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 $d \geq 0$
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>
#include<conio.h>


int main()
{
        int x,y,x1,y1,x2,y2,p,dx,dy;
        int gd=DETECT,gm=0;
        initgraph(&gd,&gm, "");
        printf("\n Enter x1 cordinate: ");
        scanf("%d",&x1);
        printf("\n Enter y1 cordinate: ");
        scanf("%d",&y1);
        printf("\n Enter x2 cordinate: ");
        scanf("%d",&x2);
        printf("\n Enter y2 cordinate: ");
        scanf("%d",&y2);

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

        putpixel (x,y, RED);
        p = (2 * dy-dx);

        while(x <= x2)
        {
```

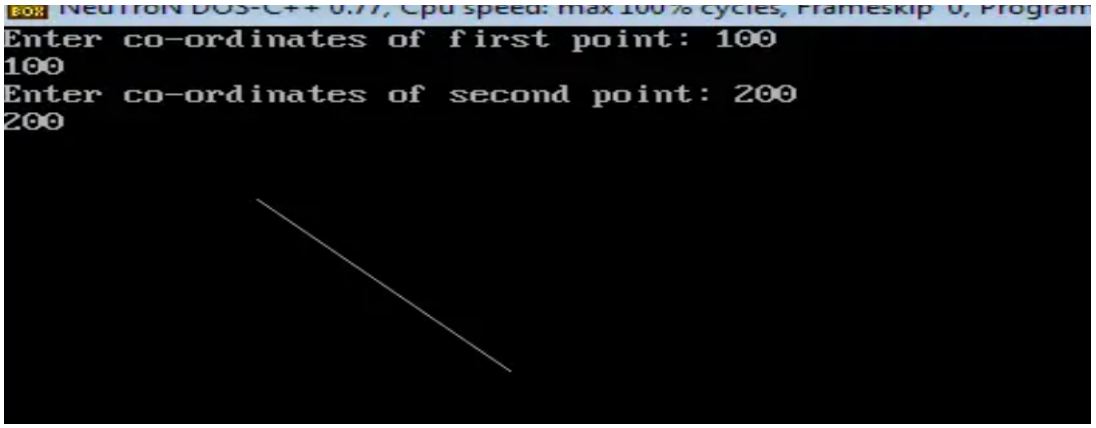
```
            if(p<0)
            {
                  x = x+1;
                  p = p + 2*dy;
            }
            else
            {
                  x = x + 1;
                  y = y + 1;
                  p = p + (2 * dy) - (2 * dx);

            }
            putpixel (x,y, RED);

      }

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

Output –



Conclusion: Comment on

1. Pixel
2. Equation for line

3. Need of line drawing algorithm
4. Slow or fast

1. A pixel, short for "picture element," is the smallest individual unit of a digital image or display, and it is typically represented as a square or rectangular point of light on a screen. In the context of Bresenham's line algorithm in computer graphics, a pixel refers to the discrete points or locations on the screen where the algorithm calculates and decides whether to turn on or off (color or leave blank) in order to draw a straight line.

2. The Bresenham's line algorithm is used to draw a straight line between two points (x0, y0) and (x1, y1) on a pixel grid. The basic equation for this algorithm is:
y = mx + b
Where:
- `y` and `x` are the coordinates of the pixel being considered.
- `m` is the slope of the line, which is calculated as `(y1 - y0) / (x1 - x0)`.
- `b` is the y-intercept, which is determined based on the pixel grid.
The algorithm calculates which pixels to turn on or off along the line by iterating through the x-coordinate of pixels from `x0` to `x1` and using the equation to find the corresponding y-coordinate. It then selects the nearest pixel on the grid for that calculated y-coordinate. The algorithm makes a decision based on the distance between the calculated y and the nearest integer y-value to determine which pixel to plot.
The main idea of the Bresenham's algorithm is to choose the pixel that is closest to the ideal line path (i.e., the one with the smallest error), ensuring an efficient and accurate way to render straight lines on a pixel grid. The algorithm's calculations are adjusted to minimize this error, making it one of the most widely used methods for line drawing in computer graphics.

3. Bresenham's line drawing algorithm fulfills several essential needs in computer graphics. Its efficiency, based on integer arithmetic, enables rapid line rendering, critical for real-time applications like video games. This algorithm ensures accuracy by selecting the closest pixel positions, resulting in clean and visually pleasing lines. Its simplicity makes it accessible to a wide range of programmers, while its adaptability accommodates various line types. With a rich historical legacy, Bresenham's algorithm serves as a classic example of how efficient algorithms enhance computer system performance and remains invaluable for creating responsive and realistic visuals in computer graphics and related fields.

4. Bresenham's line drawing algorithm is generally considered fast, especially for its primary purpose, which is to draw straight lines on a pixel grid. It is known for its efficiency, as it uses integer arithmetic and minimizes the number of calculations required to determine which pixels to plot on the line. This efficiency allows the algorithm to render lines quickly and is particularly well-suited for real-time computer graphics applications where speed is crucial, such as in video games and simulations. While there may be faster algorithms for specific cases or with modern hardware acceleration, Bresenham's algorithm remains a reliable and efficient choice for many line drawing tasks.