OpenGL & LWJGL Notes

VAO: (vertex array objects)

- Store data for 3D models
- The data is stored in attribute lists. There are about ~16 attribute lists in a VAO.
- usually store positions, colours, vectors
- Some data is stored as VBOs

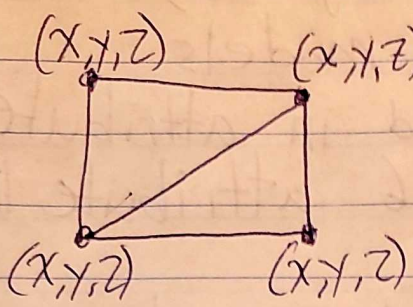
VBO: (Vertex Buffer Object)

- Data is stored as an array list of numbers.
- Data can be anything. Its just a very general storage method.

Vertex: 3 value data variable

- vertex data goes into VBOs and VBOs go into VAOs

-3D Models:

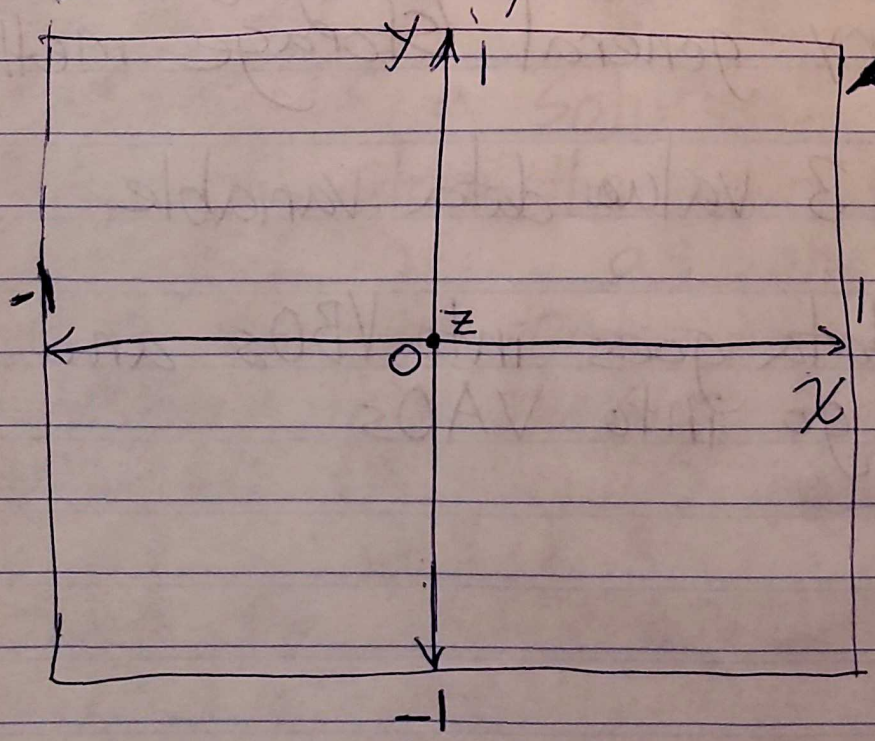


- All models are represented by triangles.

- All these vectors are stored in a single VBO.

- Each VAO has an ~~idea~~ ID that openGL keeps to render them.

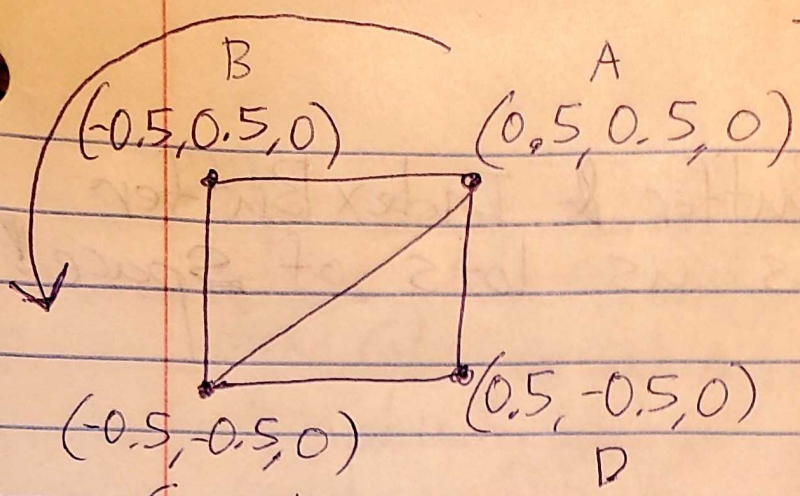
Coordinate System:



This is the entire window

- The Z axis goes towards you (the dot at the origin)

Basic Vertices



- OpenGL defines vertices in a counterclockwise order.

(0.5,0.5,0,-0.5,0.5,0,-0.5,-0.5,0,0.5,0.5,0)
A B C A

- This becomes very, very long with huge models. Many lines (edges) get repeated such as A to C because triangles can share vertices and edges.

- There is a solution!
- we will store the four vertices (A, B, C, D) as one buffer (Vertex Buffer) and the the order in which they connect into another buffer. (Index Buffer
- So If (A, B, C, D)
0 1 2 3
then the index buffer is
(0,1,2,2,3,0)
- which translated back is
A→B→C, C→D→A

- This vertexBuffer & IndexBuffer system saves us lots of space!

Shaders

- usually for graphics you would use a function. you would have a bunch of functions like addShadow(vector position).

- This works but is very limited.
- We are going to do something else.

- we will be using files called open GL shader programs. These .txt files are written in a special shader language understood by openGL. These programs allow much more direct communication with our GPU then the functions described above.

GLSL = GL Shader Language

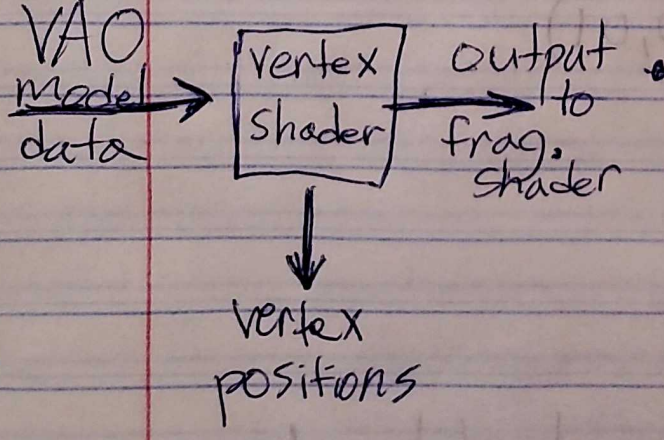
- Our code will interpret these GLSL files and feed them into OpenGL, creating our awesome graphics.

Shader Programs:

- There are two types of shaders, vertex shaders and fragment shaders.

Vertex Shader: this executes one time for each vertex per frame.
- This takes in our VAOs as input. (all our ~~verte~~ model data)

- This shader takes all the vertex positions out of the VAO and



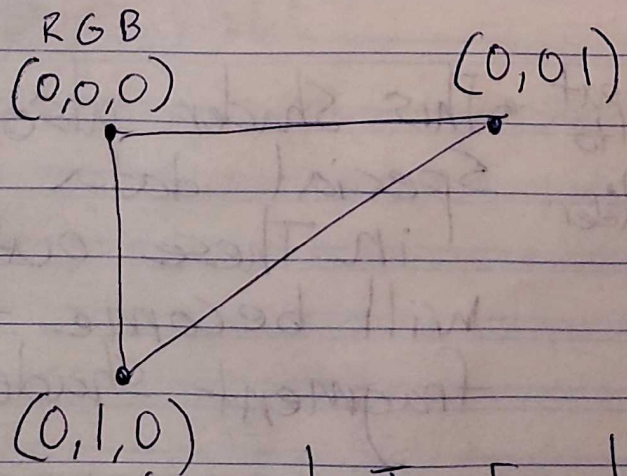
- This shader also outputs special data we program in. These outputs will become the fragment shader's inputs

- For example, the vertex shader could take in VAO data and assign a colour to each vertex. This new color data would be sent to the fragment shader.

Fragment Shader: This executes once for every pixel per model per frame.

- It takes as input the vertex shader's output. It processes this input and then assigns color values for each pixel.

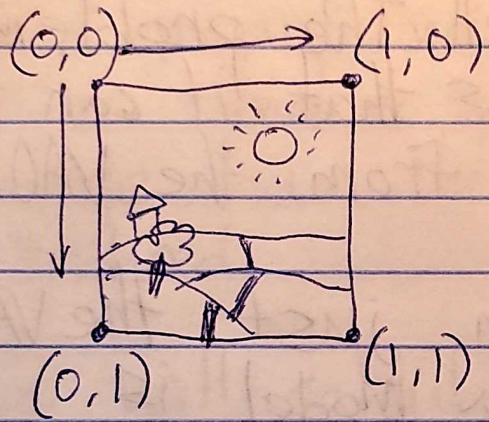
- In our example, the fragment shader takes in Red, Green, Blue values for each vertex and then calculates the colour for all the space between the vertices.



- This is a LOT of calculations! This is why GPUs are awesome.

- All textures must be .PNG files and they must be $2^n \times 2^n$ pixels, so 2x2, 4x4, 8x8, 16x16, 32x32, etc.

Texture Coordinates:



- Not (x,y), it's (u,v) because it's a special coordinate system based on vectors.

- Each vertex will now have a spacial coordinate and a texture coordinate

- These new texture coordinates will be stored in a new VBO in our VAO.