# Unit : 2

# Point

- Point plotting is done by converting a single coordinate position furnished by an application program into appropriate operations for the output device in use.
- Example: Plot point $P\ (3, 2)$
  - Line $x = 3$
  - Line $y = 2$
  - Point $P\ (3, 2)$
- To draw the point on the screen we use function
  - ✓ $setpixel\ (x, y)$
- To draw the pixel in C language we use function
  - ✓ $putpixel\ (x, y, color)$
- Similarly for retrieving color of pixel we have function
  - ✓ $getpixel\ (x, y)$

# Line

- 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.

# Line Drawing Algorithms

- The Cartesian slop-intercept equation for a straight line is
  - $y = mx + b$
  - with '$m$' representing slop and '$b$' as the intercept.
- It is possible to draw line using this equation but for efficiency purpose we use different line drawing algorithm.
  - DDA Algorithm
  - Bresenham's Line Algorithm
- We can also use this algorithm in parallel if we have more number of processors.

# Introduction to DDA Algorithm

- Full form of DDA is Digital Differential Analyzer
- DDA is scan conversion line drawing algorithm based on calculating either $\Delta y$ or $\Delta x$ using line equation.
- We sample the line at unit intervals in one coordinate and find corresponding integer values nearest the line path for the other coordinate.
- Selecting unit interval in either $x$ or $y$ direction based on way we process line.

# Derivation of DDA Algorithm

- We sample at unit $x$ interval ($\Delta x = 1$) and calculate each successive $y$ value as follow:
- $y = mx + b$
- $y_1 = m(x + 1) + b$ [*For first intermediate point*]
- $y_k = m(x + k) + b$ [*For $k^{th}$ intermediate point*]
- $y_{k+1} = m(x + k + 1) + b$ [*For $k + 1^{th}$ intermediate point*]
- Subtract $y_k$ from $y_{k+1}$
- $y_{k+1} - y_k = m(x + k + 1) + b - m(x + k) - b$
- $y_{k+1} = y_k + m$
- Using this equation computation becomes faster than normal line equation.
- As $\boldsymbol{m}$ is any real value calculated $\boldsymbol{y}$ value must be rounded to nearest integer.

# Contd.

- We sample at unit $y$ interval ($\Delta y = 1$) and calculate each successive $x$ value as follow:
- $x = (y - b)/m$
- $x_1 = ((y+1) - b)/m$ [*For first intermediate point*]
- $x_k = ((y+\mathrm{k}) - b)/m$ [*For $k^{th}$ intermediate point*]
- $x_{k+1} = ((y+\mathrm{k}+1) - b)/m$ [*For $k + 1^{th}$ intermediate point*]

Subtract $x_k$ from $x_{k+1}$

- $x_{k+1} - x_k = \{((y+\mathrm{k}+1) - b)/m\} - \{((y+\mathrm{k}) - b)/m\}$
- $x_{k+1} = x_k + 1/m$

Similarly

- for Δx = -1: we obtain $y_{k+1} = y_k - m$
- for Δy = -1: we obtain $x_{k+1} = x_k - 1/m$

**Procedure for DDA line algorithm.**

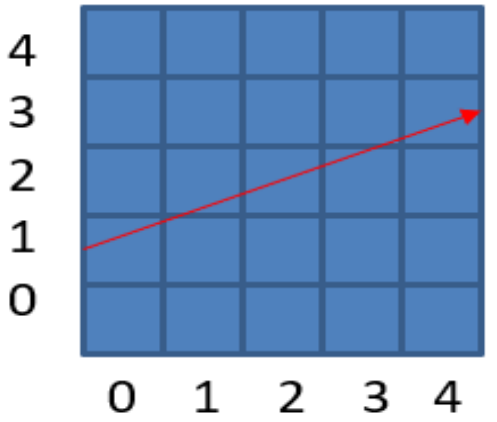
```
Void lineDDA (int xa, int ya, int xb, int yb)
{
        int dx = xb – xa, dy = yb – ya, steps, k;
        float xincrement, yincrement, x = xa, y = ya;
        if (abs(dx)>abs(dy))
        {
                Steps = abs (dx);
        }
        else
        {
                Steps = abs (dy);
        }
        xincrement = dx/(float) steps;
        yincrement = dy/(float) steps;

        setpixel (ROUND (x), ROUND (y));
        for(k=0;k<steps;k++)
        {
                x += xincrement;
                y += yincrement;
                setpixel (ROUND (x), ROUND (y));
        }
}
```

# Introduction to Bresenham's Line Algorithm

- An accurate and efficient raster line-generating algorithm, developed by Bresenham.
- It scan converts line using only incremental integer calculations.
- That can be modified to display circles and other curves.
- Based on slop we take unit step in one direction and decide pixel of other direction from two candidate pixel.
- If $|\Delta x| > |\Delta y|$ we sample at unit $\boldsymbol{x}$ interval and vice versa.

# Line Path & Candidate pixel

- Example $|\Delta X| > |\Delta Y|$, and $\Delta Y$ is "+ve".
  - Initial point $(X_k, Y_k)$
  - Line path
  - Candidate pixels $\{(X_k+1, Y_k), (X_k+1, Y_k+1)\}$




- Now we need to decide which candidate pixel is more closer to actual line.
- For that we use decision parameter $(P_k)$ equation.
- Decision parameter can be derived by calculating distance of actual line from two candidate pixel.

# Derivation Bresenham's Line Algorithm

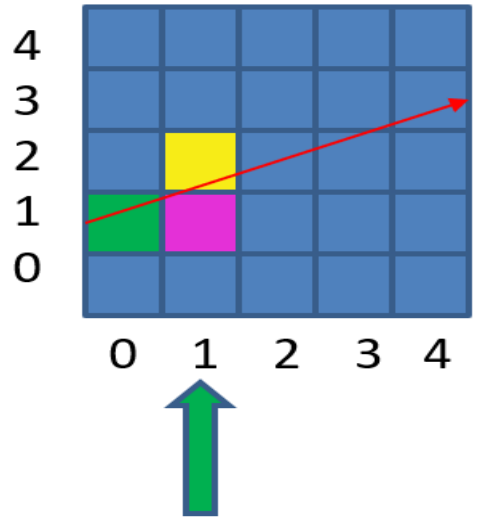
- $y = mx + b$ [*Line Equation*]
- $y = m(x_k) + b$ [*Actual Y value at $X_k$ position*]
- $y = m(x_k + 1) + b$ [*Actual Y value at $X_k$ +1position*]

Distance between actual line position and lower candidate pixel

- $d_1 = y - y_k$
- $d_1 = m(x_k + 1) + b - y_k$

Distance between actual line position and upper candidate pixel

- $d_2 = (y_k+1) - y$
- $d_2 = (y_k+1) - m(x_k + 1) - b$



# Contd.

Calculate $d_1 - d_2$

- $d_1 - d_2 = \{m(x_k + 1) + b - y_k\} - \{(y_k+1) - m(x_k + 1) - b\}$
- $d_1 - d_2 = \{mx_k + m + b - y_k\} - \{y_k + 1 - mx_k - m - b\}$
- $d_1 - d_2 = 2m(x_k + 1) - 2y_k + 2b - 1$
- $d_1 - d_2 = 2(\Delta y/\Delta x)(x_k + 1) - 2y_k + 2b - 1$ [*Put* $m = \Delta y/\Delta x$]

Decision parameter

- $p_k = \Delta x(d_1 - d_2)$
- $p_k = \Delta x \{2(\Delta y/\Delta x)(x_k + 1) - 2y_k + 2b - 1\}$
- $p_k = 2\Delta y x_k - 2\Delta x y_k + 2\Delta y + 2\Delta x b - \Delta x$
- $p_k = 2\Delta y x_k - 2\Delta x y_k + C$ [*Replacing single constant C for simplicity*]

Similarly

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

# Contd.

Subtract $p_k$ from $p_{k+1}$

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

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

  [where $(y_{k+1} - y_k) = 0$ or $1$ depending on selection of previous pixel]

# Initial Decision parameter

- The first decision parameter $p_0$ is calculated using equation of $p_k$.
- $p_k = 2\Delta y x_k - 2\Delta x y_k + 2\Delta y + 2\Delta x b - \Delta x$
- $p_0 = 2\Delta y x_0 - 2\Delta x y_0 + 2\Delta y + 2\Delta x b - \Delta x$ [*Put* $k = 0$]
- $p_0 = 2\Delta y x_0 - 2\Delta x y_0 + 2\Delta y + 2\Delta x(y_0 - m x_0) - \Delta x$

  [*Substitute* $b = y_0 - m x_0$]
- $p_0 = 2\Delta y x_0 - 2\Delta x y_0 + 2\Delta y + 2\Delta x(y_0 - (\Delta y/\Delta x) x_0) - \Delta x$

  [*Substitute* $m = \Delta y/\Delta x$]
- $p_0 = 2\Delta y x_0 - 2\Delta x y_0 + 2\Delta y + 2\Delta x y_0 - 2\Delta y x_0 - \Delta x$
- $p_0 = 2\Delta y - \Delta x$

  [*Initial decision parameter with all terms are constant*]

# Bresenham's Line Algorithm

1. Input the two line endpoints and store the left endpoint in $(x_0, y_0)$.
2. Load $(x_0, y_0)$ into the frame buffer; that is, plot the first point.
3. Calculate constants $\Delta x$, $\Delta y$, $2\Delta y$, and $2\Delta y - 2\Delta x$, and obtain the starting value for the decision parameter as

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

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

   If $p_k < 0$, the next point to plot is $(x_k + 1, y_k)$ and

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

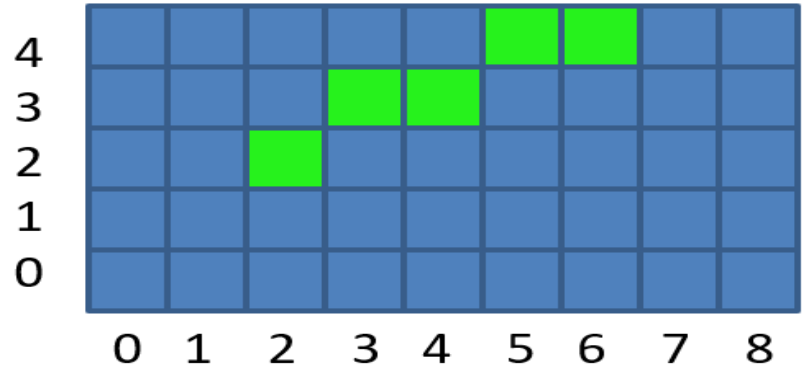
   Otherwise, the next point to plot is $(x_k + 1, y_k + 1)$ and

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

5. Repeat step-4 $\Delta x$ times.

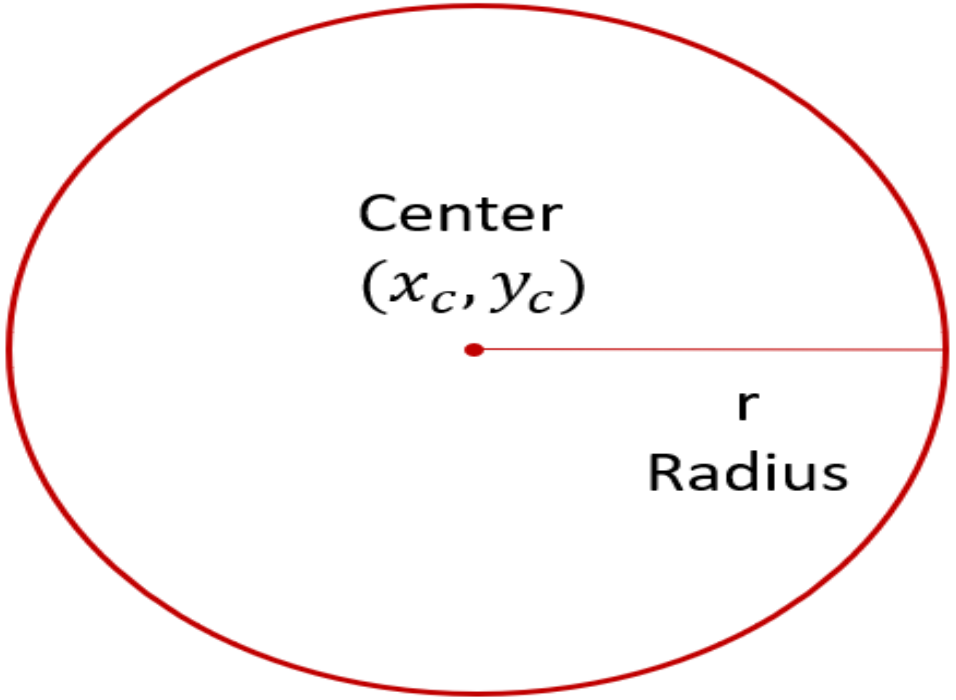
# Example Bresenham's Line Algorithm

- Example: Draw line $AB$ with coordinates $A(2,2)$, $B(6,4)$.
- Plot left end point $A(2,2)$.
- Calculate:
  - $\Delta x = 4,$
  - $\Delta y = 2,$
  - $2\Delta y = 4,$
  - $2\Delta y - 2\Delta x = -4,$
- $p_0 = 2\Delta y - \Delta x = 0$
- Now $p_0 \not< 0$ so we select upper pixel $(3,3)$.
- $p_1 = p_0 + 2\Delta y - 2\Delta x = 0 + (-4) = -4$
- Now $p_1 < 0$ so we select lower pixel $(4,3)$.
- $p_2 = p_1 + 2\Delta y = -4 + (4) = 0$
- Now $p_2 \not< 0$ so we select upper pixel $(5,4)$.
- $p_3 = p_2 + 2\Delta y - 2\Delta x = 0 + (-4) = -4$
- Now $p_3 < 0$ so we select lower pixel $(6,4)$.



# Circle

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

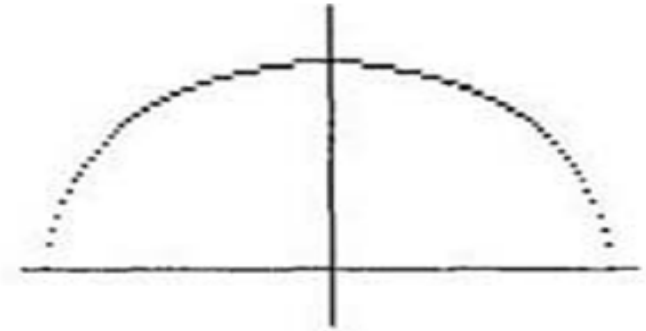


# Properties of Circle- Cartesion Coordinate

- Cartesian coordinates equation :
- $(x - x_c)^2 + (y - y_c)^2 = r^2$
- We could use this equation to calculate circular boundary points.
- We increment 1 in $x$ direction in every steps from $x_c - r$ to $x_c + r$ and calculate corresponding $y$ values at each position as:
- $(x - x_c)^2 + (y - y_c)^2 = r^2$
- $(y - y_c)^2 = r^2 - (x - x_c)^2$
- $(y - y_c) = \pm\sqrt{r^2 - (x_c - x)^2}$
- $y = y_c \pm \sqrt{r^2 - (x_c - x)^2}$

# Contd.

- But this is not best method as it requires more number of calculations which take more time to execute.
- And also spacing between the plotted pixel positions is not uniform.



- We can adjust spacing by stepping through y values and calculating x values whenever the absolute value of the slop of the circle is greater than 1.
- But it will increases computation time.

# Properties of Circle- Polar Coordinate

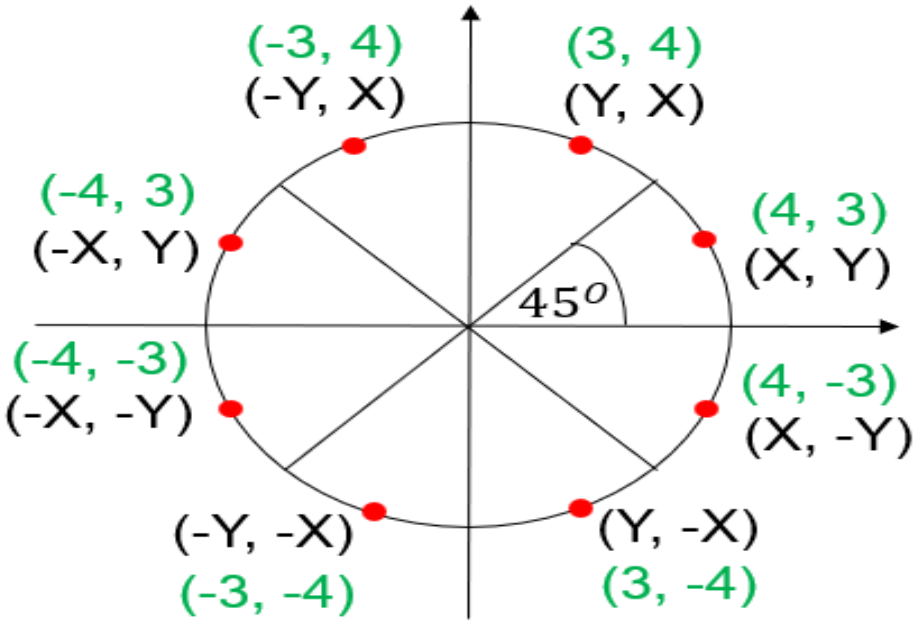
- Another way to eliminate the non-uniform spacing is to draw circle using polar coordinates ‘r’ and ‘θ’.
- Calculating circle boundary using polar equation is given by pair of equations which is as follows.

$$x = x_c + r\cos\theta$$
$$y = y_c + r\sin\theta$$

- When display is produce using these equations using fixed angular step size circle is plotted with uniform spacing.
- The step size ‘θ’ is chosen according to application and display device.
- For a more continuous boundary on a raster display we can set the step size at $\frac{1}{r}$.

# Properties of Circle- Symmetry

- Computation can be reduced by considering symmetry city property of circles.
- The shape of circle is similar in each octant.



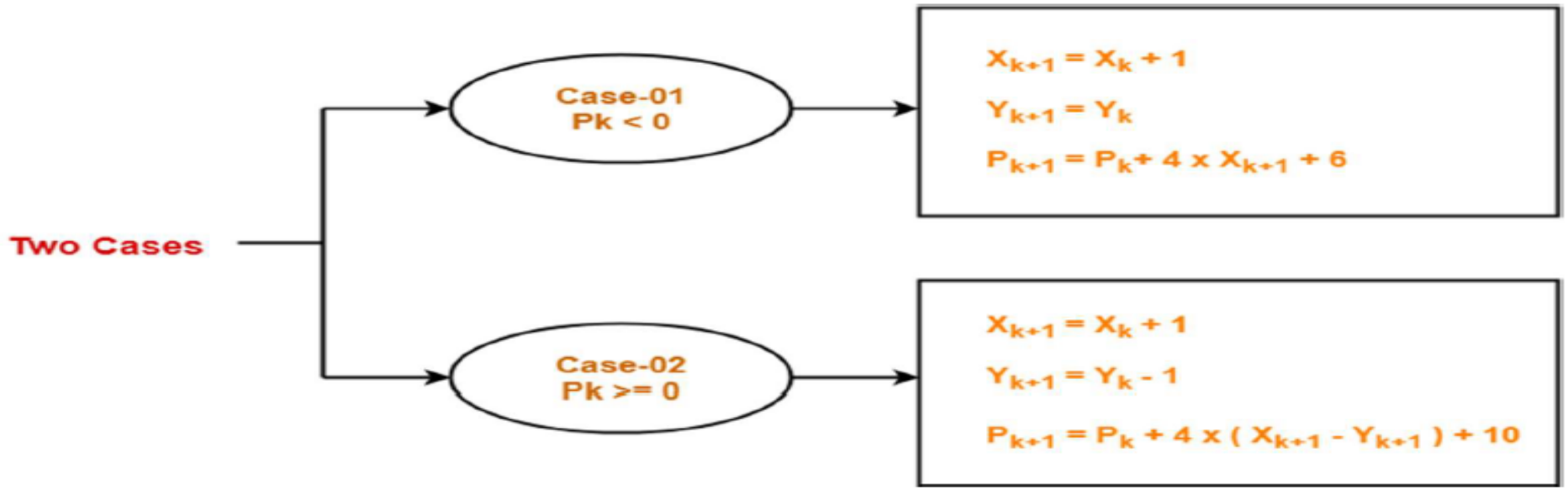
# Circle Algorithm

- 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 $x = y$.
- Determining pixel position along circumference of circle using any of two equations shown above still required large computation.
- More efficient circle algorithm are based on incremental calculation of decision parameters, as in the Bresenham line algorithm.
- Bresenham's line algorithm can be adapted to circle generation by setting decision parameter for finding closest pixel to the circumference at each sampling step.

# Contd.

- A method for direct distance comparison to test the midpoint between two pixels to determine if this midpoint is inside or outside the circle boundary.
- This method is easily applied to other conics also.
- Midpoint approach generates same pixel position as generated by bresenham's circle algorithm.
- The error involve in locating pixel positions along any conic section using midpoint test is limited to one-half the pixel separation.

# Bresenham's circle drawing algorithm:-



Initial conditions :

- x = 0
- y = r
- d = 3 - (2 * r)

**Steps:**

- **Step 1** : Set initial values of (xc, yc) and (x, y)
- **Step 2** : Calculate decision parameter d to d = 3 – (2 * r).
- **Step 3** : call displayBresenhmCircle(int xc, int yc, int x, int y) method to display initial(0,r) point.
- **Step 4** : Repeat steps 5 to 8 until x < = y
- **Step 5** : Increment value of x.

- **Step 6** : If d < 0, set d = d + (4*x) + 6
- **Step 7** : Else, set d = d + 4 * (x – y) + 10 and decrement y by 1.
- **Step 8** : call displayBresenhmCircle(int xc, int yc, int x, int y) method.
- **Step 9** : Exit.

# Introduction to Midpoint Circle Algorithm

- In this we sample at unit interval and determine the closest pixel position to the specified circle path at each step.
- 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.

# Decision Parameter Midpoint Circle Algorithm

- Position in the other seven octants are then obtain by symmetry.
- For the decision parameter we use the circle function which is:

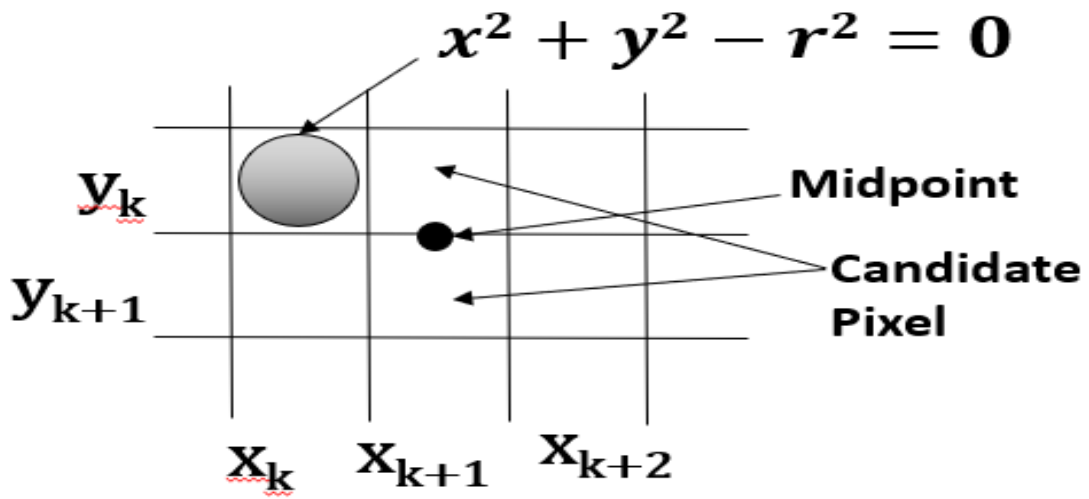
$$f_{circle}(x,y) = x^2 + y^2 - r^2$$

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

- 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.

# Midpoint between Candidate pixel

- Figure shows the midpoint between the two candidate pixels at sampling position $x_k + 1$.




- Assuming we have just plotted the pixel at $(x_k, y_k)$.
- Next we 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.

# Derivation Midpoint Circle Algorithm

- So for finding which pixel is more closer using decision parameter evaluated at the midpoint between two candidate pixels as below:
- $p_k = f_{circle}\left(x_k + 1, y_k - \frac{1}{2}\right)$
- $p_k = (x_k + 1)^2 + \left(y_k - \frac{1}{2}\right)^2 - r^2$
- If $p_k < 0$, midpoint is inside the circle and the pixel on the scan line $y_k$ is closer to circle boundary.
- Otherwise midpoint is outside or on the boundary and we select the scan line $y_k - 1$.

# Contd.

- Successive decision parameters are obtain using incremental calculations as follows:
- $p_{k+1} = f_{circle}(x_{k+1} + 1, y_{k+1} - \frac{1}{2})$
- $p_{k+1} = [(x_k + 1) + 1]^2 + (y_{k+1} - \frac{1}{2})^2 - r^2$
- Now we can obtain recursive calculation using equation of $p_{k+1}$ and $p_k$ as follow
- $p_{k+1} - p_k = \left([(x_k + 1) + 1]^2 + (y_{k+1} - \frac{1}{2})^2 - r^2\right) - \left((x_k + 1)^2 + (y_k - \frac{1}{2})^2 - r^2\right)$
- $p_{k+1} - p_k = (x_k + 1)^2 + 2(x_k + 1) + 1 + {y_{k+1}}^2 - y_{k+1} + \frac{1}{4} - r^2 - (x_k + 1)^2 - {y_k}^2 + y_k - \frac{1}{4} + r^2$

# Contd.

- $p_{k+1} - p_k = (x_k + 1)^2 + 2(x_k + 1) + 1 + {y_{k+1}}^2 - y_{k+1} + \frac{1}{4} - r^2 - (x_k + 1)^2 - {y_k}^2 + y_k - \frac{1}{4} + r^2$
- $p_{k+1} - p_k = 2(x_k + 1) + 1 + {y_{k+1}}^2 - y_{k+1} - {y_k}^2 + y_k$
- $p_{k+1} - p_k = 2(x_k + 1) + ({y_{k+1}}^2 - {y_k}^2) - (y_{k+1} - y_k) + 1$
- $p_{k+1} = p_k + 2(x_k + 1) + ({y_{k+1}}^2 - {y_k}^2) - (y_{k+1} - y_k) + 1$
- Now we can put $2x_{k+1} = 2(x_k + 1)$
- $p_{k+1} = p_k + 2x_{k+1} + ({y_{k+1}}^2 - {y_k}^2) - (y_{k+1} - y_k) + 1$

# Contd.

- $p_{k+1} = p_k + 2x_{k+1} + ({y_{k+1}}^2 - {y_k}^2) - (y_{k+1} - y_k) + 1$
- In above equation $y_{k+1}$ is either $y_k$ or $y_k - 1$ depending on the sign of the $p_k$.
- If we select $y_{k+1} = y_k$ .
- $p_{k+1} = p_k + 2x_{k+1} + 1$
- If we select $y_{k+1} = y_k - 1$.
- $p_{k+1} = p_k + 2x_{k+1} + ({y_{k+1}}^2 - {y_k}^2) - (y_{k+1} - y_k) + 1$
- $p_{k+1} = p_k + 2x_{k+1} + (y_{k+1} + y_k)(y_{k+1} - y_k) - (y_{k+1} - y_k) + 1$
- $p_{k+1} = p_k + 2x_{k+1} + (y_k - 1 + y_k)(y_k - 1 - y_k) - (y_k - 1 - y_k) + 1$
- $p_{k+1} = p_k + 2x_{k+1} + (2y_k - 1)(-1) - (-1) + 1$

# Contd.

- $p_{k+1} = p_k + 2x_{k+1} - 2y_k + 1 + 1 + 1$
- $p_{k+1} = p_k + 2x_{k+1} - (2y_k - 2) + 1$
- Now put $2y_{k+1} = 2y_k - 2.$
- $p_{k+1} = p_k + 2x_{k+1} - 2y_{k+1} + 1$

# IDP Midpoint Circle Algorithm

- The initial decision parameter(IDP) is obtained by evaluating the circle function at the start position $(x_0, y_0) = (0, r)$ as follows.
- $p_0 = f_{circle}\left(0 + 1, r - \frac{1}{2}\right)$
- $p_0 = 1^2 + \left(r - \frac{1}{2}\right)^2 - r^2$
- $p_0 = 1 + r^2 - r + \frac{1}{4} - r^2$
- $p_0 = \frac{5}{4} - r$
- $p_0 \approx 1 - r$

**Algorithm for Midpoint Circle Generation**

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
$$p_0 = \frac{5}{4} - r$$
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 $(x_k + 1, y_k)$ &
   $$p_{k+1} = p_k + 2x_{k+1} + 1$$
   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:
$$x = x + x_c, y = y + y_c$$
6. Repeat steps 3 through 5 until $x \geq y$.

# Example Midpoint Circle Algorithm

- Example: Draw circle with radius $r = 10$, and center of circle is $(1, 1)$ (Only one octant $x = 0$ to $x = y$ )
- First we find pixel position for octant $x = 0$ to $x = y$ for center $(0, 0)$
- $p_0 = 1 - r$
- $p_0 = 1 - 10 = -9$
- Now $p_0 < 0$ we select (1, 10)
- $p_1 = p_0 + 2x_1 + 1$
- $p_1 = -9 + 2 + 1 = -6$
- Now $p_1 < 0$ we select (2, 10)

| $k$ | $p_k$ | $(x_{k+1}, y_{k+1})$ |
|---|---|---|
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

# Contd.

- $p_2 = p_1 + 2x_2 + 1$
- $p_2 = -6 + 4 + 1 = -1$
- Now $p_2 < 0$ we select (3, 10)
- $p_3 = p_2 + 2x_3 + 1$
- $p_3 = -1 + 6 + 1 = 6$
- Now $p_3 \not< 0$ we select (4, 9)
- $p_4 = p_3 + 2x_4 + 1 - 2y_4$
- $p_4 = 6 + 8 + 1 - 18 = -3$
- Now $p_4 < 0$ we select (5, 9)

| $k$ | $p_k$ | $(x_{k+1}, y_{k+1})$ |
|---|---|---|
| 0 | -9 | (1, 10) |
| 1 | -6 | (2, 10) |
| | | |
| | | |
| | | |
| | | |
| | | |

# Contd.

- $p_5 = p_4 + 2x_5 + 1$
- $p_5 = -3 + 10 + 1 = 8$
- Now $p_5 \not< 0$ we select (6, 8)
- $p_6 = p_5 + 2x_6 + 1 - 2y_6$
- $p_6 = 8 + 12 + 1 - 16 = 5$
- Now $p_6 \not< 0$ we select (7, 7)
- Now Loop exit as $x \geq y$, as
- in our case $7 \geq 7$

| $k$ | $p_k$ | $(x_{k+1}, y_{k+1})$ |
|---|---|---|
| 0 | -9 | (1, 10) |
| 1 | -6 | (2, 10) |
| 2 | -1 | (3, 10) |
| 3 | 6 | (4, 9) |
| 4 | -3 | (5, 9) |
| | | |
| | | |

# Contd.

- Than we calculate pixel position for given center $(1, 1)$ using equations:
- $x = x + x_c = x + 1$
- $y = y + y_c = y + 1$

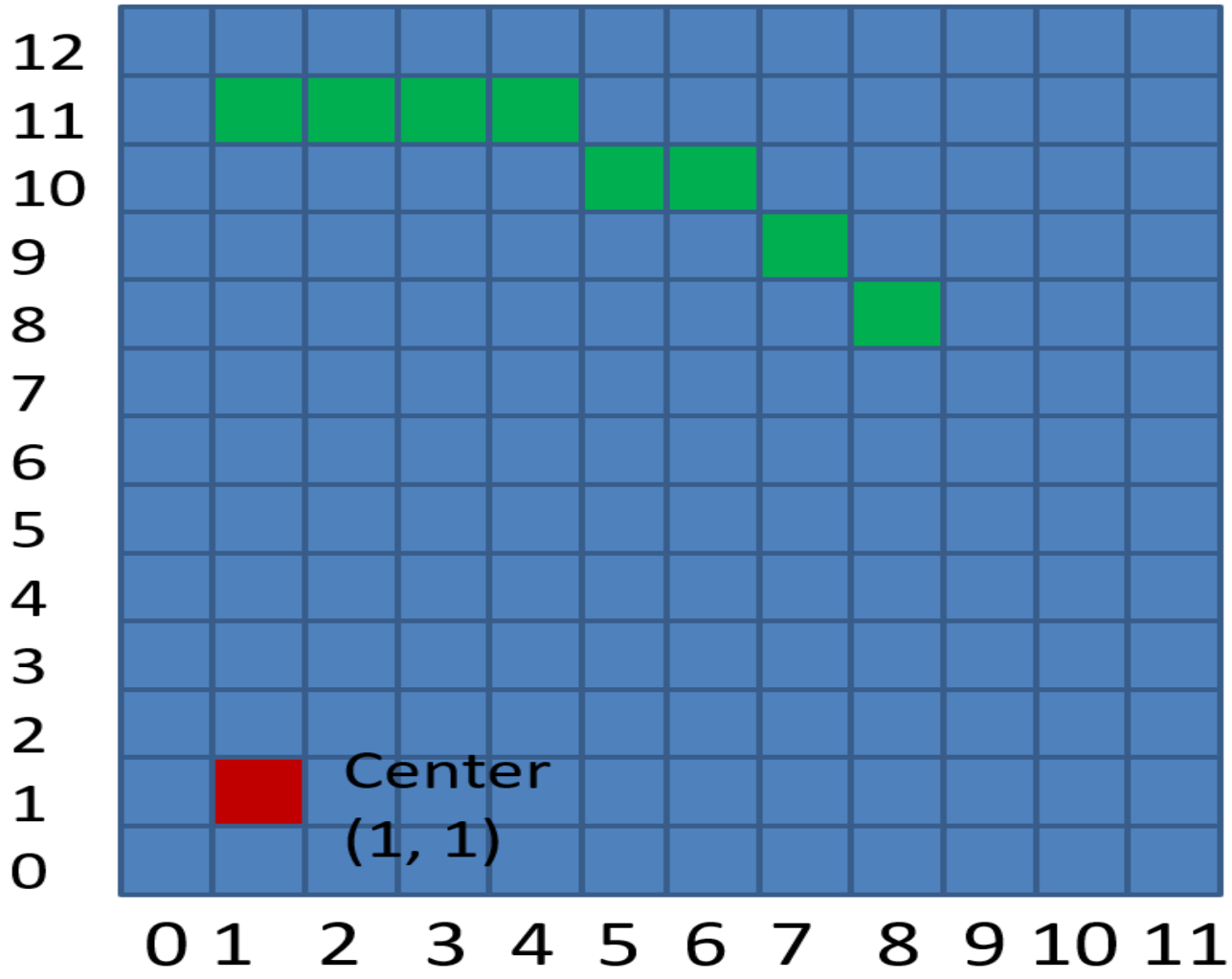
| Center (0, 0) | Center (1, 1) |
|---|---|
| (1, 10) | (2, 11) |
| (2, 10) | (3, 11) |
| (3, 10) | (4, 11) |
| (4, 9) | (5, 10) |
| (5, 9) | (6, 10) |
| (6, 8) | (7, 9) |
| (7, 7) | (8, 8) |

# Contd.

- Plot the pixel.
- First plot initial point.

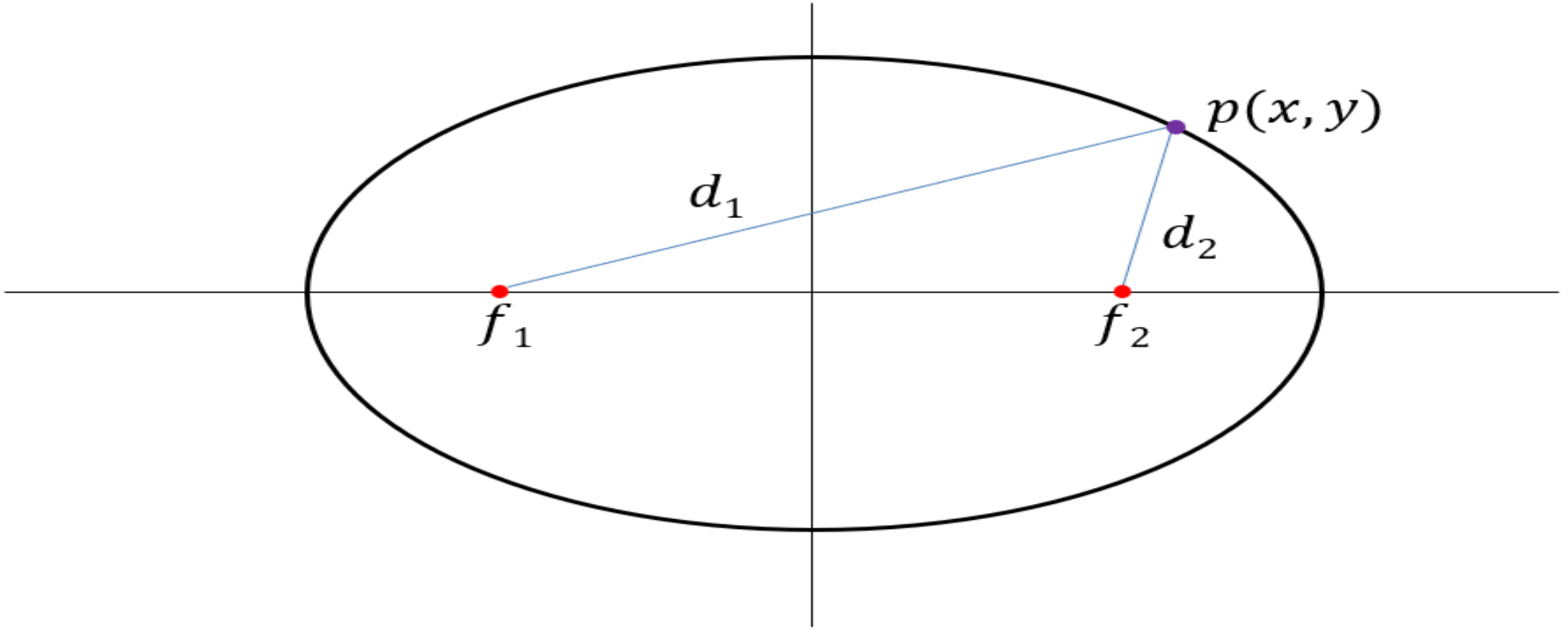
(1, 11)

| Center (1, 1) |
| --- |
| (2, 11) |
| (3, 11) |
| (4, 11) |
| (5, 10) |
| (6, 10) |
| (7, 9) |
| (8, 8) |



# 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.



# Contd.

- If we labeled distance from two foci to any point on ellipse boundary as $d_1$ and $d_2$ then the general equation of an ellipse can be written as:

$$d_1 + d_2 = Constant$$

- Expressing distance in terms of focal coordinates $f_1 = (x_1, y_1)$ and $f_2 = (x_2, y_2)$ we have
- $\sqrt{(x - x_1)^2 + (y - y_1)^2} + \sqrt{(x - x_2)^2 + (y - y_2)^2} = Constant$

  [Using Distance formula]

# Properties of Ellipse-Specifying Equations

- An interactive method for specifying an ellipse in an arbitrary orientation is to input
  - ✓ two foci and
  - ✓ a point on the ellipse boundary.
- With this three coordinates we can evaluate constant in equation:

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

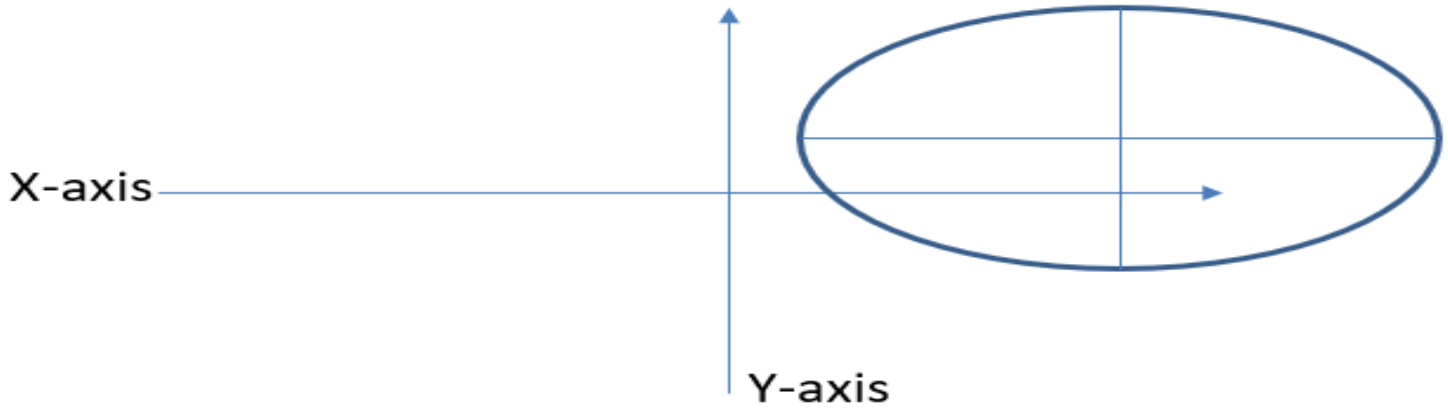
- We can also write this equation in the form

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

- Where the coefficients $A, B, C, D, E,$ and $F$ are evaluated in terms of the focal coordinates and the dimensions of the major and minor axes of the ellipse.

# Contd.

- Then coefficient in $Ax^2 + By^2 + Cxy + Dx + Ey + F = 0$ can be evaluated and used to generate pixels along the elliptical path.
- We can say ellipse is in standard position if their major and minor axes are parallel to x-axis and y-axis.
- Ellipse equation are greatly simplified if we align major and minor axis with coordinate axes i.e. x-axis and y-axis.



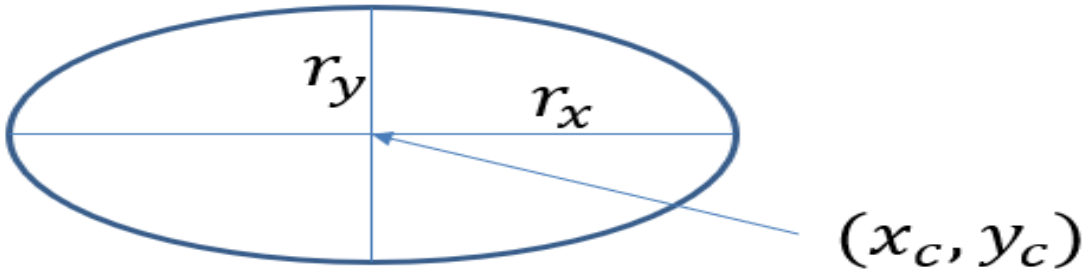
# Contd.

- Equation of ellipse can be written in terms of the ellipse center coordinates $(x_c, y_c)$ and parameters $r_x$ and $r_y$ as.

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

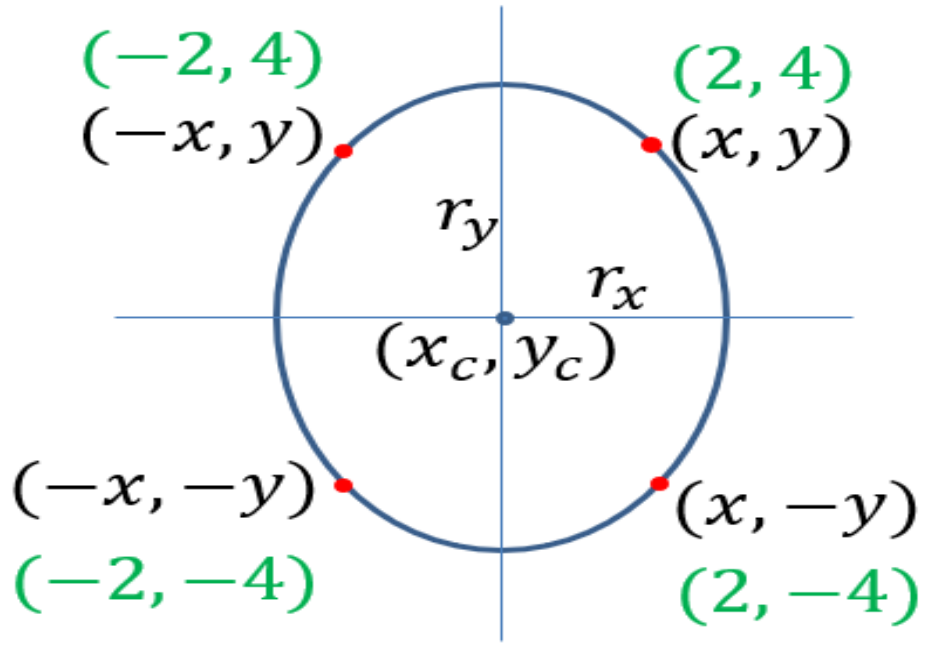
- Using the polar coordinates r and θ, we can also describe the ellipse in standard position with the parametric equations:

$$x = x_c + r_x \cos\theta$$
$$y = y_c + r_y \sin\theta$$



# Properties of Ellipse-Symmetry

- Symmetry property further reduced computations.
- An ellipse in standard position is symmetric between quadrant.

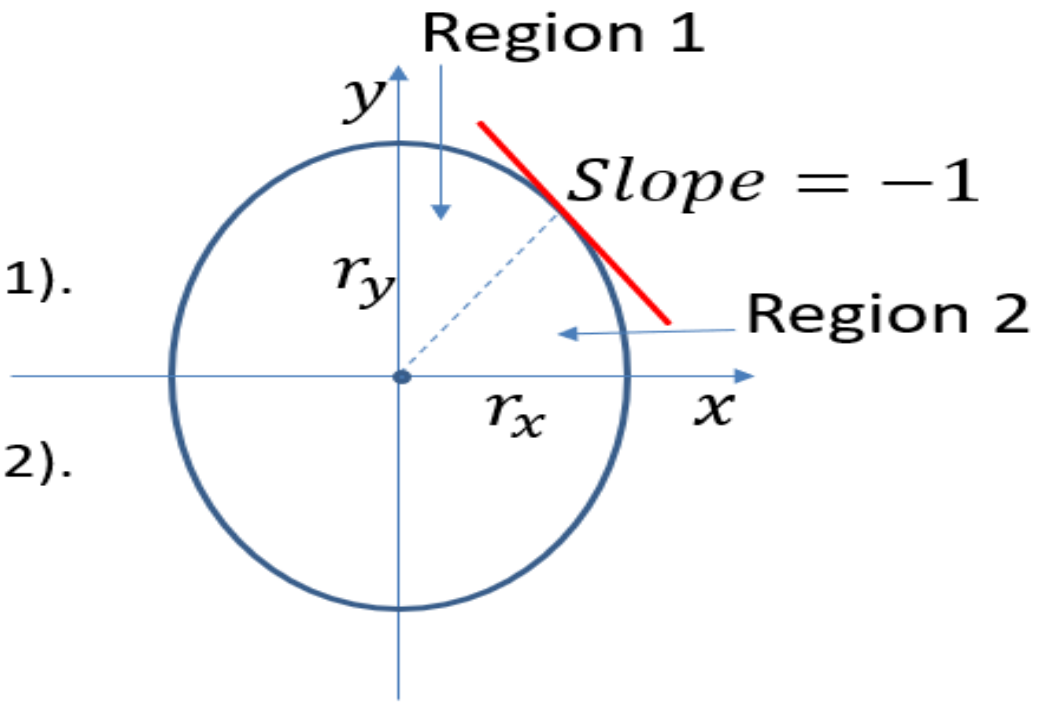


# Introduction to Midpoint Ellipse Algorithm

- Given parameters $r_x, r_y, \& (x_c, y_c)$.
- We determine points $(x, y)$ for an ellipse in standard position centered on the origin.
- Then we shift the points so the ellipse is centered at $(x_c, y_c)$.
- If we want to display the ellipse in nonstandard position then we rotate the ellipse about its center to align with required direction.
- For the present we consider only the standard position.
- We draw ellipse in first quadrant and than use symmetry property for other three quadrant.

# Regions in Midpoint Ellipse Algorithm

- In this method we divide first quadrant into two parts according to the slope of an ellipse
- Boundary divides region at
  - slope = -1.
- We take unit step in X direction
  - If magnitude of ellipse slope < 1 (Region 1).
- We take unit step in Y direction
  - If magnitude of ellipse slope > 1 (Region 2).



# Ways of Processing Midpoint Ellipse Algorithm

- We can start from $(0, r_y)$ and step clockwise along the elliptical path in the first quadrant
- Alternatively, we could start at $(r_x, 0)$ and select points in a counterclockwise order.
- With parallel processors, we could calculate pixel positions in the two regions simultaneously.
- Here we consider sequential implementation of midpoint algorithm.
- We take the start position at $(0, r_y)$ and steps along the elliptical path in clockwise order through the first quadrant.

# Decision Parameter Midpoint Ellipse Algorithm

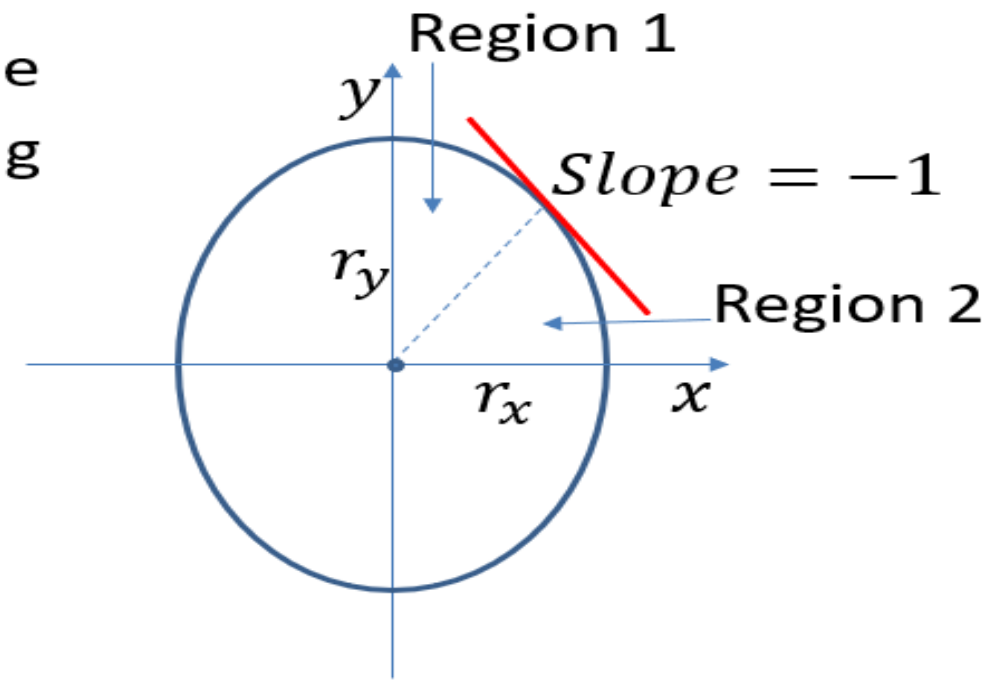
- We define ellipse function for center of ellipse at (0, 0) as follows.
- $f_{ellipse}(x, y) = r_y{}^2 x^2 + r_x{}^2 y^2 - r_y{}^2 r_x{}^2$
- Which has the following properties:

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

- Thus the ellipse function serves as the decision parameter in the midpoint ellipse algorithm.
- At each sampling position we select the next pixel from two candidate pixel.

# Processing Steps of Midpoint Ellipse Algorithm

- Starting at $(0,\ r_y)$ we take unit step in $x$ direction until we reach the boundary between region-1 and region-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.

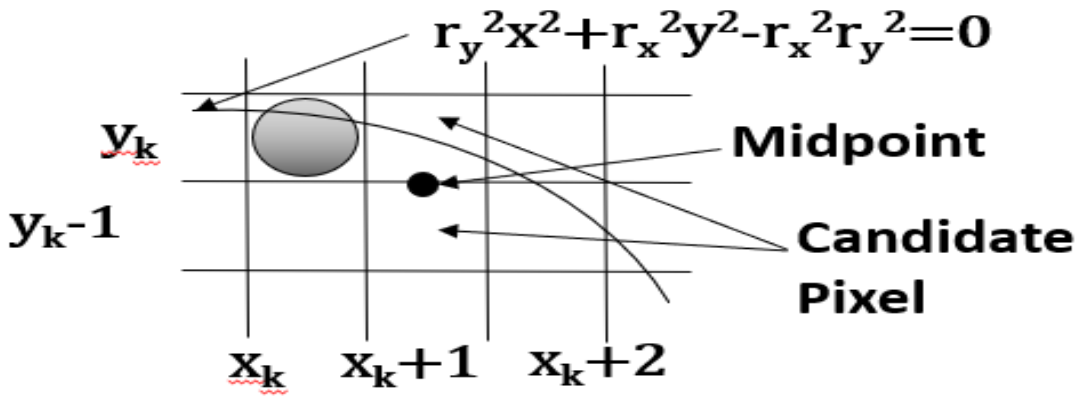


# Decide Boundary between Region 1 and 2

- The ellipse slope is calculated using following equation.
- $\frac{dy}{dx} = -\frac{2r_y{}^2 x}{2r_x{}^2 y}$
- At boundary between region 1 and 2 slope= -1 and equation become.
- $2r_y{}^2 x = 2r_x{}^2 y$
- Therefore we move out of region 1 whenever following equation is false:
- $2r_y{}^2 x \leq 2r_x{}^2 y$

# Midpoint between Candidate pixel in Region 1

- Figure shows the midpoint between the two candidate pixels at sampling position $x_k + 1$ in the first region.




- Assume we are at $(x_k, y_k)$ position and we determine the next position along the ellipse path, by evaluating decision parameter at midpoint between two candidate pixels.
- $p1_k = f_{ellipse}\left(x_k + 1, y_k - \frac{1}{2}\right)$

# Derivation for Region 1

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

# Contd.

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

# Contd.

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

# Contd.

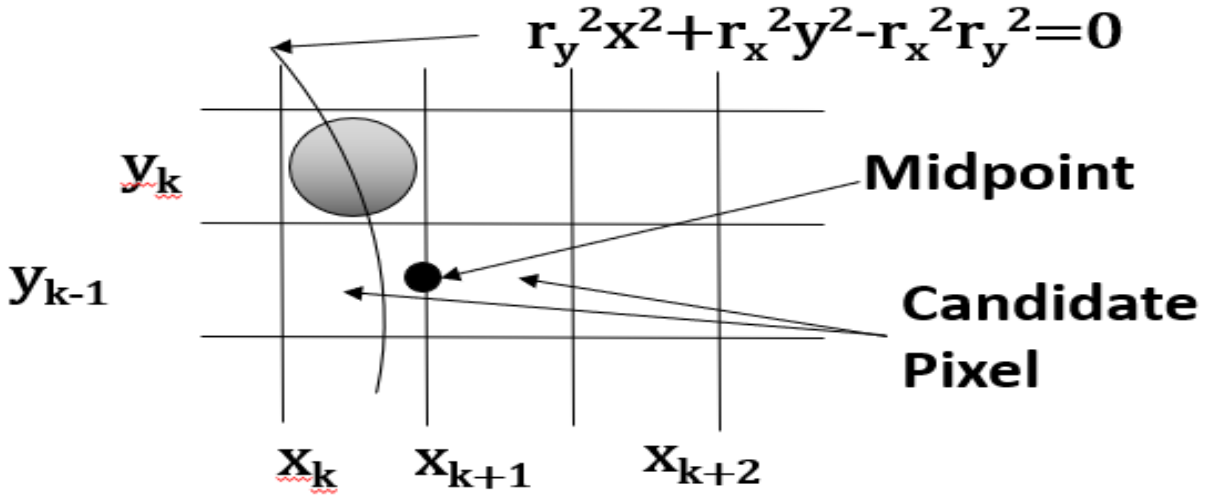
- $p1_{k+1} = p1_k + 2{r_y}^2(x_k + 1) + {r_y}^2 + {r_x}^2\left[\left(y_{k+1} - \frac{1}{2}\right)^2 - \left(y_k - \frac{1}{2}\right)^2\right]$
- $y_{k+1}$ is either $y_k$ or $y_k - 1$, depends on the sign of $p1_k$

# IDP for Region 1

- 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 between Candidate pixel in Region 2

- Now we similarly calculate over region-2.
- Unit stepping in negative $y$ direction and the midpoint is now taken between horizontal pixels at each step.




- For this region, the decision parameter is evaluated as follows.
- $p2_k = f_{ellipse}\left(x_k + \frac{1}{2}, y_k - 1\right)$

# Derivation for Region 2

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

# Contd.

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

# Contd.

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

# Contd.

- $p2_{k+1} - p2_k = -2r_x{}^2(y_k - 1) + r_x{}^2 + r_y{}^2 \left[\left(x_{k+1} + \frac{1}{2}\right)^2 - \left(x_k + \frac{1}{2}\right)^2\right]$
- Now making $p2_{k+1}$ as subject.
- $p2_{k+1} = p2_k - 2r_x{}^2(y_k - 1) + r_x{}^2 + r_y{}^2 \left[\left(x_{k+1} + \frac{1}{2}\right)^2 - \left(x_k + \frac{1}{2}\right)^2\right]$
- Here $x_{k+1}$ is either $x_k$ or $x_k + 1$, depends on the sign of $p2_k$.

# IDP for Region 2

- In region-2 initial position is selected which is last position of region one and the initial decision parameter is calculated as follows.
- $p2_0 = f_{ellipse}\left(x_0 + \frac{1}{2}, y_0 - 1\right)$
- $p2_0 = {r_y}^2\left(x_0 + \frac{1}{2}\right)^2 + {r_x}^2(y_0 - 1)^2 - {r_x}^2{r_y}^2$
- For simplify calculation of $p2_0$ we could also select pixel position in counterclockwise order starting at $(r_x, 0)$.

**Algorithm for Midpoint Ellipse Generation**

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 $x_k$ position in region 1, starting at $k = 0$, perform the following test:

   If $p1_k < 0$, than the next point along the ellipse is $(x_{k+1}, y_k)$ and

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

   Otherwise, the next point along the ellipse is $(x_{k+1}, y_k - 1)$ and

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

   With

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

   And continue until $2{r_y}^2 x \leq 2{r_x}^2 y$

**Contd.**

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

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

5. At each $y_k$ position in region-2, starting at $k = 0$, perform the following test:

   If $p2_k > 0$, the next point along the ellipse is $(x_k, y_k - 1)$ and

$$p2_{k+1} = p2_k - 2{r_x}^2 y_{k+1} + {r_x}^2$$

   Otherwise, the next point along the ellipse is $(x_k + 1, y_k - 1)$ and

$$p2_{k+1} = p2_k - 2{r_x}^2 y_{k+1} + {r_x}^2 + 2{r_y}^2 x_{k+1}$$

   Using the same incremental calculations for $x$ and $y$ as in region 1.

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:

$$x = x + x_c, y = y + y_c$$

8. Repeat the steps for region 2 until $y_k \geq 0$.

# Example Midpoint Ellipse Algorithm

- Example: Calculate intermediate pixel position (For first quadrant) for ellipse with $r_x = 8, r_y = 6$ and ellipse center is at origin
- Initial point $(0, r_y) = (0, 6)$
- $p1_0 = {r_y}^2 - {r_x}^2 r_y + \frac{1}{4}{r_x}^2$
- $p1_0 = 6^2 - 8^2 * 6 + \frac{1}{4}8^2$
- $p1_0 = -332$

# Contd.

- $p1_o = -332$
- $p1_{k+1} = p1_k + 2r_y{}^2(x_k + 1) + r_y{}^2 + r_x{}^2\left[\left(y_{k+1} - \frac{1}{2}\right)^2 - \left(y_k - \frac{1}{2}\right)^2\right]$
- $p1_1 = -332 + 2 * 6^2(0 + 1) + 6^2 + 8^2\left[\left(6 - \frac{1}{2}\right)^2 - \left(6 - \frac{1}{2}\right)^2\right] = -224$

| K | $p1_k$ | $(x_{k+1}, y_{k+1})$ |
|---|---|---|
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

*Calculation stop when* $2r_y{}^2x > 2r_x{}^2y$

*If* $p1_k < 0$ *so we select* $y_{k+1} = y_k$
Else *we select* $y_{k+1} = y_k - 1$

# Contd.

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

- $p1_2 = -224 + 2 * 6^2(1 + 1) + 6^2 + 8^2\left[\left(6 - \frac{1}{2}\right)^2 - \left(6 - \frac{1}{2}\right)^2\right] = -44$

| K | $p1_k$ | $(x_{k+1}, y_{k+1})$ |
|---|---|---|
| 0 | -332 | (1, 6) |
| 1 | -224 | (2, 6) |
| | | |
| | | |
| | | |
| | | |
| | | |

*Calculation stop when* $2r_y{}^2x > 2r_x{}^2y$

*If* $p1_k < 0$ *so we select* $y_{k+1} = y_k$
Else *we select* $y_{k+1} = y_k - 1$

# Contd.

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

- $p1_3 = -44 + 2 * 6^2(2 + 1) + 6^2 + 8^2\left[\left(6 - \frac{1}{2}\right)^2 - \left(6 - \frac{1}{2}\right)^2\right] = 208$

| K | $p1_k$ | $(x_{k+1}, y_{k+1})$ |
|---|---|---|
| 0 | -332 | (1, 6) |
| 1 | -224 | (2, 6) |
| 2 | -44 | (3, 6) |
| | | |
| | | |
| | | |
| | | |

*Calculation stop when* $2r_y{}^2x > 2r_x{}^2y$

*If* $p1_k < 0$ *so we select* $y_{k+1} = y_k$
Else *we select* $y_{k+1} = y_k - 1$

# Contd.

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

- $p1_4 = 208 + 2 * 6^2(3 + 1) + 6^2 + 8^2\left[\left(5 - \frac{1}{2}\right)^2 - \left(6 - \frac{1}{2}\right)^2\right] = -108$

| K | $p1_k$ | $(x_{k+1}, y_{k+1})$ |
|---|---|---|
| 0 | -332 | (1, 6) |
| 1 | -224 | (2, 6) |
| 2 | -44 | (3, 6) |
| 3 | 208 | (4, 5) |
| | | |
| | | |
| | | |

*Calculation stop when* $2r_y{}^2x > 2r_x{}^2y$

*If* $p1_k < 0$ *so we select* $y_{k+1} = y_k$
Else *we select* $y_{k+1} = y_k - 1$

# Contd.

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

- $p1_5 = -108 + 2 * 6^2(4 + 1) + 6^2 + 8^2\left[\left(5 - \frac{1}{2}\right)^2 - \left(5 - \frac{1}{2}\right)^2\right] = 288$

| K | $p1_k$ | $(x_{k+1}, y_{k+1})$ |
|---|---|---|
| 0 | -332 | (1, 6) |
| 1 | -224 | (2, 6) |
| 2 | -44 | (3, 6) |
| 3 | 208 | (4, 5) |
| 4 | -108 | (5, 5) |
| | | |
| | | |

*Calculation stop when* $2r_y{}^2x > 2r_x{}^2y$

*If* $p1_k < 0$ *so we select* $y_{k+1} = y_k$
Else *we select* $y_{k+1} = y_k - 1$

# Contd.

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

- $p1_6 = 288 + 2 * 6^2(5 + 1) + 6^2 + 8^2\left[\left(4 - \frac{1}{2}\right)^2 - \left(5 - \frac{1}{2}\right)^2\right] = 244$

| K | $p1_k$ | $(x_{k+1}, y_{k+1})$ |
|---|---|---|
| 0 | -332 | (1, 6) |
| 1 | -224 | (2, 6) |
| 2 | -44 | (3, 6) |
| 3 | 208 | (4, 5) |
| 4 | -108 | (5, 5) |
| 5 | 288 | (6, 4) |
| | | |

*Calculation stop when* $2r_y{}^2x > 2r_x{}^2y$

*If* $p1_k < 0$ *so we select* $y_{k+1} = y_k$
Else *we select* $y_{k+1} = y_k - 1$

# Contd.

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

- $p2_0 = 6^2\left(7 + \frac{1}{2}\right)^2 + 8^2(3 - 1)^2 - 8^2 6^2 = -23$

| K | $p2_k$ | $(x_{k+1}, y_{k+1})$ |
|---|---|---|
| | | |
| | | |
| | | |

*Calculation stop when* $y_k \leq 0$

*If* $p2_k > 0$ *so we select* $x_{k+1} = x_k$
Else *we select* $x_{k+1} = x_k + 1$

# Contd.

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

- $p2_1 = -23 - 2 * 8^2\,(3 - 1) + 8^2 + 6^2\left[\left(8 + \frac{1}{2}\right)^2 - \left(7 + \frac{1}{2}\right)^2\right] = 361$

| K | $p2_k$ | $(x_{k+1}, y_{k+1})$ |
|---|---|---|
| 0 | -23 | (8, 2) |
| | | |
| | | |

*Calculation stop when* $y_k \leq 0$

*If* $p2_k > 0$ *so we select* $x_{k+1} = x_k$
Else *we select* $x_{k+1} = x_k + 1$

# Contd.

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

- $p2_2 = 361 - 2 * 8^2\,(2 - 1) + 8^2 + 6^2\left[\left(8 + \frac{1}{2}\right)^2 - \left(8 + \frac{1}{2}\right)^2\right] = 297$

| K | $p2_k$ | $(x_{k+1}, y_{k+1})$ |
|---|---|---|
| 0 | -23 | (8, 2) |
| 1 | 361 | (8, 1) |
| | | |

*Calculation stop when* $y_k \leq 0$

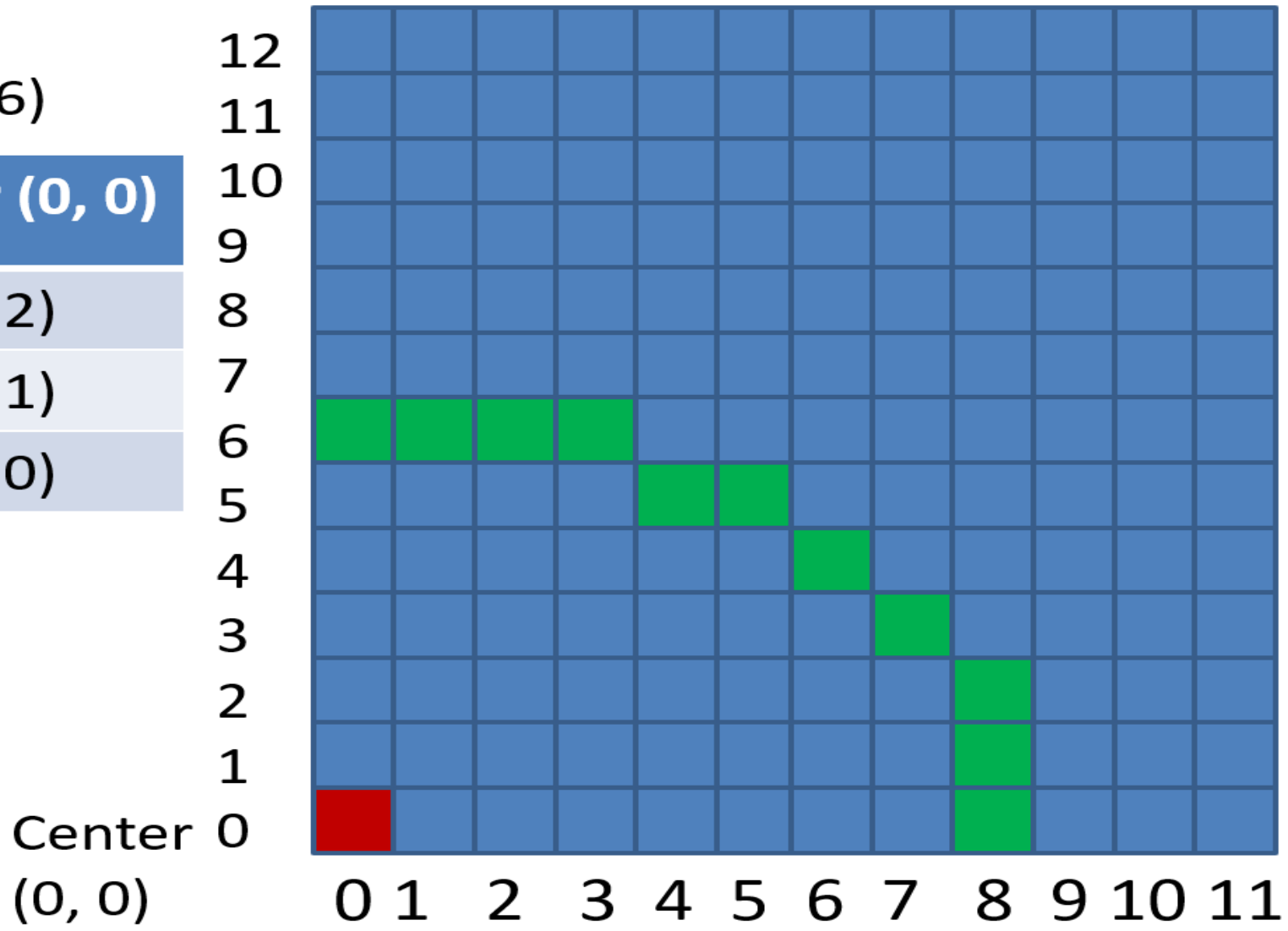
*If* $p2_k > 0$ *so we select* $x_{k+1} = x_k$
Else *we select* $x_{k+1} = x_k + 1$

# Contd.

- Plot the pixel.
- Plot initial point(0, 6)

| Center (0, 0) |
| --- |
| (1, 6) |
| (2, 6) |
| (3, 6) |
| (4, 5) |
| (5, 5) |
| (6, 4) |
| (7, 3) |

| Center (0, 0) |
| --- |
| (8, 2) |
| (8, 1) |
| (8, 0) |

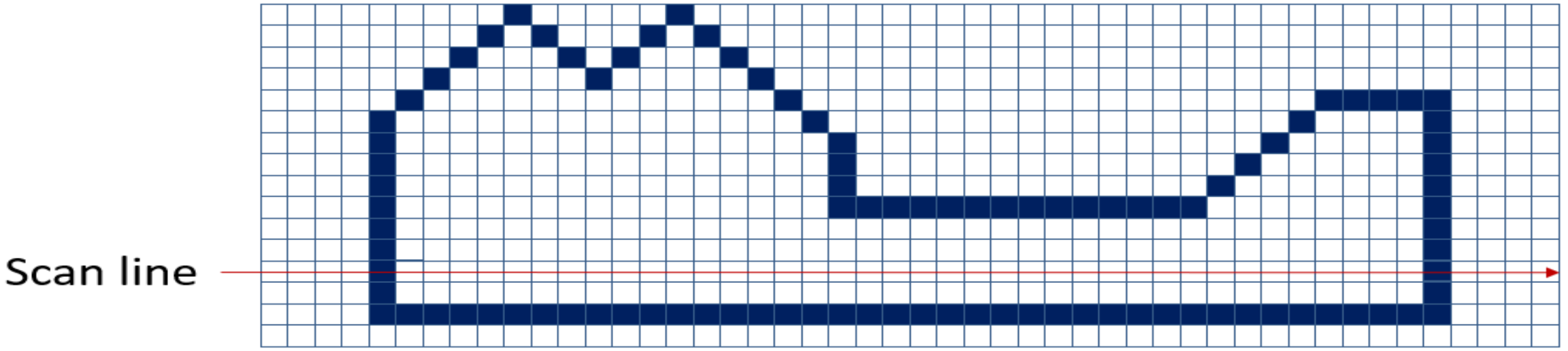


# Filled-Area Primitives

- In practical we often use polygon which are filled with some colour or pattern inside it.
- There are two basic approaches to area filling on raster systems.
  - One way to fill an area is to determine the overlap intervals for scan line that cross the area.
  - Another method is to start from a given interior position and paint outward from this point until we encounter boundary.

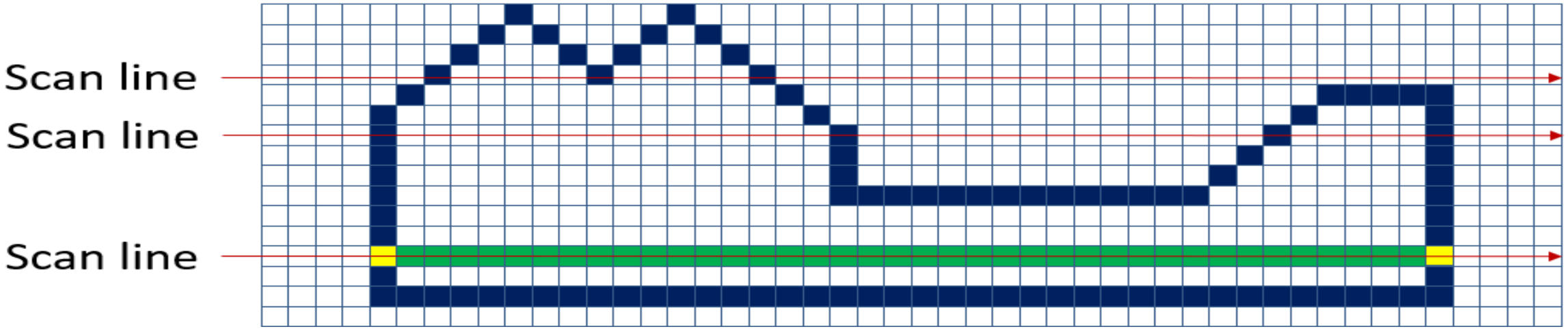
# Scan-Line Polygon Fill Algorithm

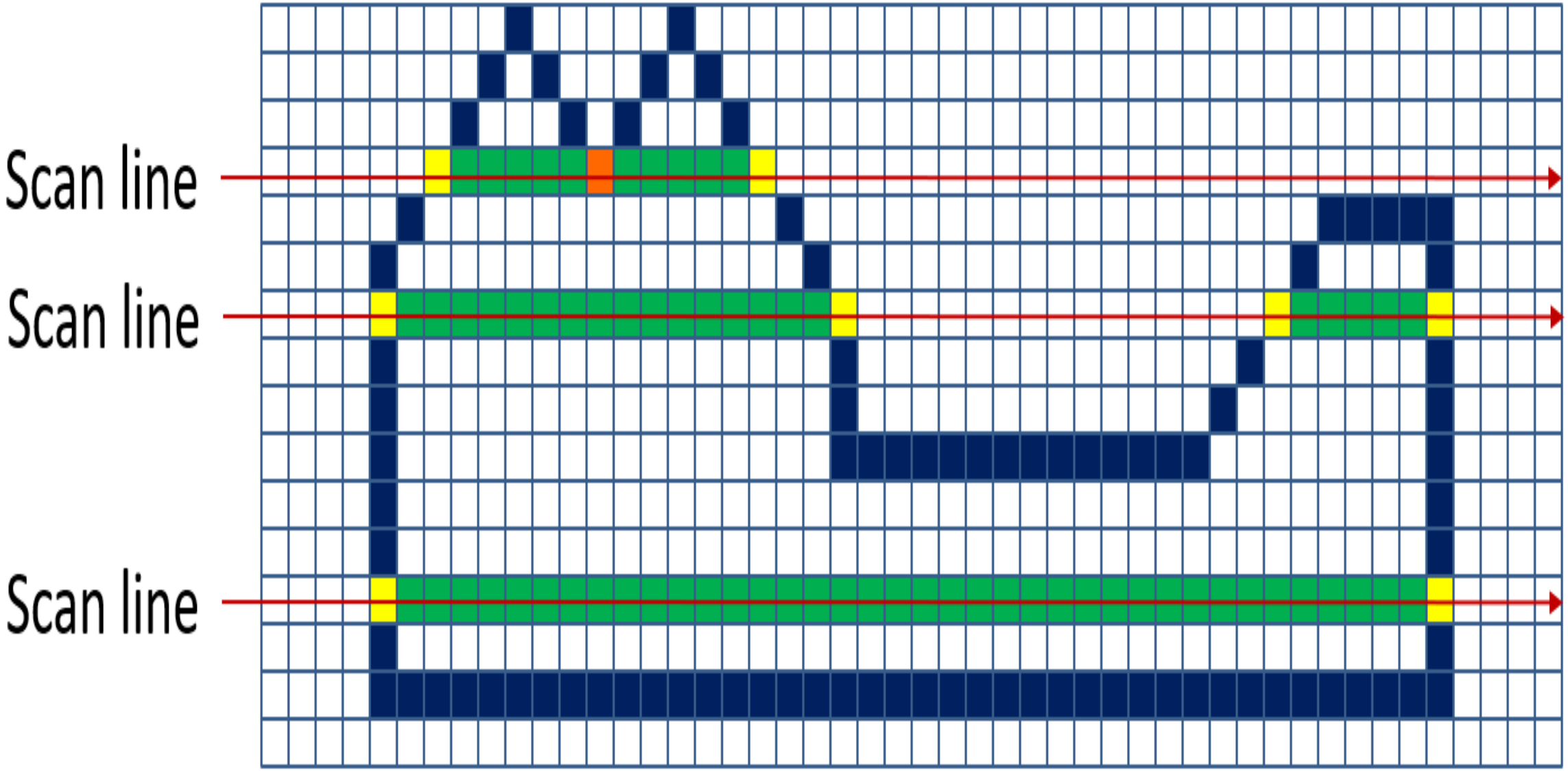
- For each scan-line crossing a polygon, the algorithm locates the intersection points are of scan line with the polygon edges.
- This intersection points are stored from left to right.
- Frame buffer positions between each pair of intersection point are set to specified fill color.



# Contd.

- Scan line intersects at vertex are required special handling.
- For vertex we must look at the other endpoints of the two line segments 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.




Scan line
Scan line
Scan line

# Edge Intersection Calculation with Scan-Line

- Coherence methods often involve incremental calculations applied along a single scan line or between successive scan lines.
- In determining edge intersections, we can set up incremental coordinate calculations along any edge by exploiting the fact that the slope of the edge is constant from one scan line to the next.
- For above figure we can write slope equation for polygon boundary as follows.
- $m = \frac{y_{k+1} - y_k}{x_{k+1} - x_k}$
- Since change in $y$ coordinates between the two scan lines is simply
- $y_{k+1} - y_k = 1$

# Contd.

- So slope equation can be modified as follows
- $m = \frac{y_{k+1} - y_k}{x_{k+1} - x_k}$
- $m = \frac{1}{x_{k+1} - x_k}$
- $x_{k+1} - x_k = \frac{1}{m}$
- $x_{k+1} = x_k + \frac{1}{m}$
- Each successive $x$ intercept can thus be calculated by adding the inverse of the slope and rounding to the nearest integer.

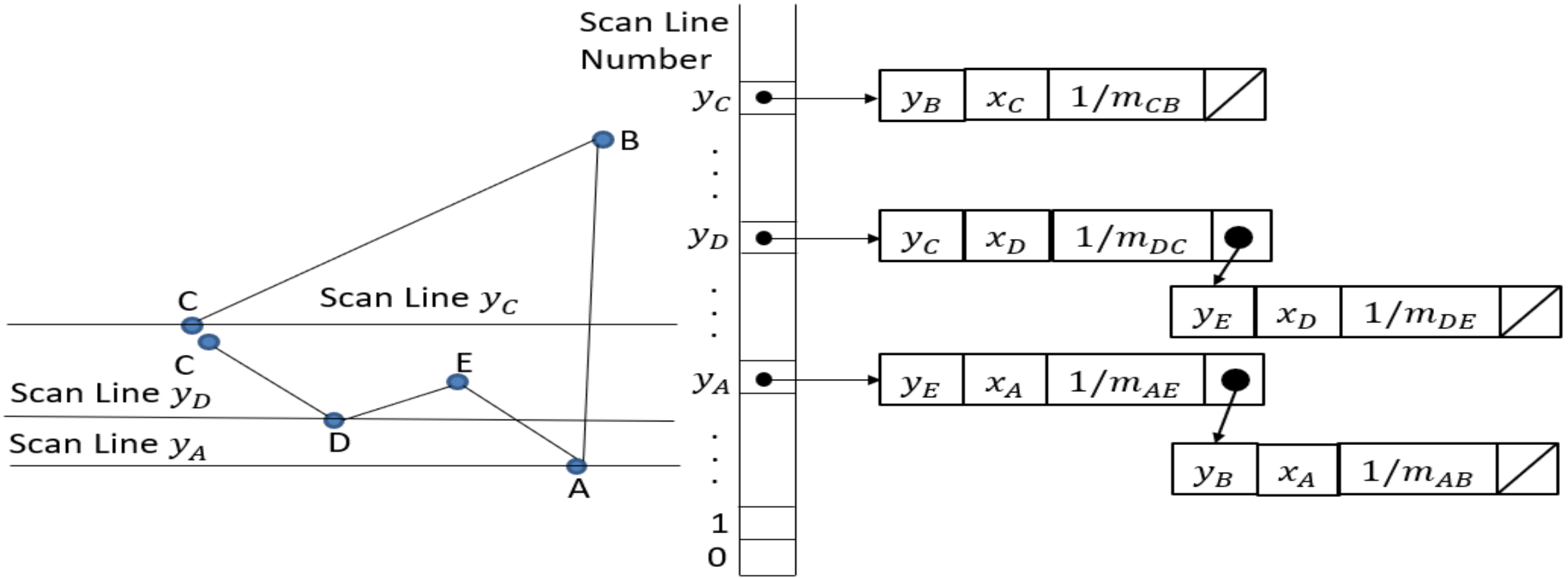
# Edge Intersection Calculation with Scan-Line for parallel execution

- For parallel execution of this algorithm we assign each scan line to separate processor in that case instead of using previous $x$ values for calculation we use initial $x$ values by using equation as.
- $x_k = x_0 + \frac{k}{m}$
- Now if we put $m = \frac{\Delta y}{\Delta x}$ in incremental calculation equation $x_{k+1} = x_k + \frac{1}{m}$ then we obtain equation as.
- $x_{k+1} = x_k + \frac{\Delta x}{\Delta y}$
- Using this equation we can perform integer evaluation of $x$ intercept.

# Use of Sorted Edge table in Scan-Line Polygon Fill Algorithm

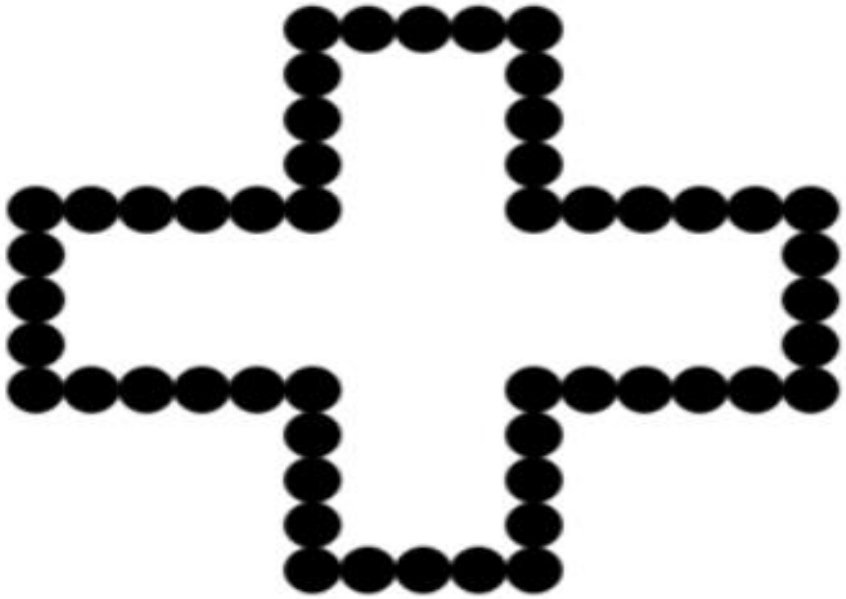
- To efficiently perform a polygon fill, we can first store the polygon boundary in a sorted edge table.
- It contains all the information necessary to process the scan lines efficiently.
- We use bucket sort to store the edge sorted on the smallest $y$ value of each edge in the correct scan line positions.
- Only the non-horizontal edges are entered into the sorted edge table.

# Contd.

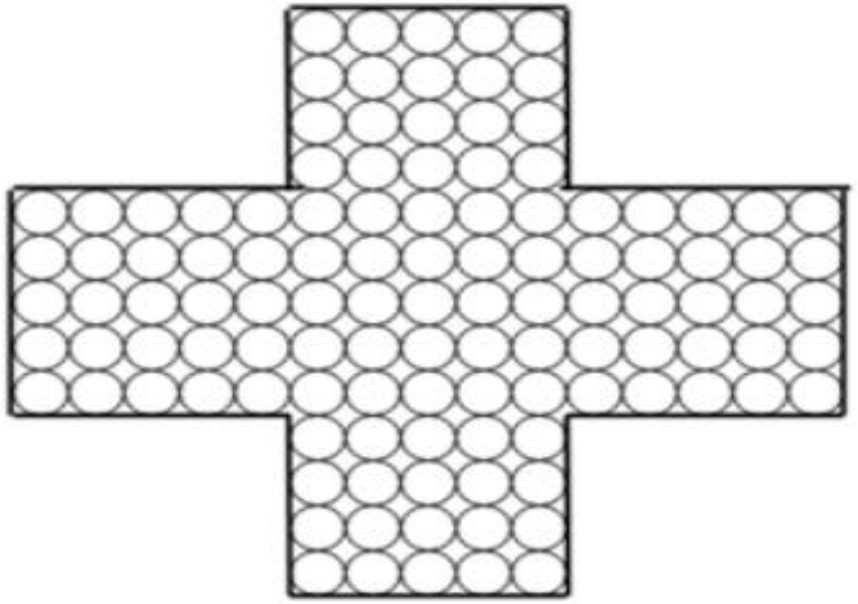

Scan Line
Number
$y_C$
$y_D$
$y_A$
1
0
B
C
Scan Line $y_C$
C
E
Scan Line $y_D$
D
Scan Line $y_A$
A
$y_B$ $x_C$ $1/m_{CB}$
$y_C$ $x_D$ $1/m_{DC}$
$y_E$ $x_D$ $1/m_{DE}$
$y_E$ $x_A$ $1/m_{AE}$
$y_B$ $x_A$ $1/m_{AB}$

# Filled Area Primitives:

Region filling is the process of filling image or region. Filling can be of boundary or interior region as shown in fig. Boundary Fill algorithms are used to fill the boundary and flood-fill algorithm are used to fill the interior.



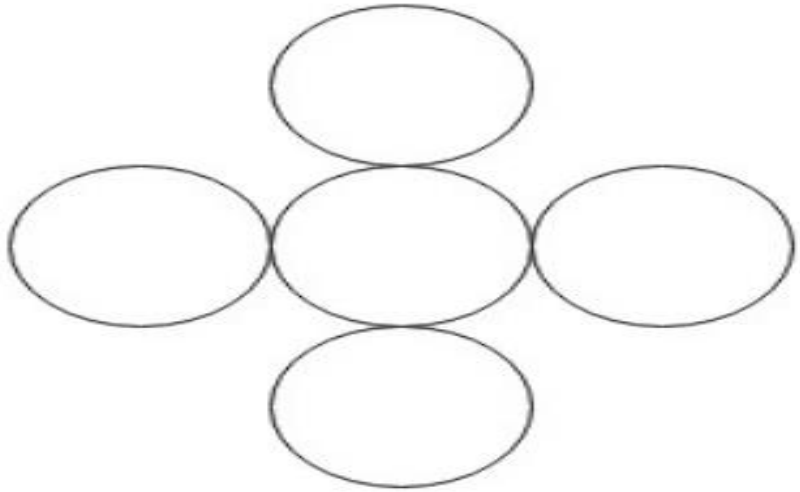
Boundary Filled Region



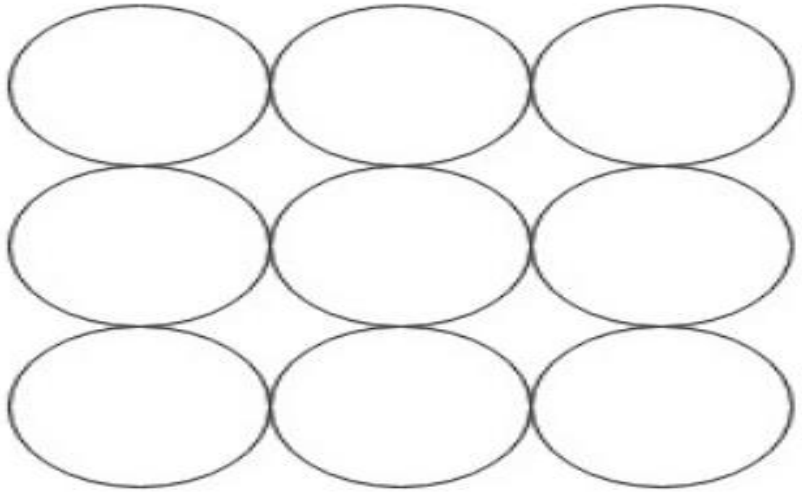
Interior or Flood Filled Region

# Boundary Filled 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



Eight Connected

# Boundary Filled Algorithm:

- Four connected approaches is more suitable than the eight connected approaches.

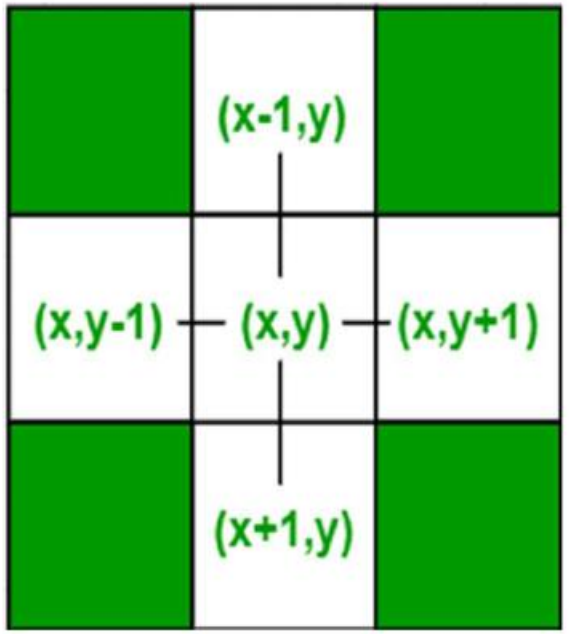
**1. Four connected approaches:** In this approach, left, right, above, below pixels are tested.

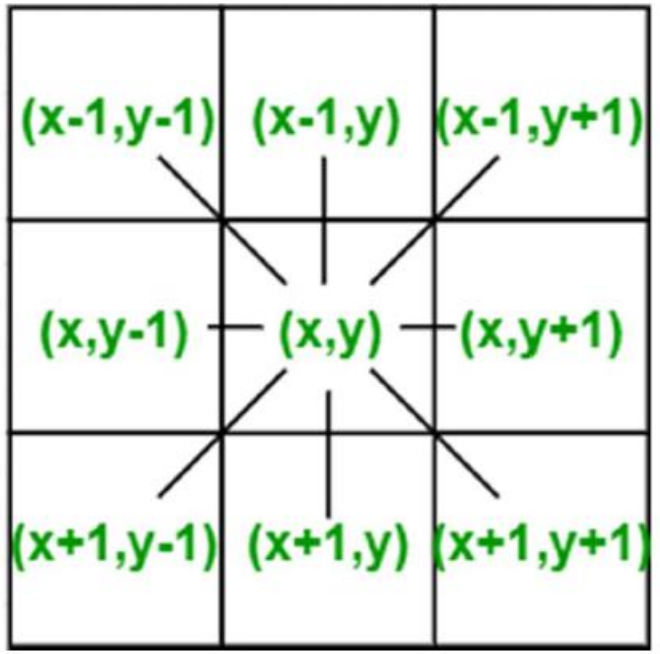
**2. Eight connected approaches:** In this approach, left, right, above, below and four diagonals are selected.

- Boundary can be checked by seeing pixels from left and right first. Then pixels are checked by seeing pixels from top to bottom. The algorithm takes time and memory because some recursive calls are needed.

# Algorithm:

```
Procedure fill (x, y, color, color1: integer)
int c;
c=getpixel (x, y);
if (c!=color) (c!=color1)
{
    setpixel (x, y, color)
    fill (x+1, y, color, color 1);
     fill (x-1, y, color, color 1);
    fill (x, y+1, color, color 1);
    fill (x, y-1, color, color 1);
}
```


(x-1,y)
(x,y-1)
(x,y)
(x,y+1)
(x+1,y)


(x-1,y-1)
(x-1,y)
(x-1,y+1)
(x,y-1)
(x,y)
(x,y+1)
(x+1,y-1)
(x+1,y)
(x+1,y+1)

# Steps of boundary fill algorithm

1. Take the position of the starting point(seed point) and the boundary colour.
2. Decide whether you want to go in 4 directions (N, S, W, E) or 8 directions (N, S, W, E, NW, NE, SW, SE).
3. Choose a fill colour.
4. Travel in those directions.
5. If the pixel you land on is not the fill colour or the boundary colour , replace it with the fill colour.
6. Repeat 4 and 5 until you've been everywhere within the boundaries.

# Introduction to Flood-Fill Algorithm

- Sometimes it is required to fill in an area that is not defined within a single colour boundary.
- In such cases we can fill areas by replacing a specified interior colour instead of searching for a boundary colour.
- 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 colour instead of boundary colour and they are replaced by new colour.
- Using either a 4-connected or 8-connected approach, we can step through pixel positions until all interior point have been filled.

# Algorithm:

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

# Steps for Flood Fill

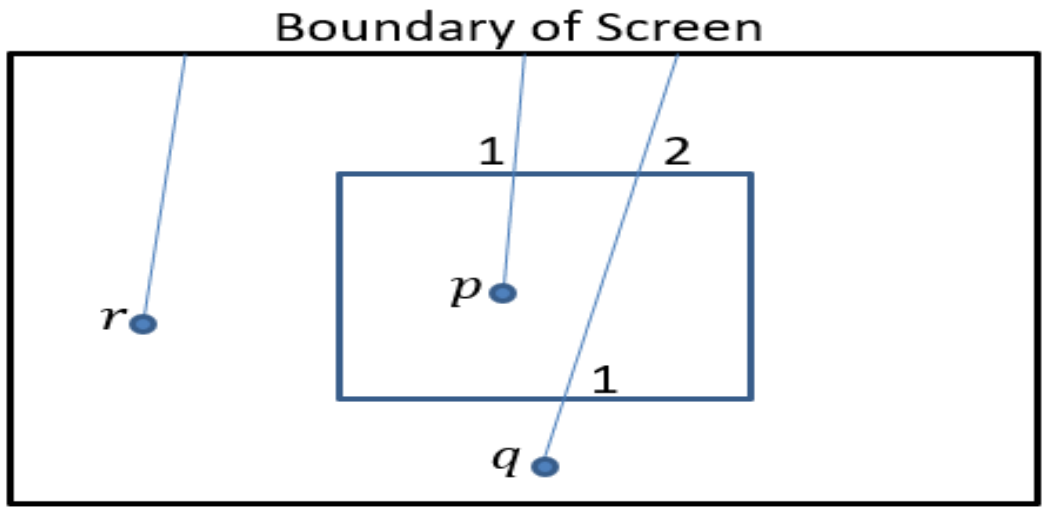
1. Take the position of the starting point.(seed)
2. Decide whether you want to go in 4 directions (**N, S, W, E**) or 8 directions (**N, S, W, E, NW, NE, SW, SE**).
3. Choose a replacement colour and a target colour. (t = green, r = purple)
4. Travel in those directions.
5. If the tile you land on is a target, replace it with the chosen colour.
6. Repeat 4 and 5 until you've been everywhere within the boundaries.

# 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
  1. Odd even rule OR
  2. Nonzero winding number rule

# Odd Even/ Odd Parity/ Even Odd Rule

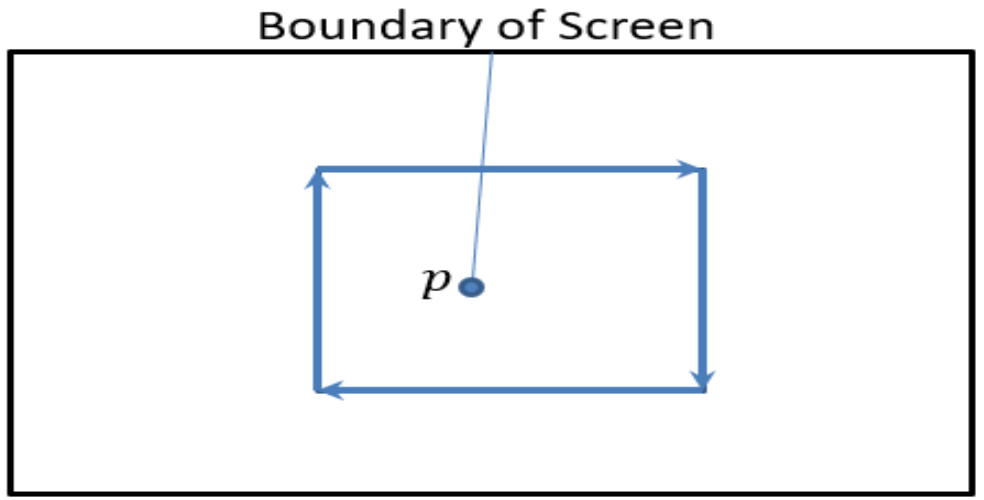
- By conceptually drawing a line from any position $p$ to a distant point outside the coordinate extents of the object.
- Than counting the number of edges crossing by this line.
  1. If Edge count is odd, than p is an interior point.
  2. Otherwise p is exterior point.
- To obtain accurate edge count we must sure that line selected is does not pass from any vertices.



# 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.
- We apply this rule by initializing winding number with 0.
- Then draw a line for any point $p$ to distant point beyond the coordinate extents of the object.

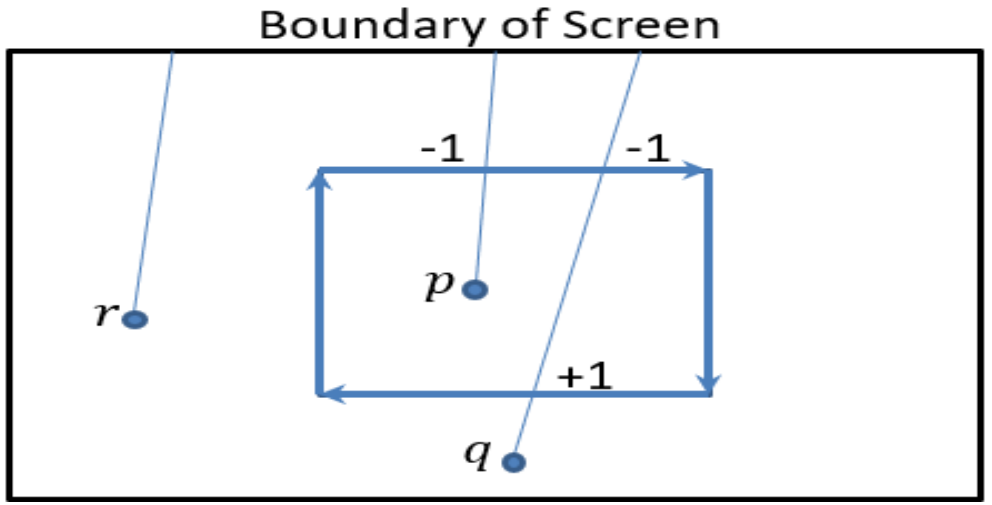
Winding number=0



# Contd.

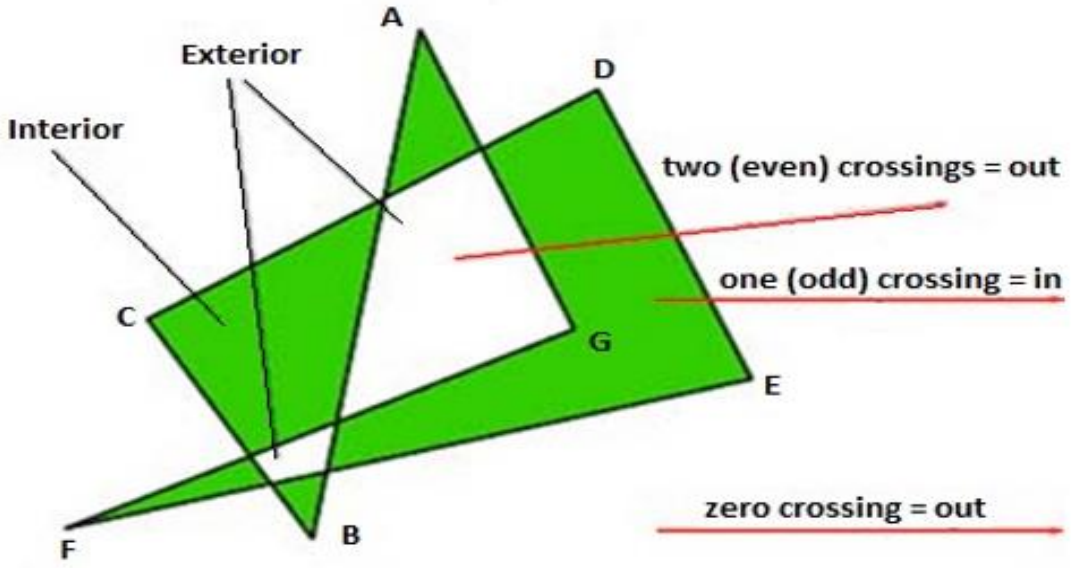
- The line we choose must not pass through vertices.
- Then we move along that line we find number of intersecting edges.
  1. If edge cross our line from right to left We add 1 to winding number
  2. Otherwise subtract 1 from winding number
- IF the final value of winding number is nonzero then the point is interior otherwise point is exterior.

**Winding number=0-1+1-1=0**

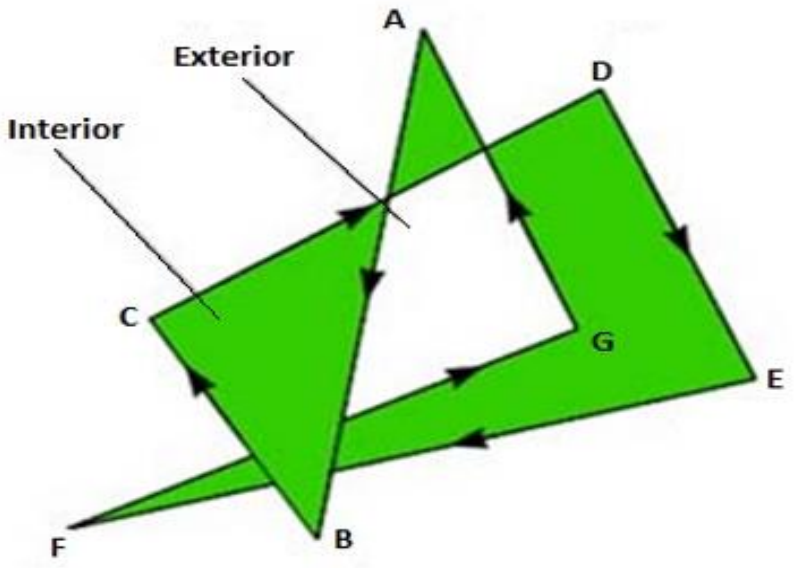


# Comparison between Odd Even Rule and Nonzero Winding Rule

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




**Odd Even Rule**




**Nonzero Winding Rule**

# Thank you