JJ Newell

6/21/25

CS330

CS330 Project Reflection

The scene I chose to develop was one of a computer desk with a mousepad, mouse, keyboard, and monitor, more specifically I used a picture of my own set-up. I chose these objects as they fitted the criteria needed for the project as they were objects which required the use of multiple 3D shapes in combination with one another to create. I also chose them because I could simply look down at my desk while working for more reference about the objects I was creating, thus eliminating the need to constantly pull up the reference photo. The mousepad was created using a thin box, the desk and wall were represented by two textured planes. The mouse was created using a flattened sphere for the body, with boxes for the side buttons and a cylinder for the scroll wheel. The keyboard was made using a series of boxes that were scaled and rotated appropriately. The monitor was made using the greatest variety of shapes. Firstly, the base was made using a flattened cylinder and the connecting support legs were made with prisms as they most closely resembled my reference photo. The support beam was made with a cylinder and the screen/connector piece to the screen were both made with boxes.

The user can navigate my 3D scene through the use of two input devices, namely the keyboard and mouse. The mouse is used to look around the scene and control camera speed with the scroll wheel. Whereas the keyboard allows you to freely move with the WASD keys, in addition to being able to use shortcut keys for a perspective view and various orthographic views (front, top, side).

There are not many custom functions I personally developed in my program as a lot of the project was understanding sample code and implementing it from an assignment into my project. However, one unique path I took was the way in which I called the SetTransformations() function for a majority of my SceneManager.cpp file where the bulk of my code was written. The following is the example way in which this stretch of code was written for a majority of the program:

// set the XYZ scale for the mesh

scaleXYZ = glm::vec3(10.0f, 1.0f, 6.0f);

// set the XYZ rotation for the mesh

XrotationDegrees = 0.0f;

YrotationDegrees = 0.0f;

ZrotationDegrees = 0.0f;

// set the XYZ position for the mesh

positionXYZ = glm::vec3(0.0f, 0.0f, 0.0f);

// set the transformations into memory to be used on the drawn meshes

SetTransformations(

scaleXYZ,

XrotationDegrees,

YrotationDegrees,

ZrotationDegrees,

positionXYZ);

SetShaderTexture("desk");

SetShaderMaterial("light1");

m\_basicMeshes->DrawPlaneMesh();

In this method, values are assigned to their corresponding variables and then called near the end in the SetTransformations() function to finalize them. The following chunk of code is the format I utilized to shorthand this process and make it more compact and quicker to write:

SetTransformations(

glm::vec3(1.0, 0.1, 1.0), // scale

0.0, 0.0, 0.0, // rotation

glm::vec3(0.0, 0.0, -3.0)); // position

SetShaderColor(0.2, 0.2, 0.2, 1); // grey

m\_basicMeshes->DrawCylinderMesh();

The way in which I assign values to the scale, rotation, and position is different as I call the SetTransformations() function and assign its values at the same time. Thus the output is the same but it is overall less lines of code which makes the process faster to replicate for other shapes. One downside however is that this approach takes away from part of the readability as it does not show the variable names of what is being assigned what value. But, this is offset considering that comments are written right next to the variables being assigned values with what exactly the value is (scale, rotation, position).