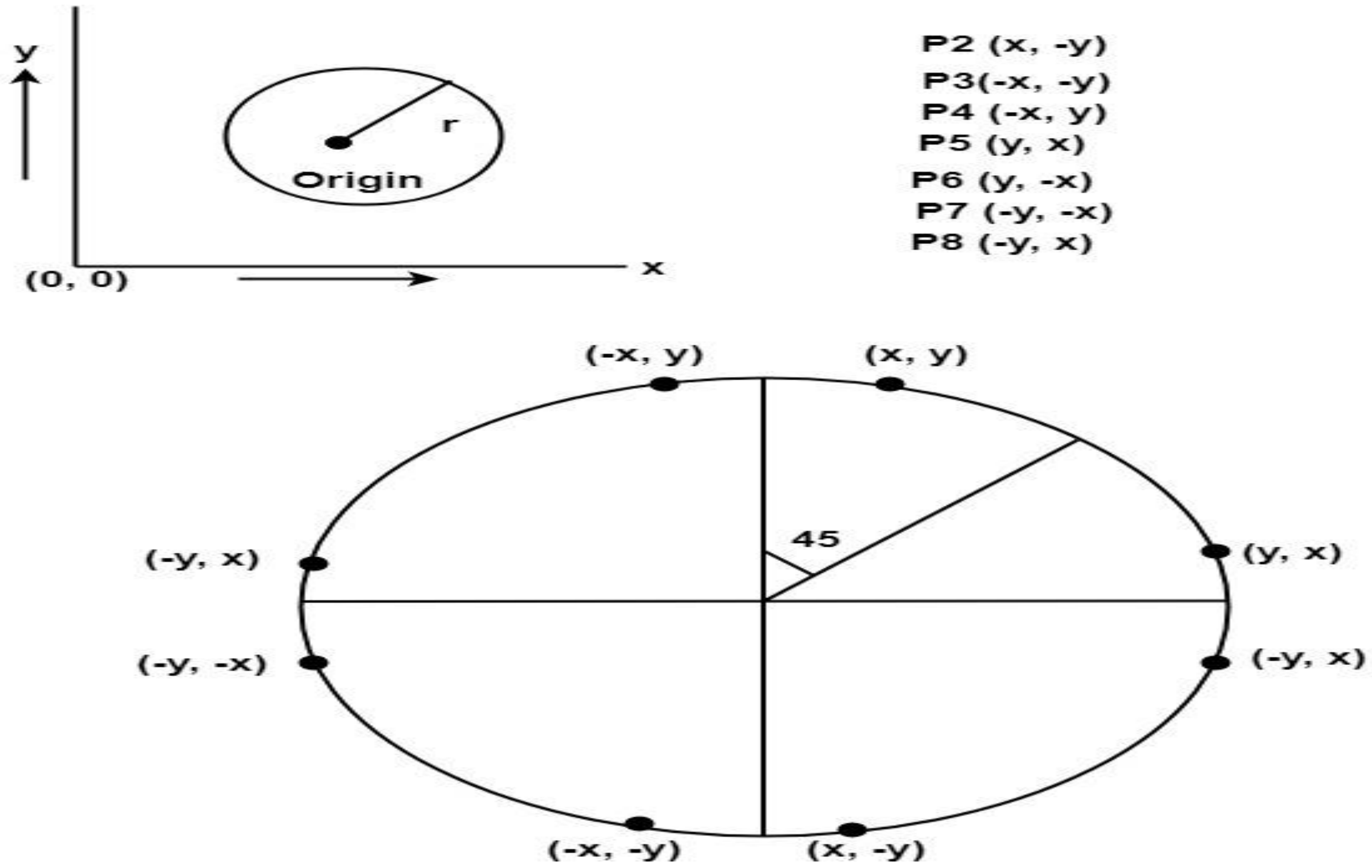
Circle Generation Algorithms

- Defining a Circle
- Defining a circle using Polynomial Method
- Defining a circle using Polar Co-ordinates
- Bresenham's Circle Algorithm
- Mid Point Circle Algorithm

Defining a Circle

- Circle is an eight-way symmetric figure.
- The shape of circle is the same in all quadrants. In each quadrant, there are two octants.
- If the calculation of the point of one octant is done, then the other seven points can be calculated easily by using the concept of eight-way symmetry.
- For drawing circle, considers it at the origin. If a point is P1(x, y), then the other seven points will be given as follows.
- So we will calculate only 45° arc. From which the whole circle can be determined easily.

Defining a Circle



Defining a Circle

- If we want to display circle on screen then the putpixel function is used for eight points as shown below:

  putpixel (x, y, color)

  putpixel (x, -y, color)

  putpixel (-x, y, color)

  putpixel (-x, -y, color)

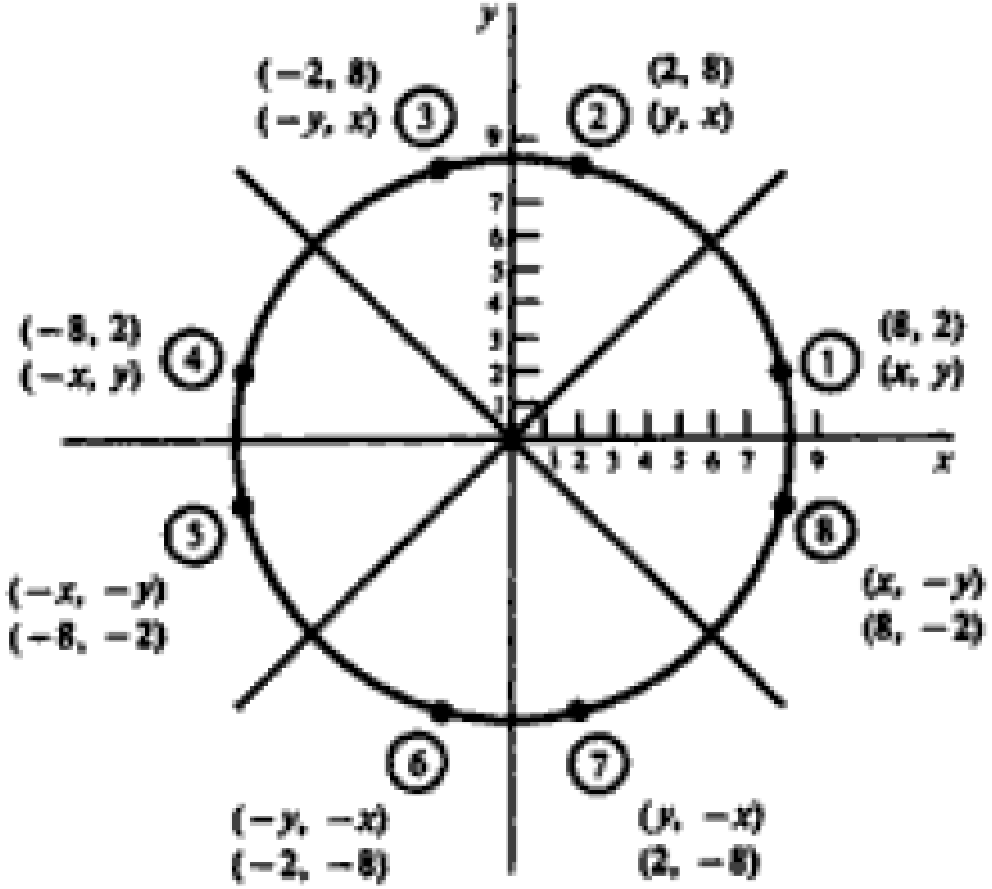
  putpixel (y, x, color)

  putpixel (y, -x, color)

  putpixel (-y, x, color)

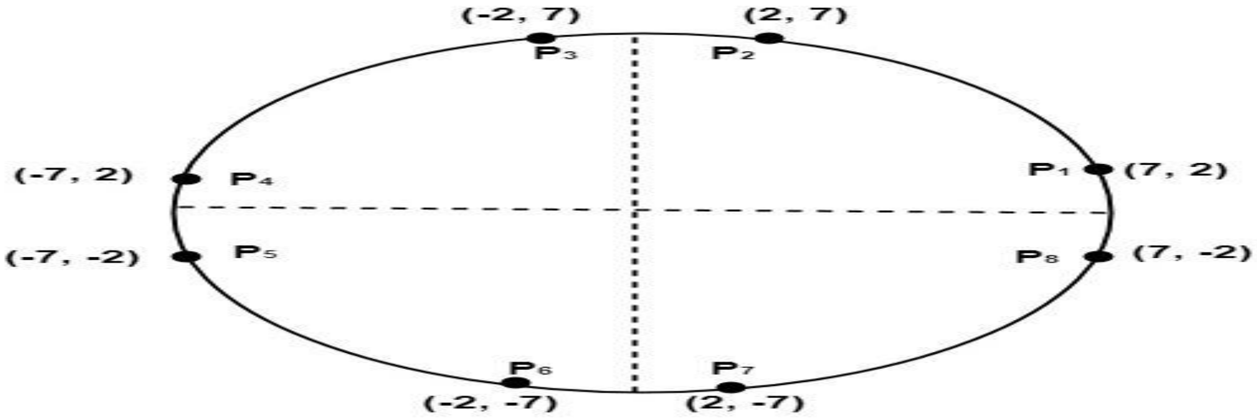
  putpixel (-y, -x, color)

Eight-way symmetry of circle



Defining a Circle

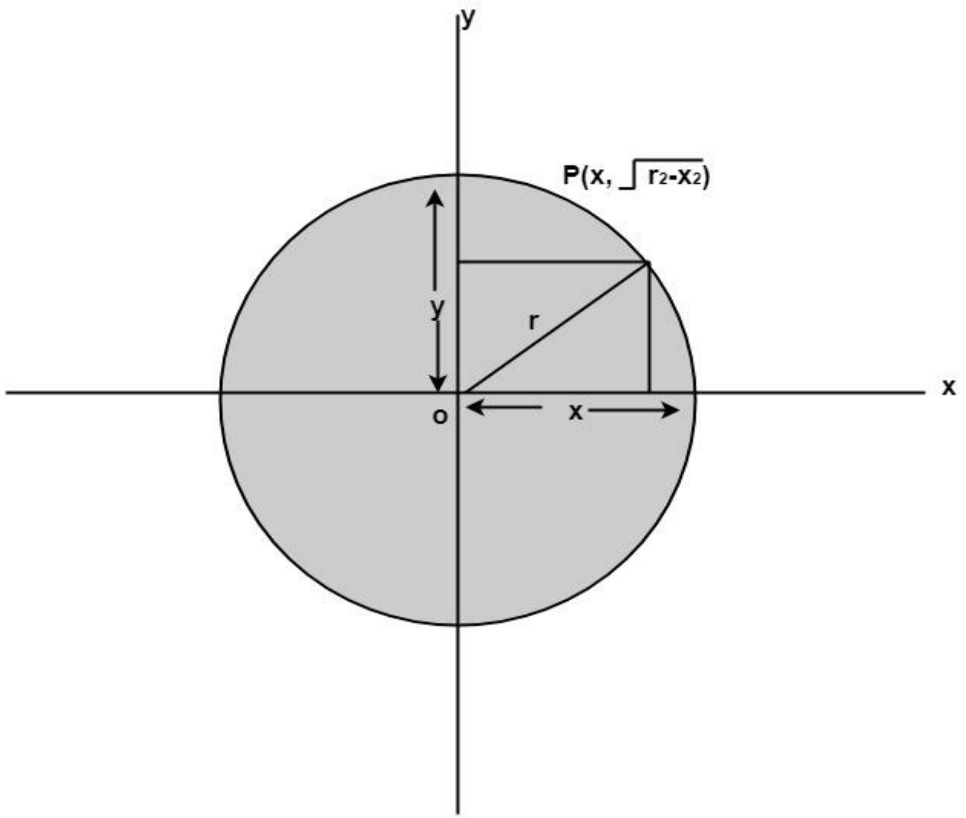
- **Example:** Let we determine a point (2, 7) of the circle then other points will be (2, -7), (-2, -7), (-2, 7), (7, 2), (-7, 2), (-7, -2), (7, -2)
- These seven points are calculated by using the property of reflection. The reflection is accomplished by reversing (x, y) co-ordinates.
- There are two standards methods of mathematically defining a circle centered at the origin. Defining a circle using Polynomial Method and Defining a circle using Polar Co-ordinates



Eight way symmetry of a Circle

Defining a Circle using Polynomial Method

- The first method defines a circle with the second-order polynomial equation as shown in fig:
- $$y^2 = r^2 - x^2$$
  Where x = the x coordinate, y = the y coordinate and r = the circle radius
- With the method, each x coordinate in the sector, from 90° to 45°, is found by stepping x from 0 to & each y coordinate is found by evaluating sqrt(r^2 - x^2) for each step of x.



Defining a Circle using Polynomial Method

Algorithm

- **Step1:** Set the initial variables

  r = circle radius

  (h, k) = coordinates of circle center

  x = 0

  i = step size

  $x_{end} = r/\sqrt{2}$

- **Step2:** Test to determine whether the entire circle has been scan-converted.

  If $(x > x_{end})$ then stop.

- **Step 3:** Compute $y = \sqrt{(r^2 - x^2)}$

- **Step4:** Plot the eight points found by symmetry concerning the center (h, k) at the current (x, y) coordinates.

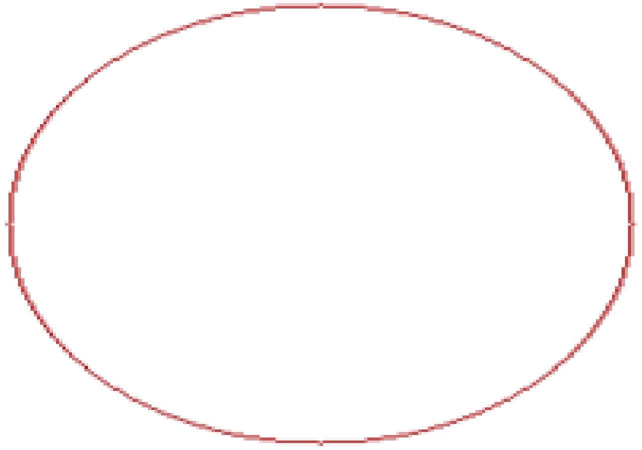
Plot (x + h, y +k); Plot (-x + h, -y + k); Plot (y + h, x + k); Plot (-y + h, -x + k); Plot (-y + h, x + k); Plot (y + h, -x + k); Plot (-x + h, y + k); Plot (x + h, -y + k)

Defining a Circle using Polynomial Method

Algorithm

- **Step 5:** Increment x = x + i
- **Step 6:** Go to step 2.
- **Step 7**: END

- **Example:** h = 200, k = 200, r = 100;

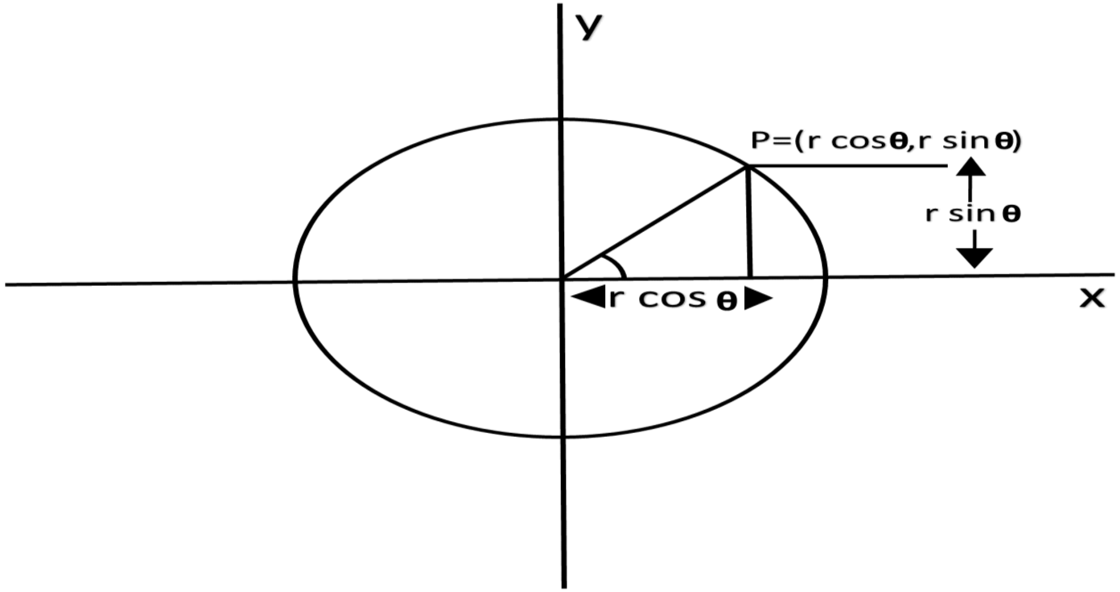


Defining a Circle using Polar Co-ordinates

- The second method of defining a circle makes use of polar coordinates as shown in fig:

$$x = r \cos \theta \qquad y = r \sin \theta$$

- Where θ=current angle, r = circle radius, x = x coordinate, y = y coordinate
- By this method, θ is stepped from 0 to ∏/4 & each value of x & y is calculated.



Defining a Circle using Polar Co-ordinates

Algorithm

- **Step1:** Set the initial variables:

  r = circle radius; (h, k) = coordinates of the circle center

  i = step size; θ_end = ∏/4; θ=0

- **Step2:** If $\theta > \theta_{end}$ then stop.
- **Step 3:** Compute x = r * cos(θ)   y = r*sin(θ)
- **Step4:** Plot the eight points, found by symmetry i.e., the center (h, k), at the current (x, y) coordinates.

  Plot (x + h, y +k); Plot (-x + h, -y + k); Plot (y + h, x + k);

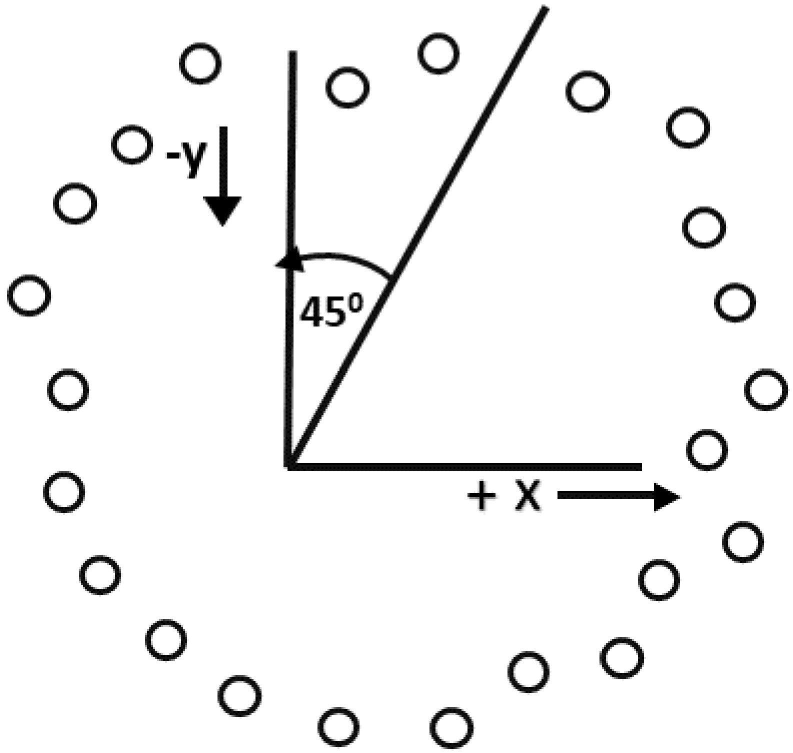
  Plot (-y + h, -x + k); Plot (-y + h, x + k); Plot (y + h, -x + k)

  Plot (-x + h, y + k); Plot (x + h, -y + k)

- **Step5:** Increment θ = θ+i
- **Step6:** Go to step 2.

Bresenham's Circle Algorithm

- Scan-Converting a circle using Bresenham's algorithm works as follows: Points are generated from 90° to 45°, moves will be made only in the +x & -y directions as shown in fig:



Bresenham's Circle Algorithm

- We cannot display a continuous arc on the raster display. Instead, we have to choose the nearest pixel position to complete the arc.
- The best approximation of the true circle will be described by those pixels in the raster that falls the least distance from the true circle. We want to generate the points from 90° to 45°.
- The decision at each step is whether to choose the pixel directly above the current pixel or the pixel which is above and to the left (8-way stepping).
- Let us assume we have a point **p (x, y)** on the boundary of the circle and with **r** radius satisfying the equation **f (x, y) = 0**
- We assume, The distance between point $\mathbf{P_3}$ and circle boundary = $\mathbf{d_1}$ and the distance between point $\mathbf{P_2}$ and circle boundary = $\mathbf{d_2}$

Bresenham's Circle Algorithm

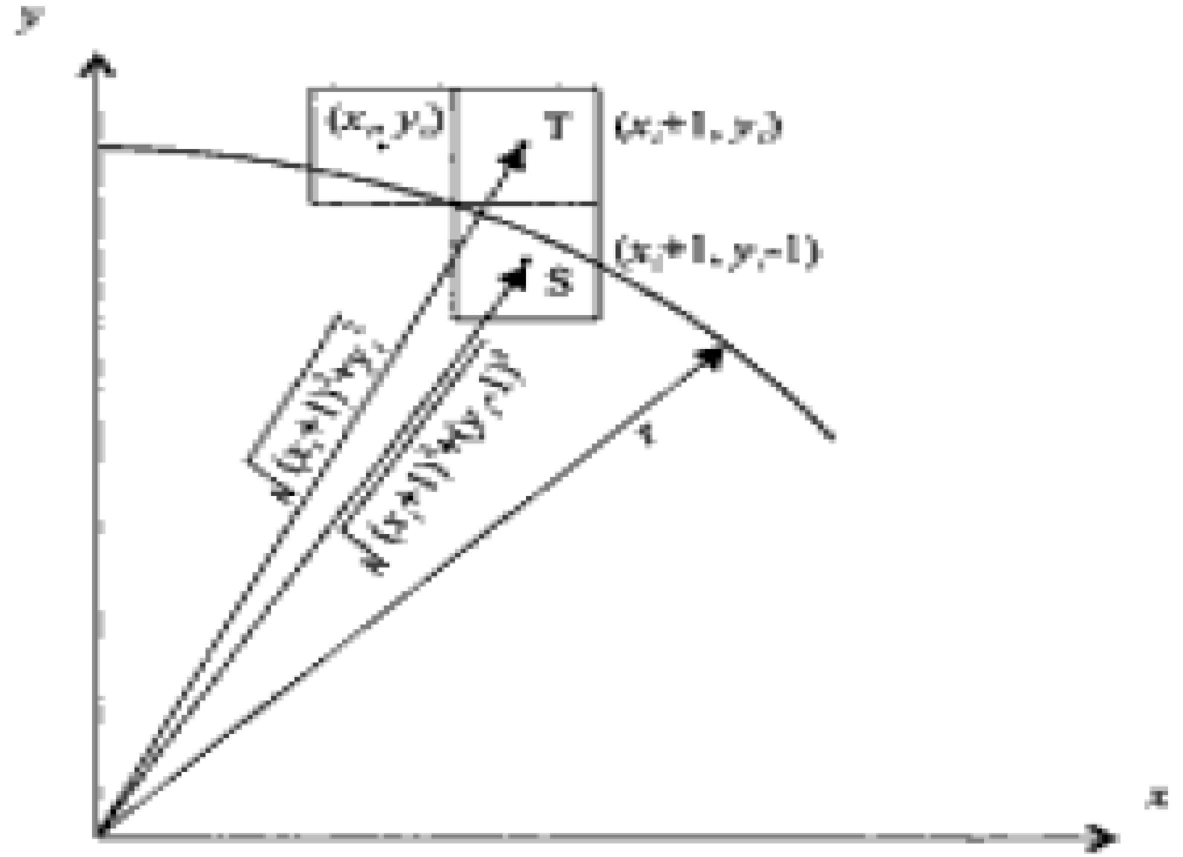
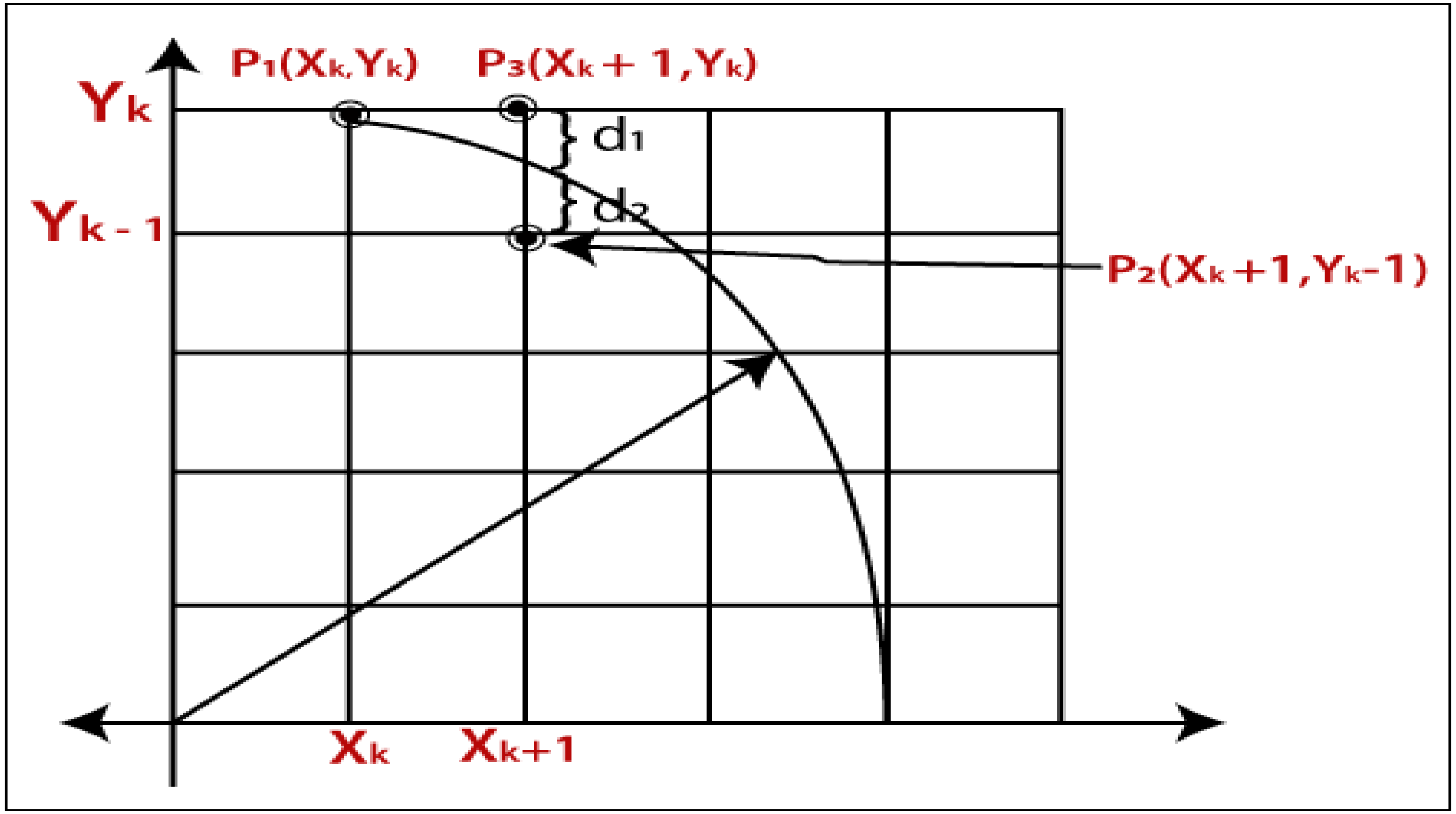


Fig. 3-8 Choosing pixels in Bresenham's circle algorithm.

Bresenham's Circle Algorithm



Bresenham's Circle Algorithm

- Now, if we select point $\mathbf{P_3}$ then circle equation will be-

  $$d_1 = (x_k + 1)^2 + (y_k)^2 - r^2$$

  {Value is +ve, because the point is outside the circle}
- If we select point $\mathbf{P_2}$ then circle equation will be-

  $$d_2 = (x_k + 1)^2 + (y_k - 1)^2 - r^2$$

  {Value is -ve, because the point is inside the circle}
- Now, we will calculate the decision parameter $\mathbf{(d_k) = d_1 + d_2}$

$$\begin{aligned} \mathbf{d_k} &= \mathbf{(x_k + 1)^2 + (y_k)^2 - r^2 + (x_k + 1)^2 + (y_k - 1)^2 - r^2} \\ &= \mathbf{2(x_k + 1)^2 + (y_k)^2 + (y_k - 1)^2 - 2r^2} \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots \mathbf{(1)} \end{aligned}$$

- If ($d_k < 0$) then Point P_3 is closer to circle boundary, and the final coordinates are- $\mathbf{(x_{k+1}, y_k) = (x_k + 1, y_k)}$
- If ($d_k >= 0$) then Point P_2 is closer to circle boundary, and the final coordinates are- $\mathbf{(x_{k+1}, y_k) = (x_k + 1, y_k - 1)}$

Bresenham's Circle Algorithm

- Now, we will find the next decision parameter $(\mathbf{d_{k+1}})$

$$(d_{k+1}) = 2(x_{k+1} + 1)^2 + (y_{k+1})^2 + (y_{k+1} - 1)^2 - 2r^2 \qquad \ldots\ldots\ldots\ldots(2)$$

- Now, we find the difference between decision parameter equation (2) – equation (1)

$$(d_{k+1}) - (d_k) = 2(x_{k+1}+1)^2 + (y_{k+1})^2 + (y_{k+1} - 1)^2 - 2r^2 - 2(x_k + 1)^2 + (y_k)^2 + (y_k - 1)^2 - 2r^2$$

$$(d_{k+1}) = d_k + 4x_k + 2(y_{k+1}{}^2 - y_k{}^2) - 2(y_{k+1} - y_k) + 6$$

- Now, we check following two conditions for decision parameter-
- **Condition 1:** If($\mathbf{d_k < 0}$) then $\mathbf{y_{k+1} = y_k}$ **(We select point P_3)**

$$(d_{k+1}) = d_k + 4x_k + 2(y_k{}^2 - y_k{}^2) - 2(y_k - y_k) + 6 = d_k + 4x_k + 6$$

- **Condition 2:** If($\mathbf{d_k >= 0}$) then $\mathbf{y_{k+1} = y_k - 1}$ **(We select point P_2)**

$$(d_{k+1}) = d_k + 4x_k + 2\{(y_k - 1)^2 - y_k{}^2\} - 2\{(y_k - 1) - y_k\} + 6$$
$$= d_k + 4(x_k - y_k) + 10$$

Bresenham's Circle Algorithm

- If it is assumed that the circle is centered at the origin, then at the first step $x = 0$ & $y = r$.
- Now, we calculate initial decision parameter **(d_0)**

$$\mathbf{d_0 = d_1 + d_2}$$

$$\mathbf{d_0 = \{1^2 + r^2 - r^2\} + \{1^2 + (r - 1)^2 - r^2\}}$$

$$\mathbf{d_0 = \quad 3 - 2r}$$

- Thereafter, if $d_i<0$,then only x is incremented.

$$x_{i+1}=x_{i+1} \qquad d_{i+1}=d_i+ 4x_i+6$$

- if $d_i \geq 0$,then x & y are incremented

$$x_{i+1}=x_{i+1} \qquad y_{i+1} =y_i+ 1$$

$$d_{i+1}=d_i+ 4 (x_i-y_i)+10$$

Bresenham's Circle Algorithm

Algorithm

- **Step1:** Start Algorithm
- **Step2:** Declare p, q, x, y, r, d variables
  p, q are coordinates of the center of the circle and r is the radius of the circle
- **Step3:** Enter the value of r
- **Step4:** Calculate d = 3 - 2r
- **Step5:** Initialize x=0 and y= r
- **Step6:** Check if the whole circle is scan converted
  If (x > = y) then Stop
- **Step7:** Plot eight points by using concepts of eight-way symmetry. Current active pixel is (x, y).

putpixel (x+p, y+q); putpixel (y+p, x+q); putpixel (-y+p, x+q); putpixel (-x+p, y+q);
putpixel (-x+p, -y+q); putpixel (-y+p, -x+q); putpixel (y+p, -x+q); putpixel (x+p, -y-q)

Bresenham's Circle Algorithm

Algorithm

- **Step8:** Find location of next pixels to be scanned

  If $(d < 0)$

  then $d = d + 4x + 6$

  increment $x = x + 1$

  $y = y$

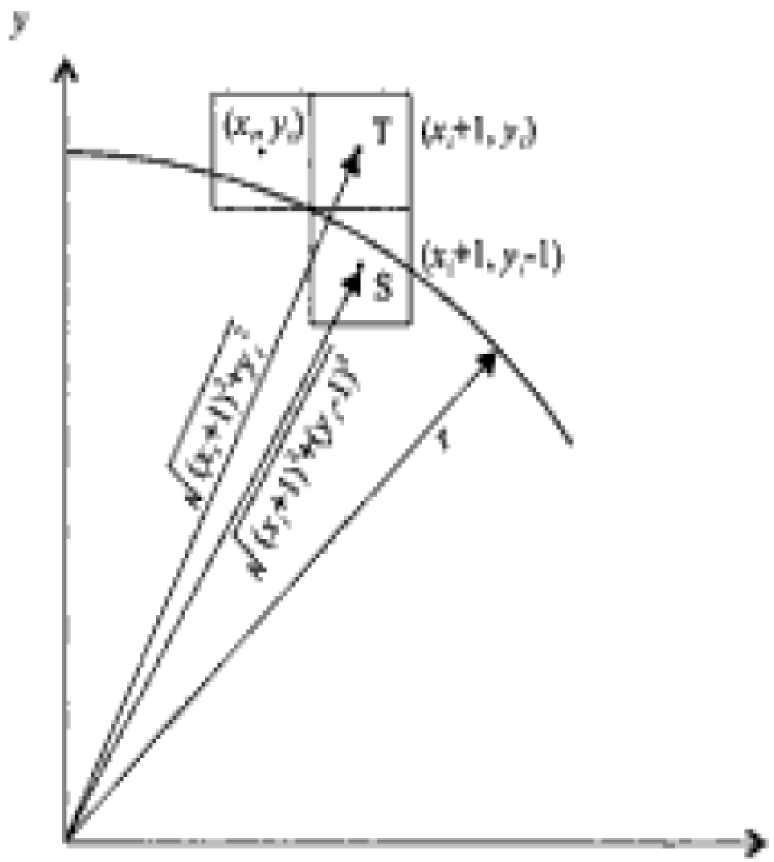
  If $d \geq 0$

  then $d = d + 4(x - y) + 10$

  increment $x = x + 1$

  decrement $y = y - 1$
- **Step9:** Go to step 6
- **Step10:** Stop Algorithm

Bresenham's Circle Algorithm



$$d_{i+1} = \begin{cases} d_i + 4x_i + 6 & \text{if } d_i < 0 \\ d_i + 4(x_i - y_i) + 10 & \text{if } d_i \geq 0 \end{cases}$$

$$d_1 = 2(0+1)^2 + r^2 + (r-1)^2 - 2r^2 = 3 - 2r$$

$$d_i = D(\mathrm{T}) + D(\mathrm{S})$$

$$d_i = 2(x_i+1)^2 + y_i^2 + (y_i - 1)^2 - 2r^2$$

$$D(\mathrm{T}) = (x_i+1)^2 + y_i^2 - r^2 \qquad D(\mathrm{S}) = (x_i+1)^2 + (y_i-1)^2 - r^2$$

Bresenham's Circle Algorithm

We can now summarize the algorithm for generating all the pixel coordinates in the 90° to 45° octant that are needed when scan-converting a circle of radius r:

```
int x = 0, y = r, d = 3 - 2r;
while (x <= y) {
   setPixel(x, y);
   if (d < 0)
      d = d + 4x + 6;
   else {
      d = d + 4(x - y) + 10;
      y--;
   }
   x++;
}
```

Bresenham's Circle Algorithm

Advantage

- The entire algorithm is based on the simple equation of Circle: X2 +Y2 = R2
- It is easy to implement

Disadvantage

- Like Mid Point Algorithm, accuracy of the generating points is an issue in this algorithm.
- This algorithm suffers when used to generate complex and high graphical images.
- There is no significant enhancement with respect to performance.

Bresenham's Circle Algorithm

Example

- **Plot 6 points of circle using Bresenham Algorithm. When radius of circle is 10 units. The circle has centre (50, 50).**

Solution

- Let r = 10 (Given)
- **Step1:** Take initial point (0, 10)

  $d = 3 - 2r = 3 - 2 * 10 = -17$

  $d < 0$, therefore $d = d + 4x + 6 = -17 + 4 (0) + 6 = -11$
- **Step2:** Plot (1, 10)

  $d = d + 4x + 6$ ($\because d < 0$) $= -11 + 4 (1) + 6 = -1$
- **Step3:** Plot (2, 10)

  $d = d + 4x + 6$ ($\because d < 0$) $= -1 + 4 \times 2 + 6 = 13$

Bresenham's Circle Algorithm

Example

- **Step4:** Plot (3, 9) d is > 0 so x = x + 1, y = y – 1

  $d = d + 4(x-y) + 10 \ (\because d > 0) = 13 + 4(3-9) + 10 = 13 + 4(-6) + 10$

  $= 23-24=-1$

- **Step5:** Plot (4, 9)

  $d = -1 + 4x + 6 = -1 + 4(4) + 6 = 21$

- **Step6:** Plot (5, 8)

  $d = d + 4(x-y) + 10 \ (\because d > 0) = 21 + 4(5-8) + 10 = 21-12 + 10 = 19$

- So P1 $(0,0) \Longrightarrow (50,50)$

  P2 $(1,10) \Longrightarrow (51,60)$

  P3 $(2,10) \Longrightarrow (52,60)$

  P4 $(3,9) \Longrightarrow (53,59)$

  P5 $(4,9) \Longrightarrow (54,59)$; P6 $(5,8) \Longrightarrow (55,58)$

Mid Point Circle Algorithm

- It is based on the following function for testing the spatial relationship between the arbitrary point (x, y) and a circle of radius r centered at the origin:

$$f(x, y) = x^2 + y^2 - r^2 \qquad \begin{bmatrix} < 0 \text{ for } (x, y) \text{ inside the circle} \\ = 0 \text{ for } (x, y) \text{ on the circle} \\ > 0 \text{ for } (x, y) \text{ outside the circle} \end{bmatrix} \text{.....equation 1}$$

- Now, consider the coordinates of the point halfway between pixel T and pixel S
- This is called midpoint ($x_{i+1}, y_i - 1/2$) and we use it to define a decision parameter:

$$P_i = f\left(x_{i+1}, y_i - \frac{1}{2}\right) = (x_{i+1})^2 + \left(y_i - \frac{1}{2}\right)^2 - r^2 \text{ ....equation 2}$$

Mid Point Circle Algorithm

- If P_i is -ve $\Rightarrow$midpoint is inside the circle and we choose pixel T
- If P_i is +ve $\Rightarrow$midpoint is outside the circle (or on the circle) and we choose pixel S.
- The decision parameter for the next step is:

$$P_{i+1}=(x_{i+1}+1)^2+(y_{i+1}-\frac{1}{2})^2-r^2 \text{............equation 3}$$

- Since $x_{i+1}=x_{i+1}$, we have

$$P_{i+1}-P_i=((x_i+1)+1)^2-(x_i+1)^2+(y_{i+1}-\frac{1}{2})^2-(y_i-\frac{1}{2})^2$$

$$=x_i^2+4+4x_i-x_i^2+1-2x_i+y_{i+1}^2+\frac{1}{4}-y_{i+1}-y_i^2-\frac{1}{4}-y_i$$

$$=2(xi+1)+1+(y_{i+1}^2-y_i^2)-(y_{i+1}-y_i)$$

$$P_{i+1}=P_i+2(x_i+1)+1+(y_{i+1}^2-y_i^2)-(y_{i+1}-y_i) \text{........................equation 4}$$

Mid Point Circle Algorithm

- If pixel T is chosen $\Longrightarrow P_i<0$ We have $y_{i+1}=y_i$
- If pixel S is chosen $\Longrightarrow P_i \geq 0$ We have $y_{i+1}=y_i-1$

$$\text{Thus,} \quad P_{i+1} = \begin{bmatrix} P_i + 2(x_i + 1) + 1, & \text{if } P_i < 0 \\ P_i + 2(x_i + 1) + 1 - 2(y_i - 1), & \text{if } P_i \geq 0 \end{bmatrix} \text{.............equation 5}$$

- We can continue to simplify this in n terms of (x_i,y_i) and get

$$P_{i+1} = \begin{bmatrix} P_i + 2x_i + 3, & \text{if } P_i < 0 \\ P_i + 2(x_i - y_i) + 5, & \text{if } P_i \geq 0 \end{bmatrix} \text{..........................equation 6}$$

- Now, initial value of P_i (0,r)from equation 2

$$P_1 = (0+1)^2 + (r - \tfrac{1}{2})^2 - r^2$$

$$= 1 + \tfrac{1}{4} - r^2 = \tfrac{5}{4} - r$$

Mid Point Circle Algorithm

We can put $\frac{5}{4} \cong 1$

$\because r$ is an integer

So, $P_1 = 1 - r$

Mid Point Circle Algorithm

The following is a description of this midpoint circle algorithm that generates the pixel coordinates in the 90° to 45° octant:

```
int x = 0, y = r, p = 1 - r;
while (x <= y) {
   setPixel(x, y);
   if (p < 0)
      p = p + 2x + 3;
   else {
      p = p + 2(x - y) + 5;
      y--;
   }
   x++;
}
```

Mid Point Circle Algorithm

It attempts to generate the points of one octant. The points for other octacts are generated using the eight symmetry property.

Given-

Center Point = (X0, Y0)

Radius = R

Step-01: Assign the starting point coordinates (X0, Y0) as-

$$X0 = 0$$

$$Y0 = R$$

Step 02: Calculate the value of initial decision parameter P0 as-

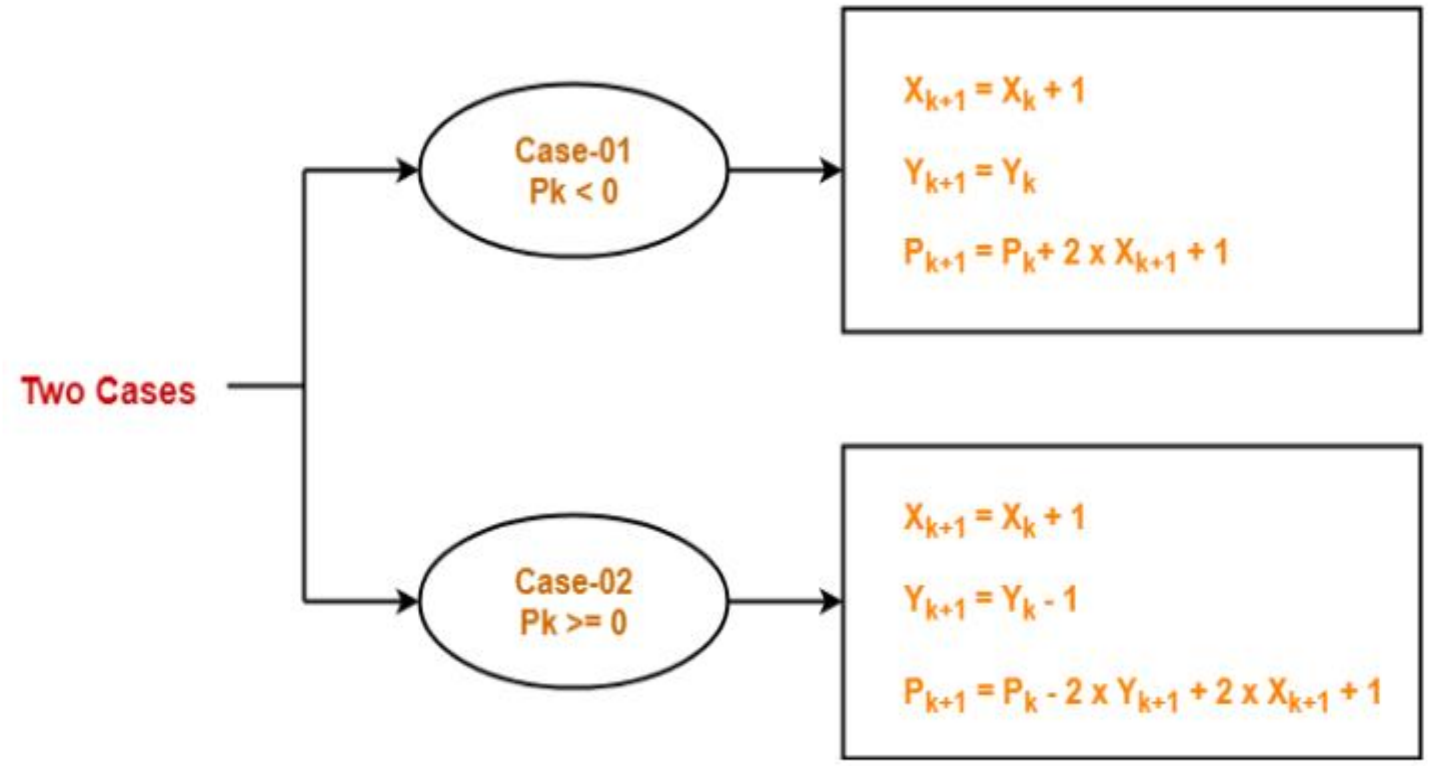
$$P0 = 1 – R$$

Step 03: Suppose the current point is (Xk, Yk) and the next point is (Xk+1, Yk+1).

Find the next point of the first octant depending on the value of decision parameter Pk.

Mid Point Circle Algorithm

Follow the below two cases-



Mid Point Circle Algorithm

Step-04: If the given centre point (X0, Y0) is not (0, 0), then do the following and plot the point-

$$Xplot = Xc + X0$$

$$Yplot = Yc + Y0$$

Here, (Xc, Yc) denotes the current value of X and Y coordinates.

Step 05: Keep repeating **Step-03 and Step-04** until

$$Xplot >= Yplot.$$

Step 06: Step-05 generates all the points for one octant. To find the points for other seven octants, follow the eight symmetry property of circle.

This is depicted by the following figure-

Mid Point Circle Algorithm

Advantage

- It is a powerful and efficient algorithm.
- The entire algorithm is based on the simple equation of circle X2 + Y2 = R2.
- It is easy to implement from the programmer’s perspective.
- This algorithm is used to generate curves on raster displays.

Disadvantage

- Accuracy of the generating points is an issue in this algorithm.
- The circle generated by this algorithm is not smooth.
- This algorithm is time consuming.

Mid Point Circle Algorithm

Example

- **Given the center point coordinates (0, 0) and radius as 10, generate all the points to form a circle.**

- **Solution**

- **Given:** Centre Coordinates of Circle (X0, Y0) = (0, 0)

  Radius of Circle (R) = 10

- **Step 01:** Assign the starting point coordinates (X0, Y0) as-

  X0 = 0

  Y0 = R = 10

- **Step 02:** Calculate the value of initial decision parameter P0 as-

  P0 = 1 – R

  P0 = 1 – 10

  P0 = -9

Mid Point Circle Algorithm

- **Step 03:** As Pinitial < 0, so case-01 is satisfied. Thus,

$$X_{k+1} = X_k + 1 = 0 + 1 = 1$$

$$Y_{k+1} = Y_k = 10$$

$$P_{k+1} = P_k + (2 * X_{k+1}) + 1 = -9 + (2 \times 1) + 1 = -6$$

- **Step 04:** This step is not applicable here as the given center point coordinates is (0, 0).
- **Step 05:** Step-03 is executed similarly until Xk+1 >= Yk+1 as follows-

P_k	P_{k+1}	(X_{k+1}, Y_{k+1})
		(0, 10)
-9	-6	(1, 10)
-6	-1	(2, 10)
-1	6	(3, 10)
6	-3	(4, 9)
-3	8	(5, 9)
8	5	(6, 8)
Algorithm Terminates These are all points for Octant-1.		

Mid Point Circle Algorithm

- Now, the points of octant-2 are obtained using the mirror effect by swapping X and Y coordinates.

Octant-1 Points	Octant-2 Points
(0, 10)	(8, 6)
(1, 10)	(9, 5)
(2, 10)	(9, 4)
(3, 10)	(10, 3)
(4, 9)	(10, 2)
(5, 9)	(10, 1)
(6, 8)	(10, 0)
These are all points for Quadrant-1.	

Mid Point Circle Algorithm

Quadrant-1 (X,Y)	Quadrant-2 (-X,Y)	Quadrant-3 (-X,-Y)	Quadrant-4 (X,-Y)
(0, 10)	(0, 10)	(0, -10)	(0, -10)
(1, 10)	(-1, 10)	(-1, -10)	(1, -10)
(2, 10)	(-2, 10)	(-2, -10)	(2, -10)
(3, 10)	(-3, 10)	(-3, -10)	(3, -10)
(4, 9)	(-4, 9)	(-4, -9)	(4, -9)
(5, 9)	(-5, 9)	(-5, -9)	(5, -9)
(6, 8)	(-6, 8)	(-6, -8)	(6, -8)
(8, 6)	(-8, 6)	(-8, -6)	(8, -6)
(9, 5)	(-9, 5)	(-9, -5)	(9, -5)
(9, 4)	(-9, 4)	(-9, -4)	(9, -4)
(10, 3)	(-10, 3)	(-10, -3)	(10, -3)
(10, 2)	(-10, 2)	(-10, -2)	(10, -2)
(10, 1)	(-10, 1)	(-10, -1)	(10, -1)
(10, 0)	(-10, 0)	(-10, 0)	(10, 0)
These are all points of the Circle			

Mid Point Circle Algorithm

Example

- **Given the center point coordinates (4, -4) and radius as 10, generate all the points to form a circle.**

Solution

- **Given:** Centre Coordinates of Circle (X0, Y0) = (4, -4)

  Radius of Circle (R) = 10
- We first calculate the points assuming the centre coordinates is (0, 0). At the end, we translate the circle
- Step-01, Step-02 and Step-03 are already completed in Problem-01.

- **Step 04:** Now, we find the values of Xplot and Yplot using the formula given in **Step-04** of the main algorithm.
- The following table shows the generation of points for Quadrant-1-
  - Xplot = Xc + X0 = 4 + X0
  - Yplot = Yc + Y0 = 4 + Y0

Mid Point Circle Algorithm

Example

- **Given the center point coordinates (4, -4) and radius as 10, generate all the points to form a circle.**

Solution

- **Step 05:** Now, we find the values of Xplot and Yplot using the formula given in **Step-04** of the main algorithm.
- The following table shows the generation of points for Quadrant-1-
  - $Xplot = Xc + X0 = 4 + X0$
  - $Yplot = Yc + Y0 = 4 + Y0$

Mid Point Circle Algorithm

Quadrant-1 (X,Y)	Quadrant-2 (-X,Y)	Quadrant-3 (-X,-Y)	Quadrant-4 (X,-Y)
(4, 14)	(4, 14)	(4, -6)	(4, -6)
(5, 14)	(3, 14)	(3, -6)	(5, -6)
(6, 14)	(2, 14)	(2, -6)	(6, -6)
(7, 14)	(1, 14)	(1, -6)	(7, -6)
(8, 13)	(0, 13)	(0, -5)	(8, -5)
(9, 13)	(-1, 13)	(-1, -5)	(9, -5)
(10, 12)	(-2, 12)	(-2, -4)	(10, -4)
(12, 10)	(-4, 10)	(-4, -2)	(12, -2)
(13, 9)	(-5, 9)	(-5, -1)	(13, -1)
(13, 8)	(-5, 8)	(-5, 0)	(13, 0)
(14, 7)	(-6, 7)	(-6, 1)	(14, 1)
(14, 6)	(-6, 6)	(-6, 2)	(14, 2)
(14, 5)	(-6, 5)	(-6, 3)	(14, 3)
(14, 4)	(-6, 4)	(-6, 4)	(14, 4)
These are all points of the Circle.			

Mid Point Circle Algo: Pseudocode

```
#include "device.h"

void circleMidpoint (int xCenter, int yCenter, int radius)
{
  int x = 0;
  int y = radius;
  int p = 1 - radius;
  void circlePlotPoints (int, int, int, int);

  /* Plot first set of points */
  circlePlotPoints (xCenter, yCenter, x, y);

  while (x < y) {
    x++;
    if (p < 0)
      p += 2 * x + 1;
    else {
      y--;
      p += 2 * (x - y) + 1;
    }
    circlePlotPoints (xCenter, yCenter, x, y);
  }
}

void circlePlotPoints (int xCenter, int yCenter, int x, int y)
{
  setPixel (xCenter + x, yCenter + y);
  setPixel (xCenter - x, yCenter + y);
  setPixel (xCenter + x, yCenter - y);
  setPixel (xCenter - x, yCenter - y);
  setPixel (xCenter + y, yCenter + x);
  setPixel (xCenter - y, yCenter + x);
  setPixel (xCenter + y, yCenter - x);
  setPixel (xCenter - y, yCenter - x);
}
```

Bresenham vs Mid Point Circle Algorithm

Bresenham Algorithm: it is the optimized form of midpoint circle. it only deals with integers because of which it consume less time as well as the memory. This type of algorithm is efficient and accurate too because it prevents from calculation of floating point.

Midpoint Algorithm: it prevents trigonometric calculations and only use adopting integers. It checks the nearest possible integers with the help of midpoint of pixels on the circle. But it still needs to use the floating point calculations.