

计算机图形学

变换与投影

Assignment 2

姓名：陈家豪

班级：2018 级计科 1 班

学号：18308013

目录

1	作业任务与要求	2
2	实验过程与结果	2
2.1	编译提供的代码并运行	2
2.2	构建透视投影矩阵	4
2.2.1	透视投影矩阵的构建过程	4
2.2.2	透视投影矩阵的代码实现	7
2.2.3	代码编译与运行结果	7
2.3	构建旋转变换矩阵	8
2.3.1	旋转变换矩阵的构建过程	8
2.3.2	旋转变换矩阵的代码实现	10
2.3.3	代码编译与运行结果	10
2.4	对四维齐次坐标的理解	13
3	作业总结与感想	13

1 作业任务与要求

- (1) 编译提供的代码并运行，将运行的结果截图；
- (2) 构建透视投影矩阵，并简述矩阵如何构建。将编译运行结果截图；
- (3) 构建旋转变换矩阵，并简述矩阵如何构建。截图三张旋转结果；
- (4) 谈谈你对四维齐次坐标的理解；

2 实验过程与结果

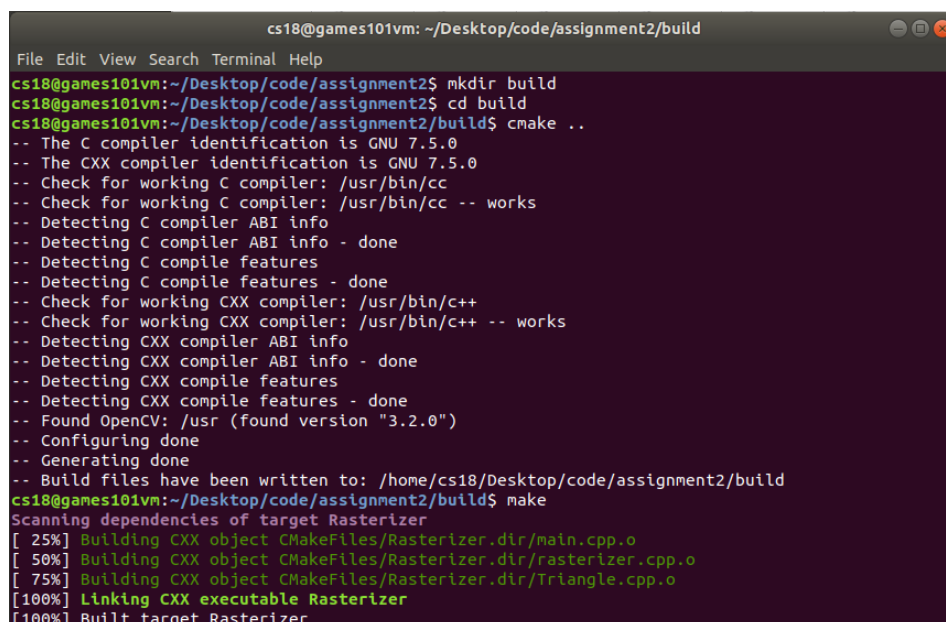
2.1 编译提供的代码并运行

本次实验的第一个任务，是将课程提供的代码放在虚拟机中编译运行。
我们将代码拖入虚拟机中，打开代码所处目录的终端，输入以下命令：

```
1 mkdir build
2 cd build
3 cmake ..
4 make
5 ./Rasterizer
```

即可编译并运行代码文件。

编译结果如图1所示：



```
cs18@games101vm: ~/Desktop/code/assignment2/build
File Edit View Search Terminal Help
cs18@games101vm:~/Desktop/code/assignment2$ mkdir build
cs18@games101vm:~/Desktop/code/assignment2$ cd build
cs18@games101vm:~/Desktop/code/assignment2/build$ cmake ..
-- The C compiler identification is GNU 7.5.0
-- The CXX compiler identification is GNU 7.5.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Found OpenCV: /usr (found version "3.2.0")
-- Configuring done
-- Generating done
-- Build files have been written to: /home/cs18/Desktop/code/assignment2/build
cs18@games101vm:~/Desktop/code/assignment2/build$ make
Scanning dependencies of target Rasterizer
[ 25%] Building CXX object CMakeFiles/Rasterizer.dir/main.cpp.o
[ 50%] Building CXX object CMakeFiles/Rasterizer.dir/rasterizer.cpp.o
[ 75%] Building CXX object CMakeFiles/Rasterizer.dir/Triangle.cpp.o
[100%] Linking CXX executable Rasterizer
[100%] Built target Rasterizer
```

图 1: 原始代码编译结果

运行结果如图2和图3所示：

```
cs18@games101vm: ~/Desktop/code/assignment2/build
File Edit View Search Terminal Help
cs18@games101vm:~/Desktop/code/assignment2/build$ ./Rasterizer
frame count: 0
frame count: 1
frame count: 2
frame count: 3
frame count: 4
frame count: 5
frame count: 6
frame count: 7
frame count: 8
frame count: 9
frame count: 10
frame count: 11
frame count: 12
frame count: 13
frame count: 14
frame count: 15
frame count: 16
frame count: 17
frame count: 18
frame count: 19
frame count: 20
frame count: 21
frame count: 22
frame count: 23
frame count: 24
frame count: 25
frame count: 26
```

图 2: 原始代码运行结果 1

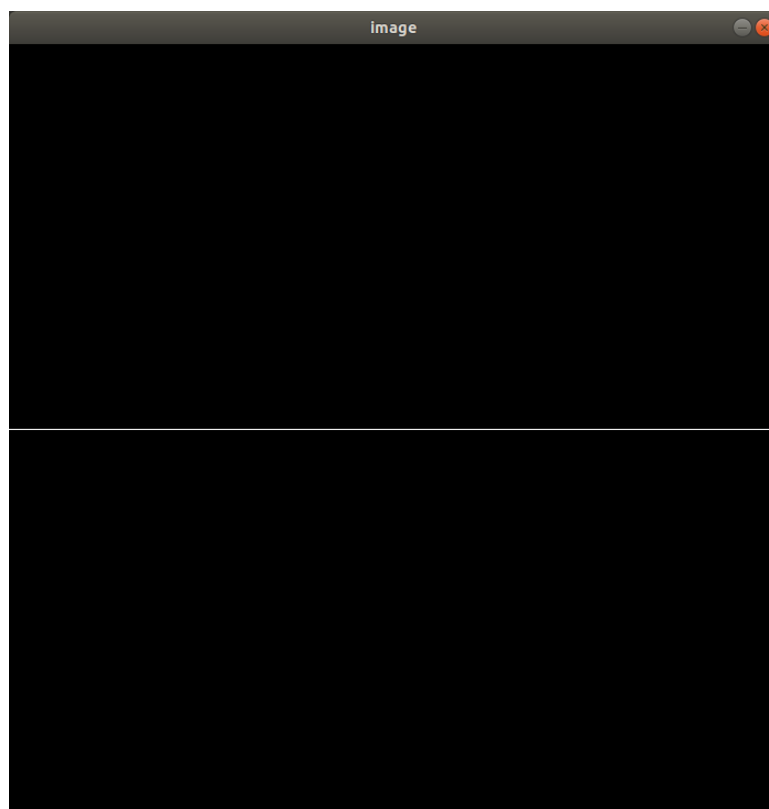


图 3: 原始代码运行结果 2

在程序运行过程中，终端不断弹出计数数字，说明画面在不断刷新。由于没有实现有关矩阵，因此图形窗口几乎没有任何显示，只有一条白线横穿窗口中心。若要退出程序，在图形窗口敲击 `esc` 键即可。

2.2 构建透视投影矩阵

2.2.1 透视投影矩阵的构建过程

透视投影是用中心投影法将形体投射到投影面上，从而实现 3D 物体到 2D 平面的转换。在计算机图形学中，透视投影就是将一个视锥体（frustum）内物体顶点的三维坐标映射到规则观察体（Canonical View Volume, CVV）。如图4所示，视锥体是一个从视点展开的四棱台，而规则观察体是一个立方体，其坐标范围是 $[-1, 1]$ 。在作业文档提供的文章中，这个立方体也称为 Normalized Device Coordinates(NDC)。

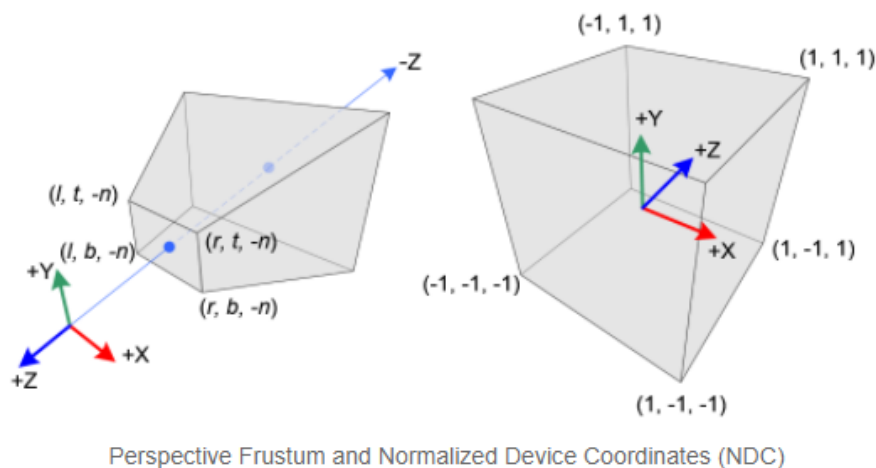


图 4: 视锥体和规则观察体

以图4所示的坐标轴为基准，假设我们已有位于视锥体内的三维坐标 $P(x, y, z)$ ，那么我们应如何将其转换为 NDC 中的坐标 (x', y', z') ？我们首先考虑坐标 x 的转换。

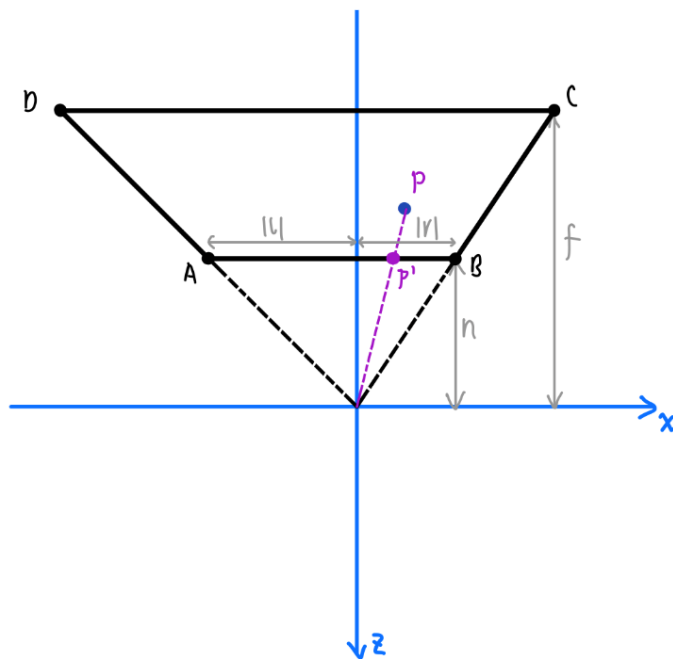


图 5: 视锥体俯视图

投影的目的就是把摄像机空间的三维点投影到近平面上。如图5所示，在视锥体的俯视图中，点 P 在视锥体的近平面上的投影点为 P' 。设点 A 坐标为 $(l, _, -n)$ ，点 B 坐标为 $(r, _, -n)$ ，点 P 坐标为

(x, y, z) ，则对于点 P' 坐标 $(x_{P'}, y_{P'}, -n)$ ，根据相似三角形性质等几何关系，我们有

$$\frac{x}{-z} = \frac{x_{P'}}{n}$$

从而得到：

$$x_{P'} = -\frac{nx}{z}$$

得到 P' 的 x 坐标后，我们将其映射到 $[-1,1]$ 范围上。对映射点的 x 轴坐标 x' ，我们有

$$\frac{x_{P'} - l}{r - l} = \frac{x' - (-1)}{1 - (-1)}$$

化简得

$$x' = \frac{2x_{P'} - r - l}{r - l} = \frac{-2nx - (r + l)z}{(r - l)z} \quad (1)$$

接下来对坐标 y 转换。

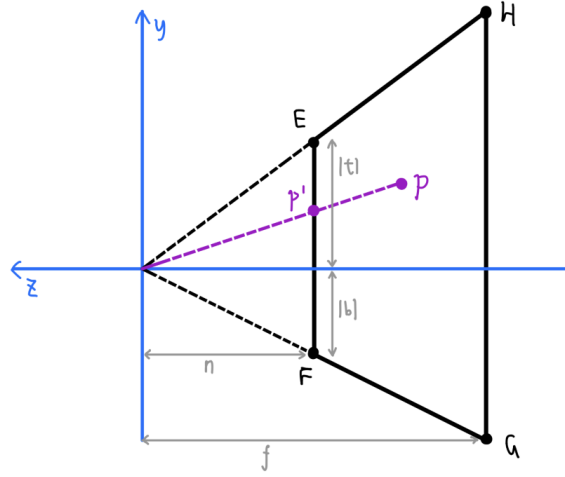


图 6: 视锥体侧视图

同样，在视锥体的侧视图6中，设点 E 坐标为 $(_, t, -n)$ ，点 F 坐标为 $(_, b, -n)$ ，点 P 坐标为 (x, y, z) ，点 P' 坐标为 $(x_{P'}, y_{P'}, -n)$ ，根据几何关系，我们可以得到：

$$\frac{y}{-z} = \frac{y_{P'}}{n}$$

化简得

$$y_{P'} = -\frac{ny}{z}$$

将 P' 的 y 坐标映射到 $[-1,1]$ ，对映射点的 y 坐标 y' ，我们有

$$\frac{y_{P'} - b}{t - b} = \frac{y' - (-1)}{1 - (-1)}$$

化简得

$$y' = \frac{2y_{P'} - t - b}{t - b} = \frac{-2ny - (t + b)z}{(t - b)z} \quad (2)$$

因此，当把原坐标 (x, y, z) 映射到 (x', y', z') 时，我们有

$$\begin{cases} x' = \frac{-2nx - (r + l)z}{(r - l)z} = \frac{2nx + (r + l)z}{(r - l)(-z)} \\ y' = \frac{-2ny - (t + b)z}{(t - b)z} = \frac{2ny + (t + b)z}{(t - b)(-z)} \end{cases} \quad (3)$$

原始图像的坐标为四维齐次坐标 $(x, y, z, 1)$ ，我们需要一个四维透视投影矩阵，使得其能转换成投影点的四维齐次坐标：

$$\begin{bmatrix} x'' \\ y'' \\ z'' \\ w'' \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (4)$$

其中根据齐次坐标的性质，我们有

$$x' = \frac{x''}{w''}, y' = \frac{y''}{w''}, z' = \frac{z''}{w''}$$

由公式3可知， x' 和 y' 具有共同的除数 $-z$ 。故我们可以大胆假设 $w'' = -z$ ，则式 4 可改为：

$$\begin{bmatrix} x'' \\ y'' \\ z'' \\ w'' \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ 0 & 0 & -1 & 0 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

那么对于 $x'' = w''x'$ ，我们有

$$\begin{cases} x'' = \frac{2nx+(r+l)z}{r-l} = \frac{2n}{r-l}x + \frac{r+l}{r-l}z \\ x'' = a_{11}x + a_{12}y + a_{13}z + a_{14} \end{cases} \Rightarrow \begin{cases} a_{11} = \frac{2n}{r-l} \\ a_{12} = 0 \\ a_{13} = \frac{r+l}{r-l} \\ a_{14} = 0 \end{cases} \quad (5)$$

对于 $y'' = w''y'$ ，我们有

$$\begin{cases} y'' = \frac{2ny+(t+b)z}{t-b} = \frac{2n}{t-b}y + \frac{t+b}{t-b}z \\ y'' = a_{21}x + a_{22}y + a_{23}z + a_{24} \end{cases} \Rightarrow \begin{cases} a_{21} = 0 \\ a_{22} = \frac{2n}{t-b} \\ a_{23} = \frac{t+b}{t-b} \\ a_{24} = 0 \end{cases} \quad (6)$$

我们最后考虑坐标 z 的转换。由上述过程可知，

$$z' = \frac{z''}{w''} = \frac{a_{31}x + a_{32}y + a_{33}z + a_{34}}{-z}$$

由几何关系可知，坐标 z 的转换就是从范围 $[-f, -n]$ 映射到 $[-1, 1]$ ，与坐标 x 、 y 无关，故 $a_{31} = a_{32} = 0$ 。当 $z = -f$ 时 $z' = 1$ ， $z = -n$ 时 $z' = -1$ ，我们可得到如下方程：

$$z' = \frac{a_{33}z + a_{34}}{-z} \Rightarrow \begin{cases} 1 = \frac{-a_{33}f + a_{34}}{f} \\ -1 = \frac{-a_{33}n + a_{34}}{n} \end{cases} \Rightarrow \begin{cases} a_{33} = \frac{n+f}{n-f} \\ a_{34} = \frac{2nf}{n-f} \end{cases} \quad (7)$$

综上所述，透视投影变换为

$$\begin{bmatrix} x'' \\ y'' \\ z'' \\ w'' \end{bmatrix} = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{n+f}{n-f} & \frac{2nf}{n-f} \\ 0 & 0 & -1 & 0 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (8)$$

在本次实验中，视锥体相对于 x 轴和 y 轴是对称的，且函数只提供了参数 eye_fov （视域范围角度）、 $aspect_ratio$ （屏幕即近平面的宽高比）、 $zNear$ （近平面到原点的距离）和 $zFar$ （远平面到原点的距离）。故由几何关系我们有

$$\begin{aligned} t &= -b = zNear * \tan \frac{eye_fov}{2} \\ r &= -l = aspect_ratio * t \\ n &= zNear \\ f &= zFar \end{aligned}$$

代入透视投影矩阵得

$$\mathbf{A} = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{n+f}{n-f} & \frac{2nf}{n-f} \\ 0 & 0 & -1 & 0 \end{bmatrix} = \begin{bmatrix} \frac{1}{aspect_ratio * \tan \frac{eye_fov}{2}} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan \frac{eye_fov}{2}} & 0 & 0 \\ 0 & 0 & \frac{zNear+zFar}{zNear-zFar} & \frac{2zNear*zFar}{zNear-zFar} \\ 0 & 0 & -1 & 0 \end{bmatrix} \quad (9)$$

2.2.2 透视投影矩阵的代码实现

实现代码如下所示，按照2.2.1的思路构建即可。

```
1 Eigen::Matrix4f get_projection_matrix(float eye_fov, float aspect_ratio,
2                                     float zNear, float zFar)
3 {
4     // Students will implement this function
5     // TODO: Implement this function
6     // Create the projection matrix for the given parameters.
7     // Then return it.
8     Eigen::Matrix4f projection = Eigen::Matrix4f::Identity();
9     float eye_fov_radian=MY_PI*eye_fov/180; // 弧度制角度
10    float t=tan(eye_fov_radian/2)*zNear;    // 半高度
11    float r=t*aspect_ratio;                  // 半宽度
12    float n=zNear;                           // 近平面距离
13    float f=zFar;                            // 远平面距离
14    projection << n/r, 0.0, 0.0, 0.0,
15                  0.0, n/t, 0.0, 0.0,
16                  0.0, 0.0, -(f+n)/(f-n), -2*f*n/(f-n),
17                  0.0, 0.0, -1.0, 0.0;
18    return projection;
19 }
```

2.2.3 代码编译与运行结果

代码编译结果如图7所示，可见编译成功。


```
cs18@games101vm: ~/Desktop/code/assignment2/build
File Edit View Search Terminal Help
cs18@games101vm:~/Desktop/code/assignment2/build$ make
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:~/Desktop/code/assignment2/build$
```

图 7: 代码编译结果

代码运行结果如图8所示，可见图形窗口绘制出了一个三角形。

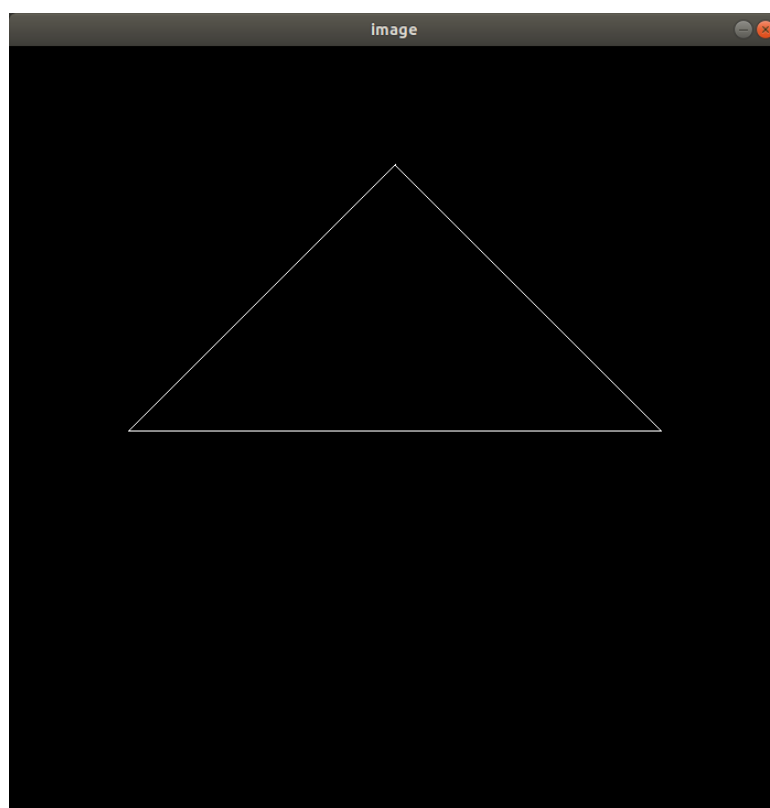


图 8: 代码运行结果

2.3 构建旋转变换矩阵

2.3.1 旋转变换矩阵的构建过程

该部分需要我们构建围绕 z 轴的旋转变换矩阵。

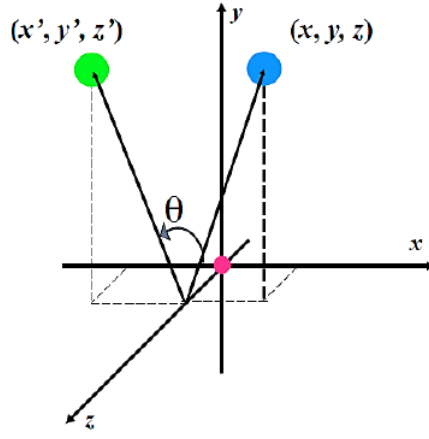


图 9: 物体顶点绕 z 轴旋转

如图9所示，当顶点 (x, y, z) 绕 z 轴旋转 θ 变成顶点 (x', y', z') 时，显然 z 坐标不会变化， $z' = z$ 。要获取其他两个坐标的值，我们可以将两个顶点对应的向量投影到 x 轴和 y 轴所在的平面上进行求解。

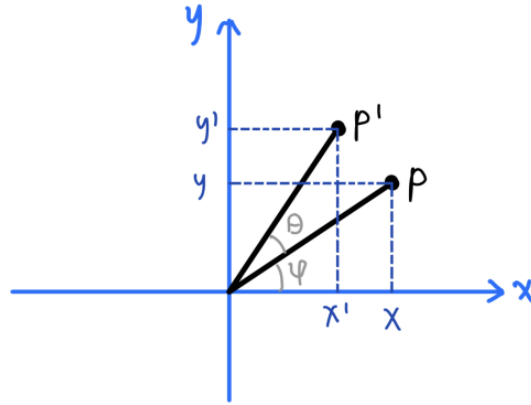


图 10: 物体顶点绕 z 轴旋转

如图10所示，设向量长度为 $|R|$ ，由几何关系可得：

$$\begin{cases} x = |R| \cos \phi \\ y = |R| \sin \phi \\ x' = |R| \cos(\phi + \theta) = |R|(\cos \theta \cos \phi - \sin \theta \sin \phi) \\ y' = |R| \sin(\phi + \theta) = |R|(\sin \theta \cos \phi + \cos \theta \sin \phi) \end{cases}$$

化简公式，我们可以得到：

$$\begin{cases} x' = \cos \theta x - \sin \theta y \\ y' = \sin \theta x + \cos \theta y \end{cases}$$

由此，我们可以得到旋转变换公式：

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

2.3.2 旋转变换矩阵的代码实现

实现代码如下所示，按照2.3.1的思路构建即可。

```
1 Eigen::Matrix4f get_model_matrix(float rotation_angle)
2 {
3     Eigen::Matrix4f model = Eigen::Matrix4f::Identity();
4
5     // TODO: Implement this function
6     // Create the model matrix for rotating the triangle around the Z axis.
7     // Then return it.
8     float rotation_angle_radian=MY_PI*rotation_angle/180;// 转换成弧度制
9     float angle_cos=cos(rotation_angle_radian);           // cos 值
10    float angle_sin=sin(rotation_angle_radian);           // sin 值
11    model << angle_cos, -angle_sin, 0.0, 0.0,
12              angle_sin,  angle_cos, 0.0, 0.0,
13              0.0,      0.0, 1.0, 0.0,
14              0.0,      0.0, 0.0, 1.0;
15    return model;
16 }
```

2.3.3 代码编译与运行结果

代码编译结果如图11所示，可见编译成功。

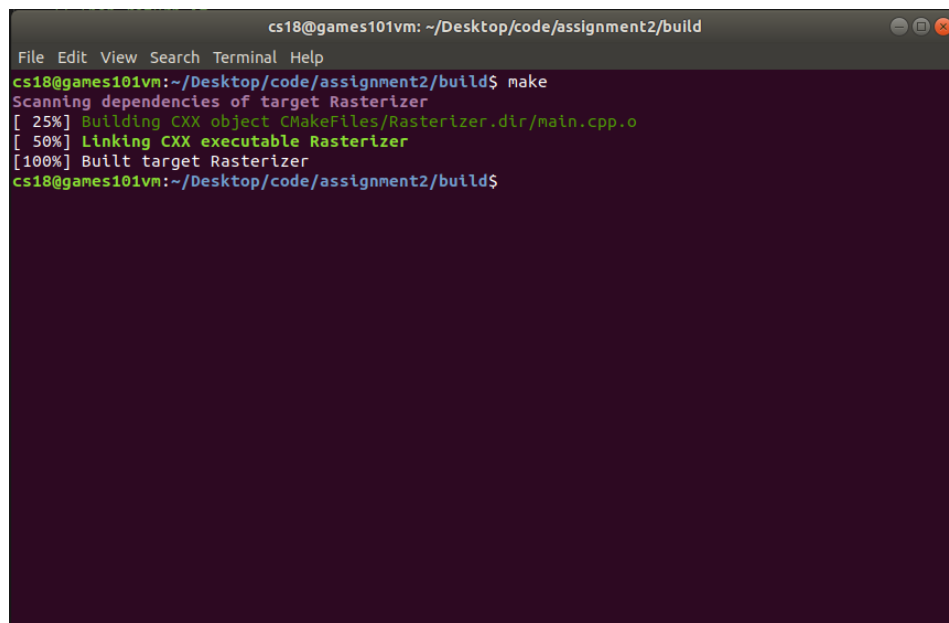
A terminal window titled 'cs18@games101vm: ~/Desktop/code/assignment2/build' showing the output of a 'make' command. The output indicates that the target 'Rasterizer' was successfully built. The steps shown are: 'Scanning dependencies of target Rasterizer', '[25%] Building CXX object CMakeFiles/Rasterizer.dir/main.cpp.o', '[50%] Linking CXX executable Rasterizer', and '[100%] Built target Rasterizer'. The prompt returns to 'cs18@games101vm:~/Desktop/code/assignment2/build\$'.

图 11: 代码编译结果

代码运行结果如图12所示，出现一个三角形。按动 A 键或 D 键可以旋转三角形，如图13、14和15所示。

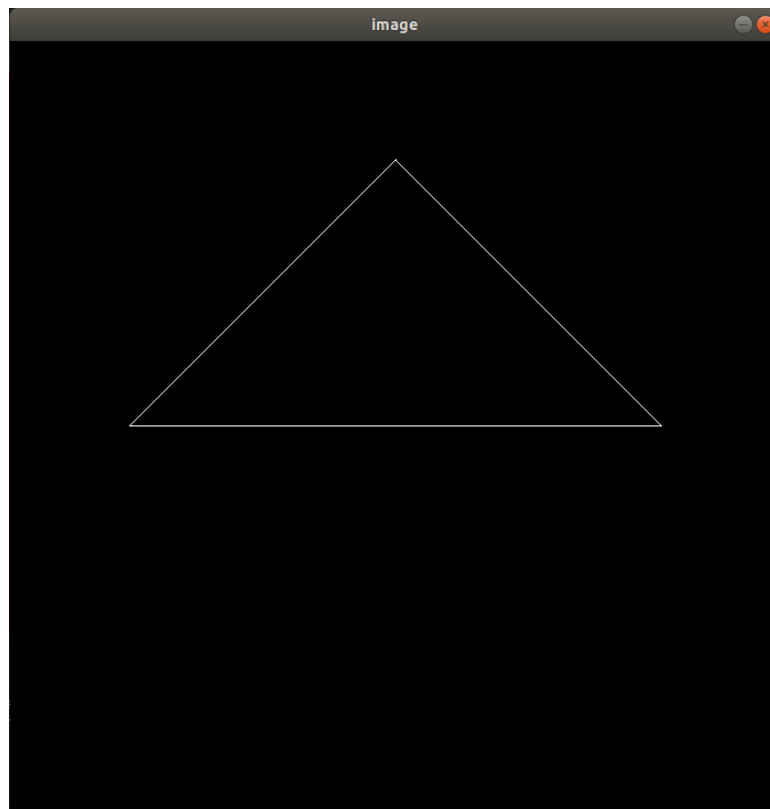


图 12: 代码运行结果

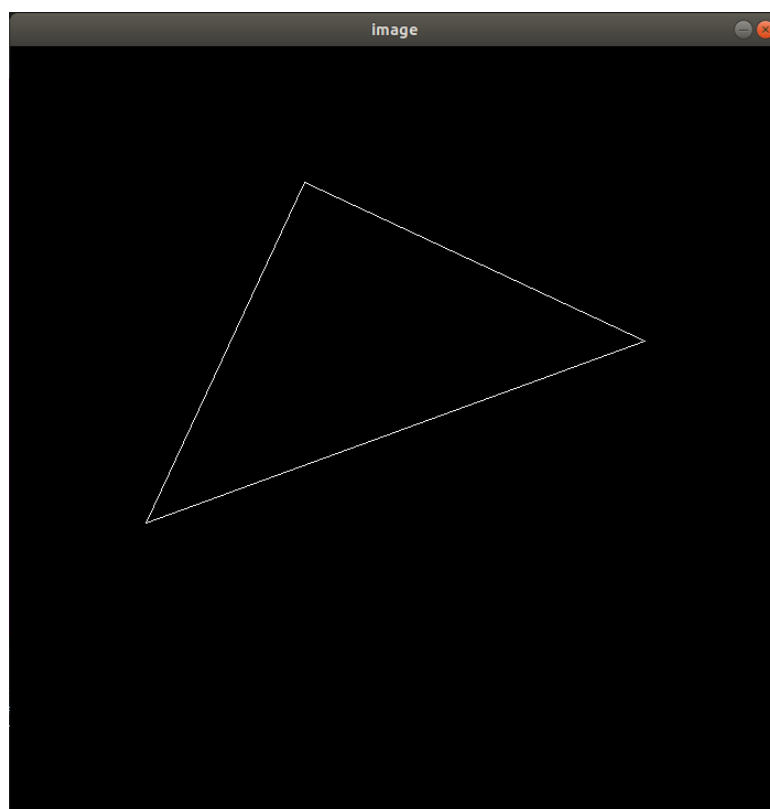


图 13: 旋转三角形 (1)

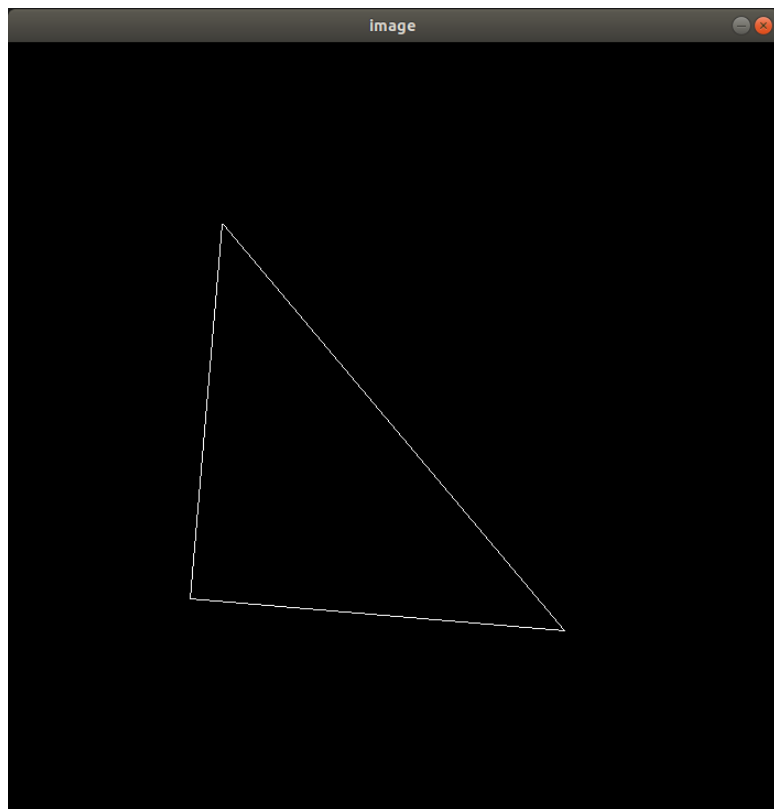


图 14: 旋转三角形 (2)

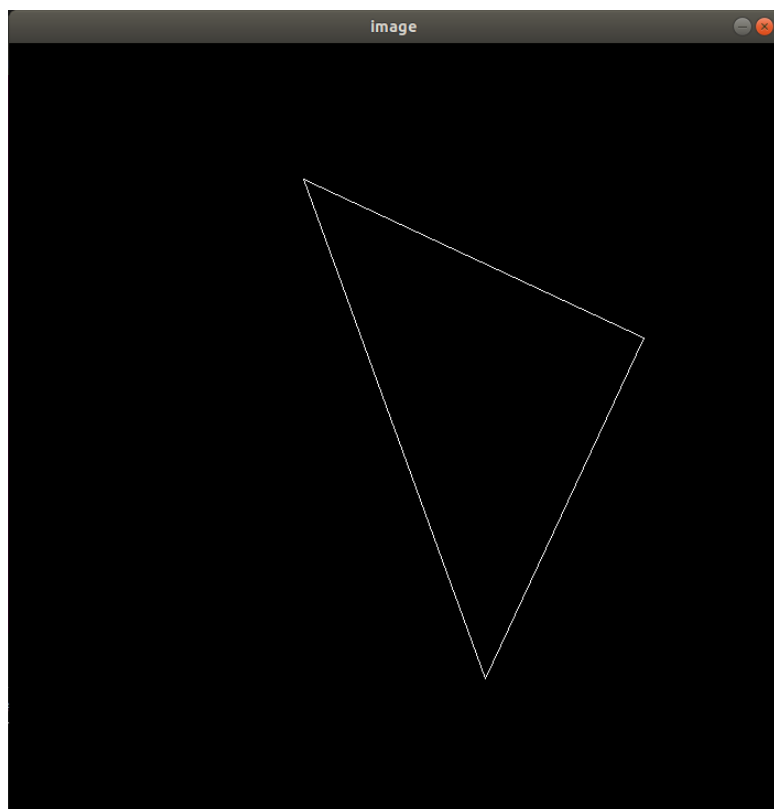


图 15: 旋转三角形 (3)

2.4 对四维齐次坐标的理解

齐次坐标就是将一个原本是 n 维的向量用一个 $n+1$ 维向量来表示。《计算机图形学 (OpenGL 版)》的作者 F.S. Hill Jr. 曾说过：“齐次坐标表示是计算机图形学的重要手段之一，它既能够用来明确区分向量和点，同时也更易用于进行仿射（线性）几何变换。”

在空间直角坐标系 $OXYZ$ 中，对于一个向量 \mathbf{v} ，我们可以用一组三维坐标 (v_1, v_2, v_3) 来表示：

$$\mathbf{v} = v_1\mathbf{X} + v_2\mathbf{Y} + v_3\mathbf{Z}$$

而对于一个点 P ，我们同样可以使用一组三维坐标 (p_1, p_2, p_3) 来表示：

$$P - O = p_1\mathbf{X} + p_2\mathbf{Y} + p_3\mathbf{Z}$$

表面上，在同一空间直角坐标系中，一个点的表示和一个向量的表示是一致的，都是使用三维坐标描述。但实际上，一个点的表示是从原点到该点的向量表示，是原点的位移。相比于向量的表示，点的表示还多了一个原点位置的隐藏条件。我们知道，一个向量的确定只需要大小和方向，但一个点的确定需要确定位置，而原点位置就是提供了这个信息。

因此，仅仅基于三维坐标，我们无法对一个点进行太多的操作。在我们对一个点的三维坐标进行操作时，实际上是对这个点代表的向量进行操作，而一个向量只能改变大小、方向，故我们只能实现缩放、旋转等线性变换，而无法实现平移变换。¹ 总而言之，平移变换对点有意义，而对向量无意义。

为了实现通过矩阵对点进行平移变换，我们就需要使用四维齐次坐标 (x, y, z, w) 来表示三维空间的一个点。设 $w = 1$ ，我们就相当于把三维空间点的坐标平移搬去 $w = 1$ 的平面上。对于一个四维空间的点 (x, y, z, w) ，其映射的三维坐标就是 $(\frac{x}{w}, \frac{y}{w}, \frac{z}{w})$ 。换言之，点 (x, y, z) 在齐次空间中有无数多个点与之对应，形式是 (kx, ky, kz, k) 。这样，我们在使用变换矩阵对四维齐次坐标变换时，操作的其实是四维空间上的向量。当把四维空间上的向量重新映射回三维空间时，就可以产生平移的效果。

第四维度 W 不仅使得几何变换的实现更加简单，也提供了区分向量和点的区分方法。按照上述定义，当 $w = 0$ 时，该齐次坐标相当于一个无穷远的“点”。此时其意义就是用于描述方向，也就相当于三维空间里的向量。综上所述，

- (1) 对于一个三维空间的顶点 (x, y, z) ，其四维齐次坐标是 $(x, y, z, 1)$ 。当 $k \neq 0$ 时， (kx, ky, kz, k) 映射的是同一个三维坐标点；
- (2) 对于一个三维空间向量 (x, y, z) ，其四维齐次坐标是 $(x, y, z, 0)$ ；
- (3) 对于一个四维齐次坐标 (x, y, z, w) ，如果 $w \neq 0$ ，则代表一个点 $(\frac{x}{w}, \frac{y}{w}, \frac{z}{w})$ ；如果 $w = 0$ ，则代表一个向量 (x, y, z) ；

3 作业总结与感想

本次作业内容不多，要求我们实现透视投影矩阵、绕 z 轴旋转的旋转变换矩阵，以及谈谈对四维齐次坐标的理解。在构造变换矩阵时，对实现思路和公式的梳理、推导比较困难，但推导完成后，使用代

¹在完成作业时，我对这点产生了疑问。似乎可以通过先旋转再缩放实现平移效果？不过最后写出来的表达式非常复杂，计算效率应该比不上使用齐次坐标。而且该变换矩阵里的数值依赖于旋转点的坐标信息，不能对原点操作，似乎根据变换矩阵的定义并不可行。

码实现起来就很轻松了。在阅读了助教提供的文章和网上的几篇博客后，我顺利完成了实验内容，取得了比较满意的结果，也对四维齐次坐标在图形学中的应用有了更深刻的理解。希望我能顺利完成下一次作业。