



INTRO TO OPENG

Build and render a spinning
cube from scratch!

Go to this link and clone the repo:

github.com/dvcalex/openglworkshop1

WHAT YOU NEED



Go to this link and clone the repo:

github.com/dvcalex/openglworkshop1

WHAT YOU WILL LEARN

- How to create a window and OpenGL context with GLFW.
- The render pipeline.
- How to define 3D geometry.
- How to upload data to the GPU.
- How to write shaders in GLSL.
- How to use transformation matrices.
- More!

I will not be teaching you c++ :)

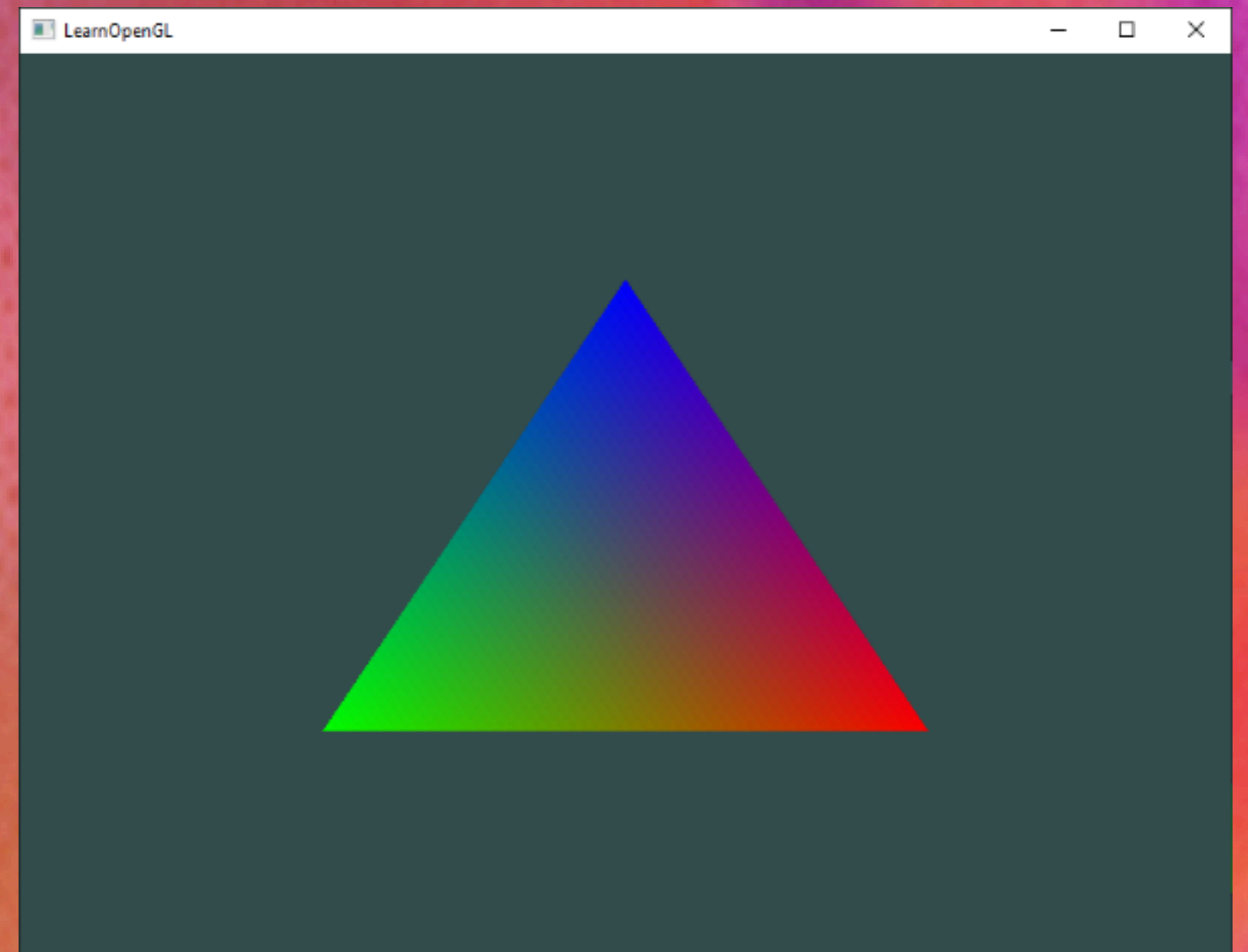
WHAT IS



- OpenGL is just a specification.
- Up to GPU manufacturers to implement OpenGL.
- Inherently cross-platform.
- Not that complicated!

'MODERN' OPENGGL VS. 'LEGACY' OPENGGL

- OpenGL first created in the 90's.
- GPU was less programmable than it is now.
- Core distinction is Shader Programs.

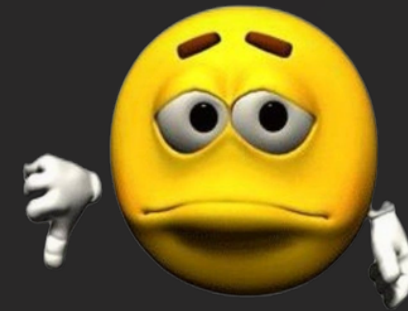


LIBRARIES

- OpenGL doesn't handle everything...
- **GLFW** – windowing and context library for OpenGL.
- **GLM** – Graphics math library.
- **stb_image** – header only image loader.

OPENGL LOADING LIBRARIES

```
// function pointer type  
typedef GLuint (*PFNGLCREATESHADERPROC)(GLenum type);
```

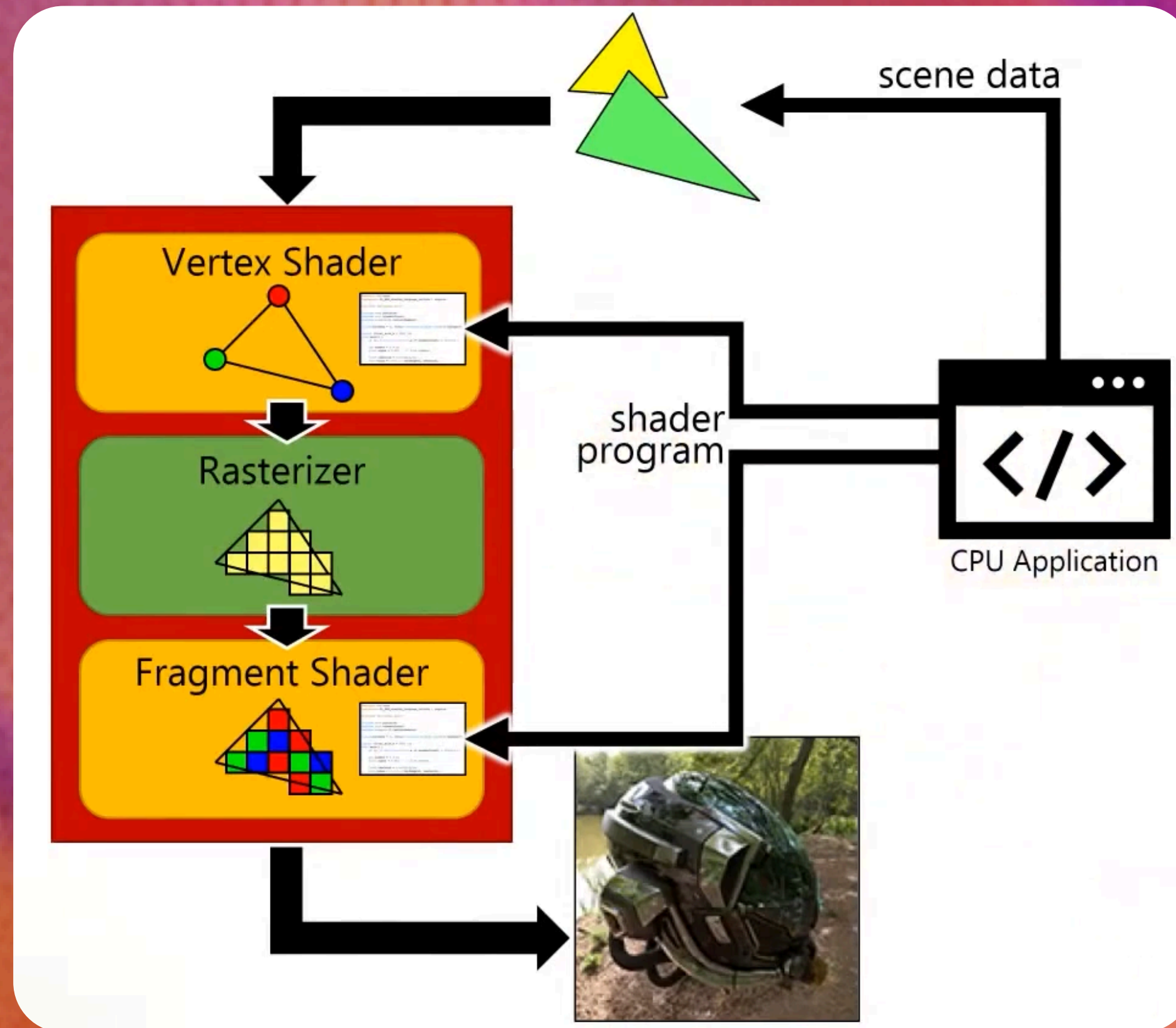


```
// ask the driver where it is  
PFNGLCREATESHADERPROC glCreateShader = (PFNGLCREATESHADERPROC)glfwGetProcAddress  
(procname: "glCreateShader");
```

```
// now you can call it  
GLuint shader = glCreateShader(type: GL_VERTEX_SHADER);
```



HIGH-LEVEL GPU PIPELINE



OPENGL IS A STATE MACHINE

```
// OpenGL is a state machine: you BIND, then ACT

// Buffers
glBindBuffer(GL_ARRAY_BUFFER, vbo);           // "I'm talking about vbo now"
glBufferData(GL_ARRAY_BUFFER, ...);           // this goes into vbo

glBindBuffer(GL_ARRAY_BUFFER, other_vbo);     // "now I'm talking about other_vbo"
glBufferData(GL_ARRAY_BUFFER, ...);           // this goes into other_vbo

// Textures
glBindTexture(GL_TEXTURE_2D, tex1);           // "I'm talking about tex1 now"
glTexParameteri(GL_TEXTURE_2D, ...);         // applies to tex1
glTexImage2D(GL_TEXTURE_2D, ...);            // uploads to tex1

// Shaders
glUseProgram(shaderA);                       // "I'm using shaderA now"
glUniform1f(loc, 0.5f);                      // sets uniform in shaderA
glDrawElements(...);                         // draws with shaderA

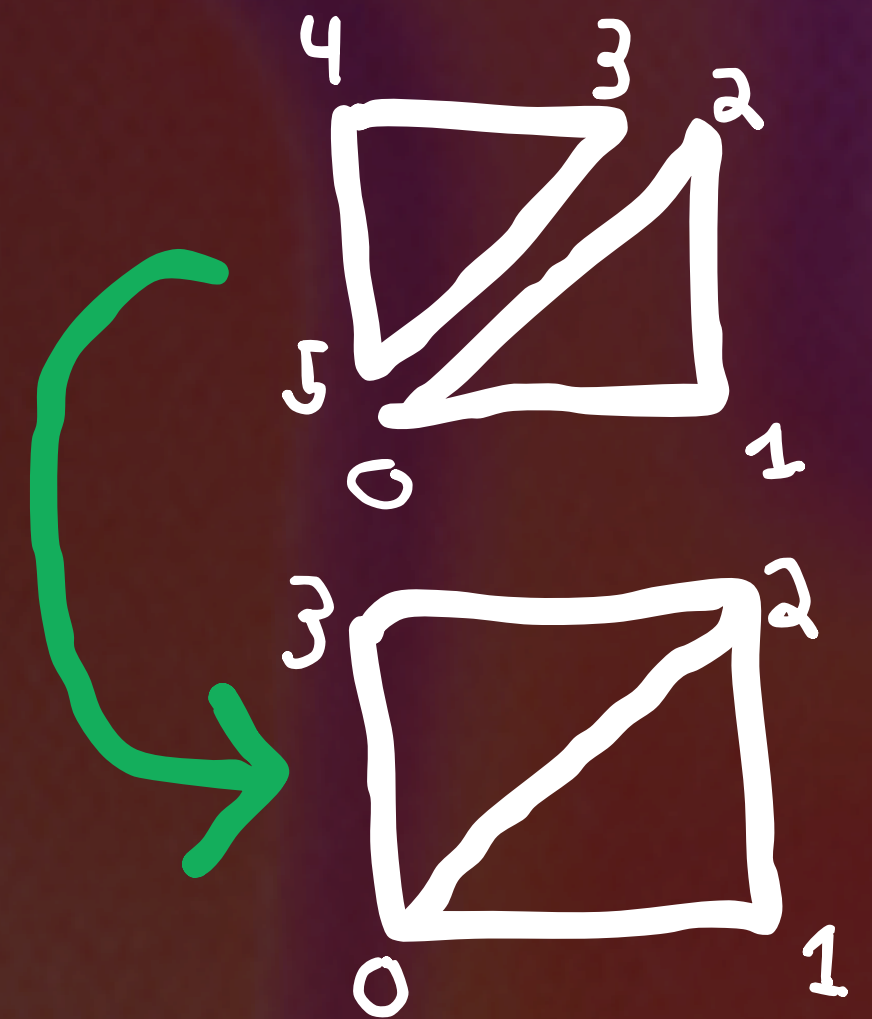
glUseProgram(shaderB);                       // "now I'm using shaderB"
glDrawElements(...);                         // draws with shaderB
```

VERTEX BUFFER OBJECT (VBO)

- What is a Vertex?
 - A bundle of attributes, typically for defining geometry.
- How do we define geometry?
 - A list of vertices.
- How do we send data to GPU?
 - VBOs!

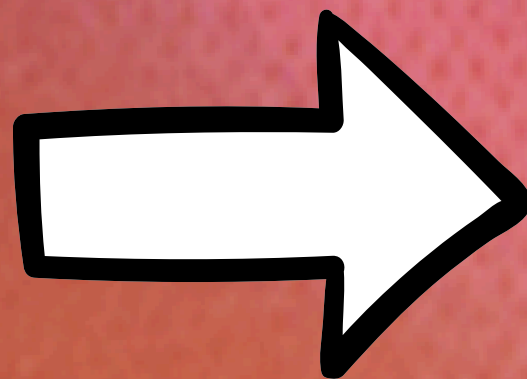
ELEMENT BUFFER OBJECT (EBO)

- OpenGL renders triangles as its primitive.
- Two triangles together make a quad.
 - Is there an optimization we can make?
- Define a buffer with index $\rightarrow$ vertex mappings (EBO) (aka Index Buffer Object or IBO).
- Only store unique vertices in VBO.



ELEMENT BUFFER OBJECT (EBO)

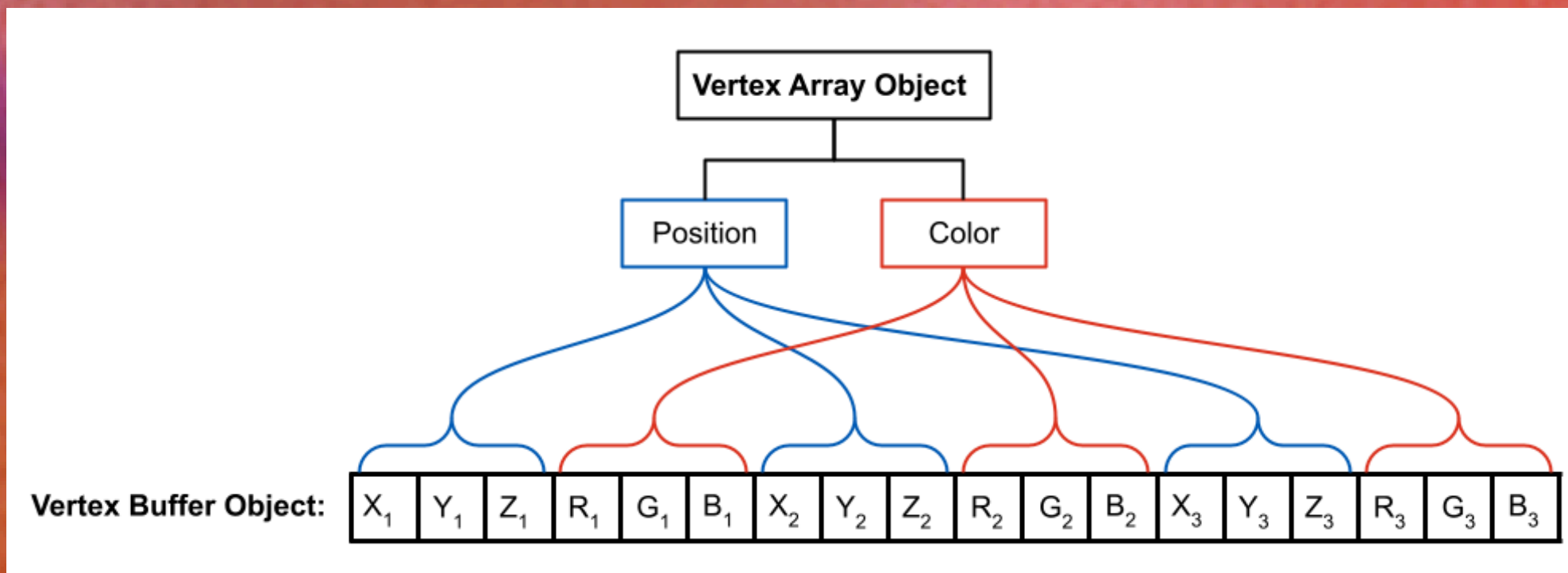
```
GLfloat quad_positions[] = {  
    // triangle one  
    -0.5f, -0.5f,  
    0.5f, -0.5f,  
    0.5f, 0.5f,  
  
    // triangle two  
    -0.5f, -0.5f,  
    0.5f, 0.5f,  
    -0.5f, 0.5f,  
};
```



```
GLfloat quad_positions[] = {  
    -0.5f, -0.5f,    // 0  
    0.5f, -0.5f,    // 1  
    0.5f, 0.5f,     // 2  
    -0.5f, 0.5f,    // 3  
};  
  
GLuint quad_indices[] = {  
    0, 1, 2,  
    2, 3, 0  
};
```


VERTEX ARRAY OBJECT (VAO)

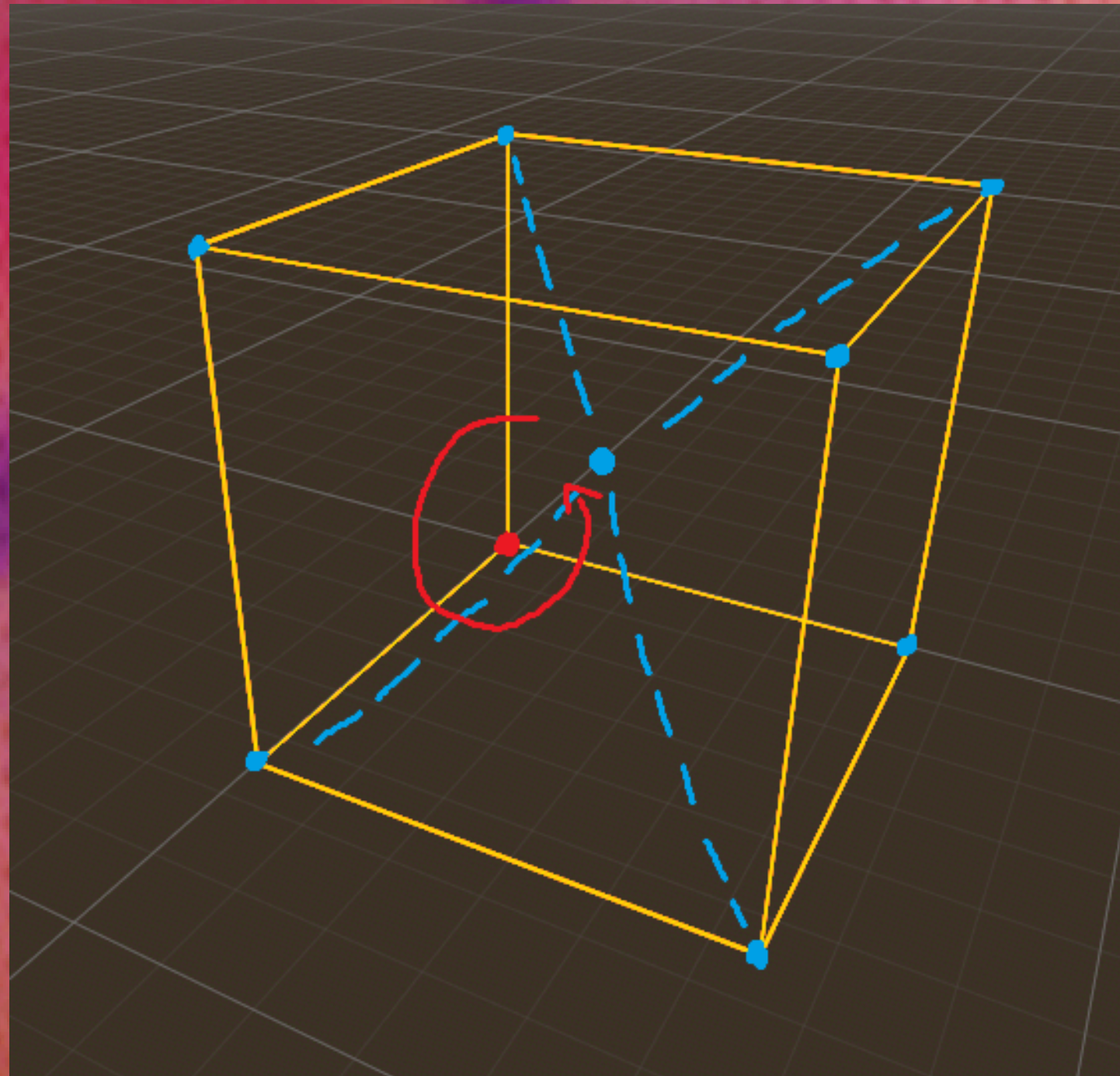
- How do we tell the GPU the types and layout of data?
 - `glVertexAttribPointer()` defines per-attribute size and layout for each vertex we send it.
 - BUT, this is stored in GLOBAL STATE.
- VAOs allow us to store state that defines a VBO and its layout.



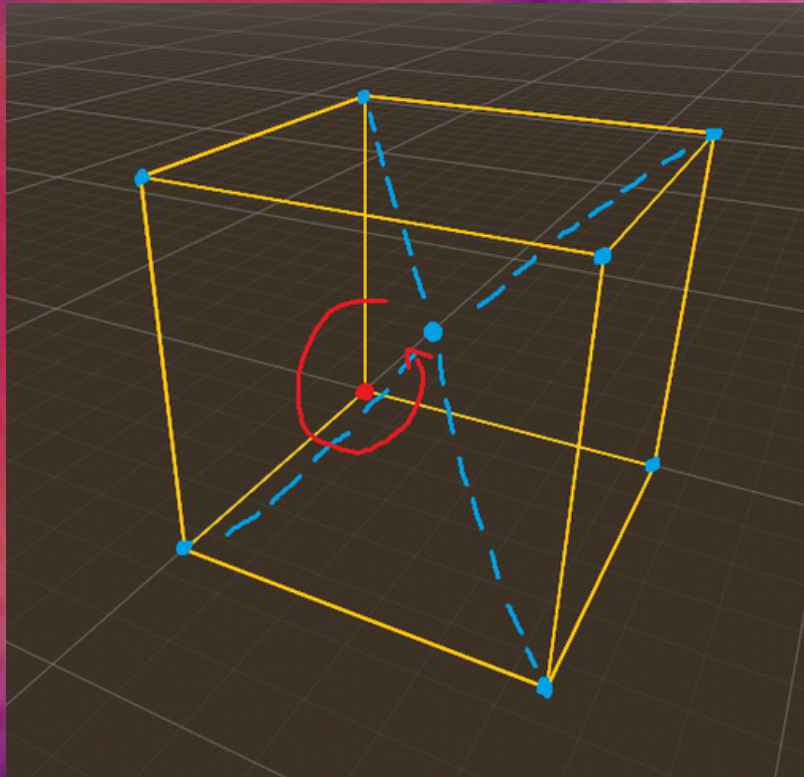
DEFINING CUBE GEOMETRY

- **Vertex Attributes:** (x, y, z, u, v) = position + texture coord.
- Cube is centered at the origin.
- **24 vertices** total (4 per face x 6 faces). Vertices aren't shared between faces so each face can map UVs independently.
- **36 indices** (2 triangles per face x 3 indices x 6 faces).
- CCW (counter-clockwise) winding order.
- Right-handed coordinate system.

vertex = ($\langle x, y, z \rangle$, $\langle u, v \rangle$)



- Transform @ cube center $\rightarrow$ all vertices
- Need $4 * 6$ vertices
(4 per face)
- Vertex buffer = all unique verts
(4 verts $\times$ 6 faces)
- Index buffer = triangles per face
(12 tris $\times$ 3 indexes)



- CCW Orientation in Index buffer

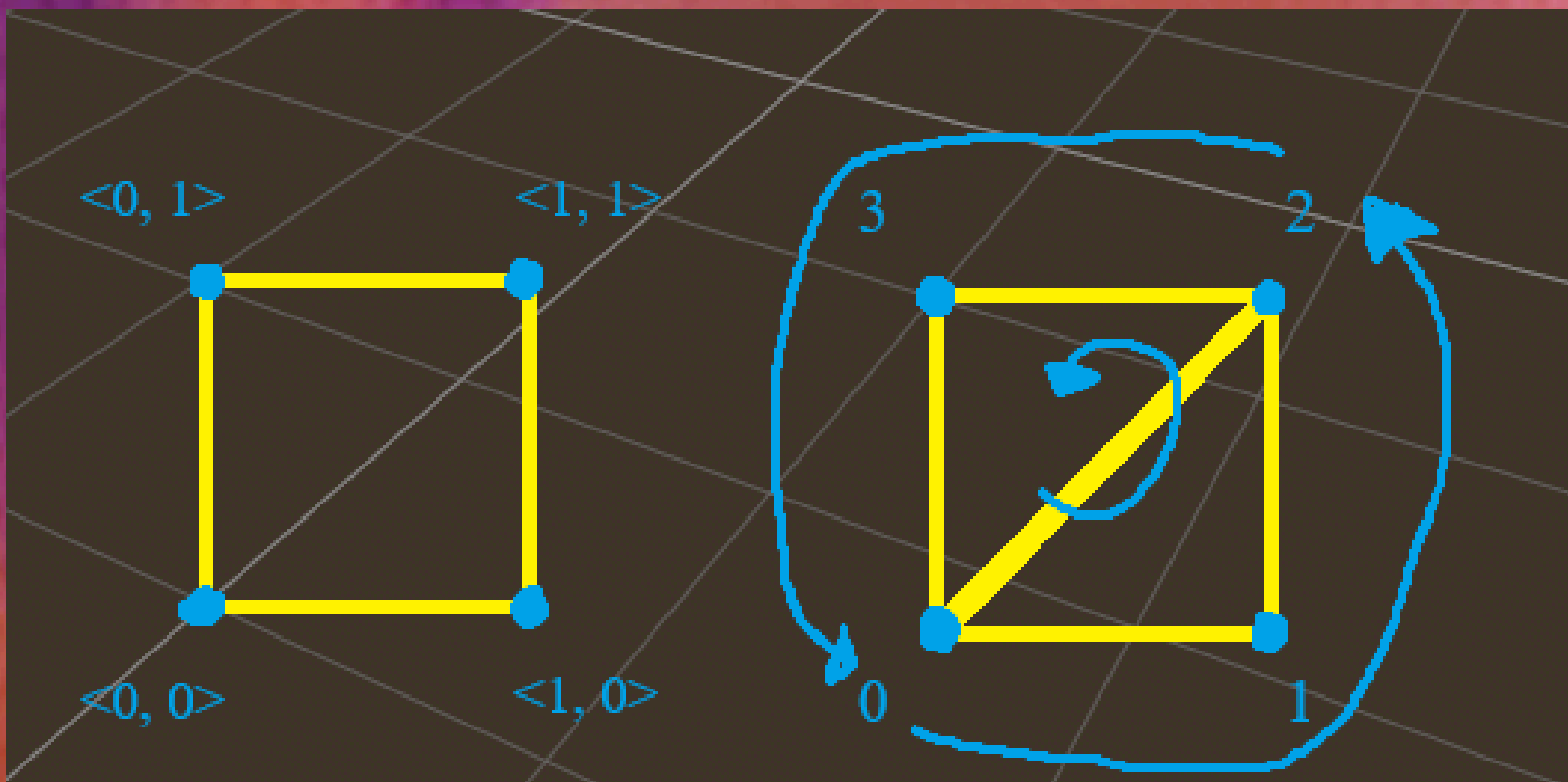
- Face ordering:

F, Ba, L, R, T, Bo

(Ba → back; Bo → bottom)

- Right-handed coordinate frame

- Winding order: CCW





Ok, coding time >:)

FURTHER READING

- learncpp.com - for ACTUALLY learning C++
- learnopengl.com - common reference for learning basic OpenGL
- scratchapixel.com - one of my favorites for all things CG



WHAT CAN I DO AFTER THIS WORKSHOP?

- Build an abstraction layer over specific OpenGL functionality
- Implement new features
 - Add more to scene
 - Lighting
 - Model Loading

CHECKOUT GPK SOCIALS AND DISCORD



THANKS FOR COMING!



Credit to Bill Quach for helping
with the slides and notes!