Tristan Lacey

Professor Brian

CS-330 Comp Graphic and Visualization

08/13/2023

7-1 Final Project: Design Decisions

Throughout this course I learned how to use OpenGL to render graphics to create 2D and 3D objects and scenes. The real objects I chose for my project were picked because they each had unique colors, shapes, and sizes. I liked how the objects contrasted with each other this way and knew it would be a good opportunity to try using different primitive shapes. Creating complex shapes from primitive shapes was certainly more challenging than I first anticipated. In terms of my development choices, to start I used a plane as the base and found a fuzzy blanket texture to recreate the blanket canvas scene. Next, I chose to use a three-sided pyramid to recreate my glass lamp shade. The texture mimics the object’s colors well and catches the light similarly. For the Doss Box, I decided to use two cylinders and stack them together, in order to reproduce the rounded edges of the device. I used a speaker mesh on the outer cylinder and found a texture for the control panel that I was able to place on the center top. These combinations of shapes and textures seemed to bring the Doss Box together. For the poke ball, I had originally used a combination of primitive shapes to re-create my own poke ball. In my final project I decided to use a sphere and apply a different poke ball texture to achieve the same effect, though the original code remains commented out since I had fun making it. The most complicated shape to reproduce was the dumbbell weight by far. I decided to use a combination of boxes to create the left and right sides of the weight, connected in the center with a cylinder for the handle. In order to render these different shapes, I needed to leverage the meshes.cpp and meshes.h files that were provided. This allowed us to call different functions when binding the vertex array, in order to draw various shapes. All of the shapes that were provided had the vertices, normal, and texture coordinates all pre-defined in the meshes.cpp. Once these shapes were added to the URender function, I was able to manipulate them by using a combination of glm scale, rotate, and translate, in that order. To add texture, we used the stb\_image.h header file to allow processing of many different image formats. I used texFilename to define the file names of the images used to texture objects in the scene. Once the images and textures were declared and loaded successfully, I was able to bind them to different shapes using glBindTexture. Lastly, I added light using the Phong lighting model to calculate ambient, diffuse, and specular lighting. I added a white key light in the center to simulate the real light above, and a tan fill light to help add contrast to the scene. This required specific shader programs to define the light characteristics, such as position in world space, intensity of the light and the light’s color.

The camera allows the user to navigate around the scene and view the objects at different angles. In Week 4 we learned how to apply navigation to our scene by using the input devices of keyboard and mouse. The user can use WASD to move the camera forward, backwards, left or right. The Q and E keys let the user move the camera up or down. The mouse controls which way the camera faces and let’s the user look around in the scene. The user can increase camera movement speed by using mouse wheel up, and slow speed by using mouse wheel down. Using the ‘P’ key toggles between perspective and orthographic views. All of this was achieved using the camera.h header file, and several functions for input such as UProcessInput, UMousePositionCallback, UMouseScrollCallback, and UMouseButtonCallback. These functions allow the program to register input device movements or button commands, and take action based on code in the camera.h/source.cpp files. One of the custom functions I created prevents the ’P’ key from sending multiple inputs when holding the key down. It will only send a single ‘P’ strike until the ‘P’ key is released. I don’t have too many other examples of truly custom functions in my code for this project, however I see some opportunities in URender to possibly refactor and reorganize how my complex shapes are created. Refactoring them into a single function would help to organize the code and make it more readable when trying to understand how the complex shapes are created. It was very challenging and rewarded working to achieve this scene, and there is still more I feel I could do to polish and add to it in the future.