COSC4370 HW3 – 3D Viewing and Shaders

In order to implement the assignment application we first separate steps to do as below:

1- Implementing the `GetViewMatrix()` function in `Camera.h`.
2- Implementing projection matrix in `main.cpp`.
3- Implementing `vertex` & `fragment` shaders for one color.
4- Implementing `vertex` & `fragment` shaders for the Phong model.

We will discuss all these steps in the following subsections:

Implementing the `GetViewMatrix` function

Actually, we are going to use lookAt() function to return the view matrix. For which it is required to pass cameraPostion, cameraFront and cameraUp as paramters that we can enter the middle argument using addition of cameraPos + cameraFront in order to keep the camera always looking at the target direction.

```
glm::mat4 GetViewMatrix(){
      return glm::lookAt(Position, Position + Front, Up);
}
```

Implementing projection matrix

In this section we need to create the perspective projection matrix using perspective function which receives camera zoom as field of view, screen width and height as aspect ratio of the scene, near and farness parameters.

```
glm::mat4 projection = glm::perspective(camera.Zoom,
(GLfloat)WIDTH / (GLfloat)HEIGHT, 0.1f, 100.0f);
```

Implementing vertex & fragment shaders for one color

In this step we need to complete vertex and fragment classes in order to add shaders for one color for which we have considered red color. In the vertex class we only need to set the position of the cube using projection matrix, view and model. Also for the fragment class we can define the color vector using the RGB format R=1, G=0 and B=0 which will reflect red color.

The output of application is demonstrated in the image 1.

- `phong.vs`

```
gl_Position = projection * view *  model * vec4(position,
1.0f);
```

- `Phong.frag`

```
color = vec4(vec3(1.f,0.f,0.f), 1.0f);   // red cube
```

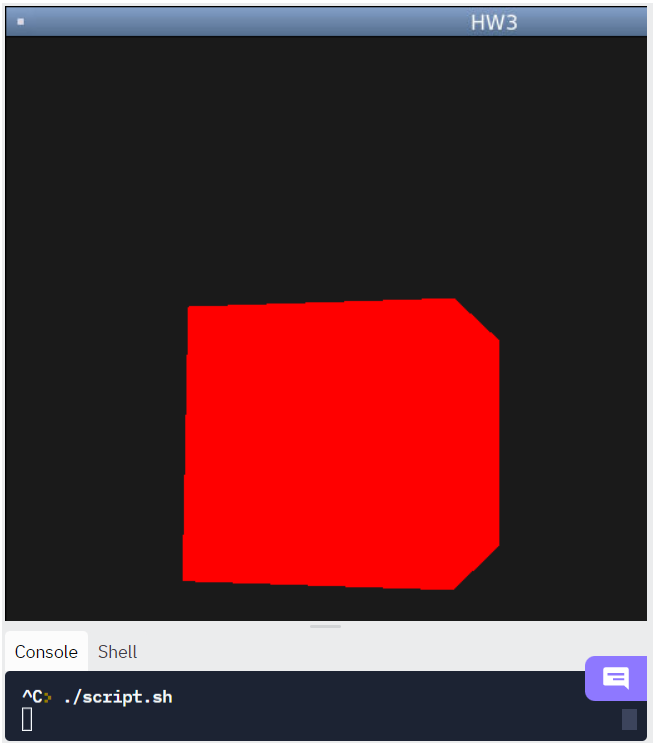

Image 1: One color cube

Implementing vertex & fragment shaders for the Phong model

This is the final step of the application to see the output. In fact, in order to build Phong model we require to create three components as ambient, diffuse and specular lighting.

After specifying positon matrix by model, view and projection matrixes in the previous steps, we can define FragmentPostion, LightPosition vectors and Normal matrix in the Phong vertex class. Then, receiving these vectors we can build our components in the fragment class.

Image two is the output of the application after applying Phong model components.

- Phong.vs

```
gl_Position = projection * view *  model * vec4(position, 1.0f);
FragPos = vec3(view * model * vec4(position, 1.0));
Normal = mat3(transpose(inverse(view * model))) * normal;
LightPos = vec3(view * vec4(lightPos, 1.0));
```

- Phong.frag

```
vec3 norm = normalize(Normal); // Diffuse
vec3 lightDir = normalize(LightPos - FragPos);
float diff = max(dot(norm, lightDir), 0.0);
vec3 diffuse = diff * lightColor;

vec3 viewDir = normalize(-FragPos);  //Specular
vec3 reflectDir = reflect(-lightDir, norm);
float spec = pow(max(dot(viewDir, reflectDir), 0.0), 32);
vec3 specular = spec * lightColor;

// 0.1 Ambient + 1.0 Diffuse + 0.5 Specular
vec3 combinedPhong = ((0.1 *lightColor) + (1.0 * diffuse) + (0.5 * specular)) * objectColor;
color = vec4(combinedPhong, 1.0);
```

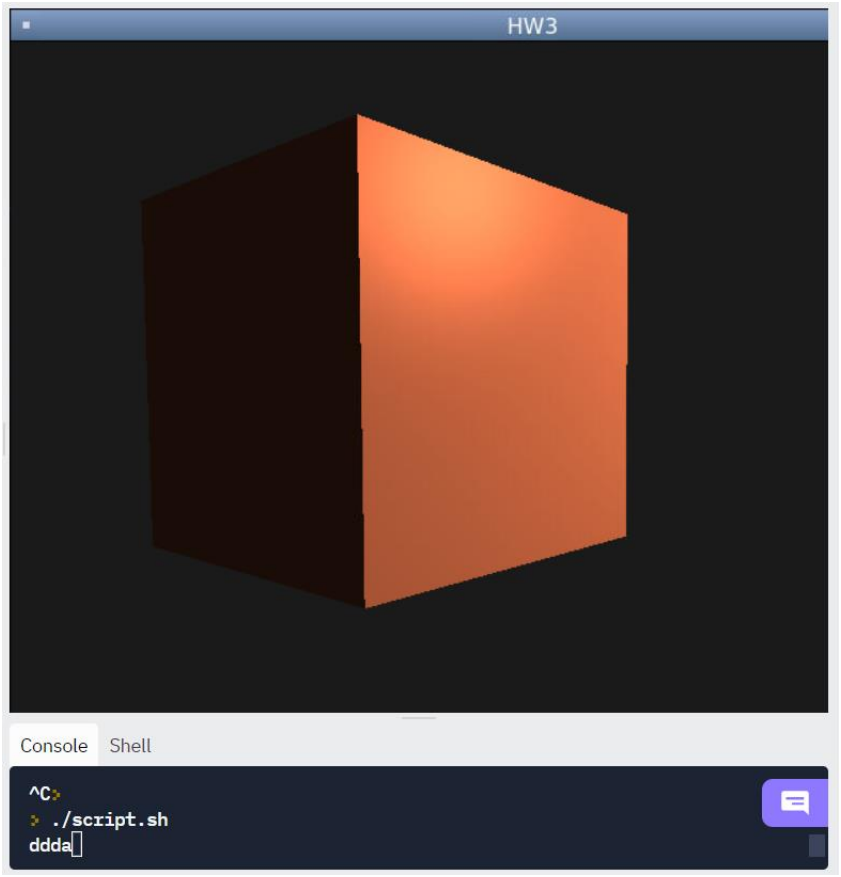

Image 1: Phong model for the cube