实验十二 三维图形几何观察实验

时间：2022年5月25日

地点：信息学院机房2202

1、实验内容

实现三维图形观察

2、实验目的

调用函数完成三维图形观察变换

3、实验代码

#include <stdlib.h>

#include<math.h>

#include <GL/glut.h>

#include<stdio.h>

class wcPt3D {

public:

GLfloat x, y, z;

};

typedef GLfloat Matrix4x4 [4][4];

typedef GLfloat Point[5][3];

float theta=0.0;//旋转角度

float s=1.0;//缩放系数

Matrix4x4 matComposite;

void matrix4x4SetIdentity (Matrix4x4 matIdent4x4) /\* Construct the 4 x 4 identity matrix. \*/

{

GLint row, col;

for (row = 0; row < 4; row++)

for (col = 0; col < 4 ; col++)

matIdent4x4 [row][col] = (row == col);

}

/\* Premultiply matrix m1 by matrix m2, store result in m2. \*/

void matrix4x4PreMultiply (Matrix4x4 m1, Matrix4x4 m2)

{

GLint row, col;

Matrix4x4 matTemp;

for (row = 0; row < 4; row++)

for (col = 0; col < 4 ; col++)

matTemp [row][col] = m1 [row][0] \* m2 [0][col] + m1 [row][1] \*

m2 [1][col] + m1 [row][2] \* m2 [2][col] +

m1 [row][3] \* m2 [3][col];

for (row = 0; row < 4; row++)

for (col = 0; col < 4; col++)

m2 [row][col] = matTemp [row][col];

}

/\* Procedure for generating 3D translation matrix. \*/

void translate3D (GLfloat tx, GLfloat ty, GLfloat tz)

{

Matrix4x4 matTransl3D;

matrix4x4SetIdentity (matTransl3D); /\* Initialize translation matrix to identity. \*/

matTransl3D [0][3] = tx;

matTransl3D [1][3] = ty;

matTransl3D [2][3] = tz;

/\* Concatenate matTransl3D with composite matrix. \*/

matrix4x4PreMultiply (matTransl3D, matComposite);

}

/\* Procedure for generating a quaternion rotation matrix. \*/

void rotate3D (wcPt3D p1, wcPt3D p2, GLfloat radianAngle)

{

Matrix4x4 matQuatRot;

float axisVectLength = sqrt ((p2.x - p1.x) \* (p2.x - p1.x) +

(p2.y - p1.y) \* (p2.y - p1.y) +

(p2.z - p1.z) \* (p2.z - p1.z));

float cosA = cosf (radianAngle);

float oneC = 1 - cosA;

float sinA = sinf (radianAngle);

float ux = (p2.x - p1.x) / axisVectLength;

float uy = (p2.y - p1.y) / axisVectLength;

float uz = (p2.z - p1.z) / axisVectLength;

translate3D (-p1.x, -p1.y, -p1.z);

/\* Initialize matQuatRot to identity matrix. \*/

matrix4x4SetIdentity (matQuatRot);

matQuatRot [0][0] = ux\*ux\*oneC + cosA;

matQuatRot [0][1] = ux\*uy\*oneC - uz\*sinA;

matQuatRot [0][2] = ux\*uz\*oneC + uy\*sinA;

matQuatRot [1][0] = uy\*ux\*oneC + uz\*sinA;

matQuatRot [1][1] = uy\*uy\*oneC + cosA;

matQuatRot [1][2] = uy\*uz\*oneC - ux\*sinA;

matQuatRot [2][0] = uz\*ux\*oneC - uy\*sinA;

matQuatRot [2][1] = uz\*uy\*oneC + ux\*sinA;

matQuatRot [2][2] = uz\*uz\*oneC + cosA;

/\* Concatenate matQuatRot with composite matrix. \*/

matrix4x4PreMultiply (matQuatRot, matComposite);

translate3D (p1.x, p1.y, p1.z);

}

/\* Procedure for generating a 3D scaling matrix. \*/

void scale3D (GLfloat sx, GLfloat sy, GLfloat sz, wcPt3D fixedPt)

{

Matrix4x4 matScale3D;

/\* Initialize scaling matrix to identity. \*/

matrix4x4SetIdentity (matScale3D);

matScale3D [0][0] = sx;

matScale3D [0][3] = (1 - sx) \* fixedPt.x;

matScale3D [1][1] = sy;

matScale3D [1][3] = (1 - sy) \* fixedPt.y;

matScale3D [2][2] = sz;

matScale3D [2][3] = (1 - sz) \* fixedPt.z;

/\* Concatenate matScale3D with composite matrix. \*/

matrix4x4PreMultiply (matScale3D, matComposite);

}

void drawPyramid(Point a) //该金字塔在以原点为中心，边长为2的立方体范围内

{

glBegin(GL\_TRIANGLES);

glColor3f(1.0f,0.0f,0.0f); //前面为红色

glVertex3f( a[0][0],a[0][1],a[0][2]); //前面三角形上顶点

glVertex3f(a[1][0],a[1][1],a[1][2]); //前面三角形左顶点

glVertex3f( a[2][0],a[2][1],a[2][2]); //前面三角形右顶点

glColor3f(0.0f,1.0f,0.0f); //右面为绿色

glVertex3f( a[0][0], a[0][1], a[0][2]); //右面三角形上顶点

glVertex3f( a[2][0],a[2][1],a[2][2]); //右面三角形左顶点

glVertex3f( a[3][0],a[3][1],a[3][2]); //右面三角形右顶点

glColor3f(0.0f,0.0f,1.0f); //背面为蓝色

glVertex3f( a[0][0], a[0][1], a[0][2]); //背面三角形上顶点

glVertex3f( a[3][0],a[3][1],a[3][2]); //背面三角形左顶点

glVertex3f(a[4][0],a[4][1],a[4][2]); //背面三角形右顶点

glColor3f(1.0f,1.0f,0.0f); //左面为黄色

glVertex3f( a[0][0], a[0][1], a[0][2]); //左面三角形上顶点

glVertex3f(a[4][0],a[4][1],a[4][2]); //左面三角形左顶点

glVertex3f(a[1][0],a[1][1],a[1][2]); //左面三角形右顶点

glEnd();

glBegin(GL\_POLYGON); //金字塔底面正方形

glColor3f(0.5f,0.5f,0.5f); //底面为灰色

glVertex3f(a[1][0],a[1][1],a[1][2]);

glVertex3f(a[2][0],a[2][1],a[2][2]);

glVertex3f(a[3][0],a[3][1],a[3][2]);

glVertex3f(a[4][0],a[4][1],a[4][2]);

glEnd();

}

void apply(Point a,Matrix4x4 Tran){

for(int i=0;i<5;i++){

int tx=a[i][0],ty=a[i][1],tz=a[i][2],tag=1;

a[i][0]=tx\*Tran[0][0]+ty\*Tran[1][0]+tz\*Tran[2][0]+tag\*Tran[3][0];

a[i][1]=tx\*Tran[0][1]+ty\*Tran[1][1]+tz\*Tran[2][1]+tag\*Tran[3][1];

a[i][2]=tx\*Tran[0][2]+ty\*Tran[1][2]+tz\*Tran[2][2]+tag\*Tran[3][2];

}

}

void displayFcn (void)

{

glClear(GL\_COLOR\_BUFFER\_BIT|GL\_DEPTH\_BUFFER\_BIT); //清空颜色和深度缓存

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

wcPt3D fixedPt;

fixedPt.x=0,fixedPt.y=0,fixedPt.z=0;//缩放中心

wcPt3D p1,p2;

p1.x=0,p1.y=0,p1.z=0;//旋转中心

p2.x=0,p2.y=1,p2.z=0;//旋转轴

matrix4x4SetIdentity (matComposite);

glPushMatrix();

rotate3D (p1, p2, theta); // First transformation: Rotate.

scale3D (s, s, s, fixedPt); // Second transformation: Scale.

//translate3D (0, 0, 0); // Final transformation: Translate.

Point a={{0,1,0},{-1,-1,1},{1,-1,1},{1,-1,-1},{-1,-1,-1}};

apply(a,matComposite);

//glTranslatef(0.0f,0.0f,-10.0f);

//glRotatef(theta,0.0f,1.0f,0.0f);

drawPyramid(a);

glPopMatrix();

glutSwapBuffers();

}

void reshape(int w, int h) //重绘回调函数，在窗口首次创建或用户改变窗口尺寸时被调用

{

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

//glFrustum(-1.0, 1.0, -1.0, 1.0, 3.1, 10.0);

//gluPerspective(45,1,0.1,10.0);

glOrtho(-2.0, 2.0, -2.0, 2.0, -2.0, 2);

}

void init()

{

glClearColor (1.0, 1.0, 1.0, 1.0);

glEnable(GL\_DEPTH\_TEST); //启动深度测试模式

}

void myKeyboard(unsigned char key, int x, int y)

{

if(key == 'a' || key == 'A')

theta += 5.0;

if(key == 's' || key == 'S')

theta -= 5.0;

if(key == 'c' || key == 'C')

exit(0);

if (theta>360) theta -=360;

if (theta<0) theta +=360;

glutPostRedisplay(); //重新调用绘制函数

}

void myMouse(int button, int state, int x, int y){//滑轮控制缩放

if(button== 3 ){

s += 0.05;

}

if(button== 4 && s>0.1){

s-=0.05;

}

glutPostRedisplay(); //重新调用绘制函数

}

int main(int argc, char\*\* argv)

{

glutInit(&argc,argv);

glutInitDisplayMode (GLUT\_DEPTH |GLUT\_DOUBLE | GLUT\_RGB);

glutInitWindowSize(500,500);

glutInitWindowPosition(0,0);

glutCreateWindow ("三维变换");

init();

glutReshapeFunc(reshape); //指定重绘回调函数

glutDisplayFunc(displayFcn);

glutKeyboardFunc( myKeyboard); //指定键盘回调函数

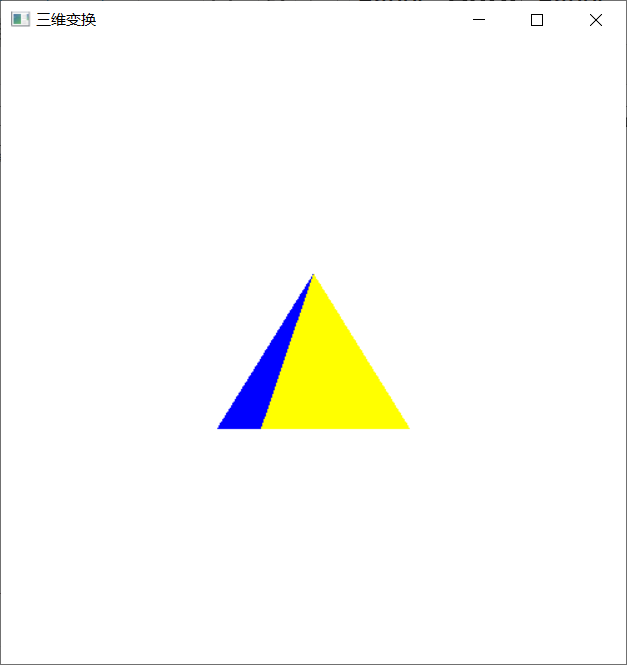
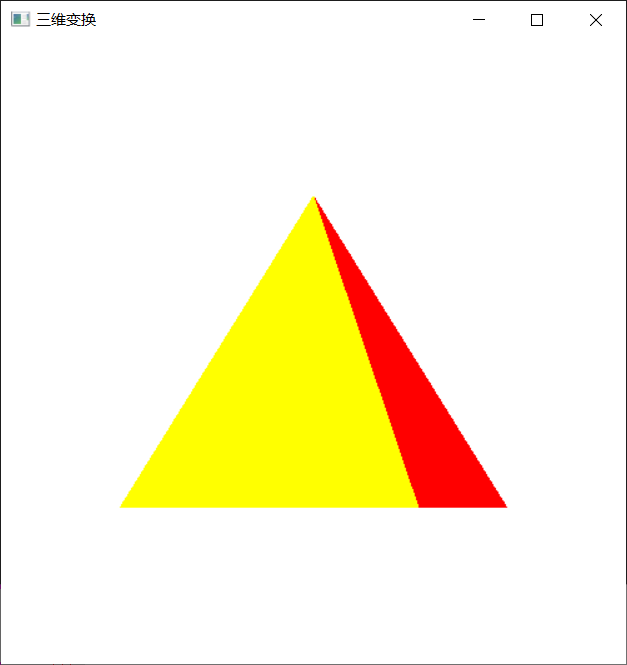
glutMouseFunc(myMouse);

glutMainLoop();

}

4、实验结果

交互式操作：按键A、S实现左右旋转，滚动鼠标滚轮实现比例变换。



5、实验总结

在进行三维变换时，需要用到一个4X4的矩阵，对立体图形上的每一个点，都进行右乘一个四维矩阵，矩阵相乘需要进行到每一行和每一列，需要大量的运算，但是，变换矩阵有好多部分为0，对于为0的部分，我们可以不做处理，这样我们就可以减少运算，大大提升程序运算速度。