Image Formation

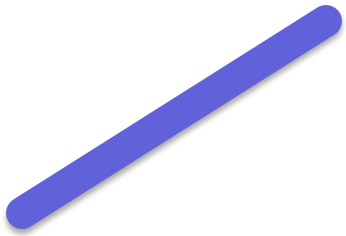
CS418 Computer Graphics

John C. Hart

Vector v. Raster Graphics

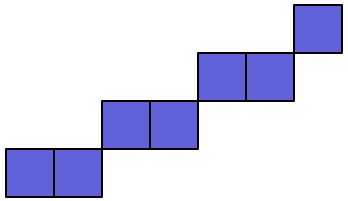
Vector Graphics

- Plotters, laser displays
- “Clip art,” illustrations
- PostScript, PDF, SVG
- Low memory (display list)
- Easy to draw line
- Solid/gradient/texture fills



Raster Graphics

- TV’s, monitors, phones
- Photographs
- GIF, JPG, etc.
- High memory (frame buffer)
- Hard to draw line
- Arbitrary fills



Raster Images

- (Spatial) Resolution
  - horizontal pixels x vertical pixels
- Image Aspect Ratio
  - width/height
  - HDTV = 1920/1080 = 1.78 = 16:9
- Pixel Aspect Ratio
  - (H/V) / (height/width) = (H/V) x (1/A)
  - Square pixels are 1:1
- Color resolution
  - Bits per pixel
  - 24 bpp = 8 bits red, green and blue
  - 8 bpp = 3 bits red, green, 2 bits blue

Interlaced v. Progressive Scan

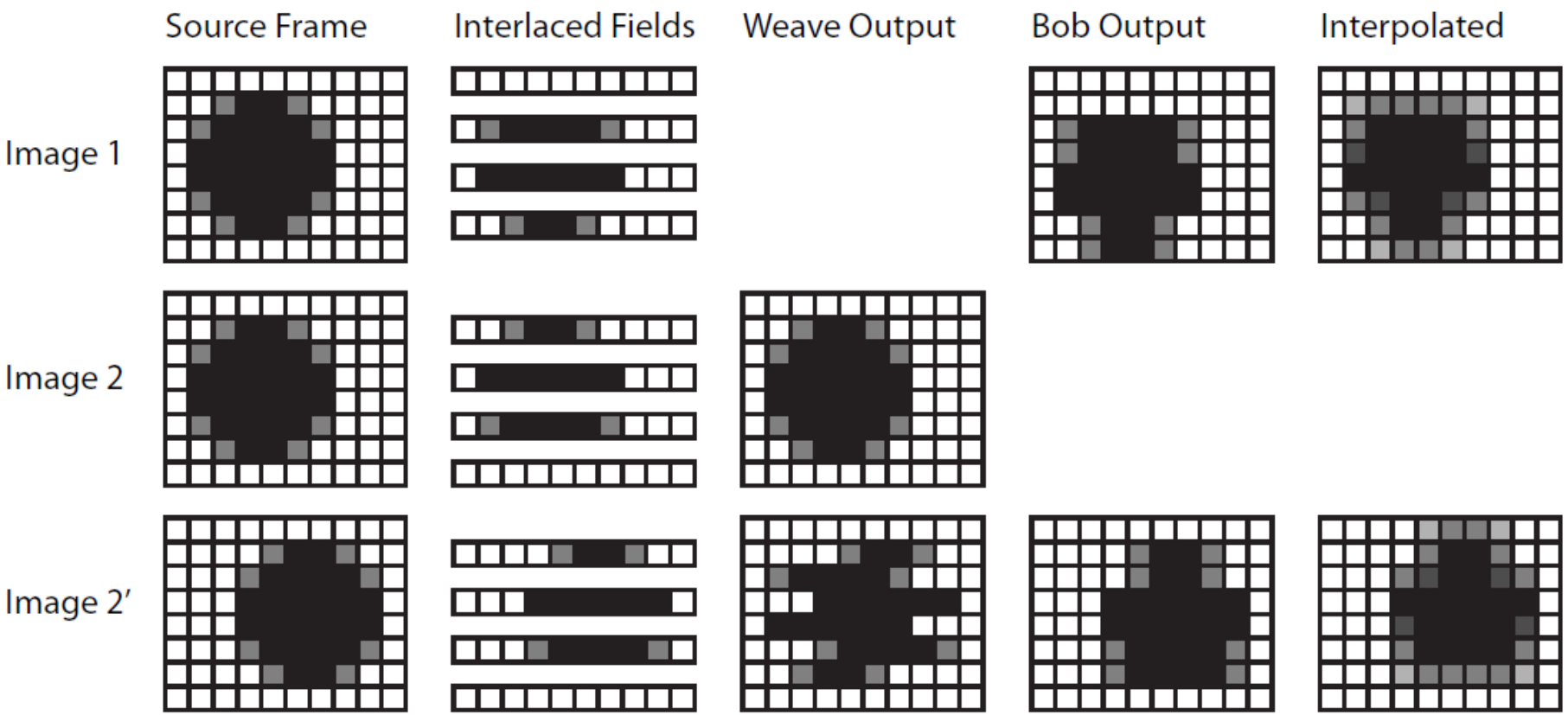


Image File Formats

Format	Fidelity	Complexity	Use
BMP	Uncompressed	**Simple**, Microsoft	Easy to process, windows icons
PPM	Uncompressed	**Simple**, open, dated	Easy to process, unix icons
GIF	Compressed Lossless (LZW)	**Only 256 colors**	Charts, graphs, diagrams, text
JPG (Exif)	Compressed **Lossy** (DCT)	Blocky, edges ring	Photographs
TIFF	Compressed Lossless (LZW)	**Flexible** but complex structure	Fax, scanning, artwork
PNG	Compressed Lossless (zlib)	**Flexible** but complex structure	Distributing images
SVG	Uncompressed	Flexible	**Vector/Line Art**

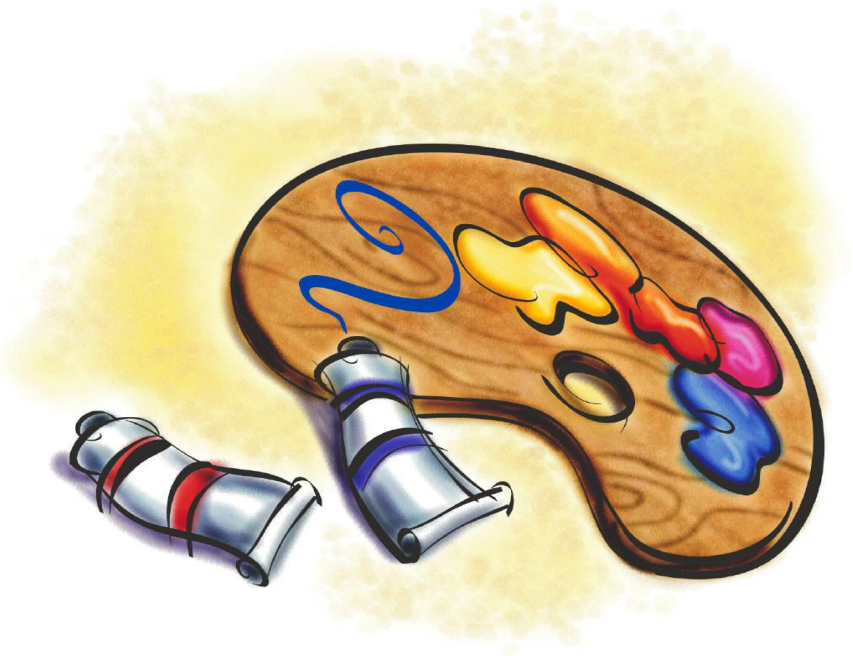
BMP Format

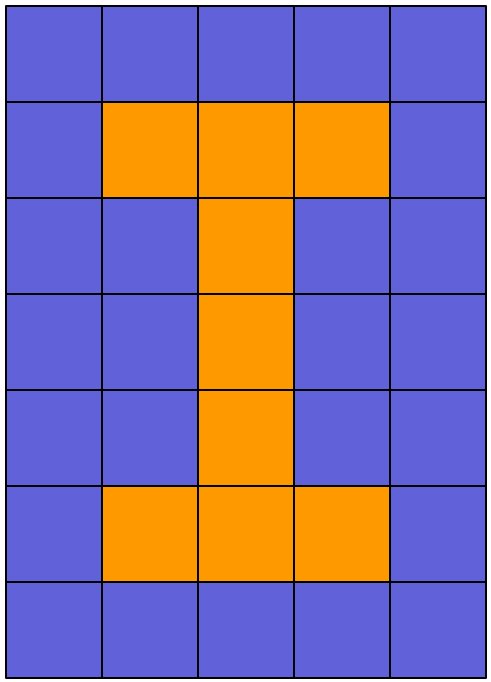
```
fput16(FILE f, short int x) {
   fputc(f,x & 255); x >>= 8; fputc(f,x & 255);
}
#define THRICE(x) x x x
fput32(FILE f, long int x) {
   THRICE(fputc(f,x & 255); x >>= 8;) fputc(f,x & 255);
}
int width, height;
int padded_row = ((24*width-1)/32 + 1)*4;
fputc(f,'B'); fputc(f,'M');
fput32(f, 54 + padded_row*height);
fput32(f, 0); fput32(f, 54); fput32(f, 40);
fput32(f, width); fput32(f, -height);
fput16(f, 1); fput16(f, 24);
fput32(f, ((24*width-1)/32 + 1)*4*height);
fput32(f, 0); fput32(f, 0); fput32(f, 0); fput32(f, 0);
```

```
for (int j = 0; j < height; j++) {
   for (int i = 0; i < width; i++) {
      fputc(red[j][i]);
      fputc(green[j][i];
      fputc(blue[j][i]);
   }
   for (int padding = 3*width;
            padding < padded_row; padding++)
      fputc(0);
   }
}
```

Color Palettes

- Store all RGB colors used in any image pixel in a table
- Store index to color in each pixel to compress data size





=

97 97 217	97 97 217	97 97 217	97 97 217	97 97 217
97 97 217	255 153 0	255 153 0	255 153 0	97 97 217
97 97 217	97 97 217	255 153 0	97 97 217	97 97 217
97 97 217	97 97 217	255 153 0	97 97 217	97 97 217
97 97 217	97 97 217	255 153 0	97 97 217	97 97 217
97 97 217	255 153 0	255 153 0	255 153 0	97 97 217
97 97 217	97 97 217	97 97 217	97 97 217	97 97 217

or

0	0	0	0	0
0	1	1	1	0
0	0	1	0	0
0	0	1	0	0
0	0	1	0	0
0	1	1	1	0
0	0	0	0	0

where:

0=

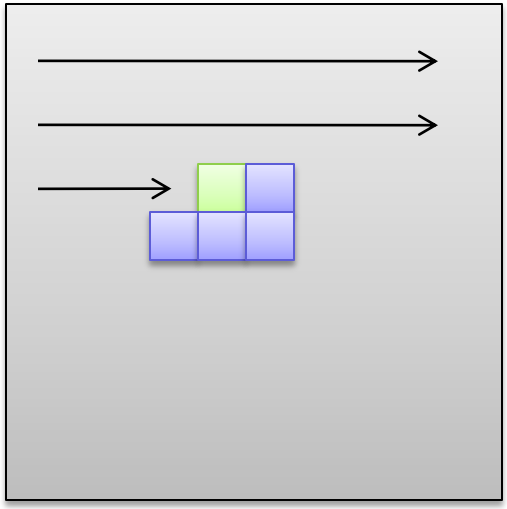
1=

Reducing Color

- How to convert a full color image into a paletted image
- “Posterize” picks color in the palette closest to full color pixel
- Dithering adds colors by combining palette colors of neighboring pixels

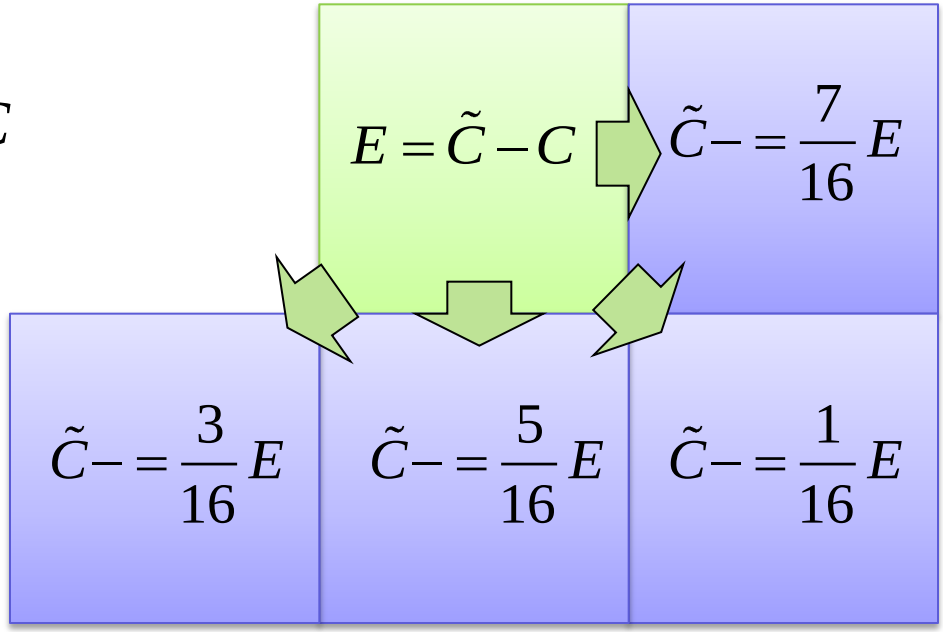
N. Damera-Venkata, B.L. Evans & V. Monga
Color Error Diffusion Halftoning,
IEEE Signal Processing 20(4), 51-58, July 2003.

Error Diffusion



Floyd-Steinberg Algorithm

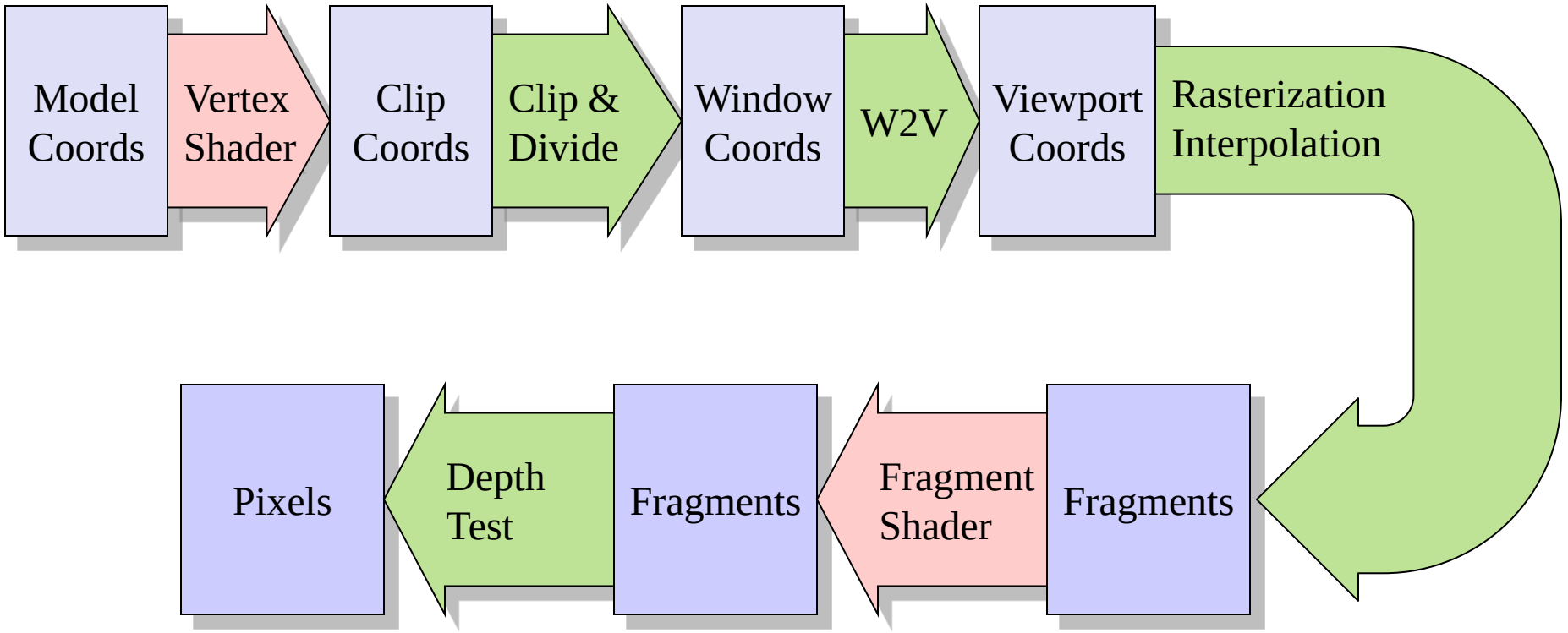
- Each pixel color C has an approximation $\tilde{C}$
- Initialize $\tilde{C}$ = approx(C) for all pixels
  - approx(C) returns closest palette color to C
- Approximation error: $E = \tilde{C} - C$
  - positive E: too bright
  - negative E: too dim
- Counteract error by changing brightness of unprocessed neighboring pixels



Dithered Images

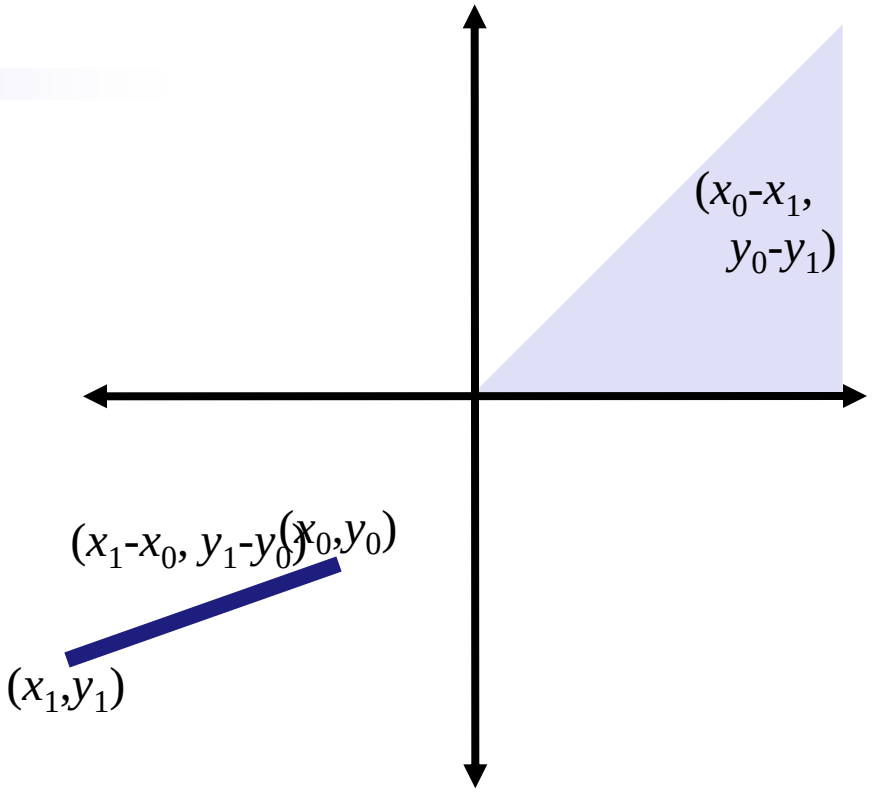
Lego Mosaics

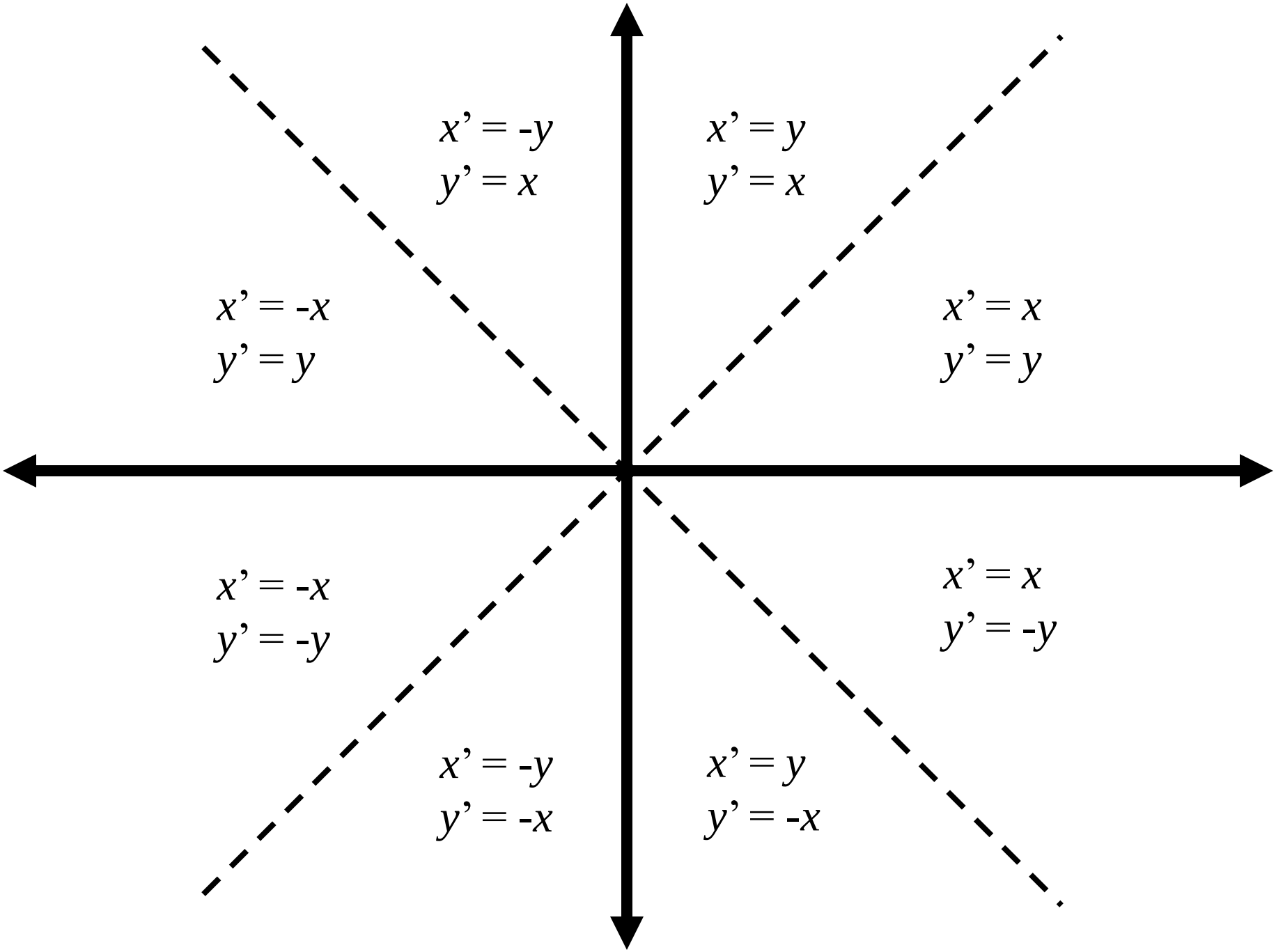
Fragment Pipeline



How to Draw a Line

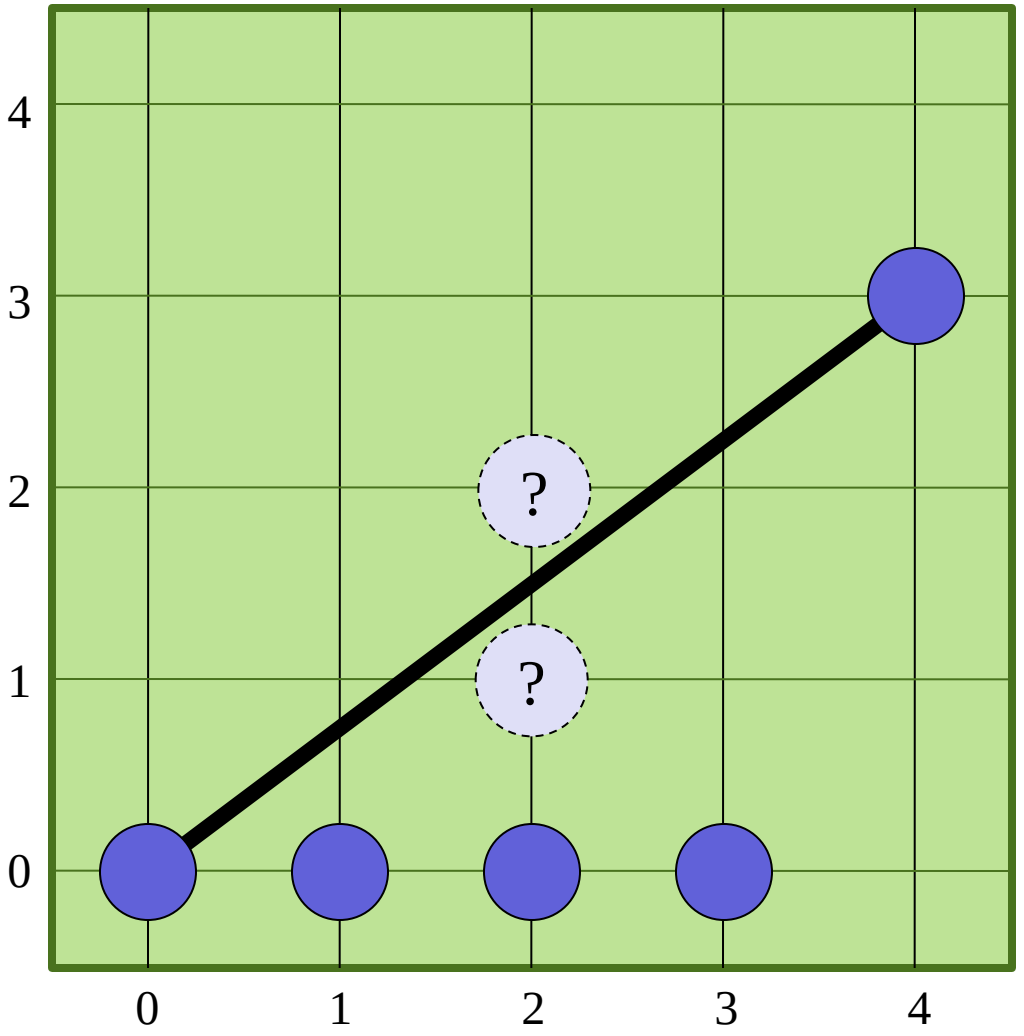
- Jack Bresenham's Algorithm
- Given a line from point (x_0,y_0) to (x_1,y_1)
- How do we know which pixels to illuminate to draw the line?
- First simplify the problem to the first octant
  - Translate start vertex to origin
  - Reflect end vertex into first octant




x’ = -y
y’ = x
x’ = y
y’ = x
x’ = -x
y’ = y
x’ = x
y’ = y
x’ = -x
y’ = -y
x’ = x
y’ = -y
x’ = -y
y’ = -x
x’ = y
y’ = -x

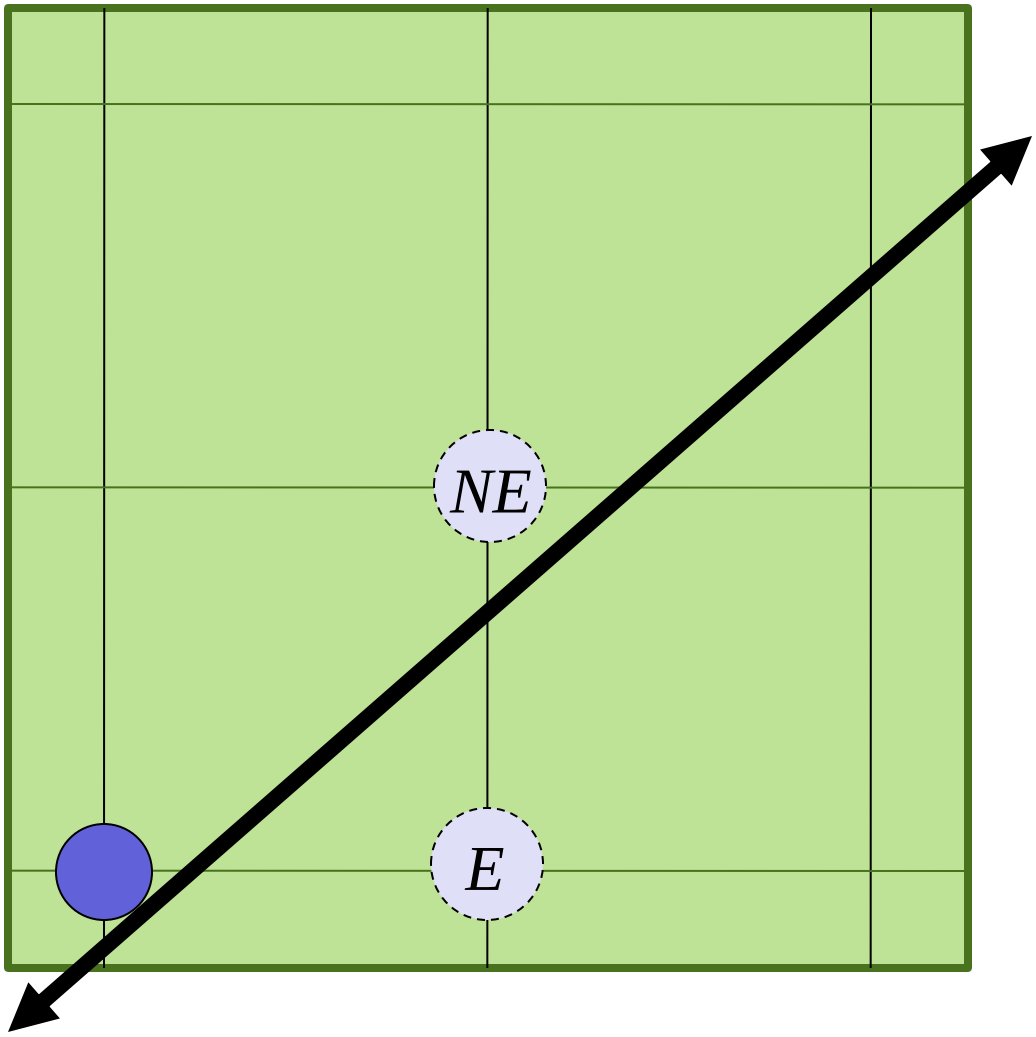
Line Rasterization

- How to rasterize a line from (0,0) to (4,3)
- Pixel (0,0) and (4,3) easy
- One pixel for each integer x-coordinate
- Pixel's y-coordinate closest to line
- If line equal distance between two pixels, pick on arbitrarily but consistently



Midpoint Algorithm

- Which pixel should be plotted next?
  - East?
  - Northeast?



Midpoint Algorithm

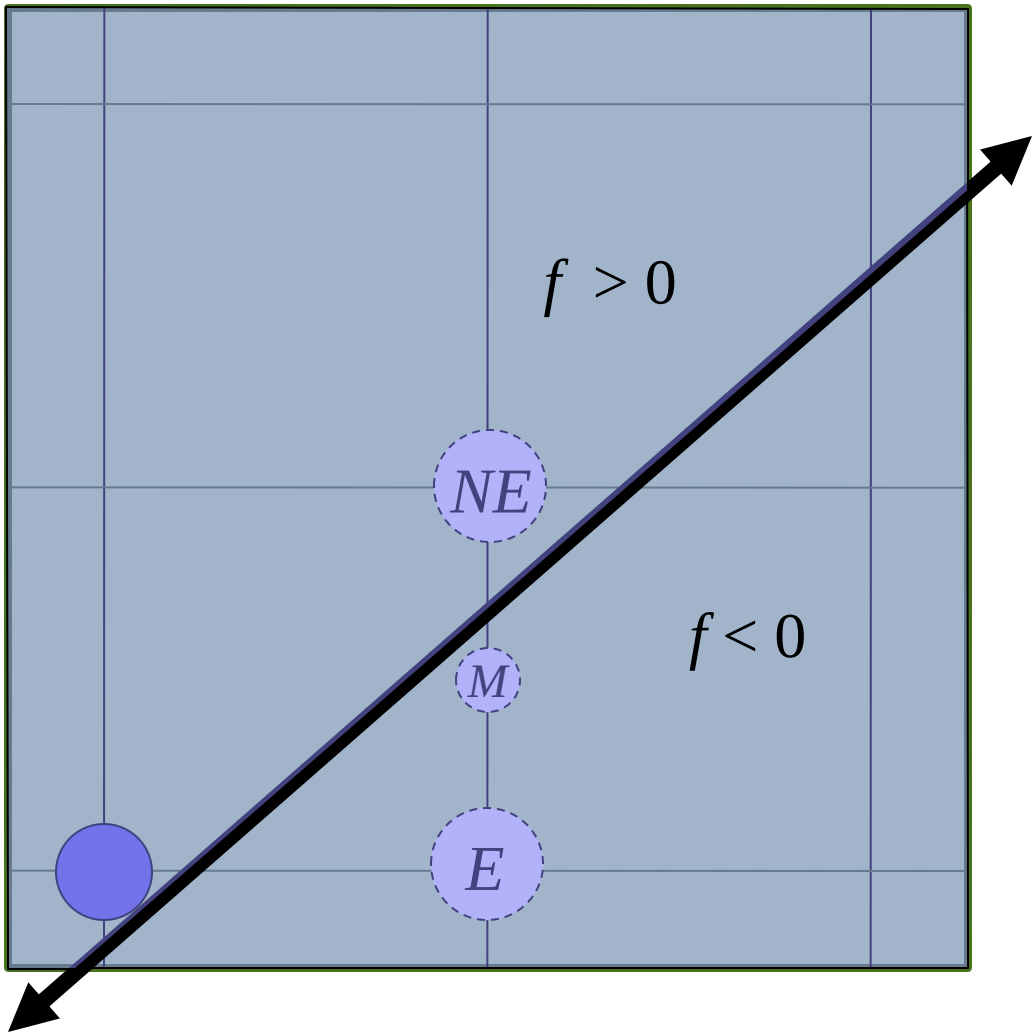
- Which pixel should be plotted next?
  - East?
  - Northeast?
- Line equation

$$y = mx + b$$

$$m = (y_1 - y_0)/(x_1 - x_0)$$

$$b = y_0 - mx_0$$

$$f(x,y) = mx + b - y$$



Midpoint Algorithm

- Which pixel should be plotted next?
  - East?
  - Northeast?
- Line equation

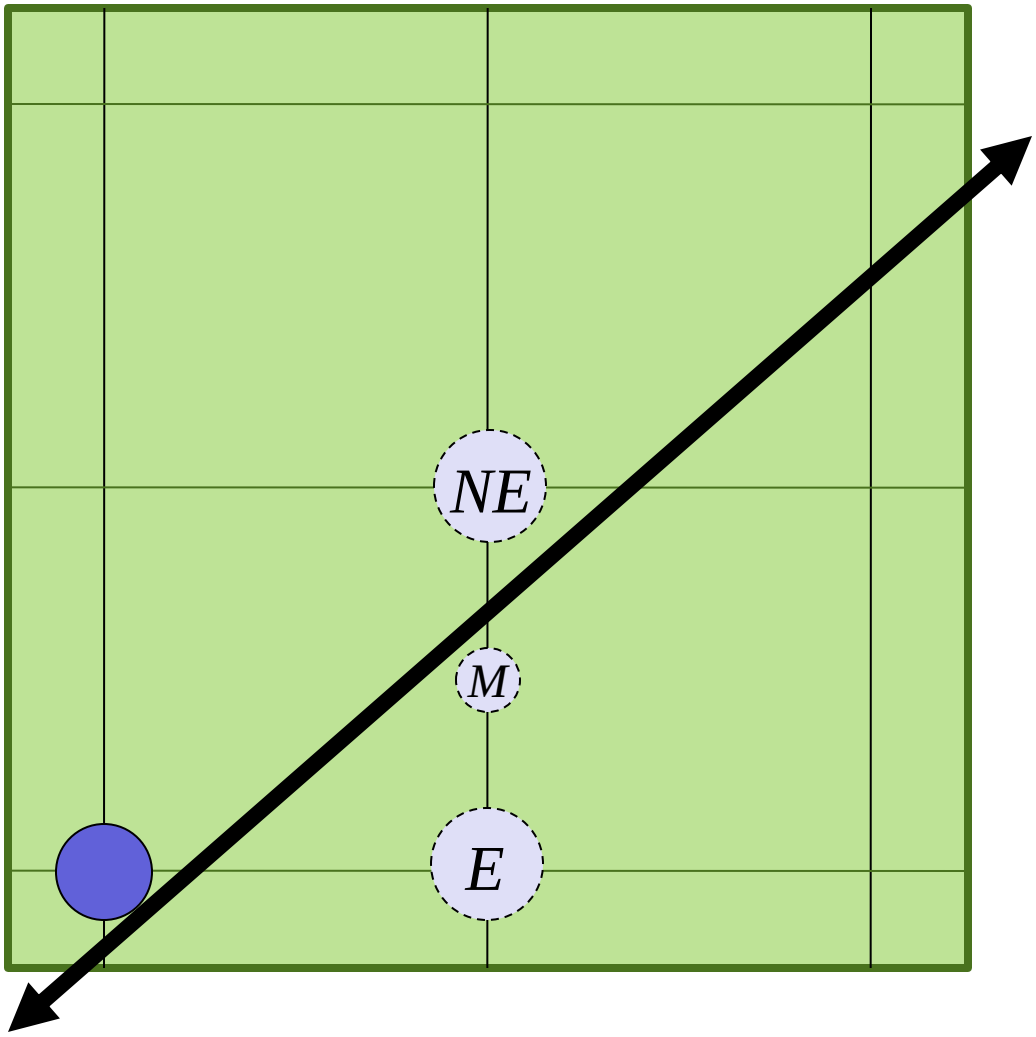
$$y = mx + b$$

$$m = (y_1 - y_0)/(x_1 - x_0)$$

$$b = y_0 - mx_0$$

$$f(x,y) = mx + b - y$$

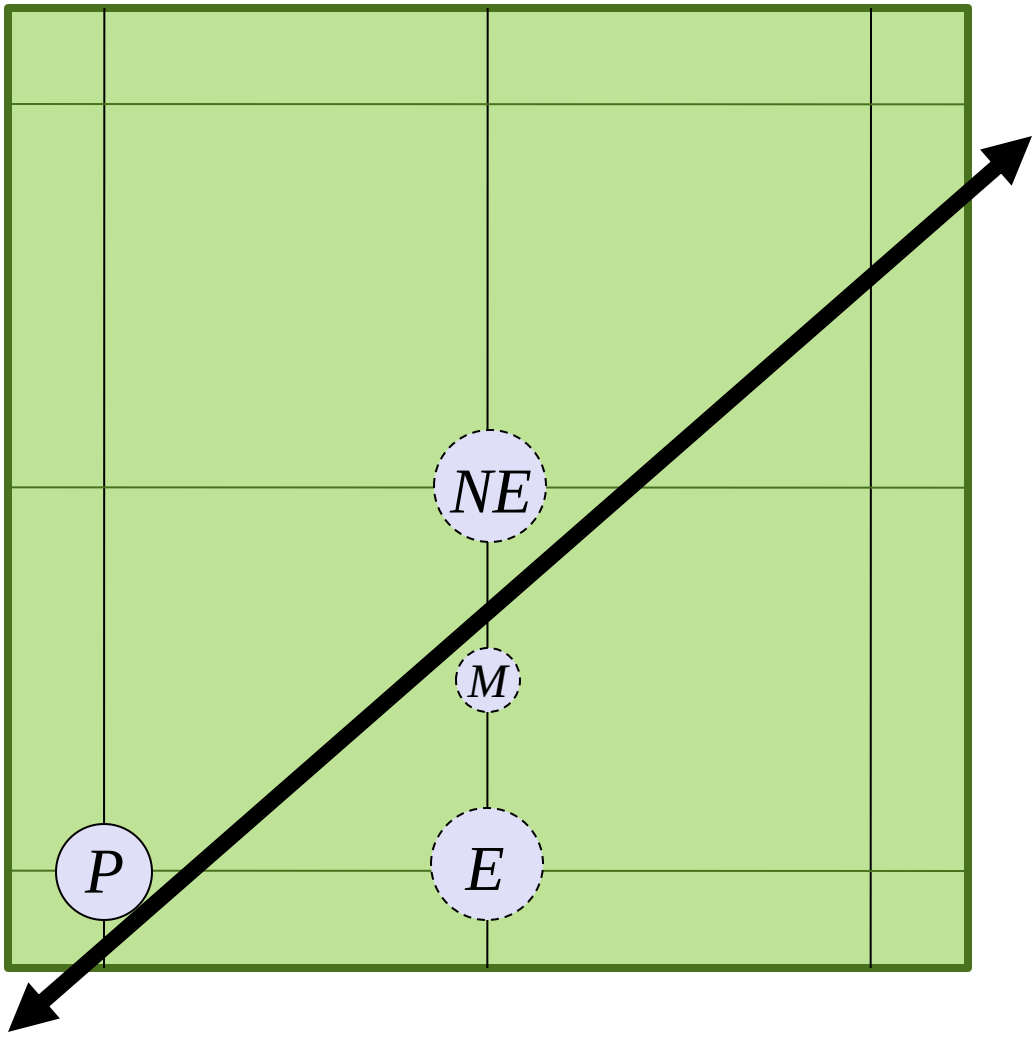
- $f(M) < 0 \rightarrow NE$
- $f(M) \geq 0 \rightarrow E$



Computing $f(M)$ Fast

If you know $f(P)$, then you can compute $f(M)$ easily:

$$
\begin{aligned}
f(x,y) &= mx + b - y \\
M &= P + (1,½) \\
f(M) &= f(x+1,y+½) \\
&= m(x+1) + b - (y+½) \\
&= mx + m + b - y - ½ \\
&= mx + b - y + m - ½ \\
&= f(P) + m - ½
\end{aligned}
$$



Preparing next $f(P)$

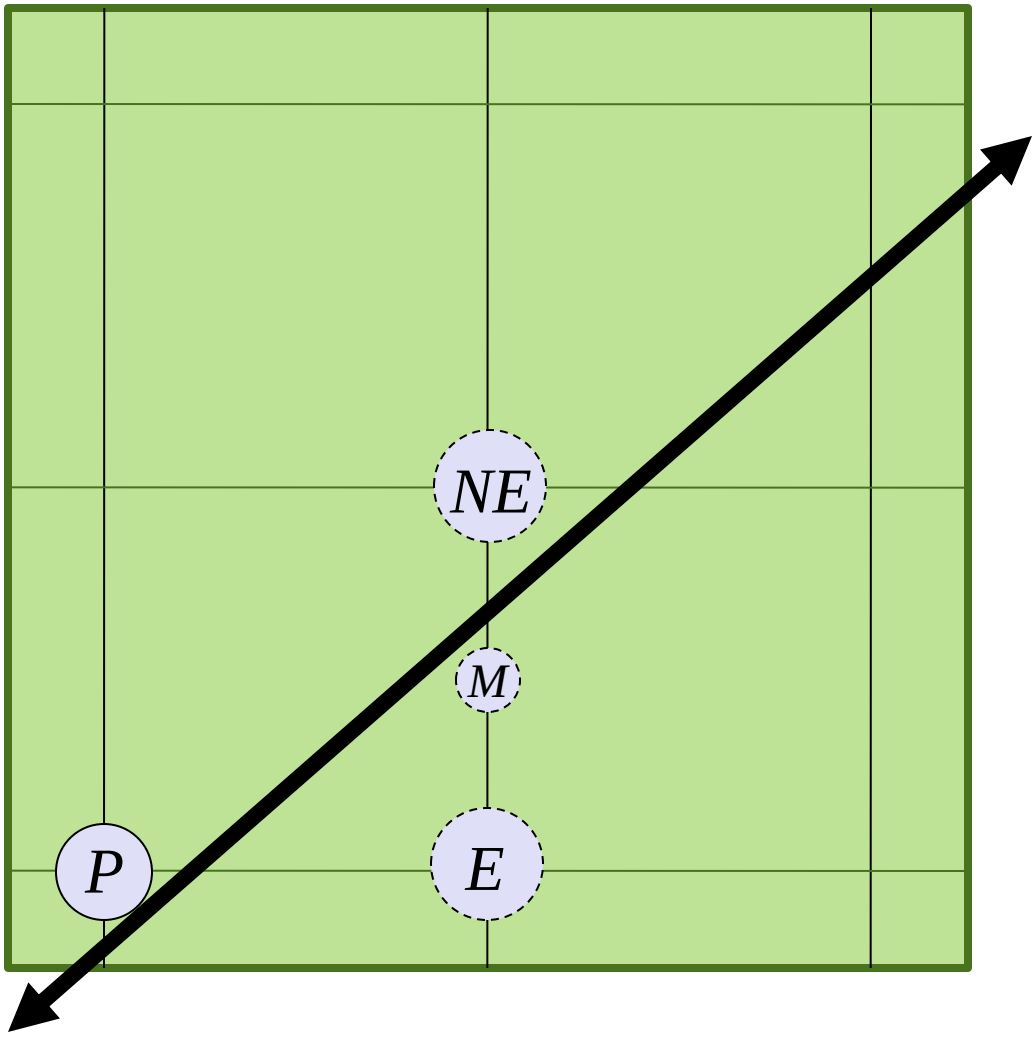
$$f(x,y) = mx + b - y$$

The next iteration's $f(P)$ is $f(E)$ or $f(NE)$ of this iteration

$$\begin{aligned} f(E) &= f(x+1,y) \\ &= f(P) + m \\ f(NE) &= f(x+1,y+1) \\ &= f(P) + m - 1 \end{aligned}$$

Also need $f(P)$ at start point:

$$f(0,0) = b$$



Midpoint Increments

$f(M) = f(P) + m - ½$

If choice is E, then next midpoint is M_E

$f(M_E) = f(x+2,y+½)$

$= m(x+2) + b - (y+½)$

$= f(P) + 2m - ½$

$= f(M) + m$

Otherwise next midpt. is M_{NE}

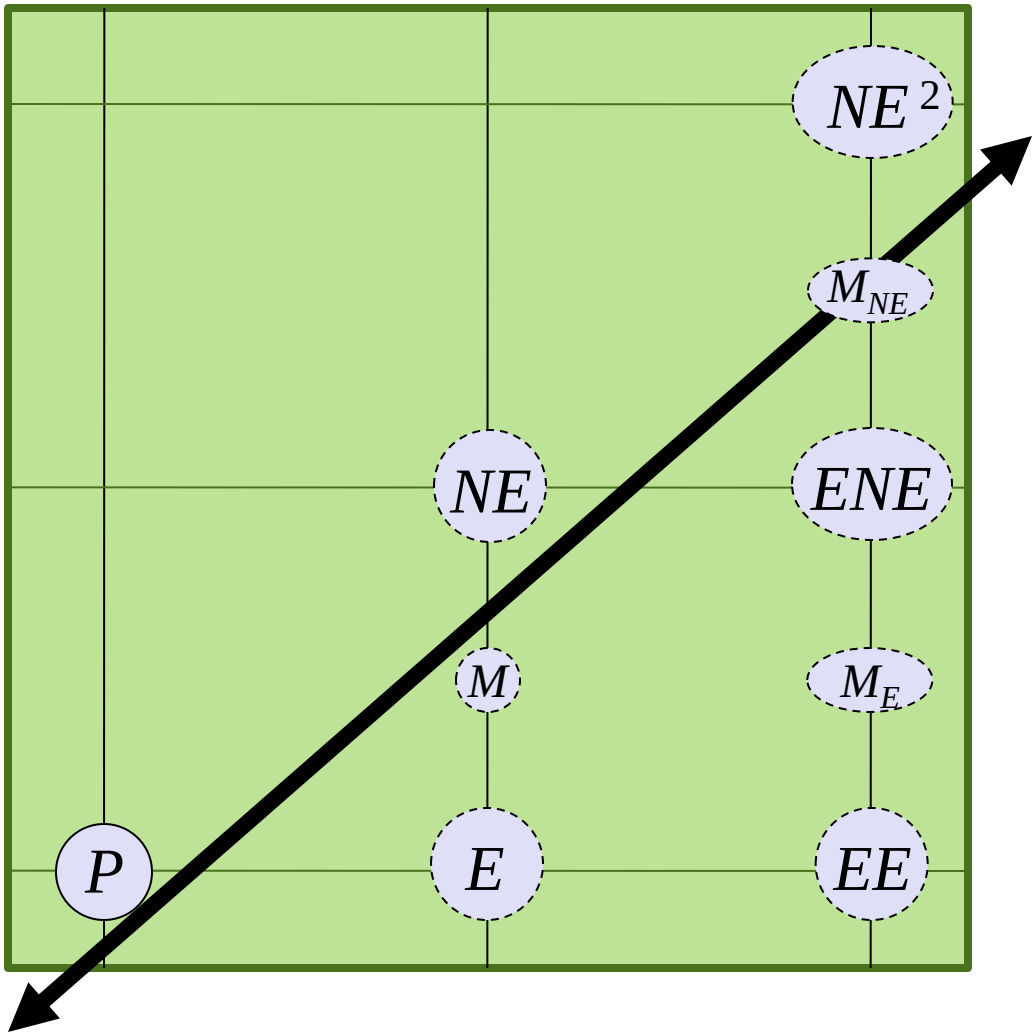
$f(M_{NE}) = f(x+2,y+1½)$

$= m(x+2) + b - (y+1½)$

$= f(P) + 2m - 1½$

$= f(M) + m - 1$

Initialize: $f(1, ½) = m + b - ½$



Integer Math

$f(M_E) = f(M) + m$

$f(M_{NE}) = f(M) + m - 1$

$f(1, ½) = m + b - ½$

$b = 0$

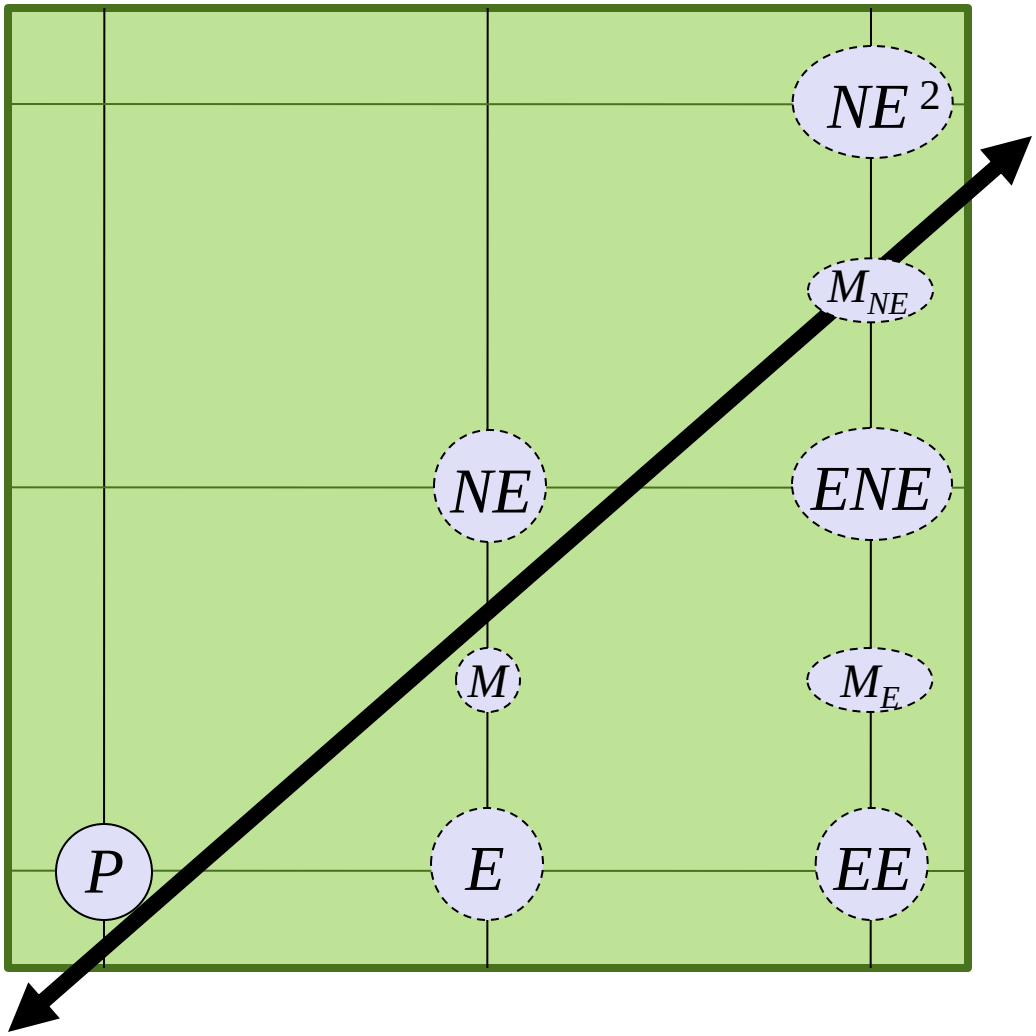
$m = (y_1 - y_0)/(x_1 - x_0)$

$= \Delta y/\Delta x$

$\Delta x\, f(M_E) = \Delta x\, f(M) + \Delta y$

$\Delta x\, f(M_{NE}) = \Delta x\, f(M) + \Delta y - \Delta x$

$\Delta x\, f(1, ½) = \Delta y - ½\, \Delta x$



Integer Math

$f(M_E) = f(M) + m$

$f(M_{NE}) = f(M) + m - 1$

$f(1, ½) = m + b - ½$

$b = 0$

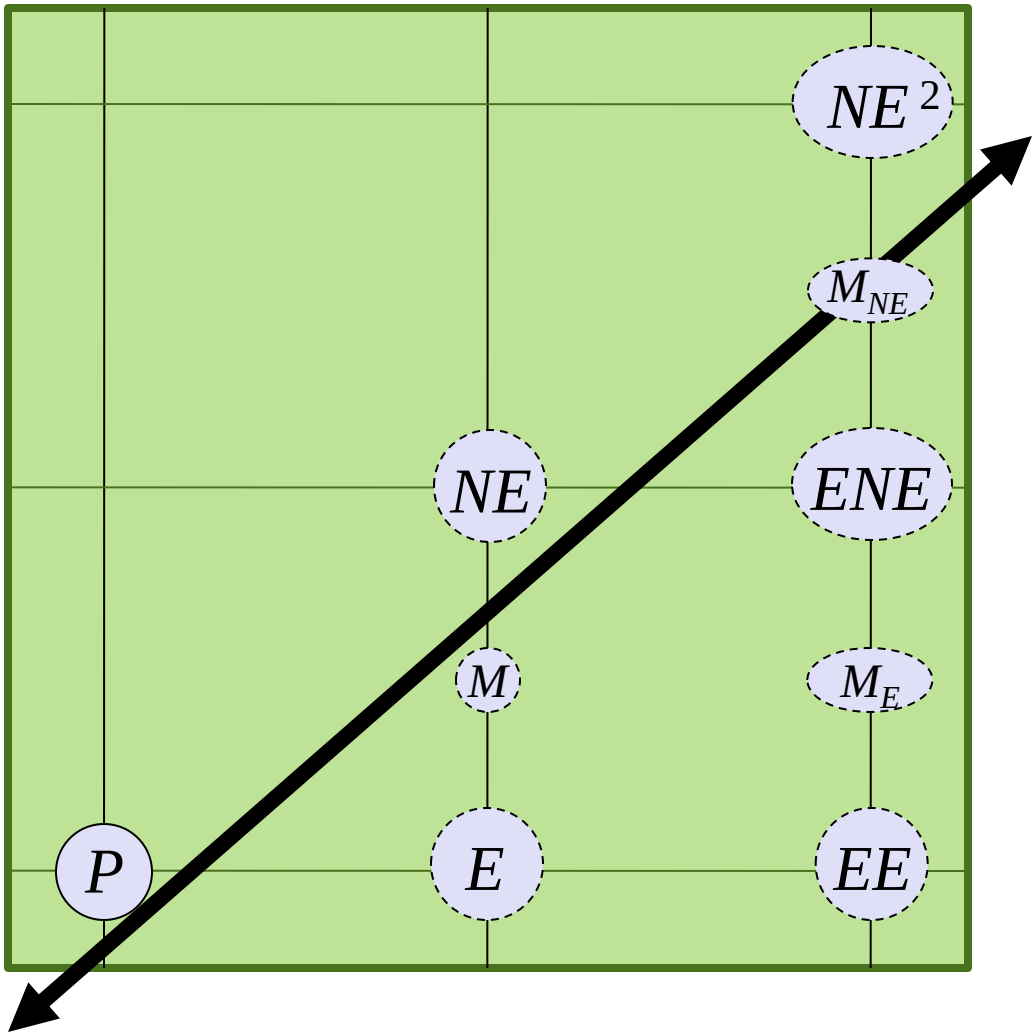
$m = (y_1 - y_0)/(x_1 - x_0)$

$= \Delta y/\Delta x$

$2\Delta x f(M_E) = 2\Delta x\, f(M) + 2\Delta y$

$2\Delta x f(M_{NE}) = 2\Delta x\, f(M) + 2\Delta y - 2\Delta x$

$2\Delta x f(1, ½) = 2\Delta y - \Delta x$



Integer Math

$f(M_E) = f(M) + m$

$f(M_{NE}) = f(M) + m - 1$

$f(1, ½) = m + b - ½$

$b = 0$

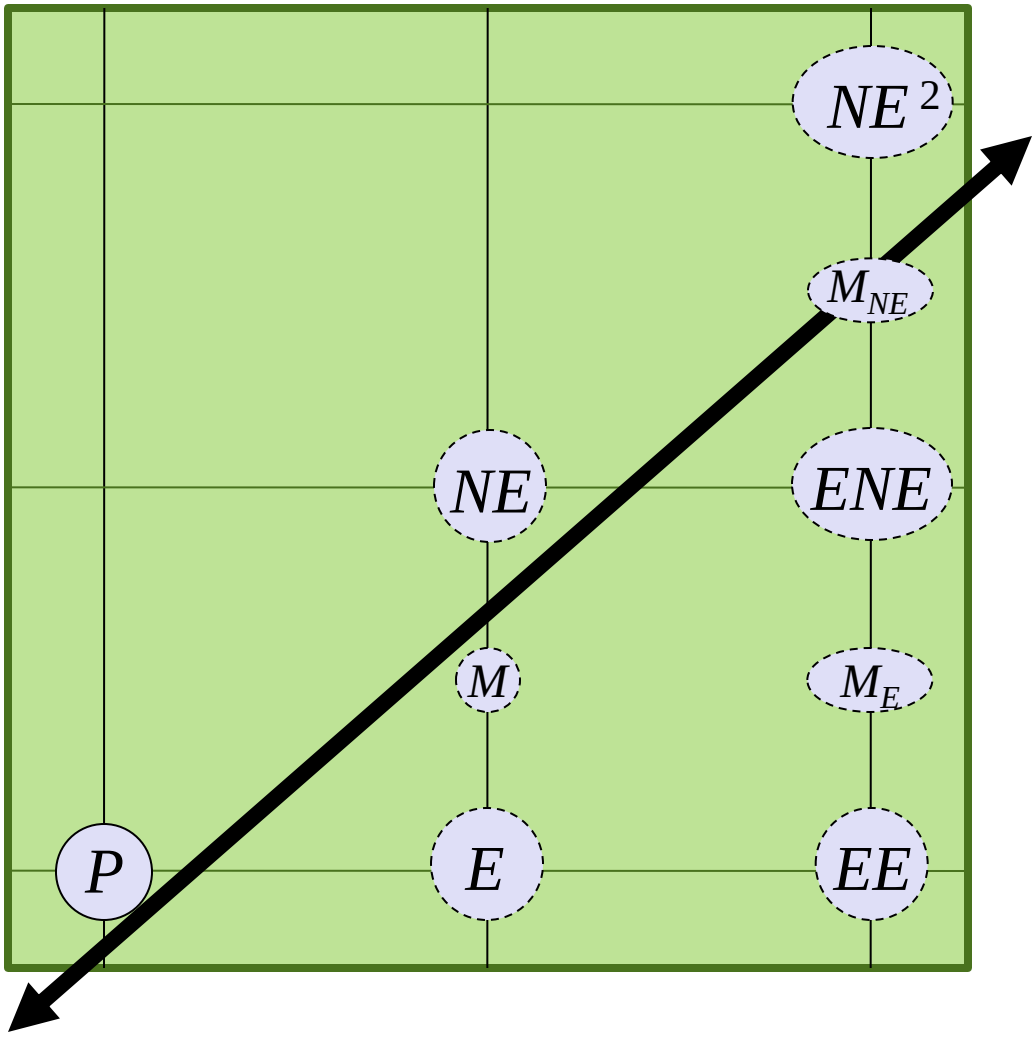
$m = (y_1 - y_0)/(x_1 - x_0)$

$= \Delta y/\Delta x$

$F(M_E) = F(M) + 2\Delta y$

$F(M_{NE}) = F(M) + 2\Delta y - 2\Delta x$

$F(1,½) = 2\Delta y - \Delta x$



Integer Math

$$f(M_E) = f(M) + m$$
$$f(M_{NE}) = f(M) + m - 1$$
$$f(1, ½) = m + b - ½$$
$$b = 0$$
$$m = (y_1 - y_0)/(x_1 - x_0)$$
$$= \Delta y/\Delta x$$
$$F(M_E) = F(M) + 2\Delta y$$
$$F(M_{NE}) = F(M) + 2\Delta y - 2\Delta x$$
$$F(1,½) = 2\Delta y - \Delta x$$

Bresenham Line Algorithm

```
line(int x0,int y0,int x1,int y1)
{
  int dx = x1 - x0;
  int dy = y1 - y0;

  int F = 2*dy - dx;

  int dFE = 2*dy;
  int dFNE = 2*dy - 2*dx;

  int y = y0;
  for (int x = x0, x < x1; x++) {
    plot(x,y);
    if (F >= 0) {
      F += dFE;
    } else {
      F += dFNE; y++;
    }
  }
}
```