**UCS1712 – GRAPHICS AND MULTIMEDIA LAB**

**Lab Exercise 8: 3-Dimensional Transformations in C++ using OpenGL**

**CODE:**

#include<gl/glut.h>

#include<iostream>

#include<utility>

#include<vector>

#include<math.h>

constexpr auto PI = 3.14;

using namespace std;

vector<vector<GLfloat>> coords(8,vector<GLfloat>(3));

int tx, ty, tz;

int ch;

double sx, sy, sz;

int xf, yf, zf;

int RotAxis;

int ang;

float rad;

void myInit()

{

glClearColor(1.0, 1.0, 1.0, 1.0);

glLoadIdentity();

glOrtho(-500.0, 500.0, -500.0, 500.0, -500.0, 500.0);

glEnable(GL\_DEPTH\_TEST);

}

void drawCube() {

glBegin(GL\_QUADS);

glColor3f(1, 0, 0);

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

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

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

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

glColor3f(0, 1, 0);

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

glVertex3f(coords[5][0], coords[5][1], coords[5][2]);

glVertex3f(coords[6][0], coords[6][1], coords[6][2]);

glVertex3f(coords[7][0], coords[7][1], coords[7][2]);

glColor3f(0, 0, 1);

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

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

glVertex3f(coords[5][0], coords[5][1], coords[5][2]);

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

glColor3f(1, 0, 1);

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

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

glVertex3f(coords[7][0], coords[7][1], coords[7][2]);

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

glColor3f(0, 1, 1);

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

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

glVertex3f(coords[6][0], coords[6][1], coords[6][2]);

glVertex3f(coords[5][0], coords[5][1], coords[5][2]);

glColor3f(1, 1, 0);

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

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

glVertex3f(coords[7][0], coords[7][1], coords[7][2]);

glVertex3f(coords[6][0], coords[6][1], coords[6][2]);

glEnd();

}

void Axis() {

glBegin(GL\_LINES);

glColor3f(0, 0, 1);

glVertex3f(0, 0, 0);

glVertex3f(0, 0, 500);

glColor3f(1, 0, 0);

glVertex3f(0, 0, 0);

glVertex3f(500, 0, 0);

glColor3f(0, 1, 0);

glVertex3f(0, 0, 0);

glVertex3f(0, 500, 0);

glEnd();

}

void translate() {

vector<vector<GLfloat>> T(4, vector<GLfloat>(4, 0));

T[0][0] = 1;

T[1][1] = 1;

T[2][2] = 1;

T[3][3] = 1;

T[0][3] = tx;

T[1][3] = ty;

T[2][3] = tz;

for (int c = 0; c < coords.size(); c++)

{

vector<GLfloat> P(4),N(4,0);

P[0] = coords[c][0];

P[1] = coords[c][1];

P[2] = coords[c][2];

P[3] = 1;

for (int i = 0; i < 4; i++) {

for (int j = 0; j < 1; j++) {

N[i] = 0;

for (int k = 0; k < 4; k++) {

N[i] += T[i][k] \* P[k];

}

}

}

coords[c][0] = N[0];

coords[c][1] = N[1];

coords[c][2] = N[2];

}

}

void scale() {

vector<vector<GLfloat>> T(4, vector<GLfloat>(4, 0));

T[0][0] = sx;

T[1][1] = sy;

T[2][2] = sz;

T[3][3] = 1;

T[0][3] = (1 - sx) \* xf;

T[1][3] = (1 - sy) \* yf;

T[2][3] = (1 - sz) \* zf;

for (int c = 0; c < coords.size(); c++)

{

vector<GLfloat> P(4), N(4, 0);

P[0] = coords[c][0];

P[1] = coords[c][1];

P[2] = coords[c][2];

P[3] = 1;

for (int i = 0; i < 4; i++) {

for (int j = 0; j < 1; j++) {

N[i] = 0;

for (int k = 0; k < 4; k++) {

N[i] += T[i][k] \* P[k];

}

}

}

coords[c][0] = N[0];

coords[c][1] = N[1];

coords[c][2] = N[2];

}

}

void rotate() {

vector<vector<GLfloat>> T(4, vector<GLfloat>(4, 0));

switch (RotAxis) {

case 1: T = {

{1, 0, 0, 0},

{0, cos(rad), -1\*sin(rad), 0},

{0, sin(rad), cos(rad), 0},

{0, 0, 0, 1}

};

break;

case 2: T = {

{cos(rad), 0, sin(rad), 0},

{0, 1, 0, 0},

{-1\*sin(rad), 0, cos(rad), 0},

{0, 0, 0, 1}

};

break;

case 3: T = {

{cos(rad), -1\*sin(rad), 0, 0},

{sin(rad), cos(rad), 0, 0},

{0, 0, 1, 0},

{0, 0, 0, 1}

};

break;

default: T = {

{1,0,0,0},

{0,1,0,0},

{0,0,1,0},

{0,0,0,1}

};

break;

}

for (int c = 0; c < coords.size(); c++)

{

vector<GLfloat> P(4), N(4, 0);

P[0] = coords[c][0];

P[1] = coords[c][1];

P[2] = coords[c][2];

P[3] = 1;

for (int i = 0; i < 4; i++) {

for (int j = 0; j < 1; j++) {

N[i] = 0;

for (int k = 0; k < 4; k++) {

N[i] += T[i][k] \* P[k];

}

}

}

coords[c][0] = N[0];

coords[c][1] = N[1];

coords[c][2] = N[2];

}

}

void myDisplay()

{

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glColor3f(0.0f, 0.0f, 0.0f);

glRotatef(-45, 0, 1, 0);

glRotatef(45, 1, 0, 0);

glRotatef(-30, 0, 0, 1);

glTranslatef(-100, 0, 0);

Axis();

drawCube();

switch (ch) {

case 1:translate();

drawCube();

break;

case 2: rotate();

drawCube();

break;

case 3:scale();

drawCube();

break;

}

glFlush();

}

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

{

GLfloat x1=100, Y1=100, z1=100,x2=200,y2=200,z2=200;

cout << "Enter cube dimensions:" << endl;

cout << "Enter min x,y,z: ";

cin >> x1>> Y1>> z1;

cout << "Enter max x,y,z: ";

cin >> x2>> y2>> z2;

cout << "Enter Transformation Operation" <<endl<< "1.Translate" << endl << "2.Rotate" << endl << "3.Scale" << endl;

cout << "Choice: ";

cin >> ch;

switch (ch) {

case 1: cout << "Enter tx,ty,tz: ";

cin >> tx >> ty >> tz;

break;

case 2: cout << "Enter Axis to rotate about X(1), Y(2), Z(3): ";

cin >> RotAxis;

cout << "Enter rotation ang: ";

cin >> ang;

rad = ang \* PI / 180;

break;

case 3: cout << "Enter sx,sy,sz: ";

cin >> sx >> sy >> sz;

cout << "Enter point to scale about x,y,z: ";

cin >> xf >> yf >> zf;

break;

default: cout << "invalid";

}

coords[0][0] = x1;

coords[0][1] = Y1;

coords[0][2] = z1;

coords[1][0] = x1;

coords[1][1] = y2;

coords[1][2] = z1;

coords[2][0] = x2;

coords[2][1] = y2;

coords[2][2] = z1;

coords[3][0] = x2;

coords[3][1] = Y1;

coords[3][2] = z1;

coords[4][0] = x1;

coords[4][1] = Y1;

coords[4][2] = z2;

coords[5][0] = x1;

coords[5][1] = y2;

coords[5][2] = z2;

coords[6][0] = x2;

coords[6][1] = y2;

coords[6][2] = z2;

coords[7][0] = x2;

coords[7][1] = Y1;

coords[7][2] = z2;

glutInit(&argc, argv);

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

glutInitWindowSize(1000, 1000);

glutCreateWindow("3d cube");

glutDisplayFunc(myDisplay);

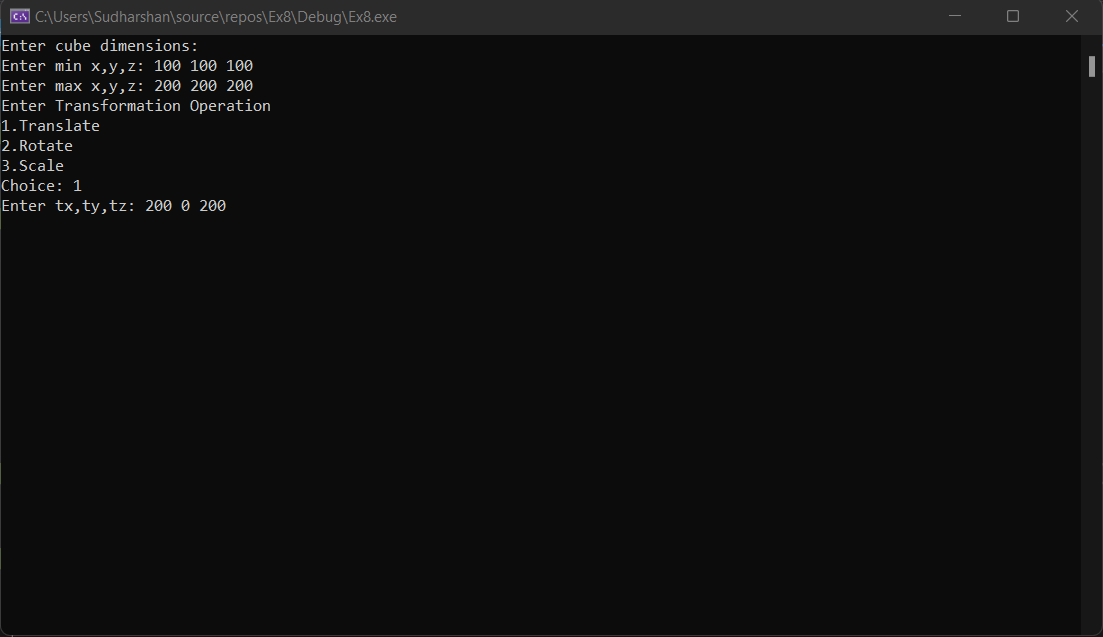
myInit();

glutMainLoop();

return 1;

}

**OUTPUT:**

****