深 圳 大 学 实 验 报 告

课程名称：计算机图形学

实验项目名称：实验三 光照与阴影

学院：计算机与软件学院

专业：计算机科学与技术

指导教师：周虹

报告人：吴嘉楷 学号：2022150168 班级：国际班

实验时间：2024 年 10 月 22 日 -- 2024 年 11 月 25 日

实验报告提交时间：2024 年 11 月 15 日

教务部制

实验目的与要求：

1. 掌握 OpenGL 三维场景的读取与绘制方法，理解光照和物体材质对渲染结果的影响，强化场景坐标系转换过程中常见矩阵的计算方法，熟悉阴影的绘制方法。
2. 创建 OpenGL 绘制窗口，读入三维场景文件并绘制。
3. 设置相机并添加交互，实现从不同位置/角度、以正交或透视投影方式观察场景。
4. 实现 Phong 光照效果和物体材质效果。
5. 自定义投影平面（为计算方便，推荐使用 y=0 平面），计算阴影投影矩阵，为三维物体生成阴影。
6. 使用鼠标点击（或其他方式）控制光源位置并更新光照效果，并同时更新三维物体的阴影。

实验过程及内容：

1. 使用实验 3.3 的代码，并对其进行项目构建

```
C:\Users\24312\Desktop\code>cmake -B.
-- Building for: Visual Studio 17 2022
CMake Deprecation Warning at CMakeLists.txt:2 (cmake_minimum_required):
  Compatibility with CMake < 3.5 will be removed from a future version of
  CMake.

  Update the VERSION argument <min> value or use a ...<max> suffix to tell
  CMake that the project does not need compatibility with older versions.


-- Selecting Windows SDK version 10.0.22621.0 to target Windows 10.0.22631
```

图 1 cmake 实验 3.3 的代码

2. 调整光源位置

若不调整光源位置，则初始时光源会出现在（0，0，2）处，无法将阴影投影在 y=0 的平面上。因此，应该调整光源位置到（0，2，2），让光源出现在球体的上方，从而产生出现在 y=0 平面上的阴影。

```
// @TODO: Task3 请自己调整光源参数和物体材质参数来达到不同视觉效果
// 设置光源位置
light->setTranslation(glm::vec3(0.0, 2.0, 2.0));
light->setAmbient(glm::vec4(1.0, 1.0, 1.0, 1.0)); // 环境光
light->setDiffuse(glm::vec4(1.0, 1.0, 1.0, 1.0)); // 漫反射
light->setSpecular(glm::vec4(1.0, 1.0, 1.0, 1.0)); // 镜面反射
```

图 2 调整光源位置

3. 调整物体材质

```
mesh->setAmbient(glm::vec4(0.2, 0.2, 0.2, 1.0)); // 环境光
mesh->setDiffuse(glm::vec4(0.7, 0.7, 0.7, 1.0)); // 漫反射
mesh->setSpecular(glm::vec4(0.2, 0.2, 0.2, 1.0)); // 镜面反射
mesh->setShininess(50.0); //高光系数
```

图 3 调整物体材质

此处，调整物体材质的目的是为了让物体的光照效果更佳，更趋近于示例效果。

4. 同步修改键盘交互复原物体材质的逻辑

```
else if(key == GLFW_KEY_MINUS && action == GLFW_PRESS && mode == 0x0000)
{
    mesh->setAmbient(glm::vec4(0.2, 0.2, 0.2, 1.0)); // 环境光
    mesh->setDiffuse(glm::vec4(0.7, 0.7, 0.7, 1.0)); // 漫反射
    mesh->setSpecular(glm::vec4(0.2, 0.2, 0.2, 1.0)); // 镜面反射
    mesh->setShininess(50.0); //高光系数
}
```

图 4 同步修改复原逻辑

5. 更改窗口的标题

```
GLFWwindow* mainwindow = glfwCreateWindow(700, 700, "2022150168_吴嘉楷_实验3", NULL, NULL);
```

图 5 修改标题栏

6. 设置灰色背景

在 main.cpp 文件的 display 函数中，使用 glClearColor 方法设置窗口背景颜色为灰色，其 rgb 值为（0.5，0.5，0.5）：

```
void display()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    // 设置灰色背景
    glClearColor(0.5, 0.5, 0.5, 1.0);
```

图 6 设置灰色背景颜色

7. 运行代码，此时效果如下：

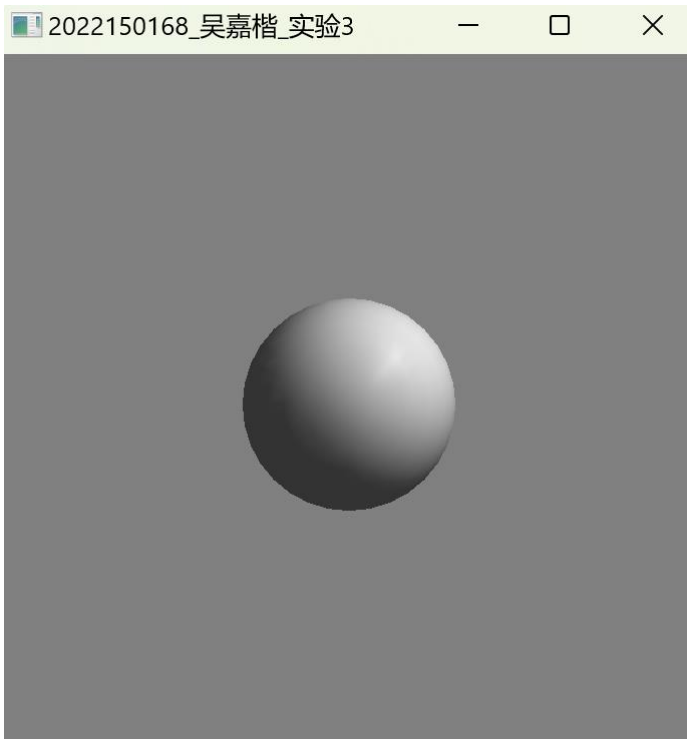

图 7 未完成时的运行效果

8. 修改默认加载的物体

原本是默认加载正方体，由于正方体不好投影，于是我们修改为默认加载球体，便于投影效果的展示。

```
}
glEnable(GL_DEPTH_TEST);
// 默认加载球体
mesh->generateCube();
mesh->readOff("./assets/sphere.off");
// Init mesh, shaders, buffer
init();
printHelp();
```

图 8 默认加载球体

9. 添加一个平面变量：用于展示物体的投影

这个变量使用 TriMesh 类型，其中记录了投影的每个面片的法向量、顶点、变换、颜色等信息。

```
openGLObject mesh_object;   // 物体对象
openGLObject plane_object;  // 平面对象

Light* light = new Light();

TriMesh* mesh = new TriMesh();

Camera* camera = new Camera();
// 平面
TriMesh* plane = new TriMesh();
```

图 9 添加一个平面变量

10. 在 init 函数中载入投影面的信息

代码截图：

```
// 将物体的顶点数据传递
bindObjectAndData(mesh, mesh_object, vshader, fshader);

// 生成平面
plane->generateSquare(glm::vec3(0.7, 0.7, 0.7));
// 设置平面的旋转位移
plane->setTranslation(glm::vec3(0.0, -0.001, 0.0));
plane->setRotation(glm::vec3(0, 90, 90));
plane->setScale(glm::vec3(3.0, 3.0, 3.0));

// 设置灰色背景
glClearColor(0.5, 0.5, 0.5, 1.0);
```

图 10 载入投影面信息

代码解释：

首先，plane->generateSquare(glm::vec3(0.7, 0.7, 0.7)); 用来生成一个平面几何体，并给它赋予一个**灰色**的颜色。颜色通过 glm::vec3(0.7, 0.7, 0.7) 设置，表示一种**中等灰色**（RGB 值分别为 0.7）。

然后，plane->setTranslation(glm::vec3(0.0, -0.001, 0.0)); 设置了平面的位移（平移操作）。这里的平移**将平面稍微下移**了 -0.001 单位，在 Y 轴上进行微调，这里是为了**确保平面不与球体重叠或穿透**。

接着，plane->setRotation(glm::vec3(0, 90, 90)); 设置了平面的旋转。旋转向量 glm::vec3(0, 90, 90) 指示平面绕 X 轴旋转 90 度，再绕 Y 轴旋转 90 度。这是为了将平面调整为正确的朝向，使其在渲染时与球体或视角对齐，从而**让投影呈现在 y＝0 的平面上**。

最后，plane->setScale(glm::vec3(3.0, 3.0, 3.0)); 通过缩放操作**将平面放大了 3 倍**。glm::vec3(3.0, 3.0, 3.0) 设置了平面在 X、Y 和 Z 轴上的缩放因子。

11. 在 display 方法中绘制阴影

在渲染场景时，阴影的生成依赖于将物体的投影与光源位置相关的变换应用于物体。我们需要通过计算一个阴影矩阵，将光源的坐标映射到物体的表面，生成相应的阴影效果。

代码截图：

```
/* 绘制阴影 */
// 光源位置
glm::vec3 lightPosition = light->getTranslation();
// 阴影矩阵
float lx = lightPosition[0];
float ly = lightPosition[1];
float lz = lightPosition[2];
glm::mat4 shadowMatrix(
    -ly, 0.0f, 0.0f, 0.0f,
    lx, 0.0f, lz, 1.0f,
    0.0f, 0.0f, -ly, 0.0f,
    0.0f, 0.0f, 0.0f, -ly
);
// 计算阴影矩阵
modelMatrix = shadowMatrix * modelMatrix;
// 传递矩阵
    // 将着色器 isShadow 设置为0，表示正常绘制的颜色，如果是1着表示阴影
glUniform1i(mesh_object.shadowLocation, 1);
glUniformMatrix4fv(mesh_object.modelLocation, 1, GL_FALSE, &modelMatrix[0][0]);
glUniformMatrix4fv(mesh_object.viewLocation, 1, GL_FALSE, &camera->viewMatrix[0][0]);
glUniformMatrix4fv(mesh_object.projectionLocation, 1, GL_FALSE, &camera->projMatrix[0][0]);
// 绘制
glDrawArrays(GL_TRIANGLES, 0, mesh->getPoints().size());
```

图 11 绘制物体的阴影

代码说明：

首先，**获取光源的位置** lightPosition，然后将光源的坐标分别赋值给 lx、ly 和 lz，这些值将用来计算一个阴影矩阵。glm::mat4 shadowMatrix 定义了一个 4x4 矩阵，用来生成光源的阴影投影。这个矩阵的元素基于光源的 ly 和 lx、lz 坐标来设置，目的是将物体的顶点从视角投射到地面上，从而得到阴影效果。

然后，将原本的物体模型矩阵与阴影矩阵相乘，**生成一个新的模型矩阵** modelMatrix，代表物体在阴影投影下的位置和形态。这个新的矩阵将应用到物体的顶点，以生成正确的阴影效果。

接着，使用 glUniform1i(mesh_object.shadowLocation, 1) 设置着色器中的 shadowLocation 变量为 1，表示当前**正在绘制的是阴影**，而不是正常的物体。这样做是为了区分阴影与正常的物体渲染。在着色器中会对阴影进行特殊处理，将会使用不同的光照模型或者不进行颜色计算。

接下来，glUniformMatrix4fv 函数将计算得到的 modelMatrix、viewMatrix 和 projectionMatrix **传递**给着色器，用于物体的阴影渲染。modelMatrix 用于描述物体在阴

影中的变换，viewMatrix 和 projectionMatrix 保持不变，用于描述相机的视角和投影设置。

最后，使用 openGL 方法 glDrawArrays **绘制阴影**，使用之前传递给着色器的矩阵和光源数据。glDrawArrays 调用将绘制阴影的顶点数据，产生最终的阴影效果。

12. 修改着色器文件 fshader.glsl 和 vshader.glsl

此处，参照实验 3.4 的代码，使用**片元着色器**来对图形进行着色，绘制效果会比顶点着色器更加平滑。

片元着色器主要的代码修改点：

```
// @TODO: 计算四个归一化的向量 N,V,L,R(或半角向量H)
vec3 N = normalize( norm );
vec3 L = normalize( light.position - position );
vec3 V = normalize( eye_position - position );
vec3 R = reflect(-L, N);
vec3 H = normalize( L + V );

// 环境光分量I_a
vec4 I_a = light.ambient * material.ambient;

// @TODO: Task2 计算系数和漫反射分量I_d
float diffuse_dot = max( dot(L, N), 0.0);
vec4 I_d = diffuse_dot *  light.diffuse * material.diffuse;

// @TODO: Task2 计算系数和镜面反射分量I_s
float specular_dot_pow = pow( max(dot(V, R), 0.0), material.shininess );
// float specular_dot_pow = pow( clamp(dot(V, R), 0.0, 1.0), material.shininess );
vec4 I_s = specular_dot_pow * light.specular * material.specular;

// 注意如果光源在背面则去除高光
if( dot(L, N) < 0.0 ) {
    I_s = vec4(0.0, 0.0, 0.0, 1.0);
}
```

图 12 片元着色器的关键代码

代码说明：

首先，代码计算了四个**归一化**的向量 **N、V、L 和 R**。其中，N 表示法向量，它通过对 norm 向量归一化得到；V 表示从当前表面点 pos 到观察者位置 eye_position 的向量；L 表示从表面点到光源位置 l_pos 的方向向量；R 是反射向量，它通过反射计算函数 reflect 得到，代表光源方向 L 关于法向量 N 的镜面反射方向。

接着，计算环境光分量 I_a。环境光反映了物体表面在阴影区域的亮度，通常是光源和物体材质的环境光颜色的乘积。

然后计算漫反射系数和漫反射光分量 I_d。首先，通过 dot(N,L) 计算法向量 N 和光源方向 L 的点积，代表光源对表面的照射强度，并取其与 0 的最大值，以防止负值影响结果。I_d 是点积结果与光源的漫反射光颜色和材质的漫反射光颜色的乘积，这个分量会模拟光线垂直照射到物体表面时的亮度。

接下来，代码计算镜面反射系数和镜面反射光分量 I_s。通过 dot(R,V) 计算反射向量 R 和观察方向 V 的点积，代表视线对反射方向的对准程度，并使用 pow 函数对其进行幂运算以实现聚光效果，幂次为材质的光泽度 material.shininess。I_s 是该幂结果与光源的镜面反射光颜色和材质的镜面反射光颜色的乘积，这一分量会在视线方向接近反射方向时产生强烈的高光。

最后，代码检查 diffuse_dot 是否小于 0.0，用于**判断光源是否在物体背面**，如果是，则设置镜面反射分量 I_s 为零，这样就不会产生不合理的高光效果。

13. 再次运行程序，观察效果

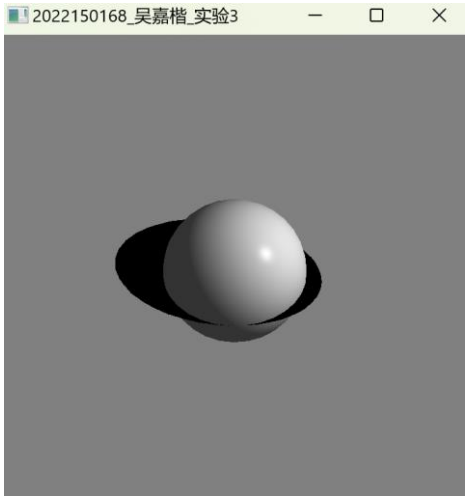

图 13 球与阴影重叠了

运行程序后，我们发现投影平面与球重叠了。这是因为物体的中心点的坐标默认为（0，0，0），存在于投影平面 y＝0 上。为了解决此问题，我们只需要将物体向上平移即可，否则，则会出现这种重叠的现象。

14. 将物体沿 y 轴向上平移

对于球体来说，由于我们球的半径 r = 0.5，为了将球的最低点与投影平面对齐，我们只需要将球体向上平移 0.5 个单位即可。

```
// 设置物体的旋转位移
mesh->setTranslation(glm::vec3(0.0, 0.5, 0.0));
mesh->setRotation(glm::vec3(0, 90.0, 0.0));
mesh->setScale(glm::vec3(1.0, 1.0, 1.0));
```

图 14.1 将球体向上平移 0.5 个单位

对于其他物体来说，我们经过实际测验，得出了以下结论：

（1）将皮卡丘向上平移 0.65 个单位

```
else if(key == GLFW_KEY_A && action == GLFW_PRESS)
{
    std::cout << "read Pikachu.off" << std::endl;
    mesh = new TriMesh();
    mesh->readOff("./assets/Pikachu.off");
    init();
    mesh->setTranslation(glm::vec3(0.0, 0.65, 0.0));
}
```

图 14.2 将皮卡丘向上平移 0.65 个单位

（2）将杰尼龟向上平移 0.6 个单位

```
else if(key == GLFW_KEY_W && action == GLFW_PRESS)
{
    std::cout << "read Squirtle.off" << std::endl;
    mesh = new TriMesh();
    mesh->readOff("./assets/Squirtle.off");
    init();
    mesh->setTranslation(glm::vec3(0.0, 0.6, 0.0));
}
```

图 14.3 将杰尼龟向上平移 0.6 个单位

（3）将粗糙球体向上平移 0.7 个单位

```
else if(key == GLFW_KEY_S && action == GLFW_PRESS)
{
    std::cout << "read sphere_coarse.off" << std::endl;
    mesh = new TriMesh();
    mesh->readOff("./assets/sphere_coarse.off");
    init();
    mesh->setTranslation(glm::vec3(0.0, 0.7, 0.0));
}
```

图 14.4 将粗糙球体向上平移 0.7 个单位

15. 再次运行程序，检查程序 bug

当我们使用鼠标交互来调整光源位置时，发现了以下 bug，如图所示：

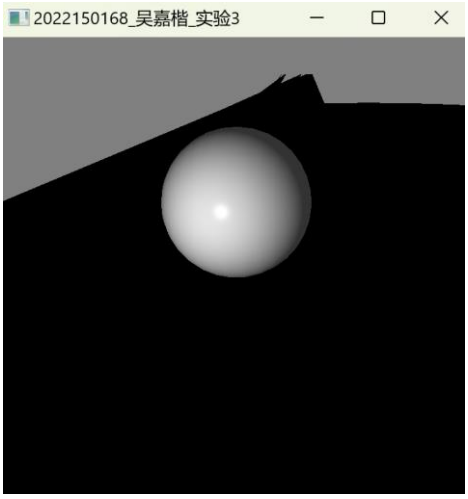

图 15.1 鼠标交互逻辑 bug

这是因为，在初始情况下，我们将光源位置向上平移了 2 个单位，但是，鼠标交互的回调函数逻辑没有同步进行修改，导致使用鼠标点击后，光源没有向上平移 2 个单位，出现在物体表面甚至内部。

因此，我们需要修改鼠标交互的回调函数：

```
void mouse_button_callback(GLFWwindow* window, int button, int action, int mods)
{
    if (button == GLFW_MOUSE_BUTTON_LEFT && action == GLFW_PRESS)
    {
        double x, y;
        glfwGetCursorPos(window, &x, &y);

        float half_winx = WIDTH / 2.0;
        float half_winy = HEIGHT / 2.0;
        float lx = float(x - half_winx) / half_winx;
        float ly = float(HEIGHT - y - half_winy) / half_winy;

        glm::vec3 pos = light->getTranslation();
        pos.x = lx;
        pos.y = ly + 2; // 2作为一个基准值
        if (pos.y < 1)pos.y = 1; // 限制y的最小值

        light->setTranslation(pos);
    }
}
```

图 15.2 修改 mouse_button_callback 函数

16. 运行结果

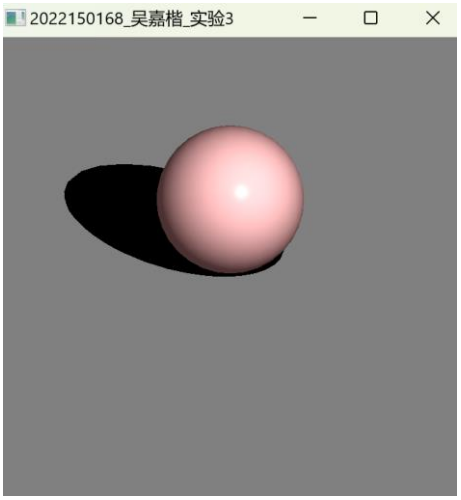

图 16.1 球体投影效果

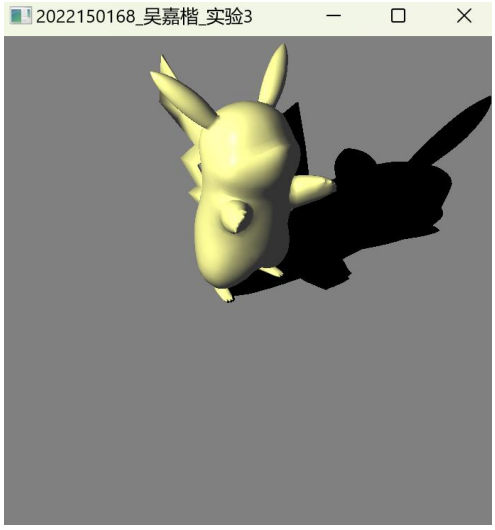

图 16.2 皮卡丘投影效果

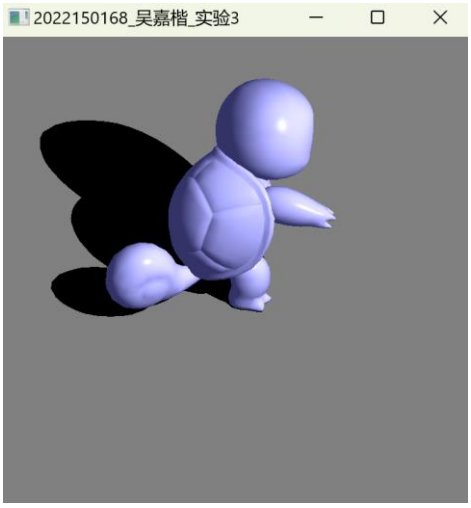

图 16.3 杰尼龟投影效果

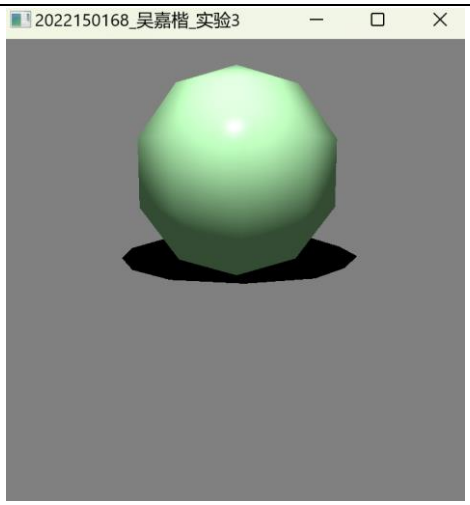

图 16.4 粗糙球体投影效果

深圳大学学生实验报告用纸

实验结论：

在本次计算机图形学实验中，我深入学习并实践了基础的光照模型，特别是**环境光、漫反射和镜面反射**的计算方法。通过实现光照模型的各个组成部分，我更好地理解了如何在计算机图形学中通过光照模拟物体的材质反应和光照效果。此外，我还加深了对向量运算、矩阵变换和图形渲染管线的理解，掌握了如何将理论知识应用到实际的渲染计算中。

实验过程中，我通过实际编写代码和调试，逐步实现了从光源到物体表面的光照计算。我学习了如何利用**法向量、视线向量、光源向量以及反射向量**来计算不同的光照分量，进而影响物体的最终渲染效果。尤其在实现镜面反射时，我加深了对反射向量的理解，并学会了如何利用**点积**和**幂次运算**来模拟真实世界中的高光反射。

通过实验，我还更加熟悉了 OpenGL 的渲染管线，理解了如何在**顶点着色器**和**片元着色器**中传递数据，如何使用 **uniform** 变量控制着色器中的材质、光源和变换矩阵。总的来说，实验帮助我将抽象的图形学理论和具体的编程实践结合起来，提高了我在计算机图形学领域的技术能力。

实验难点：

1. 光照计算中的向量和矩阵运算。

在实验的早期阶段，我遇到了如何正确计算法向量、光源向量和反射向量的问题。例如，反射向量的计算 `vec3 R = reflect(-L, N)` 就是基于光线与法线的夹角。

2. 高光计算中的指数幂

在镜面反射部分的高光计算中，我最初对 `pow(max(dot(V, R), 0.0), material.shininess)` 的作用并没有完全理解。shininess 参数决定了**高光的锐度**，而 `dot(V, R)` 计算的是视线与反射光线的夹角，这一部分的计算直接影响最终的光照效果。

3. 背面光源和阴影处理

在处理光源位于物体背面时，如何正确去除不合理的镜面反射光分量。在物体背面时，反射光线的方向与观察者的视线方向差距较大，导致镜面反射无法产生明显的高光效果。为此，我在计算 dot(L, N) 时加入了条件判断，当该值为负时，我将镜面反射光分量置为零。

4. 处理投影与物体重叠问题

因为物体的中心点的坐标默认为（0，0，0），存在于投影平面 y = 0 上，于是，会出现阴影嵌入在物体中的现象。为了解决此问题，我们只需要将物体向上平移即可

指导教师批阅意见：

成绩评定：

指导教师签字：

年　　月　　日

备注：

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。