CS-330 Michael Linsenbigler

Final Project 4/13/21

The objects I selected for the scene were chosen because I thought they represented unique shapes that would be fun to create in 3D space using OpenGL. The Xbox, which was created using a modified cube, was the base object for the scene and the other objects were created and placed in relation to it. Initially, I struggled with creating a cylinder shape to use for the top vent portion of the Xbox, so I substituted a pyramid to represent that section of the object. Later during the project’s development, I was able to create a sphere and place it properly as the Xbox vent, but I wasn’t able to apply the texture in a way that satified my expectations of the shape and object. I decided to keep the original shape of a pyramid and add the texture for the Xbox vent and am happy with the result. I also chose to rearrange the speaker to the opposite side of the scene because I felt the left side was fairly crowded with it placed in the same location as my original picture. I like the final placement of the objects and also prefer the lighting reflections and shadows more in my submitted project than in the picture I took at the beginning. I created the scene initially, so I felt I had the ability to alter it based on the experience I gained throughout the course and also based on the results I found while testing different parameters of location, shape, lighting, and texture while coding the scene. If I could go back, I would certainly lay out the objects in this way because I feel they are more appealing to the eye in their final location and form.

I wanted the scene to be viewed from the appropriate perspective angle when it opens so I set it on a 37 degree angle to start. The camera can be controlled by the user by pressing W, A, S, D, Q, E, P, and also by moving the mouse pointer and scrolling the mouse wheel. W, S, A, and D move the camera forward, backward, left and right respectively. Pressing the Q and E keys moves the camera up or down. The P key changes between a 3D (perspective) view and a 2D (orthographic) view. The user can swap between the two views to see the same objects from the same camera positions in both a 2D and a 3D environment. The mouse pointer will conrol where the camera is facing and acts as a “look around” function. The camera speed is set to 2.5 by default but can be controlled by moving the scroll wheel up to a maximum of 25 or down to a minimum speed of 0.5. Changing the camera speed will alter how fast it moves in any direction the user chooses. Using these input methods, the user can move to any point in the 3D space and view the scene from any angle they desire.

Many of the actions and processes in the project are handled by modular code functions and callbacks. This is important because it allows a function to be created once and then called back to multiple times throughout the program for process handling and returns. It is also a much more efficient way of coding instead of writing the same lines of code as many times as you need the results of that function. This modular coding method allows changes to be made once within the function instead of each time an instance of the funtion would be written out. The main source file of my project contains separate functions for shaders and creating the light sources. It also sets up creating the display window, processing user input, resizing the window, and rendering the objects for the scene as separate functions. Additionally, functions to create, allocate, and de-allocate memory for the mesh, buffers and shaders is handled through individual modules. Having these functions existing separately from each other allows me to input variable parameters and call the functions to process the input accordingly. An example of this is the function that creates a cylinder. The cylinder function exists as a separate .cpp file and outlines how a cylinder is created for the project. Code from the main source file is written with the specific parameters of the cylinder such as the radius, number of slices and height. These parameters are then passed to the cylinder funtion to be processed and are used in the render funtion to be displayed in the scene. To create multiple cylinders, I only need to call the cylinder function and provide the parameters for each one I need to create. If I needed the cylinder coding to change, I would only need to change it in the cylinder function and all of the cylinders would reflect those changes. The camera controls were also created as separate functions. Only a few lines of code were needed to assign each key to call its return from the camera header file that processed the keyboard inputs. To customize the keyboard functionality, the process keyboard function was changed in the header file instead of hard coding the changes into the main source file. This is how isolated functions can be used efficiently and can create reusable modular code.