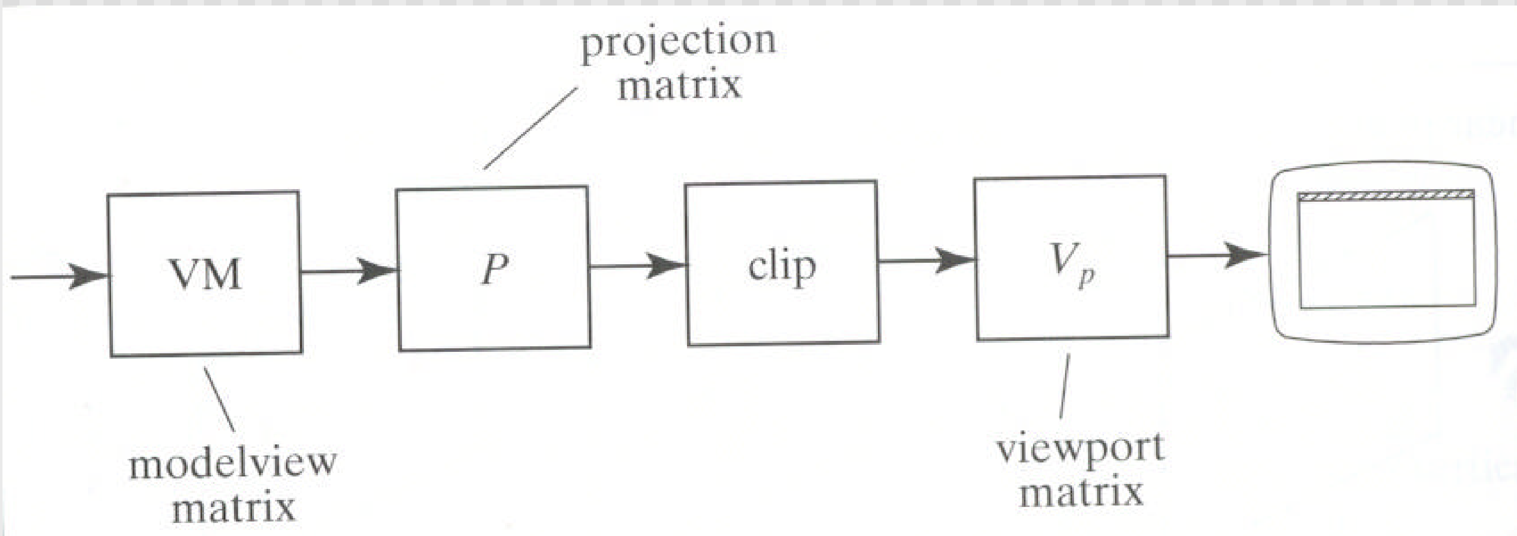
CS 543: Computer Graphics
Lecture 7 (Part II) : 3D Clipping and Viewport Transformation

Emmanuel Agu

3D Clipping



- Clipping occurs after projection transformation
- Clipping is against canonical view volume

Parametric Equations

- Implicit form

$$F(x, y) = 0$$

- Parametric forms:
  - points specified based on single parameter value
  - Typical parameter: time *t*

$$P(t) = P_0 + (P_1 - P_0) * t \qquad 0 \le t \le 1$$

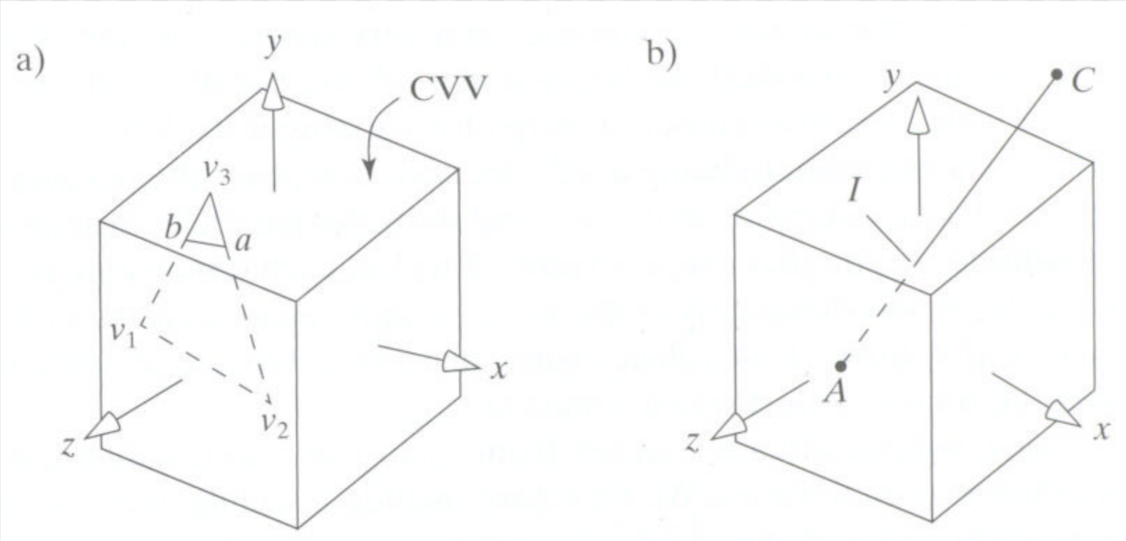
- Some algorithms work in parametric form
  - Clipping: exclude line segment ranges
  - Animation: Interpolate between endpoints by varying t

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

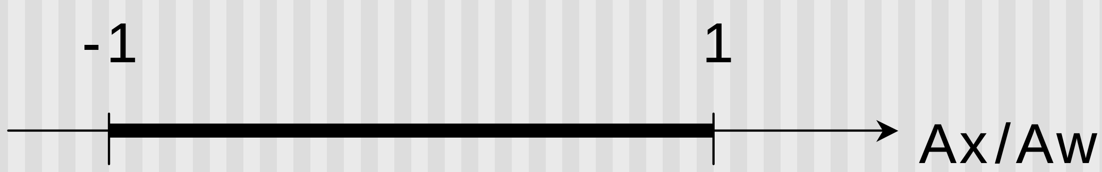
3D Clipping

- Problem:
  - Two points, A = (Ax, Ay, Az, Aw) and C = (Cx, Cy, Cz, Cw), in homogeneous coordinates
  - If segment intersects with CVV, need to compute intersection point I-=(Ix,Iy,Iz,Iw)



3D Clipping

- Represent edge parametrically as A + (C – A)t
- Intepretation: 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



3D Clipping

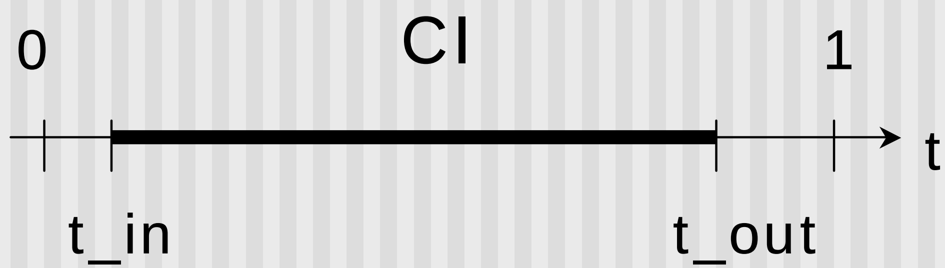
- Using notation (Aw +Ax) = w + x, write boundary coordinates for 6 planes as:

Boundary coordinate (BC)	**Homogenous coordinate**	**Clip plane**
BC0	w+x	x=-1
BC1	w-x	x=1
BC2	w+y	y=-1
BC3	w-y	y=1
BC4	w+z	z=-1
BC5	w-z	z=1

- **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



- 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:

$$Edge(t) = ((Ax + (Cx - Ax)t, (Ay + (Cy - Ay)t, (Az + (Cz - Az)t, (Aw + (Cw - Aw)t)$$

- E.g. If x = 1, $\dfrac{Ax + (Cx - Ax)t}{Aw + (Cw - Aw)t} = 1$
- 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)

3D Clipping

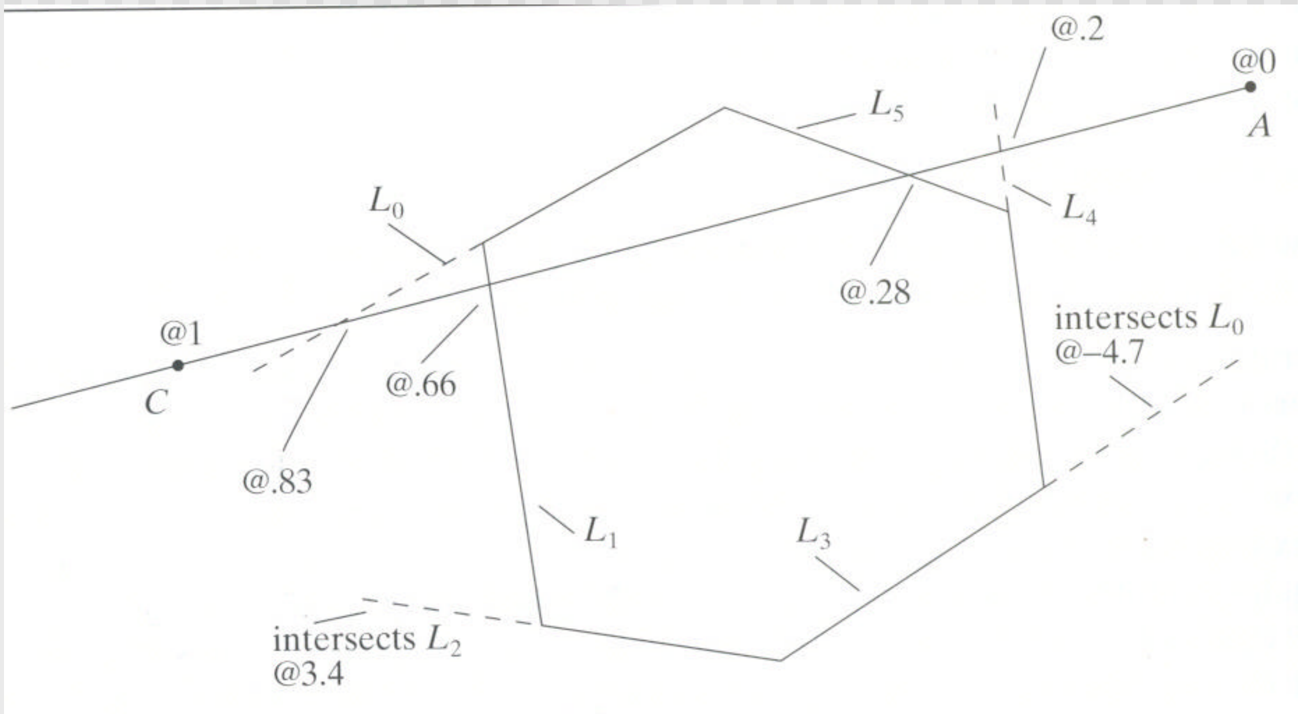
- **Algorithm:**
  - Test for trivial accept/reject (stop if either occurs)
  - Set CI to [0,1]
  - For each of 6 planes:
    - Find hit time t_hit
    - If, as t increases, edge entering, t_in = max(t_in,t_hit)
    - If, as t increases, edge leaving, t_out = min(t_out, t_hit)
    - If t_in > t_out => 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



Line test	t_{in}	t_{out}
0	0	0.83
1	0	0.66
2	0	0.66
3	0	0.66
4	0.2	0.66
5	0.28	0.66

3D Clipping

- If valid t_in, t_out, calculate adjusted edge endpoints A, C as
- A_chop = A + t_in (C – A)
- C_chop = A + t_out (C – A)

3D Clipping Implementation

- Function clipEdge( )
- Input: two points A and C (in homogenous 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

3D Clipping Implementation

- 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 tHit
  - Update t_in, t_out
  - If t_in > t_out, early exit

3D Clipping Pseudocode

```
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

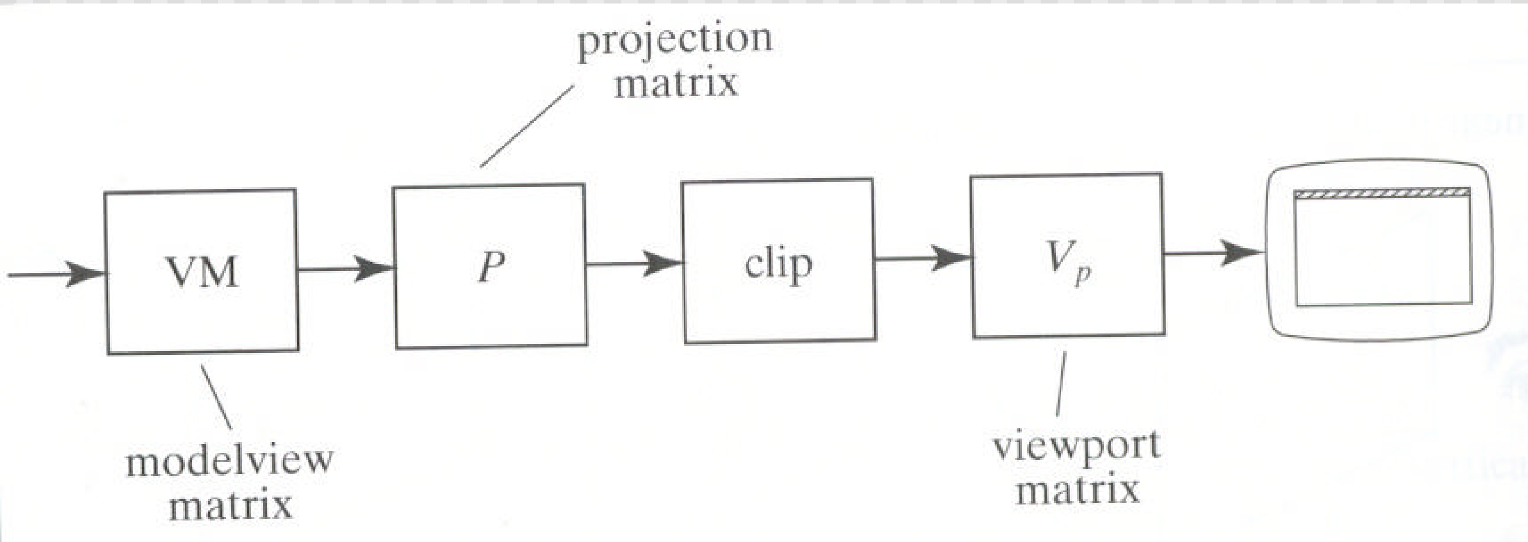
```
for(i=0;i<6;i++)  // clip against each plane
{
    if(cBC[i] < 0)  // exits: C is outside
    {
          tHit = aBC[i]/(aBC[i] – cBC[I]);
          tOut = MIN(tOut, tHit);
    }
    else if(aBC[i] < 0)  // enters: A is outside
    {
          tHit = aBC[i]/(aBC[i] – cBC[i]);
          tIn = MAX(tIn,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

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