Richard Paredes
UHID: 1492535
COSC4370

# Homework 3 Report

The purpose of this assignment was to provide knowledge reinforcement for how viewing works in 3-dimensional space using the OpenGL library. The main task involved implementing a Phong shader that renders a 3D model. In other words, a fragment shader and a vertex shader needed to be programmed to return the desired results for a Phong model, specifically a cube.

After setting up the dependencies and running the project, the first objective was to get a screen rendering. To do this, the view matrix needed to be setup in Camera.h. The view matrix is simply a transformation matrix that translates the world space relative to the camera's view. In this way, all vertices become relevant to the camera. Fortunately, the GLM library has a useful function for computing this matrix: glm::lookAt(vec3 CameraPosition, vec3 CameraTarget, vec3 upVector). Using this function made returning the view matrix a breeze.

However, to finish the process of viewing in 3-dimensional space, a projection matrix was still needed. The projection matrix accounts for the z-transformations, placing objects closer/farther from the camera based on their distance relative to the camera. To do this, the projection matrix transforms the objects in the viewing space so that they provide the illusion of a particular perspective. Again, the GLM library made this process painless with the following function: glm::perspective(float FieldOfView, float AspectRatio, float NearClippingPlane, float FarClippingPlane).

Once these matrices were set, the program was able to produce a seemingly empty viewing space – a result from the lack of a shader. To implement the Phong shader, an introduction to the GLSL was needed. After understanding how the inputs, outputs, uniforms, and setting transformation values worked, the strategy for creating the shader made more sense. The vertex shader needed to compute and set the position of the shader's output (gl_Position) using the model, view, and projection matrix. Additionally, the vertex shader needed to pass information about the normal and position to the fragment shader. The normal was passed in as-is, and the fragment position was computed using the model matrix and vertex position.

The fragment shader used the input from the vertex shader, along with uniforms specifying information about the light position, view position, and colors, to simulate lighting in the viewing space. This includes features such as ambient, diffuse, and specular. Each of these components were computed and then combined to generate the color, which was the output of the fragment shader.

With both the fragment and vertex shaders implemented, the main program now rendered more than a black screen. The resulting scene produced the Phong shader output as specified in the guidelines. As expected, the model in the scene contained all the components specified in the shader: specular, diffuse, and ambient lighting. The results were nearly identical to the guidelines.