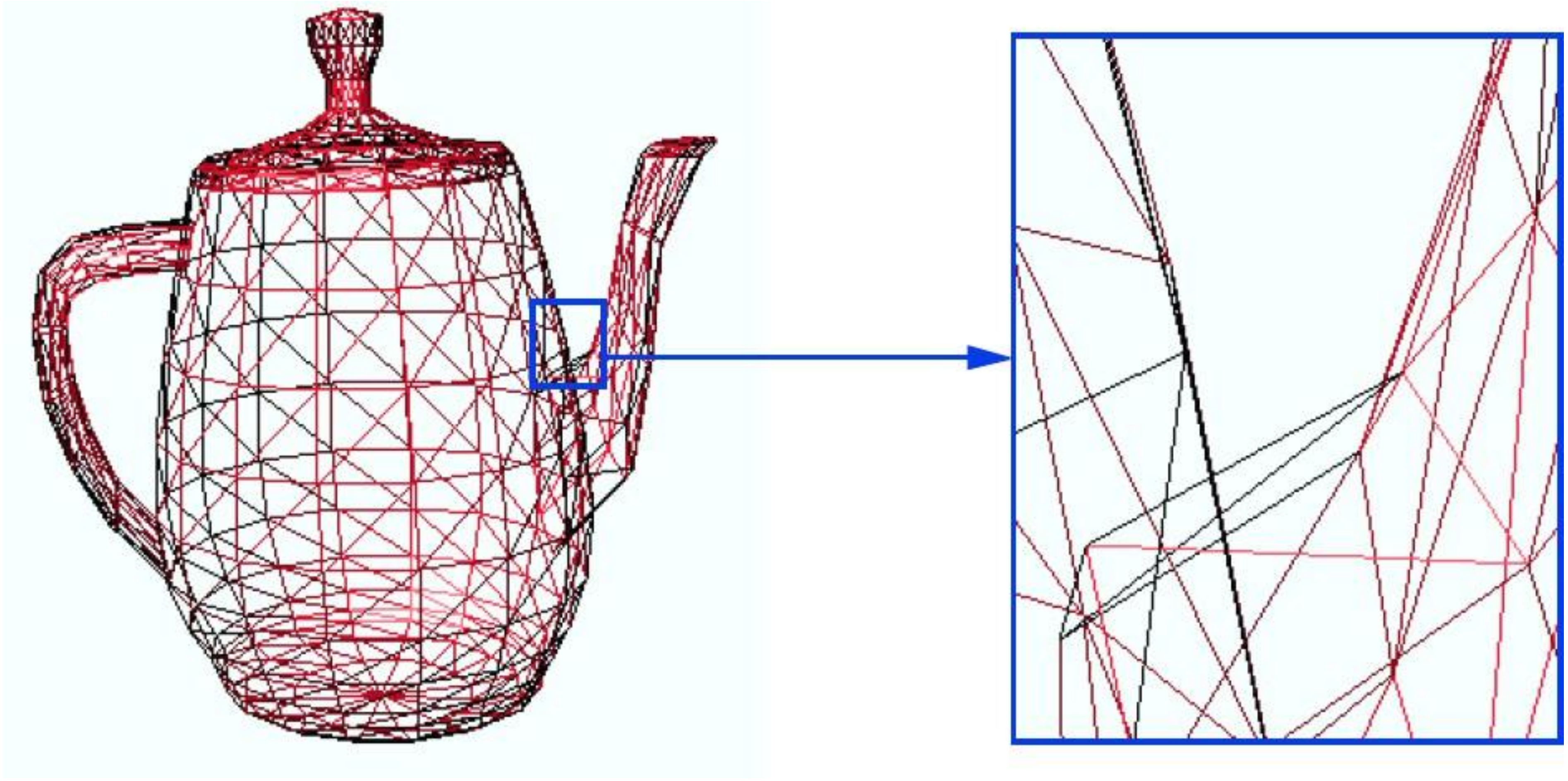
Display Primitives

Outline

- Scan Conversion
- Line Drawing Algorithms
  - Direct Line Drawing Algorithm
  - Digital Differential Analyzer(DDA) Algorithm
  - Bresenham’s Line Algorithm
  - Midpoint line drawing algorithm

Introduction

- Graphic SW and HW provide subroutines to describe a scene in terms of basic geometric structures called output primitives.
- Output primitives are combined to form complex structures.
- Simplest primitives
    - Point (pixel)
    - Line segment



The lines of this object appear ***continuous***

However they are made up of pixels

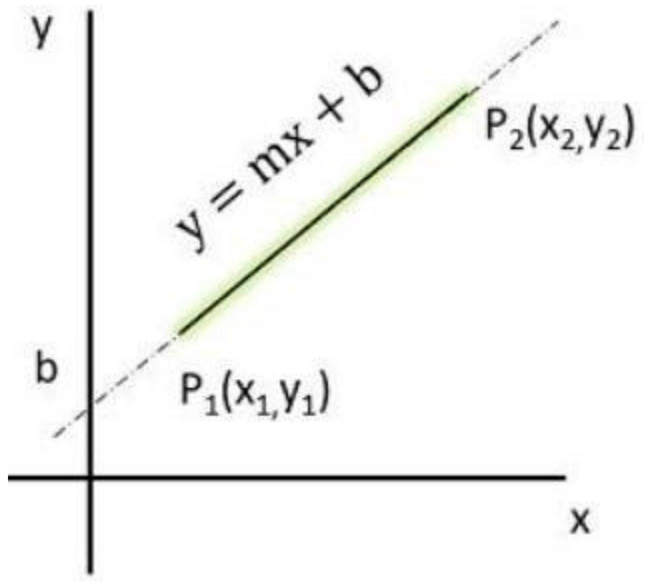
What is Scan-Conversion?

- 2D or 3D objects in real world space are made up of graphic primitives such as points, lines, circles and filled polygons.
- These picture components are often defined in a contiguous space at a higher level of abstraction than individual pixels in the discrete image space.
- For instance, a line is defined by its two endpoints and the line equation while a circle is defined by its radius, centre position, and the circle equation.
- It is the responsibility of the graphics system or the application program to convert each primitive from its geometric definition into a set of pixels that makes up the primitive in the image space.
- This conversion task is generally referred to as scan-conversion or rasterization.

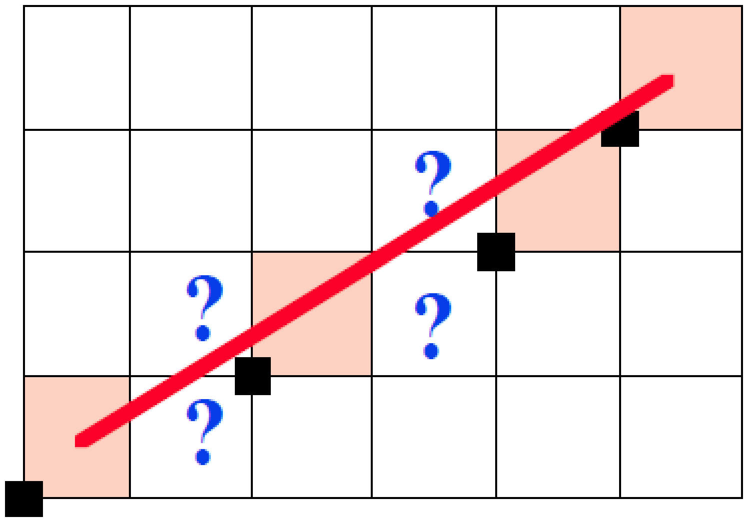
Scan Converting Lines

- A line segment is completely defined in terms of its two endpoints.
- A line segment is thus defined as:

    Line_Seg = { (x1, y1), (x2, y2) }



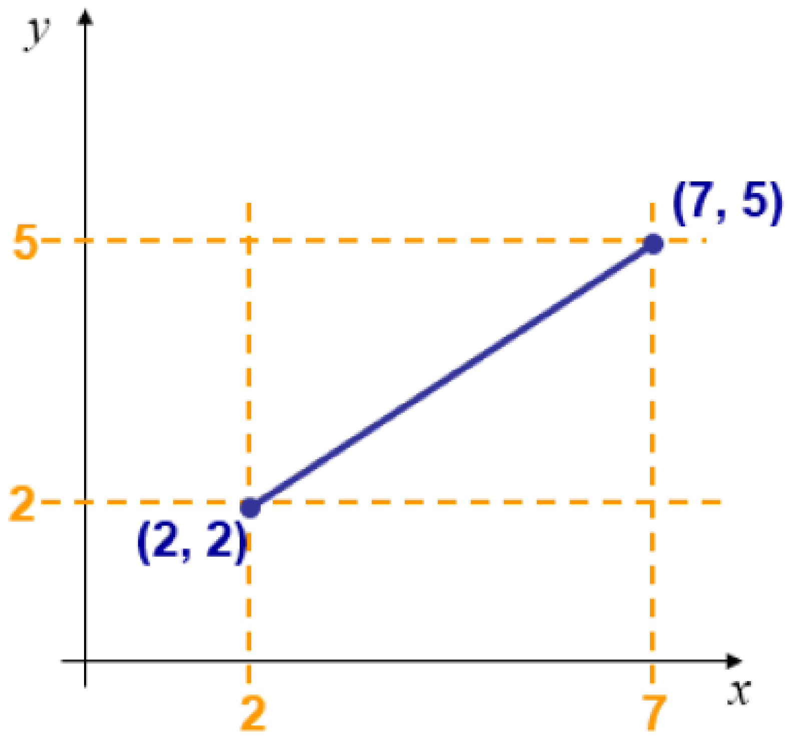
- But what happens when we try to draw line on a pixel-based display?
- How do we choose which pixels to turn on?



- The line has to look good
  - Avoid jaggies
  - The drawing has to be very fast

1. Direct Line Drawing Algorithm

- We could simply work out the corresponding *y* coordinate for each unit *x* coordinate.
- Let's consider the following example:



Direct Line Drawing Algorithm Cont..

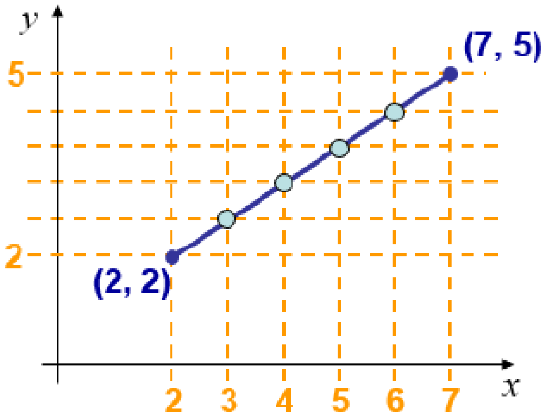
- Start at the pixel for the left-hand endpoint x1.
- Step along the pixels horizontally until we reach the right-hand end of the line, xr.
- For each pixel compute the corresponding y value.
- round this value to the nearest integer to select the nearest pixel.

```
x = xl;
while (x <= xr){
        ytrue = m*x + b;
        y = Round (ytrue);
        PlotPixel (x, y);
        /* Set the pixel at (x,y) on */
        x = x + 1;
}
```

Exercise1 :

Find out the coordinates of the pixels to draw a line having starting coordinate (2,2) and Ending coordinate(7,5).

- Example :



The Equation of the straight line is given by ***y = mx + b***

First Let's find out *m* and *b*

$m = \frac{5-2}{7-2} = \frac{3}{5}$

$b = 2 - \frac{3}{5} * 2 = \frac{4}{5}$

Now for each x value, Let's find corresponding y value.

$y(3) = \frac{3}{5} *3 + \frac{4}{5} = 2\,\frac{3}{5}$

$y(4) = \frac{3}{5} *4 + \frac{4}{5} = 3\,\frac{1}{5}$

$y(5) = \frac{3}{5} *5 + \frac{4}{5} = 3\,\frac{4}{5}$

$y(6) = \frac{3}{5} *6 + \frac{4}{5} = 4\,\frac{2}{5}$

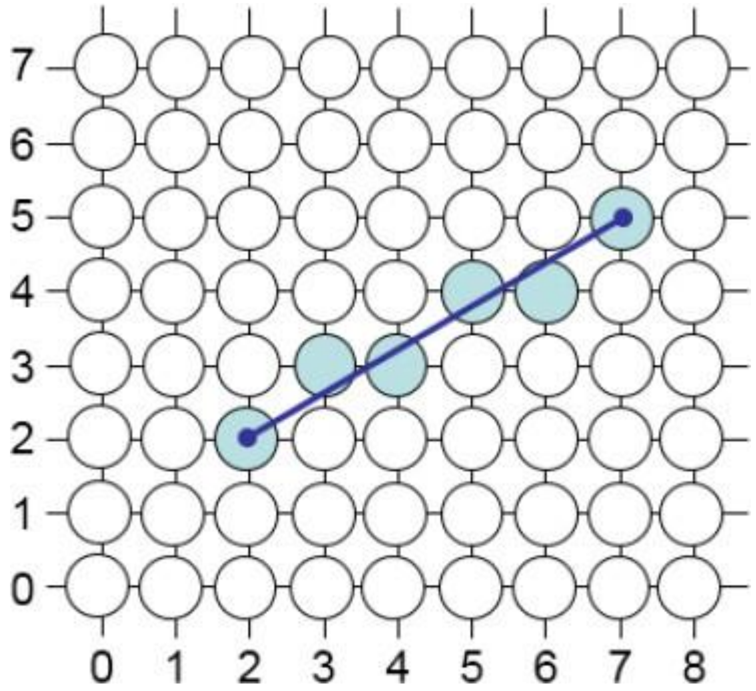
Now Just Round off the y values and turn on the pixels to draw line

$y(3) = \text{Round}\ (2\,\frac{3}{5}) \rightarrow 3$

$Y(4) = \text{Round}\ (3\,\frac{1}{5}) \rightarrow 3$

$Y(5) = \text{Round}\ (3\,\frac{4}{5}) \rightarrow 4$

$Y(6) = \text{Round}\ (4\,\frac{2}{5}) \rightarrow 4$



Limitations of the Direct Line Equation Method

- However, this approach is just way too slow as mentioned earlier
- In particular look out for:
  - The equation $y = mx + b$ requires the multiplication of m by x
  - Rounding off the resulting y coordinates
- We need a faster solution

2. Digital Differential Analyzer(DDA) Algorithm

- The *digital differential analyzer*(DDA) algorithm takes an incremental approach in order to speed up scan conversion.
- Simply calculate y_k*+1* based on y_k
- Consider the list of points that we determined for the line in our previous example:

  (2, 2), (3, 13/5), (4, 16/5), (5, 19/5), (6, 22/5), (7, 5)

- Notice that as the *x* coordinates go up by one, the *y* coordinates simply go up by the slope of the line.
- This is the key insight in the DDA algorithm.

Digital Differential Analyzer(DDA) algorithm Cont..

- We need only compute *m* once, as the start of the scan-conversion.
- For lines with $-1 < m < 1$:
  - incrementing the *x* coordinate by 1, calculate the corresponding *y* coordinates as follows:

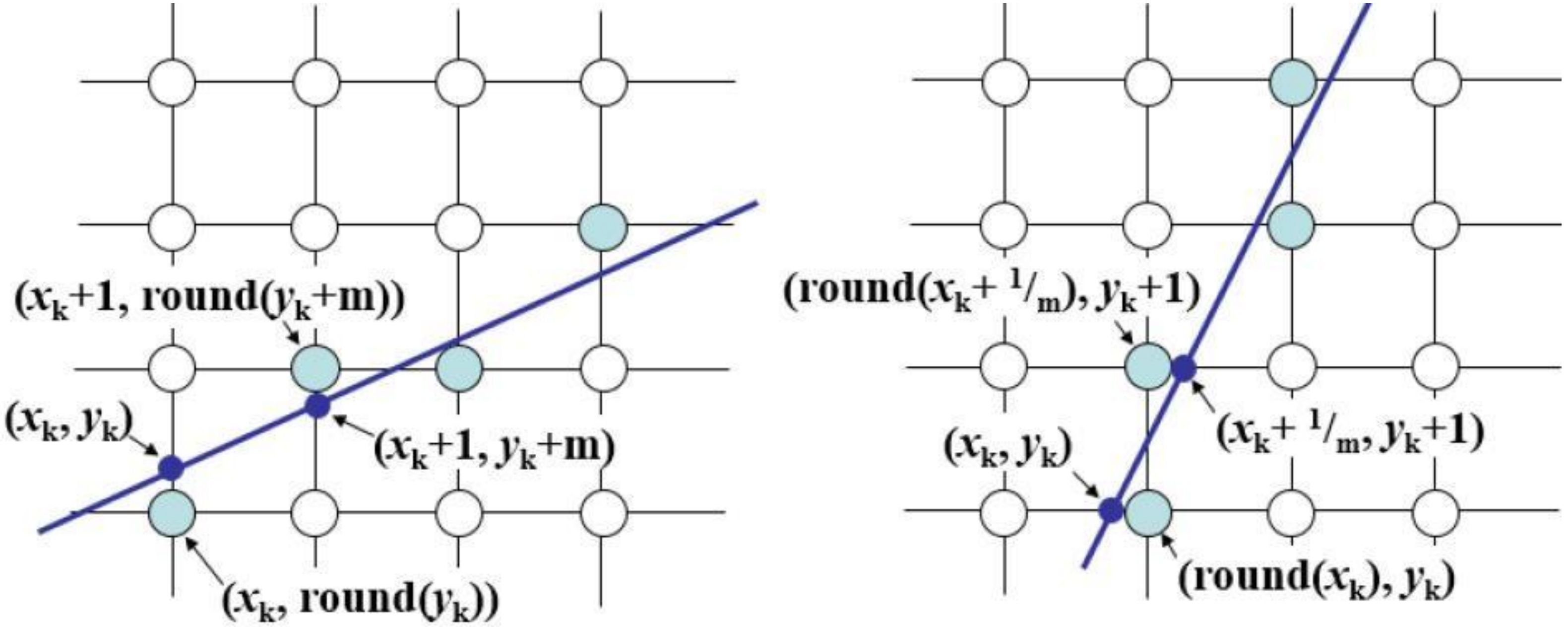
    $y_k+1 = y_k + m$

```
x = xl;
ytrue = yl;
while (x <= xr){
        ytrue = ytrue + m;
        y = Round (ytrue);
        PlotPixel (x, y);
        x = x + 1;
}
```

Digital Differential Analyzer(DDA) algorithm Cont..

- When *m>1 :*
  - increment the *y* coordinate by 1 and calculate the corresponding *x* coordinates as follows (Reverse the roles of *x* and *y* using a unit step in *y*,and *1/m* for *x*.)

$$x_k{+}1 \;= x_k + \frac{1}{m}$$



Exercise 2 :

Starting and end points of a line is given as (1,3) and (7,6). Find out the coordinates of the pixels in the line.

Starting Points – (1,3)

End Points – (7,6)

- Example :

Starting Points – (1,3) ,

End Points – (7,6)

Let's find the slope – m = $\frac{6-3}{7-1}$ = 0.5

m < 1 ;

Thus we can use y_k+1 = y_k + m to calculate y for each increment in x value

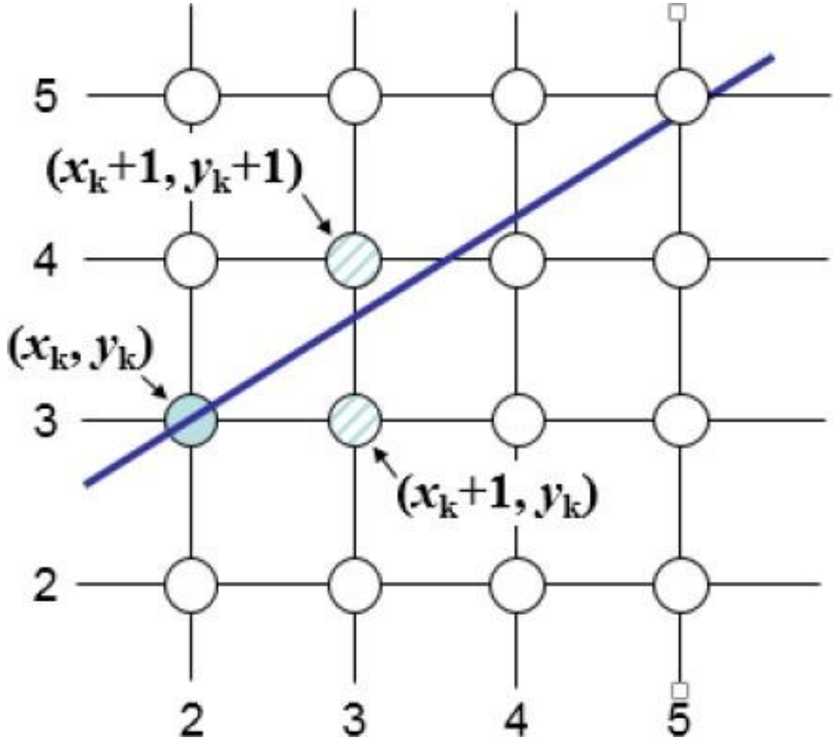
x	**y**	**Round(y)**	**(x,Round(y))**
1	3	3	1,3
2	3+0.5 = 3.5	4	2,4
3	3.5+0.5 = 4	4	3,4
4	4+0.5=4.5	5	4,5
5	4.5+0.5=5.5	5	5,5
6	5.5+0.5 = 6	6	6,6
7	6	6	6,6

Limitations of DDA

- The DDA algorithm is much faster than our previous attempt
  - In particular, there are no longer any multiplications involved.
- However, there are still two big issues:
  - Accumulation of round-off errors can make the pixelated line drift away from what was intended.
  - The rounding operations and floating point arithmetic involved are time consuming.

3. Bresenham's Line Algorithm

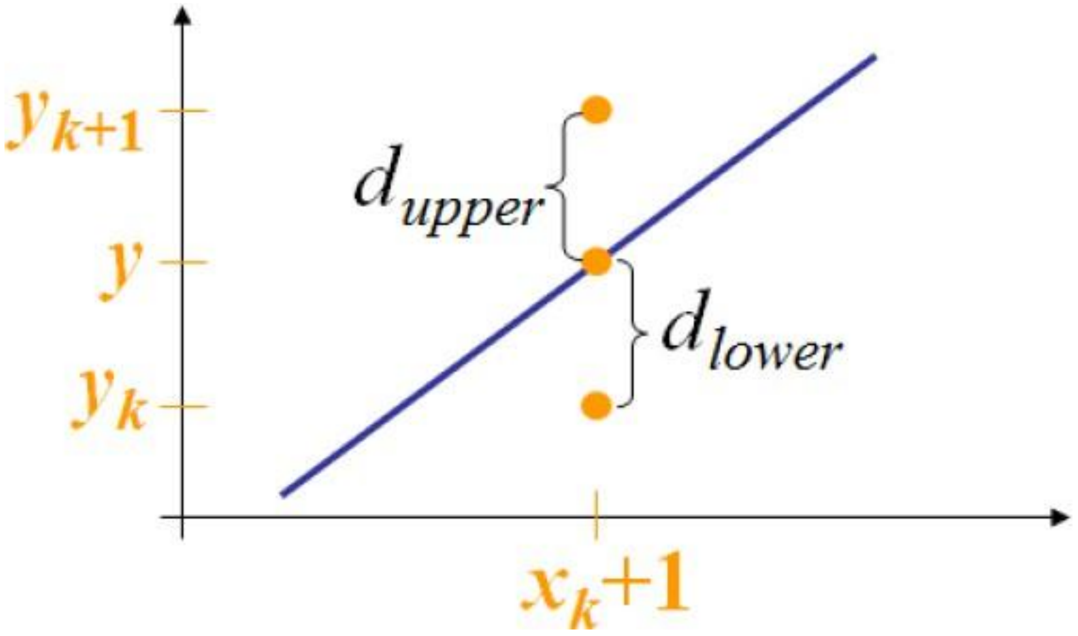
- This algorithm uses only integer arithmetic, and runs significantly faster.
- Move across the x axis in unit intervals and at each step choose between two different y coordinates
- For example, from position (2, 3) we have to choose between (3, 3) and (3, 4)
- We would like the point that is closer to the original line



Bresenham’s Line Algorithm Cont..

- At sample position x_k+1 the vertical separations from the mathematical line are labelled d_{upper} and d_{lower}
- The *y* coordinate on the mathematical line at x_k+1 is:

$$y = m(x_k+1) + b$$



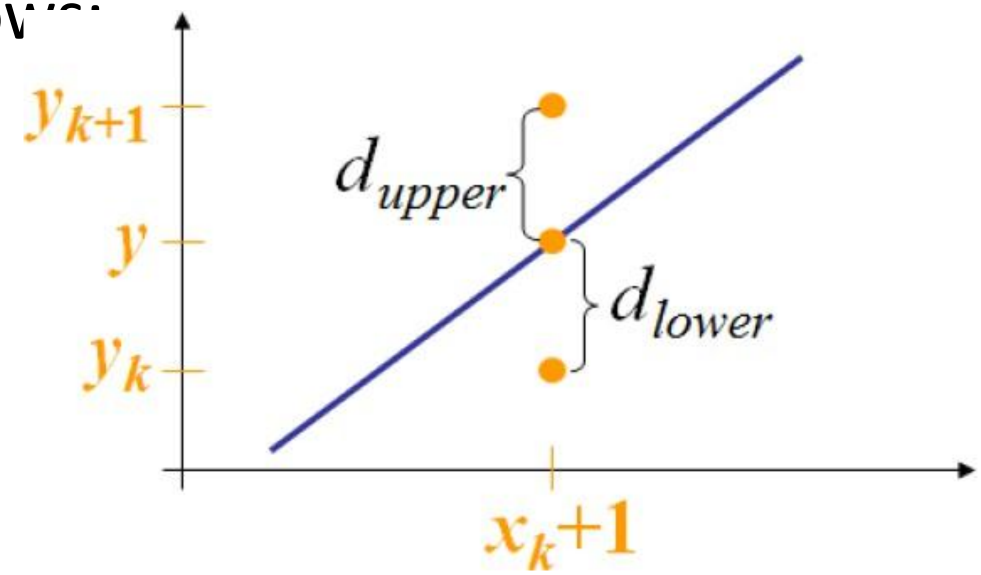
Bresenham's Line Algorithm Cont..

- So, d_{upper} and d_{lower} given as follows:

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

And

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



- We can use these to make a simple decision about which pixel is closer to the mathematical line.

Bresenham's Line Algorithm Cont..

- This simple decision is based on the difference between the two pixel positions:

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

- A decision parameter p_k will be derived from the above equation.
- The first decision parameter p0 is evaluated at (x0, y0) is given as:

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

- Depending on the sign of decision variable, we choose the lower pixel, otherwise we choose the upper pixel.

Bresenham's Line Algorithm(for $|m| < 1.0$)

1. Input the two line end-points, storing the left end-point in (*x0, y0*)
2. Plot the point (*x0, y0*)
3. Calculate the constants Δ*x*, Δ*y*, 2Δ*y*, and (2Δ*y* -2Δ*x*) and get the first value for the decision parameter as:

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

Bresenham's Line Algorithm(for $|m| < 1.0$)

4. At each x_k along the line, starting at *k = 0*, perform the following test.

    If *pk*< 0,

        the next point to plot is (xk+1,yk)

        and p_{k+1} = p_k + 2 Δ*y*

    Otherwise,

        the next point to plot is (*xk+1, yk+1*)

        and: p_{k+1} = p_k + 2 Δ*y -2* Δ*x*

5. Repeat step 4 (Δ*x* –1) times

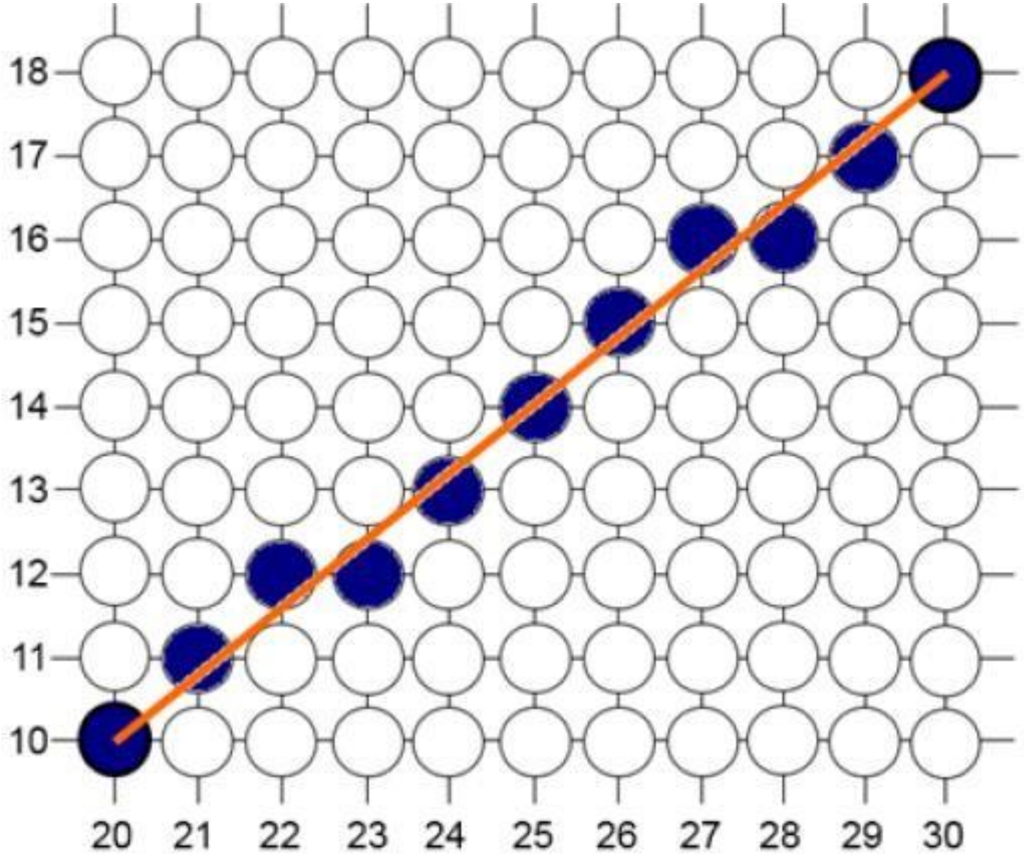
N.B.: The algorithm and derivation above assumes slopes are less than 1. For other slopes we need to adjust the algorithm slightly.

Example :

- Let's plot the line from (20, 10) to (30, 18)
- First off calculate all of the constants:

  Δx: 10 Δy: 8 $2\Delta y$: 16 $2\Delta y - 2\Delta x$: -4
- Calculate the initial decision parameter $p0$:

  $p0 = 2\Delta y – \Delta x = 6$
- Go through the steps of the Bresenham line drawing algorithm for a line going from (21,12) to (29,16).



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

Case-01
If Pk < 0

$P_{k+1} = P_k + 2\Delta Y$

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

$Y_{k+1} = Y_k$

Case-02
If Pk >= 0

Pk+1 = Pk + 2ΔY - 2ΔX

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

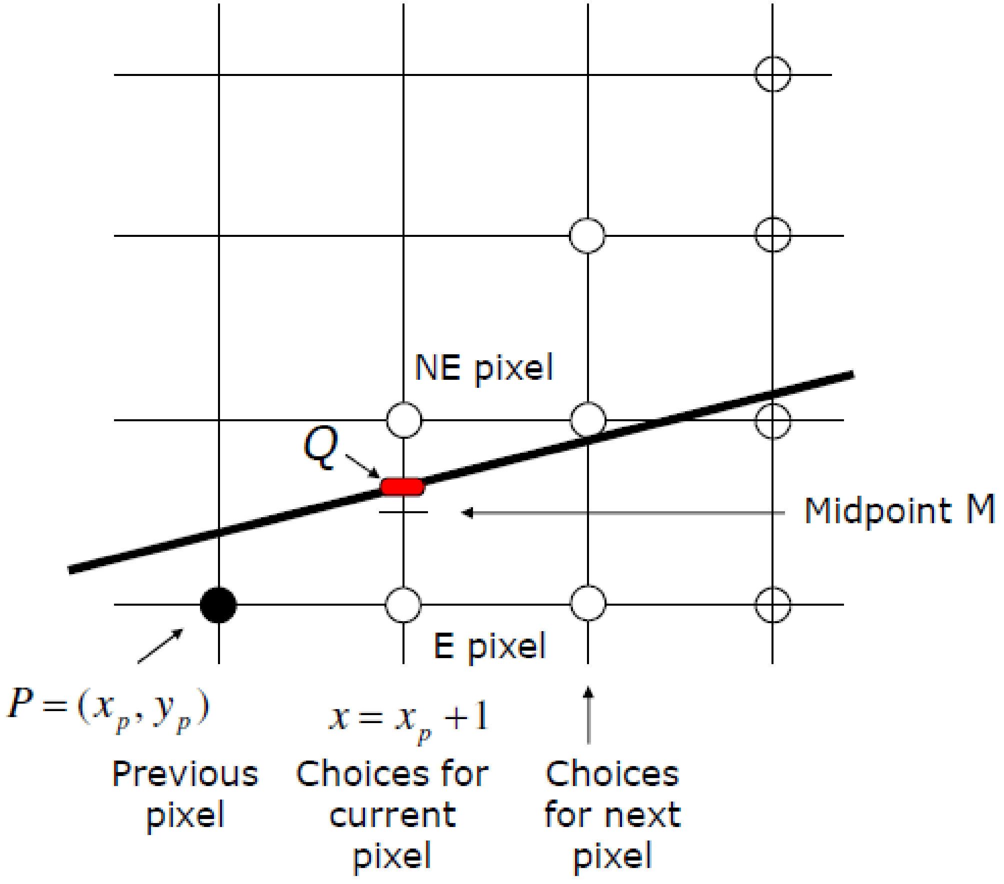
$Y_{k+1} = Y_k + 1$

Bresenham Line Algorithm Summary

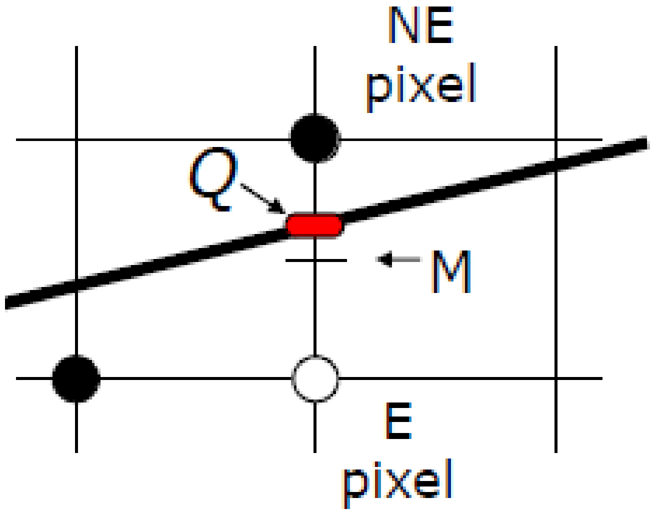
- The Bresenham's line algorithm has the following advantages:
  - A fast incremental algorithm.
  - Uses only integer calculations.
- Comparing this to the DDA algorithm, DDA has the following problems:
  - Accumulation of round-off errors can make the pixelated line drift away from what was intended.
  - The rounding operations and floating point arithmetic involved are time consuming.

4. Midpoint line drawing algorithm

- Assume that line's slope is shallow and positive (0 < slope < 1); other slopes can be handled by suitable reflections about principle axes.
- Call lower left endpoint (x0, y0) and upper right endpoint (x1, y1).
- Assume that we have just selected pixel P at (xp, yp).
- Next, we must choose between pixel to right (E pixel), or one right and one up (NE pixel).
- Let Q be intersection point of line being scan-converted and vertical line x=xp+1.



- Line passes between E and NE.
- Point that is closer to intersection point Q must be chosen.
- Observe on which side of line midpoint M lies:
  - E is closer to line if midpoint M lies above line, i.e., line crosses bottom half
  - NE is closer to line if midpoint M lies below line, i.e., line crosses top half
- Error (vertical distance between chosen pixel and actual line) is always <= ½.
- Algorithm chooses NE as next pixel for line shown.
- Now, need to find a way to calculate on which side of line midpoint lies.



Line equation as function f(x):

$$y = mx + B$$

$$y = \frac{dy}{dx} x + B$$

Line equation as implicit function:

$$f(x, y) = ax + by + c = 0$$

for coefficients a, b, c, where a, b != 0 from above,

$$f(x, y) = dy(x) + (-dx)y + Bdx = 0$$

$$\boldsymbol{a = dy\ ,\ b = -\ dx\ ,\ c = B\ dx}$$

Properties (proof by case analysis):

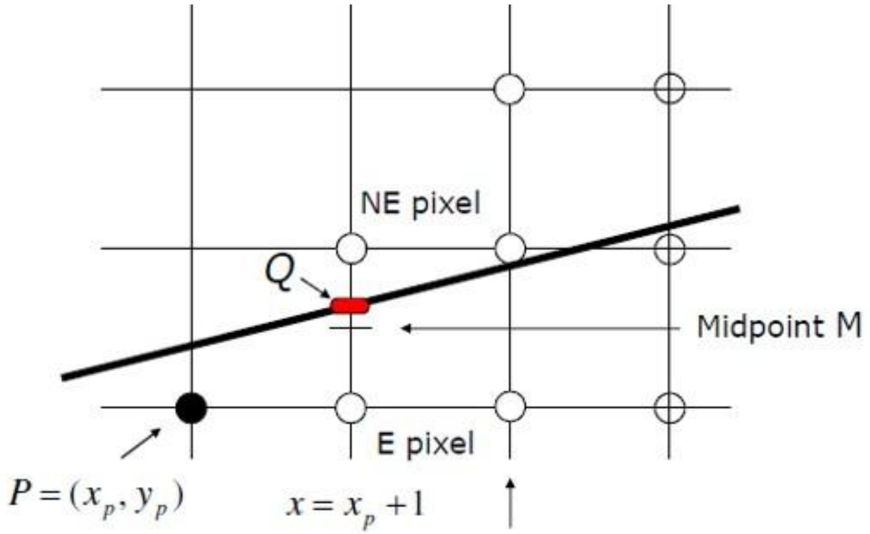
- f(xm, ym) = 0 when any point M is on line
- f(xm, ym) < 0 when any point M is above line
- f(xm, ym) > 0 when any point M is below line
- Our decision will be based on value of function at midpoint M at (xp + 1, yp + ½)

Decision Variable d:

- We only need sign of f(xp + 1, yp + ½) to see where line lies, and then pick nearest pixel

d = f(xp + 1, yp + ½).

- if d > 0 choose pixel NE
- if d < 0 choose pixel E
- if d = 0 choose either one c



How do we incrementally update d?

- On basis of picking E or NE, figure out location of M for that pixel, and corresponding value d for next grid line.
- We can derive d for the next pixel based on our current decision.

If E was chosen:

Increment M by one in x direction

d_{new} = f(xp + 2, yp + ½)

= a(xp + 2) + b(yp + ½) + c

d_{old} = a(xp + 1) + b(yp + ½) + c

d_{new} - d_{old} is the incremental difference ΔE

d_{new} = d_{old} + a

ΔE = a = dy

- We can compute value of decision variable at next step incrementally without computing F(M) directly

d_{new} = d_{old} + a = d_{old} + dy

If NE was chosen:

Increment M by both one in both x and y directions

d_{new} = F(xp + 2, yp + 3/2)

= a(xp + 2) + b(yp + 3/2) + c

ΔNE = $d_{new} - d_{old}$

d_{new} = d_{old} + a + b

ΔNE = a + b = dy – dx

Thus, incrementally,

d_{new} = d_{old} + ΔNE

d_{new} = d_{old} + dy – dx

- First midpoint for first d = d_{start} is at (x0 + 1, y0 + ½)

    f(x0 + 1, y0 + ½)

    = a(x0 + 1) + b(y0 + ½) + c

    = a * x0 + b * y0 + c + a + b/2

    = f(x0, y0) + a + b/2
- But (x0, y0) is point on line and f(x0, y0) = 0
- Therefore, d_{start} = a + b/2 = dy – dx/2
    – use d_{start} to choose second pixel, etc
- To eliminate fraction in d_{start} :
    – redefine f by multiplying it by 2; f(x,y) = 2(ax + by + c)

– this multiplies each constant and decision variable by 2, but does not change sign.

Mid point Line Algorithm for (for $|m| < 1.0$)

Step 1: Input two line endpoints (x0,y0) and (x1,y1) and calculate dy and dx

Step2 : Calculate initial decision parameters

d_{start} = 2dy-dx ; x=x0 ; y=y0 and plot (x,y)

Mid point Line Algorithm for (for $|m| < 1.0$)

```
Step3:    while(x < x1)
                    x = x + 1
                    if(d < 0)    --------- E is choosen
                              d = d+2dy
                    else          ---------- NE is choosen
                    {
                              d = d  + 2(dy-dx)
                              Y = Y+1
                    }
                         plot (x,y)
```

Step4 : Repeat 3rd step dx-1 number of times

Example

Draw line A(4,8) and B(10,12) using midpoint line drawing algorithm.

Example:

- Here Starting Point is (4,8) , End point (10,12)

  => dx = 10-4 = 6 =>dy = 12-8 = 4

  => m= dy/dx = 4/6 = 0.67 (m<1)

  Now let's find the initial decision parameter

  => d0 = 2dy –dx = 8-6 = 2

  d0 > 0 ; Therefore NE is choosen

  and next pixel to be plotted will be → (x +1,y+1) → (5,9)

$\Rightarrow$d1 = d0 + 2(dy-dx)

= 2 + 2(-2)

= -2 d1<0 → E is choosen , (x+1,y) → (6,9)

=>d2 = d1+2dy

=-2 +8

= 6 d2>0 → NE is choosen, (x+1,y+1) → (7,10)

=>d3 = d2+2(dy-dx)

= 6 +2(-2)

= 2 d3>0 → NE is choosen, (x+1,y+1) → (8,11)

=>d4 = d3+ 2(dy-dx)

=2 + 2(-2)

= -2 d4<0 → E is choosen, (x+1,y) → (9,11)

=>d5 = d4+ 2dy

= -2 + 8

= 6 d5 > 0 → (x+1,y+1) → (10,12)

K	d_k	Plotting Points
0	2	(5,9)
1	-2	(6,9)
2	6	(7,10)
3	2	(8,11)
4	-2	(9,11)
5	6	(10,12)

Midpoint line drawing algorithm Summary

- It choose the pixels closest to the line with accuracy, consistence and straightness.
- Requires only integer data and simple arithmetic calculations and no round off functions.
- Avoid complex division and multiplication and thus avoid truncate errors.
- Bresenham's line algorithm is same as mid point line alogorithm but doesn't generalize as nicely to circles and ellipses.