Date: 14/10/2025
Experiment No: 06
Experiment Name:
Apply the following 2D transformation technique to a triangle

i) Translation (tx = 2, ty = 3)
ii) Scaling (Sx = 2, Sy = 2 and Sx = 0.5, Sy = 0.5)
iii) Rotation (θ = 45°)
iv) Reflection about x and y-axis
v) Shearing about x and y-axis(Shr = 2)

Introduction

In computer graphics, 2D transformations are used to change the position, size, orientation, or shape of objects in a two-dimensional plane. This lab focuses on applying different transformation techniques to a triangle, including translation, scaling, rotation, reflection, and shearing. By performing these transformations, we can understand how the coordinates of points in a shape are modified and how these operations help in manipulating graphical objects for various applications.

Description:

In this experiment, a triangle is subjected to different 2D transformation techniques to observe how its position, size, orientation, and shape change. The transformations applied are:

1. Translation – Moving the triangle by a certain distance along the x and y axes.
2. Scaling – Enlarging or shrinking the triangle along the x and y axes.
3. Rotation – Rotating the triangle around the origin by a specified angle.
4. Reflection – Flipping the triangle about the x-axis and y-axis.
5. Shearing – Slanting the shape of the triangle along the x or y axis.

These operations help visualize the effect of coordinate changes and are fundamental in computer graphics for object manipulation.events.

Code:

```
#include <GL/glut.h>
#include <math.h>
#include <stdio.h>

float triangle[3][2] = {
   {100.0, 100.0},
   {200.0, 100.0},
   {150.0, 200.0}
};
```

```
float transformedTriangle[3][2];

float tx = 2.0, ty = 3.0;
float sx1 = 2.0, sy1 = 2.0;
float sx2 = 0.5, sy2 = 0.5;
float angle = 45.0;
float shr = 2.0;

void init() {
   glClearColor(1.0, 1.0, 1.0, 1.0);
   glMatrixMode(GL_PROJECTION);
   gluOrtho2D(-300, 300, -300, 300);

   for (int i = 0; i < 3; i++) {
      for (int j = 0; j < 2; j++) {
         transformedTriangle[i][j] = triangle[i][j];
      }
   }
}

void drawTriangle() {
   glBegin(GL_LINES);
   glVertex2f(transformedTriangle[0][0], transformedTriangle[0][1]);
   glVertex2f(transformedTriangle[1][0], transformedTriangle[1][1]);

   glVertex2f(transformedTriangle[1][0], transformedTriangle[1][1]);
   glVertex2f(transformedTriangle[2][0], transformedTriangle[2][1]);

   glVertex2f(transformedTriangle[2][0], transformedTriangle[2][1]);
   glVertex2f(transformedTriangle[0][0], transformedTriangle[0][1]);
   glEnd();
}

void display() {
   glClear(GL_COLOR_BUFFER_BIT);

   // Draw axis
   glBegin(GL_LINES);
   glColor3f(0, 0, 0);
   glVertex2f(300, 0);
   glVertex2f(-300, 0);
   glVertex2f(0, 300);
   glVertex2f(0, -300);
   glEnd();

   glColor3f(0.0, 1.0, 0.0);
```

```
    glBegin(GL_LINES);
    glVertex2f(triangle[0][0], triangle[0][1]);
    glVertex2f(triangle[1][0], triangle[1][1]);

    glVertex2f(triangle[1][0], triangle[1][1]);
    glVertex2f(triangle[2][0], triangle[2][1]);

    glVertex2f(triangle[2][0], triangle[2][1]);
    glVertex2f(triangle[0][0], triangle[0][1]);
    glEnd();

    glColor3f(1.0, 0.0, 0.0);
    drawTriangle();

    glutSwapBuffers();
}

void applyTranslation() {
    for (int i = 0; i < 3; i++) {
        transformedTriangle[i][0] = triangle[i][0] + tx;
        transformedTriangle[i][1] = triangle[i][1] + ty;
    }
}

void applyScaling(float sx, float sy) {
    float centerX = (triangle[0][0] + triangle[1][0] + triangle[2][0]) / 3.0;
    float centerY = (triangle[0][1] + triangle[1][1] + triangle[2][1]) / 3.0;

    for (int i = 0; i < 3; i++) {
        transformedTriangle[i][0] = centerX + (triangle[i][0] - centerX) * sx;
        transformedTriangle[i][1] = centerY + (triangle[i][1] - centerY) * sy;
    }
}

void applyRotation() {
    float radians = angle * M_PI / 180.0;

    float centerX = (triangle[0][0] + triangle[1][0] + triangle[2][0]) / 3.0;
    float centerY = (triangle[0][1] + triangle[1][1] + triangle[2][1]) / 3.0;

    for (int i = 0; i < 3; i++) {
        float x = triangle[i][0] - centerX;
        float y = triangle[i][1] - centerY;

        transformedTriangle[i][0] = centerX + (x * cos(radians) - y * sin(radians));
        transformedTriangle[i][1] = centerY + (x * sin(radians) + y * cos(radians));
```

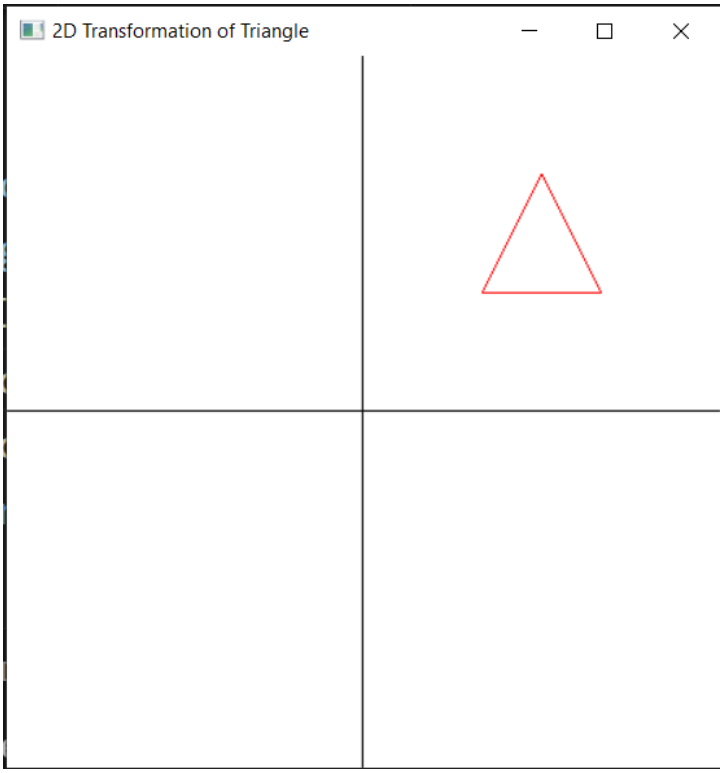
```
    }
}

void applyReflectionX() {
    for (int i = 0; i < 3; i++) {
        transformedTriangle[i][0] = triangle[i][0];
        transformedTriangle[i][1] = -triangle[i][1];
    }
}

void applyReflectionY() {
    for (int i = 0; i < 3; i++) {
        transformedTriangle[i][0] = -triangle[i][0];
        transformedTriangle[i][1] = triangle[i][1];
    }
}

void applyShearingX() {
    float centerX = (triangle[0][0] + triangle[1][0] + triangle[2][0]) / 3.0;
    float centerY = (triangle[0][1] + triangle[1][1] + triangle[2][1]) / 3.0;

    for (int i = 0; i < 3; i++) {
        float x = triangle[i][0] - centerX;
        float y = triangle[i][1] - centerY;

        float newX = x + shr * y;
        float newY = y;

        transformedTriangle[i][0] = newX + centerX;
        transformedTriangle[i][1] = newY + centerY;
    }
}

void applyShearingY() {
    float centerX = (triangle[0][0] + triangle[1][0] + triangle[2][0]) / 3.0;
    float centerY = (triangle[0][1] + triangle[1][1] + triangle[2][1]) / 3.0;

    for (int i = 0; i < 3; i++) {
        float x = triangle[i][0] - centerX;
        float y = triangle[i][1] - centerY;

        float newX = x;
        float newY = y + shr * x;

        transformedTriangle[i][0] = newX + centerX;
        transformedTriangle[i][1] = newY + centerY;
```

```
    }
}

void keyboard(unsigned char key, int x, int y) {
    switch (key) {
    case '1':
        printf("Applying Translation\n");
        applyTranslation();
        break;
    case '2':
        printf("Applying Scaling\n");
        applyScaling(sx1, sy1);
        break;
    case '3':
        printf("Applying Scaling\n");
        applyScaling(sx2, sy2);
        break;
    case '4':
        printf("Applying Rotation\n");
        applyRotation();
        break;
    case '5':
        printf("Applying Reflection about X-axis\n");
        applyReflectionX();
        break;
    case '6':
        printf("Applying Reflection about Y-axis\n");
        applyReflectionY();
        break;
    case '7':
        printf("Applying Shearin in X axis\n");
        applyShearingX();
        break;
    case '8':
        printf("Applying Shearin in Y axis\n");
        applyShearingY();
        break;
    case 'R':
        printf("Resetting to original triangle\n");
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 2; j++) {
                transformedTriangle[i][j] = triangle[i][j];
            }
        }
        break;
    case 27:
```

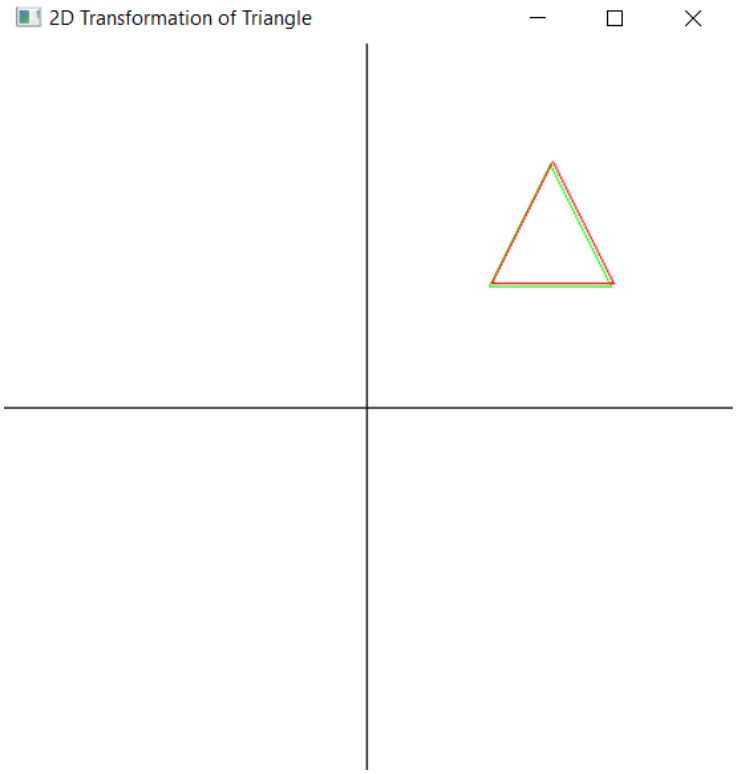
```
        exit(0);
        break;
    }

    glutPostRedisplay();
}


int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(640, 640);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("2D Transformation of Triangle");

    init();

    glutDisplayFunc(display);
    glutKeyboardFunc(keyboard);

    glutMainLoop();

    return 0;
}
```

Output:

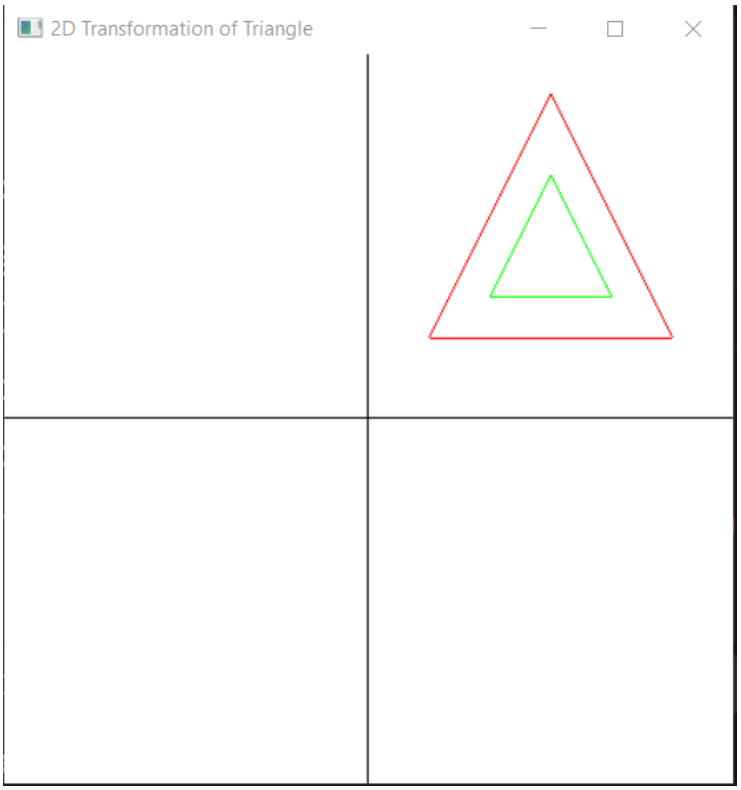
1. Initial state



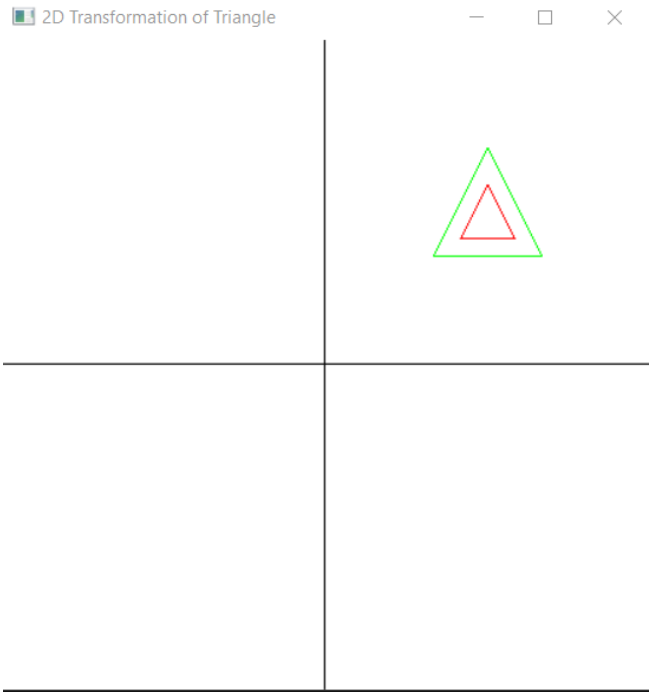
2. Applying Translation



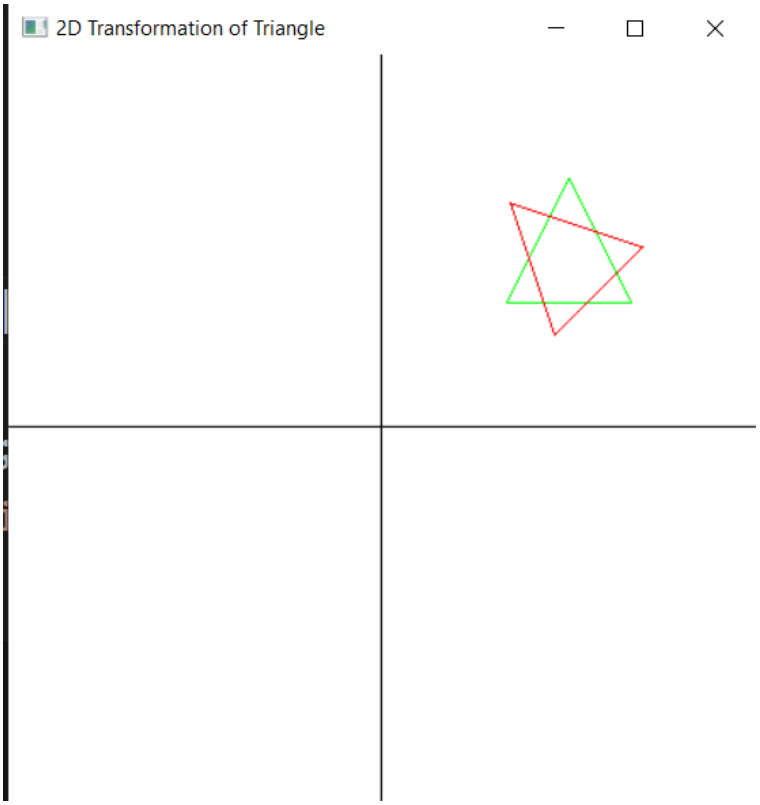
3. Applying Scaling (Large)



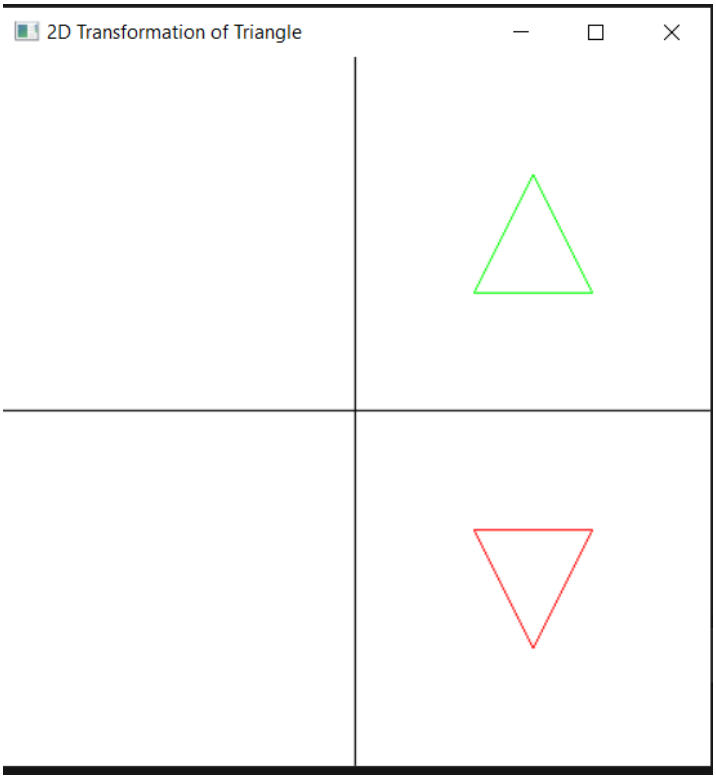
4. Applying Scaling (Make it small)



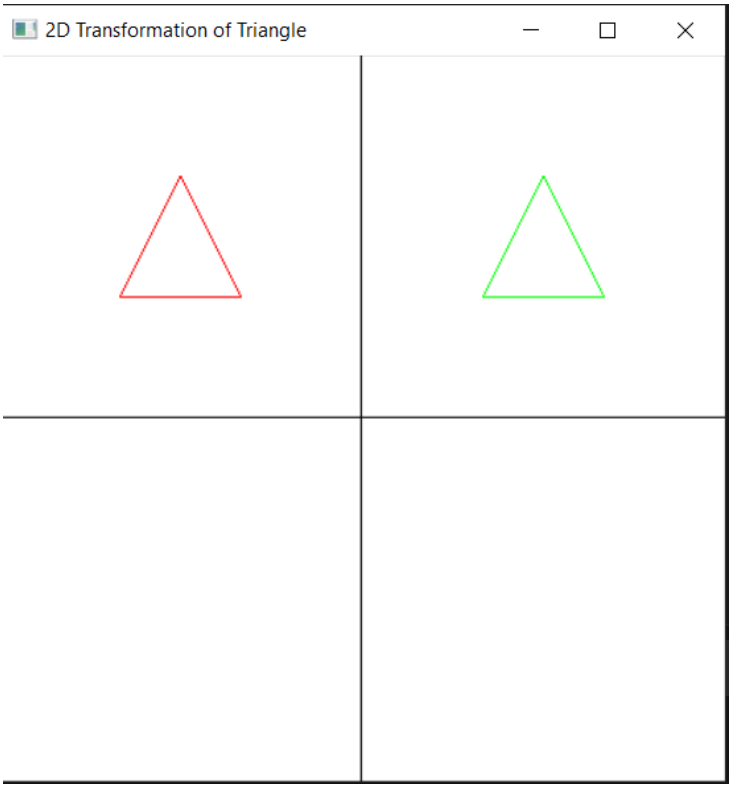
5. Applying Rotation



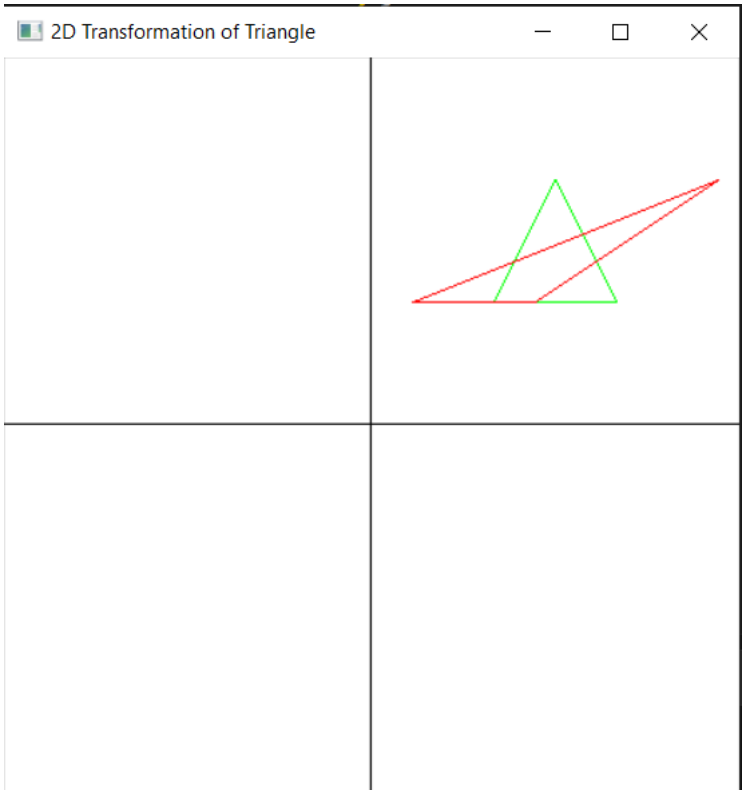
6. Applying Reflection about X-axis



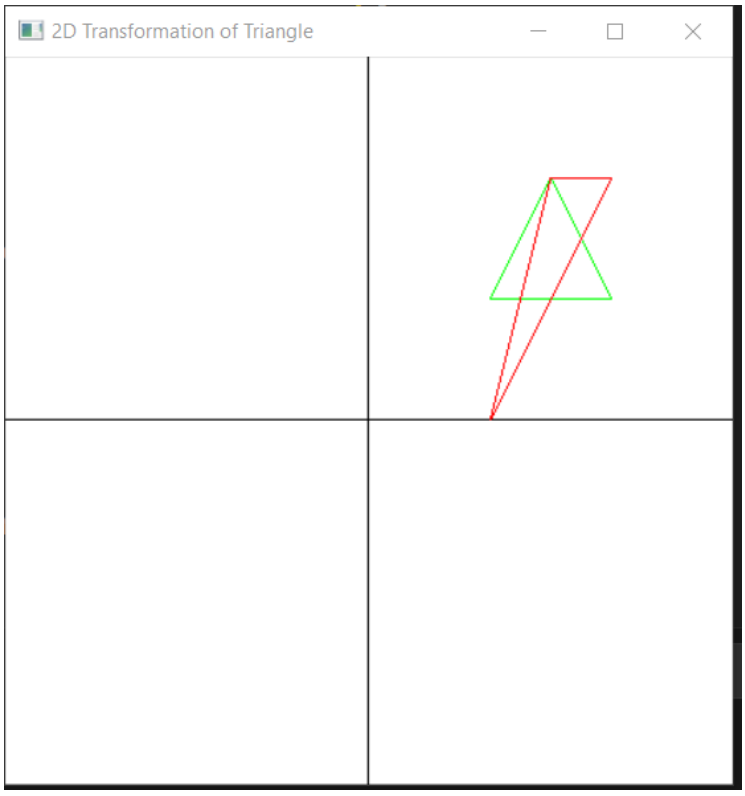
7. Applying Reflection about Y-axis



8. Applying Shearing in X axis



9. Applying Shearing in Y axis



Conclusion:

In this lab, we applied various 2D transformation techniques to a triangle, including translation, scaling, rotation, reflection, and shearing. The experiment helped us understand how the coordinates of a shape change under each transformation and how these operations are used to manipulate objects in computer graphics. Through this exercise, we gained practical knowledge of the effects of different transformations on graphical objects.