



CS335 Fall 2007
Graphics and Multimedia



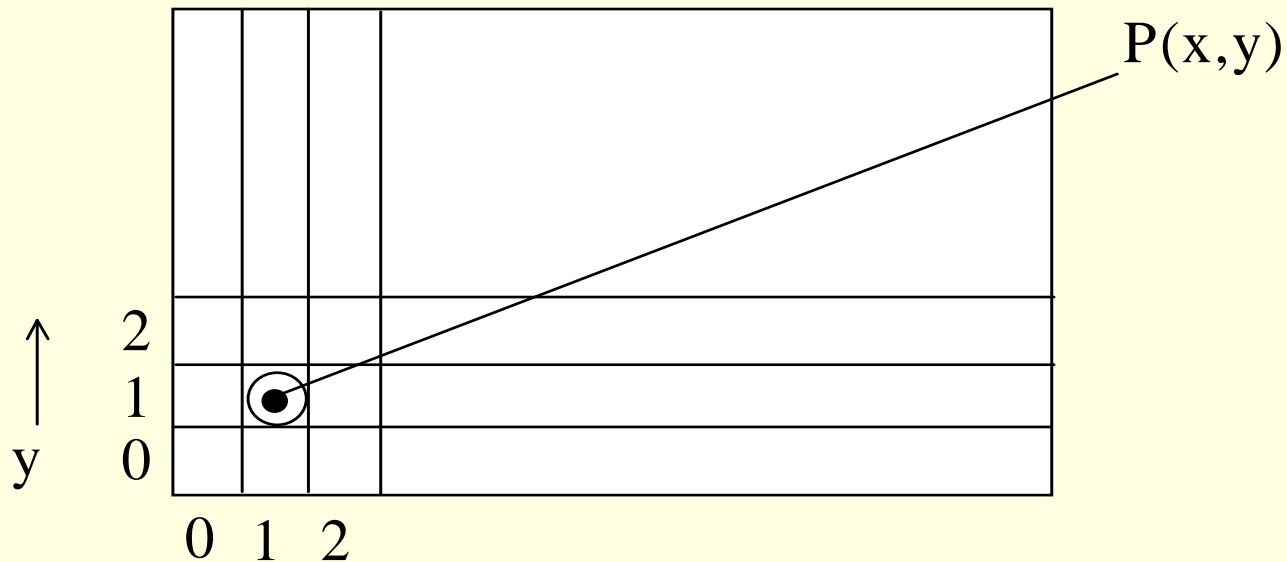
2D Drawings: Lines

Primitive Drawing Operations

Digital Concepts of Drawing in Raster Arrays

PIXEL is a single array element at (x,y)

- No smaller drawing unit exists




Scan Conversion

Pixel Value = 1 bit, 8 bit, 24 bit,

= address into hardware look up table to determine display color, intensity.

Pixel based Geometry

Points, lines, circles, conics, curves, splines, polygons, shaded polygons, text fonts & icons.

 all basically reduce to scan conversion problem:

FIND Digital algorithms for continuous geometric concepts

Drawing Lines

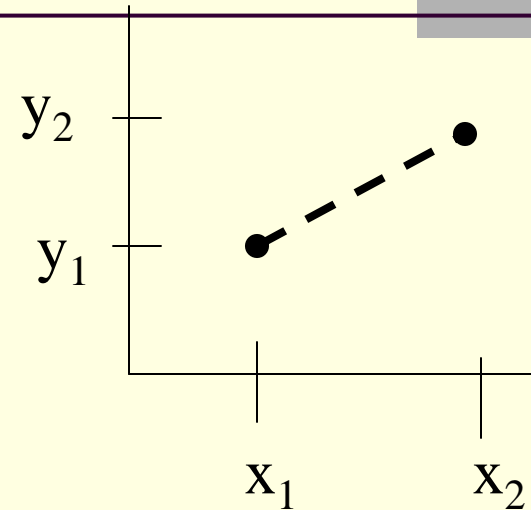
From (x_1, y_1) to (x_2, y_2)

Equation:

$$y = mx + b$$

$$m = (y_2 - y_1) / (x_2 - x_1)$$

$$b = y_1 - m * x_1$$



```
compute -- m, b
for x=x1 to x2, step 1
    y = m*x + b
    pixel( x, round(y), color)
loop
```

DDA Algorithm

(Digital Differential Analyzer)

[DDA avoids the multiple by slope m]

Equation:

$$x_{k+1} = x_k + 1$$

$$y_{k+1} = y_k + m$$

if slope $|m| > 1$

$$y_{k+1} = y_k + 1$$

$$x_{k+1} = x_k + 1/m$$

```
compute m (assume |m| < 1 )
```

```
y = y1, x = x1
```

```
pixel(round(x1), round(y1), color)
```

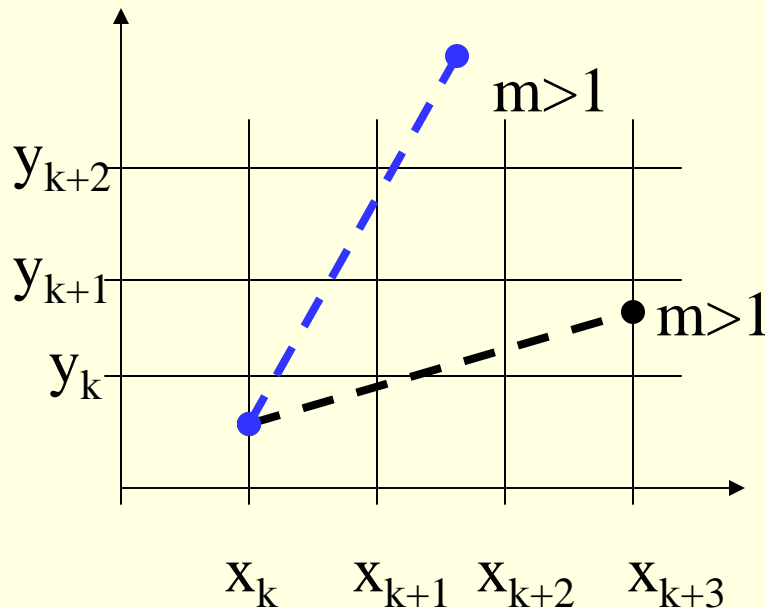
```
for x=x1 to x2, step 1
```

```
  x = x + 1
```

```
  y = y + m
```

```
  pixel(x, round(y), color)
```

```
end
```



DDA Algorithm

(Digital Differential Analyzer)

[avoid the float multiple by slope m]

slope m , y – floats

Problems:

1. Necessary to perform float addition
2. Necessary to have float representation
3. Necessary to perform -- *round()*

Float representations/operations are more expensive than integer operations. We would like to avoid them if possible.

Drawing a Fast Line

The Midpoint Algorithm (Bresenham's Algorithm)

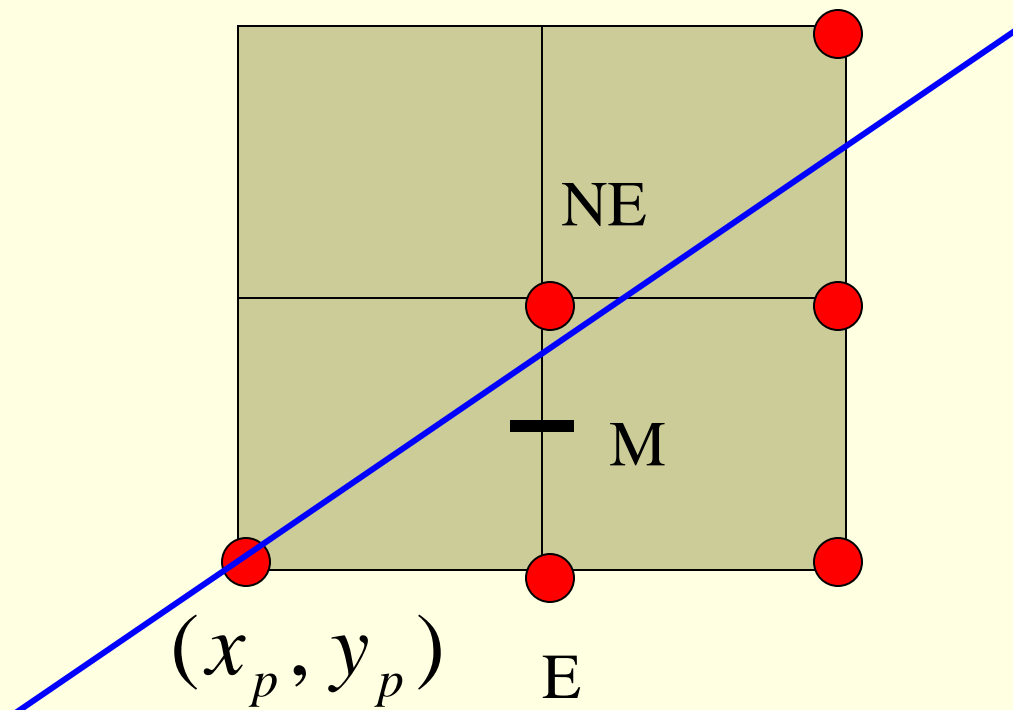
- Scan-converts lines
- Method is **incremental**
- Uses only integer arithmetic

Assume slope of line to be drawn is $0 \leq m \leq 1$

Lower-left endpoint: (x_0, y_0)

Upper-right endpoint: (x_1, y_1)

The Midpoint Algorithm

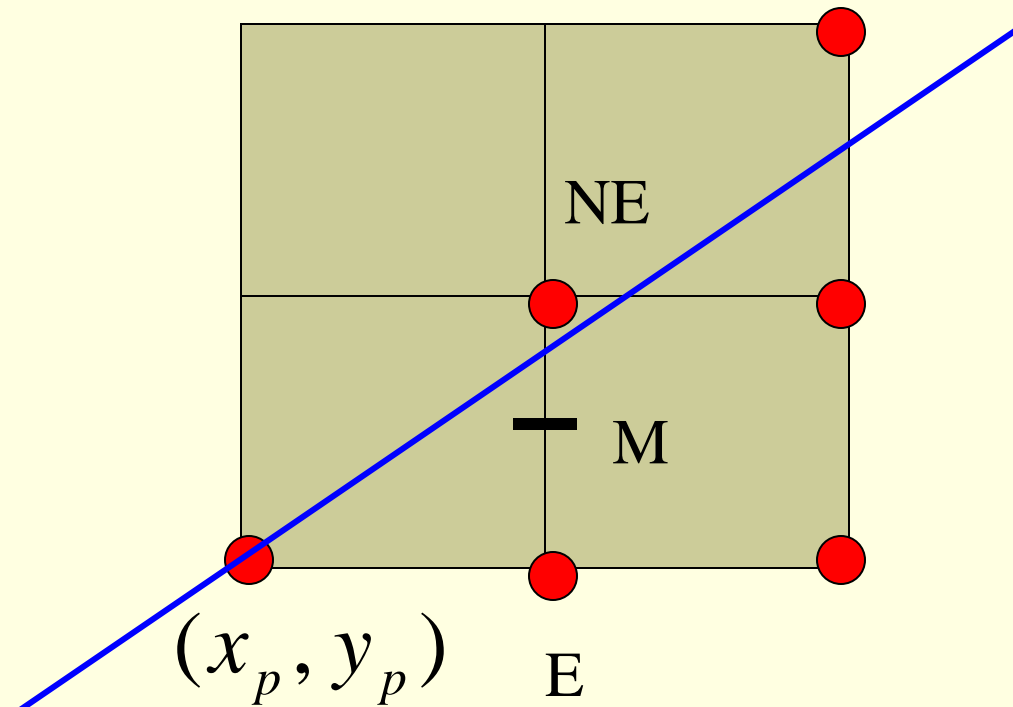


Key idea: at each step, decide which pixel (E or NE) is closest to the line. Choose that pixel and draw it.

The Implicit Line Equation

If M lies above the line, draw pixel E

If M lies below the line, draw pixel NE



Implicit Lines

$$F(x, y) = ax + by + c = 0$$

$$\text{Let } \left. \begin{array}{l} dy = y_1 - y_0 \\ dx = x_1 - x_0 \end{array} \right\} m = \frac{dy}{dx}$$

The slope-intercept form gives

$$y = \frac{dy}{dx} x + B$$

Implicit Lines

Now we can convert the slope-intercept form to the implicit form via a few substitutions and manipulations:

$$dx \ y = dy \ x + dx \ B$$

$$0 = dy \ x - dx \ y + dx \ B$$

Which is in the form $0 = ax + by + c$ by letting

$$a = dy$$

$$b = -dx$$

$$c = Bdx$$

Implicit Lines

Now we can verify an important property of this representation:

$F(x, y) = 0$ for points on the line

$F(x, y) > 0$ for points below the line

$F(x, y) < 0$ for points above the line

Example:

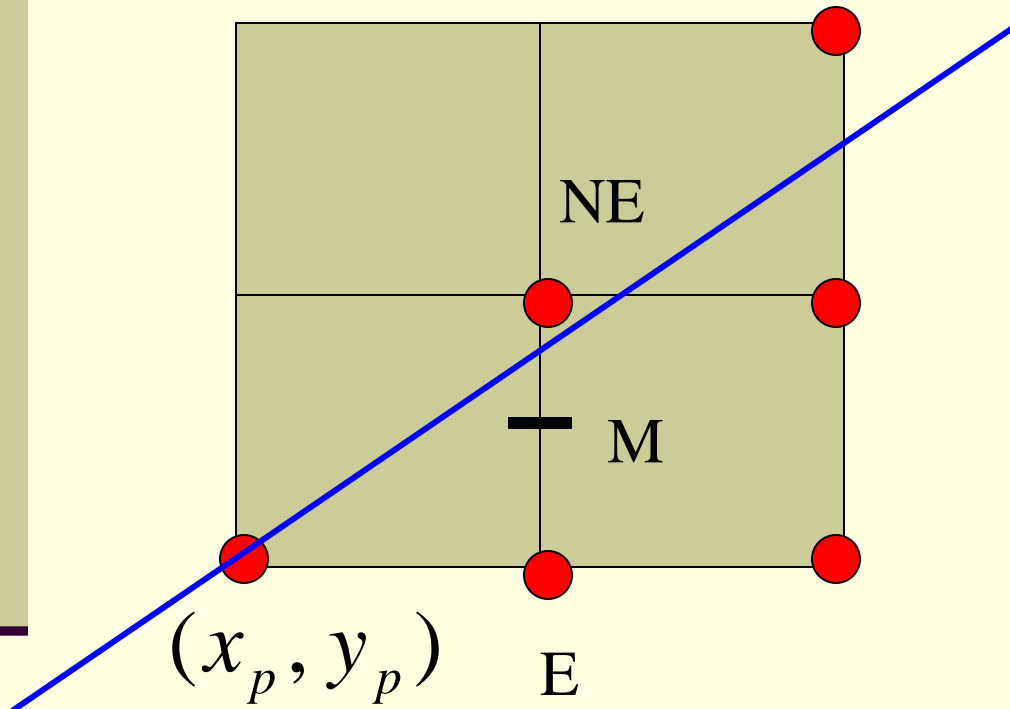
Implicit Lines

Conclusion: the sign of $F(x,y)$ reveals the location of (x,y) with respect to the line represented by F

Idea: Check the midpoint at each iteration to determine which pixel, E or NE, is closest to the line.

Use the implicit equation of the line F to check

The Midpoint Test



$$M = (x_p + 1, y_p + \frac{1}{2})$$

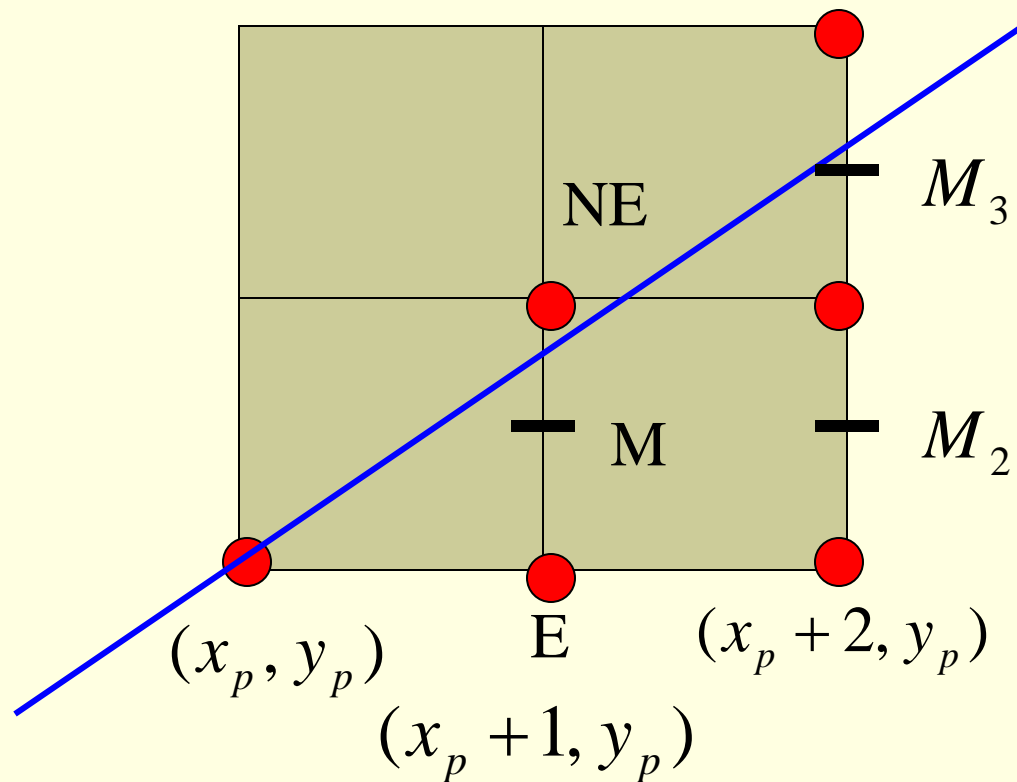
The sign of $F(M)$ gives the answer as to which pixel, E or NE, to draw

Formulating an Algorithm

Let d be a decision variable which makes the midpoint test.
Then the test to decide which pixel to draw is just

```
if (d > 0)
then
    // the midpoint is below the line
    // the line passes closer to the upper pixel
    draw the NE pixel
else
    // the midpoint is above the line
    // the line passes closer to the lower pixel
    draw the E pixel
```

Making the Algorithm Incremental



$$M_1 = (x_p + 1, y_p + \frac{1}{2})$$

$$M_2 = (x_p + 2, y_p + \frac{1}{2})$$

$$M_3 = (x_p + 2, y_p + \frac{3}{2})$$

Incremental Formulation

$$\begin{aligned}d_{first} &= F(M_1) = F(x_p + 1, y_p + \frac{1}{2}) \\ &= a(x_p + 1) + b(y_p + \frac{1}{2}) + c\end{aligned}$$

If we choose E after this computation, then we must compute $F(M_2)$ next.

What is the relationship between

$$d_{first} = F(M_1) \text{ and } F(M_2)$$

Incremental Formulation

Notice that

$$d_{first} = a(x_p + 1) + b(y_p + \frac{1}{2}) + c$$

$$d_{next} = a(x_p + 2) + b(y_p + \frac{1}{2}) + c$$

(since $d_{next} = F(M_2)$) so that we can compute

$$d_{next} = d_{first} + a$$

which is just

$$d_{next} = d_{first} + dy$$

Choosing E vs. NE

When we compute $F(M_1)$ and draw the E pixel, $F(M_2)$ can be obtained from a **single addition operation** via the equation

$$F(M_2) = F(M_1) + dy$$

What if we are drawing the NE pixel instead of the E pixel?

$$d_{first} = a(x_p + 1) + b(y_p + \frac{1}{2}) + c$$

$$d_{next} = a(x_p + 2) + b(y_p + \frac{3}{2}) + c$$

which gives this incremental computation of $F(M_3)$

$$d_{next} = d_{first} + (dy - dx)$$

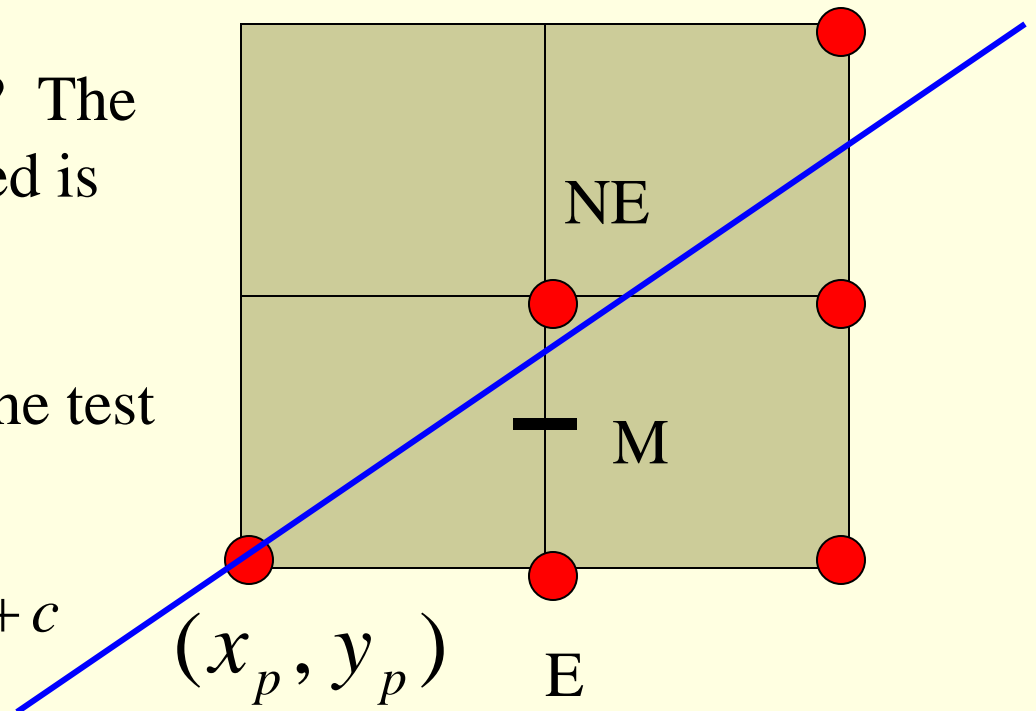
A Starting Point

What about a starting point? The first midpoint to be computed is

$$F(x_0 + 1, y_0 + \frac{1}{2})$$

Thus the starting value for the test variable is

$$d_{start} = a(x_0 + 1) + b(y_0 + \frac{1}{2}) + c$$



How can we simplify this?

Efficiencies

$$\begin{aligned}d_{start} &= a(x_0 + 1) + b(y_0 + \frac{1}{2}) + c \\ &= ax_0 + by_0 + c + a + \frac{b}{2} \\ &= F(x_0, y_0) + a + \frac{b}{2}\end{aligned}$$

Now observe that since the point $F(x_0, y_0)$ is on the line:

$$F(M_{start}) = a + \frac{b}{2} = dy - dx / 2$$

Integer Arithmetic

We can maintain and test the value of $2F(M)$ to make it integral

$$d_{start} = 2F(M_{start}) = 2dy - dx$$

If we move E, the increment to maintain the midpoint is multiplied by 2:

$$d_{next} = d_{first} + 2dy$$

If we move NE, the increment to maintain the midpoint is multiplied by 2:

$$d_{next} = d_{first} + 2(dy - dx)$$

The Algorithm

```
MidpointLine (x0, y0, x1, y1)

// assumes slope is between 0 and 1
// and that (x0, y0) is leftmost point
dx = x1-x0;
dy = y1-y0;
d = 2dy-dx;
incrE = 2dy;
incrNE = 2(dy-dx);
x = x0;
y = y0;
WritePixel(x,y);
```

Algorithm (Con't)

```
while (x < x1) do
    if (d <= 0)
        d = d + incrE
        x = x + 1
    else
        d = d + incrNE
        x = x + 1
        y = y + 1
        WritePixel (x, y)
    end
end
end
```

Summary of Concepts

- Implicit representation of a line
- Incremental algorithm for more efficient computation
- Integer algorithm by introducing harmless scale factor
- Logic must be added to handle all line orientations