Name: Nalawade Rohan Arun

Roll No: 231012

PRN: 22310407

Class: SY IT A1

Assignment 2

Aim : Implementation of the DDA Line Algorithm and Bresanham's Line drawing algorithm .

1.Digital Differential Analyzer (DDA):

The DDA algorithm utilizes floating-point arithmetic to incrementally calculate and plot points along a line between two given endpoints. It is relatively straightforward to implement and understand, making it an excellent choice for educational purposes and applications where computational resources are abundant.

Advantages:

- Simplicity: Easy to implement and understand.
- Flexibility: Can handle lines of any slope efficiently.

Disadvantages:

- Floating-Point Operations: Requires floating-point arithmetic, which can be slower and less precise on some hardware.
- Rounding Errors: Accumulation of rounding errors can affect the accuracy of the plotted line.

Algorithm:

Step 1:

Input the two end points (X0 , Y0) & (X1 , Y1)

Step 2:

Calculate the difference between the two end points .

dx = X1 – X2

dy = Y1 – Y2

Step 3:

if(dx > dy) then you need more steps in X co-ordinate , otherwise in Y co-ordinate

```
If( dx > dy ){
        Steps = absolute( dx ) ;
}
Else{
        Steps = absolute( dy) ;
```

}

Step 4:

Calculate the increment in X coordinate and in Y coordinate as follows ,

Xincrement = dx / Steps

Yincrement = dy / Steps

Step 5:

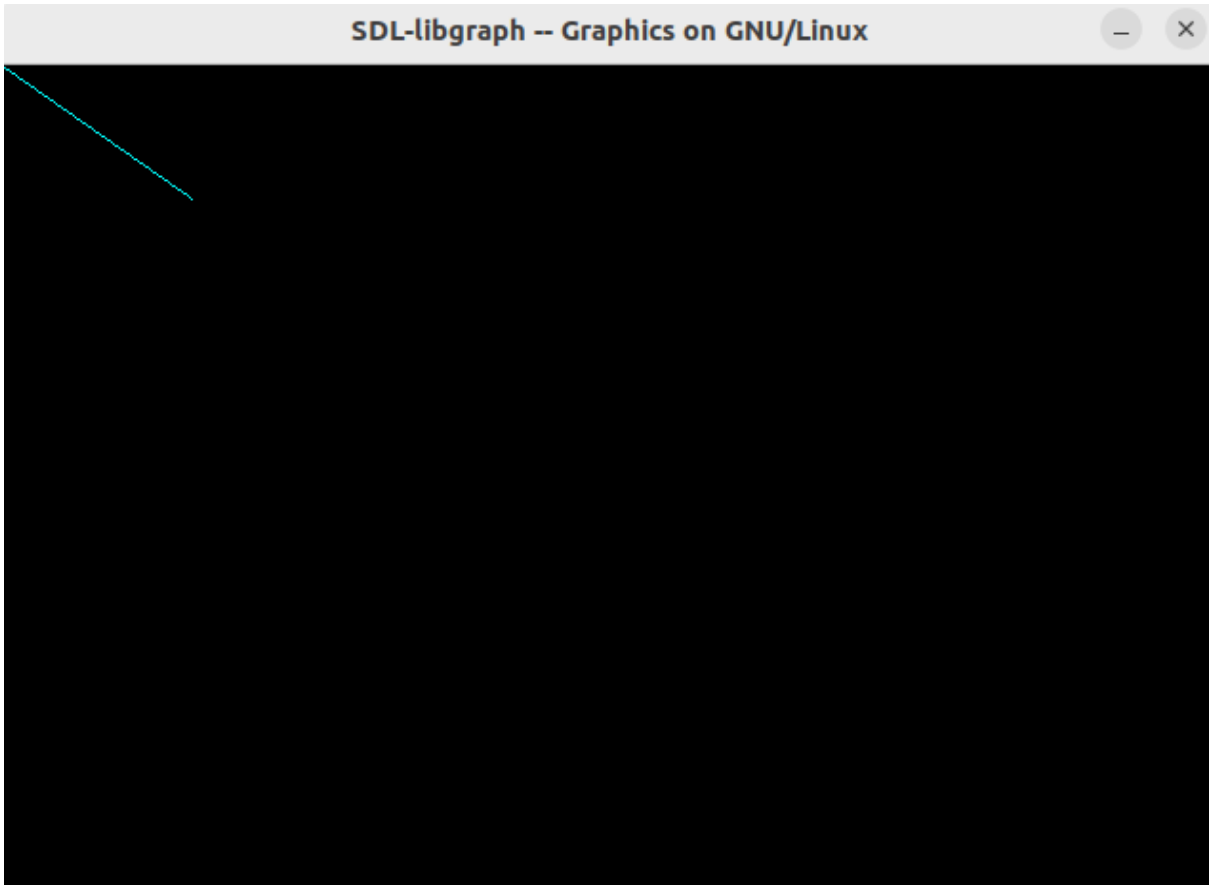
Put the pixels by successfully incrementing X & Y coordinates accordingly & complete drawing of the line .

Implementation:

```
#include <graphics.h>
#include <iostream>
#include <math.h>
using namespace std ;

int main() {
    float x1 , x2 , y1 , y2 , dx , dy , steps , Xinc , Yinc , x , y;
    int i,gd=DETECT,gm;

    initgraph(&gd,&gm , NULL);

    cout<<"Enter value of x1 and y1" ;
    cin>>x1 ;
    cin>>y1 ;

    cout<<"Enter value of x2 and y2 ";
    cin>>x2 ;
    cin>>y2 ;

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

    if(dx > dy){
       steps = abs(dx);
    }
    else{
       steps = abs(dy);
    }

    dx = dx/steps ;
    dy = dy/steps ;

    for(int v = 0 ; v<=steps ; v++){
       putpixel(x , y , 3);
       x = x + dx ;
```

```
        y = y + dy ;
        delay(100);
    }

    getch() ;
    closegraph() ;
    return 0 ;

}
```

Output:



2 . Bresanham's Line drawing algorithm:

The Bresenham line algorithm, on the other hand, employs integer arithmetic to determine the points along a line. By using only integer calculations, it achieves greater efficiency and accuracy, particularly on hardware where floating-point operations are expensive.

Advantages:

- Efficiency: Uses only integer arithmetic, making it faster and more efficient.
- Accuracy: Reduces the accumulation of rounding errors, resulting in more accurate line rendering.
- Hardware Optimization: Well-suited for hardware implementation due to its use of simple integer operations.

Disadvantages:

- Complexity: More complex to understand and implement compared to DDA.
- Special Cases: Requires handling of special cases, such as steep lines, differently.

Algorithm :

Step 1:

Input the two endpoints (X1 , Y1) & (X2 , Y2)

Step 2:

Plot the point (X1 , Y1)

Step 3:

Calcualte the constants dx , dy ,

Step 4:

The initial decision parameter will be P = 2dy – dx

Step 5:

At each point Xk along the line starting at k = 0 perform the following test ,

If(Pk < 0) the next point to be plot is (Xk + 1 , Y) and update the Pk as

Pk+1 = Pk + 2dy ,

Otherwise next point to be plot is (Xk + 1 , Yk + 1) and update Pk as

Pk + 1 = Pk + 2dy – 2dx

Step 5:

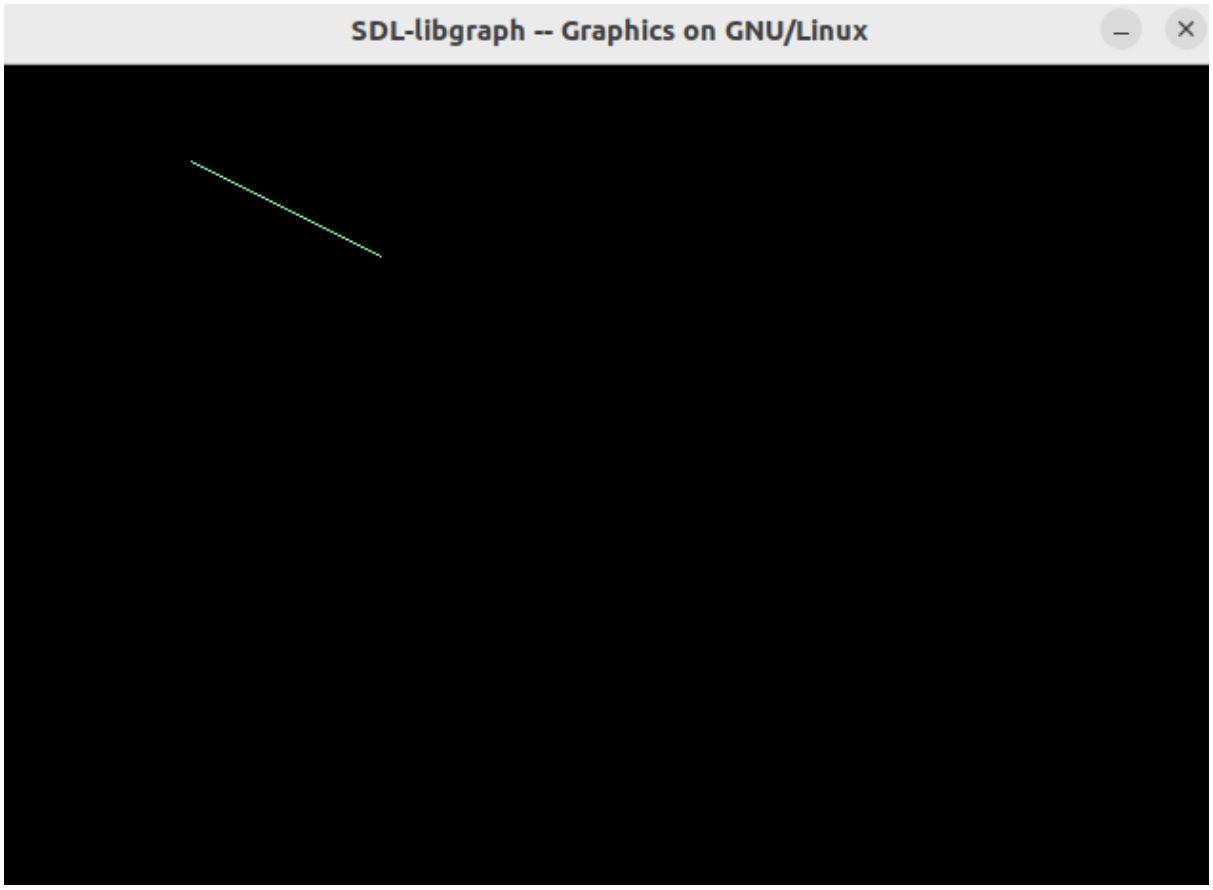
Repeat the step 4 (dx – 1) times **.**

Implementation:

```
#include <graphics.h>
#include <iostream>
#include <math.h>
using namespace std ;

class Bresanham {
      int x1 , y1 , x2 , y2 , dx , dy , x , y , P , step;

      void input () {
          cout<<"Enter the values of x1 and y1";
    cin>>x1 ;
    cin>>y1 ;
    cout<<"Enter the values of x2 and y2";
    cin>>x2 ;
    cin>>y2 ;
      }

      void drawLine() {
          dx = x2 - x1 ;
          dy = y2 - y1 ;
          P = 2*dy - dx ;
          step = x2 - x1 - 1 ;
          x = x1 ;
          y = y1 ;

         for(int i = 0 ; i<=step ; i++){
    if(P < 0){
      putpixel(x,y,10);
      P = P + 2*dy ;
      x = x + 1 ;
    }
    else{
      putpixel(x,y,10);
      P = P = 2*dy - 2*dx ;
      y = y + 1 ;
      x = x + 1 ;
    }

    delay(100);
          }
          }
      public : void run(){
         input() ;
         drawLine() ;
      }
};
```

```
int main() {
    int i,gd=DETECT,gm;
    initgraph(&gd,&gm , NULL);
    Bresanham ob ;
    ob.run();
    getch() ;
    closegraph() ;
    return 0 ;
}
```

Output:



Conclusion:

Both the DDA and Bresenham line algorithms have their unique strengths and weaknesses. The DDA algorithm is beneficial for its simplicity and straightforward implementation, making it suitable for educational purposes and environments where performance is not a critical concern. In contrast, the Bresenham algorithm excels in performance and precision, making it the preferred choice for real-time applications and environments with limited computational resources.

Ultimately, the choice between the two algorithms depends on the specific requirements of the application, such as the need for speed, accuracy, and simplicity. Understanding both algorithms provides a solid foundation for tackling a variety of line drawing challenges in computer graphics.