Outline

- Line drawing
- Digital differential analyzer
- Bresenham’s algorithm
- XPM file format

Scan-Conversion Algorithms

- Scan-Conversion: Computing pixel coordinates for ideal line on 2D raster grid
- Pixels best visualized as circles/dots – Why? Monitor hardware

Drawing a Line

- \( y = mx + B \)
- \( m = \Delta y / \Delta x \)
- Start at leftmost \( x \) and increment by 1
  \( \Delta x = 1 \)
- \( y_i = \text{Round}(mx_i + B) \)
- This is expensive and inefficient
- Since \( \Delta x = 1 \), \( y_{i+1} = y_i + \Delta y = y_i + m \)
  – No more multiplication!
- This leads to an incremental algorithm

Digital Differential Analyzer (DDA)

- If \( |\text{slope}| < 1 \)
  - \( \Delta x = 1 \)
  - else \( \Delta y = 1 \)
- Check for vertical line
- \( m = \infty \)
- Compute corresponding \( \Delta y (\Delta x) = m (1/m) \)
- \( x_{k+1} = x_k + \Delta x \)
- \( y_{k+1} = y_k + \Delta y \)
- Round \((x, y)\) for pixel location
- Issue: Would like to avoid floating point operations

Generalizing DDA

- If \( |\text{slope}| \) is less than or equal to 1
  – Ending point should be right of starting point
- If \( |\text{slope}| \) is greater than 1
  – Ending point should be above starting point
- Vertical line is a special case
  \( \Delta x = 0 \)
Bresenham’s Algorithm

- 1965 @ IBM
- Basic Idea:
  - Only integer arithmetic
  - Incremental
- Consider the implicit equation for a line:
  \[ f(x,y) = ax + by + c = 0 \]

Bresenham’s Algorithm

Given: implicit line equation: \( f(x,y) = ax + by + c = 0 \)
Let: \( d_x = r_x - q_x, \quad d_y = r_y - q_y \)
where \( r \) and \( q \) are points on the line and \( d_x, d_y \) are positive
\[ a = d_y, \quad b = -d_x, \quad c = -(q_x r_y - r_x q_y). \]

Then:
Observe that all of these are integers
and: \( f(x,y) < 0 \) for points above the line
\( f(x,y) > 0 \) for points below the line
Now......

The Algorithm

```
void bresenham(int x0, int y0, int x1, int y1) {
    int dx = x1 - x0, dy = y1 - y0; // line length and breadth
    if (dx < 0) { dx = -dx; x0 += 1; } // make dx positive
    if (dy < 0) { dy = -dy; y0 += 1; } // make dy positive
    int d = 2 * dy - dx; // initial decision value
    for (int x = x0, y = y0; x <= x1; x++) {
        writePixel(x, y); // write pixel (x, y)
        if (d < 0) d += 2 * dy; // below midpoint - go to (x+1, y)
        else { d += 2 * dy - 2 * dx; // above midpoint - go to (x+1, y+1) }
    }
}
```

Assumptions: \( q_x < r_x \)
0 ≤ slope ≤ 1
Pre-computed: \( 2d_y, 2(d_y - d_x) \)

Bresenham’s Algorithm

• Suppose we just finished \( (p_x, p_y) \)
  - (assume 0 ≤ slope ≤ 1)
other cases symmetric
• Which pixel next?
  - \( E \) or \( NE \)

Assume:
• \( Q = \) exact \( y \) value at \( x = p_x + 1 \)
• \( y \) midway between \( E \) and \( NE \): \( M = p_y + 1/2 \)
Observe:
If \( Q < M \), then pick \( E \)
Else pick \( NE \)
If \( Q = M \),
  it doesn’t matter

Bresenham’s Algorithm

• Create “modified” implicit function (2x)
  \( f(x,y) = 2ax + 2by + 2c = 0 \)
• Create a decision variable \( D \) to select, where \( D \) is the value of \( f \) at the midpoint:
  \[ D = f(p_x + 1, p_y + 1/2) = 2a(p_x + 1) + 2b \left( p_y + \frac{1}{2} \right) + 2c \]
  \[ = 2ap_x + 2bp_y + (2a + b + 2c) \]
Bresenham’s Algorithm

- If \( D > 0 \) then \( M \) is below the line \( f(x, y) \)
  - \( NE \) is the closest pixel
- If \( D \leq 0 \) then \( M \) is above the line \( f(x, y) \)
  - \( E \) is the closest pixel

\[ D_{\text{init}} = f(q_x + 1, q_y + 1/2) \]
\[ = 2a(q_x + 1) + 2b\left(q_y + \frac{1}{2}\right) + 2c \]
\[ = (2aq_x + 2bq_y + 2c) + (2a + b) \]
\[ = 0 + 2a + b \]
\[ = 2d_y - d_x \]

How to get an initial value for \( D \)?

- Suppose we start at: \((q_x, q_y)\)
- Initial midpoint is: \((q_x + 1, q_y + 1/2)\)
- Note: because we multiplied by 2\(x\), \(D\) is now an integer—which is very good news
- How do we make this incremental??

Case I: When \( E \) is next

- What increment for computing a new \( D \)?
- Next midpoint is: \( p_x + 2, p_y + (1/2) \)
  \[ D_{\text{new}} = f(p_x + 2, p_y + (1/2)) \]
  \[ = 2a(p_x + 2) + 2b\left(p_y + \frac{1}{2}\right) + 2c \]
  \[ = 2ap_x + 2bp_y + (4a + b + 2c) \]
  \[ = 2ap_x + 2bp_y + (2a + b + 2c) + 2a \]
  \[ = D + 2a = D + 2d_y \]
- Hence, increment by: \(2d_y\)

Case II: When \( NE \) is next

- What increment for computing a new \( D \)?
- Next midpoint is: \( p_x + 2, p_y + 1 + (1/2) \)
  \[ D_{\text{new}} = f(p_x + 2, p_y + 1 + (1/2)) \]
  \[ = 2a(p_x + 2) + 2b\left(p_y + \frac{3}{2}\right) + 2c \]
  \[ = 2ap_x + 2bp_y + (4a + 3b + 2c) \]
  \[ = 2ap_x + 2bp_y + (2a + b + 2c) + 2a + 2b \]
  \[ = D + 2(a + b) = D + 2(d_y - d_x) \]
- Hence, increment by: \(2(d_y - d_x)\)

The Algorithm

```c
void bresenham(intPoint qi, intPoint qj) {  
  int dx, dy, D, X, Y;  // line width and height;  
  dx = xj - xi;  // initial decision value  
  dy = yj - yi;  // start at (x, y)  
  D = 2dy - dx;  
  if (D <= 0) b = 2dy;  
  else {  // below midpoint - go to NE  
    D = 2(dy - dx);  
    b = 2dy;  
  }  
  Assumptions:  
  q_x \leq x_j  
  0 \leq \text{slope} \leq 1  
  Pre-computed: 2d_y 2(d_y - d_x)  
  }
```

Pics/Math courtesy of Dave Mount @ UMD-CP
Generalize Algorithm

- If \( q_x > r_x \), swap points
- If slope > 1, always increment \( y \), conditionally increment \( x \)
- If \(-1 \leq \text{slope} < 0\), always increment \( x \), conditionally decrement \( y \)
- If slope < -1, always decrement \( y \), conditionally increment \( x \)
- Rework D increments

Bresenham’s Algorithm: Example

\[ f(x,y) = 2Dy - X_3y + 0 \]
\[ f(x,y) = 2Dx - 7y + 0 \]

(0,0) (7,5)

Bresenham’s Algorithm: Example

Reflect line into first case
- Calculate pixels
- Reflect pixels back into original orientation
Bresenham’s Algorithm: Example

Some issues with Bresenham’s Algorithms

- Pixel ‘density’ varies based on slope
  - straight lines look darker, more pixels per unit length
  - Endpoint order
  - Line from P1 to P2 should match P2 to P1
  - Always choose E when hitting M, regardless of direction