Homework 6 - Lights and Shading

16 级数媒 颜承橹 15322244

Basic:

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

1.Cube 的绘制可沿用之前的程序。

2.关于两种光照 shading 的实现，Gouraud Shading 和 Phong Shading 都是光照反射模型（Reflection Model），用来计算镜面反射方向的光强。它们计算发射光的方法是相同的，放射光的主要结构由 3 个分量组成：环境光(Ambient)、漫反射(Diffuse)和镜面(Specular)光照。

二者的不同之处在于：Gouraud 明暗处理只在多边形顶点处采用 Phong 局部反射模型计算光强，而在多边形内的其他点采用双向线性插值，这样做的优点是高效，但是无法很好的处理镜面高光问题，依赖于其所在多面形的相对位置；

而 Phong 明暗处理是通过插值计算每个顶点的法向量（3 次插值，在 x，y，z 三个方向分别进行插值计算），然后计算每个点上的光强值。这样效果好，但计算复杂，需要付出更多的时间。

光照计算代码部分：

把环境光照添加到场景里非常简单。先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，得到的就是该环境下该物体的颜色。

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

计算漫反射光照需要法向量和定向的光线，法向量可在顶点着色器中获得，对于 Phong 算法需要令顶点着色器中输出法向量，而 Gouraud 则可在顶点着色器中直接调用。此外还要添加法线矩阵，因为法线的变换不能简单地通过乘以变换矩阵获得。

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

镜面光照是基于光的反射特性。通过把相应的摄像机位置坐标传给片段着色器，获得光照方向，然后设置反射强度，从而计算出各处反射光照的强度。

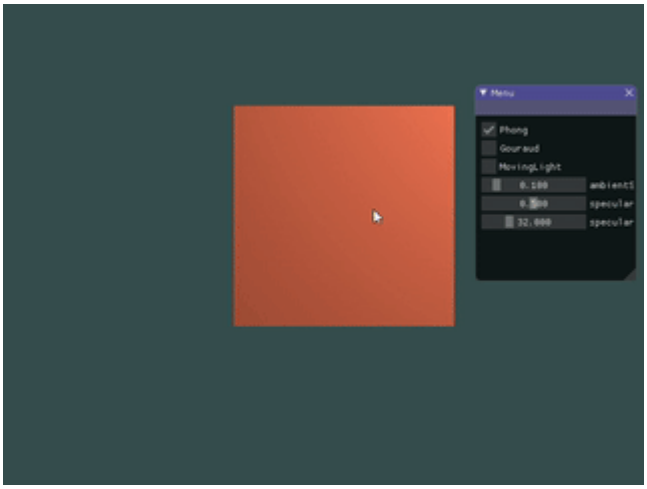
```
// specular
float specularStrength = 0.5;
vec3 viewDir = normalize(viewPos - FragPos);
vec3 reflectDir = reflect(-lightDir, norm);
float spec = pow(max(dot(viewDir, reflectDir), 0.0), specularPower);
vec3 specular = specularStrength * spec * lightColor;
vec3 result = (ambient + diffuse + specular) * objectColor;
FragColor = vec4(result, 1.0);
```

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，光照效果实时更新。

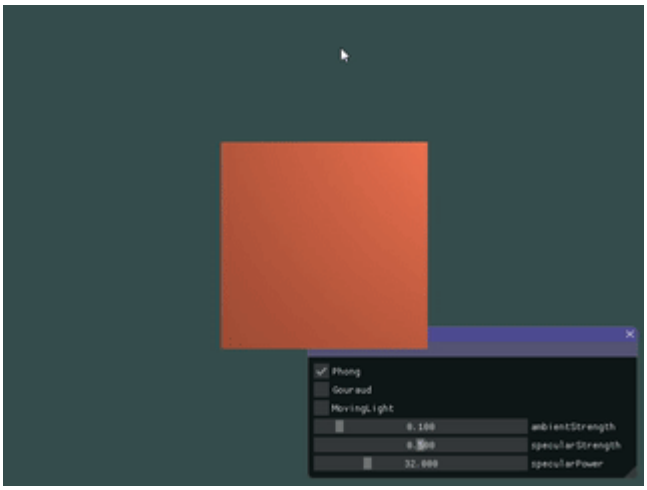
实现方法比较简单，即是将要变化的参数放入 GUI 中，得到的效果(gif 文件)如下：

（1）Shading 切换



从 gif 中能观察到使用 Gouraud Shading 时，会出现高光消失的现象。且 Gouraud 的正面明显比 Phong 亮很多，这是因为 Gourd 是先获得顶点的亮度再对三角形做插值，相比之下 Phong 的效果更加真实。而且在 Gouraud 中，随着视角的变换，当面上的顶点变暗时，能看到整个面也跟着变暗。

（2）光照因子调节



通过改变各个因子调节光照强度。

Bonus：

当前光源为静止状态，尝试使光源在场景中来回移动，光照效果实时更改。

实现方法是设置一个变量记录时间，然后控制另一个变量在 0~2 之间来回变换，将周期变换的值作为光源的坐标，从而实现光源来回移动的效果。

结果（gif 文件）如下：