Jonathan Miller

Prof. Eisen Montalvo

CS-330

October 12, 2023

Final Project Reflection

- *Justify development choices for your 3D scene.*

The objects that I chose for my 3D scene were arbitrary, but the choices were intentional. I intentionally chose arbitrary objects that I had around my house that could be realistically recreated using primitive shapes. For instance, each AA battery could be accurately recreated using two cylinders, the portable headphone amplifier could be accurately recreated using a cuboid and three cylinders, the phone box could be accurately recreated using a single cuboid, the marble could be accurately recreated using a single sphere, the table surface could be accurately recreated using a single plane, and each window could be accurately recreated using a single plane.

These primitive shapes could be built using algorithms rather than importing complex meshes. I was able to program functions to build the vertices (and indices if needed), normal vectors, and texture coordinates for each primitive type using common geometry knowledge and iteration. I had to utilize hand-draw sketches so that I could have a visual representation of what I was trying to achieve in my mesh builder functions. These mesh builder functions were then utilized in mesh creation functions to produce the various shapes in my scene. In combination with the meshes and texture files, shaders were used to bring lighting to the scene. The object shader calculates and applies the texture color and various lighting properties for each individual mesh that utilizes it. The lighting applied to each mesh was calculated using the Phong model of lighting, which is composed of ambient, diffuse, and specular lighting. The light source mesh utilizes a different shader that does not apply lighting properties to it, and instead only applies a single color to it. Each shader also applies transform matrices to each object to move the mesh, or meshes, and camera around in the 3D and 2D views.

- *Explain how a user can navigate your 3D scene.*

The user is able to navigate through my 3D scene using three-dimensional transform matrices that get fed to the shader. These matrices hold coordinates for the position of the camera and vectors for the up and front directions of the camera. Keyboard and mouse input is used to apply transformations to these matrices to move the perspective or orthographic camera around.

The *W*, *A*, *S*, *D*, *Q*, and *E* keys are used to apply transformations to the camera's position, while *mouse position offsets* are used to change the vector for the direction the camera is facing. The speed of camera movement is kept consistent by utilizing a delta time variable, and this speed can be adjusted by moving the mouse's *scroll wheel* up or down. The camera movement speed can be increased infinitely in the current implementation, but the camera movement speed can only be decreased until the speed reaches zero to prevent unwanted inversion. The *P* key is also utilized to toggle a variable that facilitates the switching between the orthographic and perspective views. When the P key is first pressed, the orthographic camera and view is utilized in each shader, and once it is pressed again, the perspective camera and view is utilized in the shaders.

- *Explain the custom functions in your program that you are using to make your code more modular and organized.*

My program is organized into many different sections. The first section, after inclusions, consists of declarations for global variables, constants, and structs. Then, custom function prototypes are declared, followed by shader program definitions. Then the main function is entered, and it contains initialization logic, the program loop, and destruction function calls. Next, the custom functions are defined.

I started off with defining the various input callback functions, followed by mesh creation functions, draw functions, setup functions, and then destruction functions. I also extracted out my mesh builder functions into separate classes to increase readability of the main program file, and they can now be included in other OpenGL projects that wish to build primitive shapes.

There are many modular functions for each of the previously mentioned function types. For instance, the input callbacks can be reused in another OpenGL program for world navigation. I also separated the mesh creation functions into a *UCreateMesh* function and functions to create each primitive shape using a mesh builder. This *UCreateMesh* function takes in a mesh ID and its associated vertices and indices vectors, and then it creates and activates the VAO, VBO, and vertex attribute pointers for the mesh. This *UCreateMesh* function can be reusable in other projects to create meshes for any shape.

The draw functions are separated into the main *URender* function that enables various render properties and calls the draw function for each defined object. I have also created a function that draws a specified object mesh with the given texture and properties. This *UDrawObjectMesh* function is utilized in each object’s draw function, and it can be reused in other OpenGL programs to render a mesh with a specified texture and Phong lighting model applied to it. The defined shaders' source code, along with the *UCreateShaderProgram* and *UCreateTexture* functions, can also be reused in other OpenGL projects to apply textures and lighting to meshes.