Homework 3: 3D Viewing and Shaders

Name: Karla Castello PSID: 2138671

March 2023

1 Problem

For this assignment, OpenGL is needed to implement 3D viewing of a cube and shade it using the Phong shading model. In order to view the cube in 3D, we need to complete the view matrix of the camera and a perspective projection matrix to view the cube in 3D space. Additionally, to be able to actually see the cube, we need to implement the ambient, diffuse, and specular lighting calculations in the fragment shader and calculate the light's position vector and the fragment's position vector in the vertex shader.

2 Method

In the Camera.h file, I used the *lookAt* matrix from *glm* to calculate the view matrix. The *lookAt* matrix takes the parameters of a position vector, target vector, and up vector in world space. Using this *lookAt* matrix as the view matrix transforms all world coordinates to our defined view space. In the main.cpp file, we will create our camera transformations. To set up the projection matrix, I used the *perspective* function from the *glm* library. This function takes the following parameters: $fieldofview$, *aspect*, *near*, and far. The field of view is the width in radians of the perspective frustum. The aspect parameter is a float ratio of the width/height of the window. The near parameter is a float that specifies the near plane of the viewing frustum. The far parameter is a float that specifies the far plane of the viewing frustum. In the phong.vs file, the projection, view, and model matrices were multiplied with the position vector to find the light's position vector. Only the model matrix was multiplied with the position vector to find the fragment's position matrix. Lastly in the phong.frag file, I used a constant float and the vector of the light's color for the ambient light. Next, I used the $Normal$ vector, the light positions, the fragment positions, and the light colors for diffuse lighting. Lastly, I used the view positions, the fragment positions, the light directions and the reflect directions, and the light color for the specular lighting. To add the shading to the object, I added the three different types of lighting and multiplied it by the object's color.

3 Implementation

3.1 View Matrix

To make the view matrix, I used the *glm* library and the *lookAt* matrix. This *lookAt* matrix is essentially an inverted rotation matrix and translation matrix. In the position parameter, I used the $Position$ vector of the camera. In the target vector parameter, I used the $Position$ vector of the camera + the $Front$ vector of the camera. The $Front$ vector is always being updated using the Euler angles when moving the space around with the mouse, therefore the camera is always facing the target direction. Lastly, for the up vector parameter, I used the $WorldUp$ vector.

3.2 Projection Matrix

To set up the projection matrix, I used the *perspective* function in the *glm* library. I used a field of view of 60 degrees, which is of medium size. The aspect ratio was the width of the window size/ height of the window size (800/600). The near plane is 0.1 therefore all coordinates in front of this near plane will not be drawn. Similarly, the far plane is 1000.0 therefore all coordinates behind the far plane will not be drawn. This matrix, as well as the view matrix, is passed through the shader.

3.3 Phong Lighting Model

There are 3 parts to the Phong Lighting Model. It is easier to first set the ambient lighting for the object to always have some type of color, then the diffuse lighting for brightness and specular lighting for shininess. At the end of calculating all three parts, we add the resulting vectors together and multiply them by the object's color to get the desired lighting of the object.

3.3.1 Ambient Lighting

The ambient lighting of the object was made with an ambient strength of 0.1 and the $lightColor$ vector multiplied together. This made the cube a very dark orange-red color all around which would ultimately be the shadowy parts of the object.

3.3.2 Diffuse Lighting

We first calculate the direction vector between the light source and the fragment's position. This was done by subtracting the light's position vector and the fragment's position vector. In order to make sure all vectors end up as unit vectors, we normalize the $Normal$ vector and the direction vectors that we just calculated. Then, we calculate the diffuse impact of the light by taking the dot product between the normalized $Normal$ vectors and the normalized $lightDir$ vectors. Since the product could end up negative, we use the max function to

return the highest of both parameters. Lastly, we multiply the diffuse impact with the $lightColor$ to get the diffuse component vectors.

3.3.3 Specular Lighting

Lastly, we can calculate the specular lighting. I used a $specularStrength$ variable of 0.5 for a mild impact on the color. Then, we calculate the view direction vector by subtracting the view positions and the fragment positions and normalizing the result. We also calculate the reflect vector by using the $reflect$ function with the first argument being $-lightDir$ (it is negative because the vector must point from the light source to the fragment's position) and the second parameter is the normal vector. Using the view direction and the reflect direction, we calculate the specular component by taking the dot product between the two. Again, we use the max function to ensure there are no negative components and then raise the result to the power of 64 for a higher shininess value. Lastly, we multiply this result by the specular strength and the light's color.

4 Results