Clipping

CPSC 314

The Rendering Pipeline

Geometry Processing

Geometry Database ➔ Model/View Transform. ➔ Lighting ➔ Perspective Transform. ➔ Clipping

Scan Conversion ➔ Texturing ➔ Depth Test ➔ Blending ➔ Frame-buffer

Rasterization

Fragment Processing

© Wolfgang Heidrich
Line Clipping

Purpose

- Originally: 2D
  - Determine portion of line inside an axis-aligned rectangle (screen or window)
- 3D
  - Determine portion of line inside axis-aligned parallelepiped (viewing frustum in NDC)
  - Simple extension to the 2D algorithms
Line Clipping

Outcodes (Cohen, Sutherland ’74)

- 4 flags encoding position of a point relative to top, bottom, left, and right boundary

- E.g.:
  - \(\text{OC}(p_1)=0010\)
  - \(\text{OC}(p_2)=0000\)
  - \(\text{OC}(p_3)=1001\)

<table>
<thead>
<tr>
<th>(p_1)</th>
<th>(p_2)</th>
<th>(p_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0110</td>
<td>0100</td>
<td>0101</td>
</tr>
</tbody>
</table>

Line Clipping

Line segment:

- \((p_1,p_2)\)

Trivial cases:

- \(\text{OC}(p_1)==0 \&\& \text{OC}(p_2)==0\)
  - Both points inside window, thus line segment completely visible (trivial accept)

- \((\text{OC}(p_1) \& \text{OC}(p_2))!=0\)
  - There is (at least) one boundary for which both points are outside (same flag set in both outcodes)
    - Thus line segment completely outside window (trivial reject)
Line Clipping

$\alpha$-Clipping

- Handling of all the non-trivial cases
- Improvement of earlier algorithms (Cohen/Sutherland, Cyrus/Beck, Liang/Barsky)
- Define *window-edge-coordinates* of a point $p=(x,y)^T$
  - $\text{WEC}_L(p)=x-x_{\text{min}}$
  - $\text{WEC}_R(p)=x_{\text{max}}-x$
  - $\text{WEC}_B(p)=y-y_{\text{min}}$
  - $\text{WEC}_T(p)=y_{\text{min}}-y$

Negative if outside!
Line Clipping

\(\alpha\)-Clipping

- Line segment defined as: \(p1 + \alpha(p2-p1)\)
- Intersection point with one of the borders (say, left):

\[
x_1 + \alpha(x_2 - x_1) = x_{\text{min}} \iff \alpha = \frac{x_{\text{min}} - x_1}{x_2 - x_1}
\]

\[
= \frac{x_{\text{min}} - x_1}{(x_2 - x_{\text{min}}) - (x_1 - x_{\text{min}})} = \frac{\text{WEC}_L(x_1)}{\text{WEC}_L(x_1) - \text{WEC}_L(x_2)}
\]

Line Clipping

\(\alpha\)-Clipping: algorithm

alphaClip( p1, p2, window ) {
    Determine window-edge-coordinates of p1, p2
    Determine outcodes OC(p1), OC(p2)

    Handle trivial accept and reject

    \(\alpha1 = 0;\) // line parameter for first point
    \(\alpha2 = 1;\) // line parameter for second point
    ...
}
Line Clipping

$\alpha$-Clipping: algorithm (cont.)

... 

// now clip point p1 against all edges
if( OC(p1) & LEFT_FLAG ) {
    \[ \alpha = \frac{\text{WEC}_L(p1)}{(\text{WEC}_L(p1) - \text{WEC}_L(p2))} \]
    \[ \alpha_1 = \max(\alpha_1, \alpha) \];
}

Similarly clip p1 against other edges
...

Line Clipping

$\alpha$-Clipping: example for clipping p1

Start configuration  After clipping to left  After clipping to top
Line Clipping

\( \alpha \)-Clipping: algorithm (cont.)

\[
\ldots
\]

// now clip point \( p_2 \) against all edges

if( \text{OC}(p_2) \& \text{LEFT\_FLAG} ) {
\begin{align*}
\alpha &= \frac{\text{WEC}_L(p_2)}{(\text{WEC}_L(p_1) - \text{WEC}_L(p_2))}; \\
\alpha_2 &= \min(\alpha_2, \alpha);
\end{align*}
}

Similarly clip \( p_1 \) against other edges

\[
\ldots
\]

Line Clipping

\( \alpha \)-Clipping: algorithm (cont.)

\[
\ldots
\]

// wrap-up

\[
\begin{align*}
\text{if}(\alpha_1 > \alpha_2 ) & \\
\text{no output;}
\end{align*}
\]

else

output line from \( p_1 + \alpha_1(p_2-p_1) \) to \( p_1 + \alpha_2(p_2-p_1) \)

\[
\ldots
\]

// end of algorithm
Line Clipping

Example

Start configuration
After clipping p1
After clipping p2

Line Clipping

Another Example

Start configuration
After clipping p1
After clipping p2
**Line Clipping in 3D**

*Approach:*
- Clip against parallelepiped in NDC (*after* perspective transform)
- Means that the clipping volume is always the same!
  - OpenGL: \(x_{\text{min}}=y_{\text{min}}=-1, x_{\text{max}}=y_{\text{max}}=1\)
- Boundary lines become boundary planes
  - *But outcodes and WECs still work the same way*
  - *Additional front and back clipping plane*
  - \(z_{\text{min}}=0, z_{\text{max}}=1\) in OpenGL

---

**Line Clipping**

*Extensions*
- Algorithm can be extended to clipping lines against
  - *Arbitrary convex polygons (2D)*
  - *Arbitrary convex polytopes (3D)*
Line Clipping

Non-convex clipping regions

- E.g.: windows in a window system!

Problem: arbitrary number of visible line segments

Different approaches:
- Break down polygon into convex parts
- Scan convert for full window, and discard hidden pixels
Polygon Clipping

Objective

- 2D: clip polygon against rectangular window
  - Or general convex polygons
  - Extensions for non-convex or general polygons
- 3D: clip polygon against parallelepiped

Not just clipping all boundary lines

- May have to introduce new line segments
Polygon Clipping

Classes of Polygons

- Triangles
- Convex
- Concave
- Holes and self-intersection

Sutherland/Hodgeman Algorithm (’74)

- Arbitrary convex or concave object polygon
  - Restriction to triangles does not simplify things
- Convex subject polygon (window)
Polygon Clipping

**Sutherland/Hodgeman Algorithm ('74)**

- Approach: clip object polygon independently against all edges of subject polygon

---

Clipping against one edge:

```java
clipPolygonToEdge( p[n], edge ) {
  for( i= 0 ; i< n ; i++ ) {
    if( p[i] inside edge ) {
      if( p[i-1] inside edge ) // p[-1]= p[n-1]
        output p[i];
      else {
        p= intersect( p[i-1], p[i], edge );
        output p, p[i];
      }
    }
  }
}
```
Polygon Clipping

Clipping against one edge (cont)

- p[i] inside: 2 cases

<table>
<thead>
<tr>
<th>inside</th>
<th>outside</th>
<th>inside</th>
<th>outside</th>
</tr>
</thead>
<tbody>
<tr>
<td>p[i-1]</td>
<td></td>
<td>p[i]</td>
<td></td>
</tr>
<tr>
<td>p[i]</td>
<td></td>
<td>p[i-1]</td>
<td></td>
</tr>
</tbody>
</table>

Output: p[i]  
Output: p, p[i]

... 
else { // p[i] is outside edge 
  if( p[i-1] inside edge ) { 
    p = intersect(p[i-1], p[i], edge ); 
    output p; 
  } 
} // end of algorithm
Polygon Clipping

Clipping against one edge (cont)

- p[i] outside: 2 cases

Output: p

Output: nothing

Example

Polygon Clipping
**Polygon Clipping**

**Sutherland/Hodgeman Algorithm**
- Inside/outside tests: outcodes
- Intersection of line segment with edge: window-edge coordinates
- Similar to Cohen/Sutherland algorithm for line clipping

---

**Discussion:**
- Works for concave polygons
- But generates degenerate cases
Polygon Clipping

Sutherland/Hodgeman Algorithm

- Discussion:
  - Clipping against individual edges independent
    - Great for hardware (pipelining)
  - All vertices required in memory at the same time
    - Not so good, but unavoidable
    - Another reason for using triangles only in hardware rendering

Polygon Clipping

Sutherland/Hodgeman Algorithm

- For Rendering Pipeline:
  - Re-triangulate resulting polygon (can be done for every individual clipping edge)
Polygon Clipping

Other Polygon Clipping Algorithms

- Weiler/Aetherton '77:
  - Arbitrary concave polygons with holes both as subject and as object polygon
- Vatti '92:
  - Self intersection allowed as well
- ... many more
  - Improved handling of degenerate cases
  - But not often used in practice due to high complexity
Coming Up:

**Tuesday, Oct 9:**
- Hidden surface removal / visibility

**Thursday, Oct 11:**
- Scan Conversion