实验一

实验目的

- 理解rasterizer.hpp和main.cpp
- 结合课堂内容修改完成main.cpp中的get_model_matrix(float rotation_angle)和get_projection_matrix(float eye_fov, float aspect_ratio, float zNear, float zFar)函数
- 提高内容：实现模型绕任意过圆点的轴的旋转代码

实验过程

1. 理解rasterizer.hpp和main.cpp

   rasterizer.hpp文件的作用是生成渲染器界面与绘制。其成员变量与函数如下：

   - Matrix4f model, view, projection: 三个变换矩阵
   - vecctor< Vector3f >frame_buf: 帧缓冲对象,用于存储需要在屏幕上绘制的颜色数据
   - set_model(const Eigen::Matrix4f& m): 将内部的模型矩阵作为参数传递给光栅化器
   - set_view(const Eigen::Matrix4f& v): 将视图变换矩阵设为内部视图矩阵
   - set_projection(const Eigen::Matrix4f& p): 将内部的投影矩阵设为给定矩阵 p,并传递给光栅化器
   - set_pixel(Vector2f point, Vector3f color): 将屏幕像素点 (x, y) 设为 (r, g, b) 的颜色,并写入相应的帧缓冲区位置

   main.cpp完成整个代码的运行逻辑。首先定义了光栅化器类的实例，然后设置了其必要的变量。然后得到一个带有三个顶点的硬编码三角形。在主函数上,定义了三个分别计算模型、视图和投影矩阵的函数,每一个函数都会返回相应的矩阵。接着,这三个函数的返回值会被 set_model(),set_view() 和 set_projection() 三个函数传入光栅化器中。最后,光栅化器在屏幕上显示出变换的结果。

2. 完成get_model_matrix(float rotation_angle)函数和完成get_projection_matrix(float eye_fov, float aspect_ratio, float zNear, float zFar)函数。

```
Eigen::Matrix4f get_model_matrix(float rotation_angle)
{
    Eigen::Matrix4f model = Eigen::Matrix4f::Identity();

    // TODO: Implement this function
    // Create the model matrix for rotating the triangle around the Z axis.
    // Then return it.
    float r = rotation_angle * M_PI / 180; // 角度制转弧度制
    model << cos(r), -sin(r), 0, 0,
             sin(r), cos(r), 0, 0,
             0, 0, 1, 0,
             0, 0, 0, 1;
    return model;
}
```

实验指出完成绕z轴旋转的矩阵，注意需要齐次化和转换角度制为弧度制，然后根据上课内容写出矩阵公式即可。

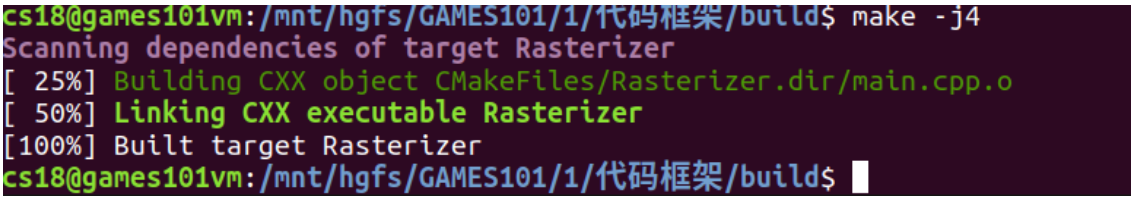
```
Eigen::Matrix4f get_projection_matrix(float eye_fov, float aspect_ratio,
                                      float zNear, float zFar) // 视角 长宽比
近裁减面 远裁减面
{
    // Students will implement this function

    Eigen::Matrix4f projection = Eigen::Matrix4f::Identity();

    // TODO: Implement this function
    // Create the projection matrix for the given parameters.
    // Then return it.
    projection << 1/(aspect_ratio * tan(eye_fov/2.0)), 0, 0, 0,
                0, 1/tan(eye_fov/2.0), 0, 0,
                0, 0, -(zNear + zFar)/(zFar - zNear), -2.0 * zFar *
zNear/(zFar - zNear),
                0, 0, -1.0, 0;
    return projection;
}
```

这里需要完成的是三维透视矩阵的函数，同样需要注意齐次化，矩阵公式可以简单计算后得出。

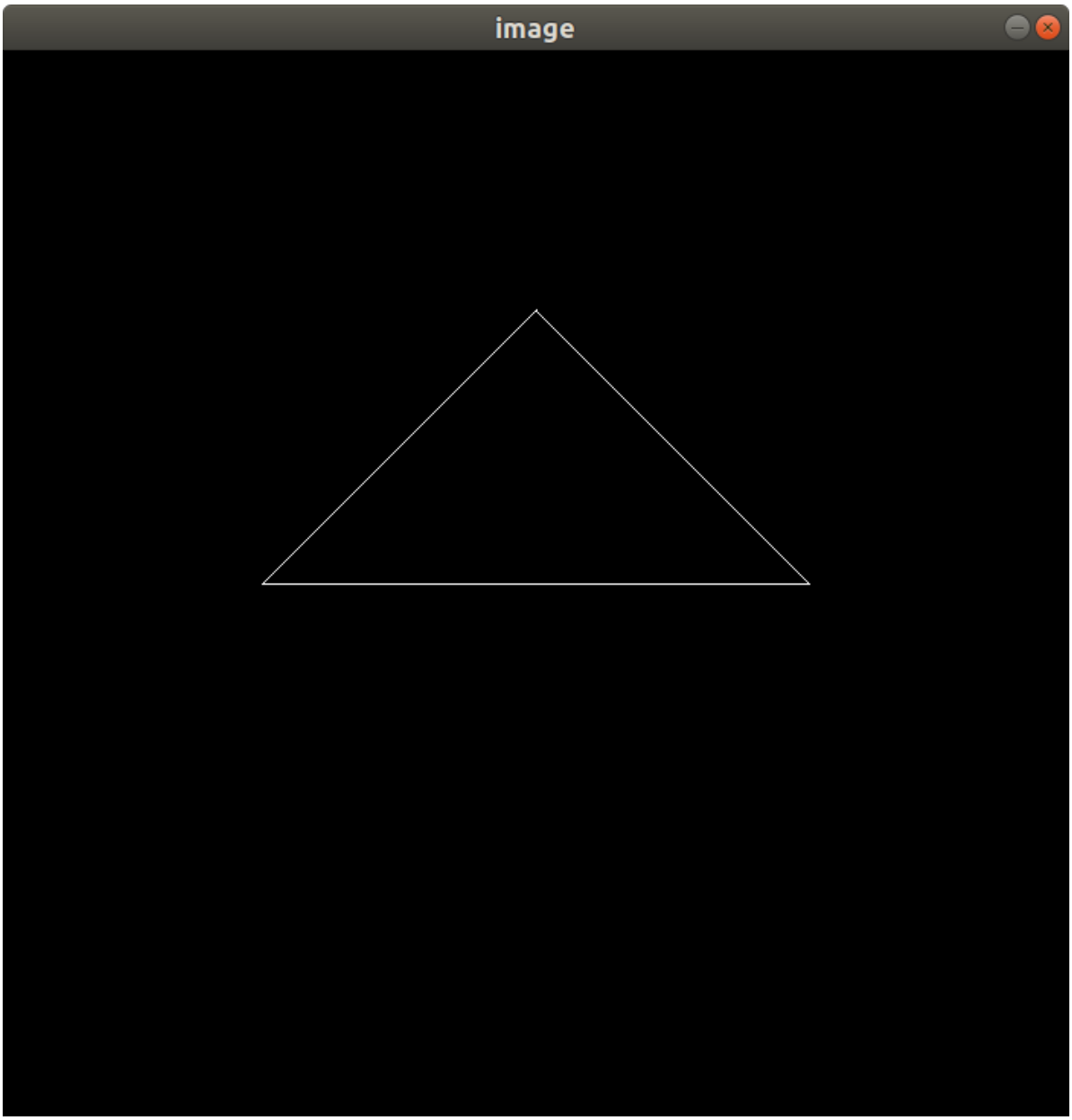
代码执行部分，在终端敲入如下命令，使用Cmake的方法完成编译然后允许：

```
mkdir build
cd build
cmake ..
make -j4
./Rasterizer
```

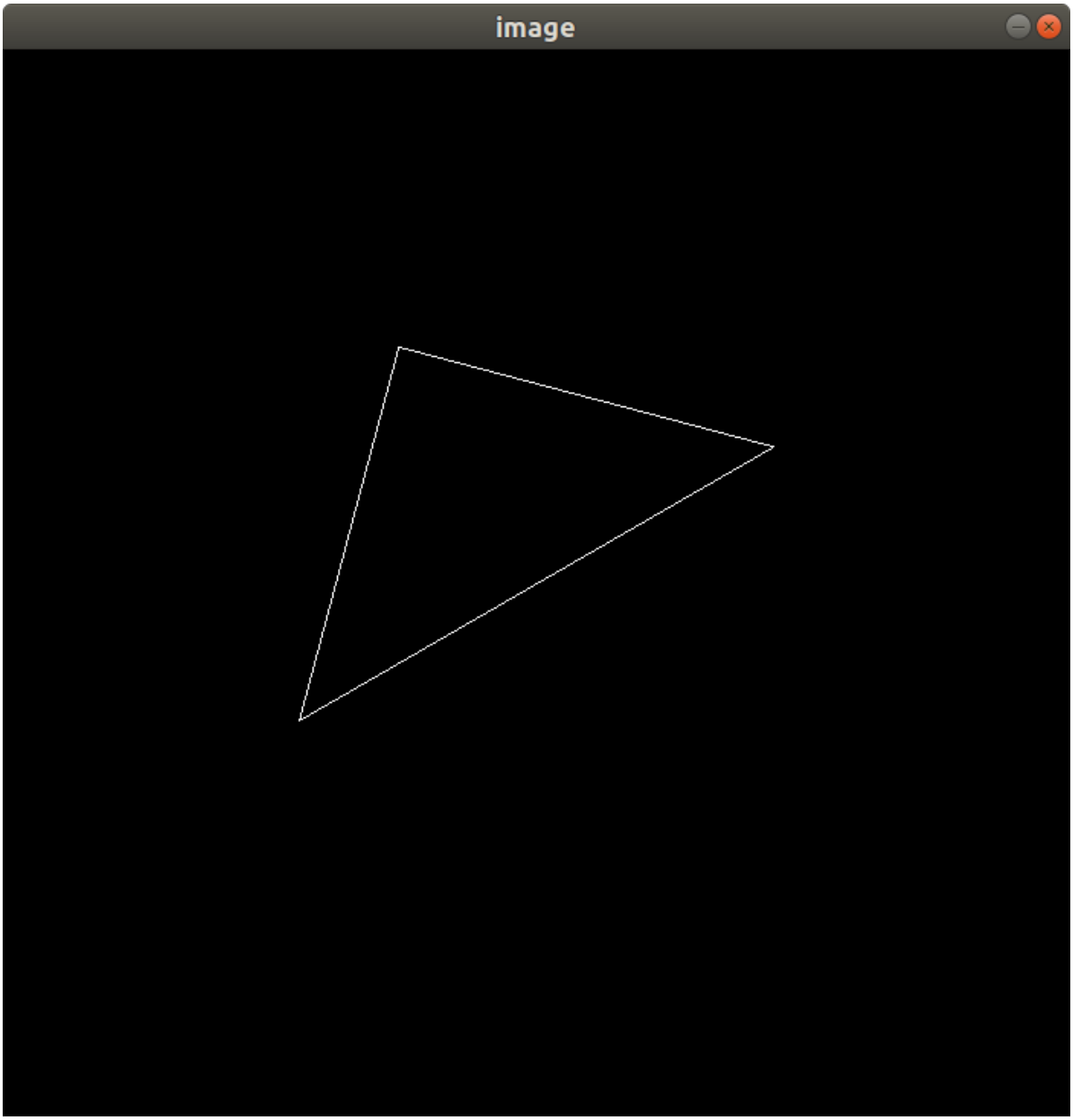
编译成功：



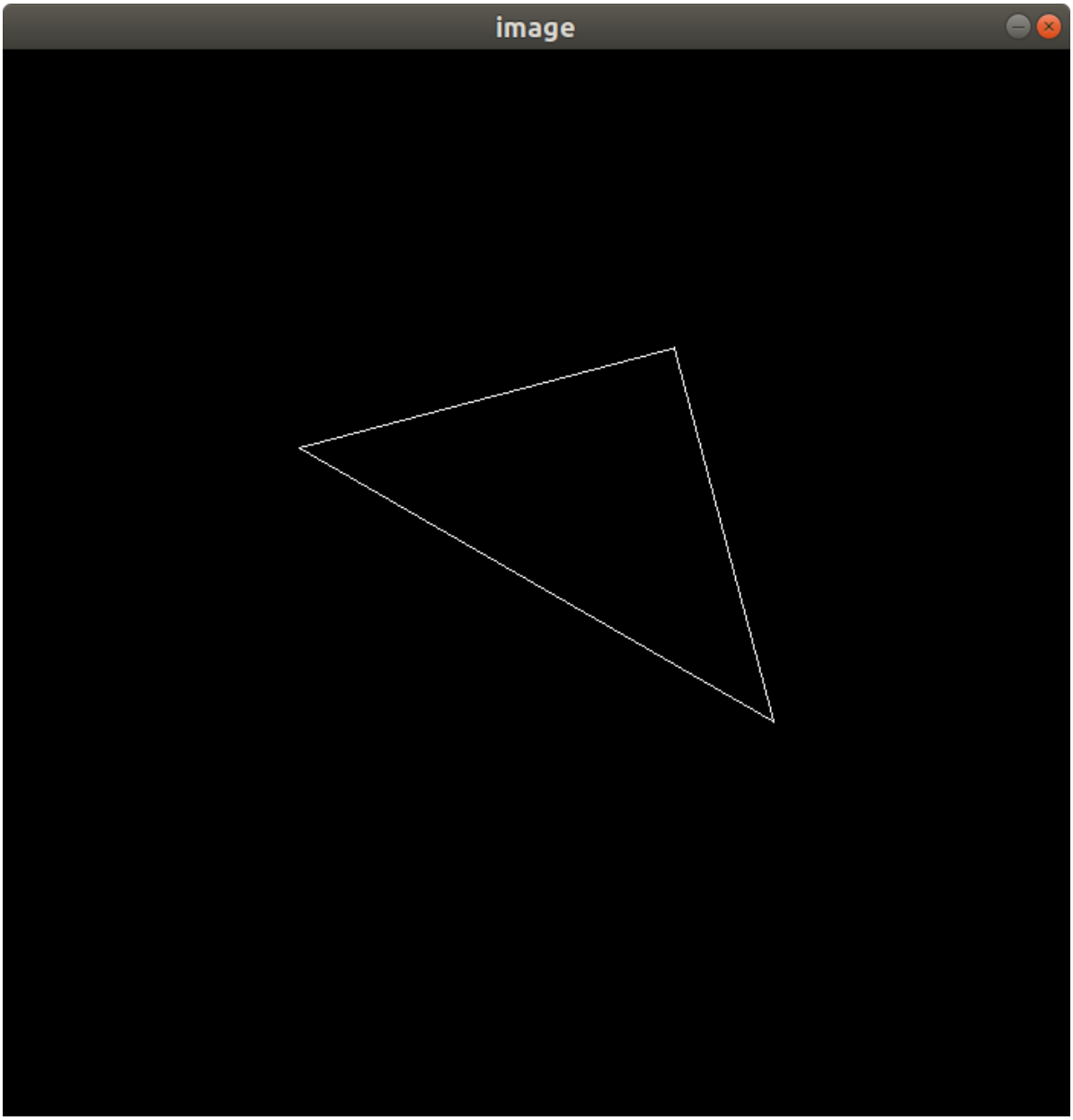
执行结果：



按键盘'a'键向左旋转



按键盘'd'键向右旋转



3. **提高部分**

   在 main.cpp 中构造一个函数,该函数的作用是得到绕任意过原点的轴的旋转变换矩阵。

   自然联想到课上讲到的罗德里格斯旋转公式。不过要注意齐次运算过程中矩阵右下角的值需要为1。

Rodrigues' Rotation Formula

Rotation by angle α around axis $\mathbf{n}$

$$\mathbf{R}(\mathbf{n},\alpha) = \cos(\alpha)\,\mathbf{I} + (1-\cos(\alpha))\,\mathbf{n}\mathbf{n}^T + \sin(\alpha)\begin{pmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{pmatrix}$$

```
Eigen::Matrix4f get_rotation(Vector3f axis, float angle)
{
    double fangle = angle / 180 * MY_PI;
    Eigen::Matrix4f I, N, Rod;
    Eigen::Vector4f axi;
```

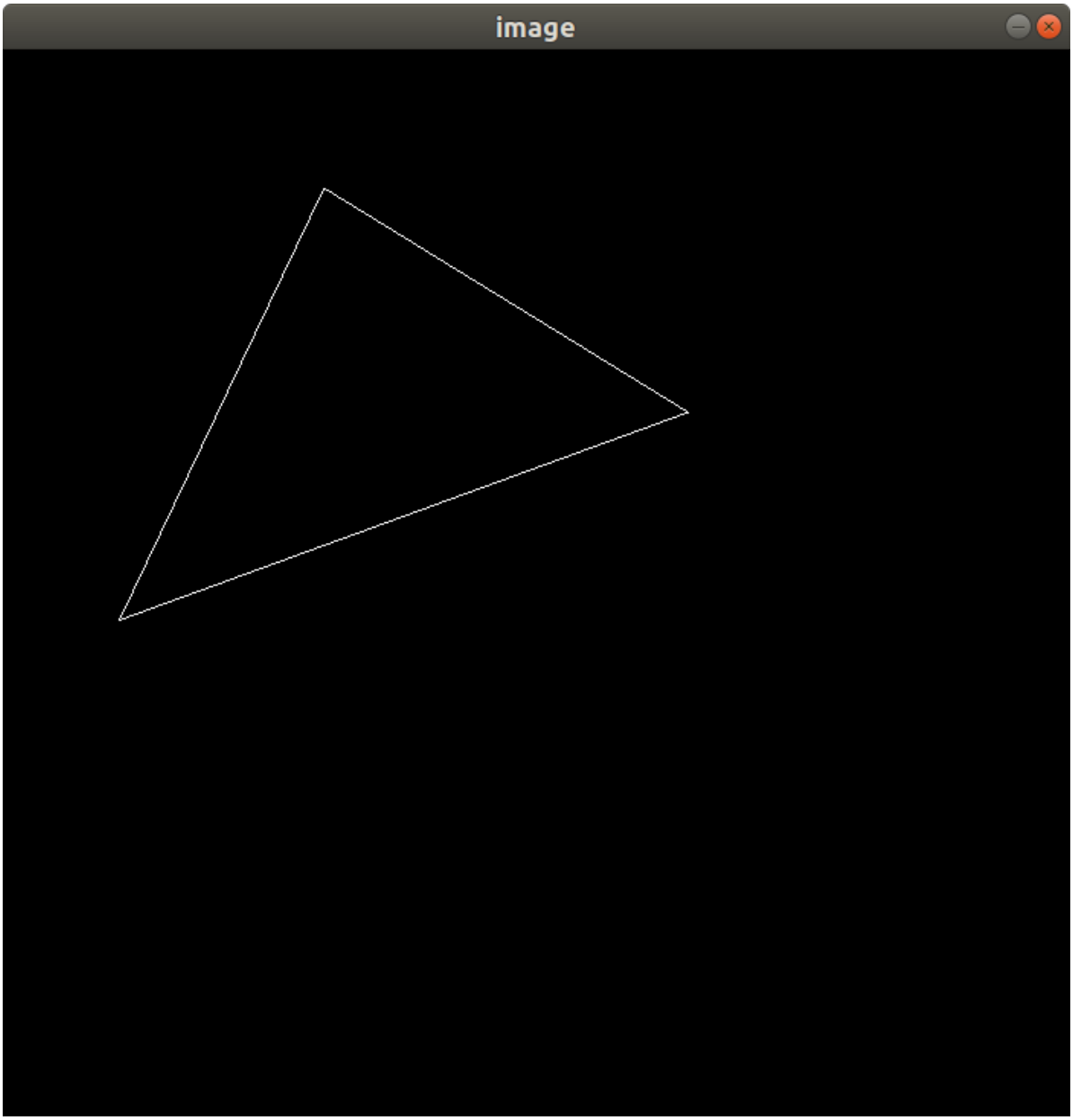
```
    Eigen::RowVector4f taxi; // 定义列向量

    axi << axis.x(), axis.y(), axis.z(), 0;
    taxi << axis.x(), axis.y(), axis.z(), 0;

    I << 1, 0, 0, 0,
         0, 1, 0, 0,
         0, 0, 1, 0,
         0, 0, 0, 1;

    N << 0, -axis.z(), axis.y(), 0,
         axis.z(), 0, -axis.x(), 0,
         -axis.y(), axis.x(), 0, 0,
         0, 0, 0, 1;

    Rod = cos(fangle) * I + (1 - cos(fangle)) * axi * taxi + sin(fangle) * N;
    Rod(3, 3) = 1; // 保证齐次位置是1
    return Rod;
}
```

在main函数中添加过原点的轴，并调用get_rotation函数（需要先注释掉原来默认绕z轴旋转的代码）：

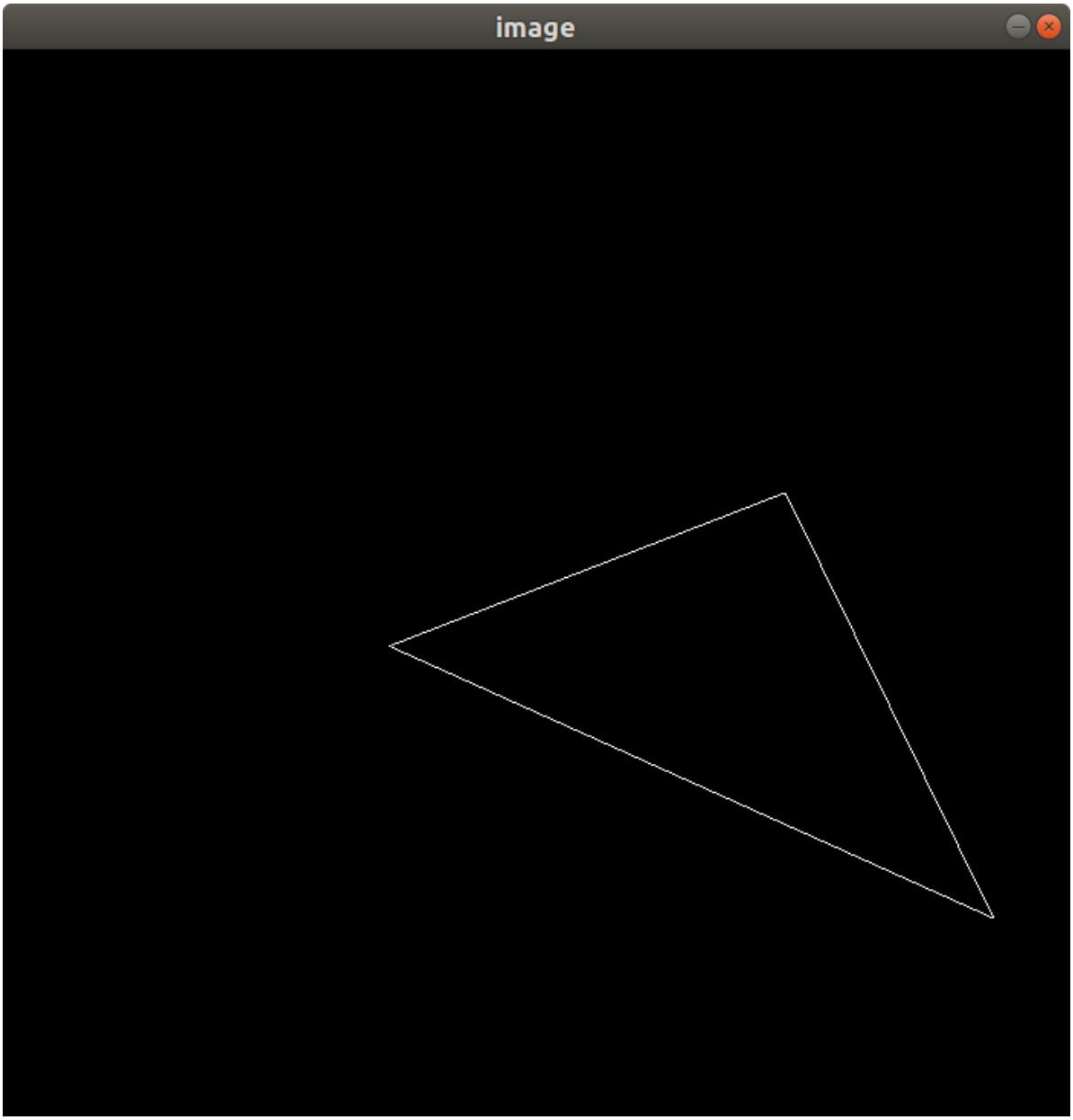
```
// 提高部分
Eigen::Vector3f axis = {1, 1, 1};
r.set_model(get_rotation(axis, angle));
// r.set_model(get_model_matrix(angle));
```

编译执行：

按'a'键后：



按'b'键后：



可以看到在z轴正方向的朝z轴负方向的视角下，三角形绕(1,1,1)旋转会发生形变，由于透视的原因，这是合理的。