**UCS1712 – GRAPHICS AND MULTIMEDIA LAB**

**Ex. No.8 3D Transformations**

**Date:** 20/9/21 **Name:** Srinath S

**Class:** CSE-C  **Roll:** 185001205

**Question:**

Perform the following basic 3D Transformations on any 3D Object.

1) Translation

2) Rotation

3) Scaling

Use only homogeneous coordinate representation and matrix multiplication to perform transformations.

Set the camera to any position on the 3D space. Have (0,0,0) at the center of the screen.

Draw X , Y and Z axis.

**Code:**

#include<GL/glut.h>

#include<iostream>

#include<math.h>

using namespace std;

#define PI 3.14159265358979323846264338327950288419716939937510582

typedef struct Point {

double x, y, z, h;

}Point;

typedef struct Face {

Point v[4];

}Face;

typedef struct Cuboid {

Point v[8];

Face faces[6];

}Cuboid;

int assignList[6][4] = { {0, 1, 3, 2}, {0, 4, 5, 1}, {0, 4, 6, 2},

{4, 5, 7, 6}, {2, 6, 7, 3}, {1, 5, 7, 3}

};

float colors[6][3] = { {0, 1, 0}, {1, 0, 0}, {0, 0, 1},

{1, 1, 0}, {0, 1, 1}, {1, 0, 1}

};

double tMat[4][4];

bool tflag = false;

Cuboid cuboid, tcuboid;

Cuboid initCuboid() {

Cuboid cuboid;

double po[8][3] = { {-25, 25, 0}, {25, 25, 0}, {-25, -25, 0}, {25,

-25, 0},

{-25, 25, 50}, {25, 25, 50}, {-25, -25, 50}, {25,

-25, 50}

};

for (int i = 0; i < 8; i++) {

cuboid.v[i].x = po[i][0];

cuboid.v[i].y = po[i][1];

cuboid.v[i].z = po[i][2];

cuboid.v[i].h = 1;

}

for (int i = 0; i < 6; i++) {

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

cuboid.faces[i].v[j] = cuboid.v[assignList[i][j]];

}

}

return cuboid;

}

void myInit() {

glClearColor(1.0, 1.0, 1.0, 0.0);

glColor3f(0, 0, 0);

glEnable(GL\_BLEND);

glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA);

glLoadIdentity();

glOrtho(-200, 200, -200, 200, -200, 200);

glEnable(GL\_DEPTH\_TEST);

}

void disp() {

glRotatef(30, 1, 0, 0);

glRotatef(30, 0, 1, 0);

}

void transformCuboid() {

tflag = true;

for (int i = 0; i < 8; i++) {

cout << cuboid.v[i].x << " " << cuboid.v[i].y << " "<<cuboid.v[i].z<<"\n";

}

cout << "\n\n";

for (int p = 0; p < 8; p++) {

double pnt[4][1], pnt1[4][1];

pnt[0][0] = cuboid.v[p].x;

pnt[1][0] = cuboid.v[p].y;

pnt[2][0] = cuboid.v[p].z;

pnt[3][0] = cuboid.v[p].h;

memset(pnt1, 0, sizeof(pnt1));

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

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

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

pnt1[i][j] += tMat[i][k] \* pnt[k][j];

}

}

}

tcuboid.v[p].x = pnt1[0][0];

tcuboid.v[p].y = pnt1[1][0];

tcuboid.v[p].z = pnt1[2][0];

tcuboid.v[p].h = pnt1[3][0];

}

for (int i = 0; i < 6; i++) {

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

tcuboid.faces[i].v[j] =

tcuboid.v[assignList[i][j]];

}

}

for (int i = 0; i < 8; i++) {

cout << tcuboid.v[i].x << " " << tcuboid.v[i].y << " "<<tcuboid.v[i].z<<"\n";

}

glutPostRedisplay();

}

void getTransformMatrix() {

memset(tMat, 0, sizeof(tMat));

tMat[0][0] = tMat[1][1] = tMat[2][2] = tMat[3][3] = 1;

int ch;

cout << "Menu:\n\t1.Translation\n\t2.Rotation\n\t3.Scaling\n\tChoice:";

cin >> ch;

switch (ch) {

case 1:

cout << "Enter translation parameters: ";

cin >> tMat[0][3] >> tMat[1][3] >> tMat[2][3];

break;

case 2:

cout << "Enter degree of rotation: ";

double deg;

cin >> deg;

deg = deg \* PI / 180;

tMat[0][0] = cos(deg);

tMat[0][1] = -sin(deg);

tMat[1][1] = tMat[0][0];

tMat[1][0] = -tMat[0][1];

break;

case 3:

cout << "Enter scaling parameters: ";

cin >> tMat[0][0] >> tMat[1][1] >> tMat[2][2];

break;

default: cout << "Incorrect choice\n";

}

transformCuboid();

}

void displayCuboid(Cuboid cuboid, double alpha = 0.6) {

for (int i = 0; i < 6; i++) {

glColor4f(colors[i][0], colors[i][1], colors[i][2], alpha);

glBegin(GL\_POLYGON);

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

glVertex3d(cuboid.faces[i].v[j].x, cuboid.faces[i].v[j].y, cuboid.faces[i].v

[j].z);

}

glEnd();

}

}

void myDisplay() {

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

glColor4f(0, 0, 0, 1);

glBegin(GL\_LINES);

glVertex3d(300, 0, 0);

glVertex3d(-300, 0, 0);

glEnd();

glBegin(GL\_LINES);

glVertex3d(0, 300, 0);

glVertex3d(0, -300, 0);

glEnd();

glBegin(GL\_LINES);

glVertex3d(0, 0, 300);

glVertex3d(0, 0, -300);

glEnd();

displayCuboid(cuboid);

if (tflag)displayCuboid(tcuboid, 1);

glFlush();

getTransformMatrix();

}

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

cuboid = initCuboid();

glutInit(&argc, argv);

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

glutInitWindowSize(960, 960);

glutInitWindowPosition(0, 0);

glutCreateWindow("3D transformations");

myInit();

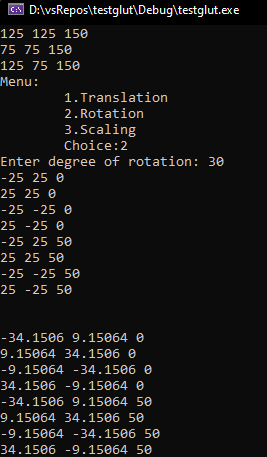
disp();

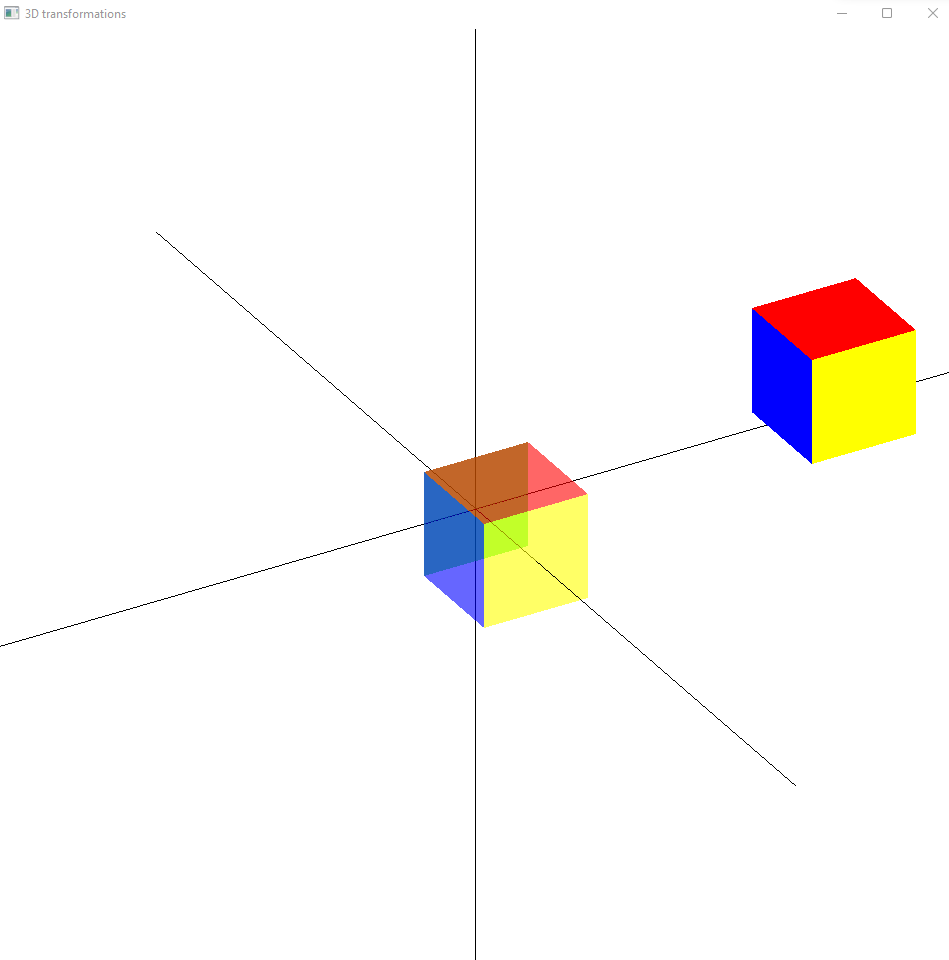
glutDisplayFunc(myDisplay);

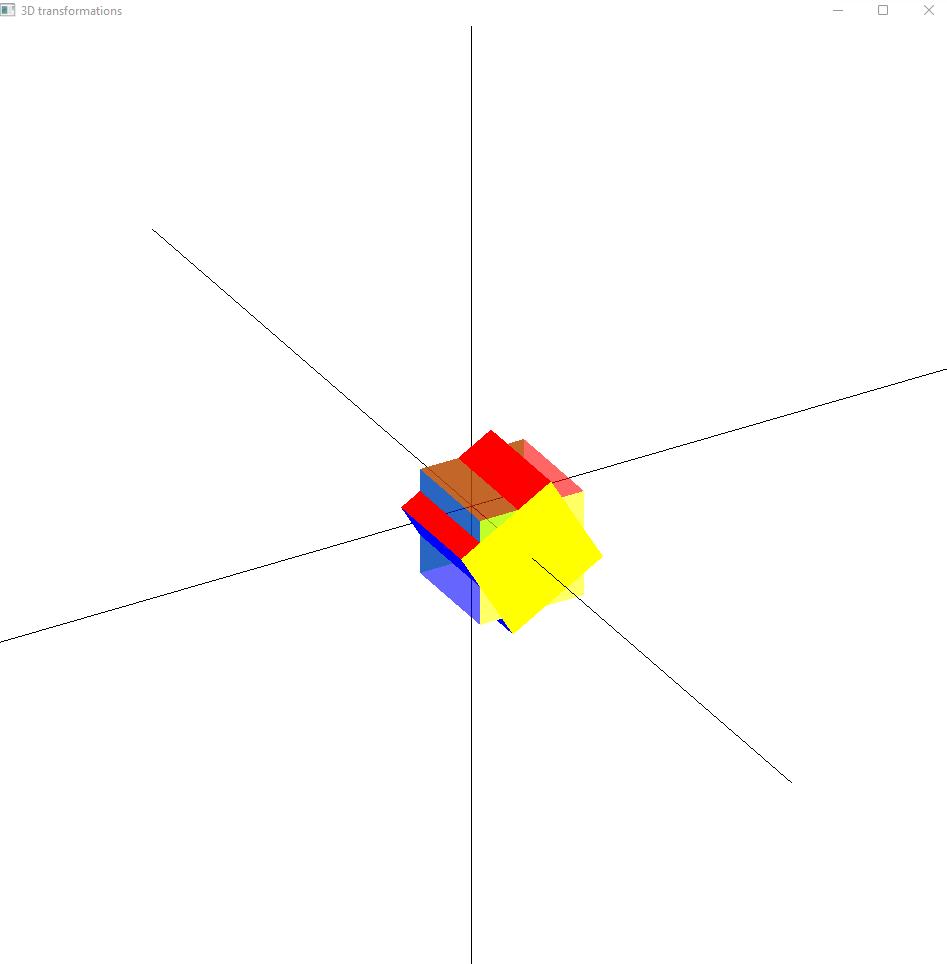
glutMainLoop();

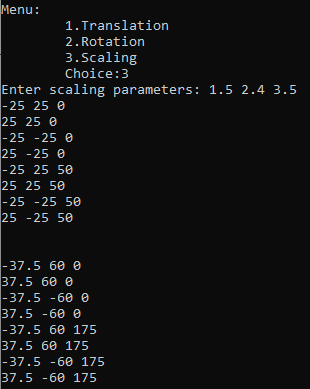
}

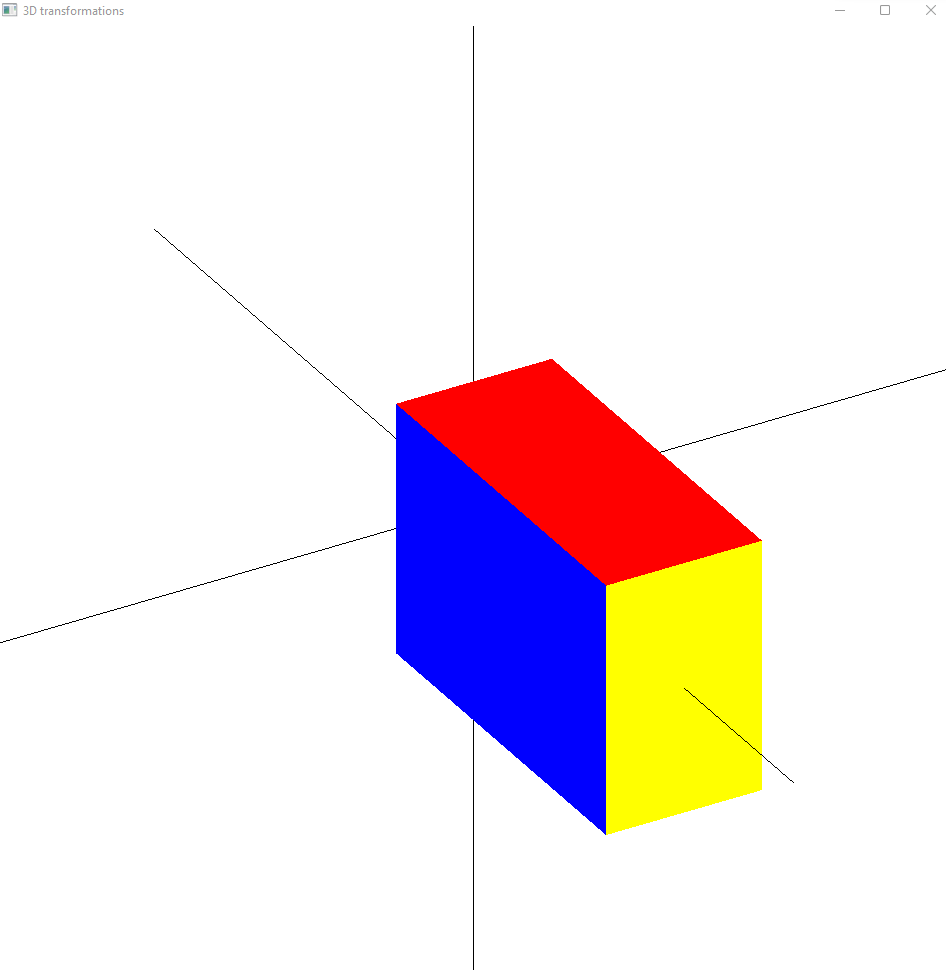
**Outputs:**

****

****

****

****

****