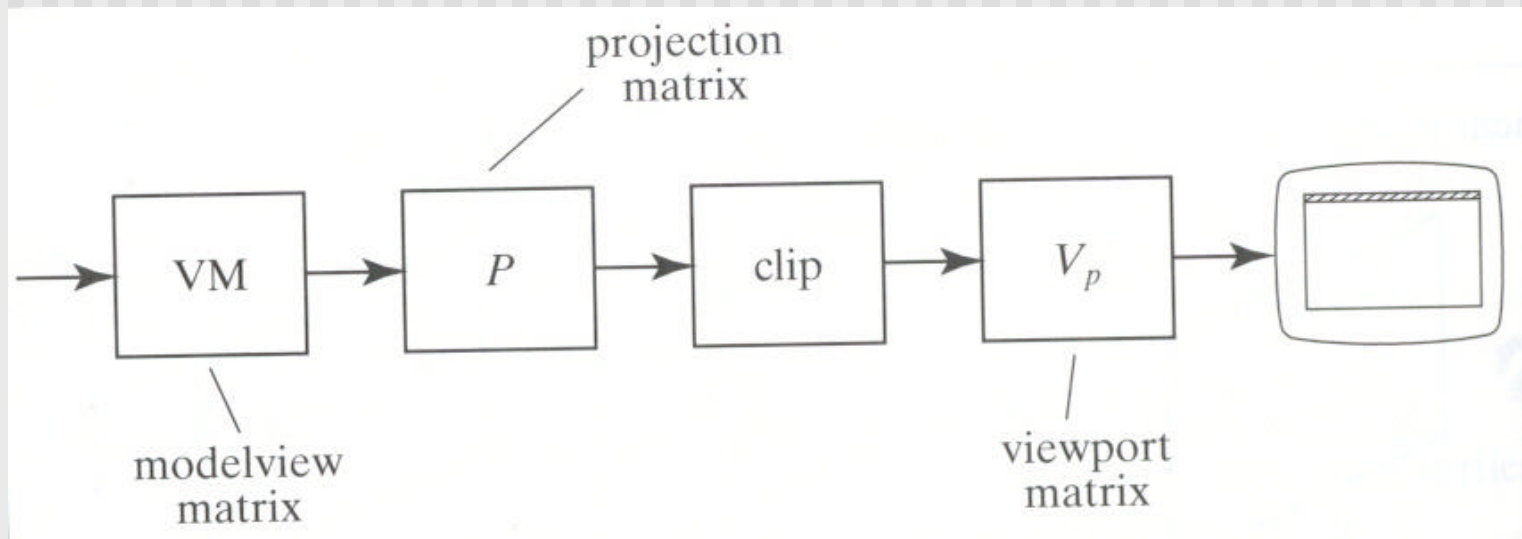


CS 543: Computer Graphics
Lecture 8: 3D Clipping and Viewport Transformation

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

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

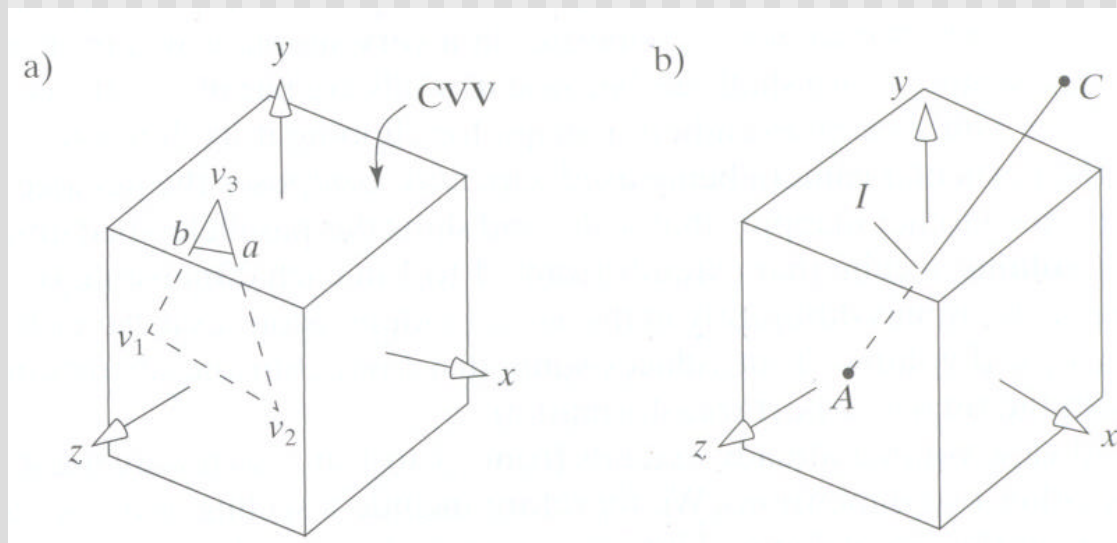
- Some algorithms work in parametric form
 - Clipping: exclude line segment ranges
 - Animation: Interpolate between endpoints by varying t

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, $A = (A_x, A_y, A_z, A_w)$ and $C = (C_x, C_y, C_z, C_w)$, in homogeneous coordinates
 - If segment intersects with CVV, need to compute intersection point $I = (I_x, I_y, I_z, I_w)$



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 $(Aw-Ax) > 0$



3D Clipping

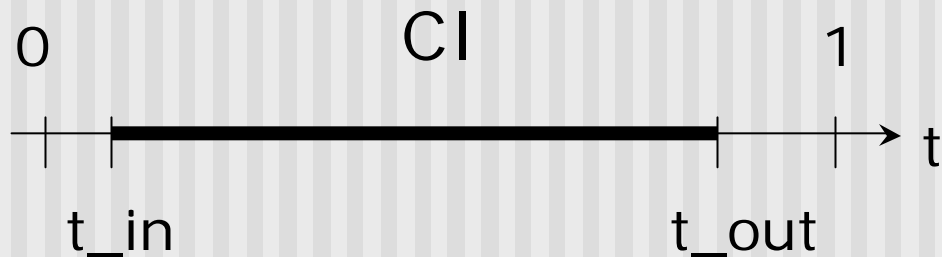
- Using notation $(Aw + Ax) = w + x$, write boundary coordinates for 6 planes as:

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:

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

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

- 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

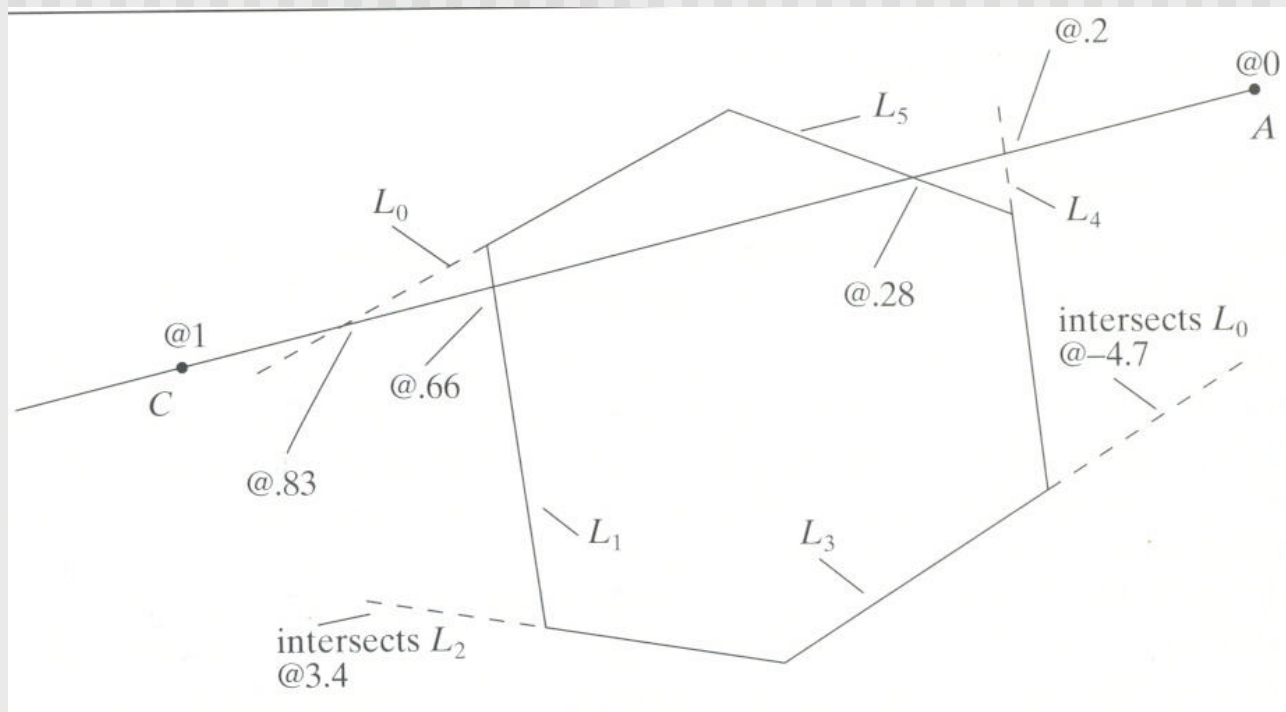
- **Algorithm:**
 - Test for trivial accept/reject (stop if either occurs)
 - Set CI to $[0,1]$
 - For each of 6 planes:
 - Find hit time t_{hit}
 - If, as t increases, edge entering, $t_{in} = \max(t_{in}, t_{hit})$
 - If, as t increases, edge leaving, $t_{out} = \min(t_{out}, t_{hit})$
 - If $t_{in} > t_{out} \Rightarrow$ 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



<u>Line test</u>	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 i th 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

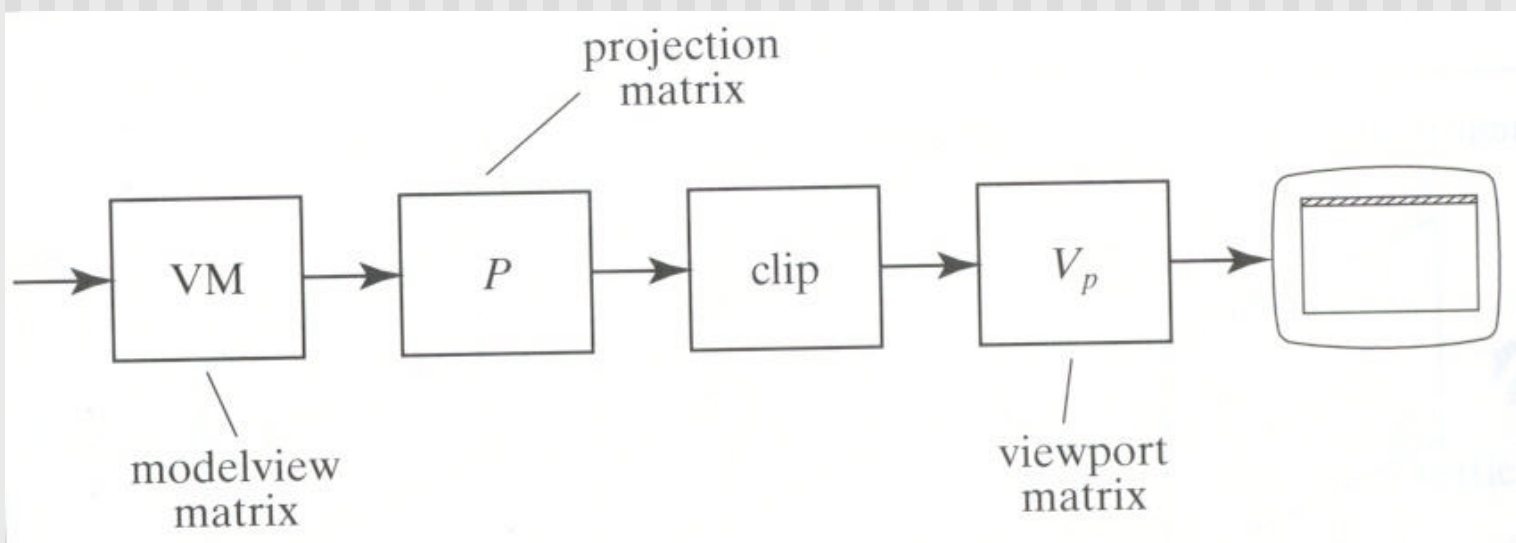
```
for(i=0;i<6;i++) // clip against each plane
{
    if(cBC[i] < 0) // exits: C is outside
    {
        tHit = aBC[i]/(aBC[i] - cBC[i]);
        tOut = MIN(tOut, tHit);
    }
    else if(aBC[i] < 0) // enters: A is outside
    {
        tHit = aBC[i]/(aBC[i] - cBC[i]);
        tIn = MAX(tIn, tHit);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

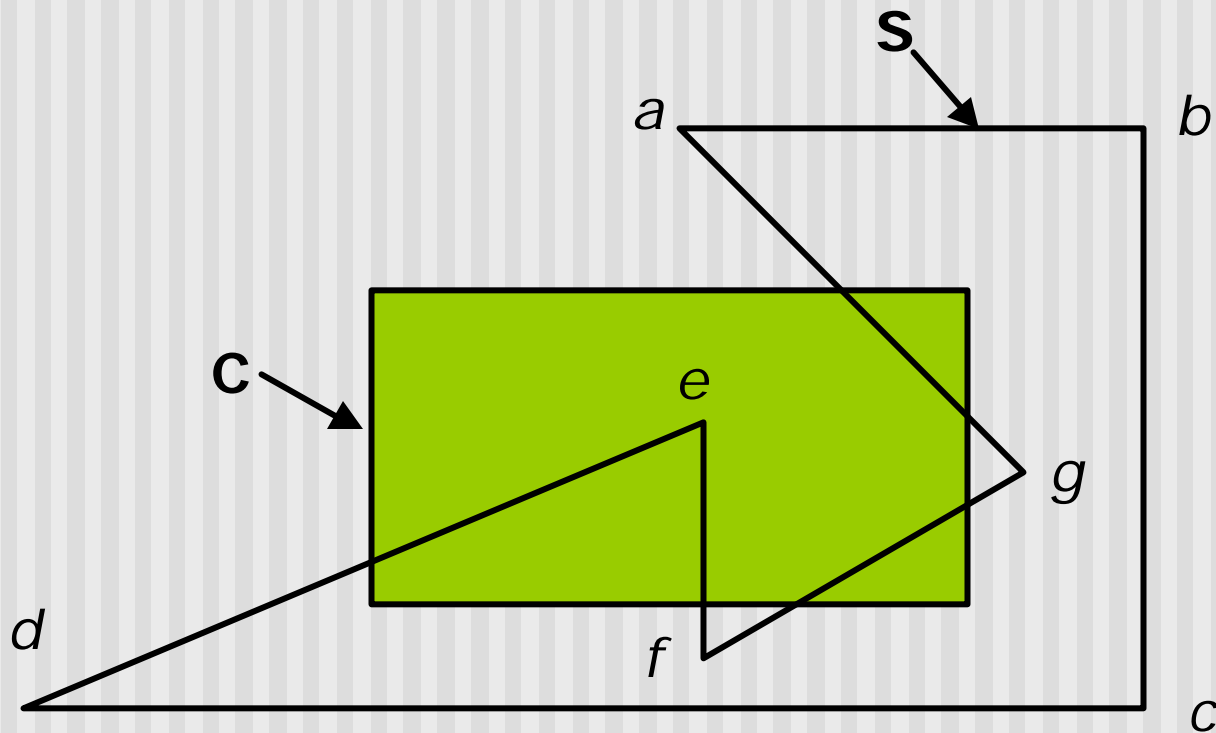


Clipping Polygons

- Cohen-Sutherland and Liang-Barsky clip line segments against each window in turn
- Polygons can be fragmented into several polygons during clipping
- May need to **add** edges
- Need more sophisticated algorithms to handle polygons:
 - *Sutherland-Hodgman*: any subject polygon against a convex clip polygon (or window)
 - *Weiler-Atherton*: Both subject polygon and clip polygon can be concave

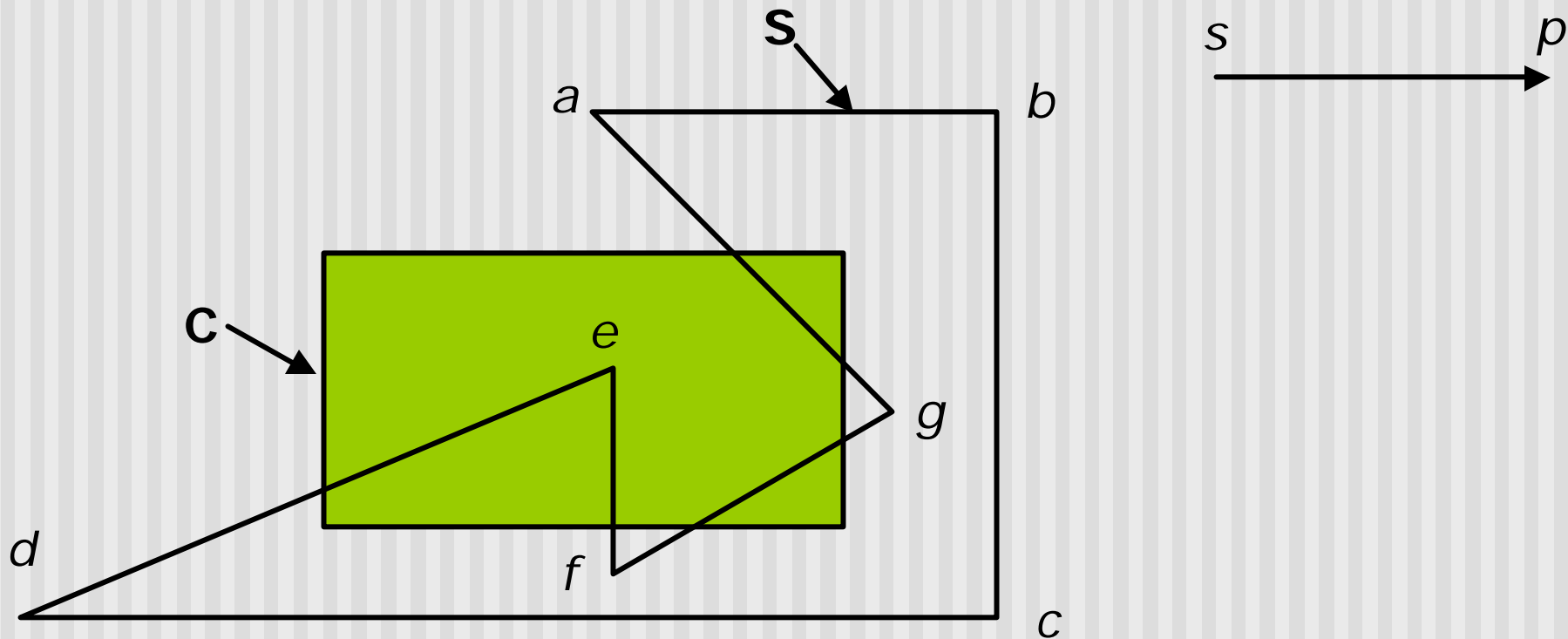
Sutherland-Hodgman Clipping

- Consider **Subject polygon, S** to be clipped against a **clip polygon, C**
- Clip each edge of S against C to get clipped polygon
- S is an ordered list of vertices $a b c d e f g$



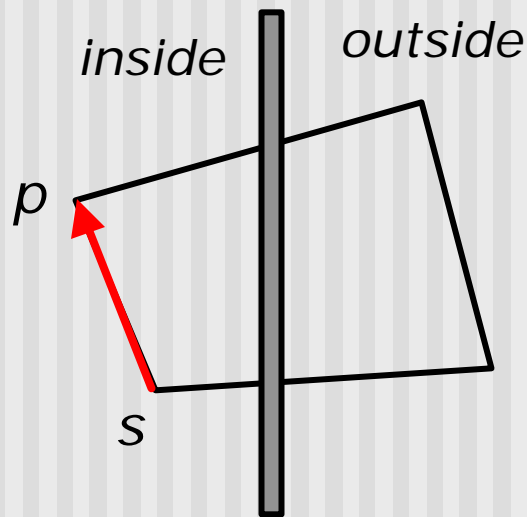
Sutherland-Hodgman Clipping

- Traverse S vertex list edge by edge
- i.e. successive vertex pairs make up edges
- E.g. ab , bc , de , ... etc are edges
- Each edge has first point s and endpoint p

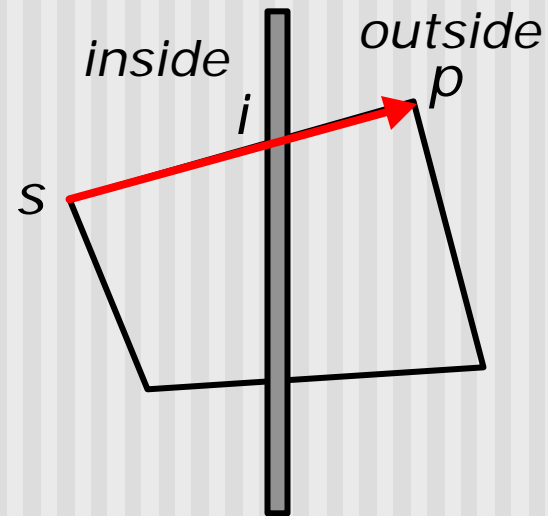


Sutherland-Hodgman Clipping

- For each edge of S , output to **new vertex** depends on whether s or/and p are inside or outside C
- 4 possible cases:



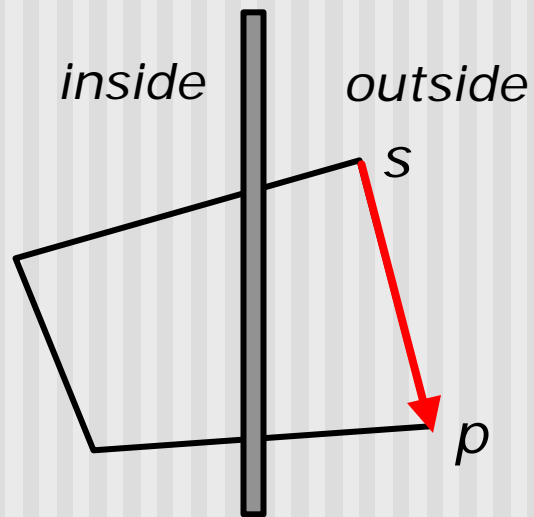
Case A: Both s and p are inside:
output p



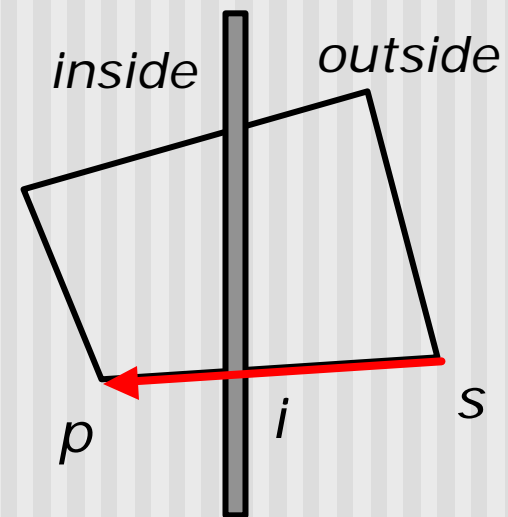
Case B: s inside, p outside:
Find intersection i ,
output i

Sutherland-Hodgman Clipping

■ And....



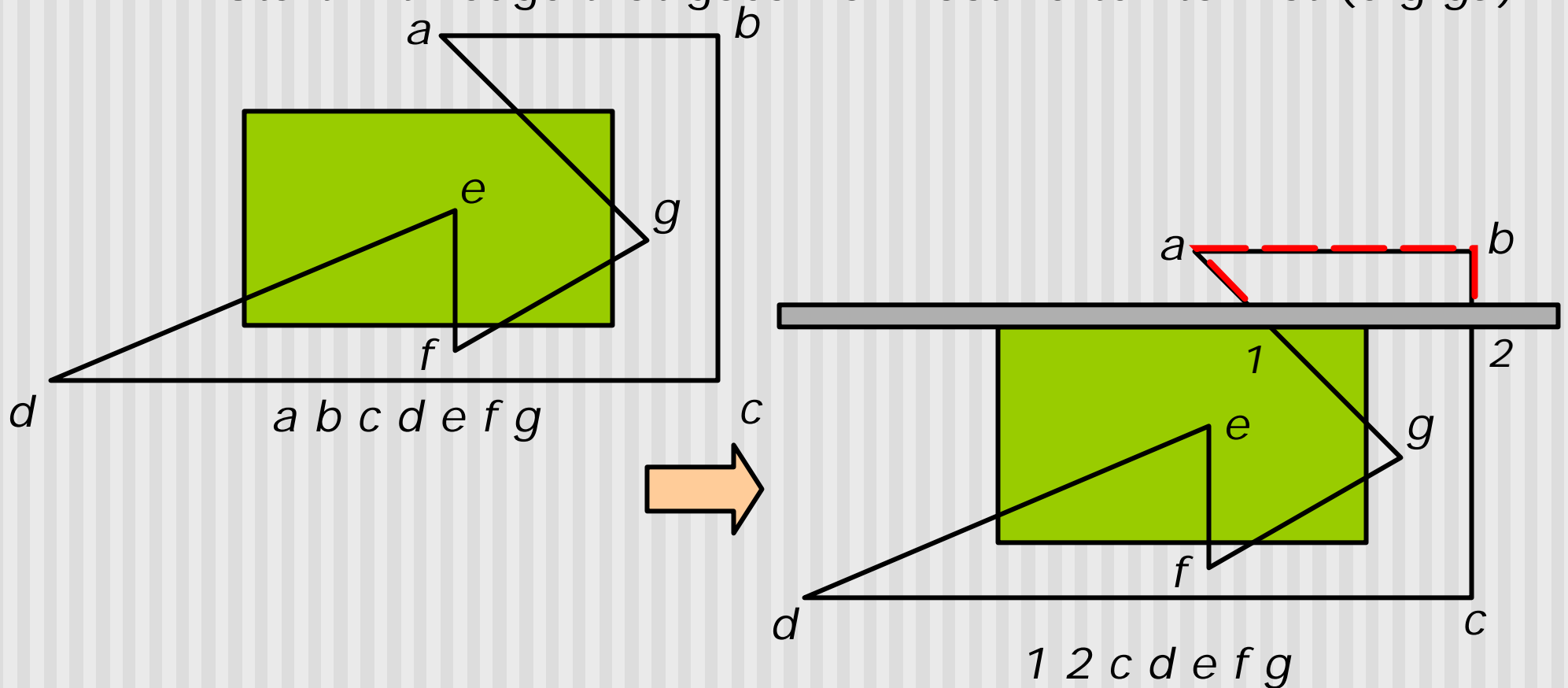
Case C: Both s and p outside:
output nothing



Case D: s outside, p inside:
Find intersection i,
output i and then p

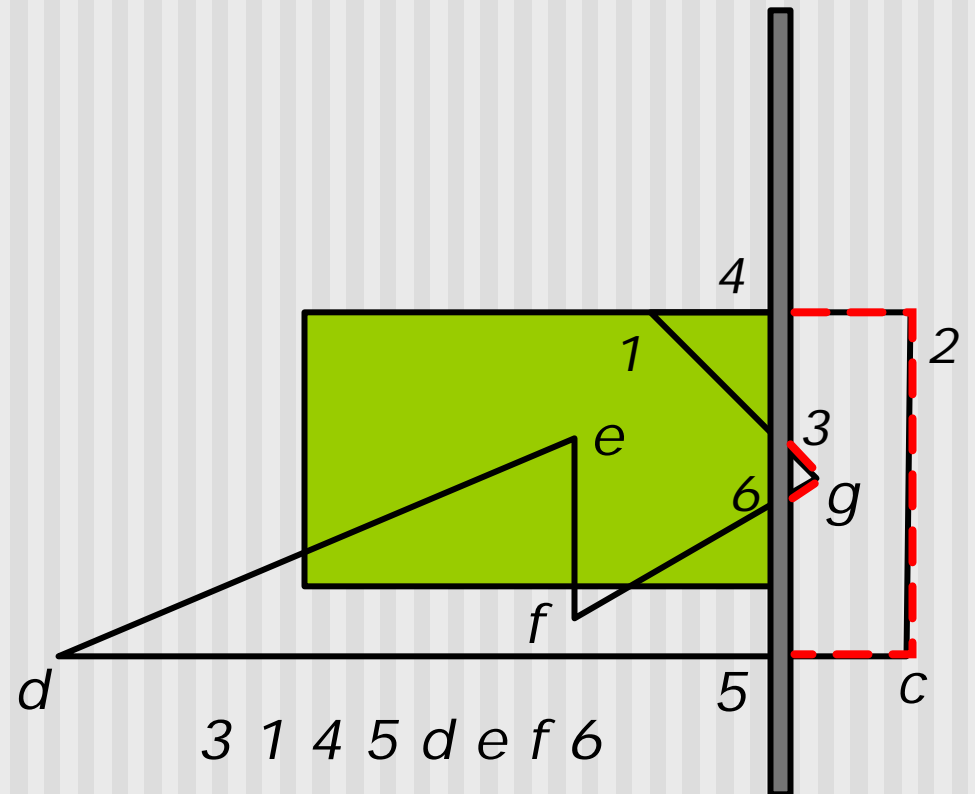
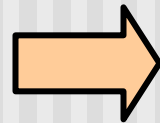
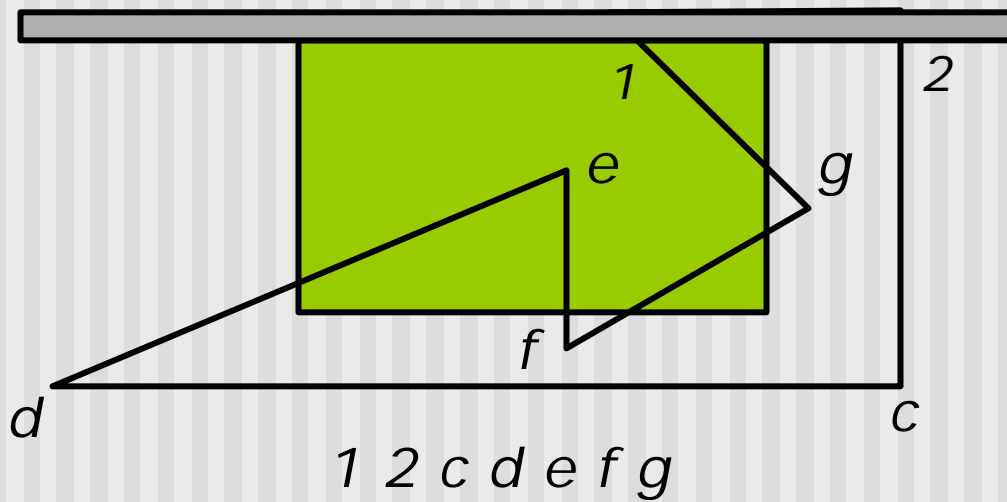
Sutherland-Hodgman Clipping

- Now, let's work through example
- Treat each edge of C as infinite plane to clip against
- Start with edge that goes from last vertex to first (e.g. ga)



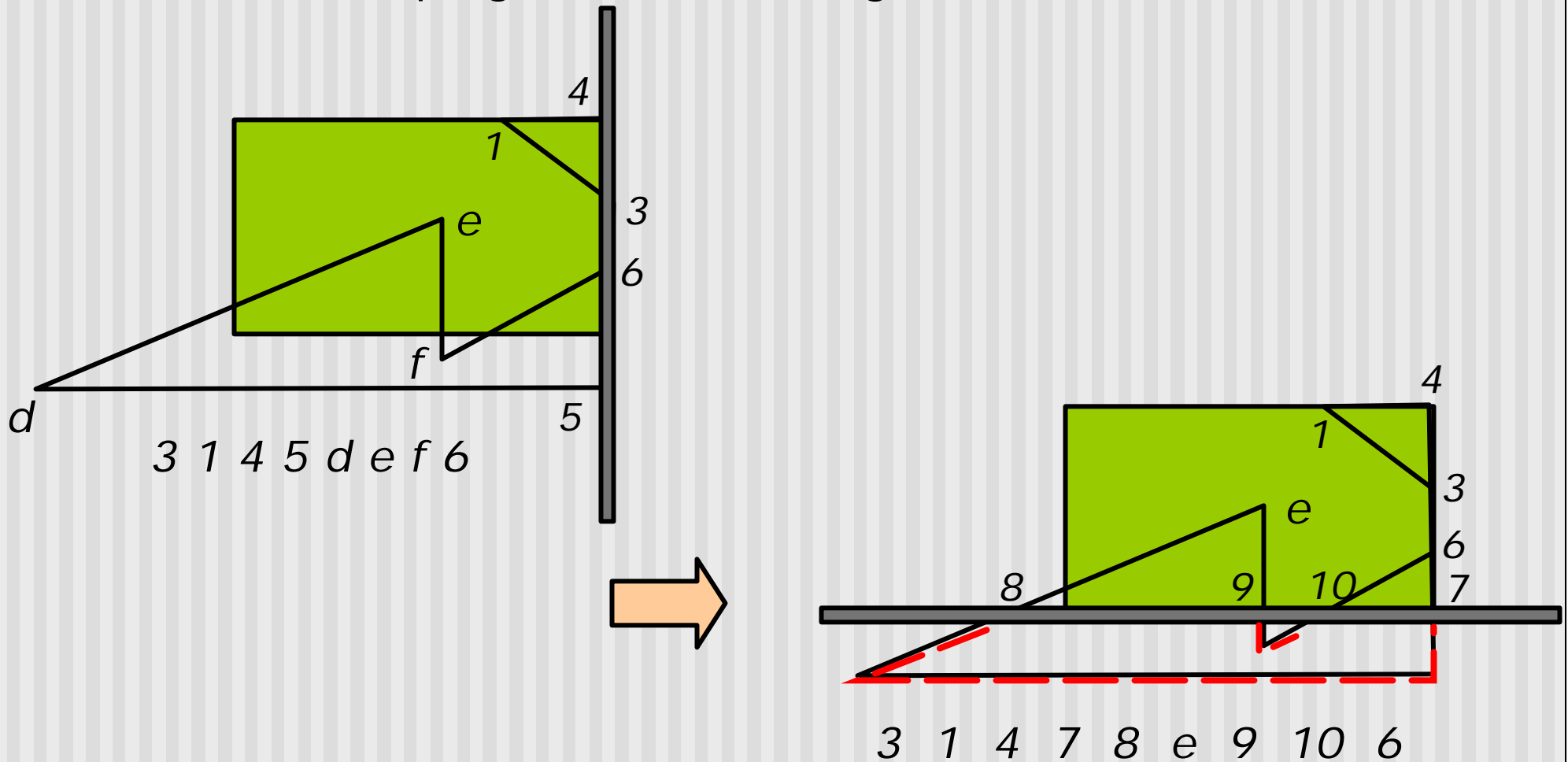
Sutherland-Hodgman Clipping

- Then chop against right edge



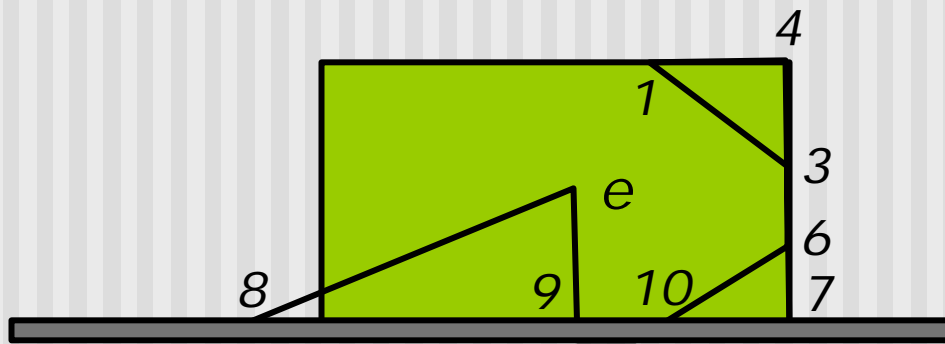
Sutherland-Hodgman Clipping

- Then chop against bottom edge

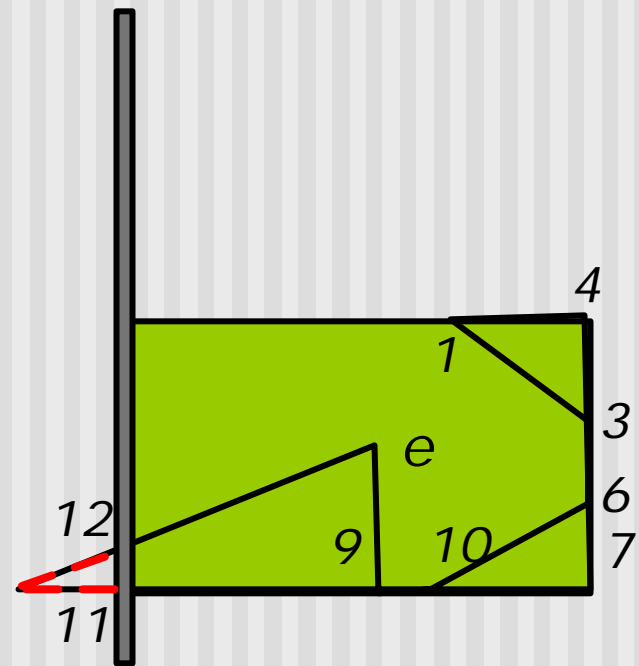
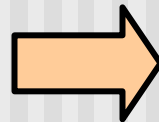


Sutherland-Hodgman Clipping

- Finally, clip against left edge



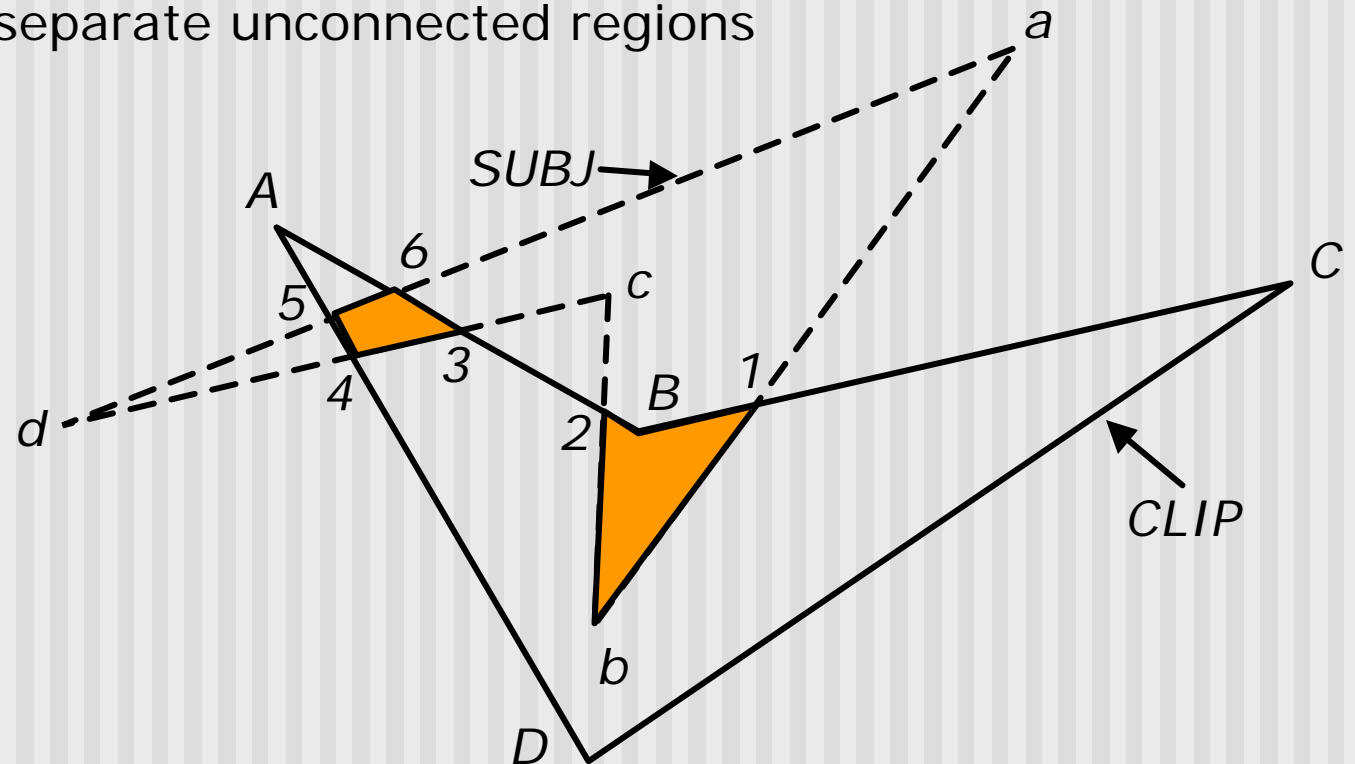
3 1 4 7 8 e 9 10 6



3 1 4 7 11 12 e 9 10 6

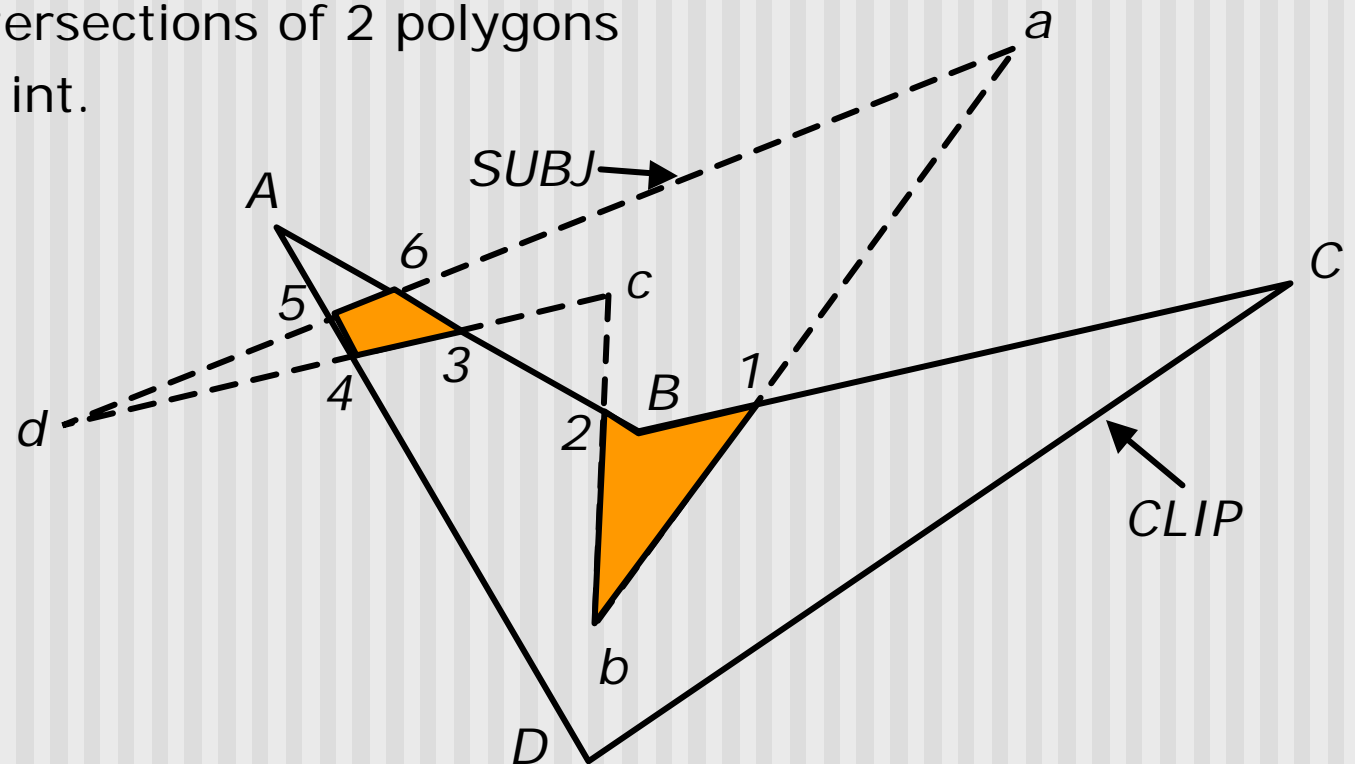
Weiler-Atherton Clipping Algorithm

- Sutherland-Hodgman required at least 1 convex polygon
- Weiler-Atherton can deal with 2 concave polygons
- Searches perimeter of SUBJ polygon searching for borders that enclose a clipped filled region
- Finds multiple separate unconnected regions



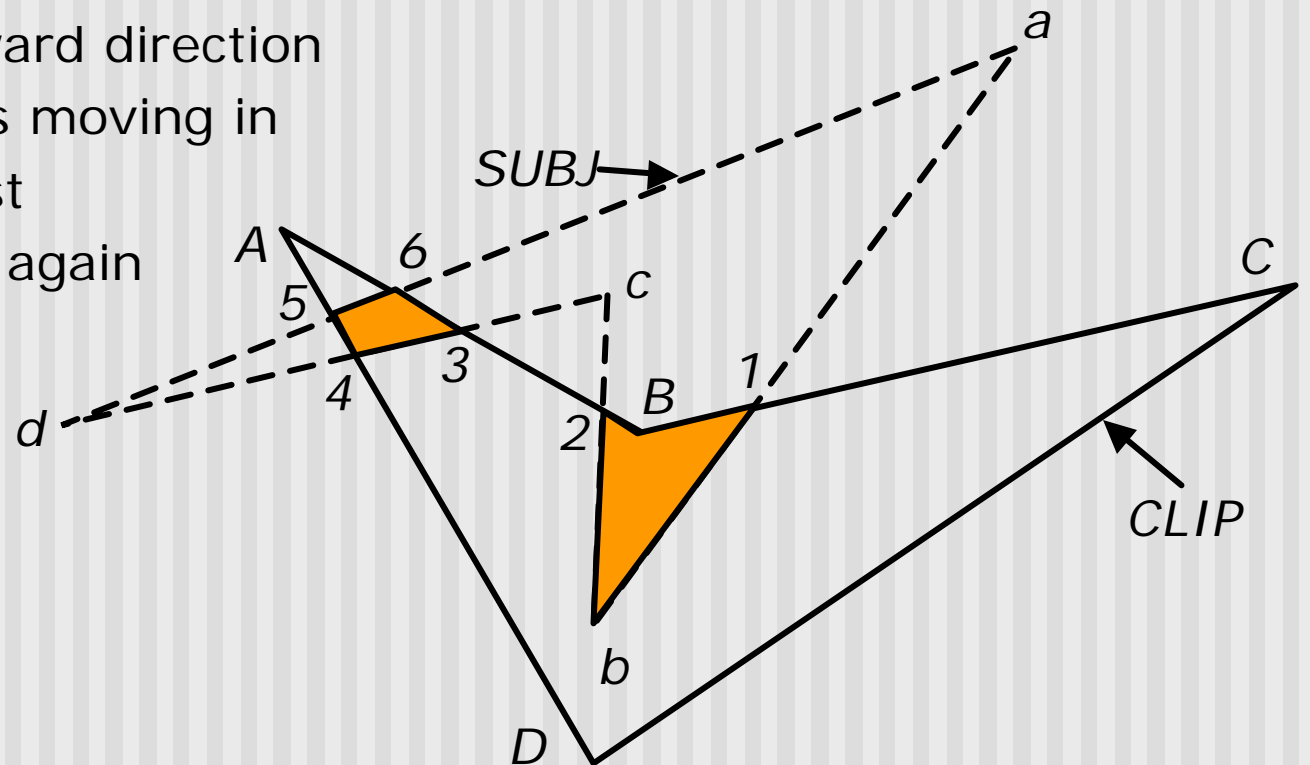
Weiler-Atherton Clipping Algorithm

- Follow detours along CLIP boundary whenever polygon edge crosses to outside of boundary
- Example: SUBJ = {a,b,c,d} CLIP = {A,B,C,D}
- Order: clockwise, interior to right
- First find all intersections of 2 polygons
- Example has 6 int.
- {1,2,3,4,5,6}



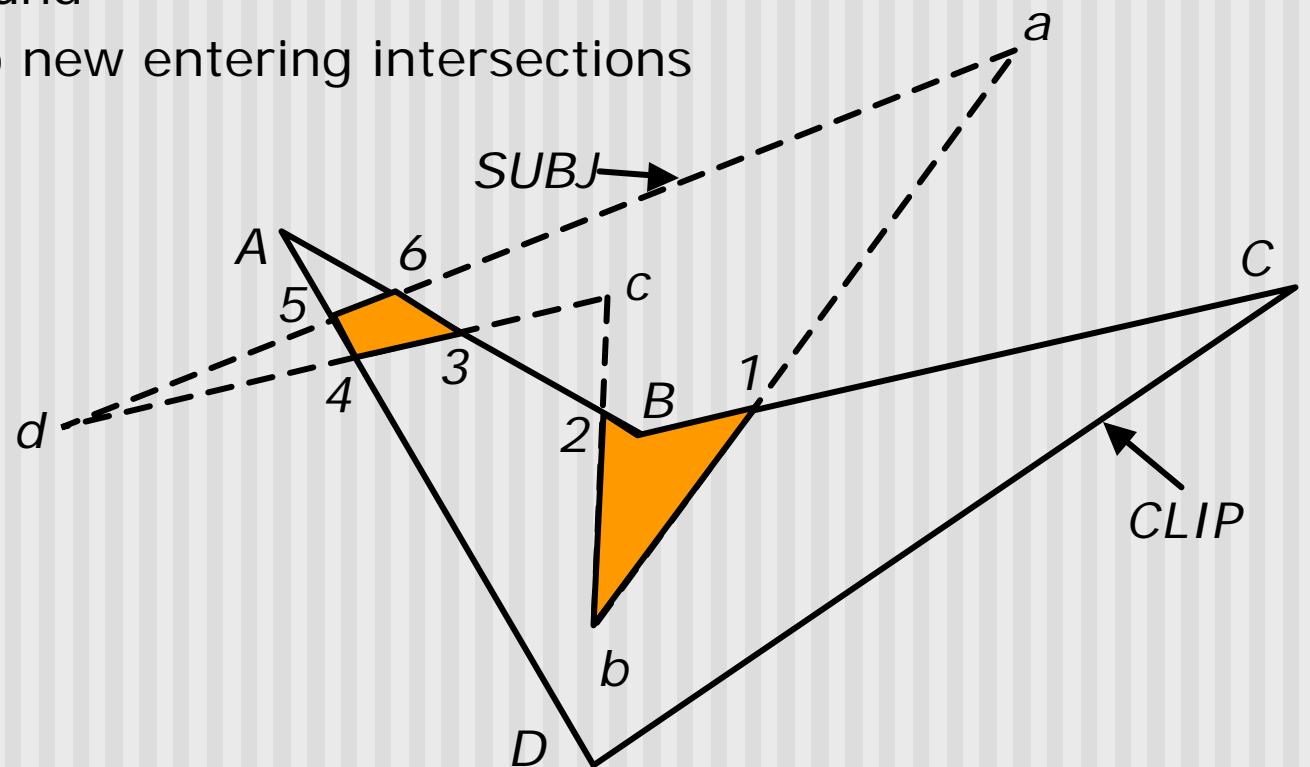
Weiler-Atherton Clipping Algorithm

- Start at a , traverse SUBJ in forward direction till first **entering intersection** (SUBJ moving outside-inside of CLIP) is found
- Record this intersection (1) to new vertex list
- Traverse along SUBJ till next intersection (2)
- Turn away from SUBJ at 2
- Now follow CLIP in forward direction
- Jump between polygons moving in forward direction till first intersection (1) is found again
- Yields: $\{1, b, 2\}$



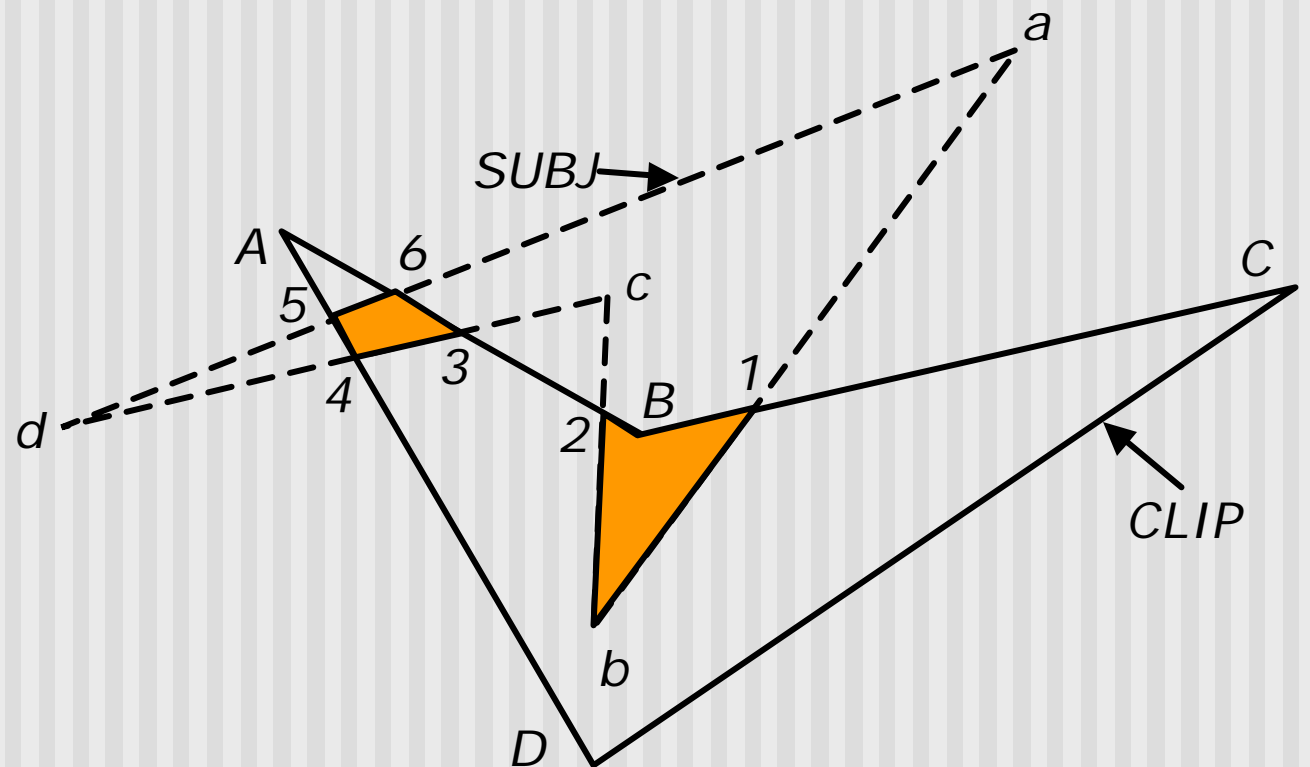
Weiler-Atherton Clipping Algorithm

- Start again, checking for next **entering intersection** of SUBJ
- Intersection (3) is found
- Repeat process
- Jump from SUBJ to CLIP at next intersection (4)
- Polygon {3,4,5,6} is found
- Further checks show no new entering intersections

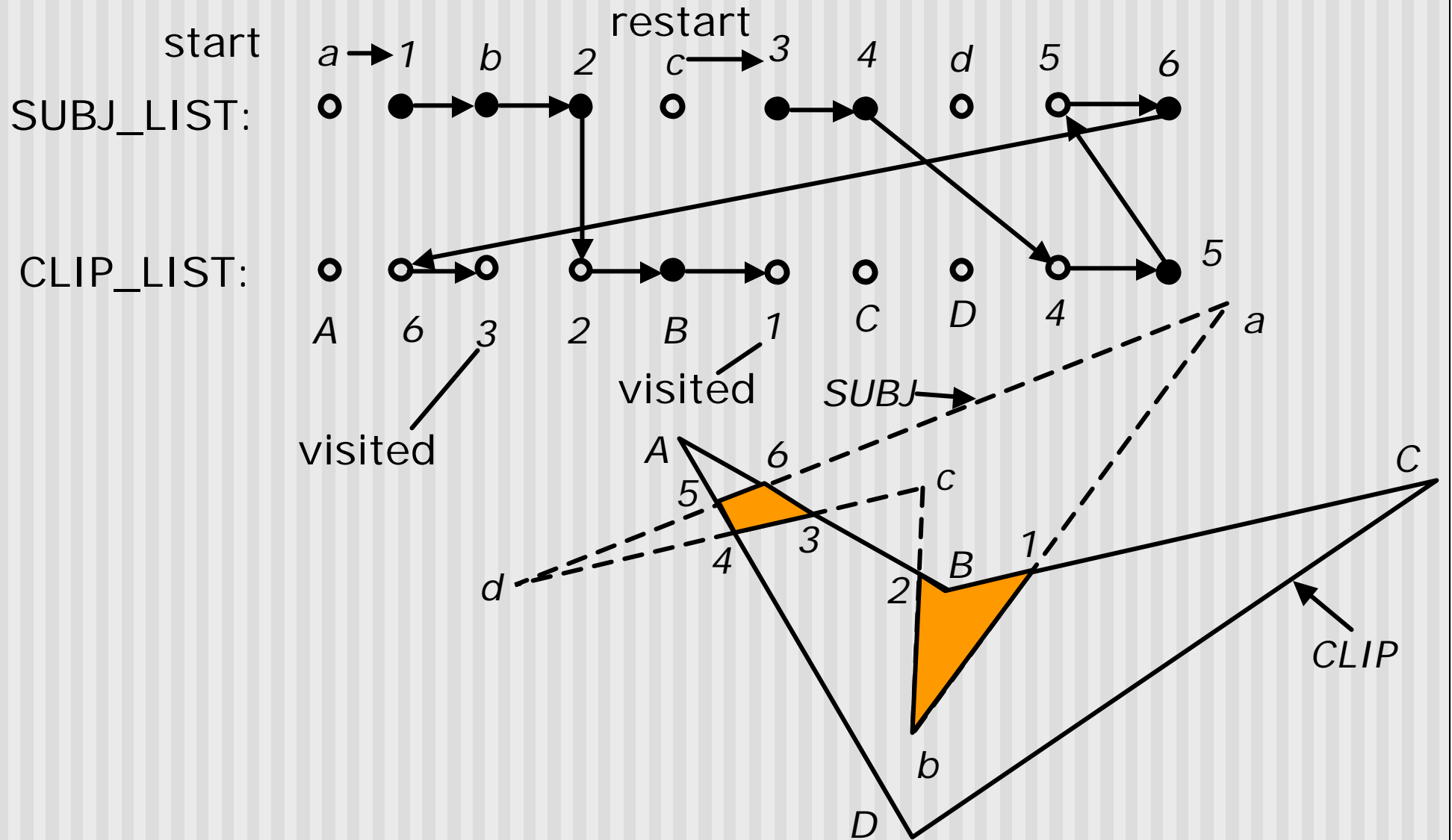


Weiler-Atherton Clipping Algorithm

- Can be implemented using 2 simple lists
- List all ordered vertices **and** intersections of SUBJ and CLIP
- SUBJ_LIST: a, 1, b, 2, c, 3, 4, d, 5, 6
- CLIP_LIST: A, 6, 3, 2, B, 1, C, D, 4, 5



Weiler-Atherton Clipping Algorithm



References

- Hill, sections 7.4.4, 4.8.2