科目	学号	姓名	邮箱
计算机图形学	16340294	张星	1401046908@qq.com

Basic

题目一

实验要求

> 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 。

实验思路

首先，需要绘制两个Cube，一个作为光源，一个作为被照射的物体。需要一个vertex数组，然后绑定两个VAO，就可以实现绘制两个不同大小和属性的cube。其次，由于光照需要每个面的法向量，所以在数组中就要将每个点的法向量事先给定。cube由六个面组成，每个面又由两个三角形组成，这样对每个三角形顶点属性后面加上法向量属性即可。

```
unsigned int VAO, VBO;
glGenBuffers(1, &VBO);
glGenVertexArrays(1, &VAO);
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);

glBindVertexArray(VAO);

glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)0);
glEnableVertexAttribArray(0);
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)(3 *
sizeof(float)));
glEnableVertexAttribArray(1);

unsigned int lightVAO;
glGenVertexArrays(1, &lightVAO);
glBindVertexArray(lightVAO);
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)0);
glEnableVertexAttribArray(0);
```

- Phong Lighting Model：环境光+漫反射+镜面反射。

- Phong Shading：先计算出每一个顶点的法向量。使用线性插值法，对每一个像素计算出法向量，然后使用Phong Lighting Model渲染出顶点颜色与亮度。所以Phong Shading是在片段着色器中完成的。优点是：更加平滑，渲染结果更为真实，可产生高光；缺点是对每一个像素都需要应用一次Phong Lighting Model，计算代价太大。

  思路：首先计算环境光，然后计算漫反射和镜面反射，由于只有一个光源，所以直接将三者相加即可，然后得到了总的光源，再与物体颜色相乘即可得到物体渲染之后的效果。

```
const char *PhongFragmentSource = "#version 330 core\n"
"in vec3 Normal;\n"
"in vec3 FragPos;\n"
"out vec4 FragColor;\n"

"uniform vec3 lightPos;\n"
"uniform vec3 viewPos;\n"
"uniform vec3 lightColor;\n"
"uniform vec3 objectColor;\n"

"uniform float ambientStrength;\n"
"uniform float diffuseStrength;\n"
"uniform float specularStrength;\n"
"uniform int reflectance;\n"

"void main()\n"
"{\n"
"   vec3 ambient = ambientStrength * lightColor;\n"
"   vec3 norm = normalize(Normal);\n"
"   vec3 lightDir = normalize(lightPos - FragPos);\n"
"   float diff = max(dot(norm, lightDir), 0.0);\n"
"   vec3 diffuse = diffuseStrength * diff * lightColor;\n"
"   vec3 viewDir = normalize(viewPos - FragPos);\n"
"   vec3 reflectDir = reflect(-lightDir, norm);\n"
"   float spec = pow(max(dot(viewDir, reflectDir), 0.0), reflectance);\n"
"   vec3 specular = specularStrength * spec * lightColor;\n"
"   vec3 result = (ambient + diffuse + specular) * objectColor;\n"
"   FragColor = vec4(result, 1.0);\n"
"}\n\0";
```

- Gouraud Shading：使用插值方法，计算出每个顶点的法向量，然后使用Phong Lighting Model渲染出顶点颜色与亮度，三个顶点之间的其他部分颜色、亮度则使用双线性插值法来计算。所以Gouraud Shading是在顶点着色器中完成的。优点是计算简单，效果也不错；缺点是无法渲染出高光，会出现马赫带效应。

  代码如下：

```
const char *GouraudVertexSource = "#version 330 core\n"
"layout(location = 0) in vec3 aPos;\n"
"layout(location = 1) in vec3 aNormal;\n"
"out vec3 LightingColor;\n"

"uniform vec3 lightPos;\n"
"uniform vec3 viewPos;"
"uniform vec3 lightColor;"
```

```
"uniform float ambientStrength;\n"
"uniform float diffuseStrength;\n"
"uniform float specularStrength;\n"
"uniform int reflectance;\n"

"uniform mat4 model;\n"
"uniform mat4 view;\n"
"uniform mat4 projection;\n"
"void main()\n"
"{\n"
"   gl_Position = projection * view * model * vec4(aPos, 1.0);\n"

"   vec3 Position = vec3(model * vec4(aPos, 1.0));\n"
"   vec3 Normal = mat3(transpose(inverse(model))) * aNormal;\n"
"   vec3 ambient = ambientStrength * lightColor;\n"
"   vec3 norm = normalize(Normal);\n"
"   vec3 lightDir = normalize(lightPos - Position);\n"
"   float diff = max(dot(norm, lightDir), 0.0);\n"
"   vec3 diffuse = diffuseStrength * diff * lightColor;\n"
"   vec3 viewDir = normalize(viewPos - Position);\n"
"   vec3 reflectDir = reflect(-lightDir, norm);\n"
"   float spec = pow(max(dot(viewDir, reflectDir), 0.0), reflectance);\n"
"   vec3 specular = specularStrength * spec * lightColor;\n"
"   LightingColor = ambient + diffuse + specular;\n"
"}\0";
```

- 参数设置：在本次作业中我使用了上次的Camera类，但是取消了鼠标调整角度的功能，因为要点击ImGui，保留了WSAD，所以能够切换视角来更仔细地观察效果 。一开始我将物体放在坐标原点，将光源放在右上位置（3.0f, 3.0f, 0.0f）。

```
float ambientStrength = 0.1;
float diffuseStrength = 1.0;
float specularStrength = 0.5;
int reflectance = 32;
float radius = 3.0f;
```

实验效果

见result.gif：确如之前所预料，Gouraud 不会产生高光，当逐渐增加漫反射系数时，Phong 是由高光部分逐渐向旁边变亮，Gouraud 也是如此，但是扩散的速度要比 Phong 快一些。

题目二

实验要求

> 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。

实验思路

关于ImGui的实现，已经用了很多次了，不再赘述。关键是如何传递这些系数进Shader，这里我使用了uniform关键字，然后通过查询文档，像之前传递model等矩阵一样传递进去，就可以进行参数调整了。

```
if (Phong_Gouraud) {
    glUseProgram(PhongShader);

    glUniform3fv(glGetUniformLocation(PhongShader, "objectColor"), 1, &ObjectColor[0]);
    glUniform3f(glGetUniformLocation(PhongShader, "lightColor"), 1.0f, 1.0f, 1.0f);
    glUniform3fv(glGetUniformLocation(PhongShader, "lightPos"), 1, &lightPos[0]);
    glUniform3fv(glGetUniformLocation(PhongShader, "viewPos"), 1, &camera.getPosition()
[0]);

    glUniform1f(glGetUniformLocation(PhongShader, "ambientStrength"), ambientStrength);
    glUniform1f(glGetUniformLocation(PhongShader, "diffuseStrength"), diffuseStrength);
    glUniform1f(glGetUniformLocation(PhongShader, "specularStrength"), specularStrength);
    glUniform1i(glGetUniformLocation(PhongShader, "reflectance"), reflectance);

    glUniformMatrix4fv(glGetUniformLocation(PhongShader, "model"), 1, GL_FALSE,
glm::value_ptr(model));
    glUniformMatrix4fv(glGetUniformLocation(PhongShader, "view"), 1, GL_FALSE, &view[0]
[0]);
    glUniformMatrix4fv(glGetUniformLocation(PhongShader, "projection"), 1, GL_FALSE,
&projection[0][0]);
    glBindVertexArray(VAO);
    glDrawArrays(GL_TRIANGLES, 0, 36);
}
```

环境光，漫反射和镜面反射等都可以直接使用参数相乘来调整，反光度我使用了课上所讲的n，通过调整其来调整镜面反射的高光集中程度，即以下的reflectance参数：

```
"   float spec = pow(max(dot(viewDir, reflectDir), 0.0), reflectance);\n"
```

Bonus

题目一

实验要求

> 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。

实验思路

我的想法是将光源像之前作业那样围绕坐标原点旋转，半径可调整。这样的话就可以实现光源在场景中来回移动，但是对于每个Shader，都需要传入光源的位置，这样才能及时地根据入射光线计算漫反射和镜面反射等，所以难点在于如何获取光源的位置。这里我使用了cos和sin函数来获取光源的x和y坐标，将其传入Shader，就完成了光源的实时更新，从而Shader也会及时渲染。

```
if (bonus) {
    float currentAngle = (float)glfwGetTime()*glm::radians(45.0f);
    lightModel = glm::rotate(model, currentAngle, glm::vec3(0.0f, 0.0f, 1.0f));
    lightModel = glm::translate(lightModel, glm::vec3(radius, 0.0f, 0.0f));
    lightModel = glm::scale(lightModel, glm::vec3(0.2f));
    lightPos.x = cos(currentAngle)*radius;
    lightPos.y = sin(currentAngle)*radius;
}
else {
    lightModel = glm::translate(lightModel, lightPos);
    lightModel = glm::scale(lightModel, glm::vec3(0.2f));
}
```

实验结果

见result.gif。