计算机图形学作业 6 实验报告

16340203 谭江华

基本要求：

1. 实现 Phong 光照模型： 场景中绘制一个 cube 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 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： GUI 里可以切换两种 shading 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&算法：

1. 首先要单独绘制一个 cube，之前已经做过类似的操作了。另外设置有一个小的 cube 作为光源，给这个光源 cube 另外写一个顶点着色器和片段着色器。

```
#version 330 core
layout (location = 0) in vec3 aPos;

uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;

void main()
{
    gl_Position = projection * view * model * vec4(aPos, 1.0);
}
```

```
#version 330 core
out vec4 FragColor;

void main()
{
    FragColor = vec4(1.0); // set alle 4 vector values to 1.0
}
```

并且设置小 cube 的位置为光源的位置

```
// also draw the lamp object
lampShader.use();
lampShader.setMat4("projection", projection);
lampShader.setMat4("view", view);
model = glm::mat4(1.0f);
model = glm::translate(model, lightPos);
model = glm::scale(model, glm::vec3(0.2f)); // a smaller cube
lampShader.setMat4("model", model);
```

2. 之后在着色器中实现 Phong Shading，包括环境光照，漫反射以及镜面光照。首先实现环境光照，我们用光的颜色乘以一个很小的常量环境因子，再乘以物体的颜色，然后将最终结果作为片段的颜色。

```
// ambient
//float ambientStrength = 0.1;
vec3 ambient = ambientStrength * lightColor;
```

接着实现漫反射，这里就需要输入 cube 六个面对应的 6 个法向量以及定向的光线，所以我们向顶点数组添加了额外的数据，并且应该更新光照的顶点着色器。

```
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;
```

输入定点坐标以及法向量。

```
float vertices[] = {
    // 顶点坐标           // 法向量
    -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,

    -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,
```

然后在片段着色器中进行计算，lightDir 是计算光源和片段位置之间的方向向量，diff 是对 norm（法向量）和 lightDir 向量进行点乘，计算光源对当前片段实际的漫发射影响

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

最后实现镜面光照，定义一个镜面强度(Specular Intensity)变量以及高光的反光度（shininess，初始值为 32）

```
// specular
//float specularStrength = 0.5;
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;
```

最后综合三种光照乘以原来物体的颜色即可。

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

3. 另外还需要实现 Gouraud 模型，它与 Phong 模型不同之处在于它是在顶点着色器中对光照进行处理的，通过对顶点的赋值来决定像素的颜色值。具体的思路是计算顶点的法向量，决定顶点的光照颜色，然后根据多边形上各点距顶点的距离进行插值，从而绘制多边形上各点投影对应的像素。所以需要重新编写一个顶点着色器来实现 Gouraud 光照模型。

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

// gouraud shading
// ------------------------
vec3 Position = vec3(model * vec4(aPos, 1.0));
vec3 Normal = mat3(transpose(inverse(model))) * aNormal;

// ambient
// float ambientStrength = 0.1;
vec3 ambient = ambientStrength * lightColor;

// diffuse
vec3 norm = normalize(Normal);
vec3 lightDir = normalize(lightPos - Position);
float diff = max(dot(norm, lightDir), 0.0);
vec3 diffuse = diff * lightColor;

// specular
// float specularStrength = 1.0; // this is set higher to better show the effect of Gouraud shading
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;

LightingColor = ambient + diffuse + specular;
```

4. 实验还要求使用 GUI 可以切换两种 Shading 并且调节参数。

切换 Shading 的实现较为简单，加入两个 checkbox 供选择即可。

```
// 创建GUI菜单栏
ImGui_ImplGlfwGL3_NewFrame();
ImGui::SetWindowFontScale(1.4);
ImGui::Text("Choose Shading");
ImGui::Checkbox("Show Phong Shading", &is_phong_shading);
ImGui::Checkbox("Show Gouraud Shading", &is_gouraud_shading);
```

选择了 Phong Shading 或者 Gouraud Shading 之后可以调节三个参数，分别是 ambientStrength（ambient 因子），specularStrength（specular 因子）和 shininess（反光度）。之后将他们输入到着色器中。

```
ImGui::SliderFloat("ambient", &ambientStrength, 0.0f, 1.0f);
ImGui::SliderFloat("specular", &specularStrength, 0.0f, 1.0f);
ImGui::SliderInt("shininess", &shininess, 0, 32);
Gouraud_lightingShader.use();
Gouraud_lightingShader.setVec3("lightPos", lightPos);
Gouraud_lightingShader.setVec3("viewPos", camera);
Gouraud_lightingShader.setVec3("objectColor", 1.0f, 0.5f, 0.31f);
Gouraud_lightingShader.setVec3("lightColor", 1.0f, 1.0f, 1.0f);
Gouraud_lightingShader.setMat4("model", model);
Gouraud_lightingShader.setMat4("view", view);
Gouraud_lightingShader.setMat4("projection", projection);
Gouraud_lightingShader.setInt("shininess", shininess);
Gouraud_lightingShader.setFloat("ambientStrength", ambientStrength);
Gouraud_lightingShader.setFloat("specularStrength", specularStrength);
glBindVertexArray(cubeVAO);
glDrawArrays(GL_TRIANGLES, 0, 36);
```

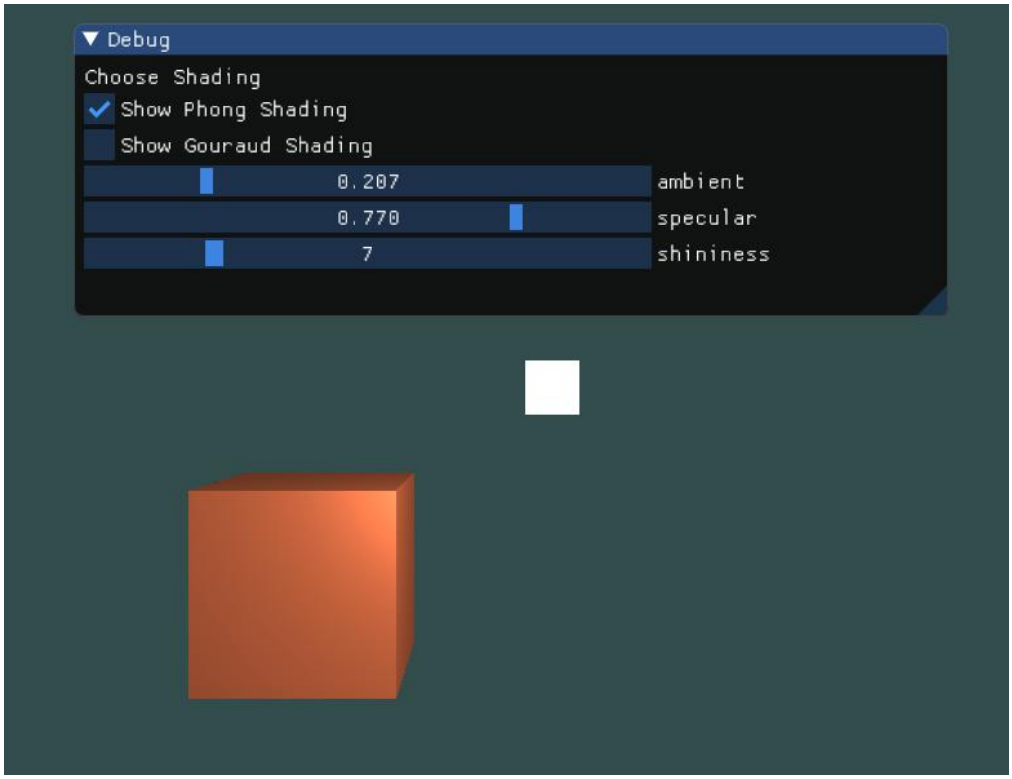
5. 解释两种 Shading 的实现原理：

Phong 光照明模型是经验模型，在 Phong 光照模型中三个主要的光照分量是环境光照（ambient）、漫反射（diffuse）和镜面反射（specular）。环境光照的处理较为简单，直接将影响系数与光源颜色相乘即可，漫反射是根据入射光的角度以及对应的法向量就可以计算出结果，镜面反射出了依赖于入射光和法向量之外，也依赖于观察者所处的位置，以及材质的发光常数。这里就需要我们对于观察者和物体、光线之间的相对位置进行处理。

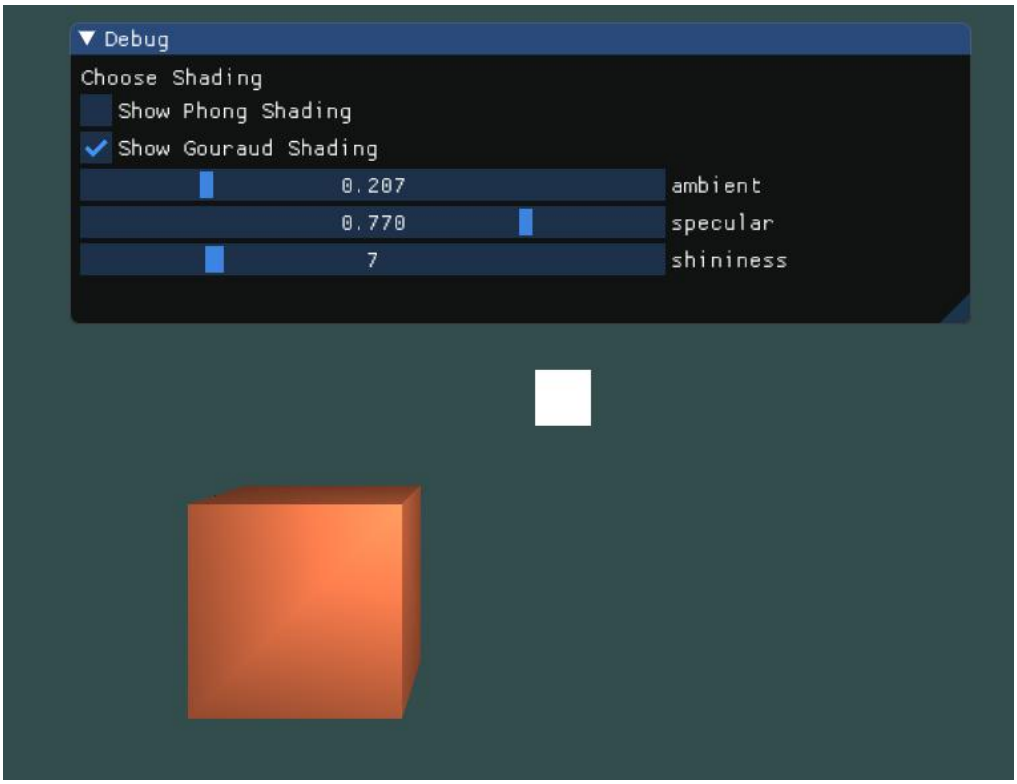
Gouraud 模型与 Phong 模型不同，是在顶点着色器中对光照进行处理。Gouraud 模型通过对顶点的赋值来决定像素的颜色值。具体的思路是计算顶点的法向量，决定顶点的光照颜色，然后根据多边形上各点距顶点的距离进行插值，从而绘制多边形上各点投影对应的像素。在对光照的处理方法上二者的方法是相似的，即 Gouraud 模型的顶点着色器中对于不同光照分量的处理与 Phong 模型的片段着色器中对于光照的处理是相似的。

最终效果如图：

Phong Shading



Gouraud Shading


Debug
Choose Shading
Show Phong Shading
Show Gouraud Shading
0.207
ambient
0.770
specular
7
shininess