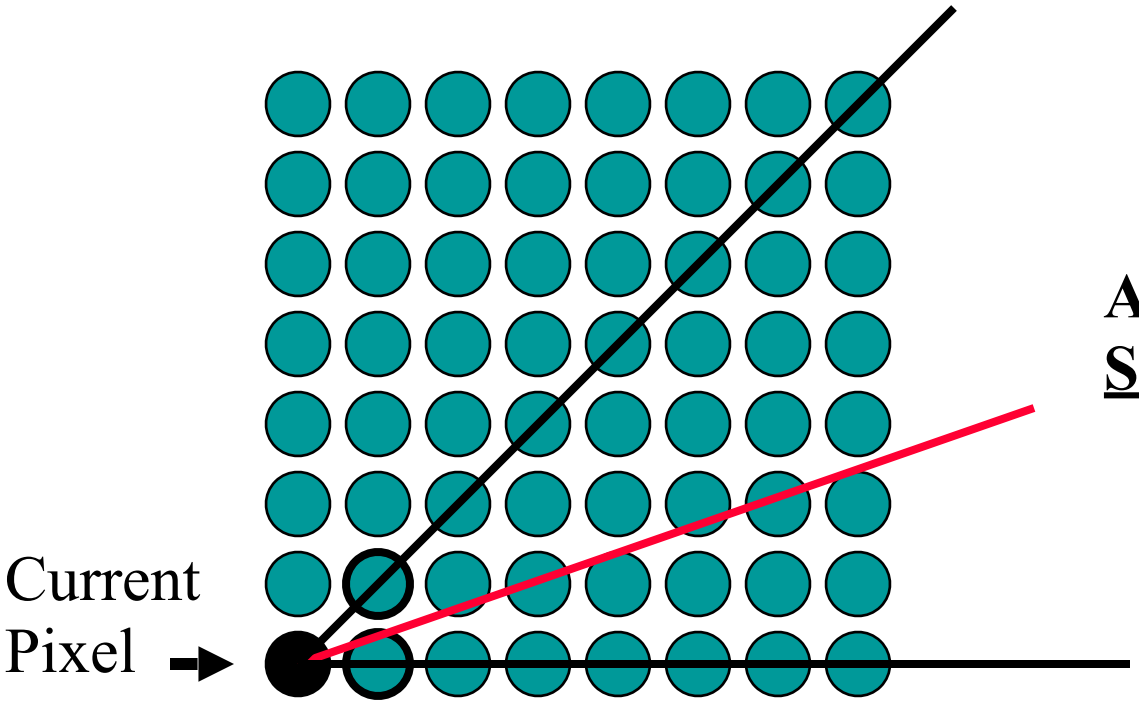
Midpoint Algorithm

Midpoint algorithm is an incremental algorithm

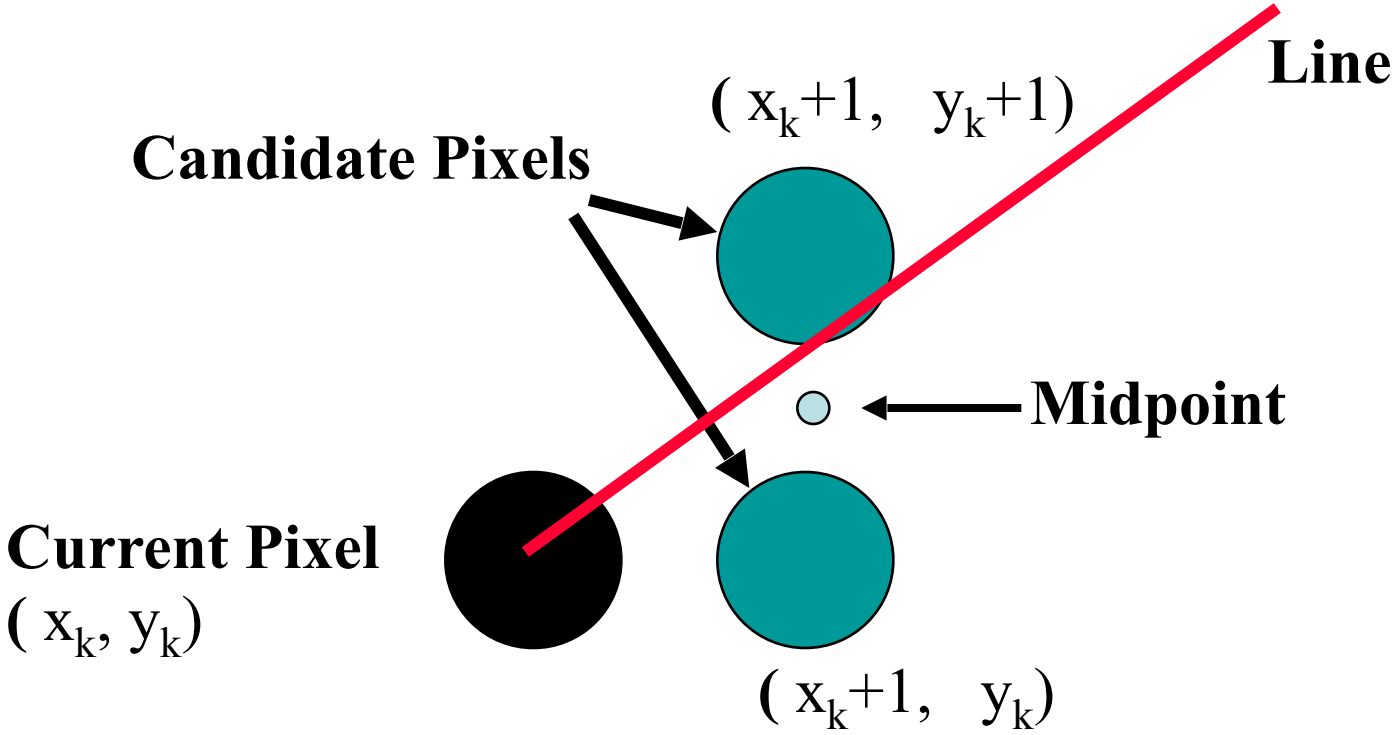


Assumption:
Slope < 1

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

$$y_{k+1} = \text{Either } y_k \text{ or } y_k + 1$$

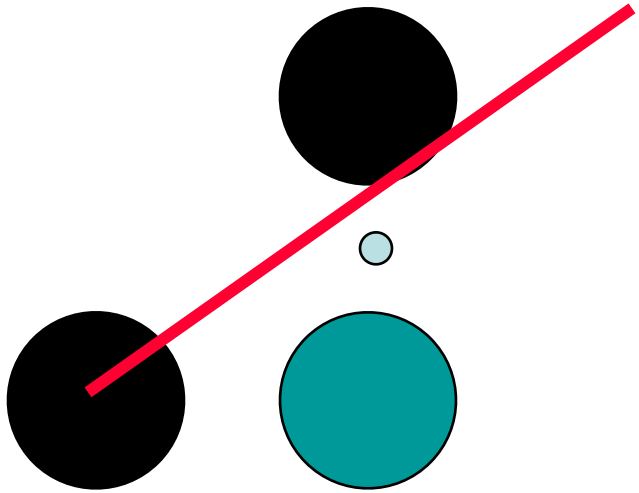
Midpoint Algorithm - Notations



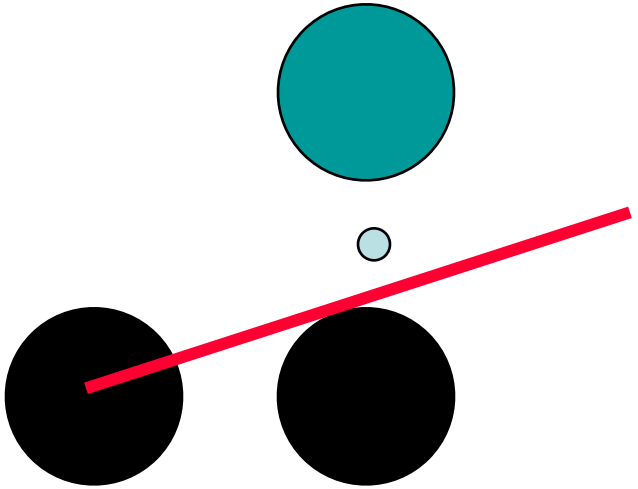
Coordinates of Midpoint = (x_k+1, y_k+(1/2))

Midpoint Algorithm:

Choice of the next pixel



Midpoint Below Line



Midpoint Above Line

- **If the midpoint is below the line, then the next pixel is (x_k+1, y_k+1).**
- **If the midpoint is above the line, then the next pixel is (x_k+1, y_k).**

Equation of a line revisited.

Equation of the line:

$$\frac{y - a_y}{b_y - a_y} = \frac{x - a_x}{b_x - a_x}$$

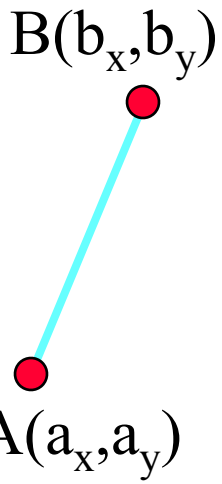
Let $w = b_x - a_x$, and $h = b_y - a_y$.

Then, $h\,(x - a_x) - w\,(y - a_y) = 0$.

(h, w , a_x , a_y are all integers).

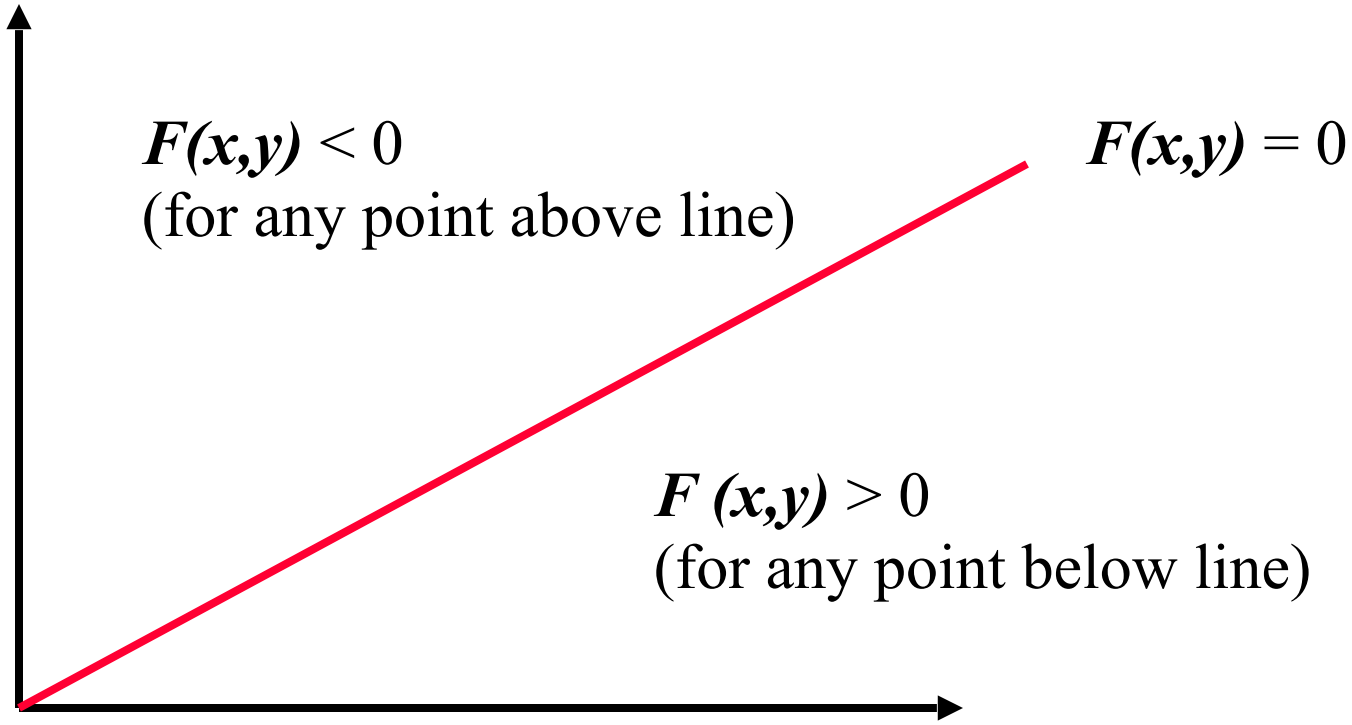
In other words, every point (x, y) on the line satisfies the equation $F(x, y) = 0$, where

$F(x, y) = h\,(x - a_x) - w\,(y - a_y)$.



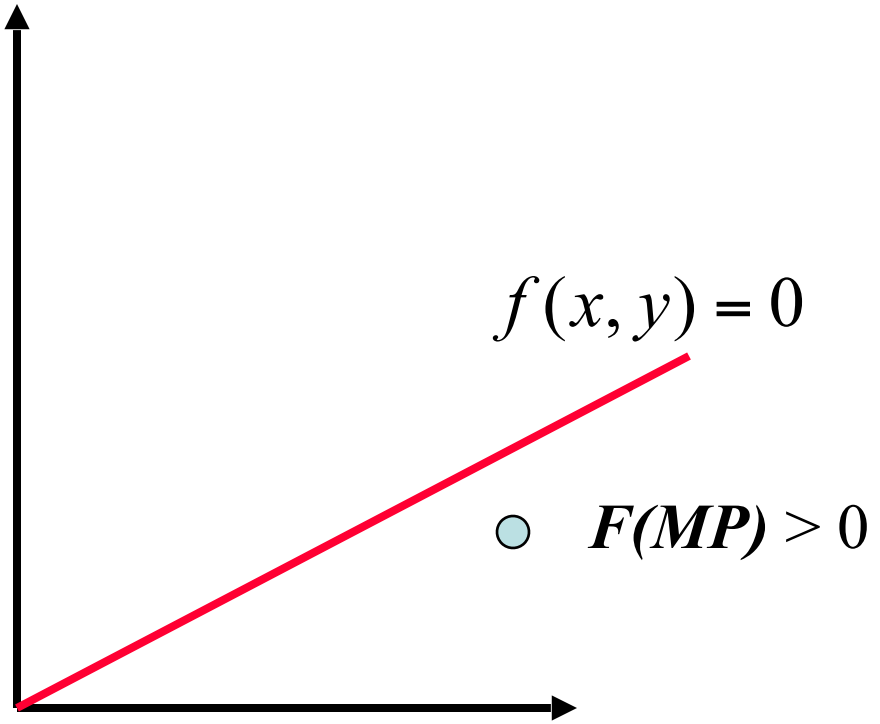
Midpoint Algorithm:

Regions below and above the line.

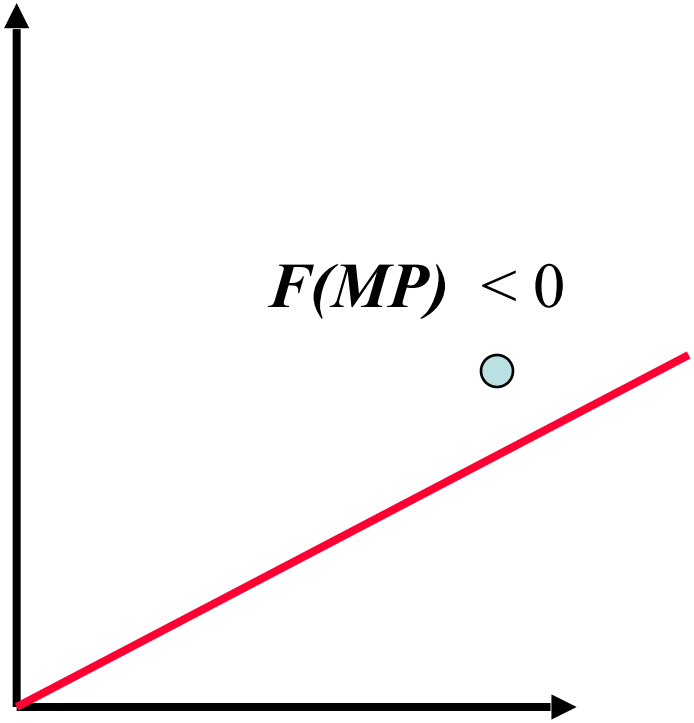


Midpoint Algorithm

Decision Criteria



Midpoint below line



Midpoint above line

Midpoint Algorithm

Decision Criteria

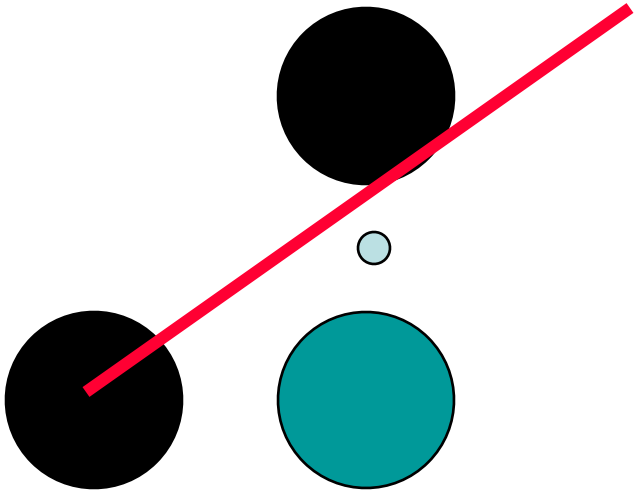
Decision Parameter

$$F(MP) = F(x_k+1, y_k+ ½) = F_k$$ (Notation)

If $F_k < 0$: The midpoint is above the line. So the next pixel is (x_k+1, y_k).

If $F_k \geq 0$: The midpoint is below or on the line. So the next pixel is (x_k+1, y_k+1).

Midpoint Algorithm – Story so far.

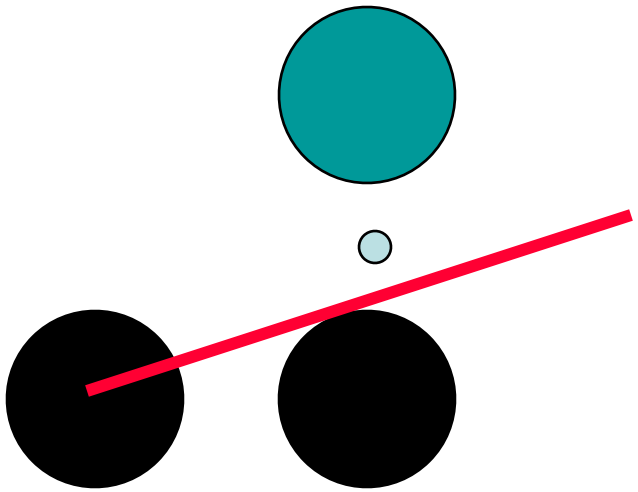


Midpoint Below Line

$$F_k > 0$$

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

Next pixel = (x_k+1, y_k+1)



Midpoint Above Line

$$F_k < 0$$

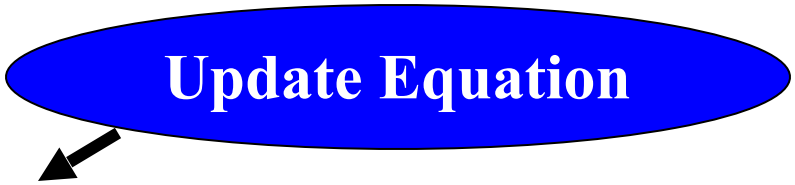
$$y_{k+1} = y_k$$

Next pixel = (x_k+1, y_k)

Midpoint Algorithm

Update Equation

$$F_k = F(x_k+1, y_k+ ½) \quad = h\,(x_k+1 - a_x) - w\,(y_k+½ - a_y)$$



But, $F_{k+1} = F_k + h - w\,(y_{k+1} - y_k)$. **(Refer notes)**

So,

$F_k < 0$: $y_{k+1} = y_k$. **Hence,** $F_{k+1} = F_k + h$.

$F_k \geq 0$: $y_{k+1} = y_k+1$. **Hence,** $F_{k+1} = F_k + h - w$.

$F_0 = h - w/2$.

Midpoint Algorithm

```
int h = by-ay;
int w = bx-ax;
float F=h-w/2;
int x=ax,  y=ay;
for (x=ax; x<=bx;  x++){
   setPixel(x, y);

   if(F < 0)
       F+ = h;
   else{
       F+ = h-w;
       y++;
   }
}
```

Bresenham's Algorithm

```
int h = by-ay;
int w = bx-ax;
int F=2*h-w;
int x=ax,  y=ay;
for (x=ax; x<=bx;  x++){
   setPixel(x, y);

   if(F < 0)
          F+ = 2*h;
   else{
          F+ = 2*(h-w);
          y++;
     }
}
```

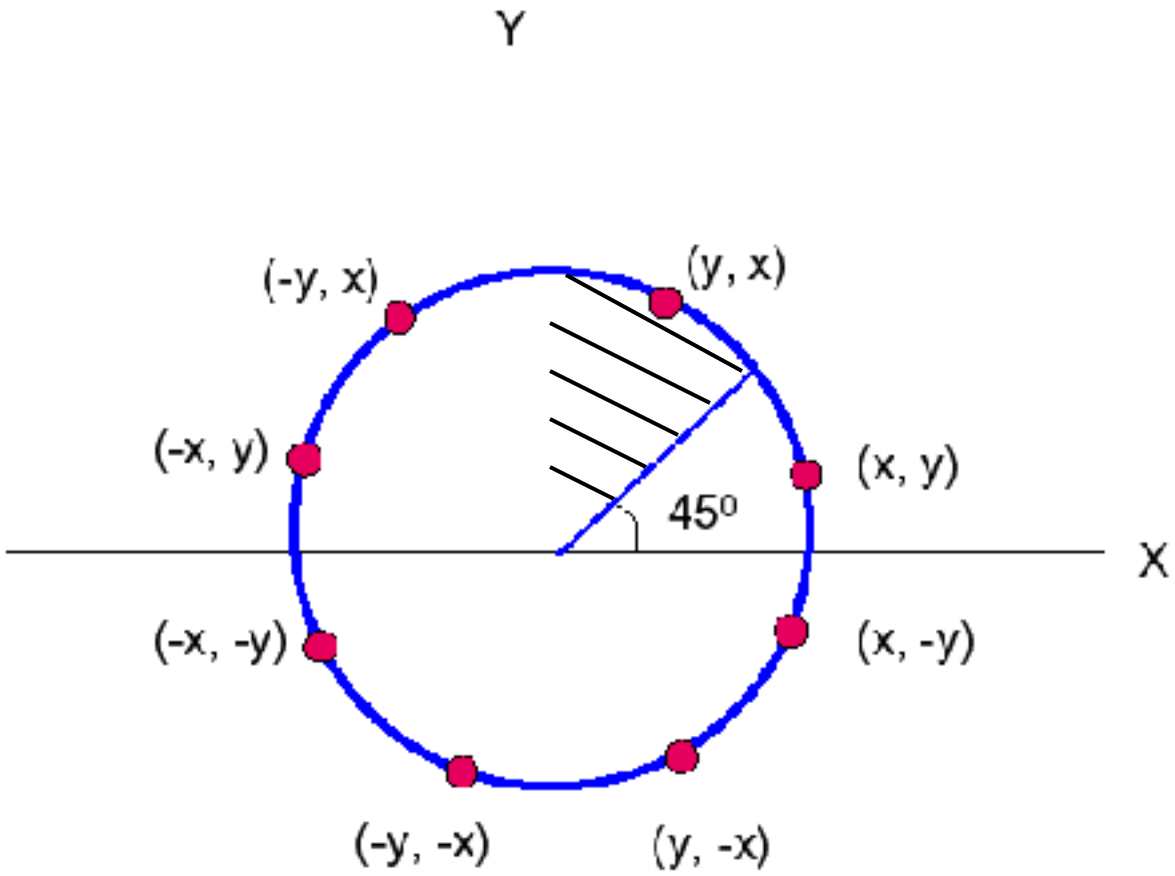
Circle Drawing Algorithms

Midpoint Circle Drawing Algorithm

- To determine the closest pixel position to the specified circle path at each step.
- For given radius r and screen center position (x_c, y_c), calculate pixel positions around a circle path centered at the coodinate origin (0,0).
- Then, move each calculated position (x, y) to its proper screen position 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 gradient varies from 0 to -1.

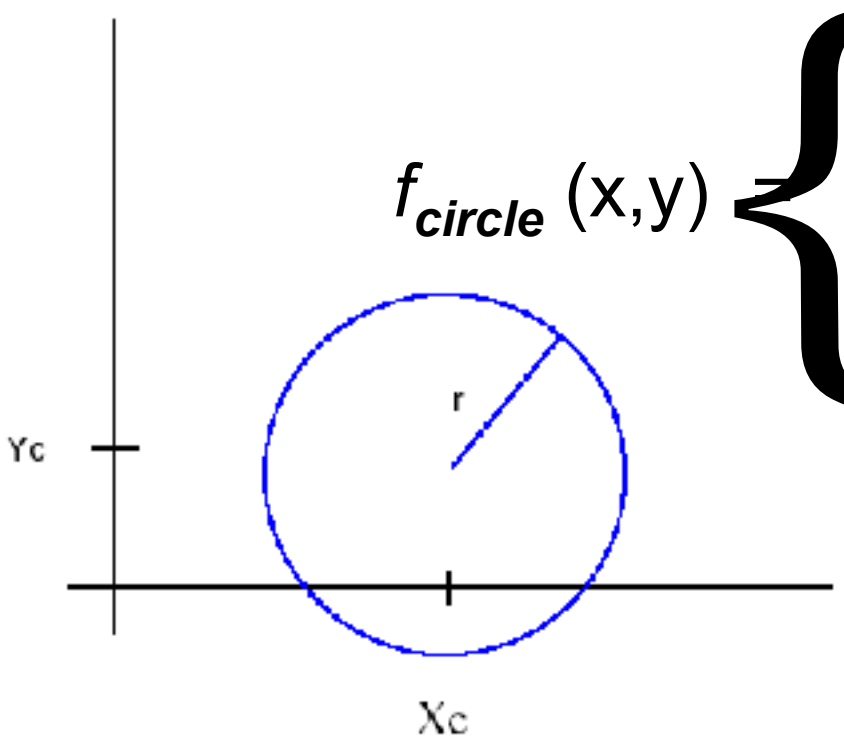
Midpoint Circle Drawing Algorithm

- 8 segments of octants for a circle:



Midpoint Circle Drawing Algorithm

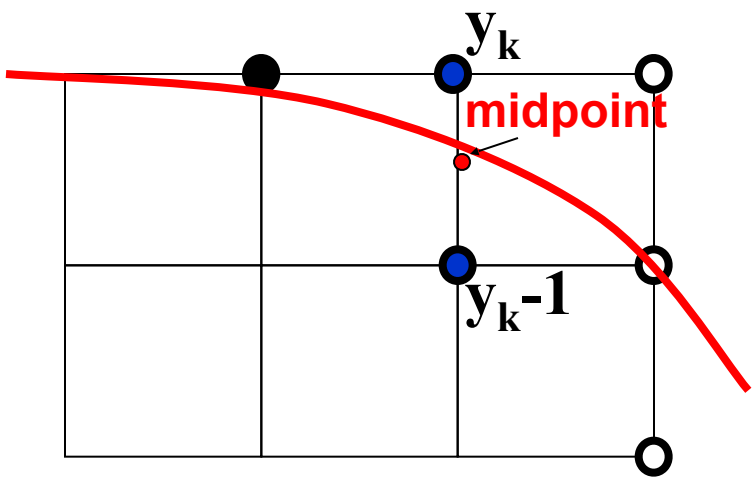
- Circle function: $f_{circle}(x,y) = x^2 + y^2 - r^2$

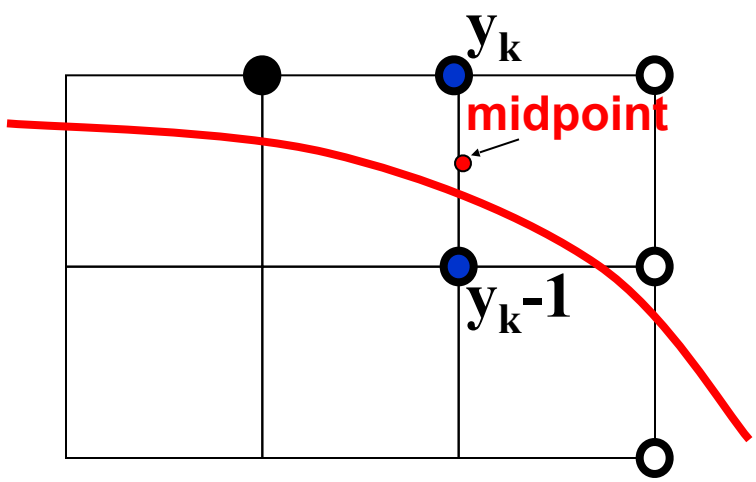


$f_{circle}(x,y) =$

- > 0, (x,y) outside the circle
- < 0, (x,y) inside the circle
- $= 0$, (x,y) is on the circle boundary

Midpoint Circle Drawing Algorithm





$F_k < 0$

$y_{k+1} = y_k$

Next pixel = (x_k+1, y_k)

$F_k >= 0$

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

Next pixel = (x_k+1, y_k-1)

Midpoint Circle Drawing Algorithm

We know $x_{k+1} = x_k + 1$,

$$F_k = F(x_k + 1, y_k - \tfrac{1}{2})$$

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

$$F_{k+1} = F(x_k + 1, y_k - \tfrac{1}{2})$$

$$F_{k+1} = (x_k + 2)^2 + (y_{k+1} - \tfrac{1}{2})^2 - r^2 \qquad \text{-------- (2)}$$

(2) – (1)

$$F_{k+1} = F_k + 2(x_k + 1) + (y^2_{k+1} - y^2_k) - (y_{k+1} - y_k) + 1$$

If $F_k < 0$, $\quad F_{k+1} = F_k + 2x_{k+1} + 1$

If $F_k >= 0$, $\quad F_{k+1} = F_k + 2x_{k+1} + 1 - 2y_{k+1}$

Midpoint Circle Drawing Algorithm

For the initial point, $(x_0, y_{0)} = (0, r)$

$$f_0 = f_{circle}(1, r-\tfrac{1}{2})$$

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

$$= \frac{5}{4} - r$$

$$\approx 1 - r$$

Midpoint Circle Drawing Algorithm

Example:

Given a circle radius = 10, determine the circle octant in the first quadrant from x=0 to x=y.

Solution:

$$f_0 = \frac{5}{4} - r$$

$$= \frac{5}{4} - 10$$

$$= -8.75$$

$$\approx -9$$

Midpoint Circle Drawing Algorithm

Initial (x_0, y_0) = (1,10)

Decision parameters are: $2x_0 = 2$, $2y_0 = 20$

k	F_k	x	y	$2x_{k+1}$	$2y_{k+1}$
0	-9	1	10	2	20
1	-9+2+1=-6	2	10	4	20
2	-6+4+1=-1	3	10	6	20
3	-1+6+1=6	4	9	8	18
4	6+8+1-18=-3	5	9	10	18
5	-3+10+1=8	6	8	12	16
6	8+12+1-16=5	7	7	14	14

Midpoint Circle Drawing Algorithm

```
void circleMidpoint (int xCenter, int yCenter, int radius)
{
        int x = 0;
        Int y = radius;
        int f = 1 – radius;

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

Midpoint Circle Drawing Algorithm

```
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);
}
```