**CS-330 Final Project**

**Imitating Life Through Code**

Damean Murphy-Short

Southern New Hampshire University

June 18, 2024

For this project, Triangle and Cube Studios was contracted by our client to recreate a 3D environment from a 2D representation of a scene. The image to be recreated is pictured below:



To do so, various primitive shapes, simple textures, and lighting were used in combination to provide an accurate draft of work. The purpose of this paper is to represent the design choices that were made, an explanation of the functionality, and the re-application of this code to a more finished project.

We will begin by describing the navigation implemented in the 3D program. There are a total of 5 dimensions of movement: the X, Y, and Z axes, in addition to pitch and yaw control. There are also two keyboard layouts present, with the possibility to add more, or to change the layout during runtime (this will be discussed later). Movement in the x axis is controlled by the ‘a’ and ‘d’ keys in the default layout and by the ‘r’ and ‘t’ keys in the colemak layout. Similarly, the y axis can be navigated using ‘q’ and ‘e’ by default and ‘space’ and ‘a’ with colemak controls. The z axis movement is controlled with ‘w’ and ‘s’ or ‘f’ and ‘s,’ respectively. Switching between these layouts can be done by pressing ‘q’ for QWERTY or ‘c’ for Colemak. The speed of these movements can be adjusted by using horizontal scroll on a pointing device.

Pitch and yaw are both controlled by pointer movement. X-axis movement controls yaw and Y-axis movement controls yaw. Similarly to panning speed, the camera speed can be changed by vertical scroll.

Now that we are able to navigate and fully examine the scene, we can discuss the rationale for the various objects represented. Only the most prominent objects were featured in the recreation: the displays, the keyboard and trackball, the desk itself, and the pc tower. This decision was made to avoid creating a number of small and complex objects that hold little importance to creating a basic draft of our client’s work. Some shapes that are more round in reality than in the the model are flattened to reduce the amount of vertices needed to represent them, and therefore the time spent creating and rendering them. Despite this, the primitive shape most close to the original shape of the object was always chosen, including slightly more complex shapes like spheres and cylinders. Some objects required a combination of multiple primitive shapes, and an effort was made to reduce this number without losing the intended form.

Lastly, we shall take a look into the code used for the program and examine how it can be used for other projects. The in-house tools of Triangle and Cube Studios already provided the necessary functionality to render basic shapes, import and use shaders, utilize imported textures, and utilize basic lighting. External dependencies include GLFW, which is used to create GL windows and interpret input, and STBI, which is used to import images as textures. Our modifications to this code accomplish several things: rendering the appropriate objects, using input to navigate, and adding complexity to the lighting shaders.

For the object rendering portions in the Source/SceneManager.cpp file, an effort was made to offset most complex objects by fixed values so that they could easily be extended or changed in the future. For example, the drawTower() function includes a float panelScale by which the scale and positions of all the tower panels are adjusted.

To implement 3D navigation, various global keys were declared to abstract the keyboard layout. In Source/ViewManager.cpp, there is an int gkey\_forward to represent forward input, and so on, so that any key can be defined at runtime as the control to any pre-declared input. This idea could be extended to load keyboard layouts from a configuration file, as is often done in 3D games and CAD software.

The complexity added to the fragment shader is accomplished by utilizing already-existing functions in the shader manager to set values for newly-created uniforms. The existing texture system is augmented by using a new bool bUseTexture2 and the mix() function of glsl to allow for the blending of two textures on the same surface. Similarly, a bUseSpecularMap bool is used to determine is a texture should be used as a map that controls how specular light interacts with a surface. Other small changes are made to fix various bugs, such as uninitialized lightSources affecting the resultant fragments. This particular bug was fixed by setting a bool to determine if a light is enabled and skip calculation of its contribution to a fragment if it is not. This will allow for an increase in the TOTAL\_LIGHTS constant without introducing more undefined behaviour.

In reflection, the sum of these changes provides a good starting point for work with our client in the recreation of various objects in 3D. In the future, we can better represent complex objects by using more textures, defining more realistic values for the object materials, and using custom shape meshes with higher vertex counts that more closely follow the form of the real objects.