Introduction to Computer Graphics
5. Clipping

National Chiao Tung Univ, Taiwan
By: I-Chen Lin, Assistant Professor

Objectives

- Introduce basic implementation strategies

- 2D Clipping
  - Lines
  - Polygons

- Clipping in 3D
Clipping

- 2D against clipping window
- 3D against clipping volume
- Easy for line segments polygons
- Hard for curves and text
  - Convert to lines or polygons first
Clipping 2D Line Segments

- Brute force approach: compute intersections with all sides of clipping window
  - Inefficient: one division per intersection
Cohen-Sutherland Algorithm

- Idea: eliminate as many cases as possible without computing intersections
- Start with four lines that determine the sides of the clipping window

\[ x = x_{\text{min}} \]
\[ y = y_{\text{max}} \]
\[ x = x_{\text{max}} \]
\[ y = y_{\text{min}} \]
Case 1

- Case 1: both endpoints of line segment inside all four lines
  - Draw (accept) line segment as is

\begin{align*}
x &= x_{\text{min}} \\
y &= y_{\text{min}} \\
y &= y_{\text{max}} \\
x &= x_{\text{max}}
\end{align*}
Case 2

- Case 2: both endpoints outside all lines and on same side of a line
  - Discard (reject) the line segment

\[ x = x_{\text{min}} \]
\[ y = y_{\text{max}} \]

\[ x = x_{\text{max}} \]

\[ y = y_{\text{min}} \]
Case 3

- One endpoint inside, one outside
  - Must do at least one intersection

```
x = x_{min}  
  \hline
|     |     |     |
  \hline
|     | y = y_{max} |     |
|     |             |     |
  \hline
y = y_{min}  
  \hline
|     | x = x_{max} |     |
|     |             |     |
  \hline
```
Case 4

- Both outside
  - May have part inside
  - Must do at least one intersection

\[
x = x_{\text{min}} \quad y = y_{\text{max}}
\]

\[
x = x_{\text{max}} \quad y = y_{\text{min}}
\]
Defining Outcodes

- For each endpoint, define an outcode
  - \([b_0 \ b_1 \ b_2 \ b_3]\)
- Outcodes divide space into 9 regions
- Computation of outcode requires at most 4 subtractions

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>
| 1001 | 1000 | 1010 | b0 = 1 if \(y > y_{\text{max}}\), 0 otherwise
| 0001 | 0000 | 0010 | b1 = 1 if \(y < y_{\text{min}}\), 0 otherwise
| 0101 | 0100 | 0110 | b2 = 1 if \(x > x_{\text{max}}\), 0 otherwise
|   |   |   | b3 = 1 if \(x < x_{\text{min}}\), 0 otherwise
Using Outcodes

- AB: outcode(A) = outcode(B) = 0

- Accept line segment
Using Outcodes

- CD: outcode (C) = 0, outcode(D) ≠ 0
  - Compute intersection
  - Location of 1 in outcode(D) determines which edge to intersect with
  - Note if there were a segment from C to a point in a region with 2 ones in outcode, we might have to do two intersections
Using Outcodes

- EF: outcode(E) logically ANDed with outcode(F) (bitwise) ≠ 0
  - Both outcodes have a 1 bit in the same place
  - Line segment is outside of corresponding side of clipping window
  - reject
Using Outcodes

- GH and IJ: same outcodes, neither zero but logical AND yields zero
  - Shorten line segment by intersecting with one of sides of window
  - Compute outcode of intersection (new endpoint of shortened line segment)
  - Reexecute algorithm

<table>
<thead>
<tr>
<th></th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0101</td>
<td>0001</td>
<td>0000</td>
<td>0010</td>
<td>0010</td>
<td></td>
</tr>
<tr>
<td>0101</td>
<td>0001</td>
<td>0000</td>
<td>0010</td>
<td>0010</td>
<td></td>
</tr>
<tr>
<td>1001</td>
<td>1000</td>
<td>1000</td>
<td>1010</td>
<td>1010</td>
<td></td>
</tr>
</tbody>
</table>
Efficiency

- In many applications, the clipping window is small relative to the size of the entire data base.
  - Most line segments can be eliminated based on their outcodes.
Polygon Clipping

- Not as simple as line segment clipping
  - Clipping a line segment yields at most one line segment
  - Clipping a polygon can yield multiple polygons

- However, clipping a convex polygon can yield at most one other polygon
Tessellation and Convexity

- One strategy is to replace nonconvex (concave) polygons with a set of triangular polygons (a tessellation).

- Also makes fill easier.
Clipping as a Black Box

Consider line segment clipping as a process that takes in two vertices and produces either no vertices or the vertices of a clipped line segment.
Pipeline Clipping of Line Segments

- Clipping against each side of window is independent of other sides
  - Can use four independent clippers in a pipeline
Pipeline Clipping of Polygons

- Sutherland-Hodgman algorithm
- Strategy used in SGI Geometry Engine
- Small increase in latency
Cohen-Sutherland Method in 3D

- Use 6-bit outcodes
  - When needed, clip line segment against planes
Check for outcodes:

\[-1 \leq x_p \leq 1, \quad -1 \leq y_p \leq 1, \quad -1 \leq z_p \leq 1\]

Since

\[
\begin{bmatrix}
  x \\
  y \\
  z \\
  1
\end{bmatrix}
\Rightarrow \ldots \Rightarrow
\begin{bmatrix}
  x_h \\
  y_h \\
  z_h \\
  h
\end{bmatrix}
\Rightarrow \ldots \Rightarrow
\begin{bmatrix}
  \frac{x_h}{h} \\
  \frac{y_h}{h} \\
  \frac{z_h}{h} \\
  1
\end{bmatrix} = \begin{bmatrix}
  x_p \\
  y_p \\
  z_p \\
  1
\end{bmatrix}
\]

To avoid unnecessary float division, We can check

\[-h \leq x_h \leq h, \quad -h \leq y_h \leq h, \quad -h \leq z_h \leq h\]
Cohen-Sutherland Method in 3D

- If `outcode(A)==outcode(B)==0`
  - Accept the whole line segment.
- If `(outcode(A) and outcode(B))!=0`
  - Reject the line segment.

- Other cases
  - Calculate an intersection (according to outcode bits)
  - Then check outcode again

- Note: use parametric forms
  - \[ x_h = x_{ha} + (x_{hb} - x_{ha})u \]
  - \[ y_h = y_{ha} + (y_{hb} - y_{ha})u \]
  - \[ z_h = z_{ha} + (z_{hb} - z_{ha})u \]
  - \[ h = h_a + (h_b - h_a)u \]
**Polygon Clipping in 3D**

- Similar to 2D clipping
  - Bounding box
  - Clipping with each clipping plane
  - Etc.....

![Diagram of polygon clipping in 3D]
Bounding Boxes

- Rather than doing clipping on a complex polygon, we can use an axis-aligned bounding box or extent
  - Smallest rectangle aligned with axes that encloses the polygon
  - Simple to compute: max and min of x and y

![Diagram showing bounding boxes and clipping decisions](image)