**Line Generation**

**Outline**
- Line generation problem
- Line equations
- Naïve approaches:
  - Based on explicit representation
  - Based on parametric representation
- Midpoint line drawing algorithm

**Line Generation Problem**
- How to display a straight line from \((x_0, y_0)\) to \((x_1, y_1)\) on a monitor?
- Select pixels that are closest to the line
- Do it efficiently
- Simplifications:
  - Lines are 1 pixel wide
  - No need for anti-aliasing

**Line Equations**
- Explicit representation:
  - \(y = mx + b\)
  - \(m = \frac{\Delta y}{\Delta x}\) (slope)
  - \(b = y_0 - mx_0\) (Y intercept)
- Parametric function:
  - \(x = x_0 + k\Delta x\)
  - \(y = y_0 + k\Delta y\)
- Implicit representation:
  - \(F(x, y) = ax + by + c = 0\)
  - \(a = \Delta y, b = -\Delta x\)
  - \(c = \Delta x \times y_0 - \Delta y \times x_0\)

**Approach Based on Explicit Representation**
- Naïve version:
  ```c
  // Naïve version:
  float m = dy / dx;
  for ( int x=x0 ; x<=x1 ; x++ ) {
    float y = y0 + (x - x0) * m;
    draw(x, round(y));
  }
  ```
- Incremental version:
  ```c
  // Incremental version:
  float m = dy / dx;
  float y = y0;
  for ( int x=x0 ; x<=x1 ; x++, y+=m )
    draw(x, round(y));
  ```

**Limitations**
- Missing pixels
- Works great if \(|m| < 1\)
- Must reverse the roles of x and y if \(|m| > 1\)
- Error accumulation
- Slope m is a fractional number
- May not be able to precisely represented in a computer
Approach Based on Parametric Representation

• Limitations:
  - No missing pixel problem but error accumulation still exists
  - Computational cost depends on the value of \( \delta k \)
  - Normally set \( \delta k = \frac{1}{\max(\Delta x, \Delta y)} \)
  - Either \( x \) or \( y \) increase by 1 each time

Midpoint Line Algorithm

• Origin:
  - First proposed by Bresenham in 1965.
  - Improved by Pitteway in 1967
• Advantage:
  - Incremental approach
  - Use integer arithmetic only
  - Fast
  - No error accumulation

Eight Cases

- \( \Delta x > 0, \Delta y > 0, |\Delta x| > |\Delta y| \)
- \( \Delta x > 0, \Delta y > 0, |\Delta y| > |\Delta x| \)
- \( \Delta x < 0, \Delta y > 0, |\Delta y| > |\Delta x| \)
- \( \Delta x < 0, \Delta y < 0, |\Delta y| > |\Delta x| \)
- \( \Delta x > 0, \Delta y < 0, |\Delta x| > |\Delta y| \)
- \( \Delta x < 0, \Delta y < 0, |\Delta x| > |\Delta y| \)
- \( \Delta x > 0, \Delta y < 0, |\Delta y| > |\Delta x| \)
- \( \Delta x < 0, \Delta y > 0, |\Delta x| > |\Delta y| \)

Consider Case 1 First

- Others can be handled by suitable reflections about the principle axes
- Case 8 is reflection over X axis
- Case 4 is reflection over Y axis
- ...

Basic Idea

- Assume \((u, v)\) is selected
- Need to chose between \((u+1, v)\) and \((u+1, v+1)\)
  - If \(M\) is above the line, pick \((u+1, v)\)
  - Otherwise, pick \((u+1, v+1)\)

Relative Position of a Point

- Position of a point relative to the line can be determined using the sign of implicit function
  - \( F(x, y) = ax + by + c = 0 \)
  - Since \( a = \Delta y > 0 \) & \( b = -\Delta x < 0 \)
    - Above the line:
      - \( F(u, v) < 0 \)
    - Below the line:
      - \( F(u, v) > 0 \)
First Decision

- Line starts from \((x_0, y_0)\)
  - \((x_0, y_0)\) should be picked
  - \(M = (x_0 + 1, y_0 + \frac{1}{2})\)
- Let \(d = F(M)\):
  - \(d = a(x_0 + 1) + b(y_0 + \frac{1}{2}) + c\)  
    - \(= ax_0 + by_0 + \left(c + a + \frac{b}{2}\right)\) 
    - \(= F(x_0, y_0) + a + \frac{b}{2}\)  
    - \(= a + \frac{b}{2}\) 
    - \(= \Delta y - \Delta x/2\)

Next Decision (Scenario 1)

- If \((u+1, v+1)\) is picked
  - \(M' = (u+2, v+\frac{1}{2})\)
  - \(d' = F(M') = a(u+2) + b(v+\frac{1}{2}) + c\)
- Previous \(d\) value is known
  - \(d = F(M) = a(u+1) + b(v+\frac{1}{2}) + c\)
- Difference:
  - \(d' - d = a = \Delta y\)
- Update function:
  - \(d' = d + \Delta y\)

Next Decision (Scenario 2)

- If \((u + 1, v + 1)\) is picked
  - \(M' = (u + 2, v + 1/2)\)
  - \(d' = F(M') = a(u + 2) + b(v + 1/2) + c\)
- Previous \(d\) value:
  - \(d = F(M) = a(u + 1) + b(v + 1/2) + c\)
- Difference:
  - \(d' - d = a = \Delta y\)
- Update function:
  - \(d' = d + \Delta y - \Delta x\)

Overall Algorithm

- int \(x0\), \(y0\);
- int \(d\);
- int \(d2\);
- while \((x < x1)\) {
  - if \((d <= 0)\) {
    - \(d += \Delta y;\)
    - \(x ++;\)
  } else {
    - \(d += \Delta y - \Delta x;\)
    - \(x ++; y ++;\)
  }
- draw \((x, y)\);
- }

Pure Integer Implementation

- int \(x0\), \(y0\);
- int \(d2\);
- while \((x < x1)\) {
  - if \((d2 <= 0)\) {
    - \(d2 += \Delta y * 2;\)
    - \(x ++;\)
  } else {
    - \(d2 += \Delta y - \Delta x;\)
    - \(x ++; y ++;\)
  }
- draw \((x, y)\);
- }

All decisions are made based on the sign of variable \(d\), not its value:
- Multiplying all variables by 2 does not change the algorithm
- The result is a pure integer implementation