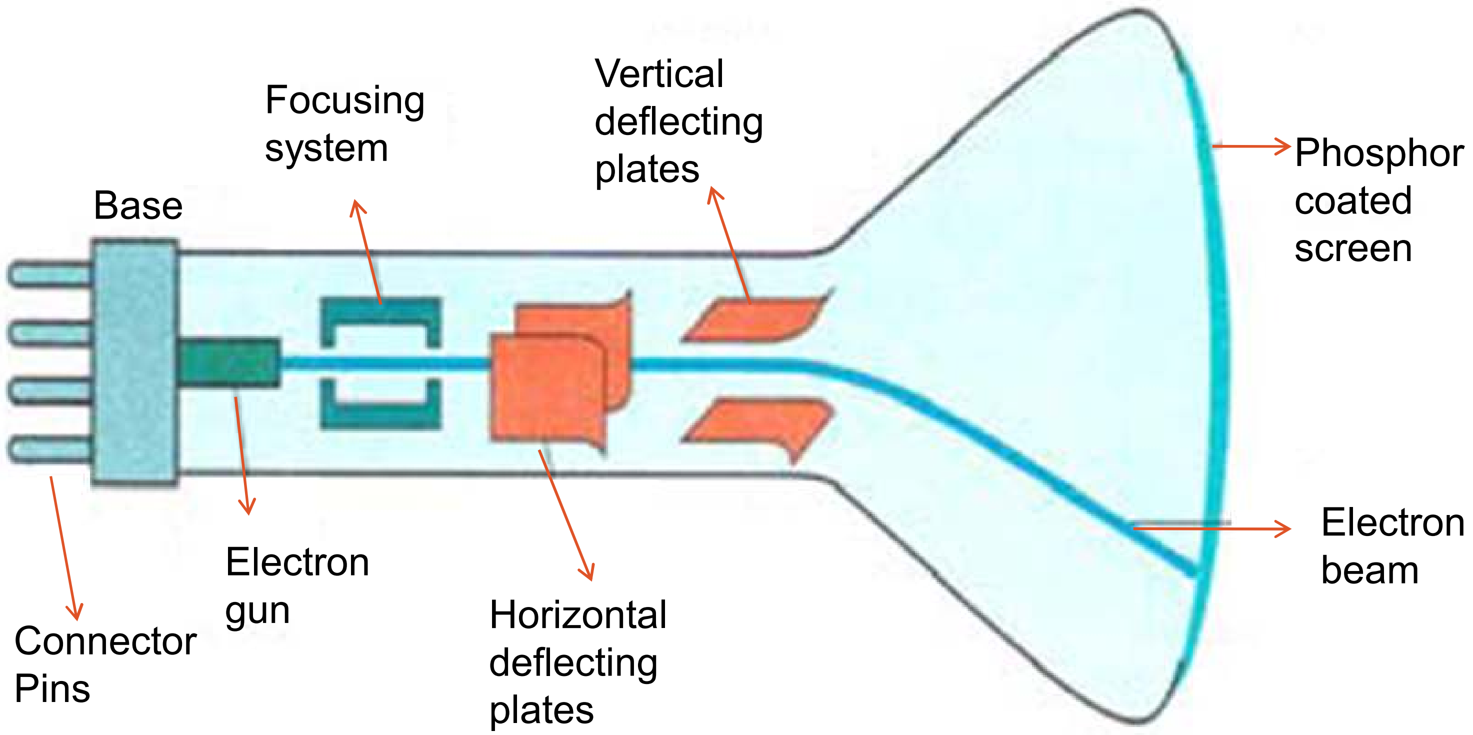
Algoritma Primitif 2D

Overview

- Display
- Primitif 2 D
- Algoritma Line
  - DDA
  - Bresenham's Line Generation
- Algoritma Circle
  - Midpoint
- Algoritma polygon filling

Display Devices

Refresh CRT

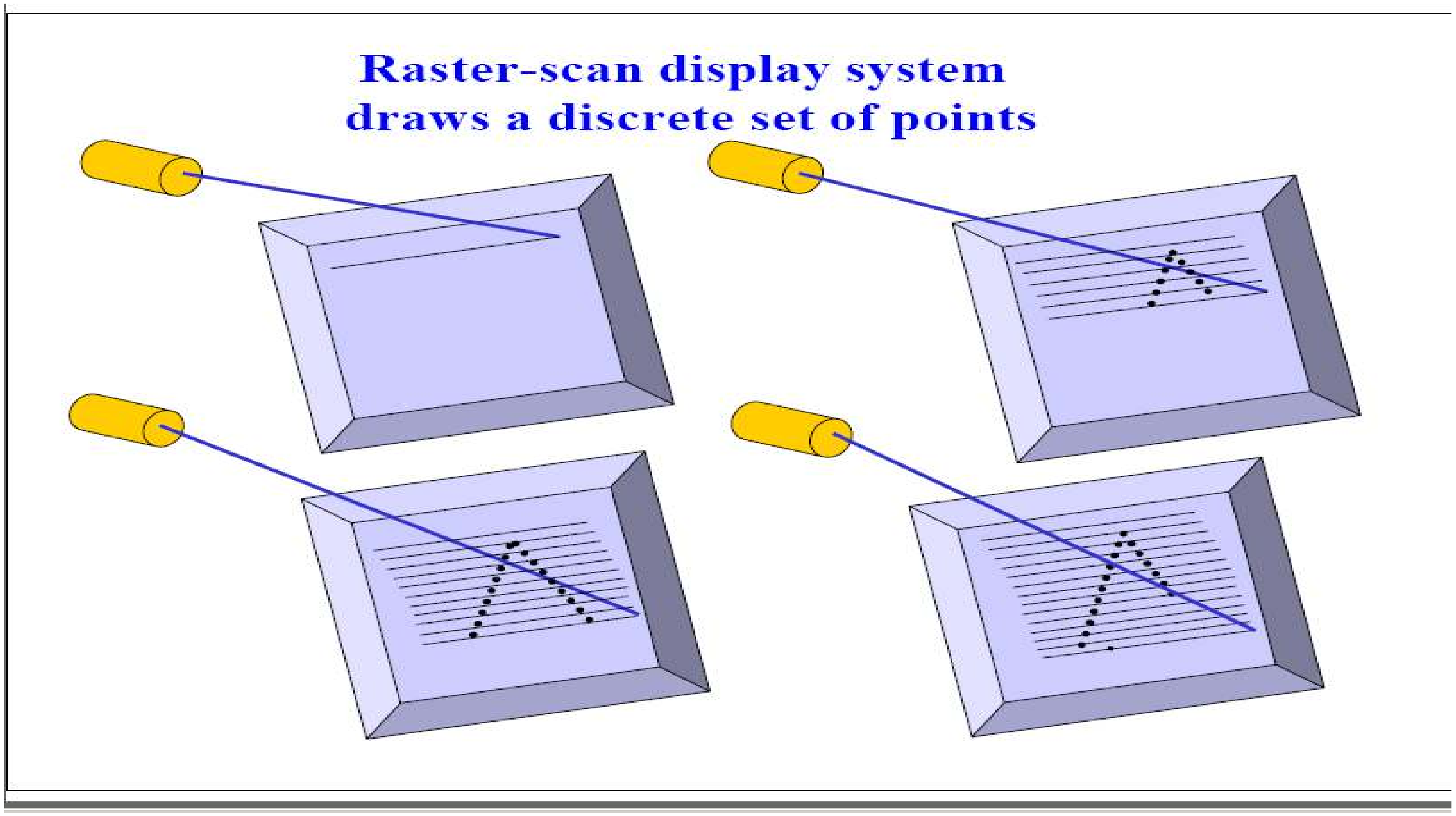

Base
Focusing system
Vertical deflecting plates
Phosphor coated screen
Electron gun
Connector Pins
Horizontal deflecting plates
Electron beam

Raster Scan Displays

- The most common type of monitor employing a CRT is raster scan display.
- In Raster scan display the electron beam is swept across the screen one row at a time from top to bottom.
- Raster: A rectangular array of points or dots
- Pixel: One dot or picture element of the raster. Its intensity range for pixels depends on capability of the system
- Scan line: A row of pixels
- Picture elements are stored in a memory called frame buffer
- A special memory is used to store the image with scan-out synchronous to the raster. We call this the *frame buffer*

Disadvantage
Raster system produces jagged lines that are plotted as discrete points

Raster Scan Displays cont..



Raster Scan Displays cont..

Horizontal Retrace

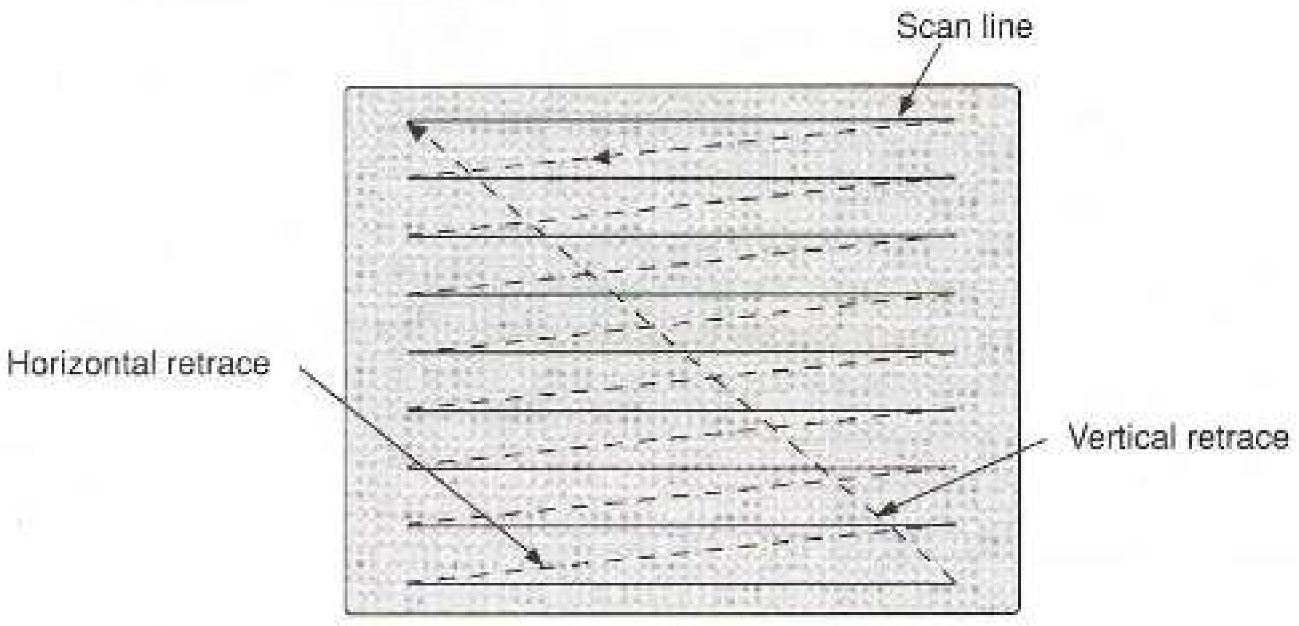
- At the end of each scan line , the electron beam returns to the left side of the screen to begin displaying the next scan line. The return to the left of the screen , after refreshing each scan line is called horizontal retrace of electron beam.

Vertical retrace

- At the end of each frame the electron beam returns to the top left corner of the screen to begin the next frame.

Raster Scan Displays cont..

- Refresh Rate
  - Usually 30~75 Hz



Random Scan Display

•The electron beam is directed only to the parts of the screen where a picture is to be drawn.

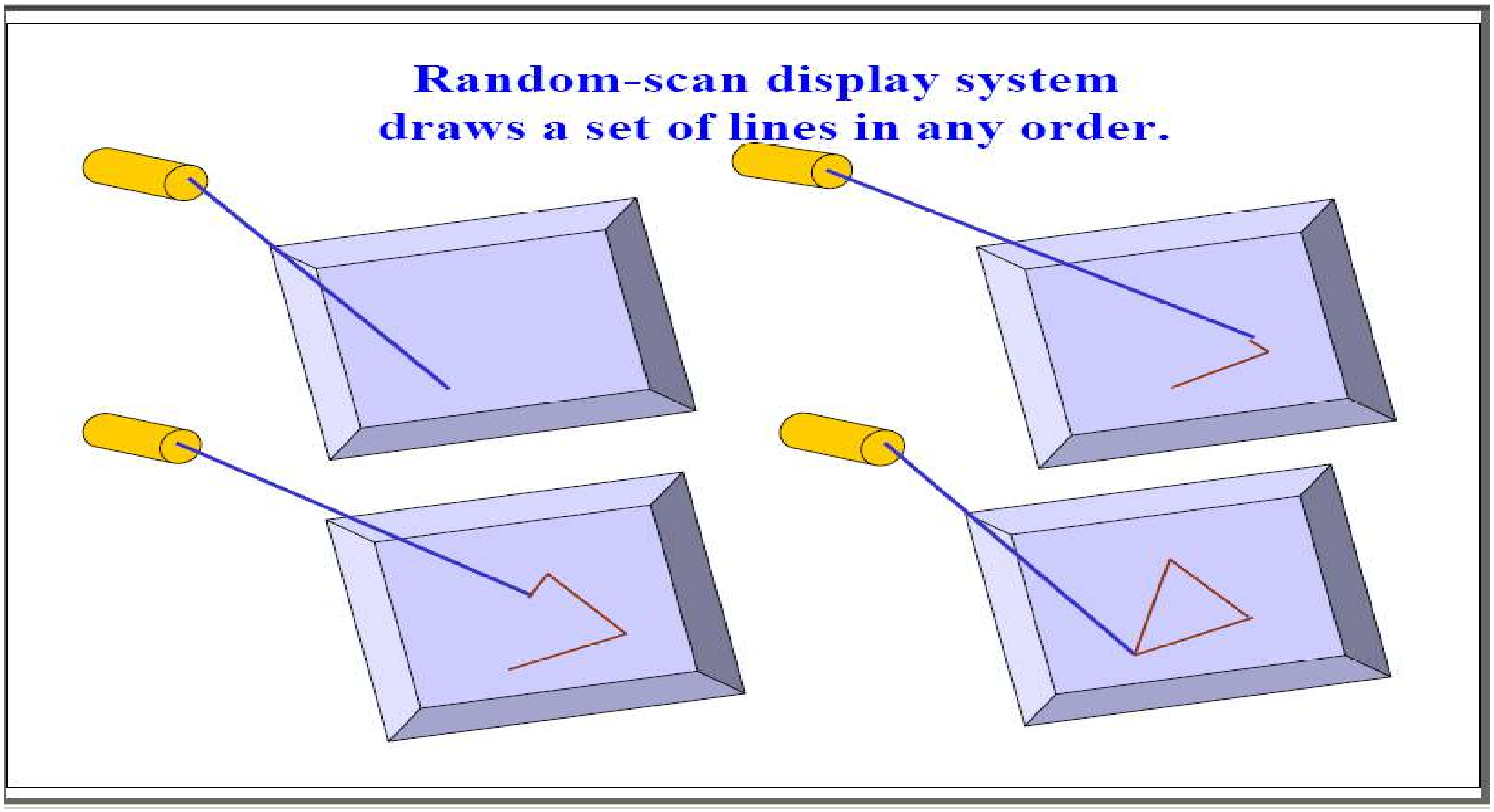
•Picture definition is stored as a set of line drawing commands in an area of memory referred to as refresh display file.

•To display a specified picture ,the system cycles through a set of commands in the display file , drawing each component line after processing all lines drawing commands the s/m cycle back to the first line command in the list.

Advantage

Has high resolution since picture definition is stored as line drawing commands

Random Scan Display cont...


Random-scan display system
draws a set of lines in any order.

Output Primitives

Output Primitives

- Basic geometric structures used to describe scenes.
- Can be grouped into more complex structures
- Example : Point, straight line segments, circles and other conic sections, polygon color areas and character strings
- Construct the vector picture

Rasterization

The process of determining the appropriate pixels for representing picture or graphic object

Scan conversion

It is the final step of rasterization . It converts picture definition into a set of pixel-intensity values for storage in the frame buffer.

ALGORITMA

Garis

Lingkaran

Polygon

Algoritma untuk menggambar garis

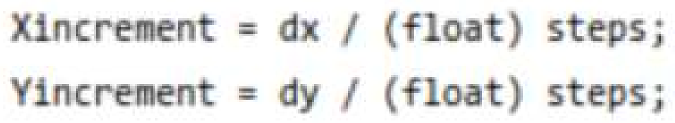
- Garis menghubungkan 2 titik
- Garis adalah elemen dasar dalam computer graphic
- 3 algoritma pembentuk garis
  - DDA
  - Bressenham
  - Midpoint
- Misal sebuah garis menghubungkan dua titik $P_0(X_0, Y_0)$ dan $P_1(X_1, Y_1)$

Algoritma Digital Differential Analyzer (DDA)

- Algoritma menggambar garis yang paling sederhana, sbb:
- Langkah 1: Dapatkan input dari 2 titik yang ingin dihubungkan (titik awal Po dan titik akhir P1)
- Langkah 2: hitung perbedaan antara kedua titip tersebut
  - dx = X1-Xo
  - dy = Y1-Yo
- Langkah 3: berdasar perbedaan yang dihitung pada langkah 2, identifikasi jumlah langkah untuk pixel selanjutnya
  - Jika dx> dy maka memerlukan lebih banyak langkah pada sumbu x, dan sebaliknya

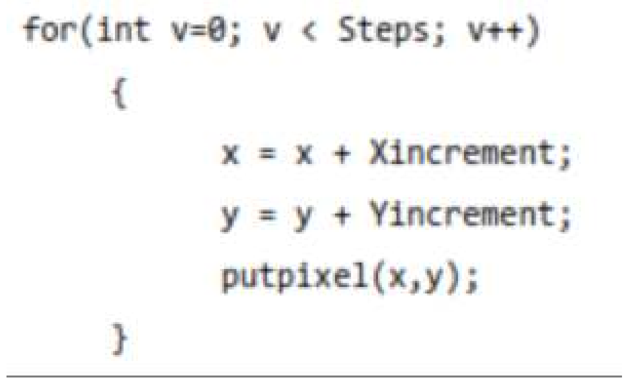
DDA -2

- Langkah 4: hitung increment di sumbu x dan sumbu y



```
Xincrement = dx / (float) steps;
Yincrement = dy / (float) steps;
```

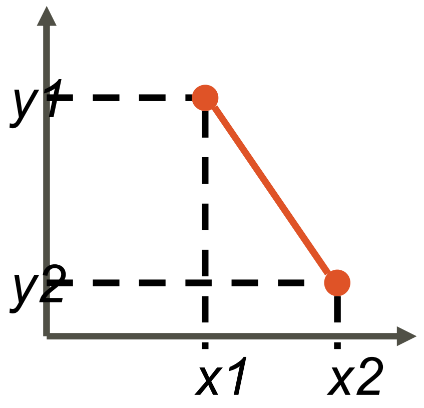
- Langkah 5: menentukan pixel dengan mennambah nilai x dan y sesuai rumus yang telah didapat



```
for(int v=0; v < Steps; v++)
    {
          x = x + Xincrement;
          y = y + Yincrement;
          putpixel(x,y);
    }
```

The DDA Algorithm cont..

- **Digital Differential Analyzer**
  - 0 < Slope <= 1
    - Unit x interval = 1
  - Slope > 1
    - Unit y interval = 1
  - -1 <= Slope < 0
    - Unit x interval = -1
  - Slope < -1
    - Unit y interval = -1



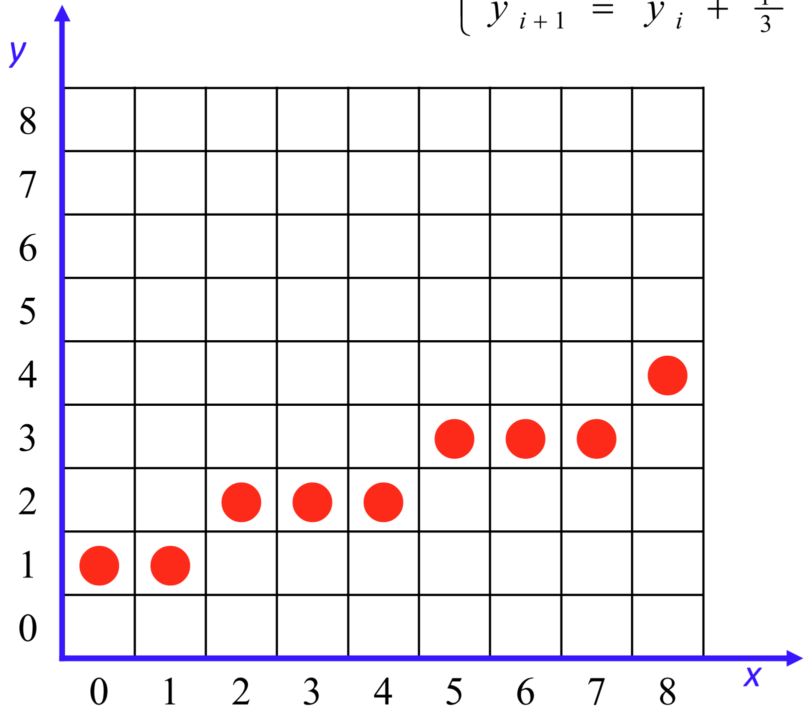
$$x_{k+1} = x_k - \frac{1}{m}$$

Example (DDA)

x	y	round(y)
0	1	1
1	4/3	1
2	5/3	2
3	2	2
4	7/3	2
5	8/3	3
6	3	3
7	10/3	3
8	11/3	4

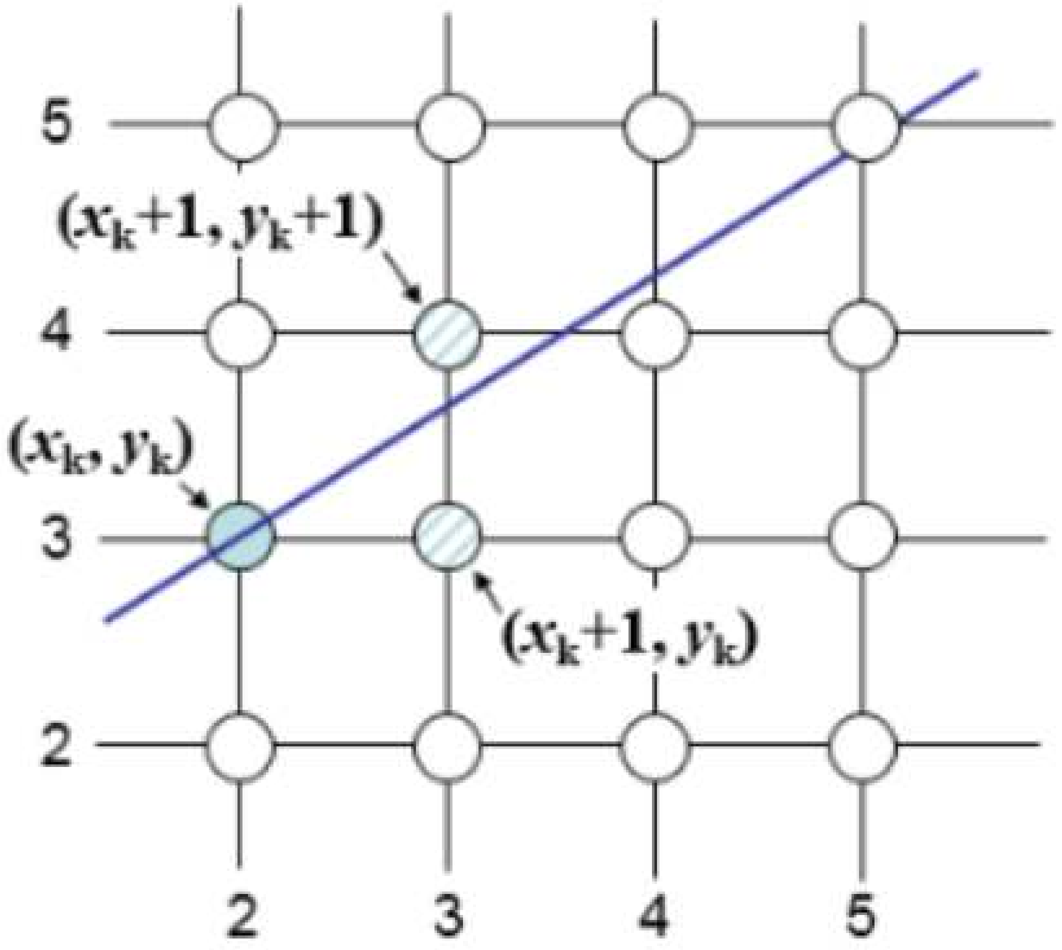
$$y = \frac{1}{3}x + 1$$

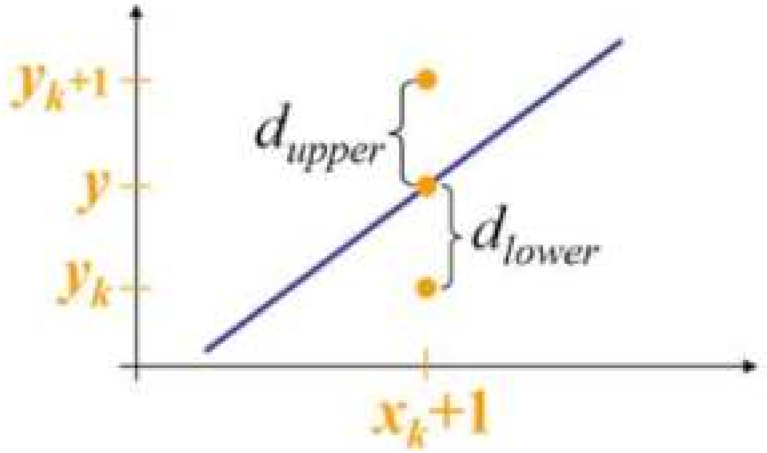
$$\begin{cases} 0 \le m \le 1 \\ x_{i+1} = x_i + 1 \\ y_{i+1} = y_i + \frac{1}{3} \end{cases}$$



Algoritma Bresenham’s

- Algoritma pembuat garis
- Juga berdasar incremental scan conversion
- Keuntungan: hanya memakai integer
- Bergerak pada sumbu x dengan suatu interval tertentu, dan pada masing-masing langkah memilih antara 2 koordinat y yang berbeda
- Memilih titik yang lebih dekat dengan garis asli


5
4
3
2
2
3
4
5
(x_k+1, y_k+1)
(x_k, y_k)
(x_k+1, y_k)


y_k+1
y
y_k
d_{upper}
d_{lower}
x_k+1

- Persamaan garis y pada titik Xk+1:

$$Y = m(X_k + 1) + b$$

- Dari persamaan tersebut

$$\begin{aligned} d_{lower} &= y - y_k \\ &= m(X_k + 1) + b - Y_k \end{aligned}$$

$$\begin{aligned} d_{upper} &= (y_k + 1) - y \\ &= Y_k + 1 - m(X_k + 1) - b \end{aligned}$$

- Keputusan dapat diambil tergantung titik yang lebih dekat dengan garis sebenarnya

$$d_{lower} - d_{upper} = 2m(x_k + 1) - 2y_k + 2b - 1$$

$$\begin{aligned} dx(d_{lower} - d_{upper}) &= dx\left(2\frac{dy}{dx}(x_k + 1) - 2y_k + 2b - 1\right) \\ &= 2dy \cdot x_k - 2dx \cdot y_k + 2dy + dx(2b - 1) \\ &= 2dy \cdot x_k - 2dx \cdot y_k + C \end{aligned}$$

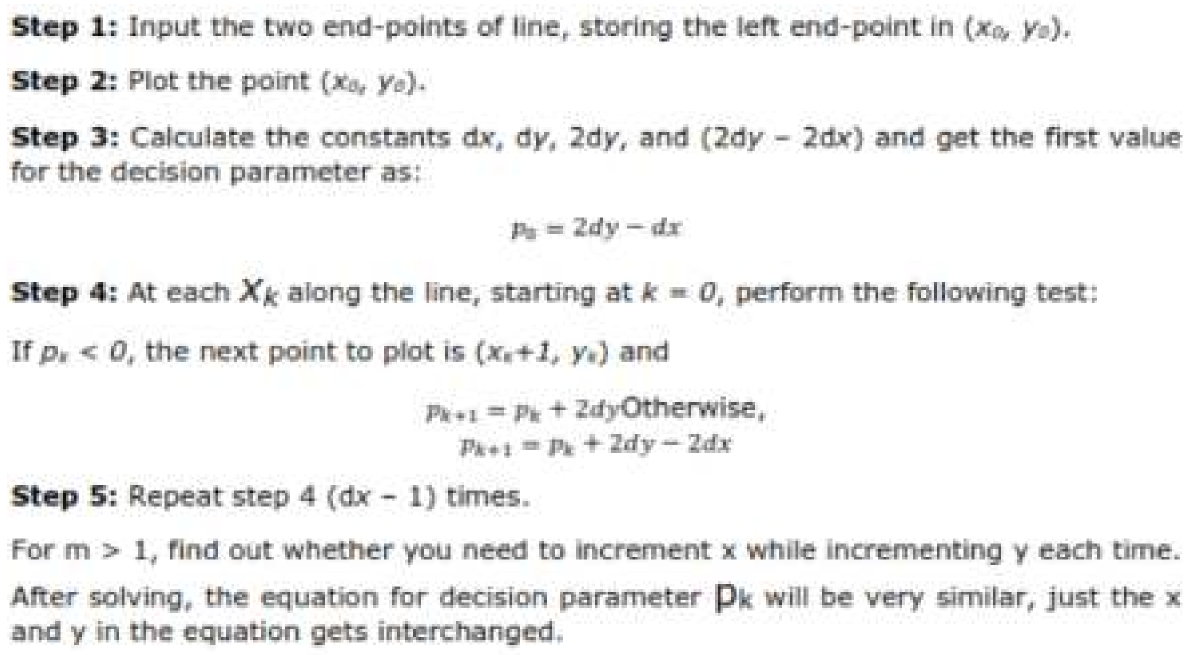
$$\begin{aligned} p_k &= dx(d_{lower} - d_{upper}) \\ &= 2dy \cdot x_k - 2dx \cdot y_k + C \end{aligned}$$

Decision parameter

Decision parameter pk

- Jika pk negative, pilih yk, otherwise pilih yk+1

algoritma



Step 1: Input the two end-points of line, storing the left end-point in (x_0, y_0).

Step 2: Plot the point (x_0, y_0).

Step 3: Calculate the constants dx, dy, 2dy, and (2dy – 2dx) and get the first value for the decision parameter as:

$$p_0 = 2dy - dx$$

Step 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

$$p_{k+1} = p_k + 2dy \text{Otherwise,}$$
$$p_{k+1} = p_k + 2dy - 2dx$$

Step 5: Repeat step 4 (dx – 1) times.

For m > 1, find out whether you need to increment x while incrementing y each time.

After solving, the equation for decision parameter p_k will be very similar, just the x and y in the equation gets interchanged.

Example

- To illustrate the algorithm, we digitize the line with endpoints (20,10) and (30,18). This line has slope of 0.8, with

  $\Delta x = 10$

  $\Delta y = 8$

- The initial decision parameter has the value

  $p_0 = 2\Delta y - \Delta x = 6$

- and the increments for calculating successive decision parameters are

  $2\,\Delta y = 16$

  $2\,\Delta y - 2\,\Delta x = -4$

Example

- We plot the initial point (x_0 , y_0)=(20,10) and determine successive pixel positions along the line path from the decision parameter as

K	p_k	(x_{k+1}, y_{k+1})	**K**	p_k	(x_{k+1}, y_{k+1})
0	6	(21,11)	5	6	(26,15)
1	2	(22,12)	6	2	(27,16)
2	-2	(23,12)	7	-2	(28,16)
3	14	(24,13)	8	14	(29,17)
4	10	(25,14)	9	10	(30,18)

Example

A plot of the pixels generated along this line path is shown in Fig.

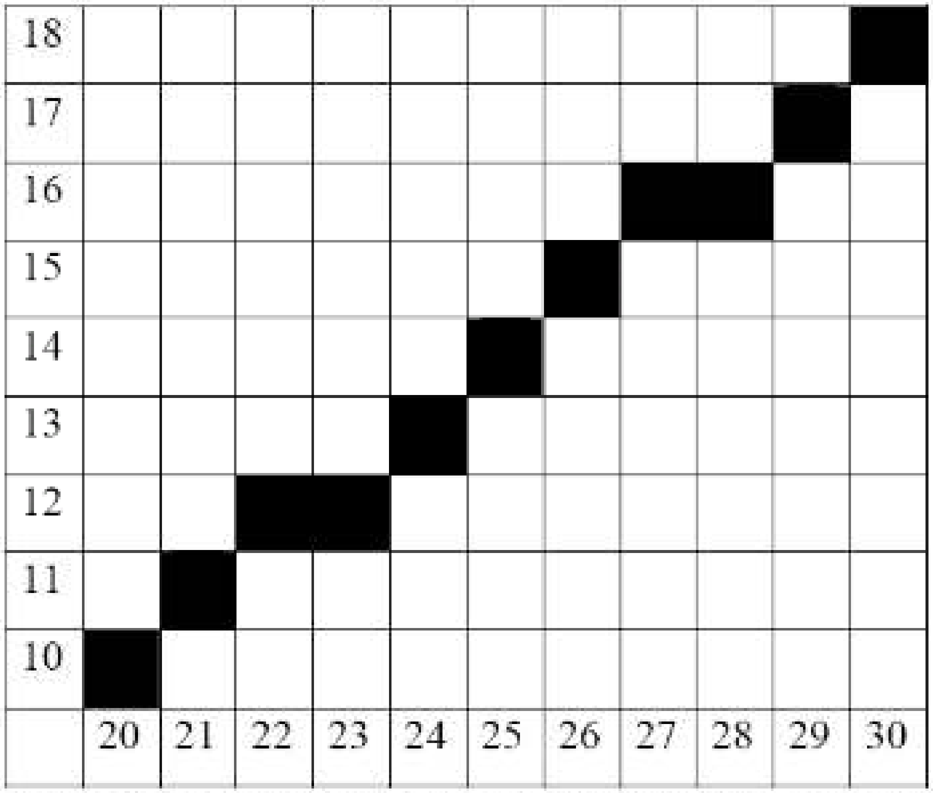


Figure: The Bresenham line from point (20,10) to point (30,18)

CIRCLE

Drawing Circles and Arcs

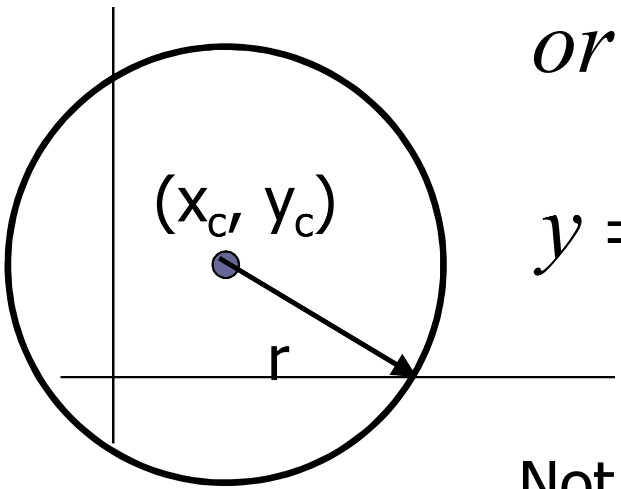
- Sama seperti algoritma pembuat garis:
  - Harus menentukan pixel yang diaktifkan
- Non linear
  - Simple equation implementation
  - Optimized

Cartesian form of Circle Equations

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

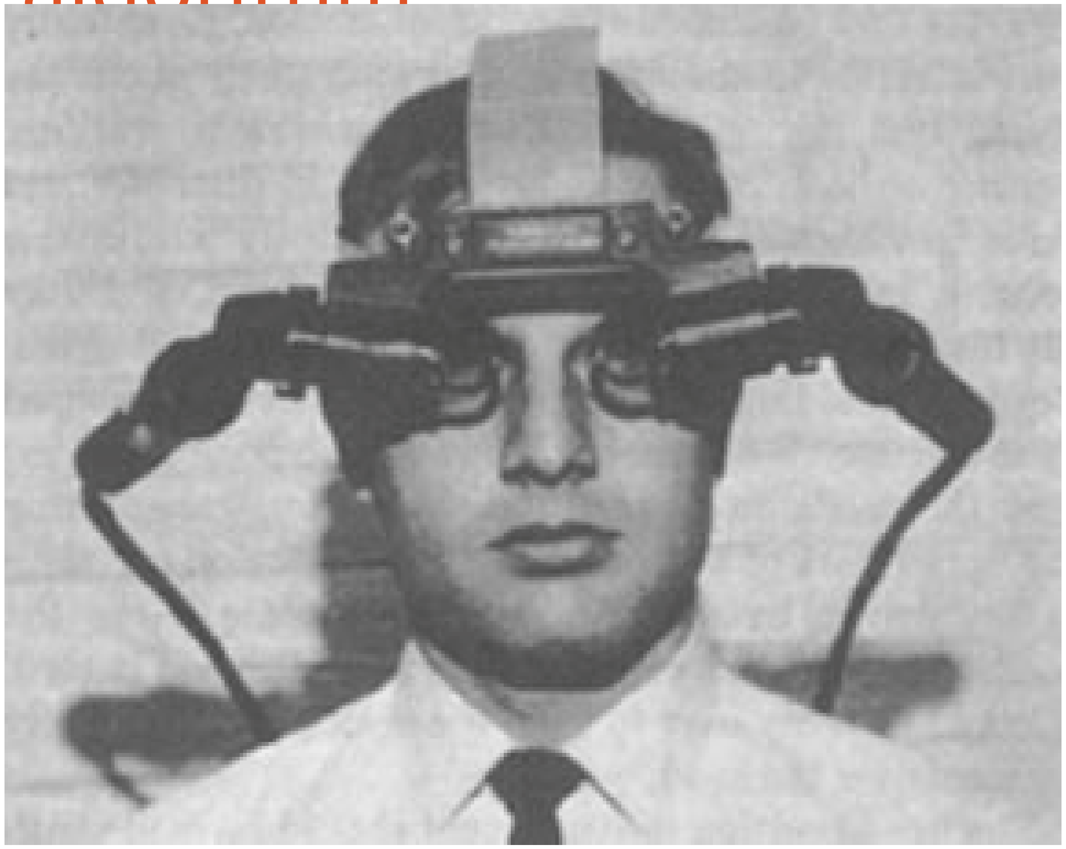
or

$$y = y_c \pm \sqrt{r^2 - (x - x_c)^2}$$

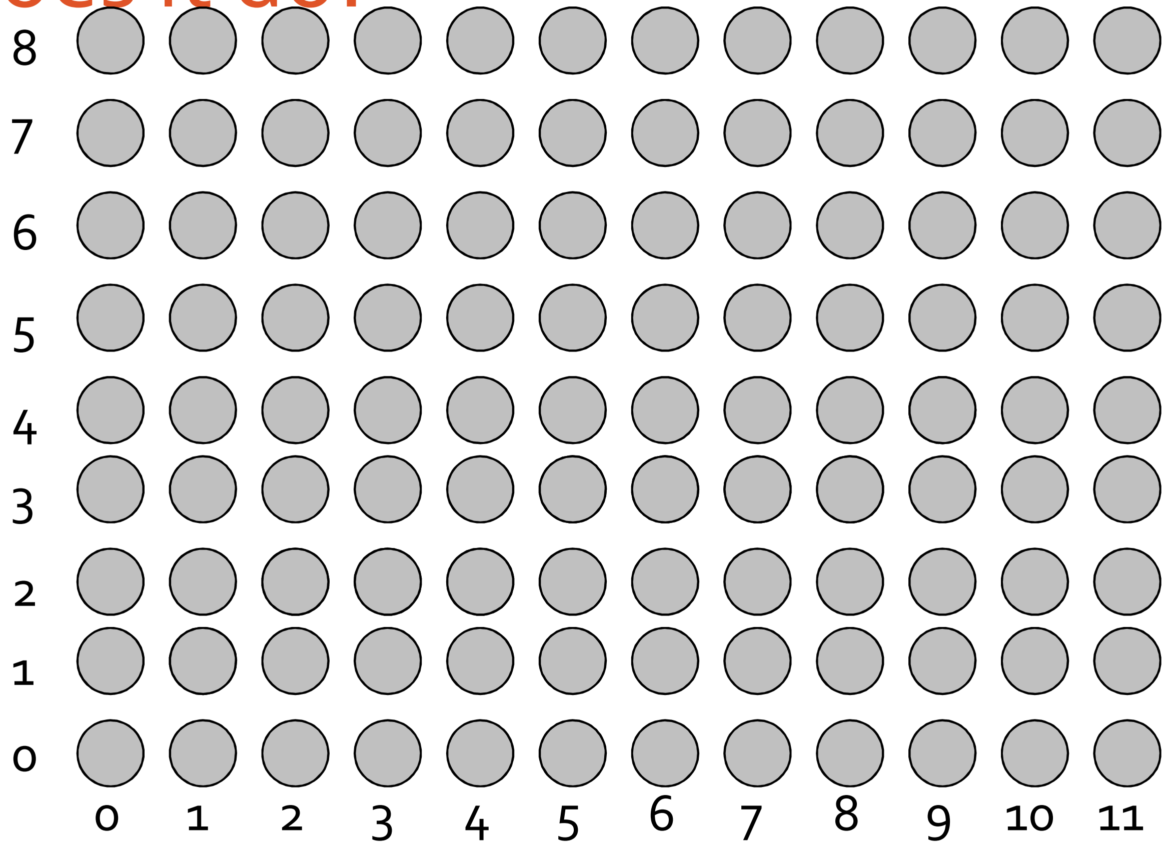


Not a very good method. Why?

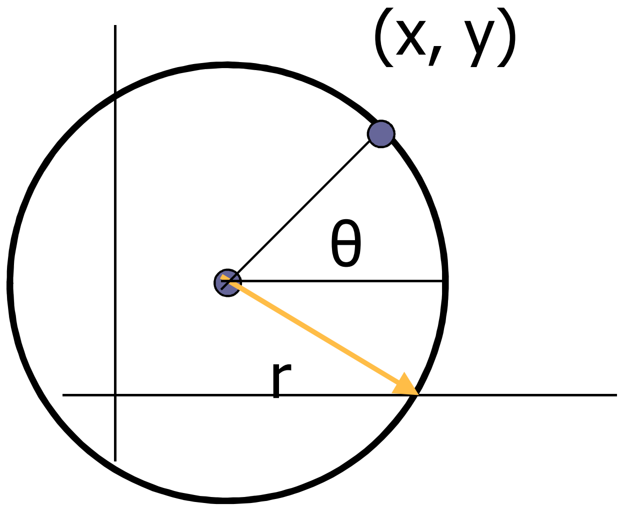
Be the algorithm!



What does it do?



Polar Coordinate Form



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

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

$$\Delta s = r\theta$$

$$\Delta s \approx 1$$

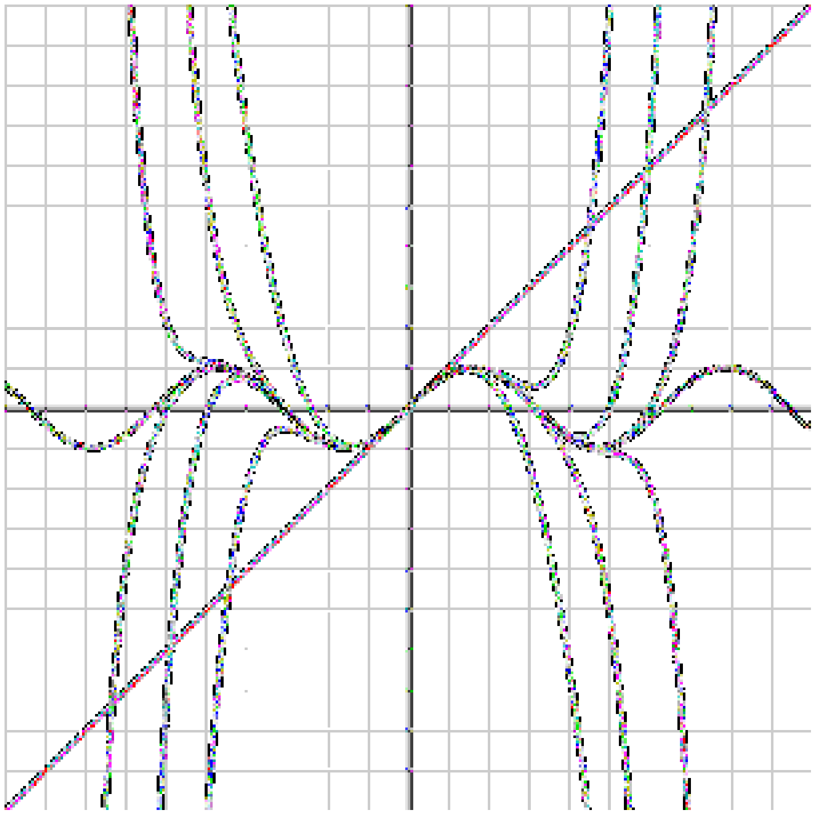
$$\Delta\theta \approx \frac{1}{r}$$

Simple method: plot directly from parametric equations

Trig functions are expensive

$$\sin\theta = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!}\theta^{2n+1}$$

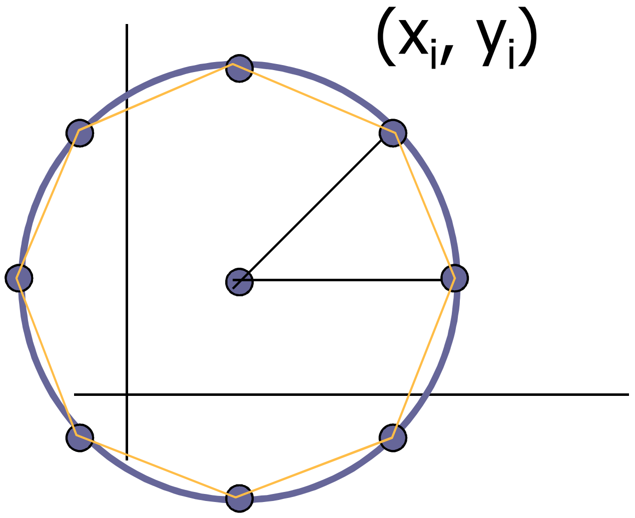
$$\cos\theta = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!}\theta^{2n}$$



Plots of first 1, 3, 5, 7, 9, 11, and 13 terms in the Taylor's series for sin

Polygon Approximation

Calculate polygon vertices
from polar equation;
connect with line algorithm



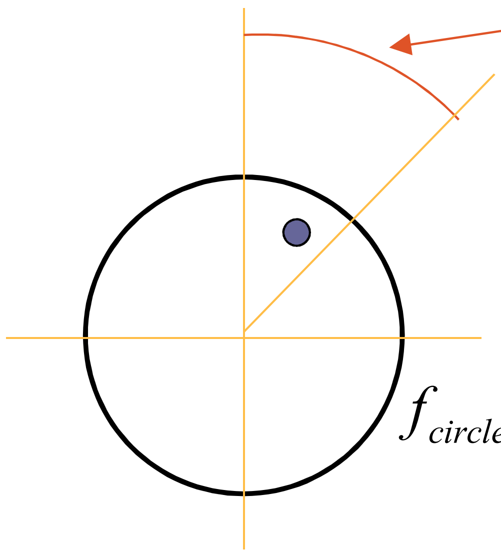
The Bresenham Algorithms

Bresenham, J. E. Algorithm for computer control of a digital plotter, *IBM Systems Journal*, 4(1), **1965**, pp. 25-30.

Bresenham, J. E. A linear algorithm for incremental digital display of circular arcs. *Communications of the ACM*, 20(2), **1977**, pp. 100-106.

Bresenham's Midpoint Circle Algorithm

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



0<|m|<1 in this region

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

The Circle Decision Parameter

Calculate f_{circle} for point midway between candidate pixels

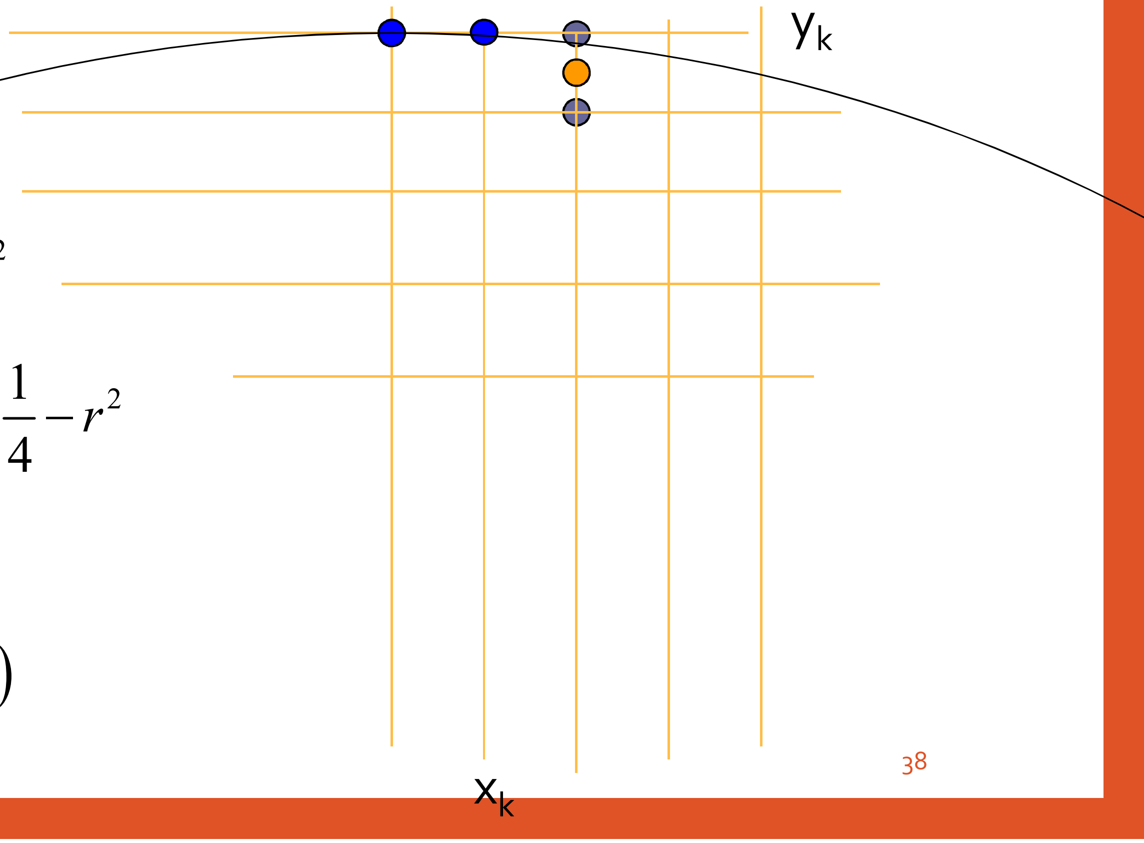
$$p_k = f_{circle}\left(x_k + 1, y_k - \frac{1}{2}\right)$$

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

$$= x_k^2 + 2x_k + 1 + y_k^2 - y_k + \frac{1}{4} - r^2$$

If $p_k < 0$: $\text{Plot}(x_k + 1, y_k)$

If $p_k >= 0$: $\text{Plot}(x_k + 1, y_k - 1)$



Calculating p_{k+1}

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

$$= (x_k + 1 + 1)^2 + \left(y_{k+1} - \frac{1}{2}\right)^2 - r^2$$

$$= x_k{}^2 + 4x_k + 4 + y_{k+1}{}^2 - y_{k+1} + \frac{1}{4} - r^2$$

Recurrence Relation for p_k

$$p_k = x_k{}^2 + 2x_k + 1 + y_k{}^2 - y_k + \frac{1}{4} - r^2$$

$$p_{k+1} = x_k{}^2 + 4x_k + 4 + y_{k+1}{}^2 - y_{k+1} + \frac{1}{4} - r^2$$

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

$$= p_k + 2x_{k+1} + 1 + \underline{\left(y_{k+1}{}^2 - y_k{}^2\right)} - \underline{\left(y_{k+1} - y_k\right)}$$

One Last Term ...

$$\left(y_{k+1}{}^{2}-y_{k}{}^{2}\right)-\left(y_{k+1}-y_{k}\right)=?$$

If $y_{k+1} = y_k$, then:

$$\left(y_{k}{}^{2}-y_{k}{}^{2}\right)-\left(y_{k}-y_{k}\right)=0$$

If $y_{k+1} = y_k - 1$, then:

$$\left(\left(y_k-1\right)^2-y_{k}{}^{2}\right)-\left(\left(y_k-1\right)-y_k\right)=$$

$$\left(y_{k}{}^{2}-2y_k+1-y_{k}{}^{2}\right)-\left(-1\right)=$$

$$-2y_k+2=-2\left(y_k-1\right)$$

Initial Values

$$(x_0, y_0) = (0, r)$$

$$p_0 = f_{circle}\left(0+1, r-\frac{1}{2}\right)$$

$$= 1 + \left(r - \frac{1}{2}\right)^2 - r^2$$

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

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

Bresenham Midpoint Circle Algorithm

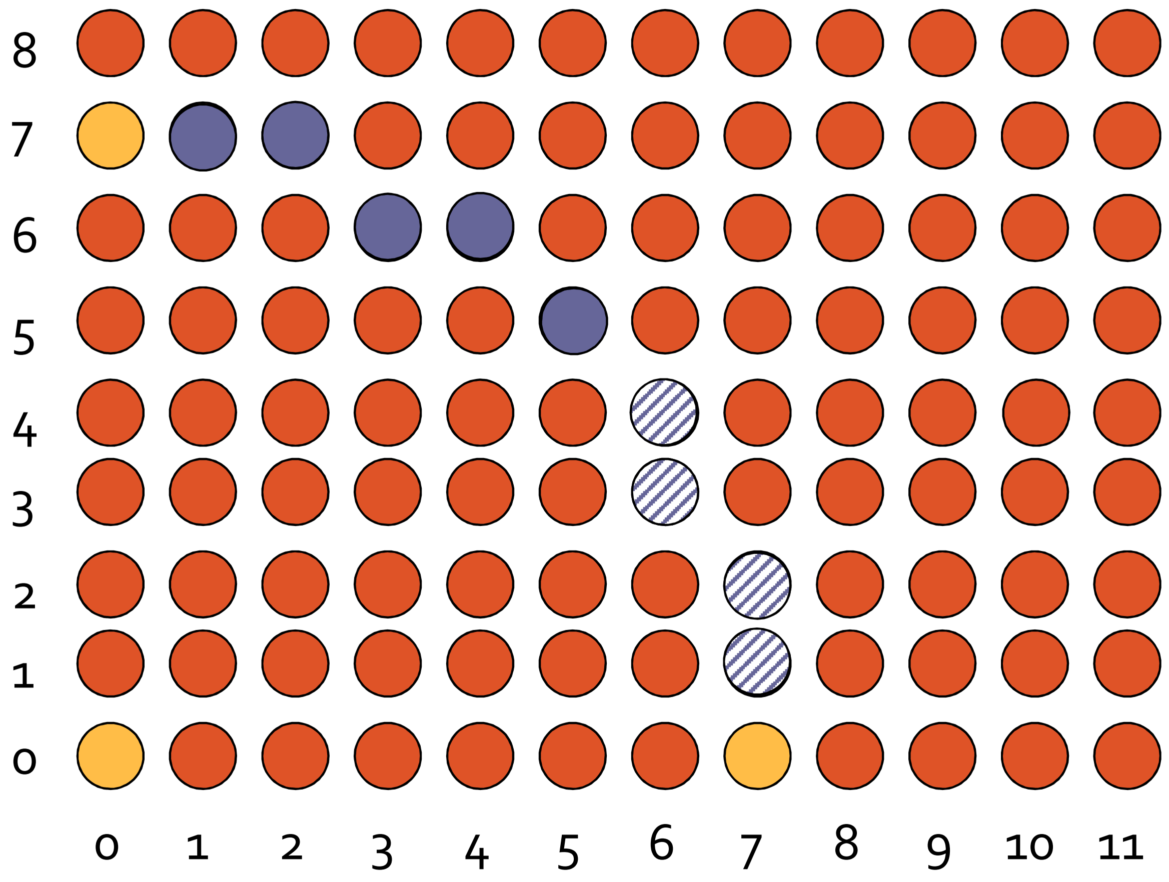
At each point:

If $p_k < 0$:
$$\text{Plot}\left(x_k + 1, y_k\right)$$
$$p_{k+1} = p_k + 2\left(x_k + 1\right) + 1$$

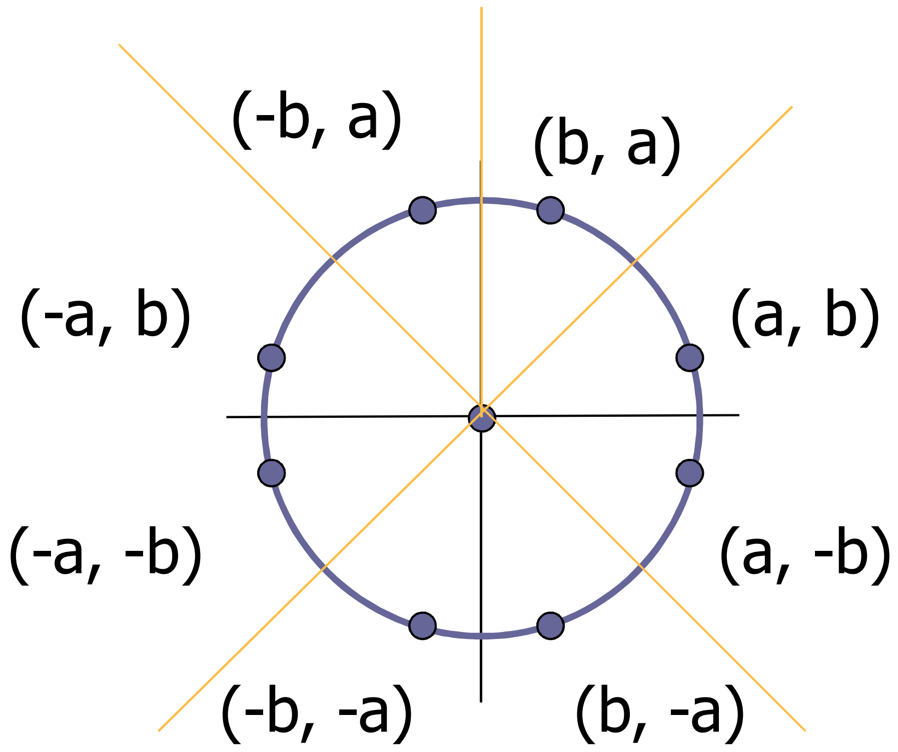
If $p_k >= 0$:
$$\text{Plot}\left(x_k + 1, y_k - 1\right)$$
$$p_{k+1} = p_k + 2\left(x_k + 1\right) + 1 - 2\left(y_k - 1\right)$$

Circle Example 1

r = 7



Symmetry Optimization



Calculate points for one octant; replicate in other seven

Midpoint Circle Algorithm

1. Input radius r and circle center (x_c, y_c), and obtain the first point on the circumference of a circle centered on the origin as

$$(x_0, y_0) = (0, r)$$

2. Calculate the initial value of the decision parameter as

$$p_0 = \frac{5}{4} - r$$

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) and

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

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

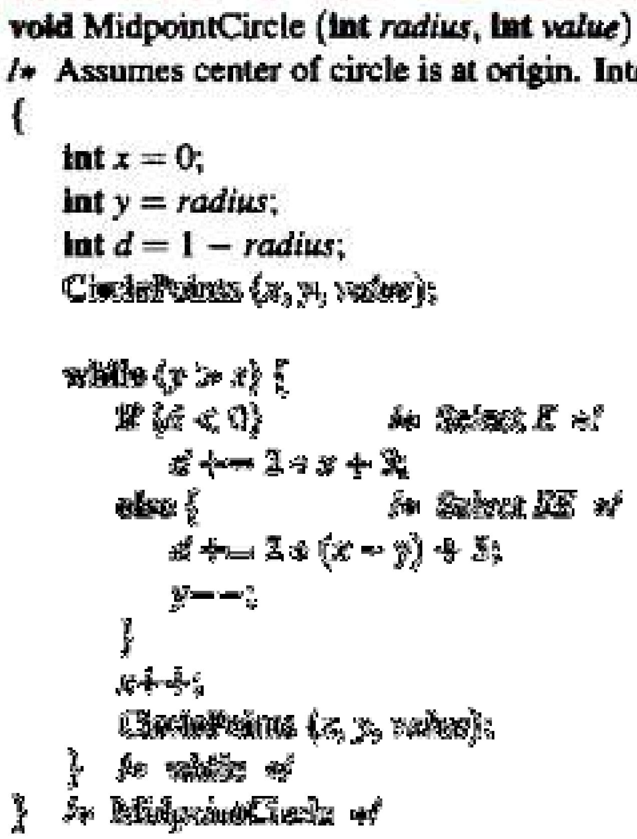
$$p_{k+1} = p_k + 2x_{k+1} + 1 - 2y_{k+1}$$

   where $2x_{k+1} = 2x_k + 2$ and $2y_{k+1} = 2y_k - 2$.
4. Determine symmetry points in the other seven octants.
5. Move each calculated pixel position (x, y) onto the circular path centered on (x_c, y_c) and plot the coordinate values:

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

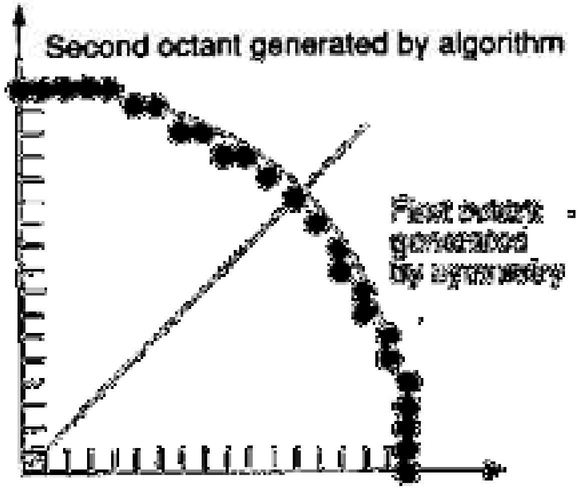
6. Repeat steps 3 through 5 until $x \geq y$.

Midpoint Circle Algorithm (cont.)



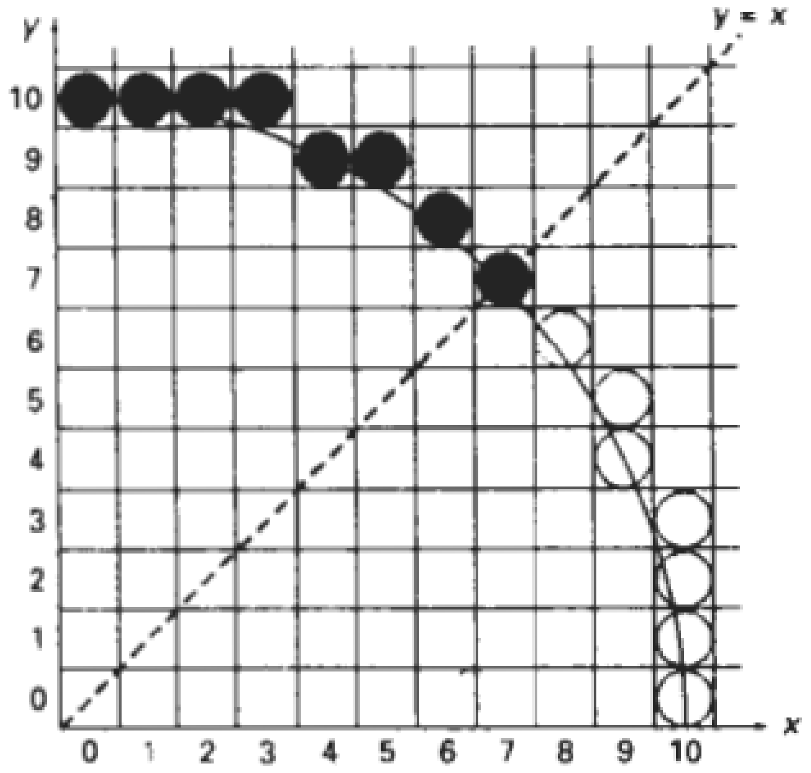
```
void MidpointCircle (int radius, int value)
/* Assumes center of circle is at origin. Integer arithmetic only */
{
    int x = 0;
    int y = radius;
    int d = 1 - radius;
    CirclePoints (x, y, value);

    while (y > x) {
        if (d < 0)              /* Select E */
            d += 2 * x + 3;
        else {                  /* Select SE */
            d += 2 * (x - y) + 5;
            y--;
        }
        x++;
        CirclePoints (x, y, value);
    } /* while */
} /* MidpointCircle */
```


Second octant generated by algorithm
First octant generated by symmetry

Example

- r=10



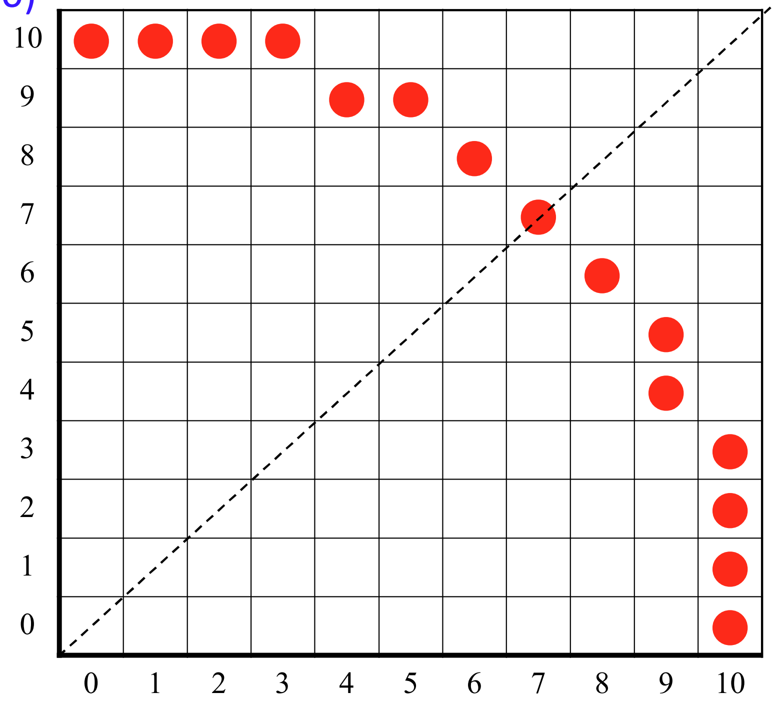
Example (Mid Point Circle Algorithm)

$r = 10$

$p_0 = 1 – r = -9$ (if r is integer round $p_0 = 5/4 – r$ to integer)

Initial point $(x_0, y_0) = (0, 10)$

i	p_i	x_{i+1}, y_{i+1}	$2x_{i+1}$	$2y_{i+1}$
0	-9	(1, 10)	2	20
1	-6	(2, 10)	4	20
2	-1	(3, 10)	6	20
3	6	(4, 9)	8	18
4	-3	(5, 9)	10	18
5	8	(6, 8)	12	16
6	5	(7, 7)	14	14



Another method:
Approximate it using a polyline.

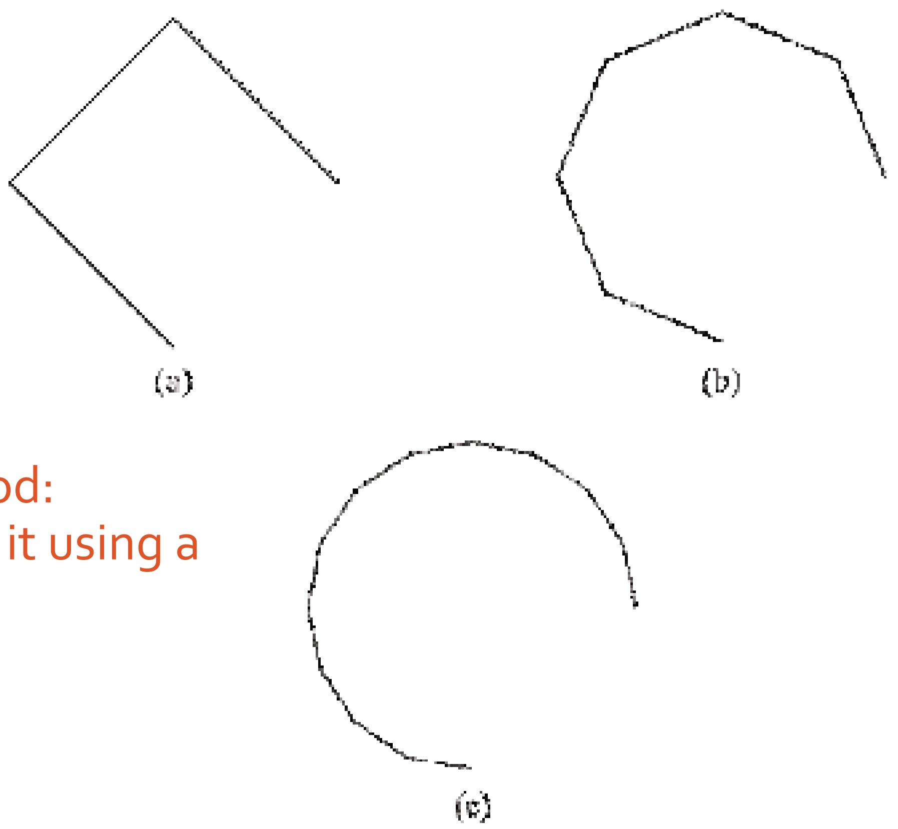


Figure 3-15

A circular arc approximated with (a) three straight-line segments, (b) six line segments, and (c) twelve line segments.

POLYGON

Filled Area Primitives

•**Polyline** - A chain of connected line segments.
•**Polygon** - When starting point and terminal point of any polyline is same, i,e when polyline is closed then it is called polygon

Filled Area Primitives cont..

Basic approaches to Area-Filling –

•Scan-line Method

Determine the overlap intervals for scan lines that cross the area. It is typically used in general graphics packages to fill polygons, circles, ellipses

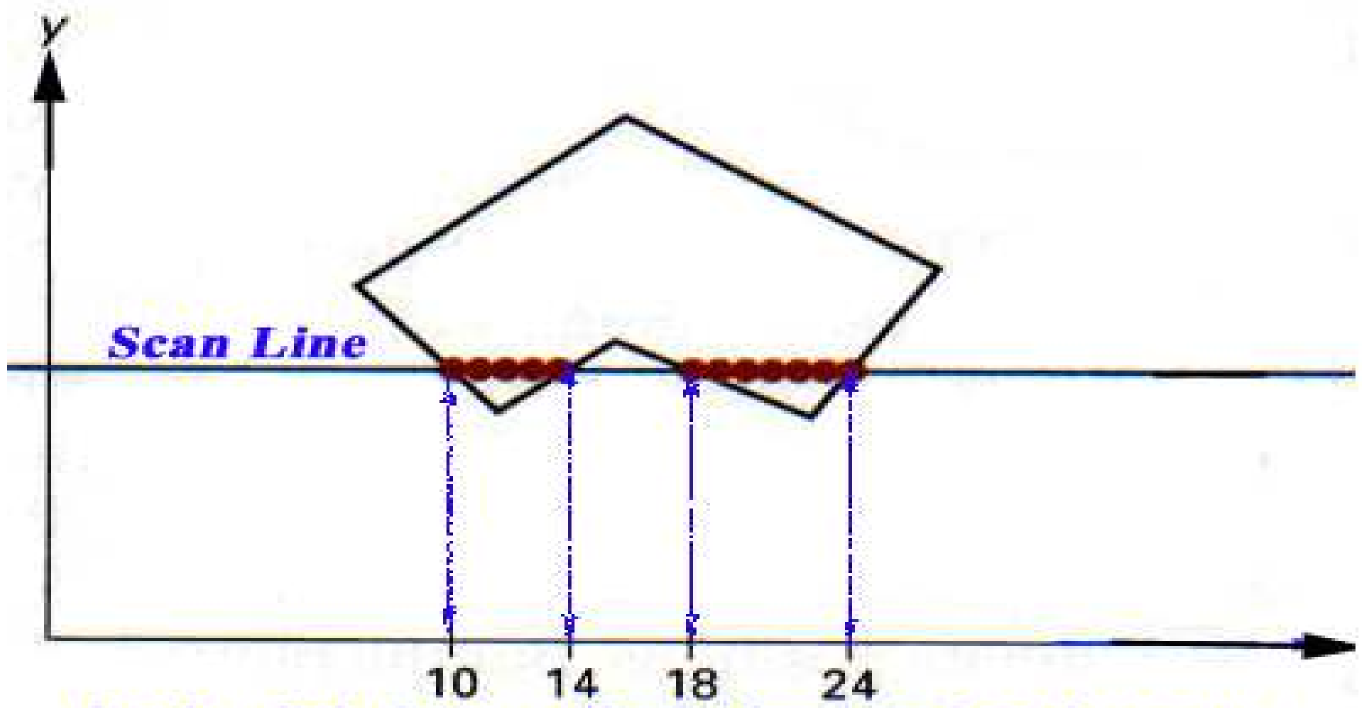
- Inside Outside Test

•Fill Method

Start from a given interior position and paint outward from this point until we encounter the specified boundary conditions. Useful with more complex boundaries and in interactive painting systems.

1. Boundary Fill
2. Flood Fill

Filled Area Primitives cont..



Interior pixels along scan line passing through a polygon area

Filled Area Primitives cont..

Boundary – Fill Algorithms

- Start at a point inside a region and paint the interior outward toward the boundary. If the boundary is specified in a single color, the fill algorithm proceeds outward pixel by pixel until the boundary color is encountered.

- It is useful in interactive painting packages, where interior points are easily selected.

- The inputs of the this algorithm are:
  - Coordinates of the interior point (x, y)
  - Fill Color
  - Boundary Color

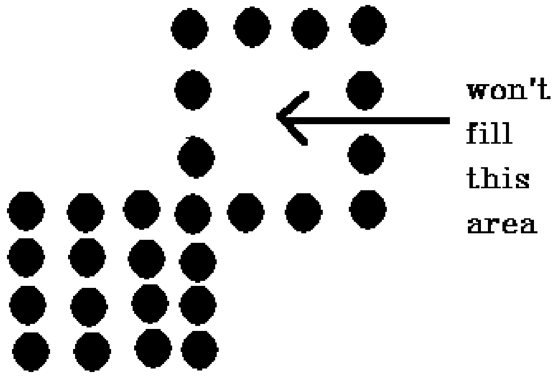
Filled Area Primitives cont..

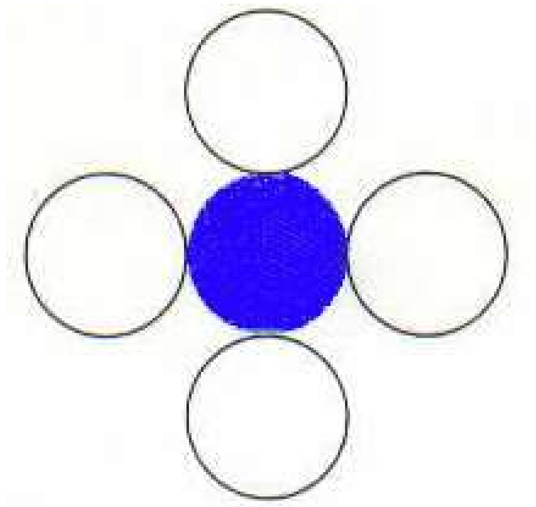
- Sometimes we want to fill in (or recolor) an area that is not defined within a single color boundary. We can paint such areas by replacing a specified interior color instead of searching for *a* boundary color value. This approach is called a flood-fill algorithm.

We start from a specified interior point (*x*, *y*) and reassign all pixel values that are currently set to a given interior color with the desired fill color.

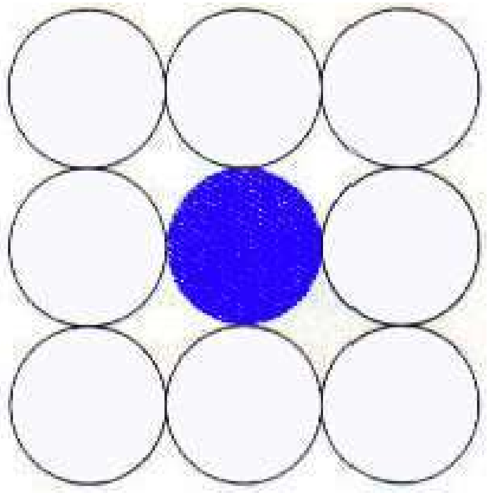
4 connected pixel

- In this technique 4-connected pixels are used as shown in the figure. We are putting the pixels above, below, to the right, and to the left side of the current pixels and this process will continue until we find a boundary with different color.

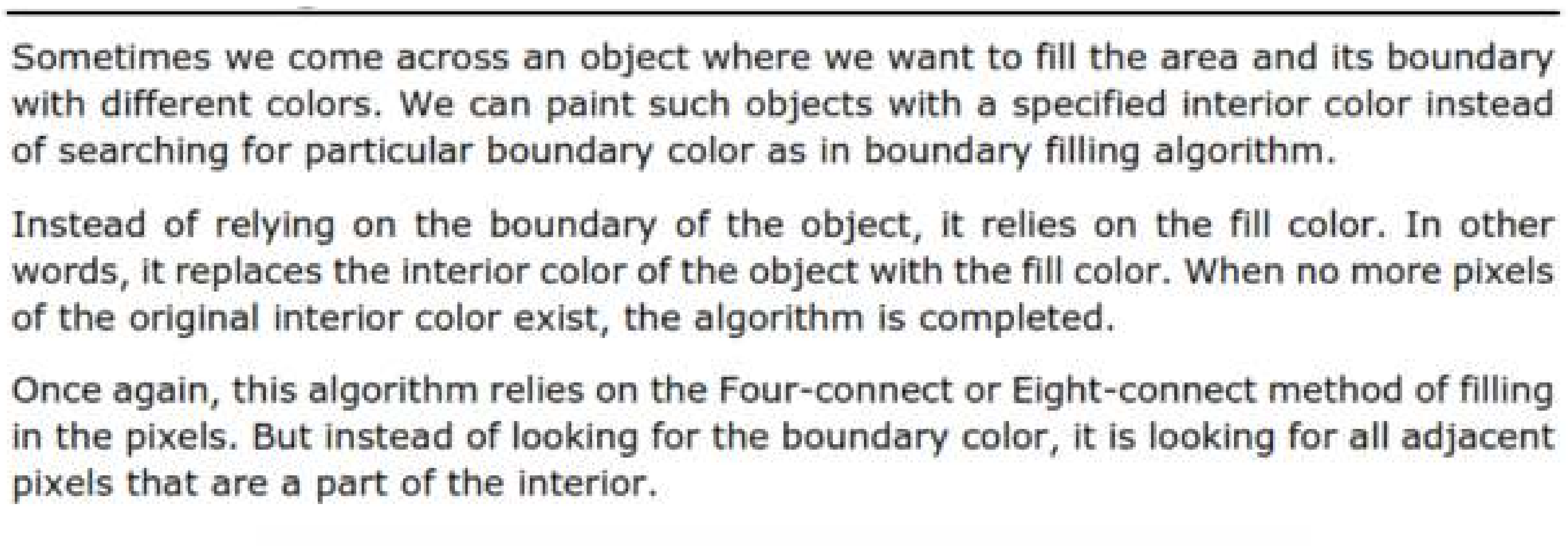




8-connected pixel



Flood Fill algorithm



Sometimes we come across an object where we want to fill the area and its boundary with different colors. We can paint such objects with a specified interior color instead of searching for particular boundary color as in boundary filling algorithm.

Instead of relying on the boundary of the object, it relies on the fill color. In other words, it replaces the interior color of the object with the fill color. When no more pixels of the original interior color exist, the algorithm is completed.

Once again, this algorithm relies on the Four-connect or Eight-connect method of filling in the pixels. But instead of looking for the boundary color, it is looking for all adjacent pixels that are a part of the interior.