COSC 4370 – HW2

Joseph Jophy

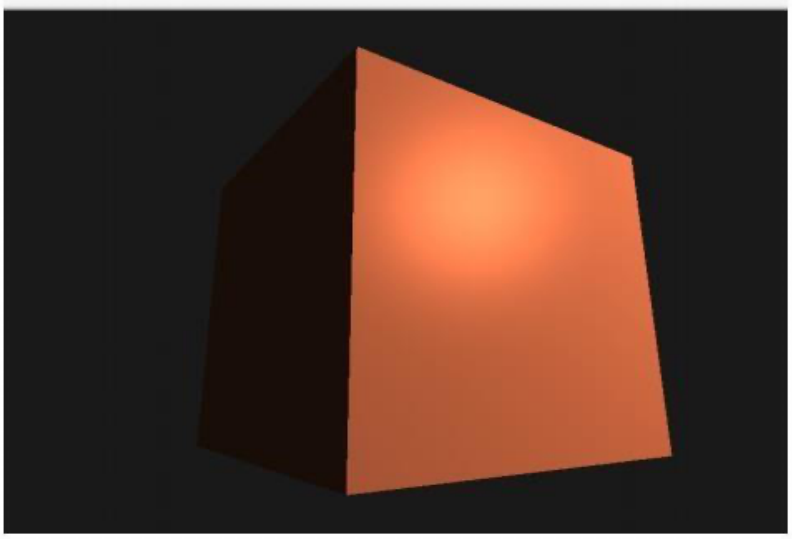
UHID#: 2048762

March 23, 2023

1 Problem

The goal of this assignment is to implement the 3D viewing and Phong shading model.

We will do this by trying to reproduce this image:



2 Method

We will first need to complete the GetViewMatrix() function in Camera.h and the projection matrix in main.cpp, which will allow us to view the object from the camera.

The vertex and fragment shaders for the Phong model will need to be implemented to shade a simple cube.

We will go into further details for each of these functions in the implementation section.

3 Implementation

ProjectionMatrix()

projection = glm::perspective(glm::radians(camera.Zoom), float(w) / float(h), 0.5f, 50.0f);

This code computes a view matrix for a camera in a 3D scene.

GetViewMatrix()

glm::lookAt(Position, Position + Front, Up) calculates a view matrix that transforms all objects in the 3D scene relative to the camera's position and orientation.

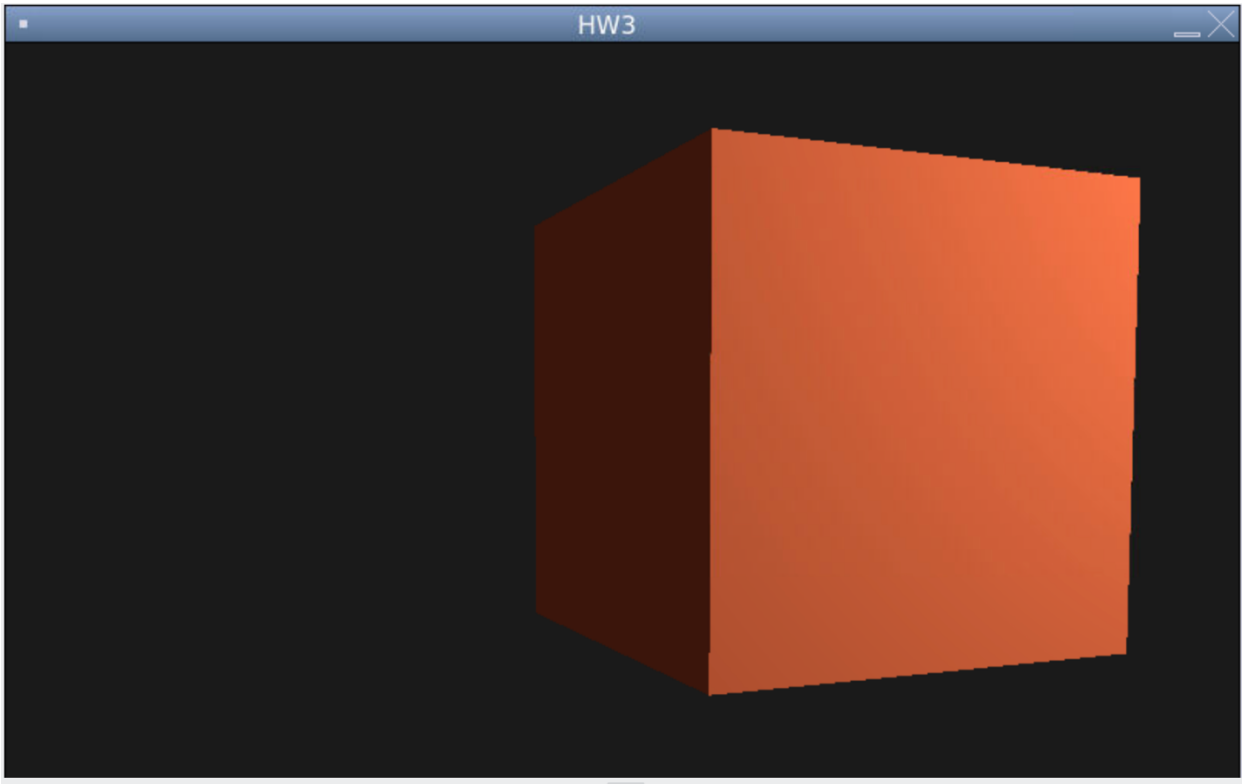
Phong.frag

This function implements the lighting equation for a fragment shader . It calculates the specular and diffuse components of the lighting equation based on the position and properties of a light source, the position of the camera (view), and the surface normal and color of the object being rendered. It then combines these components with a small ambient factor to compute the final color of the fragment. The diffuse component is based on the amount of light that is scattered by the surface of the object, while the specular component is based on the amount of light that is reflected by the surface. The glossiness value determines the size of the specular highlight, and the intensity of the specular component is multiplied by a constant value to adjust the strength of the reflection. Overall, this shader aims to produce a realistic lighting effect on the rendered object.

Phong.vs

Vertex shaders are programs that run on the GPU and process individual vertices of a 3D object. In this particular code, the vertex shader takes in vertex position and normal as inputs and calculates the transformed position of the vertex by applying model, view, and projection matrices. It also passes the normal and world position of the vertex to the fragment shader for further processing. The output of this vertex shader is the transformed position of the vertex in the clip space coordinates that will be used for rendering the object on the screen.

4 Results


HW3