# Contents

In today's lecture we'll have a look at:

- Bresenham's line drawing algorithm

# The Bresenham Line Algorithm

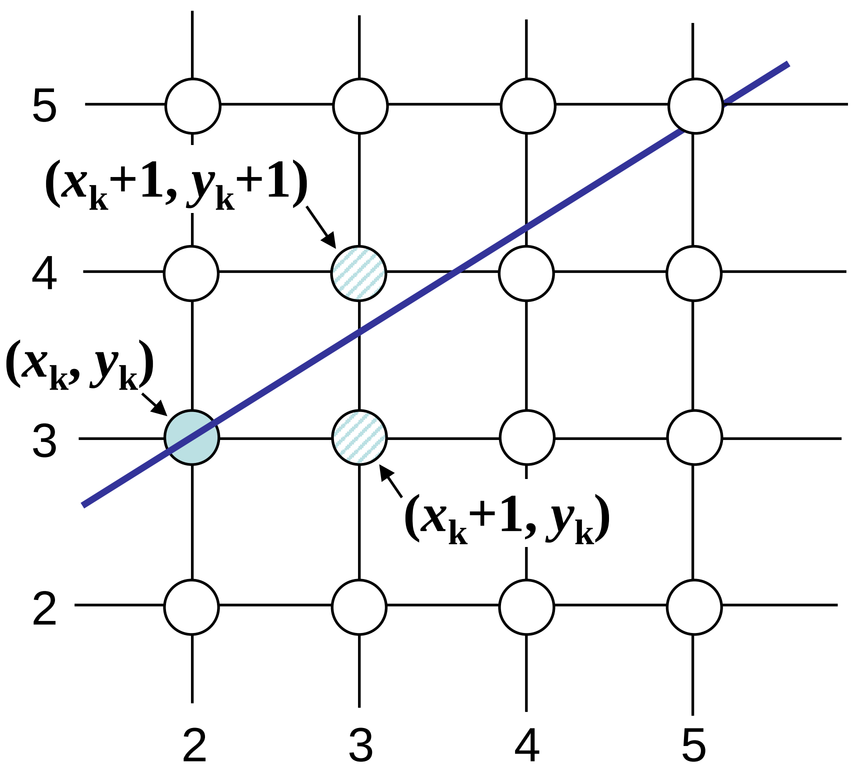
The Bresenham algorithm is another incremental scan conversion algorithm

The big advantage of this algorithm is that it uses only integer calculations

Jack Bresenham worked for 27 years at IBM before entering academia. Bresenham developed his famous algorithms at IBM in the early 1960s

# The Big Idea

Move across the $x$ axis in unit intervals and at each step choose between two different $y$ coordinates

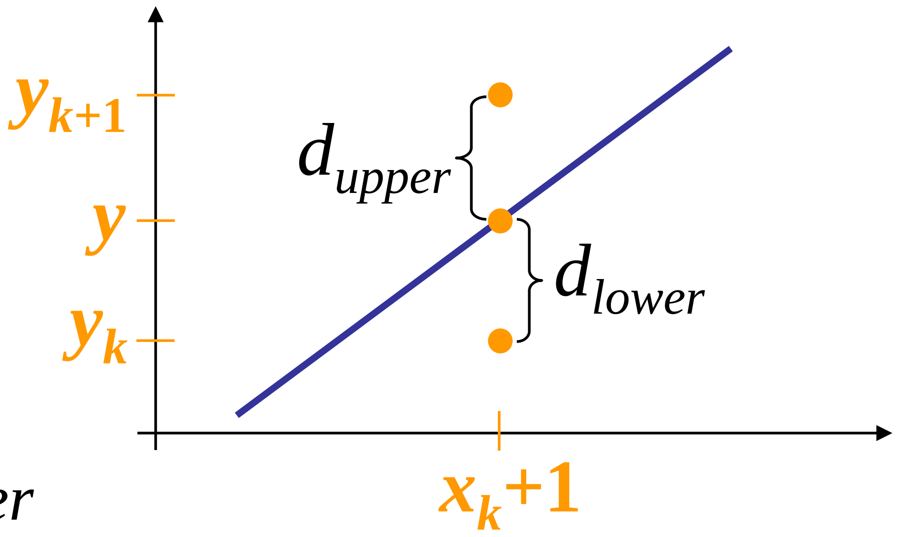



For example, from position (2, 3) we have to choose between (3, 3) and (3, 4)

We would like the point that is closer to the original line

# Deriving The Bresenham Line Algorithm

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:

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

# Deriving The Bresenham Line Algorithm (cont…)

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

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

and:

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

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

# Deriving The Bresenham Line Algorithm (cont…)

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

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

Let's substitute *m* with $\Delta y / \Delta x$ where $\Delta x$ and $\Delta y$ are the differences between the end-points:

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

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

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

# Deriving The Bresenham Line Algorithm (cont…)

So, a decision parameter $p_k$ for the $k$th step along a line is given by:

$$p_k = \Delta x(d_{lower} - d_{upper})$$
$$= 2\Delta y \cdot x_k - 2\Delta x \cdot y_k + c$$

The sign of the decision parameter $p_k$ is the same as that of $d_{lower} - d_{upper}$

If $p_k$ is negative, then we choose the lower pixel, otherwise we choose the upper pixel

# Deriving The Bresenham Line Algorithm (cont…)

Remember coordinate changes occur along the $x$ axis in unit steps so we can do everything with integer calculations

At step $k$+1 the decision parameter is given as:

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

Subtracting $p_k$ from this we get:

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

# Deriving The Bresenham Line Algorithm (cont…)

But, $x_{k+1}$ is the same as $x_k+1$ so:

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

where $y_{k+1}$ - $y_k$ is either 0 or 1 depending on the sign of $p_k$

The first decision parameter p0 is evaluated at (x0, y0) is given as:

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

# The Bresenham Line Algorithm

BRESENHAM'S LINE DRAWING ALGORITHM
(for $|m| < 1.0$)

1. Input the two line end-points, storing the left end-point in $(x_0, y_0)$
2. Plot the point $(x_0, y_0)$
3. Calculate the constants $\Delta x$, $\Delta y$, $2\Delta y$, and $(2\Delta y - 2\Delta x)$ and get the first value for the decision parameter as:

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

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:

$$p_{k+1} = p_k + 2\Delta y$$

# The Bresenham Line Algorithm (cont…)

Otherwise, the next point to plot is $(x_k+1, y_k+1)$ and:

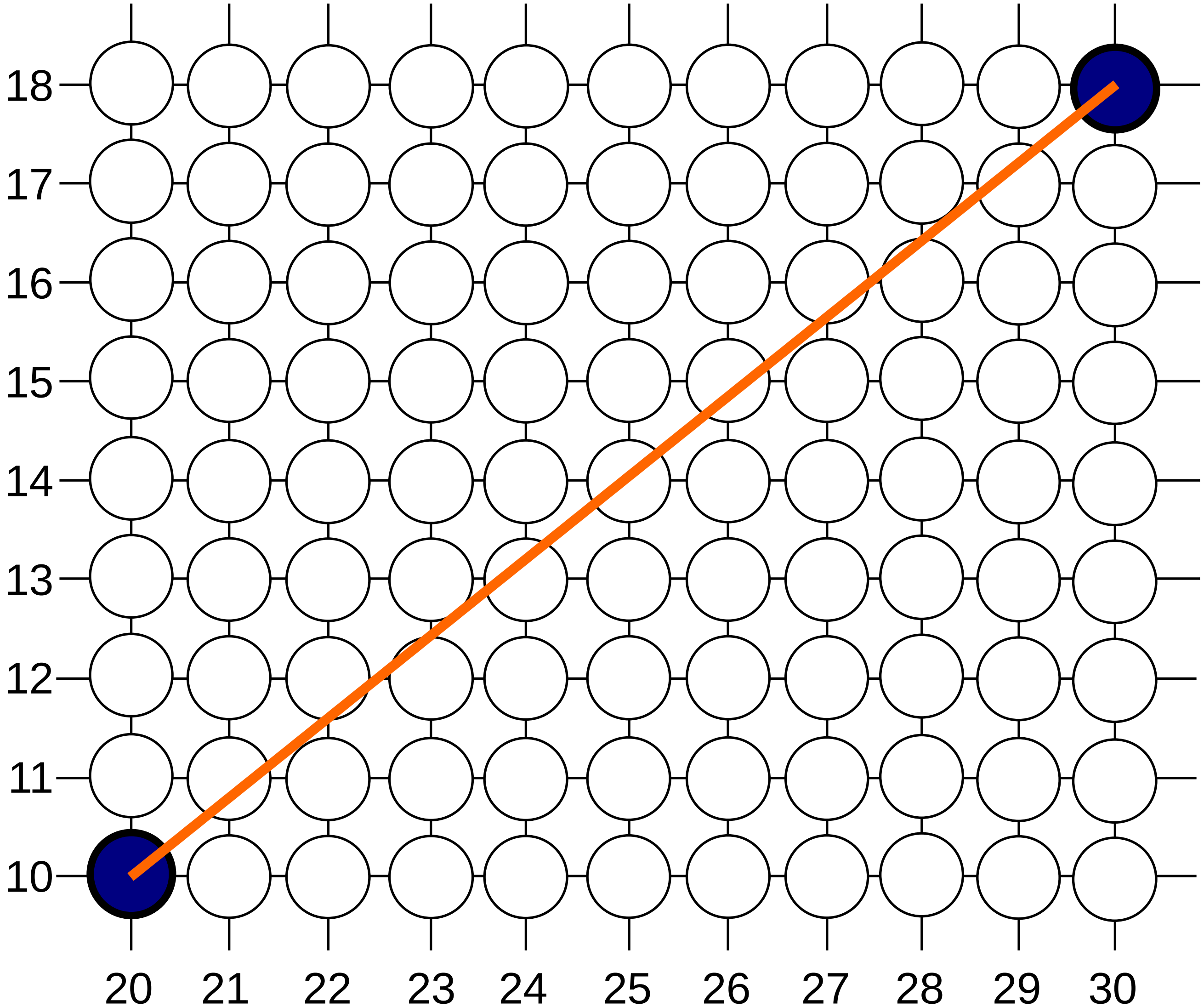
$$p_{k+1} = p_k + 2\Delta y - 2\Delta x$$

5. Repeat step 4 $(\Delta x - 1)$ times

# Bresenham Example

- Let's have a go at this
- Let's plot the line from (20, 10) to (30, 18)
- First off calculate all of the constants:
  - $\Delta x$: 10
  - $\Delta y$: 8
  - $2\Delta y$: 16
  - $2\Delta y - 2\Delta x$: -4
- Calculate the initial decision parameter $p_0$:
  - $p0 = 2\Delta y - \Delta x = 6$

# Bresenham Example (cont…)

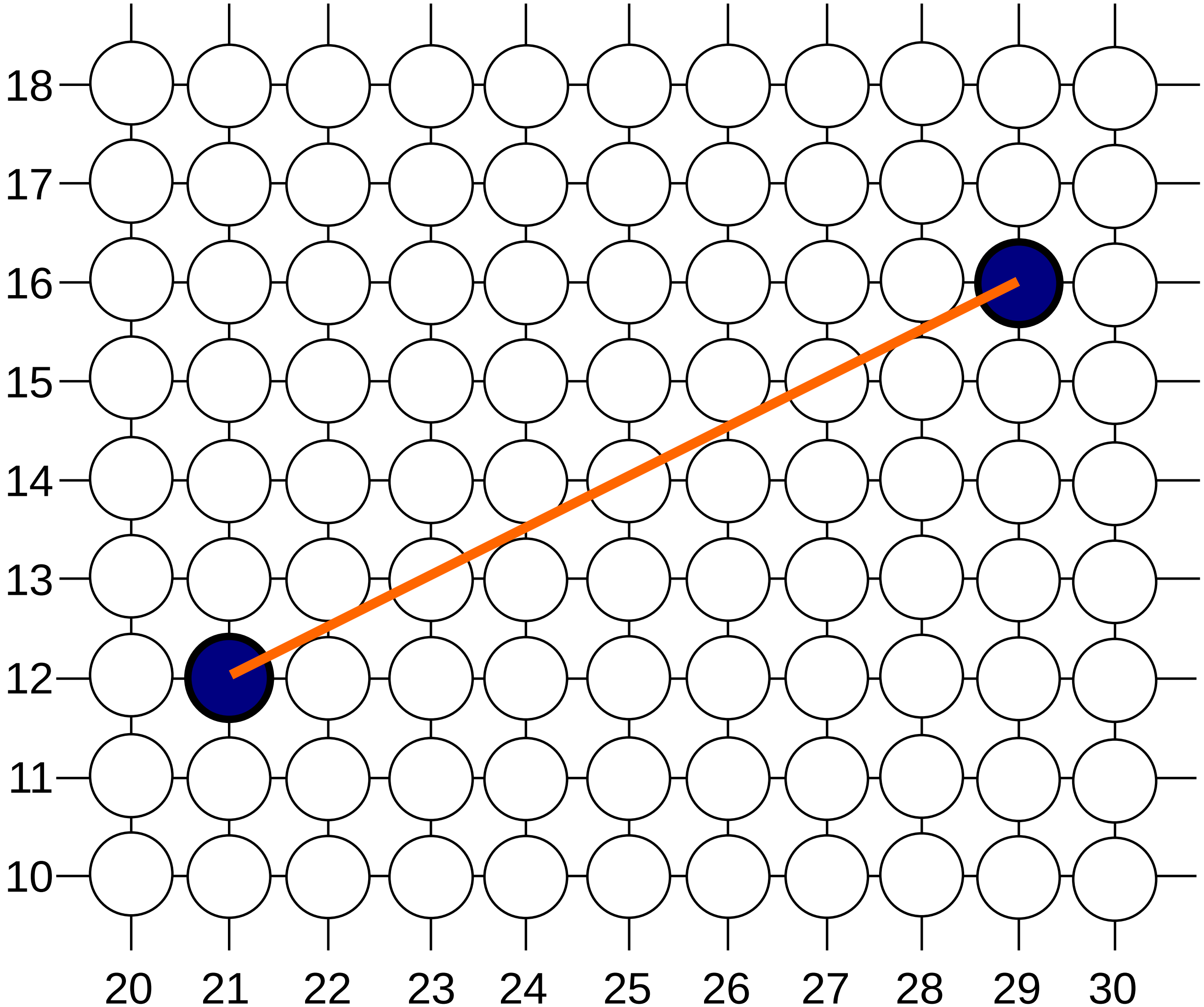



| k | $p_k$ | $(x_{k+1}, y_{k+1})$ |
|---|---|---|
| 0 | | |
| 1 | | |
| 2 | | |
| 3 | | |
| 4 | | |
| 5 | | |
| 6 | | |
| 7 | | |
| 8 | | |
| 9 | | |

# Bresenham Exercise

Go through the steps of the Bresenham line drawing algorithm for a line going from (21,12) to (29,16)

# Bresenham Exercise (cont…)




| k | $p_k$ | $(x_{k+1},y_{k+1})$ |
|---|---|---|
| 0 | | |
| 1 | | |
| 2 | | |
| 3 | | |
| 4 | | |
| 5 | | |
| 6 | | |
| 7 | | |
| 8 | | |

# Bresenham Line Algorithm Summary

The Bresenham 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