
Hong Qin
Department of Computer Science
Stony Brook University (SUNY at Stony Brook)
Stony Brook, New York 11794-2424
Tel: (631)632-8450; Fax: (631)632-8334
qin@cs.stonybrook.edu
http://www.cs.stonybrook.edu/~qin
Rasterization

Per-pixel operations: ray-casting/ray-tracing

Scan conversion of lines:
- naive version
- Bresenham algorithm
  (mid-point algorithm)

Scan conversion of polygons

Aliasing / antialiasing

Texturing
Drawing of Line Geometry

- Why line drawing – the line is the most fundamental drawing primitive with many uses
  - Charts, engineering drawings, illustrations, 2D pencil-based animation, curve approximation

- Some desirable properties for any line drawing algorithm
  - A line should be straight; endpoint interpolation; uniform density for all lines; efficient

- Our current goal – efficient and correct line drawing algorithm

- Draw-line($x_0$, $y_0$, $x_1$, $y_1$)
Line Drawing

• Convert a continuous line to a set of discretized points
• Rasterization
Algorithm Assumption

- Point samples on 2D integer lattice
- Bi-level display: on or off
- Line endpoints are all integer coordinates
- All line slopes are: $|k| \leq 1$
- Lines are ONE pixel thick
- Are the above assumptions reasonable?
Line Geometry

• Explicit representation

• \( y = mx + b \)

• The geometric meanings of these parameters: \( m \) – slope of the line; \( b \) – where it intercept y-axis (where \( x = 0 \))

• More derivations
  
  \[ \begin{align*}
  \text{– } & dy = y_1 - y_0 \\
  \text{– } & dx = x_1 - x_0 \\
  \text{– } & m = \frac{dy}{dx}
  \end{align*} \]
Simple Algorithm

• Draw-line(x0, y0, x1, y1)

1. Let $dy = y1 - y0; dx = x1 - x0$
2. For $x = x0$ to $x1$
3. $y = \text{rounding-operation}(y0 + (x - x0) (dy / dx))$
4. draw-point(x, y)
5. End for

• Why does the above procedure work?

• Explicit definition of the line geometry

$$y = \frac{dy}{dx} (x - x0) + y0 = mx + b$$
Rendering Line Segments (Rasterization)

- One of the fundamental tasks in 2D computer graphics is 2D line drawing: How to render a line segment from \((x_1, y_1)\) to \((x_2, y_2)\)?

- Use the equation
  \[ y = mx + h \text{ (explicit)} \]

- What about horizontal vs. vertical lines?
Further Improvement

- A more efficient algorithm
  1. $x = x_0; y = y_0$
  2. `draw-point(x,y)`
  3. For $x$ from $x_0 + 1$ to $x_1$
  4. $y = y + \left(\frac{dy}{dx}\right)$
  5. End for

- Note that, $m = \left(\frac{dy}{dx}\right)$, and $m$ is a float or double
DDA Algorithm

- Digital Differential Analyzer (DDA)
  
  \[
  \text{for } (x=x_1; x \leq x_2; x++)
  \]
  
  \[
  y += m;
  \]
  
  \[
  \text{draw\_pixel}(x, y, \text{color})
  \]

- Handle slopes \(0 \leq m \leq 1\); handle others symmetrically

- Does this need floating point operations?
Further Improvement

• We are now seeking an integer-ONLY algorithm to handle all line geometry
• The above procedures will fail
• We must explore new schemes (beyond the line geometry we have already know till now)
Implicit Equation

\[
f(x,y) < 0
\]

\[
f(x,y) = 0
\]

\[
f(x,y) > 0
\]
Midpoint Algorithm

• Implicit expression for the line geometry
  \[ f(x,y) = (x - x_0)(dy) - (y - y_0)(dx) \]

• What does this formulation provide us (compared with the previous derivations)?

• Fundamental ideas – spatial partitioning based on the signs!
  – If \( f(x,y) = 0 \), then \((x, y)\) is on the line
  – If \( f(x,y) > 0 \), then \((x,y)\) is below the line
  – If \( f(x,y) < 0 \), then \((x, y)\) is above the line
Midpoint Motivation
Midpoint Motivation

• We are actually considering \( d = f(x_p + 1, y_p + 0.5) \)

• There are three different cases
  – If \( d < 0 \), line is below the (current) midpoint, then choose E
  – If \( d > 0 \), line is above the midpoint, choose NE
  – If \( d = 0 \), line is passing through the midpoint, either E or NE
Recursive Algorithm

- Midpoint algorithm is a recursive algorithm!
- For any recursive algorithm, we MUST consider the subsequent steps (by traversing all cases respectively)!
- If E is chosen, then the NEW E is (xp + 2, yp), the NEW NE is (xp + 2, yp +1), the NEW midpoint is (xp + 2, yp + 0.5)
  - d_new = f(xp + 2, yp + 0.5)
  - d_old = f(xp + 1, yp +0.5)
  - d_new = d_old + (dy)
Recursive Algorithm

• If NE is chosen, the NEW E is \((xp + 2, yp + 1)\), the NEW NE is \((xp + 2, yp + 2)\), the NEW midpoint is \((xp + 2, y + 1.5)\)
  - \(d_{\text{new}} = f(xp + 2, yp + 1.5)\)
  - \(d_{\text{old}} = f(xp + 1, yp + 0.5)\)
  - \(d_{\text{new}} = d_{\text{old}} + (dy - dx)\)

• This process MUST repeat recursively, stepping along \(x\) from \(x_0\) to \(x_1\)
Midpoint Initialization
Initialization

• How about the initialization process

• At the beginning,
  – $x_p = x_0$
  – $y_p = y_0$
  – $d_{\text{old}} = f(x_0 + 1, y_0 + 0.5) = (dy) - (dx) \times (1/2)$
Midpoint Algorithm

- **draw-line***(x₀, y₀, x₁, y₁)***
  - Int *x₀*, *y₀*, *x₁*, *y₁*
  - { int *dx*, *dy*, inc_E, inc_NE, *x, y*,
  - real *d*
  - *dx* = *x₁* − *x₀*
  - *dy* = *y₁* − *y₀*
  - *d* = (dy) − (dx) *·* (1/2)
  - inc_E = dy
  - inc_NE = dy − dx
  - *y* = *y₀*
  - for *x* from *x₀* to *x₁*
  - if *d* > 0, then *d* = *d* + inc_NE, *y* + 1, else *d* = *d* + inc_E
  - end for
  - }

CSE328 Lectures
Midpoint Algorithm

• **d** is **NOT** an integer, however, **ONLY the sign MATTERS!**

• **We prefer an integer-ONLY algorithm!!!**
  
  - \( g(x, y) = 2 \ f(x, y) \)
  
  - **d** becomes \( 2d \)
  
  - then \( d = 2(dy) - (dx) \)
Modifying the Previous Algorithm

• Make it an integer-ONLY algorithm

• Our earlier assumptions
  – slopes: $0 \leq \frac{dy}{dx} \leq 1$
  – line endpoints are all integer coordinates

• How about other cases
Handling All Other Cases

• **Generalizations**
  - negative slope
  - slope larger than 1

• **If the slope is larger than 1, we use symmetry to switch x and y (you are NOT displaying (x,y), you should display (y,x))!**

• **In negative slope, we should use x and (-y)**
Bresenham’s Algorithm

• The DDA algorithm requires a floating point *add* and *round* for each pixel: can we eliminate?

• Note that at each step we will go E or NE. How to decide which?
Bresenham Decision Variable

- Bresenham algorithm uses decision variable $d = a - b$, where $a$ and $b$ are distances to NE and E pixels.

- If $d \geq 0$, go NE; if $d < 0$, go E.

- Let $d = (x_2 - x_1)(a - b) = d_x(a - b)$ [only sign matters].

- Substitute for $a$ and $b$ using line equation to get integer math (but lots of it). 

$$d = (a - b) d_x = (2j + 3) d_x - (2i + 3) d_y - 2(y_1 d_x - x_1 d_y)$$

- But note that $d_{k+1} = d_k + 2d_y (E)$ or $2(d_y - d_x) (NE)$. 

![Diagram](image_url)
Bresenham’s Algorithm

• Set up loop computing $d$ at $x_1$, $y_1$

```c
for (x=x_1; x<=x_2; )
    x++;
    d += 2dy;
    if (d >= 0) {
        y++;
        d -= 2dx;
    }
    drawpoint(x, y);
```

• Pure integer math, and not much of it

• So easy that it is built into one graphics instruction (for several points in parallel)
Extensions to Handle Curves

- Generalizations to handle all cases for line drawing
- Algorithms for circle-drawing
- Algorithms for ellipses, conic section drawing
- Algorithms for cubic curve drawing
- Algorithms to handle any type of curves?
Circles

- Implicit expression of a circle $f(x,y)=0$
  
  \[ f(x, y) = (x - x_0)^2 + (y - y_0)^2 - r^2 \]

- Remember the key idea is that, ONLY the sign matters!
  - If $f(x,y)=0$, then $(x,y)$ is on the circle
  - If $f(x,y)>0$, then $(x,y)$ is outside the circle
  - If $f(x,y)<0$, then $(x,y)$ is inside the circle

- Equations for ellipses?

- The key message: the slope is controllable!!!