Assignment no. 06

/\* Name:- Bhalsing Harshad Balaso

CRN no:- 24113070

Roll no:- 4070

Division :-SE IT

Batch :- S4 \*/

**Program :**

#include <GL/glut.h>

#include <iostream>

#include <cmath>

#define w 640

#define h 480

using namespace std;

void drawAxis() {

    glColor3f(0.0, 0.0, 0.0);

    glBegin(GL\_LINES);

    glVertex2i(0, -h);

    glVertex2i(0, h);

    glVertex2i(-w, 0);

    glVertex2i(w, 0);

    glEnd();

    glFlush();

}

class matrix {

public:

    int rows=3, cols;

    float mat[3][10];

    int vertexCount=0;

    void matmul(float temp[][3], float rslt[][10]) {

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

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

                rslt[i][j] = 0;

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

                    rslt[i][j] += temp[i][k] \* mat[k][j];

                }

            }

        }

    }

    void matmul3(float mat1[3][3], float mat2[3][3], float mat3[3][3], float rslt[][3]) {

        float temp[3][3] = {0};

        for (int i = 0; i < 3; i++) {

            for (int j = 0; j < 3; j++) {

                temp[i][j] = 0;

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

                    temp[i][j] += mat2[i][k] \* mat3[k][j];

                }

            }

        }

        for (int i = 0; i < 3; i++) {

            for (int j = 0; j < 3; j++) {

                rslt[i][j] = 0;

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

                    rslt[i][j] += mat1[i][k] \* temp[k][j];

                }

            }

        }

    }

    void plotMatrix(float temp[][10]) {

        glClear(GL\_COLOR\_BUFFER\_BIT);

        drawAxis();

        glColor3f(0.0, 0.0, 0.0);  // Object color changed to black

        glBegin(GL\_LINE\_LOOP);

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

            glVertex2f(round(temp[0][i]), round(temp[1][i]));

        }

        glEnd();

        glBegin(GL\_LINE\_LOOP);

        for(int i=0;i<cols;i++){

            glVertex2f(round(mat[0][i]),round(mat[1][i]));

        }

        glEnd();

        glFlush();

    }

    void translate() {

        float translationmat[3][3] = {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}};

        float rslt[3][10] = {0};

        float tx, ty;

        cout << "Enter the x factor: ";

        cin >> tx;

        cout << "Enter the y factor: ";

        cin >> ty;

        translationmat[0][2] = tx;

        translationmat[1][2] = ty;

        matmul(translationmat, rslt);

        plotMatrix(rslt);

    }

    void scale() {

        float scalemat[3][3] = {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}};

        float rslt[3][10] = {0};

        float sx, sy;

        cout << "Enter the x factor: ";

        cin >> sx;

        cout << "Enter the y factor: ";

        cin >> sy;

        scalemat[0][0] = sx;

        scalemat[1][1] = sy;

        matmul(scalemat, rslt);

        plotMatrix(rslt);

    }

    void rotate() {

        float rotationmat[3][3] = {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}};

        float rslt[3][10] = {0};

        float angle, flag;

        cout << "Enter Rotation angle: ";

        cin >> angle;

        cout << "Enter 1 for anticlockwise or 0 for clockwise: ";

        cin >> flag;

        angle = angle \* M\_PI / 180.0;

        if (flag == 1) {

            rotationmat[0][0] = cos(angle);

            rotationmat[0][1] = -sin(angle);

            rotationmat[1][0] = sin(angle);

            rotationmat[1][1] = cos(angle);

        } else {

            rotationmat[0][0] = cos(angle);

            rotationmat[0][1] = sin(angle);

            rotationmat[1][0] = -sin(angle);

            rotationmat[1][1] = cos(angle);

        }

        matmul(rotationmat, rslt);

        plotMatrix(rslt);

    }

    void shear() {

        float shearmat[3][3] = {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}};

        float rslt[3][10] = {0};

        float sh;

        int choice;

        cout << "Enter 1 for X-Shear, 2 for Y-Shear: ";

        cin >> choice;

        cout << "Enter Shearing factor: ";

        cin >> sh;

        if (choice == 1) shearmat[0][1] = sh;

        else shearmat[1][0] = sh;

        matmul(shearmat, rslt);

        plotMatrix(rslt);

    }

    void reflect() {

        float reflectmat[3][3] = {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}};

        float rslt[3][10] = {0};

        int choice;

        cout << "Enter 1 for Reflection about X-Axis, 2 for Y-Axis: ";

        cin >> choice;

        if (choice == 1) reflectmat[1][1] = -1;

        else reflectmat[0][0] = -1;

        matmul(reflectmat, rslt);

        plotMatrix(rslt);

    }

    void rotateAboutArbitraryPoint() {

        float translationmat1[3][3] = {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}};

        float translationmat2[3][3] = {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}};

        float rotationmat[3][3] = {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}};

        float transformationmat[3][3] = {0};

        float rslt[3][10] = {0};

        float angle, flag;

        cout << "Enter Rotation angle: ";

        cin >> angle;

        cout << "Enter 1 for anticlockwise or 0 for clockwise: ";

        cin >> flag;

        angle = angle \* M\_PI / 180.0;

        rotationmat[0][0] = cos(angle);

        rotationmat[1][1] = cos(angle);

        if (flag == 1) {

            rotationmat[0][1] = -sin(angle);

            rotationmat[1][0] = sin(angle);

        } else {

            rotationmat[0][1] = sin(angle);

            rotationmat[1][0] = -sin(angle);

        }

        float x, y;

        cout << "Enter the x coordinate of arbitrary point: ";

        cin >> x;

        cout << "Enter the y coordinate of arbitrary point: ";

        cin >> y;

        translationmat1[0][2] = -x;

        translationmat1[1][2] = -y;

        translationmat2[0][2] = x;

        translationmat2[1][2] = y;

        matmul3(translationmat2, rotationmat, translationmat1, transformationmat);

        matmul(transformationmat, rslt);

        plotMatrix(rslt);

    }

    void scaleAboutArbitraryPoint() {

        float translationmat1[3][3] = {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}};

        float translationmat2[3][3] = {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}};

        float scalemat[3][3] = {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}};

        float transformationmat[3][3] = {0};

        float rslt[3][10] = {0};

        float sx, sy, x, y;

        cout << "Enter scale factor sx and sy: ";

        cin >> sx >> sy;

        cout << "Enter the x and y coordinates of arbitrary point: ";

        cin >> x >> y;

        scalemat[0][0] = sx;

        scalemat[1][1] = sy;

        translationmat1[0][2] = -x;

        translationmat1[1][2] = -y;

        translationmat2[0][2] = x;

        translationmat2[1][2] = y;

        matmul3(translationmat2, scalemat, translationmat1, transformationmat);

        matmul(transformationmat, rslt);

        plotMatrix(rslt);

    }

};

matrix obj;

void mouseClick(int button, int state, int x, int y) {

    if (state == GLUT\_DOWN && button == GLUT\_LEFT\_BUTTON && obj.vertexCount > 0) {

        float X = (x - w/2.0f) \* 2.0f;

        float Y = (h/2.0f - y) \* 2.0f;

        obj.mat[0][obj.cols] = X;

        obj.mat[1][obj.cols] = Y;

        obj.mat[2][obj.cols] = 1;

        obj.vertexCount--;

        obj.cols++;

        if (obj.vertexCount == 0) {

            obj.plotMatrix(obj.mat);

        }

    }

}

void menu(int index) {

    if (index == 1) {

        cout << "Enter no. of vertex: ";

        cin >> obj.vertexCount;

        obj.cols = 0;

    }

    else if (index == 2) obj.translate();

    else if (index == 3) obj.scale();

    else if (index == 4) obj.rotate();

    else if (index == 5) obj.shear();

    else if (index == 6) obj.reflect();

    else if (index == 7) obj.rotateAboutArbitraryPoint();

    else if (index == 8) obj.scaleAboutArbitraryPoint();  // New Option

    else if (index == 9) exit(0);

    glFlush();

}

void display() {

    glClear(GL\_COLOR\_BUFFER\_BIT);

    drawAxis();

    glFlush();

}

void init() {

    glClearColor(1.0, 1.0, 1.0, 1.0); // Background changed to white

    glMatrixMode(GL\_PROJECTION);

    glLoadIdentity();

    gluOrtho2D(-w, w, -h, h);

}

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

    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

    glutInitWindowSize(w, h);

    glutCreateWindow("Transformation");

    glutDisplayFunc(display);

    glutMouseFunc(mouseClick);

    // Create the menu

    glutCreateMenu(menu);

    glutAddMenuEntry("DrawShape", 1);

    glutAddMenuEntry("Translate", 2);

    glutAddMenuEntry("Scale", 3);

    glutAddMenuEntry("Rotate", 4);

    glutAddMenuEntry("Shear", 5);

    glutAddMenuEntry("Reflection", 6);

    glutAddMenuEntry("Rotation about arbitrary", 7);

    glutAddMenuEntry("Scaling about arbitrary", 8);

    glutAddMenuEntry("Exit", 9);

    // Attach menu to right mouse button

    glutAttachMenu(GLUT\_RIGHT\_BUTTON);

    init();             // Initialize OpenGL settings

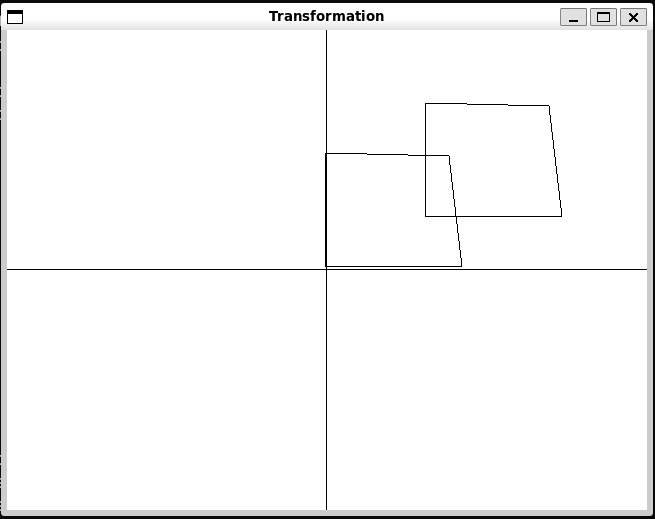
    glutMainLoop();     // Enter the event-processing loop

    return 0;

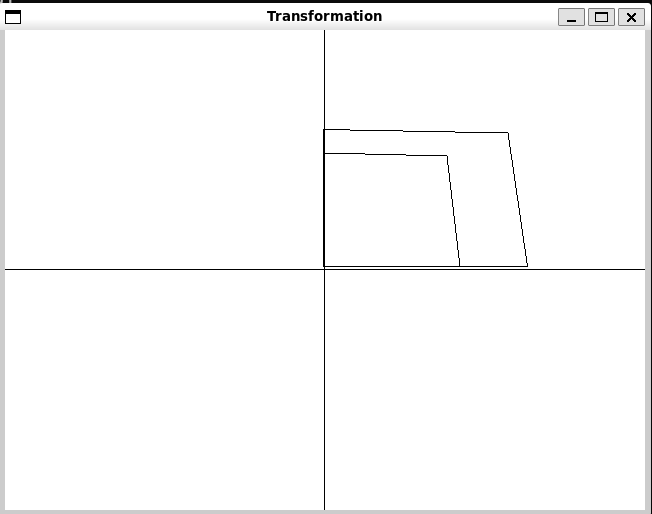
}

**Output :**

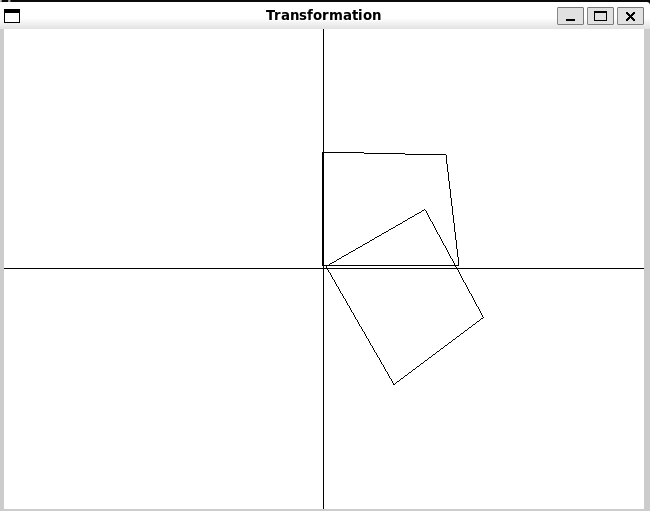
1. Translation :



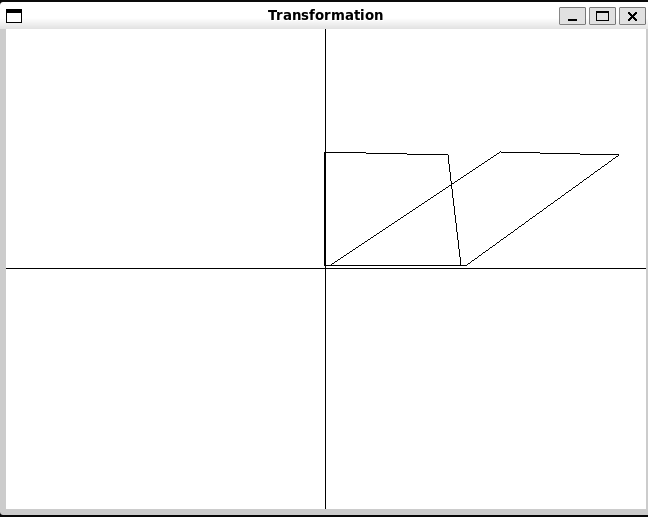
1. Scaling about Origin :



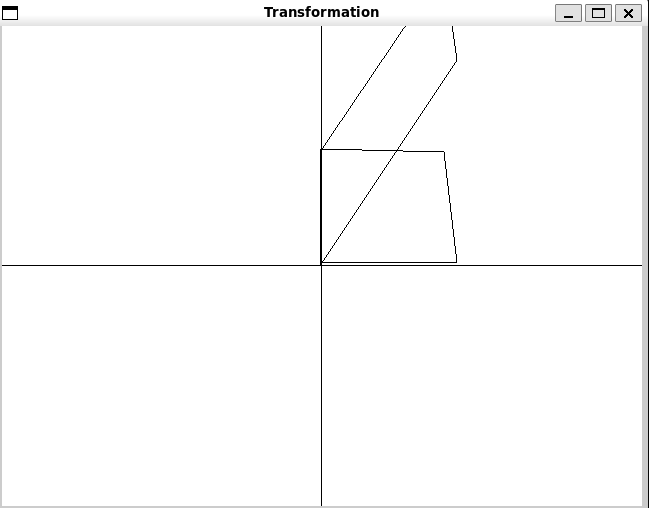
1. Rotation :



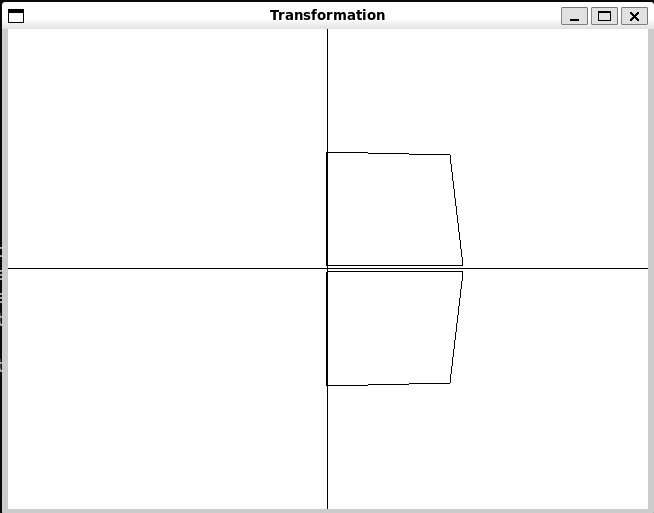
1. Shearing (x-shear) :



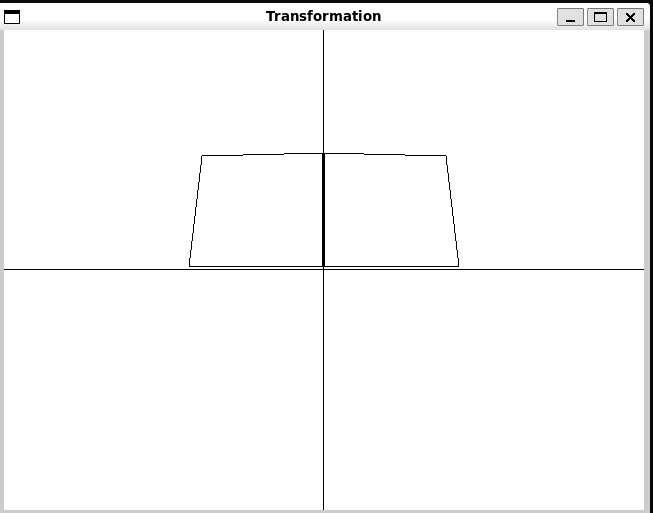
1. Shearing(Y- Shear) :



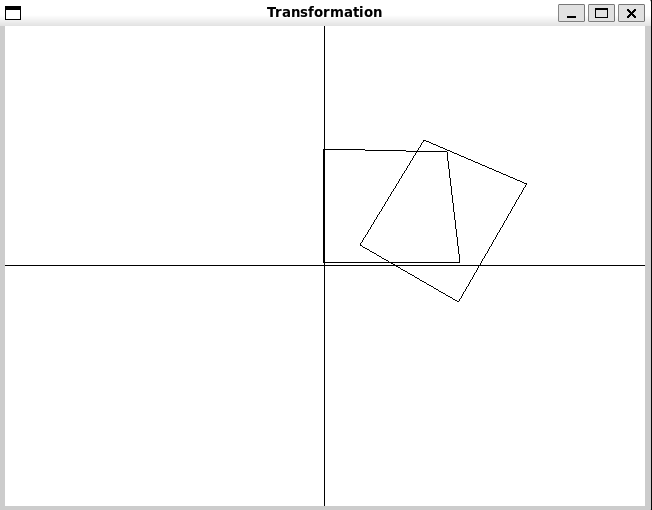
1. Reflection about x axis :



1. Reflection about y axis :



1. Rotation about arbitrary point :



1. Scaling about arbitrary point:

