WORKSHEET 2

Student Name: Sumit Kumar **UID: 22BCS10048**
Section/Group: KRG_IOT-1-A **Semester: 6th Semester**
Branch: BE-CSE **Date of Performance: 16/01/2025**
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)) o 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
      o 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 o 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.

Implementation/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
   }

int midX = (x1 + x2) / 2;     int midY =
y1;

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

void main() {
int x1, x2, y1, y2;
   int gd = DETECT, gm;
cout << "Enter the x-coordinate of the starting point: ";
cin >> x1;

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

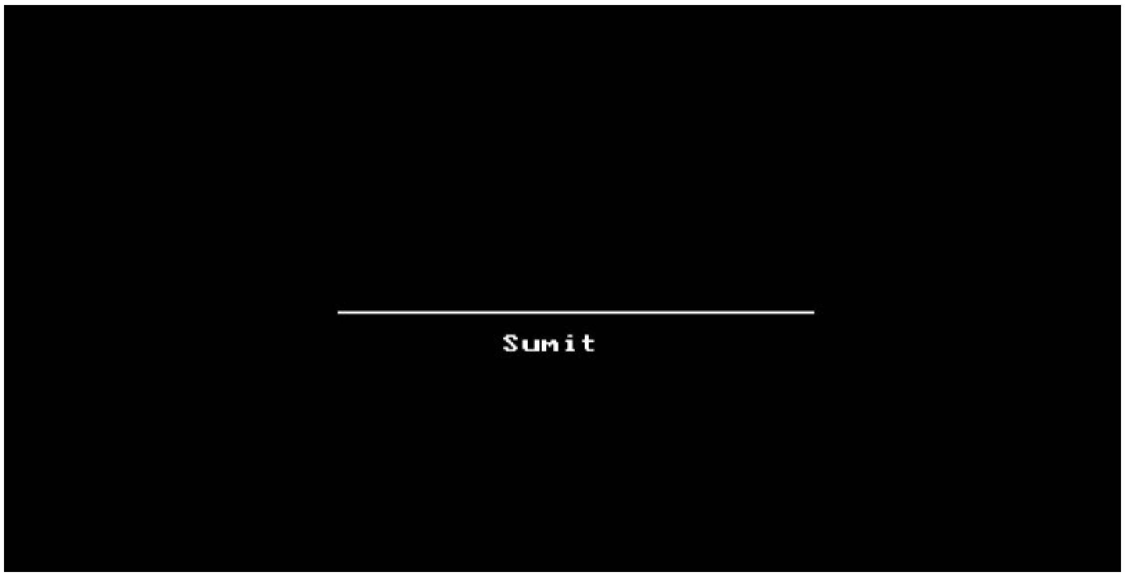
```
    cout << "Enter the x-coordinate of the ending point: ";
  cin >> x2;

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

    initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");
   dda_line(x1, y1, x2, y2);

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

Output:



```
Sumit
```

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.