中山大學
SUN YAT-SEN UNIVERSITY

《计算机图形学》

实验报告

（作业六）

学 院 名 称 ：	数据科学与计算机学院
专业（班级）：	软件工程
学 生 姓 名 ：	江炎鸿
学　　　　号 ：	16340094
时　　　　间 ：	2019 年 5 月 7 日

作业六：Lights and Shading

一、作业内容：

1. 实现Phong光照模型：

（1）场景中绘制一个cube；

（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；

（3）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：

（1）GUI里可以切换两种shading；

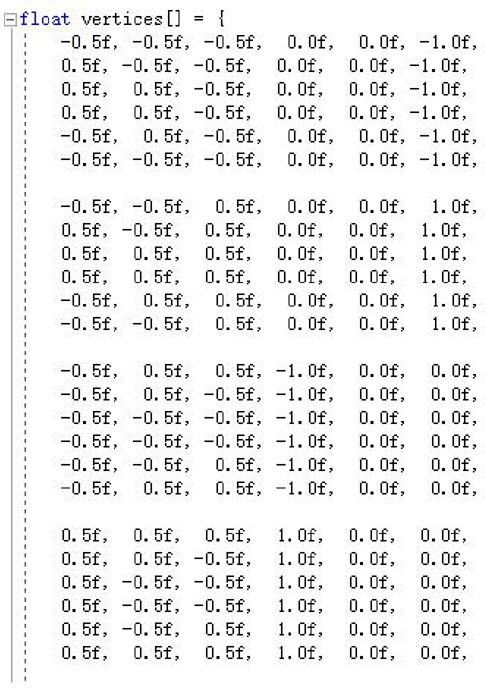
（2）使用如进度条这样的控件，使ambient因子、diffuse因子、specular因子、反光度等参数可调节，光照效果实时更改。

3. 加分项：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。

二、具体实现以及效果截图（完整效果见同目录下gif文件）

1、绘制cube：

这次不需要颜色信息，所以只需要顶点坐标和法向量：（代码如下）

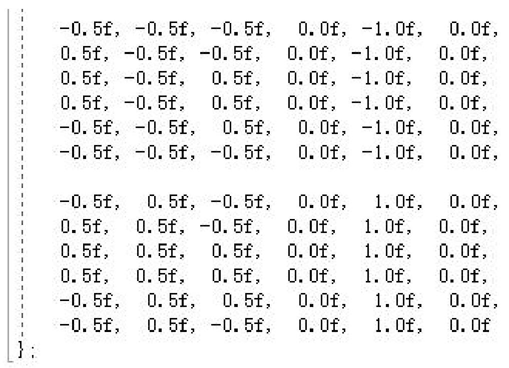


```
float vertices[] = {
    -0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
     0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
     0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
     0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
    -0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
    -0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,

    -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, -1.0f,  0.0f,  0.0f,
    -0.5f,  0.5f, -0.5f, -1.0f,  0.0f,  0.0f,
    -0.5f, -0.5f, -0.5f, -1.0f,  0.0f,  0.0f,
    -0.5f, -0.5f, -0.5f, -1.0f,  0.0f,  0.0f,
    -0.5f, -0.5f,  0.5f, -1.0f,  0.0f,  0.0f,
    -0.5f,  0.5f,  0.5f, -1.0f,  0.0f,  0.0f,

     0.5f,  0.5f,  0.5f,  1.0f,  0.0f,  0.0f,
     0.5f,  0.5f, -0.5f,  1.0f,  0.0f,  0.0f,
     0.5f, -0.5f, -0.5f,  1.0f,  0.0f,  0.0f,
     0.5f, -0.5f, -0.5f,  1.0f,  0.0f,  0.0f,
     0.5f, -0.5f,  0.5f,  1.0f,  0.0f,  0.0f,
     0.5f,  0.5f,  0.5f,  1.0f,  0.0f,  0.0f,
```



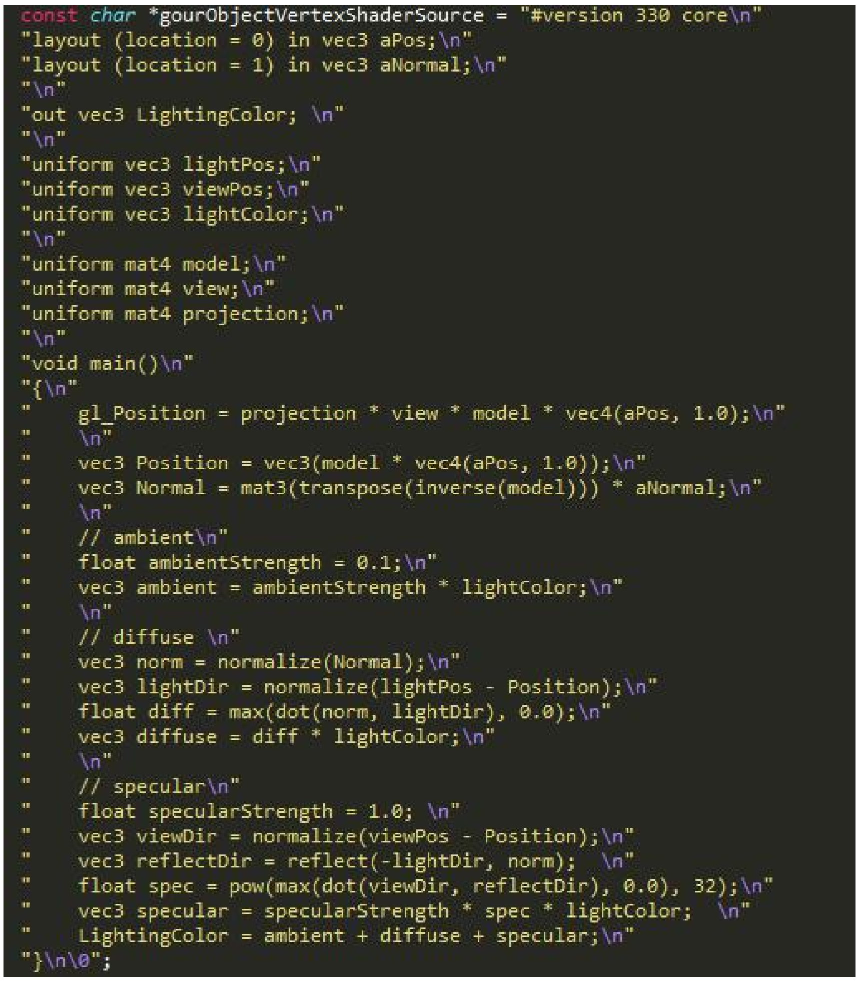
```
    -0.5f, -0.5f, -0.5f,  0.0f, -1.0f,  0.0f,
     0.5f, -0.5f, -0.5f,  0.0f, -1.0f,  0.0f,
     0.5f, -0.5f,  0.5f,  0.0f, -1.0f,  0.0f,
     0.5f, -0.5f,  0.5f,  0.0f, -1.0f,  0.0f,
    -0.5f, -0.5f,  0.5f,  0.0f, -1.0f,  0.0f,
    -0.5f, -0.5f, -0.5f,  0.0f, -1.0f,  0.0f,

    -0.5f,  0.5f, -0.5f,  0.0f,  1.0f,  0.0f,
     0.5f,  0.5f, -0.5f,  0.0f,  1.0f,  0.0f,
     0.5f,  0.5f,  0.5f,  0.0f,  1.0f,  0.0f,
     0.5f,  0.5f,  0.5f,  0.0f,  1.0f,  0.0f,
    -0.5f,  0.5f,  0.5f,  0.0f,  1.0f,  0.0f,
    -0.5f,  0.5f, -0.5f,  0.0f,  1.0f,  0.0f
};
```

2、Gouraud Shading

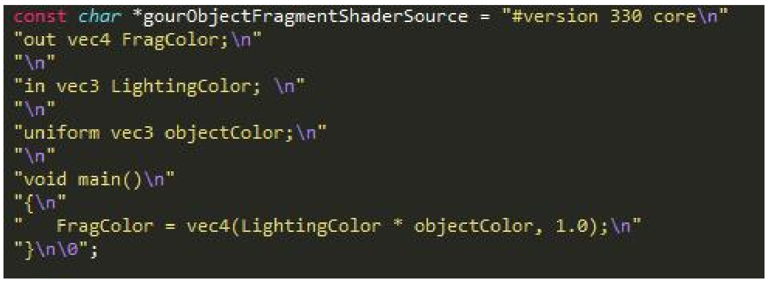
此种方法是早期实现冯氏光照模型的方法，就是在顶点着色器中实现光照模型，对于每一个顶点利用一个简单均匀的环境光代替复杂的全局光来简化模型，然后加上物体表面的漫反射和镜面反射来构成最终的该点的光照效果，然后再根据顶点的颜色插值得到其他点的颜色值，从而实现整个物体的光照模型。此种方法的顶点数较少，因此运算量比较小，但也因此导致实现出来的效果不太真实，而且无法出现高光。

顶点着色器代码如下：



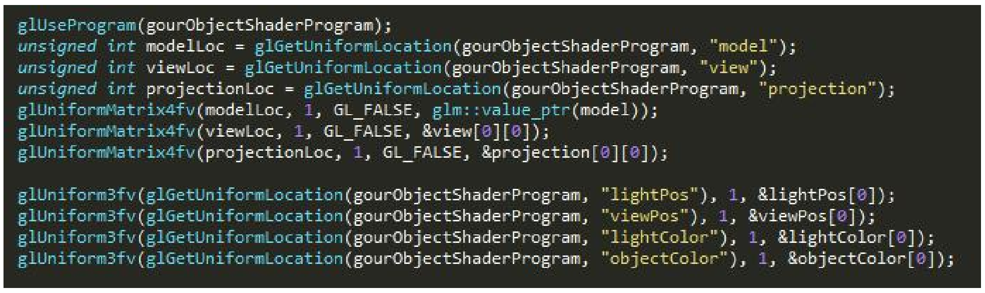
```
const char *gourObjectVertexShaderSource = "#version 330 core\n"
"layout (location = 0) in vec3 aPos;\n"
"layout (location = 1) in vec3 aNormal;\n"
"\n"
"out vec3 LightingColor; \n"
"\n"
"uniform vec3 lightPos;\n"
"uniform vec3 viewPos;\n"
"uniform vec3 lightColor;\n"
"\n"
"uniform mat4 model;\n"
"uniform mat4 view;\n"
"uniform mat4 projection;\n"
"\n"
"void main()\n"
"{\n"
"    gl_Position = projection * view * model * vec4(aPos, 1.0);\n"
"    \n"
"    vec3 Position = vec3(model * vec4(aPos, 1.0));\n"
"    vec3 Normal = mat3(transpose(inverse(model))) * aNormal;\n"
"    \n"
"    // ambient\n"
"    float ambientStrength = 0.1;\n"
"    vec3 ambient = ambientStrength * lightColor;\n"
"    \n"
"    // diffuse \n"
"    vec3 norm = normalize(Normal);\n"
"    vec3 lightDir = normalize(lightPos - Position);\n"
"    float diff = max(dot(norm, lightDir), 0.0);\n"
"    vec3 diffuse = diff * lightColor;\n"
"    \n"
"    // specular\n"
"    float specularStrength = 1.0; \n"
"    vec3 viewDir = normalize(viewPos - Position);\n"
"    vec3 reflectDir = reflect(-lightDir, norm);  \n"
"    float spec = pow(max(dot(viewDir, reflectDir), 0.0), 32);\n"
"    vec3 specular = specularStrength * spec * lightColor;  \n"
"    LightingColor = ambient + diffuse + specular;\n"
"}\n\0";
```

片段着色器代码如下：



```
const char *gourObjectFragmentShaderSource = "#version 330 core\n"
"out vec4 FragColor;\n"
"\n"
"in vec3 LightingColor; \n"
"\n"
"uniform vec3 objectColor;\n"
"\n"
"void main()\n"
"{\n"
"   FragColor = vec4(LightingColor * objectColor, 1.0);\n"
"}\n\0";
```

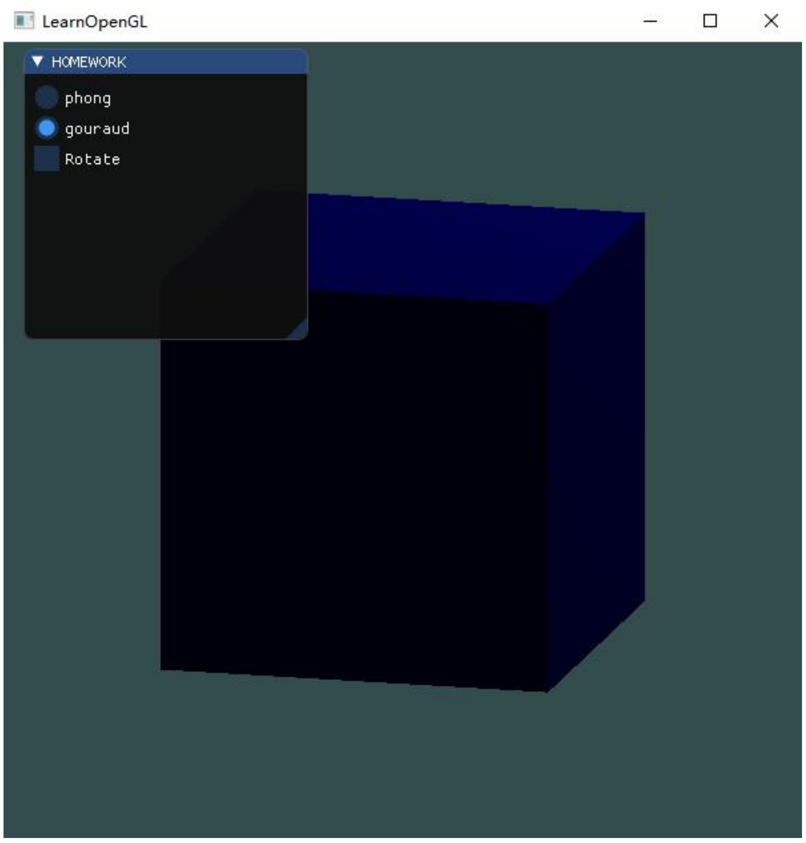
使用部分：



```
glUseProgram(gourObjectShaderProgram);
unsigned int modelLoc = glGetUniformLocation(gourObjectShaderProgram, "model");
unsigned int viewLoc = glGetUniformLocation(gourObjectShaderProgram, "view");
unsigned int projectionLoc = glGetUniformLocation(gourObjectShaderProgram, "projection");
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(viewLoc, 1, GL_FALSE, &view[0][0]);
glUniformMatrix4fv(projectionLoc, 1, GL_FALSE, &projection[0][0]);

glUniform3fv(glGetUniformLocation(gourObjectShaderProgram, "lightPos"), 1, &lightPos[0]);
glUniform3fv(glGetUniformLocation(gourObjectShaderProgram, "viewPos"), 1, &viewPos[0]);
glUniform3fv(glGetUniformLocation(gourObjectShaderProgram, "lightColor"), 1, &lightColor[0]);
glUniform3fv(glGetUniformLocation(gourObjectShaderProgram, "objectColor"), 1, &objectColor[0]);
```

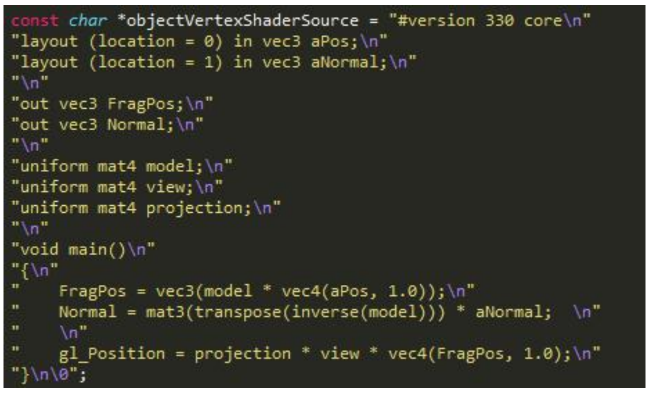
效果截图：



3、Phong Shading

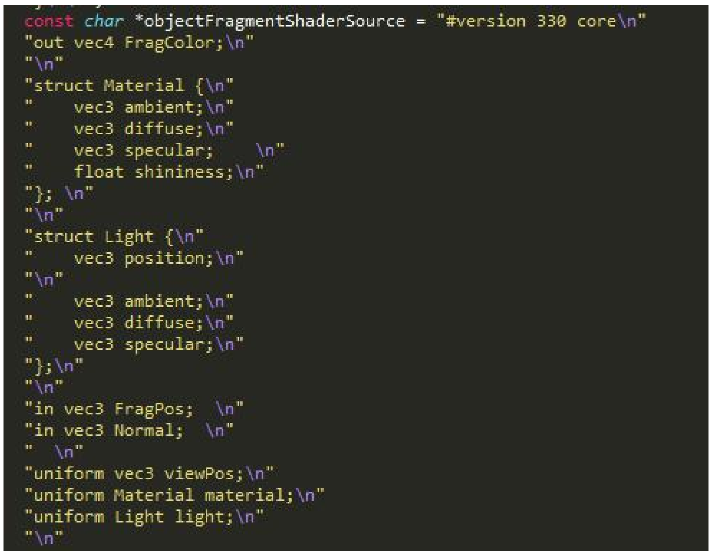
此种方法是现在比较常用的方法，是在片段着色器中实现光照模型，因此对于每一个点都会有一个独立的计算，所以实现的效果比较贴合现实。

顶点着色器代码如下：

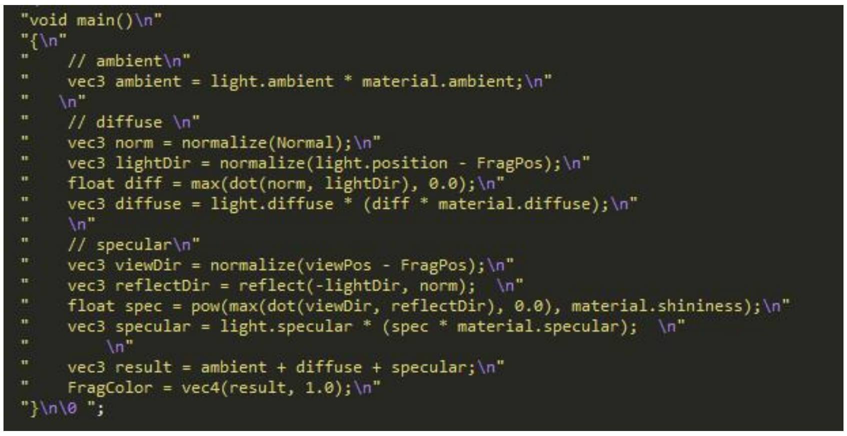


```
const char *objectVertexShaderSource = "#version 330 core\n"
"layout (location = 0) in vec3 aPos;\n"
"layout (location = 1) in vec3 aNormal;\n"
"\n"
"out vec3 FragPos;\n"
"out vec3 Normal;\n"
"\n"
"uniform mat4 model;\n"
"uniform mat4 view;\n"
"uniform mat4 projection;\n"
"\n"
"void main()\n"
"{\n"
"    FragPos = vec3(model * vec4(aPos, 1.0));\n"
"    Normal = mat3(transpose(inverse(model))) * aNormal;   \n"
"    \n"
"    gl_Position = projection * view * vec4(FragPos, 1.0);\n"
"}\n\0";
```

片段着色器代码如下：

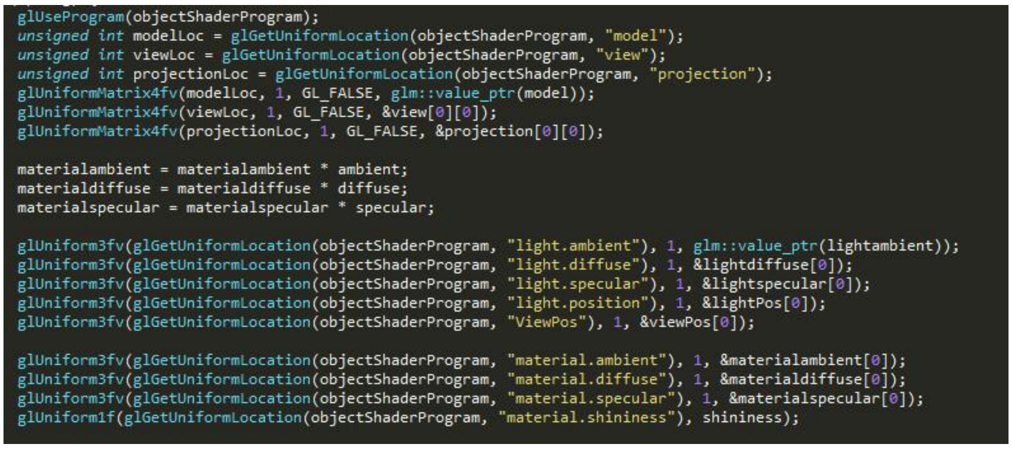


```
const char *objectFragmentShaderSource = "#version 330 core\n"
"out vec4 FragColor;\n"
"\n"
"struct Material {\n"
"    vec3 ambient;\n"
"    vec3 diffuse;\n"
"    vec3 specular;    \n"
"    float shininess;\n"
"}; \n"
"\n"
"struct Light {\n"
"    vec3 position;\n"
"\n"
"    vec3 ambient;\n"
"    vec3 diffuse;\n"
"    vec3 specular;\n"
"};\n"
"\n"
"in vec3 FragPos;  \n"
"in vec3 Normal;  \n"
"  \n"
"uniform vec3 viewPos;\n"
"uniform Material material;\n"
"uniform Light light;\n"
"\n"
```



```
"void main()\n"
"{\n"
"    // ambient\n"
"    vec3 ambient = light.ambient * material.ambient;\n"
"   \n"
"    // diffuse \n"
"    vec3 norm = normalize(Normal);\n"
"    vec3 lightDir = normalize(light.position - FragPos);\n"
"    float diff = max(dot(norm, lightDir), 0.0);\n"
"    vec3 diffuse = light.diffuse * (diff * material.diffuse);\n"
"    \n"
"    // specular\n"
"    vec3 viewDir = normalize(viewPos - FragPos);\n"
"    vec3 reflectDir = reflect(-lightDir, norm);  \n"
"    float spec = pow(max(dot(viewDir, reflectDir), 0.0), material.shininess);\n"
"    vec3 specular = light.specular * (spec * material.specular);  \n"
"        \n"
"    vec3 result = ambient + diffuse + specular;\n"
"    FragColor = vec4(result, 1.0);\n"
"}\n\0 ";
```

使用部分如下：



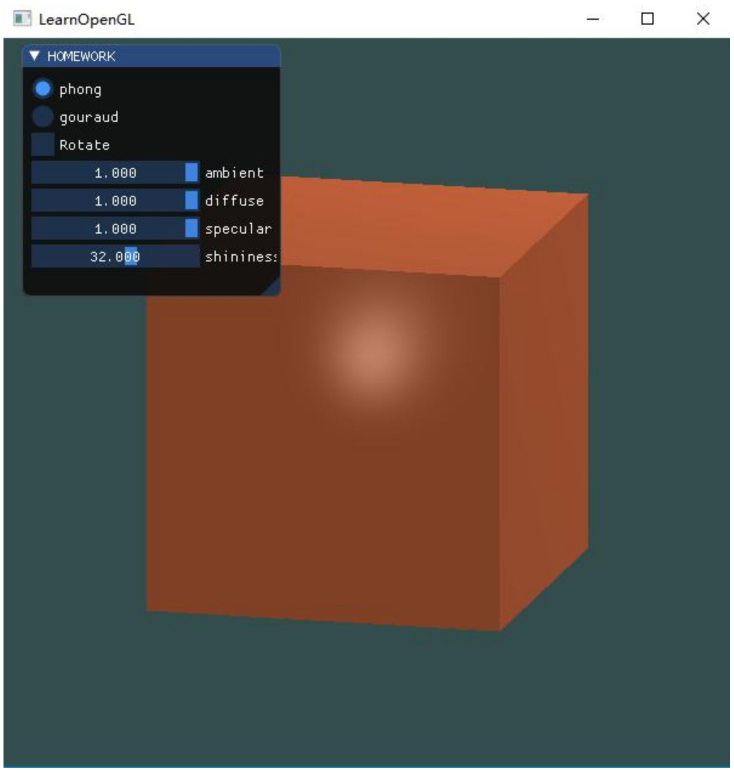
```
glUseProgram(objectShaderProgram);
unsigned int modelLoc = glGetUniformLocation(objectShaderProgram, "model");
unsigned int viewLoc = glGetUniformLocation(objectShaderProgram, "view");
unsigned int projectionLoc = glGetUniformLocation(objectShaderProgram, "projection");
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(viewLoc, 1, GL_FALSE, &view[0][0]);
glUniformMatrix4fv(projectionLoc, 1, GL_FALSE, &projection[0][0]);

materialambient = materialambient * ambient;
materialdiffuse = materialdiffuse * diffuse;
materialspecular = materialspecular * specular;

glUniform3fv(glGetUniformLocation(objectShaderProgram, "light.ambient"), 1, glm::value_ptr(lightambient));
glUniform3fv(glGetUniformLocation(objectShaderProgram, "light.diffuse"), 1, &lightdiffuse[0]);
glUniform3fv(glGetUniformLocation(objectShaderProgram, "light.specular"), 1, &lightspecular[0]);
glUniform3fv(glGetUniformLocation(objectShaderProgram, "light.position"), 1, &lightPos[0]);
glUniform3fv(glGetUniformLocation(objectShaderProgram, "ViewPos"), 1, &viewPos[0]);

glUniform3fv(glGetUniformLocation(objectShaderProgram, "material.ambient"), 1, &materialambient[0]);
glUniform3fv(glGetUniformLocation(objectShaderProgram, "material.diffuse"), 1, &materialdiffuse[0]);
glUniform3fv(glGetUniformLocation(objectShaderProgram, "material.specular"), 1, &materialspecular[0]);
glUniform1f(glGetUniformLocation(objectShaderProgram, "material.shininess"), shininess);
```

效果截图：

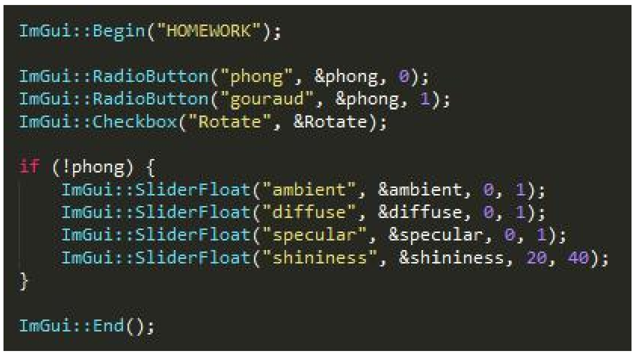


● phong
○ gouraud
☐ Rotate
1.000 ambient
1.000 diffuse
1.000 specular
32.000 shininess

4、GUI的实现

这部分其实还是跟之前的基本一样，只要按照需求设置就好了

关键代码实现如下：



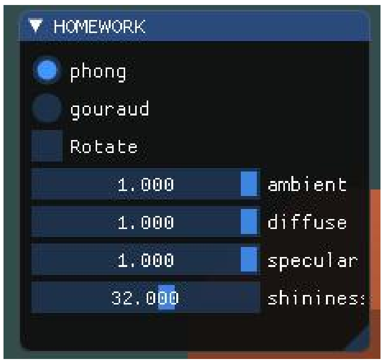
```
ImGui::Begin("HOMEWORK");

ImGui::RadioButton("phong", &phong, 0);
ImGui::RadioButton("gouraud", &phong, 1);
ImGui::Checkbox("Rotate", &Rotate);

if (!phong) {
    ImGui::SliderFloat("ambient", &ambient, 0, 1);
    ImGui::SliderFloat("diffuse", &diffuse, 0, 1);
    ImGui::SliderFloat("specular", &specular, 0, 1);
    ImGui::SliderFloat("shininess", &shininess, 20, 40);
}

ImGui::End();
```

效果截图：


▼ HOMEWORK

phong [✓]

gouraud

Rotate

1.000 ambient

1.000 diffuse

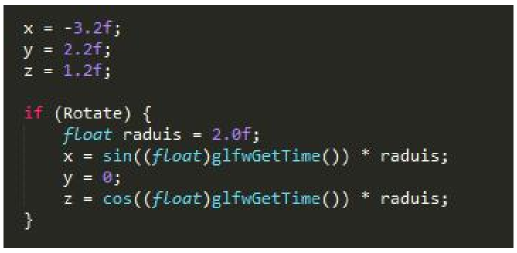
1.000 specular

32.000 shininess

5、实现光源移动：

这部分其实就是把光源当成之前作业中的地球，使其绕着cube（之前的太阳）旋转即可。

关键代码：（x，y，z即为光源位置）



```
x = -3.2f;
y = 2.2f;
z = 1.2f;

if (Rotate) {
    float raduis = 2.0f;
    x = sin((float)glfwGetTime()) * raduis;
    y = 0;
    z = cos((float)glfwGetTime()) * raduis;
}
```

实现效果见gif。