Experiment 2

Student Name: Abhiraj Patel **UID: 22BCS11329**

Branch: BE-CSE **Section/Group: KRG-IOT-1-A**

Semester: 6th **Date of Performance: 16/01/25**

Subject Name: Computer Graphics with Lab **Subject Code: 22CSH-352**

Aim: Implement and compare the performance of Simple DDA, Symmetrical DDA, and Bresenham's algorithm for positive and negative line slope.

Objective: To implement and compare the performance of Simple Digital Differential Analyzer (DDA), Symmetrical DDA, and Bresenham's line-drawing algorithms for rendering lines with positive and negative slopes, analyzing their computational efficiency, accuracy, and suitability for different scenarios in computer graphics.

Algorithm:

Calculate Differences:

- `dx = x2 - x1`
- `dy = y2 - y1`

Determine the number of steps:

- `steps = max(abs(dx), abs(dy))` **Calculate the increments**:

- `xInc = dx / steps` (For Simple DDA)
- `yInc = dy / steps` (For Simple DDA)

Set the initial points:

- `x = x1`
- `y = y1`

Error Handling (Symmetrical DDA):

- `error = 0.5` (Error term to handle precision issues)

Main Loop (Bresenham-like):

While `steps > 0`:

- Plot the point `(round(x), round(y))` ○ If `abs(dx) > abs(dy)` (Line has a shallower slope):
  - Increment `x` by `xInc` ▪ Update `error = error + dy`
  - If `error >= 0.5` , increment `y` by `yInc` and reset error: `error = error - 1`
- Else (Line has a steeper slope):
  - Increment `y` by `yInc` ▪ Update `error = error + dx`
  - If `error >= 0.5` , increment `x` by `xInc` and reset error: `error = error - 1`
    - Decrease `steps`

Handle Negative Slopes (Symmetrical DDA-like adjustment):

- If `dy < 0`, reverse the direction and handle accordingly by updating the increments (i.e., `yInc = -yInc`).

Repeat until the last point `(x2, y2)` is reached.

Code:

```
#include <iostream.h>
#include <graphics.h>
#include <conio.h>
#include <math.h>
#include <dos.h> // For delay()

#define round(a) ((int)(a + 0.5))

void dda_line(int x1, int y1, int x2, int y2) {
   int dx = x2 - x1;
   int dy = y2 - y1;
   int length;

   if (abs(dy) > abs(dx))
      length = abs(dy);
   else
      length = abs(dx);

   float xinc = dx / (float)length;
```

```
    float yinc = dy / (float)length;
    float x = x1, y = y1;

    // Draw the initial pixel
    putpixel(round(x), round(y), WHITE);

    // Draw subsequent pixels using the DDA algorithm
    for (int k = 1; k <= length; k++) {
        x += xinc;
        y += yinc;
        putpixel(round(x), round(y), WHITE);
        delay(50); // Delay in milliseconds
    }

    // Calculate the midpoint of the line
    int midX = (x1 + x2) / 2;
    int midY = y1;

    // Display name "Tanmaya" centered below the line
    setcolor(WHITE);
    outtextxy(midX - 30, midY + 10, "Abhiraj"); // Adjust text alignment for Turbo C++
}

void main() {
    int x1, x2, y1, y2;
    int gd = DETECT, gm;

    // Input coordinates for the horizontal line
    cout << "Enter the x-coordinate of the starting point: ";
    cin >> x1;

    cout << "Enter the y-coordinate of the line: ";
    cin >> y1;

    cout << "Enter the x-coordinate of the ending point: ";
    cin >> x2;

    y2 = y1; // Keep y2 same as y1 for a horizontal line

    // Initialize the graphics window
    initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");

    // Draw the line and display the name
    dda_line(x1, y1, x2, y2);

    getch();
    closegraph();
}
```

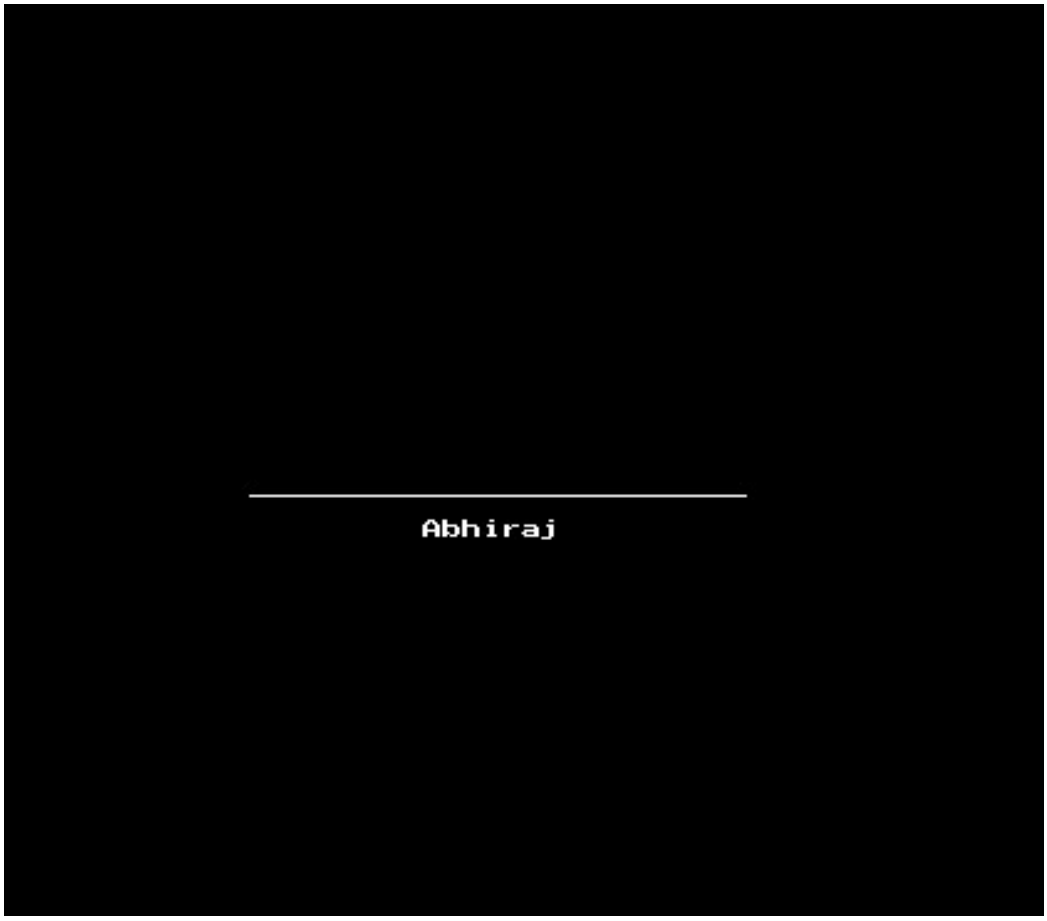
```
≡ File  Edit  Search  Run  Compile  Debug  Project  Options  Window  Help
[■]═══ PROJECT\PRINT.CPP ═══1=[↑]
#include <iostream.h>
#include <graphics.h>
#include <conio.h>
#include <math.h>
#include <dos.h> // For delay()

#define round(a) ((int)(a + 0.5))

void dda_line(int x1, int y1, int x2, int y2) {
    int dx = x2 - x1;
    int dy = y2 - y1;
    int length;

    if (abs(dy) > abs(dx))
═══ 1:1 ═══
F1 Help  F2 Save  F3 Open  Alt-F9 Compile  F9 Make  F10 Menu
```

Output:



Learning Outcomes:

1. **Line Drawing Concepts**: Understand the working of DDA and Bresenham's algorithms for line drawing.
2. **Performance Comparison**: Analyze and compare the efficiency of Simple DDA, Symmetrical DDA, and Bresenham's algorithms.
3. **Error Handling**: Learn how error terms are managed in graphics algorithms to minimize visual imperfections.
4. **Optimization**: Explore the efficiency of integer-based algorithms (like Bresenham's) vs. floating-point methods (like DDA).
5. **Coding and Debugging**: Improve coding and debugging skills through algorithm implementation and testing.