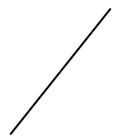
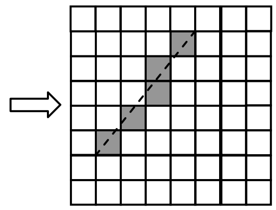
Instructor Lecture Notes: Rasterizing Lines

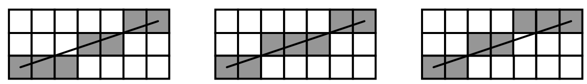
<u>“Rasterize” = “Scan convert”</u>





Draw this line ↑ On this raster ↑

Ingeneral: which pixels should be turned on? Example with 3 possibilities shown:



A. <u>Naïve approach</u>.

To draw a line use a line equation: $y_i = mx_i + b$ (*m* is the slope and *b* is the *y*-intercept). For discussion we’ll consider only the first octant. i.e., where the slope is “shallow”:

```
// Draw a line in the first octant (0 < slope < 1)
// from (x1, y1) to (x2, y2):
m = (y2-y1)/(x2-x1)    // “rise” over “run”
b = y2 - m*x2          // Use the slope just computed, and the
                       // 2nd endpoint in the line equation to
                       // get the y-intercept, b.
// Use line equation to go from 1st endpoint to 2nd ...
for( x = x1; x ≤ x2; x++ )        ← Note that we increment x by 1 each time
  y = m*x + b
  plot( x, floor(y + 0.5) )       ← Choose the pixel closest to the actual line
```

B. <u>Improvement</u>. (Floating point multiply is inefficient)

$$y_i = mx_i + b$$
$$y_{i+1} = mx_{i+1} + b$$
$$y_{i+1} = m(x_i + \Delta x) + b$$
$$y_{i+1} = mx_i + b + m\Delta x$$
$$y_{i+1} = y_i + m\Delta x$$
$$y_{i+1} = y_i + m$$

← Next *y* value is obtained by *adding*.

C. Pseudo-code.

```
// Draw a line in the first octant (0 < slope < 1)
// from (x1, y1) to (x2, y2):

compute dy
compute dx
plot(x1, y1)
compute m
y = y1
while( x1 != x2 )
  increment x1 by 1
  increment y by m
  plot( x1, floor(y + 0.5) )
```

No more multiplies in the loop: this is an "incremental algorithm" known as a "DDA" - digital differential analyzer (a differential analyzer is an analog mechanical device that solves differential equations by integration).

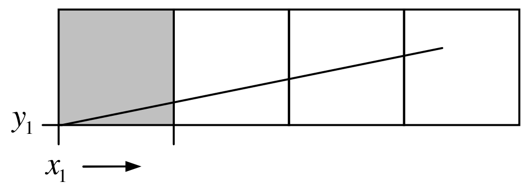
```
// DDA(x1, y1, x2, y2), all octants:

compute dy
compute dx
plot(x1, y1)

if -1 < slope < +1   // Shallow slope case.
  compute m          // Calculate in the usual way: dy/dx.

  // 1st endpoint could be on the left or the right ...
  // should we increment x (+1) or decrement x (-1)?
  dx < 0 ⇒ dx = -1, otherwise dx = +1

  m *= dx  // Same m used for pos and neg slopes.

  y = y1   // Start at 1st endpoint's scanline.

  // Go from the 1st endpoint to the 2nd.
  while( x1 != x2 )
    increment x1 by dx
    increment y by m
    plot( x1, floor(y + 0.5) )

else // Steep slope case.
  // same algorithm as above, except
  // switch x's and y's, dx's and dy's
  plot(floor(x + 0.5), y1 )
```

<u>Bresenham's Algorithm</u> (1962, Jack Bresenham, IBM, published in 1965)

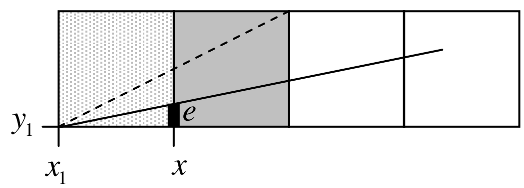
An incremental algorithm that uses only integer or bitwise operations … very fast.

Assuming the 1st octant $\Rightarrow 0 \le m \le 1$, rasterize the line from (x_1, y_1) to (x_2, y_2).

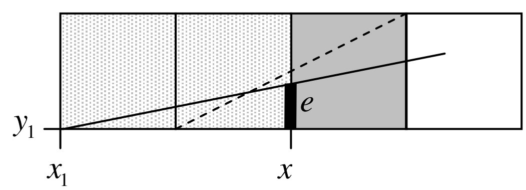
Start at $x = x_1$, increment x by 1 each time.



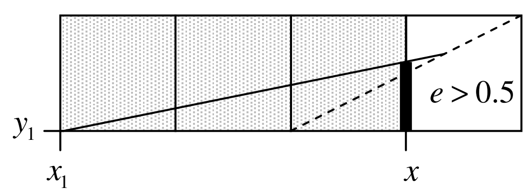
The question is: when do we increment y to go to the next scanline? Bresenham's approach: keep track of an error value, e, the distance from the current y value to the *exact* y value of the line at the current x.



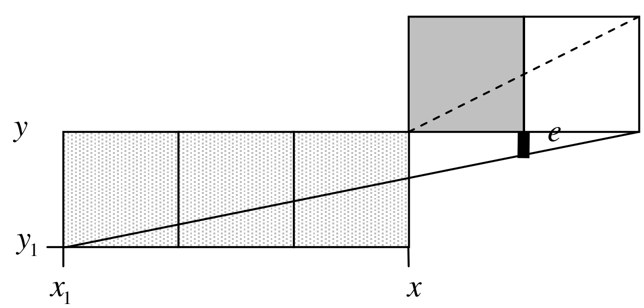
Each time we increment x, we increment e: with every incremental change in x the value of e changes by the slope of the line:



When the error value reaches or exceeds ½, the y value of the *next* scanline is closer to that of the actual line ...



... so we move to the next scanline by incrementing y, and adjust the error term to be relative to this new scanline by subtracting 1:



Pseudo-code:

```
Bresenham(x1, y1, x2, y2) // 1st octant only
 1. dx = |x2 - x1|
 2. dy = |y2 - y1|
 3. e = 0
 4. de = dy/dx              // <-- non-integer!
 5. y = y1
 6. for( x from x1 to x2 )
 7.   plot(x,y)
 8.   e = e + de
 9.   if( e ≥ 0.5 )         // <-- non-integer!
10.     y++
11.     e = e - 1.0         // <-- non-integer!
```

But this uses floating point math. How do we convert this to an integer algorithm?

(1) Multiply all floating point values by *dx*:

```
Bresenham(x1, y1, x2, y2) // 1st octant only
 1. dx = |x2 - x1|
 2. dy = |y2 - y1|
 3. e = 0
 4. de = dy                 // dy/dx ⇒ dy
 5. y = y1
 6. for( x from x1 to x2 )
 7.   plot(x,y)
 8.   e = e + de
 9.   if( e ≥ 0.5*dx )      // 0.5 ⇒ 0.5 * dx
10.     y++
11.     e = e - dx          // 1.0 ⇒ 1.0 * dx
```

(2) Remove the fraction in line 9:

```
Bresenham(x1, y1, x2, y2) // 1st octant only
 1. dx = |x2 - x1|
 2. dy = |y2 - y1|
 3. e = 0
 4. de = dy
 5. y = y1
 6. for( x from x1 to x2 )
 7.   plot(x,y)
 8.   e = e + de
 9.   if( 2e ≥ dx )         // Multiply by 2
10.     y++
11.     e = e - dx
```

This is now an ***integer*** algorithm.

Swapping x and y (this is a reflection about $y = x$) when $m > 1$ will converts the above shallow slope case to the steep slope case .

```
Bresenham(x1, y1, x2, y2) // 1st and second octants only

 1. steep = |y2 - y1| > |x2 - x1|
 2. if( steep )
 3.   swap(X1, Y1)
 4.   swap(X2, Y2)

 5. dx = |x2 - x1|
 6. dy = |y2 - y1|

 7.  e = 0
 8. de = dy
 9.  y = y1

10. for( x from x1 to x2 )
11.   if( steep ) plot(y,x) else plot(x,y)
12.   e = e + de
13.   if( 2e ≥ dx )
14.     y++
15.     e = e - dx
```

Remaining octants: reflect about $y = x$, swap endpoints, and/or decrement y, as needed.

```
Bresenham(x1, y1, x2, y2) // All octants

 1. steep = |y2 - y1| > |x2 - x1|
 2. if( steep )       // Reflecting about y = x switches
 3.   swap(x1, y1)    // x's and y's to convert steep
 4.   swap(x2, y2)    // slope to shallow slope case.

4a. if( x1 > x2 )     // Swap endpoints so algorithm only
4b.   swap(x1, x2)    // has to deal with going from left
4c.   swap(y1, y2)    // to right in x

 5. dx = x2 - x1      // Line 4b ⇒ abs value not needed
 6. dy = |y2 - y1|    // (saves one operation)

 7.  e = 0            // Intialize error term, error term
 8. de = dy           // increment, and initial scanline.
 9.  y = y1           // ystep accounts for + or - slopes
9a. if( y1 < y2 ) ystep = +1 else ystep = -1

10. for( x from x1 to x2 )
11.   if( steep ) plot(y,x) else plot(x,y)
12.   e = e + de
13.   if( 2e ≥ dx )
14.     y += ystep
15.     e = e - dx
```