Clipping

Concepts, Algorithms for line clipping
Assignment 3 will be out

Quiz2 next week
Last Time

- In the canonical volume, 2 operations are easy
  - Projection
  - Clipping
Well, that’s a lie...

- Easy to clip
  - Point?
  - Line?
  - Polygon?
Line Clipping in 2D

Clipping endpoints

- If $x_{min} < x < x_{max}$ and $y_{min} < y < y_{max}$, the point is inside the clip rectangle.
Line Clipping in 2D

- Clipping endpoints
  - If $x_{min} < x < x_{max}$ and $y_{min} < y < y_{max}$, the point is inside the clip rectangle.

- Endpoint analysis for lines:
  - if both endpoints in, do "trivial acceptance"
  - if one endpoint inside, one outside, must clip
  - if both endpoints out, don’t know
Line Clipping in 2D

- Clipping endpoints
  - If $x_{min} < x < x_{max}$ and $y_{min} < y < y_{max}$, the point is inside the clip rectangle.

- Endpoint analysis for lines:
  - if both endpoints in, do “trivial acceptance”
  - if one endpoint inside, one outside, must clip
  - if both endpoints out, don’t know

- Brute force clip: solve simultaneous equations using $y = mx + b$ for line and four clip edges
  - slope-intercept formula handles infinite lines only
  - doesn’t handle vertical lines
Parametric Line Formulation For Clipping

- Parametric form for line segment

\[
\begin{align*}
X &= x_0 + t(x_1 - x_0) \\
Y &= y_0 + t(y_1 - y_0) \\
0 \leq t \leq 1
\end{align*}
\]

\[
P(t) = P_0 + t(P_1 - P_0) = (1 - t)P_0 + t(P_1)
\]
Parametric Line Formulation For Clipping

- Parametric form for line segment

\[
X = x_0 + t(x_1 - x_0) \\
Y = y_0 + t(y_1 - y_0) \\
P(t) = P_0 + t(P_1 - P_0) = (1 - t)P_0 + t(P_1)
\]

- Line is in clip rectangle if parametric variables \( t_{\text{line}} \) and \( s_{\text{edge}} \) both in \([0,1]\) at intersection point between line and edge of clip rectangle
  - Slow, must intersect lines with all edges
Cohen-Sutherland Line Clipping in 2D

- Divide plane into 9 regions
- Compute the sign bit of 4 comparisons between a vertex and an edge
  - $y_{max} - y; \ y - y_{min}; \ x_{max} - x; \ x - x_{min}$
  - point lies inside only if all four sign bits are 0, otherwise exceeds edge
- 4 bit outcode records results of four bounds tests:
  - **First bit**: above top edge
  - **Second bit**: below bottom edge
  - **Third bit**: to the right of right edge
  - **Fourth bit**: to the left of left edge
- Compute outcodes for both vertices of each line (denoted $OC_0$ and $OC_1$)
- Lines with $OC_0 = 0$ (i.e., 0000) and $OC_1 = 0$ can be trivially accepted.
Cohen-Sutherland Line Clipping in 2D

- Divide plane into 9 regions
- Compute the sign bit of 4 comparisons between a vertex and an edge
  - $y_{max} - y; \ y - y_{min}; \ x_{max} - x; \ x - x_{min}$
  - point lies inside only if all four sign bits are 0, otherwise exceeds edge
- 4 bit outcode records results of four bounds tests:
  - First bit: above top edge
  - Second bit: below bottom edge
  - Third bit: to the right of right edge
  - Fourth bit: to the left of left edge
- Compute outcodes for both vertices of each line (denoted $OC_0$ and $OC_1$)
- Lines with $OC_0 = 0$ (i.e., 0000) and $OC_1 = 0$ can be trivially accepted.
- Lines lying entirely in a half plane outside an edge can be trivially rejected if $(OC_0 \ AND \ OC_1) \neq 0$ (i.e., they share an “outside” bit)
Cohen-Sutherland Line Clipping in 3D

- Very similar to 2D
- Divide volume into 27 regions (Picture a Rubik’s cube)
- 6-bit outcode records results of 6 bounds tests
  - **First bit**: behind back plane
  - **Second bit**: in front of front plane
  - **Third bit**: above top plane
  - **Fourth bit**: below bottom plane
  - **Fifth bit**: to the right of right plane
  - **Sixth bit**: to the left of left plane
- Again, lines with $OC_0 = 0$ and $OC_1 = 0$ can be **trivially accepted**
- Lines lying entirely in a volume outside of a plane can be **trivially rejected**: $OC_0 \text{ AND } OC_1 \neq 0$ (i.e., they share an “outside” bit)

<table>
<thead>
<tr>
<th>Back plane</th>
<th>Front plane</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000 (in front)</td>
<td>010000 (in front)</td>
</tr>
<tr>
<td>100000 (behind)</td>
<td>000000 (behind)</td>
</tr>
<tr>
<td><strong>Top plane</strong></td>
<td><strong>Bottom plane</strong></td>
</tr>
<tr>
<td>001000 (above)</td>
<td>000000 (above)</td>
</tr>
<tr>
<td>000000 (below)</td>
<td>000100 (below)</td>
</tr>
<tr>
<td><strong>Right plane</strong></td>
<td><strong>Left plane</strong></td>
</tr>
<tr>
<td>000000 (to left of)</td>
<td>0000001 (to left of)</td>
</tr>
<tr>
<td>000010 (to right of)</td>
<td>000000 (to right of)</td>
</tr>
</tbody>
</table>
Cohen-Sutherland Algorithm (1/3)

- If we can neither trivially accept/reject (T/A, T/R), divide and conquer.
- Subdivide line into two segments; then T/A or T/R one or both segments:
  - use a clip edge to cut line
  - use outcodes to choose the edges that are crossed
    - for a given clip edge, if a line’s two outcodes differ in the corresponding bit, the line has one vertex on each side of the edge, thus crosses
  - pick an order for checking edges: top – bottom – right – left
  - compute the intersection point
    - the clip edge fixes either x or y
    - can substitute into the line equation
  - iterate for the newly shortened line, “extra” clips may happen (e.g., E-I at H)
Cohen-Sutherland Algorithm (2/3)

- \[ y = y_0 + \text{slope} \times (x - x_0) \] and \[ x = x_0 + \left( \frac{1}{\text{slope}} \right) \times (y - y_0) \]

**Algorithm:**

\[
\text{ComputeOutCode}(x_0, y_0, \text{outcode0});
\]
\[
\text{ComputeOutCode}(x_1, y_1, \text{outcode1});
\]

repeat

check for trivial reject or trivial accept
pick the outside point and bit ‘1’

if TOP then
\[
x = x_0 + (x_1 - x_0) \times (y_{\text{max}} - y_0) / (y_1 - y_0);
\]
\[
y = y_{\text{max}};
\]

else if BOTTOM then
\[
x = x_0 + (x_1 - x_0) \times (y_{\text{min}} - y_0) / (y_1 - y_0);
\]
\[
y = y_{\text{min}};
\]

else if RIGHT then
\[
y = y_0 + (y_1 - y_0) \times (x_{\text{max}} - x_0) / (x_1 - x_0);
\]
\[
x = x_{\text{max}};
\]

else if LEFT then
\[
y = y_0 + (y_1 - y_0) \times (x_{\text{min}} - x_0) / (x_1 - x_0);
\]
\[
x = x_{\text{min}};
\]

if (you picked \(x_0, y_0\)) then
\[
x_0 = x; \ y_0 = y; \ \text{ComputeOutCode}(x_0, y_0, \text{outcode0})
\]

else
\[
x_1 = x; \ y_1 = y; \ \text{ComputeOutCode}(x_1, y_1, \text{outcode1})
\]

until done
Similar algorithm for using 3D outcodes to clip against canonical parallel view volume:

\[
\begin{align*}
\text{xmin} &= \text{ymin} = -1; \quad \text{xmax} = \text{ymax} = 1; \\
\text{zmin} &= -1; \quad \text{zmax} = 0;
\end{align*}
\]

\text{ComputeOutCode}(x_0, y_0, z_0, \text{outcode}0);
\text{ComputeOutCode}(x_1, y_1, z_1, \text{outcode}1);
\text{repeat}
\quad \text{check for trivial reject or trivial accept}
\quad \text{pick the point that is outside the clip rectangle}
\quad \text{if TOP then}
\quad \quad x = x_0 + (x_1 - x_0) \times \frac{(\text{ymax} - y_0)}{(y_1 - y_0)};
\quad \quad z = z_0 + (z_1 - z_0) \times \frac{(\text{ymax} - y_0)}{(y_1 - y_0)};
\quad \quad y = \text{ymax};
\quad \text{else if BOTTOM then}
\quad \quad x = x_0 + (x_1 - x_0) \times \frac{(\text{ymin} - y_0)}{(y_1 - y_0)};
\quad \quad z = z_0 + (z_1 - z_0) \times \frac{(\text{ymin} - y_0)}{(y_1 - y_0)};
\quad \quad y = \text{ymin};
\quad \text{else if RIGHT then}
\quad \quad y = y_0 + (y_1 - y_0) \times \frac{(\text{xmax} - x_0)}{(x_1 - x_0)};
\quad \quad z = z_0 + (z_1 - z_0) \times \frac{(\text{xmax} - x_0)}{(x_1 - x_0)};
\quad \quad x = \text{xmax};
\quad \text{else if LEFT then}
\quad \quad y = y_0 + (y_1 - y_0) \times \frac{(\text{xmin} - x_0)}{(x_1 - x_0)};
\quad \quad z = z_0 + (z_1 - z_0) \times \frac{(\text{xmin} - x_0)}{(x_1 - x_0)};
\quad \quad x = \text{xmin};
\quad \text{else if NEAR then}
\quad \quad x = x_0 + (x_1 - x_0) \times \frac{(\text{zmax} - z_0)}{(z_1 - z_0)};
\quad \quad y = y_0 + (y_1 - y_0) \times \frac{(\text{zmax} - z_0)}{(z_1 - z_0)};
\quad \quad z = \text{zmax};
\quad \text{else if FAR then}
\quad \quad x = x_0 + (x_1 - x_0) \times \frac{(\text{zmin} - z_0)}{(z_1 - z_0)};
\quad \quad y = y_0 + (y_1 - y_0) \times \frac{(\text{zmin} - z_0)}{(z_1 - z_0)};
\quad \quad z = \text{zmin};
\quad \text{if (x_0, y_0, z_0 is the outer point) then}
\quad \quad x_0 = x; \quad y_0 = y; \quad z_0 = z;
\quad \text{ComputeOutCode}(x_0, y_0, z_0, \text{outcode}0)
\quad \text{else}
\quad \quad x_1 = x; \quad y_1 = y; \quad z_1 = z;
\quad \text{ComputeOutCode}(x_1, y_1, z_1, \text{outcode}1)
\text{until done}
Scan Conversion after Clipping

- Don’t round and then scan convert, because the line will have the wrong slope: calculate decision variable based on pixel chosen on left edge (remember: $y = mx + B$)
- Horizontal edge problem:
  - Clipping/rounding produces pixel A; to get pixel B, round up $x$ of the intersection of line with $y = y_{\text{min}} - \frac{1}{2}$ and pick pixel above:

$$y = y_{\text{min}}$$
$$y = y_{\text{min}} - \frac{1}{2}$$

Courtesy of Andries van Dam©
Sutherland-Hodgman Polygon Clipping

- The 2D Sutherland-Hodgman algorithm generalizes to higher dimensions
  - We can use it to clip polygons to the 3D view volume one plane at a time
  - Search for BALSA Clipping on youtube (0:50, 3:45)
Cyrus-Beck/Liang-Barsky Parametric Line Clipping (1/3)

- Use parametric line formulation:
  
  \[ P(t) = P_0 + (P_1 - P_0)t \]

- Determine where line intersects the infinite line formed by each clip rectangle edge:
  - solve for \( t \) multiple times depending on the number of clip edges crossed
  - decide which of these intersections actually occur on the rectangle

- For \( P_{E_i} \): use any point on edge \( E_i \)
Now solve for the value of $t$ at the intersection of $P_0P_1$ with the edge $E_i$:

First, substitute for $P(t)$:

Next, group terms and distribute dot product:

Let $D$ be the vector from $P_0$ to $P_1 = (P_1 - P_0)$, and solve for $t$:

- note that this gives a valid value of $t$ only if the denominator of the expression is nonzero.

For this to be true, it must be the case that:

- $N_i \neq 0$ (that is, the normal should not be 0; this could occur only as a mistake)
- $D \neq 0$ (that is, $P_1 \neq P_0$)
- $N_i \cdot D \neq 0$ (edge $E_i$ and line $D$ are not parallel; if they are, no intersection).

The algorithm checks these conditions.
Cyrus-Beck/Liang-Barsky Parametric Line Clipping (3/3)

- Eliminate $t$’s outside $[0,1]$ on the line
- Which remaining $t$’s produce interior intersections?
- Can’t just take the innermost $t$ values!
- Move from $P_0$ to $P_1$; for a given edge, just before crossing:
  - if $N_i \cdot D < 0 \Rightarrow$ Potentially Entering (PE);
    - if $N_i \cdot D > 0 \Rightarrow$ Potentially Leaving (PL)
  - Pick inner PE/PL pair: $t_E$ for $P_{PE}$ with max $t$, $t_L$ for $P_{PL}$ with min $t$, and
    - $t_E > 0, t_L < 1$
  - If $t_L < t_E$, no intersection
Cyrus-Beck/Liang-Barsky Line Clipping Algorithm

Pre-calculate $N_i$ and select $P_{E_i}$ for each edge;

for each line segment to be clipped
  if $P_1 = P_0$ then line is degenerate so clip as a point;
  else
    begin
      $t_E = 0$; $t_L = 1$;
      for each candidate intersection with a clip edge
        if $N_i \cdot D \neq 0$ then {Ignore edges parallel to line}
          begin
            calculate $t$; {of line and clip edge intersection}
            use sign of $N_i \cdot D$ to categorize as PE or PL;
            if PE then $t_E = \max(t_E, t)$;
            if PL then $t_L = \min(t_L, t)$;
          end
        if $t_E > t_L$ then return nil
        else return $P(t_E)$ and $P(t_L)$ as true clip intersections
    end

Parametric Line Clipping for Upright Clip Rectangle (1/2)

- \( D = P_1 - P_0 = (x_1 - x_0, y_1 - y_0) \)

- Leave \( P_{E_i} \) as an arbitrary point on clip edge: it’s a free variable and drops out

Calculations for Parametric Line Clipping Algorithm

<table>
<thead>
<tr>
<th>Clip Edge ( _i )</th>
<th>Normal ( N_i )</th>
<th>( P_{E_i} )</th>
<th>( P_0 - P_{E_i} )</th>
<th>( t = \frac{N_i \cdot (P_0 - P_{E_i})}{-N_i \cdot D} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>left: ( x = x_{\text{min}} )</td>
<td>(-1,0)</td>
<td>((x_{\text{min}}, y))</td>
<td>((x_0 - x_{\text{min}}, y_0 - y))</td>
<td>(-\frac{(x_0 - x_{\text{min}})}{(x_1 - x_0)})</td>
</tr>
<tr>
<td>right: ( x = x_{\text{max}} )</td>
<td>(1,0)</td>
<td>((x_{\text{max}}, y))</td>
<td>((x_0 - x_{\text{max}}, y_0 - y))</td>
<td>(-\frac{(x_0 - x_{\text{max}})}{(x_1 - x_0)})</td>
</tr>
<tr>
<td>bottom: ( y = y_{\text{min}} )</td>
<td>(0,-1)</td>
<td>((x, y_{\text{min}}))</td>
<td>((x_0 - x, y_0 - y_{\text{min}}))</td>
<td>(-\frac{(y_0 - y_{\text{min}})}{(y_1 - y_0)})</td>
</tr>
<tr>
<td>top: ( y = y_{\text{max}} )</td>
<td>(0,1)</td>
<td>((x, y_{\text{max}}))</td>
<td>((x_0 - x, y_0 - y_{\text{max}}))</td>
<td>(-\frac{(y_0 - y_{\text{max}})}{(y_1 - y_0)})</td>
</tr>
</tbody>
</table>
Examine $t$:

- Numerator is just the directed distance to an edge; sign corresponds to $OC$.
- Denominator is just the horizontal or vertical projection of the line, $dx$ or $dy$; sign determines PE or PL for a given edge.
- Ratio is constant of proportionality: “how far over” from $P_0$ to $P_1$ intersection is relative to $dx$ or $dy$. 
Pros and Cons

- Cohen-Sutherland is better when many trivial accepts/reject
- Otherwise, Liang-Barsky is better