Practical No.:4

Title: Implement the following polygon filling methods.
i) Flood fill / Seed fill
ii) Boundary fill
Name: Samruddhi Sangram Yadav.
Roll No.:S564

i) Flood fill / Seed fill

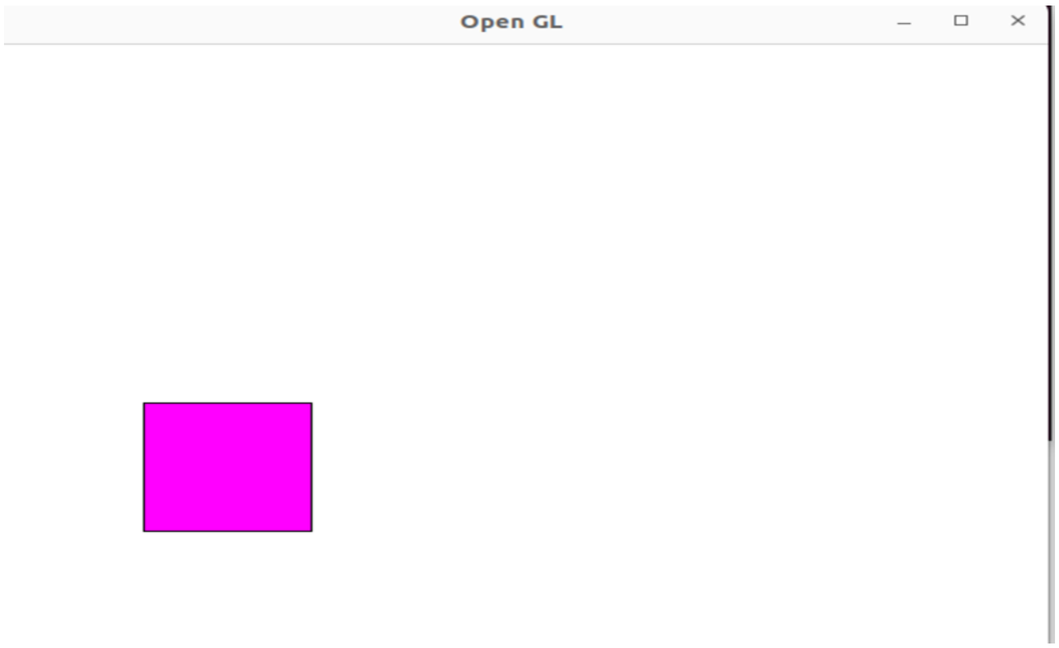
```
#include <GL/glut.h>
#include <cmath>
#include <iostream>
#include <thread>  // For std::this_thread::sleep_for
#include <chrono>  // For duration in milliseconds
struct Point {
	GLint x;
	GLint y;
};
struct Color {
	GLfloat r;
	GLfloat g;
	GLfloat b;
};
// Function to draw a line using DDA algorithm
void draw_dda(Point p1, Point p2) {
	GLfloat dx = p2.x - p1.x;
	GLfloat dy = p2.y - p1.y;
	GLfloat x1 = p1.x;
	GLfloat y1 = p1.y;
	GLfloat step = 0;
	if (abs(dx) > abs(dy)) {
		step = abs(dx);
	} else {
		step = abs(dy);
	}
	GLfloat xInc = dx / step;
	GLfloat yInc = dy / step;
	for (float i = 1; i <= step; i++) {
		glVertex2i(x1, y1);
		x1 += xInc;
		y1 += yInc;
	}
}
// Initialization of OpenGL settings
void init() {
	glClearColor(1.0, 1.0, 1.0, 0.0);
	glColor3f(0.0, 0.0, 0.0);
	glPointSize(1.0);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
```

```
        gluOrtho2D(0, 640, 0, 480);
}
// Function to get the color of a pixel
Color getPixelColor(GLint x, GLint y) {
        Color color;
        glReadPixels(x, y, 1, 1, GL_RGB, GL_FLOAT, &color);
        return color;
}
// Function to set the color of a pixel
void setPixelColor(GLint x, GLint y, Color color) {
        glColor3f(color.r, color.g, color.b);
        glBegin(GL_POINTS);
                glVertex2i(x, y);
        glEnd();
        glFlush();
}
// 4-connectivity flood fill algorithm with delay to visualize the filling process
void floodFill(GLint x, GLint y, Color oldColor, Color newColor) {
        Color color = getPixelColor(x, y);
        // Check if the color of the pixel matches the old color
        if (color.r == oldColor.r && color.g == oldColor.g && color.b == oldColor.b) {
                setPixelColor(x, y, newColor);
                // Delay for a short period to visualize the filling process
                std::this_thread::sleep_for(std::chrono::milliseconds(2));  // Adjust the delay as needed
                // Recursive calls for 4-connectivity (all 4 neighboring pixels)
                floodFill(x - 1, y, oldColor, newColor);     // Left
                floodFill(x + 1, y, oldColor, newColor);     // Right
                floodFill(x, y + 1, oldColor, newColor);     // Bottom
                floodFill(x, y - 1, oldColor, newColor);     // Top
        }
}
// Mouse click handler to initiate the flood fill
void onMouseClick(int button, int state, int x, int y) {
        // Define the colors
        Color newColor = {1.0f, 0.0f, 1.0f}; // Purple
        Color oldColor = {1.0f, 1.0f, 1.0f}; // White
        // Start flood fill at the given point (adjust for OpenGL coordinate system)
        floodFill(x, 480 - y, oldColor, newColor);
}
// Display function to draw the square and trigger the flood fill
void display(void) {
        Point p1 = {100, 100},   // bottom-right
                p2 = {200, 100},   // bottom-left (increased x by 200)
                p3 = {200, 200},   // top-right (increased x and y by 200)
                p4 = {100, 200};   // top-left (increased y by 200)
        glClear(GL_COLOR_BUFFER_BIT);
        glBegin(GL_POINTS);
                draw_dda(p1, p2);
                draw_dda(p2, p3);
                draw_dda(p3, p4);
                draw_dda(p4, p1);
        glEnd();
```

```
	glFlush();
}
// Main function to set up GLUT and OpenGL context
int main(int argc, char** argv) {
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
	glutInitWindowSize(640, 480);
	glutInitWindowPosition(200, 200);
	glutCreateWindow("Open GL");
	// Initialize OpenGL
	init();
	// Register callback functions
	glutDisplayFunc(display);
	glutMouseFunc(onMouseClick);
	// Enter the main GLUT loop
	glutMainLoop();
	return 0;
}
```

Output:

```
it@it-HP-EliteDesk-800-G2-SFF:~$ g++ fdelay.cpp -lGL -lGLU -lglut
it@it-HP-EliteDesk-800-G2-SFF:~$ ./a.out
```

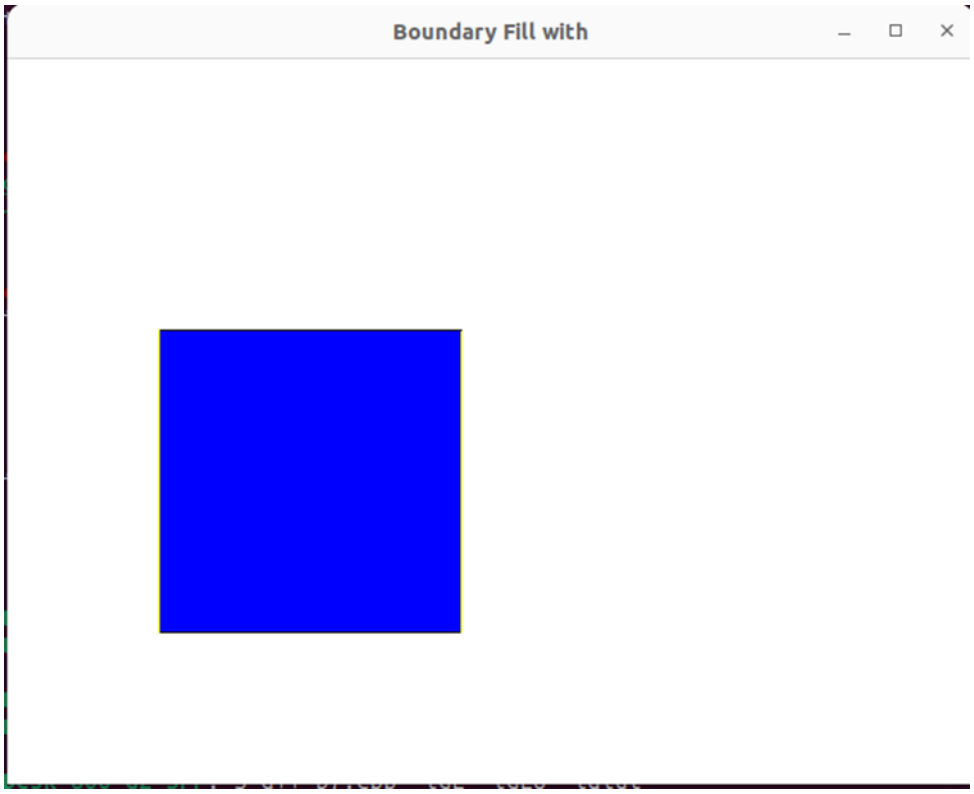


ii) Boundary fill

```
#include <GL/glut.h>
#include <iostream>
struct Point {
    GLint x, y;
};
struct Color {
    GLfloat r, g, b;
};
void draw_dda(Point p1, Point p2, Color color) {
    GLfloat dx = p2.x - p1.x;
    GLfloat dy = p2.y - p1.y;
    GLfloat steps = (abs(dx) > abs(dy)) ? abs(dx) : abs(dy);
    GLfloat xInc = dx / steps;
    GLfloat yInc = dy / steps;
    GLfloat x = p1.x, y = p1.y;
    glColor3f(color.r, color.g, color.b);
    glBegin(GL_POINTS);
    for (int i = 0; i <= steps; i++) {
        glVertex2i(x, y);
        x += xInc;
        y += yInc;
    }
    glEnd();
    glFlush();
}
void init() {
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glPointSize(1.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, 640, 0, 480);
}
Color getPixelColor(GLint x, GLint y) {
    Color color;
    GLfloat pixel[3];
    glReadPixels(x, y, 1, 1, GL_RGB, GL_FLOAT, pixel);
    color.r = pixel[0];
    color.g = pixel[1];
    color.b = pixel[2];
    return color;
}
void setPixelColor(GLint x, GLint y, Color color) {
    glColor3f(color.r, color.g, color.b);
    glBegin(GL_POINTS);
    glVertex2i(x, y);
    glEnd();
    glFlush();
}
void boundaryFill8(GLint x, GLint y, Color fillColor, Color boundaryColor1, Color boundaryColor2) {
    Color color = getPixelColor(x, y);
    if(!(color.r == boundaryColor1.r && color.g == boundaryColor1.g && color.b ==
boundaryColor1.b) &&
    !(color.r == boundaryColor2.r && color.g == boundaryColor2.g && color.b == boundaryColor2.b)
&&
    !(color.r == fillColor.r && color.g == fillColor.g && color.b == fillColor.b))
    {
        setPixelColor(x, y, fillColor);
         boundaryFill8(x - 1, y - 1, fillColor, boundaryColor1, boundaryColor2);
        boundaryFill8(x + 1, y + 1, fillColor, boundaryColor1, boundaryColor2);
        boundaryFill8(x + 1, y, fillColor, boundaryColor1, boundaryColor2);
        boundaryFill8(x, y + 1, fillColor, boundaryColor1, boundaryColor2);
        boundaryFill8(x, y - 1, fillColor, boundaryColor1, boundaryColor2);
        boundaryFill8(x - 1, y, fillColor, boundaryColor1, boundaryColor2);
        boundaryFill8(x + 1, y - 1, fillColor, boundaryColor1, boundaryColor2);
        boundaryFill8(x - 1, y + 1, fillColor, boundaryColor1, boundaryColor2);
    }
}
void onMouseClick(int button, int state, int x, int y) {
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
        Color fillColor1 = {0.0f, 0.0f, 1.0f}; // red
        Color boundaryColor1 = {0.0f, 0.0f, 0.0f}; // Black
        Color boundaryColor2 = {1.0f, 1.0f, 0.0f}; // Blue
        boundaryFill8(x, 480 - y, fillColor1, boundaryColor1, boundaryColor2);
    }
}
void display(void) {
    glClear(GL_COLOR_BUFFER_BIT);
    Point p1 = {100, 100}, p2 = {300, 100}, p3 = {300, 300}, p4 = {100, 300};
    Color red = {0.0f, 0.0f, 0.0f}; // black
    Color blue = {1.0f, 1.0f, .0f}; // Blue
    draw_dda(p1, p2, red);
    draw_dda(p2, p3, blue);
    draw_dda(p3, p4, red);
    draw_dda(p4, p1, blue);
    glFlush();
}
int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(200, 200);
    glutCreateWindow("Boundary Fill with");
    init();
    glutDisplayFunc(display);
    glutMouseFunc(onMouseClick);
    glutMainLoop();
    return 0;
}
```

Output:

```
it@it-HP-EliteDesk-800-G2-SFF:~$ g++ b7.cpp -lGL -lGLU -lglut
it@it-HP-EliteDesk-800-G2-SFF:~$ ./a.out
```


Boundary Fill with