UCS 1712 – GRAPHICS AND MULTIMEDIA LAB

ASSIGNMENT – 6

VISHAL N

185001198

CSEC

26.09.2021

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();
```

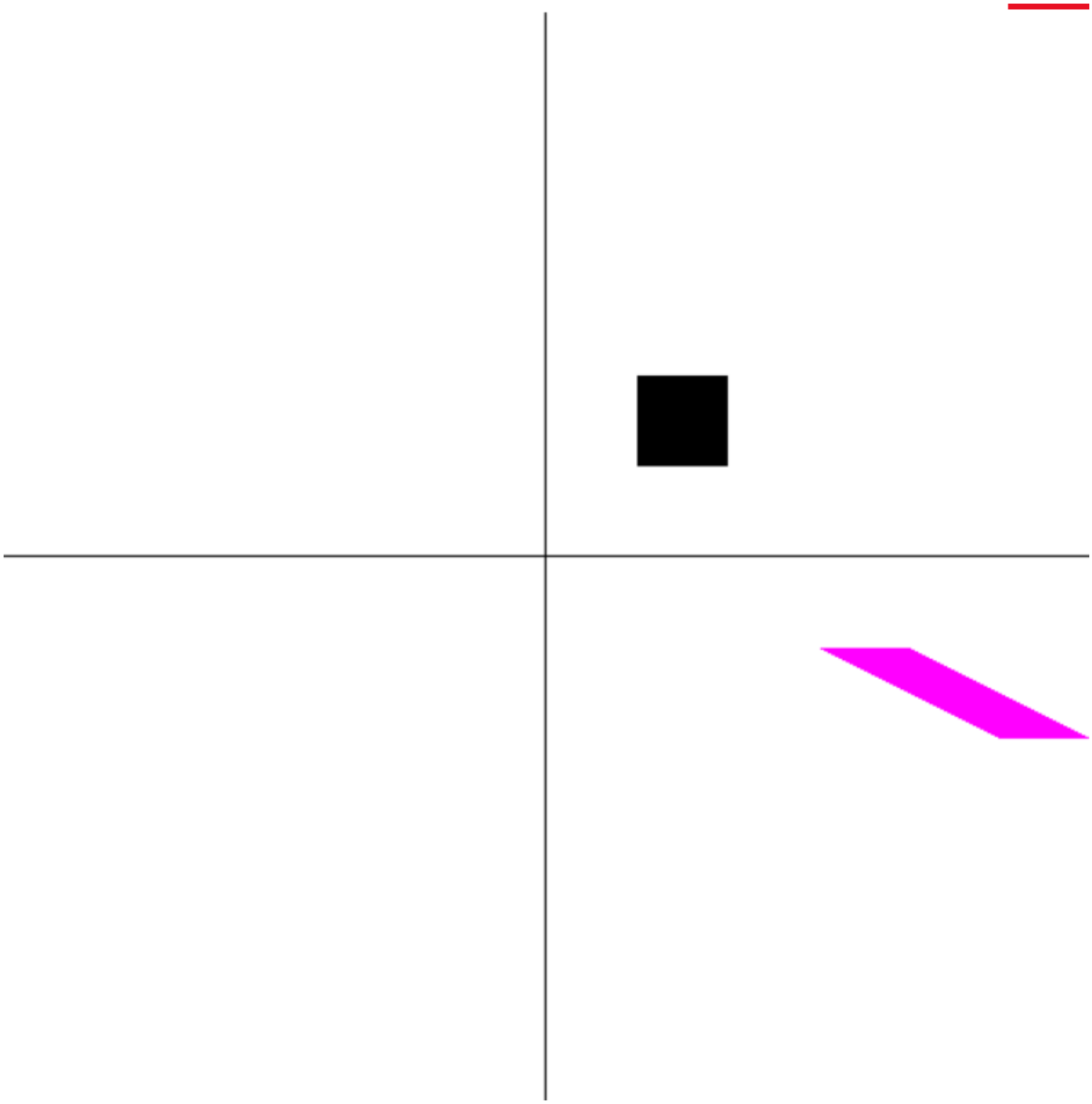
```
    glBegin(GL_LINES);
    glVertex2d(0, -600);
    glVertex2d(0, 600);
    glEnd();
}

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(rotat
ion_angle)}, {sin(rotation_angle), cos(rotation_angle), y_fixed * (1 - cos(rot
ation_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_ma
x = 200;
    double vx_org = 350, vy_org = 0, vx_min = 350, vy_min = 0, vx_max = 500, v
y_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: