SNHU: CS-330 Comp Graphics and Visualization

Module 7 – Project 2 Reflection

Matthew Bandyk

For this project, I decided to recreate a scene that held a surface that 4 objects rested upon. The objects were 2 metal boxes, one marble, and a wooden spindle that had a metal hoop around one of the ends. I chose these objects to give me the opportunity to develop multiple different shapes within the scene, all utilizing different types of textures. I believed that this would help push my understanding of what is capable within OpenGL and drive me to build my knowledge in the API. I was not disappointed in this aspect, as it did provide me with challenging situations that I had to figure out how to accomplish. When deciding how to recreate the scene, I identified basic shapes that could be utilized to build the objects in question. Those shapes included multiple cylinders, a torus, two cubes, a sphere and a plane. I will go into detail about how I was able to recreate these shapes later.

The other aspect of the final program was the navigation capabilities for users when utilizing the program. The final program allows users to fully navigate through the scene, allowing them to move the camera left, right, up, down, backward, and forwards. They are also able to utilize the mouse to move the camera’s point of view, left, right, up and down. Also utilizing the mouse, the user can control the speed at which the camera will move throughout the scene, giving them additional controls. The last aspect that I developed was the ability for the user to switch their viewing perspective, alternating between a perspective view and an orthographic view of the scene. To accomplish this, I developed an inline class called Camera, that would take the various inputs from the keyboard and the mouse and move the camera accordingly. To start, the camera is placed at a fixed location, with its x, y, and z coordinates predefined, as well as the pitch of the camera, having it look down at a 45-degree angle. This puts the camera at the same angle that was utilized when taking the 2D image that the scene was based off of.

The program runs in a constant loop to render the scene, and with each loop, inputs from the keyboard and mouse are captured and then passed to the Camera class, which will then move the camera based on the input. For example, if the user were to press the ‘W’ key, that would move the position of the camera upwards by multiplying the current y coordinate by the delta time, or the time between the current frame and the last frame, which gives it a smooth movement no matter what system the program is run on. This same process is utilized for each of the movement directions utilizing the keyboard keys ‘W’, ‘S’, ‘A’, ‘D’, ‘Q’, and ‘E’. The same is true for adjusting the camera’s point of view through the mouse, where the distance that the mouse moves is captured and utilized to move the camera’s pitch and yaw. The view that the user is currently in is captured through the press of the ‘P’ key, which if pressed, will switch from perspective view to orthographic view, or vice versa depending on the current view. When pressed, a Boolean value is switched between true and false, with each representing a view type. The last keyboard input captured is the ‘ESC’ key, which when pressed, will exit the program out of the render loop and close the program.

To develop this program, I created many custom functions to handle multiple needs. The first functions I will cover are the functions utilized to create mesh data for the various object types. I developed a function for each of the object types that would be called from the main function to create a specific object at a specific size and quality. The CreateCylinderMesh function was developed to create the vertices, normals, and UV coordinates needed to render a cylinder. The function also utilizes a struct object that holds the various variables associated with the cylinder, and also allowed me to utilize the function multiple times to create different cylinders needed in the scene. When calling the function, there are many parameters passed to the function, including, the radius and height of the cylinder, the number of sectors and stacks that should be utilized, the texture IDs associated with that object and its various faces, the translation of the object, and finally a vector that holds all of the cylinder structure objects to render later. The radius and height passed dictate the size of the cylinder to be created, while the sectors and stacks indicate the number of triangles that should be utilized to create the object. The more triangles created, the more like a true cylinder the object will look. Utilizing geometry, the function creates all of the vertices needed for all of the triangles within the cylinder, as well as identify the normals for each of the vertices for lighting and the texture coordinates that will be used to apply the texture. The function creates multiple VAO’s (Vertex Array Object), VBO’s (Vertex Buffer Objects), and EBO’s (Element Buffer Objects), and binds that data that has been created in the function to be utilized in the rendering process. It then pushes the cylinder object into the cylinder vector which is utilized to render the multiple cylinders.

The CreateTorusMesh, CreatePlane, CreateCubeMesh, and CreateSphereMesh functions are executed in much the same way, allowing me to utilize them as many times as needed to create as many of the shapes as I need for the scene being developed. This will allow me to utilize all of these functions in future projects as well, reduces the amount of code that needs to be produced, and allows for better organization of the objects being created. The biggest gain of these functions is the ability to assign different textures to the different faces of the objects, like the cylinders and the cubes. ss the objects I had chosen in my scene each utilized a different texture on each face.

To help simplify the texturing of the objects, I utilized a map to preload and hold the texture ID’s which could be passed into the functions and assigned to the individual struct objects to be called on in the rendering of the objects. This also allowed for better organization and management of the many textures being utilized within the code.

The next function that I developed was the render function which is utilized to render each of the objects in the scene. There are some nuanced transformations that are completed within the render function that allow me to manipulate multiple objects as a single object. The reason behind building this into the function is that one of the objects in my scene is built out of 4 simple objects, 3 cylinders, and a torus. When creating the mesh for the objects, I also assigned a translation to each that would set them up in the correct location in relation to each other. Then, within the render function, I create transformations that are applied to a single model matrix which allows me to rotate all of those objects together around a single point and move them as a single object to get the complex object into the correct location within the scene. Once the model matrix is created, I then apply it to each of the objects as they are rendered. I also set the light properties in the shader within the render function before rendering the objects and textures themselves. The objects are then rendered, textures applied, and then I unbind the VAO’s and textures.

To light the scene, I’ve utilized 3 point lights in total. 2 of the point lights are utilized to replicate the two lights that are producing light in the 2D image that I captured. Both are colored lights, utilizing a soft off-white, or yellowish, color to replicate that of a light bulb. They are both positioned above the objects, one behind them, and one slightly forward. The last light is placed in front of the objects and is utilized more for ambient lighting to replicate the sunlight that was passing in through windows on both sides of the room when the image was taken. This utilizes a natural sunlight color of pure white.