gradeup

Computer Graphics Part -2

Computer Graphic Part-2

Content:

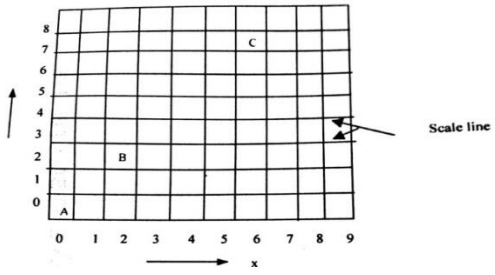
1. Graphics primitives
2. DDA
3. BRESENHAM LINE GENERATION ALGORITHM
4. Viewing and clipping
   a. CHONE SUTHERLAND ALGORITHM

GRAPHIC PRIMITIVES

POINT 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. Every pixel on the arrange surface has a finite size depending on the screen resolution and hence, a pixel cannot represent a single mathematical point. We examine every 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. Every 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 number) start at bottom and increase upwards.



Above figure 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. 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. Screen pixels are mentioned 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 45^0 lines only. Standard algorithm are available to determine which pixels provide the best approximation to the desire line, we will discuss such algorithms in our next section. Screen 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 below figure, 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.

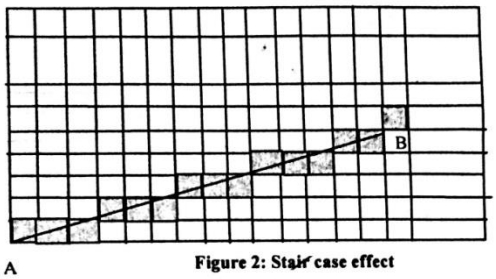
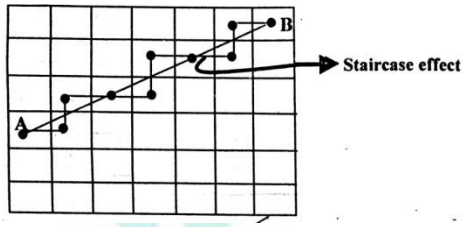


Figure 2: Stair case effect

LINE GENERATION ALGORITHMS

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 pointes say A and B is known as the Staircase effect. The concept is shown below in figure.

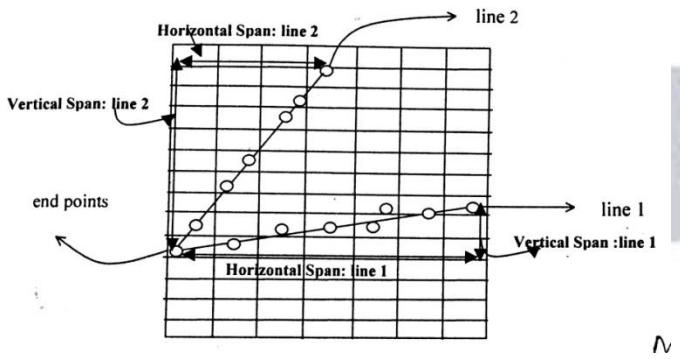


So, from the above figure, I think, it is clear to you that when a line to be drawn is simply described by its end points, then it can be plotted by making close approximation of the pixels which best suit the line, and this approximation is responsible for the staircase effect, which miss projects the 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 Analysis) Algorithm and Bresenham line generation Algorithm next.

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 45^0 lines only. Standard algorithms are available to determine which pixels provide the best approximation to the desire line, one such algorithm is the DDA (Digital Differential Analyser) algorithm. Before going to the details of the algorithms, 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 ot 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 45^0 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 below figure.



In above figure, there are two lines. Line 1 (slope < 1) and line 2 (slope > 1). Now consider the general mechanism of construction of these two liens with the DDA algorithm. As the slope of the line is a crucial factor in its construction, let us consider the algorithm in two depending on the slope of the lien 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 tie and proper x is approximated.

Assumption: The lien generation through DDA is discussed only for the 1st Quadrant, if the line lies in any other quadrant then apply respective transformation (generally reflection transformation), such that it lies in 1st 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 transformation 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 units 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 above figure, 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 step and approximate Y (this choice will also reduce staircase effect distortion in line generation) (∵ 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 → $x_i = (x_{i-1}) + 1$ 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 to be a floating number then we round its value to select the approximating pixel. This rounding off feature contributes to the staircase effect.

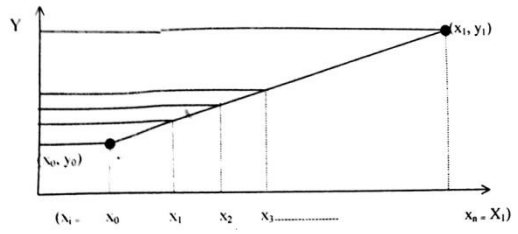
   When (infinity) > m > 1 increment Y in unit steps and approximate X, simplify (x_i, y_i) accordingly.

   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), suppose that $X_1 > X_0$ is true else swap (X_0, Y_0) and (X_1, Y_1). I f $(X_1 - X_0 > Y_1 - Y_0)$ that means slope (m) of line is < 1 hence, case 1 for line generation is applicable. We need to increase X in unit steps and approximate Y. So, from X_0 to X_1, x_i is increased 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.

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

   We will examine each case separately.



SUM UP:

In a given line the end points are (X_0, Y_0) and (X_1, Y_1). Utilise 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.

Suppose equation of line is $y = mx + c$

At $x = x_i$ we have $y_i = mx_i + c$

at $x = x_{i+1}$ we have $y_{i+1} = mx_{i+1} + c$

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

x is to be increase by 1 unit each time

$\rightarrow x_{i+1} = x_i + 1$ .....(1)

Using equation of line $y = mx + c$ we have

$$y_{i+1} = m(x_i + 1) + c$$
$$= mx_i + c + m$$
$$= y_i + m \quad .....(2)$$

Equation (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 net value of y.

Now, using the starting point (x_0, y_0) in the above equation (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 supposed that $X_0 < X_1$ ∴ the algorithm goes like:

$x \leftarrow X_0$

$y \leftarrow Y_0$

$m \leftarrow (Y_1 - Y_0) / (X_0 - X_1)$

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) $y + 1$

Simple 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, they component is increased by one unit each time and appropriate x component is to be selected. To do the task of appropriate selection of component we use the equation of Line: y = mx + c.

Unit increment should be iterative → $y_{i+1} = y_i + 1$; 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}).

→ $y_{i+1} - y_i = m(x_{i+1} - x_i)$ .....(3)

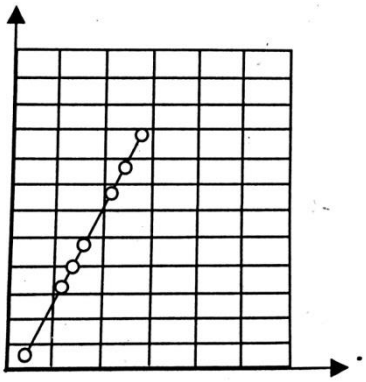
As y to be increased by unit steps

∴ $y_{i+1} - y_i = 1$ .....(4)

Using equation (4) and (3) we get

→ $1 = m\ (x_{i+1} - x_i)$

.....(5)



Rearranging (5) we get

→ $1/m = (x_{i+1} - x_i)$

→ $x_{i+1} = x_i + \frac{1}{m}$

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

Algorithm for case 2:

$x \leftarrow X_0$;

$y \leftarrow Y_0$;

$m \leftarrow (Y_1 - Y_0)/(X_1 - X_0)$;

$m_1 \leftarrow 1/m$;

{

Put-pixel (round (x), y, colour)

$y \leftarrow y + 1$;

$x \leftarrow x + m_1$;

X

}

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

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

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

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

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

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) $x_1 = x_0 + 1 = 3$

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

put pixel (x_0 round y, colour)

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

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

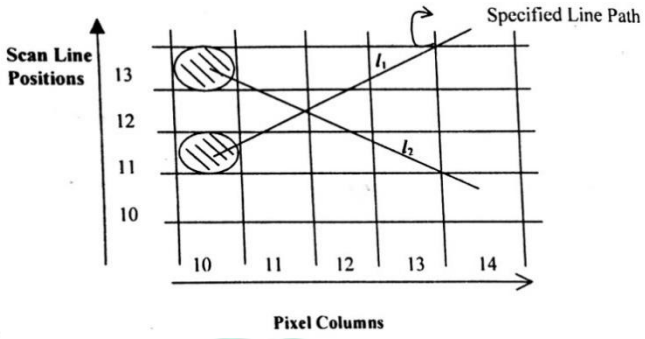
$y_2 = y_1 + m = (33/7) + 5/7 = 38/7 = 5\frac{5}{7}$

put on (4, 5)

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

BRESENHAM LINE GENERATION ALGORITHM

It 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.



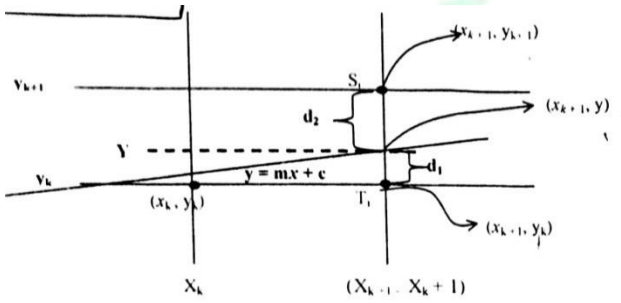
Sampling at Unit x distance in above figure, we need to decide which of the two possible pixel position is closer to the line path at each sample step.

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<0<1). If the slope is negative then, use reflection transformation 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 k^{th} step. The process is shown in below figure. 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}).



At stamping 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:

$$Y = m(x_{k+1}) + c \qquad \text{.....(1)}$$

By above figure 9:

$$d_1 = y - y_k \qquad \text{.....(2)}$$

$$m(x_{k-1}) + c - y_k \qquad \text{.....(3) (using (1)}$$

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 k^{th} step i.e., p_k is given by

$P_k = \Delta x(d_1 - d_2)$ .....(6)

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

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

$= \Delta x[2(\Delta y/\Delta x)\ (x_{k+1}) - 2y_k + 2c - 1]$

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

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

$P_k = 2\ \Delta y\ x_k - 2xy_k + b$ .....(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 above figure, $d_1 < d_2$ i.e. $(d_1 - d_2)$ is -ve i.e., p_k is -ve so pixel T_i is more appropriate chice 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+1}) 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:

$P_{k+1} = 2\Delta y\ x_{k+1} + b$ .....(9)

Subtracting (7) from (9) we get

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

depending on sign of p_k

$$\begin{cases} p_k < 0 & y_{k+1} = y_k \\ p_k > 0 & y_{k+1} = y_k + 1 \end{cases}$$

$$\boxed{p_{k+1} = p_k + 2\Delta y - 2\Delta x(y_{k+1} - y_k)} \quad \text{----------(10)}$$

This recursive calculation of decision parameter is performed at each integer positions, 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)

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

$p_0 = 2\Delta y - \Delta x$

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 Δx, Δy, $2\Delta y - 2\Delta x$ 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) Δx times.

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

$\Delta x = 5$

→ 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: Illustrated the Bresenham line generation algorithm by digitzing the line with end points (20, 10) and (30, 18).

Solution:

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

$\Rightarrow \Delta y = 8$ 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 $(x_0, y_0) = (20, 10)$ 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}) ← [use step (d) of algorithm Δx times]
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)

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 $p_{k+1} = p_k + 2\Delta y$

gradeup SUPER

Gradeup UGC NET

Super Superscription

Features:

1. 7+ Structured Courses for UGC NET Exam
2. 200+ Mock Tests for UGC NET & MHSET Exams
3. Separate Batches in Hindi & English
4. Mock Tests are available in Hindi & English
5. Available on Mobile & Desktop

Gradeup Super Subscription, Enroll Now

gradeup.co