Examples for Practice

Example 2.3.4 : *Digitize the line with end-points (10, 12) and (20, 18) using DDA algorithm.*

Example 2.3.5 : *Digitize the line with end-points (0, 3) and (– 7, 9) using DDA algorithm.*

2.3.2 Bresenham's Line Algorithm

- Bresenham's line algorithm uses only integer addition and subtraction and multiplication by 2, and we know that the computer can perform the operations of integer addition and subtraction very rapidly. The computer is also time-efficient when performing integer multiplication by powers of 2. Therefore, it is an efficient method for scan-converting straight lines.
- **The basic principle of Bresenham's line algorithm is to select the optimum raster locations to represent a straight line.** To accomplish this the algorithm always increments either x or y by one unit depending on the slope of line. The increment in the other variable is determined by examining the distance between the actual line location and the nearest pixel. This distance is called **decision variable** or the **error**. This is illustrated in the Fig. 2.3.4.

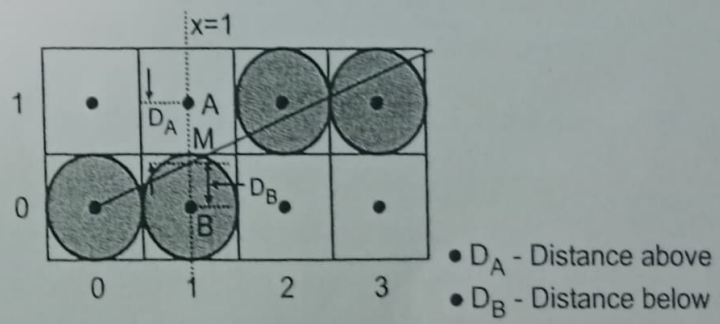


Fig. 2.3.4

- As shown in the Fig. 2.3.4, the line does not pass through all raster points (pixels). It passes through raster point (0, 0) and subsequently crosses three pixels. It is seen that the intercept of line with the line x = 1 is closer to the line y = 0, i.e. pixel (1, 0) than to the line y = 1 i.e. pixel (1, 1). Hence, in this case, the raster point at (1, 0) better represents the path of the line than that at (1, 1). The intercept of the line with the line x = 2 is close to the line y = 1, i.e. pixel (2, 1) than to the line y = 0, i.e. pixel (2, 0). Hence, the raster point at (2, 1) better represents the path of the line, as shown in the Fig. 2.3.4.
- In mathematical terms error or decision variable is defined as,

$$e = D_B - D_A \text{ or } D_A - D_B$$

- Let us define $e = D_B - D_A$. Now if $e > 0$, then it implies that $D_B > D_A$, i.e., the pixel above the line is closer to the true line. If $D_B < D_A$ (i.e. $e < 0$) then we can say that the pixel below the line is closer to the true line. Thus by checking only the sign of error term it is possible to determine the better pixel to represent the line path.
- The error term is initially set as,

$$e = 2\Delta y - \Delta x$$

where $\Delta y = y_2 - y_1$ and $\Delta x = x_2 - x_1$

Then according to value of e following actions are taken...

```
while ( e ≥ 0)
    {
        y = y + 1
        e = e – 2 * Δx
    }
x = x + 1
e = e + 2 * Δy
```

- When $e \geq 0$, error is initialized with $e = e - 2\Delta x$. This is continued till error is negative. In each iteration y is incremented by 1. When $e < 0$, error is initialized to $e = e + 2\Delta y$. In both the cases x is incremented by 1. Let us see the Bresenham's line drawing algorithm.

Bresenham's Line Algorithm for $|m = \Delta y/\Delta x| < 1$

1. Read the line end points (x_1, y_1) and (x_2, y_2) such that they are not equal. [if equal then plot that point and exit]
2. $\Delta x = |x_2 - x_1|$ and $\Delta y = |y_2 - y_1|$
3. [Initialize starting point]
   $x = x_1$
   $y = y_1$
   Plot (x, y)

4. $e = 2 * \Delta y - \Delta x$
   [Initialize value of decision variable or error to compensate for nonzero intercepts]
5. $i = 1$
   [Initialize counter]
6. while ($e \geq 0$)
   {
   $y = y + 1$
   $e = e - 2 * \Delta x$
   }
   $x = x + 1$
   $e = e + 2 * \Delta y$
7. Plot (x, y)
8. $i = i + 1$
9. if ($i \leq x$) then go to step 6.
10. Stop

Program 2.3.2 'C' code for Bresenham's line drawing algorithm

```
#include<stdio.h>
#include<graphics.h>
#include<math.h>
main()
{
      float x,y,x1,y1,x2,y2,dx,dy,e;
      int i,gd,gm;
      clrscr();

/* Read two end points of line
---------------------------------- */
printf("Enter the value of x1 :\t");
scanf("%f",&x1);
printf("Enter the value of y1 :\t");
scanf("%f",&y1);
printf("Enter the value of x2 :\t");
scanf("%f",&x2);
printf("Enter the value of y2 :\t");
scanf("%f",&y2);
/* Initialise graphics mode
---------------------------------- */
detectgraph(&gd,&gm);
initgraph(&gd,&gm,"");

dx=abs(x2-x1);
dy=abs(y2-y1);
```

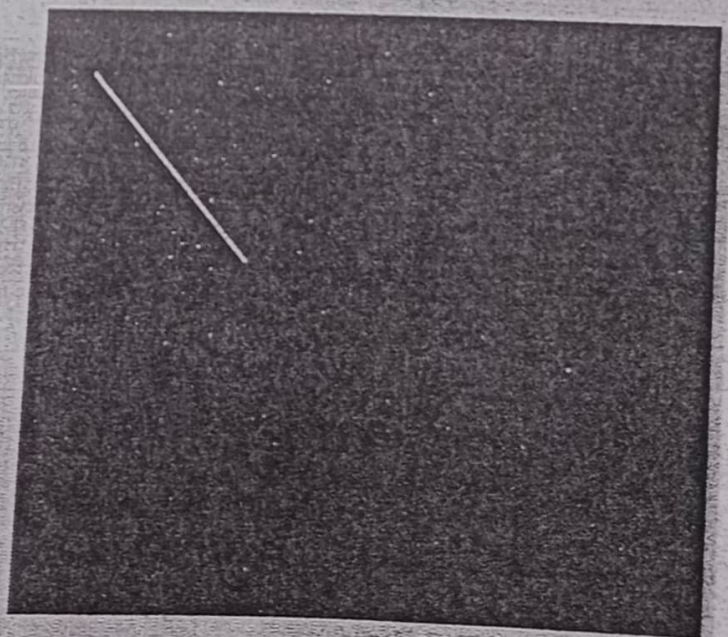
```
    /* Initialise starting point
    ----------------------------*/
    x = x1;
    y = y1;
    putpixel (x, y, 15) ;

    /* Initialise decision variable
    ---------------------------- */
    e = 2 * dy-dx;

    i = 1;   /* Initialise loop counter */
    do
    {
    while(e >= 0)
          {
          y = y + 1;
          e = e - 2 * dx;
          }
          x = x + 1;
          e = e + 2 * dy;
          i = i + 1;
          putpixel (x, y, 15);
 }
 while( i <= dx);
 getch();
 closegraph();
 }
```

Output

Enter the value of x1: 50
Enter the value of y1: 50
Enter the value of x2: 200
Enter the value of y2: 200



16) a) Consider the line from $(5,5)$ to $(13,9)$. Use the Bresenham's algorithm to rasterize the line..

(x_1, y_1 and x_2, y_2)

Soln:-

$\Delta x = x_2 - x_1$

$\Delta x = 13 - 5$

$\Delta x = 8$

$\Delta y = y_2 - y_1$

$\Delta y = 9 - 5$

$\Delta y = 4$

For table

a conditions

$P_k > 0$ $x\uparrow$ $(x+1)$ $y\uparrow$ $(y+1)$

$P_k < 0$ $x\uparrow$ $(x+1)$ y (no change)

$P_0 = 2\Delta y - \Delta x$

$= 2(4) - 8$

$P_0 = 8 - 8$

$P_0 = 0$ $P_0 > 0 \therefore x+1, y+1.$

$P_k > 0 \Rightarrow$

$P_k = P_k + 2\Delta y - 2\Delta x$

$P_1 = P_k + 2\Delta y - 2\Delta x$

$P_1 = P_0 + 2(4) - 2(8)$

$P_1 = 0 + 8 - 16$

$P_1 = -8$ $P_1 < 0 \therefore x+1, y$ no change

$P_k < 0 \Rightarrow$

$P_k = P_k + 2\Delta y$

$P_2 = P_k + 2\Delta y$

$= P_1 + 2(4)$

$P_2 = -8 + 8$

$P_2 = 0$

$P_3 = P_k + 2\Delta y - 2\Delta x$

$= P_2 + 2\Delta y - 2\Delta x$

$= 0 + 2(4) - 2(8)$

$P_3 = 8 - 16 = -8$

$$P_4 = P_k + 2\Delta y$$

$$= P_3 + 2\Delta y$$

$$= -8 + 2(4)$$

$$= -8 + 8$$

$$P_4 = 0$$

$$P_5 = P_k + 2\Delta y - 2\Delta x$$

$$= P_4 + 2(4) - 2(8)$$

$$= 0 + 8 - 16$$

$$P_5 = -8$$

$$P_6 = P_k + 2\Delta y$$

$$= P_5 + 2(4)$$

$$= -8 + 8$$

$$P_6 = 0$$

$$P_7 = P_k + 2\Delta y - 2\Delta x$$

$$= P_6 + 2(4) - 2(8)$$

$$= 0 + 8 - 16$$

$$P_7 = -8$$

Initial value (5, 5)

k	P_k	x	y	Plot (x, y)
0	$P_0 = 0$	$P_0 > 0$ $(x+1)$ $5+1 = 6$	$6^{(y+1)}$	(6, 6)
1	$P_1 = -8$	7	6	(7, 6)
2	$P_2 = 0$	8	7	(8, 7)
3	$P_3 = -8$	9	7	(9, 7)
4	$P_4 = 0$	10	8	(10, 8)
5	$P_5 = -8$	11	8	(11, 8)
6	$P_6 = 0$	12	9	(12, 9)
7	$P_7 = -8$	13	9	(13, 9) → End point

Graph :-