**Final Project: Reflection**

Robert A. Rust III

Department of Computer Science, Southern New Hampshire University

CS-320: Computational Graphics and Visualization

Professor Eisen Montalvo, M.S.

August 18, 2024

**Final Project: Reflection**

**Development Choices for the 3D Scene**

The objects selected for the 3D scene were chosen due to their ability to be replicated in a reasonable amount of time for a programmer new to graphics programming. Some of the objects were simple, requiring little adjustment to be accurate, such as the box of pasta. By adjusting the X, Y, and Z scales on the box mesh, the correct shape was made. The recipe for creating an object material for a cardboard box was simple: neutral ambient strength, a darker defuse and specular light color, and very low shininess. Other objects, such as the candles, required composing multiple primitives to construct. For example, the red candle had a short and wide cylinder for the base, a slightly smaller radius cylinder for the neck, a half-circle to connect the neck and the base, and a cylinder for the lid — each of these primitives needed to be carefully placed to convey any meaningful information. Placed correctly, they replicate the candle very well. I chose to create a room to enclose the scene, with a tiled ceiling, brick walls, and a concrete floor. The table has an oak texture and wood object material. Were I to continue, many more objects would be placed around the room to make it livelier. As it stands now, it has a lonely vibe, much like an interrogation room.

**3D Scene Navigation**

To navigate the 3D scene, both keyboard and mouse inputs are used. To control the camera, callback functions are set to execute on keyboard and mouse inputs. The GLFW library provides functions to wait for user inputs and set up callbacks. In the mouse callbacks, the X and Y offsets are calculated based on the last known position of the mouse. The mouse movement is then processed by the camera, where it moves based on a mouse sensitivity multiplier. A callback is also set up in response to scroll wheel inputs which increases or decreases the mouse sensitivity and movement speed by a constant factor. The pitch of the camera is clamped to 89 degrees in both the positive and negative direction to prevent screen flipping. Camera movement is performed in response to the keyboard’s WASD keys. The O key is bound to enable orthographic camera projection, a utility provided by the glm library.

**Custom Functions for Modularity and Organization**

Abstracting common functionality away is useful for organizing the codebase and decreasing cognitive load during development. To do this, primitives are coupled into objects which are abstracted into a function for rendering that object. Thus, to use the full object, only one function call needs to be made. There are functions for various high-level operations that are performed, such as loading textures into memory, setting up scene lights, and defining object materials. These functions are then used to set values in the shader manager. In short, there is a layer of high-level functions that are used in the main parts of the code that are worked on, and the details are abstracted away. These high-level functions call lower-level functions which perform the low-level operations, the details of which are typically not relevant to the developer, until something breaks. In reality, there are many layers of this. Variables that are shared between different parts of the code may either be lifted up into a higher scope or passed around to functions as parameters. In this project, I chose to lift the `bConstrainWorld` and `bShowLightGizmo` variables into a higher scope, making the functions impure. This may not be the best option for more complicated applications, where pure functions are typically a better design decision.