Scan-Conversion

- figuring out what pixels have to be colored in order to draw a geometric primitive on the screen
  - lines
  - circles/ellipses
  - polygons
  - text

Scan-Conversion Overview

- lines
  - DDA
  - midpoint/Bresenham
- circles
  - midpoint circle
- filled polygons
  - scan-line
- antialiasing

Scan-Conversion: Lines

the task: draw a line on a raster screen between two points

- mathematical line vs. rasterized line
- jaggies

revised task: given two integer-coordinate points on the plane, determine what pixels on a raster screen should be on to create a picture of a unit-width line segment between those points

A Simplification

- we'll consider only lines with slope $0 < m < 1$
  - horizontal and vertical lines are special cases
  - diagonal lines ($m = \pm 1$) can be special-cased or included in general case
  - can use symmetry to adapt solution for $0 < m < 1$ for $m > 1$ and negative slopes

Another Simplification

- consider line with slope $0 < m < 1$
  - generally only one pixel per column inside the unit-width line segment
  - we'll draw just one pixel per column, using the closest to the "true line" if there are two choices
  - vertical distance from point to line is proportional to perpendicular distance
Lines: Basic Algorithm
- find the equation of the line connecting endpoints P & Q
- starting with the leftmost point P, $x_i = x_{i-1} + 1$ and $y_i = m \cdot x_i + b$
- color pixel at $(x_i, \text{round}(y_i))$
- each iteration requires floating-point multiplication

Lines: DDA Algorithm
- determine slope of the line connecting endpoints P & Q
- starting with the leftmost point P, $x_i = x_{i-1} + 1$ and $y_i = y_{i-1} + m$
- color pixel at $(x_i, \text{round}(y_i))$
- each iteration still requires floating-point arithmetic and rounding
- repeated summing of fractional values can lead to roundoff problems for very long lines

Lines: Midpoint Line Algorithm
- observation: the next pixel colored is always either E or NE of the current pixel
- need a way to decide between the choices...
  - (and with only using integer arithmetic)
- let's consider the midpoint between the two choices...

The Midpoint
- if the line goes between the midpoint and the NE point, NE is closer
- if the line goes between the midpoint and the E point, E point is closer

Equation of a Line
- one way to express a line $y = mx + B = \frac{\Delta y}{\Delta x} x + B$
  - m is the slope, B is the y intercept
- another way of expressing a line $F(x, y) = ax + by + c = 0$
  - rearranging the first equation yields $(\Delta y)x - (\Delta x)y + (\Delta x)B = 0$
  - $F(x, y) = 0$ if $(x, y)$ is on the line
  - $F(x, y) < 0$ if $(x, y)$ is above the line
  - $F(x, y) > 0$ if $(x, y)$ is below the line

Putting the Bits Together
- if midpoint is below the line, then line is between midpoint and NE
  - if $F(x_m, y_m) > 0$, choose NE
- if midpoint is above the line, then line is between midpoint and E
  - if $F(x_m, y_m) < 0$, choose E
- if midpoint is on the line, then line goes through the midpoint
  - if $F(x_m, y_m) = 0$, choose either E or NE (but be consistent)
- we'll write $F(x_m, y_m)$ as d and call it the decision variable
Notation Roundup

• line endpoints are \( P = (x_p, y_p) \) and \( Q = (x_q, y_q) \)
  \[ \Delta x = x_q - x_p, \Delta y = y_q - y_p \]
  \( P \) and \( Q \) have integer coordinates

• pixels are colored one column at a time
  \((x_k, y_k)\) are the coordinates of the pixel colored in column \( k \), \(0 \leq k \leq \Delta y \) (\( k \) is an integer)

• \( d_k = F(x_k + 1, y_k + 1/2) \)
  \( d_k \) is decision variable used to choose where pixel in column \( k \) goes

An Algorithm Based on the Midpoint

• to compute \((x_{k+1}, y_{k+1})\)...
  - midpoint is \((x_k + 1, y_k + 1/2)\)
  - compute \( d_{k+1} = F(x_k + 1, y_k + 1/2) \)
  - \( x_{k+1} = x_k + 1 \)
  - if \( d_{k+1} > 0 \), \( y_{k+1} = y_k + 1 \)
  - otherwise, \( y_{k+1} = y_k \)
  - this works, but computing \( F(x_k + 1, y_k + 1/2) \) involves several floating-point additions & multiplications
  - we're trying to eliminate floating point!

Making It Incremental

• if \( d_{k+1} = d_k + \Delta \) we could save work
  \[ d_{k+1} = d_k + \Delta y \cdot \Delta x / 2 \]

• relationship between \( y_k \) and \( y_{k+1} \) depends on whether we chose NE or E pixel for \((x_k, y_k)\)

The Incremental Approach

• to compute \((x_{k+1}, y_{k+1})\)...
  - assume \( d_{k+1} \) has already been computed
  - \( x_{k+1} = x_k + 1 \)
  - if \( d_{k+1} > 0 \), \( y_{k+1} = y_k + 1 \)
  - otherwise, \( y_{k+1} = y_k \) and \( d_{k+2} = d_{k+1} + \Delta \)

Initial Values

• but what are \( x_0, y_0, d_1 \)?
  - \( x_0 \) and \( y_0 \) are the coordinates of the lower left endpoint
  \[ d_1 = F(x_0 + 1, y_0 + 1/2) \]
  \[ = (\Delta y) (x_0 + 1) - (\Delta x) (y_0 + 1/2) + (\Delta) B \]
  \[ = (\Delta y) x_0 - (\Delta x) y_0 + (\Delta) B + (\Delta) y \Delta x / 2 \]
  \[ = F(x_0, y_0) + (\Delta) y \Delta x / 2 \]
  \( (x_0, y_0) \) is an endpoint and thus on the line, so \( F(x_0, y_0) = 0 \)
  \[ d_1 = \Delta y \Delta x / 2 \]
  - but \( \Delta x / 2 \) may not be integer!
Making It Integer

\[ d_1 = \Delta y - \Delta x / 2 \]

- \( d_1 \) may not be integer
  - so, multiply by 2! (\( \Delta y \) and \( \Delta x \) are integers)
  - this doesn’t affect the sign of \( d \), which is all that matters

- we now use \( d_{k+1} = 2F(x_{k+1}, y_{k+1/2}) \)
  - so \( d_1 = 2\Delta y - \Delta x \)
  - this also means that \( \Delta_E \) and \( \Delta_{NE} \) are twice as big: \( \Delta_E = 2\Delta y \) and \( \Delta_{NE} = 2\Delta y - 2\Delta x \)

\[ 0 < m < 1 \text{ Midpoint Line Algorithm} \]

- compute \( 2\Delta x, 2\Delta y, \text{ and } 2\Delta y - 2\Delta x \)
- initialize \((x,y)\) to the lower leftmost point and \( d = 2\Delta y - \Delta x \)
- color pixel \((x,y)\)
- repeat...
  - increment \( x \)
  - if \( d > 0 \), increment \( y \) and add \( \Delta_{NE} = 2\Delta y - 2\Delta x \) to \( d \)
  - otherwise, add \( \Delta_E = 2\Delta y \) to \( d \)
  - color pixel \((x,y)\)

Midpoint Line Summary

- algorithm is integer & incremental (for speed)
- simplifications
  - one pixel per column (a hack to make it easier)
  - only consider \( 0 < m < 1 \) (other cases by symmetry)
- key observation: given pixel colored in column \( k \), next pixel is either \( E \) or \( NE \)
- need a decision variable to use to make choice

Midpoint Line Summary #2

- decision variable
  - use sign (+, -, 0) to make choice
  - the sign of \( F(x,y) \) tells which side of the line point \((x,y)\) is on
  - determine what side of line midpoint is on
    - if line is between midpoint and \( E \), \( E \) is closer to line i.e., choose \( E \) if \( F(x_m, y_m) < 0 \)
    - ditto for \( NE \) i.e., choose \( NE \) if \( F(x_m, y_m) > 0 \)
- making it integer & incremental
  - use \( 2F(x,y) \) as decision var instead of \( F(x,y) \)
  - don’t recompute \( F(x,y) \) each time - instead, increment previous value by some amount