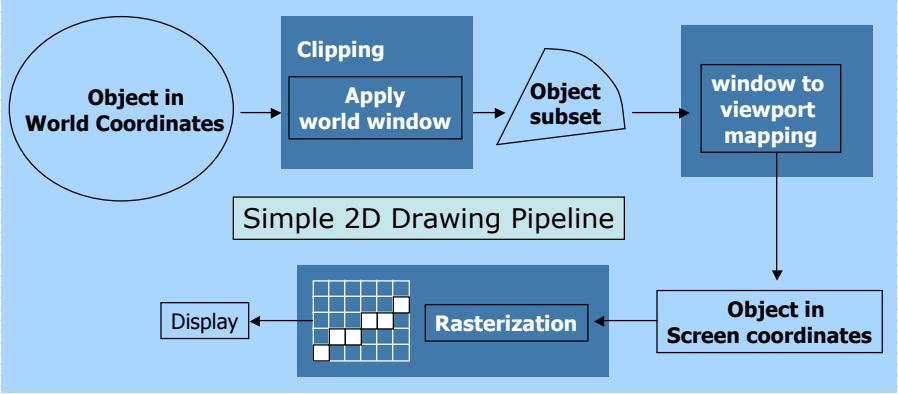
CS 543 - Computer Graphics: Raster Graphics, Part 1

by
Robert W. Lindeman
gogo@wpi.edu
(with help from Emmanuel Agu ;-)

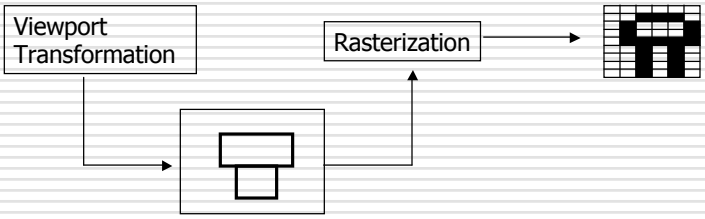
2D Graphics Pipeline

- Simplified



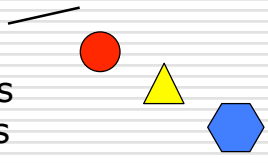
Rasterization (Scan Conversion)

- Convert high-level geometry description to pixel colors in the frame buffer
- Example: given vertex x, y coordinates, determine pixel colors to draw line
- Two ways to create an image
  - Scan existing photograph
  - Procedurally compute values (rendering)



Rasterization

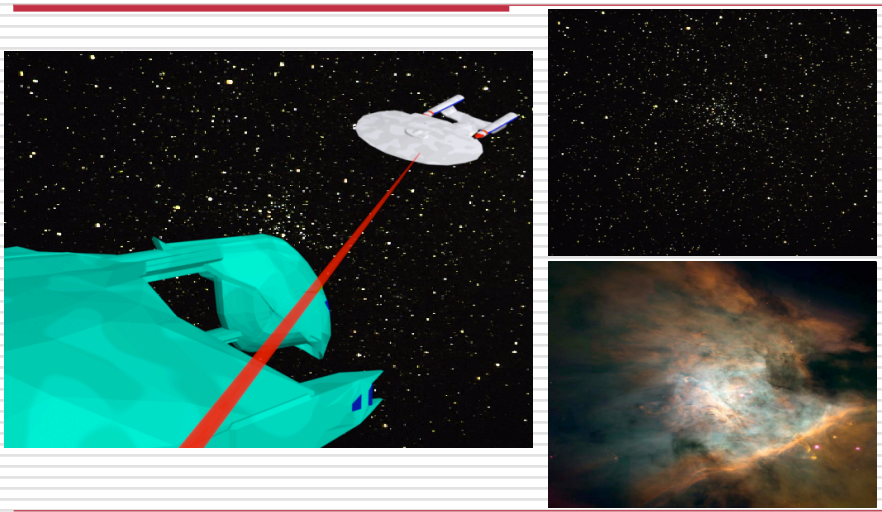
- A fundamental computer graphics function
- Determine the pixels' colors, illuminations, textures, *etc.*
- Implemented by graphics hardware
- Rasterization algorithms
  - Lines
  - Circles
  - Triangles
  - Polygons



Rasterization Operations

- Drawing lines on the screen
- Manipulating pixel maps (pixmaps): copying, scaling, rotating, *etc.*
- Compositing images, defining and modifying regions
- Drawing and filling polygons
  - Previously glBegin(GL_POLYGON), etc
- Aliasing and antialiasing methods

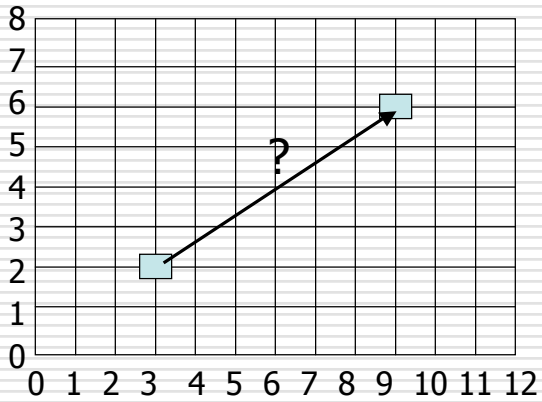
Compositing Example



Line Drawing Algorithm

- Programmer specifies (x, y) values of end pixels
- Need algorithm to figure out which intermediate pixels are on line path
- Pixel (x, y) values constrained to integer values
- Actual computed intermediate line values may be floats
- Rounding may be required, *e.g.*, computed point (10.48, 20.51) rounded to (10, 21)
- Rounded pixel value is off actual line path (jaggy!!)
- Sloped lines end up having jaggies, but vertical, horizontal lines don't

Line Drawing Algorithm (cont.)

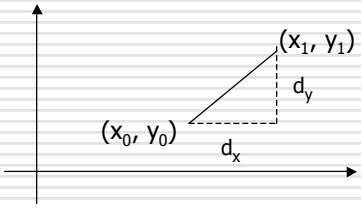


Line: (3,2) -> (9,6)

Which intermediate pixels should we light?

Line Drawing Algorithm (cont.)

- Slope-intercept line equation
  - $y = mx + b$
  - Given two end points (x_0, y_0), (x_1, y_1), how do we compute m and b?



$$m = \frac{d_y}{d_x} = \frac{y_1 - y_0}{x_1 - x_0}$$

$$b = y_0 - m * x_0$$

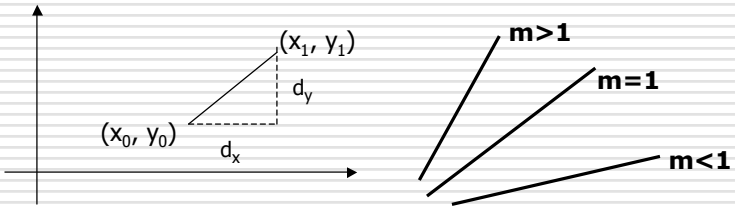
Line Drawing Algorithm (cont.)

- Numerical example of finding slope m:
- $(A_x, A_y) = (23, 41)$, $(B_x, B_y) = (125, 96)$

$$m = \frac{B_y - A_y}{B_x - A_x} = \frac{96 - 41}{125 - 23} = \frac{55}{102} = 0.5392$$

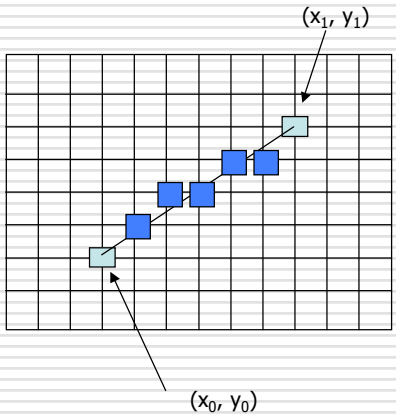
Line Drawing Algorithm: Digital Differential Analyzer (DDA)

- Walk through the line, starting at (x_0,y_0)
- Constrain x, y increments to values in [0,1] range
- Case a: x is incrementing faster (m < 1)
  - Step in x=1 increments, compute and round y
- Case b: y is incrementing faster (m > 1)
  - Step in y=1 increments, compute and round x



DDA Line Drawing Algorithm (Case a: m < 1)

$$y_{k+1} = y_k + m$$



x = x_0 y = y_0

Illuminate pixel (x, round(y))

x = x_0 + 1 y = y_0 + m

Illuminate pixel (x, round(y))

x = x + 1 y = y + m

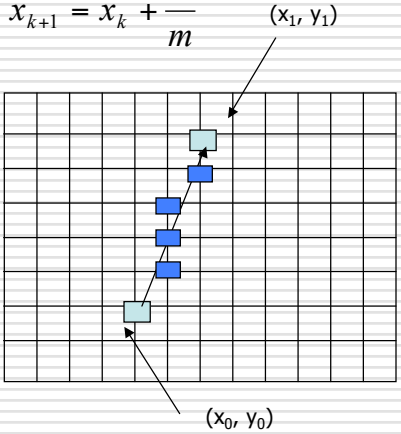
Illuminate pixel (x, round(y))

...

Until x == x_1

DDA Line Drawing Algorithm (Case b: m > 1)

$$x_{k+1} = x_k + \frac{1}{m}$$



x = x_0 y = y_0

Illuminate pixel (round(x), y)

y = y_0 + 1 x = x_0 + 1/m

Illuminate pixel (round(x), y)

y = y + 1 x = x + 1/m

Illuminate pixel (round(x), y)

...

Until y == y_1

DDA Line Drawing Algorithm Pseudocode

```
compute m;
if m < 1
  float y = y0;          // initial value
  for( int x = x0; x <= x1; x++, y += m )
    setPixel( x, round(y) );
else // m > 1
  float x = x0;          // initial value
  for( int y = y0; y <= y1; y++, x += 1/m )
    setPixel( round(x), y );
```

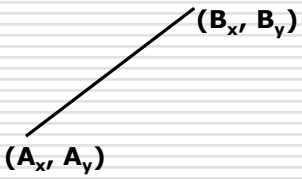
- Note: `setPixel(x, y)` writes current color into pixel in column x and row y in frame buffer

Line Drawing Algorithm Drawbacks

- DDA is the simplest line drawing algorithm
  - Not very efficient
  - Round operation is expensive
- Optimized algorithms typically used
  - Integer DDA
    - e.g., Bresenham's algorithm (Hill, 9.4.1)
- Bresenham's algorithm
  - Incremental algorithm: current value uses previous value
  - Integers only: avoid floating point arithmetic
  - Several versions of algorithm: we'll describe midpoint version of algorithm

Bresenham's Line-Drawing Algorithm

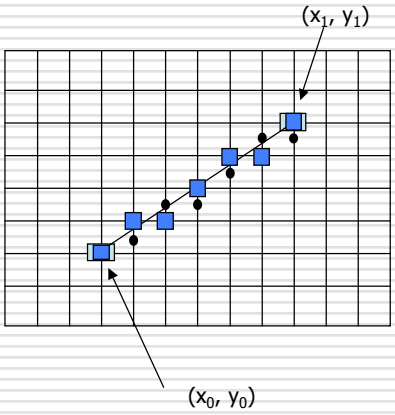
- Problem
  - Given endpoints (A_x, A_y) and (B_x, B_y) of a line, want to determine best sequence of intervening pixels
- First make two simplifying assumptions (remove later):
  - $(A_x < B_x)$ and
  - $(0 < m < 1)$
- Define
  - Width $W = B_x - A_x$
  - Height $H = B_y - A_y$



Bresenham's Line-Drawing Algorithm (cont.)

- Based on assumptions
  - W, H are positive
  - H < W
- As x steps in +1 increments, y incr/decr by <= +/−1
- y value sometimes stays same, sometimes increases by 1
  - Midpoint algorithm determines which happens

Bresenham's Line-Drawing Algorithm (cont.)



What Pixels do we need to turn on?

Consider pixel midpoint $M(M_x, M_y)$

$M = (x_0 + 1, Y_0 + ½)$

Build equation of line through and compare to midpoint

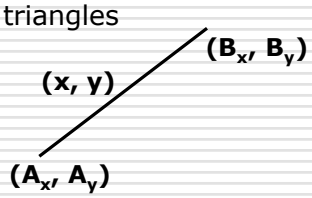
If midpoint is above line, y stays same
If midpoint is below line, y increases + 1

...

Bresenham's Line-Drawing Algorithm (cont.)

- To get a good line equation, use similar triangles

$$\frac{y - Ay}{x - Ax} = \frac{H}{W}$$



$$H(x - A_x) = W(y - A_y)$$
$$-W(y - A_y) + H(x - A_x) = 0$$

- Above is ideal equation of line through (A_x, A_y) and (B_x, B_y)
- Thus, any point (x, y) that lies on ideal line makes eqn = 0
- Double expression (to avoid floats later), and give it a name,

$$F(x, y) = -2W(y - A_y) + 2H(x - A_x)$$

Bresenham's Line-Drawing Algorithm (cont.)

- So, $F(x, y) = -2W(y - A_y) + 2H(x - A_x)$
- Algorithm
  - If:
    - $F(x, y) < 0$, (x, y) above line
    - $F(x, y) > 0$, (x, y) below line
- Hint: $F(x, y) = 0$ is on line
- Increase y keeping x constant, F(x, y) becomes more negative

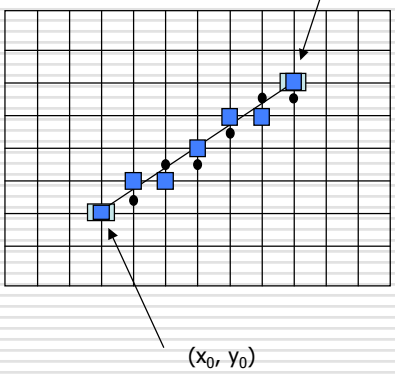
Bresenham's Line-Drawing Algorithm (cont.)

- Example
  - To find line segment between (3, 7) and (9, 11)

$$F(x,y) = -2W(y - A_y) + 2H(x - A_x)$$
$$= (-12)(y - 7) + (8)(x - 3)$$

- For points on line, e.g., (7, 29/3), F(x, y) = 0
- A = (4, 4) lies below line since F = 44 (> 0)
- B = (5, 9) lies above line since F = -8 (< 0)

Bresenham's Line-Drawing Algorithm (cont.)



What Pixels do we need to turn on?

Consider pixel midpoint $M(M_x, M_y)$

$M = (x_0 + 1, Y_0 + ½)$

If $F(M_x, M_y) < 0$, M lies above line, shade lower pixel (same y as before)

If $F(M_x, M_y) > 0$, M lies below line, shade upper pixel (y = y + 1)

...

Bresenham's Line-Drawing Algorithm (cont.)

- We can compute F(x, y) incrementally
  - Initially, midpoint M = (A_x + 1, A_y + ½)
    - $F(M_x, M_y) = -2W(y - A_y) + 2H(x - A_x)$
      $= 2H - W$
  - Can compute F(x, y) for next midpoint incrementally
  - If we increment x + 1, y stays same
    $F(M_x, M_y)$ += 2H
  - If we increment x +1, y + 1
    $F(M_x, M_y)$ += 2(W – H)

Bresenham's Line-Drawing Algorithm (cont.)

```
Bresenham( IntPoint a, InPoint b )  {
  // restriction: a.x < b.x and 0 < H/W < 1
  int y = a.y, W = b.x – a.x, H = b.y – a.y;
  int F = 2 * H – W;   // current error term
  for(int x = a.x; x <= b.x; x++)  {
    setPixel at (x, y); // to desired color value
    if F < 0
      F += 2H;
    else  {
      y++;
      F += 2(H – W)
    }
  }
}
```

- Recall: F is the equation of a line

Bresenham's Line-Drawing Algorithm (cont.)

- Final words: we developed algorithm with restrictions

  $0 < m < 1$ and $Ax < Bx$

- Can add code to remove restrictions
  - To get the same line when $A_x > B_x$ (swap and draw)
  - Lines having $m > 1$ (interchange x with y)
  - Lines with $m < 0$ (step x++, decrement y not incr)
  - Horizontal and vertical lines (pretest a.x = b.x and skip other tests)
- Important: **Read Hill 9.4.1**

References

- Hill, Chapter 9