SCHOOL OF COMPUTER SCIENCE

UNIVERSITY OF PETROLEUM AND ENERGY STUDIES
DEHRADUN, UTTARAKHAND

UPES
UNIVERSITY OF TOMORROW

COMPUTER GRAPHICS
LABORATORY FILE
(2024-2025)

For

Vth Semester

Submitted To:
Mr. Dinesh
Assistant Professor
[Vth Semester]
School of Computer Science

Submitted By:
Akshat Negi
500106533(SAP ID)
R2142220414(Roll No.)
B.Tech. CSF (Batch-1)

LAB EXPERIMENT – 4

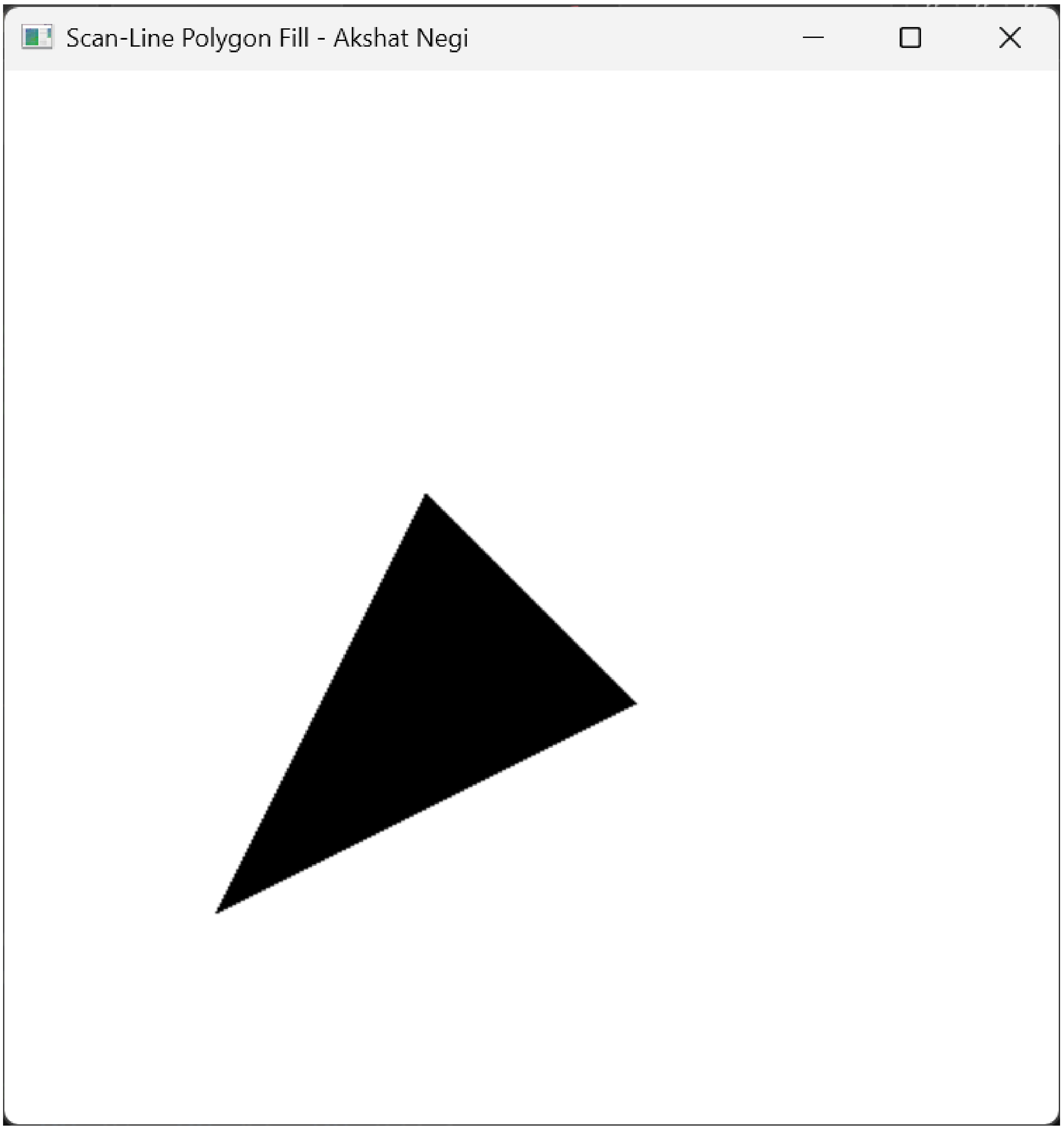
Seed Fill Algorithms

[Small Project will be given for demonstration]

# Take the value of seed point, intensity of new color as input from user.

a. WAP to fill the polygon using scan lines.

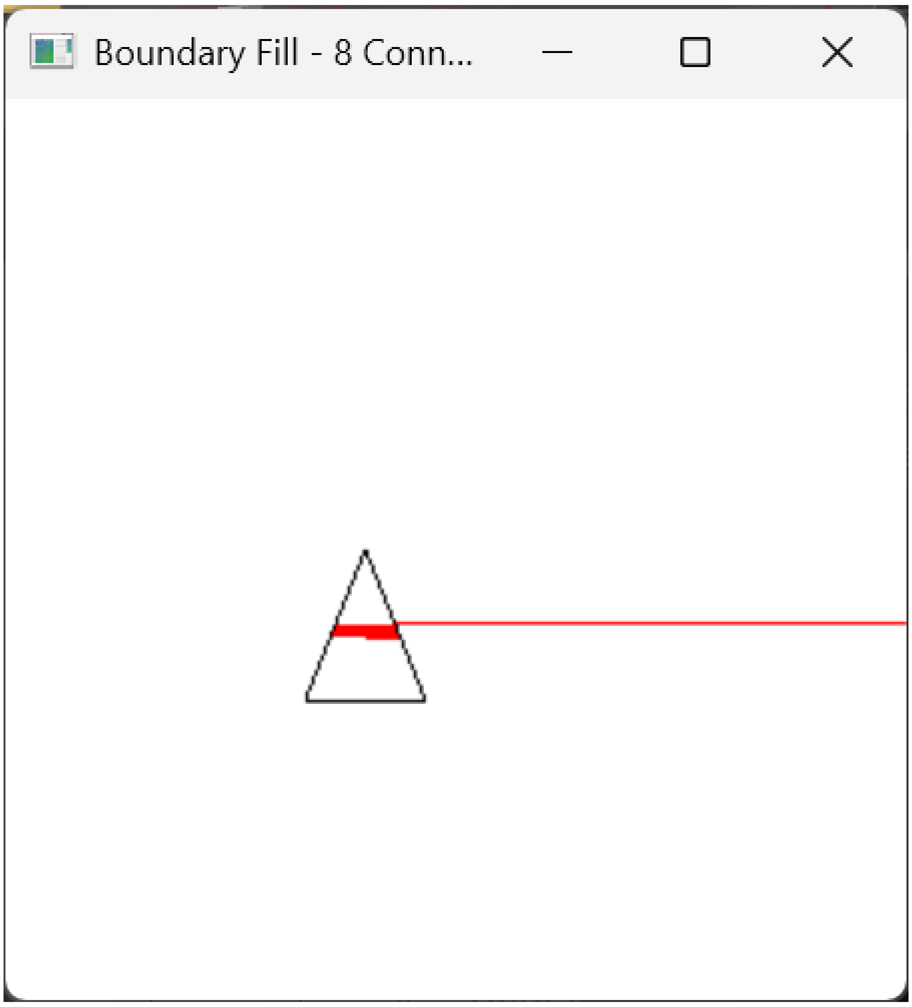
```
#include <GL/freeglut.h>
#include <iostream>
#include <vector>
#include <algorithm>

// Global Variables
std::vector<int> x_coords;
std::vector<int> y_coords;
int edges;

// Function to draw a line between two points
void drawLine(int x1, int y1, int x2, int y2) {
    glBegin(GL_LINES);
    glVertex2i(x1, y1);
    glVertex2i(x2, y2);
    glEnd();
    glFlush();
}

// Function to implement scan-line polygon filling
void scanFill()
{
    int i, j, temp;
    int xmin = *std::min_element(x_coords.begin(), x_coords.end());
    int xmax = *std::max_element(x_coords.begin(), x_coords.end());

    // Scan each scan-line within the polygon's vertical extent
    for (i = xmin; i <= xmax; i++) {
        // Initialize an array to store the intersection points
        std::vector<int> interPoints;

        for (j = 0; j < edges; j++) {
            int next = (j + 1) % edges;

            // Check if the current edge intersects with the scan line
            if ((y_coords[j] > i && y_coords[next] <= i) || (y_coords[next] > i
&& y_coords[j] <= i)) {
                int interX = x_coords[j] + (i - y_coords[j]) * (x_coords[next] -
x_coords[j]) / (y_coords[next] - y_coords[j]);
                interPoints.push_back(interX);
            }
        }

        // Sort the intersection points in ascending order
        std::sort(interPoints.begin(), interPoints.end());
        // Fill the pixels between pairs of intersection points
        for (j = 0; j < interPoints.size(); j += 2) {
            if (j + 1 < interPoints.size()) {
                drawLine(interPoints[j], i, interPoints[j + 1], i);
            }
        }
    }
}

// Display callback for OpenGL
void display() {
    glClear(GL_COLOR_BUFFER_BIT);
    scanFill();
    glFlush();
}
// Function to initialize OpenGL

void init() {
    // Set the background color to white and the drawing color to black
    glClearColor(1.0, 1.0, 1.0, 1.0);
    glColor3f(0.0, 0.0, 0.0);

    // Set up 2D orthographic projection with the window size
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(0.0, 500.0, 0.0, 500.0);  // Adjust window size as needed
}

int main(int argc, char** argv) {
    // Define the polygon vertices
    x_coords = { 100, 200, 300 };
    y_coords = { 100, 300, 200 };
    edges = 3;

    // Initialize GLUT
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);   // Window size
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Scan-Line Polygon Fill - Akshat Negi");

    init();  // Set up OpenGL

    // Register the display callback function
    glutDisplayFunc(display);

    // Enter the GLUT main loop
    glutMainLoop();

    return 0;
}
```



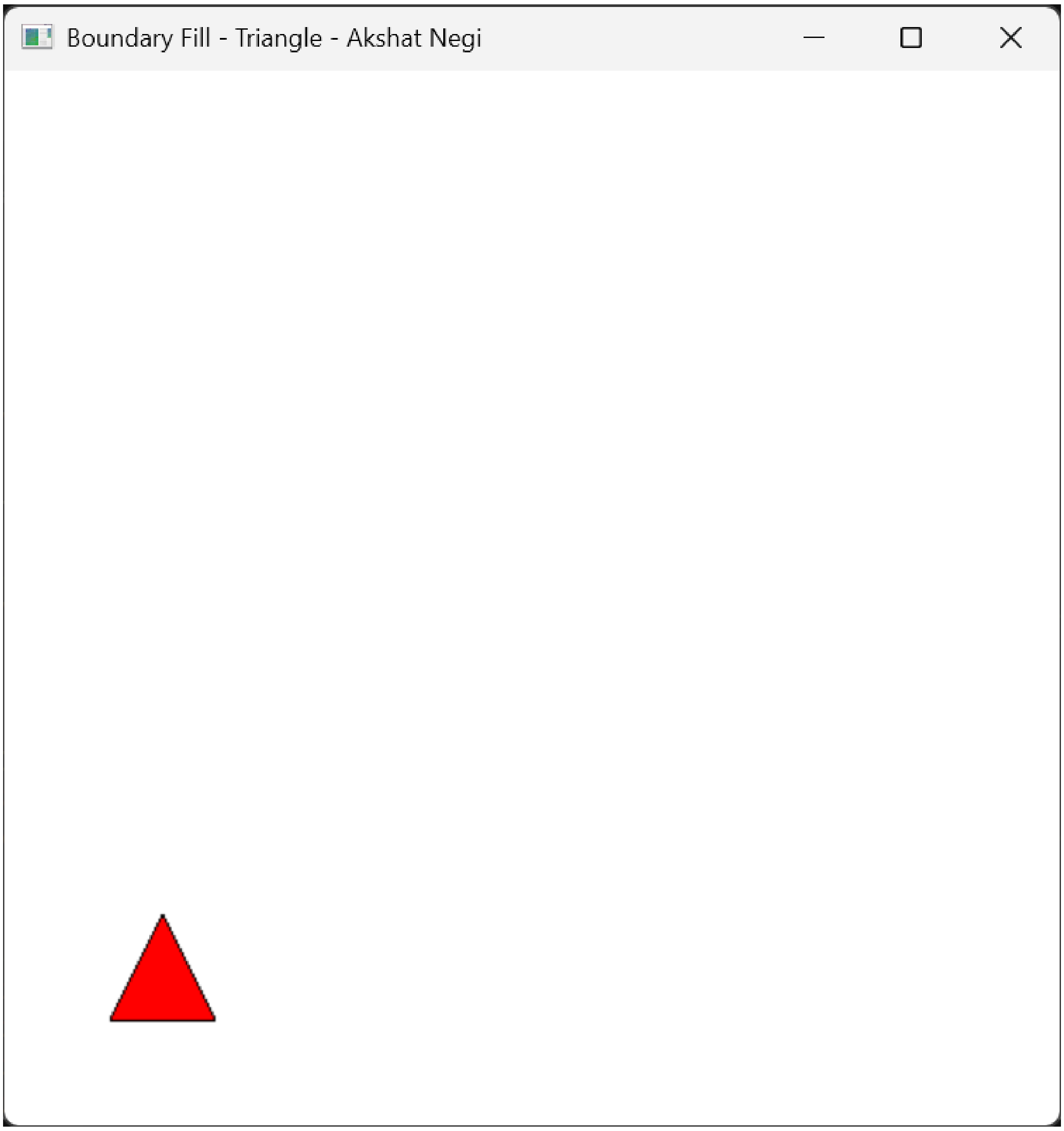
b. WAP to fill a region using boundary fill algorithm using 4 or 8 connected approaches.

```
#include <GL/freeglut.h>
#include <iostream>
#include <cmath>

float fillColor[3] = { 1.0, 0.0, 0.0 };  // Red color for filling
float borderColor[3] = { 0.0, 0.0, 0.0 }; // Black color for the boundary
float epsilon = 0.001;  // Tolerance for color comparison

// Function to set a pixel with a specific color
void setPixel(int x, int y, float* color) {
    glColor3fv(color);
    glBegin(GL_POINTS);
    glVertex2i(x, y);
    glEnd();
    glFlush();
}

// Function to get the color of a pixel at coordinates (x, y)
void getPixelColor(int x, int y, float* color) {
    glReadPixels(x, y, 1, 1, GL_RGB, GL_FLOAT, color);
}

// Helper function to compare two colors with a tolerance
bool isSameColor(float* color1, float* color2) {
    return (fabs(color1[0] - color2[0]) < epsilon &&
        fabs(color1[1] - color2[1]) < epsilon &&
        fabs(color1[2] - color2[2]) < epsilon);
}

// Boundary Fill Algorithm (8-connected)
void boundaryFill(int x, int y, float* fillColor, float* boundaryColor) {
    float currentColor[3];
    getPixelColor(x, y, currentColor);

    // If the pixel is neither the boundary nor the fill color, fill it
    if (!isSameColor(currentColor, boundaryColor) && !isSameColor(currentColor,
fillColor)) {
        setPixel(x, y, fillColor);

        // 8-connected neighbors
        boundaryFill(x + 1, y, fillColor, boundaryColor); // Right
        boundaryFill(x - 1, y, fillColor, boundaryColor); // Left
        boundaryFill(x, y + 1, fillColor, boundaryColor); // Up
        boundaryFill(x, y - 1, fillColor, boundaryColor); // Down
        boundaryFill(x + 1, y + 1, fillColor, boundaryColor); // Up-Right
        boundaryFill(x - 1, y + 1, fillColor, boundaryColor); // Up-Left
        boundaryFill(x + 1, y - 1, fillColor, boundaryColor); // Down-Right
        boundaryFill(x - 1, y - 1, fillColor, boundaryColor); // Down-Left
    }
}

// Function to draw a smaller triangle
void drawTriangle() {
    glColor3fv(borderColor); // Set border color (black)
    glBegin(GL_LINE_LOOP);
    glVertex2i(120, 150); // Top vertex
    glVertex2i(100, 100); // Bottom-left vertex
    glVertex2i(140, 100); // Bottom-right vertex
    glEnd();
    glFlush();
}

void display() {
    glClear(GL_COLOR_BUFFER_BIT);
    drawTriangle();  // Draw triangle on screen

    // Starting the boundary fill from a point inside the triangle
    boundaryFill(120, 120, fillColor, borderColor);
}

void init() {
    glClearColor(1.0, 1.0, 1.0, 1.0); // Set background color to white
    glColor3f(0.0, 0.0, 0.0);         // Set drawing color to black
    gluOrtho2D(0, 300, 0, 300);       // Set the coordinate system for the window
```

```
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(300, 300);    // Decrease window size
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Boundary Fill - 8 Connected Triangle - Akshat Negi");
    init();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

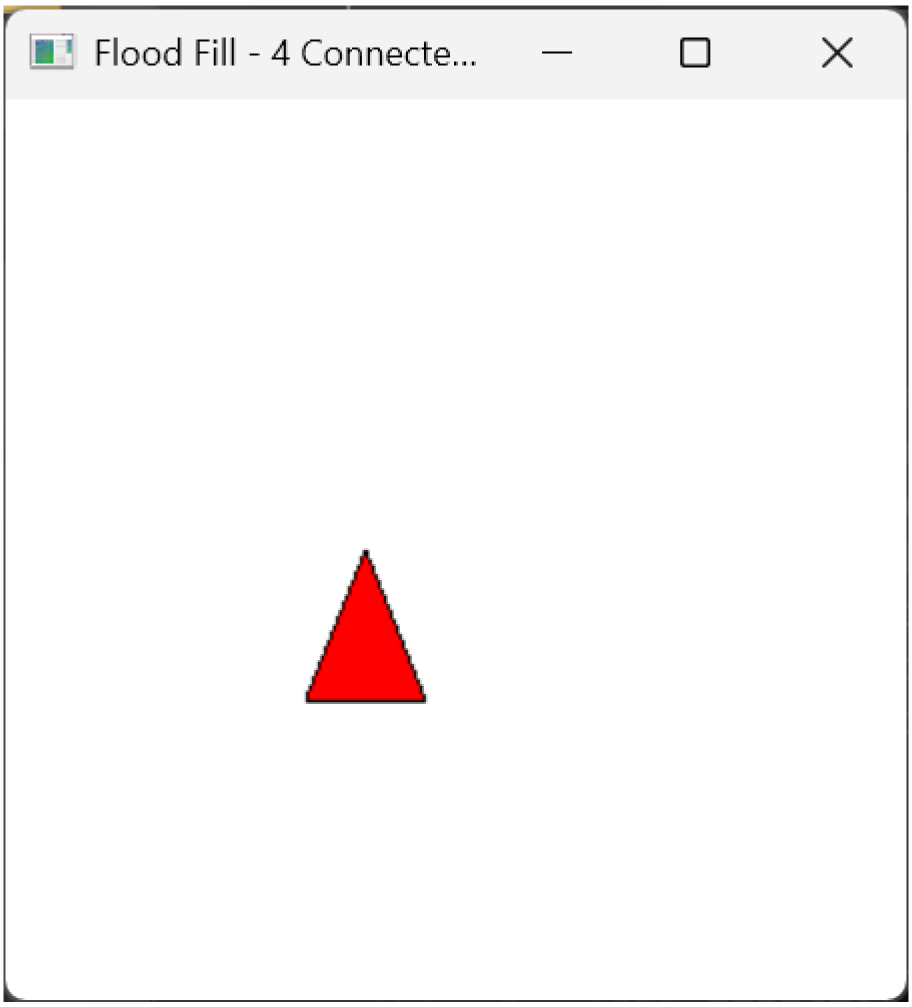


```
#include <GL/freeglut.h>
#include <iostream>
#include <cmath>

float fillColor[3] = { 1.0, 0.0, 0.0 };  // Red color for filling
float borderColor[3] = { 0.0, 0.0, 0.0 }; // Black color for the boundary
float epsilon = 0.001;  // Tolerance for color comparison

// Function to set a pixel with a specific color
void setPixel(int x, int y, float* color) {
    glColor3fv(color);
    glBegin(GL_POINTS);
    glVertex2i(x, y);
    glEnd();
    glFlush();
}

// Function to get the color of a pixel at coordinates (x, y)
void getPixelColor(int x, int y, float* color) {
    glReadPixels(x, y, 1, 1, GL_RGB, GL_FLOAT, color);
}

// Helper function to compare two colors with a tolerance
bool isSameColor(float* color1, float* color2) {
    return (fabs(color1[0] - color2[0]) < epsilon &&
        fabs(color1[1] - color2[1]) < epsilon &&
        fabs(color1[2] - color2[2]) < epsilon);
}

// Boundary Fill Algorithm (4-connected)
void boundaryFill(int x, int y, float* fillColor, float* boundaryColor) {
    float currentColor[3];
    getPixelColor(x, y, currentColor);

    // If the pixel is neither the boundary nor the fill color, fill it
    if (!isSameColor(currentColor, boundaryColor) && !isSameColor(currentColor,
fillColor)) {
        setPixel(x, y, fillColor);

        boundaryFill(x + 1, y, fillColor, boundaryColor);
        boundaryFill(x - 1, y, fillColor, boundaryColor);
        boundaryFill(x, y + 1, fillColor, boundaryColor);
        boundaryFill(x, y - 1, fillColor, boundaryColor);
    }
}

// Function to draw a triangle
void drawTriangle() {
    glColor3fv(borderColor); // Set border color (black)
    glBegin(GL_LINE_LOOP);
    glVertex2i(50, 50);  // Vertex 1 (Bottom-left corner)
    glVertex2i(100, 50); // Vertex 2 (Bottom-right corner)
    glVertex2i(75, 100); // Vertex 3 (Top corner)
    glEnd();
    glFlush();
}

void display() {
    glClear(GL_COLOR_BUFFER_BIT);
    drawTriangle();  // Draw triangle on screen

    // Starting the boundary fill from a point inside the triangle
    boundaryFill(75, 60, fillColor, borderColor); // Adjusted point for filling
}

void init() {
    glClearColor(1.0, 1.0, 1.0, 1.0); // Set background color to white
    glColor3f(0.0, 0.0, 0.0);         // Set drawing color to black
    gluOrtho2D(0, 500, 0, 500);       // Set the coordinate system for the window
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Boundary Fill - Triangle - Akshat Negi");
    init();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

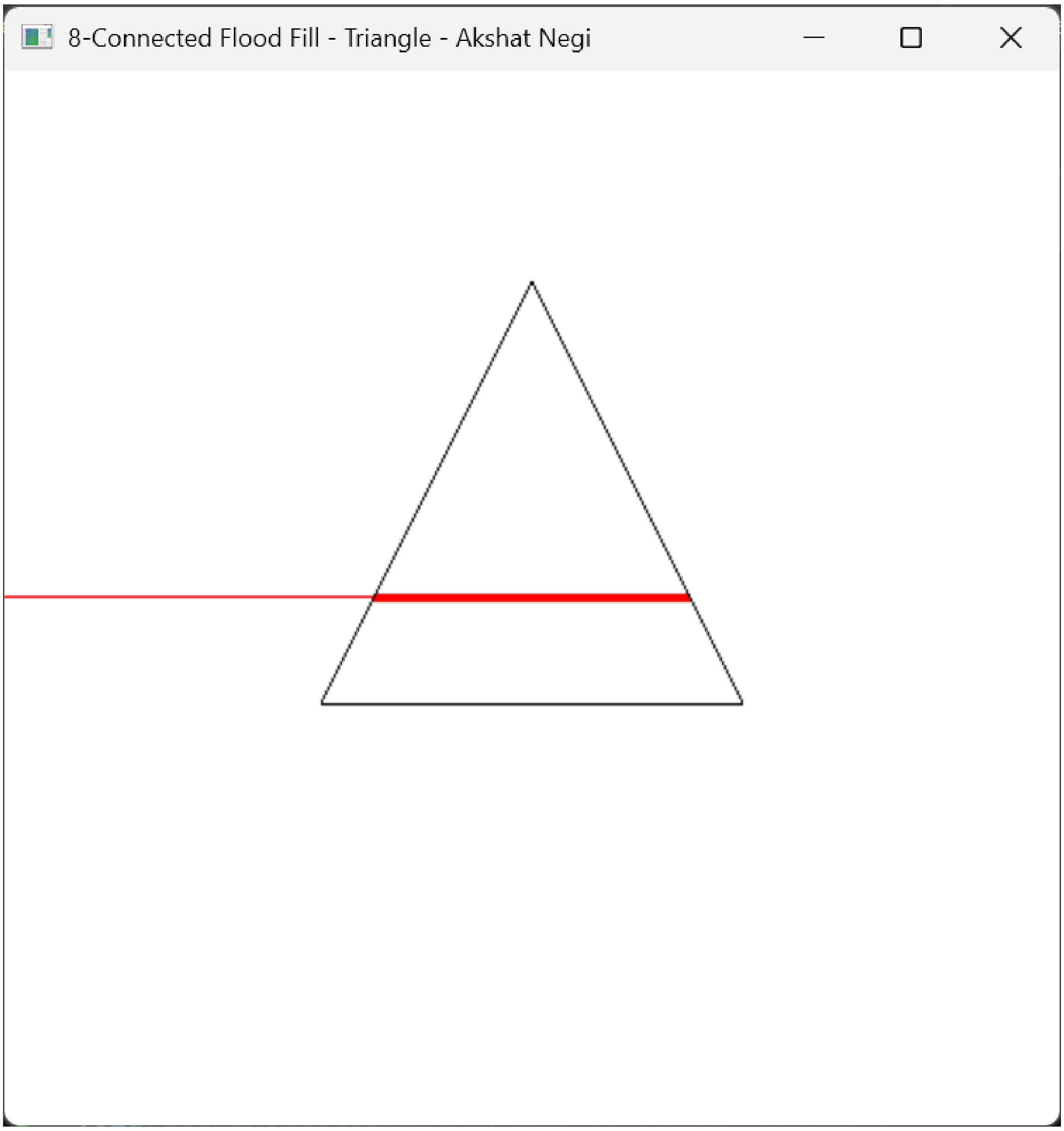


c. WAP to fill a region using flood fill algorithm using 4 or 8 connected approaches.

```
#include <GL/freeglut.h>
#include <iostream>
#include <cmath>

float fillColor[3] = { 1.0, 0.0, 0.0 };  // Red color for filling
float borderColor[3] = { 0.0, 0.0, 0.0 }; // Black color for the boundary
float epsilon = 0.001;  // Tolerance for color comparison

// Function to set a pixel with a specific color
void setPixel(int x, int y, float* color) {
    glColor3fv(color);
    glBegin(GL_POINTS);
    glVertex2i(x, y);
    glEnd();
    glFlush();
}

// Function to get the color of a pixel at coordinates (x, y)
void getPixelColor(int x, int y, float* color) {
    glReadPixels(x, y, 1, 1, GL_RGB, GL_FLOAT, color);
}

// Helper function to compare two colors with a tolerance
bool isSameColor(float* color1, float* color2) {
    return (fabs(color1[0] - color2[0]) < epsilon &&
        fabs(color1[1] - color2[1]) < epsilon &&
        fabs(color1[2] - color2[2]) < epsilon);
}

// 4-Connected Flood Fill Algorithm
void floodFill(int x, int y, float* fillColor, float* boundaryColor) {
    float currentColor[3];
    getPixelColor(x, y, currentColor);

    // If the pixel is neither the boundary nor the fill color, fill it
    if (!isSameColor(currentColor, boundaryColor) && !isSameColor(currentColor,
fillColor)) {
        setPixel(x, y, fillColor);

        // 4-connected neighbors
        floodFill(x + 1, y, fillColor, boundaryColor); // Right
        floodFill(x - 1, y, fillColor, boundaryColor); // Left
        floodFill(x, y + 1, fillColor, boundaryColor); // Up
        floodFill(x, y - 1, fillColor, boundaryColor); // Down
    }
}

// Function to draw a smaller triangle
void drawTriangle() {
    glColor3fv(borderColor); // Set border color (black)
    glBegin(GL_LINE_LOOP);
    glVertex2i(120, 150); // Top vertex
    glVertex2i(100, 100); // Bottom-left vertex
    glVertex2i(140, 100); // Bottom-right vertex
    glEnd();
    glFlush();
}

void display() {
    glClear(GL_COLOR_BUFFER_BIT);
    drawTriangle();  // Draw triangle on screen

    // Starting the flood fill from a point inside the triangle
    floodFill(120, 120, fillColor, borderColor);
}

void init() {
    glClearColor(1.0, 1.0, 1.0, 1.0); // Set background color to white
    glColor3f(0.0, 0.0, 0.0);         // Set drawing color to black
    gluOrtho2D(0, 300, 0, 300);       // Set the coordinate system for the window
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(300, 300);    // Decrease window size
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Flood Fill - 4 Connected Triangle - Akshat Negi");
    init();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```



```cpp
#include <GL/freeglut.h>
#include <iostream>
#include <cmath>

float fillColor[3] = { 1.0, 0.0, 0.0 };  // Red color for filling
float borderColor[3] = { 0.0, 0.0, 0.0 }; // Black color for the boundary
float epsilon = 0.001;  // Tolerance for color comparison

// Function to set a pixel with a specific color
void setPixel(int x, int y, float* color) {
    glColor3fv(color);
    glBegin(GL_POINTS);
    glVertex2i(x, y);
    glEnd();
    glFlush();
}

// Function to get the color of a pixel at coordinates (x, y)
void getPixelColor(int x, int y, float* color) {
    glReadPixels(x, y, 1, 1, GL_RGB, GL_FLOAT, color);
}

// Helper function to compare two colors with a tolerance
bool isSameColor(float* color1, float* color2) {
    return (fabs(color1[0] - color2[0]) < epsilon &&
        fabs(color1[1] - color2[1]) < epsilon &&
        fabs(color1[2] - color2[2]) < epsilon);
}

// Flood Fill Algorithm (8-connected)
void floodFill(int x, int y, float* fillColor, float* boundaryColor) {
    float currentColor[3];
    getPixelColor(x, y, currentColor);

    // If the pixel is neither the boundary nor the fill color, fill it
    if (!isSameColor(currentColor, boundaryColor) && !isSameColor(currentColor,
fillColor)) {
        setPixel(x, y, fillColor);

        // Recursively call floodFill for 8-connected neighbors
        floodFill(x + 1, y, fillColor, boundaryColor); // Right
        floodFill(x - 1, y, fillColor, boundaryColor); // Left
        floodFill(x, y + 1, fillColor, boundaryColor); // Up
        floodFill(x, y - 1, fillColor, boundaryColor); // Down
        floodFill(x + 1, y + 1, fillColor, boundaryColor); // Top-Right
        floodFill(x - 1, y - 1, fillColor, boundaryColor); // Bottom-Left
        floodFill(x + 1, y - 1, fillColor, boundaryColor); // Bottom-Right
        floodFill(x - 1, y + 1, fillColor, boundaryColor); // Top-Left
    }
}

// Function to draw a triangle
void drawTriangle() {
    glColor3fv(borderColor); // Set border color (black)
    glBegin(GL_LINE_LOOP);
    glVertex2i(250, 400); // Top vertex
    glVertex2i(150, 200); // Bottom-left vertex
    glVertex2i(350, 200); // Bottom-right vertex
    glEnd();
    glFlush();
}

void display() {
    glClear(GL_COLOR_BUFFER_BIT);
    drawTriangle();  // Draw triangle on screen

    // Starting the flood fill from a point inside the triangle
    floodFill(250, 250, fillColor, borderColor);
}

void init() {
    glClearColor(1.0, 1.0, 1.0, 1.0); // Set background color to white
    glColor3f(0.0, 0.0, 0.0);         // Set drawing color to black
    gluOrtho2D(0, 500, 0, 500);       // Set the coordinate system for the window
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("8-Connected Flood Fill - Triangle - Akshat Negi");
    init();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```


8-Connected Flood Fill - Triangle - Akshat Negi