Vidyavardhini's College of Engineering & Technology

Department of Artificial Intelligence and Data Science

Experiment No. 2
Topic: Bresenham's Algorithm
Name: Vinith Shetty
Roll Number: 55
Date of Performance:
Date of Submission:

Experiment No. 2

Aim: To implement Bresenham's algorithms for drawing a line segment between two given end points.

Objective:
Draw a line using Bresenham's line algorithm that determines the points of an n-dimensional raster that should be selected to form a close approximation to a straight line between two points

Theory:
In Bresenham's line algorithm pixel positions along the line path are obtained by determining the pixels i.e. nearer the line path at each step.

Algorithm –

Step1: Start Algorithm
Step2: Declare variable $x_1,x_2,y_1,y_2,d,i_1,i_2,dx,dy$
Step3: Enter value of x_1,y_1,x_2,y_2
Where $x_1,y_1$are coordinates of starting point
And x_2,y_2 are coordinates of Ending point
Step4: Calculate $dx = x_2-x_1$
Calculate $dy = y_2-y_1$

```
            Calculate $i_1$=2*dy
            Calculate $i_2$=2*(dy-dx)
            Calculate d=$i_1$-dx
Step5: Consider (x, y) as starting point and $x_{end}$ as maximum possible value of x.
            If dx < 0
                  Then x = $x_2$
                  y = $y_2$
                    $x_{end}$ = $x_1$
            If dx > 0
               Then x = $x_1$
            y = $y_1$
                  $x_{end}$ = $x_2$
Step6: Generate point at (x,y)coordinates.
Step7: Check if whole line is generated.
            If x > = $x_{end}$
            Stop.
Step8: Calculate co-ordinates of the next pixel
            If d < 0
               Then d = d + $i_1$
            If d ≥ 0
        Then d = d + $i_2$
            Increment y = y + 1
Step9: Increment x = x + 1
Step10: Draw a point of latest (x, y) coordinates
Step11: Go to step 7
Step12: End of Algorithm
```

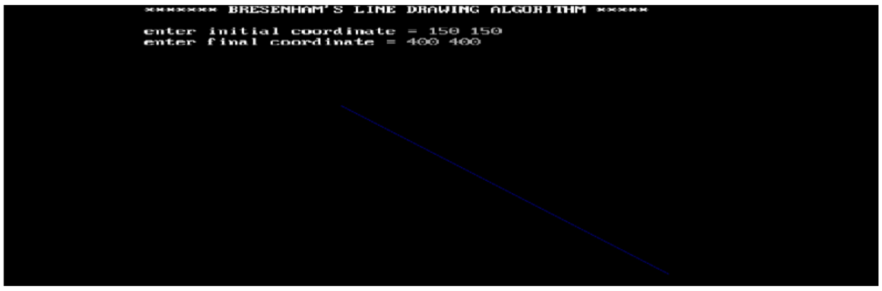
Program –

```
#include<graphics.h>
#include<stdio.h>
void main()
{
int x,y,x1,y1,delx,dely,m,grtr_d,smlr_d,d;
int gm,gd=DETECT;
initgraph(&gd,&gm,"C:\\TC\\BGI");
printf("******* BRESENHAM'S LINE DRAWING ALGORITHM *****\n\n");
printf("enter initial coordinate = ");
scanf("%d %d",&x,&y);
printf("enter final coordinate = ");
scanf("%d %d",&x1,&y1);
delx=x1-x;
dely=y1-y;
grtr_d=2*dely-2*delx;&nbsp; &nbsp;// when d > 0
smlr_d=2*dely;&nbsp; &nbsp; &nbsp; &nbsp; &nbsp; // when d< 0
d=(2*dely)-delx;

do{
```

```
putpixel(x,y,1);
if(d<0) {
d=smlr_d+d;
}
else
{
d=grtr_d+d;
y=y+1;
}
x=x+1;
}while(x<x1);
getch();
}
```

OUTPUT:



```
******** BRESENHAM'S LINE DRAWING ALGORITHM *****

enter initial coordinate = 150 150
enter final coordinate = 400 400
```

Conclusion: Comment on -

1. Pixel
2. Equation for line
3. Need of line drawing algorithm
4. Slow or fast

Pixel: The Bresenham's algorithm is utilized to determine the most appropriate pixels to render a line between two given points in a manner that minimizes computational effort. It refers to the smallest controllable element on a digital display or raster image. In the Bresenham line drawing algorithm, a pixel is represented by its coordinates (x, y) on a two-dimensional plane.

Equation for line: The equation for a line in Bresenham's algorithm doesn't rely on a classic y = mx + c form as in the DDA (Digital Differential Analyzer) algorithm. Instead, the algorithm utilizes incremental calculations to decide which pixels to plot in order to create a line between two specified points (x1, y1) and (x2, y2). The algorithm tracks the error at each step and selects the pixel that best fits the line, making efficient use of integer calculations.

Need for line drawing algorithm: Bresenham's line drawing algorithm is crucial for various applications that require efficient, accurate, and optimized rendering of lines on digital displays, making it a fundamental component of many graphical and computational systems.

Slow or fast: Bresenham's line drawing algorithm is generally considered fast and efficient in comparison to other line drawing algorithms, especially when compared to the Digital Differential Analyzer (DDA) algorithm. The algorithm's ability to determine the next best pixel based on the incremental error calculation further contributes to its speed. It chooses the pixels that most closely approximate the line path, minimizing error and ensuring an efficient rendering process.