Line Drawing Algorithms

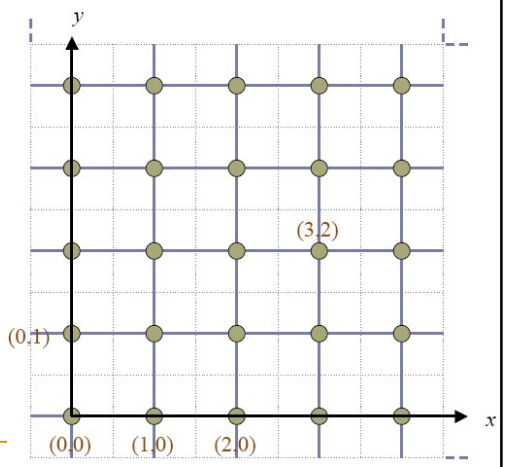
Digital Differential Analyzer (DDA)

Line Drawing

- Scan-conversion or *rasterization*: Determining which pixels to illuminate - due to the scanning nature of raster displays.
  - Convert specification of a primitive into pixels in the frame buffer
  - Process of taking high-level information such as the positions and colors of vertices and determining the colors of many pixels in a region of the frame buffer
- Algorithms are fundamental to both 2-D and 3-D computer graphics.
- Most incremental line-drawing (and other scan-conversion) algorithms were first developed for pen plotters.

Screen Coordinates

- We will use the following 2-D screen coordinate API:
- Pixel centers are at integer coordinates.
- The *y*-axis points up, *x*-axis points right

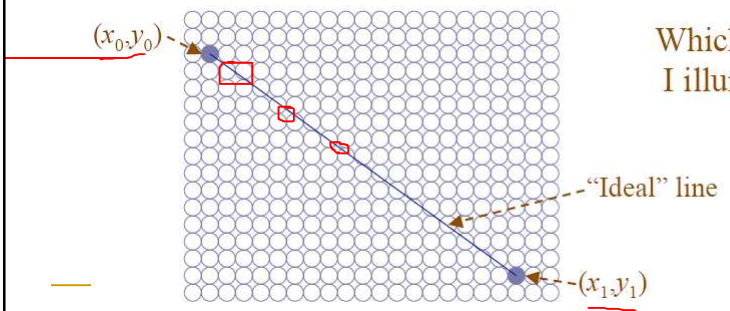


3

3

Goal

- Given integer descriptions of the endpoints of a line, produce a rasterized line
- The line may not fall on the discretized pixels (Usually won't, so pick nearest pixels to illuminate).

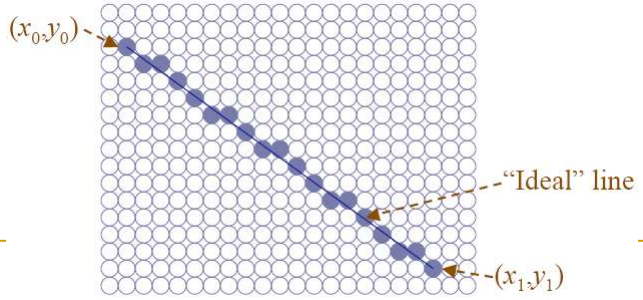


4

4

The Ideal Line

- Important qualities for a line:
  - Continuous appearance (no breaks, diagonal OK)
  - Uniform thickness and brightness
  - Accuracy (illuminate pixels nearest the ideal line)
  - Speed (how fast is the line drawn)
  - Consistency (drawn the same in both directions)



5

5

The Line

- Pixel provides a basis for all the raster graphics techniques
- A line segment is defined by the two endpoints
- A line is displayed by scan converting two endpoints into pixels on the line from one endpoint to the other
- Line Drawing Algorithms:
  - Digital Differential Analyzer (DDA) algorithm
  - Bresenham's line algorithm
  - Midpoint line algorithm

6

6

Digital Differential Analyzer (DDA) algorithm

- The Digital Differential Analyzer (DDA) algorithm is a line drawing algorithm used in computer graphics to rasterize a straight line on a display.
- It works by calculating the coordinates of points along the line at equal intervals, which can then be plotted on the display.

DDA Algorithm

Input the two endpoints of the line segment, (x1,y1) and (x2,y2).

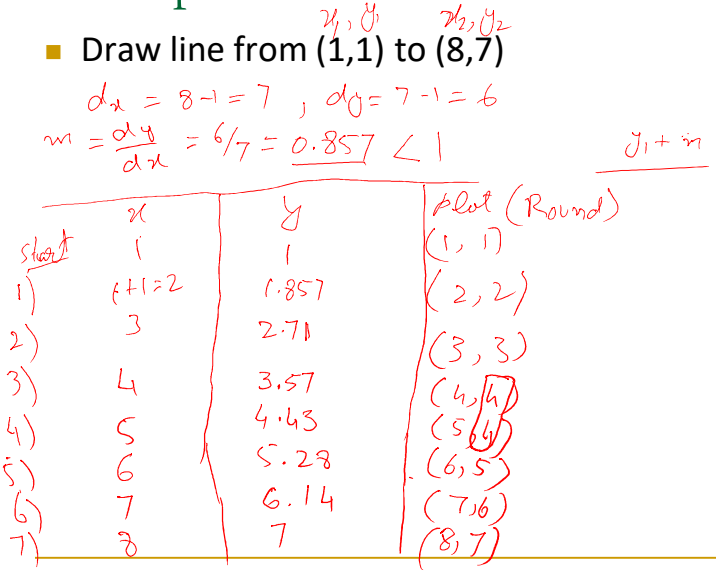
1. Calculate the difference between the x-coordinates and y-coordinates of the endpoints as dx and dy respectively.
2. Calculate the slope of the line as m = dy/dx.
3. Set the initial point of the line as (x1,y1).
4. If m<1 then
   - Loop through the x-coordinates of the line, incrementing by one each time, and calculate the corresponding y-coordinate using the equation $y = y1 + m(x - x1)$.

   If m>1 then
   - The y-coordinate is incremented by 1 at each step, and the x-coordinate is calculated as: x = x0 + (y - y0)/m

   If m=1 then
   - Increment both x and y as: x=x+1, y=y+1
5. Plot the pixel at the calculated (x,y) coordinate.
6. Repeat steps 4 and 5 until the endpoint (x2,y2) is reached.

Example

- Draw line from (1,1) to (8,7)



Example2 (m=1)

- Draw line from (1,1) to (4,4)
  - dy=4-1=3=dx
  - m=1
  - x=x+1, y=y+1
  - (1,1), (2,2), (3,3), (4,4)

Example2 (m>1)

- Draw line from (1,1) to (4,7)
  - dy=7-1=6, dx=4-1=3
  - m=2
    - m>1, y-coordinate is incremented, and the x-coordinate is calculated x=x+1/m
  - x=1+1/m=1+0.5 …..
  - (1,1), (1.5,2),(2,3),(2.5,4),(3,5),(3.5,6),(4,7)
  - After round off
  - (1,1), (2,2), (2,3),(3,4), (3,5), (4,6), (4,7)