**SSN COLLEGE OF ENGINEERING, KALAVAKKAM**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**UCS1712 – GRAPHICS AND MULTIMEDIA LAB**

------------------------------------------------------------------------------------------------------------

**Lab Exercise 5**: : 2D Transformations in C++ using OpenGL

To apply the following 2D transformations on objects and to render the final output along with the original object.

1) Translation

2) Rotation

a) about origin

b) with respect to a fixed point (xr,yr)

3) Scaling with respect to

a) origin - Uniform Vs Differential Scaling

b) fixed point (xf,yf)

4) Reflection with respect to

a) x-axis

b) y-axis

c) origin

d) the line x=y

5) Shearing

a) x-direction shear

b) y-direction shear

Note: Use Homogeneous coordinate representations and matrix multiplication to perform

transformations. Divide the output window into four quadrants. (Use LINES primitive to draw x and y axis.

***Aim:***

To implement 2D transformations on objects using C++ using OpenGL

***Algorithm:***

Application of a sequence of transformations to a point:

P’ = M2.M1.P

= M.P

Composite transformations is formed by calculating the matrix product of the individual

transformations and forming products of transformation matrix

***5a.cpp:***

#include <stdio.h>

#include <GL/glut.h>

#include <math.h>

#include <string.h>

#define pi 3.142857

void mm(double m[3][3], double v[3])

{

for (int i = 0; i < 3; ++i)

{

double temp = 0;

for (int k = 0; k < 3; ++k)

temp += m[i][k] \* v[k];

v[i] = temp;

}

}

int X = 100, Y = -50;

void draw\_pixel(int x, int y)

{

glBegin(GL\_POINTS);

glVertex2i(x, y);

glEnd();

}

void output(int x, int y, const char \*string)

{

glRasterPos2f(x, y);

int len, i;

len = (int)strlen(string);

for (i = 0; i < len; i++)

{

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_12, string[i]);

}

}

void obj(int a, int b, int c, int d, int w, int x, int y, int z)

{

glBegin(GL\_QUADS);

glVertex2d(a, b);

glVertex2d(c, d);

glVertex2d(w, x);

glVertex2d(y, z);

glEnd();

}

void myInit(void)

{

glClearColor(0.0, 0.0, 0.0, 1.0);

glColor3f(0.0, 1.0, 0.0);

glPointSize(1.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(-780, 780, -420, 420);

}

void display(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(1.0, 1.0, 1.0);

glBegin(GL\_LINES);

glVertex2d(0, 420);

glVertex2d(0, -420);

glEnd();

glBegin(GL\_LINES);

glVertex2d(780, 0);

glVertex2d(-780, 0);

glEnd();

glColor3f(0.0, 1.0, 0.0);

// TRANSLATION

double x1[3];

double x2[3];

double x3[3];

double x4[3];

x1[2] = x2[2] = x3[2] = x4[2] = 1;

x1[0] = 100;

x1[1] = 100;

x2[0] = 200;

x2[1] = 100;

x3[0] = 200;

x3[1] = 200;

x4[0] = 100;

x4[1] = 200;

obj(x1[0], x1[1], x2[0], x2[1], x3[0], x3[1], x4[0], x4[1]);

output(110, 210, "originalA:");

double T[3][3] = {{1, 0, 150}, {0, 1, 150}, {0, 0, 1}};

mm(T, x1);

mm(T, x2);

mm(T, x3);

mm(T, x4);

obj(x1[0], x1[1], x2[0], x2[1], x3[0], x3[1], x4[0], x4[1]);

output(260, 360, "translatedA:");

// ROTATION

double R[3][3] = {{cos(pi / 4), -sin(pi / 4), 0}, {sin(pi / 4), cos(pi / 4), 0}, {0, 0, 1}};

x1[0] = 100;

x1[1] = 100;

x2[0] = 200;

x2[1] = 100;

x3[0] = 200;

x3[1] = 200;

x4[0] = 100;

x4[1] = 200;

mm(R, x1);

mm(R, x2);

mm(R, x3);

mm(R, x4);

// printf("%lf%lf%lf%lf%lf%lf%lf%lf%lf", x1[0], x1[1], x2[0], x2[1], x3[0], x3[1], x4[0], x4[1]);

printf("%f%f%f%f%f%f%f%f", x1[0], x1[1], x2[0], x2[1], x3[0], x3[1], x4[0], x4[1]);

obj(x1[0], x1[1], x4[0], x4[1], x3[0], x3[1], x2[0], x2[1]);

output(10, 300, "rotatedA:");

printf("%lf", cos(pi / 4));

// PIVOTROTATION

double PR[3][3] = {{cos(-pi / 4), -sin(-pi / 4), -X \* cos(-pi / 4) + Y \* sin(-pi / 4) + X}, {sin(-pi / 4), cos(-pi / 4), -X \* sin(-pi / 4) - Y \* cos(-pi / 4) + Y}, {0, 0, 1}};

x1[0] = 100;

x1[1] = 100;

x2[0] = 200;

x2[1] = 100;

x3[0] = 200;

x3[1] = 200;

x4[0] = 100;

x4[1] = 200;

glColor3f(1.0, 1.0, 1.0);

output(X - 15, Y - 20, "(rotPivot)");

glPointSize(5);

glBegin(GL\_POINTS);

glVertex2d(X, Y);

glEnd();

glColor3f(0.0, 1.0, 0.0);

mm(PR, x1);

mm(PR, x2);

mm(PR, x3);

mm(PR, x4);

// printf("%lf%lf%lf%lf%lf%lf%lf%lf%lf", x1[0], x1[1], x2[0], x2[1], x3[0], x3[1], x4[0], x4[1]);

printf("%f%f%f%f%f%f%f%f", x1[0], x1[1], x2[0], x2[1], x3[0], x3[1], x4[0], x4[1]);

obj(x1[0], x1[1], x4[0], x4[1], x3[0], x3[1], x2[0], x2[1]);

// obj(x1[0],x1[1],x2[0],x2[1],x3[0],x3[1],x4[0],x4[1]);

// SCALING

x1[0] = -50;

x1[1] = -50;

x2[0] = -100;

x2[1] = -50;

x3[0] = -100;

x3[1] = -100;

x4[0] = -50;

x4[1] = -100;

glColor3f(1.0, 0.0, 1.0);

obj(x1[0], x1[1], x2[0], x2[1], x3[0], x3[1], x4[0], x4[1]);

output(-100, -40, "originalB:");

double S[3][3] = {{2, 0, 0}, {0, 2, 0}, {0, 0, 1}};

mm(S, x1);

mm(S, x2);

mm(S, x3);

mm(S, x4);

obj(x1[0], x1[1], x2[0], x2[1], x3[0], x3[1], x4[0], x4[1]);

output(-200, -90, "scaledB:");

// PIVOTSCALING

int px = -200, py = -200;

glColor3f(1.0, 1.0, 1.0);

output(px - 15, py - 20, "(scalePivot)");

glPointSize(5);

glBegin(GL\_POINTS);

glVertex2d(px, py);

glEnd();

glColor3f(1.0, 0.0, 1.0);

x1[0] = -50;

x1[1] = -50;

x2[0] = -100;

x2[1] = -50;

x3[0] = -100;

x3[1] = -100;

x4[0] = -50;

x4[1] = -100;

double PS[3][3] = {{2, 0, 200}, {0, 2, 200}, {0, 0, 1}};

mm(PS, x1);

mm(PS, x2);

mm(PS, x3);

mm(PS, x4);

obj(x1[0], x1[1], x2[0], x2[1], x3[0], x3[1], x4[0], x4[1]);

output(10, 110, "pivotscaledB:");

// DIISCALING

glColor3f(0.0, 1.0, 1.0);

obj(-50, 50, -100, 50, -100, 100, -50, 100);

output(-100, 110, "originalC:");

int dsx = 2, dsy = 3.5;

glColor3f(1.0, 1.0, 0.0);

obj(-50 \* dsx, 50 \* dsy, -100 \* dsx, 50 \* dsy, -100 \* dsx, 100 \* dsy, -50 \* dsx, 100 \* dsy);

output(-200, 310, "diffScaledC:");

glFlush();

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(1366, 768);

glutInitWindowPosition(0, 0);

glutCreateWindow("Transformations");

myInit();

glutDisplayFunc(display);

glutMainLoop();

}

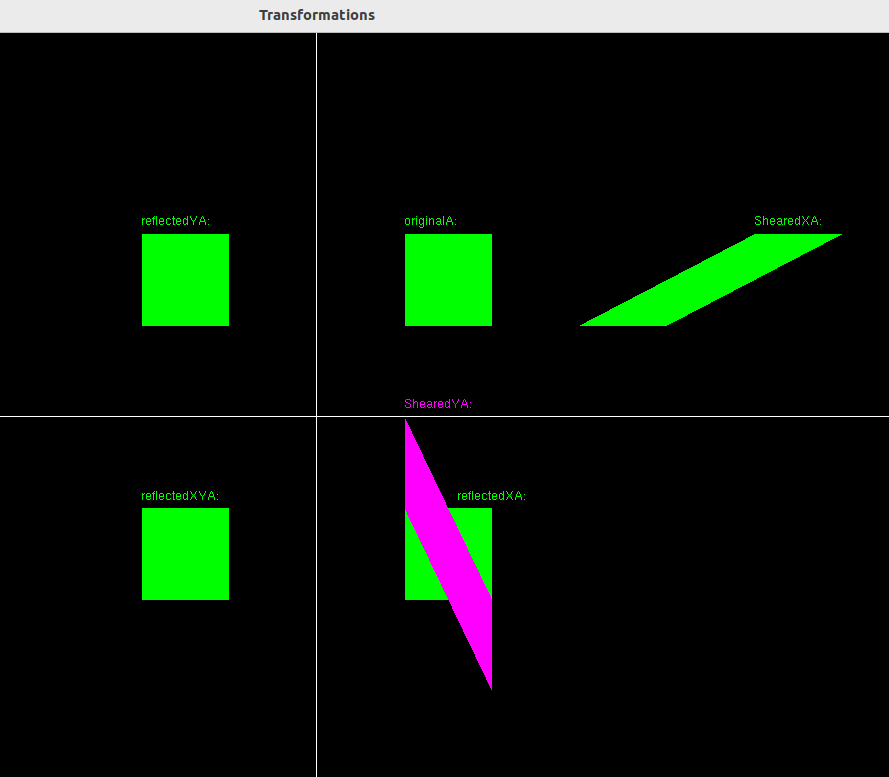
***5b.cpp:***

***run.sh:***g++ 5.cpp -lGL -lglut -lGLU

./a.out

***Sample I/O:***





***Learning Outcomes:***

Learnt to do composite transformations. Learnt to do translation, reflection, shearing, rotation and scaling.