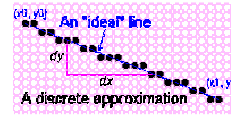


Computer Graphics

Lecture 3 Line & Circle Drawing

Towards the Ideal Line

- We can only do a discrete approximation



- Illuminate pixels as close to the true path as possible, consider bi-level display only
 - Pixels are either lit or not lit

What is an *ideal* line

- Must appear straight and continuous
 - Only possible axis-aligned and 45° lines
- Must interpolate both defining end points
- Must have uniform density and intensity
 - Consistent within a line and over all lines
 - What about antialiasing?
- Must be efficient, drawn quickly
 - Lots of them are required!!!

Simple Line

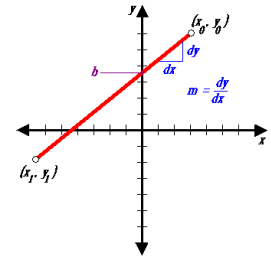
Based on *slope-intercept algorithm* from algebra:

$$y = mx + b$$

Simple approach:

increment x, solve for y

Floating point arithmetic required

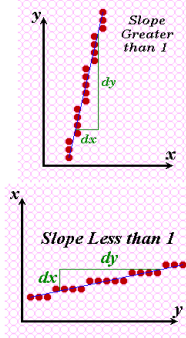


Does it Work?

It seems to work okay for lines with a slope of 1 or less,

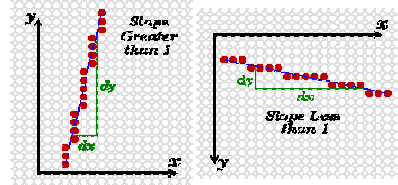
but doesn't work well for lines with slope greater than 1 – lines become more discontinuous in appearance and we must add more than 1 pixel per column to make it work.

Solution? - use *symmetry*.



Modify algorithm per octant

Rotate and Rename coordinate axes



OR, increment along x-axis if $dy < dx$ else increment along y-axis

Testing for the side of a line.

$$F(x, y) = ax + by + c = 0$$

- Need to find coefficients a,b,c.
- Recall explicit, slope-intercept form :

$$y = mx + b \text{ and so } y = \frac{dy}{dx}x + b$$

- So:

$$F(x, y) = dyx - dxy + c = 0$$

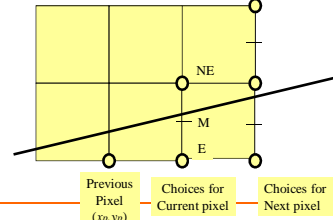
Decision variable.

Let's assume $dy/dx < 0.5$ (we can use symmetry)

Evaluate F at point M

$$d = F(x_p + 1, y_p + \frac{1}{2})$$

Referred to as *decision variable*

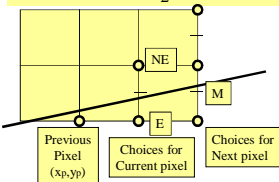


Decision variable.

Evaluate d for next pixel, Depends on whether E or NE is chosen :

If E chosen :

$$d_{new} = F(x_p + 2, y_p + \frac{1}{2}) = a(x_p + 2) + b(y_p + \frac{1}{2}) + c$$



But recall :

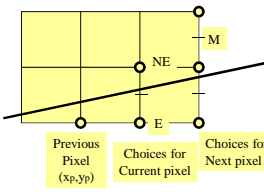
$$d_{old} = F(x_p + 1, y_p + \frac{1}{2}) = a(x_p + 1) + b(y_p + \frac{1}{2}) + c$$

$$\text{So : } d_{new} = d_{old} + a = d_{old} + dy$$

Decision variable.

If NE was chosen :

$$d_{new} = F(x_p + 2, y_p + \frac{3}{2}) = a(x_p + 2) + b(y_p + \frac{3}{2}) + c$$



So :

$$d_{new} = d_{old} + a + b = d_{old} + dy - dx$$

Summary of mid-point algorithm

- Choose between 2 pixels at each step based upon the sign of a decision variable.
- Update the decision variable based upon which pixel is chosen.
- Start point is simply first endpoint (x_1, y_1) .
- Need to calculate the initial value for d

Initial value of d .

Start point is (x_1, y_1)

$$\begin{aligned} d_{start} &= F(x_1 + 1, y_1 + \frac{1}{2}) = a(x_1 + 1) + b(y_1 + \frac{1}{2}) + c \\ &= ax_1 + by_1 + c + a + \frac{b}{2} \\ &= F(x_1, y_1) + a + \frac{b}{2} \end{aligned}$$

But (x_1, y_1) is a point on the line, so $F(x_1, y_1) = 0$

$$d_{start} = dy - dx / 2$$

Conventional to multiply by 2 to remove fraction \Rightarrow doesn't effect sign.

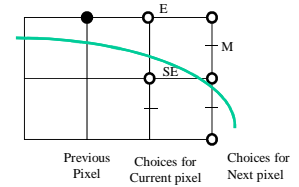
Midpoint algorithm

```

void MidpointLine(int
  x1,y1,x2,y2)
{
  int dx=x2-x1;
  int dy=y2-y1;
  int d=2*dy-dx;
  int increE=2*dy;
  int incrNE=2*(dy-dx);
  x=x1;
  y=y1;
  WritePixel(x,y);
  while (x < x2) {
    if (d <= 0) {
      d+=incrE;
      x++;
    } else {
      d+=incrNE;
      x++;
      y++;
    }
    WritePixel(x,y);
  }
}
    
```

Circle drawing.

- Can also use Bresenham to draw circles.
- Use 8-fold symmetry



Circle drawing.

- Implicit form for a circle is:

$$f(x, y) = (x - x_c)^2 + (y - y_c)^2 - r^2$$

If SE is chosen $d_{new} = d_{old} + (2x_p - 2y_p + 5)$

If E is chosen $d_{new} = d_{old} + (2x_p + 3)$

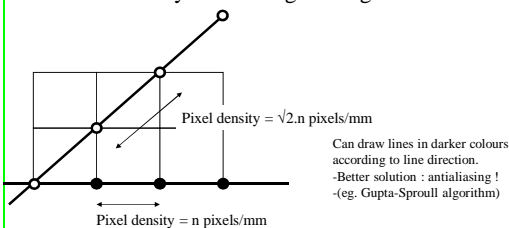
Functions are linear equations in terms of (x_p, y_p)
 -Termed *point of evaluation*

Summary of line drawing so far.

- Explicit form of line
 - Inefficient, difficult to control.
- Parametric form of line.
 - Express line in terms of parameter t
 - DDA algorithm
- Implicit form of line
 - Only need to test for 'side' of line.
 - Bresenham algorithm.
 - Can also draw circles.

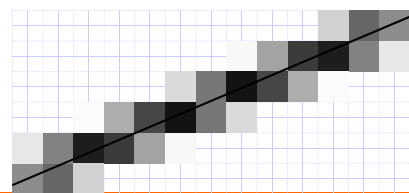
Problems with Bresenham algorithm

- Pixels are drawn as a single line \Rightarrow unequal line intensity with change in angle.



Gupta-Sproull algorithm.

- Calculate the distance of the line and the pixel center
- Adjust the colour according to the distance

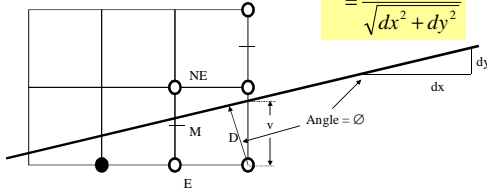


Gupta-Sproull algorithm.

Calculate distance using features of mid-point algorithm

$$D = v \cos \phi$$

$$= \frac{v dx}{\sqrt{dx^2 + dy^2}}$$



Gupta-Sproull algorithm (cont)

Recall from the midpoint algorithm:

$$F(x, y) = 2(ax + by + c) = 0 \text{ (doubled to avoid fraction)}$$

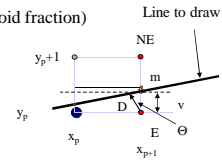
So
$$y = \frac{-(ax + c)}{-b}$$

For pixel E:

$$x_{p+1} = x_p + 1 \quad y_{p+1} = y_p \quad v = y - y_{p+1}$$

So:

$$v = \frac{a(x_p + 1) + c}{-b} - y_p$$



Gupta-Sproull algorithm (cont)

From previous slide:

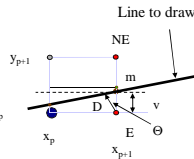
$$v = \frac{a(x_p + 1) + c}{-b} - y_p$$

From the midpoint computation,

$$b = -dx$$

So:

$$v dx = a(x_p + 1) + by_p + c = F(x_p + 1, y_p) / 2$$



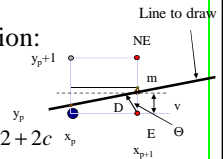
Gupta-Sproull algorithm (cont)

From the midpoint algorithm, we had the decision variable (remember?)

$$d = F(M) = F(x_p + 1, y_p + \frac{1}{2})$$

Going back to our previous equation:

$$\begin{aligned} 2v dx &= F(x_p + 1, y_p) \\ &= 2a(x_p + 1) + 2by_p + 2c \\ &= 2a(x_p + 1) + 2b(y_p + 1/2) - 2b/2 + 2c \\ &= F(x_p + 1, y_p + 1/2) - b \\ &= F(M) - b \\ &= d - b \\ &= d + dx \end{aligned}$$



Gupta-Sproull algorithm (cont)

So,

$$D = \frac{d + dx}{2\sqrt{dx^2 + dy^2}}$$

And the denominator is constant

Since we are blurring the line, we also need to compute the distances to points $y_p - 1$ and $y_p + 1$

$$\text{numerator for } y_p - 1 \Rightarrow 2(1 - v)dx = 2dx - 2vdx$$

$$\text{numerator for } y_p + 1 \Rightarrow 2(1 + v)dx = 2dx + 2vdx$$

Gupta-Sproull algorithm (cont)

If the NE pixel had been chosen:

$$\begin{aligned} 2v dx &= F(x_p + 1, y_p + 1) \\ &= 2a(x_p + 1) + 2(b y_p + 1) + 2c \\ &= 2a(x_p + 1) + 2b(y_p + 1/2) + 2b/2 + 2c \\ &= F(x_p + 1, y_p + 1/2) + b \\ &= F(M) + b \\ &= d + b \\ &= d - dx \end{aligned}$$

$$\text{numerator for } y_p + 2 \Rightarrow 2(1 - v)dx = 2dx - 2vdx$$

$$\text{numerator for } y_p \Rightarrow 2(1 + v)dx = 2dx + 2vdx$$

Gupta-Sproull algorithm (cont)

- Compute midpoint line algorithm, with the following alterations:
 - At each iteration of the algorithm:
 - If the E pixel is chosen, set numerator = $d + dx$
 - If the NE pixel is chosen, set numerator = $d - dx$
 - Update d as in the regular algorithm
 - Compute $D = \text{numerator}/\text{denominator}$
 - Color the current pixel according to D
 - Compute $D_{\text{upper}} = (2dx - 2vdx)/\text{denominator}$
 - Compute $D_{\text{lower}} = (2dx + 2vdx)/\text{denominator}$
 - Color upper and lower accordingly
-