Overview

- Cohen-Sutherland Line Clipping
- Parametric Line Clipping

Motivation

- Only draw lines inside window
- Clip lines to window boundary

Scissoring Clipping

Performed during scan conversion of the line (circle, polygon)

Compute the next point \((x, y)\)

If \(x_{\text{min}} \leq x \leq x_{\text{max}}\) and \(y_{\text{min}} \leq y \leq y_{\text{max}}\)

Then output the point.

Else do nothing

- Issues with scissoring:
  - Too slow
  - Does more work than necessary
- Better to clip lines to window, than “draw” lines that are outside of window
The Cohen-Sutherland Line Clipping Algorithm

- How to clip lines to fit in windows?
  - easy to tell if whole line falls w/in window
  - harder to tell what part falls inside
- Consider a straight line $P_0 = (x_0, y_0)$ and $P_1 = (x_1, y_1)$
- And window: $WT$, $WB$, $WL$ and $WR$

Cohen-Sutherland

Basic Idea:

- First, do easy test
  - completely inside or outside the box?
- If no, we need a more complex test
- Note: we will also need to figure out how line intersects the box

Cohen-Sutherland

Perform trivial accept and reject

- Assign each end point a location code
- Perform bitwise logical operations on a line's location codes
- Accept or reject based on result
- Frequently provides no information
  - Then perform more complex line intersection

Cohen-Sutherland

- Line is completely visible iff both code values of endpoints are 0, i.e. $C_0 \lor C_1 = 0$
- If line segments are completely outside the window, then $C_0 \land C_1 \neq 0$

Cohen-Sutherland

Otherwise, we clip the lines:

- We know that there is a bit flip, w.o.l.g. assume its $(x_0, y_0)$
- Which bit? Try 'em all!
  - suppose it's bit 4
  - Then $x_0 < WL$ and we know that $x_1 \leq WL$
  - We need to find the point: $(x_c, y_c)$
Cohen-Sutherland

- Clearly: \( x_c = WL \)
- Using similar triangles
  \[
  \frac{y_c - y_0}{y_1 - y_0} = \frac{WL - x_0}{x_1 - x_0}
  \]
- Solving for \( y_c \) gives
  \[
  y_c = \frac{WL - x_0}{x_1 - x_0} (y_1 - y_0) + y_0
  \]

Replace \((x_0, y_0)\) with \((x_c, y_c)\)
- Re-compute codes
- Continue until all bit flips (clip lines) are processed, i.e. all points are inside the clip window
Parametric Line Equation

- Line: \( P(t) = P_0 + t(P_1 - P_0) \)
- \( t \) value defines a point on the line going through \( P_0 \) and \( P_1 \)
- \( 0 \leq t \leq 1 \) defines line segment between \( P_0 \) and \( P_1 \)
- \( P(0) = P_0 \quad P(1) = P_1 \)

Intersecting Two Edges (1)

- Edge 0 : \((P_0, P_1)\)
- Edge 2 : \((P_2, P_3)\)
- \( E_0 = P_0 + t_0 * (P_1 - P_0) \quad D_0 = (P_1 - P_0) \)
- \( E_2 = P_2 + t_2 * (P_3 - P_2) \quad D_2 = (P_3 - P_2) \)
- \( P_0 + t_0 * D_0 = P_2 + t_2 * D_2 \)
- \( x_0 + dx_0 * t_0 = x_2 + dx_2 * t_2 \)
- \( y_0 + dy_0 * t_0 = y_2 + dy_2 * t_2 \)

Intersecting Two Edges (2)

- Solve for \( t \)'s
- \( t_0 = ((x_0 - x_2) * dy_2 + (y_2 - y_0) * dx_2) / (dy_0 * dx_2 - dx_0 * dy_2) \)
- \( t_2 = ((x_2 - x_0) * dy_0 + (y_0 - y_2) * dx_0) / (dy_2 * dx_0 - dx_2 * dy_0) \)
- See http://www.vb-helper.com/howto_intersect_lines.html for derivation
- Edges intersect if \( 0 \leq t_0, t_2 \leq 1 \)
- Edges are parallel if denominator = 0

Parametric Line Clipping

- Developed by Cyrus and Beck in 1978
- Used to clip 2D/3D lines against convex polygon/polyhedron
- Liang and Barsky (1984) algorithm efficient in clipping upright 2D/3D clipping regions
- Cyrus-Beck may be reduced to more efficient Liang-Barsky case
- Based on parametric form of a line
  - Line: \( P(t) = P_0 + t(P_1 - P_0) \)

The Cyrus-Beck Technique

- Cohen-Sutherland algorithm computes \((x,y)\) intersections of the line and clipping edge
- Cyrus-Beck finds a value of parameter \( t \) for intersections of the line and clipping edges
- Simple comparisons used to find actual intersection points
- Liang-Barsky optimizes it by examining \( t \) values as they are generated to reject some line segments immediately
Finding the Intersection Points

Line \( P(t) = P_0 + t(P_1-P_0) \)
Point on the edge \( P_i \)
\( N_i \) Normal to edge \( i \)

\[\begin{align*}
N_i \cdot [P(t)-P_i] &= 0 \\
N_i \cdot [P_1-P_0] &= 0 \\
N_i \cdot [P_i-P_j] + N_i \cdot [P_j-P_i] &= 0
\end{align*}\]

Let \( D = (P_i-P_j) \)
\[t = \frac{N_i \cdot [P_i-P_j]}{-N_i \cdot D} \]

Make sure
1. \( D = 0 \), or \( P_1 \neq P_0 \)
2. \( N_i \cdot D = 0 \), lines are not parallel

Calculating \( N_i \)

\( N_i \) for window edges
- WT: \((0,1)\)
- WB: \((0,-1)\)
- WL: \((-1,0)\)
- WR: \((1,0)\)

\( N_i \) for arbitrary edges
- Calculate edge direction
  \(- E = (V_1 - V_0) / |V_1 - V_0| \)
- Be sure to process edges in CCW order
- Rotate direction vector \(-90^\circ\)
  \[\begin{align*}
  N_i^x &= E \cdot y \\
  N_i^y &= -E \cdot x
  \end{align*}\]

Finding the Line Segment

- Calculate intersection points between line and every window line
- Classify points as potentially entering (PE) or leaving (PL)
- PE if crosses edge into inside half plane \( \Rightarrow \) angle \( P_0, P_1 \) and \( N_i \) greater \( 90^\circ \) \( \Rightarrow N_i \cdot D < 0 \)
- PL otherwise.
- Find \( T_e = \max(t) \)
- Find \( T_l = \min(t) \)
- If \( T_e < 0 \), \( T_e = 0 \)
- If \( T_l > 1 \), \( T_l = 1 \)
- Use \( T_e, T_l \) to compute intersection coordinates \((x_e, y_e), (x_l, y_l)\)