Clipping

- Remove portion of line outside viewport or screen boundaries
- Two approaches:
  - Clip during scan conversion: per-pixel bounds check, or span endpoint tests.
  - Clip analytically, then scan-convert the modified primitive.
Line Clipping

Basic calculations:
- Is an endpoint inside or outside the clip rectangle?
- Find the point of intersection, if any, between a line segment and an edge of the clip rectangle.

Both endpoints inside: trivial accept
One inside: find intersection and clip
Both outside: either clip or reject

Cohen-Sutherland Line-Clipping Algorithm

<table>
<thead>
<tr>
<th>View port</th>
<th>above</th>
<th>below</th>
<th>right</th>
<th>left</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>1000</td>
<td>1010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0001</td>
<td>0000</td>
<td>0010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0101</td>
<td>0100</td>
<td>0110</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

< Region code for each endpoint >

<table>
<thead>
<tr>
<th>above</th>
<th>below</th>
<th>right</th>
<th>left</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit 4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Cohen-Sutherland Line-Clipping Algorithm

- Trivially accepted
  if (both region codes = 0000)
- Trivially rejected
  if (AND of region codes ≠ 0000)
- Otherwise, divide line into two segments
  - test intersection edges in a fixed order.
    (e.g., top-to-bottom, right-to-left)

* fixed order testing and clipping cause needless clipping (external intersection)
Coen-Sutherland Line-Clipping Algorithm

- Midpoint Subdivision for locating intersections
  1. trivial accept/reject test
  2. midpoint subdivision:
     \[ x_m = \frac{x_1 + x_2}{2}, \quad y_m = \frac{y_1 + y_2}{2} \]
     (one addition and one shift)
  3. repeat step 1 with two halves of line
     \[ \Rightarrow \text{good for hardware implementation} \]

- When this is good
  - If it can trivially reject most cases
  - Works well if a window is large w.r.t. to data
  - Works well if a window is small w.r.t. to data
  - i.e., it works well in extreme cases
  - Good for hardware implementation
Parametric Line Clipping (Cyrus-beck Technique)

- Use a parametric line equation
  \[ P(t) = P_0 + t(P_1 - P_0), \quad 0 \leq t \leq 1 \]
- Reduce the number of calculating intersections by simple comparisons of parameter \( t \).

**Algorithm**

- For each edge \( E_i \) of the clip region
- \( N_i \) : outward normal of \( E_i \)
Parametric Line Clipping  
(Cyrus-beck Technique)

- Choose an arbitrary point $P_{E_i}$ on edge $E_i$ 
  and consider three vectors $P(t) - P_{E_i}$ 
  $\Rightarrow$
  $N_i \cdot (P(t) - P_{E_i}) < 0 \iff$ a point in the side halfplane
  $N_i \cdot (P(t) - P_{E_i}) = 0 \iff$ a point on the line containing the edge
  $N_i \cdot (P(t) - P_{E_i}) > 0 \iff$ a point in the outside halfplane

---

- Solve for the value of $t$ at the intersection of $P_0P_1$ with the edge:
  $N_i \cdot [P(t) - P_{E_i}] = 0.$
  $P(t) = P_0 + t(P_1 - P_0)$ and let $D = (P_1 - P_0),$
  Then
  $$t = \frac{N_i \cdot [P_0 - P_{E_i}]}{-N_i \cdot D}$$

  $N_i \neq 0,$
  $D \neq 0$ (that is $P_0 \neq P_1),$
  $N_i \cdot D \neq 0$ (if not, no intersection)
Given the four values of $t$ for a line segment, determine which pair of $t$'s are internal intersections.

If $t \not\in [0,1]$ then discard else choose a (PE, PL) pair that defines the clipped line.

PE (potentially entering) intersection:
- if moving from $P_0$ to $P_1$ causes us to cross an edge to enter the edge's inside half plane;

PL (potentially leaving) intersection:
- if moving from $P_0$ to $P_1$ causes us to leave the edge's inside half plane.

i.e., $N_i \cdot P_0P_1 < 0 \Rightarrow PE$

$N_i \cdot P_0P_1 > 0 \Rightarrow PL$

Intersections can be categorized!

Inside the clip rectangle $(T_E, T_L)$
- $T_E$: select PE with largest $t$ value $\geq 0$
- $T_L$: select PL with the smallest $t$ value $\leq 1$. 
Parametric Line Clipping
(Cyrus-beck Technique)

- This is an efficient algorithm when many line segments need to be clipped
- Can be extended easily to convex polygon windows

Liang-Barsky line clipping

- The ideas for clipping line of Liang-Barsky and Cyrus-Beck are the same. The only difference is Liang-Barsky algorithm has been optimized for an upright rectangular clip window.
- Finds the appropriate end points with more efficient computations.
Liang-Barsky Line Clipping

Let PQ be the line which we want to study.

Parametric equation of the line segment

\[
x = x_1 + (x_2 - x_1)t = x_1 + dx \times t
\]

\[
y = y_1 + (y_2 - y_1)t = y_1 + dy \times t
\]

\[t = 0 \Rightarrow P(x_1, y_1)\]
\[t = 1 \Rightarrow Q(x_2, y_2)\]

Liang-Barsky Line Clipping

1. Set \( t_{\text{min}} = 0 \) and \( t_{\text{max}} = 1 \)
2. Calculate the values of \( t_T, t_B, t_L, t_R \),

<table>
<thead>
<tr>
<th>Top edge: ( y = T )</th>
<th>Bottom edge: ( y = B )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( y_1 + t^* (y_2 - y_1) = T )</td>
<td>( y_1 + t^* (y_2 - y_1) = B )</td>
</tr>
<tr>
<td>( t_T = \frac{T - y_1}{y_2 - y_1} )</td>
<td>( t_B = \frac{B - y_1}{y_2 - y_1} )</td>
</tr>
<tr>
<td>Left edge: ( x = L )</td>
<td>Right edge: ( x = R )</td>
</tr>
<tr>
<td>( x_1 + t^* (x_2 - x_1) = L )</td>
<td>( x_1 + t^* (x_2 - x_1) = R )</td>
</tr>
<tr>
<td>( t_L = \frac{L - x_1}{x_2 - x_1} )</td>
<td>( t_R = \frac{R - x_1}{x_2 - x_1} )</td>
</tr>
</tbody>
</table>
Liang-Barsky Line Clipping

- If \( t < t_{\text{min}} \) or \( t > t_{\text{max}} \), ignore it and go to the next edge.
- Otherwise classify the \( t \) value as entering or exiting value (using the inner product to classify)
  - Let \( PQ \) be the line and \( N \) is normal vector
  - If \( N \cdot (Q - P) \leq 0 \), the parameter \( t \) is entering
  - If \( N \cdot (Q - P) > 0 \), the parameter \( t \) is exiting
  - If \( t \) is entering value, set \( t_{\text{min}} = t \); if \( t \) is exiting value set \( t_{\text{max}} = t \)

3. If \( t_{\text{min}} < t_{\text{max}} \) then draw a line from \((x_1 + dx \times t_{\text{min}}, y_1 + dy \times t_{\text{min}})\) to \((x_1 + dx \times t_{\text{max}}, y_1 + dy \times t_{\text{max}})\)
Clipping

- Clipping rotated windows, circles
  - trivial acceptance/rejection test with respect to bounding rectangle of the window
- Line clipping using nonrectangular clip windows
  - extend Cyrus-Beck algorithm

Polygon clipping

- Sutherland-Hodgeman Algorithm
  - clip against 4 infinite clip edge in succession
Sutherland-Hodgeman Algorithm

- Accept a series of vertices (polygon) and outputs another series of vertices
- Four possible outputs

1. \(V_1\rightarrow V_2\) in in
   Output: \(V_1, V_2\)
2. \(V_2\rightarrow V_1\) in in
   Output: \(V_2\)
3. \(V_1\rightarrow V_2\) in out
   Output: \(V_1\)
4. \(V_2\rightarrow V_1\) out out
   Output: none

Sutherland-Hodgeman Algorithm

- The algorithm correctly clips convex polygons, but may display extraneous lines for concave polygons.

- How clip?
How to correctly clip

[Way I] Split the concave polygon into two or more convex polygons and process each convex polygon separately.

[Way II] Modify the algorithm to check the final vertex list for multiple vertex points along any clip window boundary and correctly join pairs of vertices.

[Way III] Use a more general polygon clipper

Clipping concave polygons

- Split the concave polygon into two or more convex polygons and process each convex polygon separately.

- Vector method for splitting concave polygons
  \[\Rightarrow\] calculate edge-vector cross products in a counterclockwise order. If any z component turns out to be negative, the polygon is concave.
Weiler-Atherton Polygon Clipping

- For an outside-to-inside pair of vertices, follow the polygon boundary.
- For an inside-to-outside pair of vertices, follow the window boundary in a clockwise direction.

Polygon clipping using nonrectangular polygon clip windows

Figure 6-30
Clipping a polygon fill area against a concave-polygon clipping window using the Weiler-Atherton algorithm.
Texture Clipping

1. all-or-none text clipping: Using boundary box for the entire text
2. all-or-none character clipping: Using boundary box for each individual
3. clip individual characters
   - vector: clip line segments
   - bitmap: clip individual pixels

What we have got!