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：软件工程（腾班）

指导教师：熊卫丹

报告人：洪子敬 学号：2022155033 班级：腾班

实验时间：2024 年 11 月 20 日 至 2024 年 11 月 27 日

实验报告提交时间：2024 年 11 月 26 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. 场景和模型的绘制

创建 OpenGL 绘制窗口，然后参考实验 2.2 内容读入三维场景文件并绘制（这里默认使用球模型），借用实验 3.3 的多个物体文件，使用键盘事件进行多个模型的切换（详细可见运行时的提示字符或提交的使用手册）；同时为了和后期的阴影颜色区分，这里将窗口背景色设置为灰色。即在 init 函数最后将 RGB 设置改为：

$$\text{glClearColor}(0.5, 0.5, 0.5, 1.0);$$

2. 相机的设置

参考实验 3.1，设置相机并添加交互，这里添加了键盘事件实现了相机角度从不同位置/角度、以正交或透视投影方式观察场景；其中较为重要的是正交投影和透视投影的切换，这里编写了 display_1 函数和 display_2 函数，分别代表正交投影和透视投影，并通过按键“N”和“M”进行切换；两个函数代码详细如下所示：

```
//正交投影
void display_1()
{
    glViewport(0, 0, WIDTH, HEIGHT);

    camera->updateCamera();
    camera->viewMatrix = camera->lookAt(camera->eye, camera->at, camera->up);   // 调用 Camera::lookAt 函数计算视图变换矩阵
    camera->projMatrix = camera->ortho(-camera->scale, camera->scale, -camera->scale, camera->scale, camera->zNear, camera->zFar);

    glBindVertexArray(mesh_object.vao);

    glUseProgram(mesh_object.program);

    glm::mat4 modelMatrix = mesh->getModelMatrix(); // 物体模型的变换矩阵

    // 传递投影变换矩阵
    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]);
    // 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, 0);

    // 将材质和光源数据传递给着色器
    bindLightAndMaterial(mesh, mesh_object, light, camera);
    // 绘制
    glDrawArrays(GL_TRIANGLES, 0, mesh->getPoints().size());
    //绘制阴影
    drawShadow();
}
```

洪子敬
2022155033

```
//透视投影
void display_2()
{
    glViewport(0, 0, WIDTH, HEIGHT);

    camera->updateCamera();
    camera->viewMatrix = camera->lookAt(camera->eye, camera->at, camera->up);   // 调用 Camera::lookAt 函数计算视图变换矩阵
    camera->projMatrix = camera->perspective(camera->fov, camera->aspect, camera->zNear, camera->zFar);

    glBindVertexArray(mesh_object.vao);

    glUseProgram(mesh_object.program);

    glm::mat4 modelMatrix = mesh->getModelMatrix(); // 物体模型的变换矩阵

    // 传递投影变换矩阵
    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]);
    // 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, 0);

    // 将材质和光源数据传递给着色器
    bindLightAndMaterial(mesh, mesh_object, light, camera);
    // 绘制
    glDrawArrays(GL_TRIANGLES, 0, mesh->getPoints().size());
    //绘制阴影
    drawShadow();
}
```

洪子敬
2022155033

至于相机的角度旋转和位置变换，我们延续实验 3.3 的设置，通过 x/shift+x 实现旋转角度的增减，通过 y/shift+y 实现相机角度的旋转，通过“r/shift+r”实现相机和物体之间距离的增减以及通过 f/shift+f 实行透视投影的视场角的增减（在给定时刻可观察到的世界范围）。

3. 添加光照和材质效果

要添加光照，我们需要定义一个 light 光源变量，编写函数 bindLightAndMaterial（与实验 3.3 所给相同），并在 init 函数中调用，对光源的信息、物体的材质以及相机的位置进行传递；其次在 init 函数开始，我们需要对光源的信息进行定义，包括位置、环境光、漫反射和镜面反射等，此外，还需要对物体的旋转位移和材质进行设置；详细定义如下：

```
// 设置光源位置
// 默认y轴方向3.0
light->setTranslation(glm::vec3(3.0, 3.0, 0.0));
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

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

// 设置材质
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(1.0); //高光系数

bindObjectAndData(mesh, mesh_object, vshader, fshader);
```

添加完光照参数后，我们需要在着色器（这里选用顶点着色器进行实验）中进行处理，包括坐标系的转换、三种光分量的计算和颜色的统计；详细如下所示：

```
void main()
{
    vec4 v1 = model * vec4(vPosition, 1.0);
    // 由于model矩阵有可能为阴影矩阵，为了得到正确位置，我们需要做一次透视除法
    vec4 v2 = vec4(v1.xyz / v1.w, 1.0);
    // 考虑相机和投影
    vec4 v3 = projection * view * v2;

    gl_Position = v3;

    norm = vNormal;
    pos = vPosition;

    // 计算模视矩阵
    mat4 ModelView = view * model;

    // 将顶点坐标、光源坐标和法向量转换到相机坐标系
    vec3 pos = (ModelView * vec4(vPosition, 1.0) ).xyz;
    vec3 l_pos = (view * vec4(light.position, 1.0) ).xyz;
    vec3 norm = (ModelView * vec4(vNormal, 0.0)).xyz;

    // 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 V=normalize(eye_position-pos);
    vec3 L=normalize(l_pos-pos);
    vec3 R=reflect(-L,N);

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

    // 计算漫反射系数alpha和漫反射分量I_d
    float diffuse_dot = max(dot(N,L),0.0);
    vec4 I_d = diffuse_dot *  light.diffuse * material.diffuse;

    //计算高光系数beta和镜面反射分量I_s
    float specular_dot_pow = pow(max(dot(R,V),0.0),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);
    // }

    // 合并三个分量的颜色，修正透明度
    color = I_a + I_d + I_s;
    color.a = 1.0;
```

4. 添加阴影效果

参考实验 3.2，将前面定义的光源位置作为投影中心，为计算方便，使用 y=0 平面；在 openGL 对象中添加一个阴影变量 shadowLocation，并在着色器传递数据时将 shadowLocation 与着色器中对应位置进行绑定（bindObjectAndData 函数）；

编写绘制阴影的函数 drawShadow，计算阴影投影矩阵，设置 uniform 传递给着色器，为三维物体生成阴影，详细代码如下：（注意这里的光源直接使用 getTranslation 方法进行获取即可，其他绘制过程与实验 3.2 一致）

```
//绘制阴影
void drawShadow() {
    glBindVertexArray(mesh_object.vao);
    glUseProgram(mesh_object.program);
    //传递投影矩阵
    glm::vec3 light_pos = light->getTranslation();
    float lx = light_pos[0];
    float ly = light_pos[1];
    float lz = light_pos[2];
    glm::mat4 shadowProjMatrix(-ly, 0.0, 0.0, 0.0,
        lx, 0.0, lz, 1.0,
        0.0, 0.0, -ly, 0.0,
        0.0, 0.0, 0.0, -ly);
    glm::mat4 modelMatrix = mesh->getModelMatrix();
    glm::mat4 shadowModelMatrix = shadowProjMatrix * modelMatrix;

    glUniformMatrix4fv(mesh_object.modelLocation, 1, GL_FALSE, &shadowModelMatrix[0][0]);
    glUniformMatrix4fv(mesh_object.viewLocation, 1, GL_FALSE, &camera->viewMatrix[0][0]);
    glUniformMatrix4fv(mesh_object.projectionLocation, 1, GL_FALSE, &camera->projMatrix[0][0]);
    // 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);
    // 绘制
    glDrawArrays(GL_TRIANGLES, 0, mesh->getPoints().size());
```

5. 交互控制光源位置并更新阴影

参考实验 2.1，添加鼠标事件 mouse_button_callback，使得使用鼠标左键可以控制光源位置；使用 setTranslation 方法更新光源位置，更新光照效果，并更新三维物体的阴影；

详细代码如下：（这里使用的是 y=0 的光照平面）

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

        float half_winx = WIDTH / 2.0;
        float half_winz = HEIGHT / 2.0;
        float lx = float(x - half_winx) / half_winx;
        float lz = float(HEIGHT - z - half_winz) / half_winz;

        glm::vec3 pos = light->getTranslation();

        pos.x = lx;
        pos.z = lz;

        //重新展示阴影
        init();
        light->setTranslation(pos);
        display();
    }
}
```

注意：由于阴影变换矩阵是根据光源位置计算的，所以我们不需要更改其他地方，只需要更新光源位置即可；这里要特别注意更新的顺序，先重新初始化图形，再更新光源位置，最后再绘制图形和阴影。

6. 结果展示

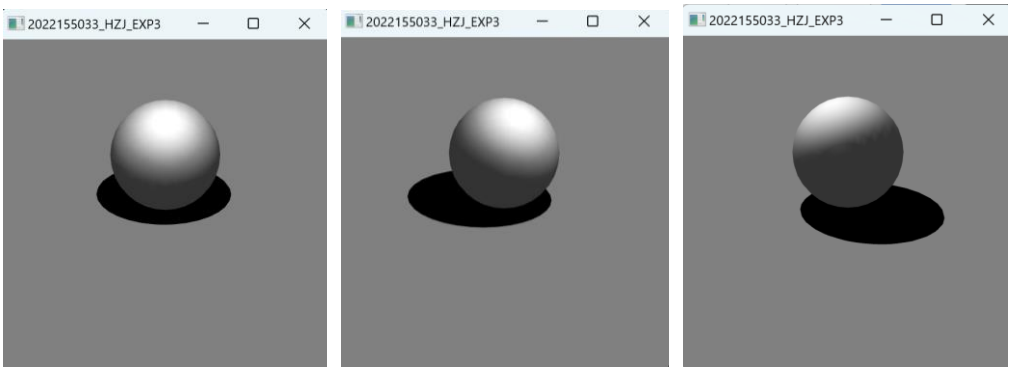
在完成前面的步骤后，我们已经可以成功进行多个物体模型在光照条件下阴影的绘制，总体的键盘事件如下所示：（此外还可以通过点击鼠标左键对光源位置进行转换）

```
Keyboard Usage
ESC:    Exit
H:      Print help message
Camera parameters options:
SPACE:  Reset Camera parameters
x/(shift+x):    Increase/Decrease the rotate angle
y/(shift+y):    Increase/Decrease the up angle
r/(shift+r):    Increase/Decrease the distance between camera and object
f/(shift+f):    Increase/Decrease FOV of perspective projection
n/(shift+n):    change to ortho projection
m/(shift+m):    change to perspective projection
Object Options:
Q:          change to sphere object
A:    change to Pikachu object
W:          change to Squirtle object
S:    change to sphere_coarse object
Materials Options:
-:              Reset material parameters
(shift) + 1/2/3:        Change ambient parameters
(shift) + 4/5/6:        Change diffuse parameters
(shift) + 7/8/9:        Change specular parameters
(shift) + 0:            Change shininess parameters
```

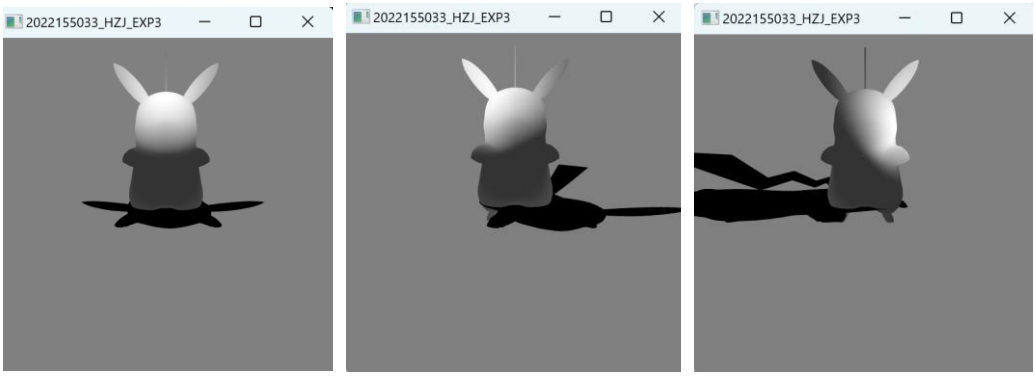
运行时若没看到阴影，可增大仰角形成俯视，此时就可以看到不同角度的阴影效果；

由于可展示效果较多，下面只展示正交投影的部分效果截图，透视投影以及其他按键效果请见提交的 main.exe 文件。

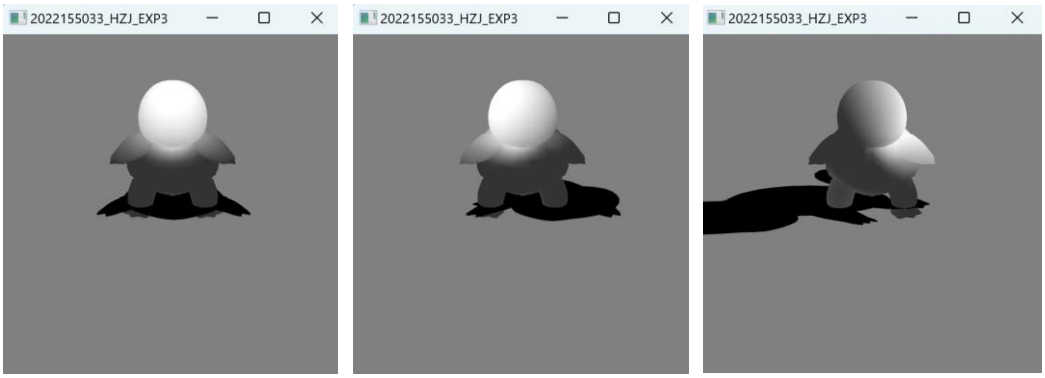
（1）对于 sphere.off 文件模型，不同光源位置下的结果如下所示：



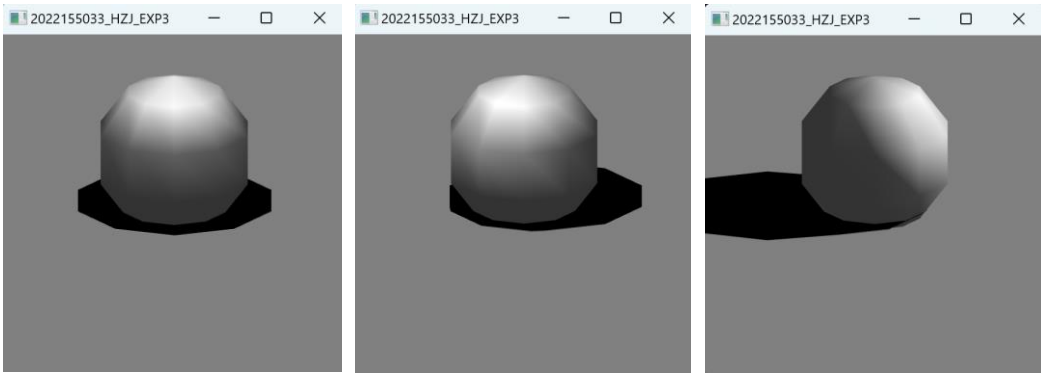
（2）对于 Pikachu.off 文件模型，不同光源位置下的结果如下所示：



（3）对于 Squirtle.off 文件模型，不同光源位置下的结果如下所示：



（4）对于 sphere_coarse.off 文件模型，不同光源位置下的结果如下所示：



深圳大学学生实验报告用纸

实验结论：

本次实验总结使用并结合了前面实验的多个方法，包括 OpenGL 三维场景的读取与绘制方法、设置相机和交互、设置 Phong 光照效果和物体材质效果、在自定义投影平面（本实验使用 y=0 平面）计算阴影投影矩阵来为三维物体生成阴影以及使用鼠标点击（或其他方式）控制光源位置来更新光照和阴影效果，最后成功实现了对多个模型文件的读取和阴影绘制，并实现了鼠标左键实现光源切换以及实时的光照阴影效果更新。本次实验圆满结束。

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。