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 \textit{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
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 - 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: consider 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:

\[
\begin{align*}
x &= x_1 + t 
\end{align*}
\]

\[
\begin{align*}
y &= y_1 + t 
\end{align*}
\]

\[
\begin{align*}
x &= x_2 - x_1 
\end{align*}
\]

\[
\begin{align*}
y &= y_2 - y_1 
\end{align*}
\]
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
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?)
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 $(X(t) - P_e) \cdot n = 0$
Computing $t$ for intersection point

Set

$$D = X_2 \square X_1$$

Then

$$X = X_1 + tD$$

And condition is

$$(P_e \square (X_1 + tD)) \cdot n = 0$$
Computing $t$ for intersection point, $X$

Condition

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

Rearrange

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

And solve

$$t = \frac{(P_e \oplus X_1) \cdot n}{D \cdot n}$$
Computing $t$ for intersection point

From previous slide \[ t = \frac{(P_e \cdot X_1) \cdot n}{D \cdot n} \]

This simplifies greatly for axis aligned rectangles

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

And \[ t = ? \]
Computing $t$ for intersection point

From previous slide \[ t = \frac{(P_e \cdot X_1) \cdot n}{D \cdot n} \]

This simplifies greatly for axis aligned rectangles

Consider left edge. Now $n=(-1,0)$ and $P_e=(x_{\text{min}},0)$

And \[ t = \frac{(x_1 \cdot x_{\text{min}})}{\square x} \]
• All four cases can expressed by:  \[ t = \frac{q_k}{p_k} \]

• Where

\[
\begin{align*}
p_1 &= \Box x & q_1 &= x_1 \Box x_{\text{min}} \\
p_2 &= \Box x & q_2 &= x_{\text{max}} \Box x_1 \\
p_3 &= \Box \Box y & q_3 &= y_1 \Box y_{\text{min}} \\
p_4 &= \Box y & q_4 &= y_{\text{max}} \Box y_1
\end{align*}
\]

• Faster derivation for this special case?
• All four cases can be expressed by:
  \[ t = \frac{q_k}{p_k} \]

• Where

  \[ p_1 = \Box x \quad q_1 = x_1 \Box x_{\text{min}} \]
  \[ p_2 = \Box x \quad q_2 = x_{\text{max}} \Box x_1 \]
  \[ p_3 = \Box y \quad q_3 = y_1 \Box y_{\text{min}} \]
  \[ p_4 = \Box y \quad q_4 = y_{\text{max}} \Box y_1 \]

• One can also get this special case directly by solving:

  \[ x_{\text{min}} \Box x_1 + t \Box x \Box x_{\text{max}} \]
  \[ y_{\text{min}} \Box y_1 + t \Box y \Box y_{\text{max}} \]
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 $D \cdot n$ which we have already computed.

\[
D = X_2 \cap X_1
\]
Cyrus-Beck/Liang-Barsky--Algorithm
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).