Chapter 7

Clipping

Rendering Pipeline

- Discard geometry outside viewport window

Copyright Alla Sheffer UBC 2011
Line and Polygon Clipping

Line/Polygon Clipping (2D)

Problem:
Given a 2D line/polygon and a window, clip the line/polygon to their regions that are inside the window.

- Objectives
  - Efficiency
  - (Parallelization)

- Two approaches
  - Explicit (continuous setting)
  - Implicit (discrete setting) – part of scan conversion

Explicit Solution: Line Segments

- Intersection of convex regions is convex
  - Why?

- L & D are convex - intersection is convex
  - single connected segment of L

- Clipping uses intersections of L with four boundary segments of window D
Basic Method

\[
\text{Clip}(P_0, P_1, x_{\text{min}}, x_{\text{max}}, y_{\text{min}}, y_{\text{max}})
\]

if \((P_0 \text{ and } P_1 \text{ inside window})\) then
draw\((P_0, P_1)\);
test if segment \((P_0, P_1)\) intersects any of the edges
if not, return;
else let \(P_i\) be the first intersection found
\[
\text{Clip}(P_0, P_i, x_{\text{min}}, x_{\text{max}}, y_{\text{min}}, y_{\text{max}});\]
\[
\text{Clip}(P_i, P_1, x_{\text{min}}, x_{\text{max}}, y_{\text{min}}, y_{\text{max}});
\]
end

- Works, but inefficient for lines OUTSIDE \(D\)
- Four intersection tests
- Note: need special care for vertices ON window edges

Segment-Segment Intersection

\[
\begin{align*}
G_1 &= \begin{cases} 
x^1(t) &= x^0_1 + (x^1_1 - x^0_1)t \\
y^1(t) &= y^0_1 + (y^1_1 - y^0_1)t
\end{cases} \quad t \in [0,1] \\
G_2 &= \begin{cases} 
x^2(r) &= x^0_2 + (x^1_2 - x^0_2)r \\
y^2(r) &= y^0_2 + (y^1_2 - y^0_2)r
\end{cases} \quad r \in [0,1]
\end{align*}
\]

Intersection: \(x\) & \(y\) values equal in both representations - two linear equations in two unknowns \((t,r)\)
test if resulting \(r\) & \(t\) are inside the \([0,1]\) range

\[
\begin{align*}
x^0_1 + (x^1_1 - x^0_1)t &= x^0_2 + (x^1_2 - x^0_2)r \\
y^0_1 + (y^1_1 - y^0_1)t &= y^0_2 + (y^1_2 - y^0_2)r
\end{align*}
\]
Intersection with axis-aligned lines

\[
\begin{align*}
G_1 &= \begin{cases}
    x^1(t) = x_0 + (x_1 - x_0) t, & t \in [0,1], \\
y^1(t) = y_0 + (y_1 - y_0) t
\end{cases} \\
G_2 &= \begin{cases}
    x^2(r) = x_0^2, \\
y^2(r) = y_0^2 + (y_1^2 - y_0^2) r
\end{cases}, \quad r \in [0,1]
\end{align*}
\]

Intersection: \( x \) \& \( y \) values equal in both representations - two linear equations in two unknowns \((r,t)\)

\[
\begin{align*}
x_0 + (x_1 - x_0) t &= x_0^2, \\
t &= \frac{x_0^2 - x_0}{x_1 - x_0}, \quad \text{if } t < 0 \text{ or } t > 1 \text{ no intersection} \\
y_0 + (y_1 - y_0) t &= y_0^2 + (y_1^2 - y_0^2) r, \quad \text{(relevantly only for segments)}
\end{align*}
\]

Line Clipping

\[
\begin{align*}
x &= x_{\text{min}}, \quad y = y_{\text{max}} \\
x &= x_{\text{max}}, \quad y = y_{\text{min}}
\end{align*}
\]
Cohen-Sutherland Algorithm

**Purpose:**
Fast treatment of line segments that are trivially inside/outside window.

\( P = (x, y) \) - point to be classified against window \( D \)

**Idea:** Assign to \( P \) a binary code consisting of a bit for each edge of \( D \), using lookup table:

<table>
<thead>
<tr>
<th>bit</th>
<th>Condition 1</th>
<th>Condition 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( y &lt; y_{\text{min}} )</td>
<td>( y \geq y_{\text{min}} )</td>
</tr>
<tr>
<td>2</td>
<td>( y &gt; y_{\text{max}} )</td>
<td>( y \leq y_{\text{max}} )</td>
</tr>
<tr>
<td>3</td>
<td>( x &gt; x_{\text{max}} )</td>
<td>( x \leq x_{\text{max}} )</td>
</tr>
<tr>
<td>4</td>
<td>( x &lt; x_{\text{min}} )</td>
<td>( x \geq x_{\text{min}} )</td>
</tr>
</tbody>
</table>

Copyright Alla Sheffer UBC 2011
Cohen-Sutherland Algorithm (cont’d)

Given \( L \) from \((x_0, y_0)\) to \((x_1, y_1)\) & rectangle \( D \).

If bitwise and of the codes of \((x_0, y_0)\) and \((x_1, y_1)\) is not zero, or the bitwise or is zero,
then \( L \) can be trivially handled (it is either totally outside or totally inside \( D \)).

Why?

Cohen-Sutherland Algorithm (cont’d)

\[
\begin{align*}
\text{C-S-Clip}(P_0 = (x_0, y_0), P_1 = (x_1, y_1), x_{\text{max}}, y_{\text{max}}, x_{\text{min}}, y_{\text{min}}) & \text{ (assumes } x_0 < x_1) \\
C_x & \leftarrow \text{code}(P_0) \\
C_y & \leftarrow \text{code}(P_1) \\
\text{if } ((C_x \text{ and } C_y) \neq 0) & \text{ then return}; \\
\text{if } ((C_x \text{ or } C_y) = 0) & \text{ then draw}(P_0, P_1); \\
\text{else if (OutsideWindow}(P_1) & \text{ then begin} \\
\text{Edge} & \leftarrow \text{Window boundary of leftmost non-zero bit of } C_y; \\
P_2 & \leftarrow P_0 \cap \text{Edge}; \\
\text{C-S-Clip}(P_2, P_1, x_{\text{max}}, y_{\text{max}}, x_{\text{min}}, y_{\text{min}}); \\
\text{end} \\
\text{end} \\
\text{Edge} & \leftarrow \text{Window boundary of leftmost non-zero bit of } C_x; \\
P_2 & \leftarrow P_0 \cap \text{Edge}; \\
\text{C-S-Clip}(P_2, P_1, x_{\text{max}}, y_{\text{max}}, x_{\text{min}}, y_{\text{min}}); \\
\text{end}
\end{align*}
\]
3D clipping

- Determine portion of line inside axis-aligned parallelepiped (viewing frustum in NDC)
- Simple extension to 2D algorithms
- After perspective transform
  - means that clipping volume always the same
    - xmin=ymin= -1, xmax=ymax= 1 in OpenGL
  - boundary lines become boundary planes
    - but bit-codes still work the same way
    - additional front and back clipping plane
      - zmin = -1, zmax = 1 in OpenGL

Triangle Clipping

- How does intersection of rectangle & triangle looks like?
  - How many sides?

- How to expand clipping to triangles?
  - Hint: it is convex
  - Will develop on the board...
Cohen-Sutherland Algorithm for convex polygons

\[
\text{C-S-Clip}(\text{poly} = (P_0, \ldots, P_n, x_{\text{min}}, x_{\text{max}}, y_{\text{min}}, y_{\text{max}}))
\]

for \( i = 1 \) to \( n \)

\( C_i \leftarrow \text{code}(P_i) \)

if \((C_0 \lor C_1 \lor \ldots \lor C_n) \neq 0\) then return;

if \((C_0 \land C_1 \land \ldots \land C_n) \equiv 0\) then draw(poly);

else

for \( i = 1 \) to \( n \)

if (OutsideWindow(P)) then

begin

Edge \( \leftarrow \text{Window boundary of leftmost non-zero bit of } C_i \);

\( P_{i,1,i} \leftarrow P_{i,1} \cap \text{Edge} \);

\( P_{i,2,i} \leftarrow P_{i,1} \cap \text{Edge} \);

\( \text{C-S-Clip}(P_0, \ldots, P_{i,1,i}, P_{i,2,i}, \ldots, P_n, x_{\text{min}}, x_{\text{max}}, y_{\text{min}}, y_{\text{max}}) \);

end

end

Other Geometric Problems

- Questions: How can these ideas be used to design an algorithm for checking if:
  - a point is inside a (convex) polygon?
    - E.g. For collision detection
  - A (convex) polygon is inside/intersects/outside a (convex) polygon?