Scan Conversion Algorithm

Unit 2

This Chapter

- ✓ ***Scan Converting a Point and a straight Line***
  - ➢ DDA Line Algorithm
  - ➢ Bresenham’s Line Algorithm
- ✓ ***Scan Converting Circle and Ellipse***
  - ➢ Mid-Point Circle and
  - ➢ Ellipse Algorithm
- ✓ ***Area Filling***
  - ➢ Scan Line Polygon fill Algorithm,
  - ➢ Inside-outside Test,
  - ➢ Scan line fill of Curved Boundary area,
  - ➢ Boundary-fill and
  - ➢ Flood-fill algorithm

Points and Lines

- ✓ Point plotting is done by converting a single coordinate position furnished by an application program into appropriate operations for the output device in use.
- ✓ Line drawing is done by calculating intermediate positions along the line path between two specified endpoint positions.
- ✓ The output device is then directed to fill in those positions between the end points with some color.
- ✓ For some device such as a pen plotter or random scan display, a straight line can be drawn smoothly from one end point to other.
- ✓ Digital devices display a straight line segment by plotting discrete points between the two endpoints.
- ✓ Discrete coordinate positions along the line path are calculated from the equation of the line.
- ✓ For a raster video display, the line intensity is loaded in frame buffer at the corresponding pixel positions.

Points and Lines

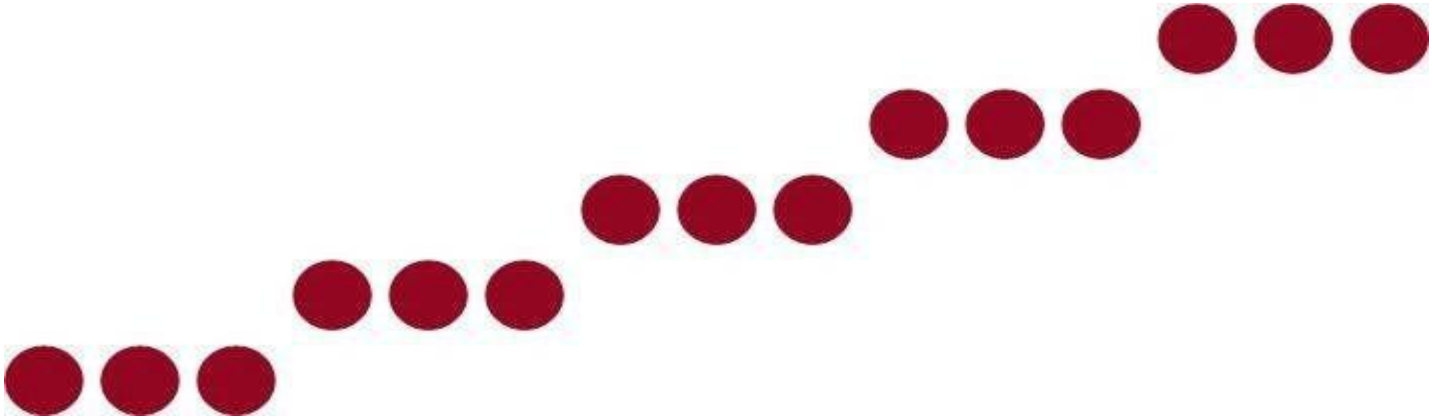


Figure: Stair step effect produced when line is generated as a series of pixel positions.

Points and Lines

- Pixel position will referenced according to scan-line number and column number which is illustrated by following figure.

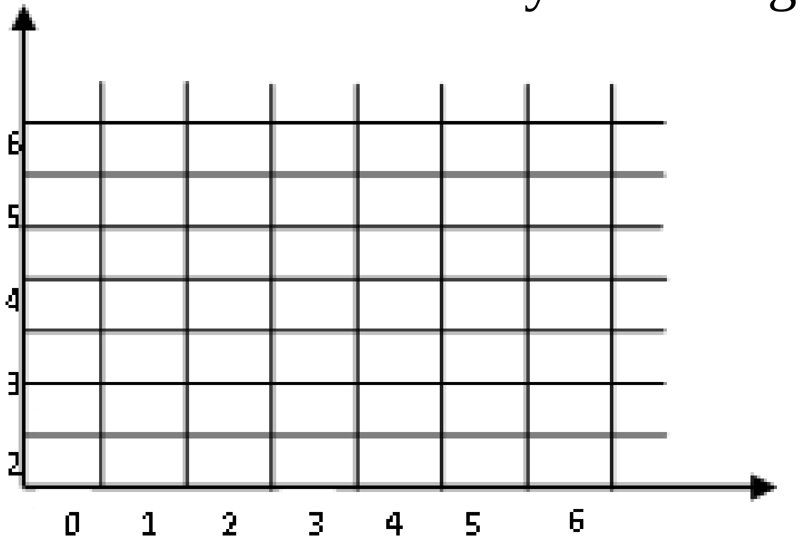


Figure: Pixel positions referenced by scan-line number and column number.

- To load the specified color into the frame buffer at a particular position, we will assume we have available low-level procedure of the form $setpixel(x, y)$.
- Similarly for retrieve the current frame buffer intensity we assume to have procedure $getpixel(x, y)$.

Line Drawing Algorithms

Line Equation

- ✓ The Cartesian slop-intercept equation for a straight line is

$$y = mx + b$$

where, m representing slop and b as the intercept.

- ✓ The two endpoints of the line are given which are say (x_1, y_1) and (x_2, y_2).

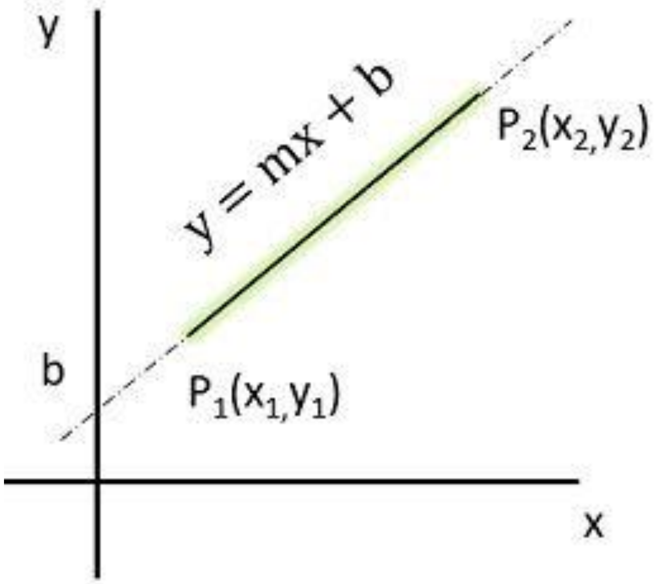


Figure: Line path between endpoint positions.

Line Drawing Algorithms

Line Equation

- ✓ We can determine values for the slope m by equation:

  $m = (y2 - y1)/(x2 - x1)$

- ✓ We can determine values for the intercept b by equation:

  $b = y1 - m * x1$

- ✓ For the given interval Δx along a line, we can compute the corresponding y interval Δy as:

  $\Delta y = m * \Delta x$

- ✓ Similarly for Δx:

  $\Delta x = \Delta y/m$

1. For line with slop $|m| < 1, \Delta x$ can be set proportional to small horizontal deflection voltage and the corresponding vertical deflection voltage is then set proportional to Δy which is calculated from above equation.
2. For line with slop $|m| > 1$, Δy can be set proportional to small vertical deflection voltage and the corresponding horizontal deflection voltage is then set proportional to Δx which is calculated from above equation.
3. For line with slop $m = 1$, $\Delta x = \Delta y$ and the horizontal and vertical deflection voltages are equal.

DDA Algorithm

- ✓ Digital differential analyzer (DDA) is scan conversion line drawing algorithm based on calculating either Δy or Δx using above equation.
- ✓ We sample the line at unit intervals in one coordinate and find corresponding integer values nearest the line path for the other coordinate.

DDA Algorithm

```
$DDA(x1, y1, x2, y2)$
{
    $dx = x2 - x1;$
    $dy = y2 - y1;$
    if( $abs(dx) > abs(dy)$ )
        $steps = abs(dx);$
    else
        $steps = abs(dy);$
    $xinc = dx/steps;$
    $yinc = dy/steps;$
    for( $i = 1; i <= steps; i++$ )
    {
        $putpixel(Round(x1), Round(y1));$
        $x1 = x1 + xinc;$
        $y1 = y1 + yinc;$
    }
}
```

DDA Algorithm Example

Q. Digitize a Line with end point $P1(2,3)$ and $P2(6,8)$, using DDA.

Solution:

Here, $dx = 6 - 2 = 4$ and $dy = 8 - 3 = 5$

steps = 5

$slop(m) = dy/dx = 5/4 = 1.25$

Here, slop is greater than 1

$xinc = dx/steps = 4/5 = 0.8 = 1/m$ and
$yinc = dy/steps = 5/5 = 1$

So, we use

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

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

DDA Algorithm Example

$\boldsymbol{k}$	$\boldsymbol{x_{k+1} = x_k + 1/m}$	$\boldsymbol{y_{k+1} = y_k + 1}$	$\boldsymbol{(x, y)}$
1	$= 2\ +\ 0.8\ =\ 2.8 \sim 3$	4	(3,4)
2	$= 2.8\ +\ 0.8 =\ 3.6 \sim 4$	5	(4,5)
3	$= 3.6\ +\ 0.8\ =\ 4.4 \sim 4$	6	(4,6)
4	$=\ 4.4\ +\ 0.8\ =\ 5.2 \sim 5$	7	(5,7)
5	$=\ 5.2\ +\ 0.8\ =\ 6$	8	(6,8)

DDA Algorithm Example

1. Digitize a Line with end point $B(2,3)$ and $A(7,8)$, using DDA
2. Digitize a Line with end point $A(2,6)$ and $B(4,2)$, using DDA
3. Digitize the line with end points (1,2) and (5,6) using digital differential analyzer method.(TU).

Advantages and Disadvantages of DDA

Advantages of DDA algorithm

- It is simple algorithm.

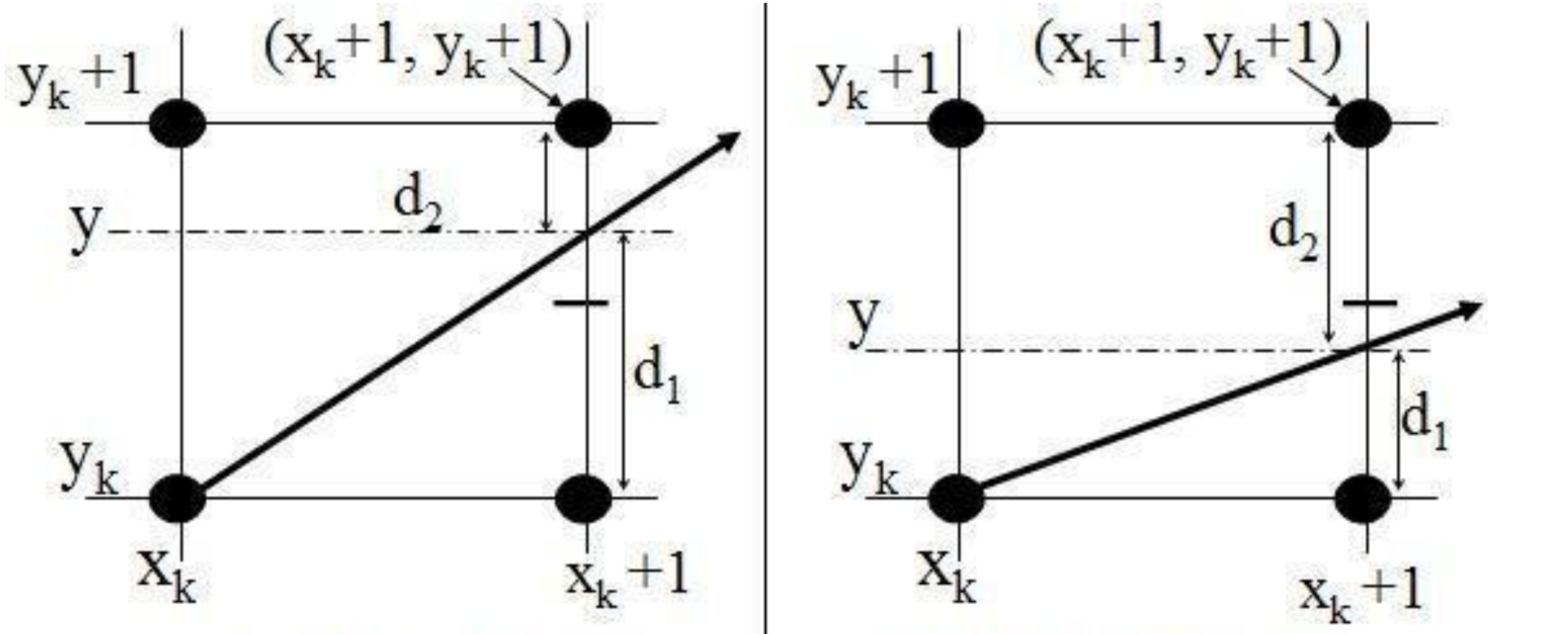
Disadvantage of DDA algorithm

- Floating point arithmetic is time consuming.
- Poor end point accuracy.

Bresenham’s Line Drawing Algorithm

- ✓ 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| ≤ 1

BLA for slope +ve and |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 $(x_k + 1, yk + 1)$.

Thus, the common equation of the line is

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

Now,

$$d1 = y - y_k = m\,(x_k + 1) + c - y_k \quad \text{and}$$

$$d2 = (y_k + 1) - y = yk + 1 – [m\,(x_k + 1) + c\,]$$

The difference bet^{n} these two separation is,

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

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

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

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

We have,

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

Define Decision parameter at kth step,

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

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

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

Now , k+1th term

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

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

Here,

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

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

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

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

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

Case 1

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

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

$y_{k+1} = y_k$

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

Case 2

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

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

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

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

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

Initial Decision Parameter (P_1)

We have,

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

$$P_1 = 2\,\Delta y\, x_1 - 2\,\Delta x\, y_1 + 2\,\Delta y + 2\,\Delta x\, c - \Delta x$$

After solving above equation. We get,

$$P_1 = 2\,\Delta y - \Delta x$$

Bresenham’s Line Drawing Algorithm

```
$Bresenham(x1, y1, x2, y2)$ {
    $x = x1;$
    $y = y1;$
    $dx = x2 - x1;$
    $dy = y2 - y1;$
    $p = 2dy - dx$
    $while(x <= x2)${
        $putpixel(x, y);$
        $x++;$
        $if(p < 0)$ {
            $p = p + 2dy;$
        }
        $else${
            $p = p + 2dy - 2dx$
            $y++;$
        }
    }
}
```

Bresenham’s Line Drawing Algorithm--Example

Example-1: Digitize the line with end points (20, 10) and (30, 18) using BLA.

Solution :

Here, Starting point of line = (x1, y1) = (20, 10) and Ending point of line = (x2, y2) = (30, 18)

Thus, slope of line, m = Δy / Δx = *y2-y1 / x2-x1*

= (18-10) / (30-20)

= 8/10

As the given points, it is clear that the line is moving left to right with the positive slop

$|m| = 0.8 < 1$

Bresenham’s Line Drawing Algorithm--Example

The initial decision parameter (P0) = 2Δy - Δx = 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,

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

$^{Y}k+1 \ ^{=} \ {}^{Y}k$

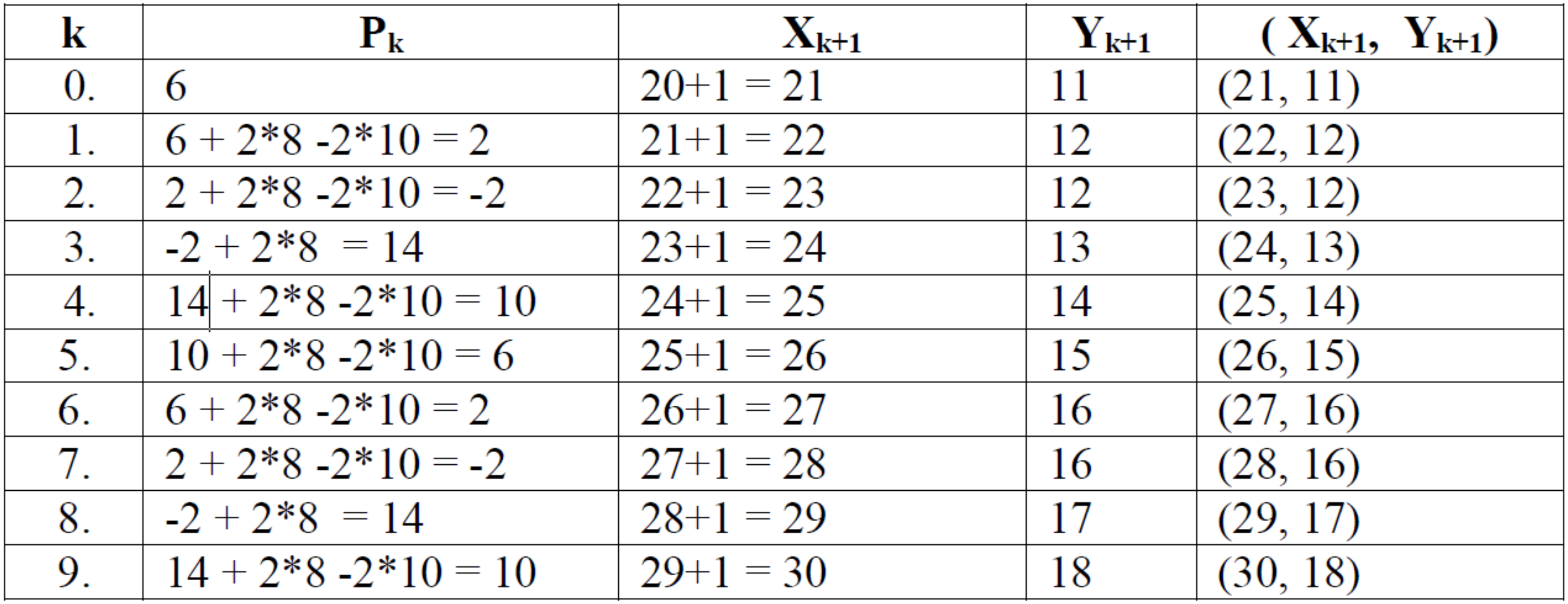
$P_k = P_k + 2\,\Delta y$

If $P_k \geq 0$ then,

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

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

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



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)

Bresenham’s Line Drawing Algorithm--Example

1. Digitize the line with end points (15, 5) and (30, 10) using BLA.
2. Digitize the line with end points (20, 5) and (25, 10) using BLA.

Circle

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

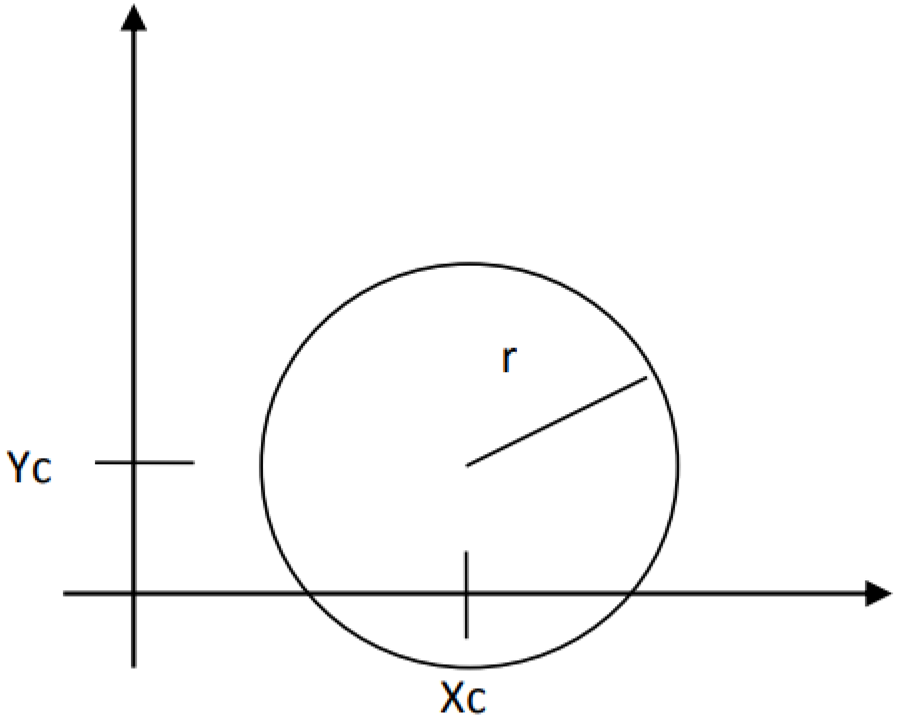


Figure: Circle with center coordinates (x_c, y_c) and radius r.

Some basics about circle

- ✓ The distance relationship is expressed by the Pythagorean theorem in Cartesian coordinates as [when center is (x_c, y_c)]

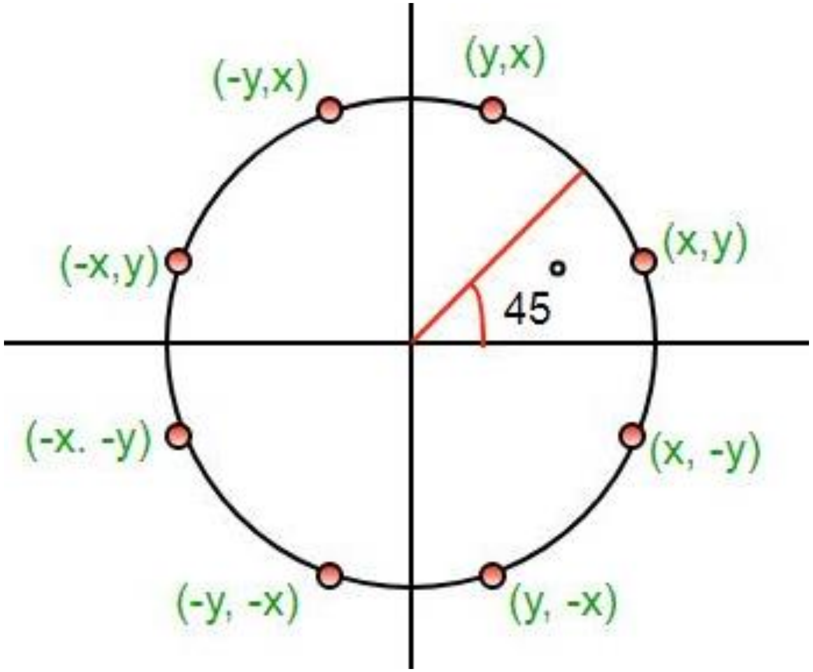
$$(x - x_c)^2 + (y - y_c)^2 = r^2$$

- ✓ The distance relationship is expressed by the Pythagorean theorem in Cartesian coordinates as [when center is origin i.e. (0,0)]

$$x^2 + y^2 = r^2$$

Symmetry Property of Circle

- ✓ This symmetry condition is shown in figure below where point (x, y) on one circle sector is mapped in other seven sector of circle.
- ✓ Taking advantage of this symmetry property of circle we can generate all pixel position on boundary of circle by calculating only one sector from $x = 0$ to $x = y$.



Mid-point Circle Drawing Algorithm--Basics

- ✓ Given radius ‘r’ and center (x_c, y_c)
- ✓ We first setup our algorithm to calculate circular path coordinates for center (0, 0). And then we will transfer calculated pixel position to center (x_c, y_c) by adding x_c to x and y_c to y.
- ✓ Along the circle section from $x = 0$ to $x = y$ in the first quadrant, the slope of the curve varies from 0 to -1 so we can step unit step in positive x direction over this octant and use a decision parameter to determine which of the two possible y position is closer to the circular path.
- ✓ Position in the other seven octants are then obtain by symmetry.

Mid-point Circle Drawing Algorithm--Basics

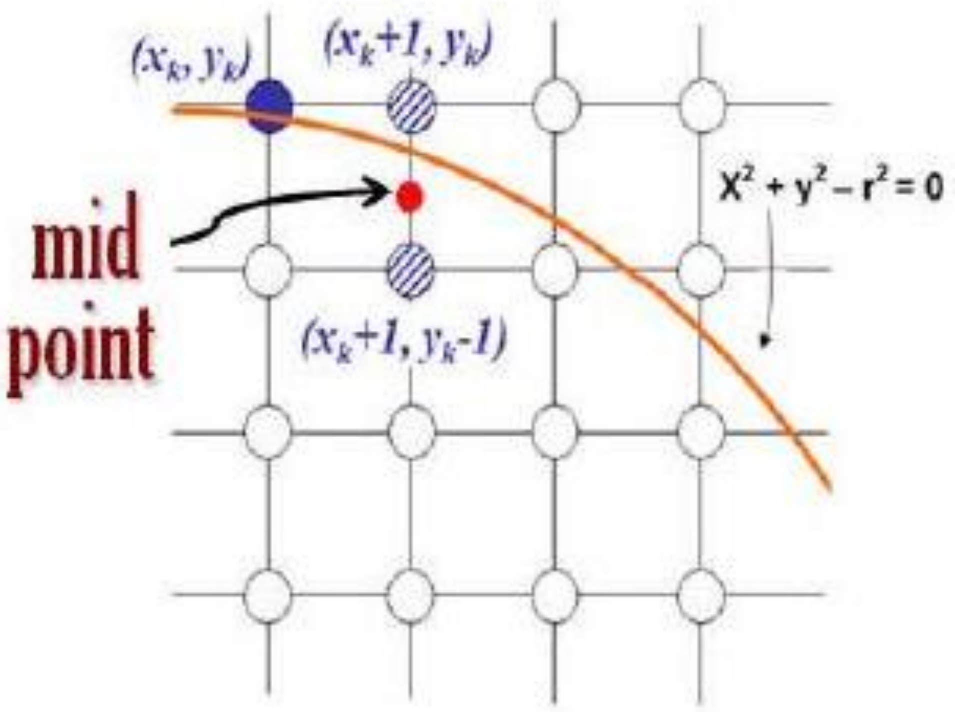


Figure: Midpoint between candidate pixel at sampling position $x_k + 1$ along circle path.

Mid-point Circle Drawing Algorithm--Basics

- ✓ For the decision parameter we use the circle function which is:

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

- ✓ Any point which is on the boundary is satisfied $f_{circle}(x, y) = 0$ if the point is inside circle function value is negative and if point is outside circle the function value is positive which can be summarize as below.

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

- ✓ Above equation we calculate for the mid positions between pixels near the circular path at each sampling step and we setup incremental calculation for this function as we did in the line algorithm.

Mid-point Circle Drawing Algorithm--Basics

- ✓ Assuming we have just plotted the pixel at (x_k, y_k) and next we need to determine whether the pixel at position $(x_k + 1, y_k)$ or the one at position $(x_k + 1,\ y_k - 1)$ is closer to circle boundary.
- ✓ So for finding which pixel is more closer using decision parameter evaluated at the midpoint between two candidate pixels as below:

$$p_k = f_{circle}\,(x_k + 1, y_k - \frac{1}{2})$$

$$p_k = (x_k + 1)^2 + (y_k\ - \frac{1}{2})^2 - r^2$$

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

Mid-point Circle Drawing Algorithm--Basics

Successive decision parameters are obtain using incremental calculations as follows:

$$p_{k+1} = f_{circle}\,(x_{k+1} + 1, y_{k+1} - \tfrac{1}{2})$$

$$p_{k+1} = (x_{k+1} + 1)^2 + (y_{k+1} - \tfrac{1}{2})^2 - r^2$$

Now,

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

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

✓ In above equation y_{k+1} is either y_k or $y_k - 1$ depending on the sign of the p_k.

Mid-point Circle Drawing Algorithm--Basics

Now we can put $2x_{k+1} = 2x_k + 2$ and when we select $y_{k+1} = y_k - 1$ we can obtain $2y_{k+1} = 2y_k - 2$.

***Case 1*:** $\boldsymbol{p_k < 0}$

$$\boxed{p_{k+1} = p_k + 2x_{k+1} + 1}$$

***Case 2*:** $\boldsymbol{p_k > 0}$

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

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

$$p_0 = fcircle\,(0 + 1, r - \tfrac{1}{2})$$

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

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

Mid-point Circle Drawing Algorithm

1. Input radius r and circle center (x_c, y_c), and obtain 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:

$$p_0 = 1 - r$$

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

- If $p_k < 0$, the next point along the circle centered on (0, 0) is $(x_k + 1, y_k)$ &

$$p_{k+1} = p \;\; + 2x_{k+1} + 1$$

- Otherwise, the next point along the circle is $(x_k + 1, y_k - 1)$ &

$$p_{k+1} = p_k + 2x_{k+1} + 1 - 2y_{k+1}$$

- Where $2x_{k+1} = 2x_k + 2$, & $2y_{k+1} = 2y_k - 2$.

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

$$x \;\; = x + x_c, \qquad y = y + y_c$$

6. Repeat steps 3 through 5 until $x \geq y$.

Tracing of Mid-point Circle Drawing Algorithm

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

Here, the initial decision parameter (P0)

$P0 = 1 - r = 1 - 9 = -8$

Since, for the Midpoint Circle Algorithm of starting point (0, r) & centre 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 $(x_k + 1, y_k - 1)$

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

Tracing of Mid-point Circle Drawing Algorithm

k	P_k	Xk+1	Yk+1	(Xk+1, Yk+1) At (0, 0)	(Xk+1, Yk+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, 8+7) = (11,15)
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)

Mid-point Circle Drawing Algorithm—Class Work

1. Digitize a circle with radius 10 at center (0,0)

Ellipse

- ✓ AN ellipse is defined as the set of points such that the sum of the distances from two fixed positions (**foci**) is same for all points.

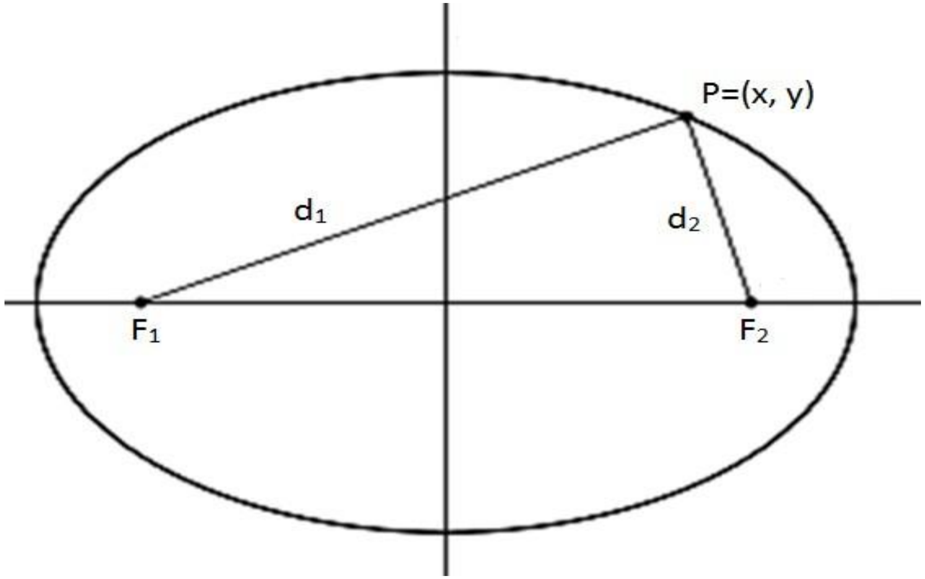


Figure: Ellipse generated about foci f1 and f2.

Properties of Ellipse

1. $d_1 + d_2 = Constant$
2. Expressing distance in terms of focal coordinates $f_1 = (x_1, y_1)$ and $f_2 = (x_2, y_2)$ we have

$$\sqrt{(x - x_1)^2 + (y - y_1)^2} + \sqrt{(x - x_2)^2 + (y - y_2)^2} = Constant$$

3. We can also write this equation in the form

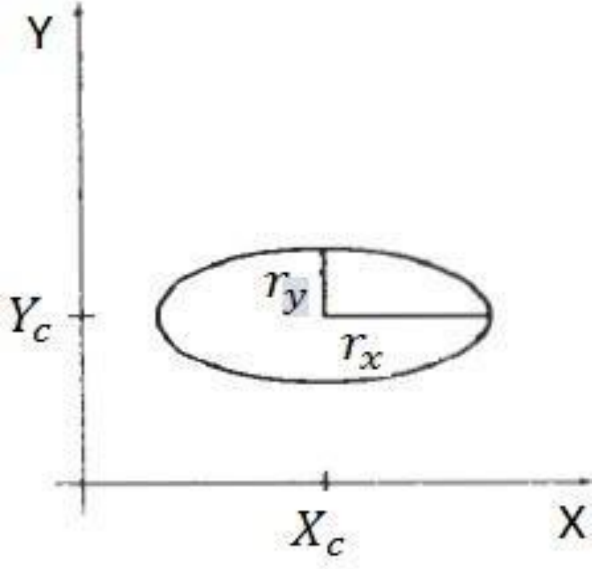
   $Ax^2 + By^2 + Cxy + Dx + Ey + F = 0$

4. $\left(\frac{x - x_c}{r_x}\right)^2 + \left(\frac{y - y_c}{r_y}\right)^2 = 1$

5. Using the polar coordinates r and θ,

$$x = x_c + r_x \cos\theta$$

$$y = y_c + r_y \sin\theta$$



Symmetry of an Ellipse

- ✓ Symmetry considerations can be used to further reduced computations.

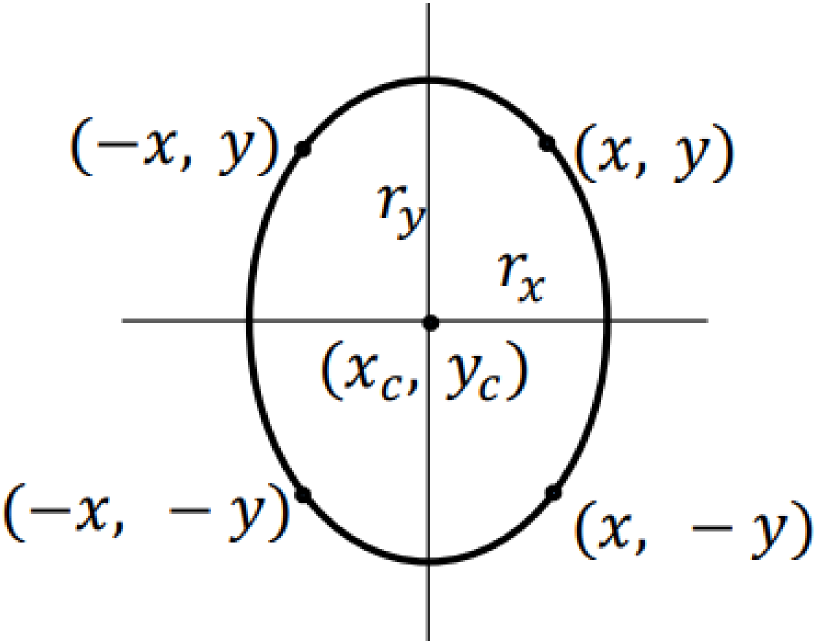
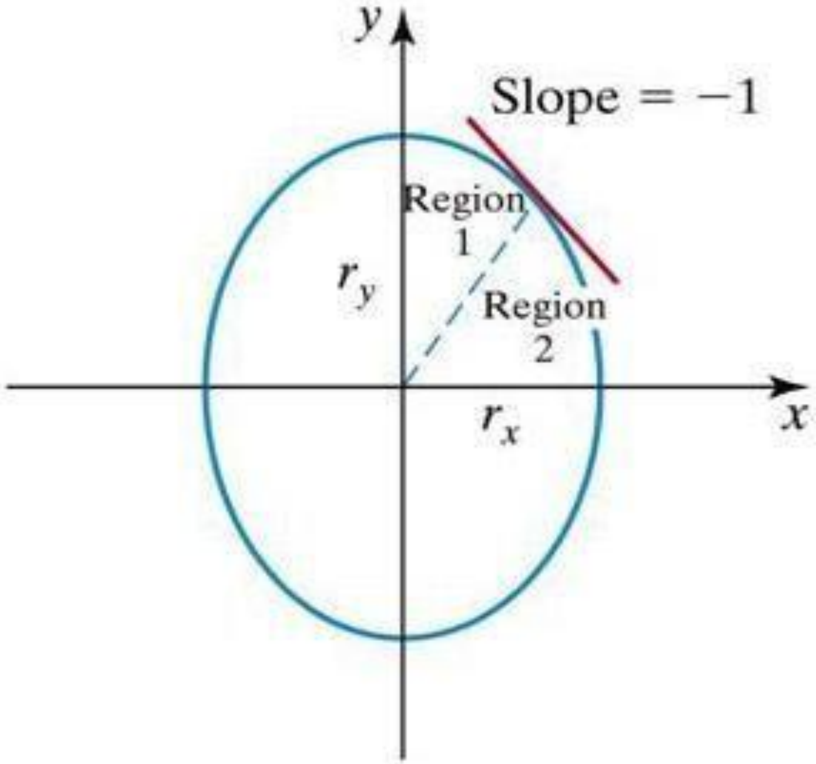


Figure: Symmetry of an ellipse.

Midpoint Ellipse Algorithm

- ✓ Given parameters r_x, r_y and (x_c, y_c) we determine points (x, y) for an ellipse in standard position centered on the origin, and then we shift the points so the ellipse is centered at (x_c, y_c).
- ✓ In this method we divide first quadrant into two parts according to the slope of an ellipse as shown in figure below.



Midpoint Ellipse Algorithm

- ✓ We take unit step in x direction if magnitude of slope is less than 1 in that region otherwise we take unit step in y direction.
- ✓ Boundary divides region at $slope = -1$.
- ✓ With $r_x < r_y$ we process this quadrant by taking unit steps in x direction in region 1 and unit steps in y direction in region 2.
- ✓ We take the start position at $(0, r_y)$ and steps along the elliptical path in clockwise order through the first quadrant.
- ✓ We define ellipse function for center of ellipse at (0, 0) as follows.

$$f_{ellipse}(x, y) = r_y^2 x^2 + r_x^2 y^2 - r_x^2 r_y^2$$

Midpoint Ellipse Algorithm

- ✓ Which has the following properties:

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

- ✓ Starting at (0, r_y) we take unit step in x direction until we reach the boundary between region 1 and 2 then we switch to unit steps in y direction in remaining portion on ellipse in first quadrant.
- ✓ At each step we need to test the value of the slope of the curve for deciding the end point of the region-1.
- ✓ The ellipse slope is calculated using following equation.

$$\frac{dy}{dx} = -\frac{2r_y^2 x}{2r_x^2 y}$$

Midpoint Ellipse Algorithm

✓ At boundary between region 1 and 2 $slope = -1$ and equation become.

$$2{r_y}^2x = 2{r_x}^2y$$

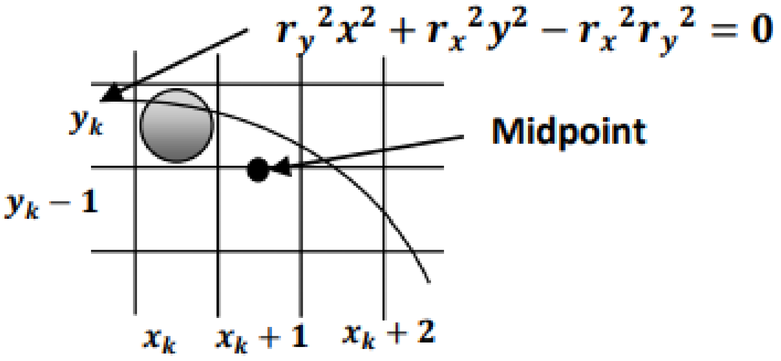
✓ Therefore we move out of region 1 whenever following equation is false

$$2{r_y}^2x \geq 2{r_x}^2y$$

✓ Assume we are at $({}_k, y_k)$ position and we determine the next position along the ellipse path by evaluating decision parameter at midpoint between two candidate pixels.

$$p1_k = f_{ellipse}\left(x_k + 1, y_k - \frac{1}{2}\right)$$

$$= r_y^2(x_k+1)^2 + r_x^2(y_k - \frac{1}{2})^2 - r_x^2 r_y^2$$



Midpoint Ellipse Algorithm

- If $p1_k < 0$, the midpoint is inside the ellipse and the pixel on scan line y_k is closer to ellipse boundary otherwise the midpoint is outside or on the ellipse boundary and we select the pixel $y_k - 1$.
- At the next sampling position decision parameter for region 1 is evaluated as.

$$p1_{k+1} = f_{ellipse}\left(x_{k+1} + 1, y_{k+1} - \frac{1}{2}\right)$$

$$p1_{k+1} = r_y{}^2[(x_k + 1) + 1]^2 + r_x{}^2\left(y_{k+1} - \frac{1}{2}\right)^2 - r_x{}^2 r_y{}^2$$

- Now subtract $p1_k$ from $p1_{k+1}$ and rearrange

$$p1_{k+1} = p1_k + 2r_y{}^2(x_k + 1) + r_y{}^2 + r_x{}^2\left[\left(y_{k+1} - \frac{1}{2}\right)^2 - \left(y_k - \frac{1}{2}\right)^2\right]$$

- Here y_{k+1} is either y_k or $y_k - 1$, depends on the sign of $p1_k$

Midpoint Ellipse Algorithm

- ✓ Decision parameters are incremented by the following amounts:

$$increment = \begin{cases} 2r_y^2 x_{k+1} + r_y^2, & if\ p1_k < 0 \\ 2r_y^2 x_{k+1} + r_y^2 - 2r_x^2 y_{k+1}, & if\ p1_k \geq 0 \end{cases}$$

- ✓ Now we calculate the initial decision parameter $p1_0$ by putting (x_0, y_0) = $(0, r_y)$ as follow.

$$p1_0 = f_{ellipse}\left(0 + 1, r_y - \frac{1}{2}\right)$$

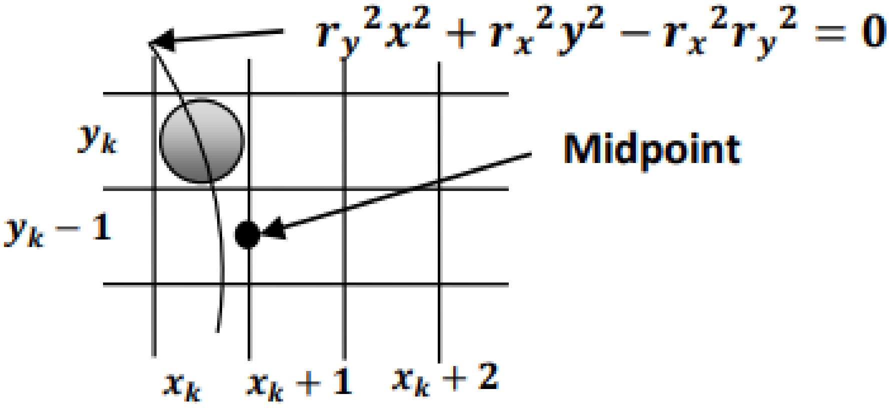
$$p1_0 = r_y{}^2(1)^2 + r_x{}^2\left(r_y - \frac{1}{2}\right)^2 - r_x{}^2 r_y{}^2$$

$$p1_0 = r_y{}^2 + r_x{}^2\left(r_y - \frac{1}{2}\right)^2 - r_x{}^2 r_y{}^2$$

$$p1_0 = r_y{}^2 - r_x{}^2 r_y + \frac{1}{4} r_x{}^2$$

Midpoint Ellipse Algorithm

- ✓ Now we similarly calculate over region 2 by unit stepping in negative *y* direction and the midpoint is now taken between horizontal pixels at each step as shown in figure below.



- ✓ For this region, the decision parameter is evaluated as follows.

$$p2_k = f_{ellipse}\left(x_k + \frac{1}{2}, y_k - 1\right)$$

$$p2_k = r_y^2\left(x_k + \frac{1}{2}\right)^2 + r_x^2(y_k - 1)^2 - r_x^2r_y^2$$

Midpoint Ellipse Algorithm

- If $p2_k > 0$ the midpoint is outside the ellipse boundary, and we select the pixel at x_k.
- If $p2_k \leq 0$ the midpoint is inside or on the ellipse boundary and we select $x_k + 1$.
- At the next sampling position decision parameter for region 2 is evaluated as.

$$p2_{k+1} = f_{ellipse}\left(x_{k+1} + \frac{1}{2}, y_{k+1} - 1\right)$$

$$p2_{k+1} = r_y{}^2\left(x_{k+1} + \frac{1}{2}\right)^2 + r_x{}^2[(y_k - 1) - 1]^2 - r_x{}^2 r_y{}^2$$

- Now subtract $p2_k$ from $p2_{k+1}$ and rearrange

$$p2_{k+1} = p2_k - 2r_x{}^2(y_k - 1) + r_x{}^2 + r_y{}^2\left[\left(x_{k+1} + \frac{1}{2}\right)^2 - \left(x_k + \frac{1}{2}\right)^2\right]$$

- Here x_{k+1} is either x_k or $x_k + 1$, depends on the sign of $p2_k$.

Midpoint Ellipse Algorithm

- ✓ In region 2 initial position is selected which is last position of region one and the initial decision parameter is calculated as follows.

$$p2_0 = f_{ellipse}\left(x_0 + \frac{1}{2}, y_0 - 1\right)$$

$$p2_0 = r_y{}^2\left(x_0 + \frac{1}{2}\right)^2 + r_x{}^2(y_0 - 1)^2 - r_x{}^2 r_y{}^2$$

Midpoint Ellipse Algorithm

1. Input r_x, r_y and ellipse center (x_c, y_c), and obtain the first point on an ellipse centered on the origin as $(x_0, y_0) = (0, r_y)$
2. Calculate the initial value of the decision parameter in region 1 as:

$$p1_0 = r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^2$$

3. At each x_k in region 1, starting from $k = 0$ until $2{r_y}^2 x \geq 2{r_x}^2 y$, test $p1_k$:
   - If $p1_k < 0$ then $plot(x_{k+1}, y_k)$ and $p1_{k+1} = p1_k + 2r_y^2 x_{k+1} + r_y^2$
   - Else $plot(x_k + 1, y_k - 1)$ and $p1_{k+1} = p1_k + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_y^2$
4. Initial value for decision parameter in region 2 using last calculated point say (x_0, y_0) in region 1 as:

$$p2_1 = r_y^2 \left(x_0 + \frac{1}{2}\right)^2 + r_x^2 (y_0 - 1)^2 - r_x^2 r_y$$

Midpoint Ellipse Algorithm

5. At each y_k, in region 2, starting at k = 0, until y=0, test $p2_k$:
   - If $p2_k$ > 0, $plot(x_k, yk - 1)$ and $p2_{k+1} = p2_k - 2r_x^2 y_{k+1} + r_x^2$
   - Else $plot(x_k$ + 1, y_k – 1) and $p2_{k+1} = p2_k + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_x^2$
6. Determine symmetry points in the other three quadrants.
7. Move each calculated pixel position (x, y) onto the elliptical path centered on (x_c, y_c) and plot the coordinate values:

$$x = x + x_c$$

$$y = y + y_c$$

Midpoint Ellipse Algorithm--Example

Example: Given input ellipse parameter r_x=8 and r_y=6, determine pixel positions along the ellipse path in the first quadrant using the midpoint ellipse algorithm

Here,

r_x = 8 and r_y = 6

Centre (x_c, y_c) = (0, 0)

Set x= 0 and y = 6; plot (0, 6)

Reason 1:

$$p1_0 = {r_y}^2 - {r_x}^2 r_y + \tfrac{1}{4} {r_x}^2$$

$$= 6^2 - 8^2.6 + \tfrac{1}{4} 8^2 = 36 - 384 + 16 = -332$$

Midpoint Ellipse Algorithm--Example

Successive midpoint decision parameter values and the pixel positions along the ellipse are listed in the following table:

k	$p1_k$	(x_{k+1},y_{k+1})	$2r_y^2\, x_{k+1}$	$2\, r_x^2 y_{k+1}$	Is $2r_y^2\, x_{k+1} \geq 2\, r_x^2 y_{k+1}$?	Or,
0	-332	(1,6)	72	768	False	= -332 +72+36 =-224
1	-224	(2,6)	144	768	False	= -224 +144 + 36 = -44
2	-44	(3,6)	216	768	False	= -44 + 216 + 36 = 208
3	208	(4,5)	288	640	False	= 208 + 288 -640+36= -108
4	-108	(5,5)	360	640	False	= -108 +360 +36 = 288
5	288	(6,4)	432	512	False	= 288 + 432 -512 +36 = 244
6	244	(7,3)	504	384	True	

Midpoint Ellipse Algorithm--Example

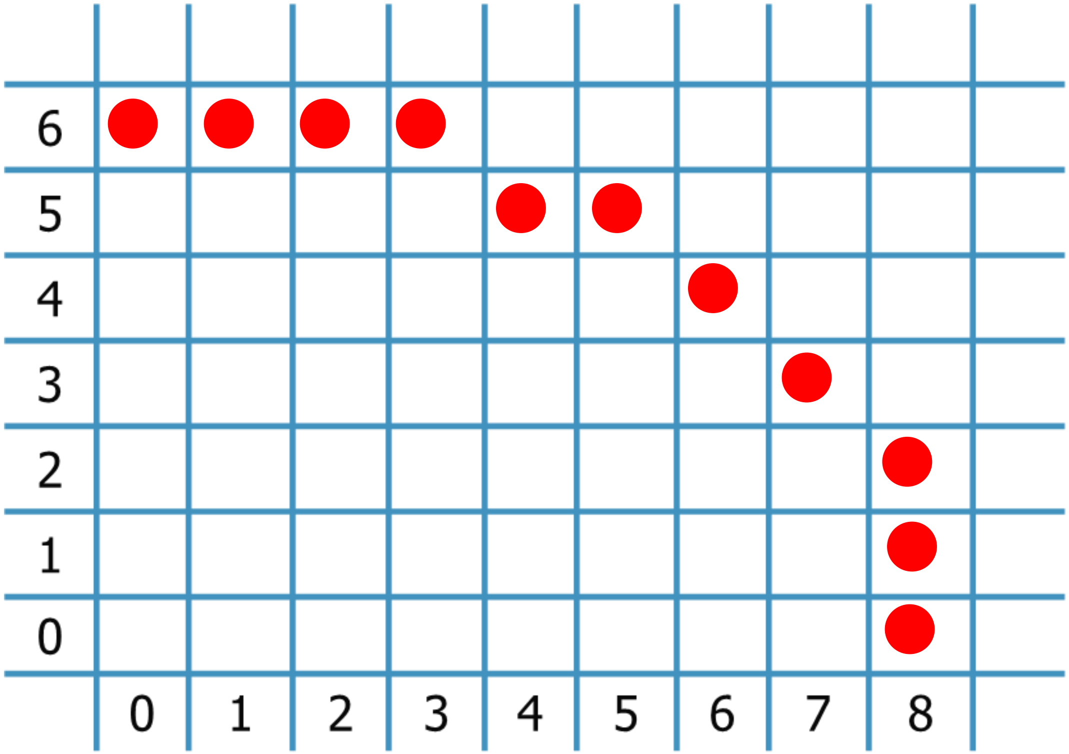
Reason 2: The initial point is $(x_0, y_0) = (7, 3)$

$$p2_0 = r_y{}^2\left(x_0 + \tfrac{1}{2}\right)^2 + r_x{}^2\left(y_0 - 1\right)^2 - r_x{}^2 r_y{}^2$$
$$= 6^2\left(7 + \tfrac{1}{2}\right)^2 + 8^2\left(3 - 1\right)^2 - 8^2 6^2 = -151$$

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

k	$p2_k$	(x_{k+1},y_{k+1})	Is $y_{k+1} = 0$?	$2r_y{}^2\, x_{k+1}$	$2\, r_x{}^2 y_{k+1}$	Or,
0	-151	(8, 2)	False	576	256	= -151 +576 – 256 + 64 = 233
1	233	(8, 1)	False	576	128	= 233 - 128 + 64 = 169
2	169	(8, 0)	True			

Midpoint Ellipse Algorithm--Example



Midpoint Ellipse Algorithm—Class Work

1. Rasterizing the ellipse in first quadrant with parameters $r_x = 7$, $r_y = 5$ and center = (100, 100) with midpoint ellipse generation algorithm.
2. Calculate the points to draw a ellipse using mid-point algorithm
   a) having radius $r_x = 7$, $r_y = 5$ and center at (0, 0)
   b) having radius $r_x = 10$, $r_y = 6$ and center at (0, 0)
   c) having radius $r_x = 6$, $r_y = 4$ and center at (2, 3)

Filled-area Primitives

- For filling polygons with particular colors, we need to determine the pixels falling on the border of the polygon and those which fall inside the polygon.
- There are two techniques used in area filling:
    - Scan-line Fill
    - Seed Fill
        - Boundary Fill
        - Flood Fill

Scan-Line Polygon Fill Algorithm

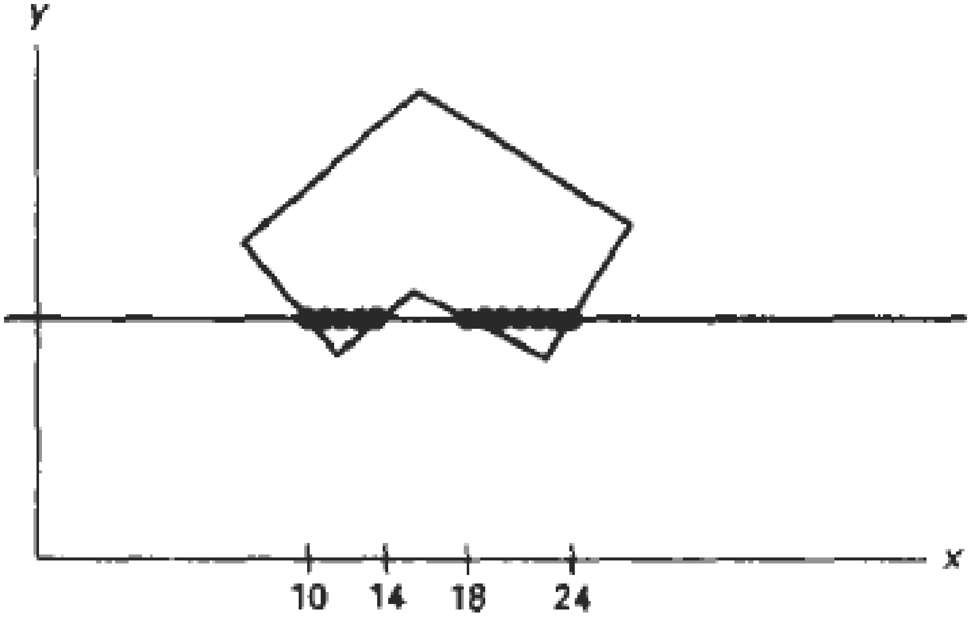


Figure: Interior pixels along a scan line passing through a polygon area.

- ✓ For each scan-line crossing a polygon, the algorithm locates the intersection points are of scan line with the polygon edges.
- ✓ This intersection points are stored from left to right.
- ✓ Frame buffer positions between each pair of intersection point are set to specified fill color.
- ✓ Some scan line intersects at vertex position they are required special handling.

Scan-Line Polygon Fill Algorithm

- ✓ For vertex we must look at the other endpoints of the two line segments of the polygon which meet at this vertex.
  - ➢ If these points lie on the same (up or down) side of the scan line, then that point is counts as two intersection points.
  - ➢ If they lie on opposite sides of the scan line, then the point is counted as single intersection.

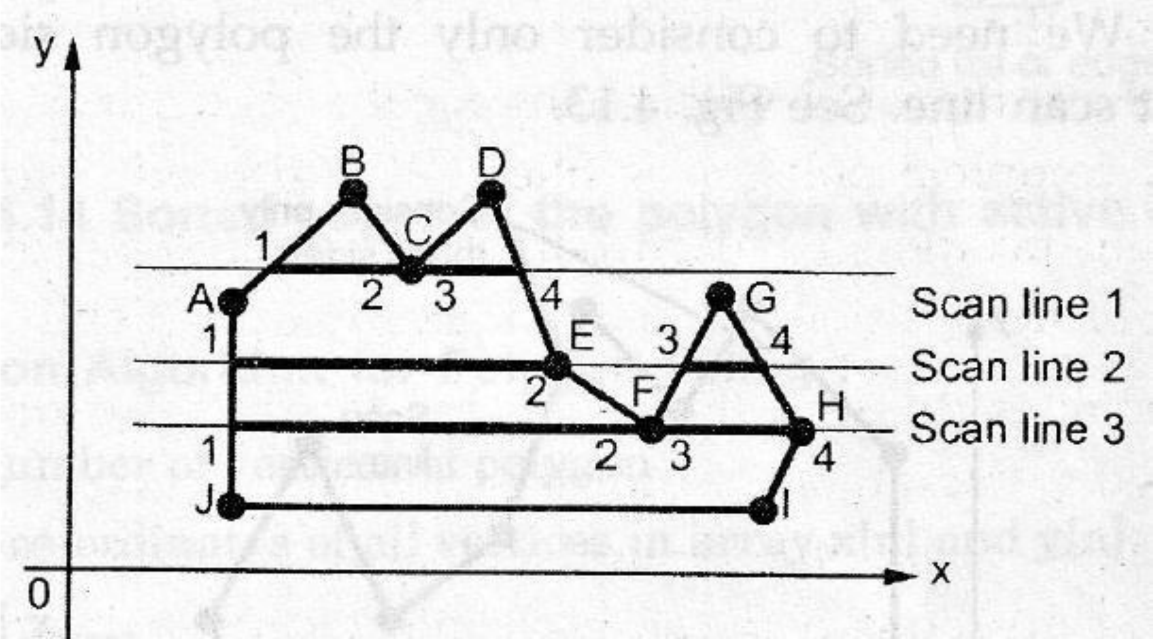


Figure: Intersection points along the scan line that intersect polygon vertices.

Scan-Line Polygon Fill Algorithm

- ✓ Vertices that require additional processing are those that have connecting edges on opposite sides of the scan line. We can identify these vertices by tracing around the polygon boundary either in clockwise or counter clockwise order.
  - ➢ One way to resolve the question as to whether we should count a vertex as one intersection or two is by splitting those vertices that should be counted as one intersection.

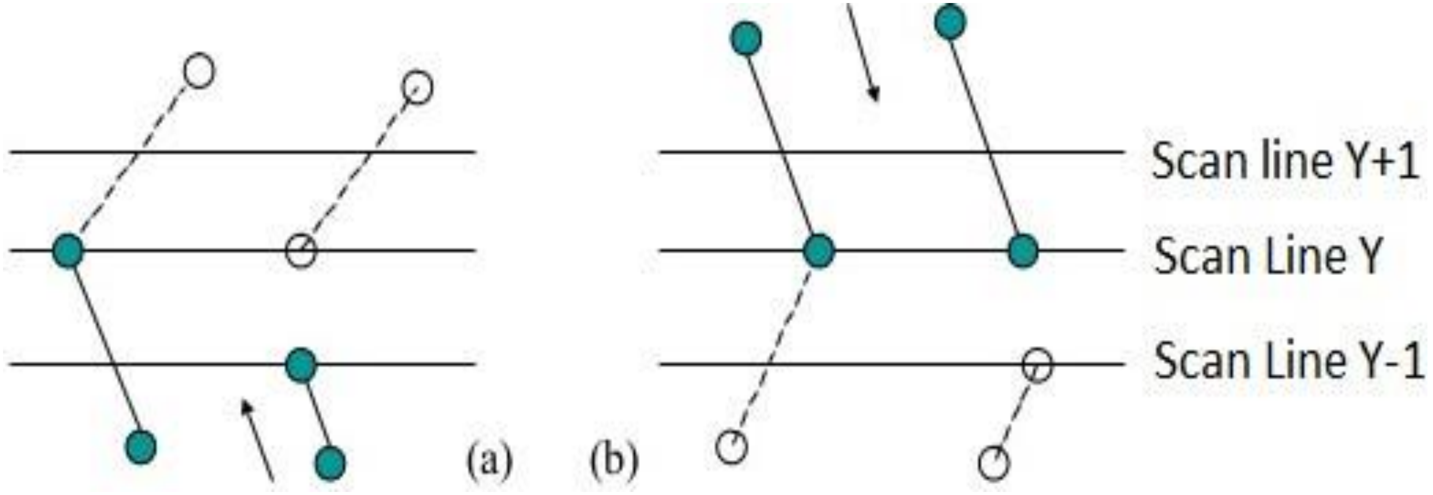


Figure: Adjacent scan line intersects with polygon edge.

Inside-Outside Tests

- ✓ In area filling and other graphics operation often required to find particular point is inside or outside the polygon.
- ✓ For finding which region is inside or which region is outside most graphics package use either *odd even rule* or *the nonzero winding number rule.*

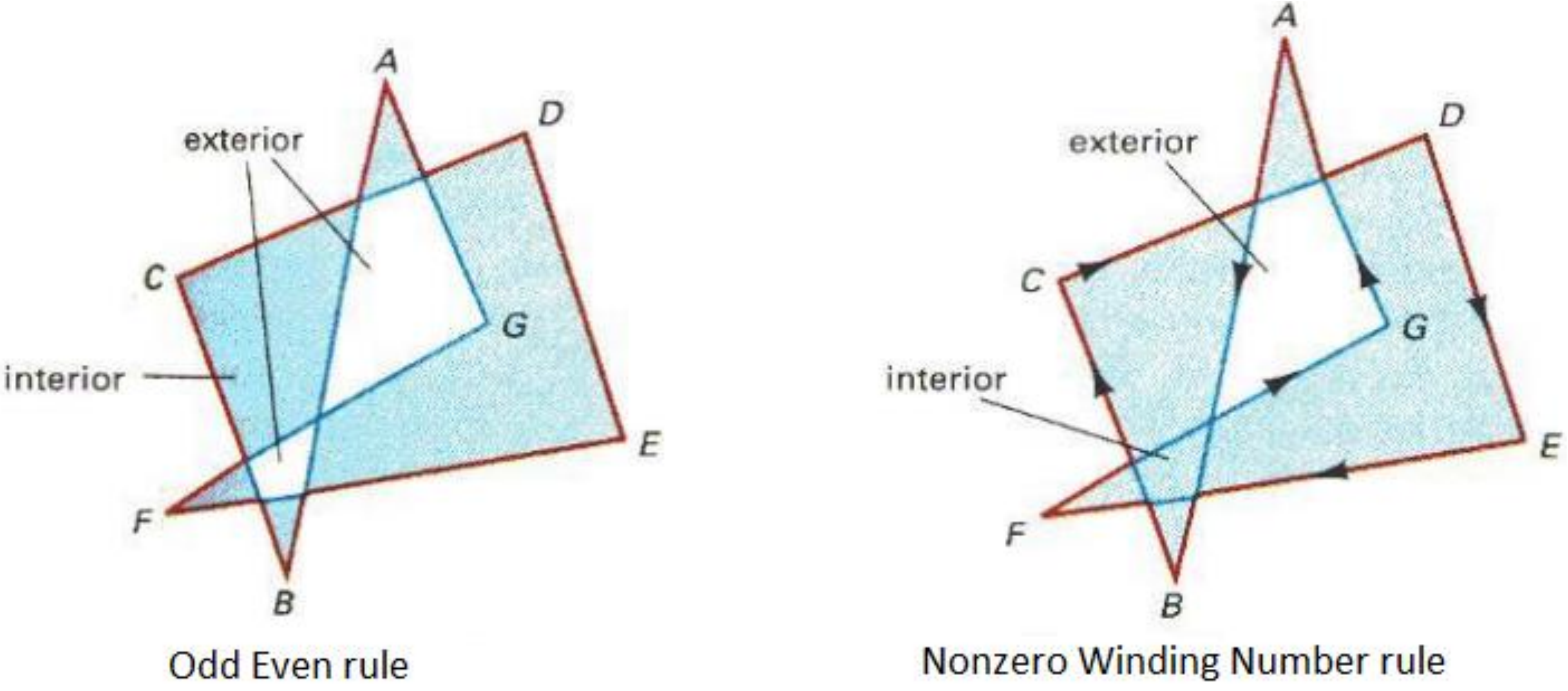
Odd Even Rule

- ➢ By conceptually drawing a line from any position *p* to a distant point outside the coordinate extents of the object and counting the number of edges crossing by this line is odd, than *p* is an interior point. Otherwise *p* is exterior point.

Nonzero Winding Number Rule

- ➢ This method counts the number of times the polygon edges wind around a particular point in the counterclockwise direction. This count is called the winding number, and the interior points of a two- dimensional object are defined to be those that have a nonzero value for the winding number.

Inside-Outside Tests



Odd Even rule

Nonzero Winding Number rule

Figure: Identifying interior and exterior region for a self-intersecting polygon.

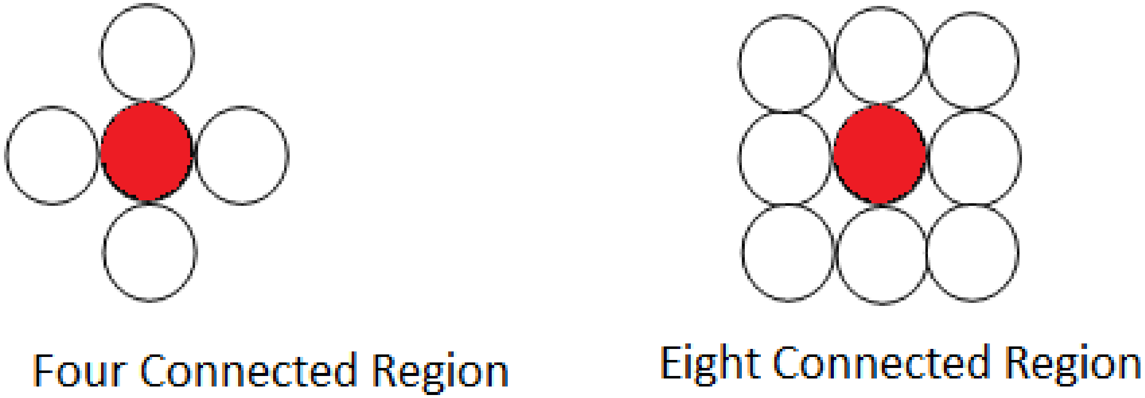
Note: For standard polygons and simple object both rule gives same result but for more complicated shape both rule gives different result

Scan-Line Fill of Curved Boundary Areas

- ✓ Scan-line fill of region with curved boundary is more time consuming as intersection calculation now involves nonlinear boundaries.
- ✓ For simple curve such as circle or ellipse scan line fill process is straight forward process.
- ✓ We calculate the two scan line intersection on opposite side of the curve.
- ✓ This is same as generating pixel position along the curve boundary using standard equation of curve.
- ✓ Then we fill the color between two boundary intersections.
- ✓ Symmetry property is used to reduce the calculation.
- ✓ Similar method can be used for fill the curve section.

Boundary Fill Algorithm/ Edge Fill Algorithm

- This algorithm uses the recursive method.
- First of all, a starting pixel called as the seed is considered.
- The algorithm checks boundary pixel or adjacent pixels are colored or not.
- If the adjacent pixel is already filled or colored then leave it, otherwise fill it.
- The filling is done using four connected or eight connected approaches.



Four Connected Region

Eight Connected Region

Figure: Neighbor pixel connected to one pixel.

Boundary Fill Algorithm/ Edge Fill Algorithm

```
boundary-fill4(x, y, f-color, b-color)
{
        if(getpixel (x,y) ! = b-color && gepixel (x, y) ! = f-color)
        {
                putpixel (x, y, f-color)
                boundary-fill4(x + 1, y, f-color, b-color);
                boundary-fill4(x, y + 1, f-color, b-color);
                boundary-fill4(x - 1, y, f-color, b-color);
                boundary-fill4(x, y - l, f-color, b-color);
        }
}
```

Flood-Fill Algorithm

- ✓ In this method, a point or seed which is inside region is selected.
- ✓ This point is called a seed point.
- ✓ Then four connected approaches or eight connected approaches is used to fill with specified color.
- ✓ 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.

Flood-Fill Algorithm

```
flood-fill4(x, y, new-color, old-color)
{
        if(getpixel (x,y) = = old-color)
        {
                putpixel (x, y, new-color)
                flood-fill4 (x + 1, y, new-color, old -color);
                flood-fill4 (x, y + 1, new -color, old -color);
                flood-fill4 (x - 1, y, new -color, old -color);
                flood-fill4 (x, y - l, new -color, old-color);
        }
}
```

Assignment #2

1. Digitize a Line with end point $B(2,3)$ and $A(7,8)$, using DDA
2. Digitize a Line with end point $A(2,6)$ and $B(4,2)$, using DDA
3. Digitize the line with end points (1,2) and (5,6) using digital differential analyzer method.(TU).
4. Digitize the line with end points (15, 5) and (30, 10) using BLA.
5. Digitize the line with end points (20, 5) and (25, 10) using BLA.
6. Digitize a circle with radius 10 at center (0,0)
7. Rasterizing the ellipse in first quadrant with parameters $r_x = 7$, $r_y = 5$ and center = (100, 100) with midpoint ellipse generation algorithm.
8. Calculate the points to draw a ellipse using mid-point algorithm
   a) having radius $r_x = 7$, $r_y = 5$ and center at (0, 0)
   b) having radius $r_x = 10$, $r_y = 6$ and center at (0, 0)
   c) having radius $r_x = 6$, $r_y = 4$ and center at (2, 3)