Brian Hartong

Professor Gray

CS-330

11 December 2022

Final Project

I chose the selected items in my scene because I thought I could create them using the primitive shapes discussed in this course. Having some background in 3D modeling helped me understand how to combine different primitive shapes to create a more complex item. This allowed me to make more informed decisions on which items to include in my scene.

I chose the shape sorter for the scene due to the multiple shapes that would be needed. I used a cylinder for the ring shape and a five-sided cylinder for the pentagon along with the cubes for the other shapes and the base. The first step of creating the cylinders was calculating the positions of the bottom and top circles. I first created an array that would contain all the x and z coordinates for each vertex of the circles then calculated the positions of those vertices using a supplied radius, the number of sides, and the number of vertices per circle (number of sides plus two). The vertices were then calculated using sin and cos to get their positioning. Once both circles were calculated, it was just a matter of creating the vertex array for each triangle of the cylinder.

I decided to include the ring stacker toy in the scene in order to use a torus shape to make the rings. The shape of the rings was changed from a torus to a short cylinder due to the complexity of the math to determine the location of the vertices of the shape. This choice was necessary for the completion of the scene because completing the necessary calculations would be too complex for this project. The ball was included in the scene to add a use of a sphere to use more types of objects.

A user can navigate my scene through use of a keyboard and mouse. The main controls use the WASD keys to move the camera through the scene and the Q and E keys to move the camera up and down. I also included the use of the arrow keys to pan and tilt the camera to allow more angles to view the scene in. To move the camera, the direction determined by which key is pressed is multiplied by a set velocity. This value is then added or subtracted to the position of the camera depending on the direction the camera needs to move. An example of this from the code is when the W key is pressed, “if (glfwGetKey(window, GLFW\_KEY\_W) == GLFW\_PRESS)”, then the camera position is moved by the set velocity in the forward direction, “Position += Front \* velocity;”. The mouse can also be used control the camera, but since this is limited to the dimensions of the application window, the arrow keys were added to give the larger rotation of the camera.

The custom functions that I used in my program to make the code more modular and organized include the use of an initializer function. This function was used to initialize the GLFW and glad along with creating the window of the application. This makes the code modular and organized by allowing other projects to use this function to quickly setup an OpenGL application. The other custom functions that I created were mesh creation functions for a cube, cylinder, and sphere. These functions created the given objects based on parameters passed to them. This also makes the code more modular by allowing other projects to use these functions to quickly make the shapes by supplying the dimensions of the shape and an identifier to access the shape. These functions make the application more organized by reducing the amount of duplicate code that would be needed to create the same shapes in the scene, thus making bug fixes easier by reducing the number of locations the code would have to be fixed in.