SSN COLLEGE OF ENGINEERING, KALAVAKKAM DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
UCS1712 - GRAPHICS AND MULTIMEDIA LAB

Assignment- 8 - 3-Dimensional Transformations in C++ using OpenGL

Name: Sabarivasan V **Reg.No: 205001085**

Ex.No: 08

Aim:

To 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 axes.

ALGORITHM:

1. Start the program.
2. Display the cube.
3. Input the translation vector tx,ty,tz.
4. Using the function line, display the object before and after translation.
5. Input the scaling factor and reference point.
6. Using the function line, display the object before and after scaling.
7. Input the rotation angle.
8. Using the function line,display the object before and after rotation.
9. Stop the Program.

CODE:

```
#include<stdio.h>
#include<GL/glut.h> //Change to <GLUT/glut.h> in Mac
#include<math.h>
#include<string.h>
#include<iostream>
using namespace std;
#define pi 3.142857
```

```
typedef float Matrix4[4][4];
Matrix4 theMatrix; static
GLfloat input[8][3] =
{{40,40,-50},{90,40,-50},{90,90,-50},{40,90,-50},{30,30,0},{80,30,0},{80,80,0},{30,80,0}};
float output[8][3]; float tx=100,
ty=100, tz=100; float sx=-2,
sy=2, sz=2; float angle=60; int
choice, choiceRot; void
setIdentityM(Matrix4 m) { for
(int i = 0; i < 4; i++) for (int j =
0; j < 4; j++) m[i][j] = (i == j);
}

//PUT SOME FUNCTION HERE

void translate(int tx, int ty, int tz) {
  for (int i = 0; i < 8; i++) {


            output[i][0] = input[i][0] + tx;
            output[i][1] = input[i][1] + ty;
            output[i][2] = input[i][2] + tz;
  } } void scale(int sx, int sy, int
sz) {
  theMatrix[0][0] = sx;
theMatrix[1][1] = sy;
theMatrix[2][2] = sz; } void
RotateX(float angle) { angle =
angle * 3.142 / 180;
theMatrix[1][1] = cos(angle);
theMatrix[1][2] = -sin(angle);
theMatrix[2][1] = sin(angle);
theMatrix[2][2] = cos(angle); }
void RotateY(float angle) {
angle = angle * 3.14 / 180;
theMatrix[0][0] = cos(angle);
theMatrix[0][2] = -sin(angle);
theMatrix[2][0] = sin(angle);
theMatrix[2][2] = cos(angle); }
void RotateZ(float angle) {
angle = angle * 3.14 / 180;
theMatrix[0][0] = cos(angle);
theMatrix[0][1] = sin(angle);
theMatrix[1][0] = -sin(angle);
theMatrix[1][1] = cos(angle);
```

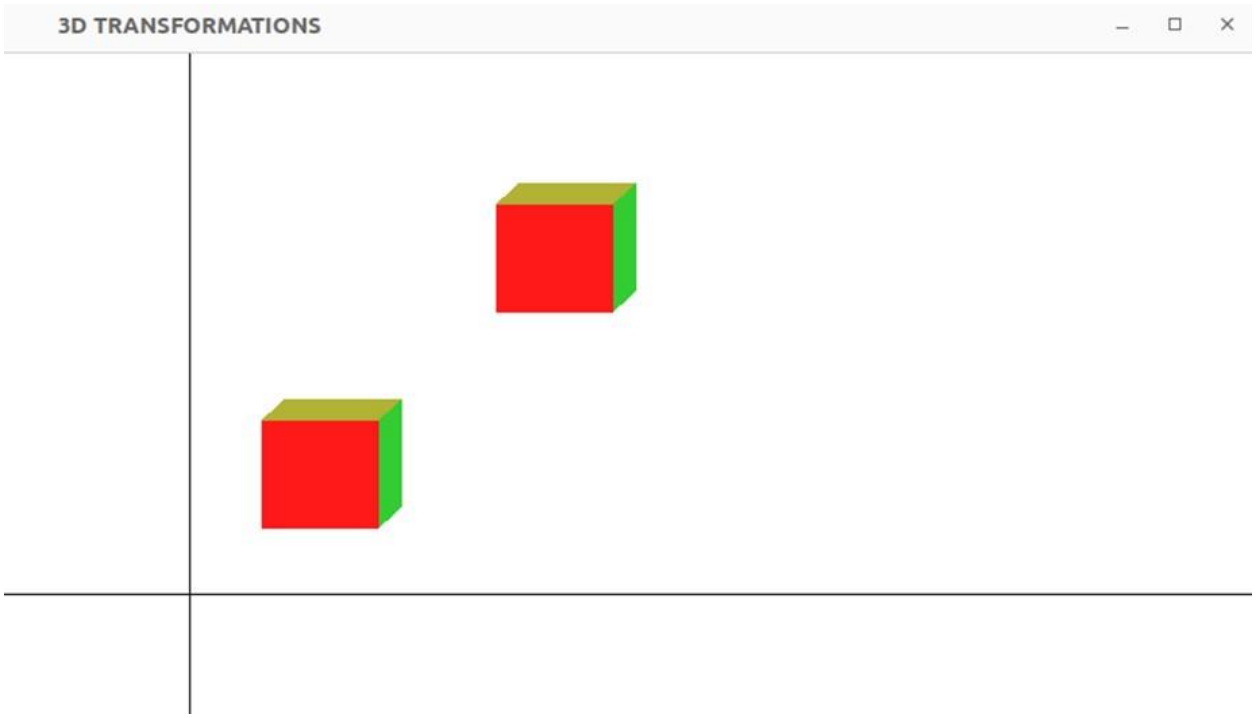
```
} void multiplyM()
{
  for (int i = 0; i < 8; i++) { for
            (int j = 0; j < 3; j++) {
            output[i][j] = 0;
            for (int k = 0; k < 3; k++) {
            output[i][j] = output[i][j] + input[i][k] * theMatrix[k][j];
            }
            }
  }
}

//To draw the solid void
draw(float a[8][3]) {
  glBegin(GL_QUADS);
  glColor3f(0.7, 0.4, 0.5); //behind
  glVertex3fv(a[0]);
  glVertex3fv(a[1]);
  glVertex3fv(a[2]);
  glVertex3fv(a[3]);
  glColor3f(0.8, 0.2, 0.4);
  //bottom glVertex3fv(a[0]);
  glVertex3fv(a[1]);
  glVertex3fv(a[5]);
  glVertex3fv(a[4]);
  glColor3f(0.3, 0.6, 0.7); //left
  glVertex3fv(a[0]);
  glVertex3fv(a[4]);
  glVertex3fv(a[7]);
  glVertex3fv(a[3]);
  glColor3f(0.2, 0.8, 0.2); //right
  glVertex3fv(a[1]);
  glVertex3fv(a[2]);
  glVertex3fv(a[6]);
  glVertex3fv(a[5]);
  glColor3f(0.7, 0.7, 0.2); //up
  glVertex3fv(a[2]);
  glVertex3fv(a[3]);
  glVertex3fv(a[7]);
  glVertex3fv(a[6]);
  glColor3f(1.0, 0.1, 0.1);
  glVertex3fv(a[4]);
  glVertex3fv(a[5]);
  glVertex3fv(a[6]);
  glVertex3fv(a[7]);
  glEnd();
```

```
}


/* This is just to call the functions also draw X and Y axis
 here and use output to label stuff) */
void display (void){

            glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
            //black glColor3f(0.0,
            0.0, 0.0);

  glBegin(GL_LINES); // Plotting X-Axis
  glVertex3d(-1000, 0, 0);
  glVertex3d(1000, 0, 0); glEnd();
  glBegin(GL_LINES); // Plotting Y-Axis
  glVertex3d(0, -1000, 0); glVertex3d(0,
  1000, 0); glEnd();
  glBegin(GL_LINES); // Plotting Z-Axis
  glVertex3d(0, 0, -1000); glVertex3d(0,
  0, 1000); glEnd();


            //Call function
   draw(input);
  setIdentityM(theMatrix);
  switch (choice) { case 1:
            translate(tx, ty, tz);
            break;
  case 2:
            scale(sx, sy, sz);
            multiplyM();
            break;
  case 3: switch (choiceRot)
            { case 1:
            RotateX(angle);
            break; case 2:
            RotateY(angle);
            break; case 3:
            RotateZ(angle);
  break; default: break;
  } multiplyM(); break;
  } draw(output);
  glFlush();

            glFlush();
}
```

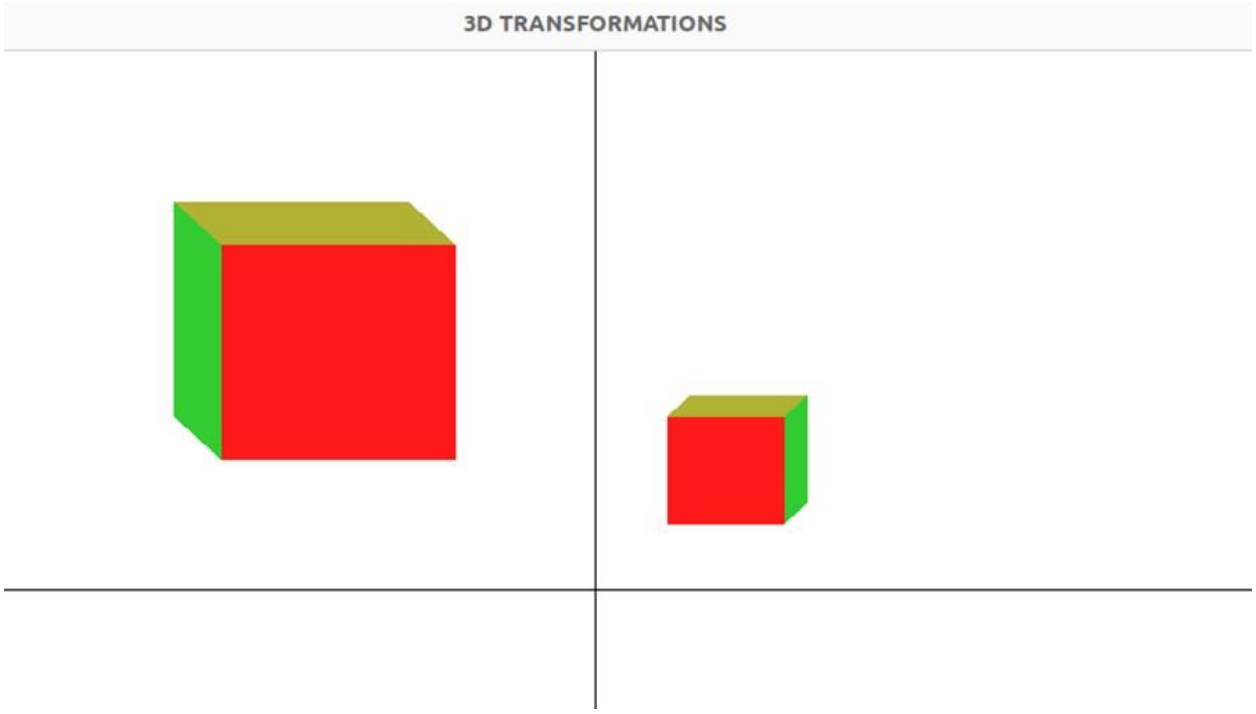
```
int main (int argc, char** argv){ /*--------WINDOW INITS-------*/
        glutInit(&argc, argv); //Mandatory. Initializes the GLUT library.
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
        glutInitWindowSize(1380, 700); //Set the size of output window (kinda optional)
        glutInitWindowPosition(200, 200); //position of output window on screen (optional)
        glutCreateWindow("3D TRANSFORMATIONS");// Giving name to window

        /*-------OTHER INITS-------*/ glClearColor(1.0, 1.0, 1.0, 1.0); //Clear the buffer values
        for color AND set these values
        /*can set initial color here also*/ glMatrixMode(GL_PROJECTION); //Uses something
        called "projection matrix" to represent glLoadIdentity(); //load the above matrix to fill with
        identity values glOrtho(-454.0, 454.0, -250.0, 250.0, -250.0, 250.0); gluPerspective(100, 100,
        100, 100); glEnable(GL_DEPTH_TEST); cout << "Enter your choice
        number:\n1.Translation\n2.Scaling\n3.Rotation\n=>";
  cin >> choice;
  switch (choice)
  { case 1: break;
  case 2:
        break;
  case 3:
        cout << "Enter your choice for Rotation about axis:\n1.parallel to X-axis." <<
        "(y& z)\n2.parallel to Y-axis.(x& z)\n3.parallel to Z-axis." <<
        "(x& y)\n =>";
        cin >>
        choiceRot; break;
  default:
        break;
  }
   glutDisplayFunc(display); //sets the display callback for the current window
   glutMainLoop(); //Enters event processing loop. Compulsory return 0;
}
```

OUTPUT:

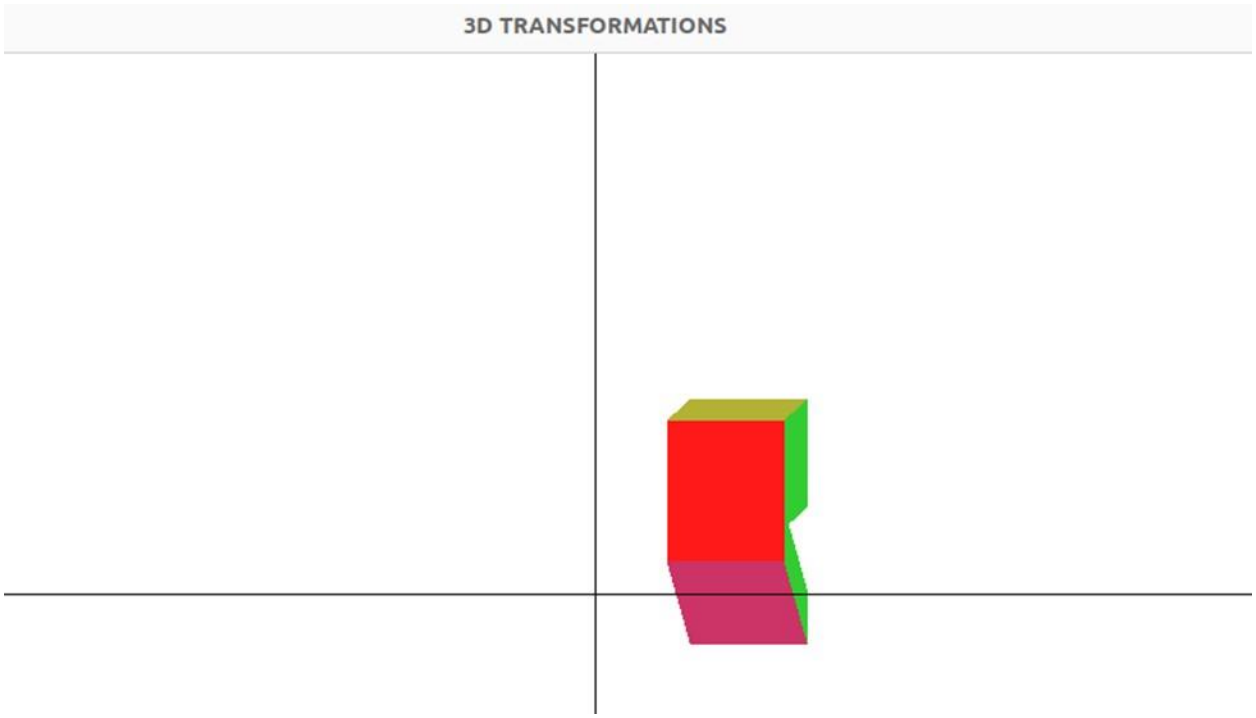
Translate along X and Y and Z



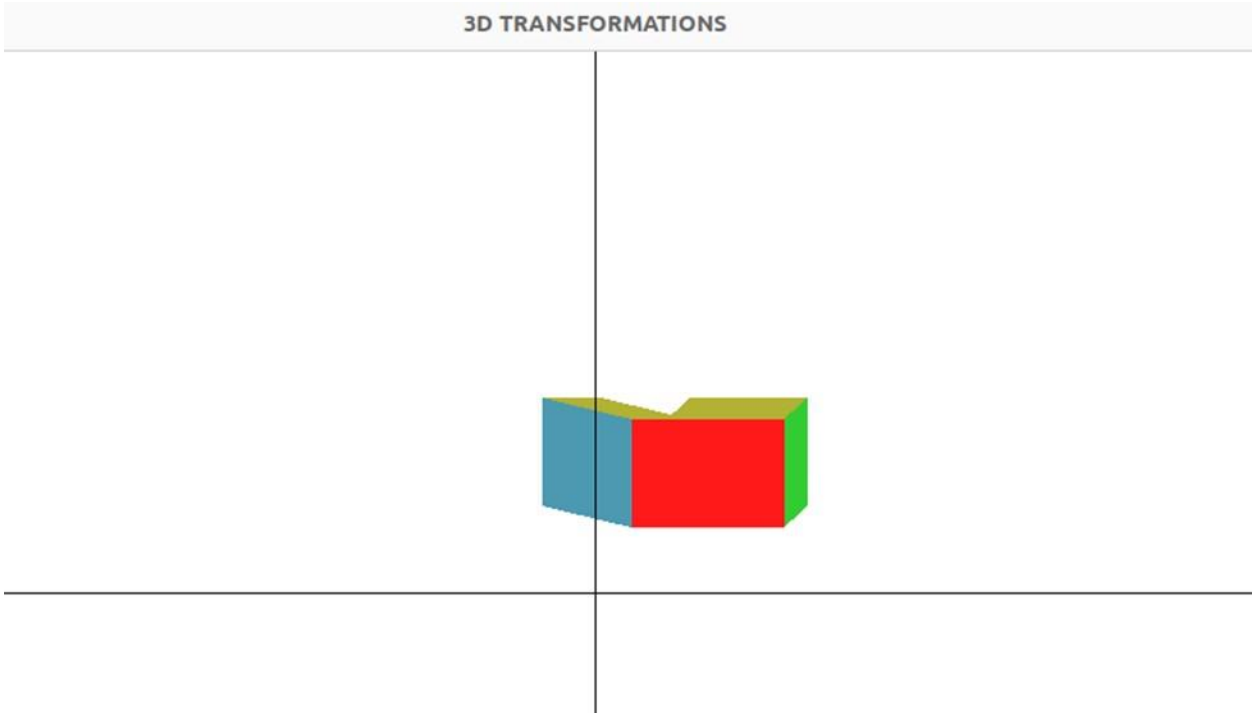
Scaling at X Y and Z



Rotation wrt X



Rotate wrt Y



Rotate wrt Z