Experiment No. 2
Topic : Bresenham's algorithms
Name: Lavanya Murudkar
Roll Number: 31
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
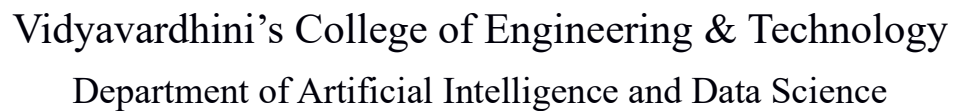
Then $x = x_2$ $y = y_2$

$x_{end} = x_1$ 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 –

```c
#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;   // when d > 0
smlr_d=2*dely;          // 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:
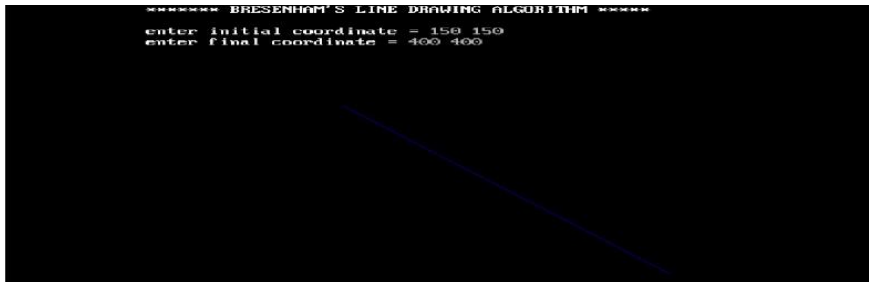
Conclusion: Comment on 1.
                    Pixel
            2. Equation for line
            3. Need of line drawing algorithm
            4. Slow or fast

Pixel: In the context of Bresenham's line drawing algorithm, a pixel refers to the smallest unit of display in a digital raster display. The algorithm is primarily used for drawing lines on a pixel grid. Bresenham's algorithm efficiently determines which pixels to illuminate in order to represent a straight line between two points on a discrete two-dimensional grid.

Equation for line: Bresenham's algorithm is specifically designed for drawing lines on a computer screen or any two-dimensional grid system. Unlike the traditional line drawing equation ($y = mx + c$) which involves floating-point arithmetic and rounding, Bresenham's algorithm uses integer arithmetic and bit manipulation to draw lines. The algorithm works based on the decision between two candidate pixels at each step, avoiding the need for floating-point calculations.

Need for line drawing algorithm: Drawing a line between two points is a fundamental operation in computer graphics and image processing. Bresenham's algorithm is particularly useful for drawing lines efficiently on digital displays, especially in older hardware where floating-point operations were computationally expensive. It is essential for the efficient and fast rendering of lines on digital screens, contributing significantly to the development of computer graphics.

Slow or fast: Bresenham's algorithm is considered fast and efficient compared to other algorithms for drawing lines. It optimizes the process of determining which pixels to illuminate to create a straight line between two points. Since it uses integer arithmetic and bitwise operations instead of floating-point arithmetic, it is more suitable for hardware implementations and significantly speeds up the line drawing process. However, it's important to note that the efficiency of the algorithm can be impacted by the hardware's capabilities and the complexity of the line being drawn