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

Objective: Draw the line using (vector) generation algorithms which determine the pixels that should be turned ON are called as digital differential analyzer (DDA).It is one of the techniques for obtaining a rasterized straight line. This algorithm can be used to draw the line in all the quadrants.

Theory:
DDA algorithm is an incremental scan conversion method. Here we perform calculations at each step using the results from the preceding step. The characteristic of the DDA algorithm is to take unit steps along one coordinate and compute the corresponding values along the other coordinate. Digital Differential Analyzer (DDA) algorithm is the simple line generation algorithm which is explained step by step here.

Algorithm:(x1,y1,x2,y2)
dx=x2-x1
dy=y2-y1
x=x1
y=y1
m=dy/dx
if abs(m)<1 then
num_of_pixels=abs(dx)
else
num_of_pixels=abs(dy)
end
xi=dx/num_of_pixels
yi=dy/num_of_pixels
putpixel(x,y)
for x=x1 to x2 do
x=x+xi
y=y+yi
end

Program: #include<stdio.h>
#include<graphics.h>
#include<math.h>

void main() {

```
  int gd = DETECT, gm;
  int dx, dy, steps;
  float xinc, yinc, x, y;
  int x1, y1, x2, y2;

  printf("Enter the coordinates of the first point (x1, y1):\n");
  scanf("%d %d", &x1, &y1);

  printf("Enter the coordinates of the second point (x2, y2):\n");
  scanf("%d %d", &x2, &y2);

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

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

  xinc = dx / (float) steps;
  yinc = dy / (float) steps;

  x = x1;
  y = y1;

  initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");

  for (int k = 1; k <= steps; k++) {
    putpixel(round(x), round(y), 4); // 4 is code for color Red
    x = x + xinc;
    y = y + yinc;

  }

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

Output:

Conclusion: Comment on -

1. Pixel:-It involves floating point operation for each pixel
2. Equation for line:-y=mx+c
3. Need of line drawing algorithm:-Does not require any special skill
4. Slow or fast:-it is faster

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.

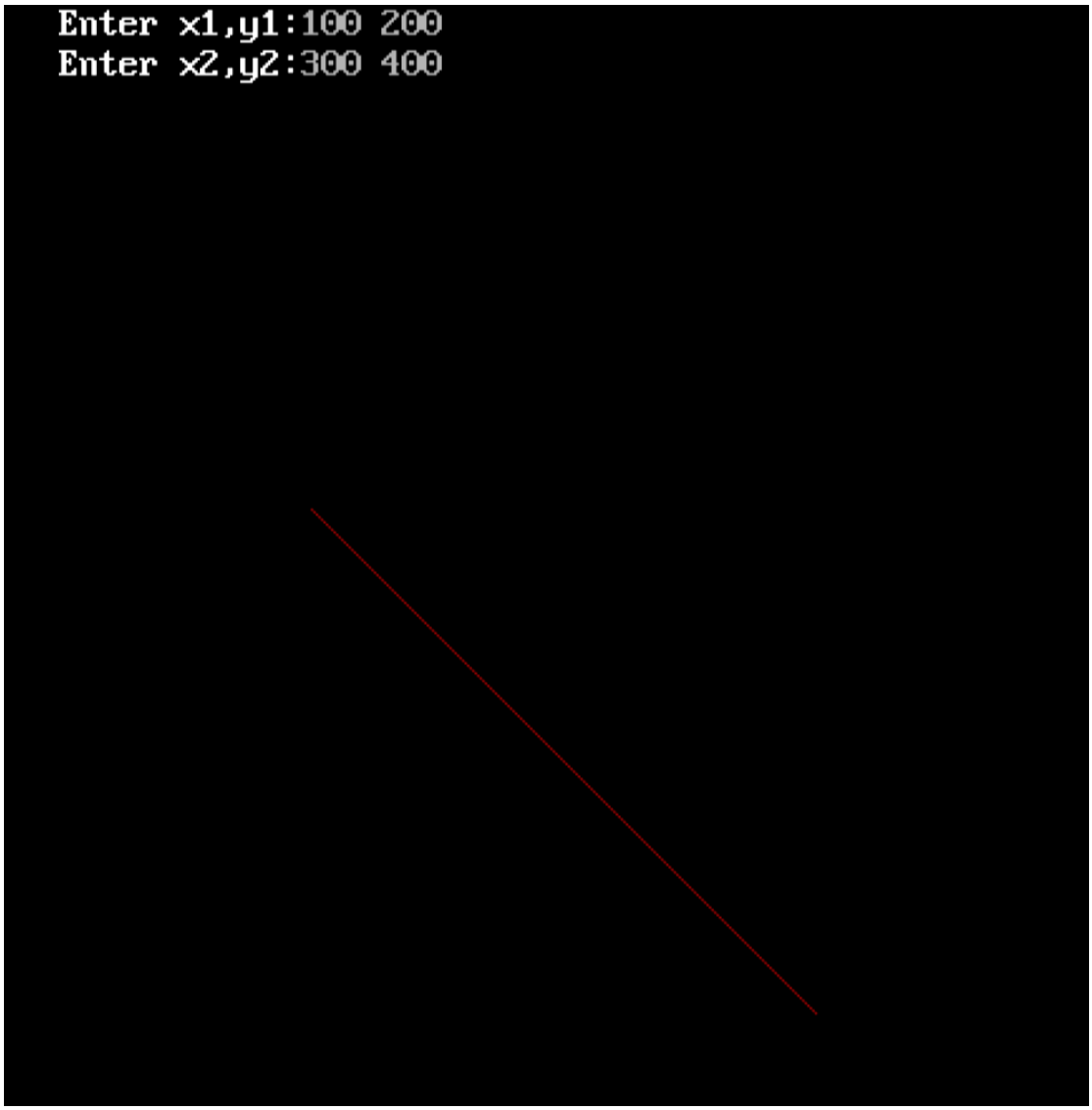
Algorithm –(x1,y1,x0,y0)
dx=x1-x0
dy=y1-y0
p0=2dy-dx
for k=0 to dx do
if pk<0 then
putpixel(xi+1,yi)
pn=pk+2dy
else
putpixel(xi+1,yi+1)
pn=pk+(2dy-2dx)
end
end

Program - #include <stdio.h>
#include <conio.h>
#include <graphics.h>

```
void Bresenham(int x1, int y1, int x2, int y2) {
   int dx, dy, x, y, p, end;
   dx = abs(x1 - x2);
   dy = abs(y1 - y2);
   p = 2 * dy - dx;
   if (x1 > x2) {
      x = x2;
      y = y2;
      end = x1;
   } else {
      x = x1;
      y = y1;
      end = x2;
   }
```

```
    putpixel(x, y, 7);
    while (x < end) {
        x = x + 1;
        if (p < 0) {
            p = p + 2 * dy;
        } else {
            y = y + 1;
            p = p + 2 * (dy - dx);
        }
        putpixel(x, y, 7);
    }
}

int main() {
    int gd = DETECT, gm;
    initgraph(&gd, &gm, RED);

    int x1, y1, x2, y2;
    printf("Enter the coordinates of the first point (x1 y1): ");
    scanf("%d %d", &x1, &y1);
    printf("Enter the coordinates of the second point (x2 y2): ");
    scanf("%d %d", &x2, &y2);

    Bresenham(x1, y1, x2, y2);


    getch();

    closegraph();
    return 0;
}
```

Output –



Conclusion: Comment on -

1. Pixel:-Bresenham's algorithm does not perform any rounding operation
2. Equation for line:-y=mx+c
3. Need of line drawing algorithm:-Involves cheaper operation like addition and subtraction

4. Slow or fast:-It is faster than DDA