Pixels

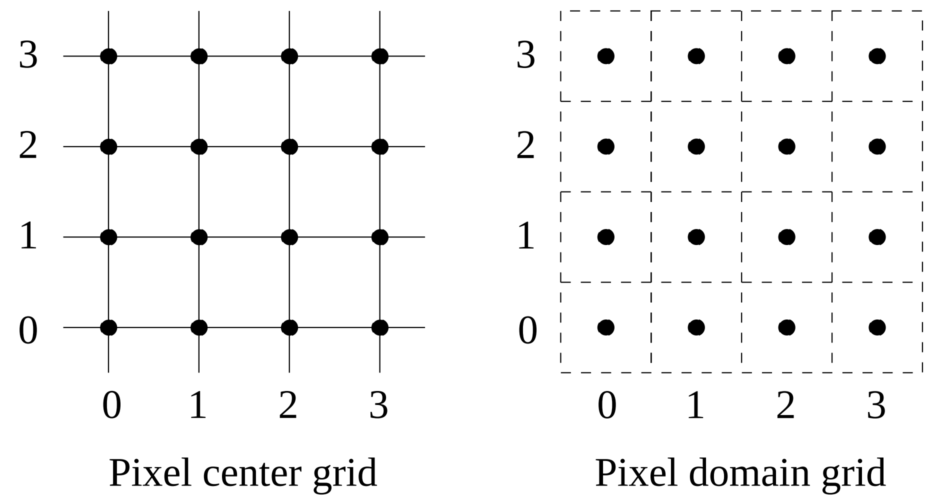
- *Pixel:* Intensity or color sample.
- *Raster Image:* Rectangular grid of pixels.
- *Rasterization:* Conversion of a primitive's geometric representation into
  - A set of pixels.
  - An intensity or color for each pixel (*shading, antialiasing*).
- For now, we will assume that the background is white and we need only change the color of selected pixels to black.

Pixel Grids

- *Pixel Centers:* Address pixels by integer coordinates (i, j)
- *Pixel Center Grid:* Set of lines passing through pixel centers.
- *Pixel Domains:* Rectangular semi-open areas surrounding each pixel center:

$$P_{i,j} = (i - 1/2, i + 1/2) \times (j - 1/2, j + 1/2)$$

- *Pixel Domain Grid:* Set of lines formed by domain boundaries.



Specifications and Representations

Each rendering primitive (point, line segment, polygon, etc.) needs both

- A geometric specification, usually "calligraphic."
- A pixel (rasterized) representation.

Standard device-level geometric specifications include:

Point: $A = (x_A, y_A) \in \mathbf{R}^2$.

Line Segment: $\ell(AB)$ specified with two points, A and B. The line segment $\ell(AB)$ is the set of all collinear points between point A and point B.

Polygon: Polygon $\mathcal{P}(A_1 A_2 \dots A_n)$ specified with an ordered list of points $A_1 A_2 \dots A_n$. A polygon is a region of the plane with a piecewise linear boundary; we connect A_n to A_1.

This "list of points" specification is flawed... a more precise definition will be given later.

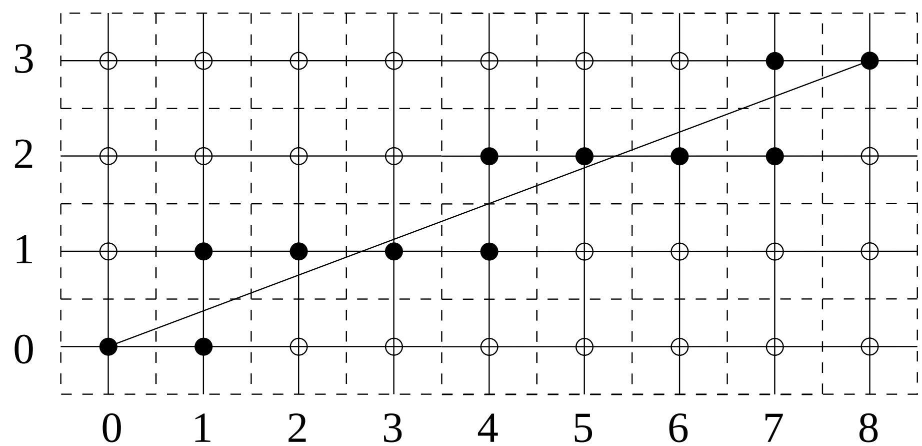
Line Segments

- Let $\ell(AB) = \{P \in \mathbf{R}^2 | P = (1-t)A + tB, t \in [0,1]\}$
- Problem: Given a line segment $\ell(AB)$ specified by two points A and B,
- Decide: Which pixels to illuminate to represent $\ell(AB)$.
- Desired properties: Rasterization of line segment should
  1. Appear as straight as possible;
  2. Include pixels whose domains contain A and B;
  3. Have relatively constant intensity (i.e., all parts should be the same brightness);
  4. Have an intensity per unit length that is independent of slope;
  5. Be symmetric;
  6. Be generated efficiently.

Line Segment Representations

1. Given AB, choose a set of pixels $L_1(AB)$ given by

$$L_1(AB) = \{(i,j) \in \mathbb{Z}^2 | \ell(AB) \cap P_{i,j}\}$$

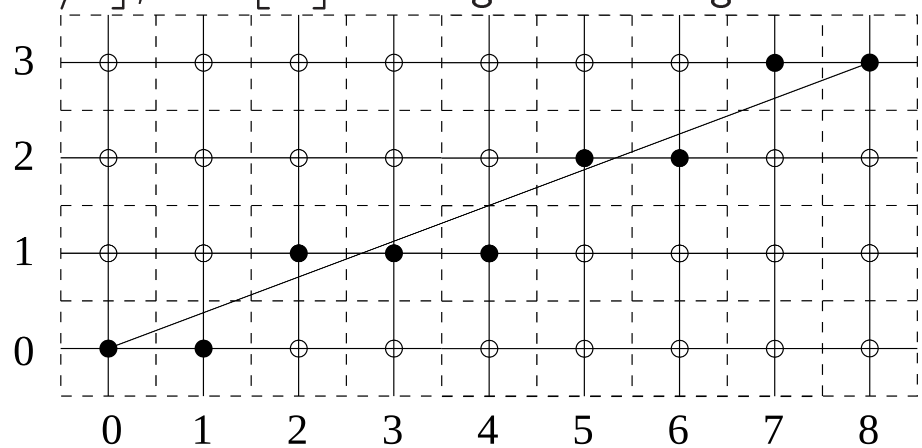


Unfortunately, this results in a very blotchy, uneven looking line.

2. Given AB, choose a set of pixels $L_2(AB)$ given by

$$L_2(AB) = \begin{cases} |x_B - x_A| \geq |y_B - y_A| \longrightarrow \\ \qquad \{(i,j) \in \mathbb{Z}^2 | (i,j) = (i,[y]), (i,y) \in \ell(AB), y \in \mathbf{R}\} \\ \qquad \cup([x_A],[y_A]) \cup ([x_B],[y_B]). \\ |x_B - x_A| < |y_B - y_A| \longrightarrow \\ \qquad \{(i,j) \in \mathbb{Z}^2 | (i,j) = ([x],j), (x,j), \in \ell(AB), x \in \mathbf{R}\} \\ \qquad \cup([x_A],[y_A]) \cup ([x_B],[y_B]). \end{cases}$$

Where $[z] = \lfloor z + 1/2 \rfloor$, and $\lfloor w \rfloor$ is the greatest integer less than or equal to w.



Line Equation Algorithm

Based on the line equation $y = mx + b$, we can derive:

```
LineEquation (int xA, yA, xB, yB)
    float m, b;
    int xi, dx;

    m = (yB - yA)/(xB - xA);
    b = yA - m*xA;
    if ( xB - xA > 0 ) then dx=1;
                       else dx = -1;
    for xi = xA to xB step dx do
         y = m*xi + b;
         WritePixel( xi, [y] );
endfor
```

Problems:

- One pixel per column so lines of slope > 1 have gaps

- Vertical lines cause divide by zero

To fix these problems, we need to use $x = m^{-1}(y - b)$ when $m > 1$.

Discrete Differential Analyzer

Observation: Roles of x and y are symmetric...

- Change roles of x and y if $|y_B - y_A| > |x_B - x_A|$

Observation: Multiplication of m inside loop...

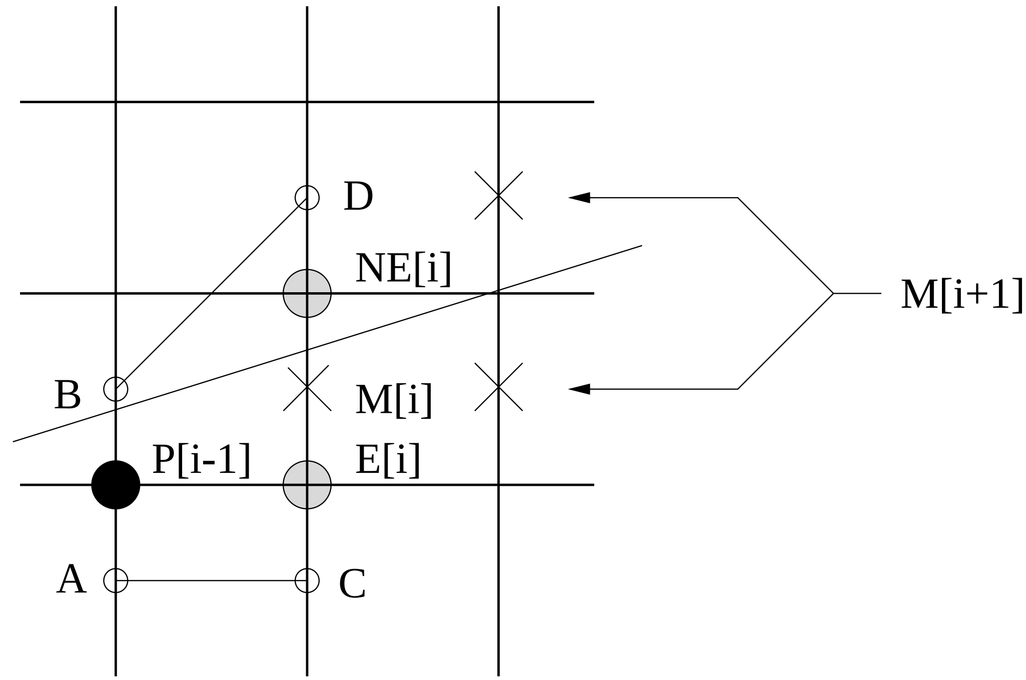
- The value of m is constant for all iterations.
- Can reduce computations inside loop: y_{i+1} and be computed incrementally from y_i

$$y_{i+1} = m(x_i + 1) + b = y_i + m$$

```
DDA (int xA, yA, xB, yB)
    int length, dx, dy, i;
    float x,y,xinc,yinc;

    dx = xB - xA;
    dy = yB - yA;

    length = max ( |dx| > |dy| );

    xinc = dx/length; # either xinc or yinc is -1 or 1
    yinc = dy/length;

    x = xA; y = yA;
    for i=0 to length do
        WritePixel( [x], [y] );
        x += xinc;
        y += yinc;
    endfor
```

Bresenham's Algorithm

- Completely integer;
- Will assume (at first) that x_A, y_A, x_B, y_B are also integer.
- Only addition, subtraction, and shift in inner loop.
- Originally for a pen plotter.
- "Optimal" in that it picks pixels closest to line, i.e., $L_2(AB)$.
- Assumes $0 \leq (y_B - y_A) \leq (x_B - x_A) \leq 1$ (i.e., slopes between 0 and 1).
- Use reflections and endpoint reversal to get other slopes: 8 cases.



- Suppose we know at step $i-1$ that pixel $(x_i, y_i) = P_{i-1}$ was chosen.
  Thus, the line passed between points A and B.
- Slope between 0 and 1 $\Rightarrow$
  line must pass between points C and D at next step $\Rightarrow$
  $E_i = (x_i + 1, y_i)$ and $N\ E_i = (x_i + 1, y_i + 1)$ are only choices for next pixel.
- If M_i above line, choose E_i;

- If M_i below line, choose $N\ E_i$.

- Implicit representations for line:

$$y = \frac{\Delta y}{\Delta x}x + b$$
$$F(x,y) = \underbrace{(2\Delta y)}_{Q}x + \underbrace{(-2\Delta x)}_{R}y + \underbrace{2\Delta x b}_{S} = 0$$

where

$$\begin{aligned}
\Delta x &= x_B - x_A \\
\Delta y &= y_B - y_A \\
b &= y_A - \frac{\Delta y}{\Delta x}x_A \Rightarrow S = 2\Delta x y_A - 2\Delta y x_A
\end{aligned}$$

Note that

1. $F(x,y) < 0 \Rightarrow (x,y)$ *above* line.
2. $F(x,y) > 0 \Rightarrow (x,y)$ *below* line.
3. Q, R, S are all integers.

- The mystery factor of 2 will be explained later.

- Look at $F(M_i)$. Remember, F is 0 if the point is on the line:
  - $F(M_i) < 0 \Rightarrow M_i$ *above* line $\Rightarrow$ choose $P_i = E_i$.
  - $F(M_i) > 0 \Rightarrow M_i$ *below* line $\Rightarrow$ choose $P_i = N\ E_i$.
  - $F(M_i) = 0 \Rightarrow$ arbitrary choice, consider choice of pixel domains...
- We'll use $d_i = F(M_i)$ as an *decision variable*.
- Can compute d_i incrementally with integer arithmetic.

- At each step of algorithm, we *know* P_{i-1} and d_i...
- Want to *choose* P_i and *compute* d_{i+1}
- Note that

$$\begin{aligned} d_i &= F(M_i) = F(x_{i-1} + 1, y_{i-1} + 1/2) \\ &= Q \cdot (x_{i-1} + 1) + R \cdot (y_{i-1} + 1/2) + S \end{aligned}$$

- If E_i is chosen then

$$\begin{aligned} d_{i+1} &= F(x_{i-1} + 2, y_{i-1} + 1/2) \\ &= Q \cdot (x_{i-1} + 2) + R \cdot (y_{i-1} + 1/2) + S \\ &= d_i + Q \end{aligned}$$

- If $N\ E_i$ is chosen then

$$\begin{aligned} d_{i+1} &= F(x_{i-1} + 2, y_{i-1} + 1/2 + 1) \\ &= Q \cdot (x_{i-1} + 2) + R \cdot (y_{i-1} + 1/2 + 1) + S \\ &= d_i + Q + R \end{aligned}$$

- Initially, we have

$$
\begin{aligned}
d_1 &= F(x_A + 1, y_A + 1/2) \\
&= Q_{x_A} + R_{y_A} + S + Q + R/2 \\
&= F(x_A, y_A) + Q + R/2 \\
&= Q + R/2
\end{aligned}
$$

- Note that $F(x_A, y_A) = 0$ since $(x_A, y_A) \in \ell(AB)$.
- Why the mysterious factor of 2?
  It makes everything integer.

```
Bresenham (int xA, yA, xB, yB)
    int d, dx, dy, xi, yi
    int incE, incNE

    dx = xB - xA
    dy = yB - yA
    incE = dy<<1                /* Q */
    incNE = incE - dx<<1;       /* Q + R */
    d = incE - dx               /* Q + R/2 */
    xi = xA; yi = yA
    WritePixel( xi, yi )
    while ( xi < xB )
        xi++
        if ( d < 0 ) then /* choose E */
             d += incE
        else /* choose NE */
             d += incNE
             yi++
        endif
```

```
    WritePixel( xi, yi )
endwhile
```

- Some asymmetries (choice when ==).
- Did we meet our goals?
  1. Straight as possible: yes, but depends on metric.
  2. Correct termination.
  3. Even distribution of intensity: yes, more or less, but:
  4. Intensity varies as function of slope.
     - Can't do better without gray scale.
     - Worst case: diagonal compared to horizontal (same number of pixels, but $\sqrt{2}$ longer line).
  5. Careful coding required to achieve some form of symmetry.
  6. Fast! (if integer math fast ...)
- Interaction with clipping?
- Subpixel positioning of endpoints?
- Variations that look ahead more than one pixel at once...
- Variations that compute from both end of the line at once...
- Similar algorithms for circles, ellipses, ...
  (8 fold symmetry for circles)

Reading Assignment and News

Chapter 8 pages 379 - 399, of Recommended Text.

(Recommended Text: Interactive Computer Graphics, by Edward Angel, 4th edition, Addison-Wesley)

Please also track the News section of the Course Web Pages for the most recent Announcements related to this course.

(http://www.cs.utexas.edu/users/bajaj/graphics24/cs354/)