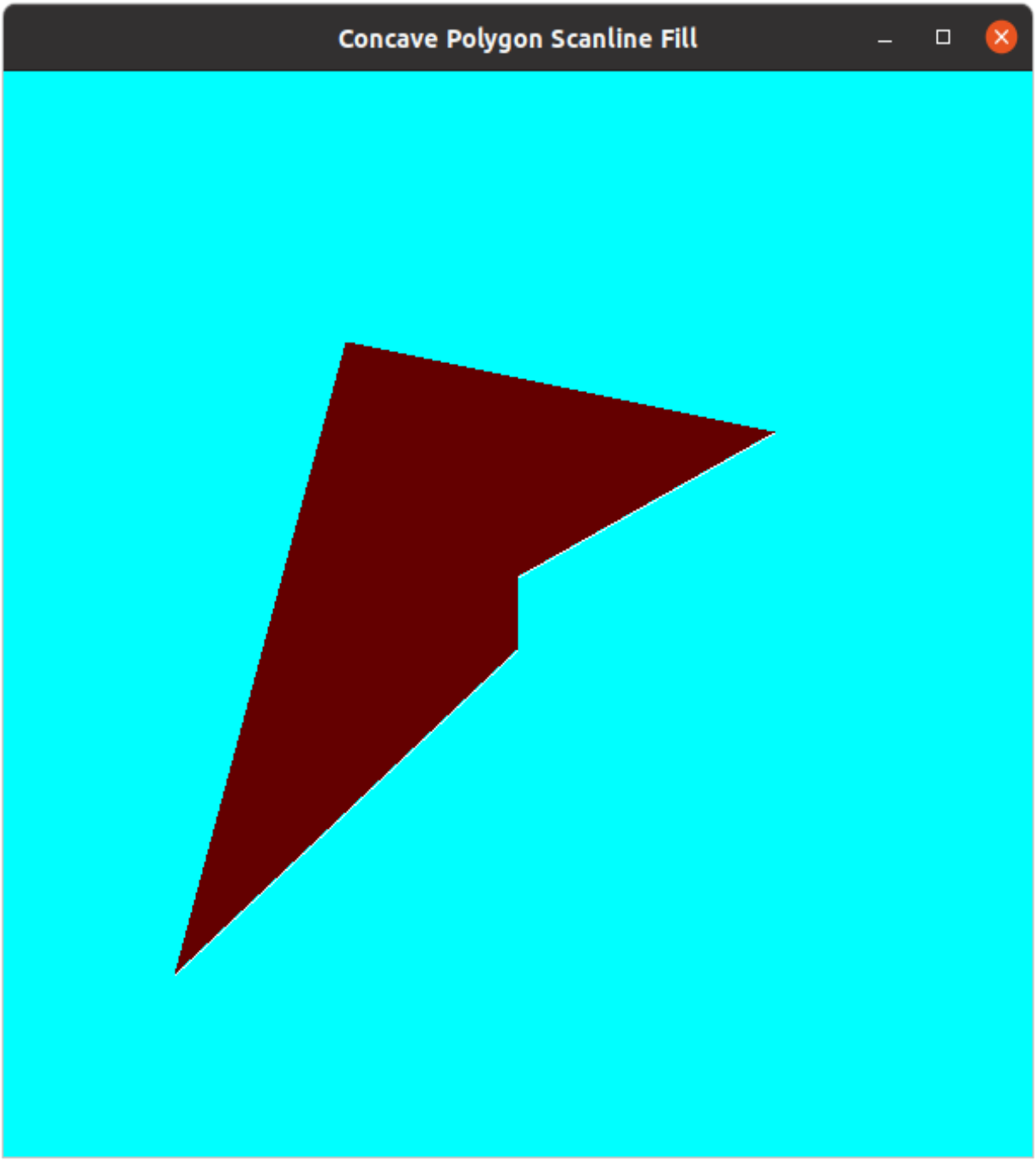
Computer graphics Practical 1

code:

```
#include <GL/glut.h>
#include <iostream>
#include <vector>
#include <climits>
#include <algorithm>

// Class to represent a polygon
class Polygon {
public:
    Polygon(std::vector<std::pair<int, int>> vertices) : vertices(vertices) {}

    void drawPolygon() {
        glBegin(GL_LINE_LOOP);
        for (const auto& vertex : vertices) {
            glVertex2i(vertex.first, vertex.second);
        }
        glEnd();
    }

    std::vector<std::pair<int, int>> getVertices() const {
        return vertices;
    }

private:
    std::vector<std::pair<int, int>> vertices;
};

// Class to fill a polygon using scanline fill algorithm
class ScanlineFill {
public:
    ScanlineFill(Polygon polygon, int fillR, int fillG, int fillB)
        : polygon(polygon), fillR(fillR), fillG(fillG), fillB(fillB) {}

    void fillPolygon() {
        std::vector<std::pair<int, int>> vertices = polygon.getVertices();
        int minY = INT_MAX, maxY = INT_MIN;

        // Find the minimum and maximum Y coordinates
        for (const auto& vertex : vertices) {
            int y = vertex.second;
            minY = std::min(minY, y);
            maxY = std::max(maxY, y);
```

```
        }

        // Scanline filling loop
        for (int y = minY; y <= maxY; y++) {
            std::vector<int> intersections;

            // Find intersections with polygon edges
            for (size_t i = 0; i < vertices.size(); i++) {
                int x1 = vertices[i].first;
                int y1 = vertices[i].second;
                int x2 = vertices[(i + 1) % vertices.size()].first;
                int y2 = vertices[(i + 1) % vertices.size()].second;

                if ((y1 <= y && y2 > y) || (y1 > y && y2 <= y)) {
                    // Calculate intersection point
                    int x = static_cast<int>((x1 + (static_cast<double>(y - y1) / (y2 - y1)) * (x2 -
x1)));
                    intersections.push_back(x);
                }
            }

            // Sort intersection points
            std::sort(intersections.begin(), intersections.end());

            // Fill between pairs of intersection points
            for (size_t i = 0; i < intersections.size(); i += 2) {
                int x1 = intersections[i];
                int x2 = intersections[i + 1];

                // Fill the scanline segment with the desired color
                glColor3ub(fillR, fillG, fillB);
                glBegin(GL_LINES);
                glVertex2i(x1, y);
                glVertex2i(x2, y);
                glEnd();
            }
        }
    }

private:
    Polygon polygon;
    int fillR, fillG, fillB;
};

void display() {
    glClear(GL_COLOR_BUFFER_BIT);

    // Define the vertices of the polygon
    std::vector<std::pair<int, int>> polygonVertices = {
        {100, 100},
```

```
        {300, 280},
        {300 ,320},
        {450, 400},
        {200, 450}
    };

    // Create a Polygon object and draw it
    Polygon polygon(polygonVertices);
    polygon.drawPolygon();

    // Create a ScanlineFill object and fill the polygon with a color (e.g., red)
    ScanlineFill scanFill(polygon, 100, 0,0);
    scanFill.fillPolygon();

    glFlush();
}

void init() {
    glClearColor(0.0, 2.0, 3.0, 0.0); // Set the background color to white
    gluOrtho2D(0, 600, 0, 600); // Set the coordinate system
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(600, 600);
    glutCreateWindow("Concave Polygon Scanline Fill");
    init();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```


Concave Polygon Scanline Fill