CSC4140 Assignment II

Computer Graphics

February 25, 2022

Transformation

This assignment is 10% of the total mark.

Strict Due Date: 11:59PM, Feb 25th, 2022

Student ID: 118010141

Student Name: Jingyu Li

This assignment represents my own work in accordance with University regulations.

Signature: JINGYU LI

1 Design

1.1 Implement Model Matrix

In this part, our task is to implement the function get_model_matrix(). Model matrix is formed by performing a translation, and then scaling and then rotation. The formulae show as the graphs.

➢ Translation

$$\mathbf{T}(t_x, t_y, t_z) = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Figure 1: Translation

➢ Scale

$$\mathbf{S}(s_x, s_y, s_z) = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Figure 2: Scale

➢ A general expression for a matrix that performs a rotation around a given axis **u** by a given angle θ:

$$\begin{bmatrix} \cos\theta + u_x^2(1-\cos\theta) & u_x u_y(1-\cos\theta) - u_z\sin\theta & u_x u_z(1-\cos\theta) + u_y\sin\theta \\ u_y u_x(1-\cos\theta) + u_z\sin\theta & \cos\theta + u_y^2(1-\cos\theta) & u_y u_z(1-\cos\theta) - u_x\sin\theta \\ u_z u_x(1-\cos\theta) - u_y\sin\theta & u_z u_y(1-\cos\theta) + u_x\sin\theta & \cos\theta + u_z^2(1-\cos\theta) \end{bmatrix}$$

Figure 3: Rotation

Replacing the parameters and multiplying the three matrices generate the result.

1.2 Implement perspective projection Matrix

In this part, our task is to implement get_projection_matrix() function. Perspective projection matrix is a little difficult to obtain. First define the variables representing left, right (r), bottom, top (t), near (zNear), far (zFar). Use eye's field of view (eye_fov) to obtain t, then use aspect ratio (ratio between width and height) to obtain r.

Next step is to transform the frustum to cubic. Using knowledge from similar triangle, we can easily obtain the relationship between new x/y values and original x/y values (e.g. $x' = \frac{x * zNear}{z}$). Now the goal is to find the relationship between new z and old z. We know transformation of z does not depend on x or y, so in the third row of 4 by 4 matrix, the first two values should be 0, set the last two as A and B, that is, the third row:

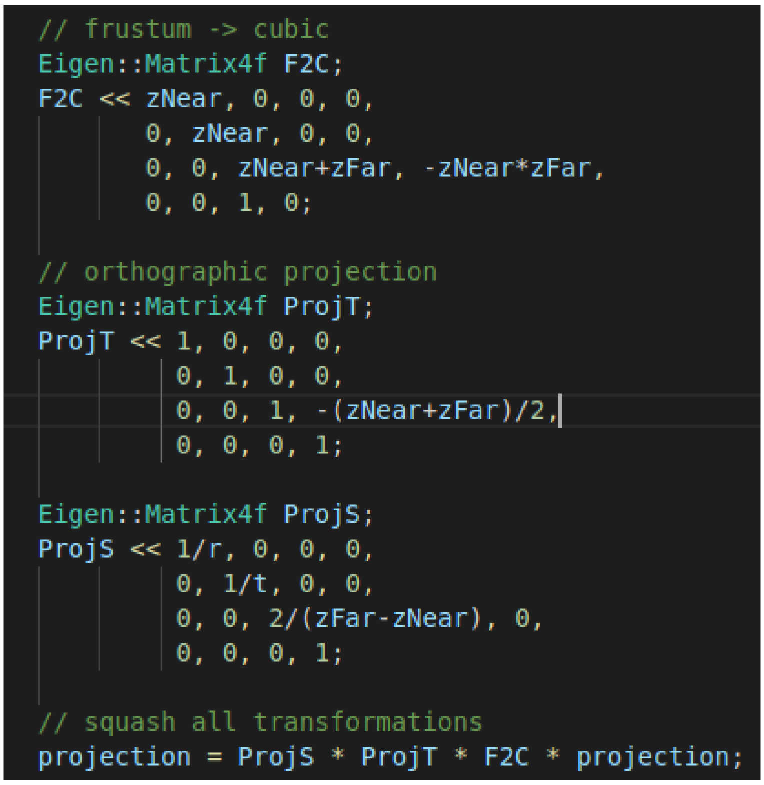
$$[0 \quad 0 \quad A \quad B]$$

We know z values of points in the near (zNear) and far planes (zFar) do not change

$$Az + B = z^{'}z$$

Replacing z with zNear and zFar we gain an equation which can give out the values of A and B, so that we obtain a matrix transforming frustum to cuboid.

Then we perform orthogonal projection, that is, to fit our cuboid to the normalized cube $[-1, 1]^3$, this operation forms by translating current cube's center to origin and then scale the length to 1. Multiplying these two matrices and the previous transformation matrix we obtain the projection matrix.



```
// frustum -> cubic
Eigen::Matrix4f F2C;
F2C << zNear, 0, 0, 0,
       0, zNear, 0, 0,
       0, 0, zNear+zFar, -zNear*zFar,
       0, 0, 1, 0;

// orthographic projection
Eigen::Matrix4f ProjT;
ProjT << 1, 0, 0, 0,
         0, 1, 0, 0,
         0, 0, 1, -(zNear+zFar)/2,
         0, 0, 0, 1;

Eigen::Matrix4f ProjS;
ProjS << 1/r, 0, 0, 0,
         0, 1/t, 0, 0,
         0, 0, 2/(zFar-zNear), 0,
         0, 0, 0, 1;

// squash all transformations
projection = ProjS * ProjT * F2C * projection;
```

Figure 4: Code

1.3 Implement main() function

In the main function, we mainly define parameters to be used. The specific values will be shown in the next section. I use two groups of parameters to test the rasterizer. Once parameters like eye position and rotation angle are identified, we can invoke the function implemented to generate relevant matrix and draw the triangle with the encapsulated class. Two modes are defined, generate the image directly and adjust the angle dynamically.

2 Parameter setting & Results

2.1 Sample 1

2.1.1 Parameter setting

Eye position: (0, 0, 10)

Triangle: ((5, 0, -5), (0, 5, -5), (-5, 0, -5)), which is an equilateral triangle

Angle: 30°

Eye_fov = 45°

Aspect_ratio = 1.0

ZNear = 0.1

ZFar = 10

2.1.2 Output Image

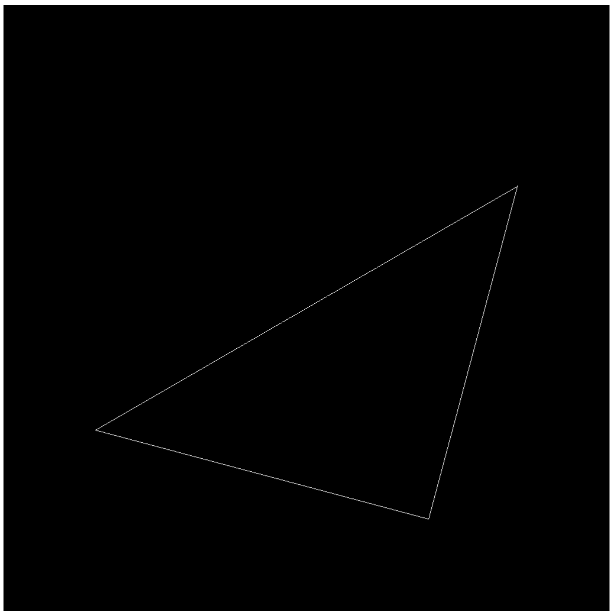
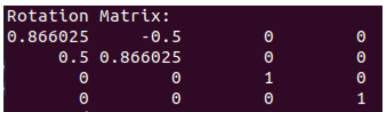


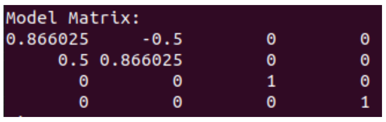
Figure 5: Sample 1, 30 degree

The corresponding rotation matrix, model matrix, view matrix and projection matrix are shown as follows:



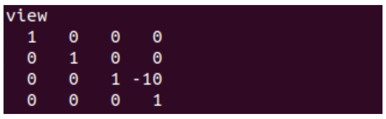
```
Rotation Matrix:
0.866025      -0.5         0         0
     0.5  0.866025         0         0
       0         0         1         0
       0         0         0         1
```

Figure 6: Rotation matrix



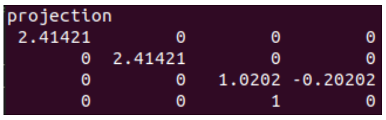
```
Model Matrix:
0.866025      -0.5         0         0
     0.5  0.866025         0         0
       0         0         1         0
       0         0         0         1
```

Figure 7: Model matrix



```
view
  1   0   0   0
  0   1   0   0
  0   0   1 -10
  0   0   0   1
```

Figure 8: View matrix



```
projection
 2.41421         0         0         0
       0   2.41421         0         0
       0         0    1.0202  -0.20202
       0         0         1         0
```

Figure 9: Projection matrix

By running with the second mode where angles can be adjusted dynamically, set angles by 60° (Click **a** for 6 times), we obtain:

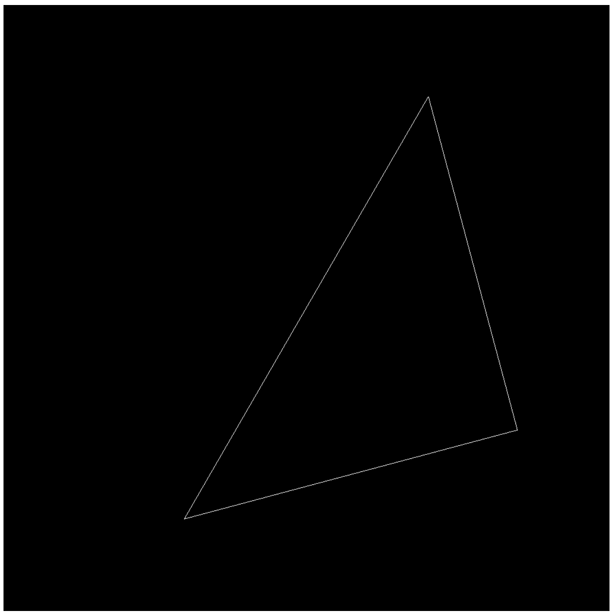


Figure 10: Sample 1, 60 degree

2.2 Sample 2

2.2.1 Parameter setting

Eye position: (3, 1.5, 5)

Triangle: ((0, 0, 0), (0, 3, 0), (4, 3, 0)), which is a right triangle

Angle: 60°

Eye_fov = 60°

Aspect_ratio = 2.5

ZNear = 0.5

ZFar = 20

2.2.2 Output Image

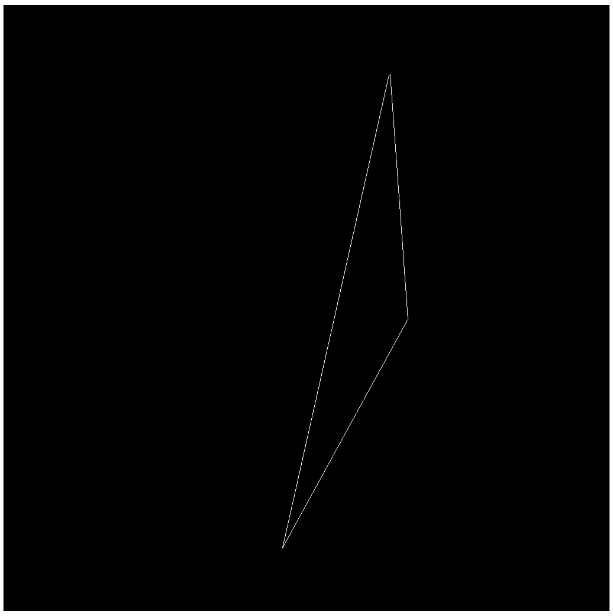
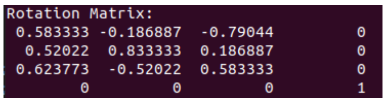


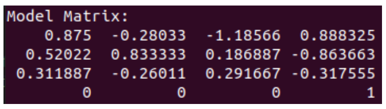
Figure 11: Sample 2, 60 degree

The corresponding rotation matrix, model matrix, view matrix and projection matrix are shown as follows:



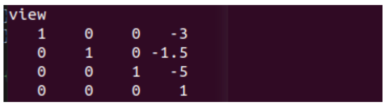
```
Rotation Matrix:
 0.583333 -0.186887  -0.79044         0
  0.52022  0.833333  0.186887         0
 0.623773  -0.52022  0.583333         0
        0         0         0         1
```

Figure 12: Rotation matrix



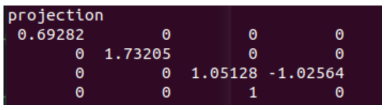
```
Model Matrix:
    0.875  -0.28033  -1.18566  0.888325
  0.52022  0.833333  0.186887 -0.863663
 0.311887  -0.26011  0.291667 -0.317555
        0         0         0         1
```

Figure 13: Model matrix



```
view
   1    0    0   -3
   0    1    0 -1.5
   0    0    1   -5
   0    0    0    1
```

Figure 14: View matrix



```
projection
 0.69282         0         0         0
       0   1.73205         0         0
       0         0   1.05128  -1.02564
       0         0         1         0
```

Figure 15: Projection matrix