练 习 题 报 告

课程名称 计算机图形学

项目名称 投影和硬阴影

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

一、 练习目的

1. 熟悉在 OpenGL 中实现正交投影变换。
2. 了解使用投影变换实现场景的硬阴影效果。

二．练习完成过程及主要代码说明

练习要求：完善 Camera.cpp 和 main.cpp，实现正交投影变换和硬阴影效果。

Task-1：在 Camera.cpp 中完善 lookAt 函数、ortho 函数和 perspective 函数。

由于这部分与实验 3.1 实现一致，而且 3.1 实验已经有详细解释，这里只展示代码；不过值得注意的是，**由于 OpenGL 中是列优先，矩阵返回时要转置**。

lookAt 函数代码如下：

```
glm::mat4 Camera::lookAt(const glm::vec4& eye, const glm::vec4& at, const glm::vec4& up)
{
    // @TODO: Task1:请按照实验课内容补全相机观察矩阵的计算
    glm::vec3 n = glm::normalize(glm::vec3(eye - at));//前向向量的计算
    glm::vec3 u = glm::normalize(glm::cross(glm::vec3(up), n));//右向向量
    glm::vec3 v = glm::normalize(glm::cross(n, u));//平面
    //设置旋转和平移部分
    glm::mat4 c = glm::mat4(1.0f);
    c[0][0] = u.x;  c[0][1] = u.y;  c[0][2] = u.z;  c[0][3] = -glm::dot(u, glm::vec3(eye));
    c[1][0] = v.x;  c[1][1] = v.y;  c[1][2] = v.z;  c[1][3] = -glm::dot(v, glm::vec3(eye));
    c[2][0] = n.x;  c[2][1] = n.y;  c[2][2] = n.z;  c[2][3] = -glm::dot(n, glm::vec3(eye));
    c[3][0] = 0.0f; c[3][1] = 0.0f; c[3][2] = 0.0f; c[3][3] = 1.0f;
    c = glm::transpose(c);
    return c;
}                                                                                    HZJ
```

ortho 函数代码如下：

```
glm::mat4 Camera::ortho(const GLfloat left, const GLfloat right,
    const GLfloat bottom, const GLfloat top,
    const GLfloat zNear, const GLfloat zFar)
{
    // @TODO: Task2:请按照实验课内容补全正交投影矩阵的计算
    //创建4*4的单位矩阵
    glm::mat4 c = glm::mat4(1.0f);
    c[0][0] = 2.0f / (right - left);
    c[0][3] = -(right + left) / (right - left);
    c[1][1] = 2.0f / (top - bottom);
    c[1][3] = -(top + bottom) / (top - bottom);
    c[2][2] = -2.0f / (zFar - zNear);
    c[2][3] = -(zFar + zNear) / (zFar - zNear);
    c = glm::transpose(c);
    return c;
}                                                                                    HZJ
```

perspective 函数代码如下：

```
glm::mat4 Camera::perspective(const GLfloat fov, const GLfloat aspect,
    const GLfloat zNear, const GLfloat zFar)
{
    // @TODO: Task3:请按照实验课内容补全透视投影矩阵的计算
    glm::mat4 c = glm::mat4(1.0f);
    GLfloat top = zNear * tan(fov * M_PI / 180 / 2);
    GLfloat right = top * aspect;
    c[0][0] = zNear / right;
    c[1][1] = zNear / top;
    c[2][2] = -(zFar + zNear) / (zFar - zNear);
    c[2][3] = -(2 * zFar * zNear) / (zFar - zNear);
    c[3][2] = -1.0f;
    c[3][3] = 0.0f;
    c = glm::transpose(c);
    return c;
}                                                                                    HZJ
```

Task-2：在 main.cpp 中完善 display()函数，完成对硬阴影部分的绘制。

有实验指导书上的内容可知，假定光源位置在(x_l, y_l, z_l)，三角形任意一个顶点为(x, y, z)，投影到投影平面上的坐标是(x_k, y_k, z_k)（这里投影平面为 y 轴，所以纵坐标为 0），投影前后的坐标关系可以表述为：

$$(x_l, y_l, z_l) = shadowMatrix * (x, y, z)$$

其中 shadowMatrix 可以表示为：

$$\begin{bmatrix} -y_l & x_l & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & z_l & -y_l & 0 \\ 0 & 1 & 0 & -y_l \end{bmatrix}$$

具体实现时只需要在前面已经绘制了一个三角形的基础上，多绘制一个新的三角形，不过要注意此时要按照上面的形式构建一个阴影矩阵，用此矩阵乘以三角形的模型矩阵得到新的阴影模型矩阵，并设置 glUniformli 为 0 表示绘制阴影即可。详细代码如下：

HZJ

```
glm::mat4 shadowMatrix = glm::mat4(1.0f);

// 计算阴影投影矩阵
shadowMatrix[0][0] = -light_position.y;
shadowMatrix[0][1] = light_position.x;
shadowMatrix[1][1] = 0.0f;
shadowMatrix[2][1] = light_position.z;
shadowMatrix[2][2] = -light_position.y;
shadowMatrix[3][1] = 1;
shadowMatrix[3][3] = -light_position.y;
shadowMatrix = glm::transpose(shadowMatrix);
// 绘制三角形阴影
glBindVertexArray(tri_object.vao);
glUseProgram(tri_object.program);

// 传递阴影变换矩阵
// 应用阴影矩阵到模型矩阵
glm::mat4 shadowModelMatrix = shadowMatrix * modelMatrix ;
glUniformMatrix4fv(tri_object.modelLocation, 1, GL_FALSE, &shadowModelMatrix[0][0]);
glUniformMatrix4fv(tri_object.viewLocation, 1, GL_FALSE, &camera->viewMatrix[0][0]);
glUniformMatrix4fv(tri_object.projectionLocation, 1, GL_FALSE, &camera->projMatrix[0][0]);

glUniform1i(tri_object.shadowLocation, 0); // 设置为0表示绘制阴影
// 绘制阴影
glDrawArrays(GL_TRIANGLES, 0, triangle->getPoints().size());
```

实验结果：运行上述代码，可以得到以下效果，练习成功完成：