**UCS 1712 – GRAPHICS AND MULTIMEDIA LAB**

**ASSIGNMENT – 6**

**VISHAL N**

**185001198**

* + 1. **CSEC**

**1. COMPOSITE 2D TRANSFORMATIONS:**

#include <GL/glut.h>

// #include <GL/freeglut.h>

#include <stdio.h>

#include <iostream>

#include <math.h>

#include <vector>

#include <cmath>

using namespace std;

vector<double> x\_coordinates;

vector<double> y\_coordinates;

int x\_trans, y\_trans, x\_fixed, y\_fixed;

double rotation\_angle, shear, x\_scale, y\_scale;

int polygon, edges, choice, x, y, axis;

double res[3][1] = {0};

double res33[3][3] = {{0}};

double const  REFLECTION\_MATRIX[4][3][3] = {{{-1, 0, 0}, {0, -1, 0}, {0, 0, 1}}, {{1, 0, 0}, {0, -1, 0}, {0, 0, 1}}, {{-1, 0, 0}, {0, 1, 0}, {0, 0, 1}}, {{0, 1, 0}, {1, 0, 0}, {0, 0, 1}}};

void myInit(void) {

    glClearColor(1.0, 1.0, 1.0, 0.0);

    glColor3f(0.0f, 0.0f, 0.0f);

    glPointSize(0.05);

    glMatrixMode(GL\_PROJECTION);

    glLoadIdentity();

    gluOrtho2D(-600.0, 600.0, -600.0, 600.0);

}

void drawAxis(void){

    glBegin(GL\_LINES);

    glVertex2d(-600, 0);

    glVertex2d(600, 0);

    glEnd();

    glBegin(GL\_LINES);

    glVertex2d(0, -600);

    glVertex2d(0, 600);

    glEnd();

}

double round(double d)

{

    return floor(d + 0.5);

}

void mutliplyMatrices31(const double tr[3][3], const double pt[3][1]){

    memset(res, 0, 3 \* 1 \* sizeof(double));

    for(int i = 0; i < 3; i++)

        for(int j = 0; j < 1; j++)

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

                res[i][j] += tr[i][k] \* pt[k][j];

}

void mutliplyMatrices33(const double tr[3][3], const double pt[3][3]){

    memset(res33, 0, 3 \* 3 \* sizeof(double));

    for(int i = 0; i < 3; i++)

        for(int j = 0; j < 3; j++)

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

                res33[i][j] += tr[i][k] \* pt[k][j];

}

void drawPolygon(void){

    if (polygon == 1){

        glBegin(GL\_LINES);

    }

    else{

        glBegin(GL\_POLYGON);

    }

    for (int i = 0; i < edges; i++)

    {

        glVertex2d(x\_coordinates[i], y\_coordinates[i]);

    }

    glEnd();

}

void drawPolygonRotationScaling(){

    double rotMat[3][3] = {{cos(rotation\_angle), -sin(rotation\_angle), x\_fixed \* (1 - cos(rotation\_angle)) + y\_fixed \* sin(rotation\_angle)}, {sin(rotation\_angle), cos(rotation\_angle), y\_fixed \* (1 - cos(rotation\_angle)) - x\_fixed \* sin(rotation\_angle)}, {0.0, 0.0, 1.0}};

    double scalMat[3][3] = {{x\_scale, 0, x\_fixed \* (1 - x\_scale)}, {0, y\_scale, y\_fixed \* (1 - y\_scale)}, {0, 0, 1}};

    mutliplyMatrices33(rotMat, scalMat);

    glBegin(GL\_QUADS);

    glColor3f(1.0, 0.0, 1.0);

    for(int i = 0; i < 4; i++){

        double pts[3][1] = {{x\_coordinates[i]}, {y\_coordinates[i]}, {1}};

        mutliplyMatrices31(res33, pts);

        glVertex2i((int)res[0][0], (int)res[1][0]);

    }

    glEnd();

}

void drawReflectionShearing(){

    if (axis == 1){

        double shear\_matrix [3][3] = {{1, shear, 0}, {0, 1, 0}, {0, 0, 1}};

        mutliplyMatrices33(REFLECTION\_MATRIX[axis], shear\_matrix);

        glBegin(GL\_QUADS);

        glColor3f(1.0, 0.0, 1.0);

        for(int i = 0; i < 4; i++){

            double pts[3][1] = {{x\_coordinates[i]}, {y\_coordinates[i]}, {1}};

            mutliplyMatrices31(res33, pts);

            glVertex2i((int)res[0][0], (int)res[1][0]);

        }

        glEnd();

        }

    else{

        double shear\_matrix [3][3] = {{1, 0, 0}, {shear, 1, 0}, {0, 0, 1}};

        mutliplyMatrices33(REFLECTION\_MATRIX[axis], shear\_matrix);

        glBegin(GL\_QUADS);

        glColor3f(1.0, 0.0, 1.0);

        for(int i = 0; i < 4; i++){

            double pts[3][1] = {{x\_coordinates[i]}, {y\_coordinates[i]}, {1}};

            mutliplyMatrices31(res33, pts);

            glVertex2i((int)res[0][0], (int)res[1][0]);

            }

        glEnd();

        }

}

void myDisplay(void){

    glClear(GL\_COLOR\_BUFFER\_BIT);

    glColor3f(0.0, 0.0, 0.0);

    drawAxis();

    drawPolygon();

    glFlush();

    while(true){

        glClear(GL\_COLOR\_BUFFER\_BIT);

        glColor3f(0.0, 0.0, 0.0);

        cout << "Enter your choice of transformation :\n";

        cout << "1. Rotation and scaling" << endl;

        cout << "2. Reflection and shearing" << endl;

        cout << "3. Exit" << endl;

        cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

        cin >> choice;

        if (choice == 3) return;

        if (choice == 1){

            cout << "Enter the angle for rotation: ";

            cin >> rotation\_angle;

            rotation\_angle \*= M\_PI / 180;

            cout << "\n Enter the fixed point: ";

            cin >> x\_fixed >> y\_fixed;

            cout << "Enter the scaling factor: ";

            cin >> x\_scale >> y\_scale;

        }

        else if (choice == 2)

        {

            cout << "Enter the axis for reflection [1 - X Axis, 2 - Y Axis, 3 - X=Y Line]: ";

            cin >> axis;

            cout << "Enter the shearing factor: ";

            cin >> shear;

        }

        if (choice == 1)

        {

            drawAxis();

            drawPolygon();

            drawPolygonRotationScaling();

        }

        else if (choice == 2)

        {

            drawAxis();

            drawPolygon();

            drawReflectionShearing();

        }

        glFlush();

    }

}

int main(int argc, char \*\*argv)

{

    cout << "Enter the number of edges: ";

    cin >> edges;

    if (edges == 2)

    {

        polygon = 1;

    }

    else

    {

        polygon = -1;

    }

    cout << "Enter vertices: \n";

    for (int i = 0; i < edges; i++)

    {

        cout << "Enter co-ordinates for vertex " << i + 1 << " (X, Y): ";

        cin >> x >> y;

        x\_coordinates.push\_back(x);

        y\_coordinates.push\_back(y);

    }

    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

    glutInitWindowSize(600, 600);

    glutInitWindowPosition(0, 0);

    glutCreateWindow("Composite 2D Transformations");

    myInit();

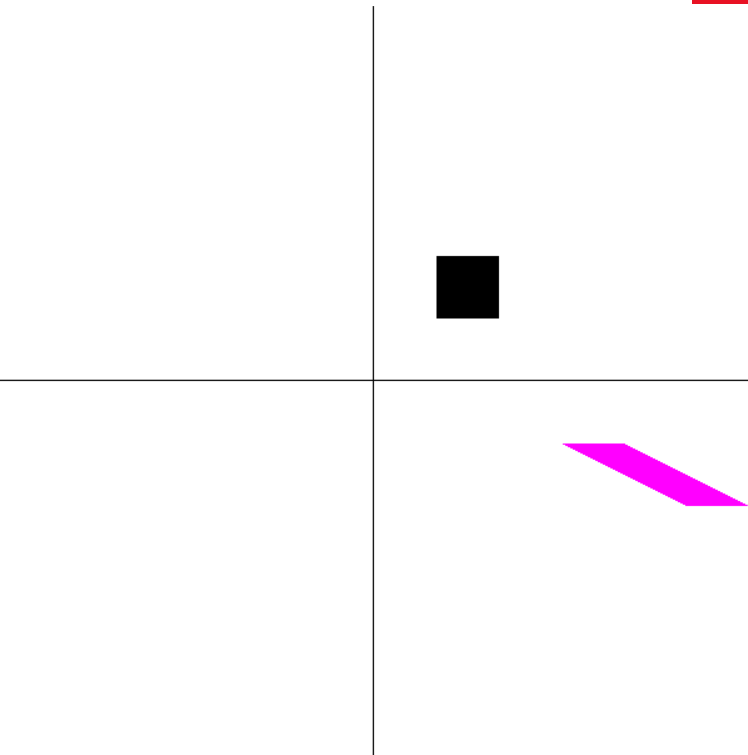
    glutDisplayFunc(myDisplay);

    glutMainLoop();

    return 1;

}

**OUTPUT:**

****

**2. WINDOWING:**

#include <GL/glut.h>

// #include <GL/freeglut.h>

#include <stdio.h>

#include <iostream>

#include <math.h>

#include <vector>

#include <cmath>

using namespace std;

class Point {

public:

    int x, y;

};

void myInit(void) {

    glClearColor(1.0, 1.0, 1.0, 0.0);

    glColor3f(0.0f, 0.0f, 0.0f);

    glPointSize(0.05);

    glMatrixMode(GL\_PROJECTION);

    glLoadIdentity();

    gluOrtho2D(-600.0, 600.0, -600.0, 600.0);

}

void drawAxis(void){

    glBegin(GL\_LINES);

    glVertex2d(-600, 0);

    glVertex2d(600, 0);

    glEnd();

    glBegin(GL\_LINES);

    glVertex2d(0, -600);

    glVertex2d(0, 600);

    glEnd();

}

void myDisplay(void){

    double wx\_org = 0, wy\_org = 0, wx\_min = 0, wx\_max = 300, wy\_min = 0, wy\_max = 200;

    double vx\_org = 350, vy\_org = 0, vx\_min = 350, vy\_min = 0, vx\_max = 500, vy\_max = 400;

    double sx = (vx\_max - vx\_min)/(wx\_max - wx\_min);

    double sy = (vy\_max - vy\_min)/(wy\_max - wy\_min);

    glClear(GL\_COLOR\_BUFFER\_BIT);

    glLoadIdentity();

    //DDA Line

    glColor3f(1.0f, 0.0f, 0.0f);

    glBegin(GL\_LINE\_LOOP);

    glVertex2f(0.f, 0.f);

    glVertex2f(300.f, 0.f);

    glVertex2f(300.f, 200.f);

    glVertex2f(0.f, 200.f);

    glEnd();

    glColor3f(0.0f, 1.0f, 0.0f);

    glBegin(GL\_LINE\_LOOP);

    glVertex2f(350.f, 0.f);

    glVertex2f(500.f, 0.f);

    glVertex2f(500.f, 400.f);

    glVertex2f(350.f, 400.f);

    glEnd();

    vector<Point> Shape(4);

    Shape[0].x = 10; Shape[0].y = 10;

    Shape[1].x = 120; Shape[1].y = 10;

    Shape[2].x = 180; Shape[2].y = 110;

    Shape[3].x = 10; Shape[3].y = 70;

    glColor3f(1.0f, 1.0f, 0.0f);

    glBegin(GL\_QUADS);

    for (Point p: Shape)

        glVertex2f(p.x, p.y);

    glEnd();

    vector<Point> Mod\_Shape;

    for (Point p: Shape)

    {

        Point newp;

        newp.x = vx\_min + (p.x - wx\_min) \* sx;

        newp.y = vy\_min + (p.y - wy\_min) \* sy;

        Mod\_Shape.push\_back(newp);

    }

    glColor3f(1.0f, 1.0f, 0.0f);

    glBegin(GL\_QUADS);

    for (Point p: Mod\_Shape)

        glVertex2f(p.x, p.y);

    glEnd();

    glFlush();

}

int main(int argc, char \*\*argv)

{

    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

    glutInitWindowSize(600, 600);

    glutInitWindowPosition(0, 0);

    glutCreateWindow("Window Viewport Transformation");

    myInit();

    glutDisplayFunc(myDisplay);

    glutMainLoop();

    return 1;

}

**OUTPUT:**

****