Computer Graphics (CSC209)

III semester, BSCCSIT

Compiled by : Ankit Bhattarai

Syllabus

Unit	Contents	Hours	Remarks
1.	Introduction of Computer Graphics	3	
2.	**Scan Conversion Algorithm**	6	Important
3.	Two-Dimensional Geometric Transformations	5	
4.	Three-Dimensional Geometric Transformation	5	
5.	3D Objects Representation	7	
6.	Solid Modeling	4	
7.	Visible Surface Detections	5	
8.	Illumination Models and Surface Rendering Techniques	5	
9.	Introduction to Virtual Reality	2	
10.	Introduction to OpenGL	3	

Lab Works in C++ or Python (Open GL)

Credit : 3

Chapter 2

(6 hrs.)

Scan Conversion Algorithm

1. Scan Converting a Point and a straight Line: DDA Line Algorithm, Bresenham's Line Algorithm
2. Scan Converting Circle and Ellipse :Mid Point Circle and Ellipse Algorithm
3. Area Filling: Scan Line Polygon fill Algorithm, Inside-outside Test, Scan line fill of Curved Boundary area, Boundary-fill and Flood-fill algorithm.

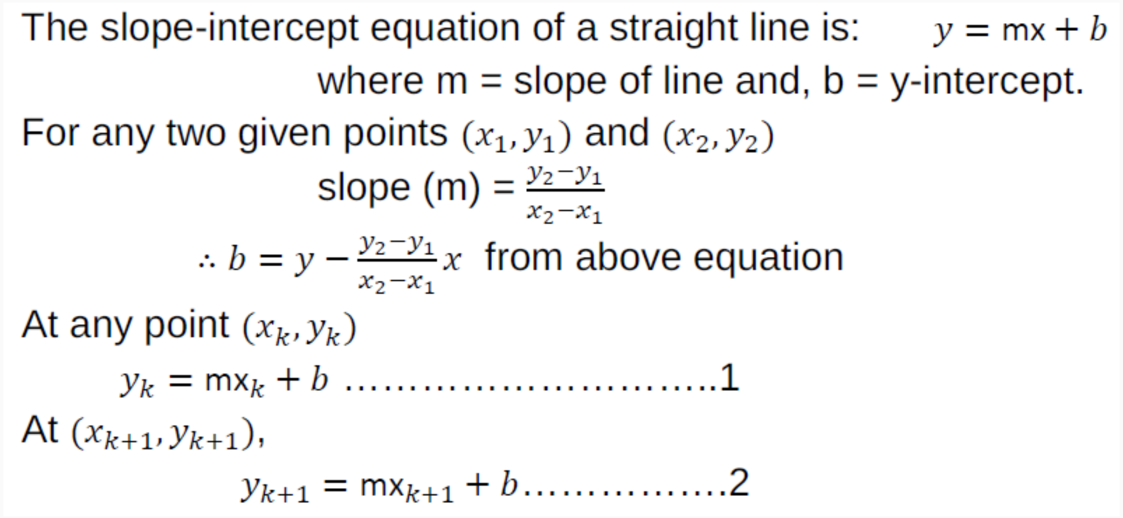
Scan Conversion

Output Primitive and Line Scan Conversion

Output primitives

- They are the basic primitives structure such as points, straight line segments, circles, curves and surfaces which are used to describe a scene.

Line Scan Conversion



The slope-intercept equation of a straight line is: $y = mx + b$

where m = slope of line and, b = y-intercept.

For any two given points (x_1, y_1) and (x_2, y_2)

$$\text{slope (m)} = \frac{y_2 - y_1}{x_2 - x_1}$$

$$\therefore b = y - \frac{y_2 - y_1}{x_2 - x_1} x \text{ from above equation}$$

At any point (x_k, y_k)

$$y_k = mx_k + b \text{ ....1}$$

At (x_{k+1}, y_{k+1}),

$$y_{k+1} = mx_{k+1} + b \text{.................2}$$

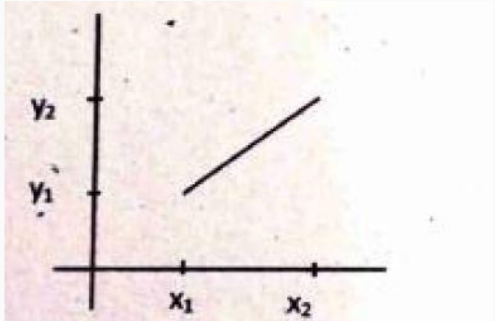
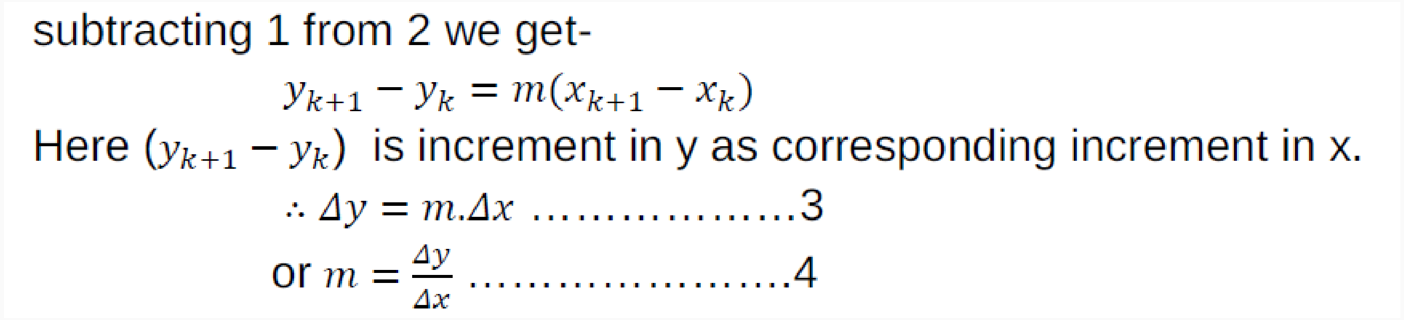


Fig. 2.1: Line path between two endpoint positions

Line Scan Conversion



subtracting 1 from 2 we get-

$$y_{k+1} - y_k = m(x_{k+1} - x_k)$$

Here $(y_{k+1} - y_k)$ is increment in y as corresponding increment in x.

$$\therefore \Delta y = m.\Delta x \;\ldots\ldots\ldots\ldots\ldots\ldots 3$$

$$\text{or } m = \frac{\Delta y}{\Delta x} \;\ldots\ldots\ldots\ldots\ldots\ldots\ldots 4$$

When the value of m is calculated from equation 4 there are three cases that arises:

Case I: |m| < 1

Case II: |m| > 1

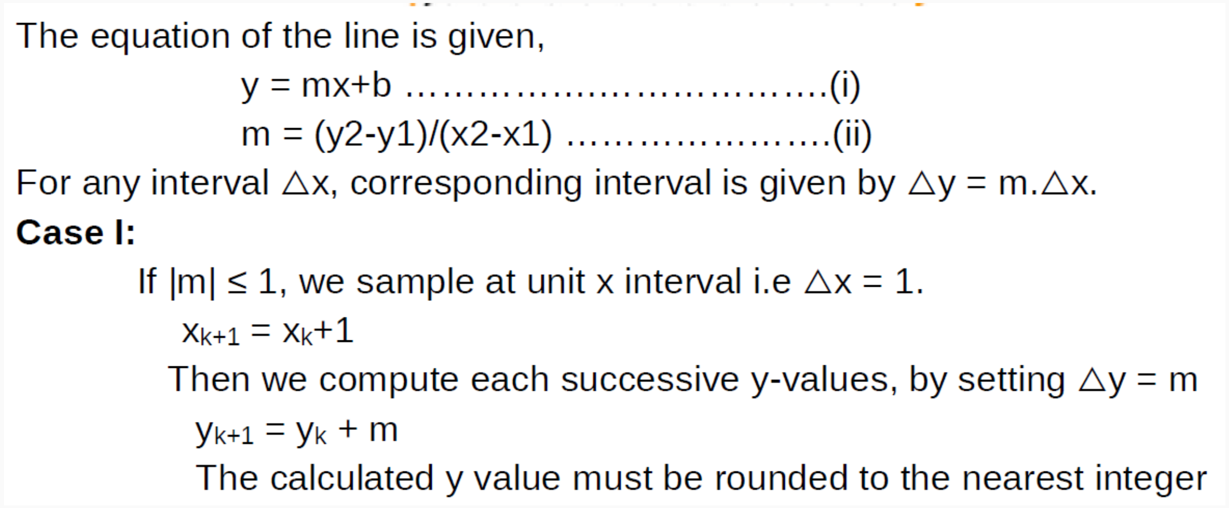
Case III: |m| = 1

Digital Differential Analyzer (DDA)

DDA

- It is a scan conversion line algorithm based on calculation either △x or △y using equation **m=△y/△x**.
- We sample the line at unit interval in one direction (x if △x is greater than △y, otherwise in y direction) and determine the corresponding integer values nearest the line path for the other coordinate..

Algorithm



The equation of the line is given,

$$y = mx+b \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (i)$$

$$m = (y2-y1)/(x2-x1) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots (ii)$$

For any interval △x, corresponding interval is given by △y = m.△x.

Case I:

If |m| ≤ 1, we sample at unit x interval i.e △x = 1.

$x_{k+1} = x_k+1$

Then we compute each successive y-values, by setting △y = m

$y_{k+1} = y_k + m$

The calculated y value must be rounded to the nearest integer

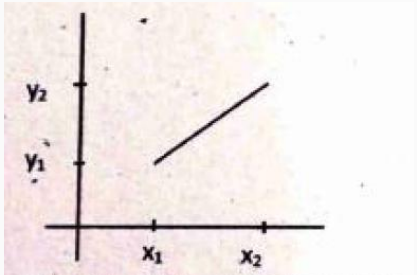
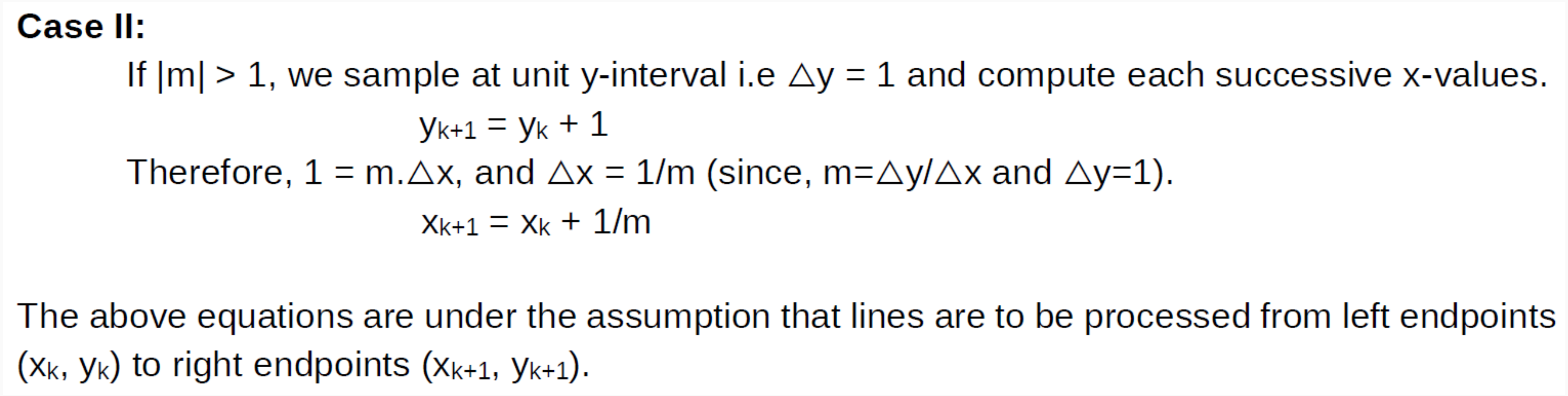


Fig. 2.1: Line path between two endpoint positions

DDA

Algorithm



Case II:

If $|m| > 1$, we sample at unit y-interval i.e $\triangle y = 1$ and compute each successive x-values.

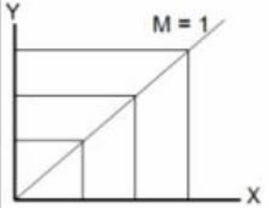
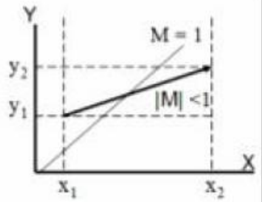
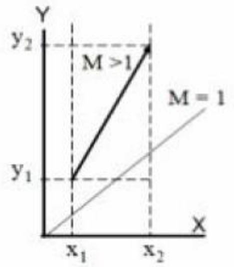
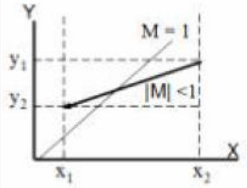
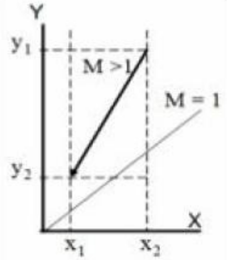
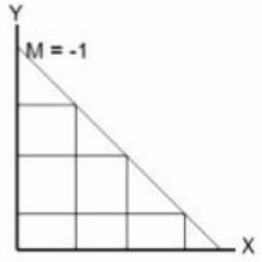
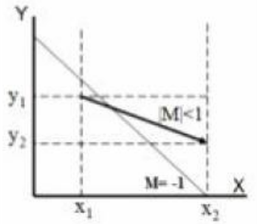
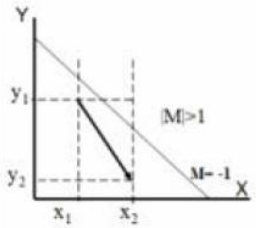
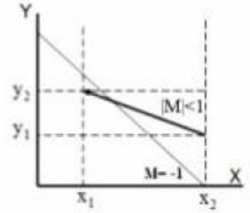
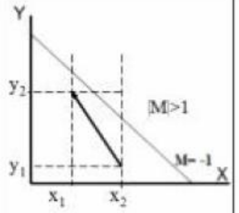
$$y_{k+1} = y_k + 1$$

Therefore, $1 = m.\triangle x$, and $\triangle x = 1/m$ (since, $m=\triangle y/\triangle x$ and $\triangle y=1$).

$$x_{k+1} = x_k + 1/m$$

The above equations are under the assumption that lines are to be processed from left endpoints (x_k, y_k) to right endpoints (x_{k+1}, y_{k+1}).

DDA

	Moving Left to Right		Moving Right to Left	
	Slope (m) < 1	Slope (m) > 1	Slope (m) < 1	Slope (m) > 1
Positive Slope  *Fig. Positive Slope*	 Y M = 1 y_2 y_1 |M| <1 X x_1 x_2 $x_{k+1} = x_k + 1$ $y_{k+1} = y_k + m$	 $x_{k+1} = x_k + \frac{1}{m}$ $y_{k+1} = y_k + 1$	 Y M = 1 y_1 y_2 |M| <1 X x_1 x_2 $x_{k+1} = x_k - 1$ $y_{k+1} = y_k - m$	 $x_{k+1} = x_k - \frac{1}{m}$ $y_{k+1} = y_k - 1$
Negative Slope  *Fig. Negative Slope*	 Y y_1 y_2 |M|<1 M= -1 X x_1 x_2 $x_{k+1} = x_k + 1$ $y_{k+1} = y_k - m$	 Y y_1 |M|>1 y_2 M= -1 X x_1 x_2 $x_{k+1} = x_k + \frac{1}{m}$ $y_{k+1} = y_k - 1$	 Y y_2 y_1 |M|<1 M= -1 X x_1 x_2 $x_{k+1} = x_k - 1$ $y_{k+1} = y_k + m$	 Y y_2 |M|>1 M= -1 y_1 X x_1 x_2 $x_{k+1} = x_k - \frac{1}{m}$ $y_{k+1} = y_k + 1$

DDA

Pros of DDA

1. It is simple to understand.
2. It is faster method than direct use of line equation $y= mx + c$. (Removes multiplication operation and only involve increments operation of x or y direction)
3. It requires no special skills for implementation

Cons of DDA

1. A floating point addition is still needed in determining each successive point which is time consuming.
2. The accumulation of round off error in successive addition of the floating-point increments may cause the calculated pixel position to drift away from the true line path for long line segment.

Numerical on DDA

DDA

Consider a line from (2,1) to (8,3). Using simple DDA algorithm, rasterize this line.

Given,

Starting Point :(x1, y1)= (2,1)

Ending Point: (x2,y2) = (8,3)

Now,

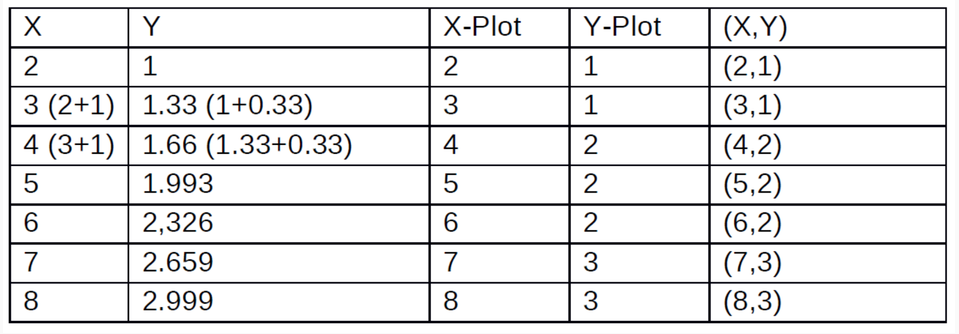
Slope m = (y2-y1)/(x2-x1)= (3-1)/(8-2)

= (1/3) = 0.333

Since |m|<1, From DDA algorithm we have

$X_{k+1} = X_k + 1$

$Y_{k+1} = Y_k + m$



X	Y	X-Plot	Y-Plot	(X,Y)
2	1	2	1	(2,1)
3 (2+1)	1.33 (1+0.33)	3	1	(3,1)
4 (3+1)	1.66 (1.33+0.33)	4	2	(4,2)
5	1.993	5	2	(5,2)
6	2,326	6	2	(6,2)
7	2.659	7	3	(7,3)
8	2.999	8	3	(8,3)

DDA

Digitized the line with end points (0,0) and (4,5) using DDA.

Given Starting Point :(x1, y1)= (0,0) and Ending

Point: (x2,y2) = (4,5)

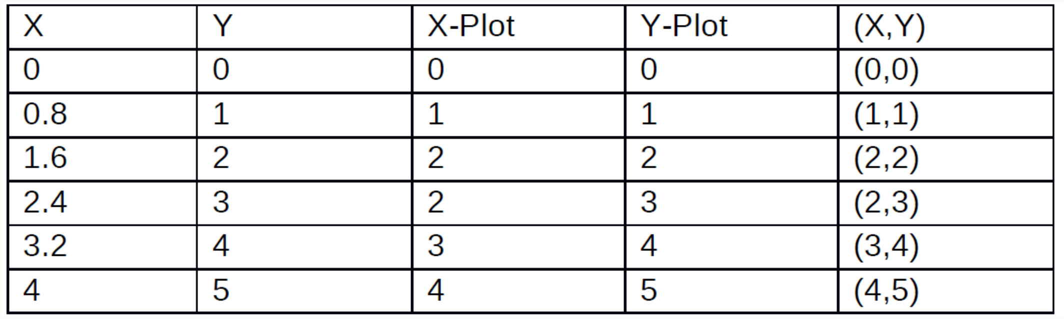
Now Slope m=(y2-y1)/(x2-x1)= (5-0)/(5-0) =(5/4) =

1.25

Since |m|>1, From DDA algorithm we have

$y_{k+1} = y_k + 1$

$x_{k+1} = x_k + (1/m)$



X	Y	X-Plot	Y-Plot	(X,Y)
0	0	0	0	(0,0)
0.8	1	1	1	(1,1)
1.6	2	2	2	(2,2)
2.4	3	2	3	(2,3)
3.2	4	3	4	(3,4)
4	5	4	5	(4,5)

DDA

Digitized the line with end points (3,7) and (8,3) using DDA

Given Starting Point :(x1, y1)= (3,7) and Ending

Point: (x2,y2) = (8,3)

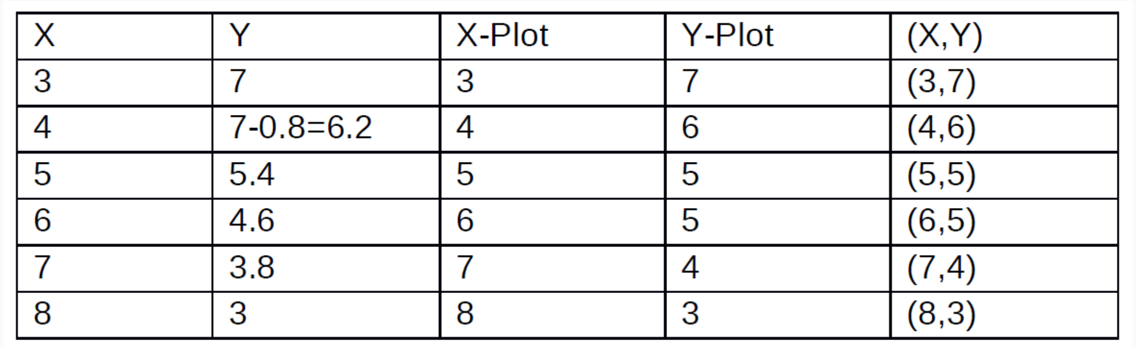
Now,

Slope m=(y2-y1)/(x2-x1)= (3-7)/(8-3) =(-4/5) = -0.8

Since |m|<1, From DDA algorithm we have

$X_{k+1} = X_k + 1$

$Y_{k+1} = Y_k + m$



X	Y	X-Plot	Y-Plot	(X,Y)
3	7	3	7	(3,7)
4	7-0.8=6.2	4	6	(4,6)
5	5.4	5	5	(5,5)
6	4.6	6	5	(6,5)
7	3.8	7	4	(7,4)
8	3	8	3	(8,3)

Bresenham's Line Algorithm (BLA)

Bresenham's Line Algorithm (BLA)

- The Bresenham Line Algorithm is an efficient method for drawing a line between two points in a discrete plane, such as a computer screen composed of pixels.
- The basic idea of the algorithm **is to determine the pixels that should be selected to approximate the straight line** between two given points.
- The algorithm **avoids floating-point arithmetic**, which was particularly beneficial in early computer systems where floating-point operations were more computationally expensive.
- The Bresenham algorithm is efficient because **it only involves integer addition and subtraction, avoiding the need for multiplication or division**. This makes it suitable for real-time graphics applications.

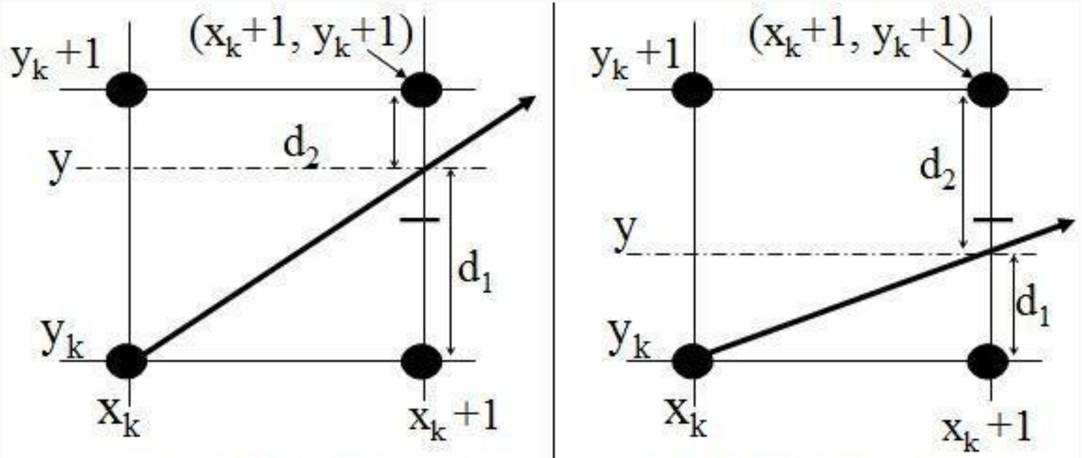
Bresenham's Line Algorithm (BLA)

The BLA is a more efficient method used to plot pixel position along a straight-line path.

Advantage of BLA over DDA

- In DDA algorithm each successive point is computed in ***floating point***, so it requires ***more time*** and ***more memory space***. While in BLA each successive point is calculated in ***integer value*** or ***whole number***. So it requires less time and less memory space.
- In DDA, since the calculated point value is floating point number, it should be rounded at the end of calculation but in BLA it does not need to round, so there is no accumulation of rounding error.
- Due to rounding error, the line drawn by DDA algorithm is not accurate, while in BLA line is accurate.
- DDA algorithm cannot be used in other application except line drawing, but BLA can be implemented in other application such as circle, ellipse and other curves.

BLA for slope +ve and |m| ≤ 1



$P_k = d_1 - d_2$ = Positive

$x_k = x_k + 1$

$y_k = y_k + 1$

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

$P_k = d_1 - d_2$ = Negative

$x_k = x_k + 1$

$y_k = y_k$

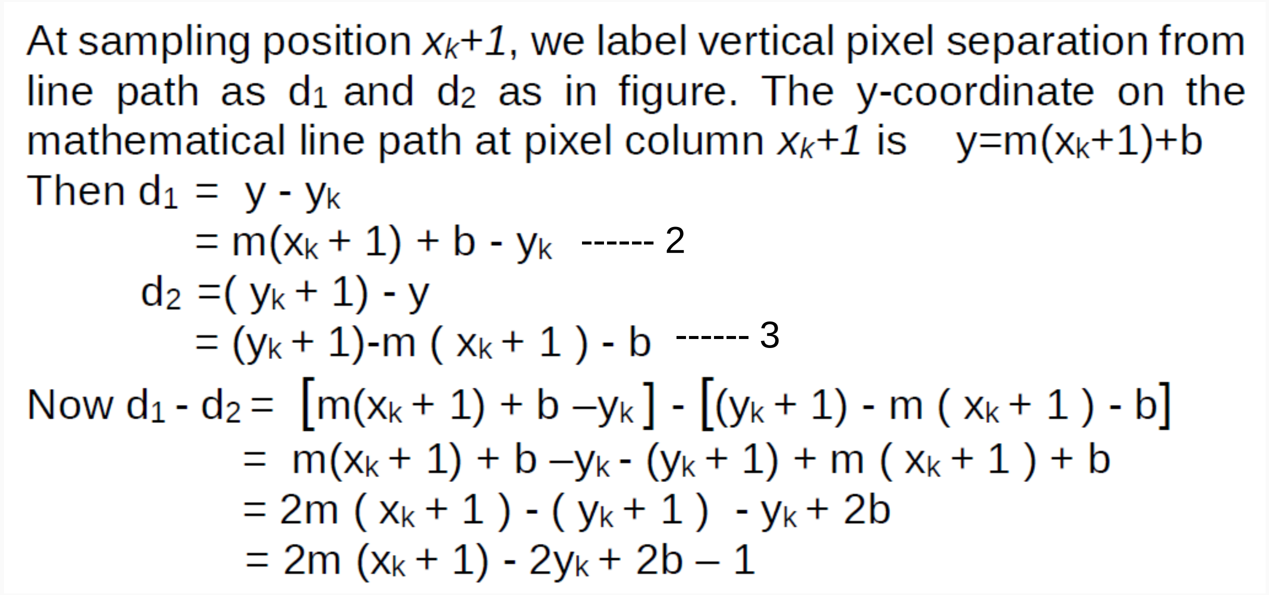
$P_k = P_k + 2\Delta y$

Fig. Bresenham's Line Algorithm for $|m| \leq 1$

BLA for slope +ve and $|m| \leq 1$

Assume that (x_k, y_k) is pixel at kth step then next point to plot may be either $(x_k + 1, y_k)$ or (x_{k+1}, y_{k+1})

$$y = m(x_k + 1) + b \text{ ------ } 1$$



At sampling position x_k*+1*, we label vertical pixel separation from line path as d_1 and d_2 as in figure. The y-coordinate on the mathematical line path at pixel column x_k*+1* is $y = m(x_k+1)+b$

Then $d_1 = y - y_k$

$= m(x_k + 1) + b - y_k$ ------ 2

$d_2 = (y_k + 1) - y$

$= (y_k + 1) - m(x_k + 1) - b$ ------ 3

Now $d_1 - d_2 = [m(x_k + 1) + b - y_k] - [(y_k + 1) - m(x_k + 1) - b]$

$= m(x_k + 1) + b - y_k - (y_k + 1) + m(x_k + 1) + b$

$= 2m(x_k + 1) - (y_k + 1) - y_k + 2b$

$= 2m(x_k + 1) - 2y_k + 2b - 1$

BLA for slope +ve and $|m| \leq 1$

A decision parameter p_k for the kth step in the line algorithm can be obtained by substituting $m = \frac{\Delta y}{\Delta x}$ in above eqn and defining

$$\begin{aligned} p_k &= \Delta x(d_1 - d_2) \\ &= \Delta x[2\frac{\Delta y}{\Delta x}(x_k + 1) - 2y_k + 2b - 1] \\ &= 2\Delta y.x_k - 2\Delta x.y_k + 2\Delta y + \Delta x(2b - 1) \quad \text{------ 4} \\ &= 2\Delta y.x_k - 2\Delta x.y_k + c \end{aligned}$$

Where the constant $c = 2\Delta y + \Delta x(2b - 1)$ which is independent.

If decision parameter p_k is negative i.e. $d_1 < d_2$, pixel at y_k is closer to the line path than pixel at y_k+1 . In this case we plot lower pixel $(x_k + 1, y_k)$ other wise plot upper pixel (x_k+1, y_k+1).

BLA for slope +ve and |m| ≤ 1

At step k+1, p_{k+1} is evaluated as:

$$p_{k+1} = 2\Delta y . x_{k+1} - 2\Delta x y_{k+1} + c$$

We get,

$$p_{k+1} - p_k = 2\Delta y(x_{k+1} - x_k) - 2\Delta x(y_{k+1} - y_k)$$

Since we are sampling at X-direction, we have $x_{k+1} = x_k + 1$

$$\therefore p_{k+1} = p_k + 2\Delta y - 2\Delta x(y_{k+1} - y_k)$$

The term y_k is either 0 or 1 depending upon the sign of p_k.

So,

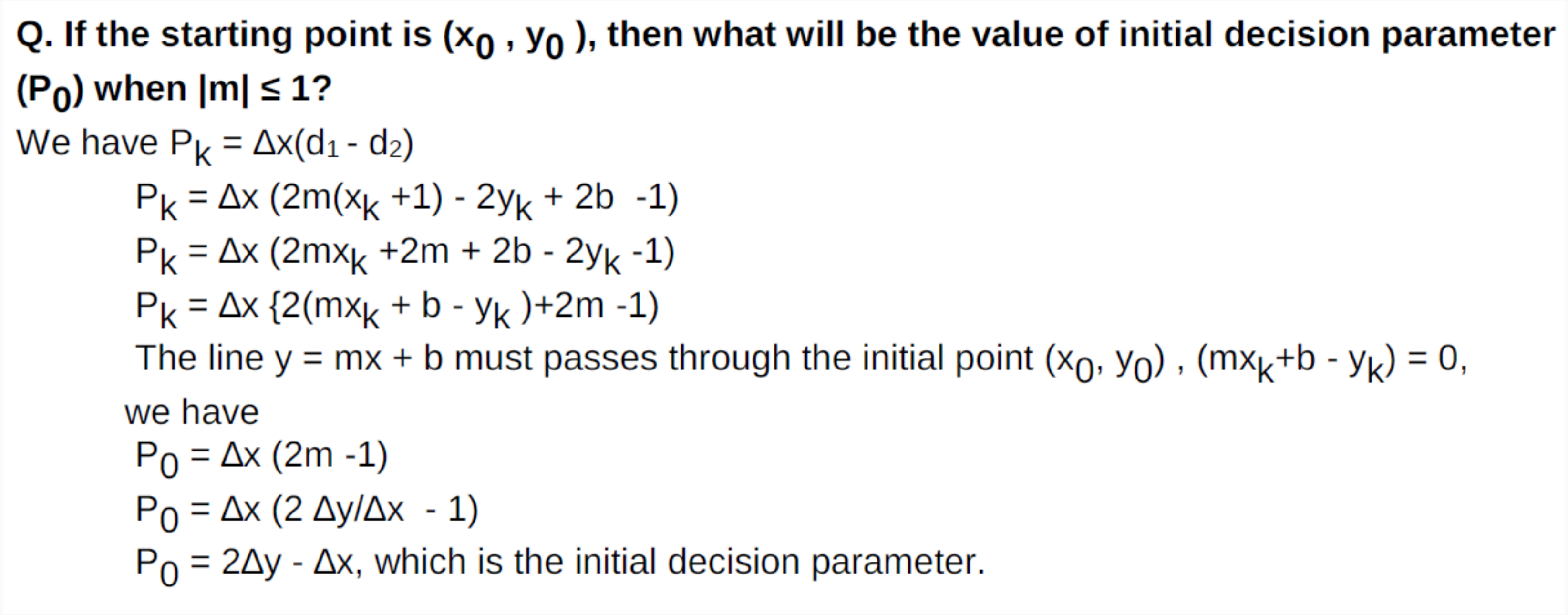
if p_k is negative

$y_{k+1} = y_k$ and $P_{k+1} = p_k + 2\Delta y$

otherwise

$y_{k+1} = y_k + 1$, then $p_{k+1} = p_k + 2\Delta y - 2\Delta x$

BLA for slope +ve and $|m| \le 1$



Q. If the starting point is (x_0, y_0), then what will be the value of initial decision parameter (P_0) when $|m| \le 1$?

We have $P_k = \Delta x(d_1 - d_2)$

$P_k = \Delta x\ (2m(x_k + 1) - 2y_k + 2b - 1)$

$P_k = \Delta x\ (2mx_k + 2m + 2b - 2y_k - 1)$

$P_k = \Delta x\ \{2(mx_k + b - y_k) + 2m - 1)$

The line $y = mx + b$ must passes through the initial point (x_0, y_0) , $(mx_k + b - y_k) = 0$, we have

$P_0 = \Delta x\ (2m - 1)$

$P_0 = \Delta x\ (2\ \Delta y/\Delta x - 1)$

$P_0 = 2\Delta y - \Delta x$, which is the initial decision parameter.

Digitize a line with endpoints (20,10) and (30,18) using BLA.

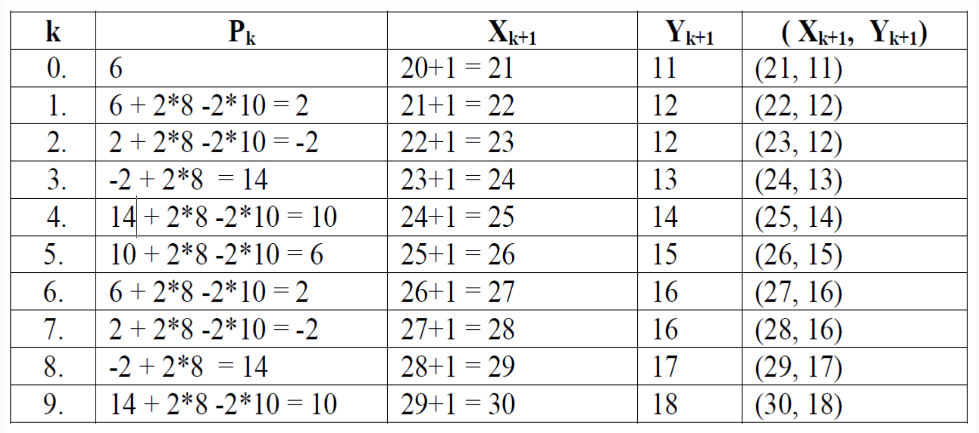
Thus, The initial decision parameter $(P0) = 2\Delta y - \Delta x = 2*8 - 10 = 6$

Since, for the Bresenham's Line drawing Algorithm of slope, $|m| \leq 1$, we have

If $P_k < 0$ (i.e. d1-d2 is Negative)
then,
$X_{k+1} = X_k+1$
$Y_{k+1} = Y_k$
$\mathbf{P_k = P_k + 2\,\Delta y}$

If $P_k \geq 0$
then,
$X_{k+1} = X_k+1$
$Y_{k+1} = Y_k+1$
$\mathbf{P_k = P_k + 2\,\Delta y - 2\Delta x}$

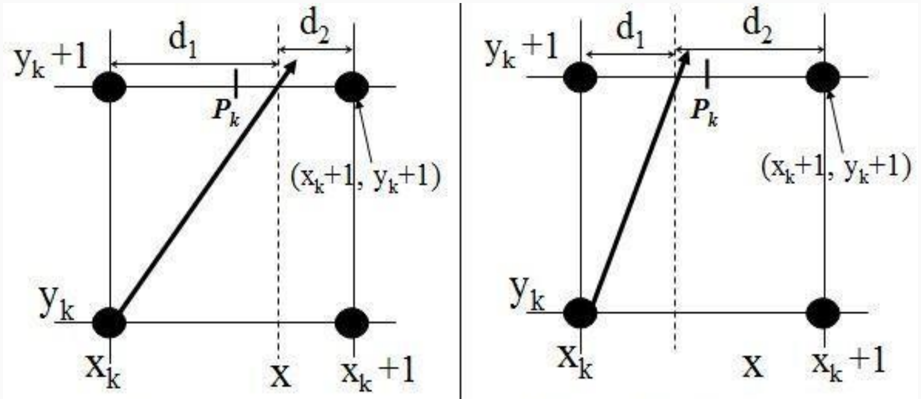
(20, 10)



k	P_k	X_{k+1}	Y_{k+1}	(X_{k+1}, Y_{k+1})
0.	6	20+1 = 21	11	(21, 11)
1.	6 + 2*8 -2*10 = 2	21+1 = 22	12	(22, 12)
2.	2 + 2*8 -2*10 = -2	22+1 = 23	12	(23, 12)
3.	-2 + 2*8 = 14	23+1 = 24	13	(24, 13)
4.	14 + 2*8 -2*10 = 10	24+1 = 25	14	(25, 14)
5.	10 + 2*8 -2*10 = 6	25+1 = 26	15	(26, 15)
6.	6 + 2*8 -2*10 = 2	26+1 = 27	16	(27, 16)
7.	2 + 2*8 -2*10 = -2	27+1 = 28	16	(28, 16)
8.	-2 + 2*8 = 14	28+1 = 29	17	(29, 17)
9.	14 + 2*8 -2*10 = 10	29+1 = 30	18	(30, 18)

BLA for slope +ve and |m| ≤ 1

BLA for slope +ve and |m| > 1



$P_k = d_1 - d_2 = \text{Positive}$

$x_k = x_k + 1$

$y_k = y_k + 1$

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

$P_k = d_1 - d_2 = \text{Negative}$

$x_k = x_k$

$y_k = y_k + 1$

$P_k = P_k + 2\Delta x$

Fig. Bresenham's Line Algorithm for |m| > 1

Thus, the initial decision parameter

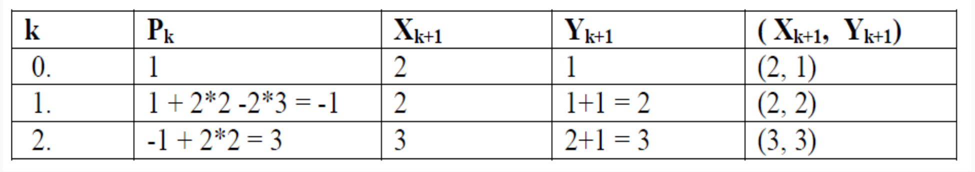
$(P0) = 2\Delta x - \Delta y = 2*2 - 3 = 1$

we have

$\Delta x = x2\text{-}x1 = 3\text{-}1 = 2$
$\Delta y = y2\text{-}y1 = 3\text{-}0 = 3$

If $P_k < 0$
then,
$X_{k+1} = X_k$
$Y_{k+1} = Y_k+1$
$\mathbf{P_k = P_k + 2\,\Delta x}$

If $P_k \geq 0$
then,
$X_{k+1} = X_k+1$
$Y_{k+1} = Y_k+1$
$\mathbf{P_k = P_k + 2\,\Delta x - 2\Delta y}$

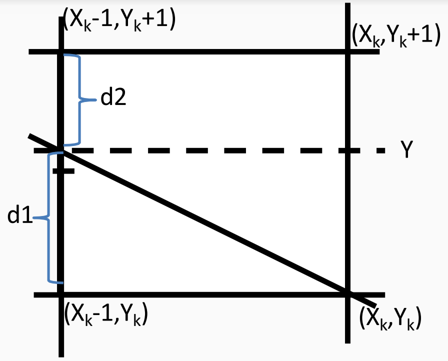


k	P_k	X_{k+1}	Y_{k+1}	(X_{k+1}, Y_{k+1})
0.	1	2	1	(2, 1)
1.	1 + 2*2 -2*3 = -1	2	1+1 = 2	(2, 2)
2.	-1 + 2*2 = 3	3	2+1 = 3	(3, 3)

Digitize a line with endpoints (1,0) and (3,3) using BLA.

BLA for slope +ve and |m| > 1

BLA: Slope – ve and |M| ≤ 1



d1-d2 > 0 (Positive)

$X_{k+1} = X_k - 1$

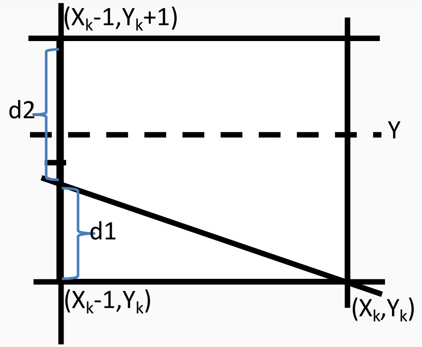
$Y_{k+1} = Y_k + 1$

If $P_k \geq 0$

$X_{k+1} = X_k - 1$

$Y_{k+1} = Y_k + 1$

$\mathbf{P_k = P_k + 2\,\Delta y - 2\Delta x}$



d1-d2 < 0 (Negative)

$X_{k+1} = X_k - 1$

$Y_{k+1} = Y_k$

If $P_k < 0$

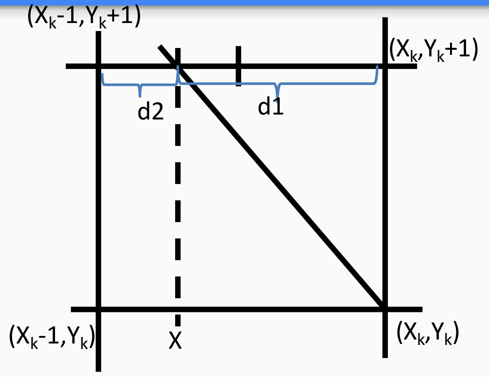
$X_{k+1} = X_k - 1$

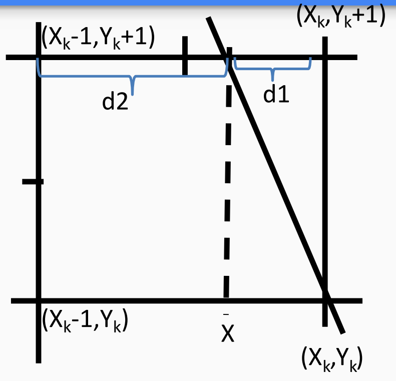
$Y_{k+1} = Y_k$

$\mathbf{P_k = P_k + 2\,\Delta y}$

$\mathbf{P_0 = 2\,\Delta y - \Delta x}$

Slope –ve and |M| > 1





d1-d2 > 0 (Positive)

$X_{k+1} = X_k - 1$

$Y_{k+1} = Y_k + 1$

If $P_k \geq 0$

$X_{k+1} = X_k - 1$

$Y_{k+1} = Y_k + 1$

$\mathbf{P_k = P_k + 2\Delta x - 2\,\Delta y}$

d1-d2 < 0 (Negative)

$X_{k+1} = X_k$

$Y_{k+1} = Y_k + 1$

If $P_k < 0$

$X_{k+1} = X_k$

$Y_{k+1} = Y_k + 1$

$\mathbf{P_k = P_k + 2\,\Delta x}$

$$\mathbf{P_0 = 2\,\Delta x - \Delta y}$$

Solution:

Here, $(X_1,Y_1) = (6,2)$ &

$(X_2,Y_2) = (3,10)$

M= (10-2)/(3-6) = 8/-3 –ve slope and |M| > 1

$\Delta x = |3-6| = 3$

$\Delta y = |10-2| = 8$

here,

Initial $(P_0) = 2\,\Delta x - \Delta y$

$= 2x3-8$

$= -2$

If $P_k < 0$

$X_{k+1} = X_k$

$Y_{k+1} = Y_k + 1$

$P_k = P_k + 2\,\Delta x$

If $P_k \geq 0$

$X_{k+1} = X_k - 1$

$Y_{k+1} = Y_k + 1$

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

(6, 2)

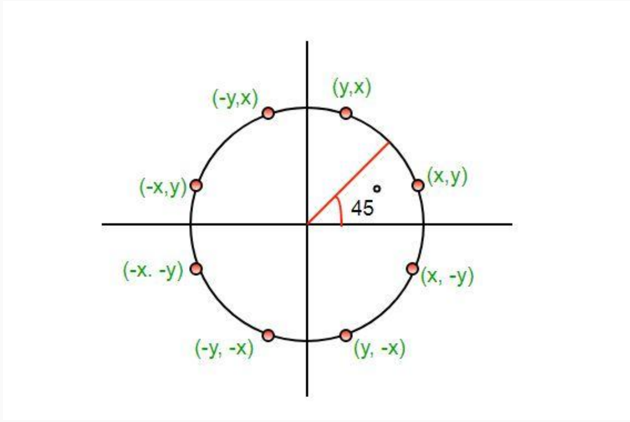
K	P_K	X_{k+1}	Y_{k+1}	(X_{k+1}, Y_{k+1})
0	-2	6	3	(6,3)
1	=-2+2x3 = 4	5	4	(5,4)
2	=4+6-16=-6	5	5	(5,5)
3	=-6+6=0	4	6	(4,6)
4	=0+6-16=-10	4	7	(4,7)
5	=-10+6= -4	4	8	(4,8)
6	=-4 +6 =2	3	9	**(3,9)**
7	= -8	3	10	(3,10)

Digitize line with end points (6,2) and (3,10) using BLA.

Circle Algorithm

What is Circle ?

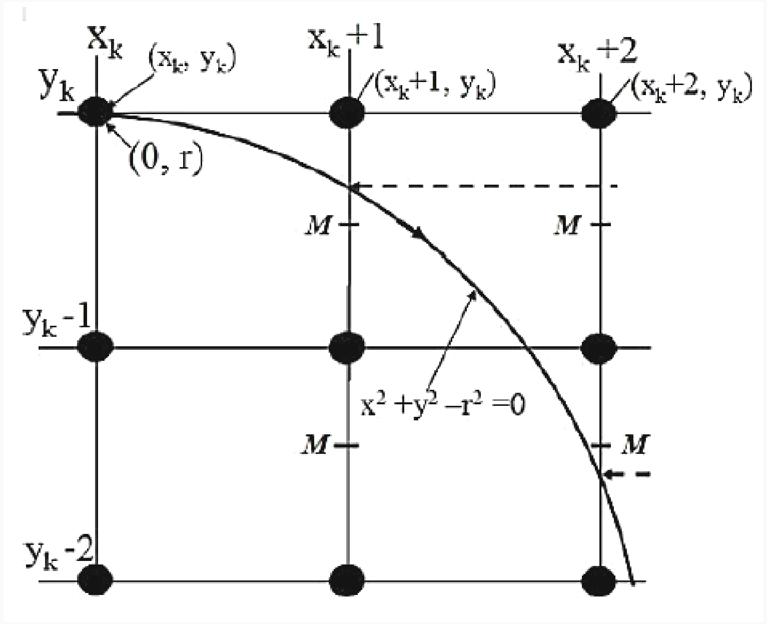
- Similarly to the case with lines, there is an incremental algorithm for drawing circles – the **mid-point circle algorithm**
- **Mid-point circle drawing algorithm is an algorithm used to determine the points needed for rasterizing a circle.**
- In the mid-point circle algorithm we use eight-way symmetry so only ever calculate the points for the top right eighth of a circle, and then use symmetry to get the rest of the points.



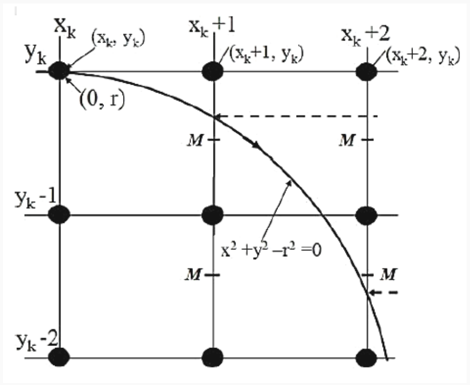
The equation of circle at (0,0) is

$x^2 + y^2 = r^2$ ………. (1)

Mid Point Circle Algorithm



Mid Point Circle Algorithm



Suppose, (x_k,y_k) is the pixel plotted, then the next pixel (x_{k+1}, y_{k+1}) will be either (x_k+1 , y_k) or (x_k+1 ,y_k-1).
Here,
the mid-point (m) = $(x_k+1, y_k-1/2)$.

Now, decision parameter $(P_k) = f_{circle}(x_k+1, y_k-1/2) = (x_k+1)^2 + (y_k-1/2)^2 – r^2$, is the circle function evaluated at midpoint.
Also, the next pixel to plot will either be $(x_{k+1} +1, y_{k+1})$ or $(x_{k+1} +1, y_{k+1} -1)$
Also, the midpoint here is $(x_{k+1} +1, y_{k+1} -1/2)$
And the next decision parameter $P_{k+1} = f_{circle}(x_{k+1} +1, y_{k+1} -1/2)$

$$P_{k+1} = (x_{k+1} +1)^2 + (y_{k+1} -1/2)^2 –r^2$$
$$P_{k+1} = \{(x_k +1) +1\}^2 + (y_{k+1} -1/2)^2 –r^2$$

Mid Point Circle Algorithm

$$P_{k+1} = (x_k + 1)^2 + 2(x_k + 1) + 1 + (y_{k+1} - 1/2)^2 - r^2$$

Now, subtracting P_{k+1} and P_k , we have

$$P_{k+1} - P_k = \{(x_k + 1)^2 + 2(x_k + 1) + 1 + (y_{k+1} - 1/2)^2 - r^2\} - \{(x_k+1)^2 + (y_k - 1/2)^2 - r^2\}$$

$$= 2(x_k + 1) + 1 + y_{k+1}^2 - y_{k+1} + ½^2 - y_k^2 + y_k - ½^2$$

$$= 2(x_k + 1) + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k) + 1$$

If $P_k < 0$, the midpoint is inside the circle and the pixel on the scan line y_k closer to the circle boundary. Otherwise, the midpoint is outside or on the circle boundary, and we select the pixel on scan line $y_k - 1$.

Case $P_k < 0$:

$$x_{k+1} = x_k + 1$$

$$y_{k+1} = y_k$$

$$P_{k+1} = P_k + 2(x_k + 1) + 1$$

$$P_{k+1} = P_k + 2\,x_{k+1} + 1$$

Mid Point Circle Algorithm

Case $P_k >= 0$:

$$x_{k+1} = x_k + 1$$
$$y_{k+1} = y_k - 1$$
$$P_{k+1} = P_k + 2(x_k + 1) + [(y_k - 1)^2 - y_k^2)] - (y_k - 1 - y_k) + 1$$
$$= P_k + 2(x_k + 1) + [(y_k^2 - 2y_k + 1 - y_k^2)] - (y_k - 1 - y_k) + 1$$
$$= P_k + 2(x_k + 1) - 2y_k + 1 + 1 + 1$$
$$= P_k + 2(x_k + 1) - 2y_k + 2 + 1$$
$$= P_k + 2(x_k + 1) – 2(y_k - 1) + 1$$
$$= P_k + 2x_{k+1} – 2y_{k+1} + 1$$

Q. Calculate the initial decision parameter (P_0) for circle

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

Here, the next pixel will either be $(1, r)$ or $(1, r-1)$ where the midpoint is $(1, r- ½)$. Thus, the initial decision parameter is given by:

$$P_0 = f_{circle}(1, r- ½)$$
$$= 1^2 + (r – ½)^2 – r^2$$
$$= 5/4 – r$$
$$\approx 1 – r$$

Thus, $P_0 = 1 – r$

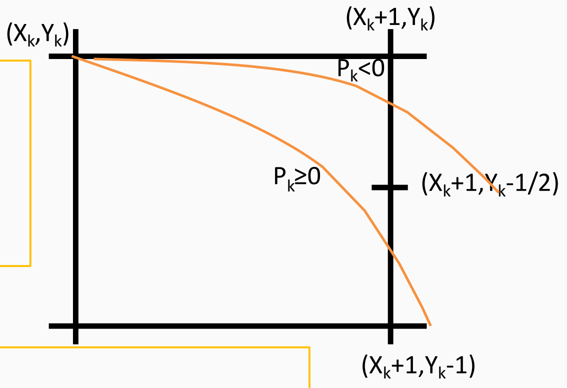
Case 1 : Pk < 0

$X_{k+1} = X_k + 1$

$Y_{k+1} = Y_k$

$P_{k+1} = Pk + 2(X_k+1) + 1$

$$\mathbf{P_{k+1} = P_k + 2X_{k+1} + 1}$$



Case 2: Pk ≥ 0

$X_{k+1} = X_k + 1$

$Y_{k+1} = Y_k - 1$

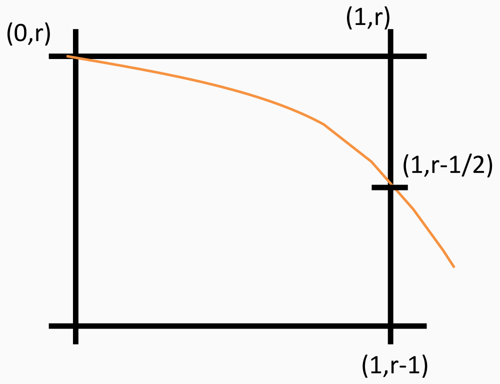
$$\begin{aligned}
P_{k+1} &= P_k + 2(x_k+1) + [(y_k - 1)^2 - y_k^2)] - (y_k - 1 - y_k) + 1 \\
&= P_k + 2(x_k+1) + [(y_k^2 - 2y_k + 1 - y_k^2)] - (y_k - 1 - y_k) + 1 \\
&= P_k + 2(x_k+1) - 2y_k + 1 + 1 + 1 \\
&= P_k + 2(x_k+1) - 2(y_k + 1) + 1 \\
\mathbf{P_{k+1}} &= \mathbf{P_k + 2X_{k+1} - 2Y_{k+1} + 1}
\end{aligned}$$

Mid Point Circle Algorithm

i.e.,
$(x_0,y_0) = (0,r)$

Here ,
$F_{circle}(1,r-1/2) = P_0$

Now,
$P_0 = 1^2+(r-1/2)^2 - r^2$
$= 1 + r^2 - r + 1/4 - r^2$
$= 5/4 - r$
$P_0 \approx 1-r$



$$\boxed{P_0 \approx 1-r}$$

Initial decision parameter

Midpoint Circle Algorithm

1. Input radius r and circle center (x_c, y_c), then set the coordinates for the first point on the circumference of a circle centered on the origin as $(x_0, y_0) = (0, r)$.

2. Calculate the initial value of the decision parameter as $p_0 = 5/4 – r$ ($1 – r$ if an integer)

3. At each x_k, from $k=0$, perform the following test:

   ○ if $p_k<0$, next point to plot along the circle centered on $(0,0)$ is $(x_k+ 1, y_k)$ and $p_{k+1} = p_k + 2 x_{k+1} + 1$

   ○ otherwise, next point to plot is $(x_k+ 1, y_k - 1)$
   and $p_{k+1} = p_k + 2 x_{k+1} + 1 - 2 y_{k+1}$

   where $2 x_{k+1} = 2 x_k + 2$, and $2 y_{k+1} = 2 y_k - 2$

Midpoint Circle Algorithm (cont.)

4. Determine symmetry points in the other seven octants.
5. Move each calculated pixel position *(x, y)* onto the circular path centered at (x_c, y_c) and plot the coordinate values:

   $x = x + x_c, y = y + y_c$
6. Repeat steps 3 through 5 until $x >= y$.

Case 1 : Pk < 0

$X_{k+1} = X_k + 1$

$Y_{k+1} = Y_k$

$$P_{k+1} = P_k + 2X_{k+1} + 1$$

Case 2: Pk ≥ 0

$X_{k+1} = X_k + 1$

$Y_{k+1} = Y_k - 1$

$$P_{k+1} = P_k + 2X_{k+1} - 2Y_{k+1} + 1$$

Solution

Here, the initial decision parameter (P0) =1 – r = 1-10 = - 9

Since, for the Midpoint Circle Algorithm of initial point(0, r)

& center at origin (0, 0) rotating at clockwise direction, we

have

Initial point (x_0,y_0) = (0,10)

$$P_0 = 1-r = 1-10 = -9$$

Thus, (0 , 10)

K	P_K	X_{k+1}	Y_{k+1}	(X_{k+1}, Y_{k+1})
0	-9	0+1 = 1	10	(1,10)
1	= -9 + 2*1 +1= -6	1+2 =2	10	(2, 10)
2	=-6 + 2*2 +1= -1	2+1=3	10	(3, 10)
3	=-1 + 2*3 +1= 6	3+1 = 4	10-1=9	(4, 9)
4	=6 + 2*4 - 2*9 +1= -3	4+1 = 5	9	(5, 9)
5	=-3 + 2*5 +1= 8	5+1 = 6	9-1=8	(6, 8)
6	=8 + 2*6 - 2*8 +1= 5	6+1 = 7	8-1=7	(7, 7)
7	=5 +2*7 – 2*7 +1 =6	7+1 = 8	7-1=6	(8,6)

Example : Digitize a circle with radius 10 at center (0,0)

Here, the initial decision parameter (P0)

$P_0 = 1 - r = 1 - 9 = -8$

Since, for the Midpoint Circle Algorithm of starting point (0, r) & center at origin (0, 0) rotating at clockwise direction, we have

If P < 0

Plot $(x_k + 1, y_k)$

$P_{k+1} = P_k + 2x_{k+1} + 1$

Else (P ≥ 0)

Plot $(x_k + 1, y_k - 1)$

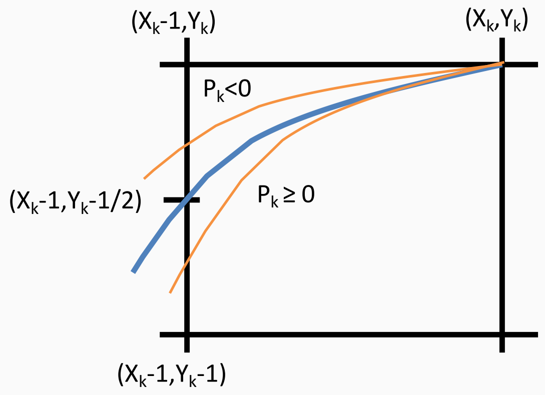
$P_{k+1} = P_k + 2x_{k+1} - 2y_{k+1} + 1$

k	P_k	X_{k+1}	Y_{k+1}	(X_{k+1}, Y_{k+1}) At (0, 0)	(X_{k+1}, Y_{k+1}) At (6, 7)
0.	-8	0+1 = 1	9	(1, 9)	(1+6, 9+7) = (7, 16)
1.	= -8 + 2*1 +1= -5	1+1 = 2	9	(2, 9)	(2+6, 9+7) = (8, 16)
2.	= -5 + 2*2 +1= 0	2+1 = 3	8	(3, 8)	(3+6, 8+7) = (9, 15)
3.	= 0 + 2*3 - 2*8 +1=-9	3+1 = 4	8	(4, 8)	(4+6, 8+7) = (10,15)
4.	= -9 + 2*4 +1= 0	4+1 = 5	7	(5, 7)	(5+6, 7+7) = (11,14)
5.	= 0 + 2*5 - 2*7 +1=-3	5+1 = 6	7	(6, 7)	(6+6, 7+7) = (12,14)
6.	= -3 + 2*6 +1= 10	6+1 = 7	6	(7, 6)	(7+6, 6+7) = (13,13)

Example: Digitize a circle with radius 9 at center (6,7)

Mid Point Circle Algorithm

Anti-Clockwise direction



$P_k < 0$
$X_{K+1} = X_k-1$
$Y_{K+1} = Y_k$

$P_k \geq 0$
$X_{K+1} = X_k-1$
$Y_{K+1} = Y_k-1$

Here,

Decision parameter(P_k) = $F_{circle}(X_k - 1, Y_k - {}^1/_2)$

$= (X_k - 1)^2 + (Y_k - {}^1/_2)^2 - r^2$

Then, K+1th term is,

$$P_{k+1} = (X_{k+1} - 1)^2 + (Y_{k+1} - {}^1/_2)^2 - r^2$$

Now,

$P_{k+1} - P_k = (X_{k+1} - 1)^2 + (Y_{k+1} - {}^1/_2)^2 - r^2 - \{(X_k - 1)^2 + (Y_k - {}^1/_2)^2 - r^2\}$

$= -2(x_k - 1) + (Y^2_{k+1} - Y^2_k) - (Y_{k+1} - Y_k) + 1$

[$\therefore X_{k+1} = X_k - 1$ in both condition]

$$\mathbf{P_{k+1} = P_k - 2x_{k+1} + (Y^2_{k+1} - Y^2_k) - (Y_{k+1} - Y_k) + 1}$$

Case 1 : $P_k < 0$

$X_{K+1} = X_k - 1$

$Y_{K+1} = Y_k$

$P_{k+1} = P_k - 2X_{k+1} + 1$

Case 2: $P_k \geq 0$

$X_{K+1} = X_k - 1$

$Y_{K+1} = Y_k - 1$

$P_{k+1} = P_k - 2X_{k+1} - 2Y_{k+1} + 1$

Initial decision parameter

i.e.,

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

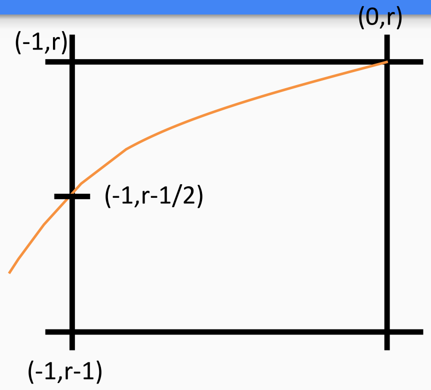
Here ,

$F_{circle}(-1, r-1/2) = P_0$

Now,

$$
\begin{aligned}
P_0 &= (-1)^2+(r-1/2)^2 - r^2 \\
&= 1 + r^2 - r + ¼ - r^2 \\
&= {}^5/_4 - r
\end{aligned}
$$

$P_0 \approx 1-r$



$$\boxed{P_0 \approx 1-r}$$

Mid point Ellipse Algorithm

Mid point Ellipse Algorithm

Our approach here is similar to that used in displaying a raster circle but the ellipse has 4-way symmetry. The midpoint ellipse method is applied throughout the first quadrant in two parts or region as shown in figure.

The region-1 just behaves as the circular property and the region-2 is slightly straight curve.

We have,

The equation of ellipse, whose centre at (0, 0) is

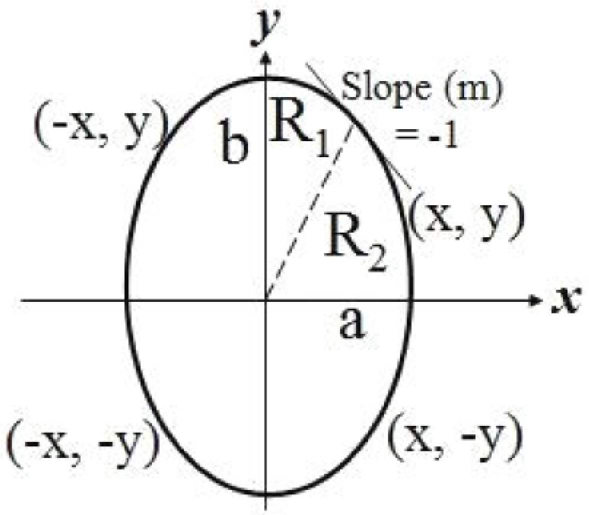
$$x^2 / a^2 + y^2 / b^2 = 1$$

Hence, we define the ellipse function for centre at origin (0, 0) as:

$$F_{\text{ellipse}}(x, y) = x^2 b^2 + y^2 a^2 - a^2 / b^2$$

This has the following properties:

$$F_{\text{ellipse}}(x, y) \begin{cases} < 0, & \text{if } (x, y) \text{ is inside the ellipse boundary} \\ = 0, & \text{if } (x, y) \text{ is on the ellipse boundary} \\ > 0, & \text{if } (x, y) \text{ is outside the ellipse boundary} \end{cases}$$



Mid point Ellipse Algorithm

Starting at (0, b), we take unit steps in the x direction until we reach the boundary between region 1 and region 2. Then we switch to unit steps in the y direction over the remainder of the curve in the first quadrant. At each step, we need to test the value of the slope of the curve. The ellipse slope is calculated by differentiating the ellipse function as:

$$2\,x\,b^2 + 2\,y\,a^2 * dy / dx = 0$$

Or

$$dy / dx = -\,2xb^2 / 2ya^2$$

At the boundary between region 1 and region 2, $dy / dx = -1$ and $2\,x\,b^2 = 2\,y\,a^2$. Therefore, we move out of region 1 whenever $2xb^2 >= 2ya^2$

For Region – 1: Condition ($2xb^2 < 2ya^2$)

Assuming that the position (x_k, y_k) has been plotted, we determine next position (x_{k+1}, y_{k+1}) as either (x_k+1, y_k) or (x_k+1, y_k-1) by evaluating the decision parameter $P1_k$ as:

$$P1_k = F_{ellipse}(x_k+1, y_k - ½)$$
$$= b^2(x_k+1)^2 + a^2(y_k - ½)^2 - a^2b^2$$

Mid point Ellipse Algorithm

At next sampling position, the decision parameter will be

$$P1_{k+1} = F_{ellipse}(x_{k+1} + 1, y_{k+1} - ½)$$
$$= b^2(x_{k+1} + 1)^2 + a^2(y_{k+1} - ½)^2 - a^2b^2$$
$$= b^2\{(x_k + 1) + 1\}^2 + a^2(y_{k+1} - ½)^2 - a^2b^2$$
$$= b^2\{(x_k + 1)^2 + 2(x_k + 1) + 1\} + a^2(y_{k+1} - ½)^2 - a^2b^2$$

Now, $P1_{k+1} - P1_k = \left(b^2\{(x_k + 1)^2 + 2(x_k + 1) + 1\} + a^2(y_{k+1} - ½)^2 - a^2b^2\right) - \left(b^2(x_k + 1)^2 + a^2(y_k - ½)^2 - a^2b^2\right)$

$$= \left(b^2(x_k + 1)^2 + 2b^2(x_k + 1) + b^2 + a^2(y_{k+1})^2 - a^2y_{k+1} + a^2 ¼ - a^2b^2\right) - \left(b^2(x_k + 1)^2 + a^2y_k^2 - a^2y_k + a^2 ¼ - a^2b^2\right)$$

$$= 2b^2(x_k + 1) + b^2 + a^2\{((y_{k+1})^2 - y_k^2) - (y_{k+1} - y_k)\}$$

If $P1_k < 0$, the midpoint is inside the ellipse and the pixel on the scan line y_k closer to the ellipse boundary. Otherwise, the midpoint is outside or on the ellipse boundary, and we select the pixel on scan line $y_k - 1$.

Case: $P1_k < 0$

$$x_{k+1} = x_k + 1$$
$$y_{k+1} = y_k$$

Mid point Ellipse Algorithm

$P1_{k+1} = P1_k + 2b^2(x_{k+1}) + b^2$

Case I: $P1_k >= 0$

$x_{k+1} = x_k + 1$

$y_{k+1} = y_k - 1$

$P1_{k+1} = P1_k + 2b^2(x_k + 1) + b^2 + a^2\{((y_{k+1})^2 - y_k^2) - (y_{k+1} - y_k)\}$

$= P1_k + 2b^2(x_k + 1) + b^2 + a^2((y_k - 1)^2 - y_k^2) - a^2(y_k - 1 - y_k)\}$

$= P1_k + 2b^2(x_k + 1) + a^2[(y_k^2 - 2y_k + 1 - y_k^2)] - a^2(y_k - 1 - y_k) + b^2$

$= P1_k + 2b^2(x_k + 1) - 2a^2 y_k + a^2 + a^2 + b^2$

$= P1_k + 2b^2(x_k + 1) - 2a^2(y_k - 1) + b^2$

$= P1_k + 2b^2 x_{k+1} - 2a^2 y_{k+1} + b^2$

Initial decision parameter ($P1_0$) for region-1 of ellipse

The initial decision parameter is obtained by evaluating the ellipse function at the start position $(x_0, y_0) = (0, b)$.

Here, the next pixel will either be (1, b) or (1, b-1) where the midpoint is (1, b- ½). Thus, the initial decision parameter is given by:

$P1_0 = F_{ellipse}(1, b - ½)$

$= b^2 + a^2(b - ½)^2 - a^2 b^2$

$= b^2 - a^2 b + a^2/4$

Mid point Ellipse Algorithm

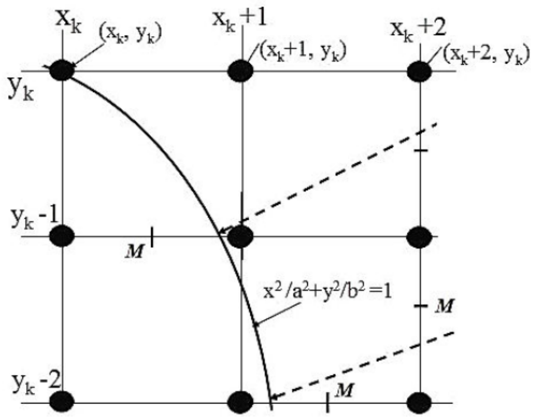
For Region – 2: Condition ($2xb^2 >= 2ya^2$)

Assuming that the position (x_k ,y_k) has been plotted, we determine next position (x_{k+1}, y_{k+1}) as either (x_k+1 , y_k-1) or (x_k, y_k-1) by evaluating the decision parameter $P2_k$ as:

$$P2_k = F_{ellipse}(x_k + ½ , y_k -1)$$
$$= b^2(x_k + ½)^2 + a^2(y_k -1)^2 – a^2b^2$$

At next sampling position, the decision parameter will be

$$P2_{k+1} = F_{ellipse}(x_{k+1} + ½ , y_{k+1} -1)$$
$$= b^2(x_{k+1} + ½)^2 + a^2(y_{k+1} -1)^2 – a^2b^2$$



Now, subtracting $P2_k$ from $P2_{K+1}$, we have

$$P2_{k+1} – P2_k = \{b^2(x_{k+1} + ½)^2 + a^2(y_{k+1}-1)^2 - a^2b^2\} - \{b^2(x_k + ½)^2 + a^2(y_{k-1})^2 – a^2b^2\}$$

$$= \{b^2(x_{k+1} + ½)^2 + a^2(y_{k+1}-1)^2 - a^2b^2 - b^2(x_k + ½)^2 - a^2(y_{k-1})^2 + a^2b^2\}$$

$$= \{b^2(x_{k+1} + ½)^2 + a^2(y_{k+1}-1)^2 - b^2(x_k + ½)^2 - a^2(y_{k-1})^2\}$$

$$= b^2(x_{k+1} + ½)^2 + a^2 \{(y_k-1)-1\}^2 - b^2(x_k + ½)^2 - a^2(y_{k-1})^2$$

$$= b^2x_{k+1}^2+ b^2x_{k+1} + b^2/4 + a^2(y_k -1)^2 - 2a^2(y_k -1) +a^2 - b^2x_k^2 – b^2x_k - b^2/4 -a^2(y_{k-1})^2$$

$$= a^2 + b^2(x_{k+1}^2 -x_k^2) - 2a^2(y_k -1) + b^2(x_{k+1} -x_k)$$

Mid point Ellipse Algorithm

If $P2_k<0$, the midpoint is inside the ellipse and the pixel on the scan line x_k is closer to the ellipse boundary.

Otherwise, the midpoint is outside or on the ellipse boundary, and we select the pixel on scan line x_k+1.

Case : $P2_k \le 0$

$$x_{k+1} = x_k + 1$$

$$y_{k+1} = y_k - 1$$

$$P2_{k+1} = P2_k + b^2 \left[(x_{k+1}^2 - x_k^2) + (x_{k+1} - x_k)\right] - 2a^2(y_k - 1) + a^2$$

$$= P2_k + b^2 \left[(x_k^2 + 2x_k + 1 - x_k^2)\right] + (x_k + 1 - x_k)] - 2a^2(y\ k - 1) + a^2$$

$$= P2_k + b^2 [2x_k + 1 + 1] - 2a^2 (y_k - 1) + a^2$$

$$= P2_k + 2b^2(x_k + 1) - 2a^2 (y_k - 1) + a^2$$

$$= P2_k + 2b^2 x_{k+1} - 2a^2 y_{k+1} + a^2$$

Case $P2_k > 0$

$$x_{k+1} = x_k$$

$$y_{k+1} = y_k - 1$$

$$P2_{k+1} = P2_k + b^2 \left[(x_{k+1}^2 - x_k^2) + (x_{k+1} - x_k)\right] - 2a^2(y_k - 1) + a^2$$

$$= P2_k - 2a^2 (y_k - 1) + a^2$$

$$= P2_k - 2a^2 y_{k+1} + a^2$$

Mid point Ellipse Algorithm

Initial decision parameter ($P2_0$) for region-2 of ellipse

When we enter region 2, the initial position (x_0, y_0) is taken as the last position selected in region 1 and the initial derision parameter in region 2 is given by:

$$P2_0 = F_{ellipse}(x_0 + 1/2,\ y_0 - 1)$$

$$= b^2 (x_0 + 1/2)^2 + a^2 (y_0 - 1)^2 - a^2 b^2$$

Mid point Ellipse Algorithm

Example: Given input ellipse parameters $r_x = a = 8$ and $r_y = b = 6$, we illustrate the steps in the midpoint ellipse algorithm by determining raster positions along the ellipse path in the first quadrant

Given a = 8 and b= 6

For Region 1

The initial point for the ellipse on the origin $(x_o, y_o) = (0, b) = (0,6)$

Calculation the initial decision parameter

$$P1_0 = b^2 – a^2 b + a^2/4$$

$$= 6^2 - 6 * (8)^2+(8^2 / 4)$$

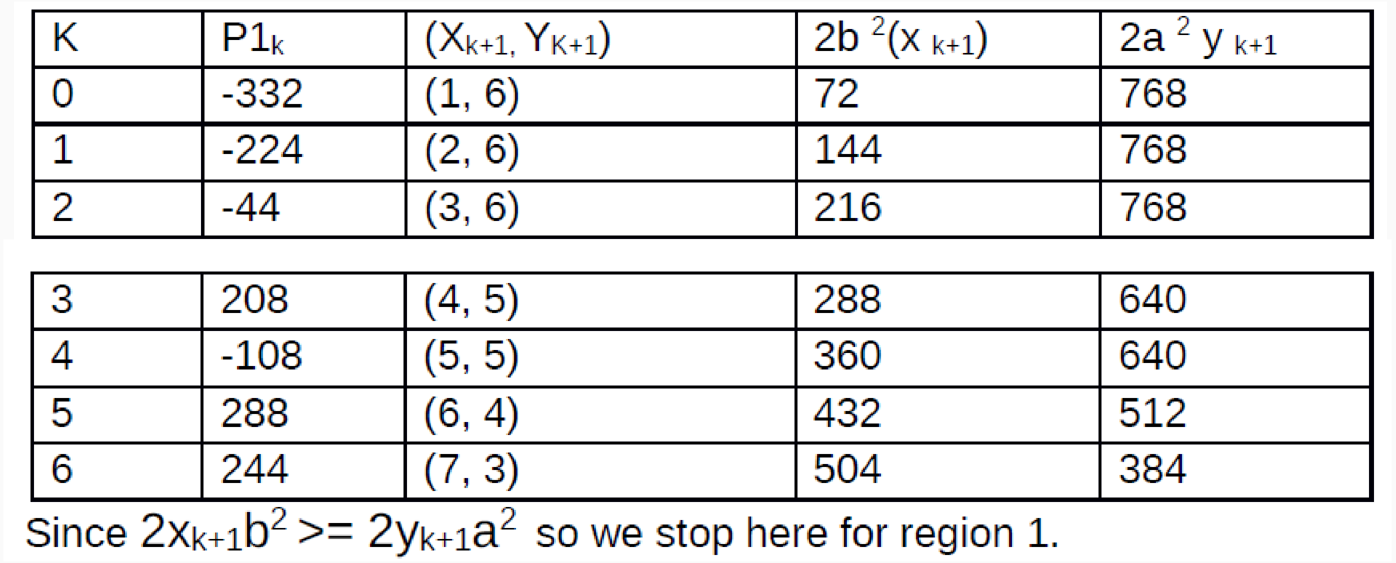
$$= -332$$

From midpoint algorithm, for Region 1 we know,

```
If($P1_k<0$) then
    $x_{k+1} = x_k +1$
    $y_{k+1} = y_k$
    $P1_{k+1}= P1_k + 2b^2x_{k+1}+ b^2$
Else (i.e. $P1_k >=0$)
    $x_{k+1} = x_k +1$
    $y_{k+1} = y_k - 1$
    $P1_{k+1}= P1_k +2b^2 x_{k+1} – 2a^2 y_{k+1} + b^2$
    And stop when $2x_{k+1}b^2 >= 2y_{k+1}a^2$
```

Mid point Ellipse Algorithm

Now successive decision parameter values and positions along the ellipse path are calculated using midpoint method as



K	$P1_k$	(X_{k+1}, Y_{K+1})	$2b^2(x_{k+1})$	$2a^2 y_{k+1}$
0	-332	(1, 6)	72	768
1	-224	(2, 6)	144	768
2	-44	(3, 6)	216	768
3	208	(4, 5)	288	640
4	-108	(5, 5)	360	640
5	288	(6, 4)	432	512
6	244	(7, 3)	504	384

Since $2x_{k+1}b^2 >= 2y_{k+1}a^2$ so we stop here for region 1.

Note: (7, 3) is the starting point for region 2]

Mid point Ellipse Algorithm

Now for Region-2,

From midpoint algorithm, for Region 2 we know,

Initial Decision Parameter:

$$P2_0 = b^2 (x_0 + 1/2)^2 + a^2 (y_0 - 1)^2 - a^2 b^2$$

```
If($P_k<0$)  then
    $X_{k+1} = X_k + 1$
    $Y_{k+1} = Y_k - 1$
    $P2_{k+1} = P2_k + 2b^2 x_{k+1} - 2a^2 y_{k+1} + a^2$
Else (i.e. $P_k >=0$)
    $X_{k+1} = X_k$
    $Y_{k+1} = Y_k - 1$
    $P2_{K+1} = P2_k - 2a^2 y_{k+1} + a^2$
```

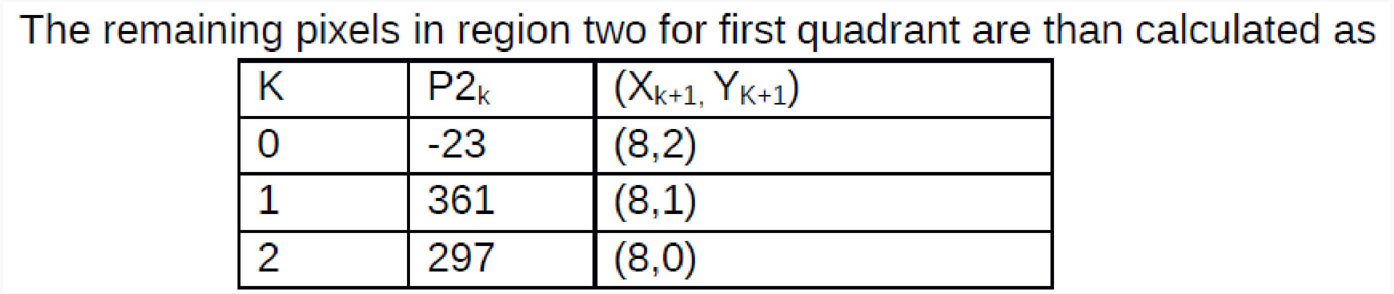
So, the initial point for region 2 is $(x_0, y_0) = (7, 3)$ and the initial decision parameter is

$$P2_0 = b^2 (x_0 + 1/2)^2 + a^2 (y_0 - 1)^2 - a^2 b^2$$

$$= 36 * (7 + 0.5)^2 + 64 * (3-1)^2 - 64 * 36$$

$$= -23$$

Mid point Ellipse Algorithm



The remaining pixels in region two for first quadrant are than calculated as

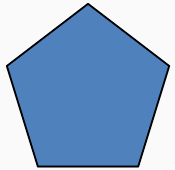
K	$P2_k$	(X_{k+1}, Y_{K+1})
0	-23	(8,2)
1	361	(8,1)
2	297	(8,0)

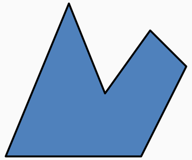
So, remaining points in other quadrants can be calculated using the symmetry property of ellipse.

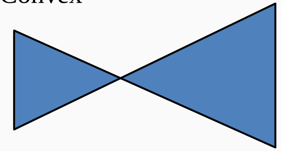
Fill Area Algorithms

Polygon Fill Algorithm

- ***Different types of Polygons***
  - Simple Convex
  - Simple Concave
  - Non-simple : self-intersecting
  - With holes


Convex


Concave


Self-intersecting

Area Fill Algorithm

- An area fill algorithm is a computer graphics algorithm used to fill a closed region or area with a specified color or pattern.
- This is often employed in the rendering of images and graphics on a computer screen.
- There are several algorithms for area filling, and the choice of algorithm depends on factors such as efficiency, simplicity, and the specific requirements of the application.
- The ***algorithm makes the following assumptions***
  - one interior pixel is known, and
  - pixels in boundary are known.

Area Fill Algorithm: Scan Line Fill Algorithm

Description: This algorithm works by scanning the image or polygon line by line, determining the intersections of the scan lines with the edges of the polygon, and filling the spaces between these intersections.

Advantages: It is relatively simple and easy to implement.

Disadvantages: It may not be efficient for complex polygons.

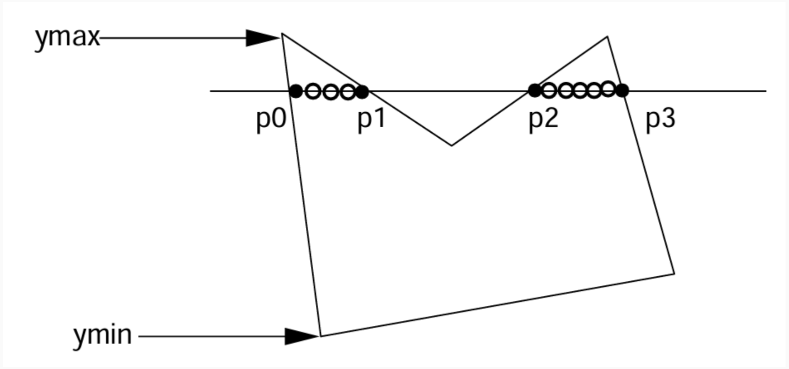
<u>Algorithm:</u>

```
For y = ymin to ymax

1)    intersect scanline y with each edge

2)    sort intersections' by increasing x  [p0,p1,p2,p3]

3)     fill pairwise  (p0 –> p1, p2–> p3, ....)
```



Area Fill Algorithm: Boundary Fill Algorithm

Description: This algorithm starts at a seed point inside a polygon and recursively fills neighboring pixels until a boundary color is encountered. It checks whether a pixel needs to be filled by comparing its color with the boundary color and the fill color.

Boundary Fill Algorithm is recursive in nature.

- ✓ It takes an interior point(x, y), a fill color, and a boundary color as the input.
- ✓ The algorithm starts by checking the color of (x, y). If it's color is not equal to the fill color and the boundary color, then it is painted with the fill color and the function is called for all the neighbours of (x, y).
- ✓ If a point is found to be of fill color or of boundary color, the function does not call its neighbours and returns. This process continues until all points up to the boundary color for the region have been tested.

Advantages: It is easy to implement.

Disadvantages: Recursive implementations may lead to stack overflow for large polygons. Also, handling certain cases (like concave polygons) can be challenging.

Area Fill Algorithm: Boundary Fill Algorithm

- The boundary fill algorithm can be implemented by 4-connected pixels or 8-connected pixels.

1. **Four neighboring points or 4-connected pixels**
    - These are the pixel positions that are right, left, above, and below the current pixel.
    - Areas filled by this method are called **4-connected.**

2. Eight neighboring points or 8-connected pixels

- This method is used to fill more complex figures.
- Here the set of neighboring points to be set includes the four diagonal pixels, in addition to the four points in the first method.
- Fill methods using this approach are called **8-connected.**

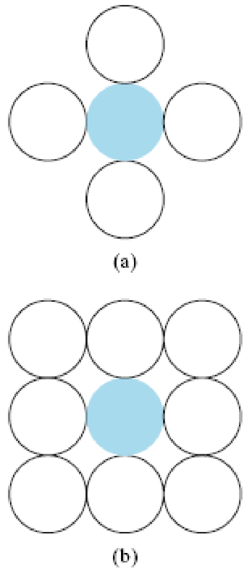


FIGURE 4-27 Fill methods applied to a 4-connected area (a) and to an 8-connected area (b). Hollow circles represent pixels to be tested from the current test position, shown as a solid color.

Area Fill Algorithm: Boundary Fill Algorithm

Four neighboring points or 4-connected pixels

- These are the pixel positions that are right, left, above, and below the current pixel.
- Areas filled by this method are called **4-connected.**

```
void boundaryFill4(int x, int y, int fill_color,int boundary_color)
{
    if(getpixel(x, y) != boundary_color &&
      getpixel(x, y) != fill_color)
    {
       putpixel(x, y, fill_color);
       boundaryFill4(x + 1, y, fill_color, boundary_color);
       boundaryFill4(x, y + 1, fill_color, boundary_color);
       boundaryFill4(x - 1, y, fill_color, boundary_color);
       boundaryFill4(x, y - 1, fill_color, boundary_color);
    }
}
```

Area Fill Algorithm: Boundary Fill Algorithm

Eight neighboring points or 8-connected pixels

- This method is used to fill more complex figures.
- Here the set of neighboring points to be set includes the four diagonal pixels, in addition to the four points in the first method.
- Fill methods using this approach are called **8-connected.**

```
void boundaryFill8(int x, int y, int fill_color,int boundary_color)
{
    if(getpixel(x, y) != boundary_color &&
       getpixel(x, y) != fill_color)
    {
        putpixel(x, y, fill_color);
        boundaryFill8(x + 1, y, fill_color, boundary_color);
        boundaryFill8(x, y + 1, fill_color, boundary_color);
        boundaryFill8(x - 1, y, fill_color, boundary_color);
        boundaryFill8(x, y - 1, fill_color, boundary_color);
        boundaryFill8(x - 1, y - 1, fill_color, boundary_color);
        boundaryFill8(x - 1, y + 1, fill_color, boundary_color);
        boundaryFill8(x + 1, y - 1, fill_color, boundary_color);
        boundaryFill8(x + 1, y + 1, fill_color, boundary_color);
    }
}
```

Area Fill Algorithm: Flood Fill Algorithm

Description: Similar to the boundary fill algorithm, flood fill starts at a seed point and fills connected pixels until a boundary is encountered. It uses a stack or a queue to keep track of pixels to be filled.

- The flood fill algorithm has many characters similar to boundary fill. But this method is more suitable for filling multiple colors boundary. When boundary is of many colors and interior is to be filled with one color we use this algorithm.

Advantages: It is often more efficient than the boundary fill algorithm, and it is less prone to stack overflow.

Disadvantages: It may not handle certain cases well, such as gradients or patterns.

Area Fill Algorithm: Flood Fill Algorithm

```
void floodfill (x, y,fill_ color, old_color: integer)   {
    if (getpixel (x, y)=old_color)
   {
    setpixel (x, y, fill_color);
    fill (x+1, y, fill_color, old_color);
     fill (x-1, y, fill_color, old_color);
    fill (x, y+1, fill_color, old_color);
    fill (x, y-1, fill_color, old_color);
     }
}
```

Area fill algorithm: Example

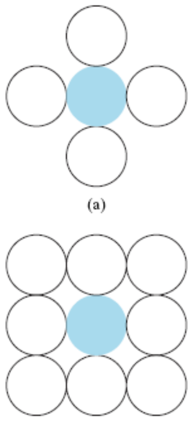


FIGURE 4-27 Fill methods applied to a 4-connected area (a) and to an 8-connected area (b). Hollow circles represent pixels to be tested from the current test position, shown as a solid color.

- Consider the Figure in the below.
- An 8-connected boundary-fill algorithm would correctly fill the interior of the area defined in the Figure.
- But a 4-connected boundary-fill algorithm would only fill part of that region.

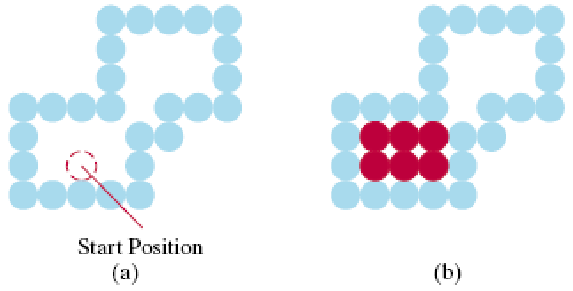


FIGURE 4-28 The area defined within the color boundary (a) is only partially filled in (b) using a 4-connected boundary-fill algorithm.

Area fill algorithm: Example

- Some times we want to fill in (or recolor) an area that is not defined within a single color boundary.
- Consider the following Figure.

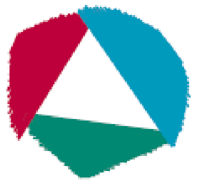
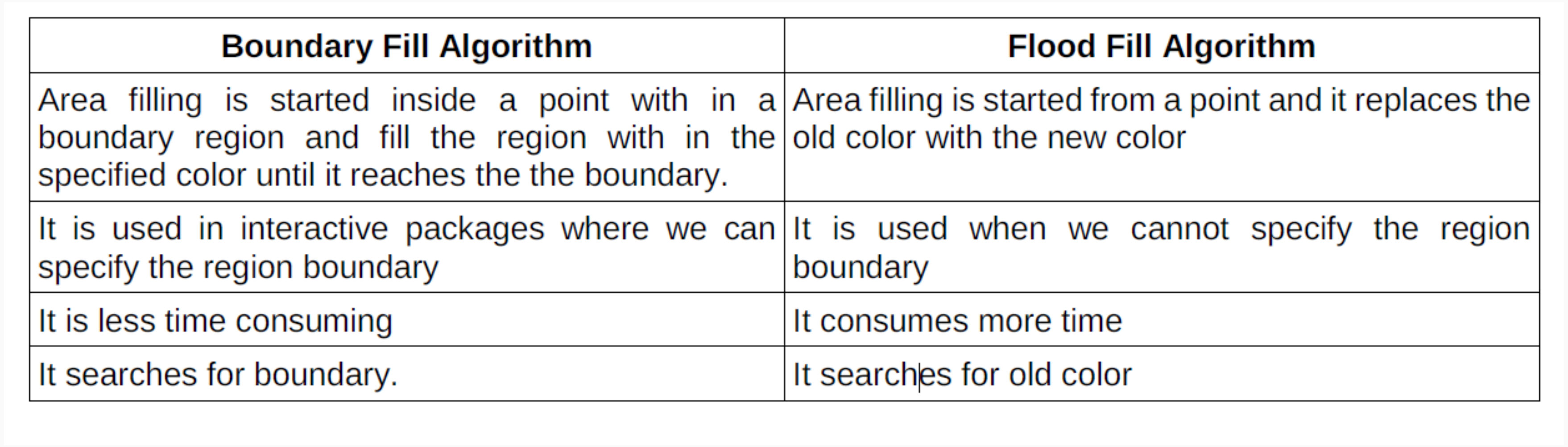


FIGURE 4-30 An area defined within multiple color boundaries.

- We can paint such areas by replacing a specified interior color instead of searching for a particular boundary color.
- This fill procedure is called a **flood-fill algorithm.**
- We start from a specified interior point (*x*, *y*) and reassign all pixel values that are currently set to a given interior color with the desired fill color.
- If the area we want to paint has more than one interior color, we can first reassign pixel values so that all interior points have the same color.
- Using either a 4-connected or 8-connected approach, we then step through pixel positions until all interior points have been repainted.
- The following procedure flood fills a 4-connected region recursively, starting from the input position.

Boundary Fill Vs Flood Fill Algorithm



Boundary Fill Algorithm	Flood Fill Algorithm
Area filling is started inside a point with in a boundary region and fill the region with in the specified color until it reaches the the boundary.	Area filling is started from a point and it replaces the old color with the new color
It is used in interactive packages where we can specify the region boundary	It is used when we cannot specify the region boundary
It is less time consuming	It consumes more time
It searches for boundary.	It searches for old color

Clipping

- The process of discarding those parts of a picture which are outside of a specified region or window is called clipping.
- The procedure using which we can identify whether the portions of the graphics object is within or outside a specified region or space is called clipping algorithm.

Applications:

i. Extracting part of a defined scene for viewing.

ii. Identifying visible surfaces in three-dimensional views.

iii. Anti aliasing line segments or object boundaries.

iv. Drawing and painting operations that allow parts of a picture to be selected for copying, moving erasing, or duplicating.

Also refer to class notes too.

THANK YOU

Any Queries ?