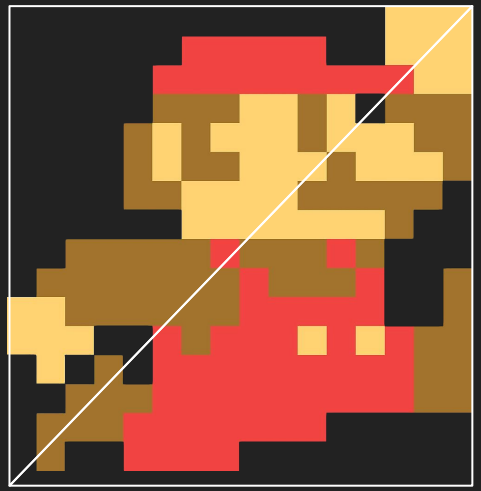
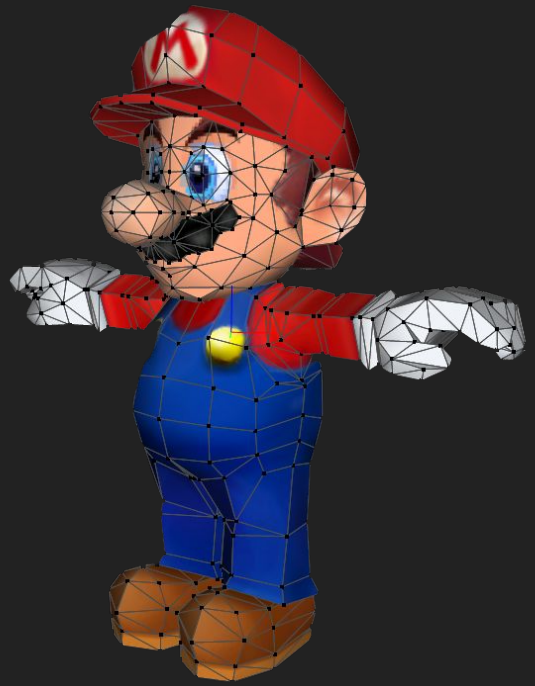
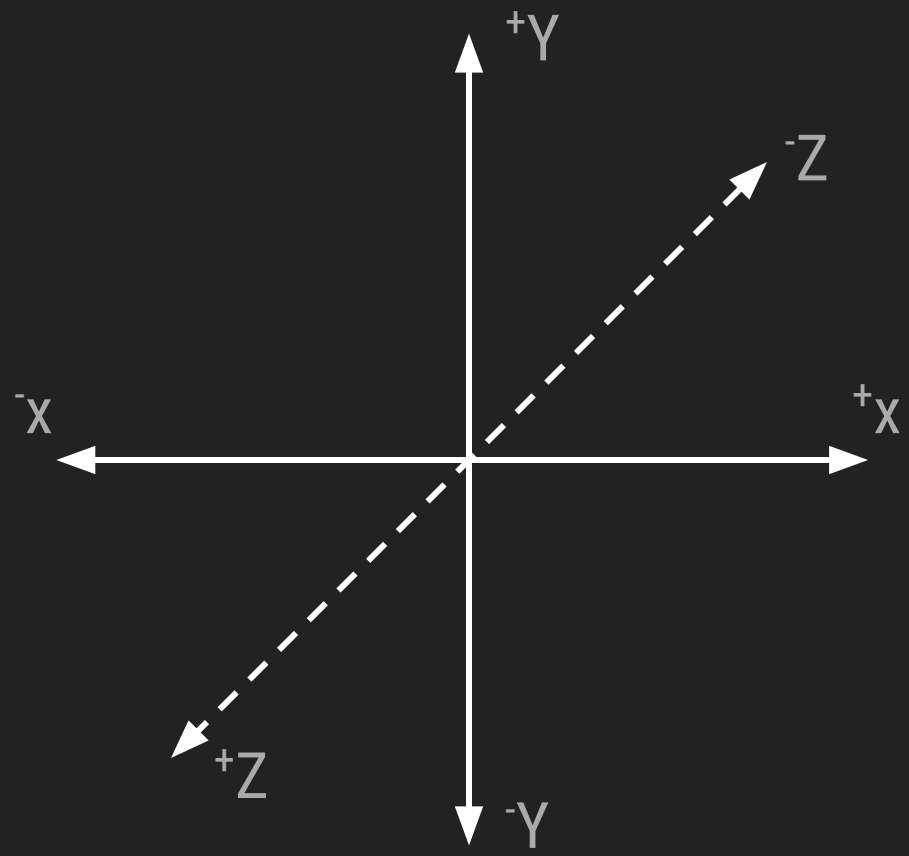
Introduction to 3D

2D



3D

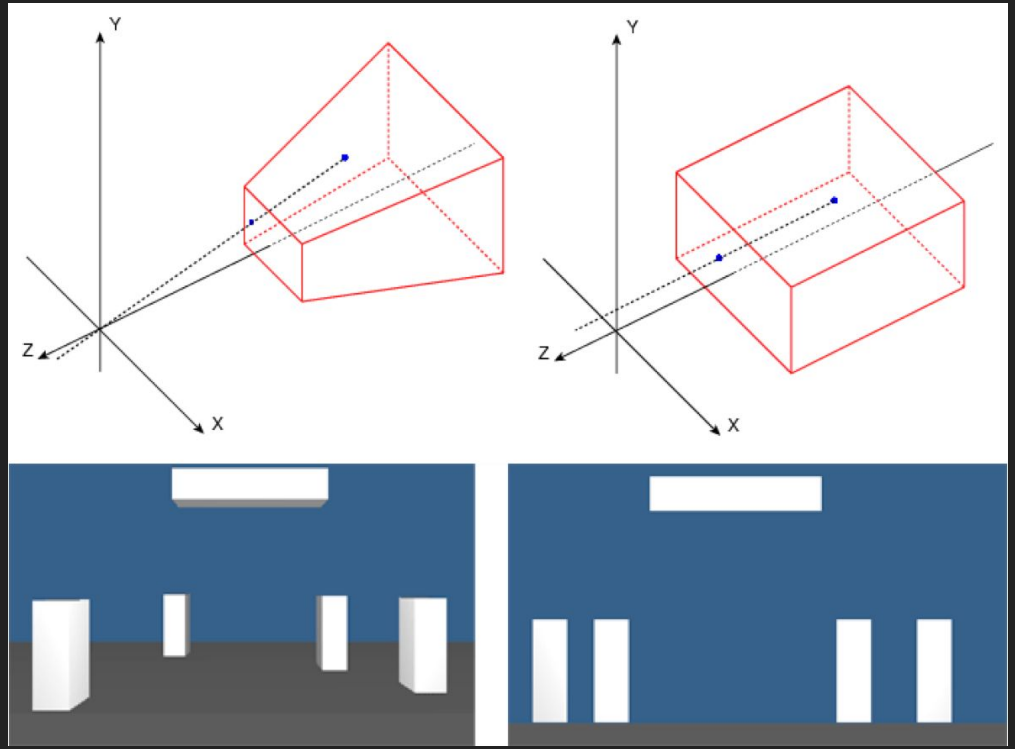



+Y
-Z
-X
+X
+Z
-Y

Switching to 3D

Perspective Projection
3 Coordinates (X, Y, Z) for Vertices
More Triangles (3D Models)
Moving in 3 Dimensions
Collision Detection

Perspective vs. Orthographic



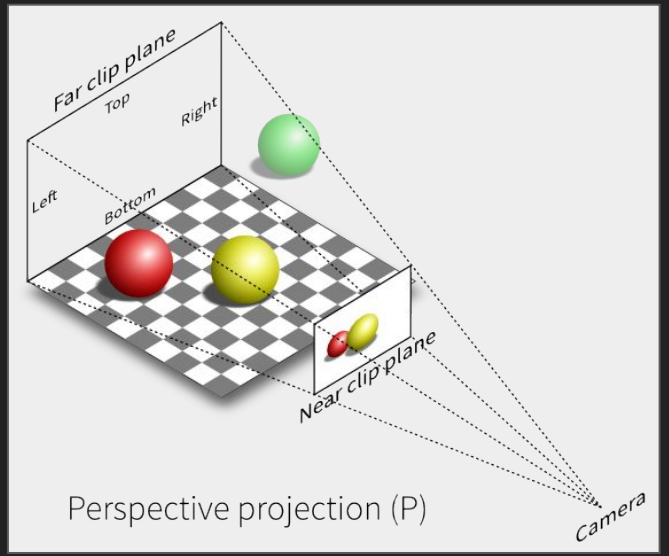
Perspective vs. Orthographic

We have been using Orthographic this entire time:

```
projectionMatrix = glm::ortho(-5.0f, 5.0f, -3.75f, 3.75f, -1.0f, 1.0f);
program.SetProjectionMatrix(projectionMatrix);
```

Perspective Projection

```
// Function definition
glm::mat4 glm::perspective(
    float fov,
    float aspectRatio,
    float nearPlane,
    float farPlane
);
```

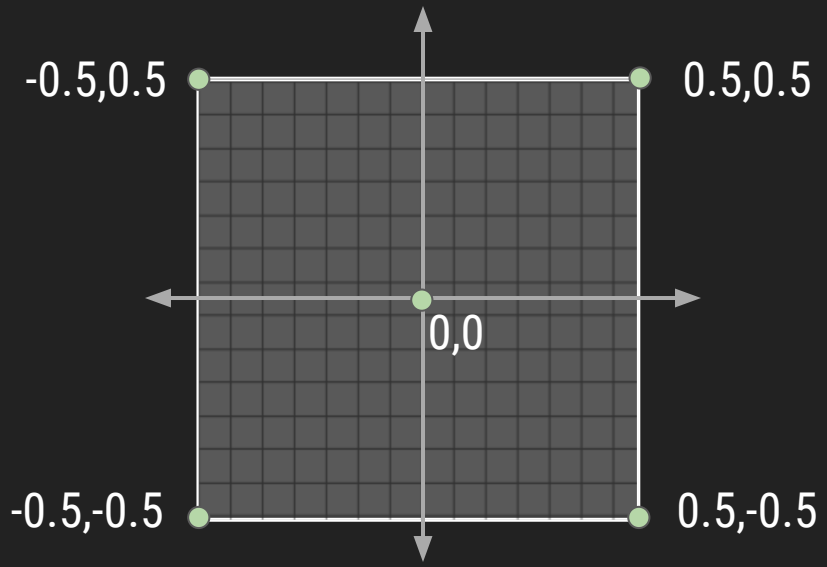



```
// Don’t forget to convert degrees to radians!
projectionMatrix = glm::perspective(glm::radians(45.0f), 1.777f, 0.1f, 100.0f);
```

Drawing a Cube

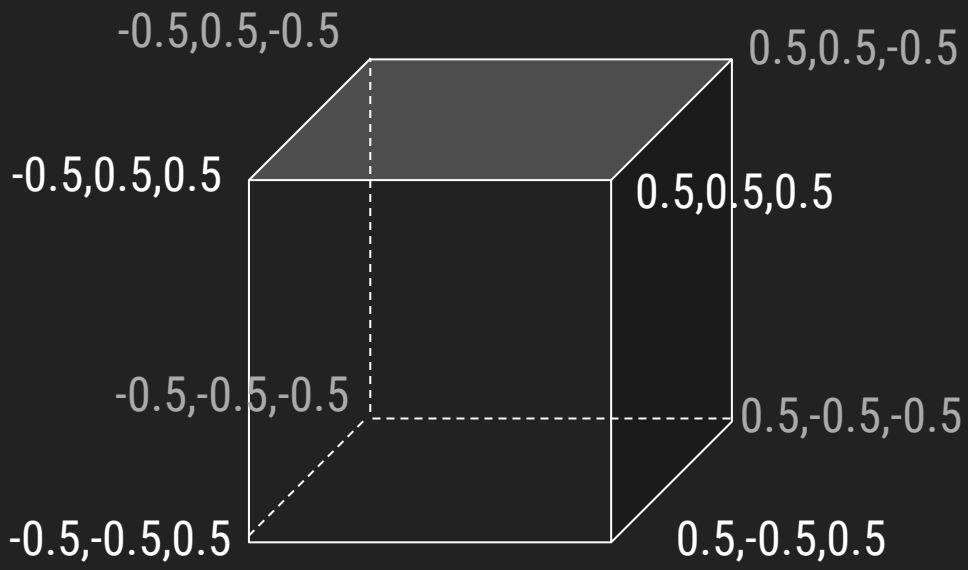
(and other objects)

Originally we used 2 values (X, Y) for each vertex.



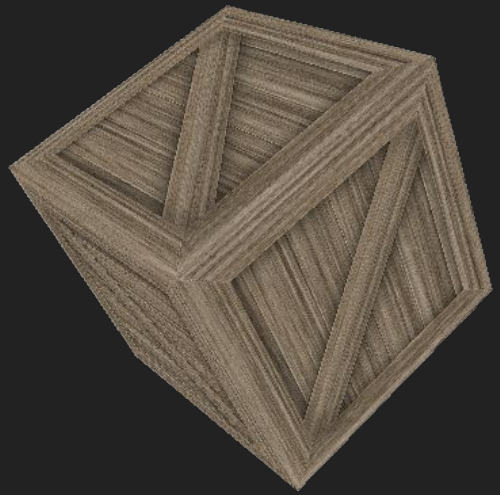

```
float vertices[] = {
    -0.5, -0.5,
     0.5, -0.5,
     0.5,  0.5,
    -0.5, -0.5,
     0.5,  0.5,
    -0.5,  0.5
};
```

Now we are going to use 3 (X, Y, Z).



```
float cubeVertices[] = {
    -0.5,  0.5, -0.5, -0.5,  0.5,  0.5,  0.5,  0.5,  0.5,
    -0.5,  0.5, -0.5,  0.5,  0.5,  0.5,  0.5,  0.5, -0.5,

     0.5, -0.5, -0.5,  0.5, -0.5,  0.5, -0.5, -0.5,  0.5,
     0.5, -0.5, -0.5, -0.5, -0.5,  0.5, -0.5, -0.5, -0.5,

    -0.5,  0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,  0.5,
    -0.5,  0.5, -0.5, -0.5, -0.5,  0.5, -0.5,  0.5,  0.5,

     0.5,  0.5,  0.5,  0.5, -0.5,  0.5,  0.5, -0.5, -0.5,
     0.5,  0.5,  0.5,  0.5, -0.5, -0.5,  0.5,  0.5, -0.5,

    -0.5,  0.5,  0.5, -0.5, -0.5,  0.5,  0.5, -0.5,  0.5,
    -0.5,  0.5,  0.5,  0.5, -0.5,  0.5,  0.5,  0.5,  0.5,

     0.5,  0.5, -0.5,  0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
     0.5,  0.5, -0.5, -0.5, -0.5, -0.5, -0.5,  0.5, -0.5
};
```

Texture Coordinates are still U, V
(we have more vertices so the list is longer)

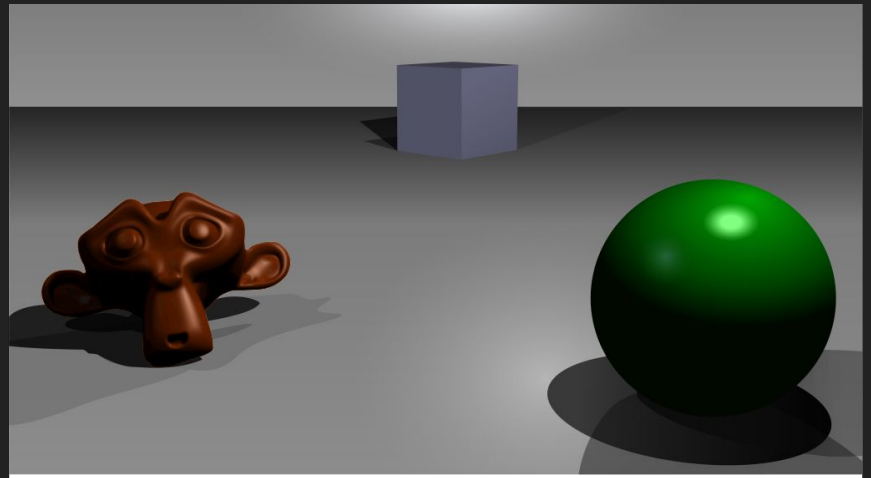


https://opengameart.org/content/3-crate-textures-w-bump-normal

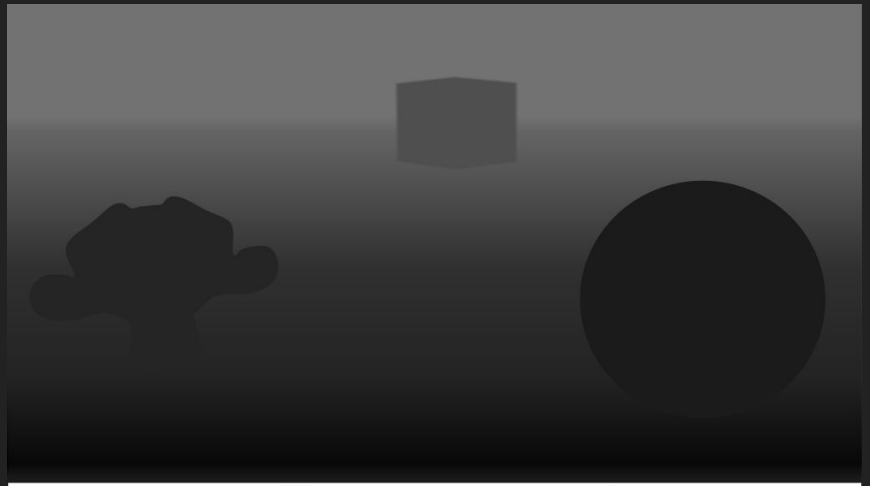
```
float cubeTexCoords[] = {
    0.0f, 0.0f, 0.0f, 1.0f, 1.0f, 1.0f,
    0.0f, 0.0f, 1.0f, 1.0f, 1.0f, 0.0f,

    0.0f, 0.0f, 0.0f, 1.0f, 1.0f, 1.0f,
    0.0f, 0.0f, 1.0f, 1.0f, 1.0f, 0.0f,

    0.0f, 0.0f, 0.0f, 1.0f, 1.0f, 1.0f,
    0.0f, 0.0f, 1.0f, 1.0f, 1.0f, 0.0f,

    0.0f, 0.0f, 0.0f, 1.0f, 1.0f, 1.0f,
    0.0f, 0.0f, 1.0f, 1.0f, 1.0f, 0.0f,

    0.0f, 0.0f, 0.0f, 1.0f, 1.0f, 1.0f,
    0.0f, 0.0f, 1.0f, 1.0f, 1.0f, 0.0f,

    0.0f, 0.0f, 0.0f, 1.0f, 1.0f, 1.0f,
    0.0f, 0.0f, 1.0f, 1.0f, 1.0f, 0.0f
};
```

Z-Buffer

(Depth Buffer)



A simple three-dimensional scene



Z-buffer representation

https://en.wikipedia.org/wiki/Z-buffering

OpenGL does not do this by default, we need to enable it.

Enable comparisons to the buffer.

```
glEnable(GL_DEPTH_TEST);
```

Enable writing to the depth buffer.

```
glDepthMask(GL_TRUE);
```

The depth function is how an incoming pixel is compared against one that is already there. GL_LEQUAL means “use the incoming pixel if its distance from the camera is less than or equal to what is there already.”

Set the depth function.

```
glDepthFunc(GL_LEQUAL);
```

Remember how we clear the screen at the beginning of each frame (inside of render)?

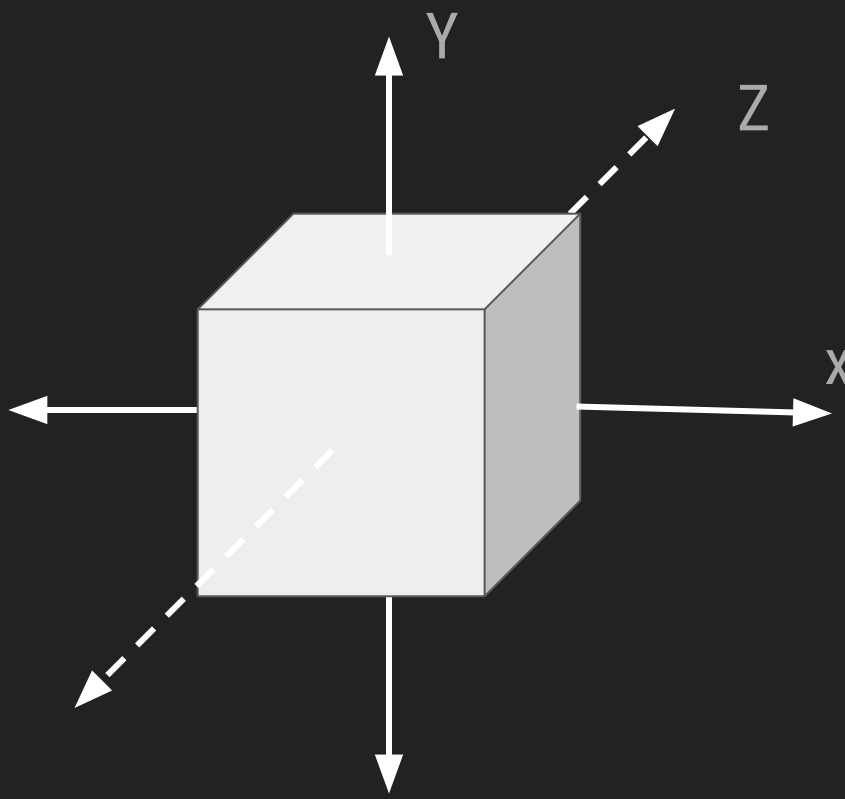
We also need to clear the depth buffer!

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

We'll need to make some updates to Entity.cpp and Entity.h

```
// Update Entity.h
float *vertices;
float *texCoords;
int numVertices;

// Update Entity.cpp
// Inside of Render, remove vertices and texCoords.
// Update the following lines below

glVertexAttribPointer(program->positionAttribute,
                      3, GL_FLOAT, false, 0, vertices);

glDrawArrays(GL_TRIANGLES, 0, numVertices);
```

Rotating in 3 Dimensions

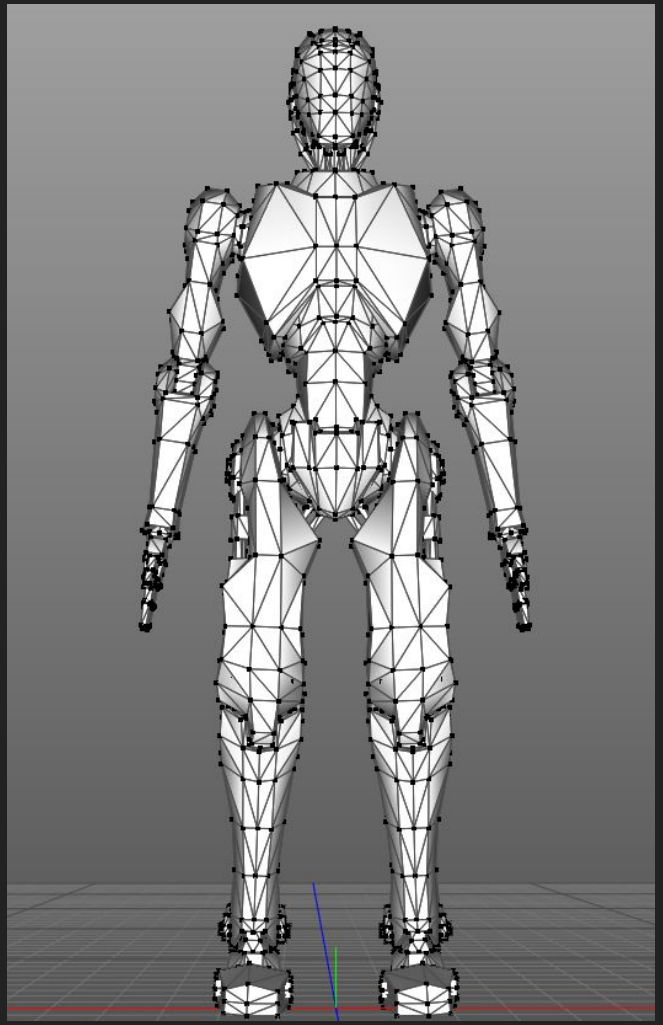

Y
Z
X

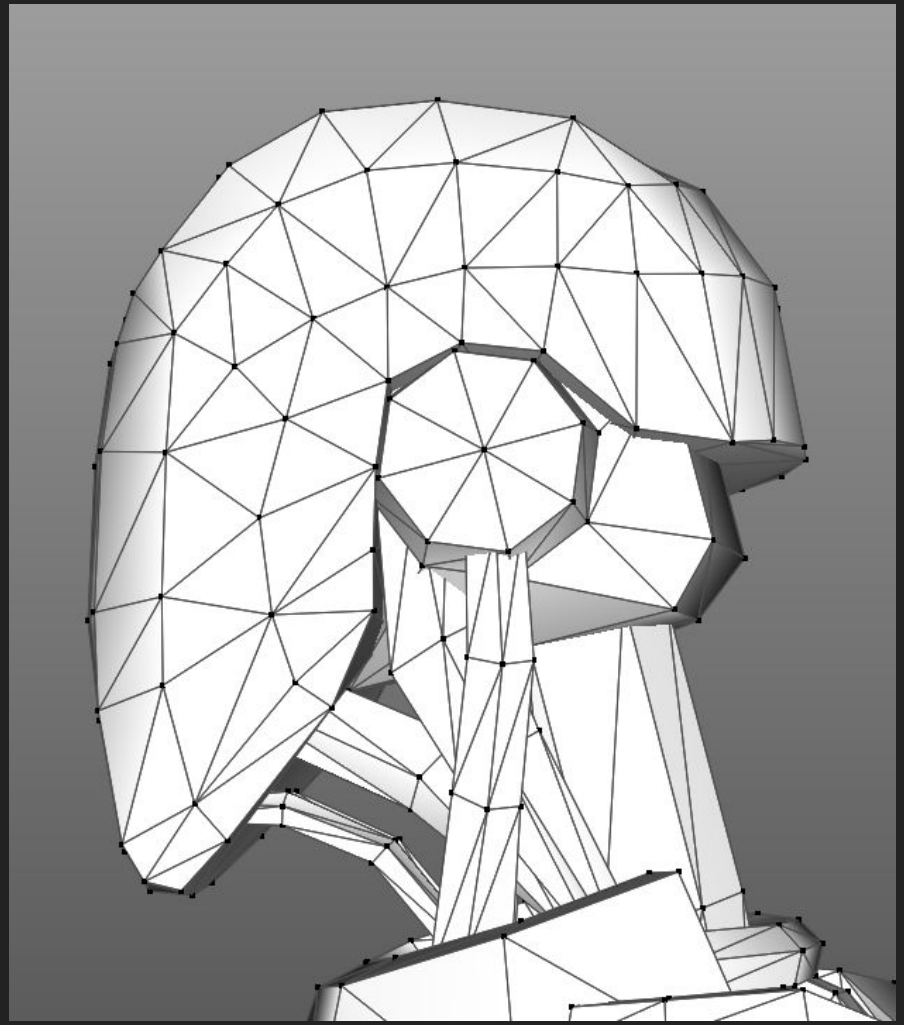
Let’s Code!

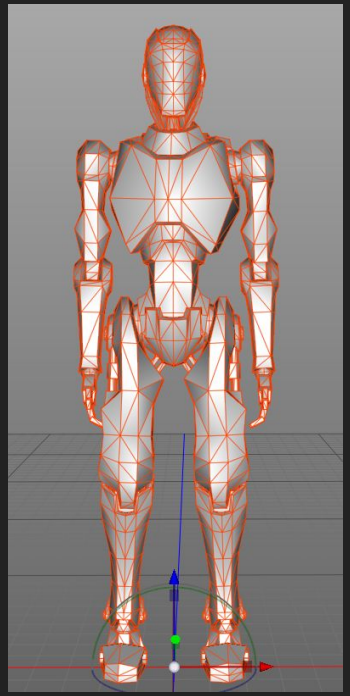
We’re going to put this all together.

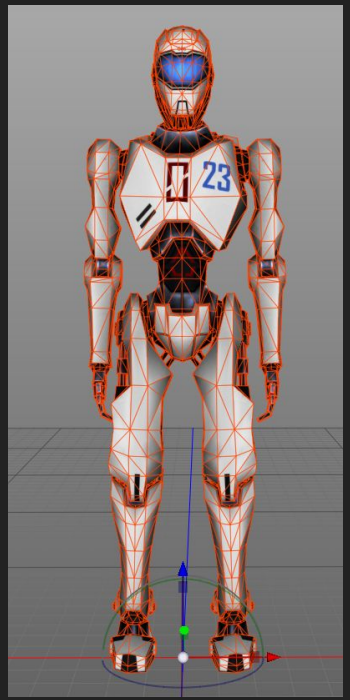
In github (inside the examples folder)
there is a file called
SDL3DStarter.zip

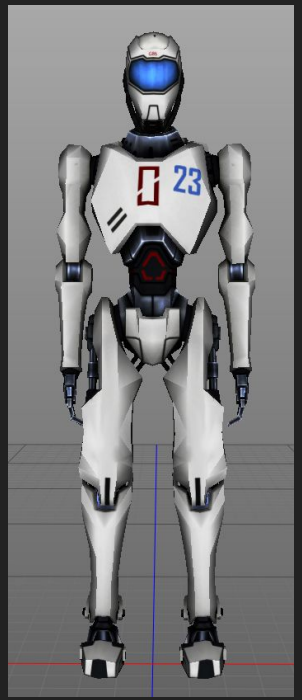
3D Models











There are various file formats for storing the list of vertices (triangles), texture coordinates (and more) that make up a 3D model.

OBJ is a very simple file format to read. Most 3D programs can export to this format.

Let’s look at cube.obj in the Assets folder in the course github.

https://github.com/carmineguida/CS3113/blob/master/Assets/3D%20Models/cube.obj

OBJ File Structure

(each line starts with a descriptor)

```
# This is a comment

# mtllib - A material
mtllib cube.mtl

# o - The name of an object
o Cube
```

OBJ File Structure

(each line starts with a descriptor)

```
# v - A vertex
v 1.000000 -1.000000 -1.000000

# vt - A texture coordinate
vt 1.000000 0.333333

# vn - A vertex normal
vn 0.000000 -1.000000 0.000000
```

OBJ File Structure

(each line starts with a descriptor)

```
# f - A face (they are in groups of 3 because it is a triangle)
# These are indices into the list.
# vertex / texture coordinate / vertex normal

f 2/1/1 3/2/1 4/3/1
```

Loading an OBJ File

Mesh.cpp and Mesh.h

Let’s Code!

Get the 3D assets from GitHub

Update Entity.h and Entity.cpp to use Meshes.
Update main.cpp to load meshes.
Get Mario and Pikachu on the screen.
Have the spaceship fly away!