Chapter 7

Line and Polygon Clipping

Rendering Pipeline

Geometry Content

Model/View Transform

Lighting

Perspective Transform

Clipping

Scan Conversion

Texturing

Depth Test

Blending

Frame-buffer

Geometric Content/View Transform

Lighting

Perspective Transform

Clipping

Scan Conversion

Texturing

Depth Test

Blending

Fragment Processing

Rasterization

Discard geometry outside viewport window

 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

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

\[ \begin{align*}
L & = \{ (x, y) \mid y \geq y_0 \} \\
D & = \{ (x, y) \mid (x, y) \text{ in window} \}
\end{align*} \]

Clip \( \{ (x, y) \mid (x, y) \text{ in window} \} \)

\( \begin{align*}
& L \cap D \\
& \text{ works, but inefficient for lines OUTSIDE D} \\
& \text{ Four intersection tests} \\
& \text{ Note: need special care for vertices ON window edges}
\end{align*} \)

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

Segment-Segment Intersection

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

Test if resulting \( r \) \& \( t \) are inside the \([0,1]\) range

Copyright Alla Sheffer UBC 2011
Intersection with axis-aligned lines

\[
G_1 = \left\{ \begin{array}{ll}
0 & x = x_1^* \\
1 & x > x_1^*
\end{array} \right.
\]

\[
G_2 = \left\{ \begin{array}{ll}
0 & y = y_1^* \\
1 & y > y_1^*
\end{array} \right.
\]

Intersection: \( x \) and \( y \) values equal in both representations - two linear equations in two unknowns (c.f.)

Line Clipping

<table>
<thead>
<tr>
<th>Point to be classified against window</th>
</tr>
</thead>
<tbody>
<tr>
<td>( (x, y) )</td>
</tr>
<tr>
<td>( x_{\min} )</td>
</tr>
<tr>
<td>( x_{\max} )</td>
</tr>
<tr>
<td>( y_{\min} )</td>
</tr>
<tr>
<td>( y_{\max} )</td>
</tr>
</tbody>
</table>

Cohen-Sutherland Algorithm

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

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>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Copyright Alla Sheffer UBC 2011
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
  - $x_{min} = y_{min} = -1$, $x_{max} = y_{max} = 1$ in OpenGL
  - boundary lines become boundary planes
  - but bit-codes still work the same way
  - additional front and back clipping plane
  - $z_{min} = -1$, $z_{max} = 1$ 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

```
C-S-Clip poly = P_0, P_1, P_2, ..., P_n, P_0
begin
  for i = 0 to n do
    C_i = code (P_i)
    if ((C_0 and C_1 and ... and C_n) = 0) then return;
    else for i = 1 to n if (Outside(Window (P_i)) = 1) then return;
    begin
      Edge = Window boundary of leftmost non-zero bit of C_i;
      P_1 = P_i + (I/E)_Edge;
      P_2 = P_i - (I/E)_Edge;
      C-S-Clip P_0, P_1, P_2, ..., P_i-1, P_{i+1}, ..., P_n, P_0
    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?