Guilherme Pereira

October 17th, 2021

CS-330

3D Scene – Reflection

The final project consists of four 3-dimensional shapes that make up a scene. In the original portrait, I had chosen a sphere, cylinder, pyramid, cube, rectangular prism, and plane. In the final iteration of this project, I was able to include everything but the sphere. For all the shapes, I utilized an array of vertices to create triangles in the world space, which would all be encompassed within the 3D object. The plane was the easiest of these as it only required six vertices. Two triangles were created to simulate a flat surface (plane) of the scene which represents the desk. The light bounces up from the desk and has a light-yellow birch texture to represent the photo more accurately. The Minecraft torchlight is composed of both a rectangular prism, the torch handle, and a cube, the torch light. This choice was made due to conflicting textures, as having the same mesh drawn would result in the same textures being used. Additionally, the cube torch light has different normal vectors, as the light passes from within the cube instead of outside, simulating the light up part of the torch. As for the textures, I used MS Paint to create them from scratch, as there were no royalty-free textures for this specific object. A shiny blue pyramid was also created, but there were no textures that could accurately represent it in the scene, therefore I decided to opt for a slate-like texture. The water bottle in the scene is composed of three different cylinders, all whose vertices were calculated manually before being drawn into the scene. A separate program was made to attempt in simulating a cylinder vertex generator, but this attempt failed when I was unable to apply those points into an array. All of these shapes were created within their respective createShapeName() function (replace ShapeName with the name of the object) and were drawn to the scene with drawShapeName(). This made generating the objects more organized and cleaner when looking into the code and executing the drawScene() function, which places all the shapes in the world space.

The control camera was created using the camera.h header file, which came with source code to move through the world space using A, S, D, and W keys. Within this file, I also decided to implement the E and Q keys to go up and down, respectively. The camera.h file sets these different directions within an enum to state that these values are not to change. The camera moves in relation to the movement speed that we set and deltaTime, a variable set to 0.0f which accounts the time in between the current frame and the previous one. We update the position of the camera in relation to the Front, Right, and Up directions, which help us move in all six directions.

My favorite custom function I created in this program is the drawScenes() function, which holds onto all of the draw functions for the different shapes and organizes them in a list of what is being drawn. Additionally, the function refreshes all of these drawing instances at the same time instead of being refreshed within their respective drawn functions – which is how the program was originally written and lead to a flickering issue in the render. A draw and create function were made to draw the shapes onto the scene and create their vertices, as well as apply textures to them. Within each draw() function, I updated the parameters to accept arguments in the following order: x-scale, y-scale, z-scale, x-position, y-position, z-position, and the angle of the object. When writing the draw() functions within the drawScene() function, you can customize how the object is oriented and modified into the scene without having to manually change the points within each object. The only limitation to this is that the angle is only done with respect to the y-axis. A processView() function was created to help change between perspective and orthographic views, and an updateCamera function was made separate from each create() function as I found it to be more modular doing so this way. Each create function() is able to call to this function so that their position and orientations are updated along with the different position of the camera.