Tribhuvan University
Institute of Science and Technology
Bachelor of Science in
Computer Science and Information Technology

3RD Semester

CSC209: Computer Graphics

-RAJESH KUMAR BAJGAIN

U2: Scan Conversion Algorithm 6 Hrs

2.1 Scan Converting a Point and a straight Line: DDA Line Algorithm, Bresenham's Line Algorithm

2.2 Scan Converting Circle and Ellipse :Mid Point Circle and Ellipse Algorithm

2.3 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

- A procedure used to digitize or rasterize pixel data available on frame buffer.

- The process of representing continuous graphics object as a collection of discrete pixels is called scan conversion.

For e.g. a line is defined by its two end points and the line equation.

Scan conversion of a point

- Scan conversion of a point requires the two data that are pixel position and color for display.

- In C, a point be scan converted using function putpixel() defined in library.

Header file is <graphics.h>

putpixel(x, y, color)

Here, x & y represent pixel position on 2D display.

Scan Conversion of line

Say $y = mx + b$ be the equation of line with end point (x_1, y_1) and (x_2, y_2) then,

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

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

Here, m represents the slope of line path where by Δx & Δy gives the deflection needed towards horizontal and vertical direction to get new pixel from current pixel position.

- Slope of line also describes the nature and characteristics of line that is going to display.

Line Drawing algorithm

a) Digital Differential Analyzer (DDA) algorithm (Incremental algorithm)

b) Bresenham's line drawing algorithm(BLA)

Digital Differential Analyzer (DDA) algorithm

It is a scan conversion line algorithm based on calculating either Δx or Δy using equation

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

The equation of the line is given as;

$y = mx + b$ ....(i)

$m = (y_2 - y_1)/(x_2 - x_1)$ .... (ii)

For any interval Δx , corresponding interval is given by $\Delta y = m\Delta x$.

If $m <= 1$, and start point is left endpoint then $\Delta x = 1$ and

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

If $m <= 1$, and start point is right endpoint, then $\Delta x = -1$ and

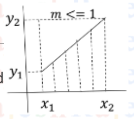
$y_{k+1} = y_k - m$

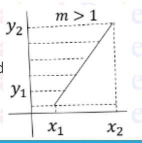
If $m > 1$, and start point is left endpoint, then $\Delta y = 1$ and

$x_{k+1} = x_k + \frac{1}{m}$

If $m > 1$, and start point is right endpoint, then $\Delta y = -1$ and

$x_{k+1} = x_k - \frac{1}{m}$

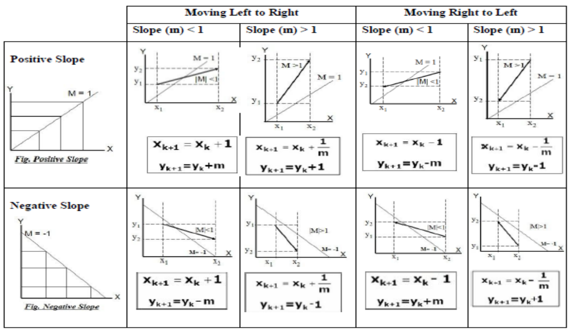




Algorithm for DDA line drawing algorithm

```
Step 1: Start
Step 2: Input the line endpoints and store the left endpoint in (x1,y1)
        and right endpoint in (x2,y2).
Step 3: Calculate the values of dx and dy
            dx=x2-x1
            dy=y2-y1
Step 4: If (abs(dx)>abs(dy)) then
            steplength=abs(dx)
        else
            steplength=abs(dy)
```

```
Step 5: Calculate the values of x-increment and y-increment as,
            xIncrement=dx/steplength
            yIncrement=dy/steplength
Step 6: Set x=x1 and y=y1
Step 7: Draw the pixel(round(x),round(y))
Step 8: for k=0 to steplength do
            x=x+xIncrement
            y=y+yIncrement
            Draw the pixel(round(x),round(y))
Step 9: End
```



	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*	$x_{k+1} = x_k + 1$ $y_{k+1} = y_k + m$	$x_{k+1} = x_k + \frac{1}{m}$ $y_{k+1} = y_k + 1$	$x_{k+1} = x_k - 1$ $y_{k+1} = y_k - m$	$x_{k+1} = x_k - \frac{1}{m}$ $y_{k+1} = y_k - 1$
Negative Slope *Fig. Negative Slope*	$x_{k+1} = x_k + 1$ $y_{k+1} = y_k - m$	$x_{k+1} = x_k + \frac{1}{m}$ $y_{k+1} = y_k - 1$	$x_{k+1} = x_k - 1$ $y_{k+1} = y_k + m$	$x_{k+1} = x_k - \frac{1}{m}$ $y_{k+1} = y_k + 1$

Advantages of DDA

- It is simple to understand.
- It requires no special skills for implementation.
- It is faster method than direct use of the line equation y=mx+c.

Disadvantages of DDA

- m is stored in floating point number.
- Accumulation of round-off error in successive additions can cause calculated pixel positions to drift away from the actual line path for long line segments.
- Rounding operations and floating-point-arithmetic are time consuming.

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

Solution:

Given,

$(x_1, y_1) = (0, 0)$

$(x_2, y_2) = (4, 5)$

$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{5-0}{4-0} = 1.25$

Since, $m > 1$, from DDA algorithm we have;

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

$x_{k+1} = x_k + \frac{1}{m}$

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)

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

Solution: Given,

$(x_1, y_1) = (3, 7)$

$(x_2, y_2) = (8, 3)$

$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{3-7}{8-3} = -0.8$

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	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)

Practice Time

Q.> Digitize a Line with end point A(2,3) and B(6,8) , using DDA Right to left.

Q.> Digitize a Line with end point A(3,2) and B(8,4) , using DDA

Q.>Digitize a Line with end point A(2,6) and B(4,2) , using DDA

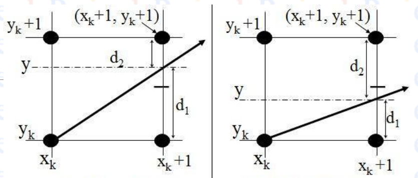
Bresenham's Line drawing Algorithm (BLA)

Advantage of BLA over DDA

- In BLA each successive point is calculated in integer unlike DDA in floating point, hence required less time and memory
- In BLA it does not need to round, so there is no accumulation of rounding error like that of DDA
- 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

BLA for slope +ve and |m| ≤ 1



$P_k = d_1 - d_2$ = Positive

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

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

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

$P_k = d_1 - d_2$ = Negative

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

$y_{k+1} = y_k$

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

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

Let us assume that pixel (x_k, y_k) is already plotted assuming that the sampling direction is along X-axis i.e. (x_k+1, y_k) or (x_k+1, y_k+1).

Thus, the common equation of the line is $y = m(x_k+1) + c$

Now, $d1 = y - y_k = m(x_k+1) + c - y_k$

and, $d2 = (y_k+1) - y = y_k+1 - \{m(x_k+1) + c\}$

The difference between these two separation is,

$d1 - d2 = [m(x_k+1) + c - y_k] - [y_k+1 - \{m(x_k+1) + c\}]$

Or, $d1 - d2 = 2m(x_k+1) + 2c - 2y_k - 1$

Since, slope of line **(m) = Δy / Δx**, We have,

$\Delta x(d1 - d2) = 2\Delta y(x_k+1) + 2\Delta x C - 2\Delta x Y_k - \Delta x$

Define Decision parameter at Kth step,

$P_k = \Delta x(d1 - d2)$

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

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

Now , K+1th term

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

Here, $P_{k+1} - P_k = 2\Delta y X_{k+1} + 2\Delta y + 2\Delta x c - 2\Delta x y_{k+1} - \Delta x - \{2\Delta y X_k + 2\Delta y + 2\Delta x c - 2\Delta x y_k - \Delta x\}$

$= 2\Delta y X_{k+1} + 2\Delta y + 2\Delta x c - 2\Delta x y_{k+1} - \Delta x - 2\Delta y X_k - 2\Delta y - 2\Delta x c + 2\Delta x y_k + \Delta x$

$P_{k+1} = P_k + 2\Delta y X_{k+1} - 2\Delta x y_{k+1} - 2\Delta y X_k + 2\Delta x y_k$

$$\mathbf{P_{k+1} = P_k + 2\Delta y(X_{k+1} - X_k) - 2\Delta x(Y_{k+1} - Y_k)}$$

Case 1

If $P_k < 0$ (i.e. d1-d2 is Negative)

then,

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

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

Case 2

If $P_k \geq 0$ (i.e. d1-d2 is Positive)

then,

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

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

Initial Decision Parameter (P_0)

$y = m(x_k+1) + c$

Let, $X_0 = 0$, $Y_0 = 0$ then $C = 0$

$\Delta x(d1 - d2) = 2\Delta y X_k + 2\Delta y + 2\Delta x c - 2\Delta x y_k - \Delta x$

$P_0 = 0 + 2\Delta y + 0 + 0 - \Delta x$

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

Digitize the line with end points (20, 10) and (30, 18) using BLA.

Here, Starting point of line = (x_1, y_1) = (20, 10) and
Ending point of line = (x_2, y_2) = (30, 18)

Thus, slope of line, $m = \Delta y / \Delta x = (y_2-y_1) / (x_2-x_1)$
= (18-10) / (30-20) = 8/10

As the given points, it is clear that the line is moving left to right with the positive slope |m| = 0.8 <1

Thus,

The initial decision parameter $(P_0) = 2\Delta y - \Delta x = 2*8 – 10 = 6$

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

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

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

(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)

Let us assume that pixel (x_k, y_k) is already plotted assuming that the sampling direction is along X-axis i.e. (x_k, y_k+1) or (x_k+1, y_k+1).

Thus, the common equation of the line is $y=mx+c$

$y_K+1 = mx + c$

$x= \{y_k+1 – c\}/m$

Now, $d1 = x - x_k = \{y_k +1 – c\}/m - x_k$

And $d2 = (x_k +1)- X = x_k+1– \{y_k+1 – c\}/m$

So, $d1 - d2 = [\{y_k +1 – c\}/m - x_k] -[x_k+1–\{\{y_k+1 – c\}/m \}]$

Or, $d1 - d2 = 2(y_k+1) /m - 2c/m – 2x_k -1$

Since, slope of line (m) = $\Delta y / \Delta x$,

we have

$P_k = \Delta y\,(d1 - d2)$
$= 2\Delta x\,(y_k+1) - 2c\,\Delta x - 2\Delta y\, x_k – \Delta y$

Now , K+1th term

$P_{k+1} = 2\Delta x\,(y_{k+1}+1) - 2c\,\Delta x - 2\Delta y\, x_{k+1} – \Delta y$

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

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

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

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

Initial Decision Parameter (P_0)

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

Let, $X_0 = 0$, $Y_0 = 0$ then C = 0

Then, $P_0 = 2\Delta x\,(y_0 +1) - 2c\,\Delta x - 2\Delta y\, x_0 – \Delta y$

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

Digitize the line with end points (1, 0) and (3, 3) using BLA.

Here, Starting point of line = $(x_1, y_1) = (1, 0)$ and

Ending point of line = $(x_2, y_2) = (3, 3)$

Thus, slope of line, $m = \Delta y / \Delta x = (y_2-y_1) / (x_2-x_1)$

$= (3-0) / (3-1) = 3/2$

As the given points, it is clear that the line is moving left to right with the positive slope, $|m| = 1.5 > 1$

Thus, the initial decision parameter

$(P_0) = 2\Delta x - \Delta y = 2*2 - 3 = 1$

$\Delta x = x_2-x_1 = 3-1 = 2$

$\Delta y = y_2-y_1 = 3-0 = 3$

we have

If $P_k < 0$ then,

$X_{k+1} = X_k$

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

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

If $P_k \geq 0$ then,

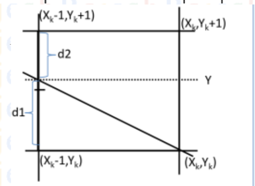
$X_{k+1} = X_k+1$

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

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

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)

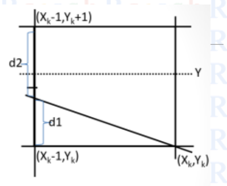
Slope –Ve and |M| ≤ 1



d1-d2 > 0 (Positive)

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

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



d1-d2 < 0 (Negative)

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

$Y_{k+1} = Y_k$

Case 1

If $P_k < 0$ (i.e. d1-d2 is Negative) then,

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

$Y_{k+1} = Y_k$

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

Case 2

If $P_k \geq 0$ (i.e. d1-d2 is Positive) then,

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

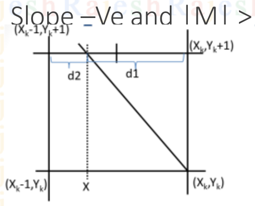
$Y_{k+1} = Y_k + 1$

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

Initial Decision Parameter (P0)

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

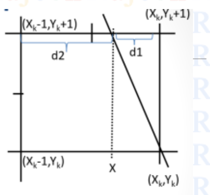
Slope –Ve and |M| > 1



d1-d2 > 0 (Positive)

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

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



d1-d2 < 0 (Negative)

$X_{k+1} = X_k$

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

Case 1

If $P_k < 0$ (i.e. d1-d2 is Negative) then,

$X_{k+1} = X_k$

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

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

Case 2

If $P_k \geq 0$ (i.e. d1-d2 is Positive) then,

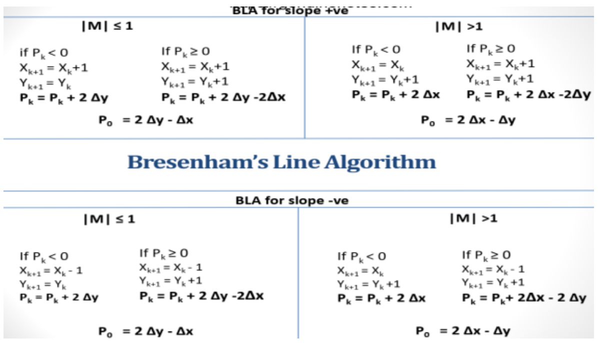
$X_{k+1} = X_k - 1$

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

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

Initial Decision Parameter (P_0)

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



BLA for slope +ve

$\|M\| \le 1$		$\|M\| > 1$	
if $P_k < 0$	If $P_k \ge 0$	If $P_k < 0$	If $P_k \ge 0$
$X_{k+1} = X_k+1$	$X_{k+1} = X_k+1$	$X_{k+1} = X_k$	$X_{k+1} = X_k+1$
$Y_{k+1} = Y_k$	$Y_{k+1} = Y_k+1$	$Y_{k+1} = Y_k+1$	$Y_{k+1} = Y_k+1$
$P_k = P_k + 2\,\Delta y$	$P_k = P_k + 2\,\Delta y - 2\Delta x$	$P_k = P_k + 2\,\Delta x$	$P_k = P_k + 2\,\Delta x - 2\Delta y$
$P_0 = 2\,\Delta y - \Delta x$		$P_0 = 2\,\Delta x - \Delta y$	

Bresenham's Line Algorithm

BLA for slope -ve

$\|M\| \le 1$		$\|M\| > 1$	
If $P_k < 0$	If $P_k \ge 0$	If $P_k < 0$	If $P_k \ge 0$
$X_{k+1} = X_k - 1$	$X_{k+1} = X_k - 1$	$X_{k+1} = X_k$	$X_{k+1} = X_k - 1$
$Y_{k+1} = Y_k$	$Y_{k+1} = Y_k+1$	$Y_{k+1} = Y_k+1$	$Y_{k+1} = Y_k+1$
$P_k = P_k + 2\,\Delta y$	$P_k = P_k + 2\,\Delta y - 2\Delta x$	$P_k = P_k + 2\,\Delta x$	$P_k = P_k + 2\Delta x - 2\,\Delta y$
$P_0 = 2\,\Delta y - \Delta x$		$P_0 = 2\,\Delta x - \Delta y$	

Step 1: Input the two line endpoints and store the left endpoint in (x_0, y_0).

Step 2: Load (x_0, y_0) into the frame buffer i.e. 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 tests:

If $P_k < 0$, then next point to plot is $(x_k + 1, y_k)$ and $P_{k+1} = P_k + 2\Delta y$

Otherwise, the next point to plot is $(x_k + 1, y_k + 1)$ and $P_{k+1} = P_k + 2\Delta y - 2\Delta x$

Step 5: Repeat step 4 Δx times.

[Note: For m>1, just interchange the role of x & y]

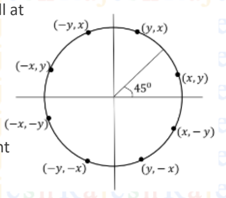
Circle Algorithm

Circle is defined as a set of points that are all at a given distance 'r' from the center position (x_c, y_c).

Equation of circle centered at (x_c, y_c) with radius 'r' is $(x - x_c)^2 + (y - y_c)^2 = r^2$

Symmetry of Circle

Calculation of circle point (x, y) in one octant yields the circle points shown for the other seven octants.



The mid-point Circle Algorithm

$P_k < 0$

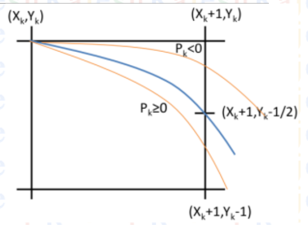
$X_{K+1} = X_k+1$

$Y_{K+1} = Y_k$

$P_k \ge 0$

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

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

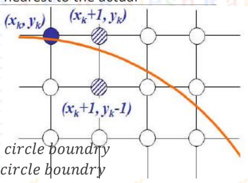


- Assume that we have just plotted point (x_k, y_k).
- The next point is a choice between (x_k+1, y_k) and (x_k+1, y_k-1).
- We would like to choose the point that is nearest to the actual circle.
- Let us define a circle function as:

$$f_{\text{circle}}(x, y) = x^2 + y^2 - r^2$$

- Any point satisfies following conditions:

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



By evaluating this function at the midpoint between the candidate pixels we can make our decision.

Our decision variable can be defined as:

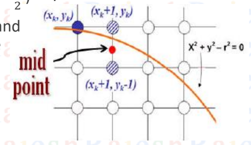
$$p_k = f_{\text{circle}}\,(x_k + 1, y_k - \tfrac{1}{2}) = (x_k + 1)^2 + (y_k - \tfrac{1}{2})^2 - r^2$$

If $p_k < 0$ the midpoint is inside the circle and the pixel at y_k is closer to the circle

Otherwise, $y_k - 1$ is closer.

The decision parameter for next position is at $x_{k+1} + 1$

$= x_k + 1 + 1$ i.e. $x_k + 2$.



$p_{k+1} = f_{\text{circle}}(x_{k+1} + 1, y_{k+1} - \frac{1}{2}) = (x_k + 2)^2 + (y_{k+1} - \frac{1}{2})^2 - r^2$

Now, $p_{k+1} - p_k = (x_k + 2)^2 + (y_{k+1} - 1/2)^2 - r^2 - (x_k + 1)^2 - (y_k - 1/2)^2 + r^2$

$p_{k+1} = p_k + x_k^2 + 4x_k + 4 + y_k{}^2{}_{+1} - y_{k+1} + 1/4 - x_k^2 - 2x_k - 1 - y_k^2 + y_k - 1/4$

$p_{k+1} = p_k + 2x_k + 3 + (y^2{}_{k+1} - y^2{}_k) - (y_{k+1} - y_k)$

$\boldsymbol{p_{k+1} = p_k + 2x_{k+1} + (y^2{}_{k+1} - y^2{}_k) - (y_{k+1} - y_k) + 1}$

Where, y_{k+1} is either y_k or y_{k-1} depending on the sign of p_k.

If $p_k < 0$; $y_{k+1} = y_k \Longrightarrow$ $\boldsymbol{p_{k+1} = p_k + 2x_{k+1} + 1}$

If $p_k \geq 0$; $y_{k+1} = y_k - 1 \Longrightarrow$ $\boldsymbol{p_{k+1} = p_k + 2x_{k+1} + 1 - 2y_{k+1}}$

For initial decision parameter: $(x_0, y_0) = (0, r)$

$p_0 = f(1, r - 1/2) = 1 + (r - 1/2)^2 - r^2 = 5/4 - r$

All increments are integer, rounding 5/4 will give 1 so,

$p_0 = 1 - r$

Step 1: Input radius 'r' and circle center (x_c, y_c) and obtain the first point on the circumference of a circle centered on origin as $(x_0, y_0) = (0, r)$

Step 2: Calculate the initial value of the decision parameter as $p_0 = 5/4 - r$

Step 3: At each x_k position, starting at k=0, perform the following test:

If $p_k < 0$, the next point on circle is $(x_k + 1, y_k)$ and
$p_{k+1} = p_k + 2x_{k+1} + 1$

Otherwise, the next point on circle is $(x_k + 1, y_k - 1)$ and
$p_{k+1} = p_k + 2x_{k+1} + 1 - 2y_{k+1}$

Step 4: Determine the symmetry point in other seven octants.

Step 5: Move each calculated pixel position (x, y) onto the circular path centered on (x_c, y_c) and plot the co-ordinate values, $x = x + x_c$, $y = y + y_c$

Step 6: Repeat step 3 through 5 until $x \geq y$.

Digitize the circle $\boldsymbol{x^2 + y^2 = 100}$ in first octant.

Here,

Center = (0, 0)

Radius (r) =10

Initial point = (0, r) = (0, 10)

Initial decision parameter $p_0 = 1 - r = 1 - 10 = -9$

From mid-point circle algorithm we have;

If $p_k < 0$;

Plot $(x_k + 1, y_k)$ and $p_{k+1} = p_k + 2x_{k+1} + 1$

$p_k \geq 0$;

Plot $(x_k + 1, y_k - 1)$ and $p_{k+1} = p_k + 2x_{k+1} + 1 - 2y_{k+1}$

If $p_k < 0$;

Plot $(x_k + 1, y_k)$ and $p_{k+1} = p_k + 2x_{k+1} + 1$

$p_k \geq 0$;

Plot $(x_k + 1, y_k - 1)$ and $p_{k+1} = p_k + 2x_{k+1} + 1 - 2y_{k+1}$

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

Digitize the circle with radius r =10 centered (3, 4) in first octant.

Note: When center is not origin, we first calculate the octants points of the circle in the same way as the center at origin then add the given circle center on each calculated pixel.

k	p_k	(x_{k+1}, y_{k+1}) at (0, 0)	$2x_{k+1}$	$2y_{k+1}$	(x_{k+1}, y_{k+1}) at (3, 4)
0	-9	(1, 10)	2	20	(4, 14)
1	-6	(2, 10)	4	20	(5, 14)
2	-1	(3, 10)	6	20	(6, 14)
3	6	(4, 9)	8	18	(7, 13)
4	-3	(5, 9)	10	18	(8, 13)
5	8	(6, 8)	12	16	(9, 12)
6	5	(7, 7)	14	14	(10, 11)

Midpoint Ellipse Algorithm

Similar approach 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.

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

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

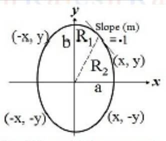


Fig. Ellipse with 4-symmetry & two region in each quadrant

Midpoint Ellipse Algorithm

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

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

This has the following properties:

$F_{ellipse}(x,y)$
- < 0, if (x,y) is inside the ellipse boundary
- = 0, if (x,y) is on the ellipse boundary
- > 0, if (x,y) is outside the ellipse boundary

Midpoint 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:
  $2xb^2 + 2ya^2 * dy/dx= 0$ Or $dy/dx = - 2xb^2 / 2ya^2$
- At the boundary between region 1 and region 2, $dy/dx = - 1$ and $2xb^2 = 2ya^2$.
- Therefore, we move out of region 1 whenever $2xb^2>=2ya^2$.

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

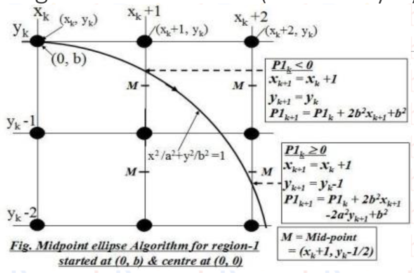


Fig. Midpoint ellipse Algorithm for region-1 started at (0, b) & centre at (0, 0)

Midpoint Ellipse Algorithm-Region1

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:

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

At next sampling position, the decision parameter will be

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

Now, Subtracting $P1_k$ from $P1_{k+1}$, We get

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

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

<u>Case 1: $P1_k<0$</u>	<u>Case 2: $P1_k \geq 0$</u>
$x_{k+1} = x_k+1$	$x_{k+1} = x_k+1$
$y_{k+1} = y_k$	$y_{k+1} = y_k - 1$
$P1_{k+1} = P1_k + 2b^2x_{k+1} +b^2$	$P1_{k+1} = P1_k + 2b^2x_{k+1} -2a^2y_{k+1} +b^2$

Initial decision parameter $(P1_0)$ for region-1 of ellipse 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 -1/2). Thus

$P1_0 = F_{ellipse}(1, b-1/2) \quad = b^2 + a^2 (b -1/2)^2 - a^2 b^2$

$P1_0 = b^2 - a^2 b^2 + a^2 * 1/4$

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

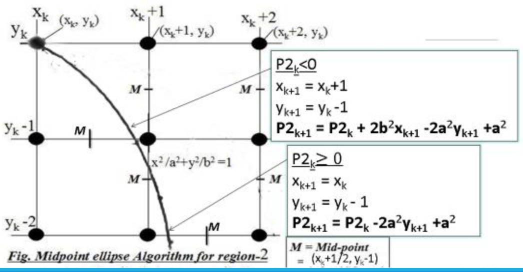


Fig. Midpoint ellipse Algorithm for region-2

Midpoint Ellipse Algorithm-Region2

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:

$P2_k = F_{ellipse}(x_k+1/2, y_k-1) \qquad = b^2(x_k+1/2)^2 + a^2(y_k-1)^2 - a^2b^2$

At next sampling position, the decision parameter will be

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

Now, Subtracting $P2_k$ from $P2_{k+1}$, We get

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

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

Case 1: $P2_k<0$	Case 2: $P2_k \geq 0$
$x_{k+1} = x_k+1$	$x_{k+1} = x_k$
$y_{k+1} = y_k-1$	$y_{k+1} = y_k - 1$
$\mathbf{P2_{k+1} = P2_k + 2b^2x_{k+1} - 2a^2y_{k+1} + a^2}$	$\mathbf{P2_{k+1} = P2_k - 2a^2y_{k+1} + a^2}$

When we enter region2, the initial position (x_0, y_0) is taken as the last position selected in region 1 and the initial decision parameter in region 2 is given by:

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

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

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.

$2b^2x = 0$	(with increment $2b^2 = 72$)
$2a^2y = 2a^2b$	(with increment $-2a^2 = -128$)

For region-1: The initial point for the ellipse centered on the origin is $(x_0, y_0) = (0,6)$, and the initial decision parameters value is:

$p1_0 = b^2 - a^2b^2 + a^2/4 = -332$

Successive decision parameter values and positions along the ellipse path are calculated using the midpoint method as:

k	$P1_k$	$(X_{k+1}, Y_{k+1})_{At\ (0,0)}$	$2b^2X_{k+1}$	$2a^2Y_{k+1}$
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)	540	384

We now move out of region 1, since $2b^2x > 2a^2y$

For region 2: The initial point is $(x_0, y_0) = (7,3)$ and the initial decision parameter is: $P2_0 = b^2(x_0+1/2)^2 + a^2(y_0-1)^2 - a^2b^2 = -151$

The remaining positions along the ellipse path in the first quadrant are then calculated as

k	$P1_k$	$(X_{k+1}, Y_{k+1})_{At\ (0,0)}$	$2b^2X_{k+1}$	$2a^2Y_{k+1}$
0.	-151	(8, 2)	576	256
1.	233	(8, 1)	576	128
2.	748	(8, 0)	-	-

Mid-point Ellipse Algorithm:

Step 1: Input r_x, r_y and center (x_c, y_c) and obtain the first point on ellipse centered at origin as $(x_0, y_0) = (0, r_y)$

Step 2: Calculate the initial value of decision parameter in region 1 as $p1_0 = r_y^2 - r_x^2r_y + \frac{1}{4}r_x^2$

Step 3: At each step x_k position in region 1, starting at k=0, perform the following test:

If $p1_k < 0$, the next point on ellipse centered at (0, 0) is $(x_k + 1, y_k)$ and

$p1_{k+1} = p1_k + 2r_y^2x_{k+1} + r_y^2$

Otherwise, the next point along the ellipse is $(x_k + 1, y_k - 1)$ and
$p1_{k+1} = p1_k + 2r_y^2x_{k+1} - 2r_x^2y_{k+1} + r_y^2$

With, $2r_y^2x_{k+1} = 2r_y^2x_k + 2r_y^2$, $2r_x^2y_{k+1} = 2r_x^2y_k - 2r_x^2$

And continue until $2r_y^2x \geq 2r_x^2y$

Step 4: Calculate the initial value of the decision parameter in region 2 using the last point (x_0, y_0) calculated in region 1 as
$p2_0 = r_y^2\,(x_0 + 1/2)^2 + r_x^2(y_0 - 1)^2 - r_x^2r_y^2$

Step 5: At each step y_k in region 2, starting at k=0, perform the following test:

If $p2_k > 0$, the next point along the ellipse centered on (0, 0) is $(x_k, y_k - 1)$ and $p2_{k+1} = p2_k - 2r_x2y_{k+1} + r_x^2$

Otherwise, the next point is $(x_k + 1, y_k - 1)$ and
$p2_{k+1} = p2_k + 2r_y^2x_{k+1} - 2r_x^2y_{k+1} + r_x^2$

Step 6: Determine symmetry points in the other three quadrants.

Step 7: Move each calculated pixel position (x, y) onto the elliptical path centered on (x_c, y_c) and plot the coordinate values:
$x = x + x_c,\ y = y + y_c$

Step 8: Repeat the steps for region 1 until $2r_y^2x \geq 2r_x^2y$ and region 2 until $(r_x, 0)$.

Area Filling

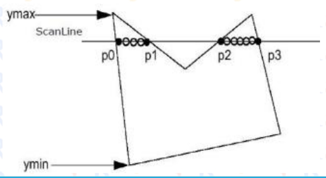
- Filling of polygon with solid color i.e. to color a polygon.
- Coloring must be done only inside its boundary.
- There are two basic approaches to area filling in raster systems:
  - To determine the overlap intervals for scan lines that crosses the area.
  - To start from a given interior position and point outward from this until a specified boundary is met.

Algorithms for filled area primitives

a) Scan line polygon fill algorithm
b) Boundary fill algorithm
c) Flood fill algorithm

Scan line polygon fill algorithm

This algorithm works by intersecting scanline with polygon edges and fills the polygon between pairs of intersections.



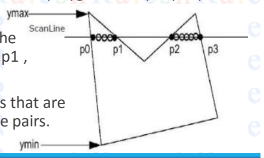
Scan line polygon fill algorithm

Step 1 – Find out the Ymin and Ymax from the given polygon.

Step 2 – ScanLine intersects with each edge of the polygon from Ymin to Ymax. Name each intersection point of the polygon. Like, p0, p1, p2, p3.

Step 3 – Sort the intersection point in the increasing order of X coordinate i.e. p0,p1 , p1,p2 , and p2,p3

Step 4 – Fill all those pair of coordinates that are inside polygons and ignore the alternate pairs.

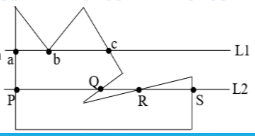


Problem: Scan line polygon fill algorithm

- For scan line L1 a, b & c are interior point, therefore we take pairwise points (a, b) & (b,c) and fill all the pixel between these points.

- For scan line L2, we should only take pairwise points (P, Q) & (R, S) because (Q, R) is not part of polygon.

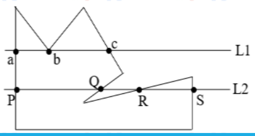
- For scan line L1, we took the vertex 'b' twice i.e.(a, b) & (b, c) but for scan line 'L2' we did not take 'Q' twice.



Solution: Scan line polygon fill algorithm

- Make a clockwise or anticlockwise traverse on the edge.

- If 'y' is monotonically increasing or decreasing and direction of 'y' changes, then we have take the vertex twice, otherwise take vertex only once.



Inside outside Test

- This test is used to identify whether a given point is inside the polygon or outside the polygon.

- There are two methods by which we can identify whether particular point is inside an object or outside.

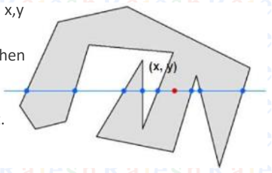
- Odd-Even Rule
- Nonzero winding number rule

Odd-Even Rule

In this technique, we will count the edge crossing along the line from any point x,y to infinity.

If the number of interactions is odd, then the point x,y is an interior point; and

if the number of interactions is even, then the point x,y is an exterior point.



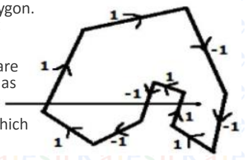
Nonzero Winding Number Rule

Give directions to all the edges of the polygon. Draw a scan line from the point to be test towards the left most of X direction

1. Give the value 1 to all the edges which are going to upward direction and all other -1 as direction values.

2. Check the edge direction values from which the scan line is passing and sum up them.

3. If the total sum of this direction value is non-zero, then this point to be tested is an interior point, otherwise it is an exterior point.



Nonzero Winding Number Rule

It can also be understood with the help of pin and rubber band test.

Fix up the pin on one of the edge of the polygon and tie-up the rubber band in it and then stretch the rubber band along the edges of the polygon.

When all the edges of the polygon are covered by the rubber band, check out the pin which has been fixed up at the point to be test.

If we find at least one wind at the point consider it within the polygon, else we can say that the point is not inside the polygon.

Scan-Line Fill of Curved Boundary area

It requires more work than polygon filling, since intersection calculation involves nonlinear boundary.

For simple curves such as circle or ellipses, performing a scan line fill is straight forward process.

Fig: Interior fill of an elliptical arc

Scan-Line Fill of Curved Boundary area

We only need to calculate the two scan-line intersection on opposite sides of the curve.

Then simply fill the horizontal spans of pixel between the boundary points on opposite side of curve.

Symmetries between quadrants are used to reduce the boundary calculation.

We can fill generating pixel position along curve boundary using mid-point method

Boundary fill Algorithm

- It accepts an input, the co-ordinate of interior point $p(x, y)$, a fill color and a boundary color.

- Starting from point $p(x, y)$, test is performed to determine whether the neighboring pixel is already filled or boundary is reached.

- If not, the neighboring pixels are filled with fill color and their neighbors are tested.

- This process is repeated till boundary is reached.

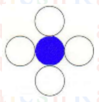
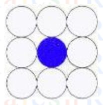
- This algorithm is used when boundary is of single color.

Boundary fill Algorithm

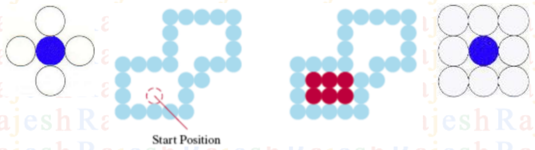
- Two mechanisms are used for finding the neighboring pixel:

 4-connected if they are adjacent horizontally and vertically.

 8-connected if they are adjacent horizontally, vertically and diagonally.

4-Connected VS 8-Connected



Boundary fill 4-connected Algorithm

```
void Boundary_fill4(int x, int y, int b_color, int fill_color) {
int value = getpixel (x, y);
if (value! =b_color && value!=fill_color) {
        putpixel (x, y, fill_color);
        Boundary_fill4 (x-1, y, b_color, fill_color);
        Boundary_fill4 (x+1, y, b_color, fill_color);
        Boundary_fill4 (x, y-1, b_color, fill_color);
        Boundary_fill4 (x, y+1, b_color, fill_color);
}
}
```

Boundary fill 8-connected Algorithm

```
void Boundary-fill8(int x,int y,int b_color, int fill_color) {
int current = getpixel (x, y);
if (current !=b_color && current!=fill_color) {
putpixel (x,y,fill_color);
Boundary_fill8(x-1, y, b_color,fill_color);
Boundary_fill8(x+1, y, b_color, fill_color);
Boundary_fill8(x, y-1, b_color, fill_color);
Boundary_fill8(x, y+1, b_color, fill_color);
Boundary_fill8(x-1, y-1, b_color,fill_color);
Boundary_fill8(x-1, y+1, b_color,fill_color);
Boundary_fill8(x+1, y-1, b_color,fill_color);
Boundary_fill8(x+1, y+1, b_color,fill_color); }}
```

Flood Fill Algorithm

- This algorithm is used when boundary is of different color.

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

- Two mechanisms are used for finding the neighboring pixel:

 4-connected if they are adjacent horizontally and vertically.

 8-connected if they are adjacent horizontally, vertically and diagonally.

Algorithm (for 4-connected)

```
void flood_fill4(int x, int y, int fill_color, int old_color) {
int current = getpixel (x,y);
if (current==old_color) {
putpixel (x,y,fill_color);
flood_fill4(x-1, y, fill_color, old_color);
flood_fill4(x+1, y, fill_color, old_color);
flood_fill4(x, y-1, fill_color, old_color);
flood_fill4(x, y+1, fill_color, old_color);   }
}
//Similarly flood fill for 8 connected can be also defined.
```

Difference between Boundary and Flood fill Algorithm

Boundary fill Algorithm	Flood fill Algorithm
Area filling is started inside a point within 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.
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.

END OF UNIT 2