

COMP3320 Introduction to OpenGL

Alex Biddulph

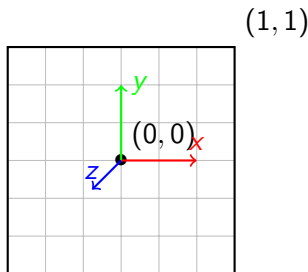
The University of Newcastle, Australia

Based on the work provided at www.learnopengl.com

Semester 2, 2019

OpenGL Coordinates

- ▶ OpenGL uses a right-handed coordinate system
- ▶ OpenGL uses normalised device coordinates



- ▶ OpenGL will map the normalised device coordinates to the viewport dimensions
 - ▶ $(-1, -1) \rightarrow (0, 0)$
 - ▶ $(1, 1) \rightarrow (800, 600)$

OpenGL Shaders

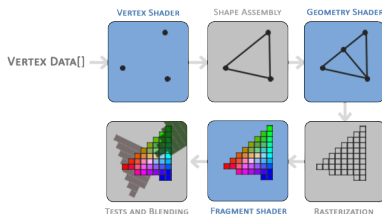


Figure: Graphics pipeline stages.

Image sourced from
learnopengl.com/Getting-started/Hello-Triangle

OpenGL Shaders

- ▶ Vertex Data: A list of 3D vertex coordinates and associated vertex attributes
- ▶ Vertex Shader: Transforms vertex coordinates from model space to clip space
- ▶ Shape Assembly: Assembles transformed vertices into a given primitive shape (e.g. triangles)
- ▶ Geometry Shader: Transforms shape geometry by emitting new vertices
- ▶ Rasterization: Maps primitives to pixel space and creates fragments
- ▶ Fragment Shader: Calculates the final colour for a pixel
- ▶ Tests and Blending: Performs depth testing and alpha blending

OpenGL Shaders

- ▶ Written in OpenGL Shader Language (GLSL)

- ▶ A simple vertex shader

```
#version 330 core
```

```
layout (location = 0) in vec3 aPosition;
```

```
void main() { gl_Position = vec4(aPosition, 1.0); }
```

- ▶ A simple fragment shader

```
#version 330 core
```

```
out vec4 FragColour;
```

```
void main() {
```

```
    FragColour = vec4(1.0f, 0.5f, 0.2f, 1.0f);
```

```
}
```

- ▶ Create a shader using `glCreateShader`
- ▶ Attach shader source code using `glShaderSource`
- ▶ Compile a shader using `glCompileShader`

OpenGL Shader Programs

- ▶ Consists of multiple OpenGL shaders
 - ▶ If you have a vertex shader, you must have a fragment shader
 - ▶ If you have a fragment shader, you must have a vertex shader
 - ▶ Geometry shader is optional
- ▶ Create a shader program using `glCreateProgram`
- ▶ Attach shaders to the program using `glAttachShader`
- ▶ Link attached shaders together into the final program using `glLinkProgram`
- ▶ Make a shader program active by using `glUseProgram`

Vertex Buffer Objects

- ▶ Stores vertex data in GPU memory
- ▶ To create a vertex buffer use `glGenBuffers`
- ▶ Program could consist of multiple different vertex buffers. To make a vertex buffer active use `glBindBuffer`
- ▶ Vertex data is defined as a list of 3D coordinates (and optionally vertex attributes)

```
float vertices[] = {-0.5f, -0.5f, 0.0f,  
                    0.5f, -0.5f, 0.0f,  
                    0.0f, 0.5f, 0.0f };
```

- ▶ To copy vertex data to GPU memory use `glBufferData`

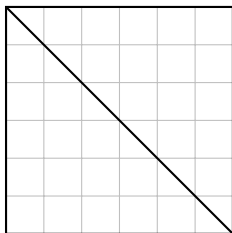
Vertex Array Objects

- ▶ Manages vertex buffer objects and vertex attributes
- ▶ To create a vertex array use `glGenVertexArrays`
- ▶ Program could consist of multiple different vertex arrays. To make a vertex array active use `glBindVertexArray`
- ▶ Bind and configure vertex buffer objects after binding the vertex array
- ▶ To configure and enable a vertex attribute use `glVertexAttribPointer` followed by `glEnableVertexAttribArray`
- ▶ To render a vertex array use `glBindVertexArray` followed by `glDrawArrays` or another OpenGL drawing command

Element Buffer Objects

- ▶ Stores indexes into the list of vertex data
- ▶ Makes it easy to draw shapes which share vertices
 - ▶ An easy example is a square, which contains 2 triangles

$(-0.5, 0.5)$ $(0.5, 0.5)$



$(-0.5, -0.5)$ $(0.5, -0.5)$

- ▶ The naive way to draw this square would be to draw two triangles by specifying six vertices. Two of the vertices would be repeated
- ▶ The better way would be to list the four vertices in the square and use an element buffer object

Element Buffer Objects

- ▶ To create an element buffer use `glGenBuffers`
- ▶ Program could consist of multiple different element buffers.
To make an element buffer active use `glBindBuffer`
- ▶ Element data is defined as a list of integer indices

```
float vertices[] = {-0.5f, -0.5f, 0.0f,
                    0.5f, -0.5f, 0.0f,
                    0.5f,  0.5f, 0.0f,
                    -0.5f,  0.5f, 0.0f };
unsigned int indices[] = {0, 1, 3,
                          1, 2, 3 };
```

- ▶ To copy index data to GPU memory use `glBufferData`
- ▶ To draw an element buffer use `glDrawElements`
- ▶ Vertex arrays will also track element buffers