重慶理工大學 两江国际学院
CQUT-KAIST
Liangjiang International College

Computer Graphics Assignment Report

Assignment:	**Assignment #3 Rotating Cube**
Name:	**游忠敏**
ID#:	**11619370216**
Date of Experiment:	**2019.4.27**
Date of Report:	**2019.4.29**
Submission:	**2019.4.30**

Ⅰ OBJECTIVES

Using OpenGL to generate a cube which keeps rotating with its central axis. Every face of this cube should have different colors.

Note: You should know how to remove the hidden surface of this cube when rendering.

Ⅱ THEORETICAL BACKGROUND

Matrices

A matrix is a rectangular array of mathematical expressions, much like a two-dimensional array. Below is an example of a matrix displayed in the common square brackets form.

$$a = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

Matrices values are indexed by (i,j) where i is the row and j is the column. That is why the matrix displayed above is called a 3-by-2 matrix. To refer to a specific value in the matrix, for example 5, the a31a31 notation is used.

Basic operations

To get a bit more familiar with the concept of an array of numbers, let's first look at a few basic operations.

Addition and subtraction

Just like regular numbers, the addition and subtraction operators are also defined for matrices. The only requirement is that the two operands have exactly the same row and column dimensions.

$$\begin{bmatrix} 3 & 2 \\ 0 & 4 \end{bmatrix} + \begin{bmatrix} 4 & 2 \\ 2 & 2 \end{bmatrix} = \begin{bmatrix} 3+4 & 2+2 \\ 0+2 & 4+2 \end{bmatrix} = \begin{bmatrix} 7 & 4 \\ 2 & 6 \end{bmatrix}$$

$$\begin{bmatrix} 4 & 2 \\ 2 & 7 \end{bmatrix} - \begin{bmatrix} 3 & 2 \\ 0 & 4 \end{bmatrix} = \begin{bmatrix} 4-3 & 2-2 \\ 2-0 & 7-4 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 2 & 3 \end{bmatrix}$$

The values in the matrices are individually added or subtracted from each other.

Scalar product

The product of a scalar and a matrix is as straightforward as addition and subtraction.

$$2 \cdot \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 2 & 4 \\ 6 & 8 \end{bmatrix}$$

The values in the matrices are each multiplied by the scalar.

Matrix-Vector product

The product of a matrix with another matrix is quite a bit more involved and is often misunderstood, so for simplicity's sake I will only mention the specific cases that apply to graphics programming. To see how matrices are actually used to transform vectors, we'll first dive into the product of a matrix and a vector.

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} a \cdot x + b \cdot y + c \cdot z + d \cdot 1 \\ e \cdot x + f \cdot y + g \cdot z + h \cdot 1 \\ i \cdot x + j \cdot y + k \cdot z + l \cdot 1 \\ m \cdot x + n \cdot y + o \cdot z + p \cdot 1 \end{pmatrix}$$

To calculate the product of a matrix and a vector, the vector is written as a 4-by-1 matrix. The expressions to the right of the equals sign show how the new x, y and z values are calculated after the vector has been transformed. For those among you who aren't very math savvy, the dot is a multiplication sign.

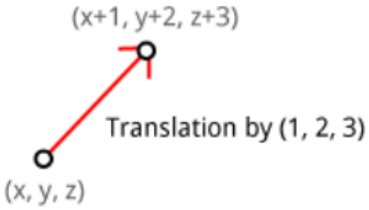
I will mention each of the common vector transformations in this section and how a matrix can be formed that performs them. For completeness, let's first consider a transformation that does absolutely nothing.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \cdot x + 0 \cdot y + 0 \cdot z + 0 \cdot 1 \\ 0 \cdot x + 1 \cdot y + 0 \cdot z + 0 \cdot 1 \\ 0 \cdot x + 0 \cdot y + 1 \cdot z + 0 \cdot 1 \\ 0 \cdot x + 0 \cdot y + 0 \cdot z + 1 \cdot 1 \end{pmatrix} = \begin{pmatrix} 1 \cdot x \\ 1 \cdot y \\ 1 \cdot z \\ 1 \cdot 1 \end{pmatrix}$$

This matrix is called the identity matrix, because just like the number 1, it will always return the value it was originally multiplied by.

Translation

To see why we're working with 4-by-1 vectors and subsequently 4-by-4 transformation matrices, let's see how a translation matrix is formed. A translation moves a vector a certain distance in a certain direction.



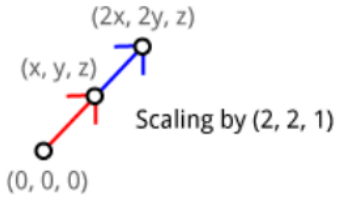
Can you guess from the multiplication overview what the matrix should look like to translate a vector by (X,Y,Z)?

$$\begin{bmatrix} 1 & 0 & 0 & X \\ 0 & 1 & 0 & Y \\ 0 & 0 & 1 & Z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x + X \cdot 1 \\ y + Y \cdot 1 \\ z + Z \cdot 1 \\ 1 \end{pmatrix}$$

Without the fourth column and the bottom 1 value a translation wouldn't have been possible.

Scaling

A scale transformation scales each of a vector's components by a (different) scalar. It is commonly used to shrink or stretch a vector as demonstrated below.



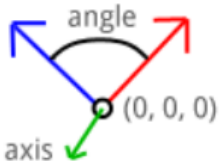
If you understand how the previous matrix was formed, it should not be difficult to come up with a matrix that scales a given vector by (SX,SY,SZ).

$$\begin{bmatrix} SX & 0 & 0 & 0 \\ 0 & SY & 0 & 0 \\ 0 & 0 & SZ & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} SX \cdot x \\ SY \cdot y \\ SZ \cdot z \\ 1 \end{pmatrix}$$

If you think about it for a moment, you can see that scaling would also be possible with a mere 3-by-3 matrix.

Rotation

A rotation transformation rotates a vector around the origin (0,0,0) using a given axis and angle. To understand how the axis and the angle control a rotation, let's do a small experiment.



Put your thumb up against your monitor and try rotating your hand around it. The object, your hand, is rotating around your thumb: the rotation axis. The further you rotate your hand away from its initial position, the higher the rotation angle.

In this way the rotation axis can be imagined as an arrow an object is rotating around. If you imagine your monitor to be a 2-dimensional XY surface, the rotation axis (your thumb) is pointing in the Z direction.

Objects can be rotated around any given axis, but for now only the X, Y and Z axis are important. You'll see later in this chapter that any rotation axis can be established by rotating around the X, Y and Z axis simultaneously.

The matrices for rotating around the three axes are specified here. The rotation angle is indicated by the theta (θ).

Rotation around X-axis:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ \cos\theta \cdot y - \sin\theta \cdot z \\ \sin\theta \cdot y + \cos\theta \cdot z \\ 1 \end{pmatrix}$$

Rotation around Y-axis:

$$\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta \cdot x + \sin\theta \cdot z \\ y \\ -\sin\theta \cdot x + \cos\theta \cdot z \\ 1 \end{pmatrix}$$

Rotation around Z-axis:

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta \cdot x - \sin\theta \cdot y \\ \sin\theta \cdot x + \cos\theta \cdot y \\ z \\ 1 \end{pmatrix}$$

But actually, in OpenGL, there is a function *glRotatef()* could help us implement rotation easily.

void glRotatef(GLfloat angle,

GLfloat x,

GLfloat y,

GLfloat z);

angle: Specifies the angle of rotation, in degrees.

x, y, z: Specify the x, y, and z coordinates of a vector, respectively.

Ⅲ PROCEDURES

First, we need to build the coordinate system. Because it revolves around the central axis, the three axes of the coordinate system are taken as the central axis, that is, the center of the circle is the center of the cube, then fill different surfaces with different colors.

```
// Drawing coordinate
glBegin(GL_LINES);
glPointSize(8);
glLineWidth(2);
glColor3f(1, 1, 1);
glVertex3f(0, -1, 0);
glVertex3f(0, 1, 0);
glVertex3f(-1, 0, 0);
glVertex3f(1, 0, 0);
glVertex3f(0, 0, -1);
glVertex3f(0, 0, 1);
glEnd();

// Drawing Cubes
glBegin(GL_LINES);
glPointSize(8);
glLineWidth(2);
glVertex3f(0.5, 0.5, -0.5);
glVertex3f(-0.5, 0.5, -0.5);
glVertex3f(-0.5, 0.5, 0.5);
glVertex3f(0.5, 0.5, 0.5);
glVertex3f(0.5, -0.5, 0.5);
glVertex3f(-0.5, -0.5, 0.5);
glVertex3f(-0.5, -0.5, -0.5);
glVertex3f(0.5, -0.5, -0.5);
glEnd();
```

Then in terms of rotation, take the X-axis as an example, because the cube has three central axes, namely, X-axis, Y-axis and Z-axis, and the codes in these three cases are different approaches but equally satisfactory results.

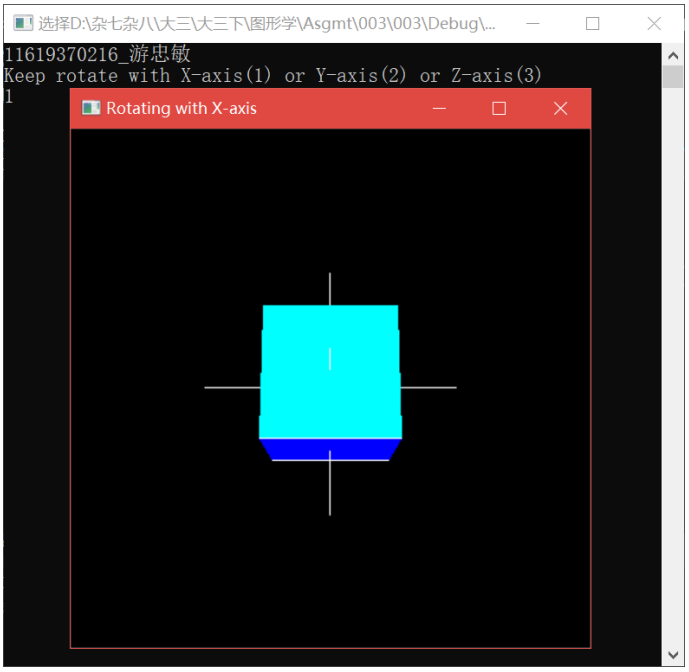
```
// Rotating with X axis
void x_rotate(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glTranslatef(0, 0, -5);
    glRotatef(angle, 1.0, 0, 0);
    cube_info();
    glutSwapBuffers();
    glPopMatrix();

    angle += 5.0f;
    Sleep(20);
}
```

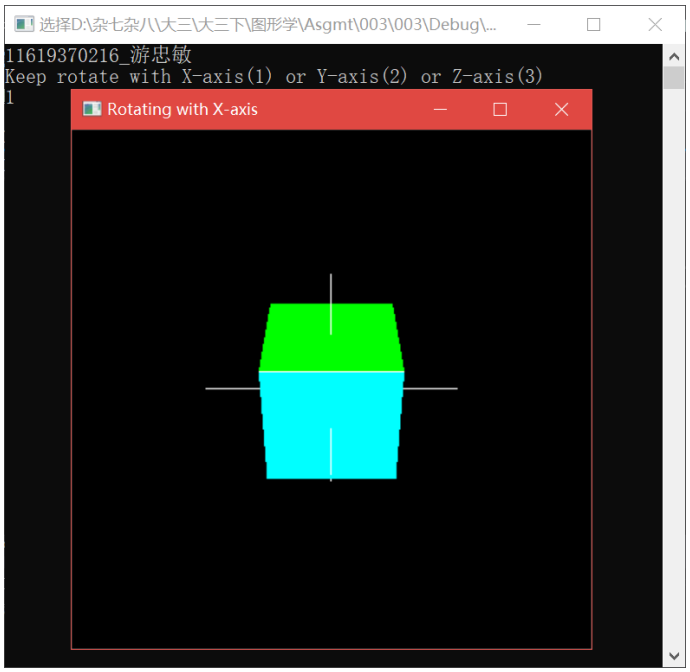
Finally, switch () is set to allow users to select the rotation axis they need.

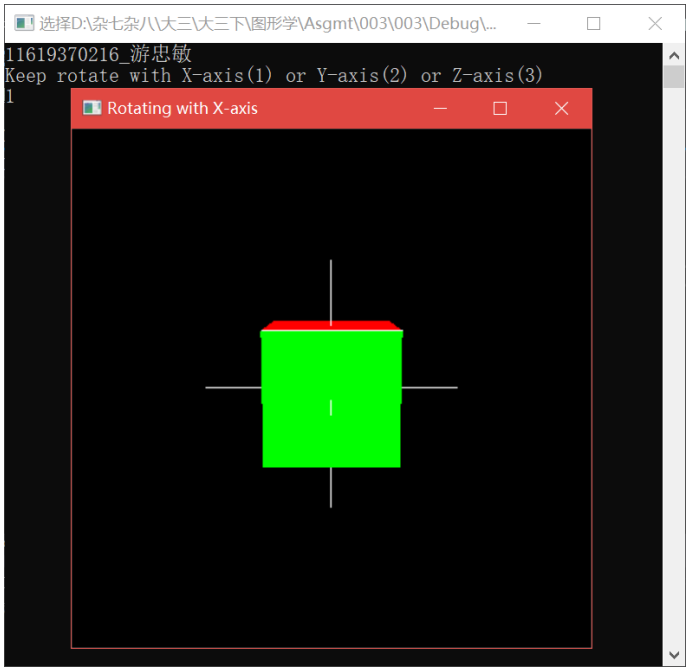
```
printf("Keep rotate with X-axis(1) or Y-axis(2) or Z-axis(3)\n");
scanf_s("%d", &i);
switch (i)
{
case 1:
    glutCreateWindow("Rotating with X-axis");
    glutDisplayFunc(x_rotate);
    glutIdleFunc(x_rotate);
    break;
case 2:
    glutCreateWindow("Rotating with Y-axis");
    glutDisplayFunc(y_rotate);
    glutIdleFunc(y_rotate);
    break;
case 3:
    glutCreateWindow("Rotating with Z-axis");
    glutDisplayFunc(z_rotate);
    glutIdleFunc(z_rotate);
    break;
default:
    printf("\n Error!!\n");
```

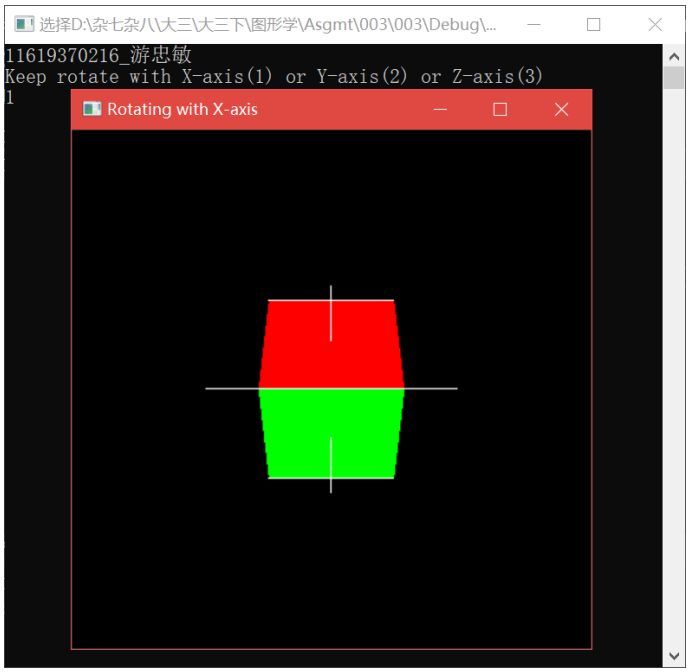
IV RESULTS

Rotating with X-axis:

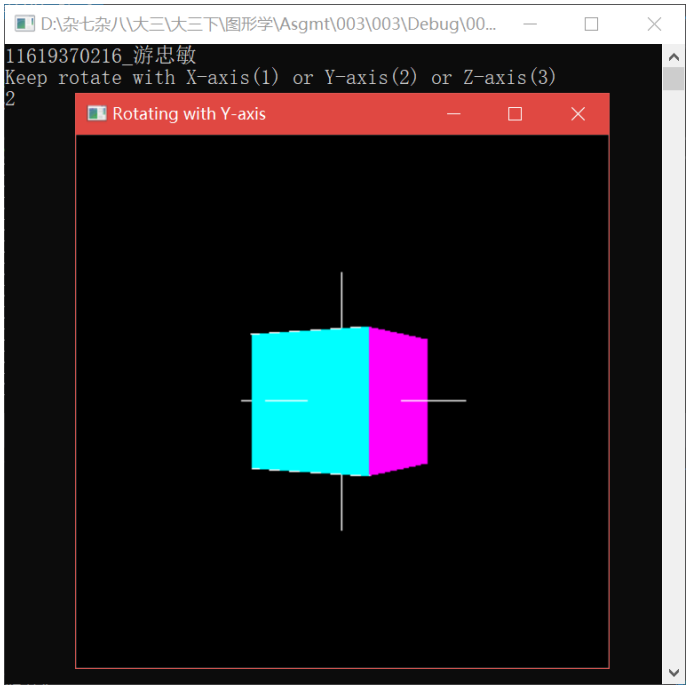


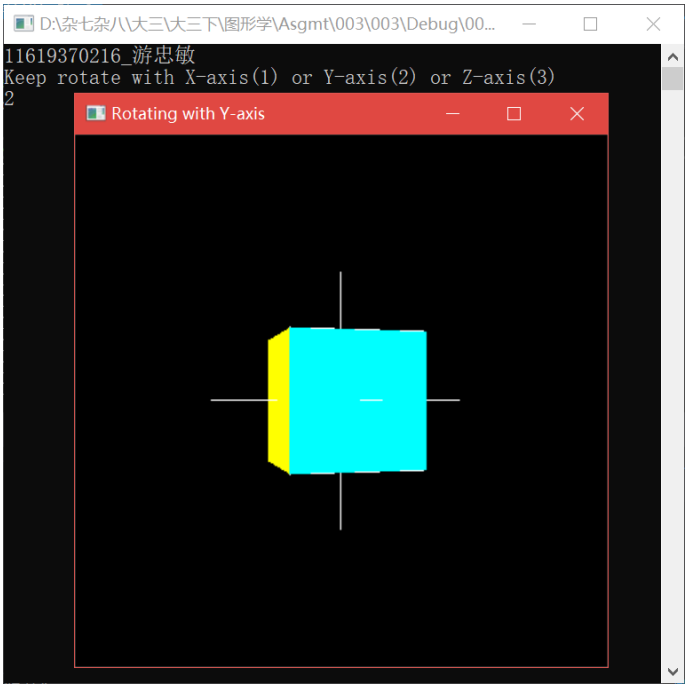

选择D:\杂七杂八\大三\大三下\图形学\Asgmt\003\003\Debug\...
11619370216_游忠敏
Keep rotate with X-axis(1) or Y-axis(2) or Z-axis(3)
1
Rotating with X-axis

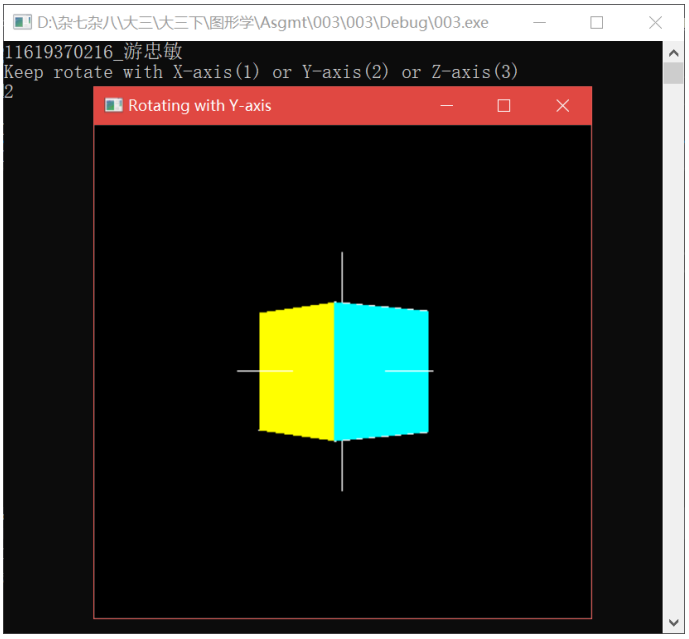

选择D:\杂七杂八\大三\大三下\图形学\Asgmt\003\003\Debug\...
11619370216_游忠敏
Keep rotate with X-axis(1) or Y-axis(2) or Z-axis(3)
1
Rotating with X-axis

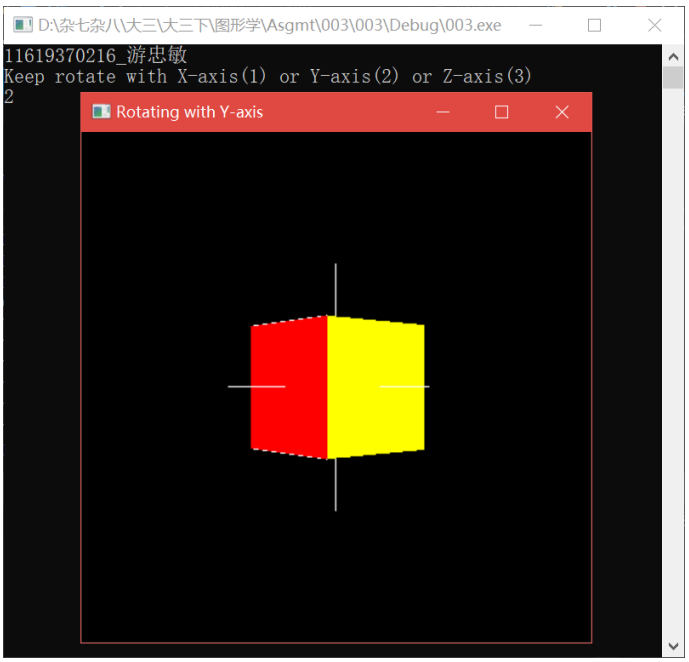


Rotating with Y-axis:

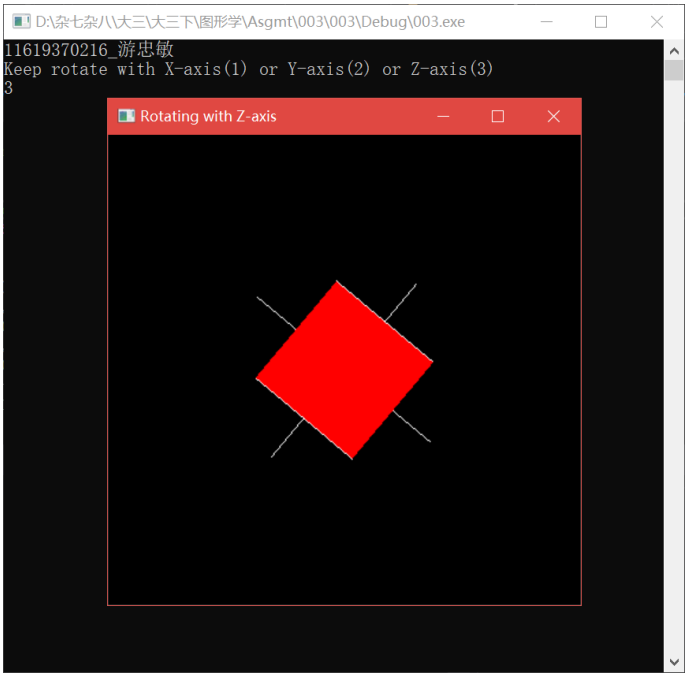


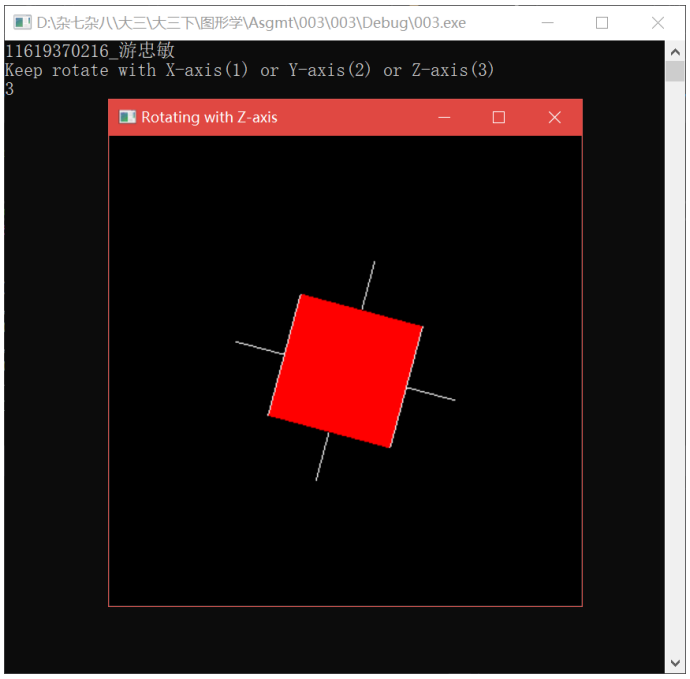

D:\杂七杂八\大三\大三下\图形学\Asgmt\003\003\Debug\00...
11619370216_游忠敏
Keep rotate with X-axis(1) or Y-axis(2) or Z-axis(3)
2
Rotating with Y-axis


D:\杂七杂八\大三\大三下\图形学\Asgmt\003\003\Debug\003.exe
11619370216_游忠敏
Keep rotate with X-axis(1) or Y-axis(2) or Z-axis(3)
2
Rotating with Y-axis



Rotating with Z-axis:





V DISCUSSION

From what we have learned in our textbooks, the rotation of an object requires a rotation matrix, but in practice, only a known function is needed to achieve it. We can always feel shallow on paper and never know that we have to go through it.

VI CONCLUSIONS

This assignment make me fully understand the algorithm of image translation and rotation, then deepen my understanding of graphics.

APPENDIX: C/JAVA/Python code

```
#include <GL/glut.h>
#include <stdio.h>
#include <Windows.h>
#include <stdlib.h>
#include <math.h>

// Basic cube information
void cube_info()
{
    // Drawing coordinate
    glBegin(GL_LINES);
    glPointSize(8);
```

```
glLineWidth(2);
glColor3f(1,1,1);
glVertex3f(0,-1,0);
glVertex3f(0,1,0);
glVertex3f(-1,0,0);
glVertex3f(1,0,0);
glVertex3f(0,0,-1);
glVertex3f(0,0,1);
glEnd();

// Drawing Cubes
glBegin(GL_LINES);
glPointSize(8);
glLineWidth(2);
glVertex3f(0.5, 0.5, -0.5);
glVertex3f(-0.5, 0.5, -0.5);
glVertex3f(-0.5, 0.5, 0.5);
glVertex3f(0.5, 0.5, 0.5);
glVertex3f(0.5, -0.5, 0.5);
glVertex3f(-0.5, -0.5, 0.5);
glVertex3f(-0.5, -0.5, -0.5);
glVertex3f(0.5, -0.5, -0.5);
glEnd();

// Fill in color
glBegin(GL_QUADS);
glColor3f(0.0,0.0,1.0);
glVertex3f( 0.5, 0.5,-0.5);
glVertex3f(-0.5, 0.5,-0.5);
glVertex3f(-0.5, 0.5, 0.5);
glVertex3f( 0.5, 0.5, 0.5);

glColor3f(0.0,1.0,0.0);
glVertex3f( 0.5,-0.5, 0.5);
glVertex3f(-0.5,-0.5, 0.5);
glVertex3f(-0.5,-0.5,-0.5);
glVertex3f( 0.5,-0.5,-0.5);

glColor3f(1.0,0.0,0.0);
glVertex3f( 0.5, 0.5, 0.5);
glVertex3f(-0.5, 0.5, 0.5);
glVertex3f(-0.5,-0.5, 0.5);
glVertex3f( 0.5,-0.5, 0.5);
```

```
    glColor3f(0.0,1.0,1.0);
    glVertex3f( 0.5,-0.5,-0.5);
    glVertex3f(-0.5,-0.5,-0.5);
    glVertex3f(-0.5, 0.5,-0.5);
    glVertex3f( 0.5, 0.5,-0.5);

    glColor3f(1.0,0.0,1.0);
    glVertex3f(-0.5, 0.5, 0.5);
    glVertex3f(-0.5, 0.5,-0.5);
    glVertex3f(-0.5,-0.5,-0.5);
    glVertex3f(-0.5,-0.5, 0.5);

    glColor3f(1.0,1.0,0.0);
    glVertex3f( 0.5, 0.5,-0.5);
    glVertex3f( 0.5, 0.5, 0.5);
    glVertex3f( 0.5,-0.5, 0.5);
    glVertex3f( 0.5,-0.5,-0.5);
    glEnd();
}

// Setting global variables for later calls to achieve angular rotation
GLfloat angle = 0.0f;

// Rotating with X axis
void x_rotate(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glTranslatef(0, 0, -5);
    glRotatef(angle, 1.0, 0, 0);
    cube_info();
    glutSwapBuffers();
    glPopMatrix();

    angle += 5.0f;
    Sleep(20);
}

// Rotating with Y axis
void y_rotate(void)
{
     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
     glLoadIdentity();
     glTranslatef(0, 0, -5);
```

```
    glRotatef(angle, 0, 1.0, 0);
    cube_info();
    glutSwapBuffers();
    glPopMatrix();

    angle += 5.0f;
    Sleep(20);
}

// Rotating with Z axis
void z_rotate(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glTranslatef(0, 0, -5);
    glRotatef(angle, 0, 0, 1.0);
    cube_info();
    glutSwapBuffers();
    glPopMatrix();

    angle += 5.0f;
    Sleep(20);
}

// Re-drew the window
void re_drew(int w, int h)
{
    if(h==0) h = 1;
    glViewport(0, 0, (GLsizei) w, (GLsizei) h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45.0, (GLfloat)w/(GLfloat)h, 0.1, 100.0);
    glMatrixMode(GL_MODELVIEW);
}

void init(int width, int height )
{
    if(height == 0) height = 1;
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glClearDepth(1.0);
    glDepthFunc(GL_LESS);
    glEnable(GL_DEPTH_TEST);
    glShadeModel(GL_SMOOTH);
```

```
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45.0, (GLfloat)width/(GLfloat)height, 1, 100.0);
    glMatrixMode(GL_MODELVIEW);
}


int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH);
    glutInitWindowPosition(300, 150);
    glutInitWindowSize(400, 400);
    int i;

    printf("11619370216_游忠敏\n");
    printf("Keep rotate with X-axis(1) or Y-axis(2) or Z-axis(3)\n");
    scanf_s("%d", &i);
    switch (i)
    {
    case 1:
        glutCreateWindow("Rotating with X-axis");
        glutDisplayFunc(x_rotate);
        glutIdleFunc(x_rotate);
        break;
    case 2:
        glutCreateWindow("Rotating with Y-axis");
        glutDisplayFunc(y_rotate);
        glutIdleFunc(y_rotate);
        break;
    case 3:
        glutCreateWindow("Rotating with Z-axis");
        glutDisplayFunc(z_rotate);
        glutIdleFunc(z_rotate);
        break;
    default:
        printf("\n Error!!\n");
    }


    glutReshapeFunc(re_drew);
    init(640, 480);
    glutMainLoop();
    return 0;
```

```
}
```