Scan Conversion of Lines

Raster devices

Most devices that are used to produce images are raster devices, that is, use rectangular arrays of dots (pixels) to display the image. This includes CRT monitors, LCDs, lasers and dot-matrix printers.

Examples of non-raster output devices include vector displays (not used anymore) and plotters still widely used.

Scan conversion = converting a continuous object such as a line or a circle into discrete pixels

Scan conversion of lines

Given two points with integer coordinates \( p_1 = [x_1, y_1] \) and \( p_2 = [x_2, y_2] \), the algorithm has to find a sequence of pixels approximating the line.

Slope: \( \frac{y_2 - y_1}{x_2 - x_1} \)

We can always reorder \( p_1 \) and \( p_2 \) so that \( x_2 - x_1 \) is nonnegative. It is convenient to look at only nonnegative slopes; if the slope is negative, change the sign of \( y \).

Slope

Slope reduction: it is convenient to have the slope of the line between 0 and 1; then we are able to step along x axis.

To handle slope > 1, swap x and y

DDA

Assume that the slope is between 0 and 1

Simplest algorithm (pompously called differential digital analyzer):

Step along x, increment y by slope at each step.

Round y to nearest pixel.

```c
float y = y1;
float slope = (y2 - y1) / (float)(x2 - x1);
int x;
for(x = x1; x <= x2; x++) {
    drawpixel(x, floor(y));
    y += slope;
}
```

Bresenham Algorithm

What is wrong with DDA?

It requires floating-point operations.

These operations are expensive to implement in hardware.

They are not really necessary if the endpoints are integers.

Idea: instead of incrementing y and rounding it at each step, decide if we just go to the right, or to the right and up using only integer quantities.
Increment decision

- Pixel corners on different sides of the line: increment both x and y.
- Pixel corners on the same side of the line: increment only x.

Need: fast way to determine on which side of a line a point is.

Half-plane test

Implicit equation can be used to perform the test.

\[(n, q - p) > 0\] the point on the same side with the normal
\[(n, q - p) < 0\] the point on the other side

Implicit line equation

The implicit equation of the line through \(p_1 = [x_1, y_1]\) and \(p_2 = [x_2, y_2]\) is

\[(n, q - p_1) = 0, \text{ with } n = [y_2 - y_1, x_2 - x_1]\]

We need to test on which side of the line is the point \(q + d_1 = [x, y] + [1/2, 1/2]\)

To do this, we need to determine the sign of \(F = (n, 2q + 2d_1 - 2p_1)\)
Note that multiplication by two makes everything integer again!
Key idea: compute this quantity incrementally.

Incremental computation

At each step \(q = [x, y]\) changes either to \([x+1, y]\) (step to the right) or to \([x+1, y+1]\) (step to the right and up); in vector form, the new value of \(q\) is either \(q + D_1\) or \(q + D_2\) with \(D_1 = [1,0]\) and \(D_2 = [1,1]\)

\[F_{next} = (n, 2q + 2D + 2d_1 - 2p_1) = (n, 2q + 2d_1 - 2p_1) + 2(n, D)\]
\[F = F + 2(n, D), \text{ where } D \text{ is } D_1 \text{ or } D_2\]
At each step, to get new \(F\) we have to increment old \(F\) either by \((n, D_1)\) or \((n, D_2)\)
\[(n, D_1) = y_2 - y_1\]
\[(n, D_2) = (y_2 - y_1) - (x_2 - x_1)\]

Bresenham algorithm

Assume the slope to be between 0 and 1.

```c
int y = y1; int dy = y2-y1;
int dxdy = y2-y1+x1-x2;
int F = y2-y1+x1-x2; int x;
for ( x = x1; x <= x2; x++ ) {
    drawpixel(x,y);
    if ( F < 0 ) {
        y++; F += dxdy;
    } else {
        F += dy;
    }
}
```

Bresenham algorithm

In your implementation you need to handle all slopes!
First, reorder endpoints so that \(x_1 \leq x_2\)
Then consider 4 cases:
\[y_2-y_1 > 0, \ x_2-x_1 > y_2-y_1, \ \text{positive slope} \leq 1\]
\[y_2-y_1 > 0, \ x_2-x_1 < y_2-y_1, \ \text{positive slope} > 1\]
\[y_2-y_1 < 0, \ x_2-x_1 > y_1-y_2, \ \text{negative slope} \geq -1\]
\[y_2-y_1 < 0, \ x_2-x_1 < y_1-y_2, \ \text{negative slope} < -1\]
In each case, make appropriate substitutions in the algorithm.