homework6

Basic

1. 实现Phong光照模型:
   a. 场景中绘制一个cube
   b. 自己写shader实现两种shading: Phong Shading 和 Gouraud Shading，并解释两种shading的实现原理
   c. 合理设置视点、光照位置、光照颜色等参数，使光照效果明显显示
2. 使用GUI，使参数可调节，效果实时更改:
   a. GUI里可以切换两种shading
   b. 使用如进度条这样的控件，使ambient因子、diffuse因子、specular因子、反光度等参数可调节，光照效 果实时更改

实现思路

- Cube的实现与以前大体一致，不过这次作业中不把顶点颜色传进去，而是将顶点位置的法向量作为顶点数据传入，从而方便shader内部计算。

```
float vertices[] = {
        -0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
        0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
        0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
        0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
        -0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
        -0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
}
```

- Phong shading实现原理

根据老师上课的教材，Phong shading 是在顶点着色器中通过插值计算每个位置的法向量，然后在片段着色器中应用phong光照模型来计算光线分量，包括三个部分：环境光、漫反射光、镜面反射光。

$$I_{total} = k_a I_{ambient} + \sum_{i=1}^{\#lights} I_i \left(k_d (\mathbf{n} \times \mathbf{l_i}) + k_s (\mathbf{v} \times \mathbf{r_i})^{n_{shiny}} \right)$$

具体代码如下:

```
// phong.vs
void main()
{
    gl_Position = projection * view * model * vec4(aPos, 1.0);
    FragPos = vec3(model * vec4(aPos, 1.0));
```

```
    Normal = mat3(transpose(inverse(model))) * aNormal;
}

// phong.fs
void main()
{
    vec3 ambient = ambientStrength * lightColor;
    vec3 norm = normalize(Normal);
    vec3 lightDir = normalize(lightPos - FragPos);
    float diff = max(dot(norm, lightDir), 0.0);
    vec3 diffuse = diff * lightColor;

    vec3 viewDir = normalize(viewPos - FragPos);
    vec3 reflectDir = reflect(-lightDir, norm);
    float spec = pow(max(dot(viewDir, reflectDir), 0.0), shininess);
    vec3 specular = specularStrength * spec * lightColor;

    vec3 result = (ambient + diffuse + specular) * objectColor;
    FragColor = vec4(result, 1.0);
}
```

- Gouraud shading 实现原理

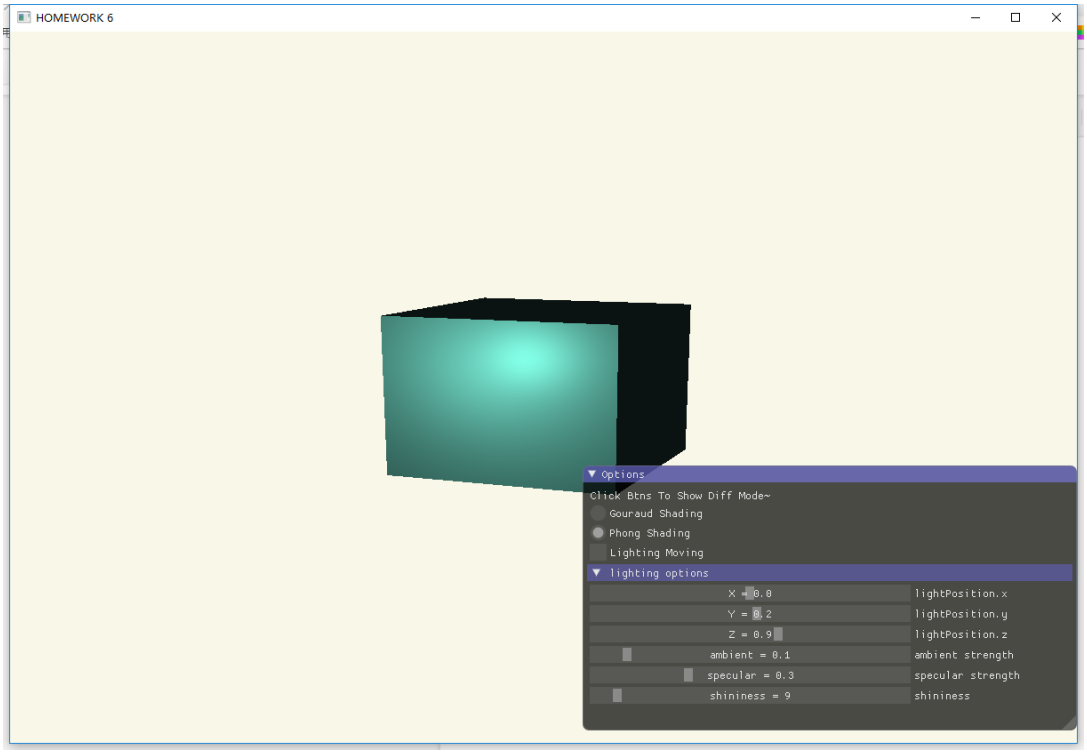
Gouraud shading 和 phong shading 的区别在于插值的内容不同。Gouraud shading是在顶点着色器中根据设置的顶点法向量计算光模型，然后在片段着色器中进行插值来得到每一个位置的颜色。因此，Gouraud shading是在顶点着色器中实现计算光照逻辑。

代码如下:

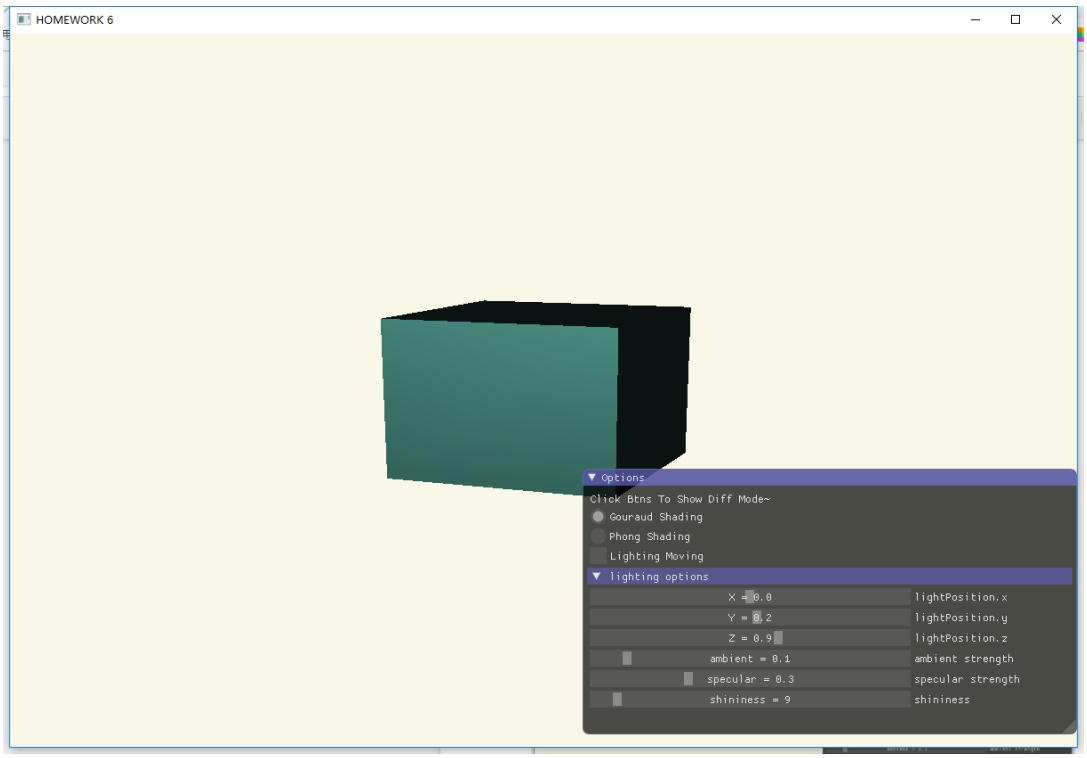
```
// gouraud.VS
void main()
{
    gl_Position = projection * view * model * vec4(aPos, 1.0);
    vec3 Position = vec3(model * vec4(aPos, 1.0));
    vec3 Normal = mat3(transpose(inverse(model))) * aNormal;
    vec3 ambient = ambientStrength * lightColor;
    vec3 norm = normalize(Normal);
    vec3 lightDir = normalize(lightPos - Position);
    float diff = max(dot(norm, lightDir), 0.0);
    vec3 diffuse = diff * lightColor;

    vec3 viewDir = normalize(viewPos - Position);
    vec3 reflectDir = reflect(-lightDir, norm);
    float spec = pow(max(dot(viewDir, reflectDir), 0.0), shininess);
    vec3 specular = specularStrength * spec * lightColor;
    result = (ambient + diffuse + specular) * objectColor;
}
```

运行效果

Phong Shading：



Gouraud Shading：



Bonus

使光源在场景中来回移动，光照效果实时更改。

实现思路：
在每一帧中使用 glfwGetTime() 函数，结合三角函数来使光源来回移动，然后将光照位置传给shader，实现光照效果实时更改。

```
if (isMovingLighting) {
    lightPosition.x = 0.5f + abs(sin(glfwGetTime())) * 1.0f;
    lightPosition.y = abs(sin(glfwGetTime() / 2.0f)) * 1.0f;
}
lightingShader.setFloat3("lightPos", glm::value_ptr(lightPosition));
```

运行效果见gif图。