Clipping
Clipping Lines and Polygons

What is clipping?

**Goal:** only render what will be visible in the drawing window; eliminate from the rendering pipeline all those primitives which will not appear in the final viewport.

World coordinate system  →  Window in WCS  →  Viewport on CRT
Clipping Approaches

Individual Pixel Tests:
- Render at the pixel level
- Check pixel for containment in the target region

Maintain Large Drawing Area:
- Render the entire set of primitives in the WCS
- Display only the portion of the set which overlaps the viewport

Analytical Methods:
- Compute the intersections of primitives with the clip region
- Eliminate outlying portions of the primitives
Analytical Clipping

Brute force method: compute all intersections of line with clip rectangle

\[ P_1 = \{ y = y_{\text{min}} \} \cap \{ y = mx + b \} \]

\[ P_2 = \{ x = x_{\text{max}} \} \cap \{ y = mx + b \} \]
Cohen-Sutherland Clipper

- Use *regions* to accept/reject lines
- Assume the clipping region is an axis-oriented rectangle

Regions are labeled according to the half-planes defined by the clipping region:

- $y = y_{\max}$
- $y = y_{\min}$
- $x = x_{\min}$
- $x = x_{\max}$
Assign a bit to each condition to form a code for the nine regions

bit 1: sign bit of \((y_{\text{max}} - y)\)
bit 2: sign bit of \((y - y_{\text{min}})\)
bit 3: sign bit of \((x_{\text{max}} - x)\)
bit 4: sign bit of \((x - x_{\text{min}})\)
The Cohen-Sutherland Algorithm

Let $c_1$ and $c_2$ be the region codes for the endpoints of the line to be clipped.

Trivially **Accept:**
Both codes are 0000 $\rightarrow$ segment is inside the clip region

Trivially **Reject:**
(code 1) && (code 2) are not equal to zero $\rightarrow$
Line is outside clip rectangle
(both endpoints must be on the same side of one of the four clip lines in order to yield a bitwise & that is non-zero)
Iterative Part of the Algorithm

Repeat until the segment cannot be trivially accepted or rejected:

• Subdivide the segment:
  
  • Pick the endpoint which is outside the clip rectangle (one must be outside)
  • Find the first non-zero bit: this corresponds to the clip edge which intersects the line
  • Compute the intersection of the line with the edge
  • Throw away the outside vertex up to the clip rectangle

\[
\{ y = mx + b \} \cap \{ y = x_{\min} \} \Rightarrow x_{\min} = mx + b \Rightarrow x = \frac{x_{\min} - b}{m}
\]
An Example

Subdivide using either A or E

E: 1010
A: 0100
&: 0000

top bottom right left
Example, Con’t

Subdivide: Third bit of D indicates that the line intersects the right clip line.
Example (Con’t)

C: 0000
A: 0100
&: 0000

Subdivide: Second bit of A indicates that the line intersects the bottom clip line
Example: Conclusion

C: 0000
B: 0000
&: 0000

Trivially accept the clipped segment BC
Cohen-Sutherland Summary

Disadvantages:
- Fixed-order decision can do needless work
- Can improve using more regions (Nicholl-Lee-Nicholl Alg)
- Can generate more efficient rejection tests
- Clipping window must be rectangular

Advantages:
- Simple to implement
- Oriented for most simple window/viewport systems
- Extends to 3-D cubic volumes
The Liang-Barsky Line clipper (parametric line clipping)

Parametric Line

\[ x = x_1 + t\Delta x \]
\[ y = y_1 + t\Delta y \]

where

\[ \Delta x = x_2 - x_1, \Delta y = y_2 - y_1 \]
\[ t \in [0,1] \]
Clipping Condition

\[ y = y_{\max} \]

\[ y = y_{\min} \]

\[ x = x_{\min} \]

\[ x = x_{\max} \]

\[ x_{\min} \leq x_1 + t\Delta x \leq x_{\max} \]

\[ y_{\min} \leq y_1 + t\Delta y \leq y_{\max} \]
Intersection Conditions

\[ x_{\text{min}} \leq x_1 + t\Delta x \leq x_{\text{max}} \]
\[ y_{\text{min}} \leq y_1 + t\Delta y \leq y_{\text{max}} \]

\[ tp_k \leq q_k \]

where

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

- If \((p_k = 0)\) – parallel line to one of the clipping edge
  - \((q_k < 0)\) – outside, reject
  - \((q_k \geq 0)\) – inside
- If \((p_k < 0)\) – from outside to inside
  - Intersection at: \(t = \frac{q_k}{p_k}\) (OIP)
- If \((p_k > 0)\) – from inside to outside
  - Intersection at: \(t = \frac{q_k}{p_k}\) (IOP)

Key Insight: we need to find two intersection points, one OIP, one IOP.
Intersection Point Computation

- For \( p_k < 0 \) (Outside to inside point)
  - \( r_k = q_k/p_k \)
  - OIP \( t_1 = \max(r_k, 0) \)

- For \( p_k > 0 \) (Inside to outside point)
  - \( r_k = q_k/p_k \)
  - IOP \( t_2 = \min(r_k, 1) \)

- If \( t_1 > t_2 \), complete outside, REJECT
- Else clipped line is between \( (t_1, t_2) \)
Liang-Barsky Line Clipping Example

\[ y_{\text{max}} = 9 \]

\[ y_{\text{min}} = 4 \]

\[ x_{\text{min}} = 10 \]

\[ x_{\text{max}} = 20 \]
Summary: Liang-Barsky

- (x, y) coordinates are only computed for the two final intersection points.
- At most 4 parameter values are computed.
- This is a non-iterative algorithm.
- Can be extended to 3-D.
Take Home Exercise

Work through the Cohen-Sutherland Clipping Algorithm using the following example

\[
\begin{align*}
(7,10) & \quad X_{\text{min}}=10 & X_{\text{max}}=20 \\
Y_{\text{min}}=4 & \quad Y_{\text{max}}=9 \\
\end{align*}
\]

<table>
<thead>
<tr>
<th>Start</th>
<th>Code</th>
<th>End</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>(7,10)</td>
<td>1001</td>
<td>(21,3)</td>
<td>0110</td>
</tr>
<tr>
<td>??</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Take Home Exercise

- Clip the following line using the Liang-Barsky algorithm

\[ y_{\text{max}} = 9 \]
\[ y_{\text{min}} = 4 \]
\[ x_{\text{min}} = 10 \]
\[ x_{\text{max}} = 20 \]