**Brandon Hobbs**

**CS-330**

**Module 7: Final Project Reflection**

**June 11, 2023**

Upon finishing the Final project for CS-330 a moment of reflection is warranted. The 3D scene chosen was created using random items from my kitchen that fit the criteria: 4 items, with one being a compound shape, I selected an orange paper box, a citrus reamer, an orange, a plastic placemat, and a plastic bag dispenser, Figure 1.

A picture containing text, yellow, orange

Description automatically generated

**Figure 1: Four Random Items Selected**

These four items had to be constructed from the OpenGL primitives:

* Cube
* Cylinder
* Plane
* Pyramid
* Sphere
* Torus

To begin recreating the scene in OpenGL the orange placemat, to serve as the base of the other items, was created. Each item was then recreated using these primitives:

* Orange is a single sphere
* Orange box is a cube
* Citrus reamer is two square pyramids rotated 45 degrees, an oblate sphere, and a tapered cylinder
* The bag dispenser is 4 elongated cubes and 4 cylinders

Besides the obvious choices for the orange and box some thought was used to select the primitive shapes of the other 2 objects. I wanted to capture the round corners of the bag dispenser so cylinders were employed. This choice doesn’t allow the inside edge to be concave, however. The straight edges were simple elongated cubes.

The citrus reamer is close to the shape of a pyramid, but has fluting. The fluting suggested that two rotated pyramids should be employed to give the effect of having 8 flutes. The handle appears to ovular in shape. To capture this an oblate sphere (2.08x aspect ratio) was used. However, the actual reamer has a flat spot at the base of the handle so a tapered cylinder was placed on top of the oblate sphere to add this feature.

To facilitate placement of objects within the scene camera controls were added via OpenGL callbacks.

* Pressing ‘P’ toggles an orthographic projection aligned with the Z-axis. This view enables quickly checking relative positioning in the XY- and ZY-planes.
* Pressing ‘W’, ‘S’, ‘A’, or ‘D’ allows the camera to move forward, backwards, left, and right, respectively. Pressing ‘Q’ or ‘E’ allows the camera to pitch up or down
* The mouse enables the camera to pitch and yaw based on user movements
* Using the scroll wheel increases or decreases the camera movement sensitivity. Sensitivity is locked between 0.01 and 3.0 units
* Clicking the scroll wheel resets the sensitivity to the default setting of 0.03 units

To improve the code readability and maintainability I tried to reduce the duplicated lines of code. For example, I have five light casters and all five use the same code to pass their attributes to the shaders. Instead of the code being duplicated five times a loop was added – this code is within the C*reateLights* function. Moreover, the loop needs string slicing but the compiler could not resolve the type casting needed so a helper function, *GetUniformLocation*, was created that could do the slicing and then return the formulated GLEW function.

Again, to save duplicate code in the light shaders two functions were written: *CalcDirLight* and *CalcPointLight*. These functions allow the code for the five lights to be reduced. Reducing duplicate code also reduces the change for bugs to be present.

All of the custom functions take advantage of 2 structs added to the parser. Again, this makes the code easier to read and maintain.

The last optimization made was to build all of the light casters in a loop. Because all of the light casters would have the same properties, except for position and color, there is no need to duplicate that code. Instead, a global list variable was created to hold the light casters’ positions. The code then loops through the code iterating over each of the light positions and sends the commands to the shader – this was added to the *Render* loop.

