Clipping

- Leaving a certain area of the scene to be displayed on the screen and discarding the other parts
- Following processes are not applied to the parts that are eliminated

Clipping

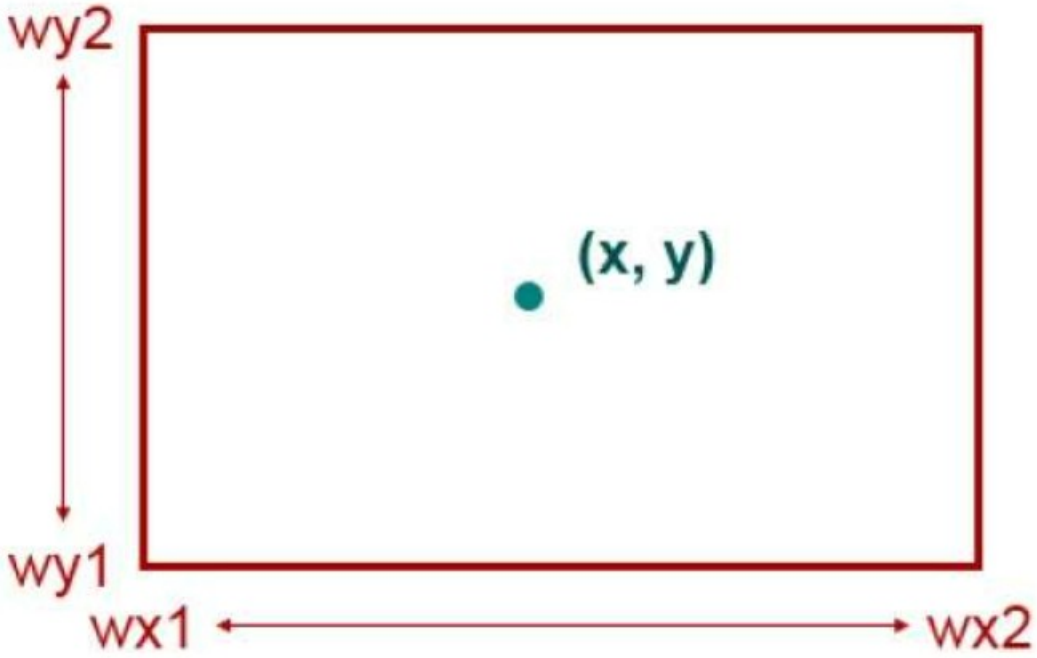
- Which shapes or what part of the shapes are inside the viewing area?
  - Viewing area: clipping window
  - Area for 2D, volume for 3D (a unit cube -with coordinates between -1 and 1)

Clipping

We have a shape and we try to understand whether this shape is inside the clipping window or not, what parts are inside

- Point clipping (simplest one)
- Line clipping
  - Cohen-Sutherland algorithm
  - Liang-Barsky algorithm
  - Nicholl-Lee-Nicholl algorithm
- Fill-area (Polygon) clipping
  - Sutherland-Hodgeman algorithm
  - Weiler-Atherton algorithm

Point Clipping



$wx1 \leq x \leq wx2$

$wy1 \leq y \leq wy2$

If the x and y coordinates of the point are in the window, the following steps in the pipeline will be applied to this point.

Cohen-Sutherland Line Clipping

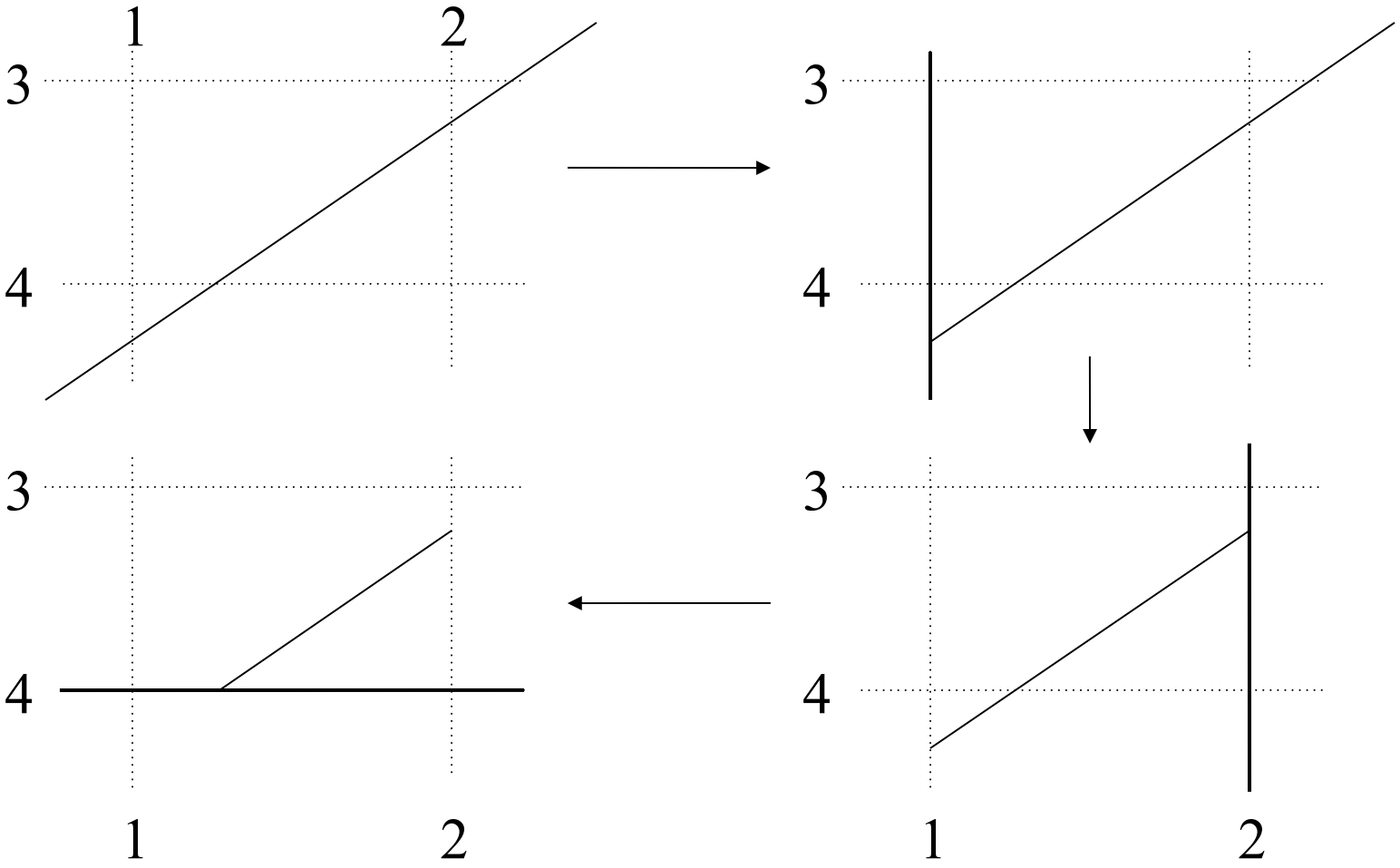
- Clip line against each edge of clip region in turn, three states (evaluate the line with respect to the four borders of the clipping window separately):

  - If both endpoints (start and end) outside, discard line and stop

  - If both endpoints in, continue to next edge (or finish)

  - If one in, one out, chop line at crossing point and continue

Cohen-Sutherland Line Clipping



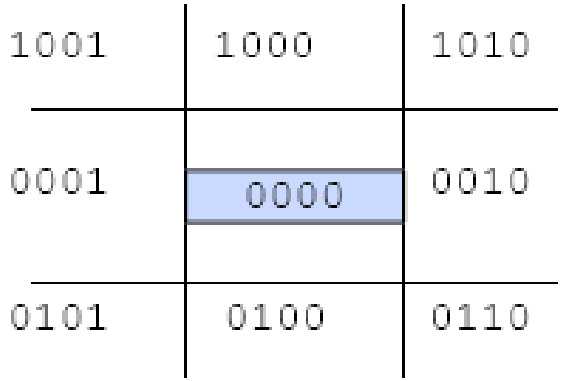
Cohen-Sutherland Line Clipping

- Some cases can lead to early acceptance or rejection (no need to calculate the intersection and clipping):
  - If both endpoints are inside all edges
  - If both endpoints are outside one edge

Cohen-Sutherland Line Clipping

outside or inside information by using 4-bit codes (outcode)

1: outside, 0: inside



Bit 1: left of the left border
Bit 2: right of the right border
Bit 3: below the bottom border
Bit 4: above the top border

Cohen-Sutherland Line Clipping

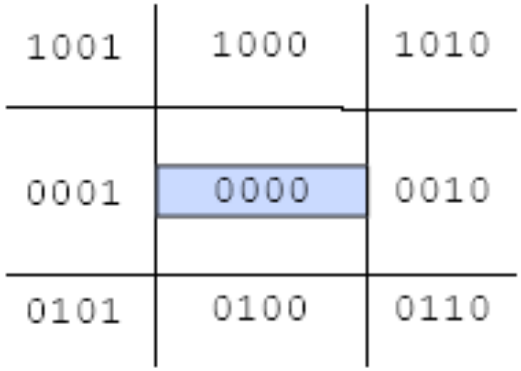
if x1 is smaller than x_{min} (left border of the clipping window) then bit 1 is 1

if x1 is greater than x_{max} (right border of the clipping window) then bit 2 is 1

if y1 is smaller than y_{min} (bottom border of the clipping window) then bit 3 is 1

if y1 is greater than y_{max} (top border of the clipping window) then bit 4 is 1

```
b1=bits(x1,y1) ; b2=bits(x2,y2);
if (b1==0 && b2==0) {
/* both end points inside
   trivial accept          */
} else if (b1 & b2) {
/* line is completely outside
   ignore it               */
} else {
/* needs further calculation*/
}
```

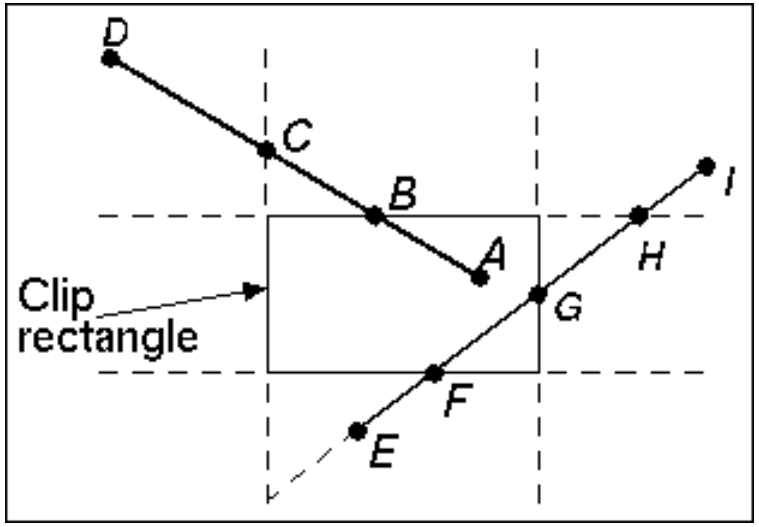


b1 AND b2 = 0 if both end points inside clipping window. Accept

b1 AND b2 ≠ 0 if line end points are in the same half-space defined by an edge of the clipping window. Reject

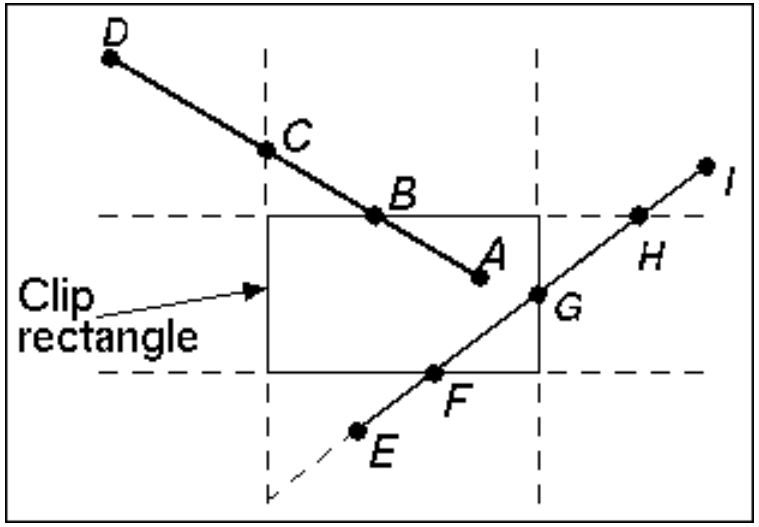
Else subdivide the line into two segments at the point where it crosses a clipping rectangle edge. Reject segment(s) outside the clipping edge

Cohen-Sutherland Line Clipping



* Consider line AD (above).
* Point A has outcode 0000 and point D has outcode 1001 (left of the left border, above the top border).
* Let the order in which the algorithm tests edges be top-to-bottom, left-to-right.
* The top edge is tested first, and line AD is split into lines DB and BA.
* Line BA is trivially accepted (both A and B have outcodes of 0000).
* Line DB is in the outside halfspace of the top edge, and gets rejected.
* What if the order of this operation was first left-to-right and then top-to-bottom control?

Cohen-Sutherland Line Clipping



* C would be the intersection point.
* The DC segment would be clipped.
* Then with respect to the top edge, B would be the intersection point and the CB above B would be clipped. There would be 2 clipping operations.
* Extra clipping can be done according to the order of operation. The order we have determined may not give better results in the same way in all cases. There is no such thing as a best order (disadvantage of the algorithm).

Cohen-Sutherland Line Clipping

To find the intersections, first find the slope of the line.

$$m = \frac{(y_2 - y_1)}{(x_2 - x_1)}$$

repeat for $border = \{\text{LEFT}, \text{RIGHT}, \text{BOTTOM}, \text{TOP}\}$

if $-(bits(x_1, y_1) \vee bits(x_2, y_2))$

Both inside, accept line and terminate

else if $bits(x_1, y_1) \wedge bits(x_2, y_2)$

Both outside same region, reject line and terminate

if $border = \text{LEFT} \vee border = \text{RIGHT}$

$$y_p = y_1 + m(x_{border} - x_1), x_p = x_{border}$$

else

$$x_p = x_1 + \frac{(y_{border} - y_1)}{m}, y_p = y_{border}$$

if $bordertest(x_1, y_1)$

$$x_1 = x_p; y_1 = y_p$$

else if $bordertest(x_2, y_2)$

$$x_2 = x_p; y_2 = y_p$$

update line endpoint coordinates

Cohen-Sutherland Line Clipping

P_1 OR P_2=0111 (all of the bits are not zero, we can not accept it)

P_1 AND P_2=0000 (none of the bits is one, no edge that both endpoints are outside at the same time, we can not reject the line

$$bits(0,0)=0101 \quad bits(8,5)=0010$$

$$bits(0,0) \wedge bits(8,5)=0000$$

$$m=\frac{(5-0)}{(8-0)}=\frac{5}{8}$$

$$\textbf{LEFT}: \quad y_p=y_1+\frac{5}{8}(1-0)=\frac{5}{8}$$

$$\text{LEFT}(P_1) \rightarrow P_1'=(1,5/8)$$

$$\textbf{RIGHT}: \quad y_p=y_1+\frac{5}{8}(7-1)=\frac{5}{8}+\frac{5}{8}6=\frac{35}{8}$$

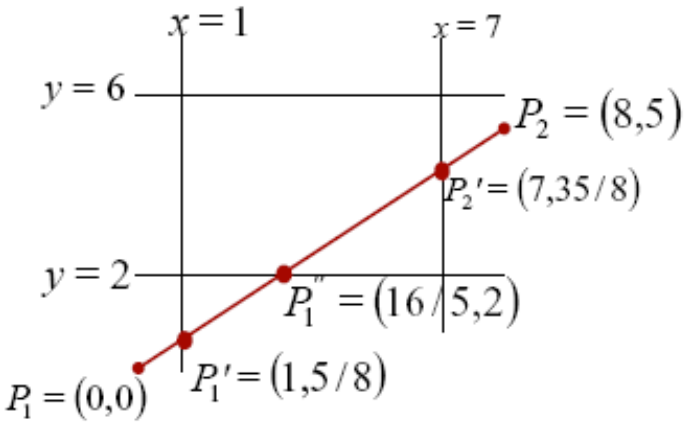
$$\text{RIGHT}(P_2) \rightarrow P_2'=(7,35/8)$$

$$\textbf{BOTTOM}: \quad x_p=1+\left(2-\frac{5}{8}\right)\frac{8}{5}=1+\frac{16}{5}-1=\frac{16}{5}$$

$$\text{BOTTOM}(P_1') \rightarrow P_1''=(16/5,2)$$

$\textbf{TOP}: P_1''$ inside, P_2' inside, so terminate

$$L=(16/5,2) \text{ to } (7,35/8)$$



if $border = \text{LEFT} \vee border = \text{RIGHT}$

$$y_p = y_1 + m(x_{border} - x_1), x_p = x_{border}$$

else

$$x_p = x_1 + \frac{(y_{border} - y_1)}{m}, y_p = y_{border}$$

if $bordertest(x_1, y_1)$

$$x_1 = x_p; y_1 = y_p$$

else if $bordertest(x_2, y_2)$

$$x_2 = x_p; y_2 = y_p$$

Cohen-Sutherland Line Clipping

- This algorithm can be very efficient if it can accept and reject primitives trivially
  - If clip window is large wrt scene data
    * Most primitives are accepted trivially
  - If clip window is much smaller than scene data
    * Most primitives are rejected trivially

Liang-Barsky Line Clipping

Avoids the unnecessary clippings of Cohen-Sutherland algorithm

Clipping: Overview of Steps

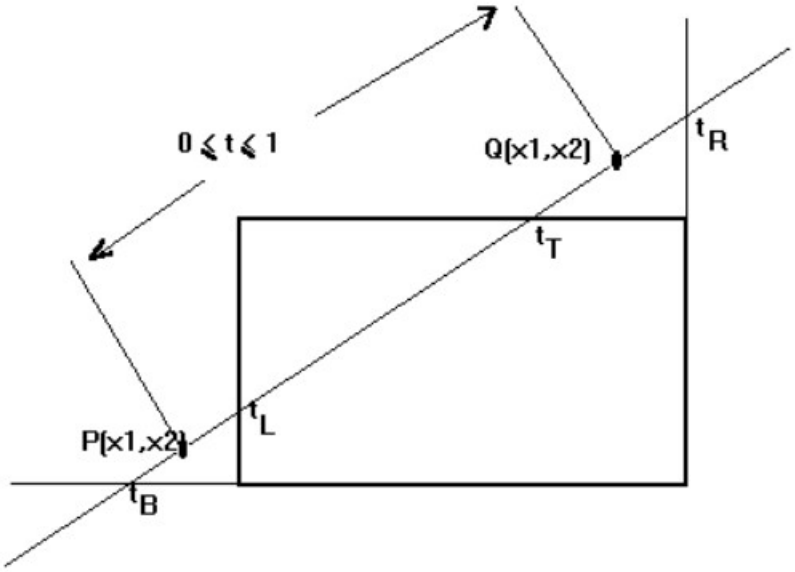
- Express line segments in parametric form
- Derive equations for testing if a point is inside the window
- Compute new parameter values for visible portion of line segment, if any
- Display visible portion of line segment

The relative speed improvement over Sutherland-Cohen algorithm is as follows:

36% for 2D lines
40% for 3D lines
70% for 4D lines

Liang-Barsky Line Clipping

- Parametrically express the line. Clipping conditions are evaluated via the parameter.
- dx and dy values are the difference between the x and y coordinates of the endpoints.
- t parameter provides scaling. If t is 0 we get the first endpoint, if t is 1 we get the second endpoint.
- t takes values between 0 and 1. If it is less than 0 OR greater than 1, points that are outside the line segment but on the same line are obtained.



$$x = x_1 + (x_2 - x_1)*t = x_1 + dx*t$$

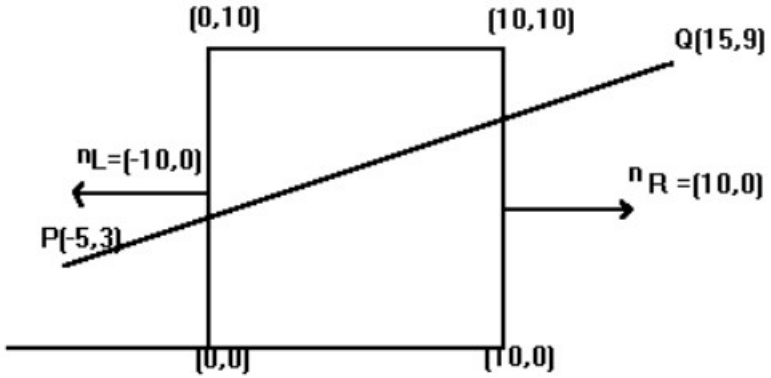
$$y = y_1 + (y_2 - y_1)*t = y_1 + dy*t$$

Liang-Barsky Line Clipping

Algorithm:

- Set tmin=0, tmax=1
- Calculate the values of tL, tR, tT, and tB
  - If t<tmin or t>tmax ignore it and go to the next edge
  - Otherwise classify the t value as entering or exiting value
  - If t value is entering set tmin=t, if t is exiting value set tmax=t
- If tmin<tmax draw a line from (x1 + dx*tmin, y1 + dy*tmin) to (x1 + dx*tmax, y1 + dy*tmax)

Liang-Barsky Line Clipping



x1=-5, y1=3 (P)
x2=15, y2=9 (Q)
dx=x2-x1=20, dy=y2-y1=6

xmin=0, ymin=0 (clipping window)
xmax=10, ymax=10 (clipping window)

p1 (left)=-dx, q1=(x1-xmin)
p2 (right)=dx, q2=(xmax-x1)
p3 (bottom)=-dy, q3=(y1-ymin)
p4 (top)=dy, q4=(ymax-y1)
tk=qk/pk

t1 (tleft)=(x1-xmin)/-(x2-x1)=(-5-0)/-(15-(-5))
=-5/-20=1/4 **pk=-20<0**
(entering)

t2 (tright)=(xmax-x1)/(x2-x1)=(10-(-5))/(15-(-5))
=15/20=3/4 **pk=20>0 (exiting)**

t3 (tbottom)=(y1-ymin)/-(y2-y1)=(3-0)/-(9-3)
=3/-6=-1/2 **t<tmin**

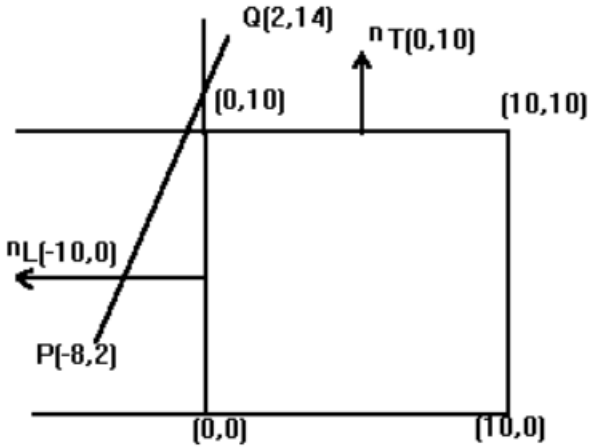
t4 (ttop)=(ymax-y1)/(y2-y1)=(10-3)/(9-3)
=7/6 **t>tmax**

tmin=1/4, tmax=3/4 --> tmin<tmax

A line from (x1 + dx*tmin, y1 + dy*tmin) to (x1 + dx*tmax, y1 + dy*tmax)

from point **(-5+(20)*(1/4), 3+(6)*(1/4))**
to point **(-5+(20)*(3/4), 3+(6)*(3/4))**

Liang-Barsky Line Clipping



x1=-8, y1=2 (P)
x2=2, y2=14 (Q)
dx=x2-x1=10, dy=y2-y1=12

xmin=0, ymin=0 (clipping window)
xmax=10, ymax=10 (clipping window)

p1 (left)=-dx, q1=(x1-xmin)
p2 (right)=dx, q2=(xmax-x1)
p3 (bottom)=-dy, q3=(y1-ymin)
p4 (top)=dy, q4=(ymax-y1)
tk=qk/pk

t1 (tleft)=(x1-xmin)/-(x2-x1)=(-8-0)/-(2-(-8))
=-8/-10=4/5 **pk=-10<0**

t2 (tright)=(xmax-x1)/(x2-x1)=(10-(-8))/(2-(-8))
=18/10=9/5 **t>tmax**

t3 (tbottom)=(y1-ymin)/-(y2-y1)=(2-0)/-(14-2)
=2/-12=-1/6 **t<tmin**

t4 (ttop)=(ymax-y1)/(y2-y1)=(10-2)/(14-2)
=8/12=2/3 **pk=12>0**

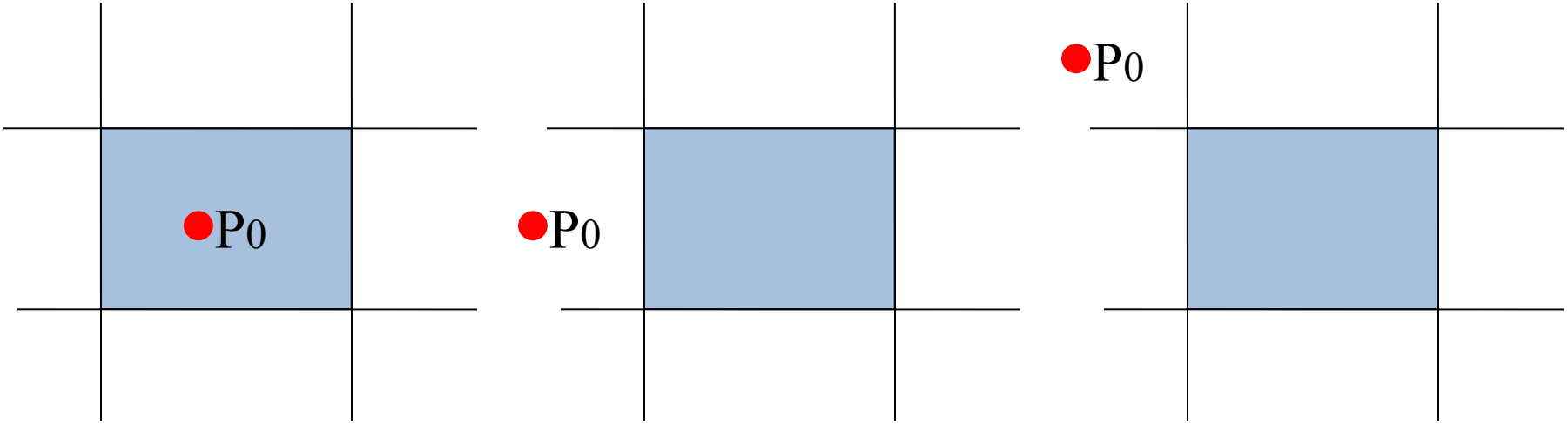
tmin=4/5 tmax=2/3

No line will be drawn.

Nicholl-Lee-Nicholl Line Clipping

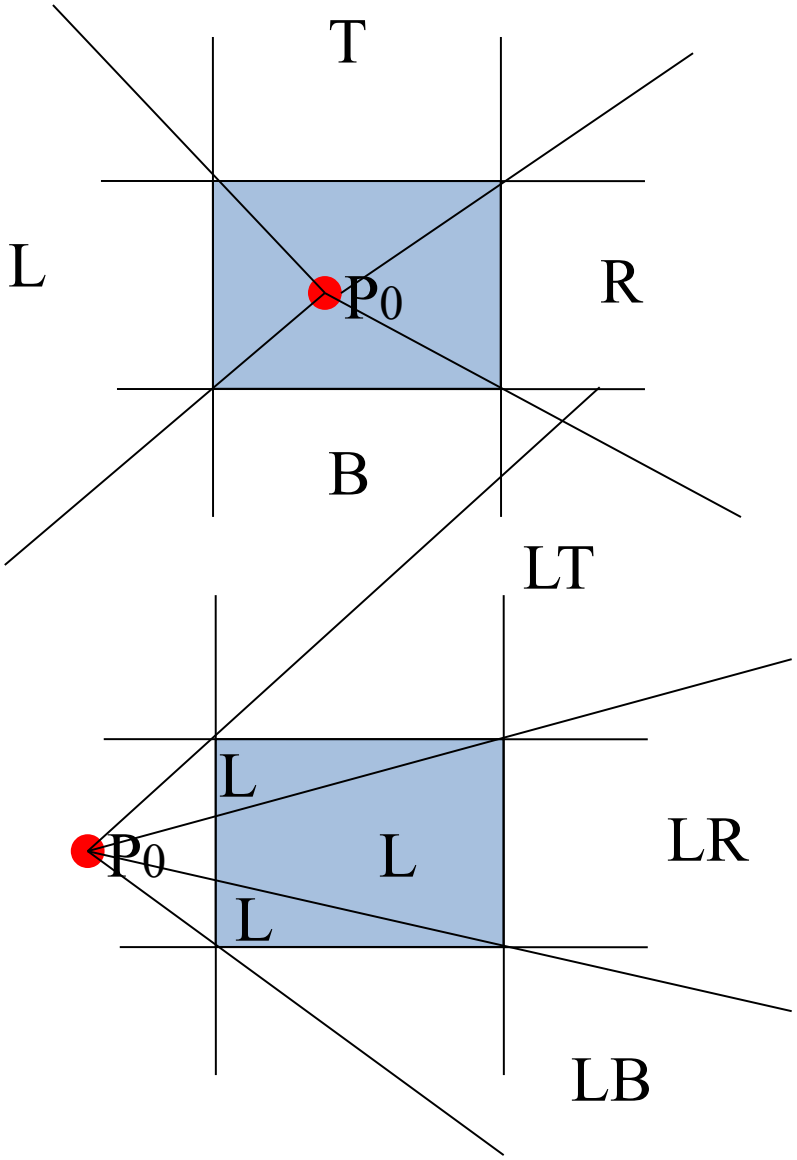
- Aim is avoiding calculations of extra intersection points
- Initial testing to determine if a line segment is completely inside the clipping window can be done using previous methods
- If trivial acceptance or rejection is not possible the NLN algorithm sets up additional regions

Nicholl-Lee-Nicholl Line Clipping



- There are three different positions to consider for a line:
  endpoints can be inside,
  outside and on the right, left, bottom or top,
  outside and in one of the corners.
- For each case of P_0, we create custom test zones for the other endpoint, P_{end}
- These regions tell us, with respect to which edges we should clip the line, by simple tests (by looking at the slope or the coordinates)

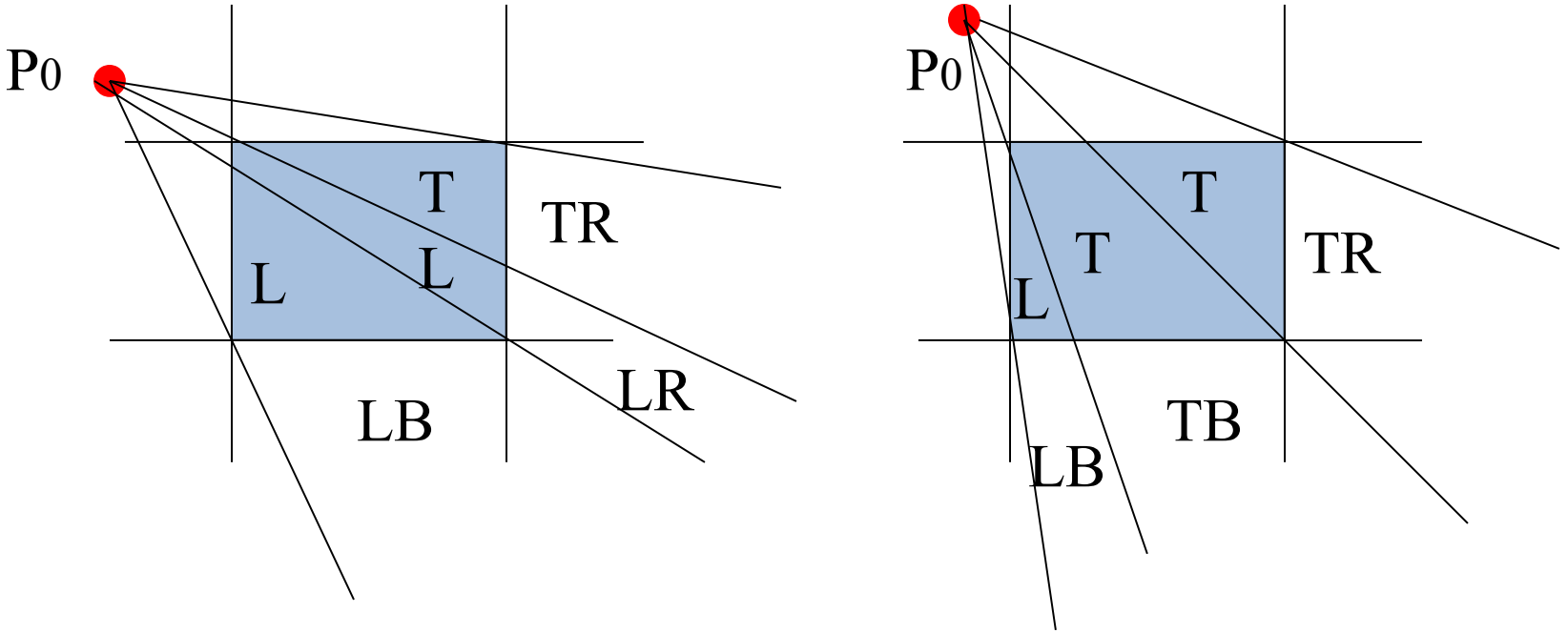
Nicholl-Lee-Nicholl Line Clipping



Find which of the four regions P_{end} lies in, then calculate the line intersection with the corresponding boundary

If P_{end} is not inside any of these regions, we will reject the line, no clipping.

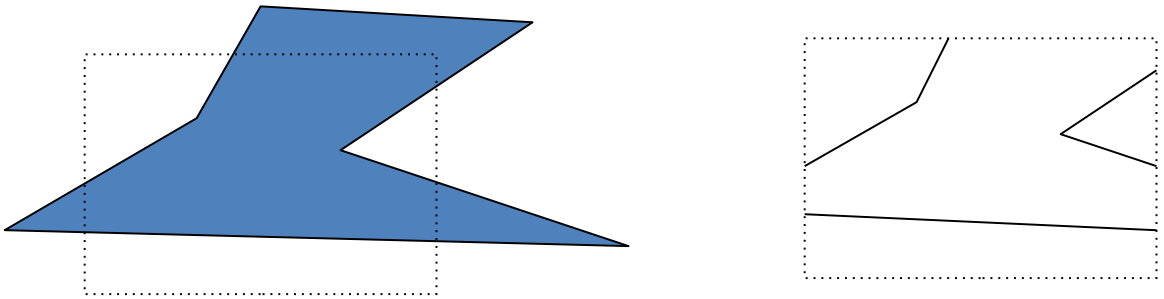
Nicholl-Lee-Nicholl Line Clipping



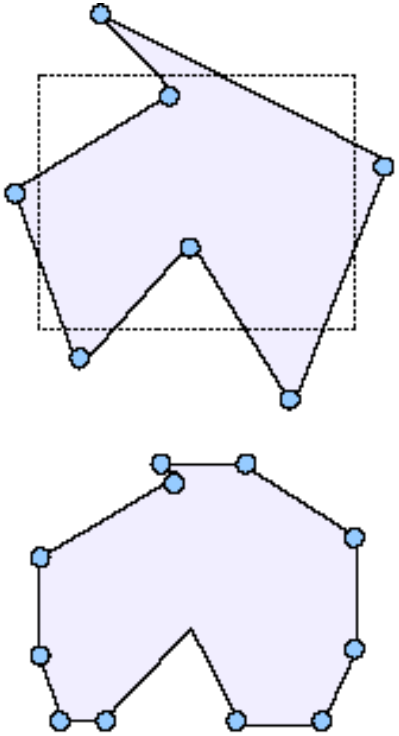
* We have 2 sub cases: We have different regions for P_{end} if P_0 is close to left of the window or top of the window
* Find which of the five regions P_{end} lies in, then calculate the line intersection with the corresponding boundary
* When you switch to 3D, the number of different cases increases much more and the algorithm ceases to be useful

Polygon Clipping

- Clipping a polygon fill area needs more than line-clipping of the polygon edges (If line clipping is applied only, we get a set of unconnected lines)
- Must generate one or more closed polylines, which can be filled with the assigned colour or pattern



Sutherland-Hodgeman Polygon Clipping

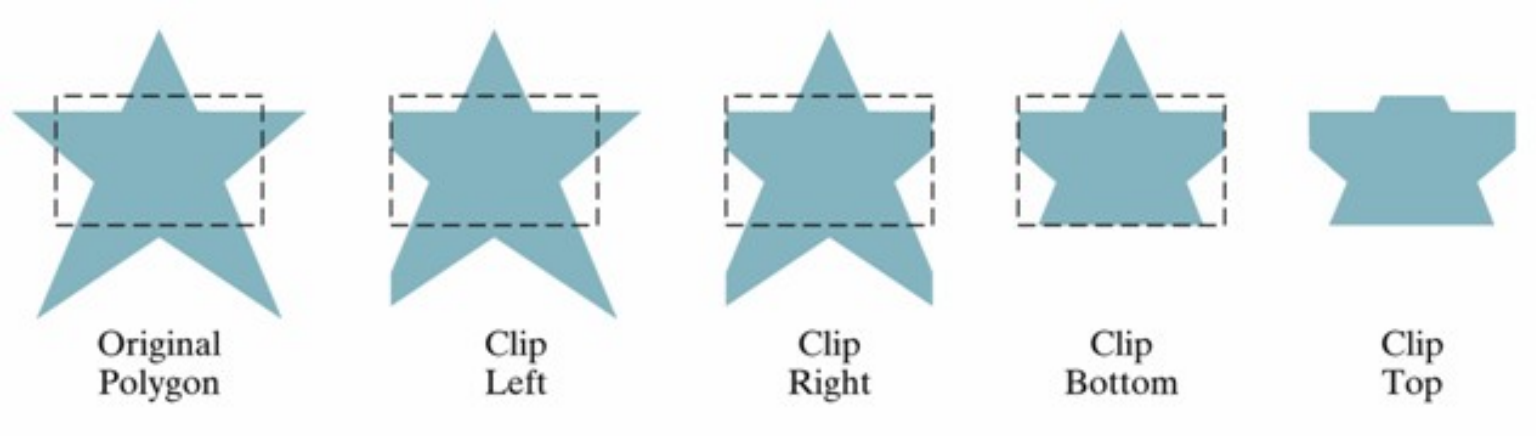


* Check each edge of the polygon against all window boundaries

* Modify the vertices based on transitions

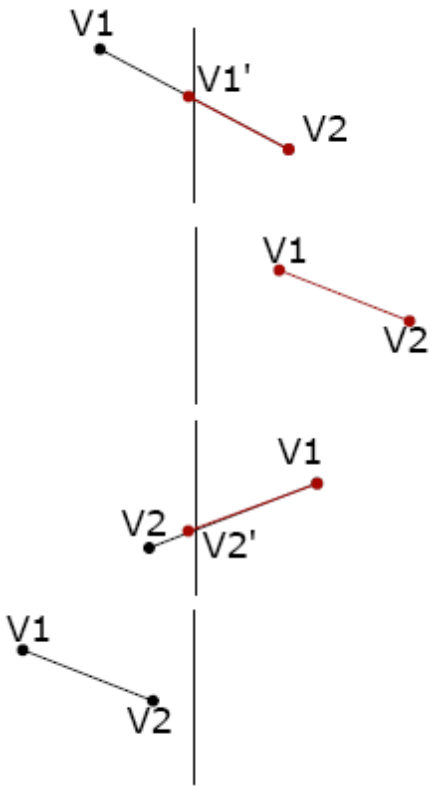
* Transfer the new edges to the next clipping boundary

Sutherland-Hodgeman Polygon Clipping



- The polygon is clipped first with respect to the left edge of the clipping window
- Afterwards, cropping is done according to the right edge with the new vertices (obtained after the previous clipping)
- Clipping is done according to the bottom and top edges also

Sutherland-Hodgeman Polygon Clipping



Polygon vertices have four cases with respect to each edge (case for left edge):

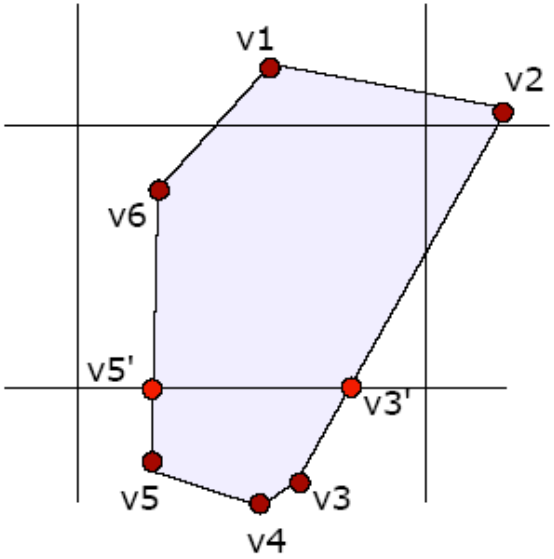
* V1 outside, V2 inside: take V1’ and V2

* V1 inside, V2 inside: take V1 and V2

* V1 inside, V2 outside: take V1 and V2’

* V1 outside, V2 outside: take none

Sutherland-Hodgeman Polygon Clipping

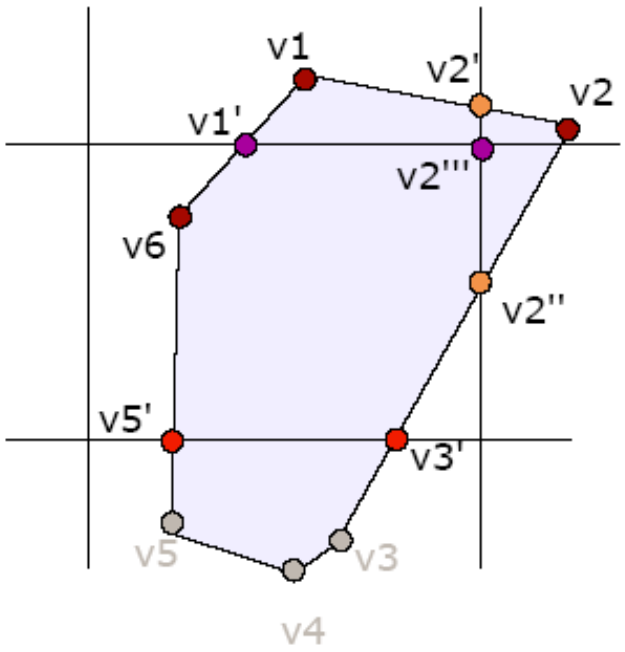


* Check wrt left border (clockwise)
    v1 and v2 inside: take v1 v2
    v2 and v3 inside: take v2 v3
    ...
    v1, v2, v3, v4, v5, v6, v1

* Check wrt bottom border (clockwise)
    v1 and v2 inside: v1 v2
    v2 inside, v3 outside: v2 v3’
    v3 and v4 outside: take none
    v4 and v5 outside: take none
    v5 outside, v6 inside: v5’ v6
    v6 and v1 inside: v6 v1

    v1, v2, v3’, v5’, v6, v1 (started with v1 and finished with v1 to get a closed area)

Sutherland-Hodgeman Polygon Clipping



* Check wrt right border (clockwise)

Input vertices: v1, v2, v3', v5', v6, v1

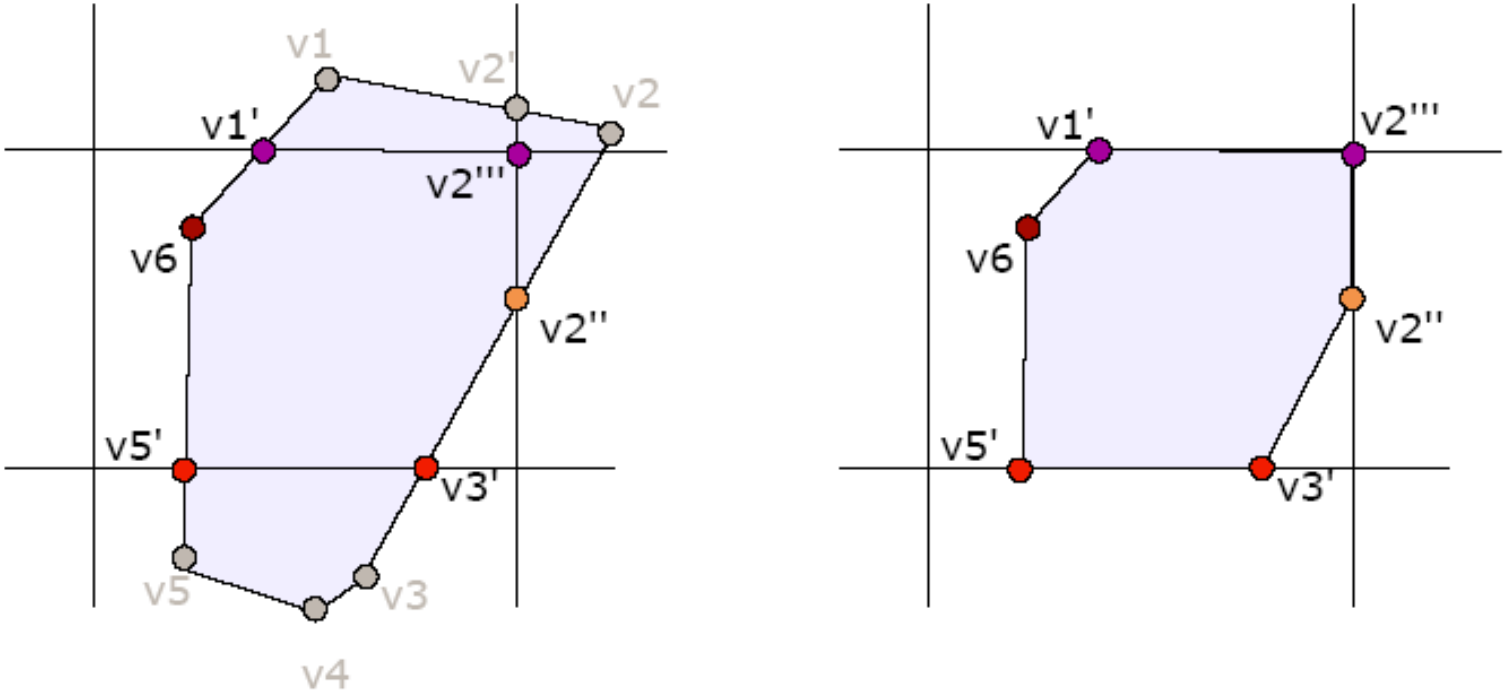
v1 inside, v2 outside: take v1 v2'
v2 outside, v3' inside : take v2'' v3'
v3' and v5' inside : take v3' v5'
v5' and v6 inside : take v5' v6
v6 and v1 inside : take v6 v1
v1, v2', v2'',v3', v5', v6, v1

* Check wrt top border (clockwise)

v1 and v2' outside: take none
v2' outside, v2'' inside: take v2''' v2''
v2'' and v3' inside: take v2'' v3'
v3' and v5' inside: take v3' v5'
v5' and v6 inside: take v5' v6
v6 inside, v1 outside: take v6 v1'

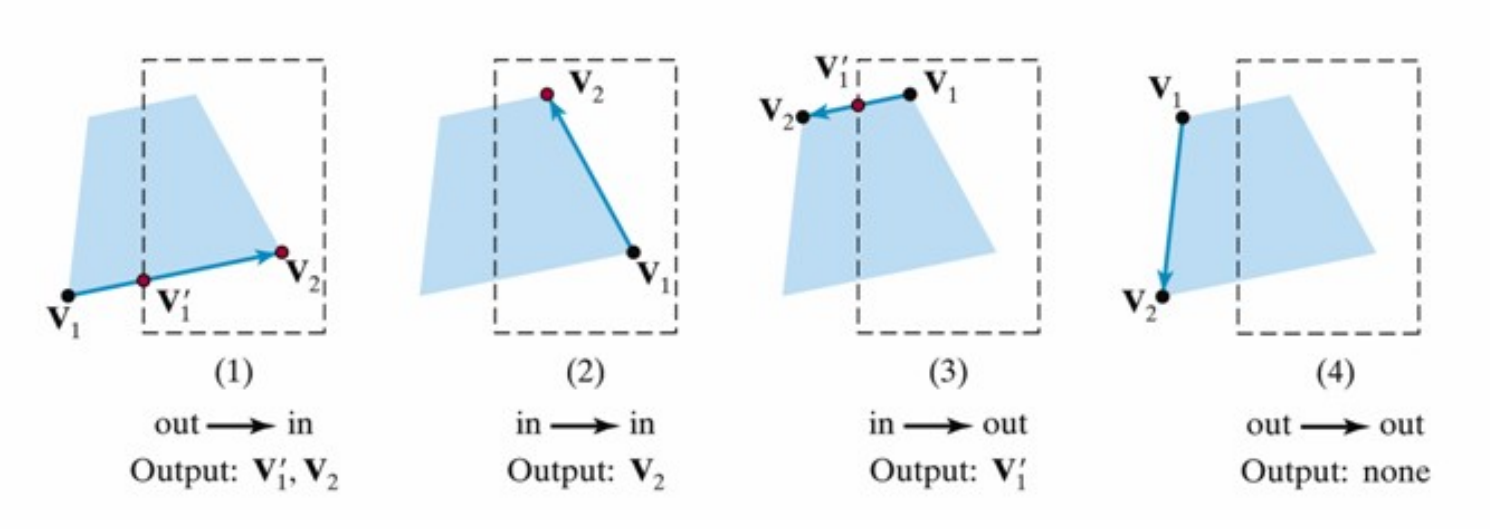
v2''', v2'', v3', v5', v6, v1'

Sutherland-Hodgeman Polygon Clipping



Input (left) & result (right)

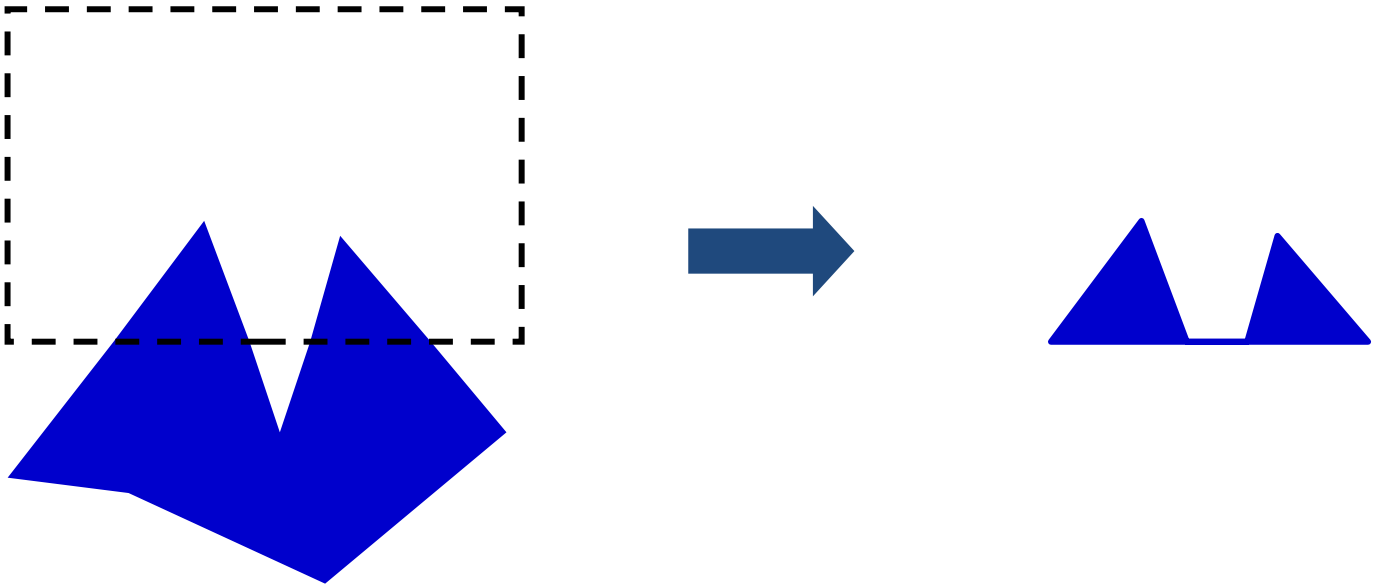
Sutherland-Hodgeman Polygon Clipping



Clipping operation of the four sides of the polygon, relative to the left edge of the clipping window

Sutherland-Hodgeman Polygon Clipping

- The algorithm correctly clips convex polygons, but may display extraneous lines for concave polygons

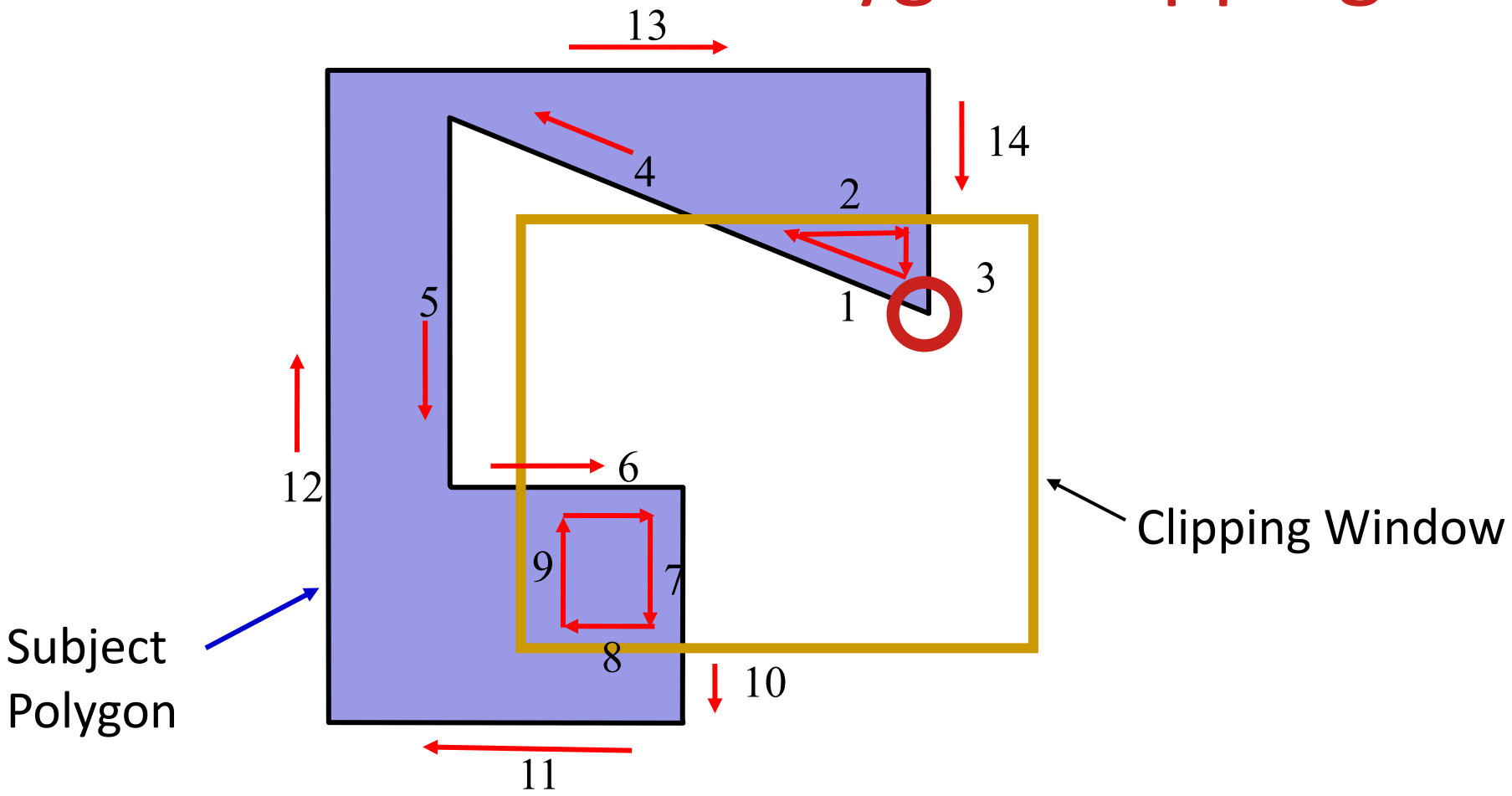


result of clipping should be 2 separate triangles, but a line is added between these 2 triangles (a triangle)

Weiler-Atherton Polygon Clipping

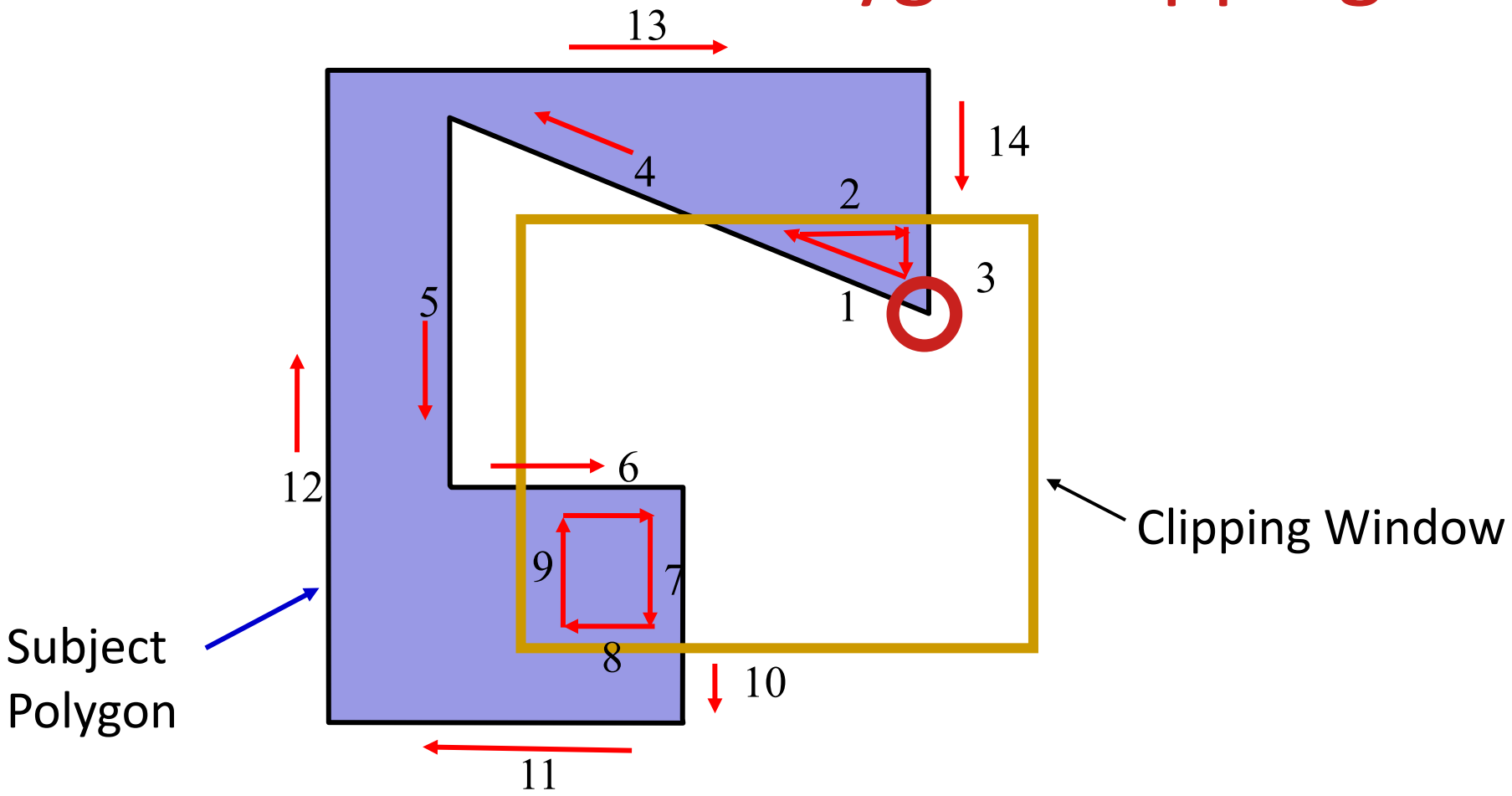
- Another approach to polygon clipping *(solves the problem that the Sutherland-Hodgeman algorithm can have for concave polygons)*
- Avoids finding unnecessary intersections
- Clockwise or counter clockwise
- Should start from a vertex inside the clipping window

Weiler-Atherton Polygon Clipping



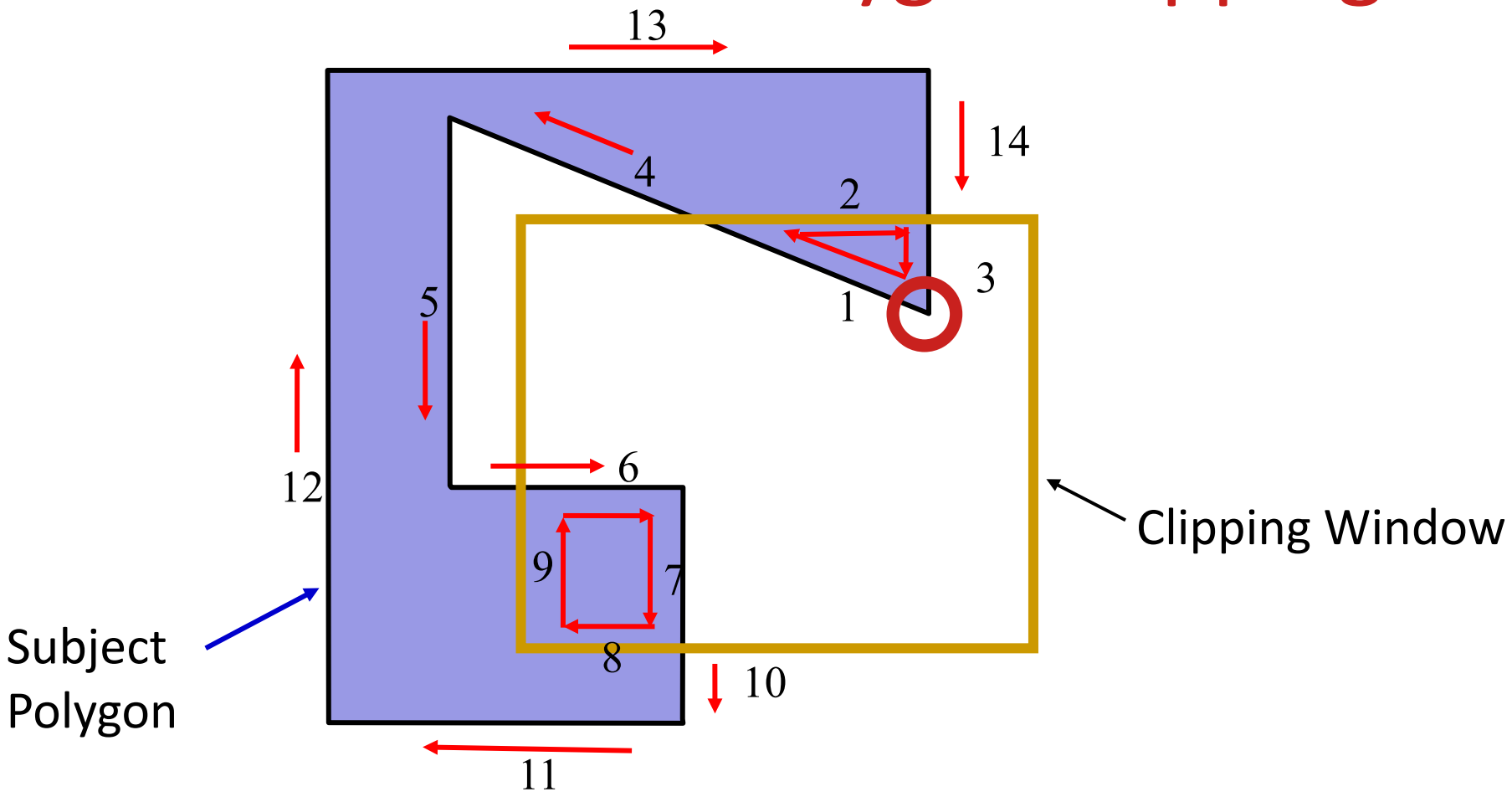
- We should start from a vertex within the clipping window. We start from the vertex inside the circle.
- If there is a transition from inside the window to the outside, we save the intersection point and turn right and continue from the window until next intersection.
- From this intersection point, turn right again, and follow subject polygon until closed. (123 is the closed area)

Weiler-Atherton Polygon Clipping



- We turned right as we ran the algorithm clockwise. If we had worked counterclockwise, we would have turned left.
- Starting from the first intersection point we recorded, we continue over the polygon. If there is a transition from outside to inside, we find and save the intersection again and continue from the polygon.

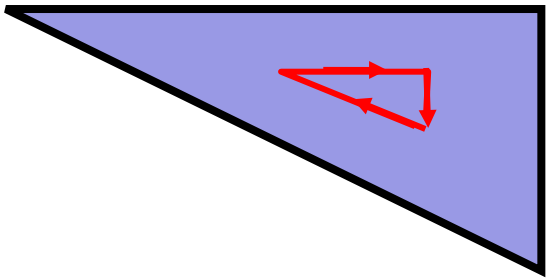
Weiler-Atherton Polygon Clipping

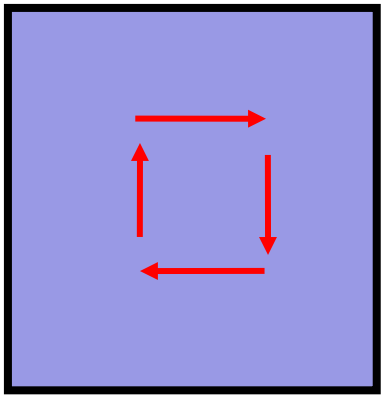


- There is a transition from inside to outside. We recorded the intersection point. We continue through the window. We created the enclosed area: 6789.
- Starting from the first intersection point (from inside to outside) we recorded, we continue over the polygon.
- 10 11 12 13 14 and we have reached the starting point.

Weiler-Atherton Polygon Clipping

Results





Weiler-Atherton Polygon Clipping

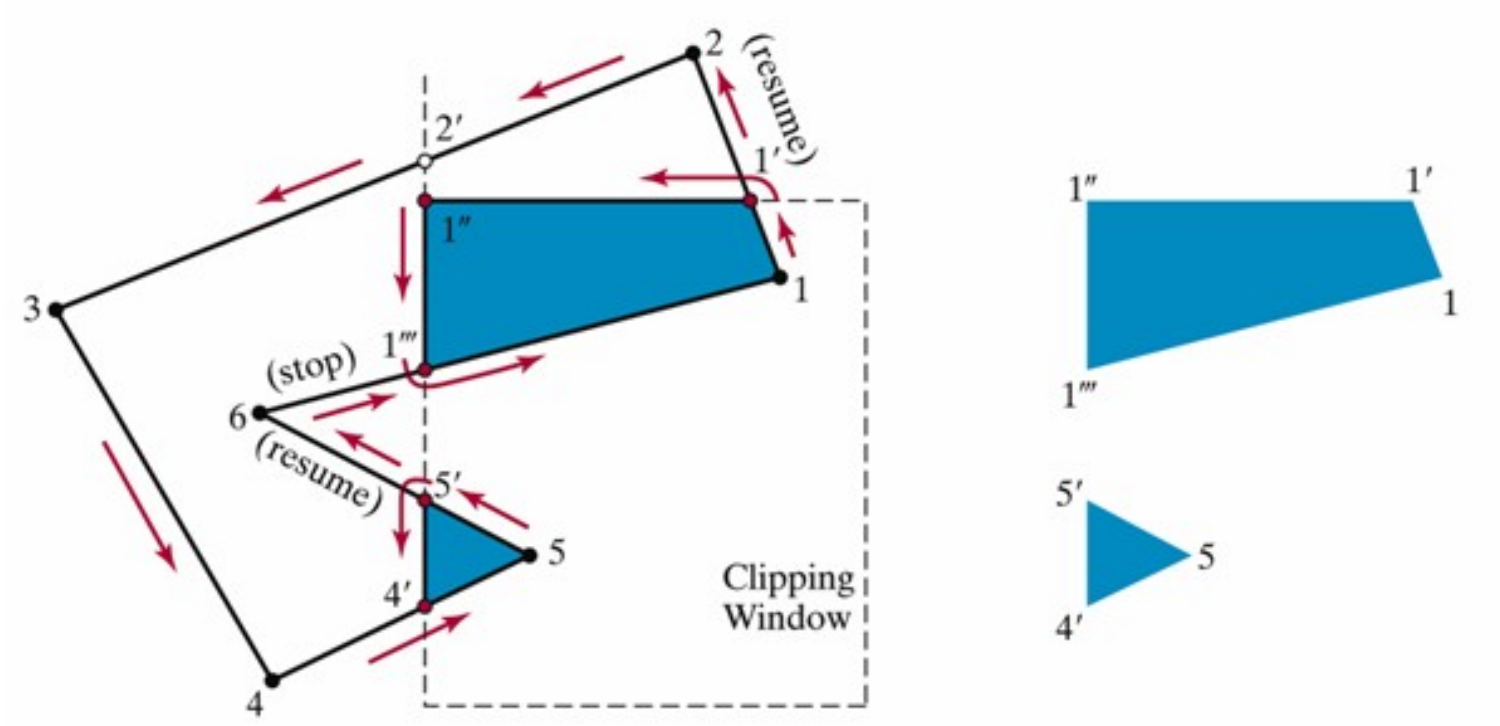
Steps of the algorithm:

- Start at first (inside) vertex
- Traverse polygon until hitting a window boundary
- Output intersection point i
- Turn right
- Follow window boundary until next intersection

Weiler-Atherton Polygon Clipping

- Output second intersection
- Turn right, again, and follow subject polygon until closed
- Continue on subject polygon from first intersection point.
- Repeat processing until complete

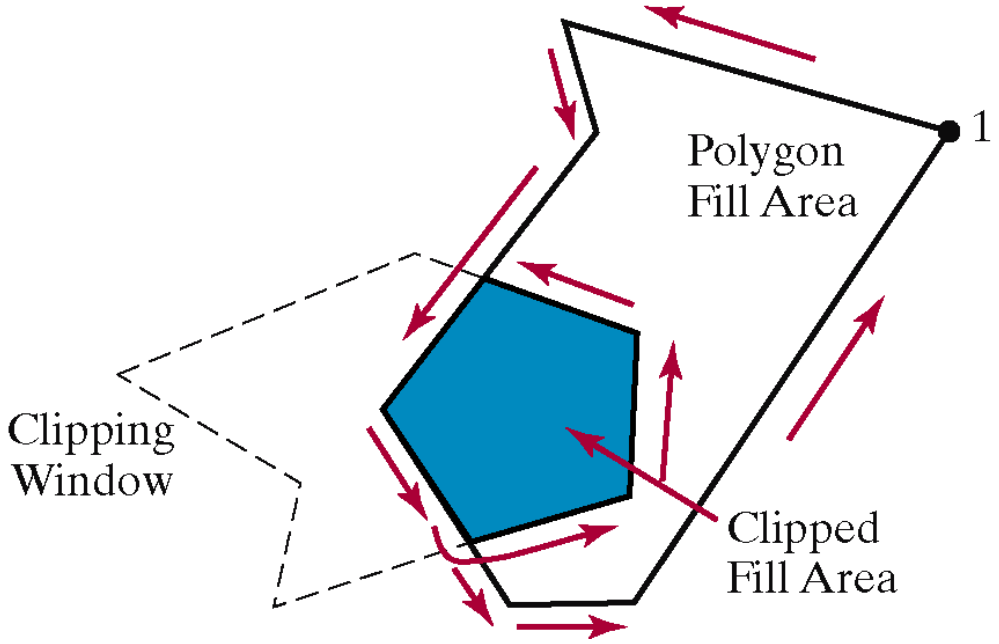
Weiler-Atherton Polygon Clipping



an example of applying the same algorithm in counter-clockwise direction

Weiler-Atherton Polygon Clipping

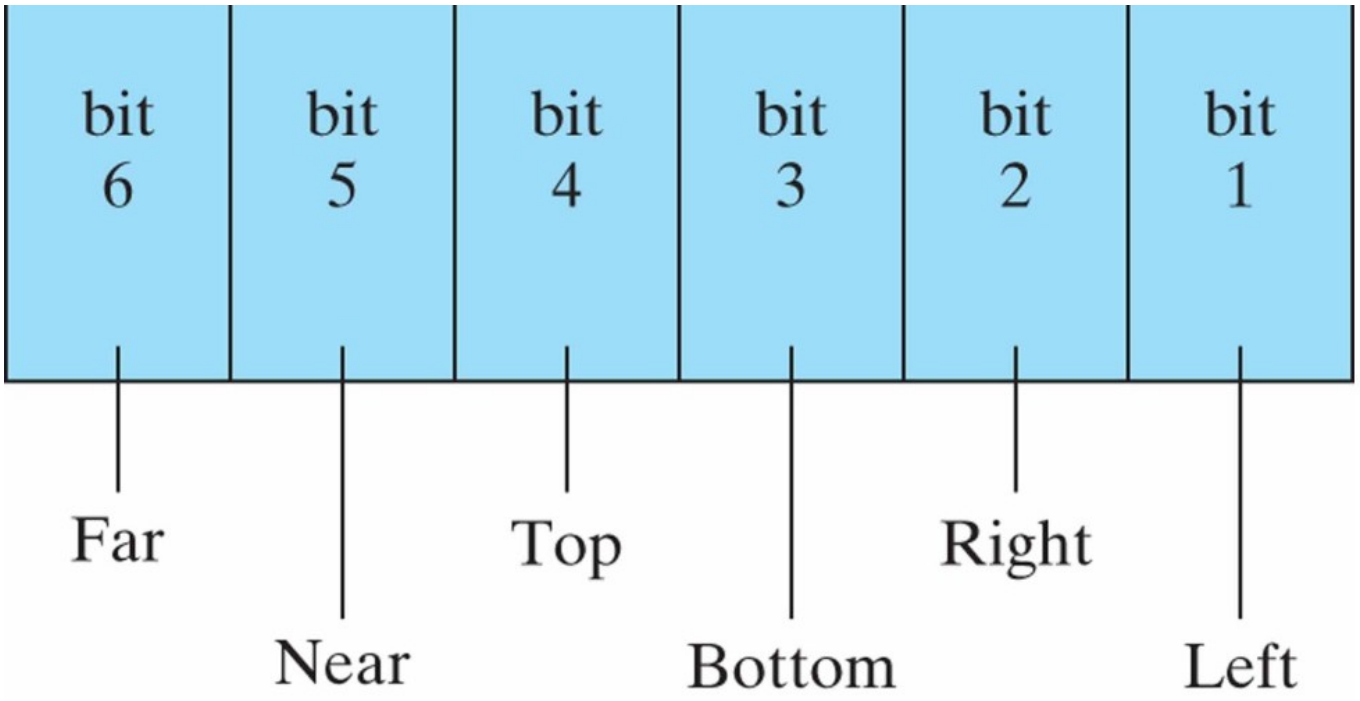
- Polygon clipping using nonrectangular polygon clipping windows



3D Clipping Algorithms

- We clip against planes parallel to Cartesian planes
- The task is to identify object sections within the cube (save parts inside and eliminate parts outside)
- Algorithms are extensions of the 2D algorithms

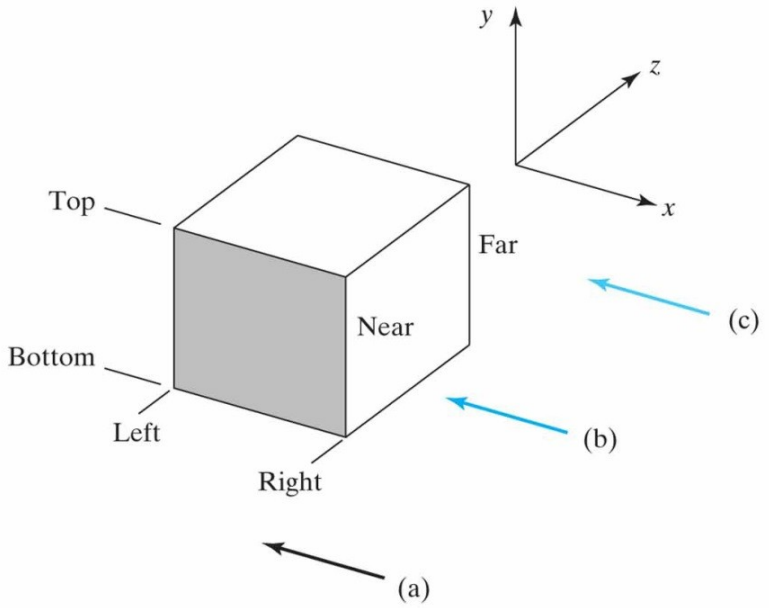
3D Clipping Algorithms



Two more bits for far and near planes

3D Clipping Algorithms

six-bit outcodes that express the positions according to the planes of the visual volume



011001	011000	011010
010001	010000	010010
010101	010100	010110

Region Codes
In Front of Near Plane
(a)

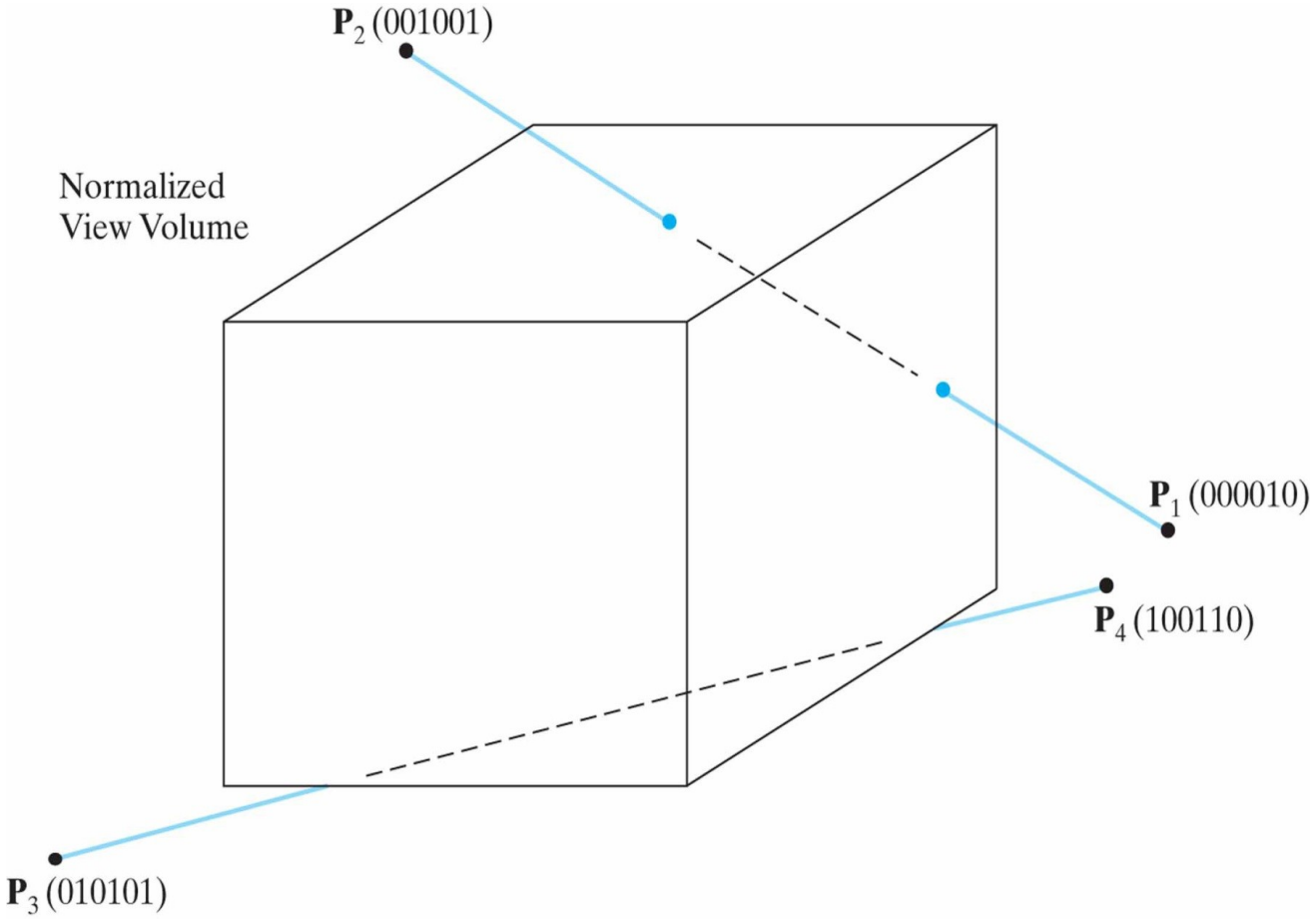
001001	001000	001010
000001	000000	000010
000101	000100	000110

Region Codes
Between Near and Far Planes
(b)

101001	101000	101010
100001	100000	100010
100101	100100	100110

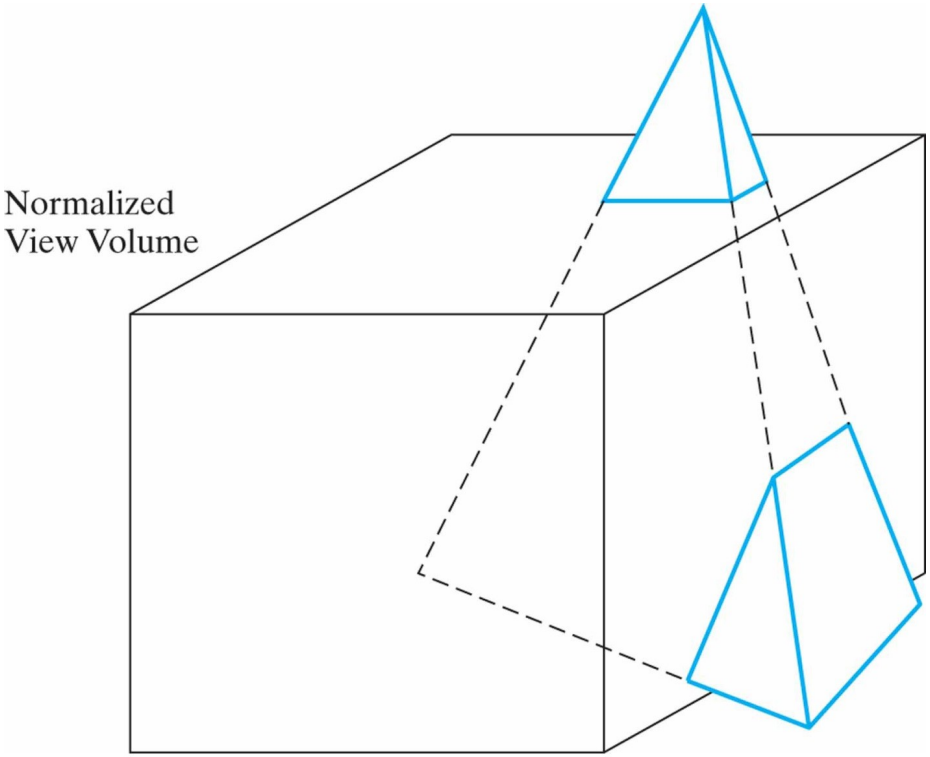
Region Codes
Behind Far Plane
(c)

3D Clipping Algorithms



3D outcodes for the two lines. The P1P2 line crosses the right and upper clipping boundaries of the visual volume, while the P3P4 line remains completely below the bottom clipping plane

3D Clipping Algorithms



3D object clipping: According to the clipping planes of the visual volume, the remaining surface parts are eliminated from the object definition. As a result, new surface polygons will need to be created.