Primitives

Line Drawing - 2 algorithms

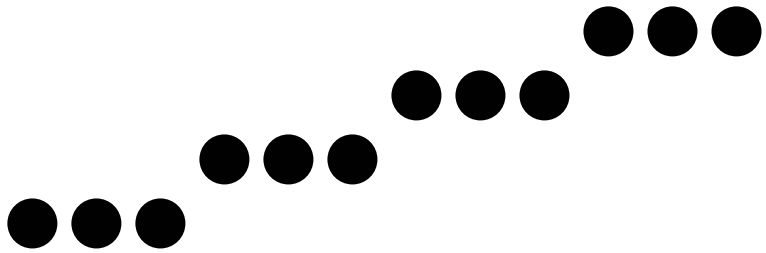
- DDA
- Bressenham's

Character Generation

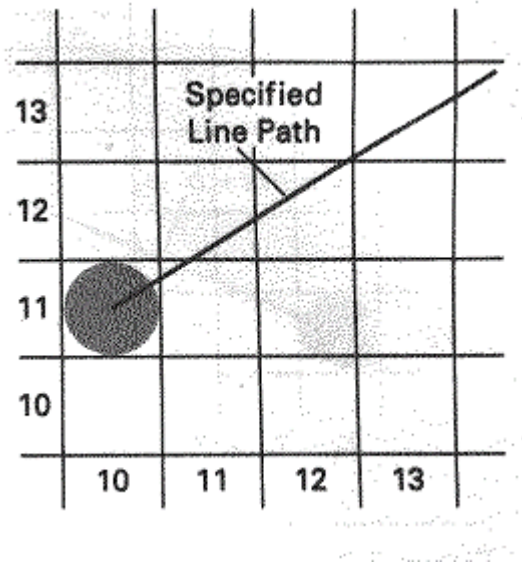
Attributes of Primitives

Filled area Primitives

Line Drawing



The Jaggies

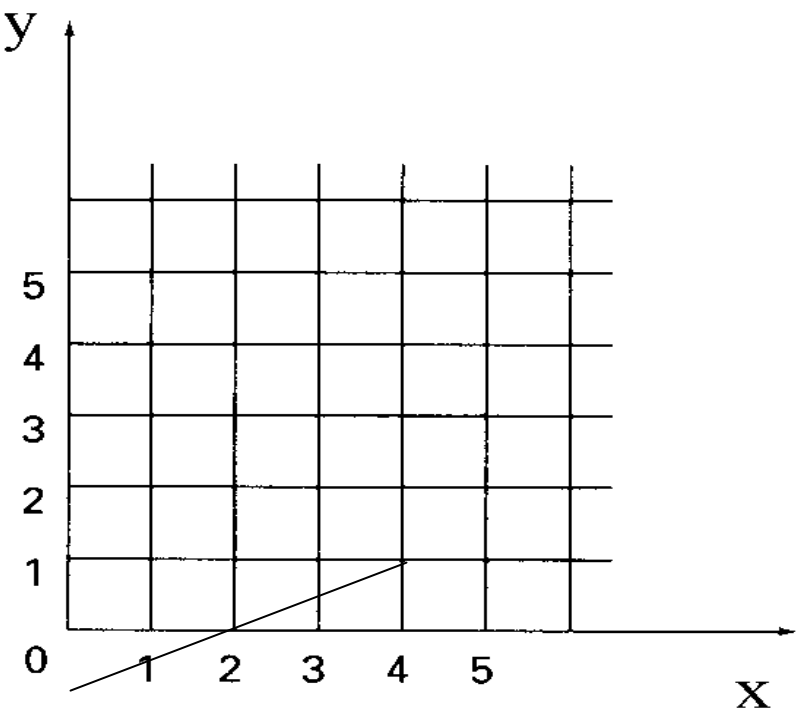


- Digital Differential Analyser (DDA)
  - line-scan conversion algorithm
  - Works out which pixels to plot for a given line definition

Example 1

$$y = \frac{1}{2}x - 1$$

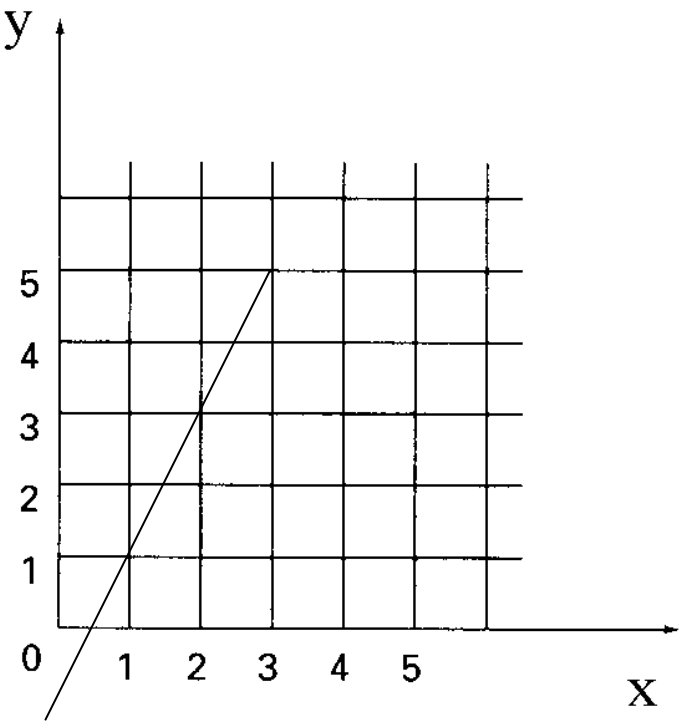
x	0	1	2	3	4
y	-1	-1/2	0	1/2	1



Example 2

$$y = 2x - 1$$

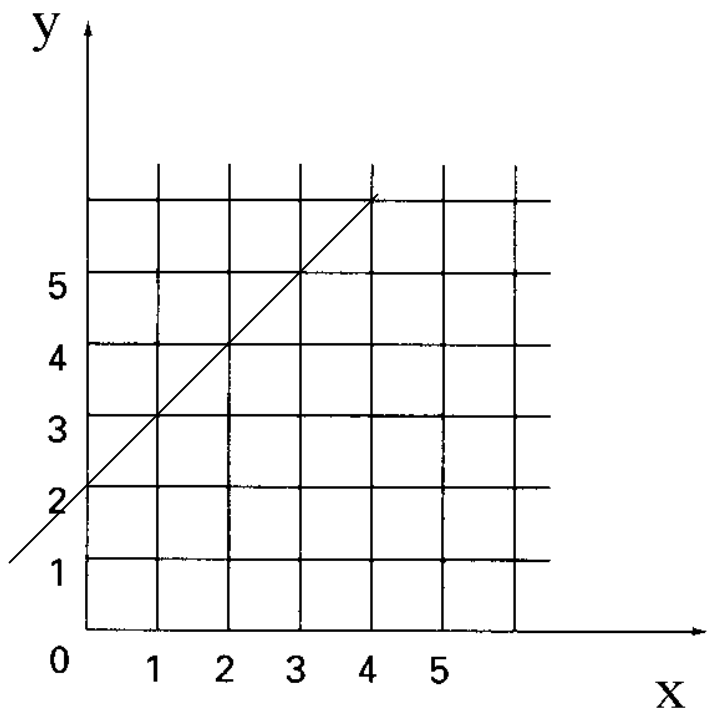
x	0	1	2	3	4
y	-1	1	3	5	7



Example 3

$$y = x + 2$$

x	0	1	2	3	4
y	2	3	4	5	6



Note that

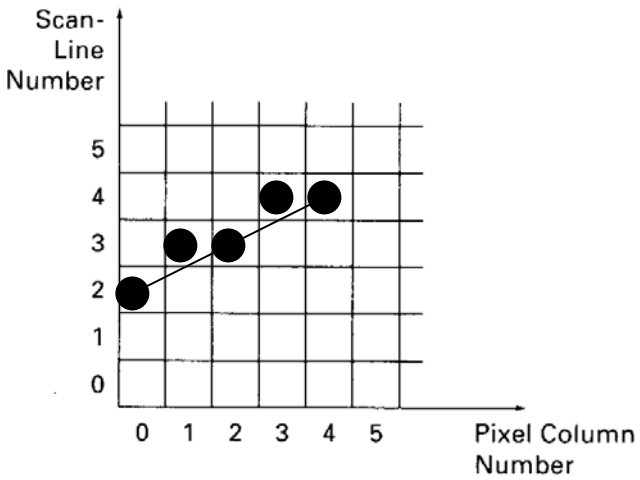
y = (gradient.x) + (y-intercept)

Gradients

- Note
  - If gradient is more than (>) 1 the slope is steep
  - If gradient is less than (<) 1 the slope is shallow
  - If the gradient is 1, the slope will run at 45º.

DDA

Differential Digital Analyser



Draw $y = \frac{1}{2}x + 2$

x	0	1	2	3	4
y	2	2.5	3	3.5	4

What is the gradient?

- If m#1 sample x at unit intervals. $^{a}x=1$.

$$y = y_{prev} + m$$

m#1for shallow slope

m>1 for a steep slope

x	0	1	2	3	4
y (y_{prev}+m)	2	2.5	3	3.5	4
y (plot)	2	3	3	4	4

Values of y to plot must be rounded to nearest integer

Gradients greater than 1

- If m>1 sample y at unit intervals. ᵃy=1.

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

- Negative gradients
  - use absolute value of gradient for sample decision
  - so if m=!1.5, sample at y unit intervals. ᵃy=!1

DDA Examples

- Produce graphs and pixel displays (as DDA OHP) for

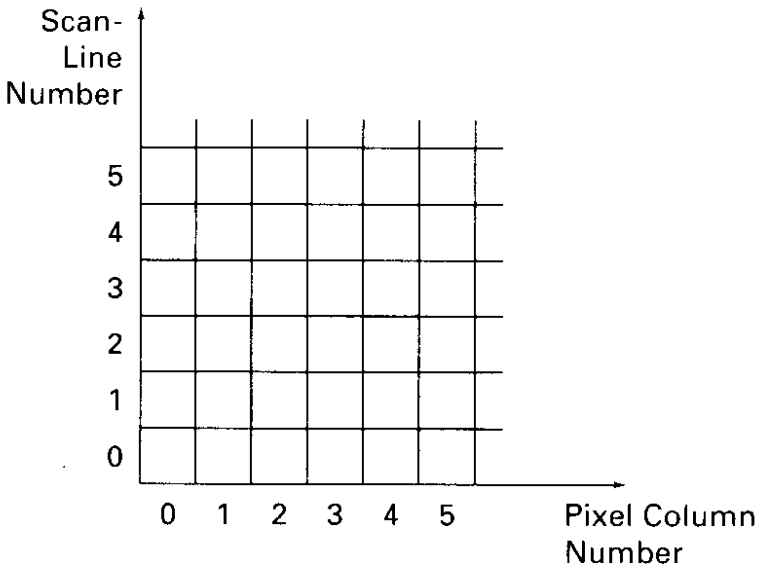
$$y = \frac{1}{2}x - 1$$

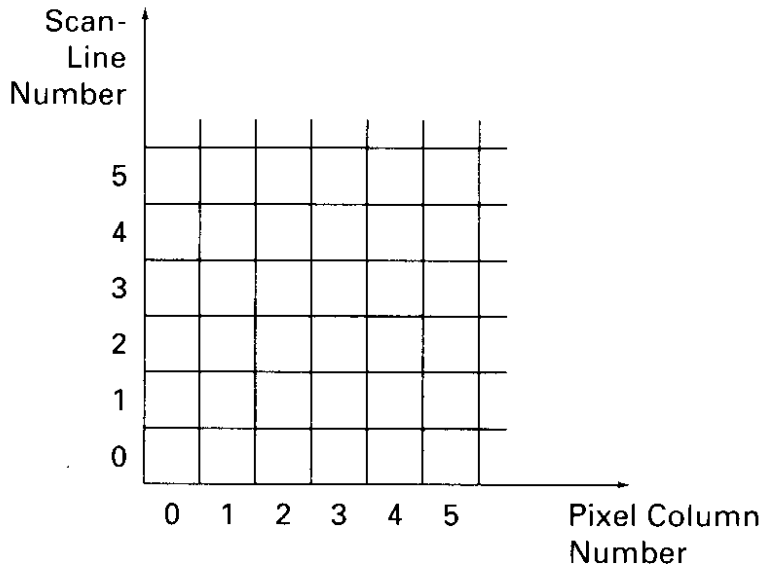
$$y = 2x - 3$$

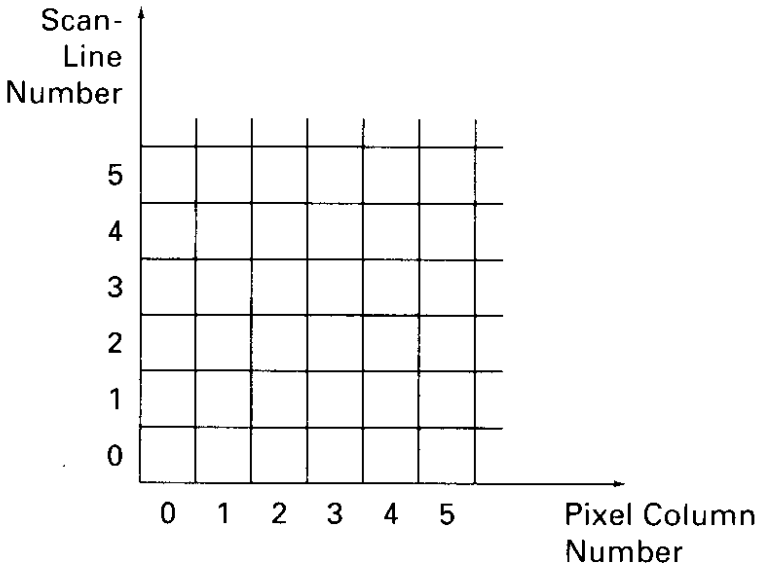
$$y = \frac{1}{2}x + 3$$

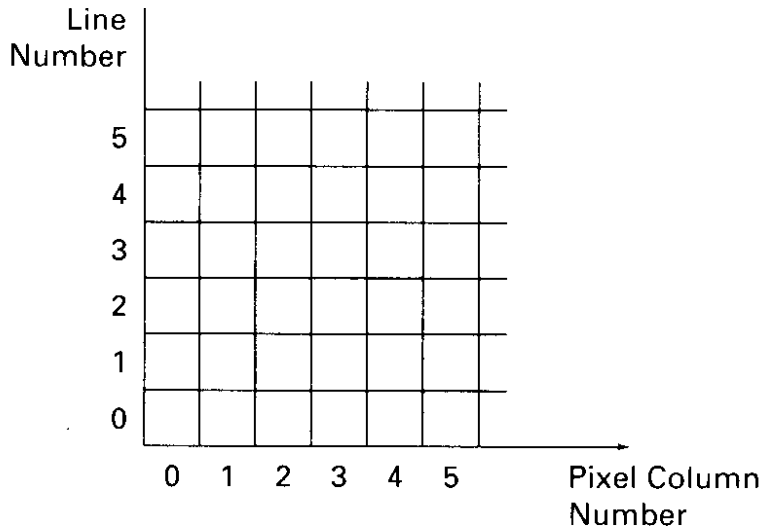
$$y = 3x - 9$$

Graph/pixel displays

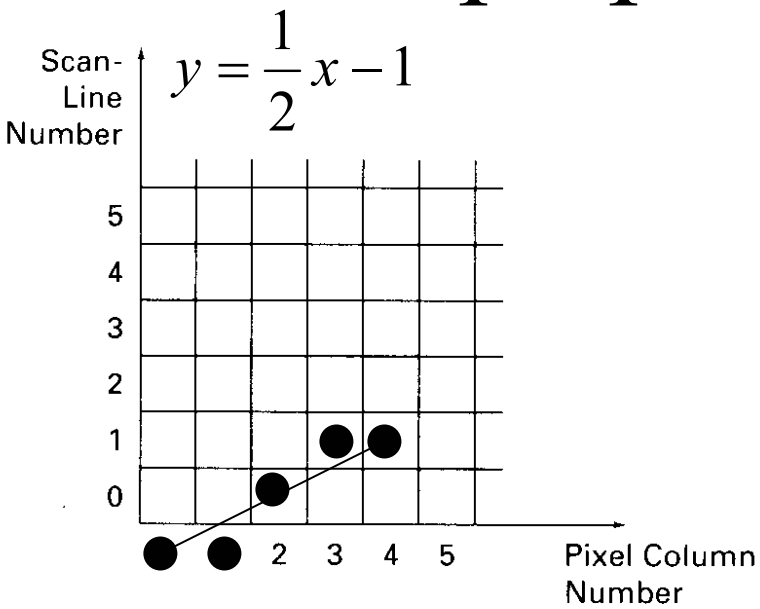


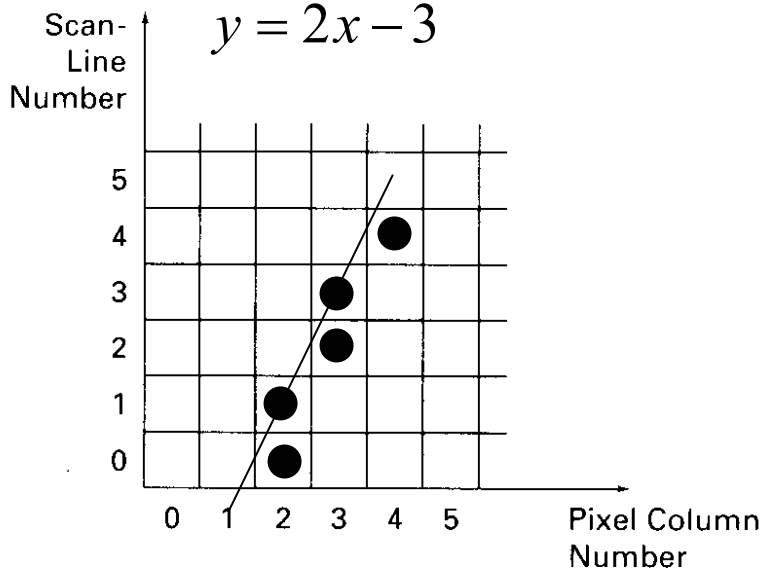


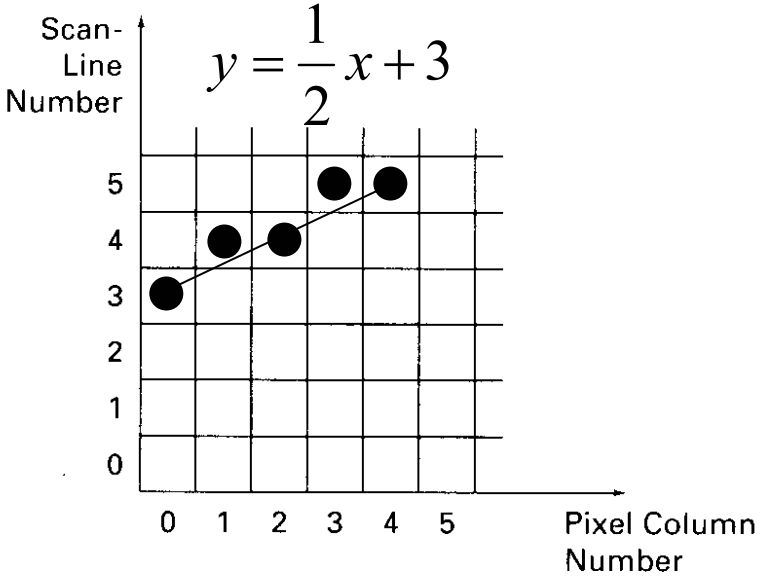


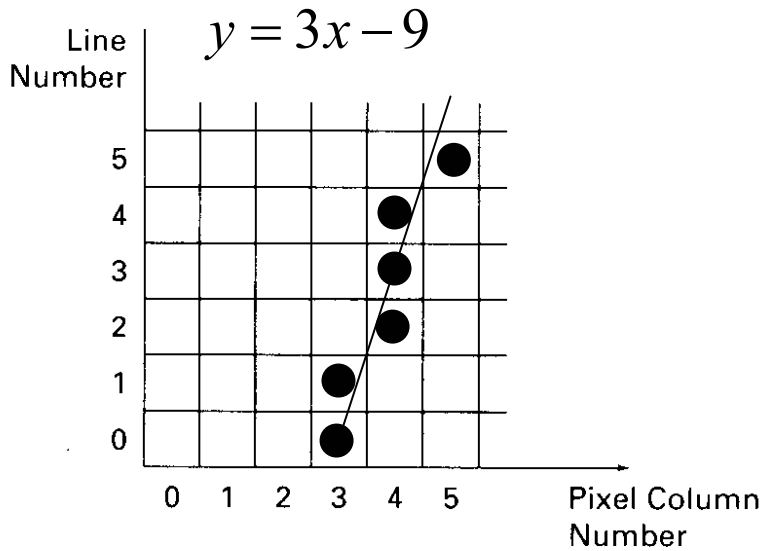


Graph/pixel Answers









DDA

- Faster than using the equation to calculate the line
- Not very accurate
  - due to roundoff
  - will float away from the true path for longer lines
- Slow as real (floating point) arithmetic
- only useful for straight lines

Bressenham

- Bressenham's line alogorithm is an improvement
- First revise finding gradients

Gradients

- What is the gradient of a line from 10,12 to 15,22?

$$\frac{y_2 - y_1}{x_2 - x_1} = m \qquad \frac{22 - 12}{15 - 10} = \frac{10}{5} = 2$$

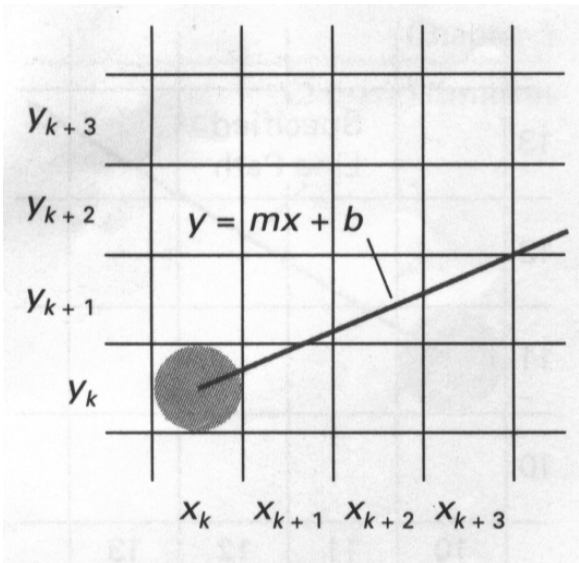
  - Find the gradients of
    - line from 1,3 to 2,9
    - line from 5,2 to 8,8

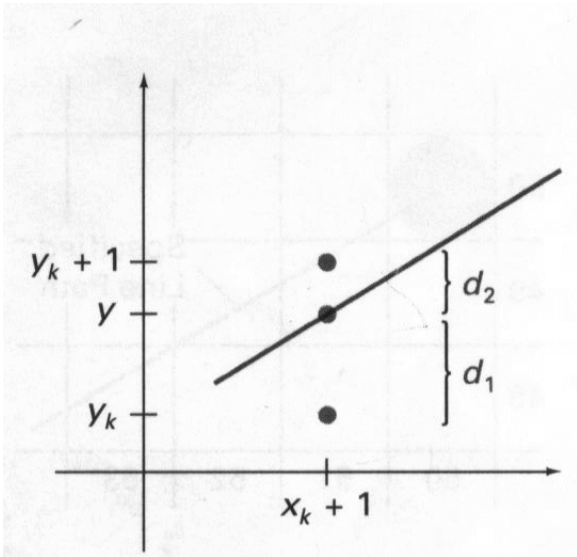
Ans 6

2

Bressenham's Line Algorithm 1

- Fast, accurate, efficient algorithm
- uses integer calculations
- can be adapted for circles and other curves
- decides which of two pixels is closer to the ideal path





Bressenham's Line Algorithm 2

- Value of $y = m(x_k+1)+b$
- Then $d_1 = y-y_k = m(x_k+1)+b-y_k$
- and $d_2 = (y_k+1)-y = y_k+1- m(x_k+1)+b$
- Difference is $d_1- d_2 = 2m(x_k+1)-2\ y_k +2b-1$
  - If answer is positive upper pixel plotted, if negative then lower pixel is plotted.
- Decision parameter (sense of p_k and $d_1- d_2$ is the same)
  - $p_k = Î x(d_1- d_2) = 2 Î y.x_k-2 Î x.y_k+c$
- From this we get a recurrence relation and can find p_k+1 in terms of p_k.

Bressenham's Line Algorithm 3

- If |m| <1
  - put left endpoint in (x_0,y_0) and plot
  - calculate dx, dy, 2dy and (2dy-2dx)
  - get start value p0=2dy-dx
  - at each x_k, from k_1, dx times
    - **test** if $p_k<0$ then plot (x_k+1, y_k) and set $p_k+1=p_k+2dy$
    - **else** plot (x_k+1, y_k+1) and set $p_k+1=p_k+2dy-2dx$
- If |m| >1 sample y rather than x
    - we will not cover a practical example of this

Bressenham's Line Algorithm 4

- For a line from 20,10 to 30,18 we have a slope of 0.8, dx=10 and dy=8
- p0=2dy-dx=6
- calculate **2dy=16**, 2dx=20, **2dy-2dx= -4**
- plot 20,10 first

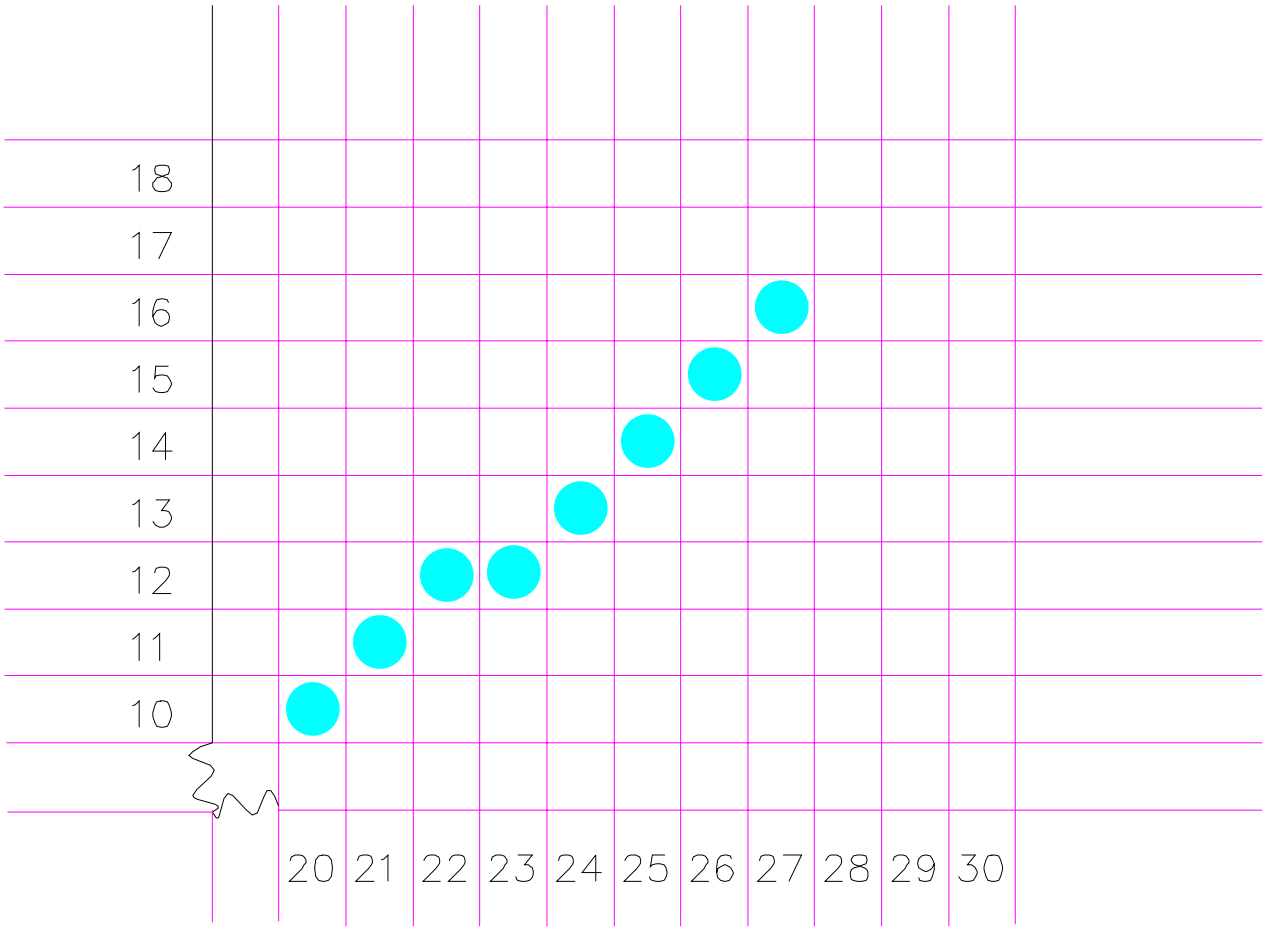
Bressenham's Line Algorithm 5

k	**p_k**	**X_k+1, y_k+1**	
0	6	21,11	
1	2	22,12	
2	-2	23,12	negative
3	14	24,13	
4	10	25,14	
5	6	26,15	
6	2	27,16	

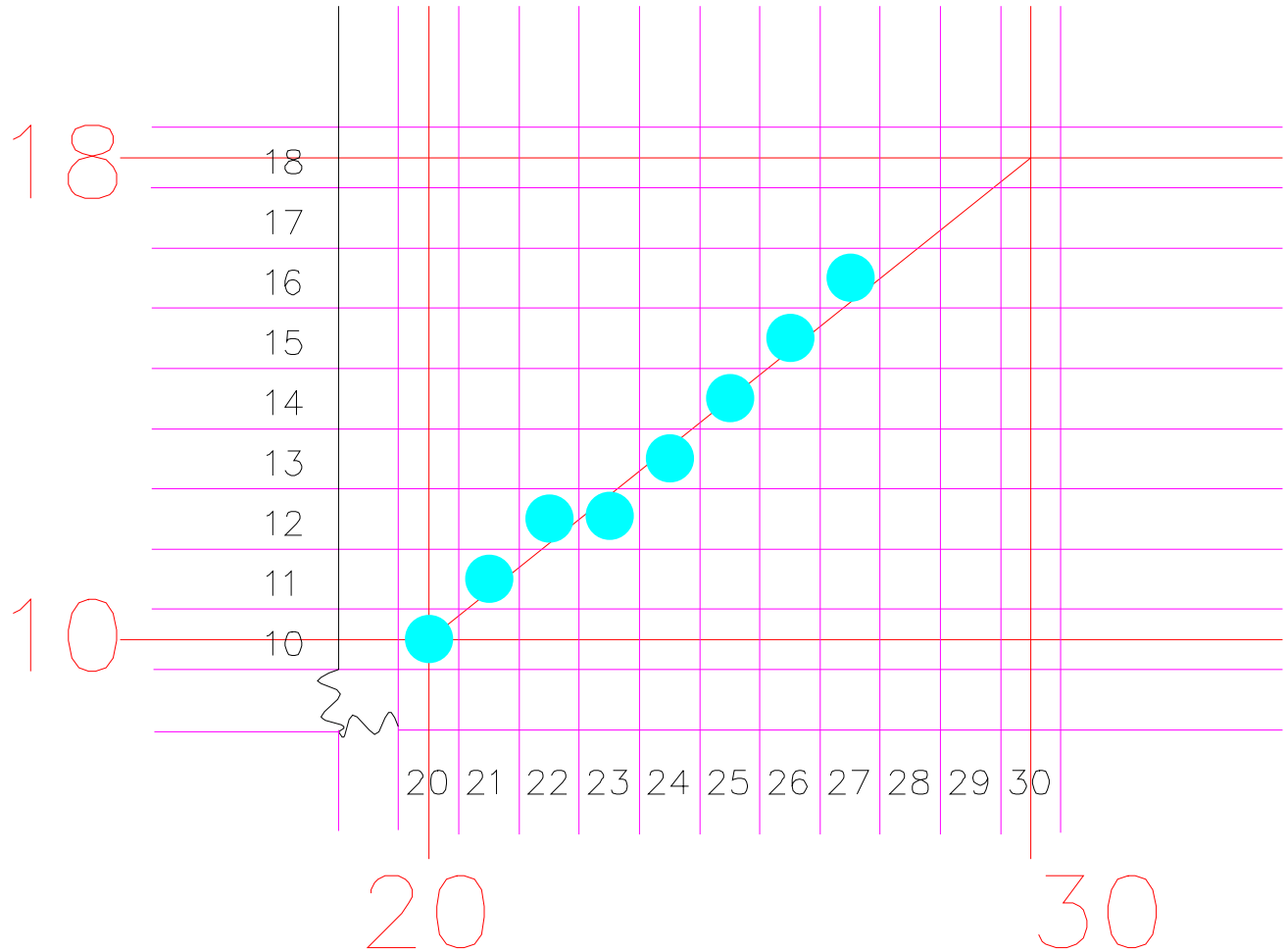
test if $p_k<0$ then plot (x_k+1, y_k) and set $p_k+1=p_k+2dy$
else plot (x_k+1, y_k+1) and set $p_k+1=p_k+2dy-2dx$

Bressenham's Line Algorithm 6

Bressenham's Line Algorithm 7



Bressenham's Line Algorithm 8



Worked Example

A line is described by the equation y=0.7x+5 For the values x=10 to x=20

1. Plot the line on a graph
2. Calculate the pixel positions using the DDA algorithm
3. On the same graph plot the DDA pixel positions
4. Calculate the pixel positions using Bressenham's algorithm.
5. On the same graph plot Bressenham's pixel positions (use another colour)

Solution 1

1 Plot the line.

Work out the y values for the start and end of the line.

X	10	20
y	12	19

Plot these values on a graph.

Solution 2

2 Use the DDA algorithm to find the plot positions

The slope is 0.7, this is less than 1 so sample x at unit intervals.

X	10	11	12	13	14	15	16…….
y_{prev+m}	12	12.7	13.4	14.1	14.8	……...	
y_{plot}	12	13	13	14……...			

3 Plot these values on the graph.

Solution 3

4 Bressenham's line Alogrithm

test if $p_k<0$ then plot (x_k+1, y_k) and set $p_k+1=p_k+2dy$
else plot (x_k+1, y_k+1) and set $p_k+1=p_k+2dy-2dx$
First value of p_k is $p_0=2dy-dx$

For a line from 10,12 to 20,19 we have a slope of 0.7, dx=10 and dy=7

calculate **2dy=14**, 2dx=20, **2dy-2dx= -6**

Calculate first value of p_k. $p_0 =2dy-dx=4$

Start point of line is known to be 10,12

Solution 4

k	**p_k**	**X_k+1, y_k+1**	**k**	**p_k**	**X_k+1, y_k+1**
0	4	11,13	5	-6	16,16
1	-2	12,13	6	…….	
2	12	13.14	7		
3	6	14,15	8		
4	0	15,16	9		

test if $p_k<0$ then plot (x_k+1, y_k) and set $p_k+1=p_k+2dy$
else plot (x_k+1, y_k+1) and set $p_k+1=p_k+2dy-2dx$

5 Plot these values on the graph using a different colour.

Problems

- Plot each of the following equations
  - as a line
  - as pixels using DDA
  - as pixels using Bressenham's

Qu 1 $y = \frac{1}{2}x + 4$ From x=0 to x=10

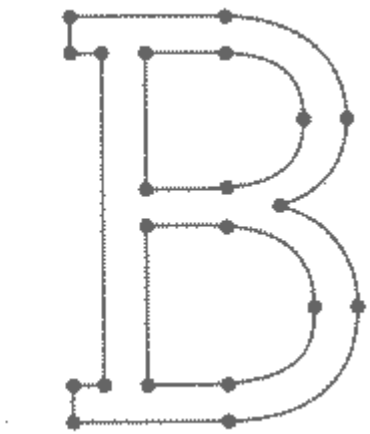
Qu 2 $y = 0.2x + 3$ From x=10 to x=20

Character Generation

- 2 different representations used to store fonts
  - Bitmaps and Outline fonts

1	1	1	1	1	1	0	0
0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	0
0	1	1	1	1	1	0	0
0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	0
1	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0

Bitmap



Outline

Bitmaps

- Rectangular grid to store pattern for each character.
- Easy to define and display
- each variation in size and format must be stored so memory heavy
- or can generate sizes/formats by scaling/transforming existing font but this produces generally poor quality results.

Outline Font

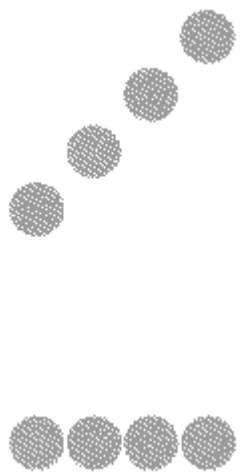
- Characters described using straight line and curved sections. This hollow character will then be filled.
- This definition may be manipulated to produce size/format variations economically
- More time-consuming to create intially

Attributes of Primitives

- Attributes - colour, width,
- Lines specify - width, linetype (dashed, dotted etc), end shape
- Characters specify - format (italic, bold etc), size
- polygons specify - texture map
- The same attributes can be applied to all the primitives of a similar type in a file or each primitive may be described separately

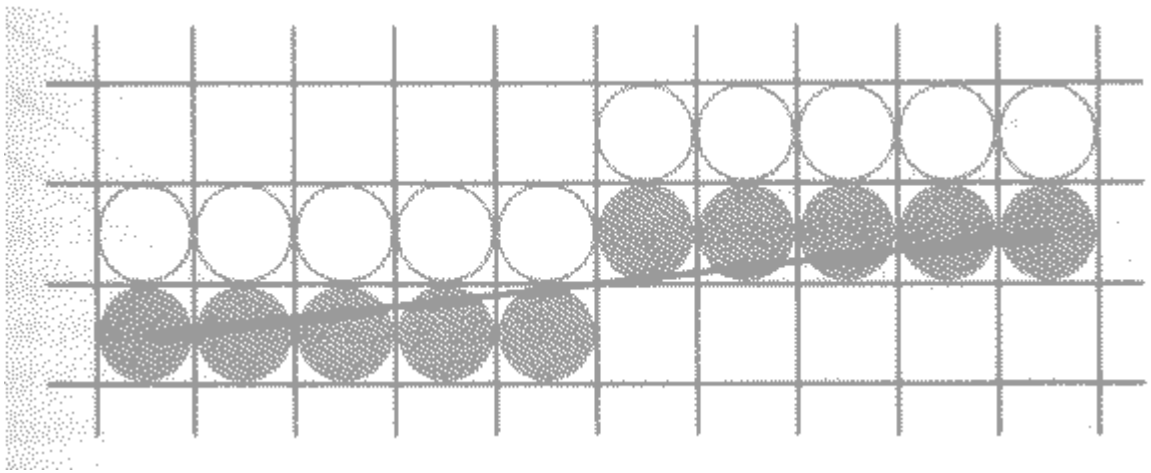
Displaying attributes

- Linetype - Dashed line
  - example 4 pixels displayed then 3 pixels space
  - fine when horiz or vert. But a diagonal line will be longer by a factor of $\sqrt{2}$.
    - This may be adjusted by an extra program which say reduces the 4 pixels to 3 for a diagonal line



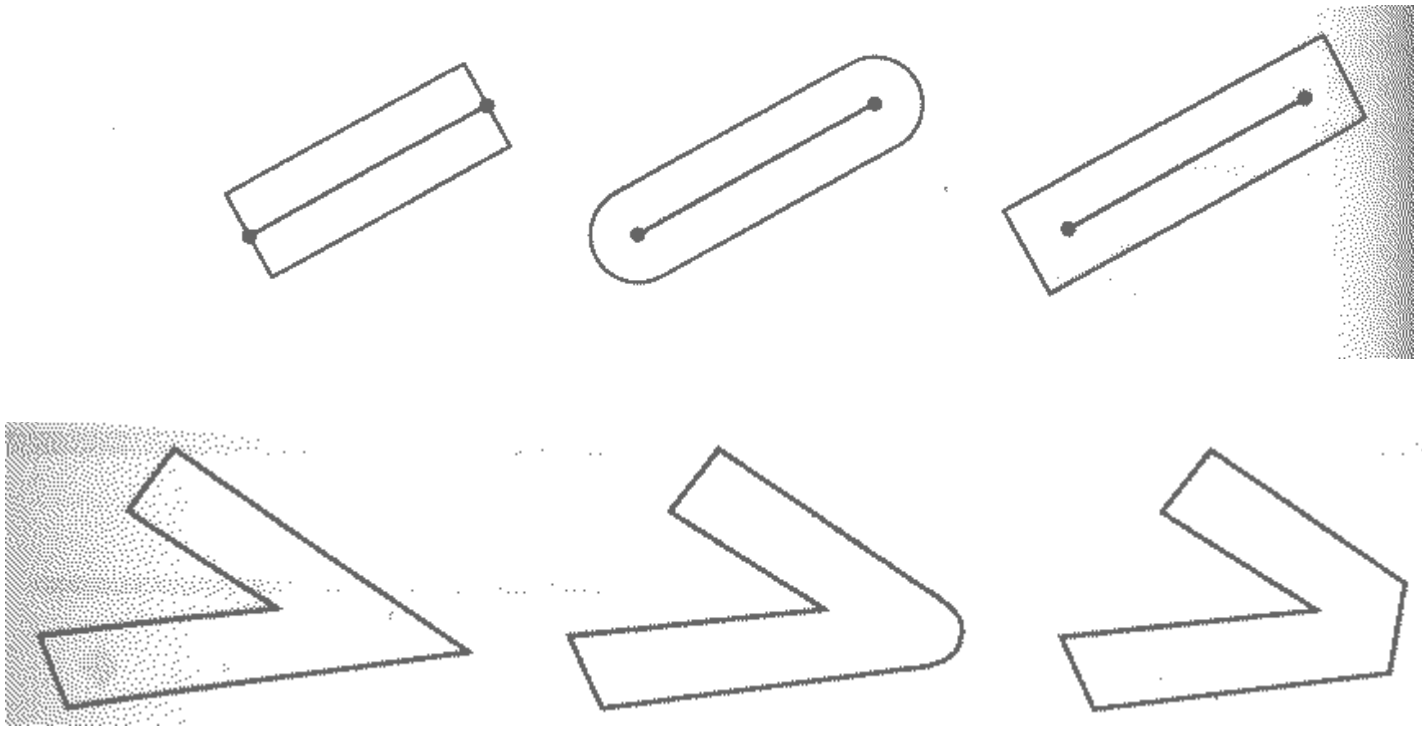
Displaying attributes 2

- Line Width
  - For thicker lines a line parallel may be displayed
  - clearly the widths that may be displayed are limited by the number of pixels on the screen



Displaying attributes 3

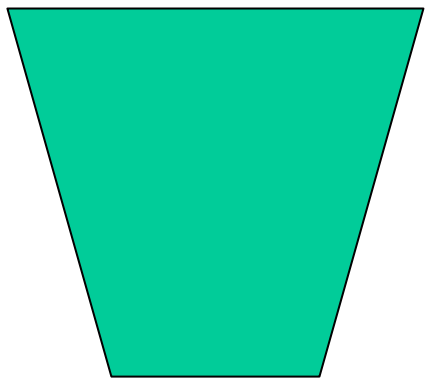
- Ends of thick lines
  - need caps otherwise the ends will be ragged



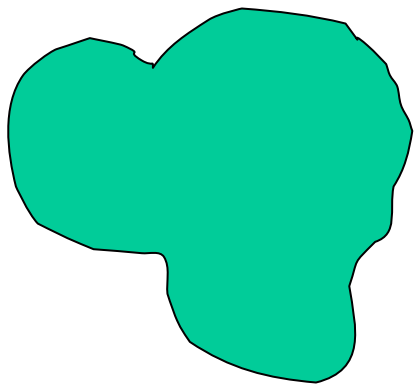
Filled Area Primitives

- 2 Approaches

Scan-line polygon fill algorithm for simple shapes and curves

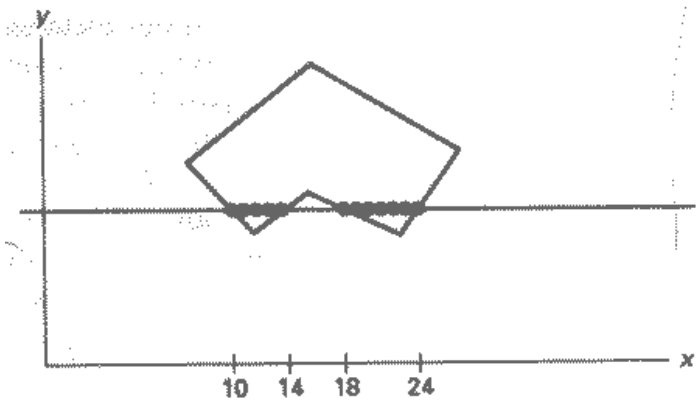


Boundary-fill algorithm for complex boundaries



Scan-line polygon fill algorithm

- Scan a line
- Count the boundaries
- start filling after an odd number



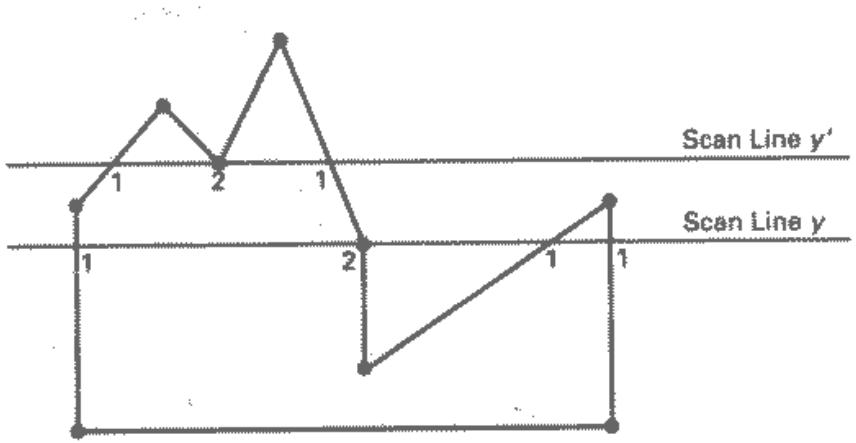
Problem

What about the vertices?

Count them as two intersections?

Scan-line at vertices

- For scan line y' counting vertices as two intersections works.
- For scan line y it doesn't work
  - **Solution - where the sign of the slope remains the same, consider that vertex to create one intersection.**

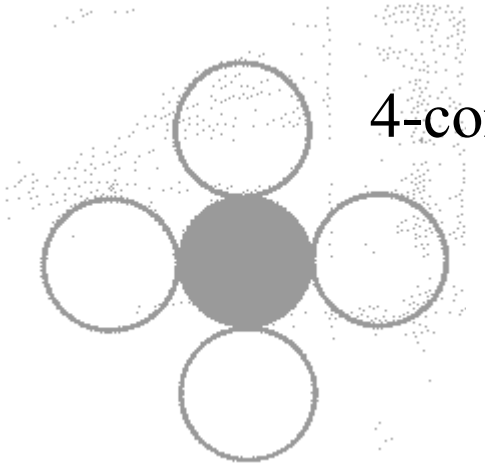


A sorted list of vertices saves time.

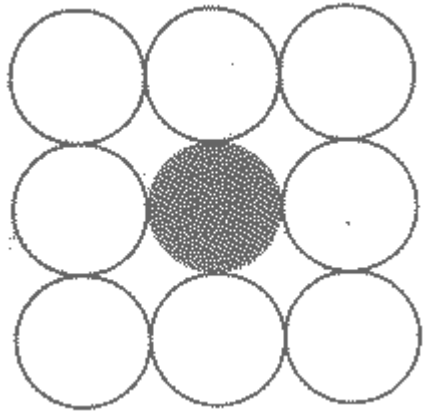
Boundary-Fill algorithm

- Used by interactive packages
- select an interior point
  - fill spreads to boundary
  - 4-connected or 8-connected algorithms

4 and 8 connected fills



4-connected



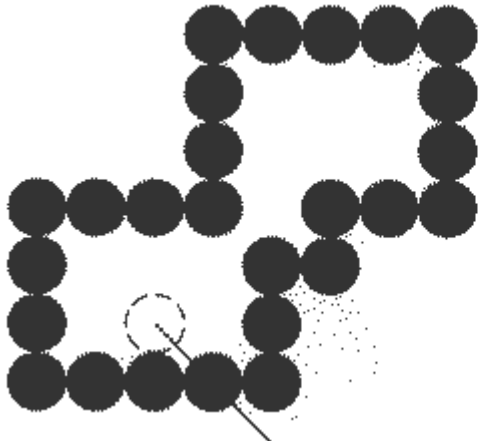
8-connected

User selects a point

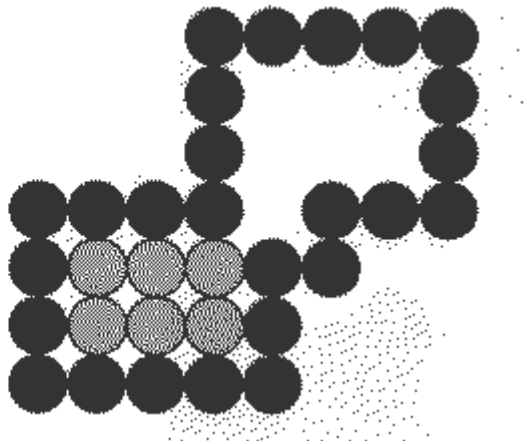
Algorithm checks neighbouring pixels (either 4 or 8) for 3 states - either boundary or fill or needs filling.

The pixel positions which are filled by the algorithm are stacked and in turn their neighbouring pixels are checked.

8-Connected boundary fill area



Start position



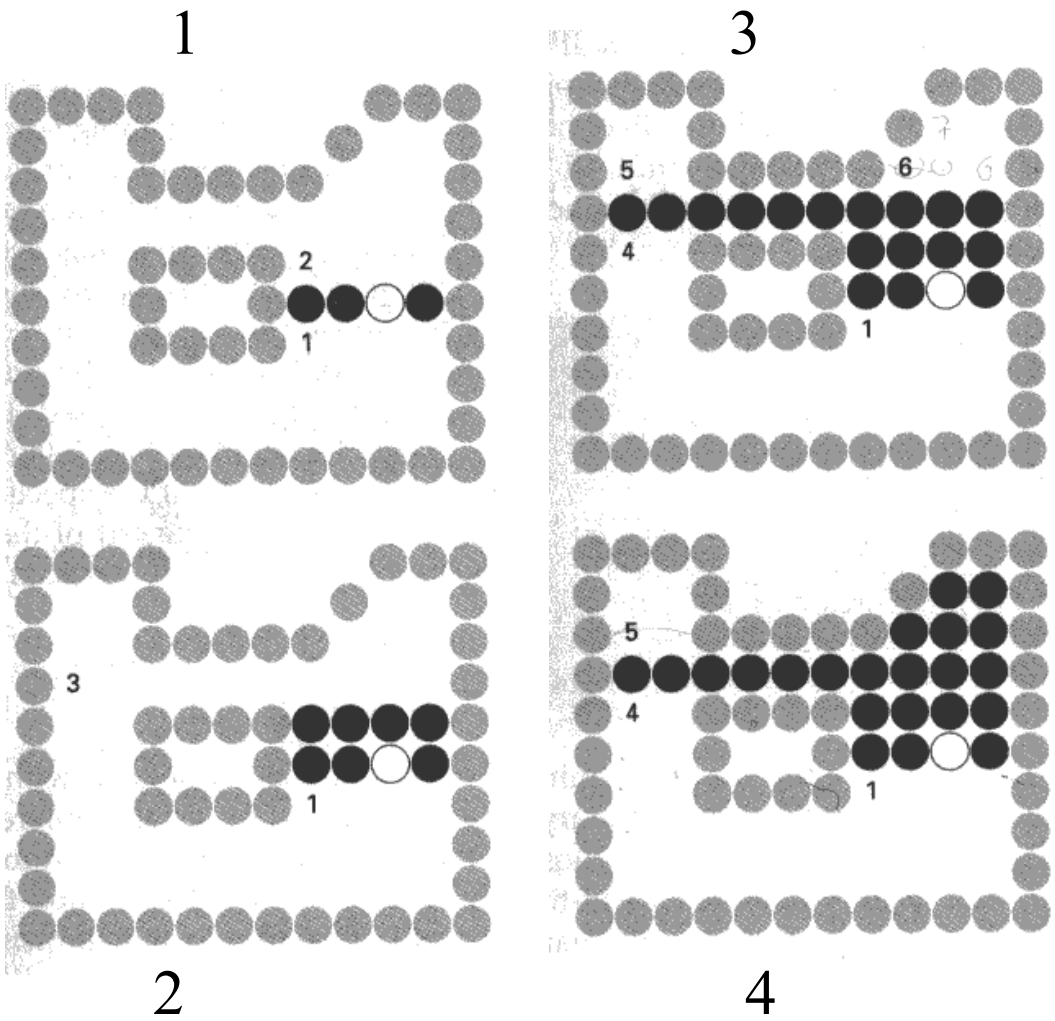
4-connected
boundary fill

Shape will not fill with 4-connected, will fill with 8-connected.

Problems with boundary fill

- May not fill correctly if fill colour is encountered
  - solution - change any pixels that are already the fill colour before applying the fill
- 4 and 8-boundary fill is inefficient as many pixel positions will be stacked and checking is duplicated
  - use a method which fills horizontally

Horizontal fill



From start point scan across, locate next start position (at left hand end of row above)

Numbers indicate stack positions

Fewer stack positions required