COSC 4370 – Homework 3

Name: Austin Thibodeaux PSID: 2037865

October 2022

# 1 Problem

The goal of this assignment is to render a simple cube using 3D viewing transforms and the Phong shading model. Boilerplate code is given to handle inputs and viewport display as well as setting up the cube for display, however the camera view transform, perspective projection, and shaders are left unfinished to be implemented.

# 2 Method

Four areas of the code needed to be updated to implement the requirements of this assignment:

* **GetViewMatrix function in Camera.h:** Creates a 4-dimension matrix using camera position data for the view of the camera to be used Phong.vs.
* **Projection matrix in main.cpp (line 200):** Calls glm::perspective to create a new 4-dimension matrix to be used in Phong.vs.
* **Phong.vs:** Phong vertex shader. Sets up gl\_Position with a perspective transformation chain including view and perspective matrix generated by the code implemented in the points above. Also sets up Normal and FragPos outputs using model’s normal at the vertex and position.
* **Phong.frag:** Phong fragment shader. Uses outputs of Phong.vs as well as four uniform parameters set at runtime (lightPos, viewPos, lightColor, objectColor) to create ambient, diffuse, and specular lighting.

# 3 Implementation

## 3.1 GetViewMatrix

This implementation is fairly simple using the glm::lookAt function provided by the glm library. The first argument of this function is the current transform location of the camera, the second is the transform of the target that the camera will look at, and the third argument is the direction of up.

The camera’s look direction should be controlled with the mouse and the camera’s position should be controlled by the keyboard WASD keys. Thus, the view transform must always look at a point in space directly in front of the camera’s front vector. Thankfully, the Camera class defined in Camera.h has two members: Position and Front. Position is the camera’s current position in world space, and Front is the vector pointing in the direction of the front of the camera.

To implement the view matrix, the Position can be used as the first argument of lookAt. The second argument needs to be a place in *world space* that is directly in front of the camera which can be accomplished by adding Position and Front together and using the result as the second argument. The third argument uses the WorldUp member variable of the Camera which is the world direction of up.

This function then returns a 4-dimensional matrix representing the view of the camera.

## 3.2 Projection matrix in main.cpp (line 200)

This implementation is very simple as it makes use of the glm library’s built-in perspective function. The function creates a new 4-dimension matrix for the perspective projection. The function has parameters for the field of view, aspect ratio, near plane and far plane. The resulting matrix is then stored into a variable called projection and then passed to the vertex shader.

## 3.3 Phong.vs

This is the vertex shader for the Phong implementation. This is where the model, view, and projection matrices are multiplied together along with the position of each vertex of the model. This part of the shader creates a flattened, perspective correct representation of the model to be viewed on a 2D screen. This matrix is then set to gl\_Position.

This shader also uses the normal of each vertex and outputs a non-uniform scale corrected normal vector to the Normal variable. It also outputs the position of the vertex in world space via the FragPos variable by multiplying the model’s position with the vertex position. These outputted variables are used as inputs in the Phong.frag file.

## 3.3 Phong.frag

This is the fragment shader where the lighting calculations are done. This shader consists of three parts which are all added together at the end: Ambient, Diffuse, and Specular.

The ambient color is simply the color of the light source multiplied by an ambientStrength constant. This will determine the color of each pixel that is not illuminated by a light source. At first, everything is colored the ambient color, but as Diffuse and Specular are computed they are added upon ambient to simulate surfaces being lit.

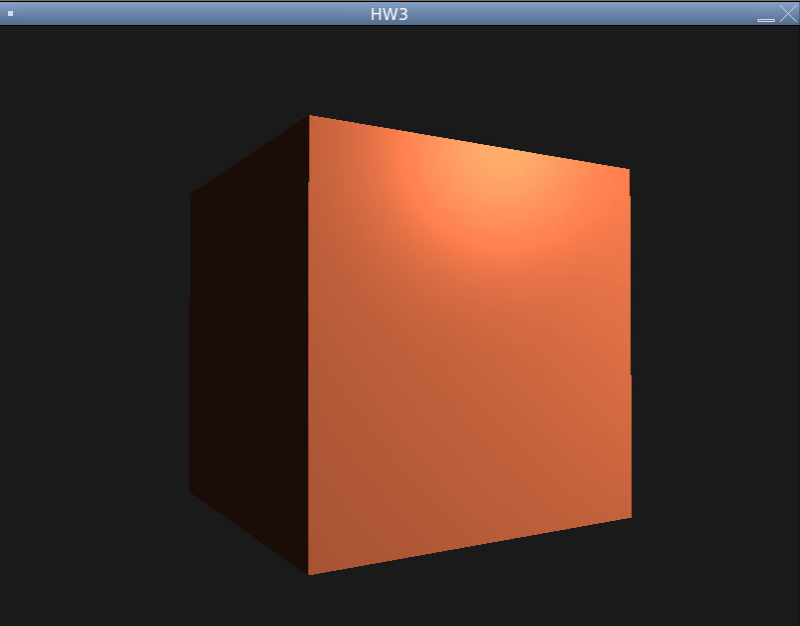
The diffuse color uses the difference of the angle of the surface normal and the angle of the light source. The closer the angle is to zero, the brighter the light on the surface. This is accomplished by calculating the dot product of the normal and light direction and multiplying the result with the light color. This results in the areas facing the light source being the brightest, and the areas facing away being the darkest (ambient color).

Finally, the specular color works in a similar way to diffuse, except the strength of the specular reflection is determined by the angle of the reflected ray of light off the surface of the fragment and the angle of the camera. The difference of the reflected ray of light and the camera is computed with a dot product so that the more parallel the camera is to the reflected light vector, the brighter the reflection is, and the less parallel they are the less powerful the reflection is. The radius of the specular reflection and its strength are determined by the specularComponent and specularStrength variables.

At the end, all three lighting calculations are added together and then multiplied by the object color. This creates the final Phong lighting model.

# 4 Results

When the program is run using script.sh (or run.sh that I created to run it outside of Replit), a window will appear showing a 3D cube that is shaded using the Phong shading model. The camera can be moved around using the mouse and keyboard to show how the cube has direct and ambient lighting as well as the specular reflection.



# 6 References

Some of the source code in this assignment are derived from <https://learnopengl.com>. Page <https://learnopengl.com/Getting-started/Coordinate-Systems> was used as a guide for the view and perspective transformation implementation and <https://learnopengl.com/Lighting/Basic-Lighting> was used as a guide for the Phong shading model implementation.

# 5 Known Bugs

Unfortunately, there is a bug that I have come across and don’t know how to fix. When running this program, a segmentation fault can happen and cause the program to crash after displaying the first frame. It seems to work fine on Replit, so I would run it on there instead.