# UNIT 2 GRAPHIC PRIMITIVES

## 2.1 INTRODUCTION

In unit 1 of this block, we have discussed refreshing display devices and its types that are Random Scan display devices and Raster Scan display devices. We have also discussed the limitations of these display devices. In Raster Scan display device, which we have discussed in the previous unit, the picture information is stored in the frame buffer, the concept of frame buffer conveys that the information for the image to be projected on the screen is stored in the form of 0s and 1s, making respective pixels activate and deactivate on the screen, and it is the concept itself which contributes to the discreteness in the picture under display. So, in this unit we are going to extend our discussion to algorithms, which are responsible for actual display of the geometries like line, circle etc., on display devices, such that there is decrease in discreteness and hence, more realistic finished image is the outcome.

## 2.2 OBJECTIVES

After going through this unit, you should be able to:

- describe the Line Generation Algorithm;
- apply Line Generation Algorithm to practical problems;
- describe the different types of Line Generation Algorithm;
- describe Circle Generation Algorithm;
- apply Circle Generation algorithm to practical problems;
- describe the different types of Circle Generation Algorithms;
- describe Polygon fill algorithm, and
- describe Scan Line Polygon fill algorithm.

## 2.3 POINTS AND LINES

In the previous unit, we have seen that in order to draw primitive objects, one has to first scan convert the objects. This refers to the operation of finding out the location of pixels to be intensified and then setting the values of corresponding bits, to the desired intensity level. Each pixel on the display surface has a finite size depending on the screen resolution and hence, a pixel cannot represent a single mathematical point. However, we consider each pixel as a unit square area identified by the coordinate of its lower left corner, the origin of the reference coordinate system being located at the lower left corner of the display surface. Thus, each pixel is accessed by a non-negative integer coordinate pair ($x$, $y$). The x values start at the origin and increase from left to right along a scan line and the y values

(i.e., the scan line numbers) start at bottom and increase upwards.

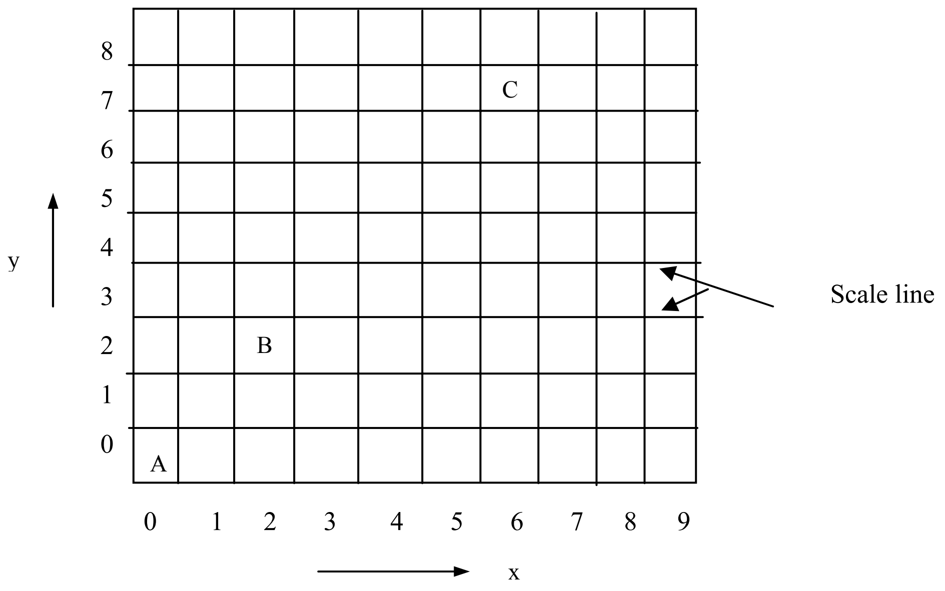


**Figure 1: Scan lines**

*Figure 1*, shows the Array of square pixels on the display surface. Coordinate of pixel A: 0, 0; B: 2, 2; C: 6, 7. A coordinate position (6.26, 7.25) is represented by C, whereas (2.3, 2.5) is represented by B. Because, in order to plot a pixel on the screen, we need to round off the coordinates to a nearest integer. Further, we need to say that, it is this rounding off, which leads to distortion of any graphic image.

Line drawing is accomplished by calculating the intermediate point coordinates along the line path between two given end points. Since, screen pixels are referred with integer values, plotted positions may only approximate the calculated coordinates – i.e., pixels which are intensified are those which lie very close to the line path if not exactly on the line path which is in the case of perfectly horizontal, vertical or 45° lines only. Standard algorithms are available to determine which pixels provide the best approximation to the desired line, we will discuss such algorithms in our next section. Screen resolution however, is a big factor towards improving the approximation. In a high resolution system the adjacent pixels are so closely spaced that the approximated line-pixels lie very close to the actual line path and hence, the plotted lines appear to be much smoother — almost like straight lines drawn on paper. In a low resolution system, the same approximation technique causes lines to be displayed with a “stairstep apperance” i.e., not smooth as shown in *Figure 2*, the effect is known as the stair case effect. We will discuss this effect and the reason behind this defect in the next section of this unit.

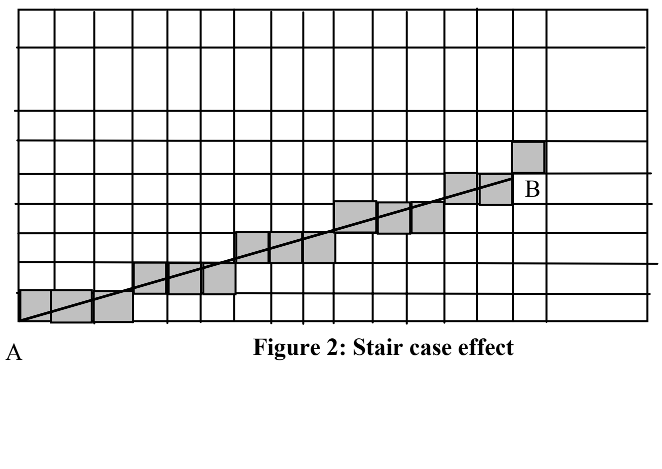


**Figure 2: Stair case effect**

## 2.4 LINE GENERATION ALGORITHMS

- In unit 1, we have discussed the case of frame buffer where information about the image to be projected on the screen is stored in an *m*n* matrix, in the form of 0s and 1s; the 1s stored in an *m* n* matrix positions are brightened on the screen and 0's are not brightened on the screen and this section which may or may not be brightened is known as the Pixel (picture element). This information of 0s and 1s gives the required pattern on the output screen i.e., for display of information. In such a buffer, the screen is also in the form of *m* n* matrix , where each section or niche is a pixel (i.e., we have *m* n* pixels to constitute the output).

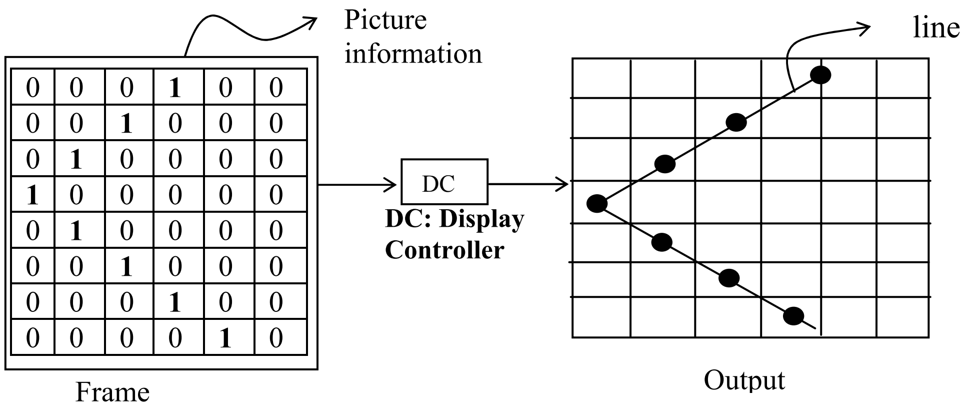


**Figure 3: Basic Line Generation**

Now, it is to be noted that the creation of a line is merely not restricted to the above pattern, because, sometimes the line may have a slope and intercept that its information is required to be stored in more than one section of the frame buffer, so in order to draw or to approximate such the line, two or more pixels are to be made ON. Thus, the outcome of the line information in the frame buffer is displayed as a stair; this effect of having two or more pixels ON to approximating a line between two points say A and B is known as the Staircase effect. **The concept is shown below in *Figure 4.***

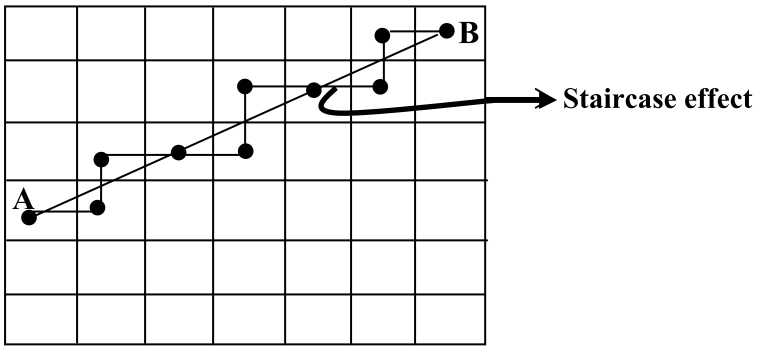


**Figure 4: Staircase effect**

So, from the *Figure 4*, I think, it is clear to you that when a line to be drawn is simply described by its end points, then it can be plotted by making close approximations of the pixels which best suit the line, and this approximation is responsible for the staircase effect, which miss projects the information of the geometry of line stored in the frame buffer as a stair. This defect known as Staircase effect is prominent in DDA Line generation algorithms, thus, to remove the defect Bresenham line generation Algorithm was introduced. We are going to discuss DDA (Digital Differential Analyser) Algorithm and Bresenham line generation Algorithm next.

### 2.4.1 DDA (Digital Differential Analyser) Algorithm

From the above discussion we know that a Line drawing is accomplished by calculating intermediate point coordinates along the line path between two given end points. Since screen pixels are referred with integer values, or plotted positions, which may only approximate the calculated coordinates – i.e., pixels which are intensified are those which lie very close to the line path if not exactly on the line path which in this case are perfectly horizontal, vertical or 45° lines only. Standard algorithms are available to determine which pixels provide the best approximation to the desired line, one such algorithm is the DDA (Digital Differential Analyser) algorithm. Before going to the details of the algorithm, let us discuss some general appearances of the line segment, because the respective appearance decides which pixels are to be intensified. It is also obvious that only those pixels that lie very close to the line path are to be intensified because they are the ones which best approximate the line. Apart from the exact situation of the line path, which in this case are perfectly horizontal, vertical or 45° lines (i.e., slope zero, infinite, one) only. We may also face a situation where the slope of the line is > 1 or < 1.Which is the case shown in *Figure 5*.

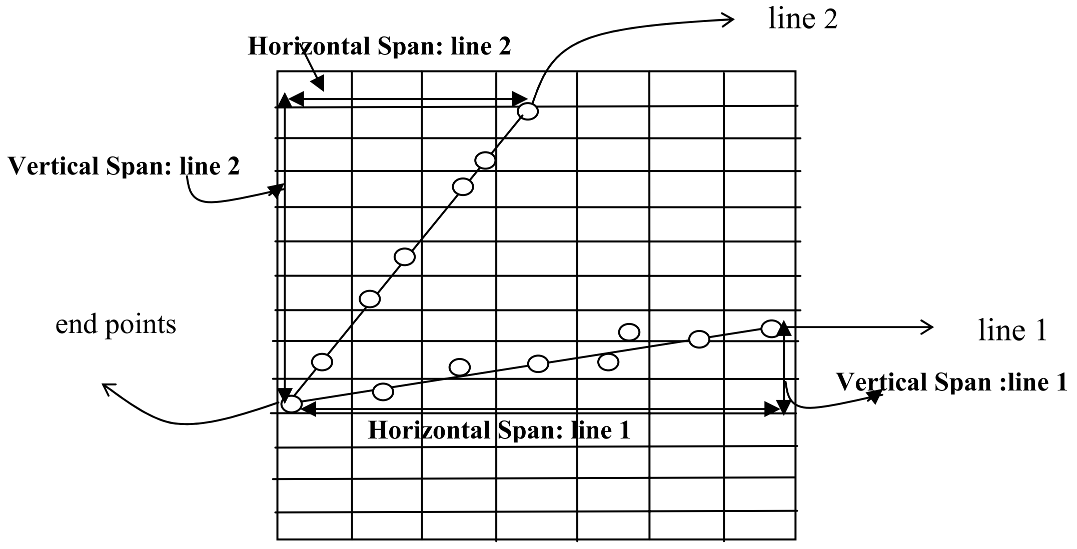


**Figure 5: DDA line generation**

In *Figure 5*, there are two lines. Line 1 (slope<1) and line 2 (slope>1). Now let us discuss the general mechanism of construction of these two lines with the DDA algorithm. As the slope of the line is a crucial factor in its construction, let us consider the algorithm in two cases depending on the slope of the line whether it is > 1 or < 1.

**Case 1: slope (*m*) of line is < 1 (i.e., line 1):** In this case to plot the line we have to move the direction of pixel in *x* by 1 unit every time and then hunt for the pixel value of the *y* direction which best suits the line and lighten that pixel in order to plot the line.

So, in Case 1 i.e., $0 < m < 1$ where *x* is to be increased then by 1 unit every time and proper *y* is approximated.

**Case 2: slope (*m*) of line is > 1 (i.e., line 2)** if $m > 1$ *i.e.*, case of line 2, then the most appropriate strategy would be to move towards the *y* direction by 1 unit every time and determine the pixel in *x* direction which best suits the line and get that pixel lightened to plot the line.

So, in Case 2, i.e., (infinity) > $m$ > 1 where $y$ is to be increased by 1 unit every time and proper $x$ is approximated.

**Assumption:** The line generation through DDA is discussed only for the I st Quadrant, if the line lies in any other quadrant then apply respective transformation (generally reflection transformation), such that it lies in I st Quadrant and then proceed with the algorithm, also make intercept Zero by translational transformation such that $(x_i, y_i)$ resolves to $(x_i, mx_i + c)$ or $(x_i, mx_i)$ and similar simplification occurs for other cases of line generation. The concept and application of transformations is discussed in Block 2 of this course.

**Note:**

1) If in case 1, we plot the line the other way round i.e., moving in $y$ direction by 1 unit every time and then hunting for $x$ direction pixel which best suits the line. In this case, every time we look for the $x$ pixel, it will provide more than one choice of pixel and thus enhances the defect of the stair case effect in line generation. Additionally, from the *Figure 5*, you may notice that in the other way round strategy for plotting line 1, the vertical span is quite less in comparison to the horizontal span. Thus, a lesser number of pixels are to be made *ON*, and will be available if we increase $Y$ in unit step and approximate $X$. But more pixels will be available if we increase $X$ in unit steps and approximate $Y$ (this choice will also reduce staircase effect distortion in line generation) ($\because$ more motion is to be made along $x$-axis).

2) Consider a line to be generated from $(X_0, Y_0)$ to $(X_1, Y_1)$. If $(X_1 - X_0 > Y_1 - Y_0)$ and $X_1 - X_0 > 0$ then slope ($m$) of line is < 1 hence case 1 for line generation is applicable otherwise case 2, i.e., If $(X_1 - X_0 < Y_1 - Y_0)$ and $X_1 - X_0 > 0$ then slope m > 1 and case 2 of line generation is applicable.
**Important: Assume that $X_1>X_0$ is true else swap $(X_0, Y_0)$ and $(X_1, Y_1)$**

3) When $0 < m < 1$ : increment $X$ in unit steps and approximate $Y$
   - Unit increment should be iterative $\Rightarrow x_i = (x_{i-1}) + 1$ such that $(x_i, y_i)$ resolves to $(x_i, mx_i + c)$ or $(x_i, mx_i)$ . *It is to be noted that while calculating $y_i$, if $y_i$ turned out to be a floating number then we round its value to select the approximating pixel. This rounding off feature contributes to the staircase effect.*

   When (infinity) > $m$ > 1 increment Y in unit steps and approximate X, simplify $(x_i, y_i)$ accordingly.

**Case 1: slope (m) of line is < 1 (or 0 < m < 1)**

Consider a line to be generated from *$(X_0, Y_0)$* to *$(X_1, Y_1)$* , **Assume that *$X_1>X_0$* is true else swap *$(X_0, Y_0)$* and *$(X_1, Y_1)$*.** Now, if $(X_1 - X_0 > Y_1 - Y_0)$ that means slope ($m$) of line is < 1 hence, case 1 for line generation is applicable. Thus, we need to increment $X$ in unit steps and approximate $Y$. So, from $X_0$ to $X_1$ , $x_i$ is incremented in unit steps in the horizontal direction, now for these unit steps the satisfying value of $Y$ can be estimated by the general equation of line $y = mx + c$.

Similarly, for case 2, let us sum up our discussion on DDA algorithm for both cases.

We will examine each case separately.

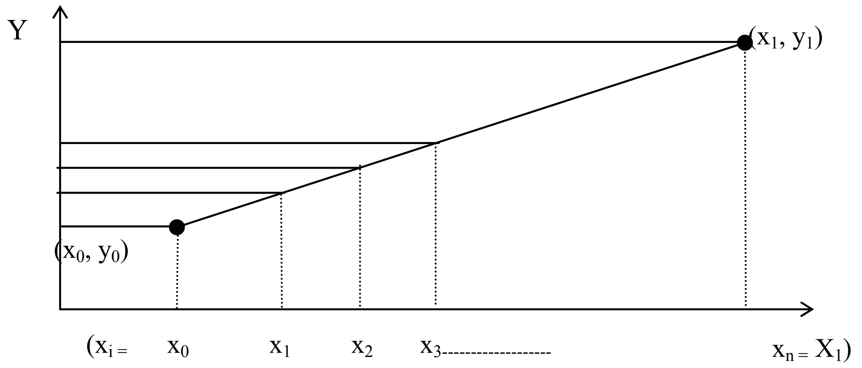


**Figure 6: Slope (*m*) of line is < 1 (i.e., line 1)**

**Sum up:**

For instance, in a given line the end points are $(X_0, Y_0)$ and $(X_1, Y_1)$. Using these end points we find the slope ($m = Y_1 - Y_0 / X_1 - X_0$) and check that the value of $m$ lies between 0 and 1 or is > 1. If 0 < m < 1 then case 1 applies, else, case 2 applies.
For case 1, increase $x$ by one Unit every time, for case 2 increase $y$ by one Unit every time and approximate respective values of $y$ and $x$.

We assume equation of line is $y = mx+c$

$$\left.\begin{array}{lll} \text{At } x = x_i & \text{we have} & y_i = mx_i+c \\ \text{Similarly at } x = x_{i+1} & \text{we have} & y_{i+1} = mx_{i+1}+c \end{array}\right\}$$

**Case 1: Slope (m) of line is 0 < *m*1 (i.e., line 1)**

Since $x$ is to be increase by 1 unit each time

$\Rightarrow x_{i+1} = x_i + 1$ ----------- (1)

So by using equation of line $y = mx+c$ we have

$$\begin{aligned} y_{i+1} &= m\,(x_i + 1) + c \\ &= mx_i + c + m \\ &= y_i + m \qquad \text{------------ (2)} \end{aligned}$$

Equations (1) and (2) imply that to approximate line for case 1 we have to move along $x$ direction by 1 unit to have next value of $x$ and we have to add slope $m$ to initial $y$ value to get next value of $y$.

Now, using the starting point $(x_0, y_0)$ in the above equations (1) and (2) we go for $x_i$ and $y_i$ ($i$ = 1, 2, ……$n$) and put colour to the pixel to be lightened.

**It is assumed that $X_0 < X_1$ $\therefore$ the algorithm goes like:**

```
            $x \leftarrow X_0$
            $y \leftarrow Y_0$
            $m \leftarrow (Y_1 - Y_0)/(X_1 - X_0)$
      while ($x <= X_1$) do
{     put-pixel ($x$, round ($y$), color)
      (new $x$-value) $x \leftarrow$      (old $x$-value) $x + 1$
      (new $y$-axis)  $y \leftarrow$      (old $y$-value) $y + m$
}
```

**Sample execution of algorithm case 1:**

at $(x_0, y_0)$ : put-pixel ($x_0$, $y_0$, colour)
$x_1 = x_0 + 1$; $y_1 = y_0 + m$

at $(x_1, y_1)$ = put pixel $(x_1, y_1$, colour)

similarly, $x_2 = x_{1+}1$; $y_2 = y_{1+}$m
at $(x_2, y_2)$ : put pixel $(x_2, y_2$, colour) and so on.

**Case 2: slope (*m*) of line is > 1 (i.e., line 2)**: Same as case 1 but, this time, the y component is increased by one unit each time and appropriate *x* component is to be selected. To do the task of appropriate selection of *x* component we use the equation of Line: $y = mx+c$.

Unit increment should be iterative $\Rightarrow y_{i+1} = y_i + 1$ ; for this $y_{i+1}$ we find corresponding $x_{i+1}$ by using equation of line $y = mx + c$ and hence get next points $(x_{i+1}, y_{i+1})$.

$\Rightarrow y_{i+1} - y_i = m\,(x_{i+1} - x_i)$ ----------- (3)

as *y* is to be increase by unit steps
$\therefore y_{i+1} - y_i = 1$ ----------- (4)

using equation (4) in (3) we get
$\Rightarrow 1 = m\,(x_{i+1} - x_i)$ ----------- (5)

rearranging (5) we get

$\Rightarrow 1/m = (x_{i+1} - x_i)$

$$\Rightarrow \boxed{x_{i+1} = x_i + \frac{1}{m}}$$

So, procedure as a whole for Case 2 is summed up as Algorithm case 2:

Assuming $Y_0 < Y_1$ the algorithm goes like

**Algorithm for case 2:**

```
        $x \leftarrow X_0$;
        $y \leftarrow Y_0$;
        $m \leftarrow (Y_1 - Y_0)/(X_1 - X_0)$;
        $m_1 \leftarrow 1/m$;
    while ($y < Y_1$) do
{
    put-pixel (round ($x$), $y$, colour)
    $y \leftarrow y + 1$;
    $x \leftarrow x + m_1$;
X
}
```

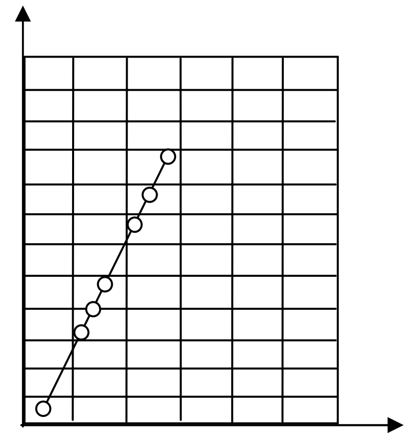


**Figure 7: Slope (*m*) of line is > 1**

**Example 1:** Draw line segment from point (2, 4) to (9, 9) using DDA algorithm.

**Solution:** We know general equation of line is given by

$y = mx+c$ where $m = (y_1 - y_0/(x_1 - x_0)$

given $(x_0, y_0) \rightarrow (2, 4)$ ; $(x_1, y_1) \rightarrow (9, 9)$

$\Rightarrow m = \dfrac{y_1 - y_0}{x_1 - x_0} = \dfrac{9-4}{9-2} = \dfrac{5}{7}$ i.e., $0 < m < 1$

$C = y_1 - mx_1 = 9 - \dfrac{5}{7} \times 9 = \dfrac{63-45}{7} = \dfrac{18}{7}$

So, by equation of line $(y = mx + c)$ we have

$$y = \frac{5}{7}x + \frac{18}{7}$$

| | |
|---|---|
| DDA Algorithm Two case: | |
| Case 1: $m < 1$ | $x_{i+1} = x_i + 1$ |
| | $y_{i+1} = y_i + m$ |
| Case 2: $m > 1$ | $x_{i+1} = x_i + (1/m)$ |
| | $y_{i+1} = y_i + 1$ |

As $0 < m < 1$ so according to DDA algorithm case 1

$x_{i+1} = x_i + 1$ $y_{i+1} = y_i + m$

given $(x_0, y_0) = (2, 4)$

1) $x_1 = x_0 + 1 = 3$

$y_1 = y_0 + m = 4 + \dfrac{5}{7} = \dfrac{28+5}{7} = \dfrac{33}{7} = 4\dfrac{5}{7}$

put pixel ($x_0$, round y, colour)

i.e., put on (3, 5)

2) $x_2 = x_1 + 1 = 3 + 1 = 4$

$y_2 = y_1 + m = (33/7) + \dfrac{5}{7} = 38/7 = 5\dfrac{5}{7}$

put on (4, 5)

Similarly go on till (9, 9) is reached.

## ☞ Check Your Progress 1

1) What do you mean by the term graphic primitives?

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

2) What is the Staircase effect?

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

3) What is the reason behind the Staircase effect?

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

4) How does the resolution of a system affect graphic display?

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

5) Modify the DDA algorithm for negative sloped lines ; discuss both the cases i.e., slope > 1 and 0 < slope < 1.

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

6) Using DDA algorithm draw line segments from point (1,1) to (9,7).

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

### 2.4.2 Bresenham Line Generation Algorithm

Bresenham algorithm is accurate and efficient raster line generation algorithm. This algorithm scan converts lines using only incremental integer calculations and these calculations can also be adopted to display circles and other curves.

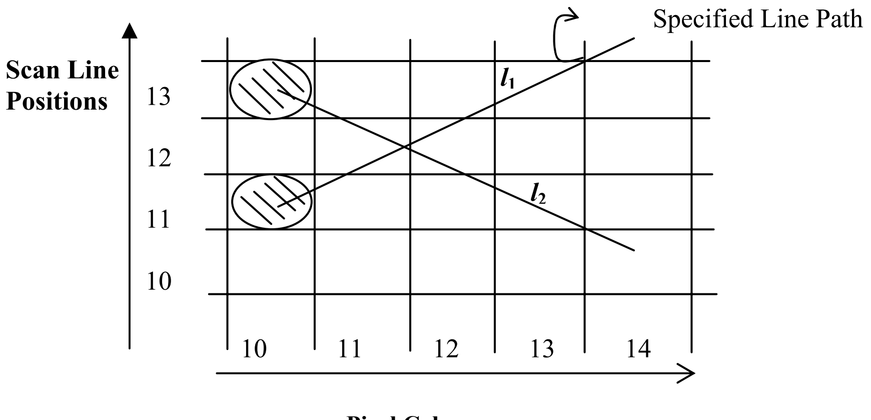


**Figure 8: Bresenham line generation**

Sampling at Unit *x* distance in *Figure 8*, we need to decide which of the two possible pixel position is closer to the line path at each sample step.

**In $l_1$ :** we need to decide that at next sample position whether to plot the pixel at position (11, 11) or at (11, 12).

Similarly, **In $l_2$**: the next pixel has to be (11, 13) or (11, 12) or what choice of pixel is to be made to draw a line is given by Bresenham, by testing the sign of the integer parameter whose value is proportional to the difference between the separation of the two pixel positions from actual line path. In this section, we will discuss the Bresenham line drawing algorithm for + ve slope ( $0 < m < 1$). If the slope is negative then, use reflection transformation to transform the line segment with negative slope to line segment with positive slope. Now, let us discuss the generation of line again in two situations as in the case of DDA line generation.

**Case 1: Scan Conversion of Line with the slope (0 < *m* < 1)**

Here the pixel positions along the line path are determined by sampling at Unit *x* intervals *i.e.,* starting from the initial point. $(x_0, y_0)$ of a given line we step to each successive column. (*i.e.*, *x*-position) and plot the pixel whose scanline *y* value is closest to the line path. Assume we proceed in this fashion up to the kth step. The process is shown in *Figure 9*. Assuming we have determined the pixel at $(x_k, y_k)$. We need to decide which pixel is to be plotted in column $x_{k+1.}$ Our choices are either $(x_{k+1}, y_k)$ or $(x_{k+1}, y_{k+1})$.

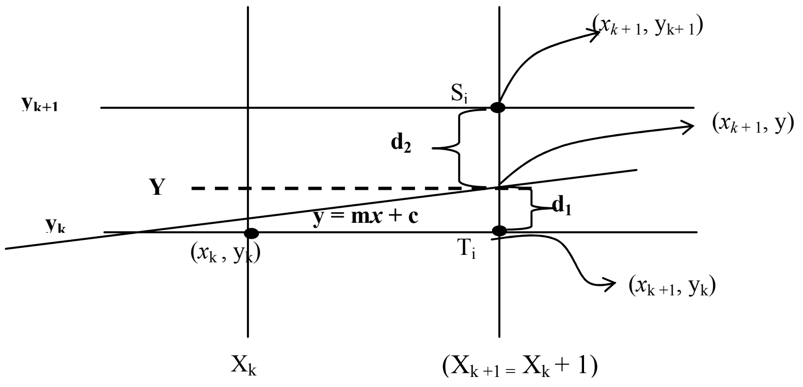


**Figure 9: Scan conversion 0 < m < 1**

At sampling position $X_{k+1}$ the vertical pixel (or scan line) separation from mathematical line ($y = mx + c$) is say $d_1$ and $d_2$.

Now, the *y* coordinate on the mathematical line at pixel column position $X_{k+1}$ is:

$y = m\,(x_{k+1}) + c$ ---------------------(1)

by *Figure 9*:

$d_1 = y - y_k$ ---------------------(2)

$= m\,(x_{k+1}) + c - y_k$ ---------------------(3) (using (1))

similarly, $d_2 = (y_{k+1}) - y = y_{k+} 1 - m\,(x_{k+1}) - c$ ---------------------(4)

using (3) and (4) we find $d_1 - d_2$

$d_1 - d_2 = [m\,(x_{k+} 1) + c - y_k] - [y_k + 1 - m\,(x_{k+} 1) - c]$

$= mx_k + m + c - y_k - y_k - 1 + mx_k + m + c$

$= 2m\,(x_k + 1) - 2y_k + 2c - 1$ ---------------------(5)

Say, decision parameter p for kth step i.e., $p_k$ is given by

$$p_k = \Delta x(d_1 - d_2) \quad \text{---------------------(6)}$$

Now, a decision parameter $p_k$ for the kth step in the line algorithm can be obtained by rearranging (5) such that it involves only integer calculations. To accomplish this substitute $m = \Delta y/\Delta x$ where, $\Delta y$ and $\Delta x \Rightarrow$ vertical and horizontal separations of the end point positions.

$p_k = \Delta x\,(d_1 - d_2) = \Delta\, x\,[2m\,(x_k + 1) - 2y_k + 2c - 1]$

$= \Delta x\,[2(\Delta y/\Delta x)\ (x_k + 1) - 2y_k + 2c - 1]$

$= 2\,\Delta y\,(x_k + 1) - 2\,\Delta x y_k + 2\,\Delta x c - \Delta x$

$= 2\,\Delta y\,x_k - 2\,\Delta x\,y_k + [2\,\Delta y + \Delta x\,(2c - 1)]$

$p_k = 2\,\Delta y\,x_k - 2\Delta x y_k + b$ ------------------- (7)

where b is constant with value $b = 2\Delta y + \Delta x\,(2c - 1)$ --------------------(8)

**Note:** *sign of* $p_k$ is same as sign of $d_1 - d_2$ since it is assumed that $\Delta x > 0$
[As in *Figure 9*, $d_1 < d_2$ *i.e.* $(d_1 - d_2)$ is –ve i.e., $p_k$ is –ve so pixel $T_i$ is more appropriate choice otherwise pixel $S_i$ is the appropriate choice.
*i.e.,* (1) if $p_k < 0$ choose $T_i$, so next pixel choice after $(x_k, y_k)$ is $(x_{k+1}, y_k)$
else (2) if $p_k > 0$ choose $S_i$, so next pixel choice after $(x_k, y_k)$ is $(x_{k+1}, y_{k+1})$.

At step $k + 1$ the decision parameter is evaluated by writing (7) as:

$$p_{k+1} = 2\Delta y\, x_{k+1} - 2\Delta x\, y_{k+1} + \mathrm{b} \qquad \text{---------------------(9)}$$

Subtracting (7) from (9) we get

$$p_{k+1} - p_k = 2\Delta y(\underbrace{x_{k+1} - x_k}_{\mathbf{1}}) - 2\Delta x\,(\underbrace{y_{k+1} - y_k}_{\mathbf{0\ or\ 1}}) = 2\Delta y - 2\Delta x\,(y_{k+1} - yk)$$

$$\text{depending on sign of } p_k \quad \begin{bmatrix} p_k < 0 & y_{k+1} = y_k \\ p_k > 0 & y_{k+1} = y_k + 1 \end{bmatrix}$$

$$\boxed{\phantom{xxxxxxxxxxxxxxxxxxxxxxxxxxxx}} \qquad \text{---------------------(10)}$$

This recursive calculation of decision parameter is preformed at each integer position, beginning with the left coordinate end point of line.

This first parameter $p_0$ is determined by using equation (7), and (8) at the starting pixel position $(x_0, y_0)$ with $m$ evaluated as $\Delta y / \Delta x$ (*i.e.,* intercept $c = 0$)

$$p_0 = 0 - 0 + b = 2\Delta y + \Delta x\,(2 * 0 - 1) = 2\Delta y - \Delta x$$

$$p_0 = 2\Delta y - \Delta x \qquad \text{-----------------------(10 A)}$$

**Summary:**

(Bresenham line drawing algorithm for +ve slope ( $0 < m < 1$).

- If slope is negative then use reflection transformation to transform the line segment with negative slope to line segment with a positive slope.

- Calculate constants $2\Delta y$, $2\Delta y - 2\Delta x$, $\Delta y$, $\Delta x$ at once for each line to be scan converted, so the arithmetic involves only integer addition/subtraction of these constants.

Remark : The algorithm will be exactly the same if we assume $| m | < 1$

- **Algorithm $| m | < 1$:**
  (a) Input two line end points and store left end point in $(x_0, y_0)$
  (b) Load $(x_0, y_0)$ on frame buffer *i.e.,* plot the first point.
  (c) Calculate $\Delta x$, $\Delta y$, $2\Delta y$, $2\Delta y - 2\Delta x$ and obtain the starting value of decision parameter as $p_0 = 2\Delta y - \Delta x$
  (d) At each $x_k$ along the line, starting at $k = 0$, perform following test:

      if $p_k < 0$, the next plot is $(x_{k+1}, y_k)$ and $p_{k+1} = p_k + 2\Delta y$
      else next plot is $(x_{k+1}, y_{k+1})$ and $p_{k+1} = p_k + 2(\Delta y - \Delta x)$

  (e) Repeat step (D) $\Delta x$ times.

## Bresenham Line Generation Algorithm (| m | < 1)

```
$\Delta x \leftarrow x_1 - x_0$
$\Delta y \leftarrow y_1 - y_0$
$p_0 \leftarrow 2\Delta y - \Delta x$
```

```
while $(x_0 < = x_1)$ do
          {puton $(x_0, y_0)$
          if (pi > 0) then
          {$x_0 \leftarrow x_0 + 1$;
             $y_0 \leftarrow y_0 + 1$;
          $p_{i+1} \leftarrow pi + 2\ (\Delta y - \Delta x)$;

          }
       if (pi < 0) then
       {$x_0 \leftarrow x_0 + 1$
        $y_0 \leftarrow y_0$
     $p_{i+1} \leftarrow pi + 2\ \Delta y$
       }
      }
```

**Note:**

- Bresenhams algorithm is generalised to lines with arbitrary slopes by considering the symmetry between the various octants and quadrants of the coordinate system.
- For line with +ve slope (m) such that m > 1, then we interchange the roles of *x* and *y* direction i.e., we step along *y* directions in unit steps and calculate successive *x* values nearest the line path.
- for –ve slopes the procedures are similar except that now, one coordinate decreases as the other increases.

**Example 2:** Draw line segment joining (20, 10) and (25, 14) using Bresenham line generation algorithm.

**Solution:** $(x_0, y_0) \rightarrow (20, 10)$ ; $(x_1, y_1) \rightarrow (25, 14)$

$$m = \frac{y_1 - y_0}{x_1 - x_0} = \frac{14-10}{25-20} = \frac{4}{5} < 1$$

As, $m = \frac{\Delta y}{\Delta x} = \frac{4}{5} \Rightarrow \Delta y = 4$

$\Delta x = 5$

$\rightarrow$ plot point (20, 10)

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

$i = 1$: $p_i = 2 * 4 - 5 = 3$

*as* $p_1 > 0$ *so* $x_0 \leftarrow 21;\ y_0 \leftarrow 11$

*now plot (21,11)*

$i = 2$ *as* $p_1 > 0$

$\therefore p_2 = p_1 + 2(\Delta y - \Delta x)$

$= 3 + 2\ (4 - 5) = 3 - 2 = 1$

$p_2 > 0$ *so* $x_0 \leftarrow 22;\ y_0 \leftarrow 12$

*plot (22,12)*

$i = 3$ *as* $p_2 > 0$

$\therefore p_3 = p_2 + 2\ (\Delta y - \Delta x) = 1 + 2\ (4 - 5) = -1$

$p_3 < 0$ $\therefore x_0 \leftarrow 23$

$y_0 \leftarrow 12$

*plot (23, 12)*

$i = 4$ *as* $p_3 < 0$

$\therefore p_4 = p_3 + 2\Delta y$

$= -1 + 2 * 4 = 7$

$\therefore x_0 \leftarrow 24; y_0 \leftarrow 13$

*plot (24, 13)*

$i = 5$ *as* $p_4 > 0$

$\therefore p_5 = p_4 + 2(\Delta y - \Delta x)$

$= 7 + 2(4 - 5) = 5$

$x_0 \leftarrow 25; y_0 \leftarrow 14$

plot (25, 14)

{for i = 6, $x_0$ will be > $x_i$ so algorithm terminates

**Example 3:** Illustrate the Bresenham line generation algorithm by digitzing the line with end points (20, 10) and (30,18)

**Solution:** $m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{\Delta y}{\Delta x} = \frac{18-10}{30-20} = \frac{8}{10} = 0.8$ -------------------(1)

$\Rightarrow \Delta y = 8$ and $\Delta x = 10$ -------------------(2)

value of initial decision parameter ($p_0$) = $2\Delta y - \Delta x = 2 * 8 - 10 = 6$ -----------------(3)

value of increments for calculating successive decision parameters are:

$2\Delta y = 2 * 8 = 16;$ -------------------(4)

$2\Delta y - 2\Delta x = 2 * 8 - 2 * 10 = -4$ -------------------(5)

plot initial point ($x_0$, $y_0$) = (20, 10) in frame buffer now determine successive pixel positions along line path from decision parameters value (20, 10).

| $k$ | $p_k$ | $(x_{k+1}, y_{k+1})$ |
|---|---|---|
| 0 | 6 | (21, 11) |
| 1 | 2 | (22, 12) |
| 2 | – 2 | (23, 12) |
| 3 | 14 | (24, 13) |
| 4 | 10 | (25, 14) |
| 5 | 6 | (26, 15) |
| 6 | 2 | (27, 16) |
| 7 | – 2 | (28, 16) |
| 8 | 14 | (29, 17) |
| 9 | 10 | (30, 18) |

← [use step (d) of algorithm $\Delta x$ times]

If $p_k > 0$ then increase both $X$ and $Y$ and $p_{k+1} = p_k + 2\Delta y - 2\Delta x$

If $p_k < 0$ then increase $X$ and not $Y$ and $p_{k+1} = p_k + 2\Delta y$

## ☞ Check Your Progress 2

1) Compare Bresenham line generation with DDA line generation.

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

2) Illustrate the Bresenham line generation algorithm by digitising the line with end points (15, 5) and (25,13).

……………………………………………………………………………………

……………………………………………………………………………………

## 2.5 CIRCLE-GENERATION ALGORITHMS

Circle is one of the basic graphic component, so in order to understand its generation, let us go through its properties first:

### 2.5.1 Properties of Circles

In spite of using the Bresenham circle generation (i.e., incremental approval on basis of decision parameters) we could use basic properties of circle for its generation.

These ways are discussed below:

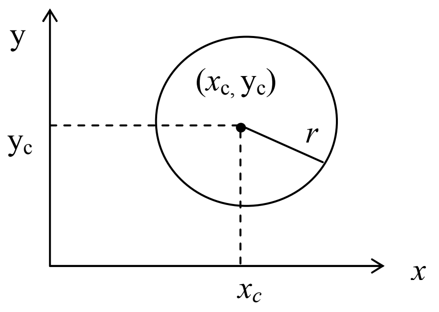


**Figure 8: Property of circle**

### Generation on the basis of Cartesian Coordinates:

A circle is defined as the set of points or locus of all the points, which exist at a given distance $r$ from center ($x_c$, $y_c$). The distance relationship could be expressed by using Pythagonous theorem in Cartesian coordinates as

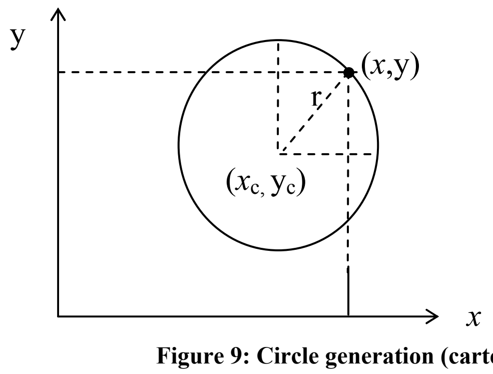


**Figure 9: Circle generation (cartesian coordinate system)**

$$(x - x_c)^2 + (y - y_c)^2 = r^2 \qquad \text{--------------------(1)}$$

Now, using (1) we can calculate points on the circumference by stepping along $x$-axis from $x_c – r$ to $x_c + r$ and calculating respective $y$ values for each position.

$$y = y_c \pm \sqrt{r^2 - (x - x_c)^2} \qquad \text{---------------------(2)}$$

**Generation on basis of polar coordinates (*r* and θ)**

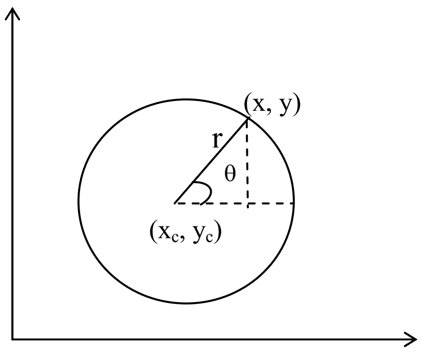


**Figure 10: Circle generation (polar coordinate system)**

Expressing the circle equation in parametric polar form yields the following equations

$x = x_c + r \cos \theta$

$y = y_c + r \sin \theta$ --------------------------(3)

Using a fixed angular step size we can plot a circle with equally spaced points along the circumference.

**Note:**

- *Generation of circle with the help of the two ways mentioned is not a good choice ∵ both require a lot of computation equation (2) requires square root and multiplication calculations where as equation (3) requires trigonometric and multiplication calculations. A more efficient approach is based on incremental calculations of decision parameter. One such approach is Bresenham circle generation. (mid point circle algorithm*).

- This *Bresenham circle generation (mid point circle algorithm*) is similar to line generation. Sample pixels at Unit *x* intervals and determine the closest pixel to the specified circle path at each step.

- For a given radius and center position (*x, y,*) we first setup our algorithm to calculate pixel position around the path of circle centered at coordinate origin (0, 0) *i.e.,* we translate $(x_c, y_c) \rightarrow (0, 0)$ and after the generation we do inverse translation $(0, 0) \rightarrow (x_c, y_c)$ hence each calculated position (*x*, *y*) of circumference is moved to its proper screen position by adding $x_c$ to *x* and $y_c$ to *y*.

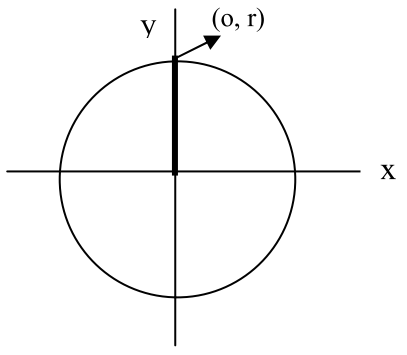
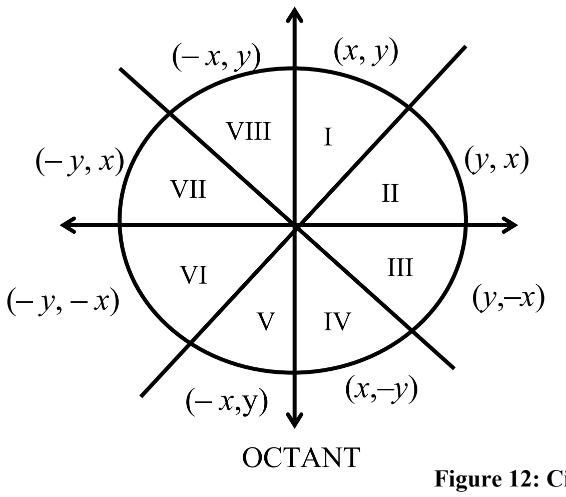


**Figure 11: Circle generation (initialisation)**

- In *Bresenham circle generation (mid point circle algorithm)* we calculate points in an octant/quadrant, and then by using symmetry we find other respective points on the circumference.



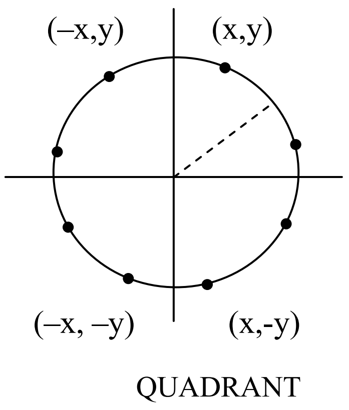


**Figure 12: Circle generation approaches**

## 2.5.2 Mid Point Circle Generation Algorithm

We know that equation of circle is $x^2 + y^2 = r^2$. By rearranging this equation, we can have the function, to be considered for generation of circle as $f_c(x, y)$.

$$f_c(x, y) = x^2 + y^2 - r^2 \qquad \text{-------------(1)}$$

The position of any point $(x, y)$ can be analysed by using the sign of $f_c(x, y)$ i.e.,

$$\text{decision parameter} \rightarrow f_c(x, y) = \begin{cases} < 0 \text{ if } (x, y) \text{ is inside circle boundary} \\ = 0 \text{ if } (x, y) \text{ is on the circle boundary} \\ > 0 \text{ if } (x, y) \text{ is outside circle boundary} \end{cases} \qquad \text{---------(2)}$$

i.e., we need to test the sign of $f_c(x, y)$ are performed for mid point between pixels near the circle path at each sampling step.

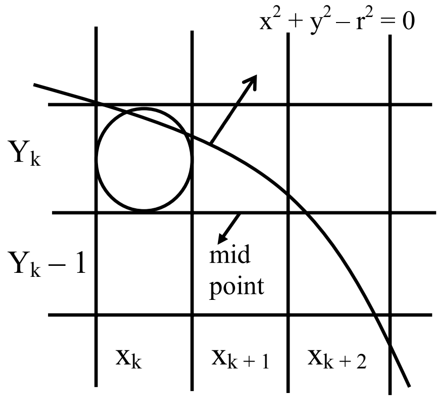


**Figure 13: Mid point between Candidate Pixel at sampling Position $x_{k+1}$ along Circular Path**

*Figure 13* shows the mid point of two candidate pixel $(x_{k+1}, y_k)$ and $(x_{k+1}, y_{k-1})$ at sampling position $x_k + 1$.

Assuming we have just plotted the $(x_k, y_k)$ pixel, now, we need to determine the next pixel to be plotted at $(x_{k+1}, y_k)$ or $(x_{k+1}, y_{k-1})$. In order to decide the pixel on the circles path, we would need to evaluate our decision parameter $p_k$ at mid point between these two pixels i.e.,

$$p_k = f_c(x_{k+1}, y_k - \frac{1}{2}) = (x_{k+1})^2 + (y_k - \frac{1}{2})^2 - r^2 \qquad \text{---------------------(3)}$$

(using (1)

If $p_k < 0$ then $\Rightarrow$ midpoint lies inside the circle and pixel on scan line $y_k$ is closer to circle boundary else mid point is outside or on the circle boundary and we select pixel on scan line $y_k - 1$ successive decision parameters are obtained by using incremental calculations.

The recursive way of deciding parameters by evaluating the circle function at sampling positions $x_{k+1} + 1 = (x_k + 1) + 1 = x_k + 2$

$$p_{k+1} = f_c\left(x_{k+1} + 1,\ y_{k+1} - \frac{1}{2}\right) = \left[(x_k + 1) + 1)^2 + \left(y_{k+1} - \frac{1}{2}\right)^2 - r^2\right] \text{ --------(4)}$$

subtract equation (3) from (4) we get

$$p_{k+1} - p_k = \left\{[(x_k + 1) + 1]^2 + \left(y_{k+1} - \frac{1}{2}\right)^2 - r^2\right\} - \left\{(x_k + 1)^2 + \left(y_k - \frac{1}{2}\right)^2 - r^2\right\}$$

$$p_{k+1} - p_k = [(x_k + 1)^2 + 1 + 2(x_k + 1).\ 1] + \left[y^2_{k+1} + \frac{1}{4} - \cancel{2}.\frac{1}{\cancel{2}}.\ y_{k+1}\right] - \cancel{r^2} - (x_k + 1)^2$$

$$- \left[y_k^2 + \frac{1}{4} - \cancel{2}.\ \frac{1}{\cancel{2}}.\ y_k\right] + \cancel{r^2}$$

$$= \cancel{(x_k + 1)^2} + 1 + 2\,(x_k + 1) + y^2_{k+1} + \cancel{\frac{1}{4}} - y_{k+1} - \cancel{(x_k + 1)^2} - y_k^2 - \cancel{\frac{1}{4}} + y_k$$

$$p_{k+1} = p_k + 2\,(x_k + 1) + (y^2_{k+1} - y_k^2) - (y_{k+1} - y_k) + 1 \text{ --------------(5)}$$

Here, $y_{k+1}$ could be $y_k$ or $y_{k-1}$ depending on sign of $p_k$

$$\left[\begin{array}{l} \text{so, increments for obtaining } p_{k+1} \text{ are : -} \\ \text{either } 2x_{k+1} + 1 (\text{if } p_k < 0)\,(i.e., y_{k+1} = y_k) \\ or\ 2x_{k+1} + 1 - 2y_{k-1} (\text{if } p_k > 0)\,(i.e., y_{k+1} = y_k - 1)\, So(2y_{k+1} = 2(y_k - 1) = 2y_k - 2 \end{array}\right.$$

**How do these increments occurs?**

- $y_{k+1} = y_k$ : use it in (5) we get

  $p_{k+1} = p_k + 2\,(x_k + 1) + 0 - 0 + 1 = p_k + (2x_{k+1} + 1)$

$\Rightarrow$ $\boxed{\phantom{xxxxxxxxxxxx}}$ ----------------(6)

- $y_{k+1} = y_{k-1}$ use it in (5) we get

  $p_{k+1} = p_k + 2x_{k+1} + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k) + 1$

  $= p_k + 2x_{k+1} + \dfrac{(y_{k+1} - y_k)\quad [(y_{k+1} + y_k) - 1]}{} + 1$

$\boxed{p_{k+1} = p_k + (2x_{k+1} - 2y_{k-1} + 1)}$ ------------------(7)

Also,

$2x_{k+1} = 2\,(x_k + 1) = 2x_k + 2$ ------------------ (8)

$2y_{k+1} \rightarrow 2y_{k-1} = 2\,(y_k - 1) = 2y_k - 2$ -------------------(9)

**Note:**

- At starting position $(0, r)$ the terms in (8) and (9) have value 0 and $2r$ respectively. Each successive value is obtained by adding 2 to the previous value of $2x$ and subtracting 2 from the previous value of $2y$.

- The initial decision parameter is obtained by evaluating the circle function at start position $(x_0, y_0) = (0, r)$

  *i.e.,* $p_0 = f_c\left(1, r - \frac{1}{2}\right) = 1 + \left(r - \frac{1}{2}\right)^2 - r^2 = \frac{5}{4} - r$ (using 1)

  If radius $r$ is specified as an integer then we can round $p_0$ to

  $\boxed{p_0 = 1 - r}$ (for $r$ as an integer).

**Midpoint Circle Algorithm**

(a) Input radius $r$ and circle, center $(x_c, y_c)$ and obtain the first point on the circumference of the circle centered on origin as

$$(x_0, y_0) = (0, r)$$

(b) Calculate initial value of decision parameter as

$$p_0 = \frac{5}{4} - r \sim 1 - r$$

(c) At each $x_k$ position starting at $k = 0$ perform following test.
1) If $p_k < 0$ then next point along the circle centered on $(0, 0)$ is $(x_{k+1}, y_k)$ and $p_{k+1} = p_k + 2x_{k+1} + 1$
2) Else the next point along circle is $(x_{k+1}, y_k - 1)$ and
$p_{k+1} = p_k + 2x_{k+1} - 2y_{k-1}$
where $2x_{k+1} = 2(x_k + 1) = 2x_k + 2$ and $2y_{k-1} = 2(y_k - 1) = 2y_k - 2$

(d) Determine symmetry points in the other seven octants.
(e) Move each calculated pixel position $(x, y)$ onto the circular path centered on $(x_c, y_c)$ and plot coordinate values $x = x + x_c,\ y = y + y_c$
(f) Repeat step (c) through (e) until $x \geq y$.

**Example 4:** Given a circle radius $r = 10$ determine positions along the circle octants in 1st Quadrant from $x = 0$ to $x = y$.

**Solution:** An initial decision parameter $p_0 = 1 - r = 1 - 10 = -9$
For circle centered on coordinate origin the initial point $(x_0, y_0) = (0, 10)$ and initial increment for calculating decision parameter are:

$$2x_0 = 0, 2y_0 = 20$$

Using mid point algorithm point are:

| $k$ | $p_k$ | $(x_{k+1}, y_{k-1})$ | $2x_{k+1}$ | $2y_{k-1}$ |
|---|---|---|---|---|
| 0 | – 9 | (1, 10) | 2 | 20 |
| 1 | – 6 | (2, 10) | 4 | 20 |
| 2 | – 1 | (3, 10) | 6 | 20 |
| 3 | 6 | (4, 9) | 8 | 18 |
| 4 | – 3 | (5, 9) | 10 | 18 |
| 5 | 8 | (6, 8) | 12 | 16 |
| 6 | 5 | (7, 7) | 14 | 14 |

## 2.6 POLYGON FILLING ALGORITHM

In many graphics displays, it becomes necessary to distinguish between various regions by filling them with different colour or at least by different shades of gray. There are various methods of filling a closed region with a specified colour or equivalent gray scale. The most general classification appears to be through following algorithms:

1) Scan Line polygon fill algorithm.
2) Seed fill algorithm./Flood fill algorithm which are further classified as:
   (a) Boundary fill algorithm
   (b) Interior fill algorithm

We restrict our discussion to scan line polygon fill algorithm, although we will discuss the others in brief at the end of this section.

## Scan Line Polygon Fill Algorithm

In this, the information for a solid body is stored in the frame buffer and using that information all pixels i.e., of both boundary and interior region are identified and, are hence plotted.

Here we are going to do scan conversion of solid areas where the region is bounded by polygonal lines. Pixels are identified for the interior of the polygonal region, and are then filled plotted with the predefined colour.

Let us discuss the algorithm briefly, then we will go into details on the same.

This algorithm checks and alters the attributes (i.e., characteristics and parameters) of the pixels along the current raster scan line. As soon as it crosses over from the outside to the inside of a boundary of the specified polygon it starts resetting the colour (or gray) attribute. In effect filling the region along that scan line. It changes back to the original attribute when it crosses the boundary again. *Figure 14* shows variations of this basic idea.

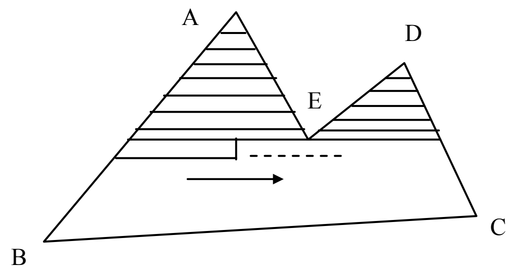


**Figure 14: Concept of scan line polygon filling**

So as to understand Scan Line Polygon Fill Algorithm in detail consider *Figure 15*:

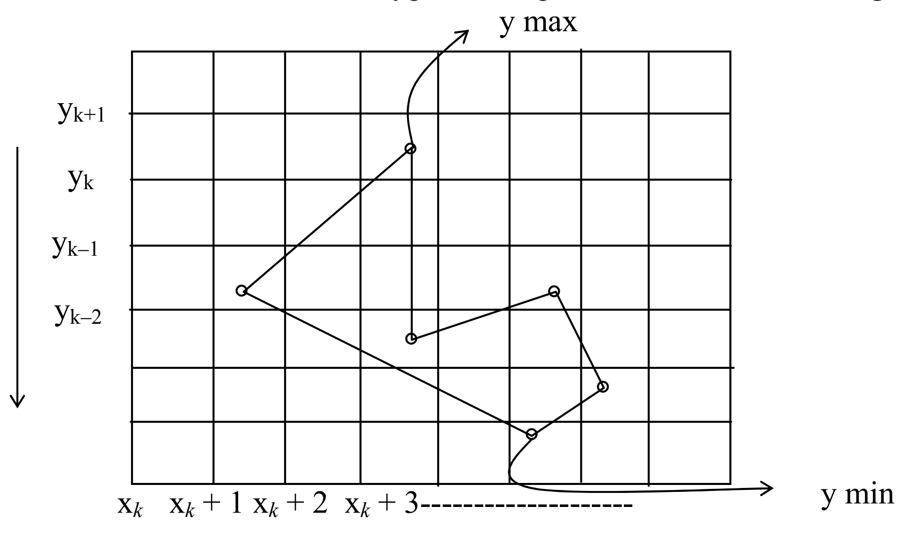


**Figure 15: Scan line polygon filling**

Here, in *Figure 15,*

- $y_k, y_{k-1}, \ldots$ → are scan lines from top to bottom (value of y at top or scan line at top has maximum value and the scan line at bottom has minimum y value).
- $x_k, x_k + 1 \ldots$ →are consecutive pixels on a scan line *i.e.,* $(x_k, y_k)$, $(x_{k+1}, y_k), \ldots \Rightarrow$ a sea of pixels for scan lines $y_k, y_{k-1}, y_{k-2}, \ldots, y_i$ and for the chosen scan line we find pixels satisfying the condition as mentioned above.
- For a given scan line, let us assume that it intersects edges of the given polygonal region at the points *P1, P2, ..., Pk.* Sort these points *P1, P2, ..., Pk* in terms of the *X*-coordinates in increasing order.

Now, for a chosen scan line there are three cases:

(a) Scan line passes through the edges in between shown by point a in *Figure 16.*
(b) Scan line passes through the vertex of the polygon whose neighbouring vertices lie completely on one side of the scan line (point b, point c in *Figure 16*).
(c) Scan line passes through some bottom vertex of polygon whose neighbouring vertices lie on both sides of the scan line.

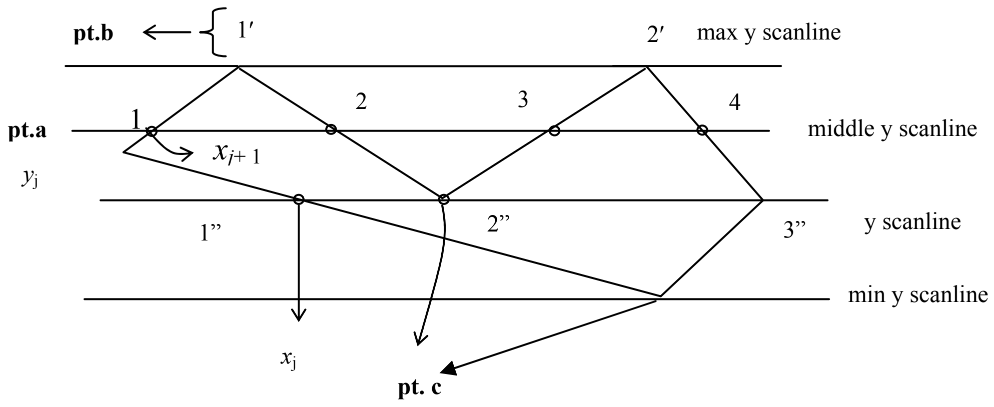


**Figure 16: Cases for scan line polygon filling**

**In case a**, *i.e.,* when a scan line intersects an edge at a point but not a vertex point of the edge (as in case of point 1, 2, 3, 4) then that point of intersection is considered as a single point and it is taken as ON/OFF point depending on the requirement. Therefore, in this case the intersection points are taken as 1234. Colour of the pixels between the intersection points 1 and 2 is replaced with the filling colour but the colour of the pixels between the points 2 and 3 are left unchanged. Again the colour of the pixels between 3 and 4 are changed by the filling colour.

**In case b and c**, case *c* i.e., when a scan line intersects an edge E1 at a vertex V1 i.e., a vertex point of the edge E1 whose *y* coordinate is greater (or less ) than the *y* coordinate values of the other two vertex points V2 and V3 where, for example, edge E1 has end vertex points V1, V2 and edge E2 having end vertex points V1, V3. That is the vertices V2 and V3 will lie either completely above V1 or both V2 and V3 will lie completely below the vertex V1. In this case, the point of intersection, i.e., E1, will be counted two times. For example, the vertices 1′ and 2′ are such that their y-coordinate values are grater than the y-coordinate values of their neighbouring vertices and therefore they are counted twice.1′1′ 2′2′ i.e. the colour of the pixels between 1′, 1′ is replaced with the filling colour but the colour of the pixels between 1′ , 2′ is left unchanged. Again the colour of the pixels between 2′, 2′ is changed.

Assume that the scan line passes through a vertex point V1 of the polygonal region having neighbouring vertices V0 and V2., i.e. let E1 and E2 be two edges of the polygon so that V0, V1 and V1, V2 be respectively their end vertices. Suppose we assume that the vertex V1 is such that the y-coordinate for the neighbouring vertex V0 is grater than the y-coordinate for V1 but the y-coordinate for the neighbouring vertex

V2 is less than the y-coordinate for V1. In this case, the intersection vertex point V0 will be taken as a single point and the algorithm will remain the same as above.

The point of intersection is to be considered two times, i.e., 1″ 2″ 2″ 3″. 1″ 2″ ⇒ make pixels ON from 1″ to 2″, 2″ 2″ don't make pixel ON, 2″, 3″⇒ make pixels ON from 2″ to 3″.

Now, the requirements for scanning an image are:

1) Determine the point of intersection of an edge and scan line

2) Sort the points on scan line w.r.t $x$ value. Get the respective pixels ON.

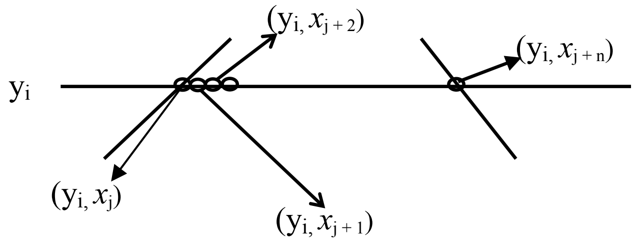


**Figure 17: Requirement for image scanning**

**Sum up:** To fill a polygonal image we have to identify all edges and vertices of intersections while moving from scan line $y_{max}$ to $y_{min}$, for each scan line. Then check for the case and perform the algorithm accordingly.

**Recursive way to scan a figure**

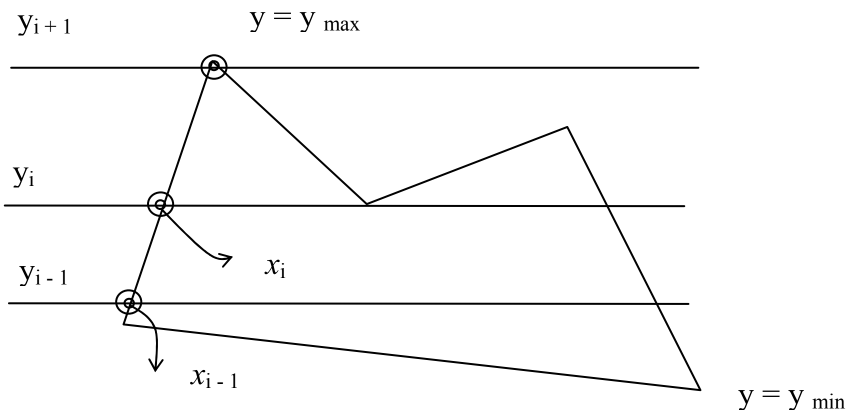


**Figure 18: Scan line polygon filling (a recursive approach)**

For a scan line $y = y_i$ identify the edges and use the data of the point of intersection $(x_i, y_i.)$. Say $x_0$ is point on scan line $y_0$ and $x_1$ is point on scan line $y_1$. Now between $(x_0, y_0)$ and $(x_1, y_1)$ we find slope (this all is to find recursive way to scan a figure).

| $x_0$ | $y_0$ | $x_1$ | $y_1$ | $1/m$ |
|---|---|---|---|---|

$$\because \quad m = y_1 - y_0/x_1 - x_0 = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}$$

$$\Rightarrow m\,(x_{i+1} - x_i) = (y_{i+1} - y_i) \qquad \text{-------------(1)}$$

now $y_{i+1}$ is scan line next to $y_i$ i.e. $\boxed{\phantom{xxxxxxxx}}$ -------------(2)

($\because$ moving from top to bottom we are moving from $y_{max}$ to $y_{min}$)

Using (2) in (1) we get

$$m\,(x_{i+1} - x_i) = (y_{i+1} - y_i) = (y_i + 1 - y_i)$$

$$\Rightarrow \quad \boxed{x_{i+1} = x_i - (1/m)}$$

Using $y_{i+1}$ and $x_{i+1}$ expressions we can find consecutive points. $x$ for respective scan lines $y_i$ . If $x_i$ comes out to be real then round the value to integer values.

**Note:** Although Flood fill algorithms are not part of this block let me briefly describe Flood Fill Algorithms. The algorithm starts with the coordinates of a specified point (known as the "seed" pixel) inside the polygon to be filled, and proceeds outwards from it, testing the adjacent pixels around it in orderly fashion until the spreading wave reaches the boundary in every direction, as suggested in *Figure 19*.

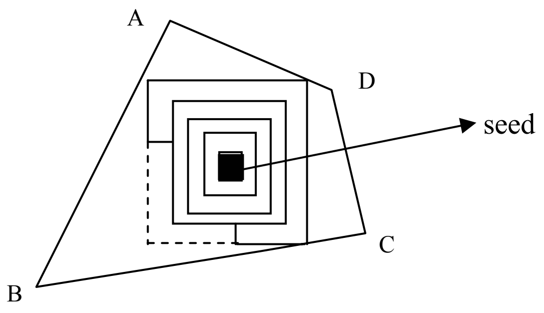


**Figure 19: Flood filling method**

To economise in storage and memory management, a combination of the two techniques may be used. Starting from an interior seed, first the scan-line through the seed is painted until the left and right boundaries are reached, then the line above and below are similarly covered. The procedure is repeated until the entire region is filled.

Filling may be solid (painted with a single colour), or it may be patterned or tiled (filled with different patterns).

Fill options in standard packages such as MS-Word are grouped generally under the heads: (a) Gradient, variations from light to dark of specified colour in specified directions; (b) Texture; (c) Pattern (i.e., Hatching); and (d) Picture, of the user's choice. Some of these will be discussed later.

## ☞ Check Your Progress 3

1) Do we need to generate the full circumference of the circle using the algorithm, or can we can generate it in a quadrant or octant only and then use it to produce the rest of the circumference?

……………………………………………………………………………………………

……………………………………………………………………………………………

……………………………………………………………………………………………

……………………………………………………………………………………………

……………………………………………………………………………………………

2) Given a circle radius $r = 5$ determine positions along the circle octants in 1st Quadrant from $x = 0$ to $x = y$?

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

3) Distinguish between Scan line polygon fill and Seed fill or Flood fill algorithm?

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

## 2.7 SUMMARY

In this unit, we have discussed the basic graphic primitives that is point, line and circle; we have also discussed both theoretical and practical application of different algorithms related to their generation. At the end of the unit, we have emphasised on different seed fill and flood fill type of polygon filling algorithms. The filling algorithms are quite important as they give privilege to quickly fill colours into the graphics created by you.

## 2.8 SOLUTIONS/ANSWERS

### Check Your Progress 1

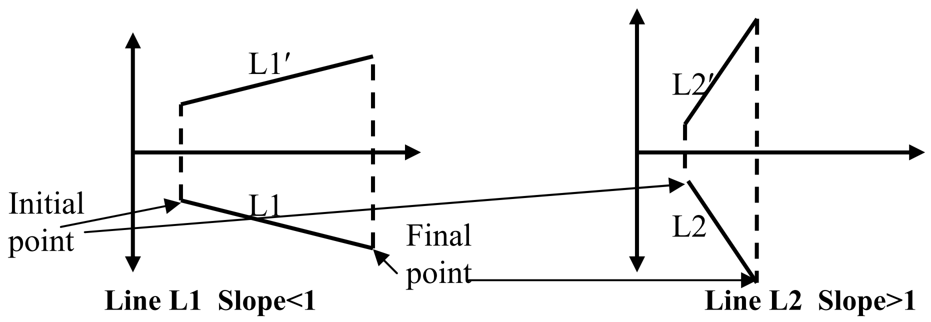
1) Graphic primitives are the fundamental graphic objects which can be united in any number and way to produce a new object or image.

2) Due to low resolution and improper approximation of pixel positions the images are generally distorted and the Zig-Zags (i.e., distortions) produced in the display of straight line or any other graphic primitive is the staircase effect.

3) The approximation involved in the calculation for determination of pixel position involved in the display of lines and other graphic primitives is the actual cause of this effect.

4) In a high resolution system the adjacent pixels are so closely spaced that the approximated line-pixels lie very close to the actual line path and hence the plotted lines appear to be much smoother — almost like straight lines drawn on paper. In a low resolution system, the same approximation technique causes lines to be displayed with a "stairstep apperance" i.e., not smooth.

5)



For the generation of lines with negative slopes:

Slope < 1 : successively increase y and respectively decrease x
Slope > 1 : successively increase x and respectively decrease y

*Aliter*: This hint you will understand after going through Block 2.

In the shown Figure say L1 and L2 are lines with negative slope with a magnitude of <1 and >1 respectively. Then for the generation of such a line, you may take reflection of the line about the concerned axis and follow the usual algorithm of line generation. After that you may inverse reflection the transformation, and this will provide you the line generation of negative slope.

6) We know that the general equation of the line is given by
$y = mx+c$ where $m =( y_1 - y_0/( x_1 - x_0)$

given $(x_0, y_0) \rightarrow (1, 1) ; (x_1, y_1) \rightarrow (9, 7)$

$\Rightarrow m = (7-1)/(9-1) =6/8$
$C = y_1 - mx_1 = 7 - (6/8) *9 = 1/4$

So, by equation of line $( y = mx + c)$ we have
$y = (6/8)x+(1/4)$

| DDA Algorithm Two case: | |
|---|---|
| Case 1: $m < 1$ | $x_{i+1} = x_i + 1$ |
| | $y_{i+1} = y_i + m$ |
| Case 2: $m > 1$ | $x_{i+1} = x_i + (1/m)$ |
| | $y_{i+1} = y_i + 1$ |

as m < 1 so according to DDA algorithm case 1
$x_{i+1} = x_i + 1$ $y_{i+1} = y_i + m$
given $(x_0, y_0)$ = (1, 1)

1) $x_1 = x_0 + 1 = 2$
$y_1 = y_0 + m = 1+(6/8) = 7/4$

put pixel ($x_0$, round y, colour)
i.e., put on (2, 2)

Similarly, go on till (9, 7) is arrived at.

**Check Your Progress 2**

1) Bresenham line generation algorithm is better than DDA because it has better solution to the concept of approximation of pixel position to be enlightened for the display of any graphic primitive, thus, reduces the staircase effect, for more details refer to 2.4.2

2) Here we are using the Bresenham line generation algorithm by digitising the line with end points (15, 5) and (25,13).

$$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{\Delta y}{\Delta x} = \frac{13-5}{25-15} = \frac{8}{10} = 0.8 \qquad \text{------------------(1)}$$

$\Rightarrow \Delta y = 8$ and $\Delta x = 10$ -------------------(2)

value of initial decision parameter ($p_0$) = $2\Delta y - \Delta x = 2 * 8 - 10 = 6$ ------------(3)

value of increments for calculating successive decision parameters are:

$2\Delta y = 2 * 8 = 16$; -------------------(4)

$2\Delta y - 2\Delta x = 2 * 8 - 2 * 10 = -4$ -------------------(5)

plot initial point $(x_0, y_0) = (15, 5)$ in the frame buffer now determine successive pixel positions along line path from decision parameters value (15, 5).

| ***k*** | $p_k$ | $(x_{k+1}, y_{k+1})$ | ← **[use step (d) of algorithm $\Delta x$ time]** |
|---|---|---|---|
| 0 | 6 | (16, 6) | |
| 1 | 2 | (17, 7) | |
| 2 | – 2 | (18, 7) | |
| 3 | 14 | (19, 8) | |
| 4 | 10 | (20, 9) | |
| 5 | 6 | (21, 10) | |
| 6 | 2 | (22, 11) | |
| 7 | – 2 | (23, 11) | |
| 8 | 14 | (24, 12) | |
| 9 | 10 | (25, 13) | |

If $p_k > 0$ then increase both X and Y and $p_{k+1} = p_k + 2\Delta y - 2\Delta x$

If $p_k < 0$ then increase X and not Y and $p_{k+1} = p_k + 2\Delta y$

## Check Your Progress 3

1) No, there is no need to generate the full circumference of the circle using the algorithm. We may generate it in a quadrant or octant only and then use it to produce the rest of the circumference by taking reflections along the respective axis.

2) Refer to the discussed example 4 on page 16.

3)

| **Scan Line Polygon** | **Flood Fill Algorithms** |
|---|---|
| • This algorithm checks and alters the attributes (i.e., characteristics and parameters) of the pixels along the current raster scan line. As soon as it crosses from outside to the inside of a boundary of the specified polygon or other region, it starts resetting the colour (or gray) attribute, in effect filling the region along that scan line. It changes back to the original attribute when it crosses the boundary again. | • Flood Fill Algorithms**.** In these, the algorithm starts with the coordinates of a specified point (known as the "seed") inside the polygon to be filled, and proceeds outwards from it, testing the adjacent pixels around it in orderly fashion, until the spreading wave reaches the boundary in every direction. |
| • Polygon filling is time consuming. | • Polygon filling is quite quick. |