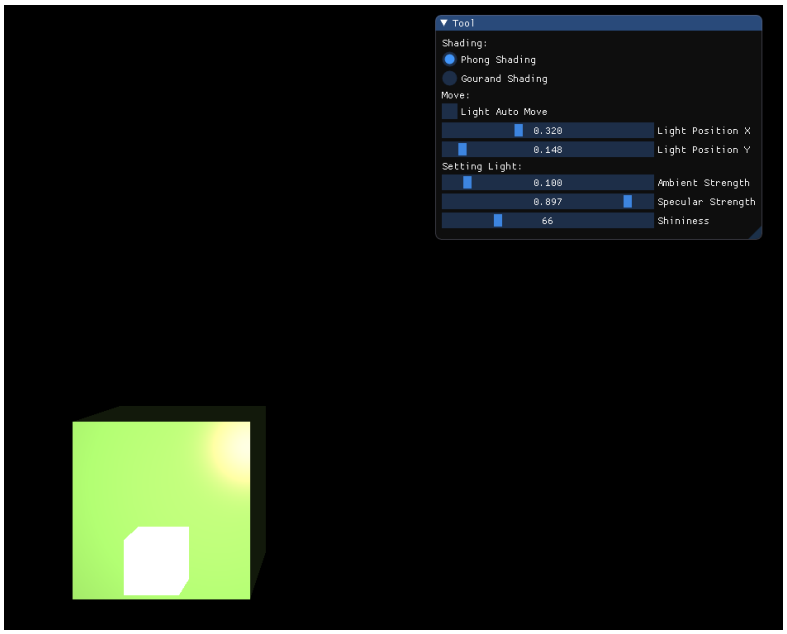
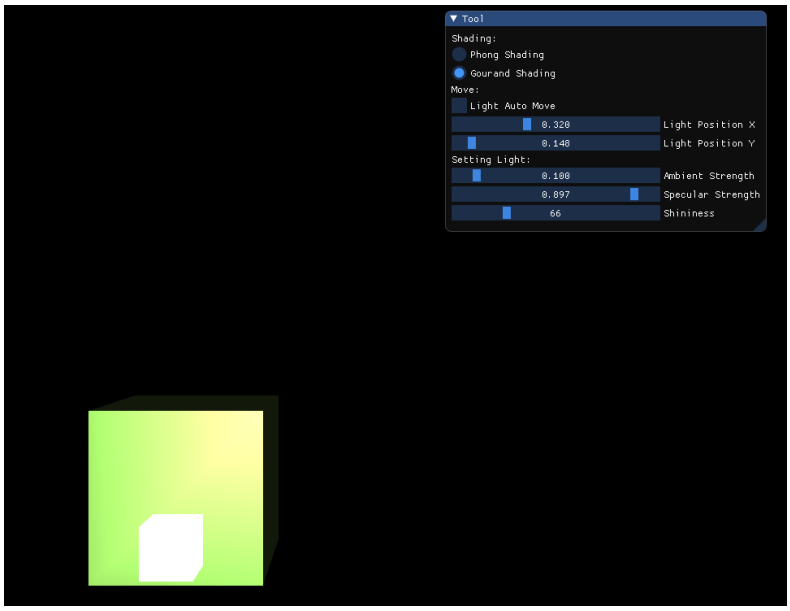
Homework 6 - Lights and Shading

一、实现结果

✧ Phong Shading



✧ Gouraud Shading



需要了解更多见 doc 文件下的演示视频

二、实现思路

1. 场景中绘制一个 cube

➢ 为这个 cube 设置每个顶点的位置，然后再设置一个世界坐标的位置，利用 model，view，projection 矩阵，就可以将一个正方体在场景中绘制。

2. 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

2.1 Phong Shading

- 冯氏光照模型主要由 3 个分量组成环境、漫反射和镜面光照。这三个分量相加然后与光源颜色相乘可以简单的创建出一个冯氏光照模型

2.1.1 环境光照

- 在世界上因为会有一些发光的星球会传到地球上，所以物体不会完全黑暗，为了模拟这个需要使用一个环境光照，永远给物体一些颜色。

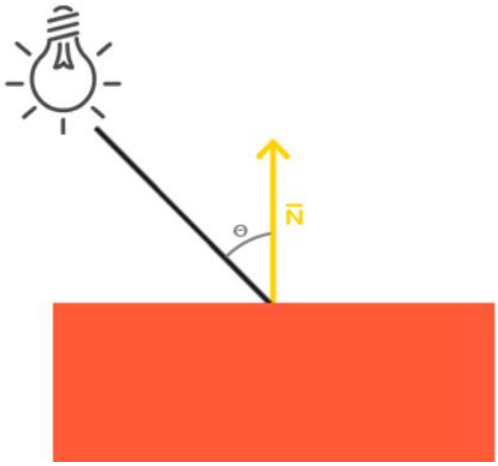
为了模拟环境光照，使用一个环境光照强度常量乘以物体的颜色，将结果作为最终物体的颜色，这样物体始终会有一些比自身颜色暗的颜色来模拟环境光照。代码如下：

```
// 环境光照强度
uniform float ambientStrength;
// 环境光照
vec3 ambient = ambientStrength * lightColor;
```

2.1.2 漫反射光照

- 在现实生活中，面对光源的一面颜色要比背对光源的一面要亮一些，所以物体上与光线越接近的片段能从光源处获得更多的亮度。这就是漫反射光照。

如果有一个光源，它所发出的光线落在物体的一个片段上。需要测量这个光线是以什么角度接触到这个片段的，为了知道光线和片段的角度，利用片段的法向量，使用点乘计算出这两个向量之间的角度。



首先计算法向量，就是垂直这个片段的单位向量，在这个程序中我是手动指定这个片段的单位向量，而不是自动计算的，作为 Phong Shading 所有光照的计算都是在片段着色器里，所以手动定义的片段法向量要传入片段着色器中。法向量只是一个方向向量，不能表达空间中的特定位置。如果模型矩阵执行了不等比缩放，顶点的改变会导致法向量不再垂直于表面了。所以还要为法向量专门定制的模型矩阵叫做法线矩阵。

- 在顶点着色器中：

```
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;
…
out vec3 Normal;
…
Normal = mat3(transpose(inverse(model))) * aNormal;  // 与法线矩阵相乘
```

第二个就是定向的光线，利用光源位置与片段的位置之间向量差的方向向量。将两个向量进行点乘，得到的结果再乘以光的颜色，得到漫反射分量。(如果两个向量之间的角度大于 90 度，点乘的结果就会变成负数，所以当为负数的时候，漫反射分量为 0)

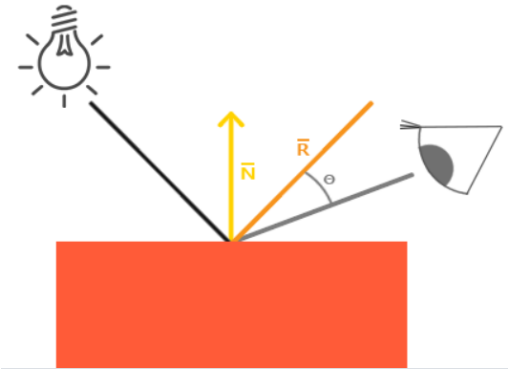
- 在片段着色器中：

```
// 镜面光照强度
in vec3 Normal;
…
uniform float specularStrength;
…
// 漫反射光照
vec3 norm = normalize(Normal);
vec3 lightDir = normalize(lightPos - FragPos);    // 光源的方向向量
float diff = max(dot(norm, lightDir), 0.0);
vec3 diffuse = diff * lightColor;
```

2.1.3 镜面光照

➢ 像现实生活中光照射在表面类似镜面的物体上，它的光可能会变成一个圆形点，这就是镜面高光，和漫反射光照一样，镜面光照也是依据光的方向向量和物体的法向量来决定的，但是它也依赖于观察方向。

需要求一个片段反射的光的反射向量与我们视角方向的角度差，如果夹角越小，那么镜面光的影响就会越大。所以需要计算视线方向与反射方向的点乘，然后取它的幂，之后再乘上镜面强度以及光源颜色。



首先需要得到观察视角向量，这里摄像机就是我们的观察眼睛，得到摄像机的坐标，通过摄像机位置与片段位置相减得到一个视角向量。

```
vec3 viewDir = normalize(viewPos - FragPos);
```

然后需要计算出反射光的反射向量，使用入射光与法线向量再使用函数 reflect()可以求出反射向量，reflect 函数要求第一个向量是从光源指向片段位置的向量，所以要将我们之前计算的光的方向取反

```
vec3 reflectDir = reflect(-lightDir, norm);
```

最后定义一个镜面强度变量，确定镜面高光的强度以及一个反光度，一个物体的反光度越高，反射光的能力越强，散射得越少，高光点就会越小。

```
float spec = pow(max(dot(viewDir, reflectDir), 0.0), shininess);
vec3 specular = specularStrength * spec * lightColor;
```

2.1.4 冯氏着色

- 冯氏着色是在片段着色器中实现光照计算，最后将各个分量相加然后乘以物体的颜色就形成了最终的光照效果。

```
vec3 result = (ambient + diffuse + specular) * objectColor;
```

2.2 Gouraud Shading

- Gouraud 着色是在顶点着色器中实现冯氏光照模型，其优势是相比片段来说，顶点要少得多，因此会更高效，对于开销大光照计算频率会变低，但是之前的作业中，为每个顶点定义了颜色，最后生成的三角形中间的颜色更像是顶点颜色根据一定权重的混合，同理只在顶点着色器中计算的颜色仅仅只是那个顶点的颜色值，最后片段的颜色值是由插值光照颜色所得来的。所以这种光照会看起来不太真实。冯氏着色能产生更平滑的光照效果。

Gouraud 着色就是在顶点着色器中将冯氏光照模型中的各个分量相加然后乘以物体的颜色就形成了最终的光照效果。

```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;    // 片段法向量

// 传入片段着色器的计算好的颜色
out vec3 LightingColor;

uniform vec3 lightPos;
uniform vec3 viewPos;
uniform vec3 lightColor;

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

// 环境光照强度
uniform float ambientStrength;
```

```
// 镜面光照强度
uniform float specularStrength;
// 反光度
uniform int shininess;
void main()
{
    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;   // 与法线矩阵相
乘

    //  环境光照
    vec3 ambient = ambientStrength * lightColor;

    // 漫反射光照
    vec3 norm = normalize(Normal);
    vec3 lightDir = normalize(lightPos - Position);          // 光源的方向向量
    float diff = max(dot(norm, lightDir), 0.0);
    vec3 diffuse = diff * lightColor;

    // 镜面光照
    vec3 viewDir = normalize(viewPos - Position);          // 视角向量
    vec3 reflectDir = reflect(-lightDir, norm);            // 反射向量
    float spec = pow(max(dot(viewDir, reflectDir), 0.0), shininess);
    vec3 specular = specularStrength * spec * lightColor;

    LightingColor = ambient + diffuse + specular;
}
```

3.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

在着色器中将这些变量定义为 uniform 类型，并且在渲染中与 ImGui 的 Slider 进行绑定。

```
// 光照的属性
float ambientStrength = 0.1;
float specularStrength = 0.5;
int shininess = 32;

ImGui::Text("Setting Light:");
ImGui::SliderFloat("Ambient Strength", &ambientStrength,0.0f,1.0f);
ImGui::SliderFloat("Specular Strength", &specularStrength, 0.0f, 1.0f);
```

```
ImGui::SliderInt("Shininess", &shininess, 2, 256);
```

通过更改光源的位置，可以改变漫反射因子，让一些不同的面与光照的夹角改变。光源有两种移动方式，第一种是通过 Slider 改变光源的 X、Y 坐标，第二种是在正方体前面画圆改变照射情况。

```
if (!isAutomove)
{
    lightPos.x = lightPosX;
    lightPos.y = lightPosY;
}
else
{
    float radius = 0.65f;
    lightPos.x = sin((float)glfwGetTime()) * radius;
    lightPos.y = cos((float)glfwGetTime()) * radius;
}
```