2D Scan-line Conversion

CS 4610/7610 Computer Graphics

2D Scan-line Conversion

- Naïve
- DDA algorithm
- Bresenham’s algorithm

2D Scan-line Conversion

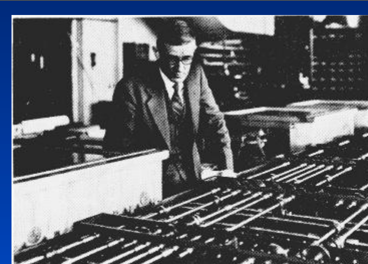
- Naïve: Explicitly evaluate equation of line using multiply and add:

$$y = mx + b$$

DDA Line Drawing

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



The original differential analyzer was a physical machine developed by Vannevar Bush at MIT in the 1930's in order to solve ordinary differential equatiom.

DDA Algorithm

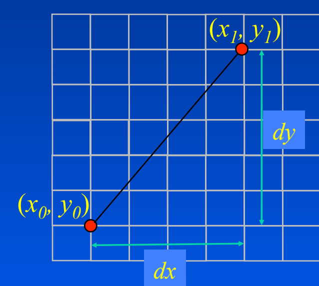
- Simple line drawing algorithm
- Named after Digital Differential Analyzer

$$m = \frac{y_1 - y_0}{x_1 - x_0} = \frac{dy}{dx}$$

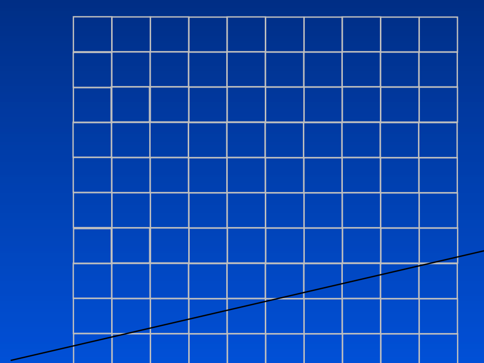
$$0 \le m \le 1$$

$$\Delta y = m\Delta x, \quad \Delta x = 1$$

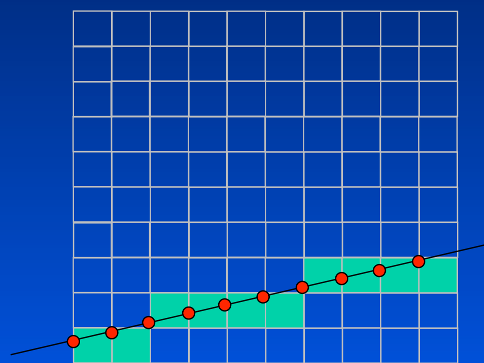
$$So, \Delta y = m$$



DDA Algorithm

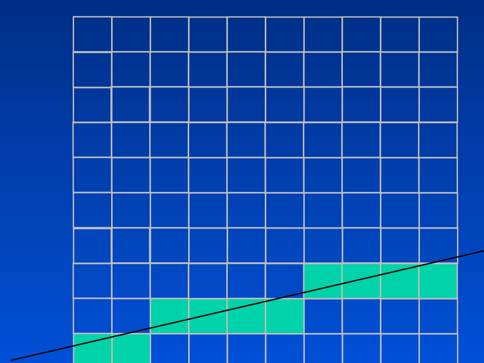


DDA Algorithm



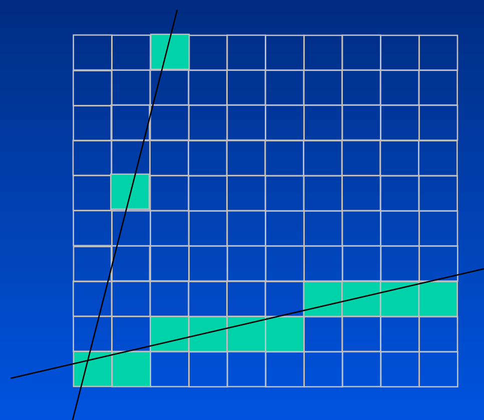
University of Missouri at Columbia

DDA Algorithm



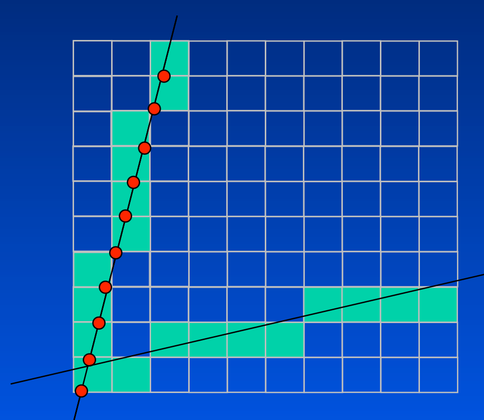
University of Missouri at Columbia

DDA Algorithm – Problem with Steep Lines



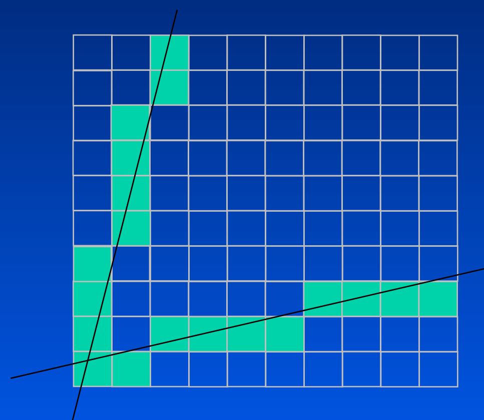
Avoid jaggies

DDA Algorithm – Steep Lines



Swap the role of x and y

DDA Algorithm



2D Rasterization of Lines

- DDA algorithm
  - Uses floating point variables
  - No multiplication only addition
  - Needs rounding which is expensive and can accumulate error/drift
- Bresenham's algorithm
  - Fast, Integer variables only
  - No floating point computations (*, /)
  - No rounding

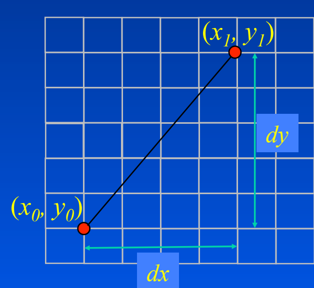
The Bresenham algorithm is another incremental scan conversion algorithm

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

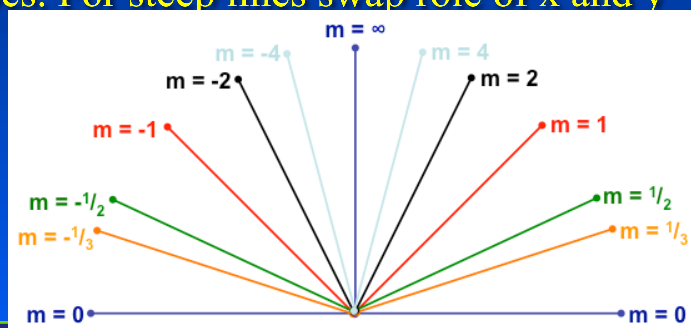
Bresenham's Midpoint Algorithm

- DDA is simple, efficient, but needs floating point add
- Bresenham uses only integer additions



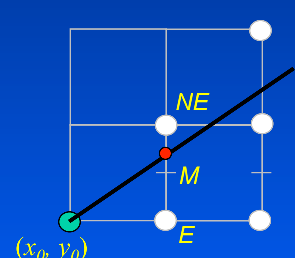
Lines and Slopes

- Slope of a line (m) is defined by its start and end coordinates, rise over run
- Diagram shows examples of lines and their slopes. For steep lines swap role of x and y

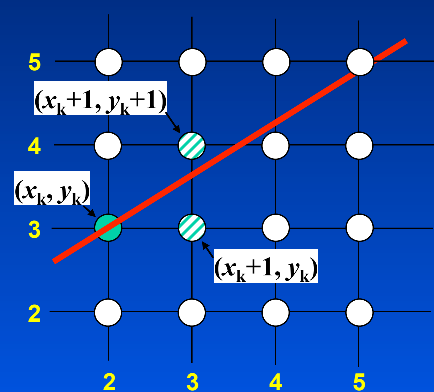


Bresenham's Midpoint Algorithm

- Select from two choices: *NE* or *E* pixel depending on the relative position of the midpoint *M* and the line
- Choose *E* pixel if *M* is above the line
- Choose *NE* pixel if *M* is below the line



Basic Idea for 0<m<1



Take unit steps in x and at each step choose between y-coord

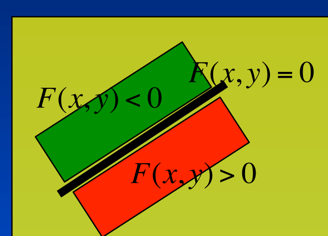
Implicit Equation of Line

Line equation:

$$y = mx + b = \frac{\Delta y}{\Delta x} x + b$$

Implicit form:

$$F(x, y) = \Delta y \cdot x - \Delta x \cdot y + b \cdot \Delta x = 0$$



For points above the line F(x,y) is negative

For points below the line F(x,y) is positive

Bresenham's Midpoint Algorithm

- Choose *NE* if decision variable *p* is positive
- Choose *E* if decision variable *p* is negative

Line equation:

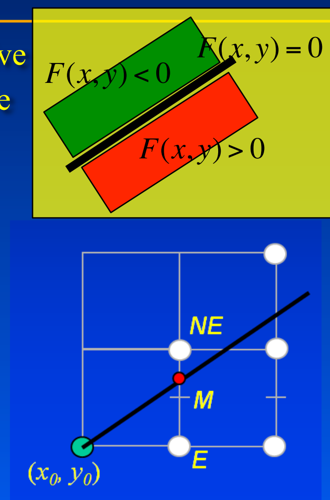
$$y = mx + b = \frac{\Delta y}{\Delta x} x + b$$

Implicit form:

$$F(x, y) = \Delta y \cdot x - \Delta x \cdot y + b \cdot \Delta x = 0$$

Since point (x_0, y_0) is on the line, so

$$F(x_0, y_0) = \Delta y \cdot x_0 - \Delta x \cdot y_0 + b \cdot \Delta x = 0$$



Bresenham's Midpoint Algorithm - Initialization

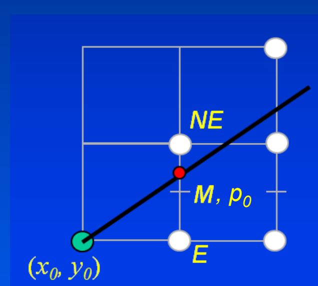
- Choose *NE* if decision variable *p* is positive
- Choose *E* if decision variable *p* is negative

Check the sign of a decision variable p_0 at point M:

$$F(x_0 + 1, y_0 + \tfrac{1}{2}) = \Delta y - \Delta x \cdot \tfrac{1}{2}$$

$$2F(x_0 + 1, y_0 + \tfrac{1}{2}) = 2 \cdot \Delta y - \Delta x$$

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



Incremental Calculation of the Decision Variable p_k

Since $p_k = 2F(x_k+1, y_k+\frac{1}{2}) = 2\Delta y\cdot(x_k+1) - 2\Delta x\cdot(y_k+\frac{1}{2}) + 2b\cdot\Delta x$

and $p_{k+1} = 2F(x_{k+1}+1, y_{k+1}+\frac{1}{2}) = 2\Delta y\cdot(x_{k+1}+1) - 2\Delta x\cdot(y_{k+1}+\frac{1}{2}) + 2b\cdot\Delta x$

Taking the difference gives:

$p_{k+1} - p_k = 2(F(x_{k+1}+1, y_{k+1}+\frac{1}{2}) - F(x_k+1, y_k+\frac{1}{2}))$

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

There are two possible values for $(y_{k+1}-y_k)$:

if $p_k < 0$ we choose East pixel,

$p_{k+1} = 2F(x_k+2, y_k+\frac{1}{2})$

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

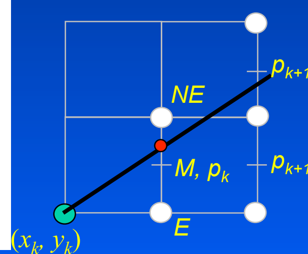
$= p_k + 2\Delta y$

if $p_k \ge 0$ we choose NE pixel,

$p_{k+1} = 2F(x_k+2, y_k+\frac{3}{2})$

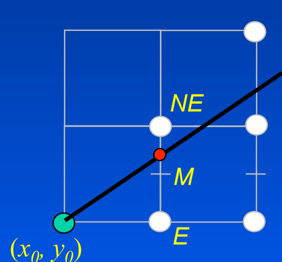
$= 2\Delta y\cdot(x_k+2) - 2\Delta x\cdot(y_k+\frac{3}{2}) + 2b\cdot\Delta x$

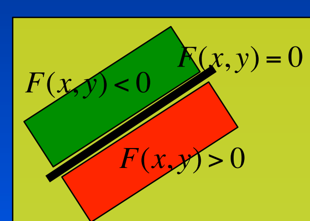
$= p_k + 2\Delta y - 2\Delta x$



Bresenham's Midpoint Algorithm

$$p_{k+1} = \begin{cases} p_k + 2\cdot\Delta y, & \text{if } p_k < 0 \text{ choose } E \\ p_k + 2\cdot\Delta y - 2\cdot\Delta x, & \text{if } p_k \ge 0 \text{ choose } NE \end{cases}$$





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 Δx, Δy, $2\Delta y$, 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:

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

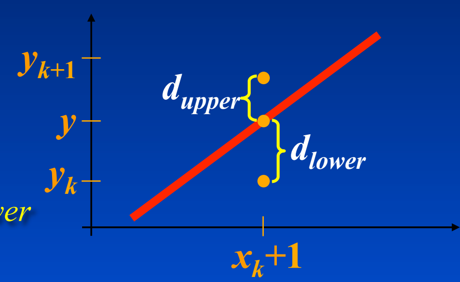
5. Repeat Step 4, $(\Delta x - 1)$ times

•**WARNING!** The algorithm and derivation above assumes slopes are less than 1. For other slopes adjust the algorithm slightly

University of Missouri at Columbia

Bresenham Line Algorithm – Alternative Derivation

At position x_k+1 the vertical distance from the true line are labelled d_{upper} and d_{lower}



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

University of Missouri at Columbia

Bresenham Line Algorithm – Alternative Derivation

Expressions for d_{upper} and d_{lower} are:

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

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

Determine which pixel is closer to the mathematical line

University of Missouri at Columbia

Bresenham Line Algorithm – Alternative Derivation

Decision based on the difference between upper and lower pixel positions:

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

Substitute m with $\Delta y / \Delta x$ where Δx and Δy are the differences between the end-points:

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

Bresenham Line Algorithm – Alternative Derivation

Decision parameter p_k for the kth 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

Bresenham Line Algorithm – Alternative Derivation

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 gives incremental update:

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

Bresenham Line Algorithm – Alternative Derivation

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 p_0 is evaluated at (x_0, y_0) is given as:

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

Bresenham Line Algorithm - Example

Plot the line from (20, 10) to (30, 18)

First calculate the constants (outside of loop):

Δx = 10, Δy = 8

$2\Delta y$ = 16

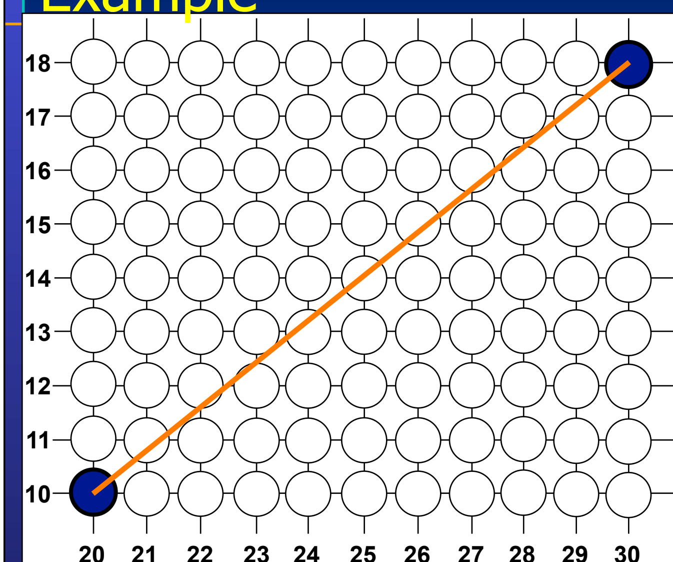
$2\Delta y - 2\Delta x$ = -4

Calculate the initial decision parameter p_0:

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

University of Missouri at Columbia

Bresenham Line Algorithm - Example



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)

Bresenham Line Algorithm - Example

Plot the same line in the reverse direction from right to left: (30, 18) to (20, 10)

First calculate the constants (outside of loop):

Δx = 10, Δy = 8

$2\Delta y$ = 16

But increments are -1

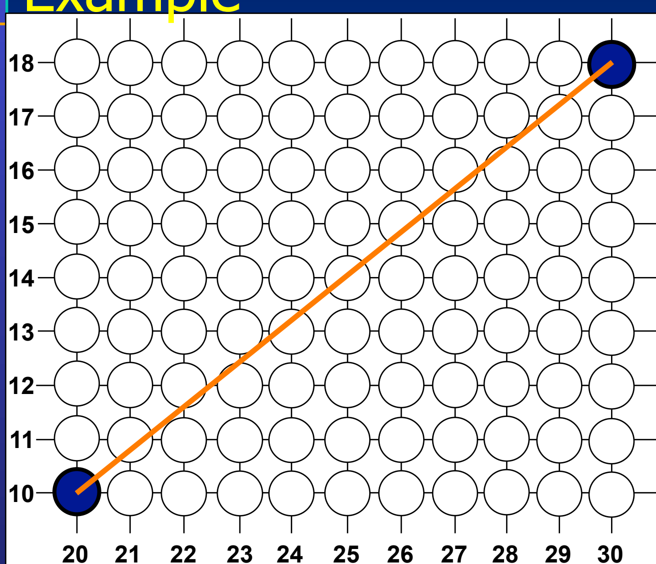
$2\Delta y - 2\Delta x$ = -4

Calculate the initial decision parameter p_0:

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

University of Missouri at Columbia

Bresenham Line Algorithm - Example

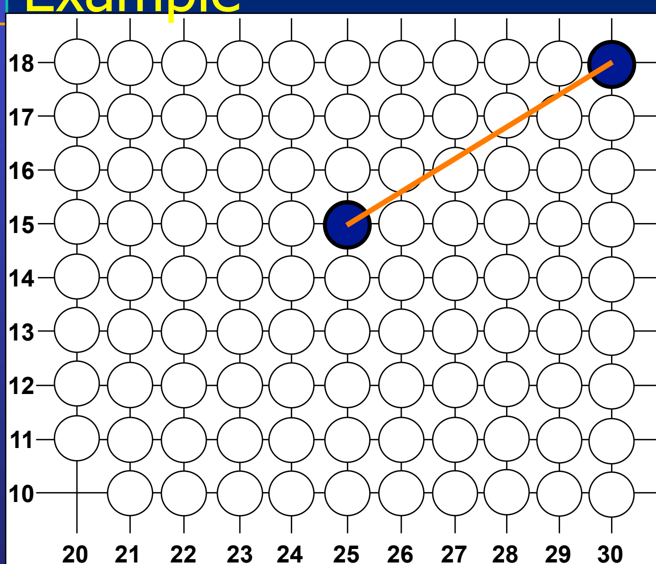


k	p_k	(x_{k+1},y_{k+1})
0	6	(29,17)
1	2	(28,16)
2	-2	(27,16)
3	14	(26,15)
4	10	(25,14)
5	6	(24,13)
6	2	(23,12)
7	-2	(22,12)
8	14	(21,11)
9	10	(20,10)

Bresenham Line Algorithm - Example

Compute the points on the Bresenham line going from (30,18) to (25,15)

Bresenham Line Algorithm - Example



k	p_k	(x_{k+1}, y_{k+1})
0		
1		
2		
3		
4		