Homework6——Lights and Shading

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

Phong Shading

Phong Shading操作要在片段着色器文件中进行，因为它的目标是每个像素。

1. 环境光

环境光比较简单，只是一个反射率和环境光强的乘积，它对每一个像素都是一样的：

```
vec3 ambient = ambientStrength * lightColor;
```

2. 漫反射光

漫反射光是发光源光照在法向量上的分量，也就是说与平面夹角越小，漫反射光越弱，可以用向量的点乘来计算，漫反射光也有一个反射率：

```
vec3 norm = normalize(Normal);  #法向量的标准化
vec3 lightDir = normalize(lightPos - FragPos); #光照的方向
loat diff = max(dot(norm, lightDir), 0.0);  #法向量与光照的角度不能大于90度，点成都得到光照在法向量上的投影
vec3 diffuse = diff * lightColor * diffuseStrength; #得到最终的光强
```

3. 镜面反射光

镜面反射多了一个视线因素，随着视线的不同，光强也会有所不同。离反射光线夹角越大，光强越小。镜面发射有一个反射强度（反射光线强度与它有关）specularStrength和一个反光度（反射光线的散射强度与它有关）specN：

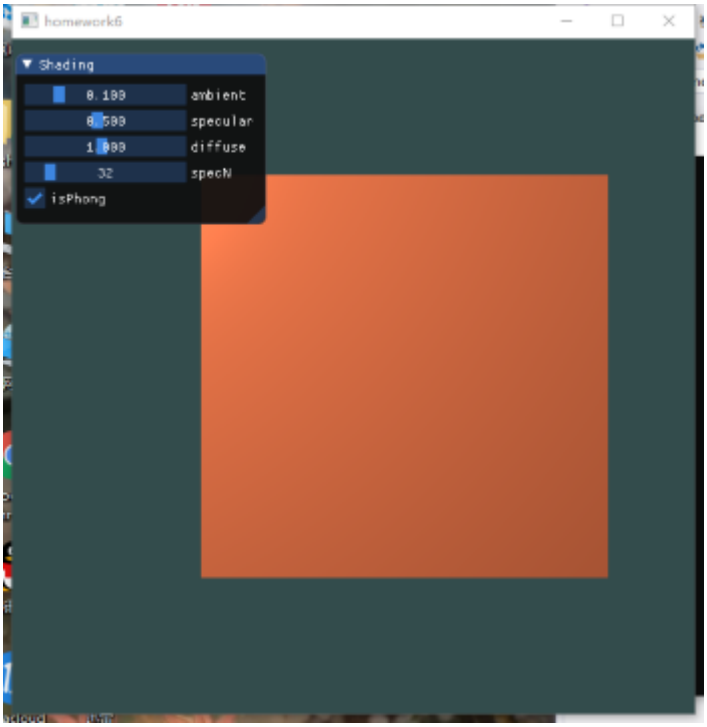
```
vec3 viewDir = normalize(viewPos - FragPos);     #视线方向
vec3 reflectDir = reflect(-lightDir, norm);      #反射方向
float spec = pow(max(dot(viewDir, reflectDir), 0.0), specN);     #视线与反射光的点乘
vec3 specular = specularStrength * spec * lightColor;    #最终光强
```

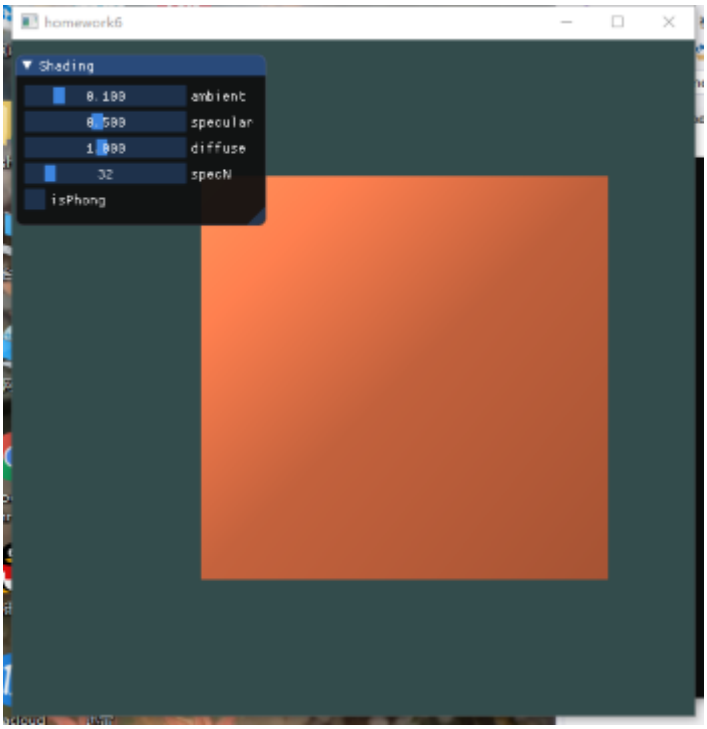
Gouraud Shading

Gouraud Shading操作在顶点着色器文件中进行，由于顶点数大幅减小，计算时间会很快，结构与Phong Shading是相同的。

结果

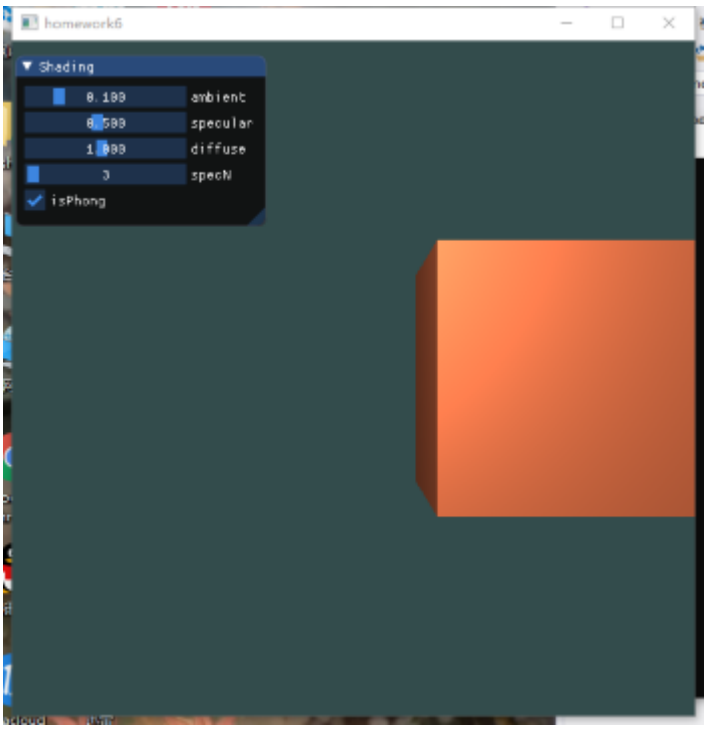
相同条件下的Phong SHding和Gouraud Shading：

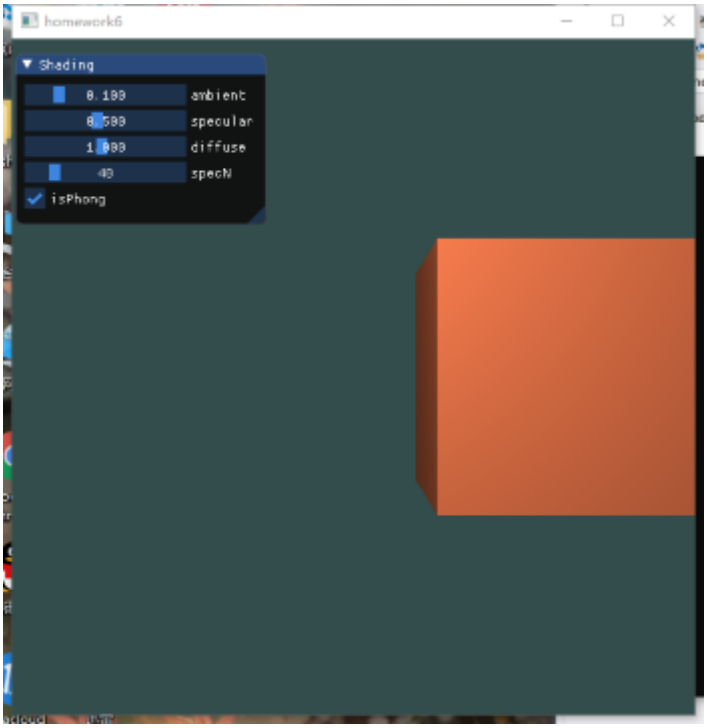




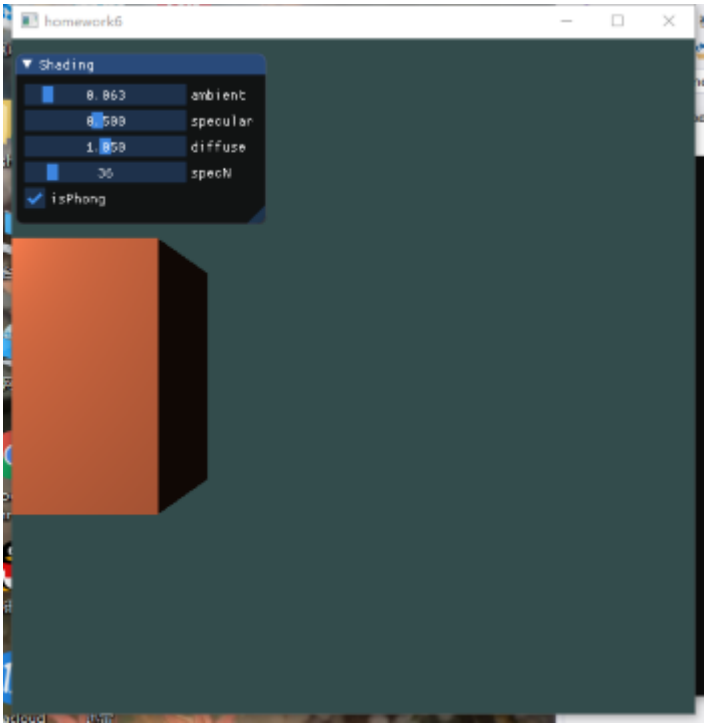
Gouraud Shading在对角线出有一道不自然的过渡。

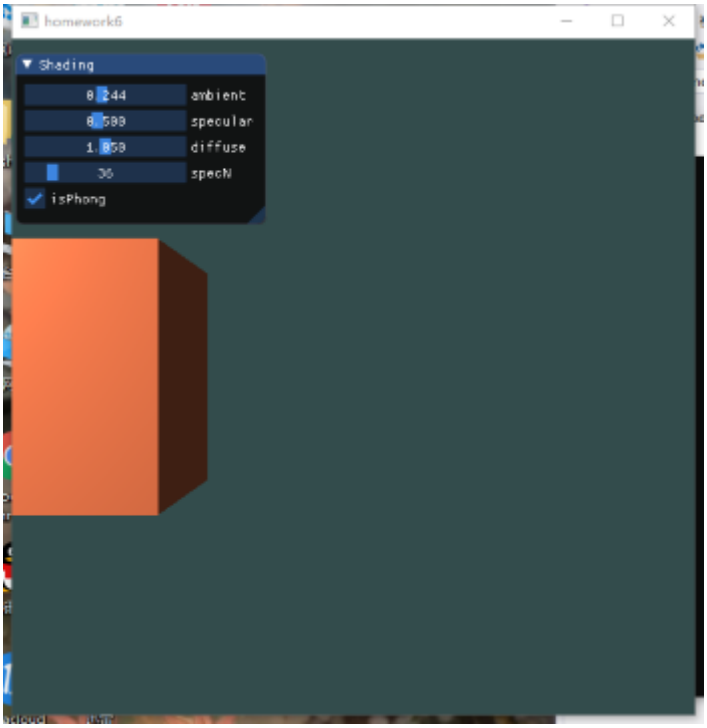
specN为3和40时，镜面反射的效果不同：



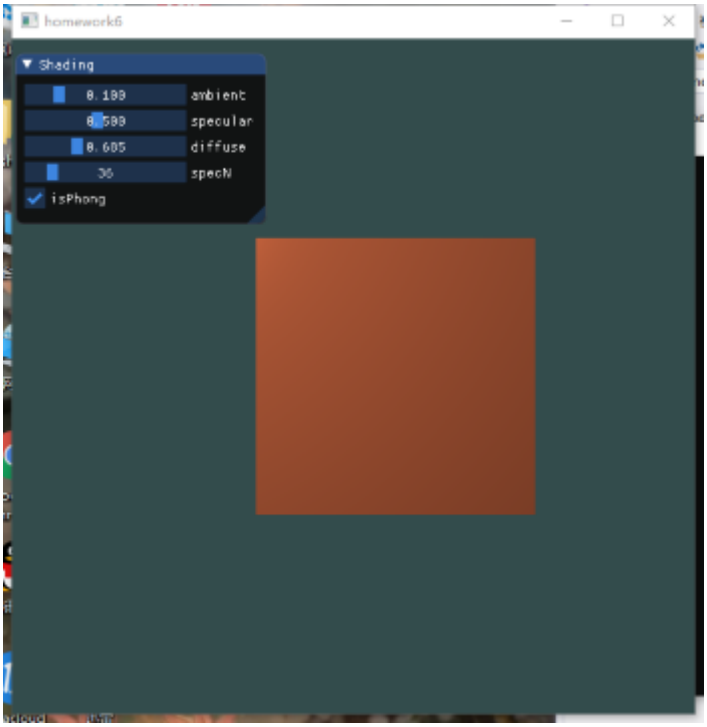


环境光反射较弱和较强的效果：

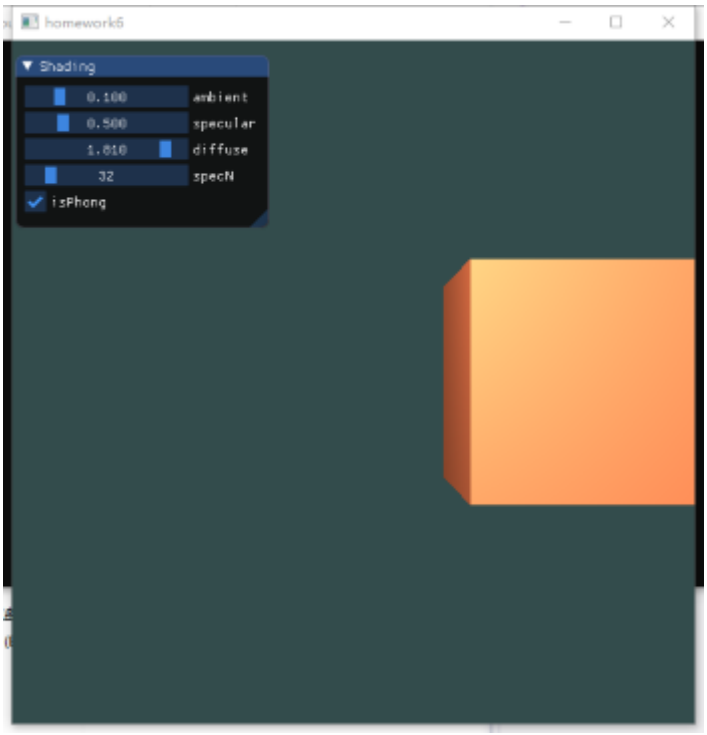

homework6
Shading
0.863
ambient
0.500
specular
1.000
diffuse
36
specN
isPhong

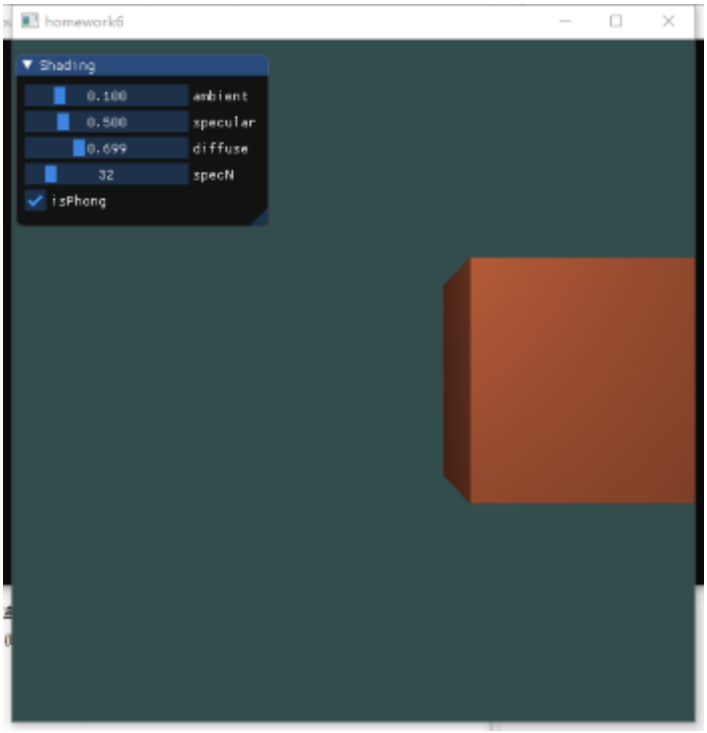

homework6
Shading
0.244
ambient
0.500
specular
1.000
diffuse
36
specN
isPhong

漫反射因子调小后效果：



镜面反射强度较强和较弱





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

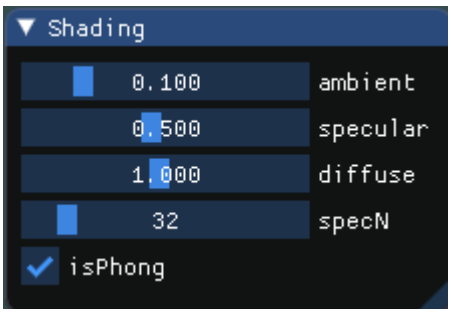
设置变量isPhong判断两种Shading，传入两个着色器中，不同的情况下执行不同的部分，比如顶点着色器：

```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;

out vec3 FragPos;
out vec3 Normal;

#Gouraud Shading的情况下的结果
out vec3 LightingColor;

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

#下面是两个着色器共有的uniform变量
uniform bool isPhong;

uniform float ambientStrength;
uniform float specularStrength;
uniform float diffuseStrength;
uniform int specN;
uniform vec3 lightColor;
uniform vec3 lightPos;
uniform vec3 viewPos;
```

```
void main()
{
    FragPos = vec3(model * vec4(aPos, 1.0));
    Normal = aNormal;

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

    #什么都不必做
    if(isPhong){
        LightingColor = vec3(1.0f);
    }
    #进行Shading算法
    else{
        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 * diffuseStrength;

        vec3 viewDir = normalize(viewPos - Position);
        vec3 reflectDir = reflect(-lightDir, norm);
        float spec = pow(max(dot(viewDir, reflectDir), 0.0), 32);
        vec3 specular = specularStrength * spec * lightColor;

        LightingColor = ambient + diffuse + specular;
    }

}
```

调节的参数就像上一问中提到的那样。

GUI截图：



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。

这个要求比较好实现，传入着色器变量中的lightPos就是光源位置，将它设置为随时间旋转的即可，这个在上次作业中有实现过绕半径旋转：

```
//环绕光源
        float lamPosX = sin(clock() / 1000.0)*radius;
        float lamPosZ = cos(clock() / 1000.0)*radius;
        glm::vec3 lightPos = glm::vec3(lamPosX, 2.0f, lamPosZ);

...
//传入着色器
    cubeShader.setVec3("lightPos", lightPos);
```

效果见目录下的gif图。