3D Clipping

- 3D clipping against canonical view volume (CVV)
- Automatically clipping after projection matrix
- Liang-Barsky algorithm (embellished by Blinn)
- CVV == 6 infinite planes (x=-1,1; y=-1,1; z=-1,1)
- Clip edge-by-edge of the an object against CVV
- Chopping may change number of sides of an object. E.g. chopping tip of triangle may create quadrilateral
3D Clipping

- Represent edge parametrically as $A + (C - A)t$
- Interpretation: a point is traveling such that:
  - at time $t=0$, point at $A$
  - at time $t=1$, point at $C$
- Like Cohen-Sutherland, first determine trivial accept/reject
- E.g. to test edge against plane, point is:
  - Inside (right of plane $x=-1$) if $Ax/Aw > -1$ or $(Aw + Ax) > 0$
  - Inside (left of plane $x=1$) if $Ax/Aw < 1$ or $(Aw - Ax) > 0$

\[
\begin{array}{c|c|c}
\text{Clip plane} & \text{boundary coordinate (BC)} & \text{Homogenous coordinate} \\
\hline
\text{C0} & x + x & -1 \\
\text{C1} & y + y & -1 \\
\text{C2} & z + z & -1 \\
\text{C3} & y + y & -1 \\
\text{C4} & z + z & -1 \\
\text{C5} & x + x & -1 \\
\end{array}
\]

- Trivial accept: 12 BCs (6 for pt. $A$, 6 for pt. $C$) are positive
- Trivial reject: Both endpoints outside of same plane

3D Clipping

- If not trivial accept/reject, then clip
- Define Candidate Interval (CI) as time interval during which edge might still be inside CVV. i.e. $CI = t_{in}$ to $t_{out}$

\[
\begin{array}{c}
0 \quad CI \quad 1 \\
\hline
\end{array}
\]

- Conversely: values of $t$ outside $CI$ = edge is outside CVV
- Initialize CI to $[0,1]$

3D Clipping

- How to calculate $t_{hit}$?
- Represent an edge $t$ as:

\[
Edget(x) = ((Ax + (Cx - Ax)y), (Ay + (Cy) - Ay)y), (Az + (Cz - Az)x), (Aw + (Cw - Aw)x))
\]
- E.g. If $x = 1$, $\frac{Ax + (Cx - Ax)y}{Aw + (Cw - Aw)x} = t$
- Solving for $t$ above,

\[
t = \frac{Aw - Ax}{(Aw - Ax)(Cw - Cx)}
\]
3D Clipping

- Test against each wall in turn
- If BCs have opposite signs = edge hits plane at time $t_{hit}$
- Define: "entering" = as $t$ increases, outside to inside
- i.e. if pt. $A$ is outside, $C$ is inside
- Likewise, "leaving" = as $t$ increases, inside to outside ($A$ inside, $C$ outside)

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, as $t$ increases, edge entering, $t_{in} = \max(t_{in}, t_{hit})$
  - If, as $t$ increases, edge leaving, $t_{out} = \min(t_{out}, t_{hit})$
- If $t_{in} > t_{out} =>$ exit (no valid intersections)

Note: seeking smallest valid CI without $t_{in}$ crossing $t_{out}$

3D Clipping

Example to illustrate search for $t_{in}$, $t_{out}$

Note: CVV is different shape. This is just example

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

- $A_{chop} = A + t_{in} (C - A)$
- $C_{chop} = C + t_{out} (C - A)$
### 3D Clipping Implementation

- **Function clipEdge( )**
- **Input:** two points A and C (in homogeneous coordinates)
- **Output:**
  - 0, if no part of line AC lies in CVV
  - 1, otherwise
  - Also returns clipped A and C
- Store 6 BCs for A, 6 for C

- Use outcodes to track in/out
  - Number walls 1...6
  - Bit i of A’s outcode = 0 if A is inside ith wall
  - 1 otherwise
  - Trivial accept: both A and C outcodes = 0
  - Trivial reject: bitwise AND of A and C outcodes is non-zero
- If not trivial accept/reject:
  - Compute THR
  - Update t_in, t_out
  - If t_in > t_out, early exit

### 3D Clipping Pseudocode

```c
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;
    for(i=0;i<6;i++)  // clip against each plane
    {
        if(cBC[i] < 0)  // exits: C is outside
            tHit = aBC[i]/(aBC[i] – cBC[i]);
        else if(aBC[i] < 0)  // enters: A is outside
            tHit = aBC[i]/(aBC[i] – cBC[i]);
        tIn = MAX(tIn, tHit);
        tOut = MIN(tOut, tHit);
        if(tIn > tOut) return 0; // CI is empty: early out
    }
    // return clipped A and C
}
```
3D Clipping Pseudocode

Point4 tmp; // stores homogeneous coordinates
If (aOutcode != 0) // A is out: tIn has changed
  tmp.x = A.x + tIn * (C.x – A.x);
  // do same for y, z, and w components
}
If (cOutcode != 0) // C is out: tOut has changed
  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

Viewport Transformation

- After clipping, do viewport transformation
- We have used glViewport(x, y, wid, ht) before
- Use again here!
- glViewport shifts x, y to screen coordinates
- Also maps pseudo-depth z from range [-1,1] to [0,1]
- Pseudo-depth stored in depth buffer, used for Depth testing (Will discuss later)

References

- Hill, sections 7.4.4, 4.8.2