Experiment No:04

Experiment Name: Implementation of 2D Transformation in Computer Graphics.

(Rotation)

Description:

Rotation is a fundamental 2D transformation technique in computer graphics, used to rotate an object about a specific point in a two-dimensional plane. In this experiment, a 2D geometric shape (triangle or rectangle) is rotated by a specified angle, either around the origin or its own centroid. The process uses mathematical formulas involving sine and cosine to calculate new positions of each point after rotation.

Rotation is implemented using the following equations:

- $x' = x * \cos(\theta) - y * \sin(\theta)$
- $y' = x * \sin(\theta) + y * \cos(\theta)$

Where (x, y) are the original coordinates, (x', y') are the rotated coordinates, and θ is the rotation angle in radians.

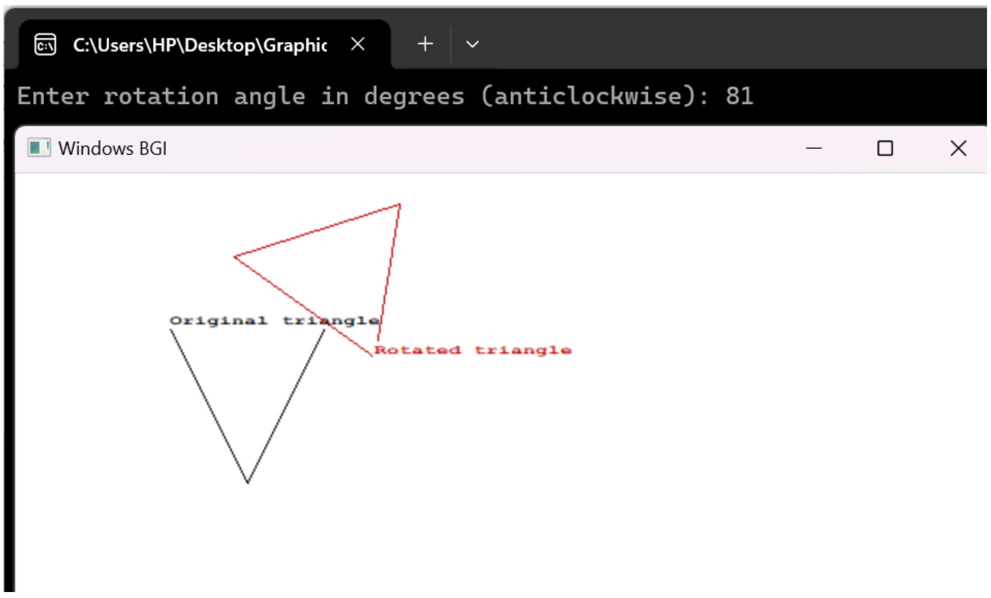
Algorithm for 2D Rotation:

```
1. Start the graphics mode using initgraph().
2. Define the coordinates of the shape (triangle or rectangle).
3. Calculate the centroid (if rotating around the center of the shape).
4. Ask the user to enter the rotation angle (in degrees).
5. Convert the angle from degrees to radians: angleRad = angleDeg × π / 180.
6. For each vertex of the shape:
    o Translate the point to the origin (if rotating around centroid).
    o Apply rotation using the formulas:
        ▪ $x' = x * \cos(\theta) - y * \sin(\theta)$
        ▪ $y' = x * \sin(\theta) + y * \cos(\theta)$
    o Translate back (if required).
7. Draw the original and rotated shapes with different colors.
8. Close the graphics window after a key is pressed.
```

Source Code:

```
#include <graphics.h>

#include <iostream>

#include <cmath>

using namespace std;

void rotatePoint(int x, int y, double angleRad, int &xr, int &yr) {
    xr = round(x * cos(angleRad) - y * sin(angleRad));
    yr = round(x * sin(angleRad) + y * cos(angleRad));
}int main() {
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "");
    setbkcolor(WHITE);
    cleardevice();
// Define triangle vertices (relative to origin)
    int x1 = 100, y1 = 100;  // Point A
    int x2 = 200, y2 = 100;  // Point B
    int x3 = 150, y3 = 200;  // Point C
double angleDeg;
    cout << "Enter rotation angle in degrees (anticlockwise): ";
    cin >> angleDeg;
double angleRad = angleDeg * M_PI / 180.0;
// Draw original triangle
    setcolor(BLACK);
    line(x1, y1, x2, y2);
    line(x2, y2, x3, y3);
    line(x3, y3, x1, y1);
    outtextxy(x1, y1 - 10, "Original triangle");
// Rotate each point
    int rx1, ry1, rx2, ry2, rx3, ry3;
    rotatePoint(x1, y1, angleRad, rx1, ry1);
    rotatePoint(x2, y2, angleRad, rx2, ry2);
    rotatePoint(x3, y3, angleRad, rx3, ry3);
// Shift origin to center of screen
    int cx = getmaxx() / 2;
    int cy = getmaxy() / 2;
// Draw rotated triangle
    setcolor(RED);
    line(cx + rx1, cy - ry1, cx + rx2, cy - ry2);
    line(cx + rx2, cy - ry2, cx + rx3, cy - ry3);
    line(cx + rx3, cy - ry3, cx + rx1, cy - ry1);
outtextxy(cx + rx1, cy - ry1 - 10, "Rotated triangle");
 getch();
   closegraph();
   return 0;}
```

Output:



```
Enter rotation angle in degrees (anticlockwise): 81
```

Figure name: Rotation of Triangle.

Conclusion:

In this experiment, we successfully implemented 2D rotation using C++ and the graphics.h library. By applying basic trigonometric formulas, we could visualize how each point of a shape is transformed under rotation. This helped in understanding how rotation matrices work and how transformations can be applied in real-time rendering. The experiment demonstrates the practical use of mathematical transformations in computer graphics applications such as animation, gaming, and simulations.