Experiment 2

Student Name: Dhruv Sharma **UID: 22BCS13922**
Branch: BE-CSE **Section/Group: 22BCS_IOT-612/B**
Semester: 6 **Date of Performance: 15/01/2025**
Subject Name: Computer Graphics Lab **Subject Code: 22CSH-352**

1. Aim:

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

2. Objective:

The objective of this practical is to implement and compare the performance of Simple DDA, Symmetrical DDA, and Bresenham's line-drawing algorithms for lines with both positive and negative slopes. The comparison focuses on computational efficiency, accuracy, and their ability to render lines on a raster display.

3. 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.

4. Implementation/Code:

a) DDA:

```
#include <iostream.h>
#include <graphics.h>
#include <conio.h>
#include <math.h>
#include <dos.h>
#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, yinc, x = x1, y = y1;
   xinc = dx / (float)length;
   yinc = dy / (float)length;
   putpixel(round(x), round(y), 15);
   for (int k = 1; k <= length; k++)
   {
      x = x + xinc;
      y = y + yinc;
      putpixel(round(x), round(y), 15);
      delay(100);
   }
}
void main()
{
```

```
    clrscr();
    int x1, y1, x2, y2;
    int gd = DETECT, gm;
    cout << "Enter the x-coordinate of starting point: ";
    cin >> x1;
    cout << "Enter the y-coordinate of starting point: ";
    cin >> y1;
    cout << "Enter the x-coordinate of ending point: ";
    cin >> x2;
    cout << "Enter the y-coordinate of ending point: ";
    cin >> y2;
    initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");
    dda_line(x1, y1, x2, y2);
    getch();
    closegraph();
}
```

b) Using Symmetrical DDA:

```
#include <conio.h>
#include <iostream.h>
#include <graphics.h>
#include <dos.h>
#include <math.h>
#define ROUND(a) ((int)(a + 0.5))
void symDDA(int xa, int ya, int xb, int yb)
{
    int dx = xb - xa, dy = yb - ya;
    float length;
    float xinc, yinc, x = xa, y = ya;
    if (abs(dx) > abs(dy))
        length = abs(dx);
    else
        length = abs(dy);
    float n = log10(length) / log10(2);
    xinc = dx / (pow(2, n));
    yinc = dy / (pow(2, n));
    putpixel(ROUND(x), ROUND(y), 15);
    delay(50);
    for (int i = 0; i < length; i++)
    {
```

```
        x = x + xinc;
        y = y + yinc;
        putpixel(ROUND(x), ROUND(y), 15);
        delay(50);
    }
}
void main()
{
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "C:\\Turboc3\\BGI");
    int xa, xb, ya, yb;
    cout << "Enter the points (xa ya xb yb): ";
    cin >> xa >> ya >> xb >> yb;
    cleardevice();
    symDDA(xa, ya, xb, yb);
    getch();
    closegraph();
}
```

c) Using bresenham's algorithm

```
#include <iostream.h>
#include <conio.h>
#include <graphics.h>
#include <math.h>
#include <dos.h>
int sign(int x)
{
    if (x < 0)
        return -1;
    if (x > 0)
        return 1;
    return 0;
}
void lineBres(int xa, int ya, int xb, int yb)
{
    int sx, sy, t, length, flag;
    int x = xa, y = ya;
    int dx = abs(xa - xb), dy = abs(ya - yb);
    sx = sign(xb - xa);
    sy = sign(yb - ya);
```

```
        if (dy > dx)
        {
            t = dx;
            dx = dy;
            dy = t;
            length = dy;
            flag = 1;
        }
        else
        {
            length = dx;
            flag = 0;
        }
        int p = (2 * dy) - dx;
        int twoDx = 2 * dx, twoDy = 2 * dy;
        putpixel(x, y, 15);
        delay(50);
        for (int i = 0; i < length; i++)
        {
            while (p > 0)
            {
                if (flag == 1)
                    x = x + sx;
                else
                    y = y + sy;
                p = p - twoDx;
            }
            if (flag == 1)
                y = y + sy;
            else
            {
                x = x + sx;
                p = p + twoDy;
            }
            putpixel(x, y, 15);
            delay(50);
        }
    }
    void main()
```

```
{
   int gd = DETECT, gm;
   initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");
   int xa, ya, xb, yb;
   cout << "Enter the starting point of x: ";
   cin >> xa;
   cout << "Enter the starting point of y: ";
   cin >> ya;
   cout << "Enter the ending point of x: ";
   cin >> xb;
   cout << "Enter the ending point of y: ";
   cin >> yb;
   cleardevice();
   lineBres(xa, ya, xb, yb);
   getch();
   closegraph();
}
```

5. Output:

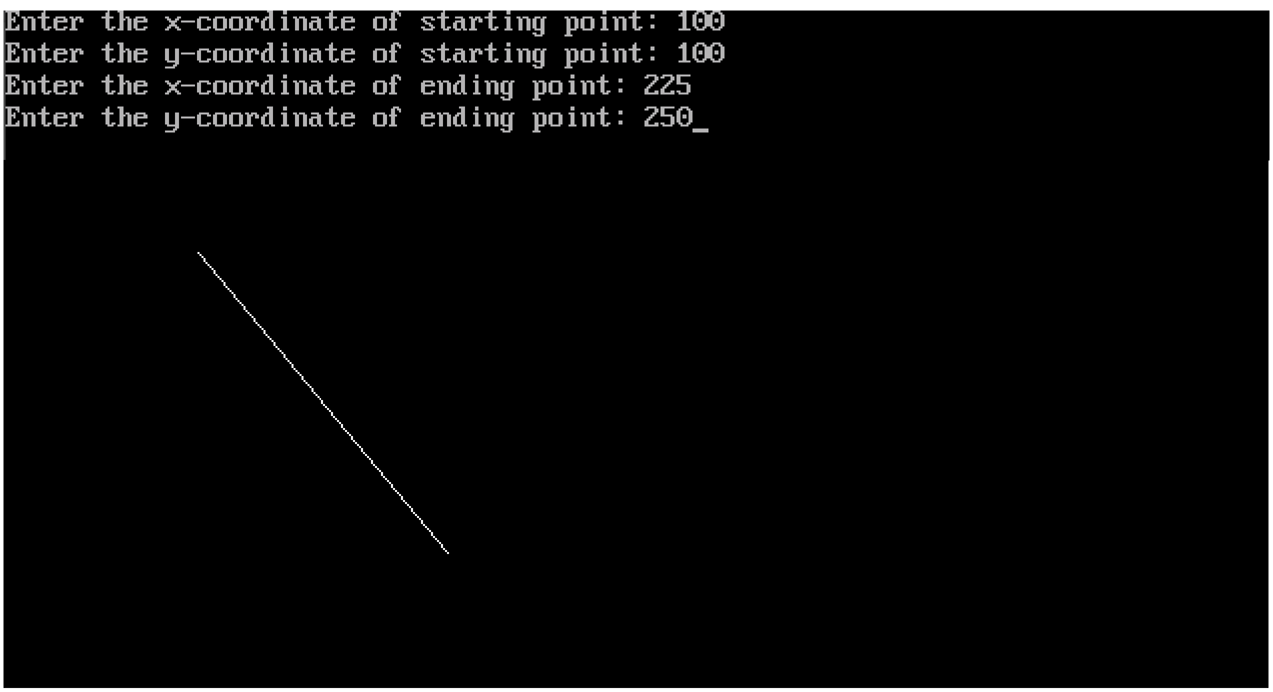

Fig 1: DDA

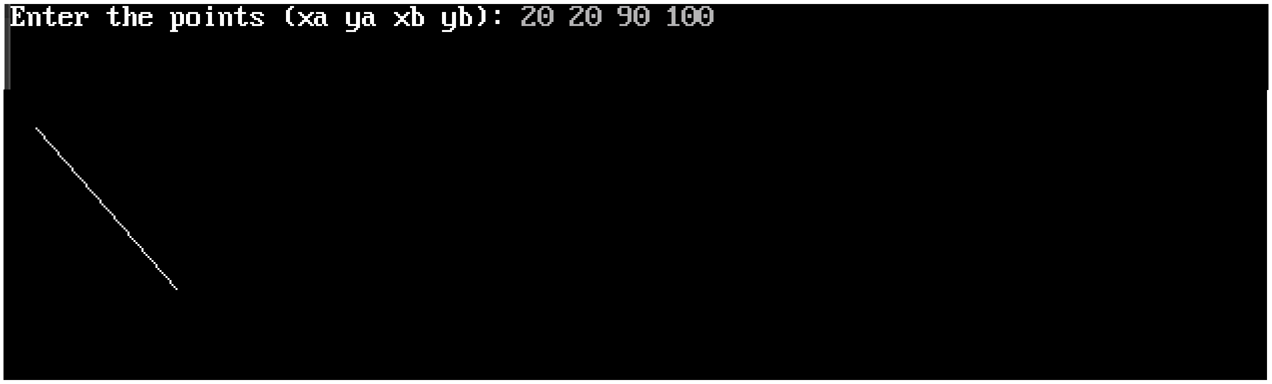

Fig 2: Using Symmetrical DDA

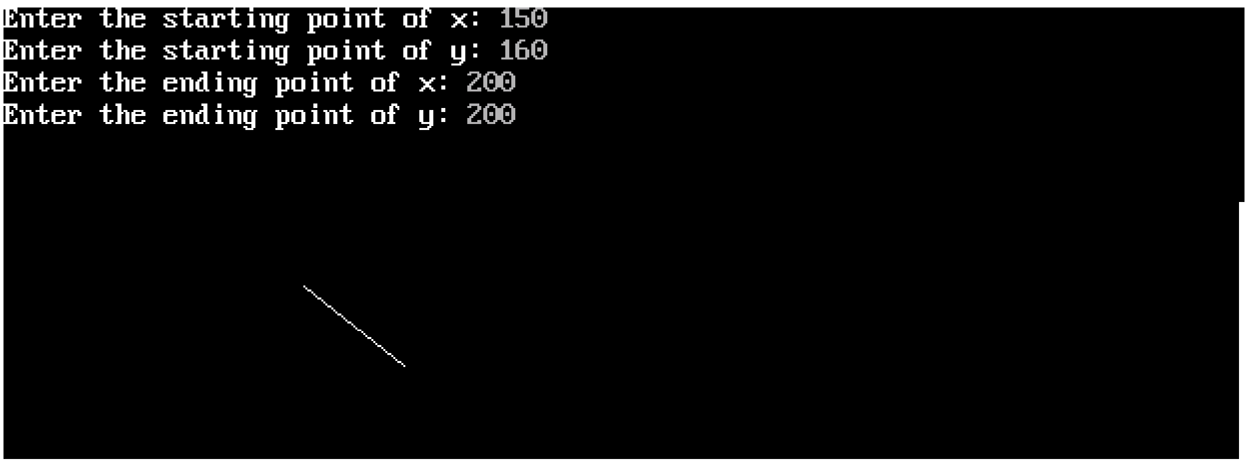

Fig 3: Using bresenham's algorithm

6. Learning Outcome:

- Simple DDA is a basic algorithm that uses floating-point calculations to plot points along the line. It helps in understanding the concept of line generation and the trade-offs between accuracy and performance.
- Symmetrical DDA optimizes the DDA algorithm by reducing floating-point operations, which improves efficiency without compromising the accuracy of the plotted line.
- Bresenham's Algorithm is the most efficient algorithm among the three, utilizing only integer calculations. It provides the highest performance for line drawing, especially when dealing with large lines or real-time graphics.