COMPUTER GRAPHICS

Disclaimer

This document is part of teaching materials for COMPUTER GRAPHICS under the Pokhara University syllabus for Bachelors in computer and IT Engineering (BE-CMP and BE-IT). This document does not cover all aspect of learning COMPUTER GRAPHICS, nor are these be taken as primary source of information. As the core textbooks and reference books for learning the subject has already been specified and provided to the students, students are encouraged to learn from the original sources because this document cannot be used as a substitute for prescribed textbooks.

Various text books as well as freely available material from internet were consulted for preparing this document. Contents in This document are ***copyrighted*** *to the instructor and authors of original texts where applicable.*

Unit-2 Scan Conversion Algorithms

2.1 Scan Conversion

- It is a process of representing graphics objects as a collection of pixels. The graphics objects are continuous. The pixels used are discrete. Each pixel can have either on or off state.
- The circuitry of the video display device of the computer is capable of converting binary values (0, 1) into a pixel on and pixel off information. O is represented by pixel off. 1 is represented using pixel on. Using this ability graphics computer represent picture having discrete dots.
- Any model of graphics can be reproduced with a dense matrix of dots or points. Most human beings think graphics objects as points, lines, circles, ellipses. For generating graphical object, many algorithms have been developed.

Advantage of developing algorithms for scan conversion

- Algorithms can generate graphics objects at a faster rate.
- 2. Using algorithms memory can be used efficiently.
- 3. Algorithms can develop a higher level of graphical objects.

Examples of objects which can be scan converted

- Point
- Line
- Sector
- Arc
- Ellipse
- Rectangle
- Polygon
- Characters
- Filled Regions

- The process of converting is also called as rasterization.
- The algorithms implementation varies from one computer system to another computer system.
- Some algorithms are implemented using the software. Some are performed using hardware or firmware.

- Some are performed using various combinations of hardware, firmware, and software.

Output Primitives:

Initial phase of computer science→ computing and textual form printing

Later on, → graphics evolved→ drawing of different objects like line, circle, ellipse, rectangle, spline curve etc.

Hence, these graphical components are called output primitives.

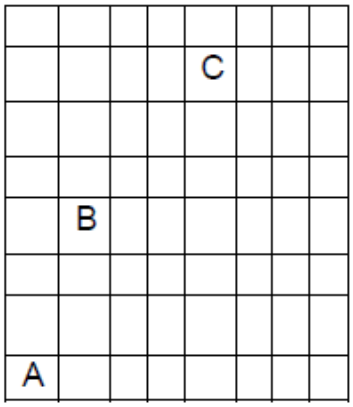
Output primitives are the geometric structure such as straight-line segments and polygon areas used to describe the shapes and color of the objects.

In case of the two-dimensional algorithm, we mostly deal with the line, circle, ellipse etc. as they are the basic output primitives.

Scan Converting a Point and a Straight Line

Point:

- Pixel is a unit square area identified by the coordinate of its lower left corner.
- Each pixel on the display surface has a finite size depending on the screen resolution & hence a pixel can't represent a single mathematical number.
- Origin of the reference coordinate system being located of the lower left corner of the display surface.
- Each pixel is accused by non-negative integer coordinate pair (x, y).
- The x values start at the origin &increase from left to right along a scan line & y values start at the bottom & increase upwards.



- In the above diagram the coordinate of pixel A:0,0, B:1,4, C:4,7.C:4,7.
- A coding position (4. 2, 7. 2) is represented by C whereas (1.5, 4.2) is represented by B.
- In order to half a pixel on the screen we need to round off the coordinate to a nearest integer.

Line:

- Line drawing is accomplished by calculating intermediate point coordinates along the line path between two given end points.
- Screen pixel is referred with integer values, plotted positions may only approximate the calculate coordinates, what is pixel which are intensified are those which lie very close to the line path.
- In a high-resolution system, the adjacent pixels are so closely spread that the approximated line pixels lie very close to actual line path and hence the plotted lines appear to be much smooth-almost like straight line drown on paper.
- In low resolution system the same approximation technique causes to display with stair step appearance that is not smooth.

2.2. Line Drawing Algorithm

The equation of a straight line is Y=mX + b

Where **m** representing slope of the line and **b** as the y intercept.

Consider the two end points of a line segment are (x_1, y_1) & (x_2, y_2) then the straight line can be written as

$$y_1 = \frac{y_2 - y_1}{x_2 - x_1} x_1 + b \quad \ldots\ldots\ldots\ldots\ldots\ldots (1)$$

We can determine the volume for the slope 'm' & y intercept 'b' with the following calculation.

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

$$b = y_1 - mx_1$$

$$\Rightarrow y_1 = mx_1 + b$$

For any given x interval Δx along a line, we can compute the corresponding y interval Δy from the above equation

$$m = \frac{\Delta y}{\Delta x} \Rightarrow \Delta y = m\Delta x$$

Similarly, we can obtain the x interval Δx corresponding to a specified Δy as

$$\Delta x = \frac{\Delta y}{m}$$

For a line with slope magnitude |m|<1,

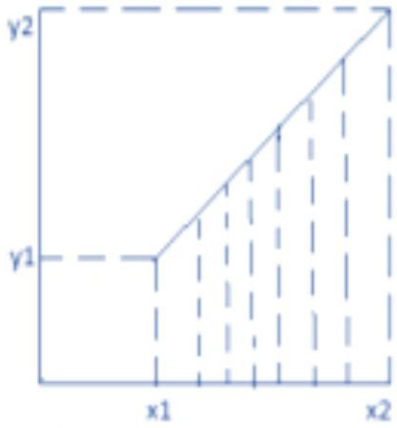
Δx can be set proportional to a small horizontal deflection voltage & the corresponding vertical deflection is then set proportional to Δy as calculate from the equation

$\Delta y = m\, \Delta x$

For a line whose slopes have magnitudes |m|>1,

Δy can be set proportional to a small vertical deflection voltage with the corresponding horizontal deflection voltage set proportional to Δx calculate from the equation

$$\Delta x = \frac{\Delta y}{m}$$



For a line with m=1,

Then $\Delta x = \Delta y$ and vertical & horizontal deflection voltages are equal.

In each case a smooth line with slope 'm' is generated between specified end point.

2.2.1. Digital Differential Analyzer (DDA)

- DDA is a scan-conversion line algorithm base on either Δx or Δy.

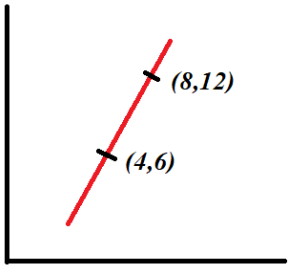
⬇

The process of representing continuous graphics objects as a collection of discrete pixels is called ***scan conversion***. The graphics objects are continuous. The pixels used are discrete.

- 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:

Consider the line having positive slope



The equation of the line is given,

$y = mx+b$ …….(i)

$m = (y2-y1)/(x2-x1)$ …….(ii)

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

Case I: If $|m| \leq 1$, we sample at unit x interval

i.e $\triangle x = 1$.

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

Then we compute each successive y-values, by setting $\triangle y = m$

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

The calculated y value must be rounded to the nearest integer

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}).

Advantages

- DDA algorithm is a faster method for calculating pixel position then the equation of a pixel position. Y = mx + b
- Simplex and fast method

Disadvantages

- Accumulation of round of error is successive addition of the floating-point increments is used to find the pixel position but it takes lot of time to compute the pixel position.
- Poor end point accuracy

Problem: 01: Digitized the line with end points (0, 0) and (4, 5) using DDA.

Solution:

Given,

Starting Point: P $(x_1, y_1) = (0, 0)$

Ending Point: Q $(x_2, y_2) = (4, 5)$

Now Slope m= $\frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1}$ = (5-0) / (5-0) = (5/4) = 1.25 i.e. Slope |m|>1,

From DDA algorithm we have

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

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

Hence,

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)

Problem: 02: Consider a line from M (2, 1) to P (8, 3). Using DDA algorithm, rasterize this line.

Solution:

Given,

Starting Point: M (x_1, y_1) = (2, 1)
Ending Point: P (x_2, y_2) = (8, 3)

Now,

Slope m= = $\frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1}$ = (3-1) / (8-2) = (1/3) = 0.333 i.e. slope |m|<1

From DDA algorithm we have

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

Hence,

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)

Practice:

Q1. Consider a line from A (0, 0) to B (5, 5). Using DDA algorithm, rasterize this line.

2.2.2. Bresenham's Line Drawing Algorithm

We have to different condition for Bresenhams line drawing algorithm for positive slope.

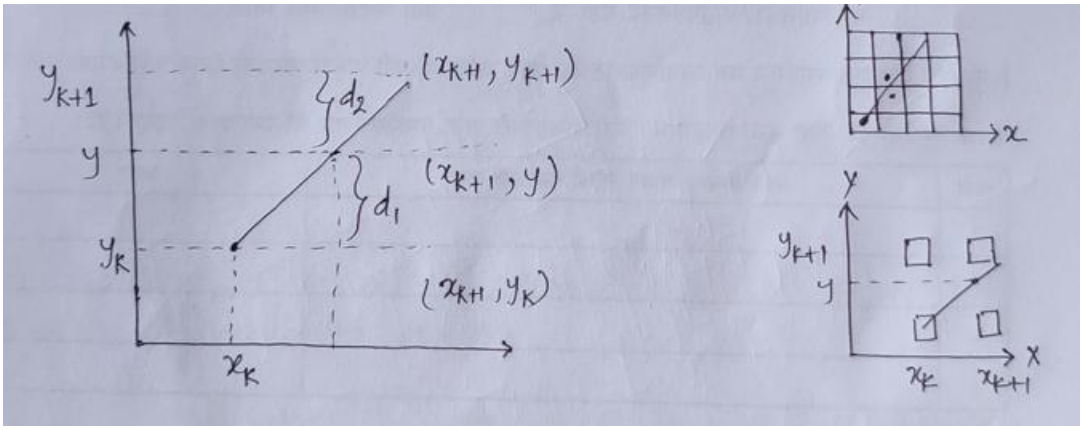
1. **For +Ve Slope less than 1 (i.e. m<1)**

Objective:

To decide which of two possible pixel position is closer to the line path ar each sample step.

If we consider the +ve slope < 1 then pixel position along line path are determined by sampling at X- interval

Let us start from left end point (x_0, Y_0) of the given line we step to each successive column (X position) and plot the pixel whose scan line y-value is closest to the line path.



Let, (x_k, y_k) be the pixel position of starting point of line the next pixel to be plotted is either (x_{k+1}, y_k) or (x_{k+1}, y_{k+1}).

At sampling position x_{k+1} we label vertical pixel separation from mathematical line path as d_1 and d_2.

The y coordinate on the mathematical line at pixel column position x_{k+1} is calculated as:

$y= mx+b \Rightarrow m(x_k+1)+b$ ---------- (1)

From above figure,

$$d_1 = y- y_k \text{ and } d_2 = (y_{k+1}) - y$$

difference between these two separations is $d_1 - d_2$ then

$$d_1 - d_2 = (y- y_k) - [(y_k +1)-y] \Rightarrow y- y_k -y_k -1 +y \Rightarrow 2y -2y_k -1$$

from equation (1)

$d_1 - d_2 = 2[m(x_k +1) + b] - 2y_k -1$

$= 2m(x_{k+1})+2b - 2y_{k-1}$ .....(2)

If $d_1 - d_2 < 0$ i.e. Distance d_1 is small (d1 <d2) so pixel x_{k+1}, y_k is plotted

If $d_1 - d_2 > 0$ i.e. Distance d_2 is small (d1 >d2) so pixel x_{k+1}, y_{k+1} is plotted

In Equation (2), Slope (m) is present and 'm' gives float value so we need to remove it.

Put m = dy / dx and multiply both side by dx, we get

$$dx(d_1 - d_2) = 2 \cdot dx \cdot dy/dx\,(x_k + 1) + 2b\,dx - 2y_k\,dx - dx$$

$$\Rightarrow 2\,dy.\,x_k + 2dy + 2b \cdot dx - 2y_k\,dx - dx$$

from this expression we can decide the nearest distance the pixel. Hence consider the decision parameter as P^k for the k^{th} step / position.

$$\text{Therefore, } P_k = dx(d1 - d2) = 2\,dy.\,x_k + 2\,dy + 2b.\,dx - 2y_k\,dx - dx$$

$$\text{Hence, } P_k = 2dy.\,X_k - 2\,dx \cdot y_k + c$$

where, c = 2dy+2b.dx-dx is constant in above expression which doesn't affect the decision.

For the k^{th} position P_k is decision parameter and it changes towards every position.

Now,

To decide the next nearest pixel to be plotted, we have to find next P_k i.e. P_{k+1}

$$P_{k+1} = 2dy.\,x_{k+1} - 2dx.\,y_{k+1} \text{ or, } P_{next} = 2dy.x_{next} - 2dx.\,y_{next}$$

To find how decision Parameter changes,

$$P_{k+1} - P_k = 2dy.\,x_{k+1} - 2dx.\,y_{k+1} - [2dy.\,x_k - 2\,dx \cdot y_k]$$

$$= 2dy.\,x_{k+1} - 2dx.\,y_{k+1} - 2dy.\,x_k + 2\,dx \cdot y_k$$

Case-I: If $P_{k+1} - P_k < 0$, i.e. $P_k < 0$ => point to be plotted (x_{k+1}, y_k)

case-II: If $P_{k+1} - P_k >= 0$, i.e. $P_k >= 0$ => point to be plotted (x_{k+1}, y_{k+1})

From Case I:

$$P_{k+1} - P_k = 2dy.\,x_{k+1} - 2dx.\,y_{k+1} - 2dy.\,x_k + 2\,dx \cdot y_k$$

$$= (2\,dy\,(x_k + 1)) - 2dx.y_k - 2dy \cdot x_k + 2dx.\,y_k$$

$$= 2dy\,x_k + 2dy - 2dx.y_k - 2dy \cdot x_k + 2dx.\,y_k$$

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

Similarly, from Case II:

$P_{k+1}-P_k = [2dy. x_{k+1} – 2dx. y_{k+1}] – [2dy. x_k - 2 dx . y_k]$

On Solving,

$P_{k+1}-P_k = 2dy- 2dx$

$P_{k+1} = P_k + 2dy-2dx$

Now,

Calculate the decision parameter for initial condition

We know, for k^{th} position

$P_k = 2 dy. x_k -2 dx y_k +2 dy + 2b. dx – dx$

Let, P_0 is the initial decision parameter,

$$P_0 = 2 dy\, x_0 – 2 dx\, y_0 + 2 dy-2 dx\, b-dx$$

$$\Rightarrow 2 dy\, x_0 – 2 dx\, y_0 + 2 dy-2 dx\, [y_0 – (dy/dx).x_0]-dx$$

As we know, $y=mx+b$

$y_o=mx_0+b$

$b= Y_o-m.x_o$

$b= Y_o- (dy/dx)*x_0$

- ⇨ $2 dy\, x_0 – 2 dx\, y_0 + 2dy – 2 dx\, y_0 – 2dy\, x_0 - dx$
- ⇨ **$P_0 = 2dy -dx$** Which is required initial decision parameter

Algorithm (for m<1)

Step 1 : Input the two line end points & store left end point in (x_0 ,y_0)

Step 2 : load (x_0 ,y_0) into the frame buffer and plot the first point.

Step 3 : Calculate constants Δx , Δy, $2\Delta y$ and $2\Delta y - 2\Delta x$ and obtain the starting value for the decision parameter as **$P_0 = 2\Delta y - \Delta x$**

Step 4 : At each x_k along the line starting at k=0, perform the following.

If pk <0

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

$Y_{k+1} = y_k$

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

Then, the next point to be plotted is $(x_k + 1, y_k)$

Else (i.e. **pk >=0**

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

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

$P_{k+1} = p_k + 2\,\Delta y - 2\,\Delta x$

Then, the next point to be plotted is $(x_k + 1, y_k + 1)$

Step 5 : End

2. Bresenham's Line Drawing Algorithm for slope (m) >1

Algorithm

Step 1 : Input the two line end points &store left end point in (x_0, y_0)

Step 2 : load (x_0, y_0) into the frame buffer plot the first point.

Step 3 : Calculate constants Δx , Δy, $2\Delta y$ and $2\Delta x - \Delta y$ and obtain the starting value for the decision parameter as $\mathbf{P_0 = 2\Delta x - \Delta y}$

Step 4 : At each x_k along the line starting at k=0, perform the following last.

If pk <0

$X_{k+1} = x_k$

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

$P_{k+1} = p_k + 2\Delta x$

Then, the next point to be plotted is $(x_k, y_k + 1)$

Else (i.e. **pk >=0**

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

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

$P_{k+1} = p_k + 2\Delta x - 2\Delta y$

Then, the next point to be plotted is $(x_k + 1, y_k + 1)$

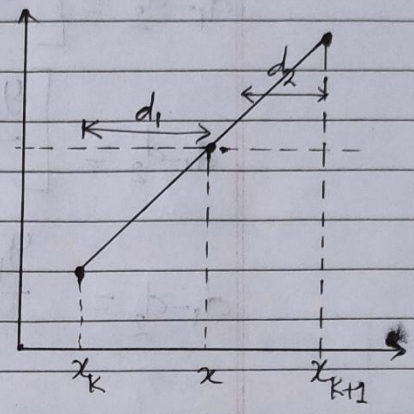
Step 5 : End

Derivation:

Bresenham's Line drawing Algorithm

for $|m| > 1$

⇒ Let (x_k, y_k) be the pixel position determined then the next to the be plotted is either (x_{k+1}, y_{k+1}) or (x_k, y_{k+1})



Let, d_1 and d_2 be the seperation of pixel positions (x_k, y_{k+1}) and (x_{k+1}, y_{k+1}) for the actual line path.

We know $y = mx + b$ then

$$x = \frac{y-b}{m} \quad \text{—— (1)}$$

Now, Sampling position at y_{k+1}

$$x = \frac{(y_k + 1 - b)}{m} \quad \text{—— (11)}$$

from above figure,

$$d_1 = x - x_k$$
$$d_2 = x_k + 1 - x$$

Now,

To find closes pixel,

$$d_1 - d_2 = (x - x_k) - (x_k + 1) - x$$

put x from (ii) we get,

$$d_1 - d_2 = \left[\frac{(y_k+1-b)}{m} - x_k\right] - \left[x_k+1 - \frac{y_k+1-b}{m}\right]$$

$$d_1 - d_2 = \left[\frac{y_k+1-b-mx_k}{m}\right] - \left[\frac{mx_k+m-y_k-1+b}{m}\right]$$

$$d_1 - d_2 = \frac{y_k+1-b-mx_k-mx_k-m+y_k+1-b}{m}$$

$$m(d_1-d_2) = 2y_k - 2mx_k - 2b + 2 - m$$

To remove m put $m = \frac{\Delta y}{\Delta x}$ and multiply both side by Δx then,

$$\Delta x \cdot \frac{\Delta y}{\Delta x}(d_1-d_2) = \Delta x\, 2y_k - 2\Delta x \cdot \frac{\Delta y}{\Delta x} x_k - 2b\Delta x + 2\Delta x - \Delta x \cdot \frac{\Delta y}{\Delta x}$$

$$\Delta y(d_1-d_2) = 2y_k\Delta x - 2x_k\Delta y - 2b\Delta x + 2\Delta x - \Delta y$$

Let, P_k is the decision parameter, then above expression contains the decision variable x_k and y_k then,

$$P_k = \Delta y(d_1-d_2) = 2y_k\Delta x - 2x_k\Delta y + c$$

where, $c = 2\Delta x - 2b\Delta x - \Delta y$

then the next decision parameter will be P_{k+1}.

$$P_{k+1} = 2\Delta x y_{k+1} - 2\Delta y x_{k+1}$$

$$P_{next} = 2\Delta x y_{next} - 2\Delta y x_{next}$$

Therefore,

$$P_{k+1} - P_k = [2\Delta x y_{k+1} - 2\Delta y x_{k+1}] - [2\Delta x y_k - 2\Delta y x_k]$$

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

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

$$\boxed{P_{k+1} = P_k + 2\Delta x (y_{k+1} - y_k) - 2\Delta y (x_{k+1} - x_k)}$$

OR

$$P_{next} = P_{current} + 2\Delta x (y_{next} - y_{current}) - 2\Delta y (x_{next} - x_{current})$$

Case-I: if $P_k \geq 0$ then

$$x_{k+1} = x_k + 1$$
$$y_{k+1} = y_k + 1 \quad \text{and}$$
$$P_{k+1} = P_k + 2\Delta x - 2\Delta y$$

Case-II: if $P_k < 0$ then

$$x_{k+1} = x_k \text{ and } y_{k+1} = y_k + 1$$

$$P_{k+1} = P_k + 2\Delta x$$

To find initial value of decision parameter

As we know

$$P_k = 2\Delta x y_k - 2\Delta y x_k + c$$

$$P_0 = 2\Delta x y_0 - 2\Delta y x_0 + 2\Delta x - 2b\Delta x - \Delta y$$

$$= 2\Delta x y_0 - 2\Delta y x_0 + 2\Delta x - 2(y_0 - m x_0)\Delta x - \Delta y$$

$$= 2\Delta x y_0 - 2\Delta y x_0 + 2\Delta x - 2\Delta x y_0 + 2\Delta x \cdot m \cdot x_0 - \Delta y$$

$$= -2\Delta y x_0 + 2\Delta x + 2\Delta x \cdot \frac{\Delta y}{\Delta x} \cdot x_0 - \Delta y$$

$$= -2\Delta y x_0 + 2\Delta x + 2\Delta y x_0 - \Delta y$$

$$\boxed{P_0 = 2\Delta x - \Delta y}$$

Which is Required Expression.

2.3. Circle Generation Algorithm (Mid-Point)

Circle: A Circle is defined as a S\set of points that are all at a given distance 'r' from the center position (Xc, Yc).

This distance relationship is expressed by the $(x-x1)^2 + (y-y1)^2 = r^2$

OR

A Circle is a Set of all points lie at an equal distance (called radios) from a fixed point called Centre.

This is also a scan converting algorithm.

equation of the Circle with center (0,0) is $\mathbf{x^2 + y^2 = r^2}$

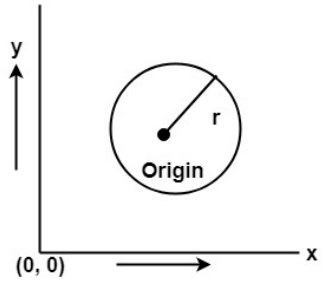
The equation of Circle having center (h, k) is $\mathbf{(x-h)^2 + (y-k)^2 = r^2}$

In the circle drawing

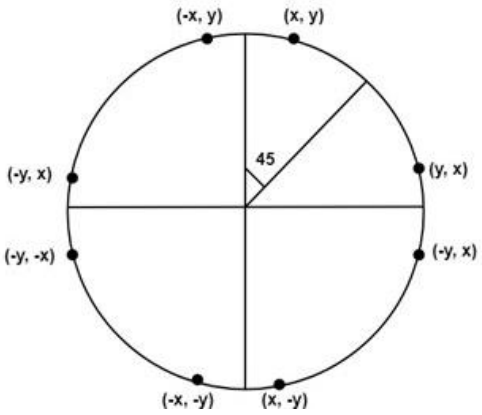
We use symmetry of Circle

For Symmetry, if we draw only one octant, we can easily draw points on other seven octants using reflection procedure.

Let us consider a point P(x, y) then the other seven points will be calculated by using symmetrical properties of circle.

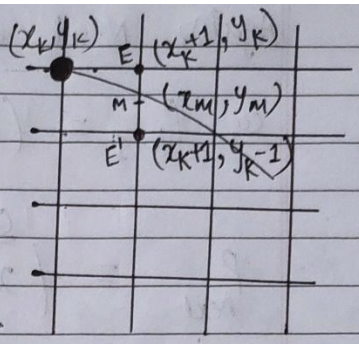


P2 (x, -y)
P3(-x, -y)
P4 (-x, y)
P5 (y, x)
P6 (y, -x)
P7 (-y, -x)
P8 (-y, x)

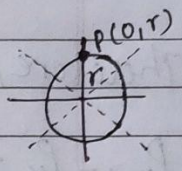


Derivation:

Here, we have to find points for the first octants, when $x \geq y$ then it indicates the end of octant 1st.



The first co-ordinate point is $(0, r)$. to draw the circle, The x-value increasing as unit interval and y-value decreasing along the value of x.



Now, The next pixel to draw is either (x_k+1, y_k) or (x_k+1, y_k-1). so the mid point of these two pixel is (x_m, y_m).

then,
$$x_m = \frac{x_k+1 + x_k+1}{2} = x_k+1$$

and
$$y_m = \frac{y_k + y_k-1}{2} = y_k - \frac{1}{2}$$

(∵ mid-point formulae
$x_m = \frac{x_1+x_2}{2}$
$y_m = \frac{y_1+y_2}{2}$)

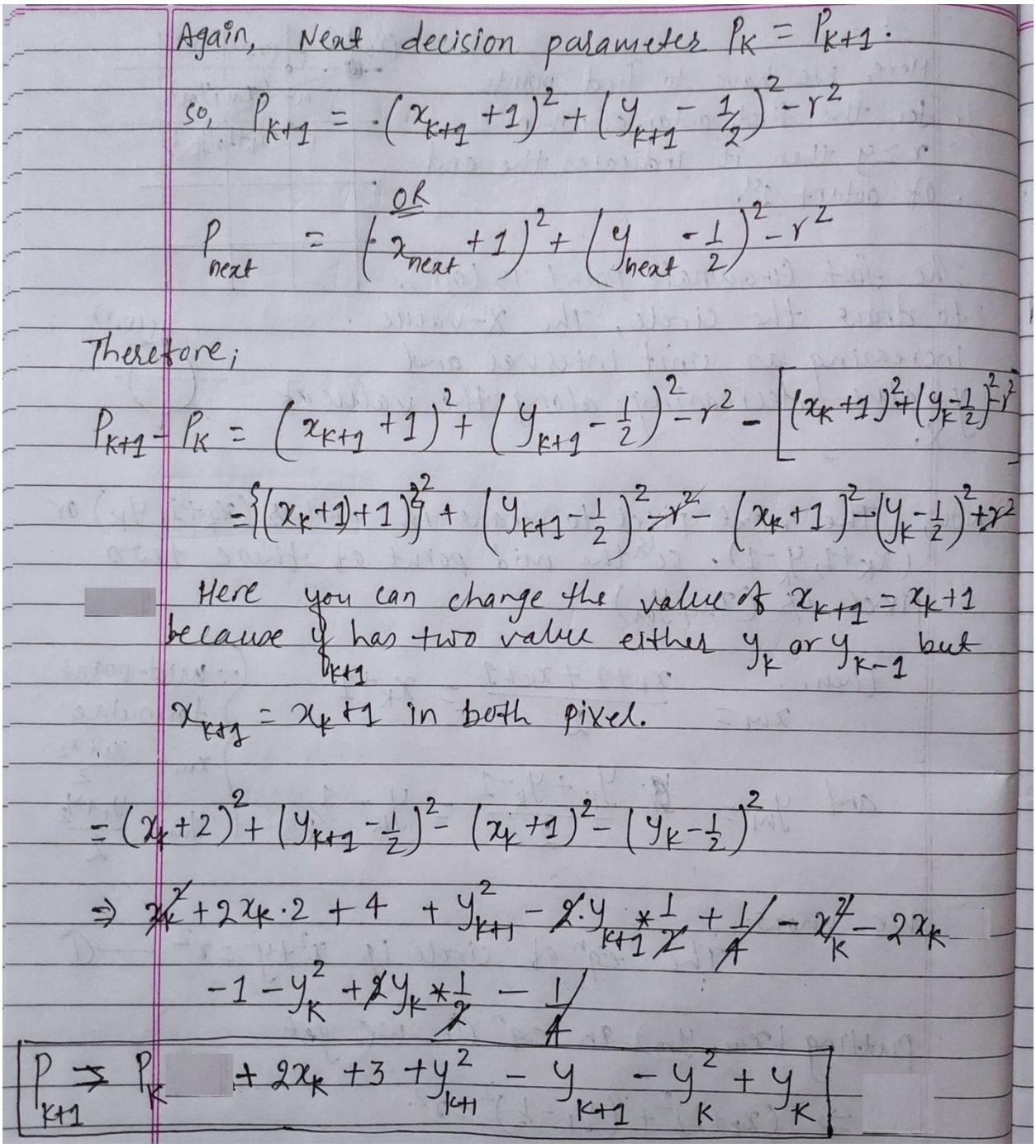
We know
The eqⁿ of circle is $x^2+y^2=r^2$ — ①

putting (x_m, y_m) in eqⁿ ① we get

$$\Rightarrow (x_k+1)^2 + (y_k - \tfrac{1}{2})^2 = r^2$$

Let P_k is the decision parameter then

$$\boxed{P_k = (x_k+1)^2 + (y_k - \tfrac{1}{2})^2 - r^2} \quad \text{— (II)}$$



Here,

X_{k+1} always will be X_k +1 but Y_{k+1} will be either Y_k or Y_k-1 depending on closer distance

Therefore,

$$P_{k+1} = P_k + 2x_k + y_{k+1}^2 - y_k^2 - y_{k+1} + y_k + 3$$

case-I: if $P_k < 0$ then the midpoint is inside the circle and the pixel on the scan line y_k closer to circle boundary.

then next point to plot is:

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

and, $P_{K+1} = P_k + 2X_{K+1} + 1$

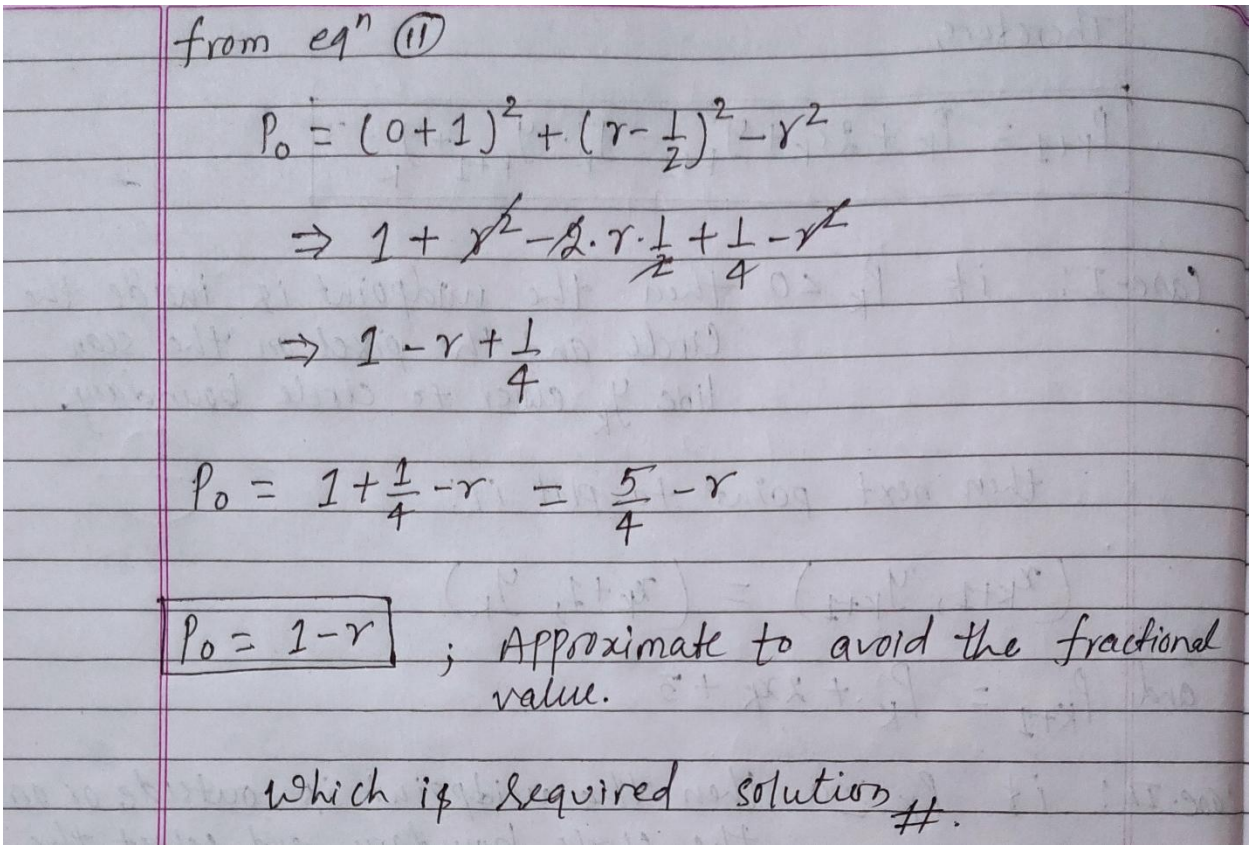
case-II: if $P_k \geq 0$ then the midpoint is outside or on the circle boundary, and select the pixel on scan line $y_k - 1$.

next point to plot is: $(x_{k+1}, y_{k+1}) = (x_k + 1, y_k - 1)$

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

Now, Find the Initial Decision Parameter (P_0)

The starting point is $(0, r)$ so, $x_k = 0$ & $y_k = r$
let the initial decision parameter is P_0.



Algorithm:

1. The input radios 'r' and center (Xc, Yc) to obtain the first octant point in the circumference of a circle is centered on the origin as (X0, Y0) = (0,r)
2. calculate the initial decision parameter **P0= 1- r**
3. at each x k position starting k=0 perform following operation

   3.1. if P k <0 then plotting point will be (x_{k+1}, y_k) and $P_{k+1} = P_k + 2 * x_{k+1} + 1$

   3.2. else plot (x_{k+1}, y_{k-1}) and $P_{k+1} = P_k + 2 * x_{k+1} - 2 * y_{k+1} + 1$
4. Determine the symmetry points in the other octants
5. move at each point by given center i.e. (X+ Xc and Y+Yc) if necessary
6. Repeat step 3 to 5 until the X>=Y.

2.4. Ellipse Generation Algorithm

Ellipse Generation Algorithms

→ Ellipse is an elongated circle, therefore elliptical curves can be generated by modifying circle drawing algorithm's procedures.

→ Circle → has one radious in all direction
ellipse → has two radii

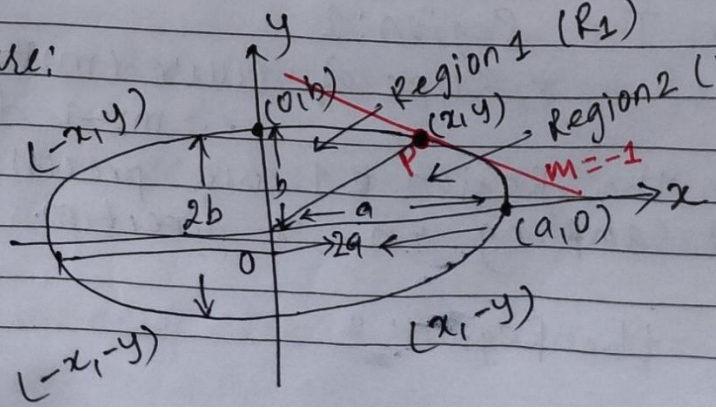
- one in x-axis (called major axis) (2a)
- neat in y-axis (called minor axis) (2b)

→ Due to these two different radii circle and ellipse has different symmetry.

→ In circle 8 point symmetry
In ellipse 4 point symmetry

→ In circle, we need to plot only one octant of any quardant but in ellipse we need to plot two octants (i.e. one complete quardant to plot entire ellipse).

→ See figure:



→ Ellipse generating algorithm is applied through 1st quadant according to the slope of ellipse.

→ slope is obtained by differentiating the ellipse function.

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

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

$$x^2b^2 + y^2a^2 - a^2b^2 = 0 \quad -(1)$$

differentiating (1) w.r to x

$$2xb^2 + 2ya^2\frac{dy}{dx} = 0$$

$$\frac{dy}{dx} = -\frac{2xb^2}{2ya^2}$$

→ At the boundary point between Region 1 and Region 2 of ellipse slope is -1.

i.e. at point P, $\frac{dy}{dx} = -1$. { see figure for point P.

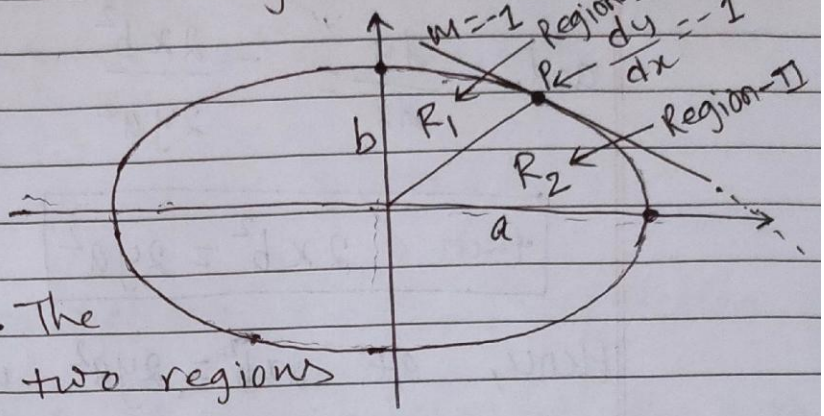
→ For The Region: 1

* slope of curve (m) is greater than -1.
i.e. $m > -1$ for R_1.

So, For the Region: 1 we process by taking unit sampling in x-direction.

→ For the Region: 2

* slope is less than 1. i.e. $m < -1$ for R_2.
* sample in y-direction by unit interval



Derivation:

Let us consider one quarter of an ellipse. The curve is divided into two regions R_1 and R_2.

In the region-I, the slope on the curve is greater than -1 while in region-II slope is less than -1.

Also, consider the general equation of an ellipse,

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

$$\Rightarrow \quad x^2b^2 + y^2a^2 - a^2b^2 = 0$$

where, a is horizontal radious and b is vertical radious

Now,

Lets start from the point (0,b) in region-I and take unit step sampling towards x-dir^n untill the boundary between R_1 and R_2.

and, In R_2 we sample at y-direction to test the slope at each point of curve.

At the boundary region between R_1 and R_2

$\frac{dy}{dx} = -1$

and, $\frac{dy}{dx} = -\frac{2xb^2}{2ya^2}$

$\therefore \frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$

diff.

$2xb^2 - 2ya^2\frac{dy}{dx} = 0$

at boundary $\frac{dy}{dx} = -1$

so, $2xb^2 = 2ya^2$

then $2xb^2 = 2ya^2$

Hence, at $2xb^2 = 2ya^2$ we are in Region-I & Region-II boundary point.

if $2xb^2 > 2ya^2$ we move out of Region-I

Therefore

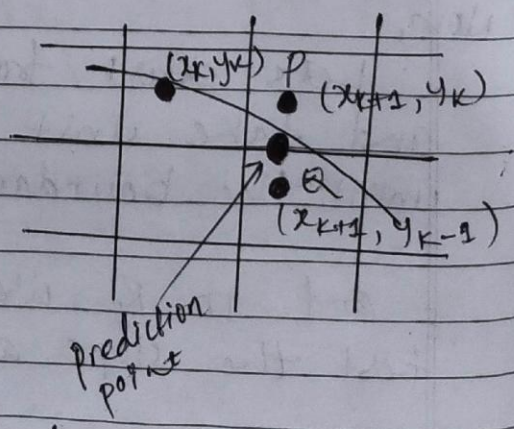
for R_1 $m < 1$ and $2xb^2 < 2ya^2$

for R_2 $m > 1$ and $2xb^2 > 2ya^2$

In Region-I

We know $\frac{dy}{dx} > -1$ and x is always incremented on each step.

also, we know that at R_1, $2xb^2 < 2ya^2$



When x is always incremented on each step

i.e. $x_{k+1} = x_k + 1$

and $y_{k+1} = y_k$ if P is selected

$= y_k - 1$ if Q is selected

i.e (y_{k+1} is either y_k or $y_k - 1$)

To make the decision between P & Q, a prediction $(x_k + 1, y_k - \frac{1}{2})$ is set at the middle bt^n the two candidate pixels. This is calculated by $\begin{cases} x_m = \frac{x_1 + x_2}{2} \\ y_m = \frac{y_1 + y_m}{2} \end{cases}$

put $(x_k + 1, y_k - \frac{1}{2})$ in the eq^n of ellipse then

$$E_{function} = b^2(x_k+1)^2 + a^2(y_k - \tfrac{1}{2})^2 - a^2b^2$$

Let, P_k be the decision parameter as recent expression contains value satisfying standard equation of ellipse.

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

for Region-1 denote P_k as P_{1K}

$$\boxed{P_{1K} = b^2(x_k+1)^2 + a^2(y_k - \tfrac{1}{2})^2 - a^2b^2}$$

$$P_{1(K+1)} = b^2(x_{k+1}+1)^2 + a^2(y_{k+1} - \tfrac{1}{2})^2 - a^2b^2$$

i.e. $$\boxed{P_{1next} = b^2(x_{next}+1)^2 + a^2(y_{next} - \tfrac{1}{2})^2 - a^2b^2}$$

Now we are sampling at x direction so

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

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

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

Now,

$$P_{1(k+1)} - P_k = \left[\{b^2(x_k+1)^2 + 2(x_k+1)+1\} + a^2\left(y_{k+1}-\frac{1}{2}\right)^2 - a^2b^2\right] - \left[b^2(x_k+1)^2 + a^2\left(y_k-\frac{1}{2}\right)^2 - a^2b^2\right]$$

$$\Rightarrow \left[b^2(x_k+1)^2 + 2b^2(x_k+1) + b^2 + a^2(y_k+1)^2 - a^2 * y_{k+1} + a^2 * \frac{1}{4} - a^2b^2\right] - \left[b^2(x_k+1)^2 + a^2y_k^2 - a^2y_k + a^2 * \frac{1}{4} - a^2b^2\right]$$

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

Case-I If $P_{1k} < 0$ then midpoint is inside the ellipse boundary and the pixel on scan line y_k is closer to boundary of ellipse

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

$y_{k+1} = y_k$

and, P_{k+1} for region-1 (Let's denote by $P_{1(k+1)}$)

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

Page

Case-II if $P1_K \geq 0$ then predicted midpoint is outside or on the ellipse boundary then we select pixel y_{K-1} on same scan line.

Hence, $x_{K+1} = x_K + 1$

$y_{K+1} = y_K - 1$

and

$$P1_{K+1} = P1_K + 2b^2 x_{K+1} - 2a^2 y_{K+1} + b^2$$

To Find Initial Decision Parameter for R1.

We have the starting point for the ellipse is $(0, b)$.

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

Then, The next point will be either $(1, b)$ or $(1, b-1)$.

from $(1, b)$ and $(1, b-1)$ we can calculate mid point and i.e $(1, b - \frac{1}{2})$

let, $P1_0$ is the initial decision parameter for R1.

then, $P1_0 = b^2 + a^2(b - \frac{1}{2})^2 - a^2b^2$ {∵putting $(1, b-\frac{1}{2})$ in eqⁿ of ellipse

$$\boxed{P1_0 = b^2 - a^2 b + \frac{a^2}{4}}$$

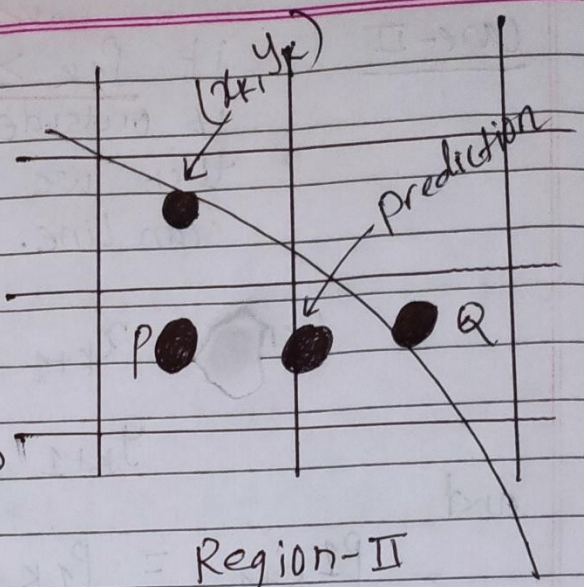
For Region-2

In R2 $\Rightarrow$ $2xb^2 \geq 2ya^2$

In R2, y is always decremented in each step i.e.

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

then x_k has two possibilities



if P is selected

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

if Q is selected $x_{k+1} = x_k + 1$.

Therefore, we have to decide by taking mid point of $(x_k, x_k + 1)$ as prediction pixel.

mid point of PkQ is $\left(x_k + \frac{1}{2}, y_k - 1\right)$

putting this point in standered equation of Ellipse we get,

$$E_{function} = \left(x_k + \frac{1}{2}, y_k - 1\right)$$

$$b^2\left(x_k + \frac{1}{2}\right)^2 + a^2(y_k - 1)^2 - a^2b^2$$

Let, $P2_k$ is the decision parameter for Region-II

then $\boxed{P2_k = b^2\left(x_k + \frac{1}{2}\right)^2 + a^2(y_k - 1)^2 - a^2b^2}$

In Region-2 we are sampling at y-direction

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

$$P2_{k+1} = b^2\left(x_{k+1}+\frac{1}{2}\right)^2 + a^2\left(y_{k+1}-1\right)^2 - a^2b^2$$

i.e. $P2_{next} = b^2\left(x_{next}+\frac{1}{2}\right)^2 + a^2\left(y_{next}-1\right) - a^2b^2$

Now

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

Here only y_{k+1} is $y_k - 1$.

$$\Rightarrow \left\{b^2\left(x_{k+1}+\frac{1}{2}\right)^2 + a^2\left(y_{k+1}-1\right)^2 - a^2b^2 - b^2\left(x_k+\frac{1}{2}\right)^2 - a^2\left(y_{k-1}\right)^2 + a^2b^2\right\}$$

$$\Rightarrow \left\{b^2\left(x_{k+1}+\frac{1}{2}\right)^2 + a^2\left(y_{k+1}-1\right)^2 - b^2\left(x_k+\frac{1}{2}\right)^2 - a^2\left(y_{k-1}\right)^2\right\}$$

$$\Rightarrow b^2\left(x_{k+1}+\frac{1}{2}\right)^2 + a^2\left\{(y_k-1)-1\right\}^2 - b^2\left(x_k+\frac{1}{2}\right)^2 - a^2\left(y_{k-1}\right)^2$$

$$\Rightarrow b^2x_{k+1}^2 + b^2x_{k+1} + \frac{b^2}{4} + a^2\left(y_k-1\right)^2 - 2a^2\left(y_k-1\right) + a^2 - b^2x_k^2 - b^2x_k - \frac{b^2}{4} - a^2\left(y_{k-1}\right)^2$$

$$\Rightarrow \boxed{a^2 + b^2\left(x_{k+1}^2 - x_k^2\right) - 2a^2\left(y_k-1\right) + b^2\left(x_{k+1}-x_k\right)}$$

Case-I if $P2_k < 0$, then the predicted mid point is inside the boundary of ellipse and pixel x_k is closer to ellipse boundary

then,

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

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

and,

$$p2_{k+1} = p2_k + b^2[(x_{k+1}^2 - x_k^2) + (x_{k+1} - x_k)] - 2a^2 (y_k - 1) + a^2$$

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

on solving

$$p2_{k+1} = p2_k + 2b^2 x_{k+1} - 2a^2 y_{k+1} - a^2$$

Case-II if $p2_k \geq 0$ then the predicted mid point is outside or on boundary of ellipse then we choose $x_k + 1$ as closer pixel.

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

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

and, $p2_{k+1} = p2_k + b^2[(x_{k+1}^2 - x_k^2) + (x_{k+1} - x_k)] - 2a^2(y_k - 1) + a^2$

$\Rightarrow p2_k - 2a^2(y_k - 1) + a^2$

$$\boxed{p2_{k+1} \Rightarrow p2_k - 2a^2 y_{k+1} + a^2}$$

To find Initial Decision parameter for R2.

When we enter in Region-II of ellipse, the initial position (x_0, y_0) is taken the last position selected in Region-1 and the initial decision parameter in R2 is:

$$P2_0 = f\left(x_0 + \frac{1}{2}, y_0 - 1\right)$$

put this point in the standerd eqn of ellipse

$$\boxed{P2_0 = b^2\left(x_0 + \frac{1}{2}\right)^2 + a^2(y_0 - 1)^2 - a^2b^2}$$

Findings

for R1

$P1_0 = b^2 + 0.25a^2 - a^2b$

if $P1_K < 0$

$x_{K+1} = x_K + 1$

$y_{K+1} = y_K$

$P1_{K+1} = P1_K + 2b^2(x_{K+1}) + b^2$

else,

$x_{K+1} = x_K + 1$

$y_{K+1} = y_K - 1$

$P1_{K+1} = P1_K + 2b^2x_{K+1} - 2a^2y_{K+1} + b^2$

for R2

$P2_0 = b^2(x0+0.5)^2 + a^2(y-1)^2 - a^2b^2$

if $P2_K < 0$

$x_{K+1} = x_K + 1$

$y_{K+1} = y_K - 1$

$P2_{K+1} = P2_K + 2b^2x_{K+1} - 2a^2y_{K+1} + a^2$

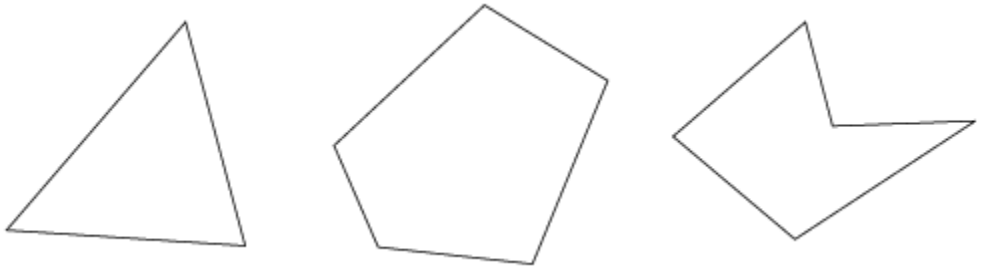
else,

$x_{K+1} = x_K$

$y_{K+1} = y_K - 1$

$P2_{K+1} = P2_K - 2a^2y_{K+1} + a^2$

2.5. Filled Area Primitives



- To provide more realistic view on the various objects filling of color is one of the best ways.
- For filling a picture or object with color's, we have 2 different algorithmic approaches
  1. Area filling scan line algorithm
  2. Seed Fill Algorithm
     - Seed fill Algorithm is further categorized in 2 different algorithms

     2.1 Boundary Fill Algorithm

     2.2 Flood Fill Algorithm

Instead of Using filling algorithms such as Flood fill algorithm, Boundary fill algorithm and scanline polygon fill algorithm, programmatically we can color the objects by Using inbuilt graphics functions such as **floodfill(), setfillstyle()** we can fill the object with color's directly without using any filling algorithm in C program.

2.5.1. Scan Line Polygon Fill Algorithm (Area filling scan line algorithm)

- Also called scan line polygon fill algorithm
- The basic scan-line algorithm is as follows:

1. find the intersections of the scan line with all edges of the polygon

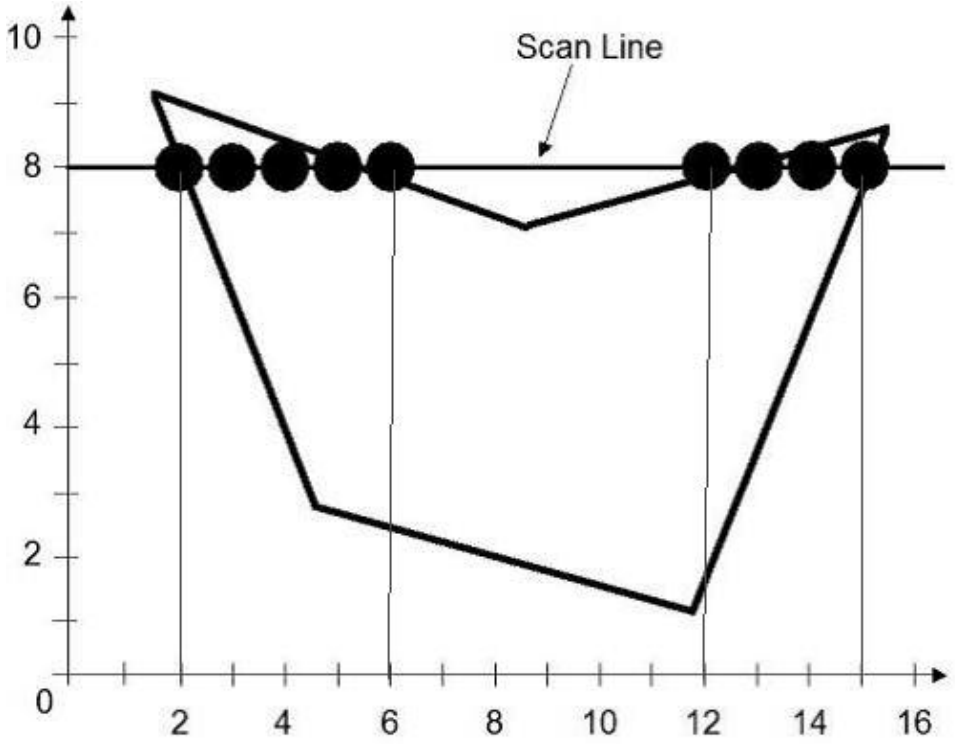
2. Sort the intersections by increasing x coordinate (i.e. short in ascending order of X coordinate)

3. Make pair of two x coordinate

4. Fill in all pixels between pairs of intersections that lie interior to the polygon

- ❖ The scan-line polygon-filling algorithm involves
- ✓ the **horizontal scanning** of the polygon from its **lowermost** to its **topmost** vertex,
- ✓ identifying which edges intersect the scan-line, and
- ✓ Finally drawing the interior horizontal lines with the specified fill color.

For example,



In this polygon ,

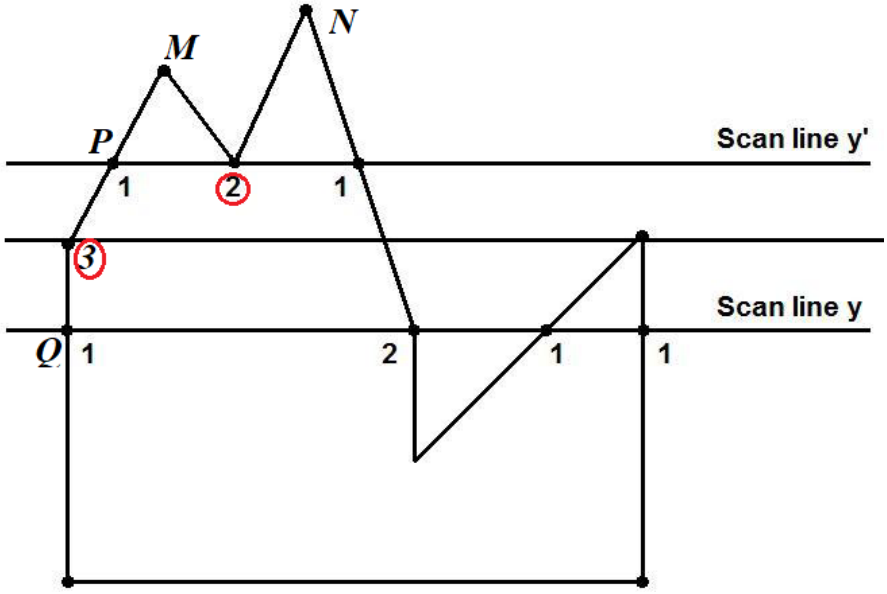
Scan line cuts the polygon edges in x= 2, x=6, x=12 and x=15.

Now sort the x cordinate in ascending order we get – [2,6,12,15]

Then make a pair of sorted pixel as (2,6) and (12,15)

Finally fill the two pair (2,6) and (12,15) with a color of your choice.

1. ***Special case:*** If the scan line intersects in any vertex point



As shown in above polygon, scan line Y' intersects in the vertex 2. Similarly next scan line intersects in vertex 3.

In this case, if all the vertices that are connected in intersected vertex are in same direction of scan line, then repeat the vertex value and pair with other intersecting point.

As→ M and N are two vertex that are connected with vertex 2. M and N both are in same direction of scan line y' (i.e. both M, N are in up direction of scan line or on top). In this case repeat the value of vertex 2 during sorting x direction value in ascending order and make two pair as (1, 2) and (2, 1). Then fill color.

As→P and Q are two vertex that are connected with vertex 3. P and Q both are in opposite direction of second scan line (i.e. P in above the scan line and Q in below of scan line). In this case don't repeat the value of vertex 3 during sorting x direction value in ascending order. Just take it once and make pair with other intersecting point of polygon boundary. Then fill color

2.5.2. Boundary Fill Algorithm

- Also called Edge Fill Algorithm
- An alternative approach for filling an area is to start at a point inside the area and "paint" the interior, point by point, out to the boundary
- This is a particularly useful technique for filling areas with irregular borders

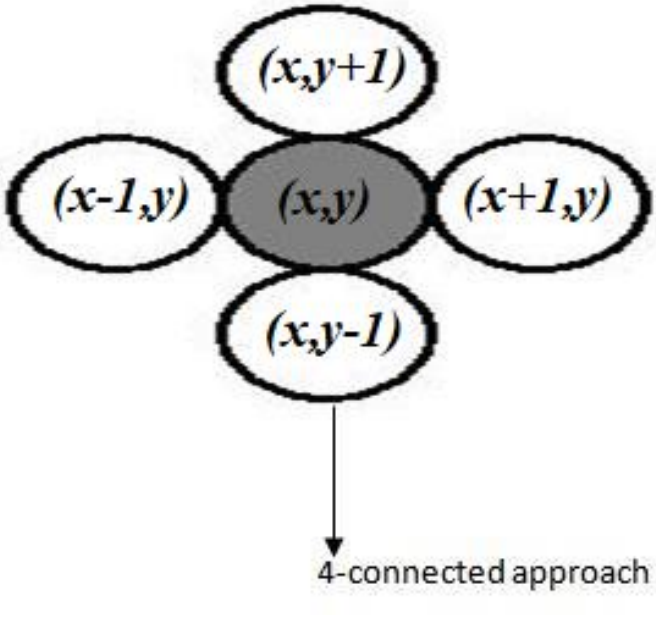
The algorithm makes the following assumptions

- ❖ one interior pixel is known, and
- ❖ Pixels in boundary are known.
- ❖ Boundary pixels are filled with a single color

- ✓ If the boundary of some region is specified in a single color, we can fill the interior of a region, pixel by pixel, until the boundary color is encountered.
- ✓ Basically, a boundary-fill algorithm starts from an interior point (x, y) and sets the neighboring points to the desired color
- ✓ This procedure continues until all pixels are processed up to the designated boundary for the area.
- ✓ Boundary fill algorithm is recursive in nature.

There are two methods for processing neighboring pixels from a current point

1. **4-Connected**
   - ✓ Also called 4-neighbouring points method
   - ✓ These are the pixel positions that are right, left, above, and below the current pixel.



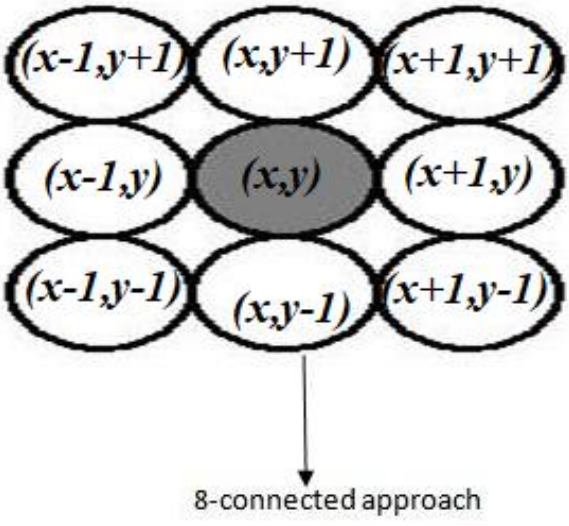
   - ✓ In 4 connected approaches, we can fill an object in only 4 directions. We have 4 possibilities for proceeding to next pixel from current pixel.

Function for 4 connected approaches:

```
Void boundary_fill (int x, int y, int fill_color, int boundary_color)
{
    if ((getpixel(x, y) != boundary_color) && (getpixel(x, y) != fill_color))
        {
            delay (10);
            putpixel(x, y, fill_color);
            boundary_fill(x + 1, y, fill_color, boundary_color);
            boundary_fill(x - 1, y, fill_color, boundary_color);
            boundary_fill(x, y + 1, fill_color, boundary_color);
            boundary_fill(x, y - 1, fill_color, boundary_color);
        }
}
```

2. 8-Connected

- ✓ Also called 8-neighbouring points method
- ✓ This method is used to fill more complex figures.
- ✓ These are the pixel positions that are right, left, above, below and four diagonal points of the current pixel.



- ✓ In 8 connected approaches, we can fill an object in 8 directions. We have 8 possibilities for proceeding to next pixel from current pixel

Function for 8 connected approaches:

```
void boundary_fill(int x, int y, int fill_color, int boundary_color)
{
if ((getpixel(x, y) != boundary_color) && (getpixel(x, y) != fill_color))
                {
                 delay(10);
                                putpixel(x, y, fill_color);
                                boundary_fill(x + 1, y, fill_color, boundary_color);
                                boundary_fill(x , y+1, fill_color, boundary_color);
                                boundary_fill(x+1, y + 1, fill_color, boundary_color);
                                boundary_fill(x-1, y - 1, fill_color, boundary_color);
                                boundary_fill(x-1, y, fill_color, boundary_color);
                                boundary_fill(x , y-1, fill_color, boundary_color);
                                boundary_fill(x-1, y + 1, fill_color, boundary_color);
                                boundary_fill(x+1, y - 1, fill_color, boundary_color);
                }
}
```

Algorithm

1. Start from an interior point.

2. If the current pixel is not already filled and if it is not an edge point, then set the pixel with the fill color, and store its neighboring pixels (**4 or 8-connected**). Store only neighboring pixel that is not already filled and is not an edge point.

3. Select the next pixel from the stack, and continue with step 2.

Flood Fill Algorithm

- ❖ Sometimes it is required to fill in an area that is not defined within a single-color boundary.

- In such cases we can fill areas by replacing a specified interior color instead of searching for a boundary color.
- This approach is called a flood-fill algorithm. Like boundary fill algorithm, here we start with some seed and examine the neighbouring pixels.
- However, here pixels are checked for a specified interior color instead of boundary color and they are replaced by new color.
- ***Using either a 4-connected or 8-connected approach***, we can step through pixel positions until all interior point have been filled.
- The following procedure illustrates the recursive method for filling 4-connected region using flood-fill algorithm.

Procedure:

```
Void  flood-fill(x, y, new-color, old-color)
 {
 if(getpixel (x,y) = = old-color)
 {
delay(10);
putpixel (x, y, new-color)
flood-fill (x + 1, y, new-color, old -color);
flood-fill (x, y + 1, new -color, old -color);
flood-fill (x - 1, y, new -color, old -color);
flood-fill (x, y - l, new -color, old-color);
 }
}
```

In above program *'getpixel()'* function gives the color of. specified pixel and *'putpixel()*' function draws the pixel with specified color.

Algorithm insight

1. We start from a specified interior pixel (x, y) and reassign all pixel values that are currently set to a given interior color with the desired fill color.

2. If the area has more than one interior color, we can first reassign pixel values so that all interior pixels have the same color.

3. Using either 4-connected or 8-connected approach, we then step through pixel positions until all interior pixels have been repainted.

Difference between Boundary fill and Flood Fill **imp**

Boundary Fill Algorithm	**Flood Fill Algorithm**
1. 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 boundary.	Area filling is started from a point and it replaces the old color with the new color
2. It is used in interactive packages where we can specify the region boundary	It is used when we cannot specify the region boundary
3. It is less time consuming	It consumes more time
4. It searches for boundary.	It searches for old color
5. Write algorithmic program yourself from above.	Write algorithmic program yourself from above.

****End of Unit 2****