计算机学院实验报告

实验题目： 实验 2：变换与投影		学号：202300130183
日期：2025/3/10	班级：23 级智能班	姓名：宋浩宇

Email：2367651943@qq.com
202300130183@mail.sdu.edu.cn

实验目的：

本次作业的任务是填写一个旋转矩阵和一个透视投影矩阵。给定三维下三个点 v0(2.0, 0.0, −2.0), v1(0.0, 2.0, −2.0), v2(−2.0, 0.0, −2.0), 你需要将这三个点的坐
标变换为屏幕坐标并在屏幕上绘制出对应的线框三角形 (在代码框架中，我们已经提供了 draw_triangle 函数，所以你只需要去构建变换矩阵即可)。简而言之，我们需要进行模型、视图、投影、视口等变换来将三角形显示在屏幕上。在提供的代码框架中，我们留下了模型变换和投影变换的部分给你去完成。

实验环境介绍：

软件环境：
主系统：Windows 11 家庭中文版 23H2 22631.4317
虚拟机软件：Oracle Virtual Box 7.1.6
虚拟机系统：Ubuntu 18.04.2 LTS
编辑器：Visual Studio Code
编译器：gcc 7.3.0
计算框架：Eigen 3.3.7

硬件环境：
CPU：13th Gen Intel(R) Core(TM) i9-13980HX 2.20 GHz
内存：32.0 GB (31.6 GB 可用)
磁盘驱动器：NVMe WD_BLACKSN850X2000GB
显示适配器：NVIDIA GeForce RTX 4080 Laptop GPU

解决问题的主要思路：

本实验的主要思路如下：

1. 首先我们先解决最简单的绕 z 轴旋转的其次旋转矩阵的计算，由三维空间中的数学推导可以得到新的坐标的值：

$$x^{'} = |OP| \cdot \cos(\alpha + \beta) = |OP| \cdot (cos\alpha \cdot cos\beta - sin\alpha \cdot sin\alpha) = x \cdot cos\beta - y \cdot sin\beta$$
$$y^{'} = |OP| \cdot sin(\alpha + \beta) = |OP| \cdot (sin\alpha \cdot cos\beta + cos\alpha \cdot sin\alpha) = x \cdot sin\beta + y \cdot cos\beta$$
$$z^{'} = z$$

因为我们使用的是三维空间中的其次坐标来表示点的位置，因此我们可以将上述变换关系整理成如下矩阵：

$$\begin{bmatrix} cos\theta & -sin\theta & 0 & 0 \\ sin\theta & cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

由此我们可以得到绕 z 轴旋转矩阵的表达形式，这个矩阵左乘三维空间的齐次变量即可完成变换。

2. 然后我们需要完成透视投影矩阵的计算。

首先，我们知道一个点和它的全局坐标$P(x_u, y_u, z_u)$，我们根据视点的坐标计算出这个点相对于视点的视点坐标$P(x_v, y_v, z_v)$，根据相似关系我们可以求出来这个点在近裁剪平面上的投影的坐标$P(x_v^{'}, y_v^{'}, z_v^{'})$，因为我们默认这个以 z 轴作为摄像机的正方向，因此我们的 z 坐标是不变的，我们可以得到：

$$\frac{y_v^{'}}{-zNear} = \frac{y_v}{z_v}$$

$$\frac{x_v^{'}}{-zNear} = \frac{x_v}{z_v}$$

然后可以计算出：

$$y^{'} = -\frac{y_v \cdot zNear}{z_v}$$
$$x^{'} = -\frac{x_v \cdot zNear}{z_v}$$
$$z^{'} = z$$

然后我们的目的是计算出这个点在我们最终渲染出来的二维画面上的坐标，为了求出这个数据，我们还需要视窗的大小。由$\frac{H}{2} = zNear \cdot tan\left(\frac{fovy}{2}\right)$可以得到高，又因为给出的 aspect 的值，我们还可以得到视窗的宽度，即$\frac{W}{2} = aspect \cdot zNear \cdot tan\left(\frac{fovy}{2}\right)$，至此我们就可以得到这个点在我们的视窗上的坐标，只要计算 x，y 的值即可，即：

$$y'' = \frac{y_v'}{\frac{H}{2}}$$

$$y'' = \frac{y_v'}{zNear \cdot tan(\frac{fovy}{2})}$$

$$y'' = -\frac{y_v}{z_v \cdot tan(\frac{fovy}{2})}$$

对于 x 轴的坐标来说，同理

$$x'' = \frac{x_v'}{\frac{W}{2}}$$

$$x'' = \frac{x_v'}{zNear \cdot aspect \cdot tan(\frac{fovy}{2})}$$

$$x'' = -\frac{x_v}{z_v \cdot aspect \cdot tan(\frac{fovy}{2})}$$

当我们补充上 z 轴的坐标后，这就是一个规范化坐标。
于是我们可以得到 P 点的齐次坐标的一部分：

$$P''(-\frac{x_v}{z_v \cdot aspect \cdot tan(\frac{fovy}{2})}, -\frac{y_v}{z_v \cdot tan(\frac{fovy}{2})}, z_v'', 1)$$

这里的 z 对应的坐标仍然不知道，考虑到齐次坐标的表示方法，我们可以给每一项都乘一个$-z_v$得到。

$$P''(\frac{x_v}{aspect \cdot \tan\left(\frac{fovy}{2}\right)}, \frac{y_v}{\tan\left(\frac{fovy}{2}\right)}, -z_v'' \cdot z_v, -z_v)$$

那么我们就可以得到这个投影矩阵的一部分项：

$$\begin{bmatrix} \frac{1}{aspect \cdot \tan\left(\frac{fovy}{2}\right)} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan\left(\frac{fovy}{2}\right)} & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

我们还需要求出 a 和 b 的值，有如下等式：

$$a \cdot z_v + b = -z_v'' \cdot z_v$$

化简得到：

$$a + \frac{b}{z_v} = -z_v''$$

我们知道z_v的取值范围是[-zFar,zNear]，我们又知道在规范化坐标下$-z_v''$分别对应取值 1 和-1，因此我们可以得到：

$$a+\frac{b}{zNear}=1$$

$$a+\frac{b}{zFar}=-1$$

解这两个方程即可得到：

$$a=\frac{zNear+zFar}{zNear-zFar}$$

$$b=\frac{2\cdot zNear\cdot zFar}{zNear-zFar}$$

至此我们得到整个透视投影矩阵的所有项：

$$\begin{bmatrix} \frac{1}{aspect\cdot\tan\left(\frac{fovy}{2}\right)} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan\left(\frac{fovy}{2}\right)} & 0 & 0 \\ 0 & 0 & \frac{zNear+zFar}{zNear-zFar} & \frac{2\cdot zNear\cdot zFar}{zNear-zFar} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

3. 然后我们需要完成的是提高项问题，即如何获取绕任意过原点的轴旋转的旋转矩阵。只需要一点点空间想象能力，我们就能知道，让一个点绕任意过原点的轴旋转，实际上就是让这个点与这个旋转轴垂直的分向量旋转。对于一个要绕 n 轴旋转θ的点/向量 v，我们有以下分解：

$$v_{\parallel}=(v\cdot\frac{n}{|n|})\cdot\frac{n}{|n|}$$

$$v_{\perp}=v-(v\cdot\frac{n}{|n|})\cdot\frac{n}{|n|}$$

我们用 e 表示 n 的方向向量，可以化简上述式子：

$$v_{\parallel}=(v\cdot e)\cdot \mathrm{e}$$

$$v_{\perp}=v-(v\cdot e)\cdot \mathrm{e}$$

我们将$v_{\perp}$与 e 的叉乘结果记为 w，有：

$$w=e\times v$$

然后我们可以计算出旋转之后的$v_{\perp}^{'}$，即：

$$v_{\perp}^{'}=v_{\perp}cos\theta+wsin\theta$$

因为平行于旋转轴的向量不会变，因此旋转后的向量为：

$$v^{'}=v_{\perp}^{'}+v_{\parallel}$$

展开后得到：

$$v^{'}=(v-(v\cdot e)\cdot e)cos\theta+(e\times v)sin\theta+(v\cdot e)\cdot e$$

这个变换关系得到后，我们接下来要做的就是获取这个变换矩阵。

我们只需要将原本的基向量分别代入这个变换即可获得变换后的坐标系的

基向量。将这组新的基向量组成一个变换矩阵，这个就是我们需要的变换矩阵。即：

对于 i = [1,0,0]有：

$$i^{'} = [e_x^2(1-cos\theta)+cos\theta, e_xe_y(1-cos\theta)+e_zsin\theta, e_xe_z(1-cos\theta)-e_ysin\theta]$$

对于 j = [0,1,0]有：

$$j^{'} = [e_xe_y(1-cos\theta)-e_zsin\theta, e_y^2(1-cos\theta)+cos\theta, e_xe_z(1-cos\theta)+e_xsin\theta]$$

对于 k = [0,0,1]有：

$$k^{'} = [e_xe_z(1-cos\theta)+e_ysin\theta, e_ye_z(1-cos\theta)-e_xsin\theta, e_z^2(1-cos\theta)+cos\theta]$$

至此，我们就可以组合出这个变换矩阵，将其齐次化之后为：

$$\begin{bmatrix} e_x^2(1-cos\theta) & cos\theta, e_xe_y(1-cos\theta)+e_zsin\theta & e_xe_z(1-cos\theta)-e_ysin\theta & 0 \\ e_xe_y(1-cos\theta)-e_zsin\theta & e_y^2(1-cos\theta)+cos\theta & e_xe_z(1-cos\theta)+e_xsin\theta & 0 \\ e_xe_y(1-cos\theta)-e_zsin\theta & e_y^2(1-cos\theta)+cos\theta & e_xe_z(1-cos\theta)+e_xsin\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

我们用这个变换矩阵左乘要变换的向量即可。

实验步骤与实验结果：

1. 首先完成绕 z 轴旋转矩阵的代码编写：

```
Eigen::Matrix4f get_model_matrix(float rotation_angle)
{
    Eigen::Matrix4f model = Eigen::Matrix4f::Identity();
    rotation_angle =  rotation_angle * MY_PI / 180;
    // TODO: Implement this function
    // Create the model matrix for rotating the triangle around the
Z axis.
    // Then return it.
    Eigen::Matrix4f translate = Eigen::Matrix4f::Identity();
    translate <<
std::cos(rotation_angle),-std::sin(rotation_angle),0,0,std::si
n(rotation_angle),std::cos(rotation_angle),0,0,0,0,1,0,0,0,0,1
;
    model = translate;
    return model;
}
```

这段代码会返回绕 z 轴旋转的齐次旋转矩阵。

2. 然后完成透视投影矩阵的代码的编写：

```
Eigen::Matrix4f get_projection_matrix(float eye_fov, float
aspect_ratio,
                                      float zNear, float zFar)
{
    // Students will implement this function

    Eigen::Matrix4f projection = Eigen::Matrix4f::Identity();
    // TODO: Implement this function
    // Create the projection matrix for the given parameters.
    // Then return it.
```

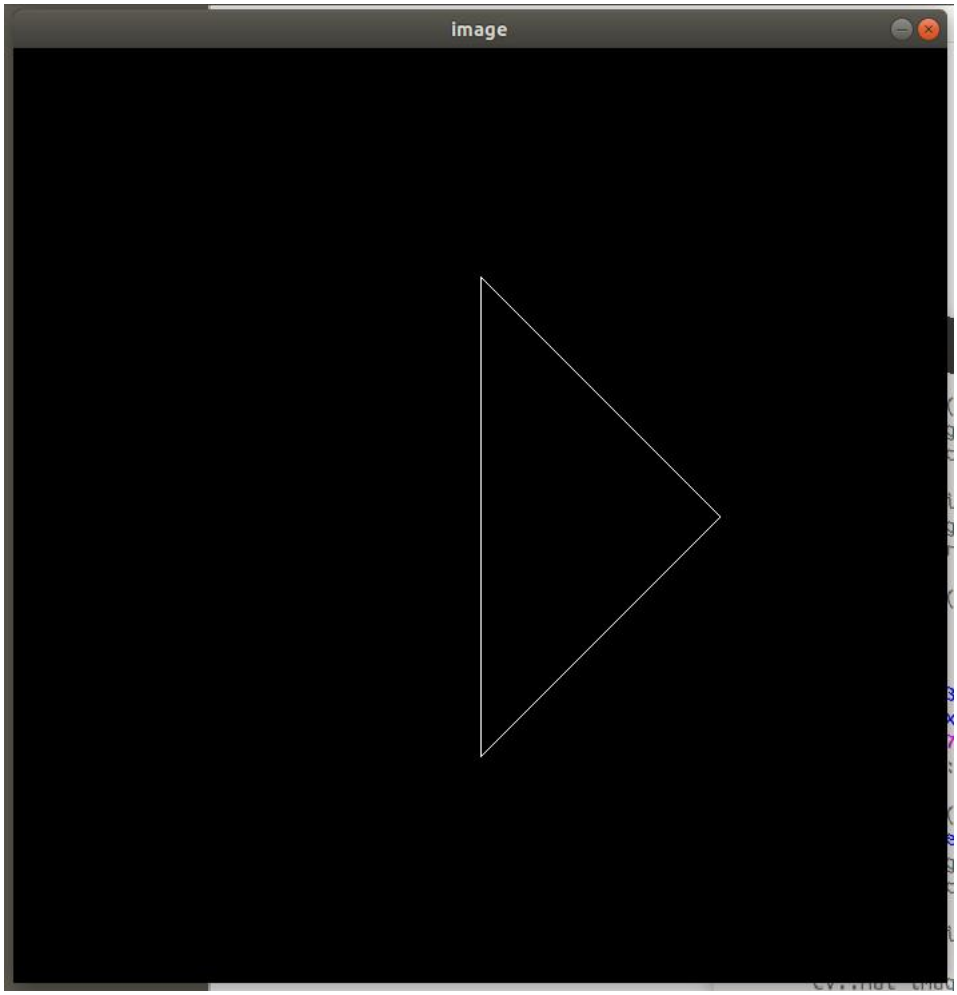
```
    float t = std::tan(eye_fov/2);
    projection << 1/(t * aspect_ratio),0,0,0,
                  0,1/t,0,0,
                  0,0,(zNear + zFar)/(zNear - zFar),(2*zNear*
zFar)/(zNear - zFar),
                0,0,-1,0;
    return projection;
}
```

这段代码会返回一个由垂直方向视场角、裁剪平面的宽高比、摄像机与远、近裁剪平面的距离决定的一个透视投影矩阵。

3. 编译运行

```
cs18@games101vm:~/games101/实验2/代码框架$ make -j4
Scanning dependencies of target Rasterizer
[ 25%] Building CXX object CMakeFiles/Rasterizer.dir/main.cpp.o
[ 50%] Linking CXX executable Rasterizer
[100%] Built target Rasterizer
cs18@games101vm:~/games101/实验2/代码框架$ ./Rasterizer
```

效果为：



该图为按 d 键 9 次之后的结果。

4. 然后是完成提高项问题的绕任意轴的旋转矩阵的代码：

```
Eigen::Matrix4f get_rotation(Eigen::Vector3f axis, float angle)
```

```
{
    angle = angle * MY_PI / 180;
    axis = axis.normalized();
    float tem = 1 - std::cos(angle);
    float cos_angle = std::cos(angle);
    float sin_angle = std::sin(angle);
    Eigen::Matrix4f translate = Eigen::Matrix4f::Identity();
    translate <<
        axis[0] * axis[0] * tem + cos_angle,
        axis[0] * axis[1] * tem - axis[2] * sin_angle,
        axis[0] * axis[2] * tem + axis[1] * sin_angle,
        0,
        axis[1] * axis[0] * tem + axis[2] * sin_angle,
        axis[1] * axis[1] * tem + cos_angle,
        axis[1] * axis[2] * tem - axis[0] * sin_angle,
        0,
        axis[2] * axis[0] * tem - axis[1] * sin_angle,
        axis[2] * axis[1] * tem + axis[0] * sin_angle,
        axis[2] * axis[2] * tem + cos_angle,
        0,
        0,
        0,
        0,
        1;
    return translate;
}
```

为了让画面成功地显示出来，我们还需要更改 main 函数中的内容。

```
    Eigen::Vector3f axis = Eigen::Vector3f::Identity();
    std::cin >> axis[0] >> axis[1] >> axis[2];
    while (key != 27) {
        r.clear(rst::Buffers::Color | rst::Buffers::Depth);

        //r.set_model(get_model_matrix(angle));
    r.set_model(get_rotation(axis,angle));
    r.set_view(get_view_matrix(eye_pos));
        r.set_projection(get_projection_matrix(90, 1, 0.1, 50));

        r.draw(pos_id, ind_id, rst::Primitive::Triangle);
        cv::Mat image(700, 700, CV_32FC3,
r.frame_buffer().data());
        image.convertTo(image, CV_8UC3, 1.0f);
        cv::imshow("image", image);
        key = cv::waitKey(10);
        std::cout << "frame count: " << frame_count++ << '\n';
```

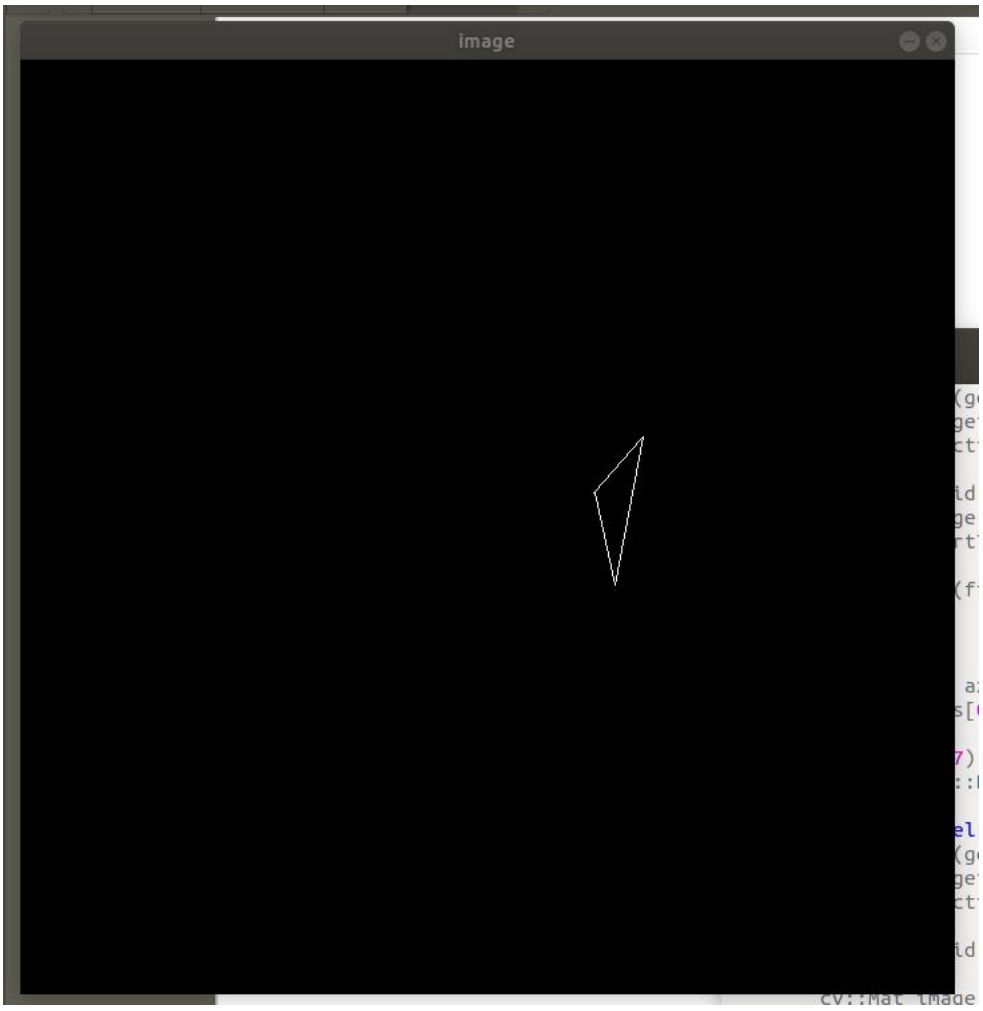
```
        if (key == 'a') {
            angle += 10;
        }
        else if (key == 'd') {
            angle -= 10;
        }
    }
```

我们的修改包括：增加一个旋转轴 axis 的输入、将绕 z 轴旋转矩阵的获取该为任意轴旋转矩阵的获取、更改 pov 为 90；在做完这个修改之后，三角形才能完整的显示在画面中。

编译运行：

```
cs18@games101vm:~/games101/实验2/代码框架$ make -j4
Scanning dependencies of target Rasterizer
[ 25%] Building CXX object CMakeFiles/Rasterizer.dir/main.cpp.o
[ 50%] Linking CXX executable Rasterizer
[100%] Built target Rasterizer
cs18@games101vm:~/games101/实验2/代码框架$ ./Rasterizer
7 4 9
```

获取结果如下图(使用旋转向量(1,4,0)，图为按 d 键 12 次后的结果)：



实验中存在的问题及解决：

问题 1：在绘制图形时，当图像旋转到一定位置后，会出现 Segmentation fault 的运行时错误，怎么解决？

答 1：这个问题的原因似乎是图像中的某一个点旋转到了被绘制的棱台的空间以外的地方，导致 OpenCV 出现了意外的错误。将 fovy 设置为 90°后，图像能完整绘制，并且也不会出现在这个运行时错误了。