Computer Graphics (CS 543)
Lecture 12a: 3D Clipping

Prof Emmanuel Agu

Computer Science Dept.
Worcester Polytechnic Institute (WPI)
Consider an edge going from A to C

- 2 end-points of edge: $A = (Ax, Ay, Az, Aw)$ and $C = (Cx, Cy, Cz, Cw)$
Liang-Barsky 3D Clipping

- **Goal:** Clip object edge-by-edge against Canonical View volume (CVV)

- **Problem:**
  - 2 end-points of edge: $A = (Ax, Ay, Az, Aw)$ and $C = (Cx, Cy, Cz, Cw)$
  - If edge intersects with CVV, compute intersection point $I = (Ix, Iy, Iz, Iw)$
Problem: Determine if point \((x,y,z)\) is inside or outside CVV?

CVV == 6 infinite planes \((x=-1,1;\ y=-1,1;\ z=-1,1)\)

Point \((x,y,z)\) is **inside CVV** if
\[
(-1 \leq x \leq 1) \\
\text{and} \ (-1 \leq y \leq 1) \\
\text{and} \ (-1 \leq z \leq 1)
\]

else point **is outside CVV**
Determining if point is inside CVV

- If point specified as (x,y,z,w)
- Test \((x/w, \ y/w, \ z/w)\)!

Point \((x/w, \ y/w, \ z/w)\) is inside CVV

if \((-1 <= \ x/w \ <= 1)\)
and \((-1 <= \ y/w \ <= 1)\)
and \((-1 <= \ z/w \ <= 1)\)

else point is outside CVV
Modify Inside/Outside Tests Slightly

Our test: \((-1 < \frac{x}{w} < 1)\)

Point \((x,y,z,w)\) inside plane \(x = 1\) if

\[ x/w < 1 \]

\[ \Rightarrow w - x > 0 \]

Point \((x,y,z,w)\) inside plane \(x = -1\) if

\[-1 < x/w \]

\[ \Rightarrow w + x > 0 \]
Numerical Example: Inside/Outside CVV Test

- Point \((x,y,z,w)\) is
  - inside plane \(x=-1\) if \(w+x > 0\)
  - inside plane \(x=1\) if \(w-x > 0\)

- Example Point \((0.5, 0.2, 0.7)\) inside planes \((x = -1,1)\) because \(-1 \leq 0.5 \leq 1\)

- If \(w = 10\), \((0.5, 0.2, 0.7) = (5, 2, 7, 10)\)
- Can either divide by \(w\) then test: \(-1 \leq 5/10 \leq 1\) OR
  - To test if inside \(x = -1\), \(w + x = 10 + 5 = 15 > 0\)
  - To test if inside \(x = 1\), \(w - x = 10 - 5 = 5 > 0\)
3D Clipping

- Do same for y, z to form boundary coordinates for 6 planes as:

<table>
<thead>
<tr>
<th>Boundary coordinate (BC)</th>
<th>Homogenous coordinate</th>
<th>Clip plane</th>
<th>Example (5,2,7,10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BC0</td>
<td>w+x</td>
<td>x=-1</td>
<td>15</td>
</tr>
<tr>
<td>BC1</td>
<td>w-x</td>
<td>x=1</td>
<td>5</td>
</tr>
<tr>
<td>BC2</td>
<td>w+y</td>
<td>y=-1</td>
<td>12</td>
</tr>
<tr>
<td>BC3</td>
<td>w-y</td>
<td>y=1</td>
<td>8</td>
</tr>
<tr>
<td>BC4</td>
<td>w+z</td>
<td>z=-1</td>
<td>17</td>
</tr>
<tr>
<td>BC5</td>
<td>w-z</td>
<td>z=1</td>
<td>3</td>
</tr>
</tbody>
</table>

- Consider line that goes from point A to C
  - **Trivial accept:** 12 BCs (6 for pt. A, 6 for pt. C) > 0
  - **Trivial reject:** Both endpoints A, C outside (-ve) for same plane
Edges as Parametric Equations

- Implicit form \( F(x, y) = 0 \)

- Parametric forms:
  - points specified based on single parameter value
  - Typical parameter: time \( t \)

\[
P(t) = P_0 + (P_1 - P_0)*t \quad 0 \leq t \leq 1
\]

- Some algorithms work in parametric form
  - Clipping: exclude line segment ranges
  - Animation: Interpolate between endpoints by varying \( t \)

- Represent each edge parametrically as \( A + (C - A)t \)
  - at time \( t=0 \), point at \( A \)
  - at time \( t=1 \), point at \( C \)
Inside/outside?

- Test A, C against 6 walls \((x=-1,1; \ y=-1,1; \ z=-1,1)\)
- There is an intersection if BCs have opposite signs. i.e. if either
  - A is outside \(<0\), C is inside \(>0\) or
  - A inside \(>0\), C outside \(<0\)
- Edge intersects with plane at some \(t_{hit}\) between \([0,1]\)
Calculating hit time (t_hit)

- How to calculate t_hit?
- Represent an edge t as:

\[ \text{Edge}(t) = ((Ax + (Cx - Ax)t, (Ay + (Cy - Ay)t, (Az + (Cz - Az)t, (Aw + (Cw - Aw)t) \]

- E.g. If x = 1,
  \[
  \frac{x}{w} = \frac{Ax + (Cx - Ax)t}{Aw + (Cw - Aw)t} = 1
  \]

- Solving for t above,

\[
t = \frac{Aw - Ax}{(Aw - Ax) - (Cw - Cx)}
\]
Inside/outside?

- t_hit can be “entering (t_in)” or “leaving (t_out)”
- Define: “entering” if A outside, C inside
  - Why? As t goes [0-1], edge goes from outside (at A) to inside (at C)
- Define “leaving” if A inside, C outside
  - Why? As t goes [0-1], edge goes from inside (at A) to inside (at C)
Definition: Candidate Interval

- Candidate Interval (CI): time interval during which edge might still be inside CVV. i.e. CI = t_in to t_out
- Initialize CI to [0,1]
- For each of 6 planes, calculate t_in or t_out, shrink CI

Conversely: values of t outside CI = edge is outside CVV
Example: Chop step by Step against 6 planes

- Initially

  \[
  C \\
  t = 1
  \]

  \[
  t_{in} = 0, \quad t_{out} = 1 \\
  \text{Candidate Interval (CI) = [0 to 1]}
  \]

- Chop against each of 6 planes

  \[
  A \\
  t = 0
  \]

  t_{in} = 0, \quad t_{out} = 0.74

  \text{Candidate Interval (CI) = [0 to 0.74]}

  Plane \( y = 1 \)

  Why \( t_{out} \)?
Chop step by step against 6 planes

- Initially
  
  $t_{\text{out}} = 0.74$
  
  $t_{\text{in}} = 0, \quad t_{\text{out}} = 0.74$
  
  Candidate Interval (CI) $= [0 \text{ to } 0.74]$

- Then
  
  Plane $x = -1$

  $t_{\text{in}} = 0.36, \quad t_{\text{out}} = 0.74$
  
  Candidate Interval (CI) $= [0.36 \text{ to } 0.74]$

Why $t_{\text{in}}$?
Shortening Candidate Interval

**Algorithm:**
- Test for trivial accept/reject (stop if either occurs)
- Set CI to [0,1]
- For each of 6 planes:
  - Find hit time $t_{hit}$
  - If $t_{in}$, new $t_{in} = \max(t_{in}, t_{hit})$
  - If $t_{out}$, new $t_{out} = \min(t_{out}, t_{hit})$
  - If $t_{in} > t_{out} \Rightarrow$ exit (no valid intersections)

**Note:** seeking smallest valid CI without $t_{in}$ crossing $t_{out}$
Calculate chopped A and C

- If valid $t_{in}$, $t_{out}$, calculate adjusted edge endpoints $A$, $C$ as

- $A_{chop} = A + t_{in} \cdot (C - A)$ (calculate for $Ax,Ay,Az$)
- $C_{chop} = A + t_{out} \cdot (C - A)$ (calculate for $Cx,Cy,Cz$)
3D Clipping Implementation

- Function `clipEdge()`
- Input: two points A and C (in homogenous coordinates)
- Output:
  - 0, if AC lies complete outside CVV
  - 1, complete inside CVV
  - Returns clipped A and C otherwise
Store BCs as Outcodes

- Calculate 6 BCs for A, 6 for C
- Use outcodes to track in/out
  - Number walls $x = +1, -1; y = +1, -1$, and $z = +1, -1$ as 0 to 5
  - Bit $i$ of A’s outcode = 1 if A is outside $i$th wall
  - 1 otherwise
- **Example:** outcode if point outside walls 1, 2, 5

<table>
<thead>
<tr>
<th>Wall no.</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>OutCode</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Trivial Accept/Reject using Outcodes

- **Trivial accept**: inside (not outside) all walls

<table>
<thead>
<tr>
<th>Wall no.</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>A Outcode</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C OutCode</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

  Logical bitwise test: \( A \mid C == 0 \)

- **Trivial reject**: point outside **same** wall. Example Both A and C outside wall 1

<table>
<thead>
<tr>
<th>Wall no.</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>A Outcode</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>C OutCode</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

  Logical bitwise test: \( A \& C != 0 \)
3D Clipping Implementation

- Compute BCs for A,C store as outcodes
- Test A, C outcodes for trivial accept, trivial reject
- If not trivial accept/reject, for each wall:
  - Compute tHit
  - Update t_in, t_out
  - If t_in > t_out, early exit
int clipEdge(Point4& A, Point4& C) {
    double tIn = 0.0, tOut = 1.0, tHit;
    double aBC[6], cBC[6];
    int aOutcode = 0, cOutcode = 0;

    .....find BCs for A and C
    .....form outcodes for A and C

    if((aOutCode & cOutcode) != 0) // trivial reject
        return 0;
    if((aOutCode | cOutcode) == 0) // trivial accept
        return 1;
3D Clipping Pseudocode

for(i=0;i<6;i++)  // clip against each plane
{
    if(cBC[i] < 0)  // C is outside wall i (exit so tOut)
    {
        tHit = aBC[i]/(aBC[i] – cBC[i]);  // calculate tHit
        tOut = MIN(tOut, tHit);
    }
    else if(aBC[i] < 0)  // A is outside wall i (enters so tIn)
    {
        tHit = aBC[i]/(aBC[i] – cBC[i]);  // calculate tHit
        tIn = MAX(tIn, tHit);
    }
    if(tIn > tOut) return 0; // Cl is empty: early out
}
Point4 tmp; // stores homogeneous coordinates
If(aOutcode != 0) // A is outside: tIn has changed. Calculate A_chop
{
    tmp.x = A.x + tIn * (C.x – A.x);
    // do same for y, z, and w components
}
If(cOutcode != 0) // C is outside: tOut has changed. Calculate C_chop
{
    C.x = A.x + tOut * (C.x – A.x);
    // do same for y, z and w components
}
A = tmp;
Return 1; // some of the edges lie inside CVV
Polygon Clipping

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

- Clipping a **convex** polygon can yield at most one other polygon
Clipping Polygons

- Need more sophisticated algorithms to handle polygons:
  - **Sutherland-Hodgman**: clip any given polygon against a convex clip polygon (or window)
  - **Weiler-Atherton**: Both clipped polygon and clip polygon (or window) can be concave
Tessellation and Convexity

- One strategy is to replace nonconvex (concave) polygons with a set of triangular polygons (a tessellation)
- Also makes fill easier
References