In this project, I demonstrated a 3D rendered scene that I built using OpenGL. In the scene it features multiple different textured objects with multiple light sources using the Phong Lighting model. In this scene I used three different lighting components including ambient, diffuse, and specular. The scene is controlled by the user using a keyboard and mouse. Using the mouse keys, you can pan up or down, left or right and control forward and backward zoom. I also added controls for the mouse too. You can use the mouse to look around while the scroll wheel adjusts your camera movement speed.

For my scene I chose a 2D image that met the clients’ requirements. This includes a complex object that consists of two or more simple objects together and 3 additional objects. This scene also highlights some of OpenGL’s’ elements such as camera movement, textures, and lighting. The different objects and textures in this scene show how different objects and materials react under Phong lighting. I was able to control what was going on in the scene by using different functions such as translate, rotate, and scale to give the scene a realistic positioning and orient them in a way that matched the 2D image. From there I could load textures from the utility folder and set them up using the UV coordinates to ensure they were mapped properly. I created two lights for this scene and gave each of them ambient color, diffuse color, and specular color values to give the scene realistic lighting and some color. From there I created an object material structure that sets lighting properties to each of the textures to further enhance the realism in the scene by controlling the ambient strength, specular color, and shininess depending on the texture material.

I added user controls that allow the user to navigate the scene using their mouse and keyboard as input devices. This controls the virtual cameras position using the “WASD” Keys to pan left and right or zoom in or out. They can also use the “QE” keys to pan up or down. There is a camera class that maintains the position, direction, and orientation of the camera as the user moves about the virtual space. The user can also change their views between a 3D perspective view or a 2D orthographic view using the “PO” keys. This gives the user an option on how they want to view or navigate the space. The camera’s view is calculated per frame using the “lookat()” function in the glm library to generate a view matrix. This transforms the environment based on the users’ positioning. The keyboard input is handled using GLFW key callback system. So, when the user presses one of the keys mentioned above it changes the camera’s view. I also included mouse controls to give the user the option to look around from a single place and adjust their camera speed. For this I implemented another callback system that tracks the change in cursor placement and updates the camera’s yaw and pitch angles to simulate a first-person experience. I added adjustments to the camera speed using the mouse wheel so when a user scrolls up or down, they can increase or decrease their movement speed allowing them to navigate at their own pace.

I organized my program by using managers. This includes a SceneManager, ViewManager, and a ShaderManager to keep the code modular and easy to follow. By doing this it allows portions of the program to manage specific portions within the rendering pipeline. For example, my SceneManager handles the creation, transformation, and rending of my objects in the scene. Each object can be created and added to this scene using the same method calls which helps me stay organized and reduces redundancy. Another example is my view manager. It is used to manage the view and projection matrix updates such as keyboard strokes or mouse movements. These are reusable across other projects or scenes that require camera control. For my Shader manager, this has functions to load and apply shaders. Using a modular approach this allowed me to easily change and load shaders without having to rewrite code. In addition to these managers, I also implemented functions for material and lighting setups. For example, “SetMaterialProeprties()” takes input for special lighting effects such as ambient, diffuse, and specular values and sends them to the shader so that I could set material settings for multiple different objects without having to duplicate code. I also used a “ConfigureLight()” function to standardize light positions, color, and strength to shader in the same way as setting the material.