Andrew McPherson  
CS-330  
6/19/2022

**7-1 Project Reflection**

For my 3D scene, I chose objects that I thought may prove to be slightly complex while also still being able to be created with more low-poly and primitive shapes. Since I was new to computer graphics, I didn’t want to choose something that would be incredibly difficult to faithfully recreate, but I also didn’t want to be too simple where there wouldn’t be any challenge to it. That is why I chose a Nintendo Switch as one of my main pieces, because it has the potential of being quite challenging. Though I didn’t go the route of adding every single button on its surface, I did spend considerable time adding in appropriate textures to those areas so it will look the part regardless of adding in those extra shapes.

One development choice I made was to not have to reuse or recreate a new vertex array just for the Arizona Green Tea can, it is using the same vertices I created for the coaster, just each object is being transformed differently. I had thought about changing it up and maybe even making the can an octagon instead of a hexagon, but I decided it was better to reuse what I already had to create both objects. This came with a struggle of using the textures correctly. How can I map two textures for two completely different things all on the same set of vertexes sharing texture coordinates? At the time I only had the coaster, and thus the coaster texture which was just a square image with the round coaster on it. Since I wanted the side of the coaster to be a different texture, I had the struggle of changing it all up for when I created the can. This choice led me to changing up my coaster texture file and creating them both in the same way. One texture for the side that is tall and long, from (0.0, 0.0) to (0.5, 1.0) *the left half*, was the texture I would need for the side, and from (0.5, 0,5 to 1.0, 1.0) *the top right quarter,* I could use for the top of the cylinders to get that coaster texture and can top texture.

To navigate my 3D scene, you can use the standard movement keys you might find on a PC game of WASD, W for forward, A for strafing left, S for backing up, and D for strafing right. I chose these keys due to the immediate familiarity anyone may have with computer movement, because it is essentially the common “arrow keys” of the keyboard that is used in games. Of course, I could have also set up the actual arrow keys instead, but that felt redundant. Aside from those movement keys I added Q and E for vertical elevation changes (Q is up, E is down). It isn’t a “up” so much as it changes the Y value, so if you were looking at the scene from straight up and pressed Q, it wouldn’t move you up related to the direction you were looking, but instead just vertical height.

Then for moving the camera you have basic mouse control to rotate the camera around, and I also set up the mouse wheel to be able to change the speed of which you move with the camera. There isn’t necessarily a maximum speed, but there is a minimum speed of how slow you can go, but this allows it to be easier to navigate to be able to change your speed to what suits you best.

Regarding modular functions and reusability there is a lot more to be desired here, I could have set up every single object I created to be made in a different way by using functions and loops to recreate and rotate vertices, but I did not go this route mostly due to time constraints. This means that I had to take the time to plot out every vertex essentially in my head to get the desired effect, but there are more practical ways to go about this. My code isn’t completely un-modular however, as I am using things like UCreateMesh which stores all the vertices I created, sets them to their own Vertex Buffer Object /Vertex Array Object, which those are then later used in the URender function to bind, transform, and draw each object. Though one example I did mention regarding reusability is how I as able to use the same array of vertices for two completely different objects and have them both be textured accordingly, that is one way that I kept things reusable.

But, as I said there is still much to be desired in reusability that I was not able to implement, so instead this function is incredibly long and scrolling through the list of vertices takes considerable time. Other ways I could have made this more modular is by putting this function as its own class, that way the bulk of the code is not visible in the main .cpp file, on top of modifying it so that once I created a certain shape, I could define that as its own function to call when I needed that specific shape to create alongside other varying objects.