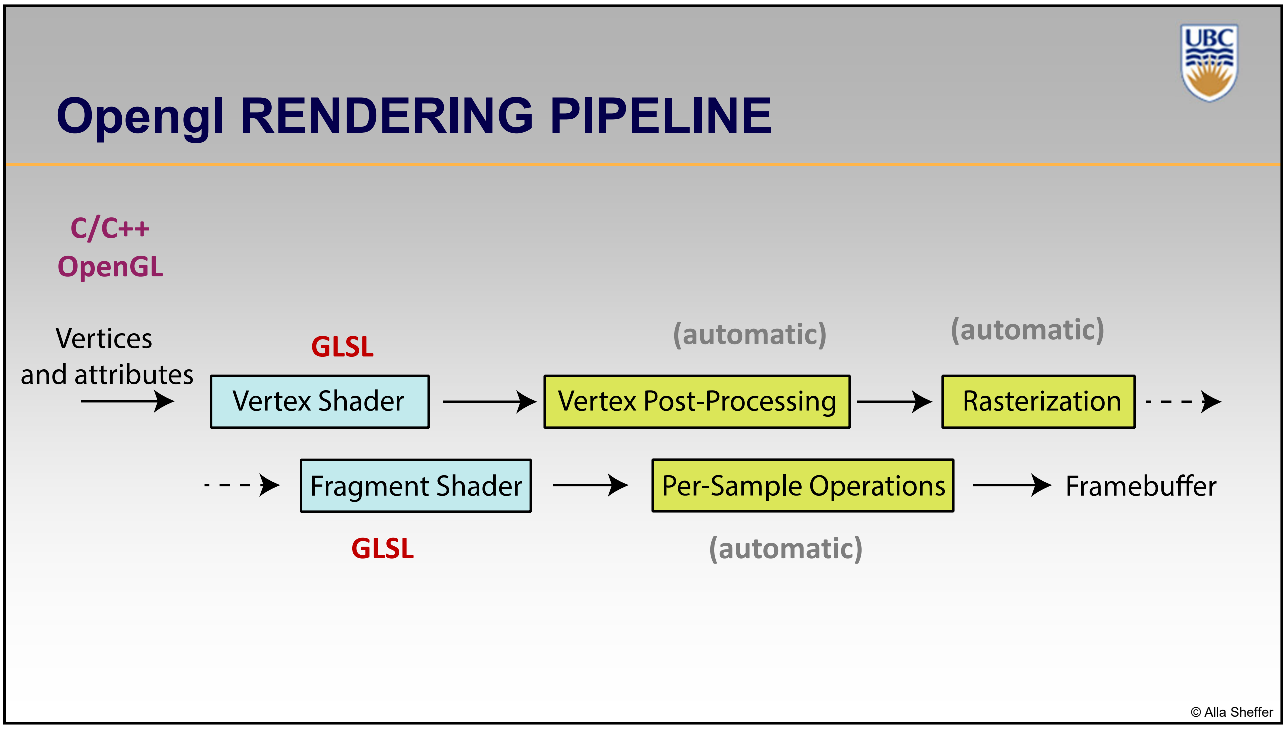
→

Fragment Shader
- Texturing/…
- Lighting/shading

→

Per-Sample Operations
- Depth test
- Blending

→ Framebuffer



Opengl RENDERING PIPELINE

C/C++
OpenGL

Vertices and attributes → Vertex Shader (GLSL) → Vertex Post-Processing (automatic) → Rasterization (automatic) - - →

- - → Fragment Shader (GLSL) → Per-Sample Operations (automatic) → Framebuffer

openGL

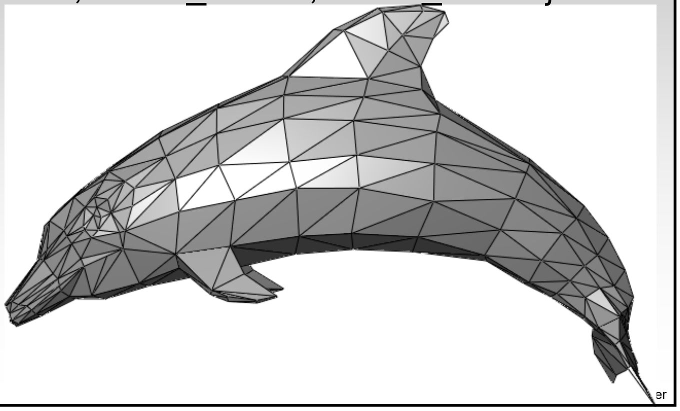
- Low-level graphics API
- C Interface accessed from C++

- Mesh: **Vertex Buffers** and **Index Buffers**
- Materials: **Shaders**, **Textures**, **Samplers** and **Uniforms**
- Camera: **(View)** and **(Projection)** matrices

GEOMETRY

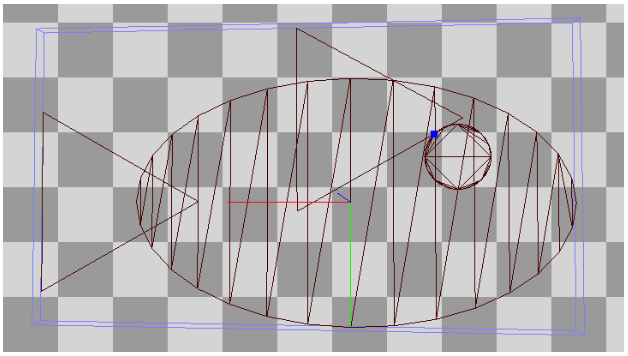
Triangle meshes

- Set of vertices
- Triangle defines as {vertex_index1, vertex_index2, vertex_index3}



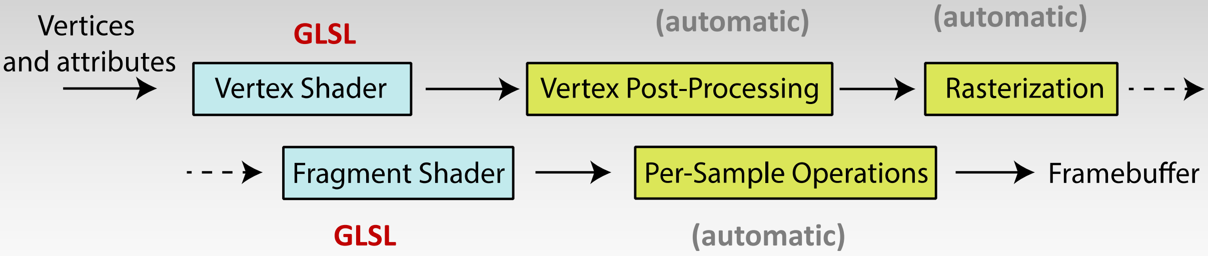
```
vertices[0].position = { -0.54, +1.34, -0.01 };
vertices[1].position = { +0.75, +1.21, -0.01 };
..
vertices[150].position = { -1.22, +3.59, -0.01 };

uint16_t indices[] = { 0, 3, 1,.. , 152, 150 };
Gluint ibo, vbo;
glGenBuffers(vbo);
glBindBuffer(vbo);
glBufferData(vbo, vertices);

glGenBuffers(ibo);
glBindBuffer(ibo);
glBufferData(ibo, indices);
```

GEOMETRY
C/C++ OPENGL



Opengl RENDERING PIPELINE

C/C++
OpenGL



Vertices and attributes → Vertex Shader (GLSL) → Vertex Post-Processing (automatic) → Rasterization (automatic) - - →

- - → Fragment Shader (GLSL) → Per-Sample Operations (automatic) → Framebuffer

Vertex Shader

Vertices
and attributes
Vertex Shader

- Called SEPARATELY for each vertex

- Default: No connectivity info

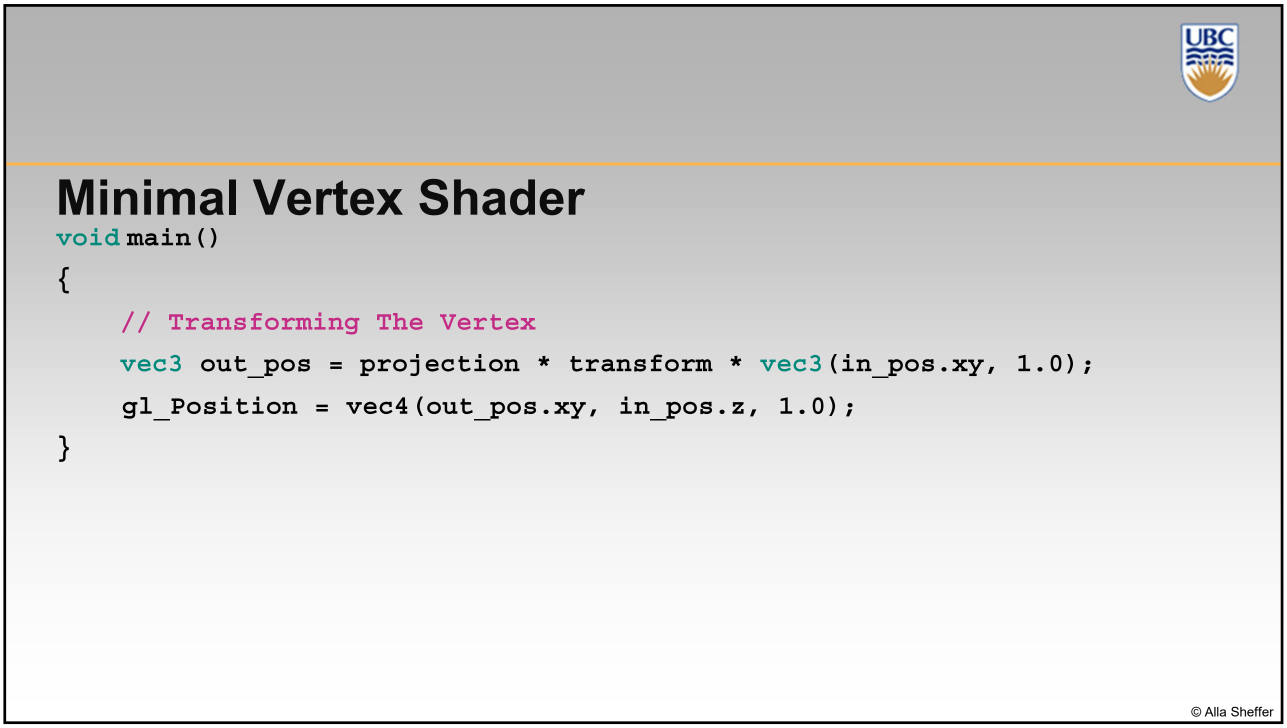
- **Input:** vertex coordinates in Object Coordinate System
- **Main goal:** set **gl_Position**

Object coordinates -> WORLD coordinates -> VIEW coordinates/Clip Coordinates

FRAGMENT SHADER

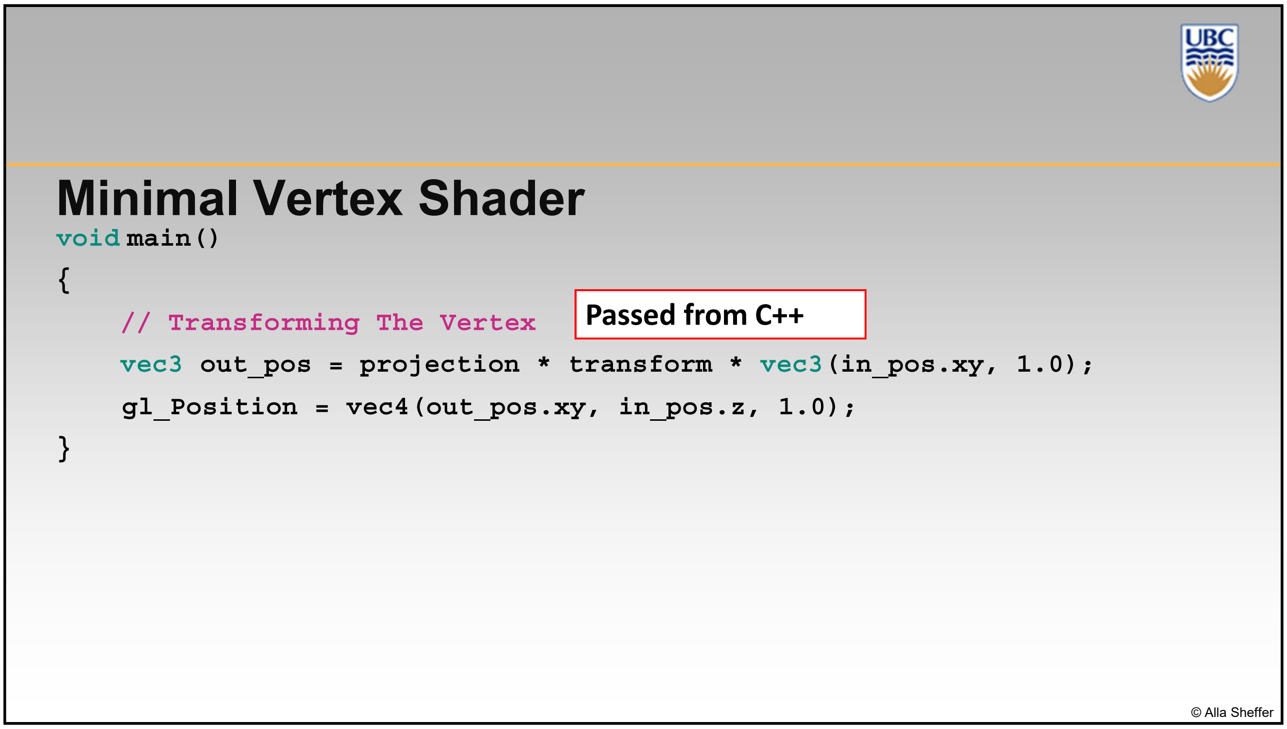
- ***Common Tasks:***
  - texture mapping
  - per-pixel lighting and shading

- ***Fragment Shader = Pixel Shader***



Minimal Vertex Shader

```
void main()
{
    // Transforming The Vertex
    vec3 out_pos = projection * transform * vec3(in_pos.xy, 1.0);
    gl_Position = vec4(out_pos.xy, in_pos.z, 1.0);
}
```



Minimal Vertex Shader

```
void main()
{
    // Transforming The Vertex
    vec3 out_pos = projection * transform * vec3(in_pos.xy, 1.0);
    gl_Position = vec4(out_pos.xy, in_pos.z, 1.0);
}
```

Passed from C++

Minimal Vertex Shader

```
void main()
{
    // Transforming The Vertex
    vec3 out_pos = projection * transform * vec3(in_pos.xy, 1.0);
    gl_Position = vec4(out_pos.xy, in_pos.z, 1.0);
}
```

Passed from C++

$$\begin{pmatrix} x \\ y \\ z \\ (w) \end{pmatrix}$$

Minimal Vertex Shader

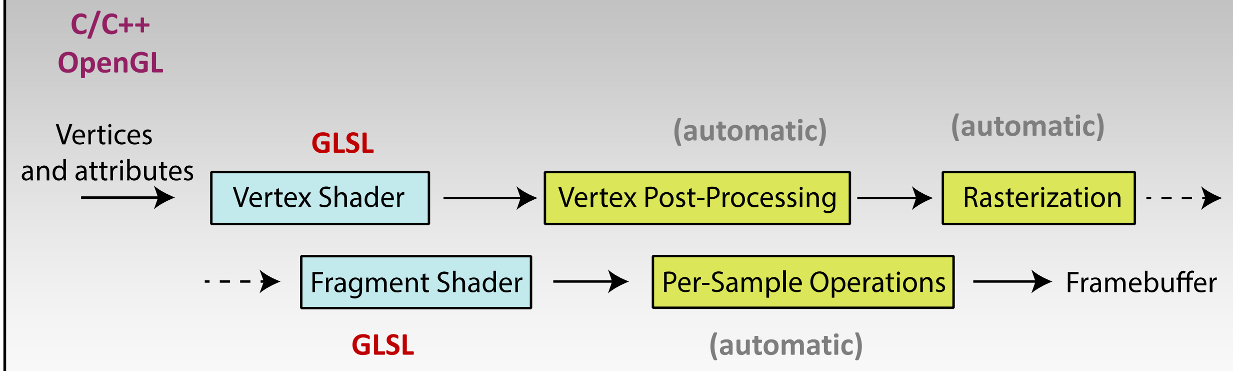
```
void main()
{
    // Transforming The Vertex
    vec3 out_pos = projection * transform * vec3(in_pos.xy, 1.0);
    gl_Position = vec4(out_pos.xy, in_pos.z, 1.0);
}
```

Passed from C++

$$\begin{pmatrix} x \\ y \\ z \\ (w) \end{pmatrix}$$

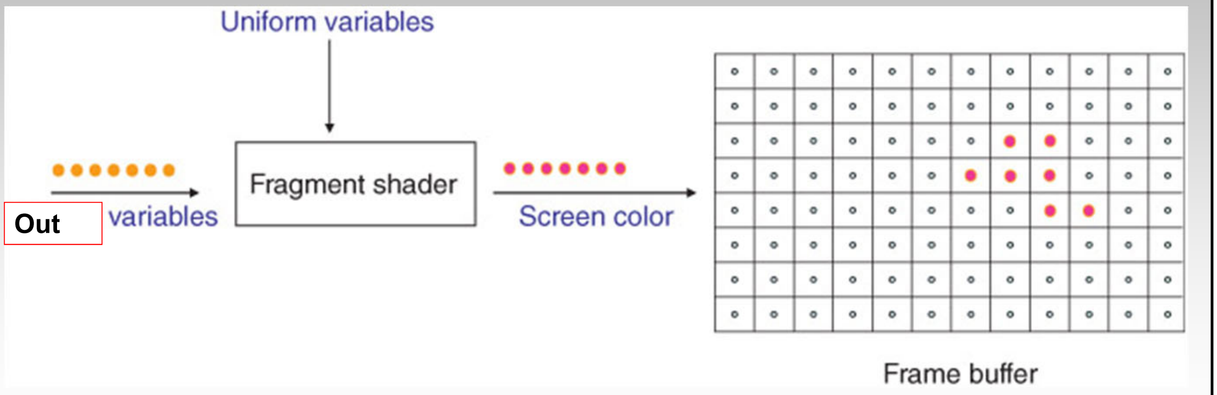
View coordinate system

Opengl RENDERING PIPELINE

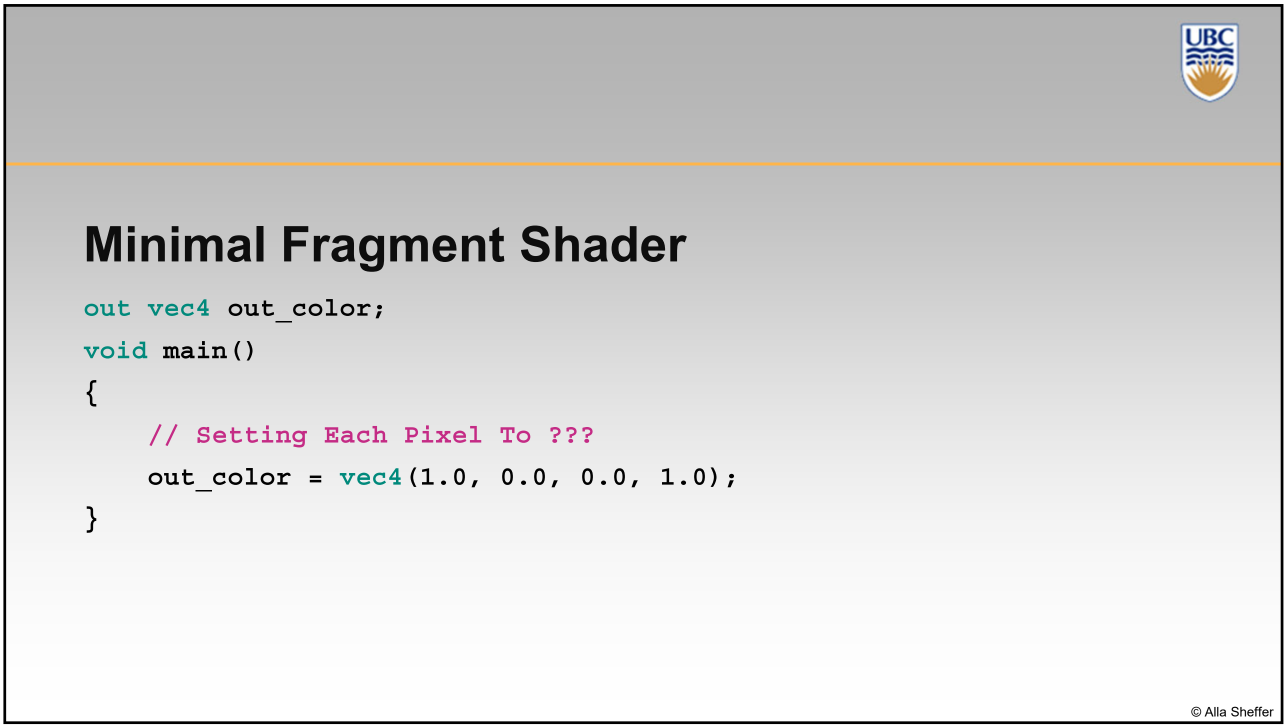


© Alla Sheffer

Fragment SHADER



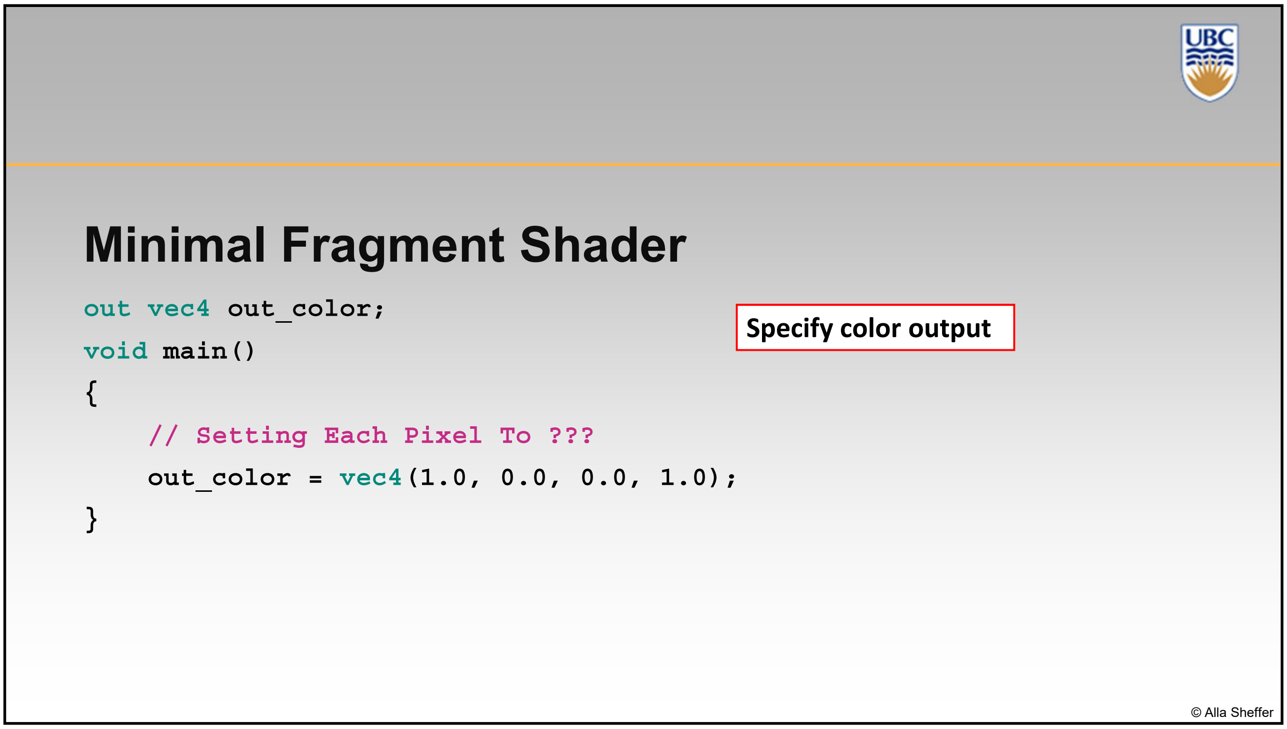
© Alla Sheffer



Minimal Fragment Shader

```
out vec4 out_color;
void main()
{
    // Setting Each Pixel To ???
    out_color = vec4(1.0, 0.0, 0.0, 1.0);
}
```

© Alla Sheffer

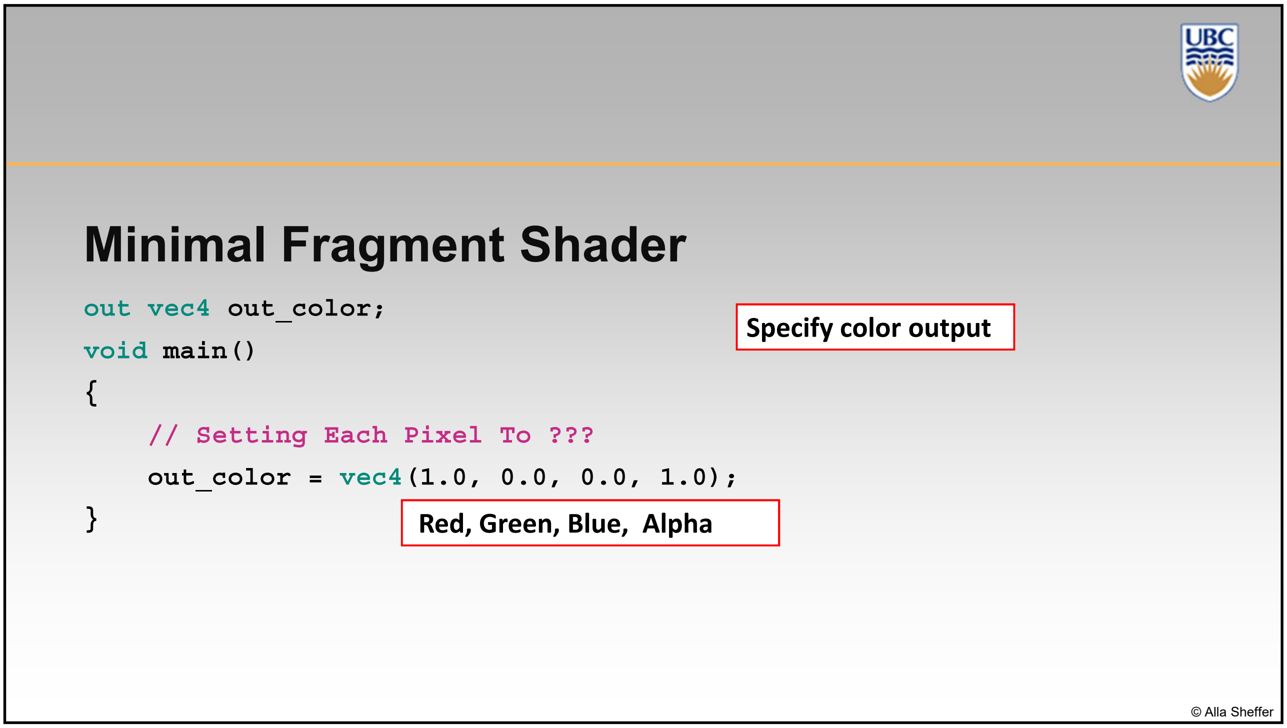


Minimal Fragment Shader

```
out vec4 out_color;
void main()
{
    // Setting Each Pixel To ???
    out_color = vec4(1.0, 0.0, 0.0, 1.0);
}
```

Specify color output

© Alla Sheffer

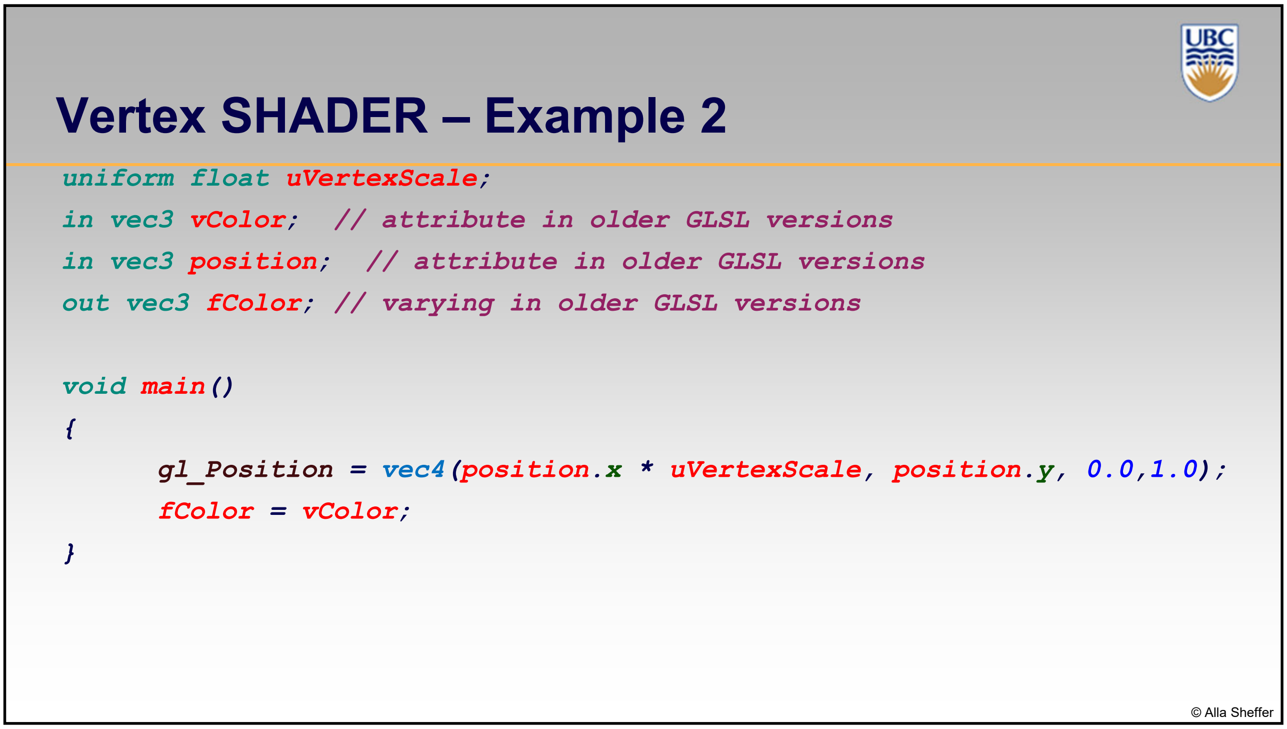


Minimal Fragment Shader

```
out vec4 out_color;
void main()
{
    // Setting Each Pixel To ???
    out_color = vec4(1.0, 0.0, 0.0, 1.0);
}
```

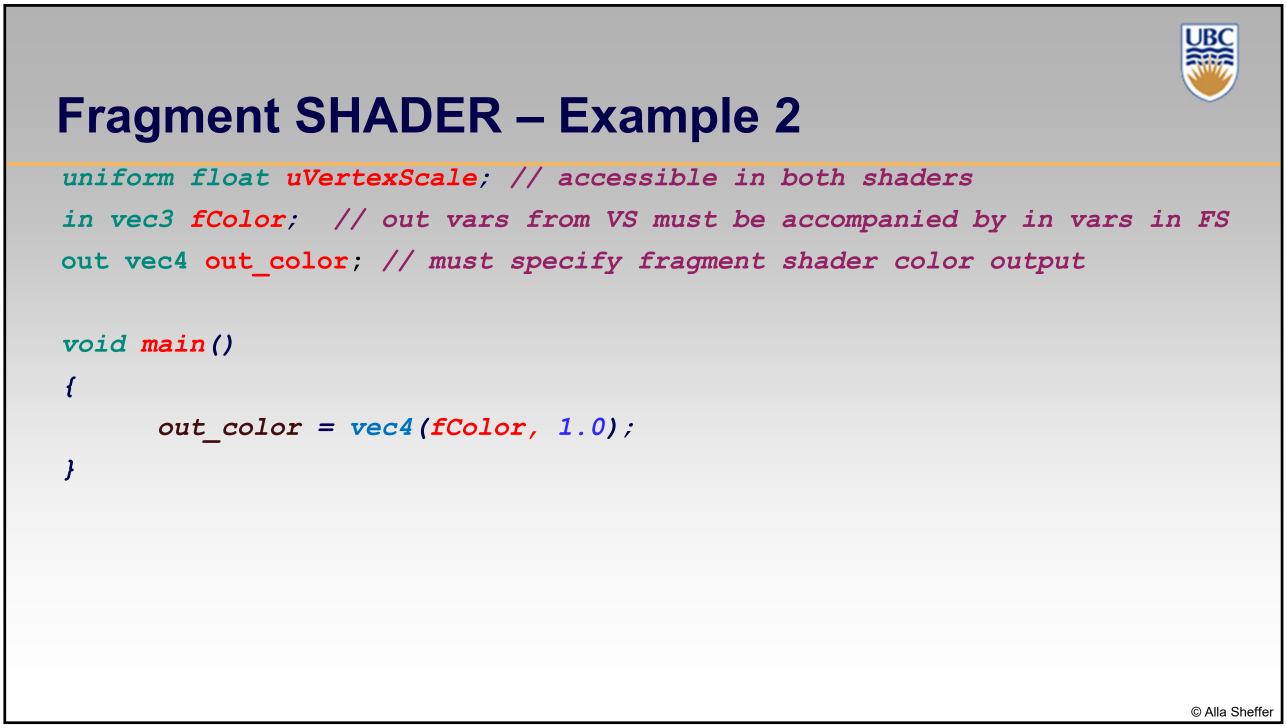
Specify color output

Red, Green, Blue, Alpha



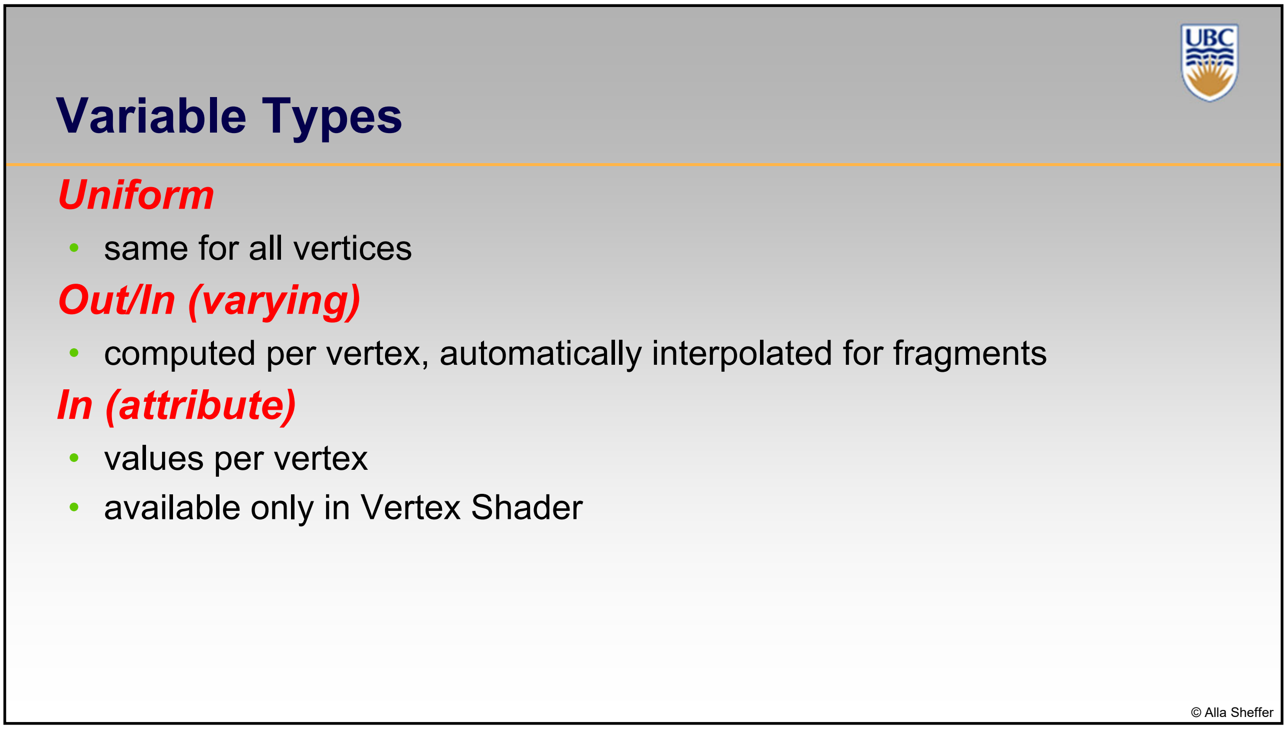
Vertex SHADER – Example 2

```
uniform float uVertexScale;
in vec3 vColor;  // attribute in older GLSL versions
in vec3 position;  // attribute in older GLSL versions
out vec3 fColor; // varying in older GLSL versions

void main()
{
      gl_Position = vec4(position.x * uVertexScale, position.y, 0.0,1.0);
      fColor = vColor;
}
```



Fragment SHADER – Example 2

```
uniform float uVertexScale; // accessible in both shaders
in vec3 fColor;  // out vars from VS must be accompanied by in vars in FS
out vec4 out_color; // must specify fragment shader color output

void main()
{
      out_color = vec4(fColor, 1.0);
}
```



Variable Types

Uniform

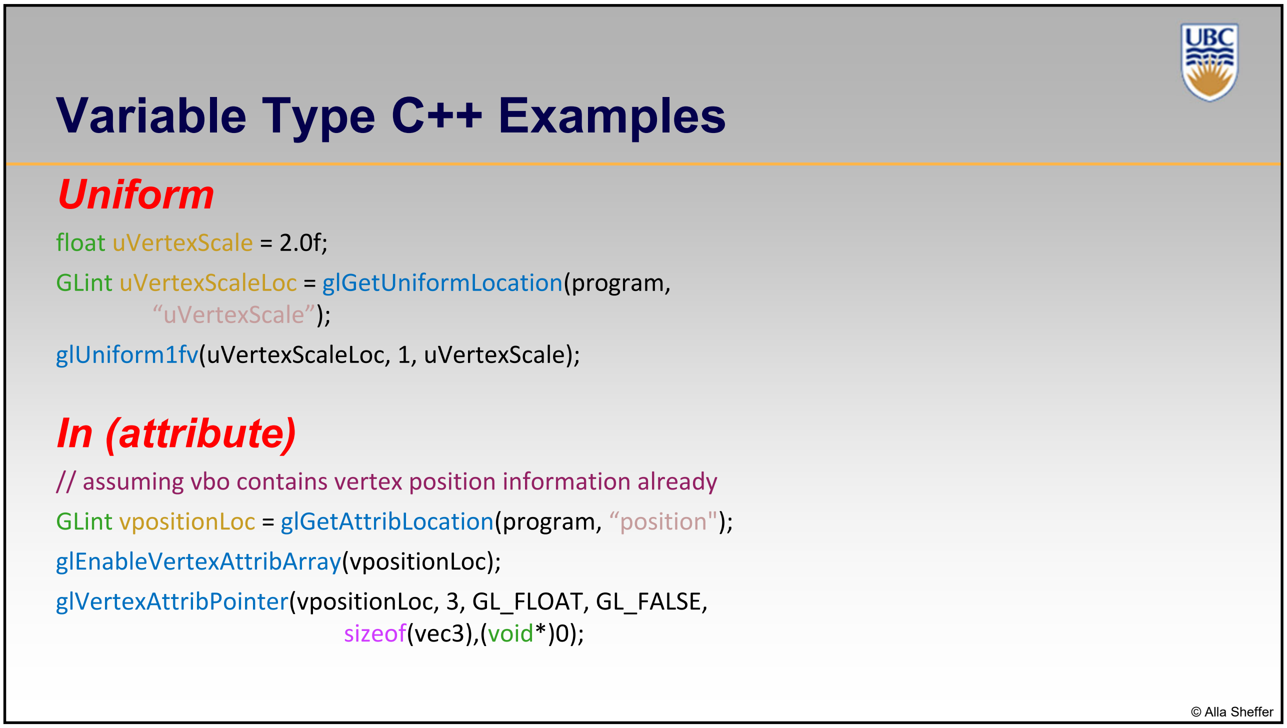
- same for all vertices

Out/In (varying)

- computed per vertex, automatically interpolated for fragments

In (attribute)

- values per vertex
- available only in Vertex Shader



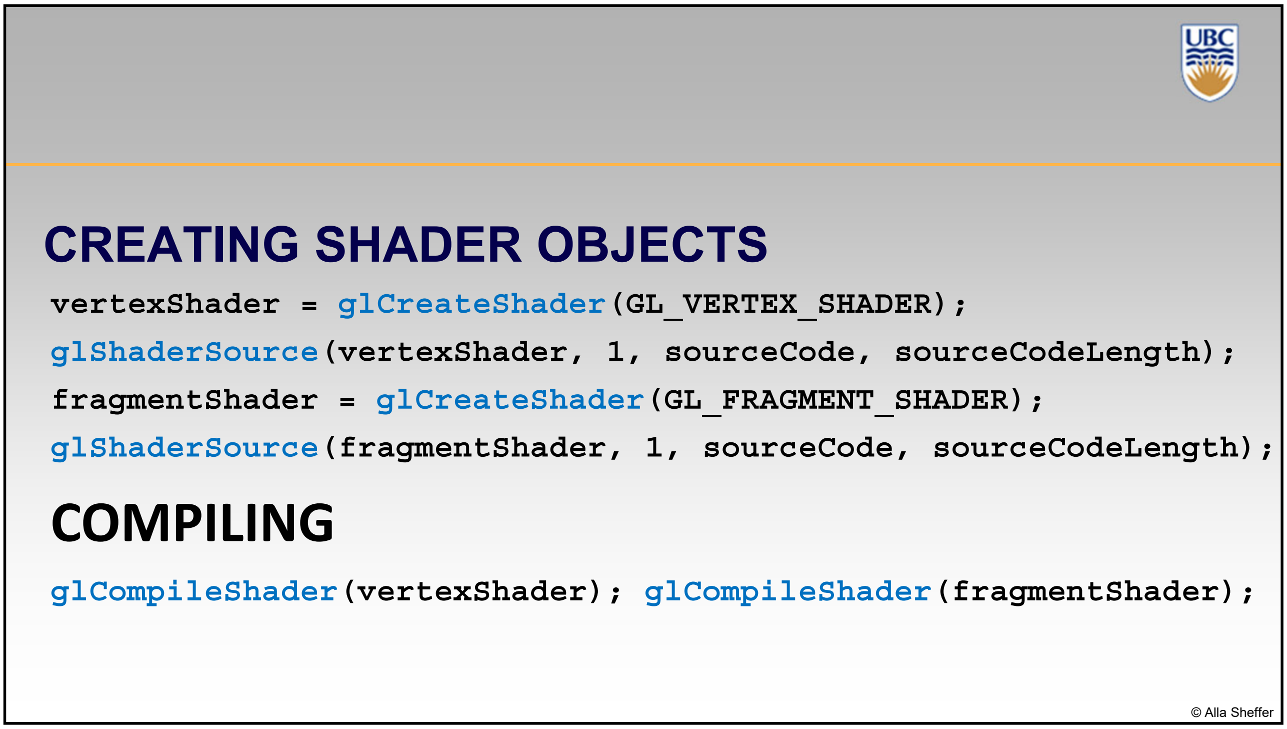
Variable Type C++ Examples

Uniform

```
float uVertexScale = 2.0f;
GLint uVertexScaleLoc = glGetUniformLocation(program,
          “uVertexScale”);
glUniform1fv(uVertexScaleLoc, 1, uVertexScale);
```

In (attribute)

```
// assuming vbo contains vertex position information already
GLint vpositionLoc = glGetAttribLocation(program, “position");
glEnableVertexAttribArray(vpositionLoc);
glVertexAttribPointer(vpositionLoc, 3, GL_FLOAT, GL_FALSE,
                              sizeof(vec3),(void*)0);
```

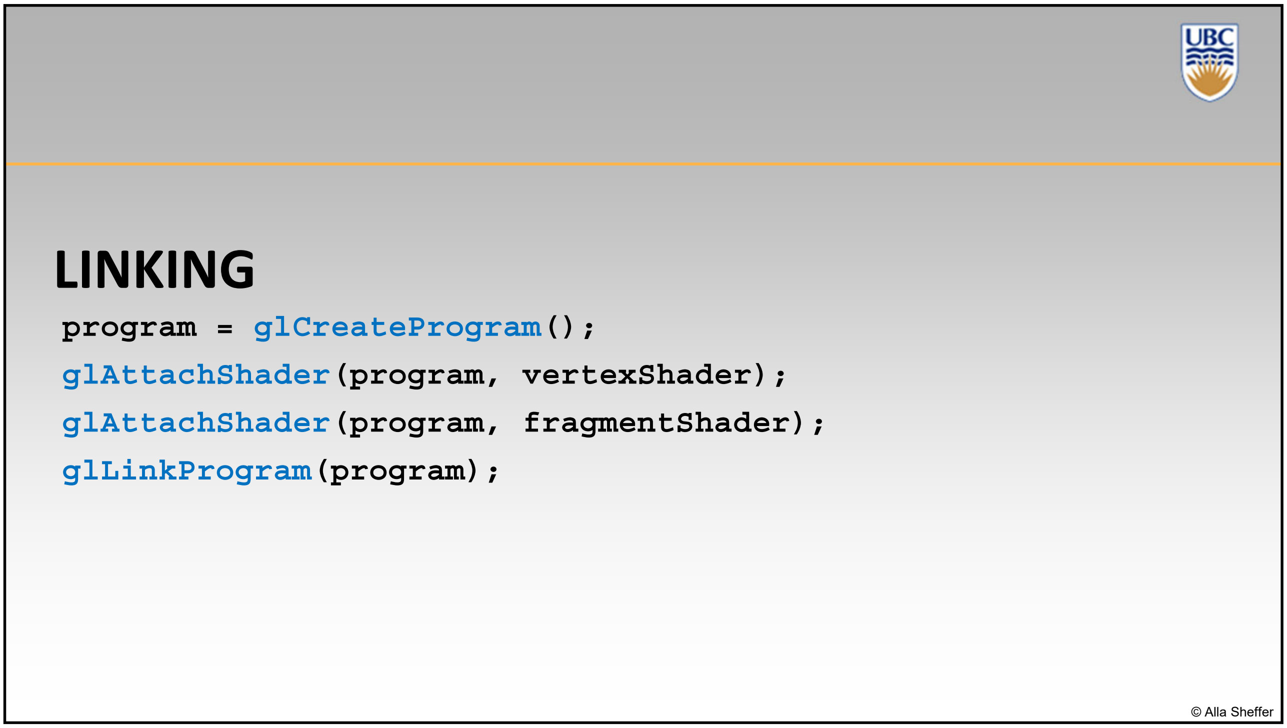


CREATING SHADER OBJECTS

```
vertexShader = glCreateShader(GL_VERTEX_SHADER);
glShaderSource(vertexShader, 1, sourceCode, sourceCodeLength);
fragmentShader = glCreateShader(GL_FRAGMENT_SHADER);
glShaderSource(fragmentShader, 1, sourceCode, sourceCodeLength);
```

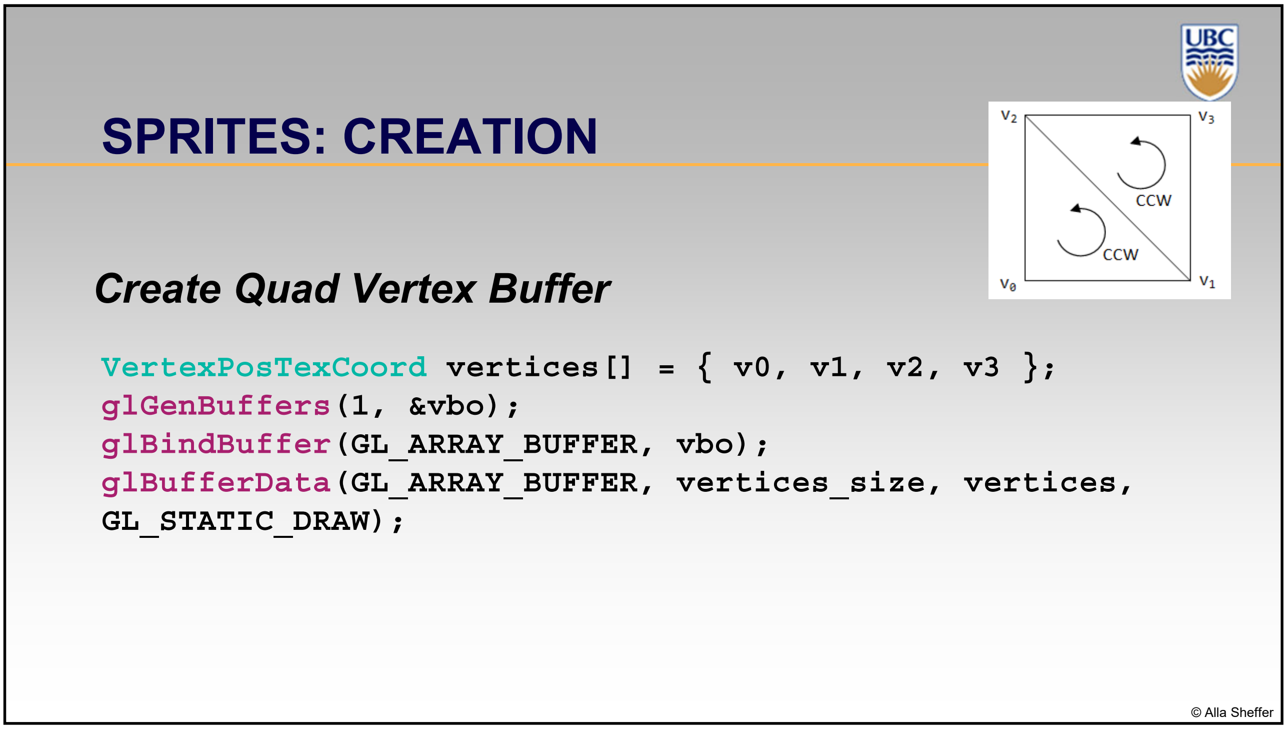
COMPILING

```
glCompileShader(vertexShader); glCompileShader(fragmentShader);
```



LINKING

```
program = glCreateProgram();
glAttachShader(program, vertexShader);
glAttachShader(program, fragmentShader);
glLinkProgram(program);
```

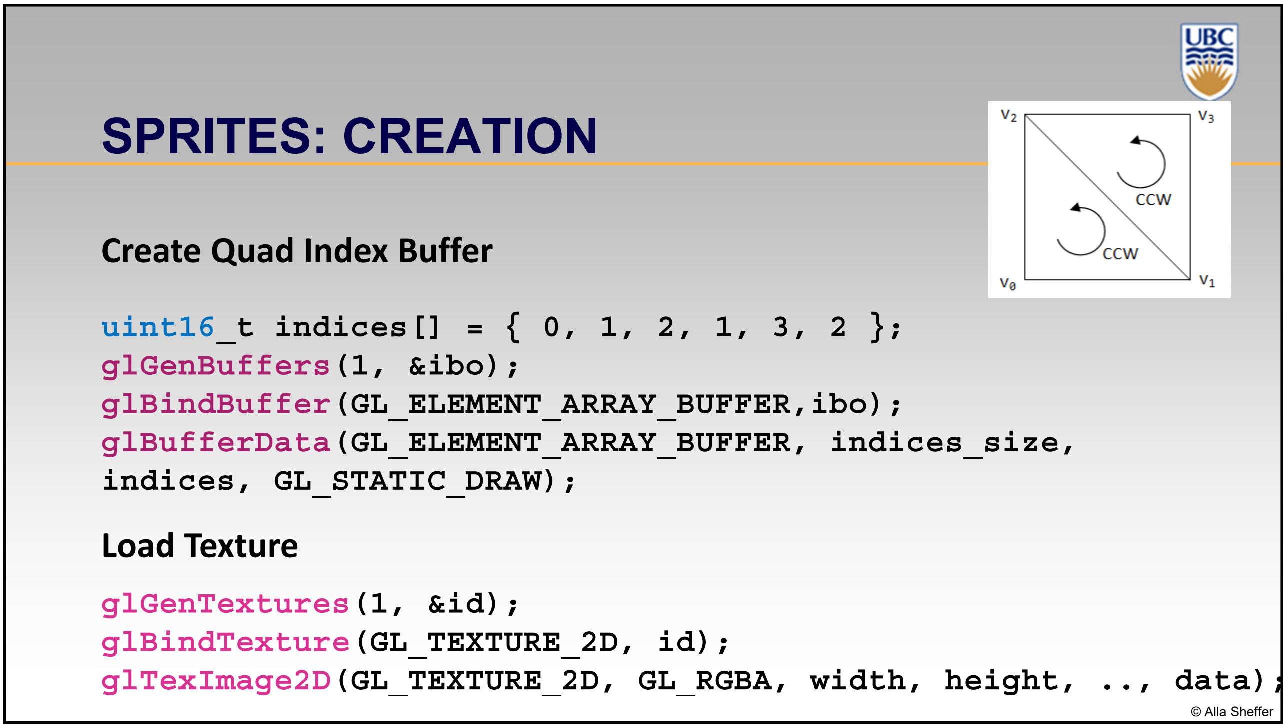


SPRITES: CREATION

V₂
V₃
CCW
CCW
V₀
V₁

Create Quad Vertex Buffer

```
VertexPosTexCoord vertices[] = { v0, v1, v2, v3 };
glGenBuffers(1, &vbo);
glBindBuffer(GL_ARRAY_BUFFER, vbo);
glBufferData(GL_ARRAY_BUFFER, vertices_size, vertices,
GL_STATIC_DRAW);
```



SPRITES: CREATION

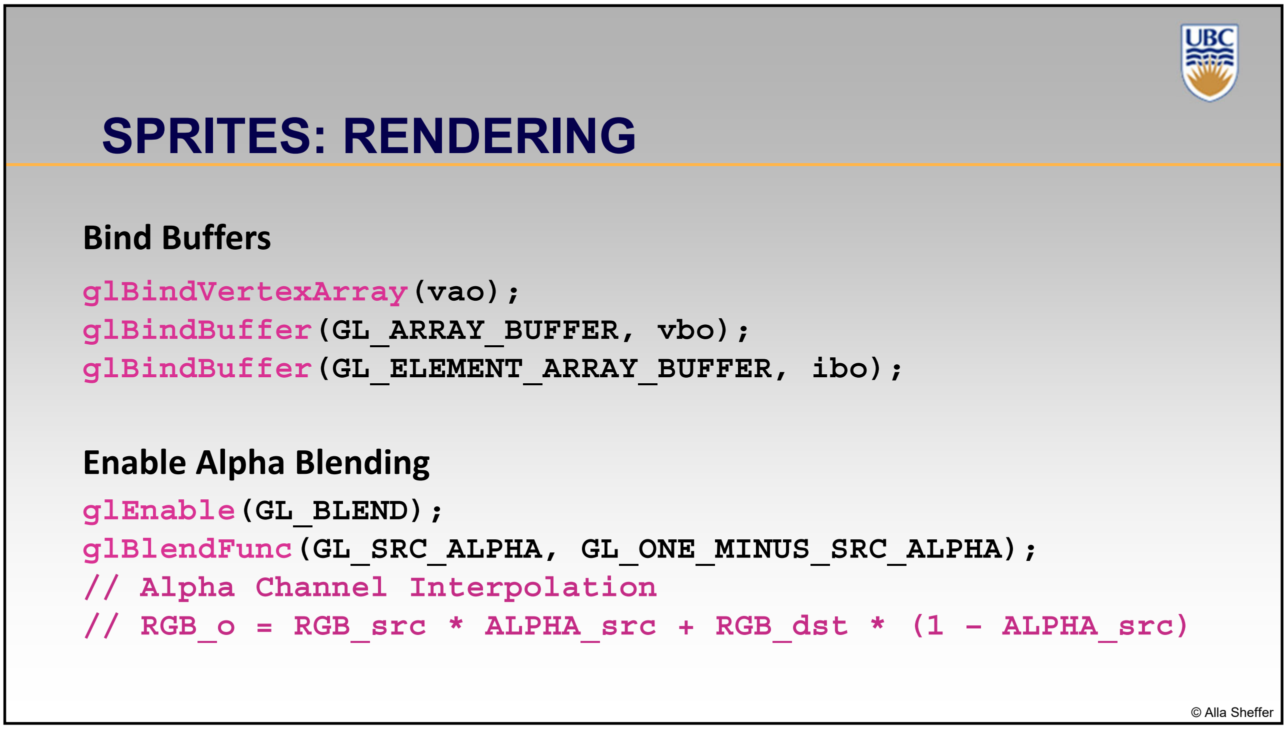
V₂
V₃
CCW
CCW
V₀
V₁

Create Quad Index Buffer

```
uint16_t indices[] = { 0, 1, 2, 1, 3, 2 };
glGenBuffers(1, &ibo);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER,ibo);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices_size,
indices, GL_STATIC_DRAW);
```

Load Texture

```
glGenTextures(1, &id);
glBindTexture(GL_TEXTURE_2D, id);
glTexImage2D(GL_TEXTURE_2D, GL_RGBA, width, height, .., data);
```



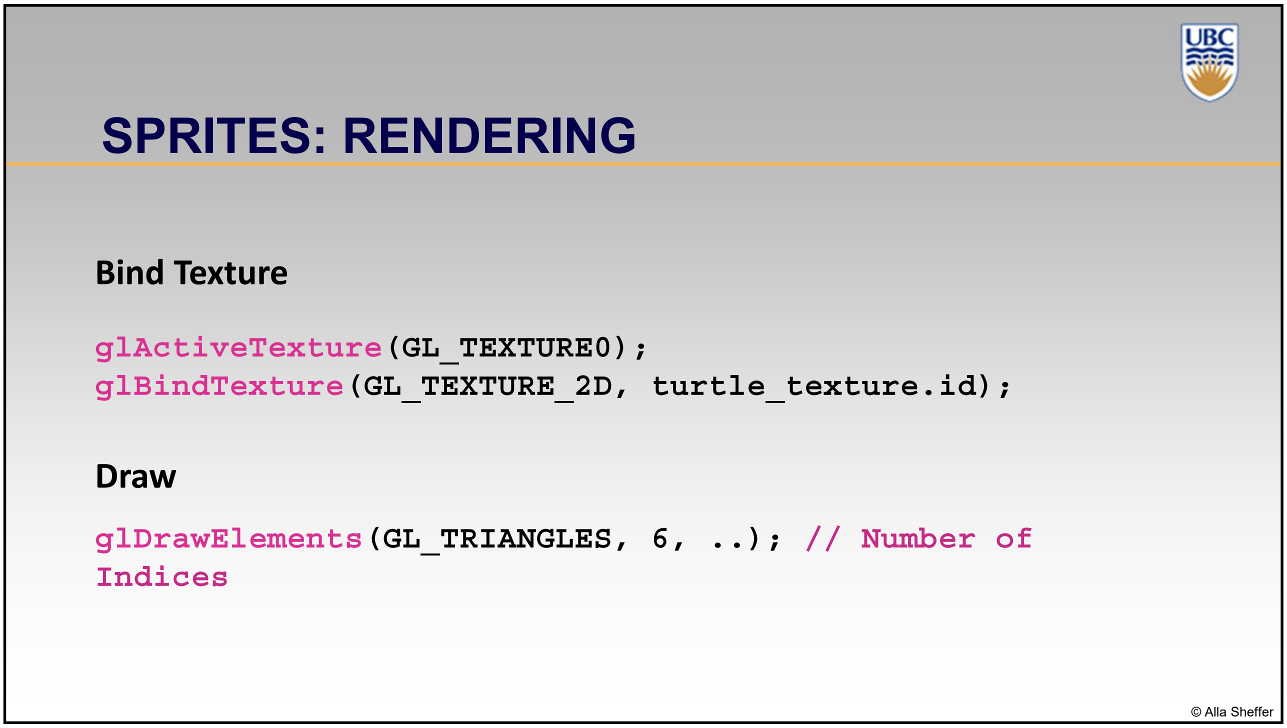
SPRITES: RENDERING

Bind Buffers

```
glBindVertexArray(vao);
glBindBuffer(GL_ARRAY_BUFFER, vbo);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, ibo);
```

Enable Alpha Blending

```
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
// Alpha Channel Interpolation
// RGB_o = RGB_src * ALPHA_src + RGB_dst * (1 - ALPHA_src)
```



SPRITES: RENDERING

Bind Texture

```
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, turtle_texture.id);
```

Draw

```
glDrawElements(GL_TRIANGLES, 6, ..); // Number of
Indices
```