Computer Graphics

- Clipping -

Philipp Slusallek
Clipping

• Motivation
  – Projected primitive might fall (partially) outside of screen window
    • E.g., if standing inside a building
  – Eliminate non-visible geometry early in the pipeline to process visible parts only
  – Happens after transformation from 3D to 2D
  – Must cut off parts outside the window
    • Outside geometry might not be representable (e.g., in fixed point)
    • Cannot draw outside of window (e.g., plotter (hardly exist anymore))
  – Must maintain information properly
    • Drawing the clipped geometry should give the correct results:
      – E.g., correct interpolation of colors across triangle even when clipped
    • Type of geometry might change
      – Cutting off a vertex of a triangle produces a quadrilateral (up to hexagon)
      – Might need to be split into triangles again
    • Polygons must remain closed after clipping
Line Clipping

• Definition of clipping
  – Cut off parts of objects which lie outside/inside of a defined region
  – Often clip against viewport (2D) or canonical view-volume (3D)

• Let’s focus first on lines only
### Brute-Force Method

- **Brute-force line clipping at the viewport**
  - If both end points $p_b$ and $p_e$ are inside viewport
    - Accept the whole line
  - Otherwise, clip the line at each edge
    - $p_{\text{intersection}} = p_b + t_{\text{line}}(p_e - p_b) = e_b + t_{\text{edge}}(e_e - e_b)$
    - Solve for $t_{\text{line}}$ and $t_{\text{edge}}$
      - Intersection within segment if both $0 \leq t_{\text{line}}, t_{\text{edge}} \leq 1$
    - Replace suitable end points for the line by the intersection point
  - Unnecessarily tests many cases that are irrelevant
Cohen-Sutherland (1974)

- **Advantage: divide and conquer**
  - Efficient trivial accept and trivial reject
  - Non-trivial case: divide and test

- **Outcodes of points**
  - Bit encoding (outcode, OC)
    - Each viewport edge defines a half space
    - Set bit if vertex is outside w.r.t. that edge

- **Trivial cases**
  - Trivial accept: both are in viewport
    - \((\text{OC}(p_b) \text{ OR } \text{OC}(p_e)) = 0\)
  - Trivial reject: both lie outside w.r.t. *at least one common edge*
    - \((\text{OC}(p_b) \text{ AND } \text{OC}(p_e)) \neq 0\)
  - Line has to be clipped to all edges where XOR bits are set, i.e. the points lies on different sides of that edge
    - \(\text{OC}(p_b) \text{ XOR } \text{OC}(p_e)\)
Cohen-Sutherland

- **Clipping of line (p1, p2)**

  \[
  \text{ocl} = \text{OC}(p1); \quad \text{oc}2 = \text{OC}(p2); \quad \text{edge} = 0;
  \]
  
  do {
  
  if (((ocl AND oc2) != 0)) \text{ // trivial reject of remaining segment}
  
  return REJECT;

  else if (((ocl OR oc2) == 0)) \text{ // trivial accept of remaining segment}
  
  return (ACCEPT, p1, p2);

  if (((ocl XOR oc2)[edge]) {
  
  if (ocl[edge]) \text{ // p1 outside}
  
  \{ p1 = \text{cut}(p1, p2, edge); \text{ocl} = \text{OC}(p1); \}

  else \text{ // p2 outside}
  
  \{ p2 = \text{cut}(p1, p2, edge); \text{oc}2 = \text{OC}(p2); \}
  
  }
  
  } while (++edge < 4); \text{ // Not the most efficient solution}

  return ((ocl OR oc2) == 0) ? (ACCEPT, p1, p2) : REJECT;

- **Intersection calculation for \( x = x_{\text{min}} \)**

  \[
  \frac{y - y_b}{y_e - y_b} = \frac{x_{\text{min}} - x_b}{x_e - x_b}
  \]

  \[
  y = y_b + (x_{\text{min}} - x_b) \frac{y_e - y_b}{x_e - x_b}
  \]
Cyrus-Beck (1978)

- **Parametric line-clipping algorithm**
  - Only convex polygons: max 2 intersection points
  - Use edge orientation, via “normals” pointing out

- **Idea: clipping against polygons**
  - Clip line \( p = p_b + t_i (p_e - p_b) \) against each edge
  - Intersection points sorted by parameter \( t_i \)
  - Select
    - \( t_{in} \): entry point \( ((p_e - p_b) \cdot N_i < 0) \) with largest \( t_i \)
    - \( t_{out} \): exit point \( ((p_e - p_b) \cdot N_i > 0) \) with smallest \( t_i \)
  - If \( t_{out} < t_{in} \), line lies completely outside (akin to ray-box intersect.)

- **Intersection calculation**
  \[
  (p - p_{edge}) \cdot N_i = 0 \\
  t_i (p_e - p_b) \cdot N_i + (p_b - p_{edge}) \cdot N_i = 0 \\
  t_i = \frac{(p_{edge} - p_b) \cdot N_i}{(p_e - p_b) \cdot N_i}
  \]
Liang-Barsky (1984)

- **Cyrus-Beck for axis-aligned rectangles**
  - Using window-edge coordinates (with respect to an edge $T$)
  $$WEC_T(p) = (p - p_T) \cdot N_T$$

- **Example: top ($y = y_{\text{max}}$)**

  $$N_T = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \quad p_b - p_T = \begin{pmatrix} x_b - x_{\text{max}} \\ y_b - y_{\text{max}} \end{pmatrix}$$

  $$WEC_T(p_b) = \frac{(p_b - p_T) \cdot N_T}{(p_b - p_e) \cdot N_T} = \frac{WEC_T(p_b)}{WEC_T(p_b) - WEC_T(p_e)} = \frac{y_b - y_{\text{max}}}{y_b - y_e}$$

- Window-edge coordinate (WEC): decision function for an edge
  - Directed distance to edge
    - Only sign matters, similar to Cohen-Sutherland opcode
  - Sign of the dot product determines whether the point is in or out
  - Normalization unimportant
Line Clipping - Summary

- **Cohen-Sutherland, Cyrus-Beck, and Liang-Barsky algorithms readily extend to 3D**

- **Cohen-Sutherland algorithm**
  - Efficient when majority of lines can be trivially accepted / rejected
    - Very large clip rectangles: almost all lines inside
    - Very small clip rectangles: almost all lines outside
    - Repeated clipping for remaining lines
    - Testing for 2D/3D point coordinates

- **Cyrus-Beck (Liang-Barsky) algorithms**
  - Efficient when many lines must be clipped
  - Testing for 1D parameter values
  - Testing intersections always for all clipping edges (in the Liang-Barsky trivial rejection testing possible)
Polygon Clipping

- Extended version of line clipping
  - Condition: polygons have to remain closed
    - Filling, hatching, shading, ...
Sutherland-Hodgeman (1974)

- **Idea**
  - Iterative clipping against each edge in sequence

- Four different local operations based on sides of $p_{i-1}$ and $p_i$

```
<table>
<thead>
<tr>
<th>$p_{i-1}$</th>
<th>$p_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>inside output: $p_i$</td>
<td>outside</td>
</tr>
<tr>
<td>$p_i$</td>
<td>$p_{i-1}$</td>
</tr>
<tr>
<td>inside output: $p$</td>
<td>outside</td>
</tr>
<tr>
<td>$p$</td>
<td>$p_i$</td>
</tr>
<tr>
<td>inside output: -</td>
<td>outside</td>
</tr>
<tr>
<td>$p$</td>
<td>$p_i$</td>
</tr>
<tr>
<td>inside output: $p$</td>
<td>outside</td>
</tr>
<tr>
<td>$p$</td>
<td>$p_i$</td>
</tr>
</tbody>
</table>
| 1st output: $p$    | 2nd output: $p_i$
```
Enhancements

• **Recursive polygon clipping**
  – Pipelined Sutherland-Hodgeman

```
p_0, p_1, ...
```

```
Top → Bottom → Left → Right
```

```
p_0, p_1, ...
```

• **Problems**
  – Degenerated polygons/edges
    • Elimination by post-processing, if necessary
Other Clipping Algorithms

- **Weiler & Atherton (’77)**
  - Arbitrary concave polygons with holes against each other

- **Vatti (’92)**
  - Also with self-overlap

- **Greiner & Hormann (TOG ’98)**
  - Simpler and faster as Vatti
  - Also supports Boolean operations
  - Idea:
    - Odd winding number rule
      - Intersection with the polygon leads to a winding number $\pm 1$
    - Walk along both polygons
    - Alternate winding number value
    - Mark point of entry and point of exit
    - Combine results

Non-zero WN: in
Even WN: out
A in B

B in A

(A in B) U (B in A)
3D Clipping agst. View Volume

- **Requirements**
  - Avoid unnecessary rasterization
  - Avoid overflow on transformation at fixed point!

- **Clipping against viewing frustum**
  - Enhanced Cohen-Sutherland with 6-bit outcode
  - After perspective division
    - $-1 < y < 1$
    - $-1 < x < 1$
    - $-1 < z < 0$
  - Clip against side planes of the canonical viewing frustum
  - Works analogously with Liang-Barsky or Sutherland-Hodgeman
3D Clipping agst. View Volume

- **Clipping in homogeneous coordinates**
  - Use canonical view frustum, but avoid costly division by $W$
  - Inside test with a linear distance function (WEC)
    - Left: $X / W > -1 \Rightarrow W + X = WEC_L(p) > 0$
    - Top: $Y / W < 1 \Rightarrow W - Y = WEC_T(p) > 0$
    - Back: $Z / W > -1 \Rightarrow W + Z = WEC_B(p) > 0$
    - ...
  - Intersection point calculation (before homogenizing)
    - Test: $WEC_L(p_b) > 0$ and $WEC_L(p_e) < 0$
    - Calculation:
      
      $$WEC(p_b + t(p_e - p_b)) = 0$$
      $$W_b + t(W_e - W_b) + X_b + t(X_e - X_b) = 0$$
      $$t = \frac{W_b + X_b}{(W_b + X_b) - (W_e + X_e)} = \frac{WEC_L(p_b)}{WEC_L(p_b) - WEC_L(p_e)}$$

- **Negative w**
  - Points with $w < 0$ or lines with $w_b < 0$ and $w_e < 0$
    - Negate and continue
  - Lines with $w_b \cdot w_e < 0$ (NURBS)
    - Line moves through infinity
      - External „line“
    - Clipping two times
      - Original line
      - Negated line
    - Generates up to two segments
Practical Implementations

• **Combining clipping and scissoring**
  - Clipping is expensive and should be avoided
    - Intersection calculation
    - Variable number of new points, new triangles
  - Enlargement of clipping region
    - (Much) larger than viewport, but
    - Still avoiding overflow due to fixed-point representation
  - Result
    - Less clipping
    - Applications should avoid drawing objects that are outside of the viewport/viewing frustum
    - Objects that are still partially outside will be implicitly clipped during rasterization
    - Slight penalty because they will still be processed (triangle setup)