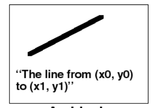
Scan conversion

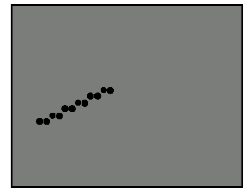
Problem

...to this.

From this...



An Ideal Geometric Primitive



A Bag of Pixels

Drawing on a raster

- Have to represent continuous shapes on a fixed resolution grid
- primitives: lines, polygons, curves
- antialiasing: use changes in intensity to compensate for resolution
- algorithms have to be fast
- algorithms have to be consistent

Primitives

Curves:

	implicit	parametric
general	$f(x, y) = 0$	$(x(t), y(t))$
line	$f(x, y) = ax + by - c$	$x(t) = s_x + t(e_x - s_x)$ $y(t) = s_y + t(e_y - s_y)$
circle	$f(x, y) = (x - c_x)^2 + (y - c_y)^2 - r^2$	$x(t) = c_x + r\cos(2\pi t)$ $y(t) = c_y + r\sin(2\pi t)$

Primitives

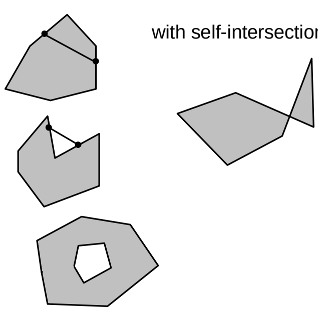
Polygons:

convex

with self-intersections

non-convex

with holes



DDA

- Take unit steps in *x*, size *m* steps in *y*. Round *y* to the nearest pixel. (Reflect if slope > 1.)

```
void line (int x0, int y0, int x1, int y1){
    float y = y0;
    float slope = (y1 - y0)/ (float) (x1 - x0);
    int x;
    for(x=x0;x<=x1;x++){
        draw_pixel(x,Round(y));
        y += slope;
    }
}
```

- This algorithm goes by the grandiose name of *Differential Digital Analyzer* (DDA.)
- Problem: Round and floating-point add are expensive (esp. if you're planning to build custom pixel-painting engines!)

Bresenham algorithm

```
/* this does the right thing (same as DDA) at */
/* a cost of 2 or 3 integer adds per point.   */
/* (assumes sorted endpoints, 0<slope<1)      */

void draw_line(int x0, int y0, int x1, int y1)
    int x,y=y0;dx = 2*(x1-x0),dy = 2*(y1-y0)
    int dydx = dy - dx, d = dy - dx/2;

    for(x=x0;x<=x1;x++){
        draw_pixel(x,y);
        if(d<0) {d+= dy;}
        else    {y+=1; d += dydx;}
    }

}
```

Line $\mathbf{L}$ *from* $[x_0, y_0]$ *to* $[x_1, y_1]$.

$\mathbf{P}_0 = [x_0, y_0]$,

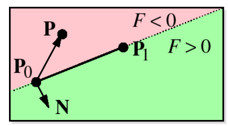
$\mathbf{P}_1 = [x_1, y_1]$.

$dx = x_1 - x_0,\ dy = y_1 - y_0$

$\mathbf{N} = [dy, -dx]$

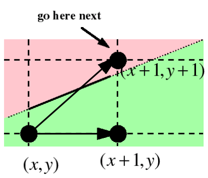
implicit function : $F(\mathbf{P}) = 2\mathbf{N} \cdot (\mathbf{P} - \mathbf{P}_0)$

$F = 0 \rightarrow \mathbf{P}$ *is on* $\mathbf{L}$



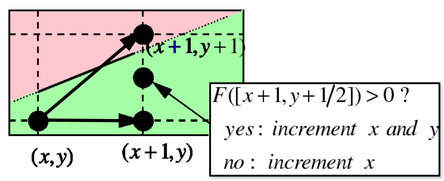
Why the factor of 2? Because we're going to divide by 2 later.

Which pixel is next?



- **Assume $0 < \text{slope} < 1$, sorted endpoints.**
- **At each step:**
  - We're at *(x,y)*.
  - Should we increment both x and y, or only x?
  - Answer: pixel that's closest to the actual line.
- **We can use the implicit function to decide…**

Using ther implicit equation



NOTE: Blinn's law of sign flips…

Incremental update for F

$$F(\mathbf{P}) = 2\mathbf{N} \cdot (\mathbf{P} - \mathbf{P}_0)$$

$$F(\mathbf{P} + \Delta) = 2\mathbf{N} \cdot (\mathbf{P} + \Delta - \mathbf{P}_0)$$

$$= F(\mathbf{P}) + 2\mathbf{N} \cdot \Delta$$

- **Computing $F(\mathbf{P})$ from scratch requires a dot product, but computing a new value given the old one and a position offset only requires an add. (an *integer add,* for us.)**
- **The thing you add in doesn't depend on P so you can precompute and re-use it.**

Bresenham Algorithm

$F_0 = F(\mathbf{P}_0 + [1, 1/2])$

$\quad = F(\mathbf{P}_0) + \mathbf{N} \cdot [2, 1]$

$F' = F + 2\mathbf{N} \cdot \Delta$

where

$\Delta = [1,0]$ *or* $[1,1]$

$\mathbf{N} = [dy, -dx]$

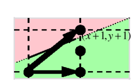
so

$F' = F + 2dy$

or

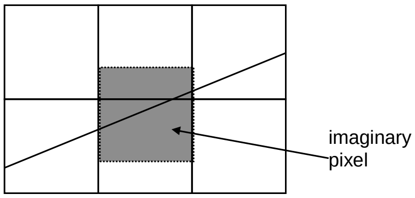
$F' = F + 2dy - 2dx$

- **Initialize x,y,F**
- **Eval F one over and 1/2 up.**
- **increment x.**
- **if F>0, increment y, too.**
- **increment F using the appropriate constant (integer) update.**
- **Oh, yeah. Draw the pixel.**
- **Continue to end of line.**



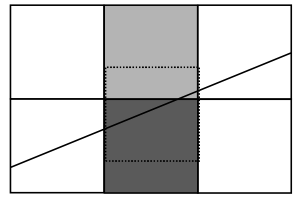
Anti-aliasing

Instead of binary decision, adjust intensity proportoinal to the area:



Anti-aliasing

Instead of binary decision, adjust intensity proportoinal to the area:

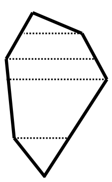


Scan Conversion of Convex Polygons

General idea:

decompose polygon into tiles

scan convert each tile, moving along one edge



Convex Polygons

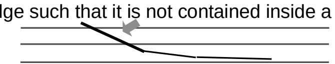
Scan convert a convex polygon:

```
ScanY( Vertex v[], int nv, int bot )
```

array of vertices in counterclockwise order — array size — number of the vertex with min. y coordinate

1. Find left edge of a tile:

•go around **clockwise,** starting from v[bot], until find an edge such that it is not contained inside a scan line:



2. Similarly, find the right edge of a tile.
3. Scan convert all scan lines going from left to right edges

Convex Polygons

```
ScanY( Vertex v[], int nv, int bot ){
    int li = bot, ri = li;
    int y = ceil( v[bot].y ), ly = y, ry = y;
    Vertex l, dl, r, dr;
    for( int rem = nv; rem > 0; ){
        while( ly <= y && rem-- > 0 ){
            int i = li; li = mod( li-1, nv ); ly = ceil( v[li].y );
            if( ly > y ) DiffY( v[i], v[li], l, dl, y );
        }
        while( ry <= y && rem-- > 0 ){
            int i = ri; ri = mod( ri+1, n ); ry = ceil( v[ri].y );
            if( ry > y ) DiffY( v[i], v[ri], r, dr, y );
        }
        for( ; y < ly && y < ry; y++ ) {
            ScanX( l, r, y ); l += dl; r += dr; }
    }
}
```

find left boundary edge of a tile (first while loop)

find right boundary edge of a tile (second while loop)

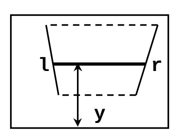
scan convert lines (for loop)

By P. Schröder

Convex Polygons

Scan conversion of a single scan line:

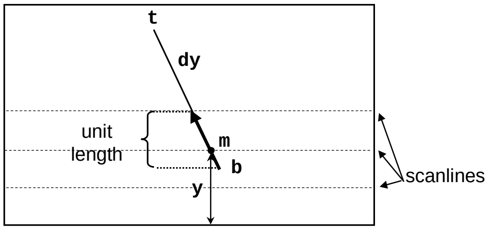
```
ScanX( Vertex& l, Vertex& r, float y ){
    int lx = ceil( l.x ), rx = ceil( r.x );
    if( lx < rx ){
        for( int x = lx; x < rx; x++ ){
            Point( x, y );
        }
    }
}
```



Convex Polygons

Compute increment in y direction and starting/ending point for the first scan

```
DiffY( Vertex& b, Vertex& t, Vertex& m, Vertex& dy, int y ){
   dy = (t - b) / (t.y - b.y); m = b + (y - b.y) * dy;
}
```



Texture mapping for polygons

- **map an image to a polygon (e.g. quadrilateral)**
- **can be done during scan conversion**
- **can be done using scaling but the scale factor depends on pixel**
- **to figure out the dependence, need to know the map from the quadrilateral to the image.**