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

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

- DDA = Digital Differential Analyser
  - finite differences
- Treat line as parametric equation in t:

\[
\begin{align*}
  x(t) &= x_1 + t(x_2 - x_1) \\
  y(t) &= y_1 + t(y_2 - y_1)
\end{align*}
\]

Start point - \((x_1, y_1)\)
End point - \((x_2, y_2)\)

DAA Algorithm

- Start at \(t = 0\)
- At each step, increment \(t\) by \(dt\)
- Choose appropriate value for \(dt\)
- Ensure no pixels are missed:
  - Implies: \(\frac{ds}{dt} < 1\) and \(\frac{dy}{dt} < 1\)
- Set \(dt\) to maximum of \(dx\) and \(dy\)

DDA algorithm

```c
line(int x1, int y1, int x2, int y2)
{
    float x, y;
    int dx = x2-x1, dy = y2-y1;
    int n = max(abs(dx),abs(dy));
    float dt = n, dxdt = dx/dt, dydt = dy/dt;
    x = x1;
    y = y1;
    while( n-- ) {
        point(round(x),round(y));
        x += dxdt;
        y += dydt;
    }
}
```

Observation on lines.

Testing for the side of a line.

- Need a test to determine which side of a line a pixel lies.
- Write the line in implicit form:

\[
F(x, y) = ax + by + c = 0
\]

- If \((b<0) F<0\) for points above the line, \(F>0\) for points below.
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:
  \[ y = mx + b \text{ and so } y = \frac{dy}{dx} + b \]
- So:
  \[ F(x, y) = dyx - dxy + c = 0 \]

**Decision variable.**

Let's assume \( \frac{dy}{dx} < 0.5 \) (we can use symmetry).

Evaluate \( F \) at point \( M \)

Referred to as decision variable

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

Start point is simply first endpoint \((x_1, y_1)\).

Need to calculate the initial value for \( d \)

**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 \((x, y)\).
- Need to calculate the initial value for \( d \)

Conventional to multiply by 2 to remove fraction \( \frac{1}{2} \) doesn’t effect sign.
Midpoint algorithm

```c
void MidpointLine(int x1, y1, x2, y2) {
    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 (cont)

Recall from the midpoint algorithm:
\[ F(x, y) = 2(ax + by + c) = 0 \]

For pixel E:
\[ x_{p+1} = x_p + 1 \quad y_{p+1} = y_p \quad v = y - y_{p+1} \]

So:
\[ y = \frac{ax_{p+1} + c}{-b} \]

Gupta-Sproull algorithm (cont)

From the midpoint algorithm, we had the decision variable (remember?)
\[ d = F(M) = F(x_{p+1}, y_{p+1}) \]

Going back to our previous equation:
\[ 2dx = F(x_p + 1, y_p) \]
\[ = 2a(x_p + 1) + 2by_p + 2c \]
\[ = 2a(x_p + 1) + 2b(y_p + 1/2) - 2b/2 + 2c \]
\[ = F(x_p + 1, y_p + 1/2) - b \]
\[ = F(M) - b \]
\[ = d - b \]
\[ = d + dx \]

Gupta-Sproull algorithm (cont)

If the NE pixel had been chosen:
\[ 2dx = F(x_p + 1, y_p + 1) \]
\[ = 2a(x_p + 1) + 2(by_p + 1) + 2c \]
\[ = 2a(x_p + 1) + 2b(y_p + 1/2) + 2b/2 + 2c \]
\[ = F(x_p + 1, y_p + 1/2) + b \]
\[ = F(M) + b \]
\[ = d + b \]
\[ = d + dx \]

And the denominator is constant
Since we are blurring the line, we also need to compute the distances to points \( y_p - 1 \) and \( y_p + 1 \)

numerator for \( y_p - 1 \) \( \Rightarrow 2(1 - v)dx = 2dx - 2dx \)

numerator for \( y_p + 1 \) \( \Rightarrow 2(1 + v)dx = 2dx + 2dx \)
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 = numerator/denominator
  - Color the current pixel according to D
  - Compute $D_{upper} = (2dx-2vdx)/denominator$
  - Compute $D_{lower} = (2dx+2vdx)/denominator$
  - Color upper and lower accordingly