Experiment 4 – Scan Line Polygon Fill Algorithm

Objective:

- To fill a polygon using the **Scan Line Polygon Filling algorithm** in computer graphics.
- To understand how **pixel-by-pixel filling** works for arbitrary polygons.

Theory:

- The Scan Line Polygon Fill algorithm works by:
    1. Finding the edges of the polygon.
    2. Determining the intersection points of the polygon with each horizontal scan line.
    3. Sorting intersection points by x-coordinate.
    4. Filling pixels between pairs of intersection points.
- This algorithm is widely used because it efficiently fills convex and concave polygons.

Requirements / Equipment:

- OpenGL, GLUT
- Code::Blocks
- C++

Procedure / Description:

1. Define polygon vertices.
2. Find min and max y of polygon.
3. For each scan line (y = ymin to ymax):
    - Find intersections with polygon edges.
    - Sort intersections.
    - Fill pixels between pairs of intersections.
4. Display the polygon on the screen.

Source Code:

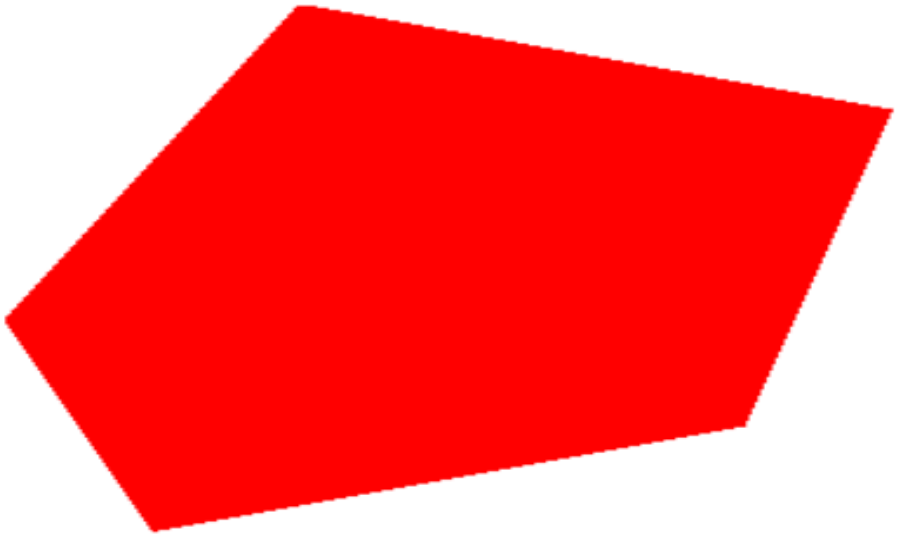
```
#include <windows.h>
#include <GL/gl.h>
#include <GL/glut.h>
#include <vector>
#include <algorithm>


struct Point {
    int x, y;
};
// Define a simple polygon (convex or concave)
std::vector<Point> polygon =
{{100,100},{300,150},{350,300},{150,350},{50,200}};
void drawPixel(int x, int y)
{
    glBegin(GL_POINTS);
    glVertex2i(x, y);
    glEnd();
}


/* Scan Line Polygon Fill Algorithm */
void scanLineFill(std::vector<Point> poly) {
```

```
int ymin = poly[0].y, ymax = poly[0].y;

for(auto p : poly) {
    if(p.y < ymin) ymin = p.y;
    if(p.y > ymax) ymax = p.y;
}

for(int y = ymin; y <= ymax; y++) {
    std::vector<int> intersections;

    int n = poly.size();
    for(int i=0; i<n; i++) {
        Point p1 = poly[i];
        Point p2 = poly[(i+1)%n];

        if((y >= std::min(p1.y,p2.y)) && (y < std::max(p1.y,p2.y))) {
            int x = p1.x + (float)(y - p1.y) * (p2.x - p1.x) / (p2.y - p1.y);
            intersections.push_back(x);
        }
    }
```

```
        std::sort(intersections.begin(), intersections.end());

        for(size_t i=0; i+1<intersections.size(); i+=2) {
            for(int x=intersections[i]; x<=intersections[i+1]; x++)
                drawPixel(x, y);
        }
    }
}

void display() {
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1, 0, 0); // Red polygon fill
    scanLineFill(polygon);
    glFlush();
}
void init() {
    glClearColor(1,1,1,1);
    glPointSize(2);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0,500,0,500);
```

```
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500,500);
    glutCreateWindow("Scan Line Polygon Fill Algorithm");
    init();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

```
// Define a simple polygon (convex or concave)
std::vector<Point> polygon = {{100,100},{300,150},{350,300},{150,350},{50,200}};
```



Conclusion / Discussion:

- Scan Line Fill efficiently fills polygons line by line.
- Works for both **convex and concave polygons**.
- Demonstrates how **raster graphics** fills shapes in 2D space.