Problem Statement

WAP that uses the Digital Differential Analyzer (DDA) algorithm to draw a straight line between two user-defined points, in a graphics window.

Background theory

The DDA line drawing algorithm offers an effective method for rendering lines in computer graphics by computing intermediate pixel positions. It handles all types of slopes efficiently and produces a smooth approximation of a straight line. This lab provided valuable insight into how mathematical concepts can be practically applied to pixel-level graphics programming using C++.

Algorithm

1. Begin with the coordinates of the start and end points: (x1, y1) and (x2, y2).

2. Compute the change in each direction:
   dx = x2 - x1
   dy = y2 - y1

3. Determine the total number of steps to take by finding the greater of the absolute values of dx and dy:
   steps = max(|dx|, |dy|)

4. Calculate the amount to increment x and y in each step:
   x_inc = dx / steps
   y_inc = dy / steps

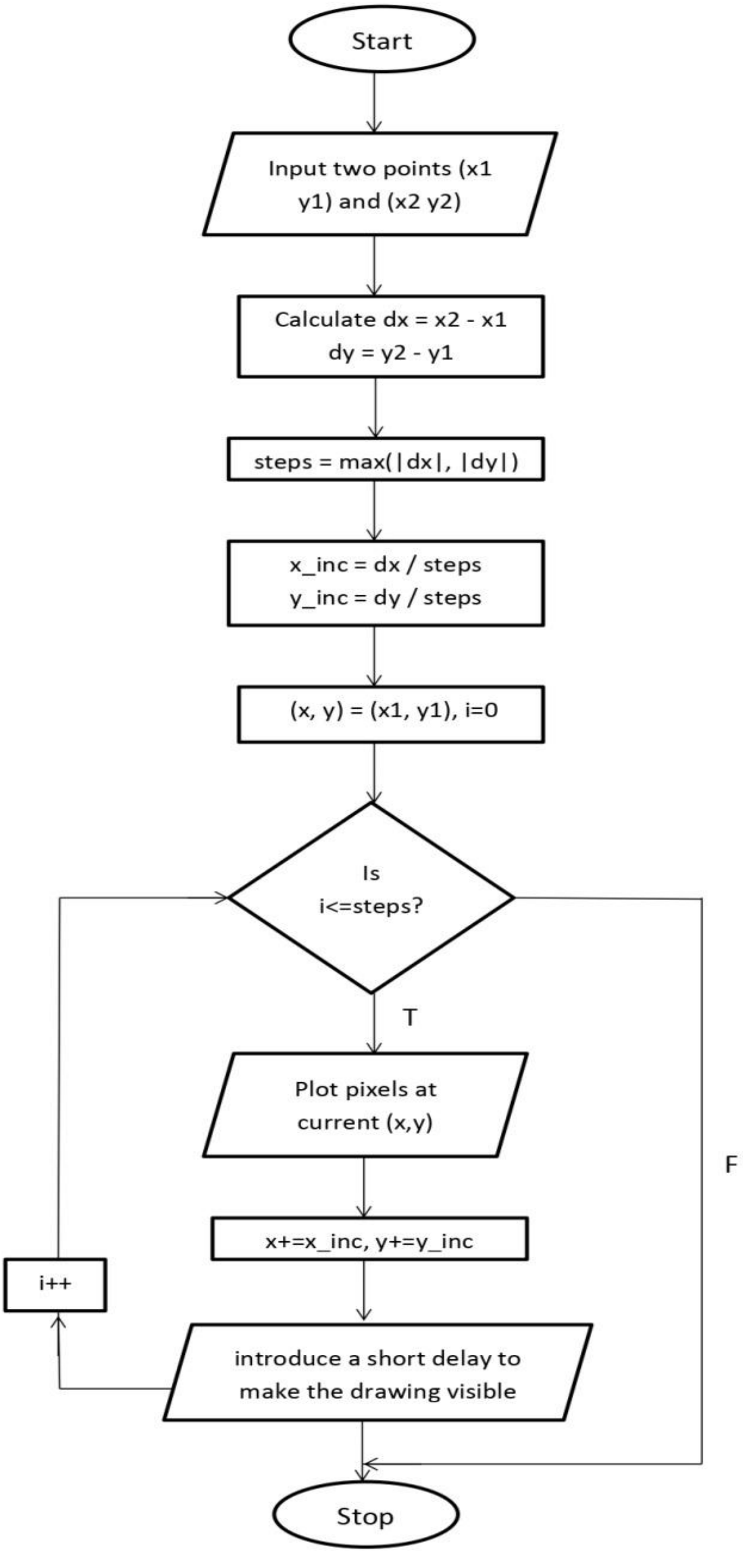
5. Initialize the current position at (x, y) = (x1, y1). For each step, repeat the following:

6. Plot the current point (x, y) using a pixel-drawing function.

7. Add x_inc to x and y_inc to y.

8. Optionally introduce a short delay to make the drawing visible.

Flowchart


Start
Input two points (x1 y1) and (x2 y2)
Calculate dx = x2 - x1
dy = y2 - y1
steps = max(|dx|, |dy|)
x_inc = dx / steps
y_inc = dy / steps
(x, y) = (x1, y1), i=0
Is i<=steps?
T
F
Plot pixels at current (x,y)
x+=x_inc, y+=y_inc
introduce a short delay to make the drawing visible
i++
Stop

Source code

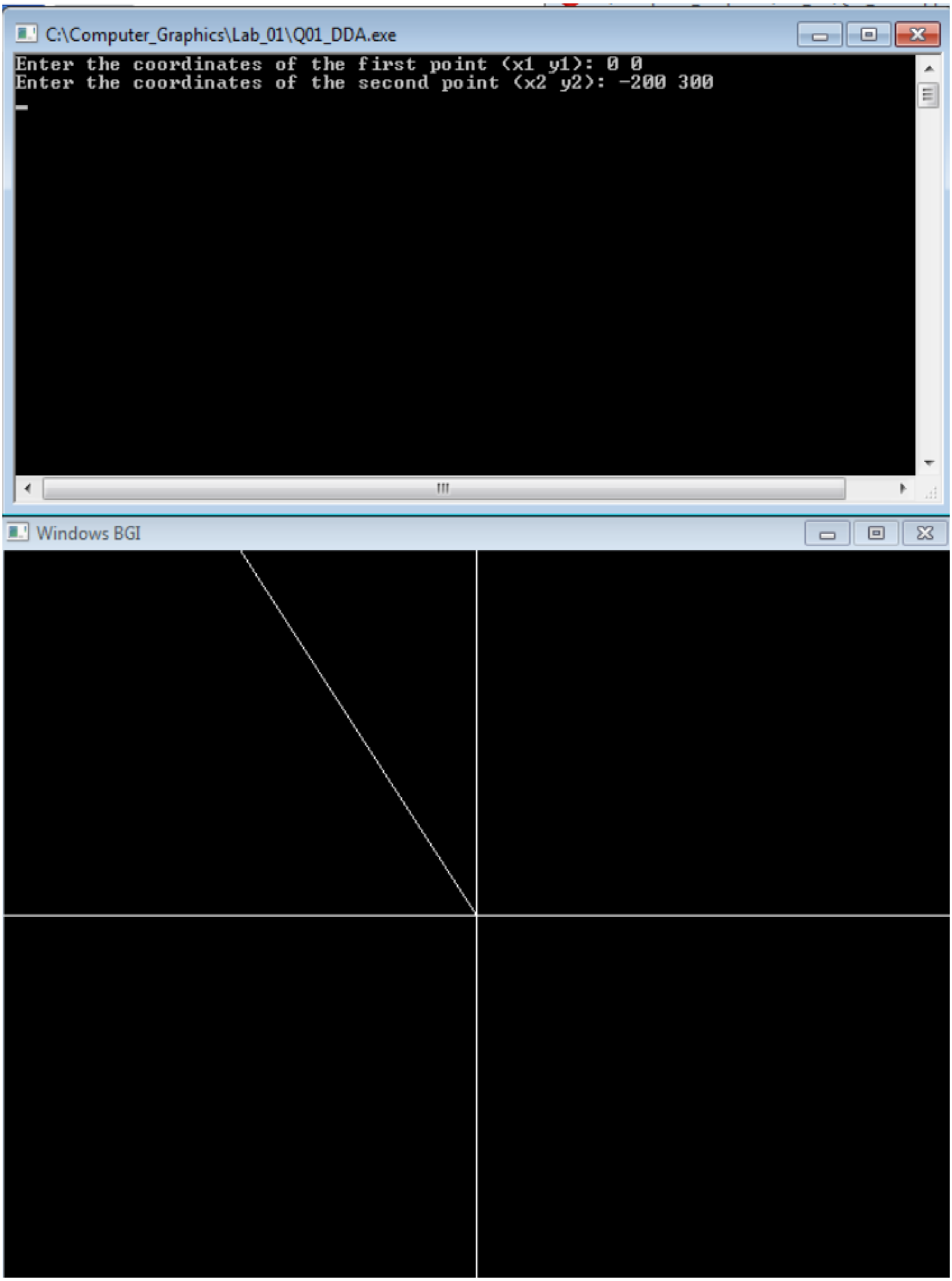
```
#include <iostream>
#include <cmath>
#include <graphics.h>
using namespace std;
void drawLineDDA(int x1, int y1, int x2, int y2, int maxHeight, int maxWidth)
{
    float x_inc, y_inc, x, y, steps;
    int dx = x2 - x1;
    int dy = y2 - y1;

    steps = (abs(dx) > abs(dy)) ? abs(dx) : abs(dy);
    x_inc = dx / steps;
    y_inc = dy / steps;
    // Adjust for screen center and invert Y-axis
    x = x1 + maxWidth / 2;
    y = maxHeight / 2 - y1;  // subtract instead of add

    for (int i = 0; i <= steps; i++) {
        if (x >= 0 && x < maxWidth && y >= 0 && y < maxHeight)
            putpixel(round(x), round(y), WHITE);
        x += x_inc;
        y -= y_inc;  // invert y-axis by subtracting
        delay(10);
    }
}
int main() {
    int gd = DETECT, gm;
    initgraph(&gd, &gm, NULL);
    int maxWidth = getmaxx();
    int maxHeight = getmaxy();
    line(maxWidth/2, 0, maxWidth/2, maxHeight);
    line(0, maxHeight/2, maxWidth, maxHeight/2);

    int x1, y1, x2, y2;
    cout << "Enter the coordinates of the first point (x1 y1): ";
    cin >> x1 >> y1;
    cout << "Enter the coordinates of the second point (x2 y2): ";
    cin >> x2 >> y2;
    drawLineDDA(x1, y1, x2, y2, maxHeight, maxWidth);
    getch();
    closegraph();
    return 0;
}
```

Sample input and output



```
Enter the coordinates of the first point (x1 y1): 0 0
Enter the coordinates of the second point (x2 y2): -200 300
_
```

Conclusion

We can conclude that the DDA line drawing algorithm offers an effective method for rendering lines in computer graphics by computing intermediate pixel positions. It handles all types of slopes efficiently and produces a smooth approximation of a straight line. This lab provided valuable insight into how mathematical concepts can be practically applied to pixel-level graphics programming using C++.