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?

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 w.r.t each edge. One bit per clipping edge, 1 if out, 0 if in.
Cohen Sutherland - details

- Trivial reject:
  -
- Trivial accept:
  -
- Clipping line against vertical/horizontal edge is easy:
  - line has endpoints \((x_s, y_s)\) and \((x_e, y_e)\)
  -
  -
  -

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

- 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:

  - Line is:

  \[
  x = x_2 + t(x_2 - x_1) \\
  y = y_2 + t(y_2 - y_1)
  \]
Computing $t$ for intersection point

Think of $X$ moving along the line shown. What is the condition that it is on the other line as well (i.e., intersects?)

Simplest to work from condition $(X(t) - P_e) \cdot n = 0$

Consider left edge. Now $n = ?$ and $P_e$ = ?

And $t = ?$

From previous slide $t = \frac{D \cdot X_1 - n}{D \cdot n}$

This simplifies greatly for axis aligned rectangles
Computing \( t \) for intersection point

From previous slide 

\[ t = \frac{\mathbf{P}(X_1) \cdot \mathbf{m}}{\mathbf{D} \cdot \mathbf{m}} \]

This simplifies greatly for axis aligned rectangles

Consider left edge. Now \( \mathbf{m} = (-1,0) \) and \( \mathbf{P}(x_{min},0) \)

And 

\[ t = \frac{(x_{min},0)}{(1,0)} \]

• All four cases can expressed by: 
  \[ t = \frac{q}{p_1} \]
  Where
  \[ p_1 = \begin{bmatrix} x \end{bmatrix} \quad q_1 = x_1 \quad x_{max} \]
  \[ p_2 = \begin{bmatrix} x \end{bmatrix} \quad q_2 = x_{max} \quad x_1 \]
  \[ p_3 = \begin{bmatrix} y \end{bmatrix} \quad q_3 = y_{min} \quad y_{max} \]
  \[ p_4 = \begin{bmatrix} y \end{bmatrix} \quad q_4 = y_{max} \quad y_1 \]

• Faster derivation for this special case?

Cyrus-Beck/Liang-Barsky (cont)

• Next step: Use the \( t \)'s to determine the clip points
  • Recall that only \( t \) in \((0,1)\) is relevant, but we need additional logic to determine clip endpoints from multiple \( t \)'s inside \((0,1)\).
  • We imagine going from \( X_1 \) to \( X_2 \) and classify intersections as either potentially entering (PE) or potentially leaving (PL) if they go across a clip edge from outside in, or inside out.
  • Whether an edge is PE or PL is easily determined from the sign of \( \mathbf{D} \cdot \mathbf{n} \) which we have already computed.

\[ \mathbf{n} \]

\[ \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} \]

\[ \mathbf{D} = X_1 \cup X_2 \]
Cyrus-Beck/Liang-Barsky--Algorithm

- Compute incoming (PE) t values, which are \( q_k/p_k \) for each \( p_k < 0 \)
- Compute outgoing (PL) t values, which are \( q_k/p_k \) for each \( p_k > 0 \)
- Parameter value for small t end of the segment is:
  \( t_{\text{small}} = \max(0, \text{incoming values}) \)
- Parameter value for large t end of the segment is:
  \( t_{\text{large}} = \min(1, \text{outgoing values}) \)
- If \( t_{\text{small}} < t_{\text{large}} \), there is a segment portion in the clip window - compute endpoints by substituting t values (otherwise reject as it is outside).