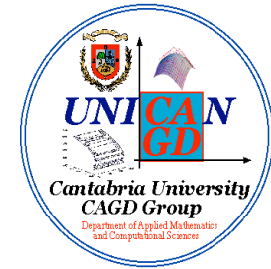




*Department of Applied Mathematics
and Computational Sciences*
University of Cantabria
UC-CAGD Group



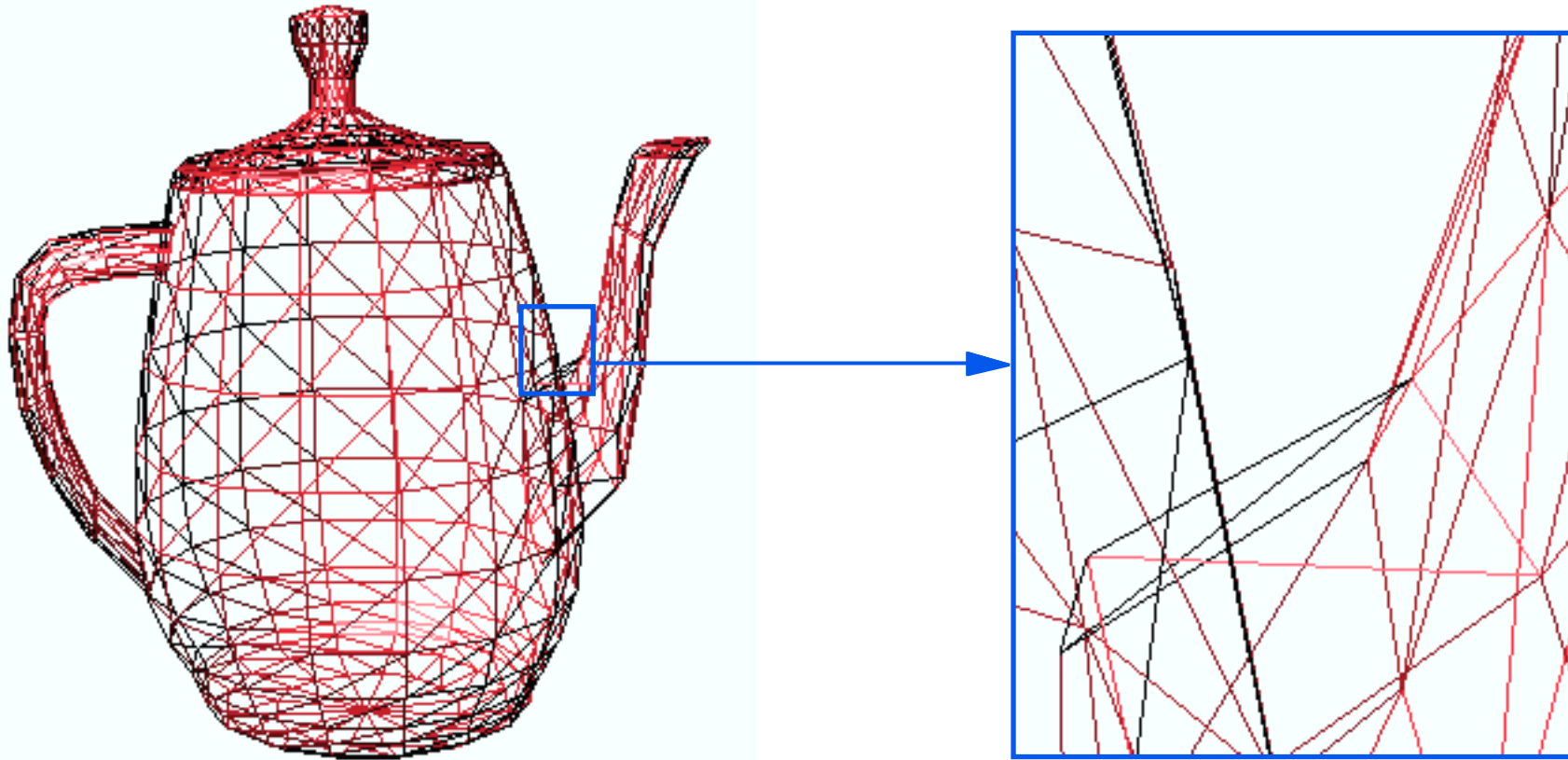
**COMPUTER-AIDED GEOMETRIC DESIGN
AND COMPUTER GRAPHICS:
LINE DRAWING ALGORITHMS**

Andrés Iglesias

e-mail: iglesias@unican.es

Web pages: <http://personales.unican.es/iglesias>
<http://etsiso2.macc.unican.es/~cagd>

Line Drawing Algorithms



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

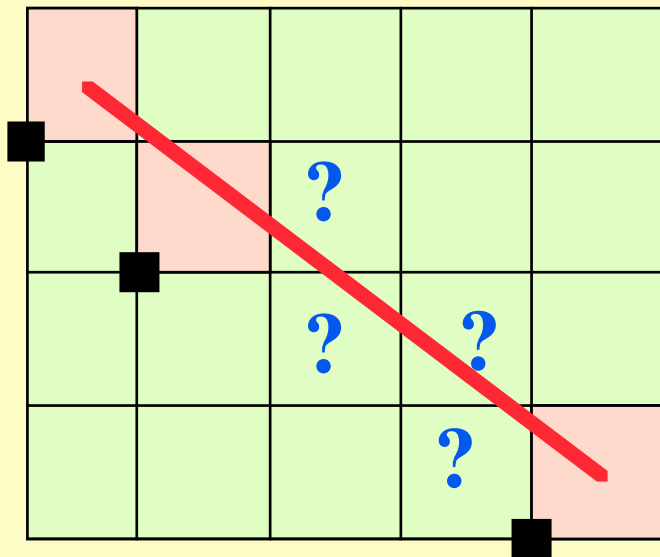
*However, they are **made of pixels***

Line Drawing Algorithms

We are going to analyze how this process is achieved.

Some useful definitions

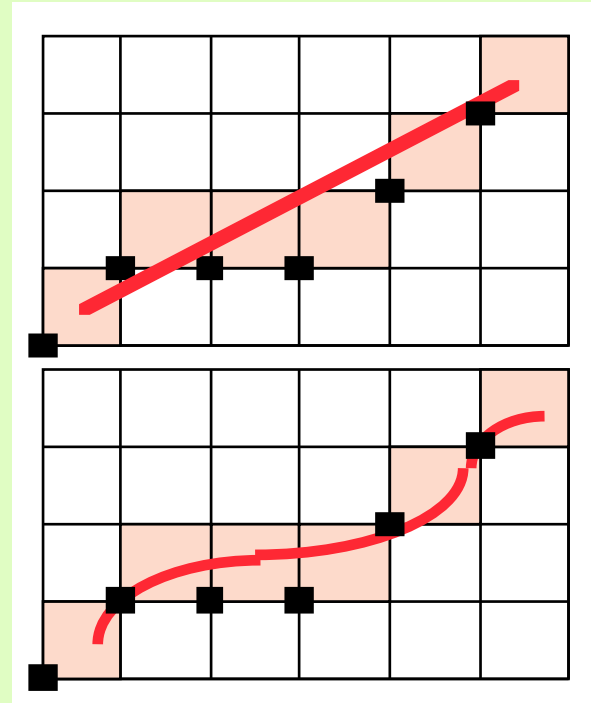
Rasterization: Process of determining which pixels provide the best approximation to a desired line on the screen.



Scan Conversion: Combination of rasterization and generating the picture in scan line order.

General requirements

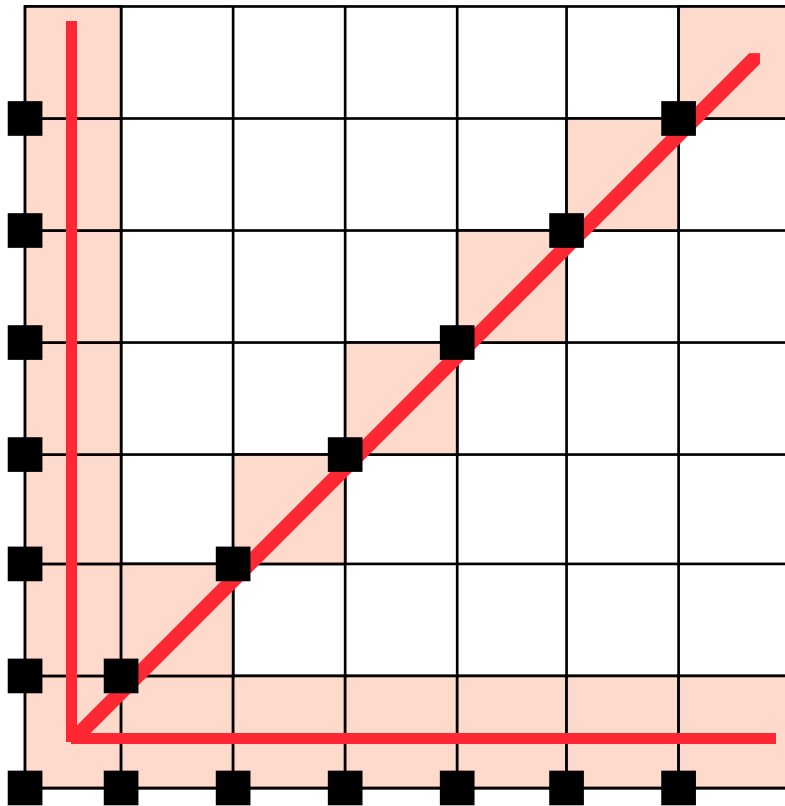
- **Straight lines** must appear as **straight lines**.



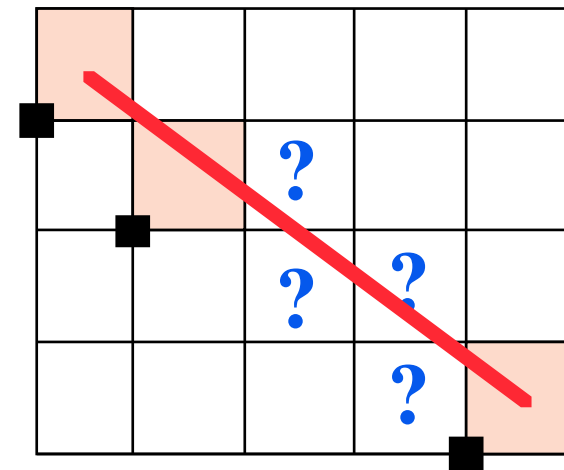
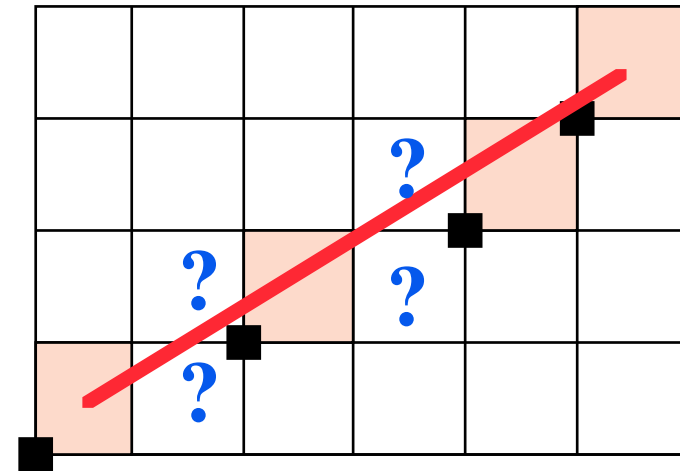
- They must **start** and **end accurately**
- Lines should have **constant brightness** along their length
- Lines should draw rapidly

Line Drawing Algorithms

For horizontal, vertical and 45° lines, the choice of raster elements is obvious. This lines exhibit constant brightness along the length:

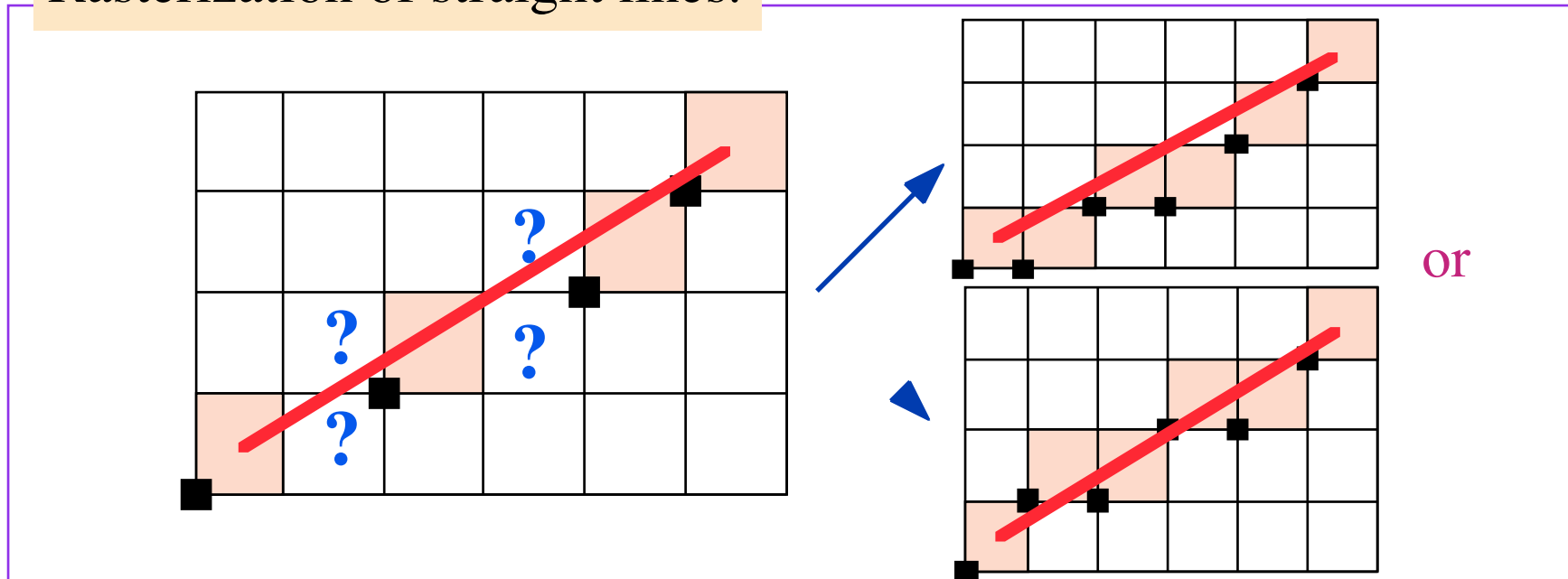


For any other orientation the choice is more difficult:



Line Drawing Algorithms

Rasterization of straight lines.



Rasterization yields **uneven brightness**: Horizontal and vertical lines appear brighter than the 45° lines.

For fixing so, we would need:

1. Calculation of square roots
(increasing CPU time)
2. Multiple brightness levels



Compromise:

1. Calculate only an approximate line
2. Use integer arithmetic
3. Use incremental methods

Line Drawing Algorithms

The equation of a straight line is given by: $y = m \cdot x + b$

Algorithm 1: Direct Scan Conversion

1. Start at the pixel for the left-hand endpoint x_l
2. Step along the pixels horizontally until we reach the right-hand end of the line, x_r
3. For each pixel compute the corresponding y value
4. 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;
}
```

The algorithm performs a **floating-point multiplication for every step in x** . This method therefore requires an enormous number of floating-point multiplications, and is therefore **expensive**.

Line Drawing Algorithms

Algorithm 2: Digital Differential Analyzer (DDA)

The differential equation of a straight line is given by:

$$\frac{dy}{dx} = \text{constant}$$

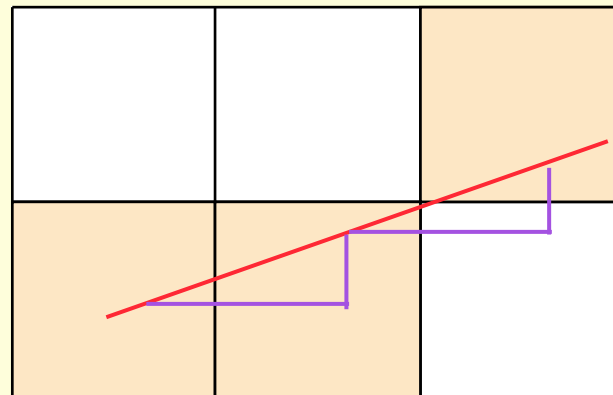
or

$$\frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1}$$

The solution of the finite difference approximation is:

$$x_{i+1} = x_i + \Delta x$$

$$y_{i+1} = y_i + \frac{y_2 - y_1}{x_2 - x_1} \Delta x$$



DDA uses repeated addition

We need only compute m once, as the start of the scan-conversion.

The DDA algorithm runs rather slowly because it requires **real arithmetic** (floating-point operations).

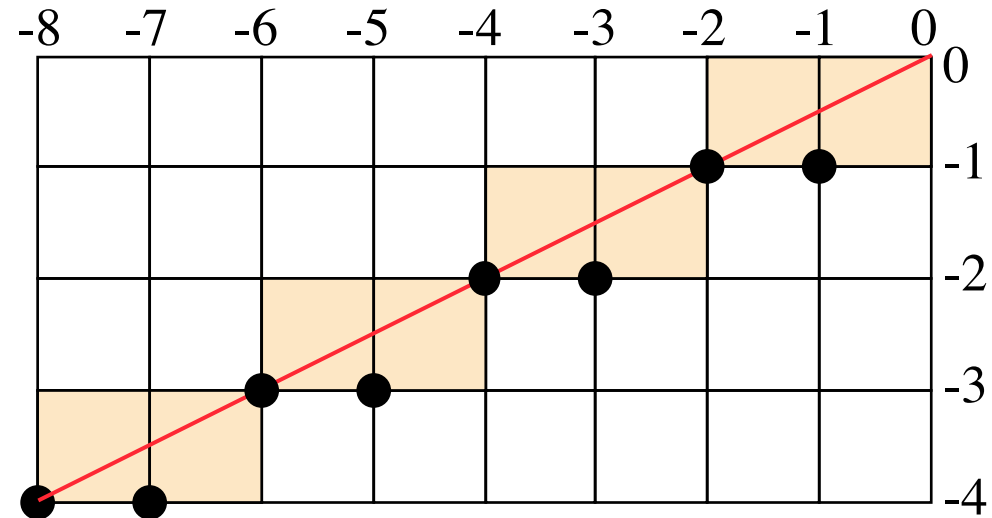
Line Drawing Algorithms

DDA algorithm for lines with $-1 < m < 1$

Example: Third quadrant

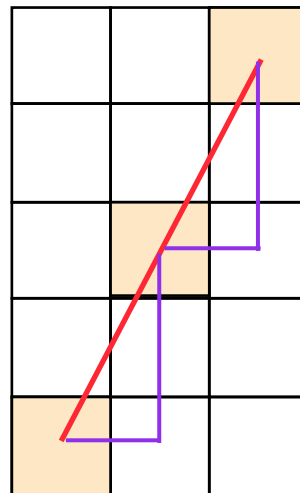
```

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

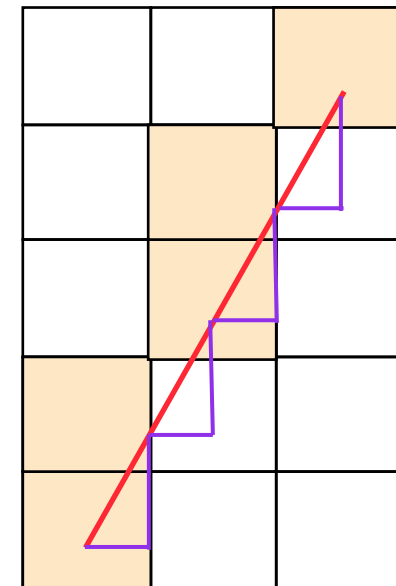


Switching the roles of x and y when $m > 1$

Gaps occur when $m > 1$



Reverse the roles of x and y using a unit step in y , and $1/m$ for x .

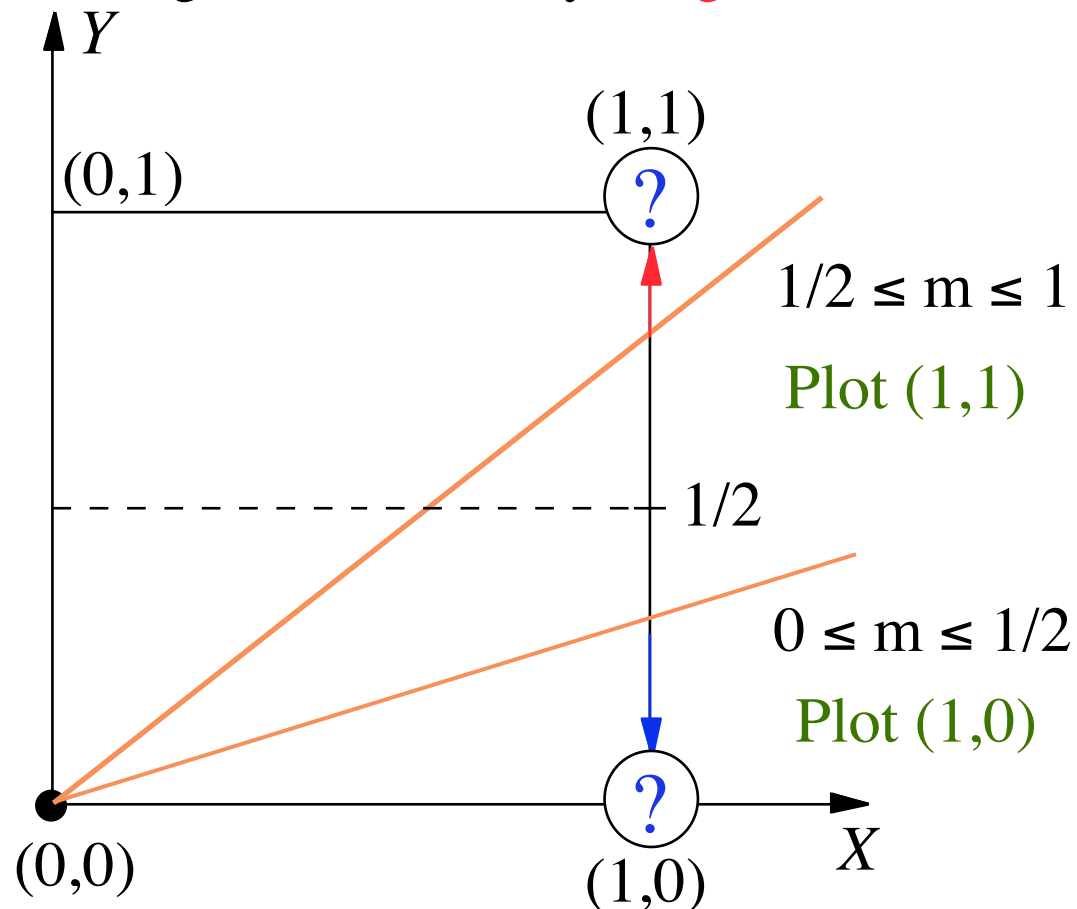


Line Drawing Algorithms

Algorithm 3: Bresenham's algorithm (1965)

Bresenham, J.E. *Algorithm for computer control of a digital plotter*, IBM Systems Journal, January 1965, pp. 25-30.

This algorithm uses only **integer arithmetic**, and runs significantly faster.



Key idea: distance between the actual line and the nearest grid locations (**error**).

Initialize error:

$$e = -1/2$$

Error is given by:

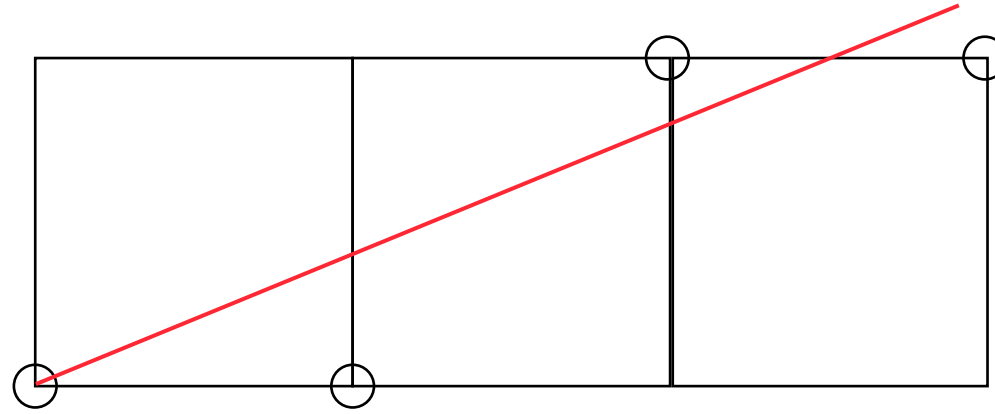
$$e = e + m$$

Reinitialize error:

when $e > 0$

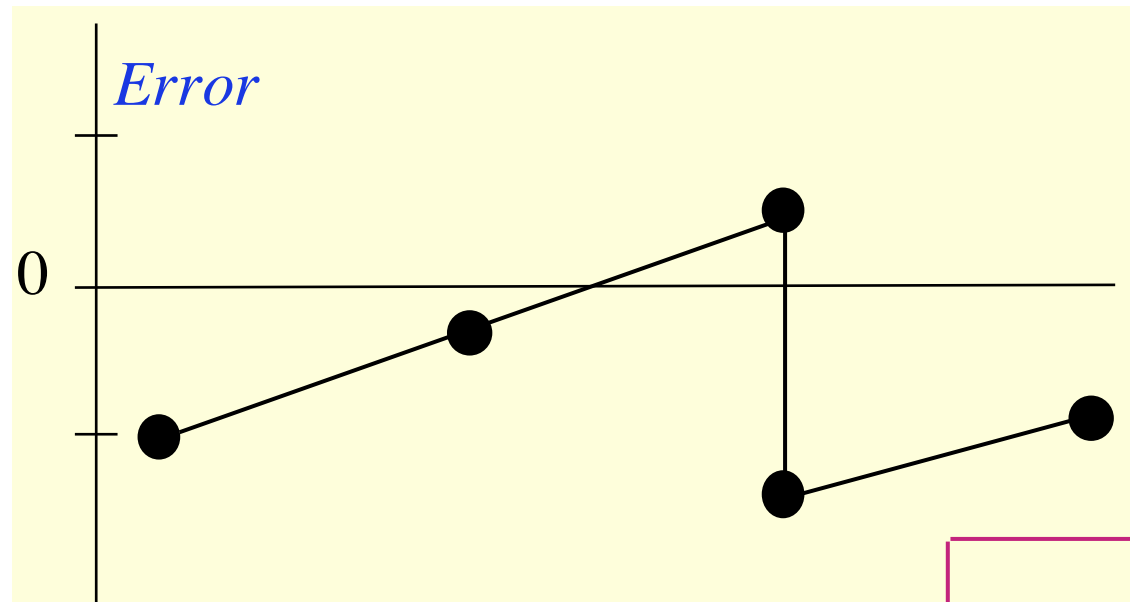
Line Drawing Algorithms

Example: $m=3/8$



If $e < 0$ below
else above

below below above above



Reinitialize
error:
 $e = 1/4 - 1$
 $= -3/4$

Error: $e=e+m$

Initial value: $e = -1/2$	$e = -1/2 + 3/8$ $= -1/8$	$e = -1/8 + 3/8$ $= 1/4$	$e = -3/4 + 3/8$ $= -3/8$
------------------------------	------------------------------	-----------------------------	------------------------------

Line Drawing Algorithms

However, this algorithm **does not lead to integer arithmetic**. Scaling by: $2 * dx$

```
void Bresenham (int xl, int yl, int xr, int yr)
{
    int x,y;                /* coordinates of pixel being drawn */
    int dy, dx;
    int ne;                 /* integer scaled error term */
    x = xl; y = yl;        /* start at left endpoint */
    ie = 2 * dy - dx;      /* initialize the error term */
    while (x <= xr){      /* pixel-drawing loop */
        PlotPixel (x,y);  /* draw the pixel */
        if (ie > 0) {
            y = y + 1;
            ne = ne - 2 * dx; /* replaces e = e - 1 */
        }
        x = x + 1;
        ne = ne + 2 * dy;    /* replaces e = e + m */
    }
}
```

Line Drawing Algorithms

Bresenham's algorithm also applies for **circles**.

Bresenham, J.E. *A linear algorithm for incremental digital display of circular arcs*
Communications of the ACM, Vol. 20, pp. 100-106, 1977.

Key idea: compute
the initial octant only

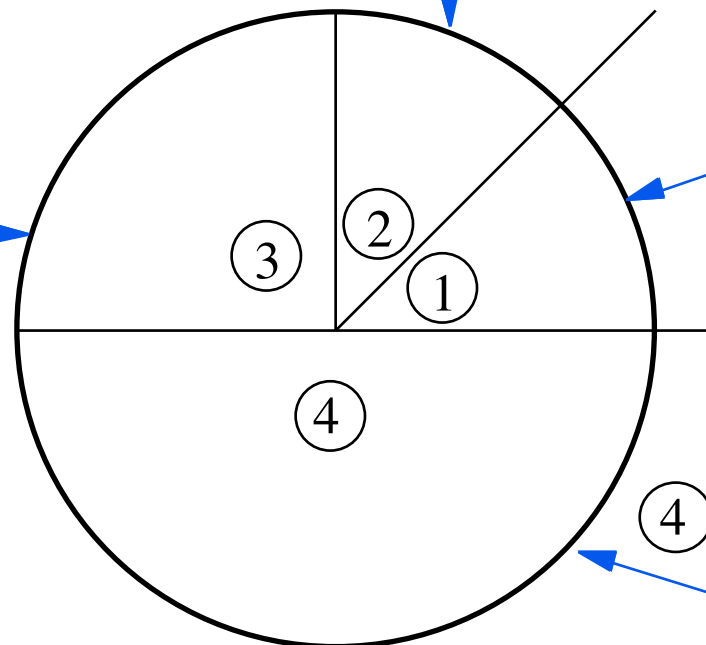
$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

② Reflect first octant about $y=x$

③ Reflect first
quadrant
about $x=0$

$$\begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}$$

① Generate first
octant



④ Reflect upper
semicircle
about $y=0$

$$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

Line Drawing Algorithms

Bresenham's incremental circle algorithm.

Example:

circle of radius 8

Bright pixels:

initial pixel → (0,8)

(1,8)

(2,8)

(3,7)

(4,7)

(5,6)

(6,5)

(7,4)

(7,3)

(8,2)

(8,1)

end pixel → (8,0)

