**Project Reflection**

Justin Winstead

Southern New Hampshire University

CS 330: Comp Graphic and Visualization

Dr. Diesch

June 19, 2022

**Development Choices**

The objects in my scene were interesting to break down into primitive objects and code. For the box of Cheez-Its, I modified a set of cube vertices to create a shape that accurately represents the item. A cube was the most obvious shape to work with when creating a box, and it worked well in the scene with the texture I applied to it. When creating the bottle of olive oil, I utilized a sphere and two cylinders created with sphere.h and cylinder.h, respectively. These shapes, when paired with my glass and label textures, created a faithful representation of a glass bottle. Next, I used a sphere and two cylinders to create the Apple pen in the front. Initially, I planned on using the concepts behind a pyramid to visualize the tip of the pen. However, cylinder.h provided the ability to use differing radii for my cylinder which made making a cone much easier. These decisions allowed me to create the most accurate pen possible. The final object I created was a perfume bottle. This object was essentially just a cylinder in real life. Therefore, a cylinder was the right choice for this item. My primitive shape choices shifted throughout the development process. However, the result I arrived at was a strong visual representation of the scene I had chosen.

When coding this program, I utilized a variety of learning resources and currently existing header files to ensure that I was creating a scene that was not only visually appealing, but functional as well. I used a series of mesh structs to house the primitive shapes that I had created. These meshes passed vertex information to the shaders I created alongside the textures and changes in the model, view, and projection matrices to draw the objects I desired. The shaders I created utilized two sources of light for the scene. The main light was white and shone on the front of the objects at full intensity. The secondary light was a softer yellow color that was placed behind the items in the program. These lights gave my textures an appealing look while minimizing how much of my objects’ surfaces were obscured by shadows.

**Navigation**

A user can navigate through the scene I have created using their keyboard and mouse. The program moves the user forward, back, left, right, up, and down when pressing the W, S, A, D, Q, and E keys, respectively. These keys move the camera throughout the scene without changing the direction the camera is pointed. The user can also press the escape key to exit the program. The P key is supposed to swap the user from 3D to 2D viewing and back. However, I was unable to get this to work in my program, despite trying. This functionality was achieved using my processInput function. I used glfwGetKey to monitor what keys are being pressed and what should happen on a key press. The user can move their mouse to adjust the direction of the camera and utilize the scroll wheel to change the speed of their movement throughout the scene. Camera.h provided most of the processing for this functionality. With the camera direction adjustment, I used previous and current x and y positions for the cursor to calculate where the mouse had moved. This allowed me to use the ProcessMouseMovement function from Camera.h to adjust the viewing direction for the camera. Similarly, when using the scroll wheel, I utilized the ProcessMouseScroll function in Camera.h to handle processing scroll wheel movement. However, I adjusted the file to ensure that the scroll wheel would adjust the speed of movement rather than the zoom of the camera.

**Custom Functions**

I utilized a variety of custom functions when creating this program. The most important function I created was render. This function housed all the code for drawing the scene. It is easily modifiable to allow me to adjust any objects or to add new ones. The next set of functions I created is the createMesh series. These functions were responsible for creating the mesh structs that were needed to draw the primitive shapes in my scene. Some of these are more reusable than others. For example, createSphereMesh and createCylinderMesh can initialize a mesh struct using a Sphere and Cylinder object, respectively. In contrast, createBoxMesh can only create a mesh for the specific set of vertices defined inside it. The input processing functions processInput, processMousePosition, and processMouseScroll are used to convert inputs into movements through the scene. Although implemented differently, they can all be modified easily to adjust what actions the inputs perform. processInput can be used in the render loop to consistently monitor keyboard inputs. createTexture is utilized to process image files to be mapped onto the objects I had created. This can be used to process a variety of images and I processed .jpg and .png photos for this project using this function. The last custom function that should be covered is createShaderProgram. This function takes the source code for a vertex and fragment shader and initializes a shader program using the code. I utilized it for 3 distinct shader programs in my project and it worked well.