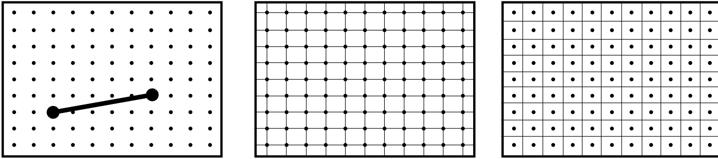
Drawing Lines

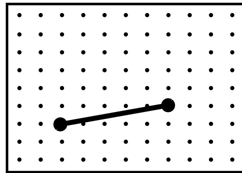
- frame buffer coordinates

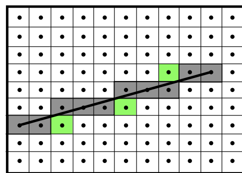


Drawing Lines

(scan conversion/rasterization)

- Given: (x_1, y_1) , (x_2, y_2)
- want:
  - straight
  - consistent density
  - speed
    - integer
    - incremental





Drawing Lines

- incremental methods

$$y = mx + b$$

$$m = \frac{\Delta y}{\Delta x}$$

$$y_{i+1} = m(x_{i+1}) + b = m(x_i + \Delta x) + b$$

$$y_{i+1} = y_i + m\Delta x$$

if $\Delta x = 1$

$$y_{i+1} = y_i + m$$

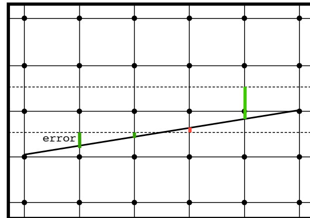
Simple DDA

```
DrawLine(int x1, int y1, int x2, int y2) {
 int x
 double dx, dy, y, m
 dx = x2-x1
 dy = y2-y1
 m = dy/dx
 y = y1;
 for (x = x1; i <= x2; x++) {
  SetPixel(x, round(y))
  y += m
 }
}
```

- *assumes -1<m<1*
- *problems:*
  - *floating pt*
  - *round()*
- *workaround:*
  - *add 0.5 to y and truncate*

Bresenham's Alg 1962

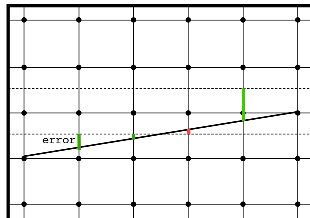
```
DrawLine(int x1,int y1,int x2,int y2){
 int x, y
 double dx, dy, error
 dx = x2-x1
 dy = y2-y1
 error = dy/dx - 0.5
 y = y1;
 for (x = x1; i <= x2; x++) {
  SetPixel(x, y)
  if (error > 0.0) {
    y++
    error = error - 1.0
  }
  error = error + dy/dx
 }
}
```



- *error = distance between half-way point and true line at next step*
- *problems:*
  - *division*
  - *floating point*

Bresenham's Alg

```
DrawLine(int x1,int y1,int x2,int y2){
 int x, y, dx, dy, error
 dx = x2-x1
 dy = y2-y1
 error = 2*dy-dx
 y = y1;
 for (x = x1; i <= x2; x++) {
  SetPixel(x, y)
  if (error > 0) {
    y++
    error = error - 2*dx
  }
  error = error + 2*dy
 }
}
```

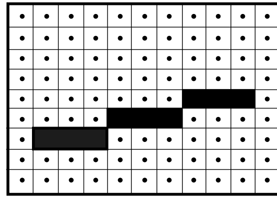


- *multiply error by 2*dx (only care about sign)*
- *developed in 1960s*
- *pen plotters*

Smoothing Lines

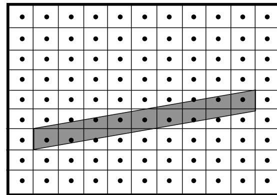
Anti-Aliasing

- jagged lines
- staircasing
- also known as aliasing (need to perform anti-aliasing)
- static bad, motion worse



Area Averaging

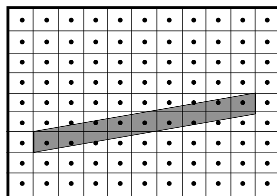
- contribution proportional to area within pixel square

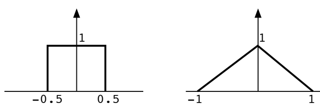


$$I'(x_0, y_0) = \int_{x_0-0.5}^{x_0+0.5} \int_{y_0-0.5}^{y_0+0.5} I(x,y)dxdy$$

Weighed Area Averaging

- filtering theory encourages use of weighed area averaging
- filter has higher value closer to centre



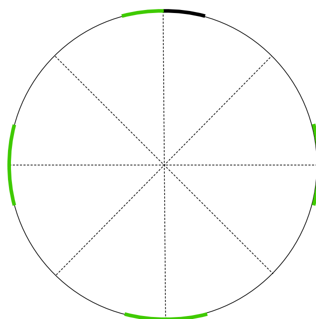


$$I'(x_0, y_0) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} I(x,y)F(x_0 - x, y_0 - y)dxdy$$

- *overlap across pixel squares*
- *weighed filter normalized*

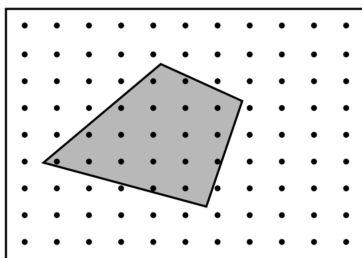
Drawing Circles

- 8-fold symmetry
- do just 1 octant and mirror the rest
- modified Bresenham



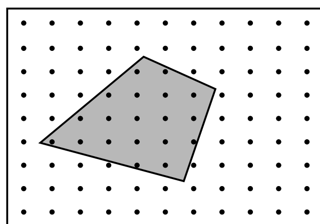
Polygon Rasterization

- given: list of vertices
  generate: interior pixels



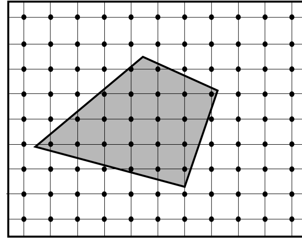
Polygon Rasterization

- given: list of vertices
  generate: interior pixels
- types of polygons:
  - rectangular, axis-aligned
  - convex
  - concave
  - concave, with holes, self-intersecting



Convex Polygons

- find top vertex
- go down left and right sides
- compute intersections with scanline
- draw horizontal runs on each scan line

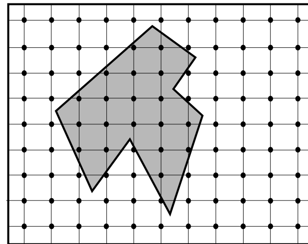


- *incremental*
- *amortize edge computations*

Concave

For each scanline:

- find intersection of scanline and edges of polygon
- sort intersections by increasing x value
- fill in pixels between pairs of intersection points

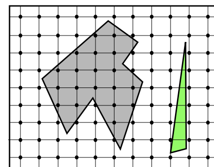


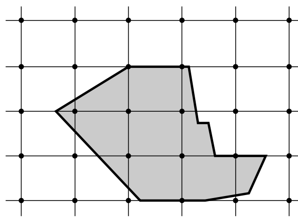
can also handle holes

Issues

- slivers
- horizontal edges
- vertices with integer y coords

[…)

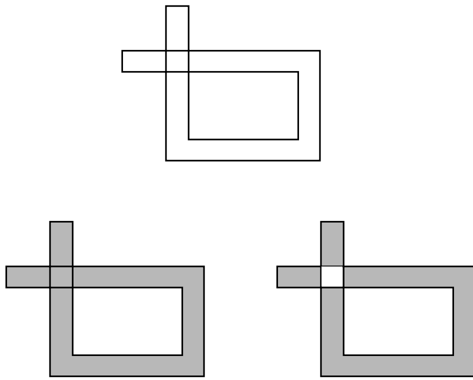




Speeding Up Concave

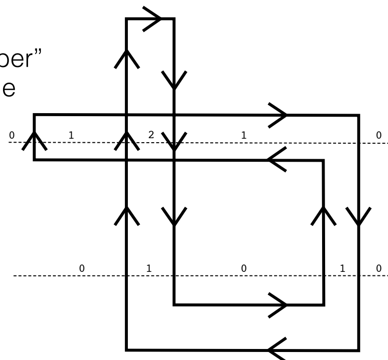
- rely on edge and scanline **coherence** (the extent to which image is locally the same)
- sort edges w.r.t. max y
- compute active edges (edges that intersect current scanline) by going down sorted edge list and adding new edges
- intersect scanline with active edges to get intersection points
- draw visible spans
- go to next scanline (throw out old edges, get new active edges)

Self-Intersecting?



Newell & Sequin

- VLSI
- compute "winding number" as you go along scanline
- edge going up: +1
- edge going down: -1
- draw non-zero spans



Newell & Sequin

- can also do holes
- hole edges labeled in opposite direction