Project Design Decisions

As an employee of Triangle & Cube Studios, a client had provided a 2D image that they wanted us to use to create a 3D version. This image required the use of basic shapes to create each object in the image, as well as texturing and lighting. Since the client would like this project to be done in a 3D environment, added controls for camera movement were also necessary to see the entire scene from various positions and angles.

The scene was a picture of a round wooden table, with two tankards on their sides, three plates, a candle and four small, black pyramid shaped objects (possibly intended to be “nails” holding the table together?). The tankards and plates were round cylindrical shapes, so a function was created to create different sized cylinders that would allow for different radius, height, and “steps”, which is basically how many sides the cylinder would have. To make a rounder cylinder, more sides are necessary.

Looking at the tankards, they both include a handle. To make a basic handle on each one, multiple cubes can be created that “join” at a specific point, and the outer sides reach onto the tankards to create a finished object. One of the tankards in the given image is more intricate, but I retained a basic look, using two cubes for each handle.

The “nails” are black pyramid shaped objects. This is relatively simple as we have experience creating pyramids throughout our course, and a function was created that would use a mesh of position, texture, and normal coordinate points.

Navigation in this 3D scene was necessary to be able to view all the objects at different angles and positions. Using keyboard input, directional navigation can be observed as: W = forward, S = backward, A = left, D = right, E = up, Q = down. There are additional controls that allow the user to do other things, including pressing the P key to change the view between orthogonal and perspective. The keys 1, 2, 3, 4 are used to change the texture wrapping between repeat, mirrored repeat, clamp to edge, and clamp to border, respectively. The right and left bracket keys also increase or decrease the scale.

While development of this project included a great deal of functionality used throughout the course (UCreateShaderProgram, UInitialize, UProcessInput, etc.), there are many new, custom functions developed to modularize and organize the code. There are four different shader programs, one for the plane (table), one for displaying the lamp light sources, one for the candle (since this object doesn’t seem to be nearly as shiny or reflective as anything else in the scene) ,and one for all other objects. Adding additional objects, colors, etc. is easily done in the main fragment shader, and it is relatively organized to continue a consistent flow. There is also a function to render frames. Enabling different shader programs can be done here, to create various objects and lamp sources.

Since we have objects such as cubes, cylinders, pyramids, lamps, and a plane, it is necessary to have different, custom functions for each shape type, that can be reused as often as we need. Our renderCube function uses data in a mesh function, binds a texture to a textureUnit (both included as arguments in the call to renderCube). If we look at the function that includes the mesh, there are two arrays for vertices and indices; the vertices include position, texture, and normal (for lighting) coordinates, and the indices tell the program which vertices are used to create cubes. The renderPyramid is very similar but uses a different mesh in the function UCreateMesh2. Two more functions, renderCylinder and renderCircPlane, use various arguments and some slightly complicated math to figure out vertices for coords, texture coords and normal. Finally, renderLamp is used to create lamp objects. This uses an array of position coordinates to create a basic square lamp, and in the render function, uniform variables for lamp position and color are added to change these attributes.

All these functions to render objects use vertex array objects and vertex buffer objects to store and organize their vertex data. Since buffers can be used to store data, we assign different buffers for position, texture, and normal coordinates. These are used in our different vertex shaders to obtain the data so that we can apply textures and lighting correctly.

The functions to render different objects are reusable as each one simply creates an instance of an object. Making a call to these functions, such as a cylinder, requires arguments for things such as height, radius, steps, texture, etc., so someone viewing the code could easy add additional objects with more calls to that specific function in the renderFrame function, based on their 3D scene requirements. While it could be more organized, I believe the functions make sense, are easy to read and understand, and another user could add their own objects, functions, etc. in the future.