Title

Bresenham Line Drawing Algorithm Implementation.

Introduction

In this lab task, I implemented the Bresenham Line Drawing Algorithm using OpenGL and GLUT. The purpose of this experiment was to draw a line between two user-defined points using only integer arithmetic, making the process faster and more efficient compared to algorithms that rely on floating-point calculations. Bresenham's algorithm determines the next pixel position based on a decision parameter, allowing the line to be rendered with high accuracy and minimal computational cost. Through this task, I learned how to initialize an OpenGL environment, take input from the user, execute the Bresenham algorithm, and display the resulting line on the screen. This lab enhanced my understanding of raster graphics and how precise lines can be generated using incremental decision-based logic.

Contents

In this lab task :

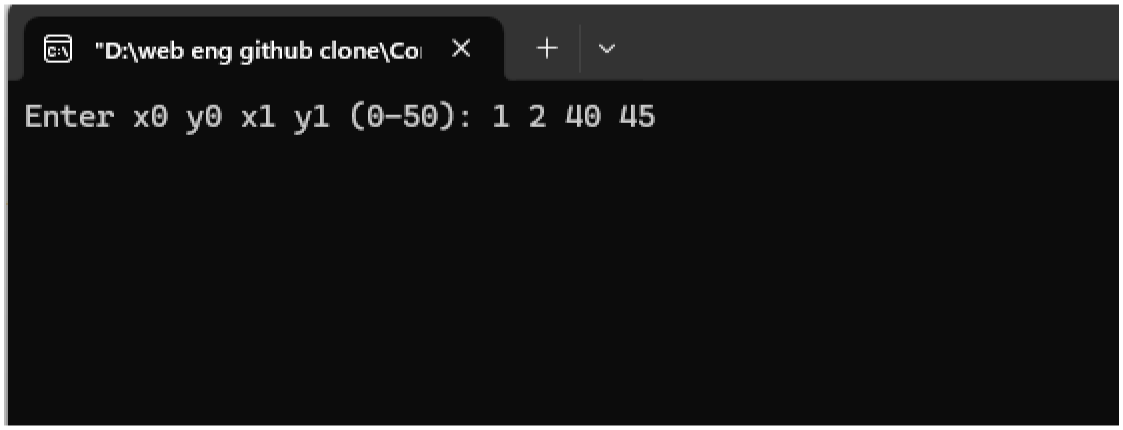
1. Functions used
   - 'glClear() – clears the display
   - glColor3f() – sets the drawing color
   - glBegin() / glEnd() – starts and ends point plotting
   - glVertex2i() – plots each selected pixel on the line
   - glFlush() – outputs the rendered graphics
   - drawLineBresenham() – applies Bresenham's decision formula to determine the next pixel for the line.
2. Inputs taken:
   - Starting point (x_0, y_0)
   - Ending point (x_1, y_1)

Code

```
#include <GL/gl.h>

#include <GL/glut.h>

#include <stdlib.h>

#include <stdio.h>


int x0_user, y0_user, x1_user, y1_user;
```

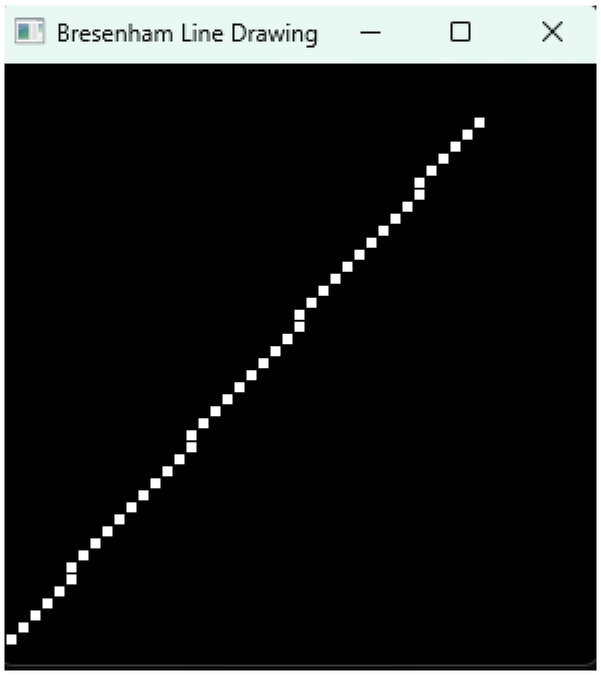
```
void init(void)
{
    glClearColor(0.0, 0.0, 0.0, 0.0);
    gluOrtho2D(0, 50, 0, 50);
    glPointSize(5.0);
}

void drawLineBresenham(int x0, int y0, int x1, int y1)
{
    int dx = abs(x1 - x0);
    int dy = abs(y1 - y0);
    int sx = x0 < x1 ? 1 : -1;
    int sy = y0 < y1 ? 1 : -1;
    int err = dx - dy;

    glColor3f(1.0, 1.0, 1.0);
    glBegin(GL_POINTS);
    while (1)
    {
        glVertex2i(x0, y0);
        if (x0 == x1 && y0 == y1) break;
        int e2 = 2 * err;
        if (e2 > -dy)
        {
            err -= dy;
            x0 += sx;
        }
        if (e2 < dx)
        {
            err += dx;
            y0 += sy;
        }
    }
    glEnd();
}

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    drawLineBresenham(x0_user, y0_user, x1_user, y1_user);
    glFlush();
}

int main(int argc, char** argv)
{
    printf("Enter x0 y0 x1 y1 (0-50): ");
    scanf("%d %d %d %d", &x0_user, &y0_user, &x1_user, &y1_user);

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(300, 300);
    glutCreateWindow("Bresenham Line Drawing");
    init();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

Input



```
Enter x0 y0 x1 y1 (0-50): 1 2 40 45
```

Output



Discussion

In this lab task, I successfully implemented the Bresenham Line Drawing Algorithm using OpenGL. Unlike the DDA algorithm, Bresenham uses integer calculations and a decision parameter to determine whether the next plotted pixel should move horizontally or diagonally. This makes the algorithm faster and avoids floating-point rounding errors. Each selected pixel along the line was plotted using glVertex2i(), while glColor3f() was used to give the line a clear visual appearance. The program demonstrated how Bresenham's approach provides accurate and efficient rasterized line drawing on a pixel grid. Through this lab, I gained a deeper understanding of how optimized algorithms can enhance rendering performance while maintaining line precision in computer graphics.