Evan Malia

Prof. Enekma

CS 330 Desgin Decisons

I chose the entertainment center in my living room for this project as the 3D scene. In hindsight, I could have narrowed down the scope to just the shelves of my entertainment center and focused on the objects on those shelves. However, I opted for a broader approach, taking a more expansive view of the space. Even though I could have reduced the number of objects, I chose to challenge myself by creating objects with small details. At first glance, the scene might not seem complex, some objects have intricate details like handles or often are overlooked. One example is I managed to make the stands of both the TV and the entertainment center from tapered cylinders, as they closely resembled these in real life. Most of the other objects had modular parts that were easily discernible, as most of the meshes I utilized in the program were box meshes, including the screen, which allowed me to add a dynamic texture to it. I just made the depth of the “screen” very small so it looked like it was a screen. However, even though I used basic objects, I had to create every shelf, every wall of the entertainment center, and the tower to the right of the table. The most arduous part of this project was getting all the objects properly aligned on the XYZ axes, as it required precise optimization to emulate reality.

The user can navigate the scene by controlling the camera using keyboard and mouse inputs. From the keyboard, the user has many options for movement: Forward, Backward, Left, Right, Up, and Down, with the keys associated with these movements being W, S, A, D, Q, and E, respectively. These commands are the only way to navigate the camera through the scene. However, the user can also toggle between different projections of the scene: orthographic and perspective views. Orthographic view omits the plane and represents the 3D objects as 2D from the perspective of the camera. The perspective view provides an organic view of the scene, showing each object in 3D and offering an excellent default position of the camera if the user wants to navigate through the scene. To toggle between these different projections, the user would have to press ‘O’ and ‘P’, respectively. From the mouse input, the user could scroll the wheel forward to increase the speed or backward to decrease the speed of the camera in the scene. If the user moved the mouse the program would change the orientation of the camera allowing the user to look left, right, up, and down.

Several pre-defined classes, headers, and methods made this project a modular experience, making it much easier to manage and edit the code as a developer, as most of the tools were abstracted. The key was to be able to use the methods each class provided, as it was not essential to understand how each class worked. The functions I created were to generate the different objects in the scene, including the floor and the backdrop. In the RenderScene() function where the objects were created in the project that would become a lot of lines when more meshes were added to the scene, I added functions for each object render. As the code became gradually more complex, having each object in a function was a time-saver. I could toggle a full view of the function or close it, making the code more easily navigable, which came in handy when I had to add textures and lighting to the scene. The way I used these functions can easily be used again in similar projects but some coordinates might need to be adjusted. Overall the code provided before it was edited was a great environment to prepare the scene creation and can be carried over to other projects that utilize the OpenGL API; apart from the addition of more meshes in the ShapeMeshes.cpp and ShapeMeshes.h files.