Austin Little

SNHU

CS-330

12/10/2021

**Final Reflection**

1. **Justify development choices for your 3D scene:**

**The objects that I chose were all chosen for simplicity reasons. I chose to create a table, a ball, a cologne bottle, a pyramid, and a glass cube. The table is three cylinders that vary in size. The ball is a sphere, the glass cube is just a textured cube, the pyramid is just a textured pyramid, and the cologne bottle consists of two cylinders of varying sizes.**

**My reasoning for using these objects also falls upon the ease-of-implementation into my project. I should be able to create and texture these items with simple shapes by creating a vertex array object for each shape. Each vertex array object can then be activated and rendered accordingly. Finally, each vertex array object will be deactivated when the program is closing.**

**The objects that are made with cylinder and spheres both referenced the header files corresponding to those shapes. These header files used mathematical functions and created the shape when it was called in the main function. For the pyramid and box, I created vertices that could be used within the vertex array object(s) and vertex buffer object(s). This allowed for the use of “GL\_DRAW\_TRIANGLES” to render these shapes accordingly.**

1. **Explain how a user can navigate your 3D scene:**

**Having camera controls is essential to any graphical program. I chose to implement the camera controls in the source file. By creating a function to process input, I was able to check if a specific key was pressed and move the camera accordingly. I achieved this by using the “glfwGetKey” function and checking if a key was pressed. If the key was pressed, I would update the variable for camera positioning to accommodate for the given movement. The speed that the camera would move depends on the cameraSpeed variable which was raised or lowered depending on the users scroll wheel.**

**We also needed to allow for perspective changes within the scene. This was done by creating a global variable called “ortho” that would start out set to false and be the answer to switching perspectives. By implementing another conditional within the “processInput” function to check if the “P” key was pressed, we could change the ortho variable to be “not ortho” (!ortho) and this could change the variables value easily. This would do nothing as is, but that is why I implemented a conditional in the while loop in the main function. I started by declaring a mat4 variable called projection with glm. I could then use an if/else statement and check if ortho was true or false. Depending on the value of this variable, the scene would change the projection variables value and allow the scene to switch from perspective to ortho view.**

**The final part of navigation was the mouse inputs. The “mouse\_callback” function would check the coordinates of the user’s mouse in the scene and offset the view accordingly by using and manipulating the x-offset and y-offset variables. This allowed for updating the coordinates of the mouse constantly and allowed the user to look around the scene just by moving their mouse.**

**As for the movement speed, I created a function called “scroll\_callback” which increased or decreased a global variable for movement speed depending on how the user scrolled their mouse wheel. Scrolling the mouse wheel affects the y-offset and in turn changes the movement speed of the camera. I also implemented a range in which the movement speed must stay. The final piece to making this work was creating a variable in the process input function that was set equal to the movement speed variable multiplied by delta time.**

1. **Explain the custom functions in your program that you are using to make your code more modular and organized:**

The main function that allows for modularization would be the function that processes input. This function can have conditionals that check if the user presses a key and modify the program accordingly. Having this function saves having to look throughout the entire code base multiple times for each input conditional.

The other helpful components of this project are the sphere and cylinder header files. There are many header files contained that all increase modularization, although these were the two most useful for quick implementation of shapes within the scene. These headers contained the mathematical functions needed to fully create varying sizes of cylinders and spheres. This allowed for less lines of code within the main function and a simple call to render the shape from the header file.