Rasterization

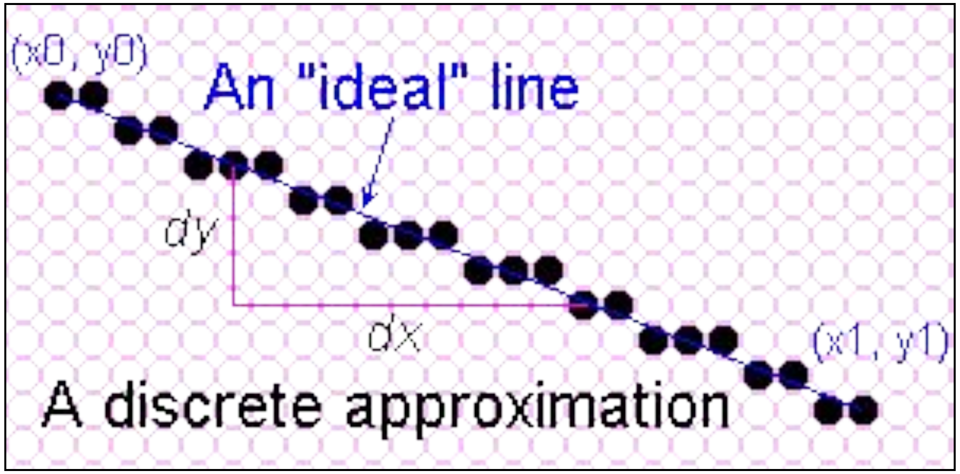
(Scan Conversion)

- a background process in the pipeline, one of the most basic problems
- after primitive assembling and clipping, primitives are still described with vertices:
  - a line is defined with 2 endpoints
  - a polygon is defined with an ordered vertex list
- for a raster-scan system, the pixels that are on or inside a primitive needs to be determined
- this task is figuring out which pixels to draw or fill on the screen

Line Drawing Algorithms

- we must "sample" a line at discrete positions
- idea: a line is sampled at unit intervals in one coordinate and the corresponding integer values nearest the line path are determined for the other coordinate

Towards the Ideal Line



- * An ideal line,
  - – Must appear straight and continuous
    - Only possible with axis-aligned lines and lines having 45° with the axis
  - – Must have uniform density and intensity
- * Must be drawn very quickly
- * Two algorithms: Digital Differential Analyser (DDA) Algorithm and Bresenham Algorithm

Using Cartesian Slope-Intercept Equation

* Before elaborating algorithms try to represent the lines

* $y = mx + b$

m: slope of the line

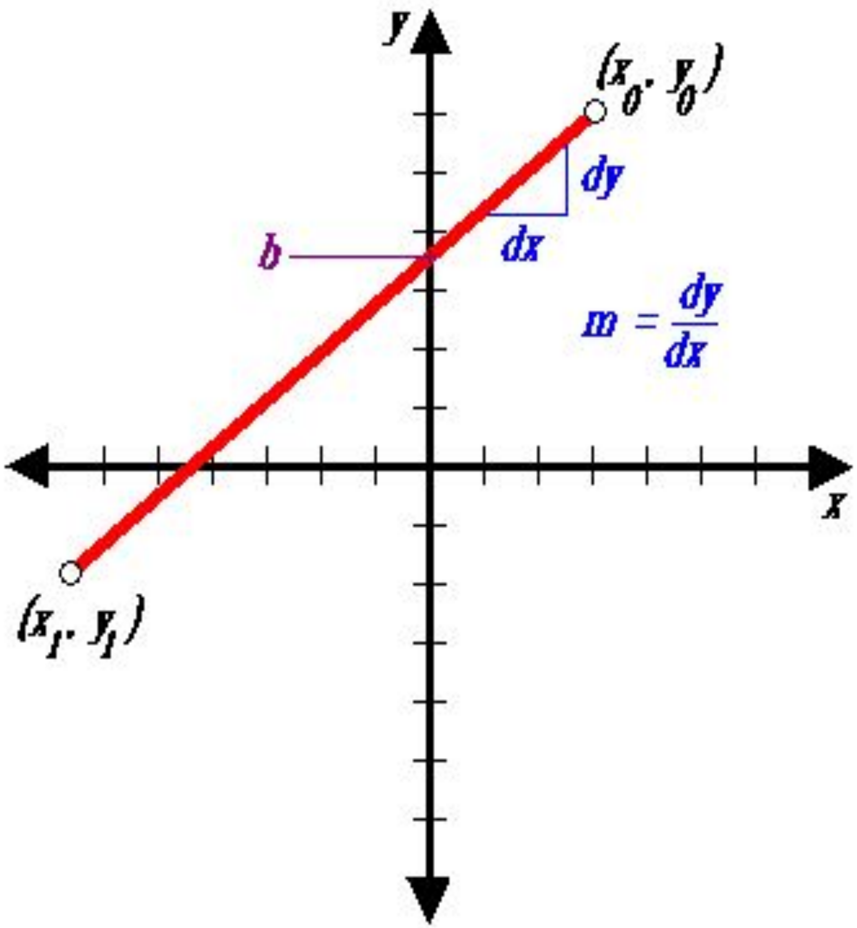
b: y intercept

$m = (y_1 - y_0) / (x_1 - x_0)$

$b = y_0 - m.\ x_0$

* y and x intervals

$dy = m.dx$



DDA Algorithm

A line is sampled at unit intervals in one coordinate and the corresponding integer values nearest the line path are determined for the other coordinate.

eq.: $y = mx + b$; 2 endpoints: $(x1, y1)$ and $(x2, y2)$.
m (slope) = $(y2 – y1) / (x2 – x1)$
(x_k, y_k) is one point on the line, (x_{k+1}, y_{k+1}) is the next point.
m (slope) = $(y_{k+1} – y_k)/ (x_{k+1} – x_k)=dy/dx$

Case 1: If $m < 1$ then x coordinate tends to the Unit interval ($dx=1, dy<1$)

$x_{k+1} = x_k + 1$
$y_{k+1} = y_k + m$
Round y_{k+1} to nearest integer value, increment k by 1 for each step

Case 2: If $m > 1$ then y coordinate tends to the Unit interval ($dy=1, dx<1$)

$y_{k+1} = y_k + 1$
$x_{k+1} = x_k + 1/m$
Round x_{k+1} to nearest integer value, increment k by 1 for each step

Case 3: If $m = 1$ then x and y coordinate tend to the Unit interval ($dx=dy=1$)

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

C function for DDA

```
inline int round (const float a)  { return int (a + 0.5); }

void lineDDA (int x0, int y0, int xEnd, int yEnd)
{
    int dx = xEnd - x0,  dy = yEnd - y0,  steps,  k;
    float xIncrement, yIncrement, x = x0, y = y0;

    if (fabs (dx) > fabs (dy))
      steps = fabs (dx);
    else
      steps = fabs (dy);
    xIncrement = float (dx) / float (steps);
    yIncrement = float (dy) / float (steps);

    setPixel (round (x), round (y));
    for (k = 0; k < steps; k++) {
      x += xIncrement;
      y += yIncrement;
      setPixel (round (x), round (y));
    }
}
```

DDA Algorithm

- Simple but needs a lot of floating point arithmetic:
  - ‘round’s and 2 additions per pixel, sometimes the point position is not accurate
- Is there a simpler way?
- Can we use only integer arithmetic?

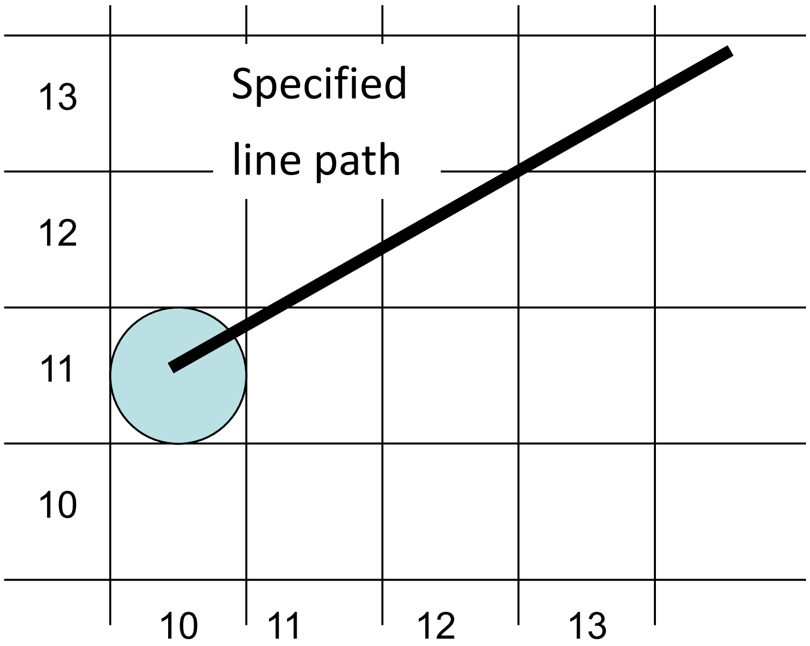
Bresenham's Line Algorithm

- Accurate and efficient
- Only incremental integer calculations

The method is described for a line segment with a positive slope less than one (0<=m<=1) and x2>x1

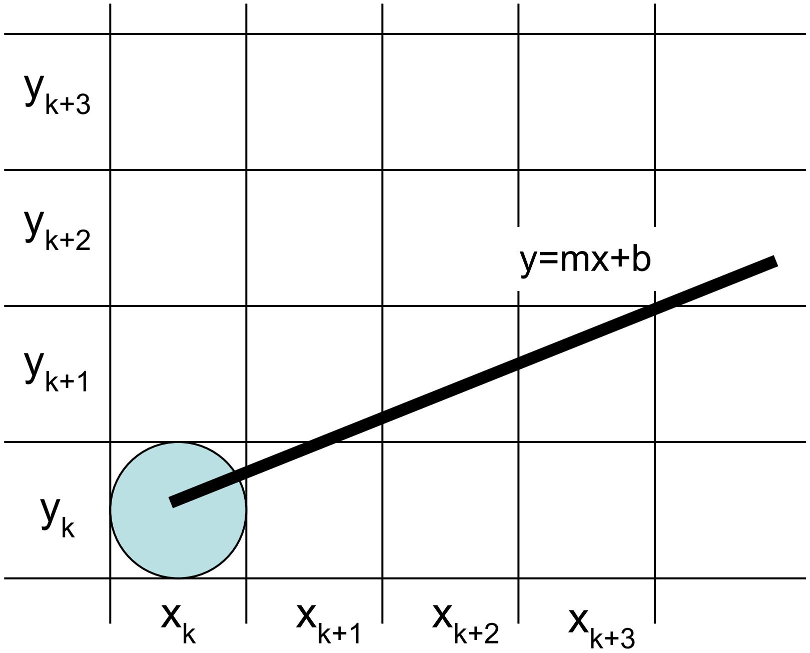
Generalizes to line segments with other slopes by considering the symmetry between the various octants (1/8) and quadrants (1/4) of the xy plane

Bresenham's Line Algorithm



- Initial coordinates (10,11)
- Decide what is the next pixel position: right or upper right
  - (11,11) or (11,12)

Bresenham's Line Algorithm

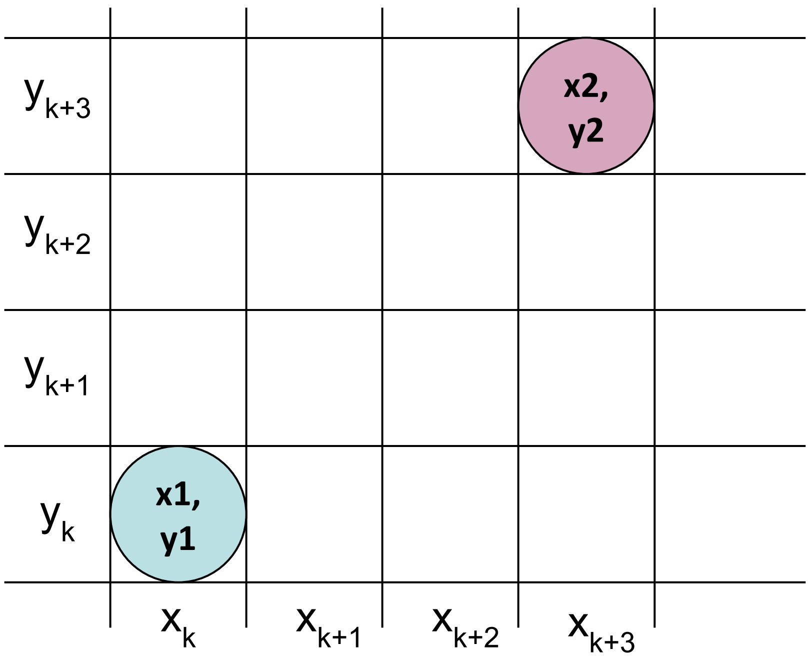


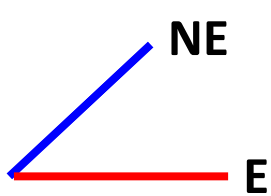
In general;

For the pixel position $x_{k+1}=x_k+1$, which one we should choose:

$$(x_{k+1},y_k) \text{ or } (x_{k+1}, y_{k+1})$$

Bresenham's Line Algorithm





Q: How we will decide?

* Starting point (x1,y1)

* Ending point (x2,y2)

* dx=x2-x1, dy=y2-y1

* decision variable d=2dy-dx (initial value, will be changed on each step)

* △E=2dy (east, right),

△NE=2(dy-dx) (north east, upper right)

These values won't be changed

* If d<=0 → choose E

x=x1+1, y=y1, d=d+△E

* If d>0 → choose NE

x=x1+1, y=y1+1, d=d+△NE

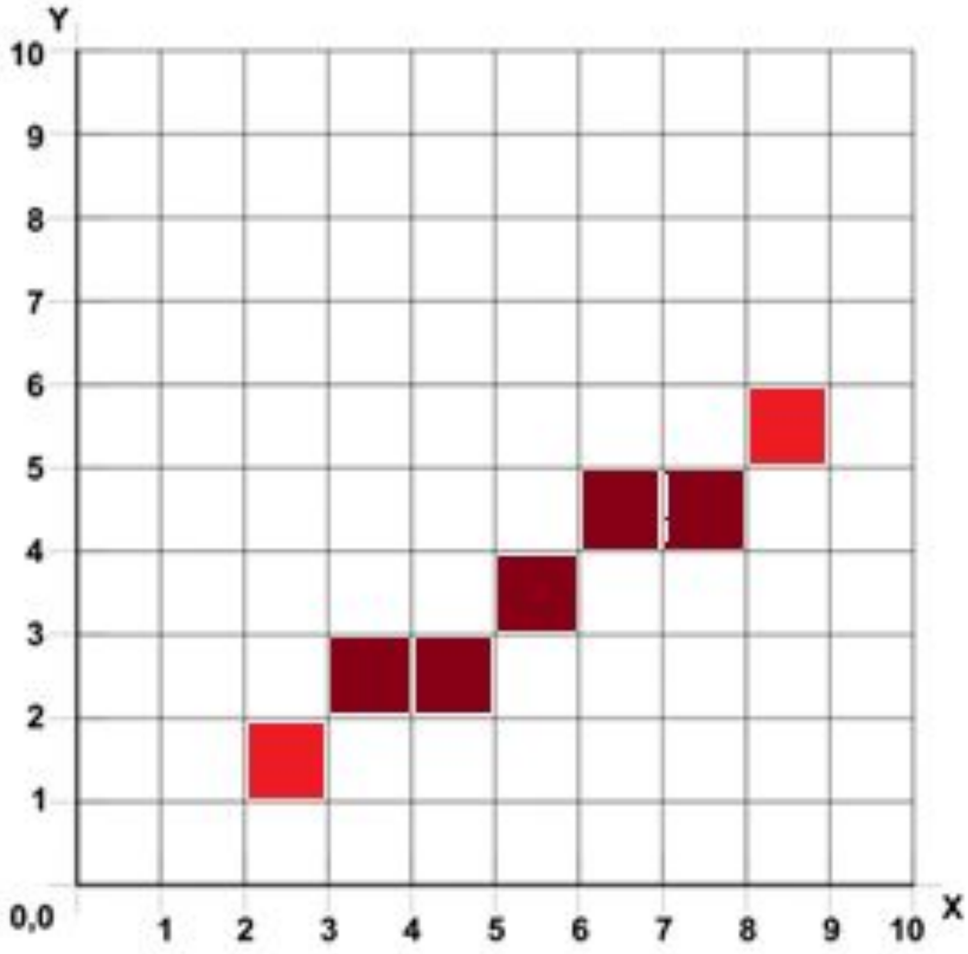
* Continue until x=x2

Bresenham's Line Algorithm

Draw a line from (2,1) to (8,5)

dx=x2-x1=8-2=6; dy=y2-y1=5-1=4

d=2dy-dx=8-6=2 (initially); △E=2dy=8; △NE=2(dy-dx)=2(4-6)=-4



x	y	d	Next Pixel
2	1	2	NE
3	2	-2	E
4	2	6	NE
5	3	2	NE
6	4	-2	E
7	4	6	NE
8	5		

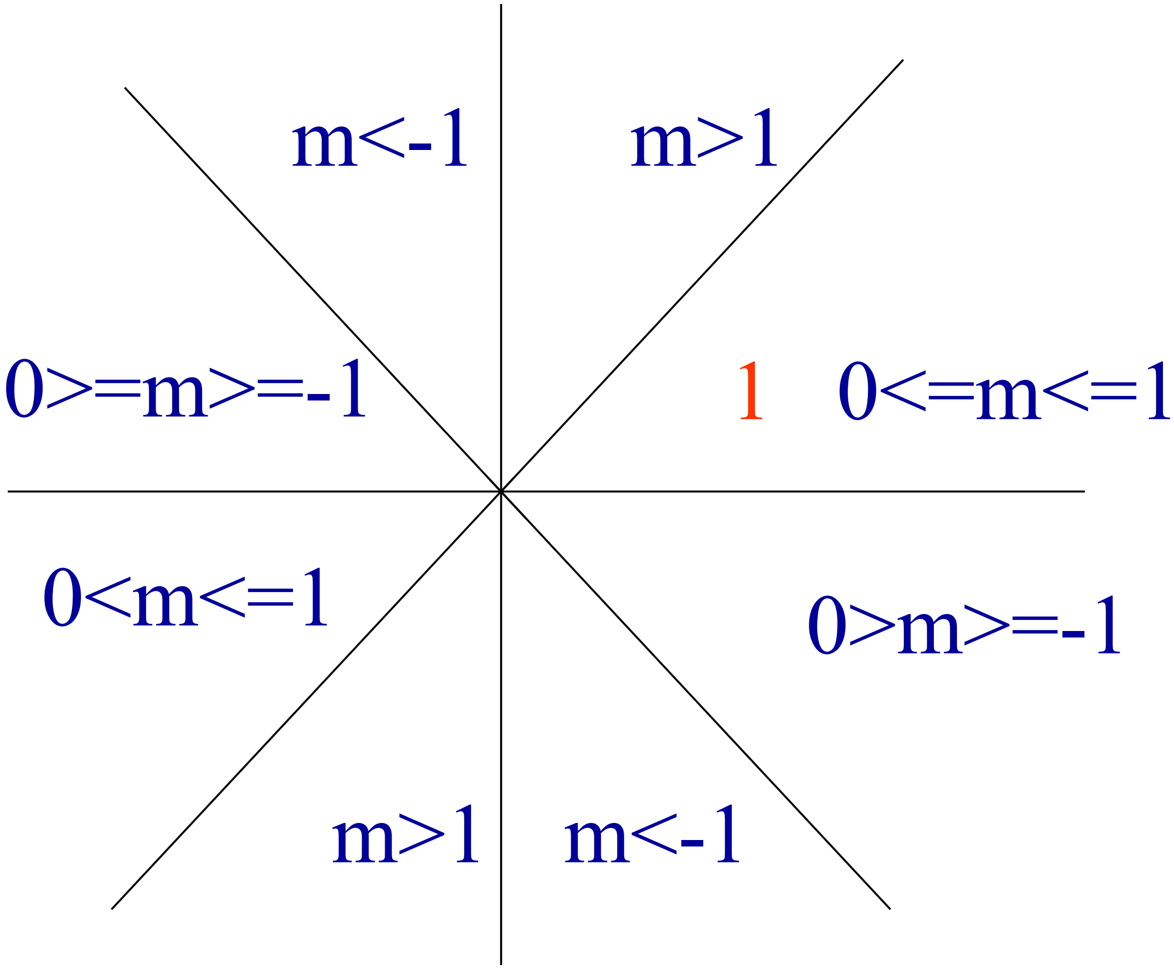
C function for Bresenham

```
/*  Bresenham line-drawing procedure for |m| < 1.0.  */
 void lineBres (int x0, int y0, int xEnd, int yEnd)
 {
   int dx = fabs (xEnd - x0),  dy = fabs(yEnd - y0);
   int x, y, p = 2 * dy - dx;
   int twoDy = 2 * dy,  twoDyMinusDx = 2 * (dy - dx);

   /* Determine which endpoint to use as start position.  */
   if (x0 > xEnd) {
     x = xEnd;    y = yEnd;   xEnd = x0;
   }
   else {
     x = x0;    y = y0;
   }
   setPixel (x, y);

   while (x < xEnd) {
     x++;
     if (p < 0)
       p += twoDy;
     else {
       y++;
       p += twoDyMinusDx;
     }
     setPixel (x, y);
   }
 }
```

Bresenham's Line Algorithm



* If x2<x1, take (x2, y2) as the starting point and (x1, y1) as the endpoint

* if m<0 get the line with a positive slope by reflecting the original line around the X-axis, perform the algorithm and reflect back around the X-axis

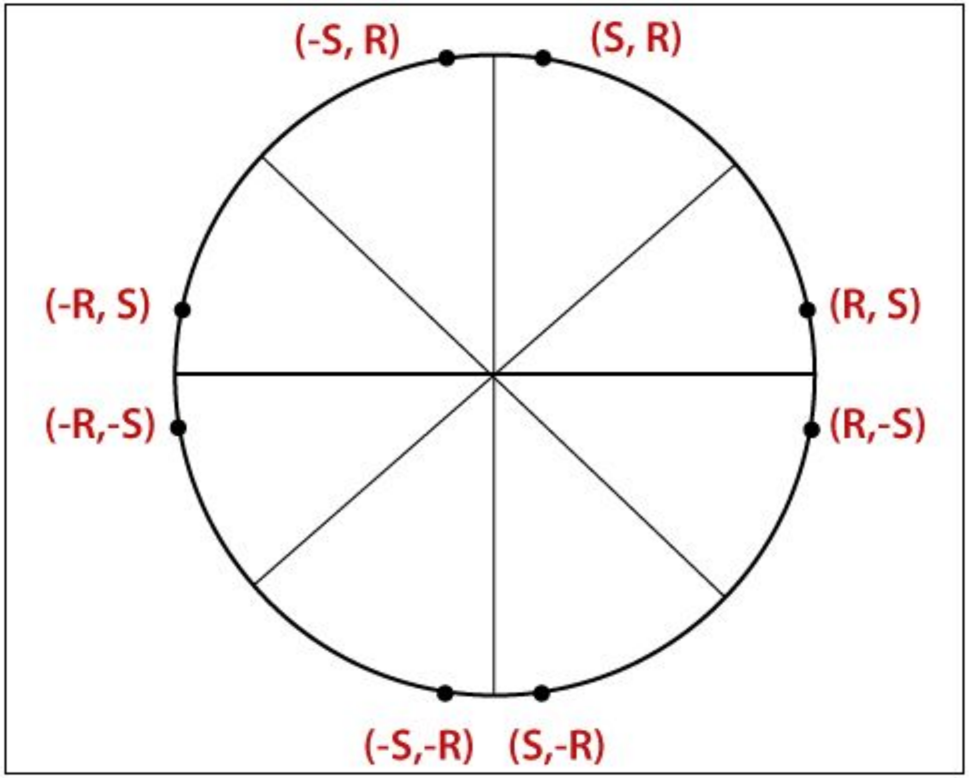
* if m> 1, exchange the x and y values, perform the algorithm and exchange the x and y values back

Scan Converting Circles

* a circle is an eight-way symmetric shape, all quadrants of a circle are the same

* can be defined as a combination of points that all points are at the same distance (or radius) from the center point

* for a point P1(R, S) we can represent the other seven points:



$P_2(R, -S)$

$P_3(-R, -S)$

$P_4(-R, S)$

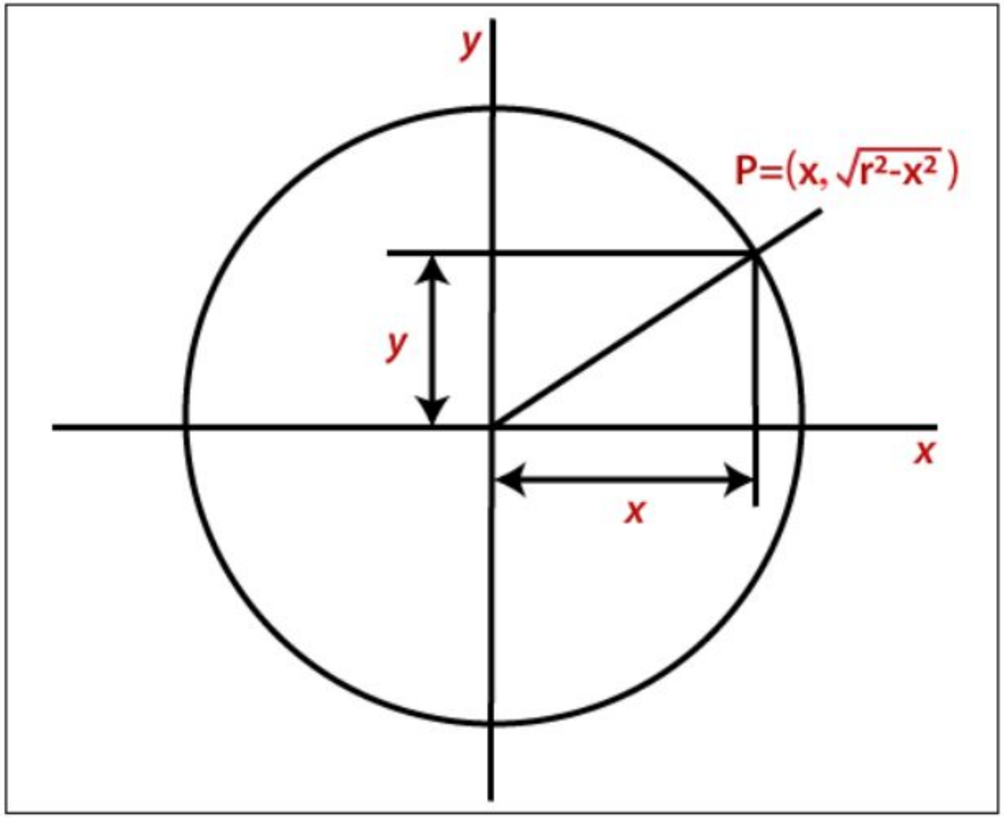
$P_5(S, R)$

$P_6(S, -R)$

$P_7(-S, -R)$

$P_8(-S, R)$

Scan Converting Circles

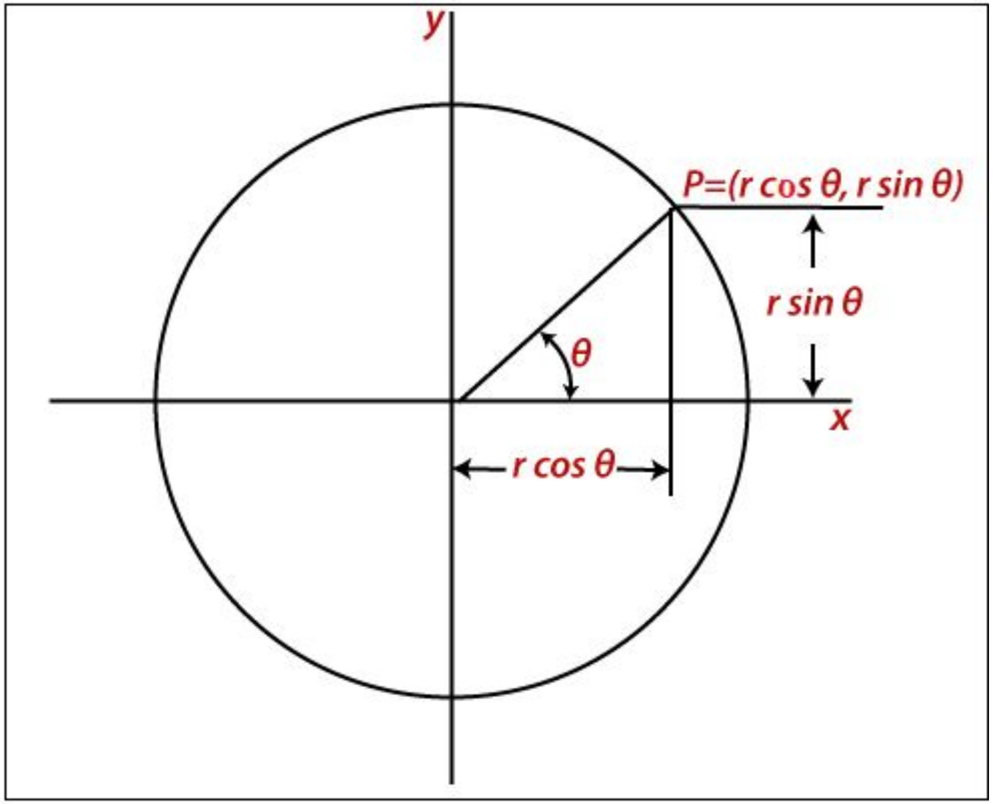


$$y^2 = r^2 - x^2$$

2 standard methods to define a circle mathematically:

- A circle with a second-order polynomial equation
- A circle with trigonometric/ polar coordinates

Scan Converting Circles



$x = r \cos \Theta$
$y = r \sin \Theta$

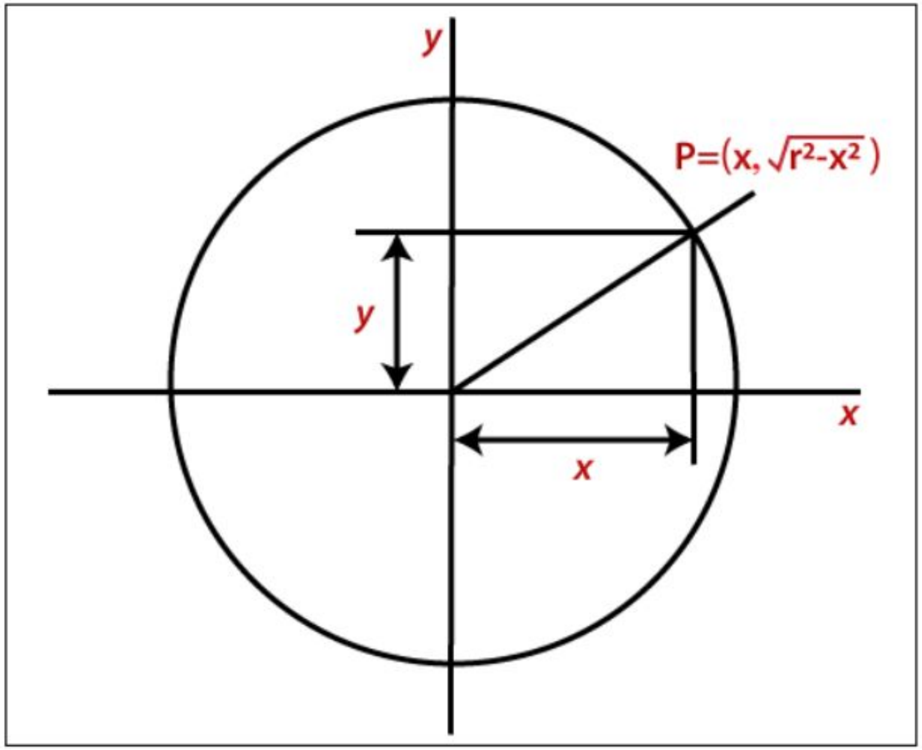
2 standard methods to define a circle mathematically:

- A circle with a second-order polynomial equation
- A circle with trigonometric/ polar coordinates

2 algorithms to draw a circle:

- Bresenham's Circle drawing Algorithm
- Midpoint Circle Drawing Algorithm

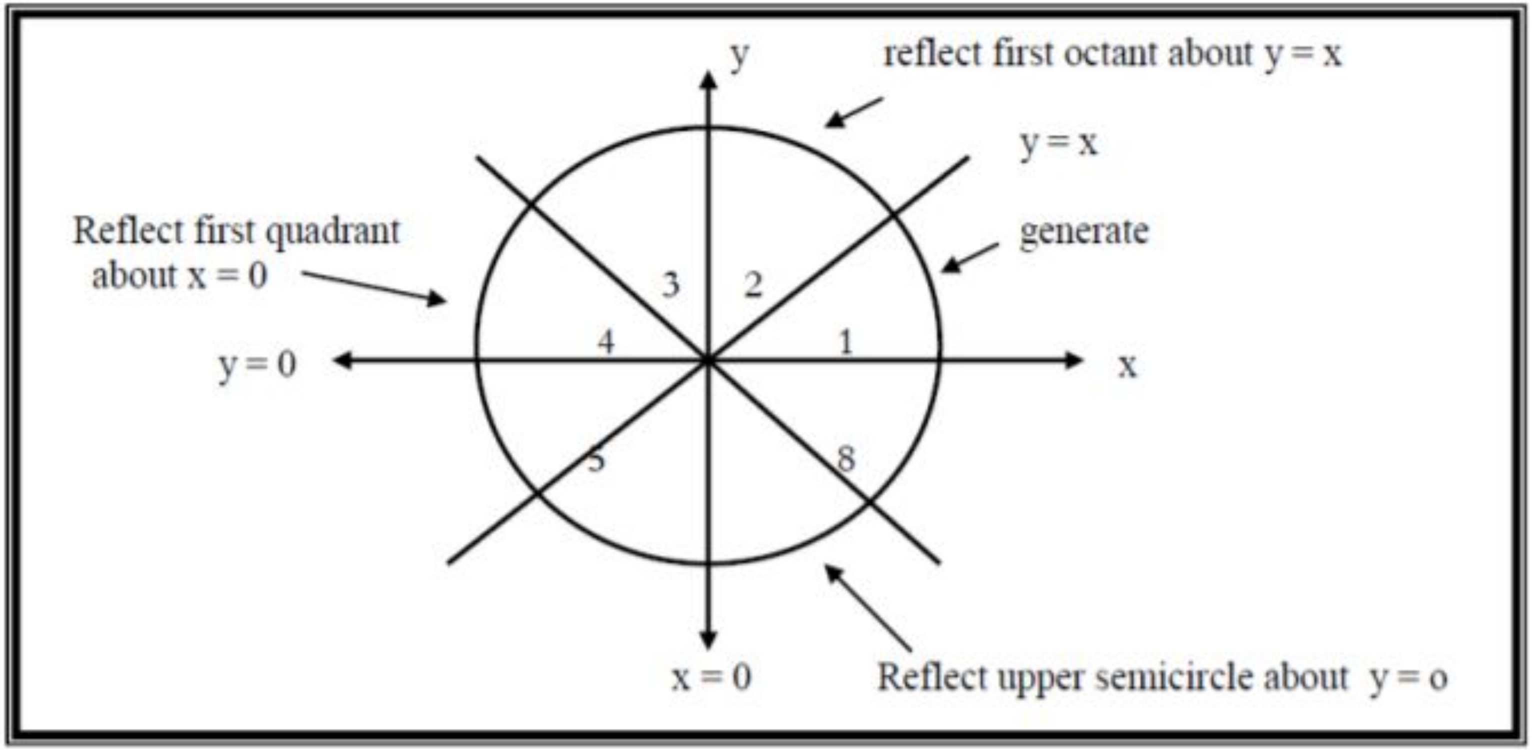
Bresenham's Circle Algorithm



a point P, select the closest pixel position to complete the arc
$f(x,y) = x^2+y^2-r^2$ (equation of the circle)

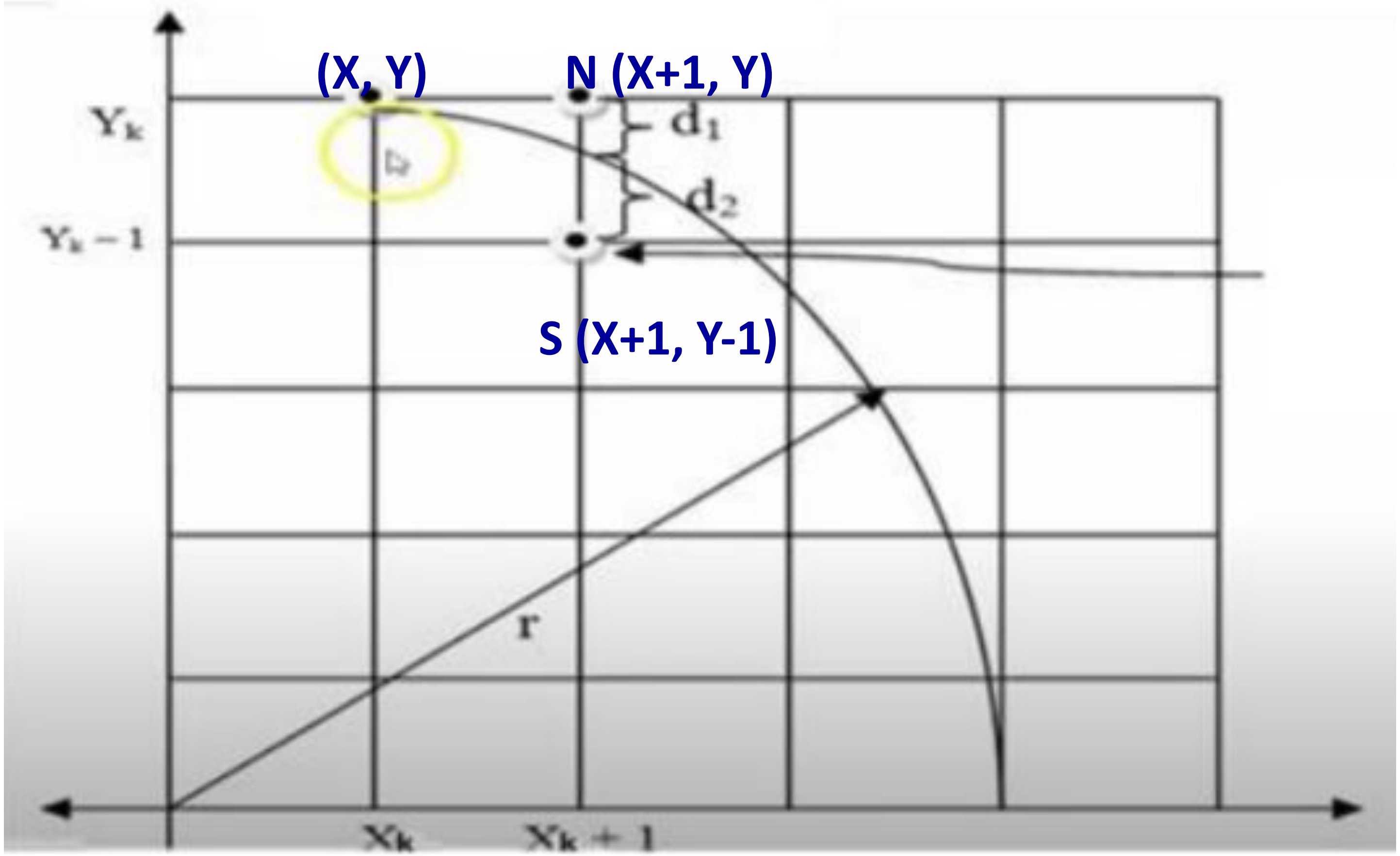
If $f(x,y) = 0$ then it is on the circle.
$f(x,y) > 0$ then it is outside the circle.
$f(x,y) < 0$ then it is inside the circle.

Bresenham's Circle Algorithm



* Only one octant of the circle need be generated, other parts can be obtained by successive reflections.
* If the first octant (0 to 45) is generated, the second octant can be obtained by reflection through the line y=x to yield the first quadrant
* The results in the first quadrant are reflected through the line x=0 to obtain the second quadrant
* Upper semicircle is reflected through the line y=0 to complete the circle

Bresenham's Circle Algorithm



for a point (X,Y), decide the next point (N or S)

P: decision parameter

If P ≤ 0, choose N(X+1, Y)

If P > 0, choose S(X+1, Y-1)

Bresenham's Circle Algorithm

Step-1: Input radius r and circle center (X_C, Y_C), the first point (X_0,Y_0)

Step-2: Calculate the initial value of decision parameter (P) as

$$P_0 = 3-2r$$

Step-3: Assume the starting coordinates are (X_K,Y_K). Find the next point (X_{K+1},Y_{K+1}) according to the value of the decision parameter P_K

If $P_K < 0 \longrightarrow (X_K+1, Y_K)$, $P_{K+1}=P_K+4X_K+6$

Otherwise $(P_K \geq 0) \longrightarrow (X_K+1, Y_K-1)$, $P_{K+1}=P_K+4(X_K-Y_K)+10$

Step-4: Move each pixel position (X,Y) into circular path:

$X=X+X_C$ and $Y=Y+Y_C$

Step-5: Repeat step 3 and 4 until $X \geq Y$

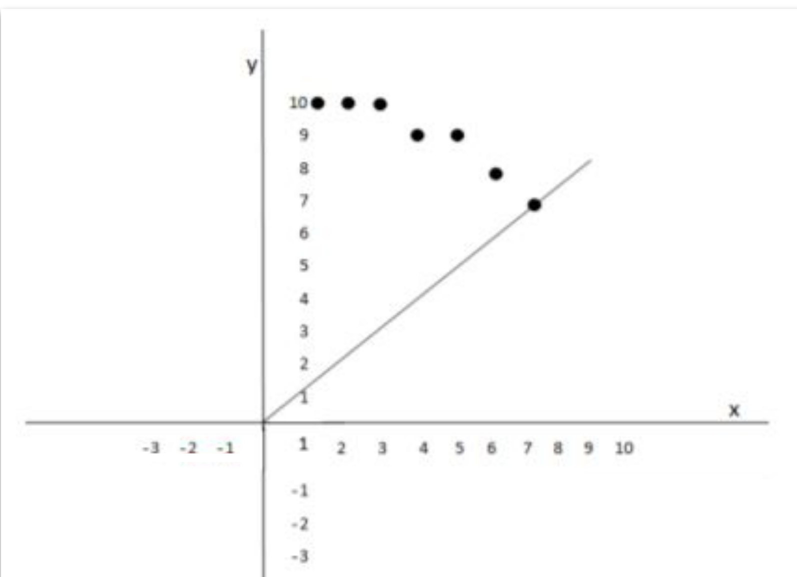
Step-6: Determine the symmetry points of the calculated points in other seven octant

Bresenham's Circle Algorithm

Center $(X_C,Y_C)=(0,0)$; starting point $(X_0,Y_0)=(0,10)$; $r=10$

find the pixels for the first quadrant

$P_0=3-2r=3-20=-17$	$P_0<0$	$\rightarrow (X_1,Y_1)=(X_0+1,Y_0)=(1,10)$
$P_1=P_0+4X_0+6=-17+0+6=-11$	$P_1<0$	$\rightarrow (X_2,Y_2)=(X_1+1,Y_1)=(2,10)$
$P_2=P_1+4X_1+6=-11+4+6=-1$	$P_2<0$	$\rightarrow (X_3,Y_3)=(X_2+1,Y_2)=(3,10)$
$P_3=P_2+4X_2+6=-1+8+6=13$	$P_3\geq 0$	$\rightarrow (X_4,Y_4)=(X_3+1,Y_3-1)=(4,9)$
$P_4=P_3+4(X_3-Y_3)+10=13+4(3-10)+10=-5$	$P_4<0$	$\rightarrow (X_5,Y_5)=(X_4+1,Y_4)=(5,9)$
$P_5=P_4+4X_4+6=-5+16+6=17$	$P_5\geq 0$	$\rightarrow (X_6,Y_6)=(X_5+1,Y_5-1)=(6,8)$
$P_6=P_5+4(X_5-Y_5)+10=17+4(5-9)+10=11$	$P_6\geq 0$	$\rightarrow (X_7,Y_7)=(X_6+1,Y_6-1)=(7,7)$



X>=Y stop here

* simple and easy to implement
* less accurate than the midpoint algorithm

Midpoint Circle Algorithm

Step-1: Input radius r and circle center (X_C, Y_C), the first point (X_0,Y_0)

Step-2: Calculate the initial value of decision parameter (d_0) as

$$d_0 = 1-r$$

Step-3: Assume the starting coordinates are (X_K,Y_K). Find the next point (X_{K+1},Y_{K+1}) according to the value of the decision parameter (d_K)

If $d_K < 0 \longrightarrow X_{K+1}=X_K+1,\ Y_{K+1}=Y_K,\ d_{K+1}=d_K+2X_{K+1}+1$

Otherwise $(d_K \geq 0) \longrightarrow X_{K+1}=X_K+1,\ Y_{K+1}=Y_K-1,\ d_{K+1}=d_K-2(Y_{K+1}-2X_{K+1})+1$

Step-4: Move each pixel position (X,Y) into circular path:

$X=X+X_C$ and $Y=Y+Y_C$

Step-5: Repeat step 3 and 4 until $X \geq Y$

Step-6: Determine the symmetry points of the calculated points in other seven octant

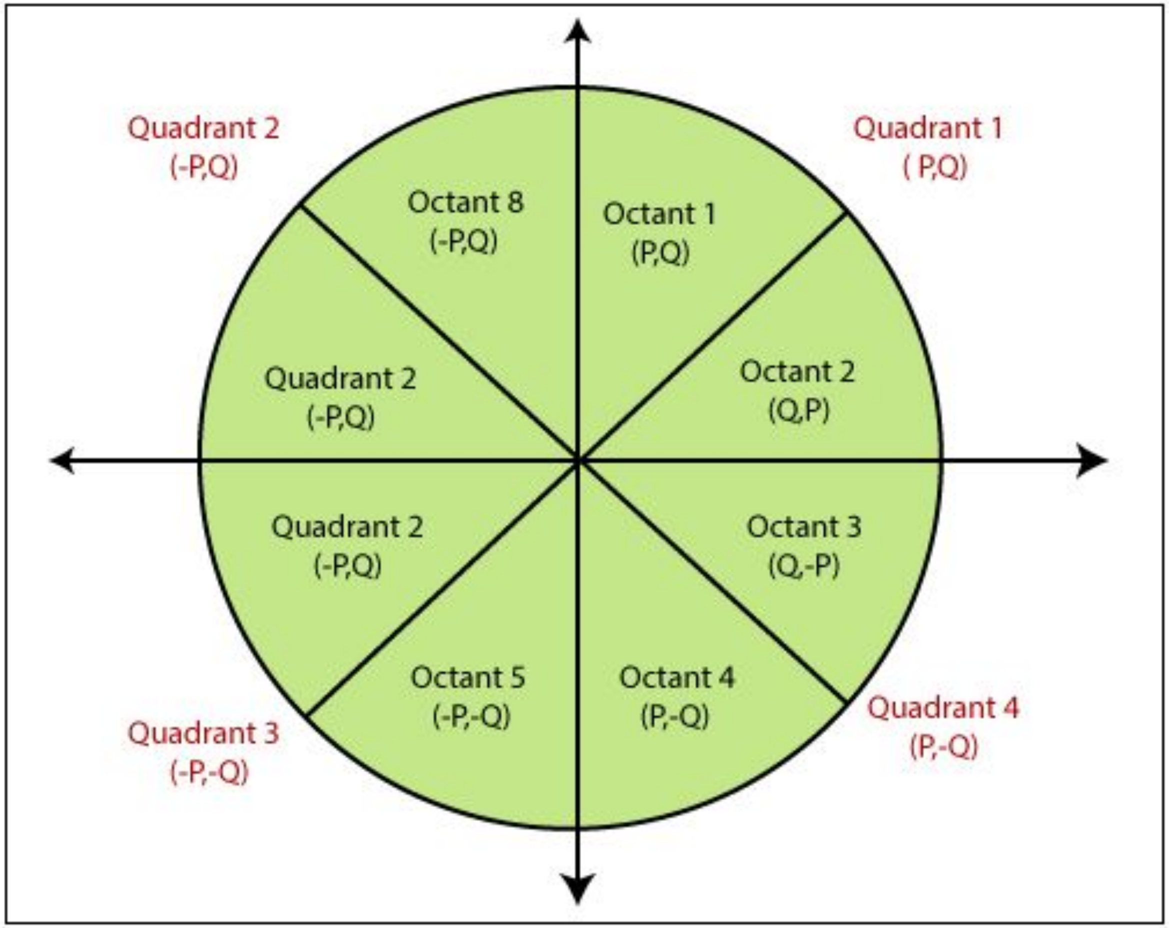
Midpoint Circle Algorithm

Center $(X_C,Y_C)=(0,0)$; starting point $(X_0,Y_0)=(0,10)$; $r=10$

find the pixels for the first quadrant

$d_0=1-r=1-10=-9$	$d_0<0$	$\longrightarrow (X_1,Y_1)=(X_0+1,Y_0)=(1,10)$
$d_1=d_0+2X_1+1=-9+2+1=-6$	$d_1<0$	$\longrightarrow (X_2,Y_2)=(X_1+1,Y_1)=(2,10)$
$d_2=d_1+2X_2+1=-6+4+1=-1$	$d_2<0$	$\longrightarrow (X_3,Y_3)=(X_2+1,Y_2)=(3,10)$
$d_3=d_2+2X_3+1=-1+6+1=6$	$d_3\geq 0$	$\longrightarrow (X_4,Y_4)=(X_3+1,Y_3-1)=(4,9)$
$d_4=d_3-2(Y_4-X_4)+1=6-2(9-4)+1=-3$	$d_4<0$	$\longrightarrow (X_5,Y_5)=(X_4+1,Y_4)=(5,9)$
$d_5=d_4+2X_5+1=-3+10+1=8$	$d_5\geq 0$	$\longrightarrow (X_6,Y_6)=(X_5+1,Y_5-1)=(6,8)$
$d_6=d_5-2(Y_6-X_6)+1=8-2(8-6)+1=5$	$d_6\geq 0$	$\longrightarrow (X_7,Y_7)=(X_6+1,Y_6-1)=(7,7)$
		X>=Y stop here

Determining the Symmetry Points



determine the symmetry points of the calculated points in other seven octant

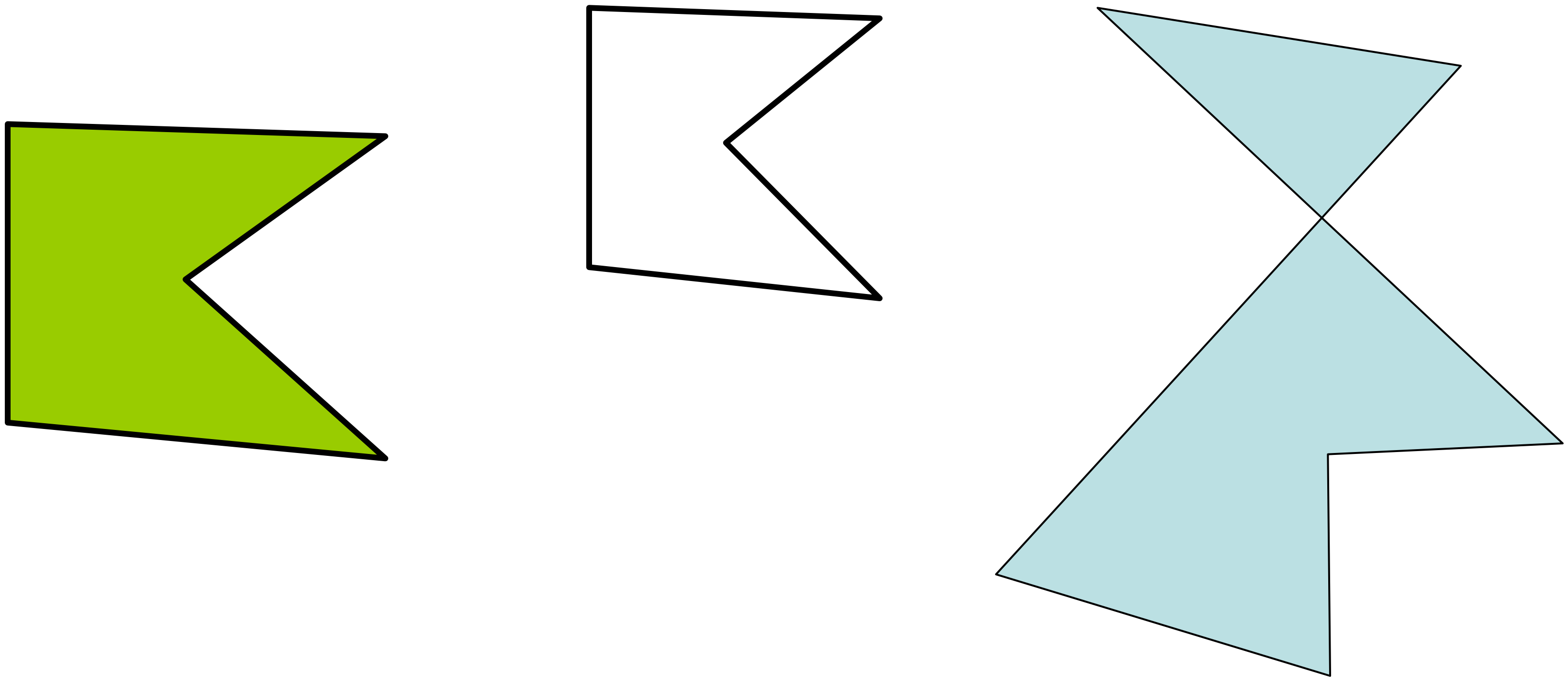
Determining the Symmetry Points

Points of Octant 1	Points of Octant 2
(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)

Determining the Symmetry Points

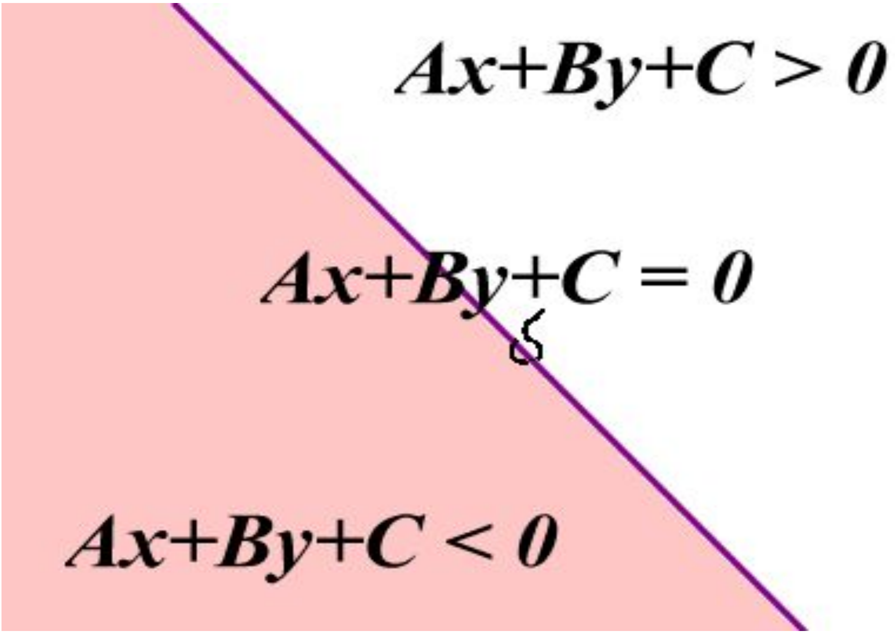
Quadrant 1 (p, q)	Quadrant 2 (-p, q)	Quadrant 3 (-p, -q)	Quadrant 4 (p, -q)
(0, 10)	(0, 10)	(0, -10)	(0, -10)
(1, 10)	(-1, 10)	(-1, -10)	(1, -10)
(2, 10)	(-2, 10)	(-2, -10)	(2, -10)
(3, 10)	(-3, 10)	(-3, -10)	(3, -10)
(4, 9)	(-4, 9)	(-4, -9)	(4, -9)
(5, 9)	(-5, 9)	(-5, -9)	(5, -9)
(6, 8)	(-6, 8)	(-6, -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)

Area Filling



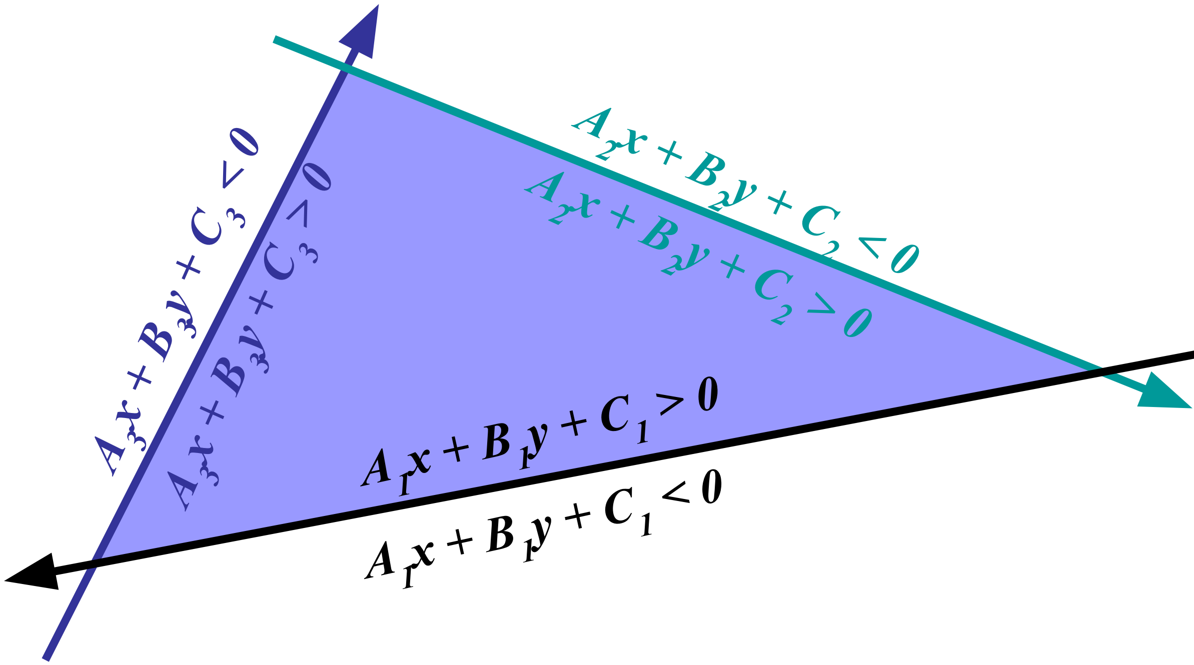
Q: how can we generate a solid color/patterned polygon area?

Edge Equations



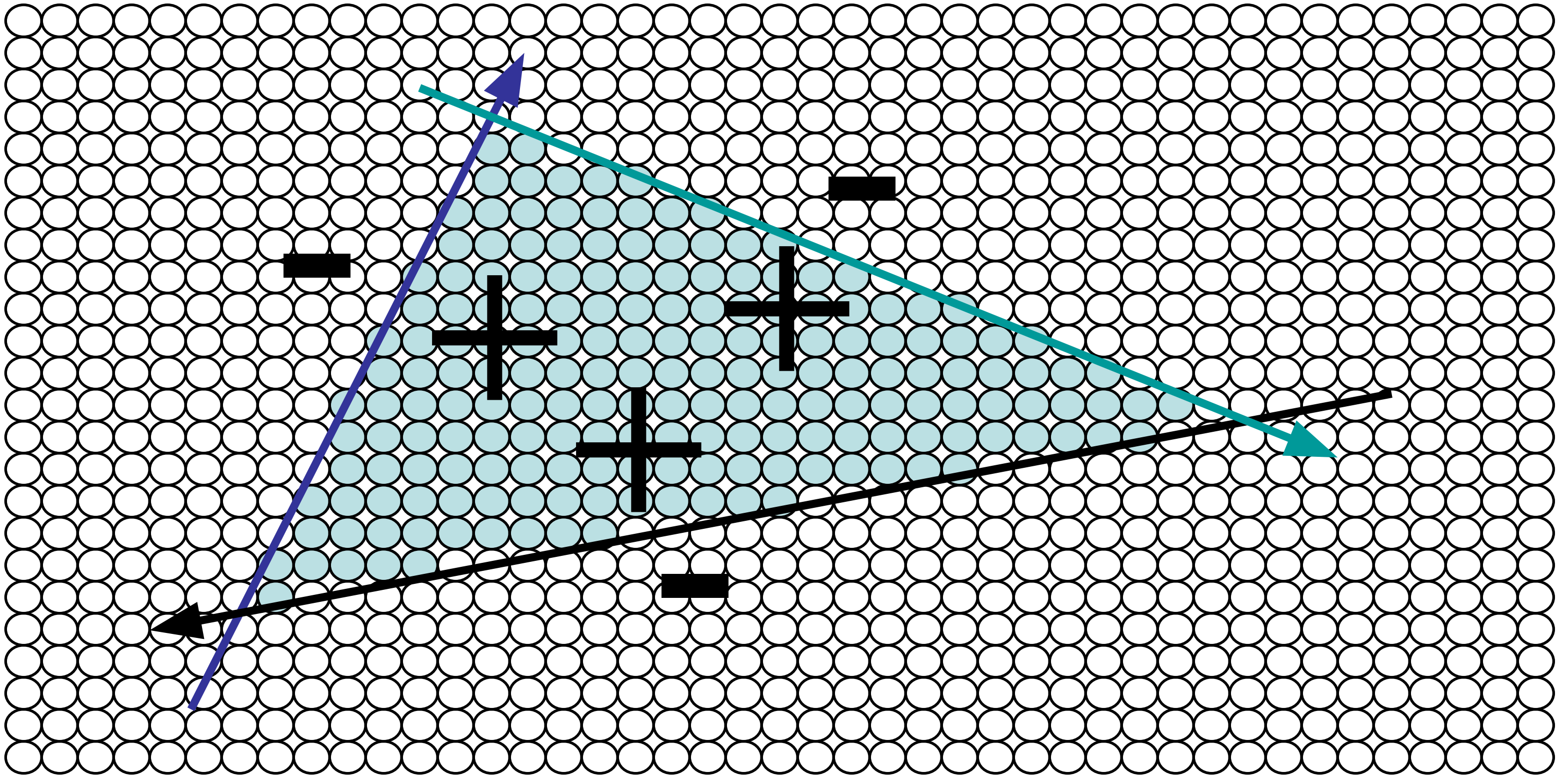
* We can make use of edge equations
* Edge equations define the edges
* Each line defines 2 half-spaces: <0 and >0
* =0 is the edge

Edge Equations



a triangle can be defined as the intersection of three positive half-spaces

Edge Equations

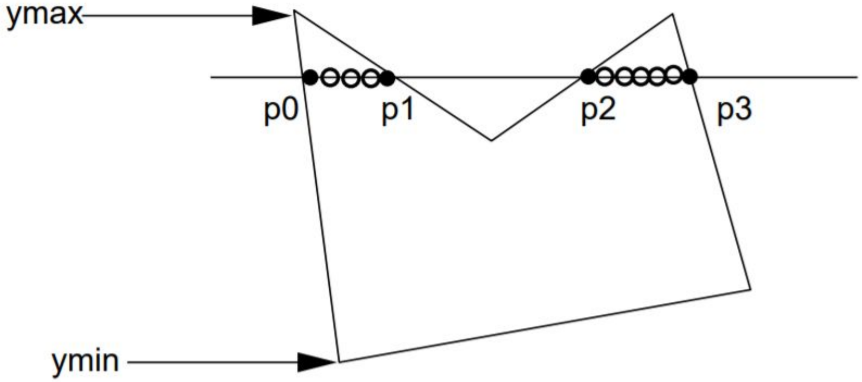


to fill a triangle shaped area, turn on those pixels for which, all edge equations evaluate to >0 case

Area Filling

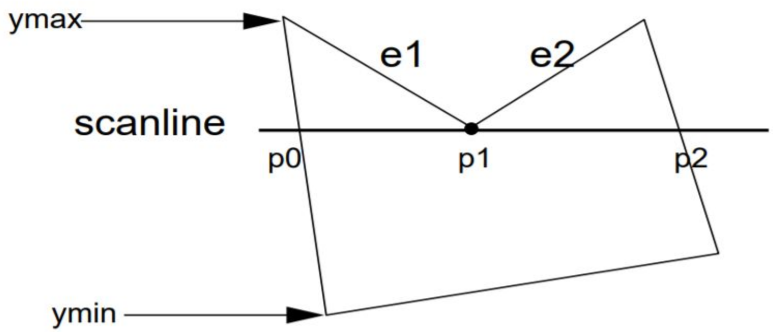
- Scan Line Algorithm
- Boundary Fill Algorithm
- Flood Fill Algorithm

Scan Line Algorithm

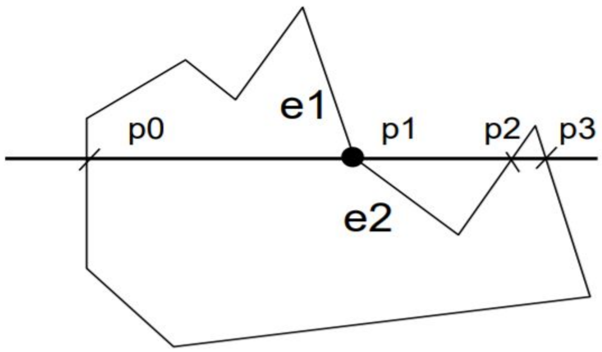


- Find out the ymin and ymax from the polygon
- Find each intersection point of the polygon with the scan line *(p0,p1,p2,p3)*
- Sort the intersection points in the increasing order of X coordinate *(p0,p1,p2,p3)*
- Fill pairwise *(from p0 to p1 and from p2 to p3)*

Special cases:



fill p0 to p1 and p1 to p2



if p1 is counted twice, p1 to p2 will be filled erroneously

Boundary Fill Algorithm

A recursive algorithm: If we have a specified boundary in a single color, then the algorithm proceeds pixel by pixel until the boundary color is encountered

Boundary fill (x, y, fill, boundary)

- Initialize boundary of the region, and variable “fill with color”
- Let the interior pixel (x,y)

  current=getpixel(x,y)
- If current is not equal to boundary and current is not equal to fill then

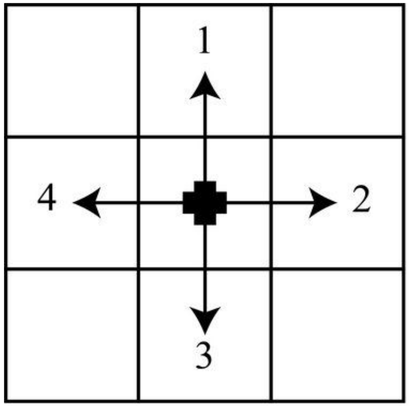
  set pixel (x, y, fill)

  boundary fill 4(x+1,y,fill,boundary)

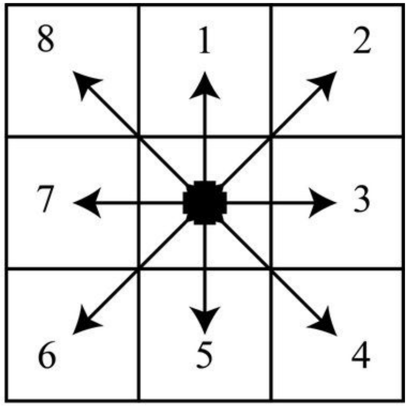
  boundary fill 4(x-1,y,fill,boundary)

  boundary fill 4(x,y+1,fill,boundary)

  boundary fill 4(x,y-1,fill,boundary)
- End



4-Connected



8-Connected

Flood Fill Algorithm

** We can recolor an area that is not defined within a single color boundary*
** We can paint such areas by replacing a color instead of searching for a boundary color value*

```
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);
     }
}
```

Boundary Fill vs. Flood Fill

Flood fill: more than one boundary colours
Boundary fill: single boundary colour

Flood fill: replaces every point color
Boundary fill: checks for the boundary colour

Flood fill: high memory requirements
Boundary fill: less amount of memory