Elliot Huh

CS-330

Module 7-1

Project Reflection

**Justify Development choices**

I chose the following objects to replicate: an apple pencil, a small wooden plank, a drinking glass, and a small bookshelf. I chose these objects because they are simpler objects that will be easier to replicate in a 3d environment. The apple pencil was created using cylinders, for the body, a tapered cylinder combined with a cone for the tip, and a sphere at the opposite end to represent the rounded end. The wooden plan was created using a box a the body combined with 2 flattened cylinders on the sides to represent its rounded sides. The drinking glass was represented with a large tapered cylinder and the bookshelf was created using multiple boxes. These objects had a simpler path and direction that made it easier to replicate.

These objects had clear distinct textures and materials, they did not have a mix of varying differing textures and materials. This simplified the process of applying the proper resources to the complicated objects making the required functionality of the scene simpler in terms of the objects. The apple pencil had some variety although it was minimal and easy to work through. The scene was also on top of a carpet inside a room, this would help define the lighting and additional textures and materials. Textures were sourced from wiki advanced search for JPG images. These images were then applied onto each basic shape within the numerous complex objects.

Additional each texture had its defined material with distinct shininess and other aspects that help define its relationship to lighting. This would help define shinier objects that reflect light. Lighting was then applied in a way to try and match the setting in a room. All these developments were applied to the SceneManger.cpp file.

**Navigation in 3d scene**

Navigation primarily revolved around the keyboard input device and a mouse. We can see this with the GLFW input command function with the scroll input codes and the keyboard input codes. Each key that had a function was clearly defined within the code. W and the S keys were utilized to zoom in and out, A and D keys were utilized to pan the camera left and right, Q and E was to pan the camera up and down. The mouse scroll wheel was utilized to manipulate the speed of the camera movement. The mouse movement would manipulate the view of the camera, the camera’s view and perspective would move alongside the movement of the mouse. Orthographic projection views were utilized. This view was defined within the code and its proper settings and configurations were defined to show the scene in an orthographic projection. This was set to the O key and the P key was assigned to return to the original setting perspective view.

Additional input devices can be utilized with further modifications and considerations to the code. Further functions and utilization of the GLFW Input functions would need to be defined and utilized. All changes were applied to the ViewManager.cpp.

**Functions that make my code more modular and organized**

The numerous function that was added to scenemanager.cpp file was created to replicate the objects within the 3d scene, render their textures, define the shininess of the object, and create the lighting within the scene. These functions build the scene and replicate the 2d environment in a 3d setting. The functions that was added to viewmanger.cpp was focused on the user and how they interact with the scene through their input devices. The numerous keys and abilities that the user had at their disposal was defined.

To maintain modularity and organization each function was divided up into their own class and section within the code. Within the Scenemanager.cpp file their were specific classes and sections for each core aspect of the scene. The object class had its specific section within the code where objects were created defined and had their textures redendered. This was seen within the render scene class. Other examples can be seen with loadtexture class where the textures were defined and loaded onto the project and defineobjectmaterial which did the same thing for materials. The Viewmanager.cpp followed a similar trend. By splitting up each function into their own classes and sections it helps the code maintain reusability.