Lecture 4: Line Conversion and Antialiased Line

- Purpose: Given a line $y = mx + b$ which goes through the two points (x_0, y_0) and (x_1, y_1) plot the pixels which are closest to the line.

- Criteria:
  - Convert the floating point coordinates to interger coordinates;
  - Minimize the number of calculations (e.g., multiplies and divides)

- Method:
  - brute-force method (e.g., digital differential analyzer (DDA))
  - Bresenham algorithm
  - Midpoint algorithm

1. Scan convering lines: DDA

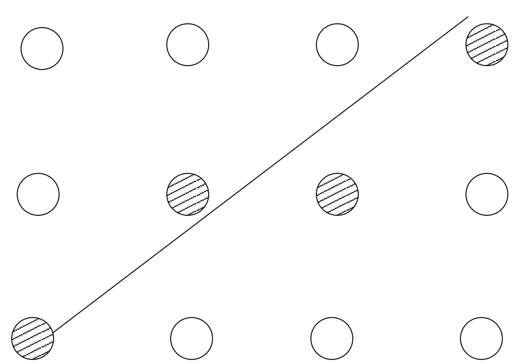
$$y_i = mx_i, \qquad m = \frac{\Delta y}{\Delta x} \tag{1}$$

$$Round(y_i) = Floor(0.5 + y_i) \tag{2}$$

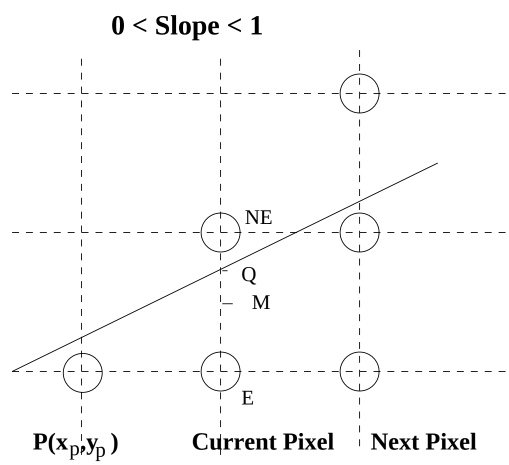
—— select the closest pixel to the true line
$y_{i+1} = y_i + m$
$\Rightarrow y_{i+1} = mx_{i+1} + b = mx_i + m + b = y_i + m$

Drawback:

- time consuming (rounding y $\rightarrow$ integer)
- variable y and m are real values



2. Brensenham Algorithm – integer arithmetic algorithm



Bresenham algorithm: $d = Q^{NE} - Q^{E}$

if $d > 0 \Rightarrow E$

if $d < 0 \Rightarrow NE$

Mid-point algorithm: observe mid-point M

if M lies above the line $\Rightarrow E$

if M lies below the line $\Rightarrow NE$

Note: error $< \frac{1}{2}$ pixel.

3. Mid-point Algorithm

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

$y = \frac{dy}{dx}x + B$
$\Rightarrow F(x, y) = dy \times x - dx \times y + b \times dx = 0$
$F(x, y) = 0 \rightarrow$ point (x,y) on the line
$F(x, y) > 0 \rightarrow$ point (x,y) below the line
$F(x, y) < 0 \rightarrow$ point (x,y) above the line

Mid-point: $(x_p + 1,\ y_p + \frac{1}{2})$
Define d as a *decision variable*:
$d = F(x_p + 1, y_p + \frac{1}{2})$
$= (x_p + 1)a + (y_p + \frac{1}{2})b + c$
if $d > 0 \Rightarrow NE$
if $d < 0 \Rightarrow E$
if $d = 0 \Rightarrow E$

3. Mid-point Algorithm (Cont'd)

What happens to the next grid?

- if the current pixel chooses E $\rightarrow$ next mid-point is $(x_p + 2, y_p + \frac{1}{2})$
  $d_{new} = F(x_p + 2, y_p + \frac{1}{2})$
  $d_{old} = F(x_p + 1, y_p + \frac{1}{2})$
  $\Rightarrow d_{new} = d_{old} + a$
  $\Rightarrow \ \Delta E = d_{new} - d_{old} = a = dy$

- if the current pixel chooses NE $\rightarrow$ next mid-point is $(x_p + 2, y_p + \frac{3}{2})$
  $d_{new} = F(x_p + 2, y_p + \frac{3}{2})$
  $d_{old} = F(x_p + 1, y_p + \frac{1}{2})$
  $\Rightarrow d_{new} = d_{old} + a + b$
  $\Rightarrow \ \Delta NE = d_{new} - d_{old} = a + b = dy - dx$
  note: $dy = a, \ dx = -b$

3. Mid-point Algorithm (Cont'd)

- The choose of E or NE is decided by the sign of the decision variable

- **Initial d:**
  $d_{start} = F(x_0 + 1, y_0 + \frac{1}{2}) = F(x_0 + y_0) + a + \frac{b}{2}) = dy - \frac{dx}{2}$
  In order to eliminate the fraction of d_{start}, define $F(x, y) = 2(ax + by + c)$

- **Procedure:**
  $dx = x_1 - x_0$;
  $dy = y_1 - y_0$;
  $d = 2dy - dx$;
  $\Delta E = 2dy$;
  $\Delta NE = 2(dy - dx)$;
  $setpixel(x_0, y_0)$;
  $while(x < x_1)$ {
  $if(d \leq 0)\{d = d + \Delta E; x + +\}$
  $else\{d = d + \Delta NE; x + +; y + +\}$

  $setpixel(x, y)$;
  }

4. Gupta-Sproull Antialiased lines

- **Problem of line converting:** Jagged edges (staircasing or aliasing problem) due to the digitized array which shows the line with all-or-nothing drawback.

- **Possible solution:**

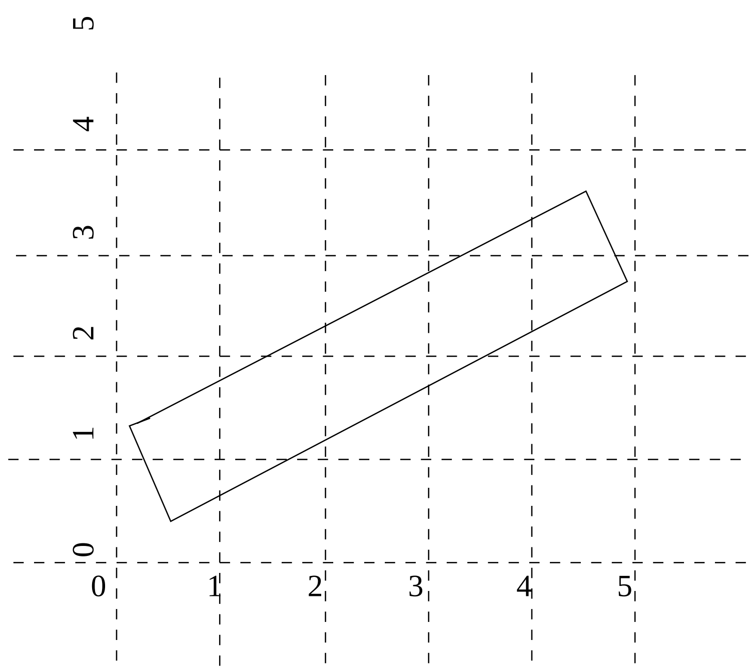
  – Increasing resolution: Jaggie size reduces half in x and y if the display resolution is twice in vertical and horizontal directions.

  – Creating thick line: treat the line as a thin rectangle and computing appropriate intensities for the multiple pixels in each column that lie in or near the rectangle.

4. Gupta-Sproull Antialiased lines (cont'd)

Line model:

- ideal line: width = 0
- real line: width = 1
- line shape: rectangle
- pixel shape: square



4. Gupta-Sproull Antialiased lines (cont'd)

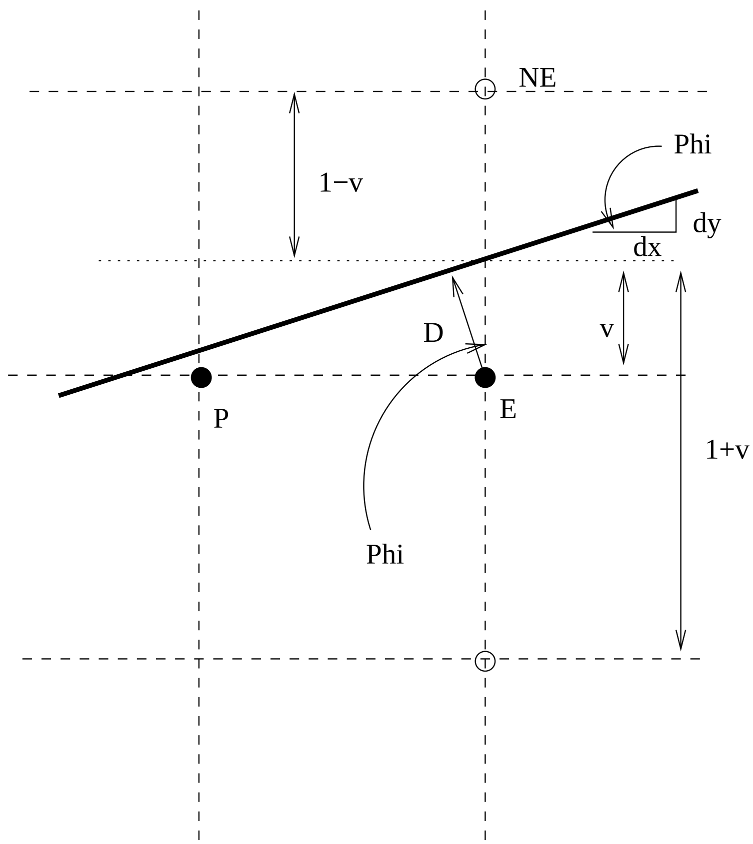
Principle:

- The modified version of Mid-point algorithm
- A line of unit thickness (with slope < 1) intersects three supports in a column.
- Decision variable d: choose E or NE pixel. $d = F(M) = F(x_p + 1, y_p + \frac{1}{2})$
- The pixel intensity to which a line contributes is proportional to the percentage of the pixel that the line covers.
- Chosen pixels and its two vertical neighbors will be set the intensity based on the distance D from these pixels to the line.

4. Gupta-Sproull Antialiased lines (cont'd)

Algorithm:

$$D = v\cos(\phi) = \frac{vdx}{\sqrt{dx^2+dy^2}}$$



Incremental computation of "$v \rightarrow D$"

4. Gupta-Sproull Antialiased lines (cont'd)

$F(x, y) = 2(ax + by + c) = 0$
$\Rightarrow y = (ax + c)/(-b)$

$v = y - y_p$
$\Rightarrow (ax + c)/(-b) - y_p = a(x_p + 1) + c/(-b) - y_p$

Multiply by $(-b)$:
$\Rightarrow -bv = a(x_p + 1) + by_p + c = F(x_p + 1, y_p)/2$

$b = -dx$
$\Rightarrow vdx = F(x_p + 1, y_p)/2$

4. Gupta-Sproull Antialiased lines (cont'd)

Case E: $2vdx = F(x_p + 1, y_p) = d + dx$
$\Rightarrow D = \frac{d+dx}{2\sqrt{dx^2+dy^2}}$

Define $L = 2\sqrt{dx^2 + dy^2}$

Two other neighbor pixels:

at $y_p + 1 \rightarrow$: $D = \frac{2(1-v)dx}{L} = \frac{2dx-2vdx}{L}$
at $y_p - 1 \rightarrow$: $D = \frac{2(1+v)dx}{L} = \frac{2dx+2vdx}{L}$

CASE NE: $2vdx = F(x_p + 1, y_p + 1) = d - dx$
at $y_p + 2 \rightarrow$: $D = \frac{2(1-v)dx}{L} = \frac{2dx-2vdx}{L}$
at $y_p \rightarrow$: $D = \frac{2(1+v)dx}{L} = \frac{2dx+2vdx}{L}$

4. Gupta-Sproull Antialiased lines (cont'd)

Note:

- In this algorithm, fractional arithmetic is used instead of integer

- Intensity value of the pixels is inversely propotional to the distance D, we can define them as a look up table (LUT):
  $f(D) = filter(D)$
  $f(0) = 1, f(1) = 15/16, ..., f(15) = 1/16.$

4. Gupta-Sproull Antialiased lines (cont'd)

Procedure:

$dx = x_1 - x_0;$

$dy = y_1 - y_0;$

$d = 2dy - dx;$

$\Delta E = 2dy;$

$\Delta NE = 2(dy - dx);$

$2vdx = 0;$

$setpixel(x_0, y_0);$

$setpixel(x_0, y_0 + 1);$

$setpixel(x_0, y_0 - 1);$

$while(x < x_1)$

$\{$

$if(d \leq 0)\{2vdx = d + dx; d = d + \Delta E; x + +; \}$

$else\{2vdx = d - dx; d = d + \Delta NE; x + +; y + +; \}$

$setpixel(x, y);$

$setpixel(x, y + 1);$

$setpixel(x, y - 1);$

$\}$