Dodging floating point

• $1/m = D_x/D_y$, which is a rational number.
• $x = x_{\text{int}} + x_{\text{num}}/D_y$
• store $x$ as $(x_{\text{int}}, x_{\text{num}})$,
• then $x \rightarrow x + 1/m$ is given by:
  – $x_{\text{num}} = x_{\text{num}} + D_x$
  – 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 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/Anti-Aliasing

• Some anti-aliasing approaches implicitly deal with boundary ambiguity
• Problem with “slivers” (page 90) is really an aliasing problem.
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
Choice of neighbours is important

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 may also 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 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 some 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.
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 edge, 1 if out, 0 if in.
Cohen Sutherland - details

• Trivial reject:
  – outcode(p1) & outcode(p2) != 0

• Trivial accept:
  – outcode(p1) | outcode(p2) == 0

• Clipping line against vertical/horizontal edge is easy:
  – line has endpoints (x_s, y_s) and (x_e, y_e)
  – e.g. (vertical case) clip against x=a gives the point
    \[(a, y_s + (a - x_s)((y_e - y_s)/(x_e - x_s)))\]
  – new point replaces the point for which outcode() is true

• Algorithm is valid for any convex clipping region (intersections are slightly more difficult)
Cohen Sutherland - Algorithm

• Compute outcodes for endpoints
• While not trivial accept and not trivial reject:
  – clip against a problem edge (i.e. one for which an outcode bit is 1)
  – compute outcodes again
• Return appropriate data structure
Cyrus-Beck/Liang-Barsky clipping

- Parametric clipping: view line in parametric form and reason about the parameter values

- More efficient, as we don’t compute the coordinate values at irrelevant vertices

- Line is:

  \[
  \begin{align*}
  x &= x_1 + t D_x \\
  y &= y_1 + t D_y \\
  \end{align*}
  \]

  \[
  \begin{align*}
  D_x &= x_2 - x_1 \\
  D_y &= y_2 - y_1 \\
  \end{align*}
  \]
Cyrus-Beck/Liang-Barsky clipping

• Consider the parameter values, t, for each clip edge
• Only t inside (0,1) is relevant
• Assumptions
  – \( X_1 \neq X_2 \)
  – Ignore case where line is parallel to a clip edge (has no effect, but would lead to divide by zero).
  – We have a normal, \( n \), for each clip edge pointing outward
  – For axis aligned rectangle (the usual case) these are ?:
Cyrus-Beck/Liang-Barsky clipping

- Consider the parameter values, t, for each clip edge
- Only t inside (0,1) is relevant
- Assumptions
  - \( X_1 \neq X_2 \)
  - Ignore case where line is parallel to a clip edge (has no effect, but would lead to divide by zero).
  - We have a normal, \( n \), for each clip edge pointing outward
  - For axis aligned rectangle (the usual case) these are:
    - left \((-1,0)\)  right \((1,0)\)  top \((0,1)\)  bottom \((0,-1)\)
Computing $t$ for intersection point, $X$
Computing $t$ for intersection point, $X$

Simplest to work from condition

$$(X(t) - P_e) \cdot n = 0$$
Computing $t$ for intersection point, $X$

Set
$$D = X_2 \parallel X_1$$

Then
$$X = X_1 + tD$$

And condition is
$$(P_e \parallel (X_1 + tD)) \cdot n = 0$$
Computing $t$ for intersection point, $X$

Condition

$$(P_e \parallel (X_1 + tD)) \cdot n = 0$$

Rearrange

$$(P_e \parallel X_1) \cdot n = tD \cdot n$$

And solve

$$t = \frac{(P_e \parallel X_1) \cdot n}{D \cdot n}$$