CSE-303: COMPUTER GRAPHICS

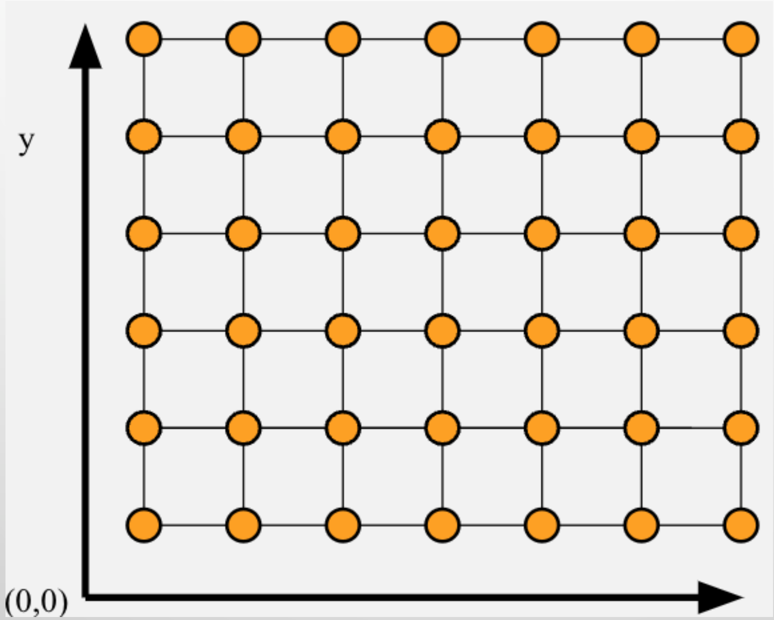
PROFESSOR DR. SANJIT KUMAR SAHA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

JAHANGIRNAGAR UNIVERSITY, SAVAR, DHAKA

DISPLAYS - PIXELS

- Pixel: the smallest element of picture.
  - integer position (i, j)
  - color information (r, g, b)



SCAN CONVERSION

- **Scan conversion** is defined as the process of representing continuous ***graphic object*** as a collection of discrete pixels.
  - Various graphic objects are
    - Point
    - Line
    - Rectangle, Square
    - Circle, Ellipse
    - Sector, Arc
    - Polygons

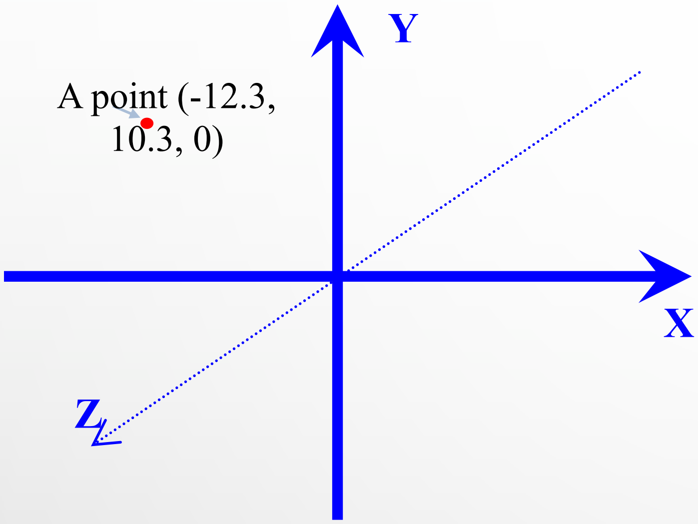
SCAN CONVERSION…

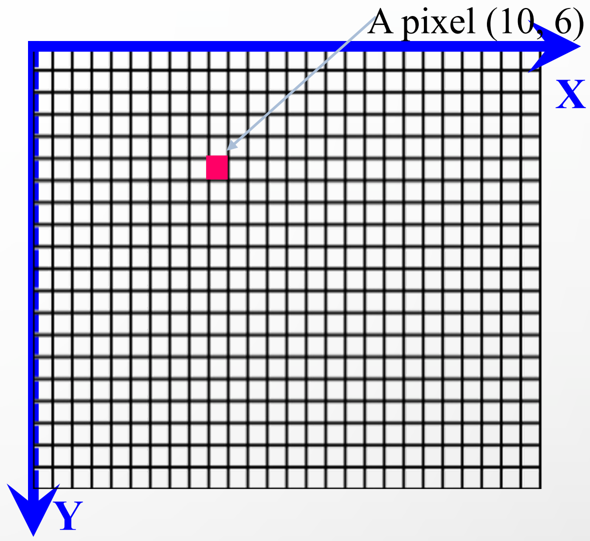
- **Scan conversion** is required to convert vector data to raster format for a scan line display device
  - Convert each graphic object to a set of regular pixels.
  - Determine inside/outside areas when *filling polygons*.
  - Scan-convert curves

SCAN CONVERSION ALGORITHMS

1. **Scan conversion of point**
2. Scan conversion of line
3. Scan conversion of circle
4. Scan conversion of ellipse
5. Scan conversion of polygons

SCAN CONVERTING A POINT





MODELLING CO-ORDINATES

- Mathematically vectors are defined in an infinite, “real-number” Cartesian co-ordinate system

SCREEN COORDINATES

- Also known as device co-ordinates, pixel co-ordinates
- On display hardware we deal with finite, discrete coordinates
- X, Y values in positive integers
- 0,0 is measured from top-left usually with +Y pointing down

SCAN CONVERTING A POINT

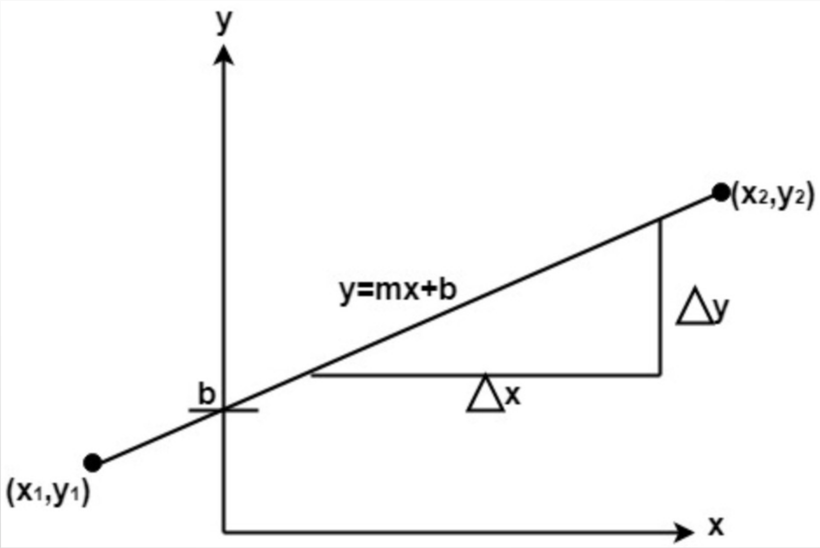
- Each pixel on graphic display does not represent a mathematical point like P(2.6,3.33). But it can be accommodated to the nearest position by applying few mathematical functions such as
  - Ceil p ≈ (3,4)
  - Floor p ≈ (2,3)
  - Greatest integer function p ≈ (3,3)
  - Round p ≈ (3,3)

SCAN CONVERSION

1. Scan conversion of point
2. **Scan conversion of line**
3. Scan conversion of circle
4. Scan conversion of ellipse

SCAN CONVERSION OF LINE

- A straight line may be defined by two endpoints and an equation. In fig the two endpoints are described by (x_1,y_1) and (x_2,y_2). The equation of the line is used to determine the x, y coordinates of all the points that lie between these two endpoints.



PROPERTIES OF GOOD LINE DRAWING ALGORITHM

- **Line should appear straight:** We must appropriate the line by choosing addressable points close to it. If we choose well, theline will appear straight, if not, we shall produce crossed lines.

Fig: O/P from a poor line generating algorithm

- **Lines should terminate accurately:** Unless lines are plotted accurately, they may terminate at the wrong place.

Fig: Uneven line density caused by bunching of dots.

- **Lines should have constant density:** Line density is proportional to the no. Of dots displayed divided by the length of the line.

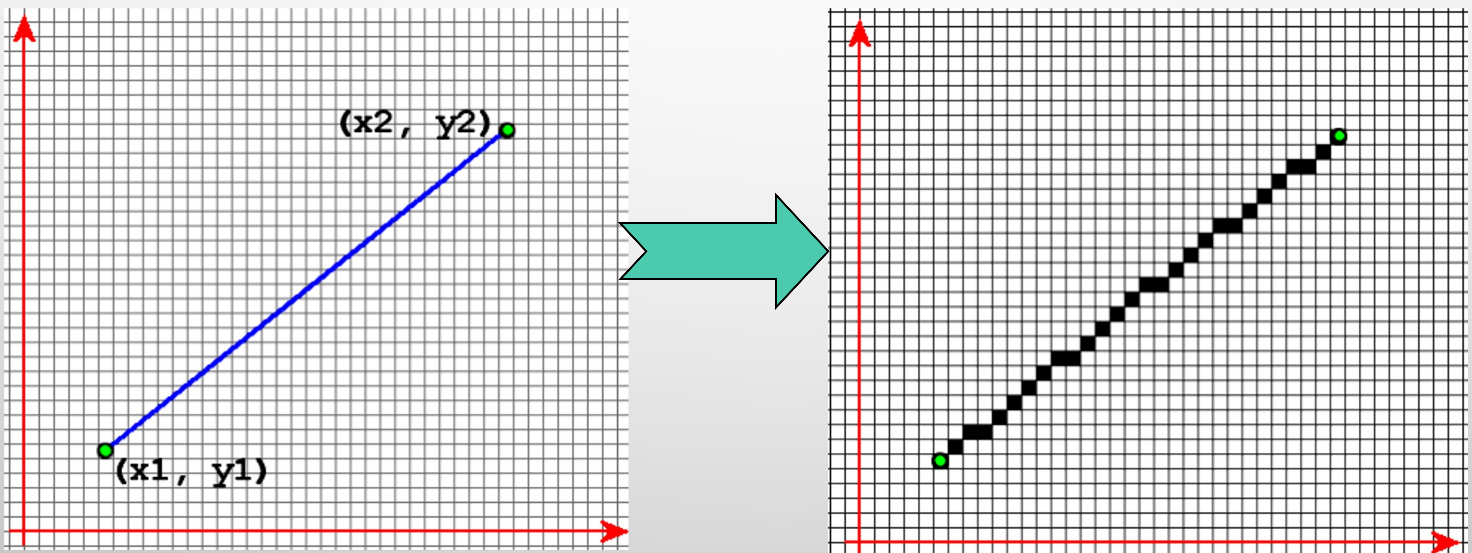
To maintain constant density, dots should be equally spaced.

- **Line density should be independent of line length and angle:** This can be done by computing an approximating line-length estimate and to use a line-generation algorithm that keeps line density constant to within the accuracy of this estimate.
- **Line should be drawn rapidly:** This computation should be performed by special-purpose hardware.

SCAN CONVERTING A LINE

How does a machine draw lines?

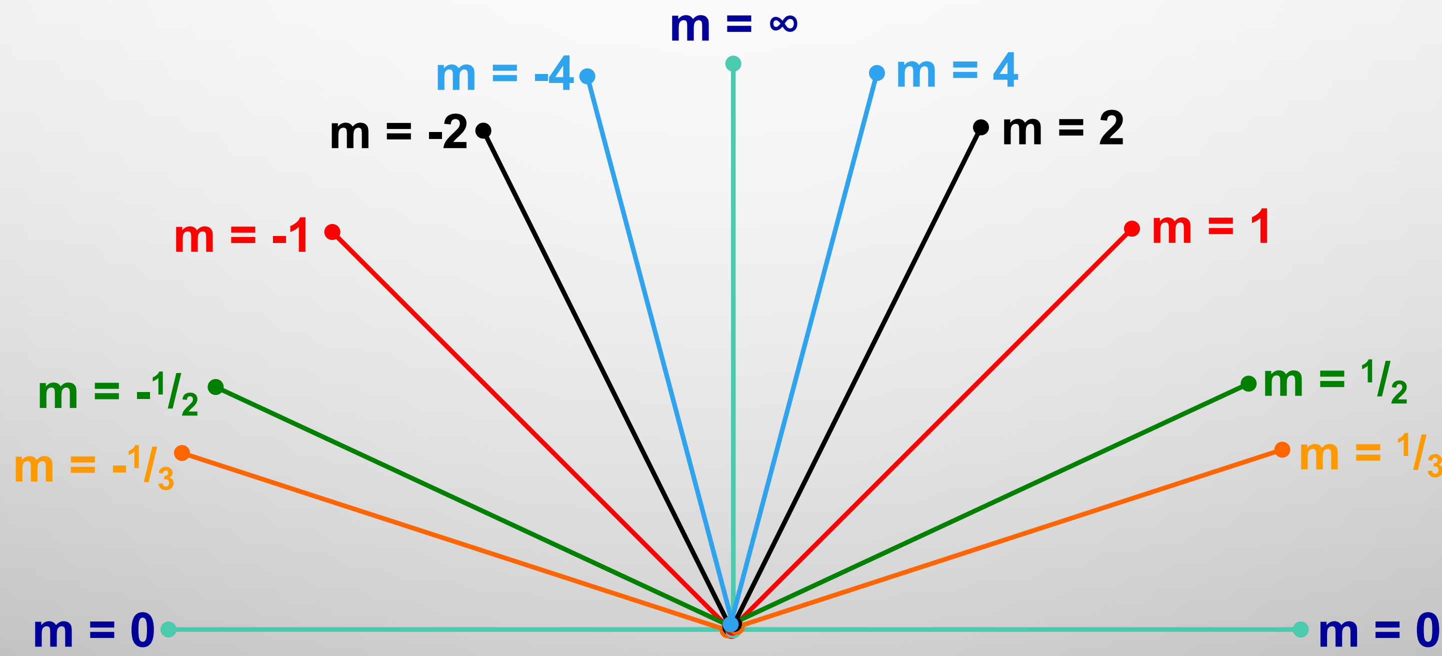
1. Give it a start and end position.
2. Figure out which pixels to colour in between these…
    - How do we do this?
    - Line-drawing algorithms: DDA, Bresenham's algorithm



SCAN CONVERTING A LINE

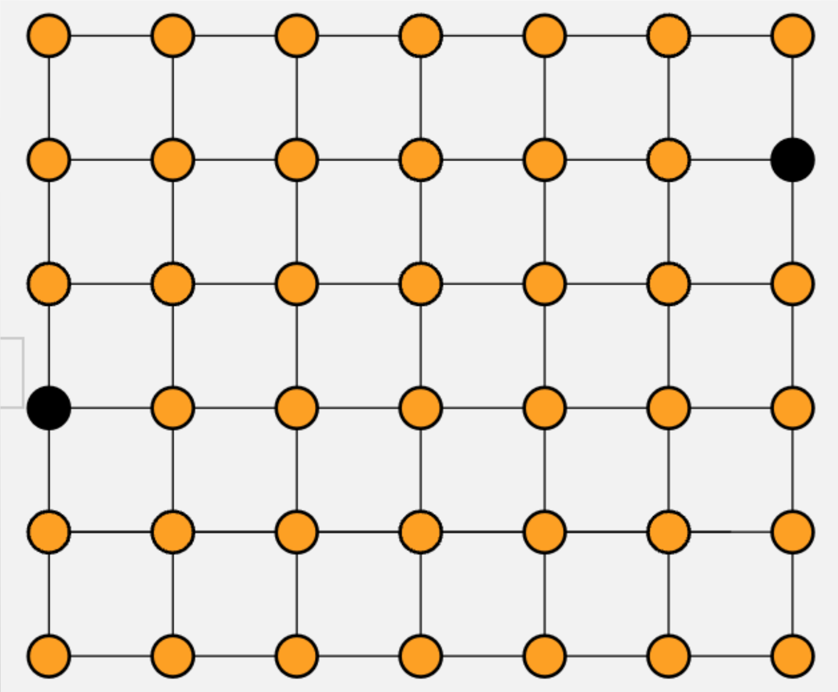
Line and its slope

- The slope of a line (*m*) is defined by its start and end coordinates
- The diagram below shows some examples of lines and their slopes



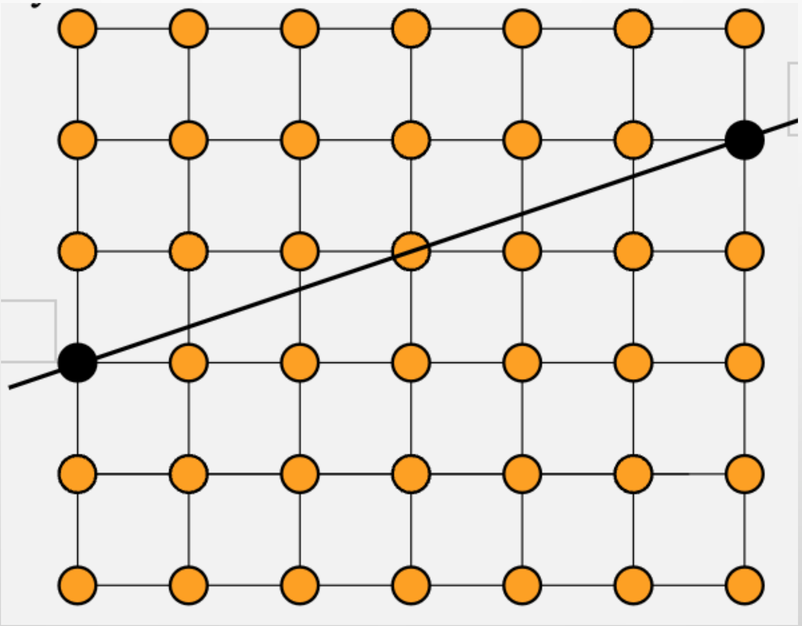
PROBLEM

- Given two points (P,Q) on the screen (with integer coordinates) determine which pixels should be drawn to display a unit width line.



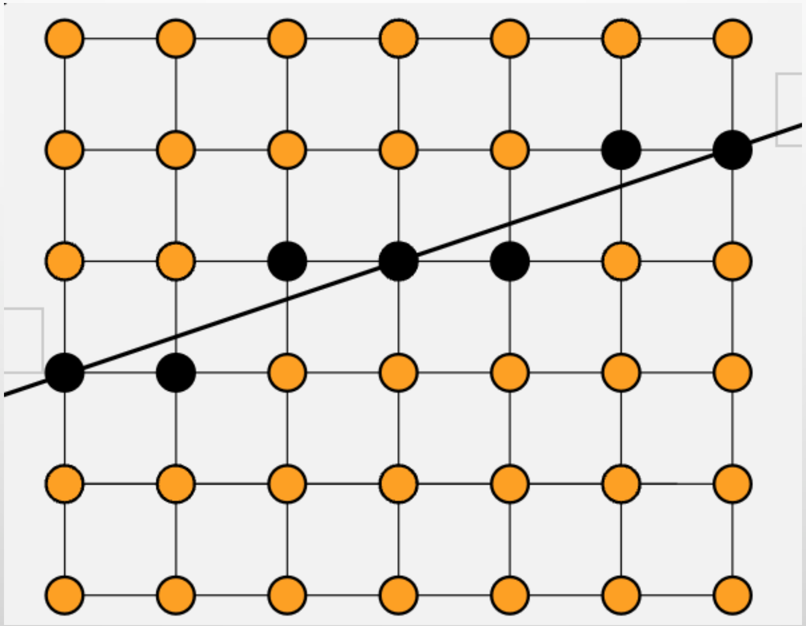
PROBLEM…

- Given two points (P,Q) on the screen (with integer coordinates) determine which pixels should be drawn to display a unit width line.



PROBLEM…

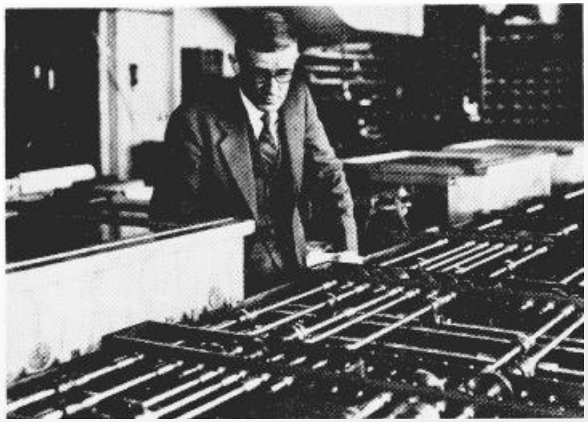
- Given two points (P,Q) on the screen (with integer coordinates) determine which pixels should be drawn to display a unit width line.



LINE DRAWING ALGORITHMS

1. **DDA Line Algorithm**
2. Bresenham's Line Algorithm

DDA ALGORITHM



- Digital differential analyser is an algorithm for scan-converting lines
- The original differential analyzer was a physical machine developed by Vannevar Bush at MIT in the 1930's in order to solve ordinary differential equations.
- It calculates pixel positions along a line by taking unit step increment along one direction and calculating corresponding coordinate position based on the *rate of change* of the coordinate (δx or δy) (***incremental approach)***

DDA ALGORITHM

Basic concept

- For each part of the line the following holds true:

$$m = \frac{\Delta y}{\Delta x} \quad \Rightarrow \quad \Delta y = m\Delta x$$

- If δx = 1 i.e. 1 pixel then … $\Delta y = m$
- i.e. For each pixel we move right (along the x axis), we need to move down (along the y-axis) by m pixels.
- In pixels, the gradient represents how many pixels we step upwards (δy) for every step to the right (δx)

DDA ALGORITHM

Derivation

Assume that $0<m<1$, $\delta x > 0$ and $\delta y >0$

For a point $p(x_i, y_i)$ on a line we know that

$$Y_i = mx_i + b$$

At next position $p(x_{i+1}, y_{i+1})$

$$y_{i+1} = mx_{i+1} + b$$

Having unit step increment along x-axis means $x_{i+1} = x_i + 1$

Therefore $y_{i+1} = m(x_i + 1) + b$

$$= mx_i + m + b$$

$$= mx_i + b + m$$

$$= y_i + m$$

DDA ALGORITHM

Simple algorithm

```
1.  Input (x1,y1) and (x2,y2)
2.  Let x = x1; y = y1;
    M = (y2-y1)/(x2-x1);
3.  Draw pixel (x, y)
4.  WHILE (x < x2) //i.e. We reached the second
    endpoint
5.    {
       X = x + 1; //step right by one pixel
       Y = y + m; //step down by m pixels
       Draw pixel (round(x), round(y));
      }
```

DDA ALGORITHM

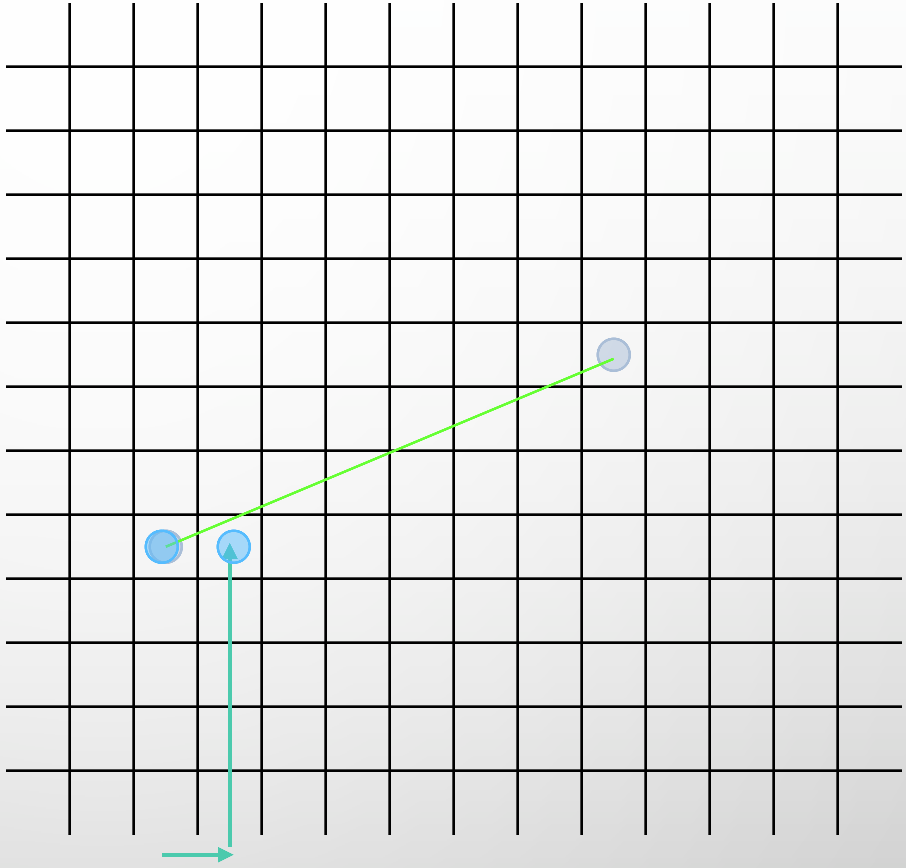
Example

Sample at unit x:

$$x_{k+1} = x_k + \Delta x$$
$$= x_k + 1$$

Corresponding y pos.:

$$y_{k+1} = y_k + \Delta y$$
$$= y_k + m \cdot \Delta x$$
$$= y_k + m \cdot (1)$$



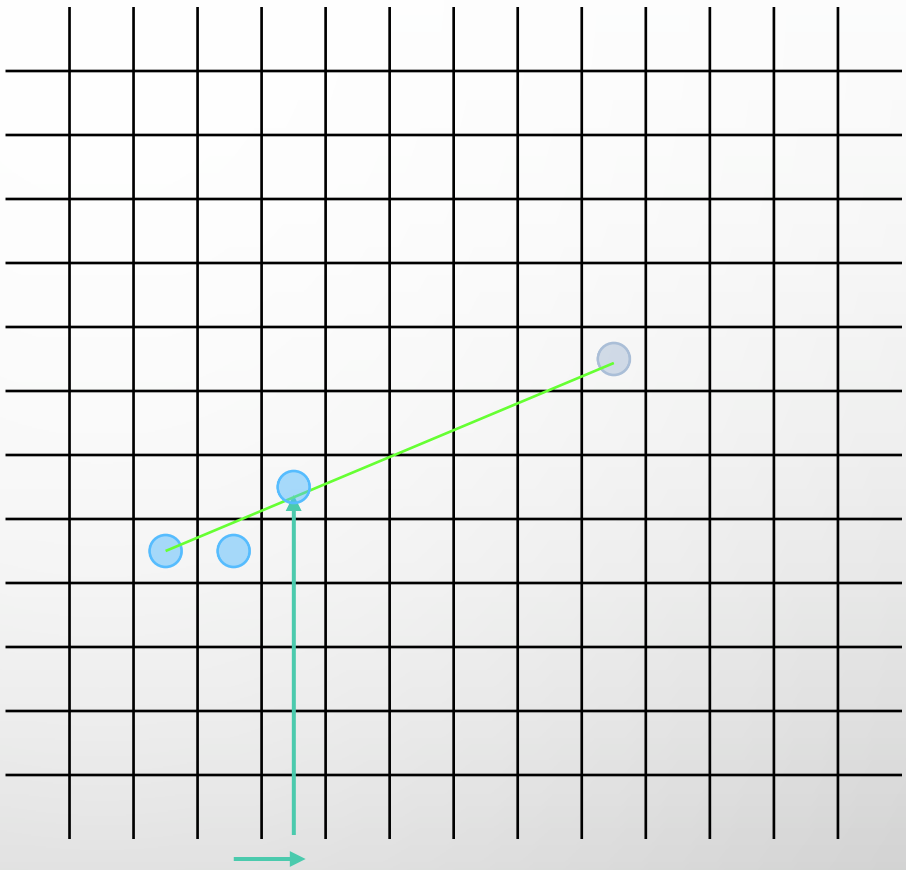
Example

Sample at unit x:

$$x_{k+1} = x_k + \Delta x$$
$$= x_k + 1$$

Corresponding y pos.:

$$y_{k+1} = y_k + \Delta y$$
$$= y_k + m \cdot \Delta x$$
$$= y_k + m \cdot (1)$$



DDA ALGORITHM

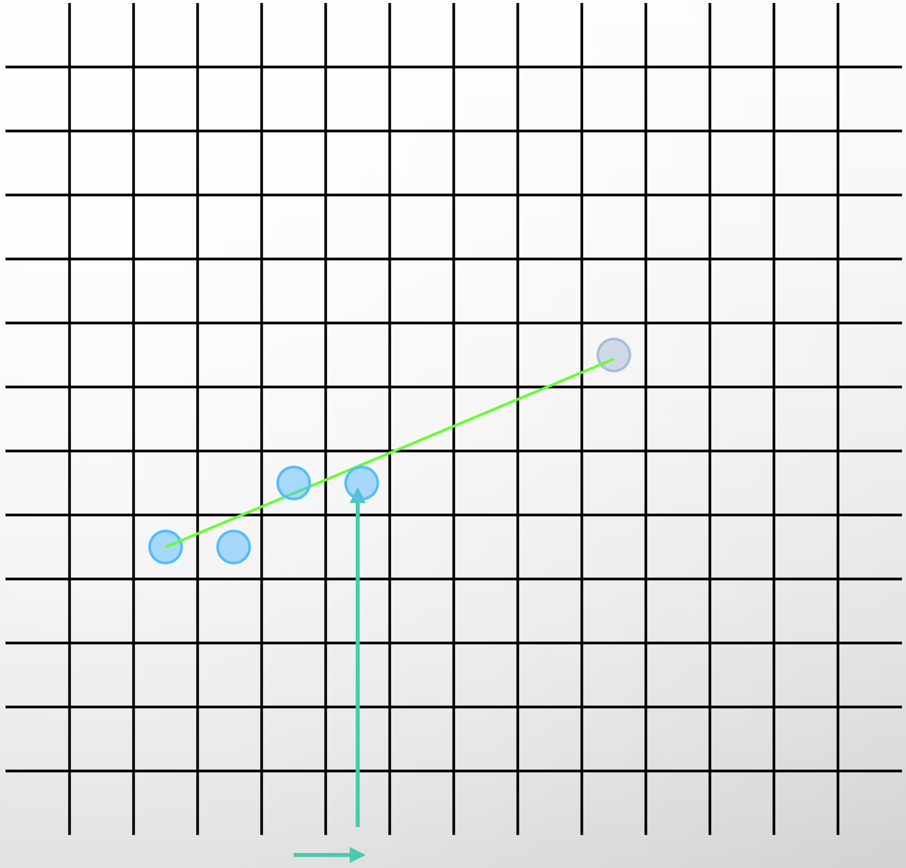
Example

Sample at unit x:

$$x_{k+1} = x_k + \Delta x$$
$$= x_k + 1$$

Corresponding y pos.:

$$y_{k+1} = y_k + \Delta y$$
$$= y_k + m \cdot \Delta x$$
$$= y_k + m \cdot (1)$$



DDA ALGORITHM

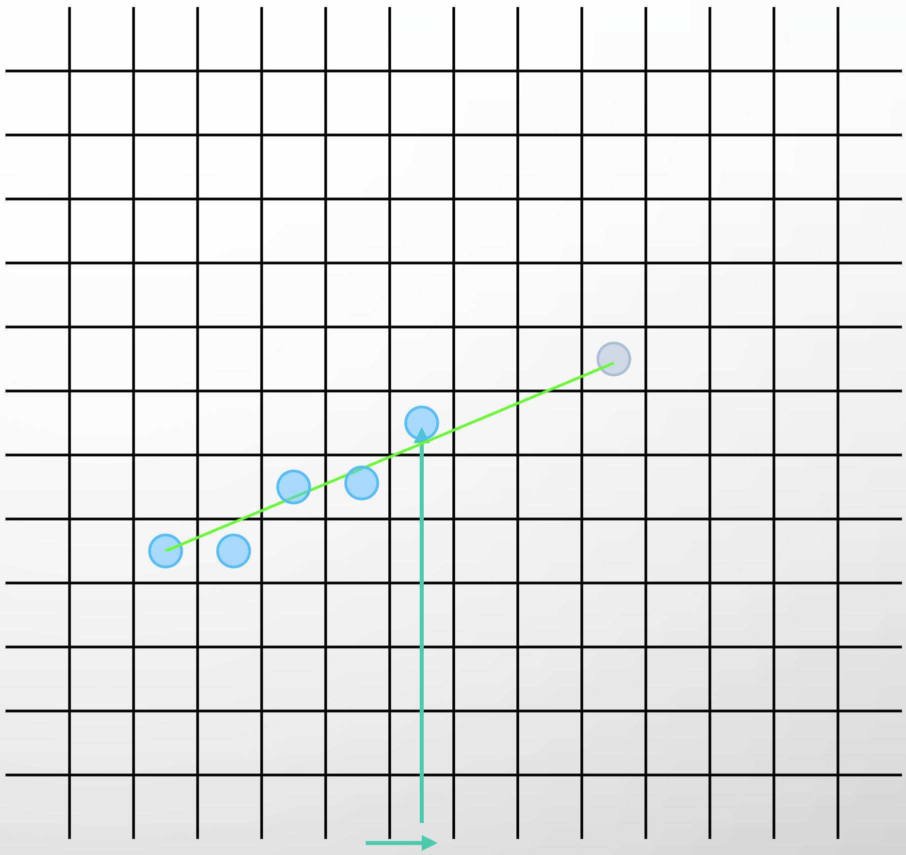
Example

Sample at unit x:

$$x_{k+1} = x_k + \Delta x$$
$$= x_k + 1$$

Corresponding y pos.:

$$y_{k+1} = y_k + \Delta y$$
$$= y_k + m \cdot \Delta x$$
$$= y_k + m \cdot (1)$$



DDA ALGORITHM

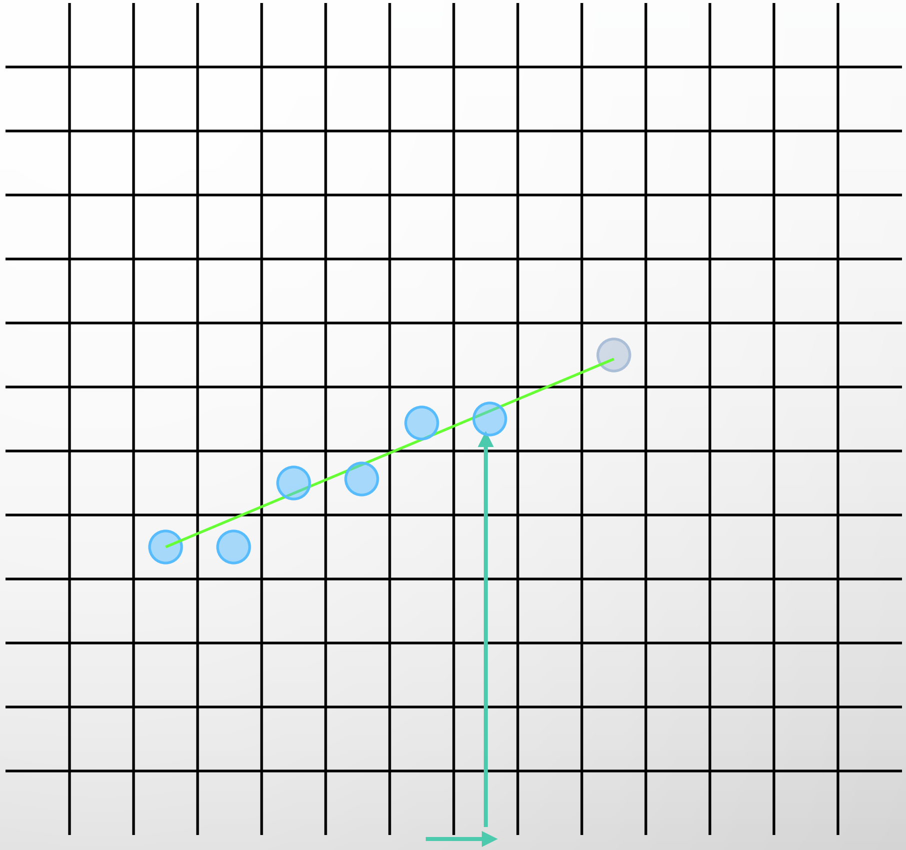
Example

Sample at unit x:

$$x_{k+1} = x_k + \Delta x$$
$$= x_k + 1$$

Corresponding y pos.:

$$y_{k+1} = y_k + \Delta y$$
$$= y_k + m \cdot \Delta x$$
$$= y_k + m \cdot (1)$$



DDA ALGORITHM

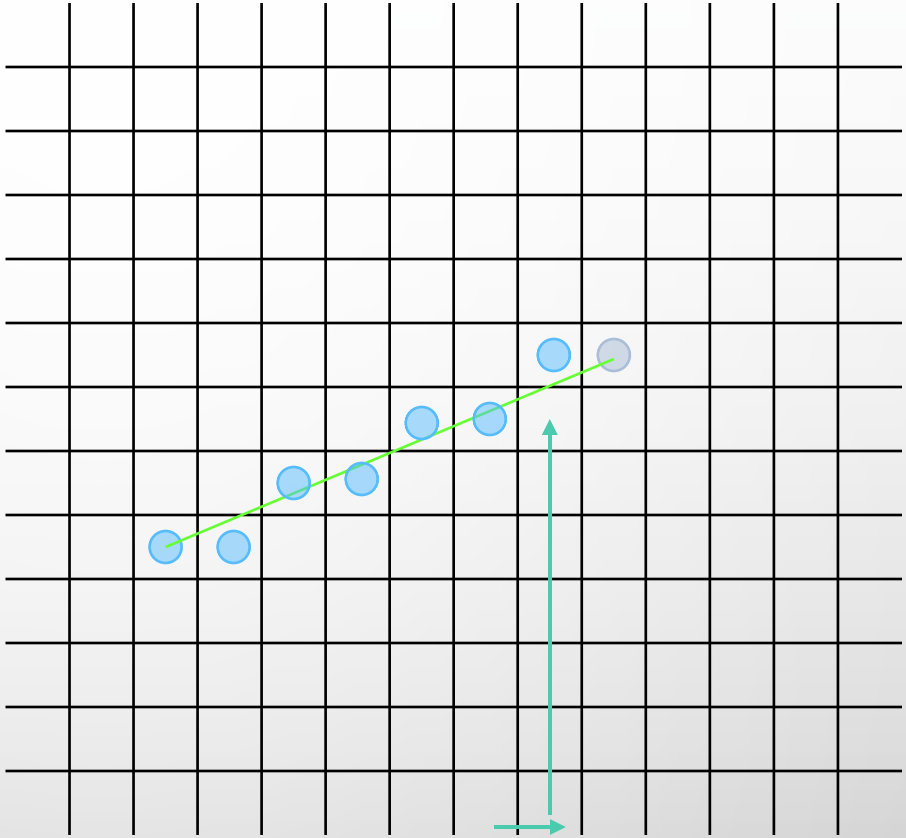
Example

Sample at unit x:

$$x_{k+1} = x_k + \Delta x$$
$$= x_k + 1$$

Corresponding y pos.:

$$y_{k+1} = y_k + \Delta y$$
$$= y_k + m \cdot \Delta x$$
$$= y_k + m \cdot (1)$$



DDA ALGORITHM

Example

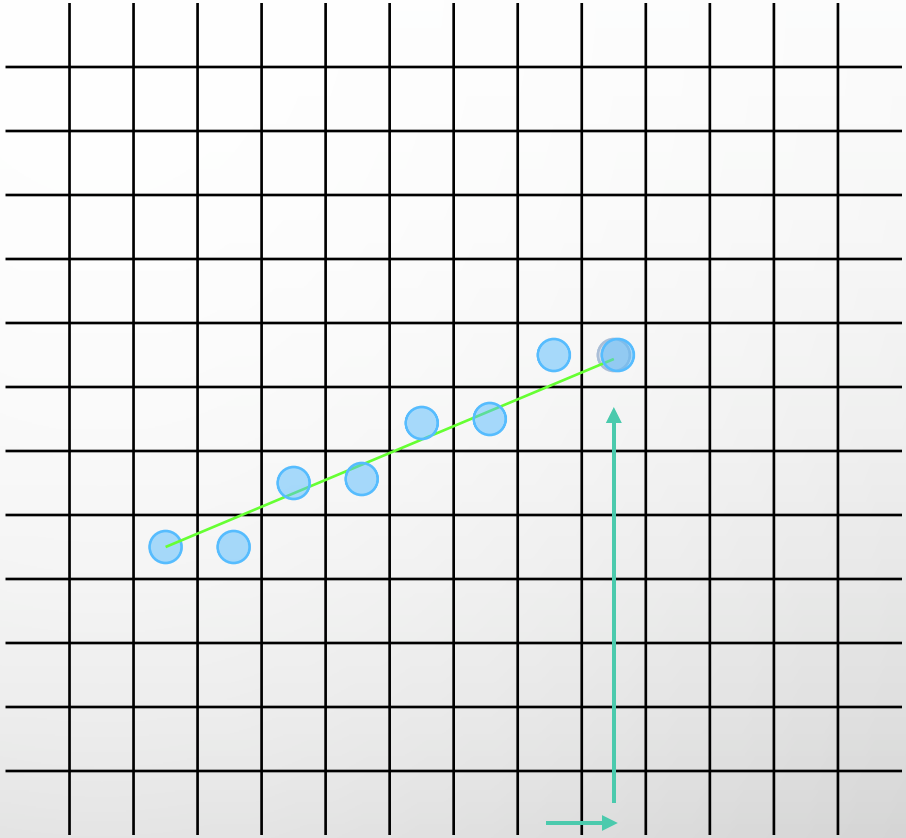
Sample at unit x:

$$x_{k+1} = x_k + \Delta x$$
$$= x_k + 1$$

Corresponding y pos.:

$$y_{k+1} = y_k + \Delta y$$
$$= y_k + m \cdot \Delta x$$
$$= y_k + m \cdot (1)$$

Consider endpoints:
P1(0,0), P2(7,4)



DDA ALGORITHM

Exercise

1. Consider endpoints:

   P1(0,0), p2(6, 4)

   Calculate the points that made up the line P1 P2

2. Now, consider endpoints:

   P3(0,0), P4(4, 6)

   Calculate the points that made up the line P3 P4

What happened with P3P4??????

DDA ALGORITHM

Limitations

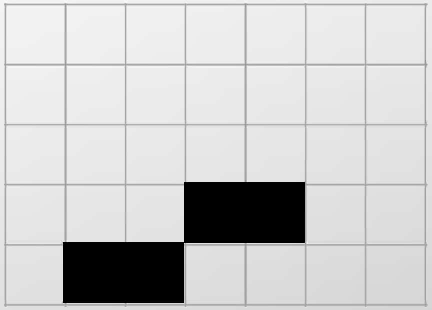
- Rounding integers takes time
- Variables y and m must be a real or fractional binary because the slope is a fraction
- Real variables have limited precision, summing an inexact slope m repetitively introduces a cumulative error buildup

DDA ALGORITHM

Rounding error

- Note that the actual pixel position is actually stored as a REAL number (in C/C++/java a float or a double)
- But we round off to the nearest whole number just before we draw the pixel.
- e.g. If m=0.333 …

X	Y	Rounded { x, y }
1.0	0.33	{ 1, 0 }
2.0	0.66	{ 2, 0 }
3.0	0.99	{ 3, 1 }
4.0	1.32	{ 4, 1 }



LINE DRAWING ALGORITHMS

1. DDA line algorithm
2. **Bresenham’s line algorithm**

BRESENHAM'S LINE ALGORITHM

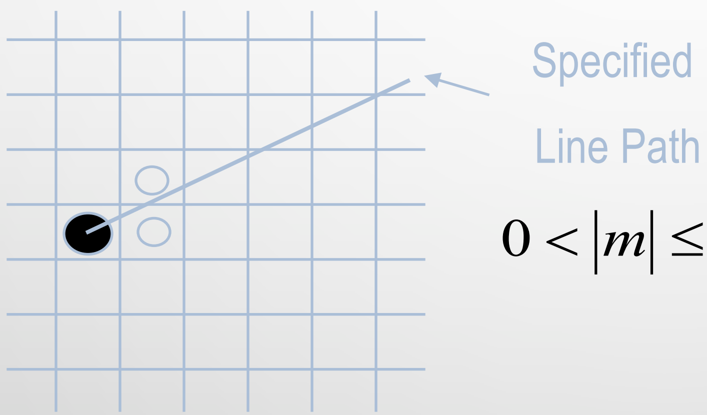
Introduction

- One disadvantage of DDA is the ***rounding part*** which can be expensive
- Developed by jack Bresenham
  at IBM in the early 1960s
- One of the earliest algorithms in computer graphics
- The algorithm is based on essentially the same principles but is completely based on integer variables

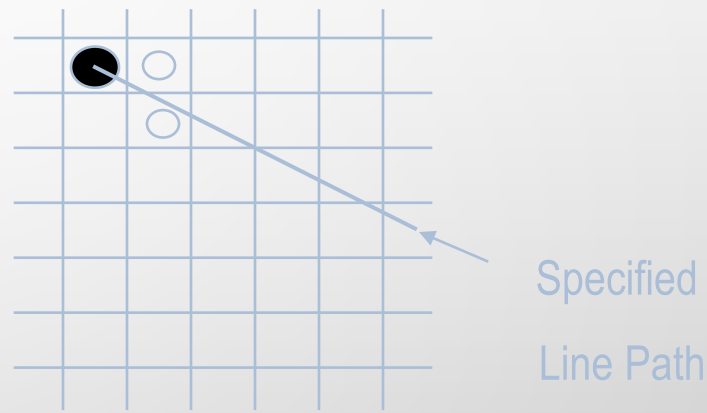
BRESENHAM'S LINE ALGORITHM

Basic concept

- Find the closest integer coordinates to the actual line path using only integer arithmetic
- Candidate for the next pixel position



$$0 < |m| \le 1$$



- No division, efficient comparison, no floating point operations

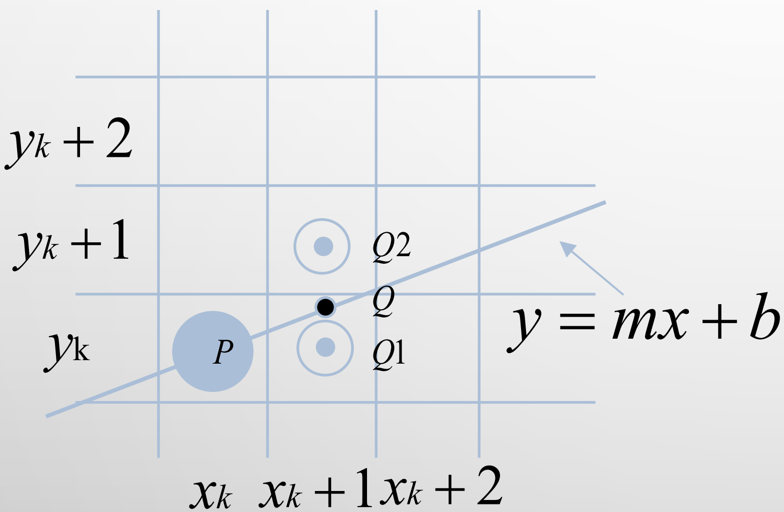
BRESENHAM’S LINE ALGORITHM

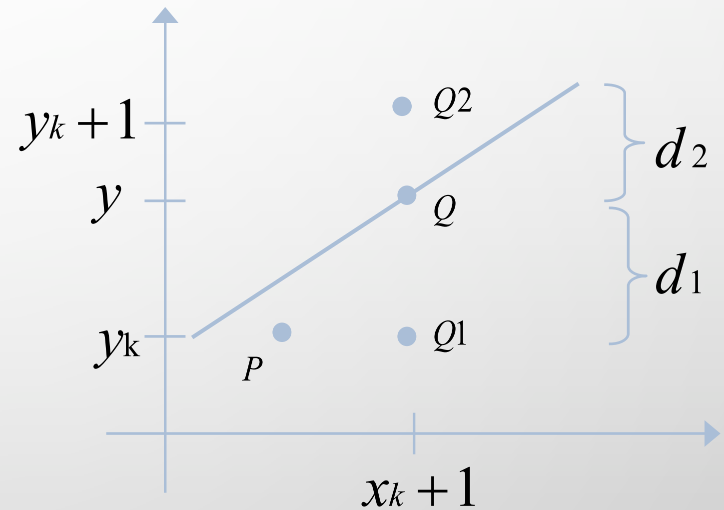
Derivation

- The algorithm is derived for a line having slope $0 < m < 1$ in the first quadrant.
- Pixel position along the line are plotted by taking unit step increments along the x-direction and determining y-coordinate value of the nearest pixel to the line at each step.
- Can be generalized for all the cases

BRESENHAM'S LINE ALGORITHM

- Let us assume that $p(x_k, y_k)$ is the currently plotted pixel. $Q(x_{k+1}, y_{k+1}) \leftrightarrow (x_{k+1}, y)$ is the next point along the actual path of line. We need to decide next pixel to be plotted from two candidate positions $q1(x_k+1, y_k)$ or $q2(x_k+1, y_k+1)$





$$0 < m \le 1, \quad x_k < x_l, \quad k < l$$

BRESENHAM'S LINE ALGORITHM

Given the equation of line

$$y = mx + b$$

Thus actual value of y at $x = x_{k+1}$ is given by

$$y = mx_{k+1} + b = m(x_k + 1) + b$$

Let $d_1 = |QQ_1| =$ *distance of* y_k *from actual value of* y

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

$d_2 = |QQ_2| =$ *distance of actual value of* y *from* $y_k + 1$

$$= y_{k+1} - y = (y_k + 1) - [m(x_k + 1) + b]$$

The difference between these 2 separations is

$$d_1 - d_2 = 2m(x_k + 1) + 2b - y_k - (y_k + 1)$$

$$= 2m(x_k + 1) - 2y_k + 2b - 1$$

BRESENHAM'S LINE ALGORITHM

We can define a decision parameter p_k for the k[th] step to by simplifying above equation such that the sign of p_k is the same as the sign of d_1-d_2, but involves only integer calculations.

Define $p_k = \delta x\,(d_1 - d_2)$

$$= \Delta x(2m(x_k + 1) - 2y_k + 2b - 1)$$

$$= \Delta x(2\frac{\Delta y}{\Delta x}(x_k + 1) - 2y_k + 2b - 1)$$

$$= 2\Delta y \cdot x_k - 2\Delta x \cdot y_k + 2\Delta y + \Delta x(2b - 1)$$

$$= 2\Delta y \cdot x_k - 2\Delta x \cdot y_k + c$$

where $c = 2\Delta y + \Delta x(2b - 1)$ a constant

BRESENHAM'S LINE ALGORITHM

If $p_k < 0$

$\Rightarrow (d_1 - d_2) < 0$

$\Rightarrow$ Distance d_1 is less than d_2

$\Rightarrow y_k$ is closer to line-path

hence $q_1(x_k+1, y_k)$ is the better choice

Else

$q_2(x_k+1, y_k+1)$ is the better choice

Thus if the parameter p_k is negative lower pixel is plotted else upper pixel is plotted

BRESENHAM'S LINE ALGORITHM

To put p_k in the iterative form, we derived that

$$p_k = 2\Delta y.x_k - 2\Delta x.y_k + c$$

Replacing $k = k + 1$

$$p_{k+1} = 2\Delta y.x_{k+1} - 2\Delta x.y_{k+1} + c$$

subtract p_k from p_{k+1}

$$p_{k+1} - p_k = 2\Delta y(x_{k+1} - x_k) - 2\Delta x(y_{k+1} - y_k)$$

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x(y_{k+1} - y_k)$$

$$y_{k+1} = \begin{cases} y_k & \text{if } p_k < 0 \\ y_k + 1 & \text{otherwise} \end{cases}$$

$$\therefore\ p_{k+1} = \begin{cases} p_k + 2\Delta y & \text{if } p_k < 0 \\ p_k + 2\Delta y - 2\Delta x & \text{otherwise} \end{cases}$$

BRESENHAM'S LINE ALGORITHM

The first parameter p_0 is directly computed as:

$$p_0 = 2\ \delta y.x_0 - 2\ \delta x.y_0 + c$$

$$= 2\ \delta y.x_0 - 2\ \delta x.y_0 + 2\ \delta y + \delta x\ (2b-1)$$

Since (x_0,y_0) satisfies the line equation , we also have

$$y_0 = \delta y / \delta x * x_0 + b$$

$$b = y_0 - \delta y / \delta x * x_0$$

Combining the above 2 equations , we will have

$$p_0 = 2\delta y - \delta x$$

The constants $2\delta y$, $2\delta y - \delta x$ and $2\delta y\text{-}2\delta x$ are calculated once.

BRESENHAM'S LINE ALGORITHM

Steps for Bresenham's line drawing algorithm (for $|m| < 1.0$)

1. **Input the two line end-points (x_0, y_0) and (x_1, y_1)**
2. **Plot the point (x_0, y_0)**
3. **Compute** $\delta x = x_1 - x_0$, $\delta y = y_1 - y_0$
4. **Initialize** $p_0 = 2\delta y - \delta x$
5. **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)

   $p_k = p_k + 2\delta y$

   Else

   the next point to plot is (x_k+1, y_k+1)

   $P_k = p_k + 2\delta y - 2\delta x$

6. **Repeat step 5 (δx – 1) times**
7. **Exit**

BRESENHAM'S LINE ALGORITHM

Exercise

Calculate pixel positions that made up the line connecting endpoints: (12, 10) and (17, 14).

1. (x_0, y_0) = ?
2. Δx = ?, Δy =?, $2\Delta y$ = ?, $2\Delta y - 2\Delta x$ =?
3. $p_0 = 2\Delta y - \Delta x$ =?

k	p_k	(x_{k+1}, y_{k+1})

Bresenham's Line Algorithm

Exercise

Calculate pixel positions that made up the line connecting endpoints: (12, 10) and (17, 14).

1. $(x_0, y_0) = (12,10)$
2. $\Delta x = 5$, $\Delta y = 4$, $2\Delta y = 8$, $2\Delta y - 2\Delta x = -2$
3. $p_0 = 2\Delta y - \Delta x = 3$

k	p_k	(x_{k+1}, y_{k+1})
0	3	
1		
2		

Bresenham's Line Algorithm

Exercise

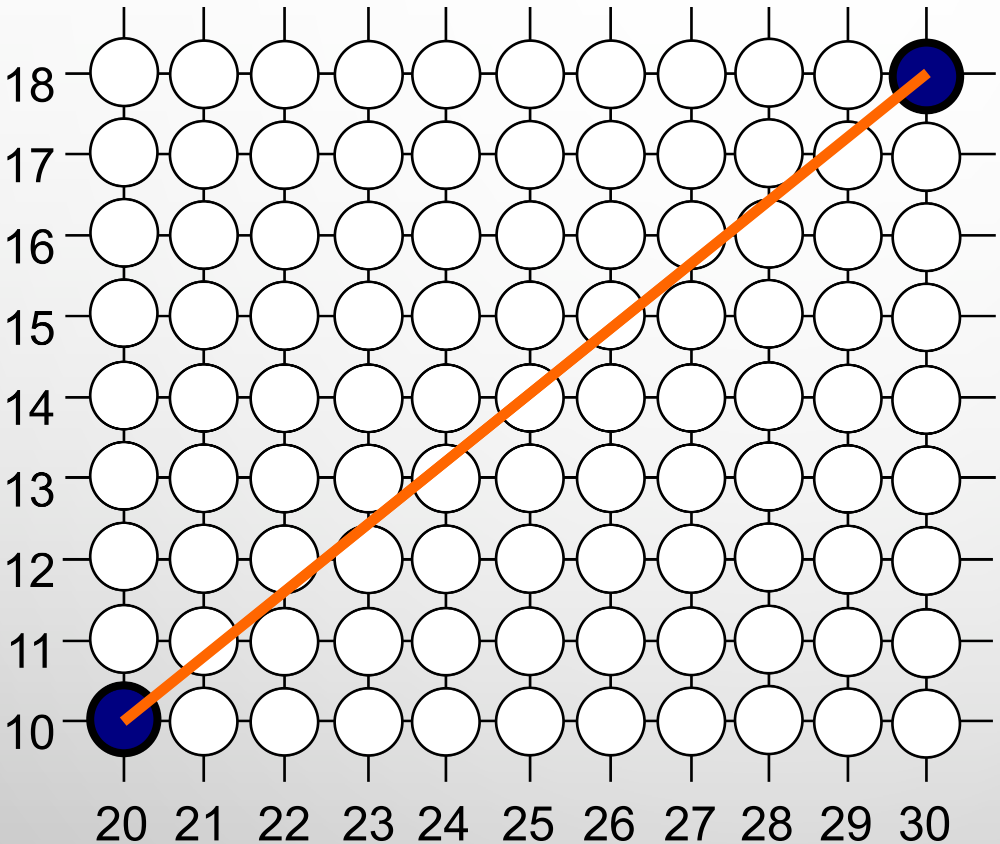
Calculate pixel positions that made up the line connecting endpoints: (12, 10) and (17, 14).

1. $(x_0, y_0) = (12,10)$
2. $\Delta x = 5$, $\Delta y = 4$, $2\Delta y = 8$, $2\Delta y - 2\Delta x = -2$
3. $p_0 = 2\Delta y - \Delta x = 3$

k	p_k	(x_{k+1}, y_{k+1})
0	3	(13, 11)
1	1	(14, 12)
2	-1	(15, 12)
3	7	(16, 13)
4	5	(17, 14)

BRESENHAM'S LINE ALGORITHM

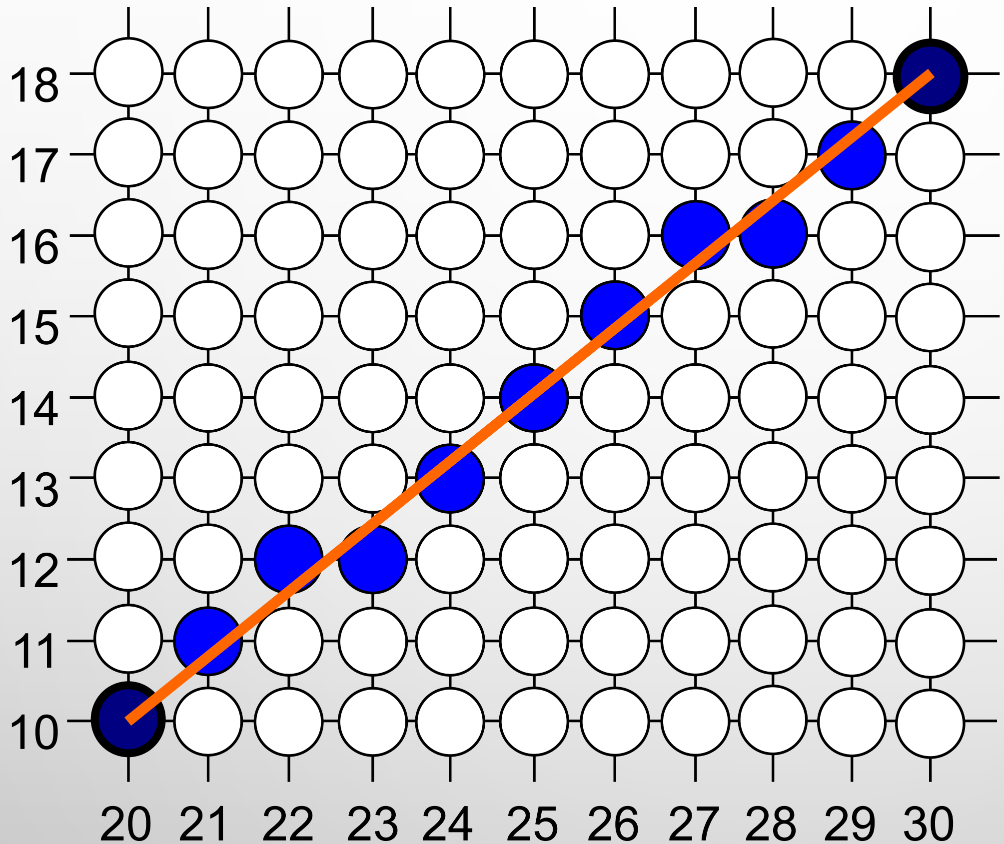
Exercise: trace for (20,10) to (30,18)



k	p_k	(x_{k+1}, y_{k+1})

BRESENHAM'S LINE ALGORITHM

Answer



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)