Spans

- Fill the bottom horizontal span of pixels; move up and keep filling
- Assume we have xmin, xmax.
- Recall—for non integral xmin (going from outside to inside), round up to get first interior point, for non integral xmax (going from inside to outside), round down to get last interior point
- Recall—convention for integral points gives a span closed on the left and open on the right
- Thus: fill from ceiling(xmin) up to but not including ceiling(xmax)

Algorithm

- For each row in the polygon:
  - Throw away irrelevant edges (horizontal ones, ones that we are done with)
  - Obtain newly relevant edges (ones that are starting)
  - Fill spans
  - Update spans

The next span - 1

- for an edge, have \( y = mx + c \)
- hence, if \( y_{n+1} = m \cdot x_{n+1} + c \), then \( y_{n+1} = y_{n} + 1 \cdot m \cdot (x_{n+1} + 1/m) + c \)
- hence, if there is no change in the edges, have: \( x += (1/m) \)

The next span - 2

- Horizontal edges are irrelevant (typically would be pruned at the outset)
- Edge becomes relevant when \( y >= y_{\text{min}} \) of edge (note appeal to convention)*
- Edge becomes irrelevant - when \( y >= y_{\text{max}} \) of edge (note appeal to convention)*

*Because we add edges and check for irrelevant edges before drawing, bottom horizontal edges are drawn, but top ones are not.
Filling in details -- 1

- For each edge store: x-value, maximum y value of edge, $1/m$
  - x-value starts out as x value for $y_{\text{min}}$
  - m is never 0 because we ignore horizontal ones
- Keep edges in a table, indexed by minimum y value (Edge Table==ET)
- Maintain a list of active edges (Active Edge List==AEL).

Filling in details -- 2

- For row = min to row=max
  - AEL=append(AEL, ET(row)); (add edges starting at the current row)
  - remove edges whose ymax=row
    - OK since we are assuming integral coordinates; otherwise one would use ceil(ymax)
  - sort AEL by x-value
  - fill spans
    - use parity rule
    - remember convention for integral $x_{\text{min}}$ and $x_{\text{max}}$
  - update each edge in AEL
    - $x += (1/m)$

Compute the edge table (ET) to begin. Then fill polygon and update active edge list (AEL) row by row.

Format of edge entries

\[
\begin{array}{ccc}
  x & \frac{1}{m} & y_{\text{max}} \\
\end{array}
\]

\[
\begin{array}{c}
  \text{ET} \\
  3 & 0 & 5 \\
  2 & \cdot \\
  1 & \\
  0 & 1 & 3 & 5 & 0 & 5 \\
\end{array}
\]
AEL just before filling

Row=5
3 0 5 5 0 5

Row=4
3 0 5 5 0 5

Row=3
1 0 3 5 0 5

Row=2
1 0 3 5 0 5

Row=1
1 0 3 5 0 5

Row=0
1 0 3 5 0 5

Format of edge entries

(This is all there is in the ET---why?)

ET

1 1 4
4 -1 4
0

Comments

- Sort is quite fast, because AEL is usually almost in order.
- Nonetheless, OpenGL limits to convex polygons, so two and only two elements in AEL at any time, and no sorting.
- With additional logic to keep track of what color to use, can fill in many polygons at a time.
- Can be done without division/floating point.
Dodging division and floating point

- \(1/m = Dx/Dy\), which is a rational number.
- \(x = x_{\text{int}} + x_{\text{num}}/Dy\)
- store \(x\) as \((x_{\text{int}}, x_{\text{num}})\),
- then \(x \rightarrow x+1/m\) is given by:
  - \(x_{\text{num}} = x_{\text{num}} + Dx\)
  - if \(x_{\text{num}} \geq x_{\text{denom}}\)
    - \(x_{\text{int}} = x_{\text{int}} + 1\)
    - \(x_{\text{num}} = x_{\text{num}} - x_{\text{denom}}\)

- Advantages:
  - no division/floating point
  - can tell if \(x\) is an integer or not (check \(x_{\text{num}} = 0\)), and get \(\text{truncate}(x)\) easily, for the span endpoints.

Aliasing/Anti-Aliasing

- Analogous to the case of lines
- Anti-aliasing is done using graduated gray levels computed by smoothing and sampling

Aliasing

Ideal

Boundary fill

- Basic idea: fill in pixels inside a boundary
- Recursive formulation:
  - to fill starting from an inside point
    - if point has not been filled,
      - fill
      - call recursively with all neighbours that are not boundary pixels

Aliasing/Anti-Aliasing

- Some anti-aliasing approaches implicitly deal with boundary ambiguity
- Problem with “slivers” is really an sampling problem and is handled by filtering and sampling.
Choice of neighbours is important

![4-connected and 8-connected diagrams]

- 4-connected
- 8 connected

4 connected fill of a four connected boundary doesn't work

Pattern fill

- Use coordinates as index into pattern

Clipping

- 2D elements are laid out in a convenient (often user based) coordinate system--perhaps km for a map--and then transformed to a frame buffer coordinate system.
- Objects that are to be drawn must lie inside frame buffer, and may have to lie inside particular region - e.g. viewport.
- We want to dodge additional expensive operations on objects or parts of objects that won’t be displayed.
- How do we ensure line/polygon lies inside a region?

Clipping in the 2D pipeline

- Element in modelling coordinates
- Transform into frame buffer coordinates
- Clip
- Convert to pixels in frame buffer

Clipping references

- Hearn and Baker
  - C-S (lines): p 317
  - L-B (lines): p 322
  - N-L (lines): p 325
  - S-H (poly): p 331
  - W-A(poly): p 335
- Foley at al.
  - C-S (lines): p 103
  - L-B (lines): p 107
  - N-L (lines): N.A.
  - S-H (poly): p 112
  - W-A(poly): N.A.
Clipping lines

Have

Need

Cohen-Sutherland clipping (lines)

- Clip line against convex region.
- For each edge of the region, clip line against that edge:
  - line all on wrong side of any edge? throw it away (trivial reject—e.g. red line with respect to bottom edge)
  - line all on correct side of all edges? doesn’t need clipping (trivial accept—e.g. green line).
  - line crosses edge? replace endpoint on wrong side with crossing point (clip)

Cohen Sutherland - details

- Only need to clip line against edges where one endpoint is inside and one is outside.
- The state of the outside endpoint (e.g., in or out, w.r.t a given edge) changes due to clipping as we proceed—need to track this.
- Use “outcode” to record endpoint in/out wrt each edge. One bit per clipping edge, 1 if out, 0 if in.
Outcode example

Frame buffer
1 2 3
4
Outcode for P1?
Outcode for P2?

Outcode example

Frame buffer
1 2 3
4
Outcode for P1?
Outcode for P2?

Note: As we process the four edges, the outcodes change