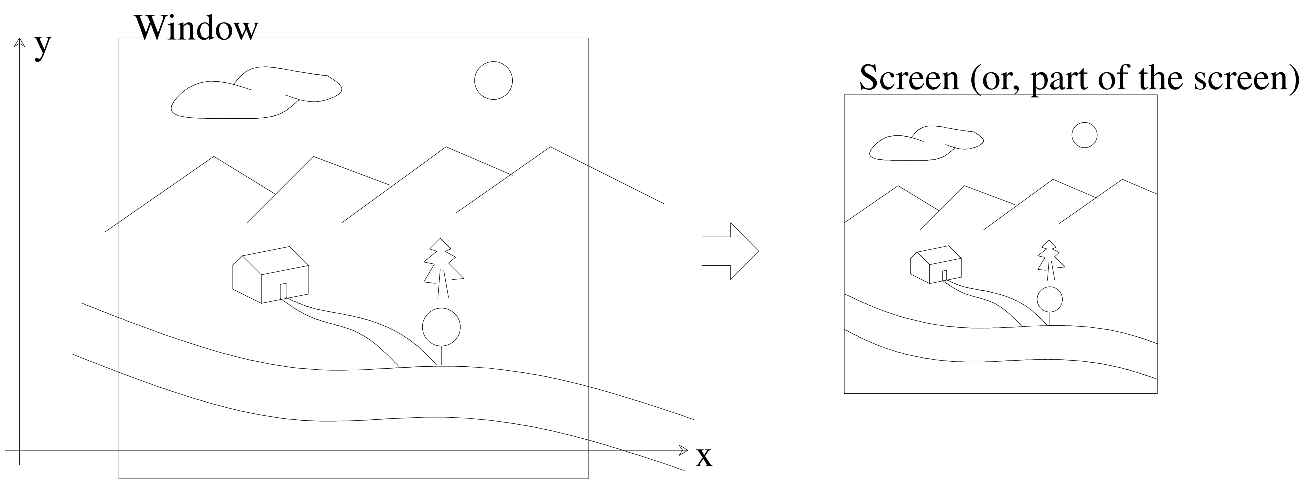
Clipping Output Primitives

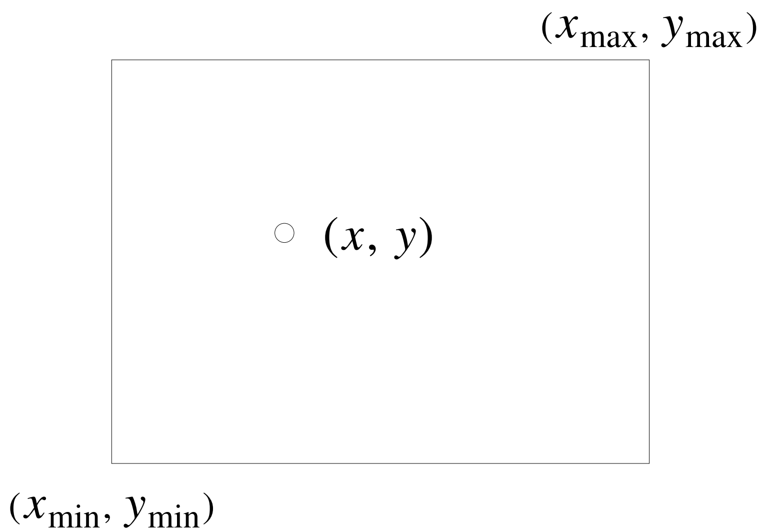
- The process of removing the invisible portions of the output primitives while working with the world coordinate system (WCS)
- Clipping is necessary to avoid the "wrap-around" and "internal register overflow" problems
- Points and lines lying on the window border are considered inside.



- **Clipping** and **mapping** are the responsibility of the application programmer

Primitives: points, lines, polygons, text

Point clipping:



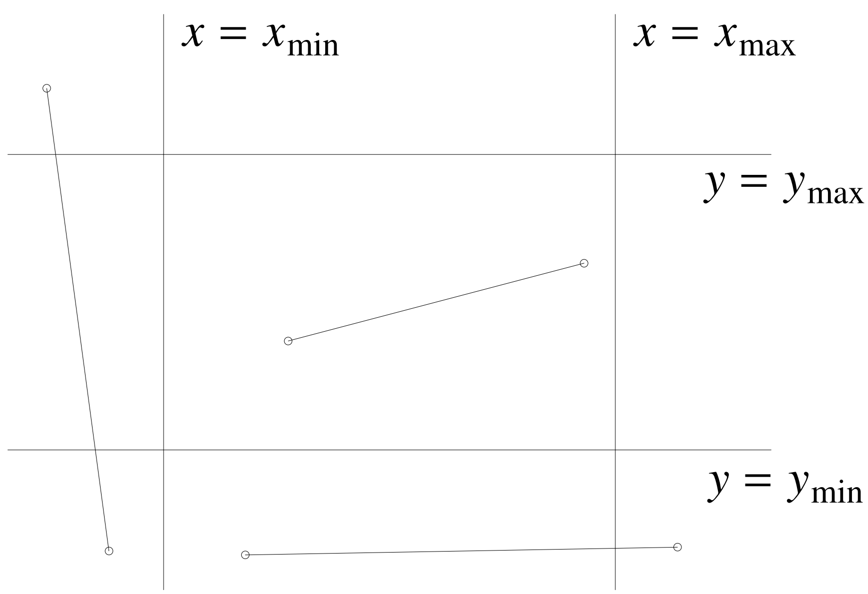
To determine if a point (x, y) is inside a window defined by (x_{min}, y_{min}), lower-left corner, and (x_{max}, y_{max}), upper-right corner, simply test if

$$x_{min} \le x \le x_{max}$$

$$y_{min} \le y \le y_{max}$$

Line clipping: (Cohen-Sutherland algorithm)

- To avoid unnecessary computation, perform tests on trivially accepted cases and trivially rejected cases first
- If both endpoints are inside the window, then the line segment is inside the window
- If both endpoints are to the left ($x < x_{min}$), to the right ($x > x_{min}$), below ($y < y_{min}$), or above ($y > y_{min}$) the window, then the line segment is outside the window



To perform the tests efficiently, divide the world coordinate system into 9 regions and assign each of them a four-bit code

1001	1000	1010
0001	0000	0010
0101	0100	0110

bit 4	bit 3	bit 2	bit 1
top	bottom	right	left

bit 1: sign bit of $(x - x_{\min})$

bit 2: sign bit of $(x_{\max} - y)$

bit 3: sign bit of $(y - y_{\min})$

bit 4: sign bit of $(y_{\max} - y)$

The Cohen-Sutherland Algorithm

1. Compute the codes for the endpoints of the line segment to be clipped

2. Repeat until the line segment is either trivially accepted or rejected

   2.1 [Trivial Acceptance Test]

   If bitwise OR of the codes is 0000 (line segment is inside the window), draw the line segment and stop.

   3. [Trivial Rejection Test]

   If bitwise AND of the codes is not 0000 (line segment is outside the window), discard the line segment and stop.

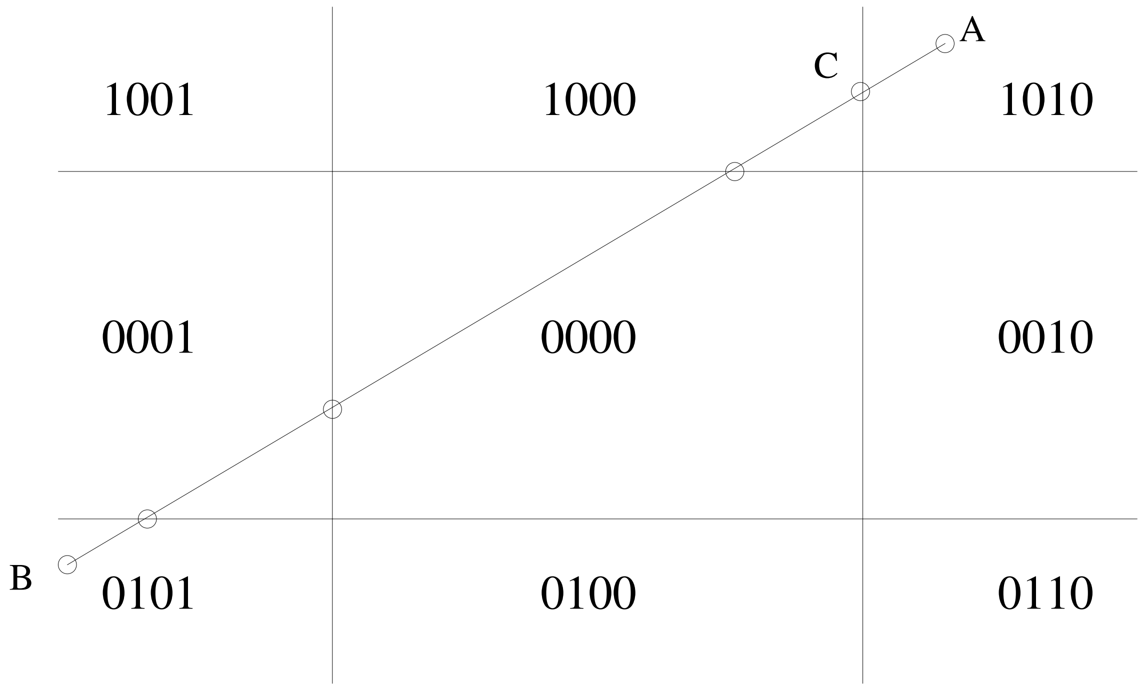
   4. [Subdivide the segment]

      4.1 Pick an endpoint whose code is non-zero (the endpoint that is outside the window)

      4.2 Find the first non-zero bit: this corresponds to the window edge which intersects the line segment

      4.3 Compute the intersection point and replace the outside endpoint with the intersection point

An Example




```
     A:  1010            A:  1010
/    B:  0101       &    B:  0101
-------------       -------------
         1111                0000
```

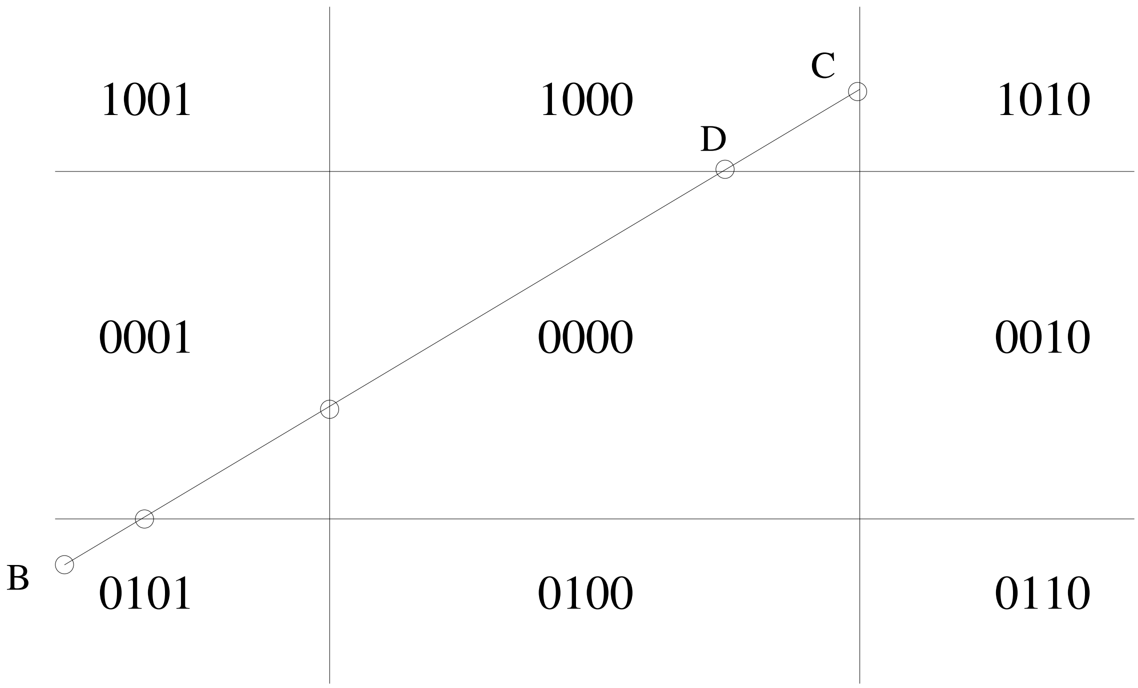
bit 4	bit 3	bit 2	bit 1
top	bottom	right	left

Use bit 2 of A (right clipping edge) to do the subdivision

Subdivide at C (Find y coordinate of C)

$$y = m \cdot x_{\max} + b$$

Example (con’t)



	C:	1000
/	B:	0101
		1101

	C:	1000
&	B:	0101
		0000

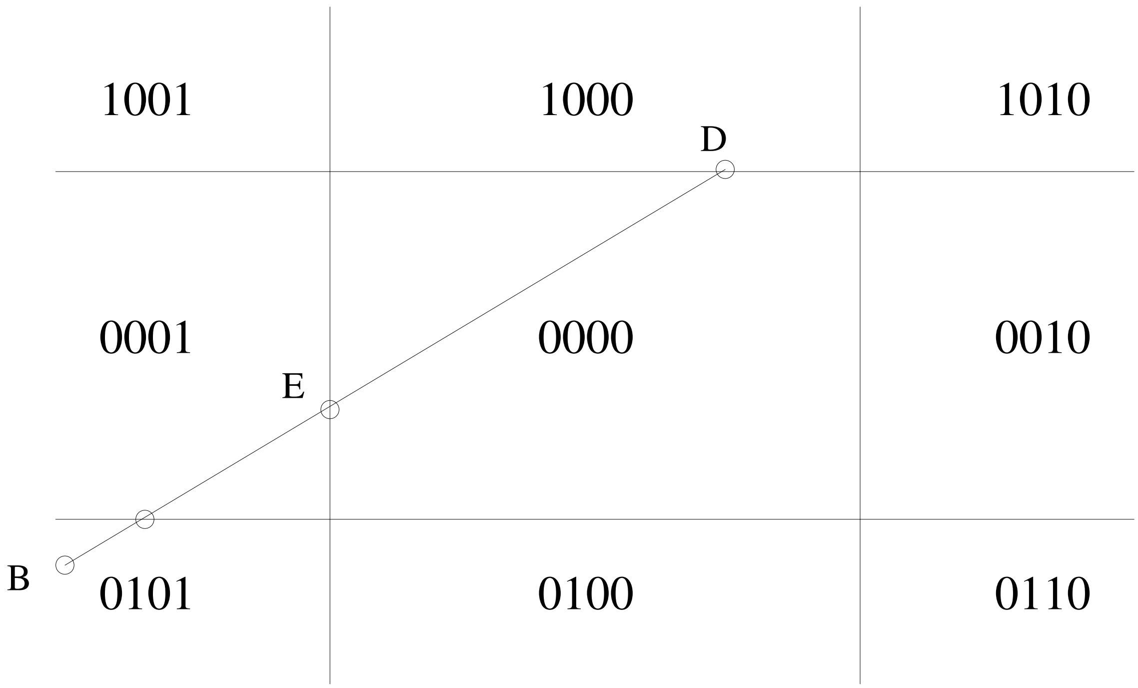
bit 4	bit 3	bit 2	bit 1
top	bottom	right	left

Use bit 4 of C (top clipping edge) to do the subdivision

Subdivide at D (need to find x coordinate of D)

$x = (y_{\text{max}} - b)/m$

Example (con't)




```
      D:   0000              D:   0000
/     B:   0101        &     B:   0101
      ----------             ----------
           0101                   0000
```

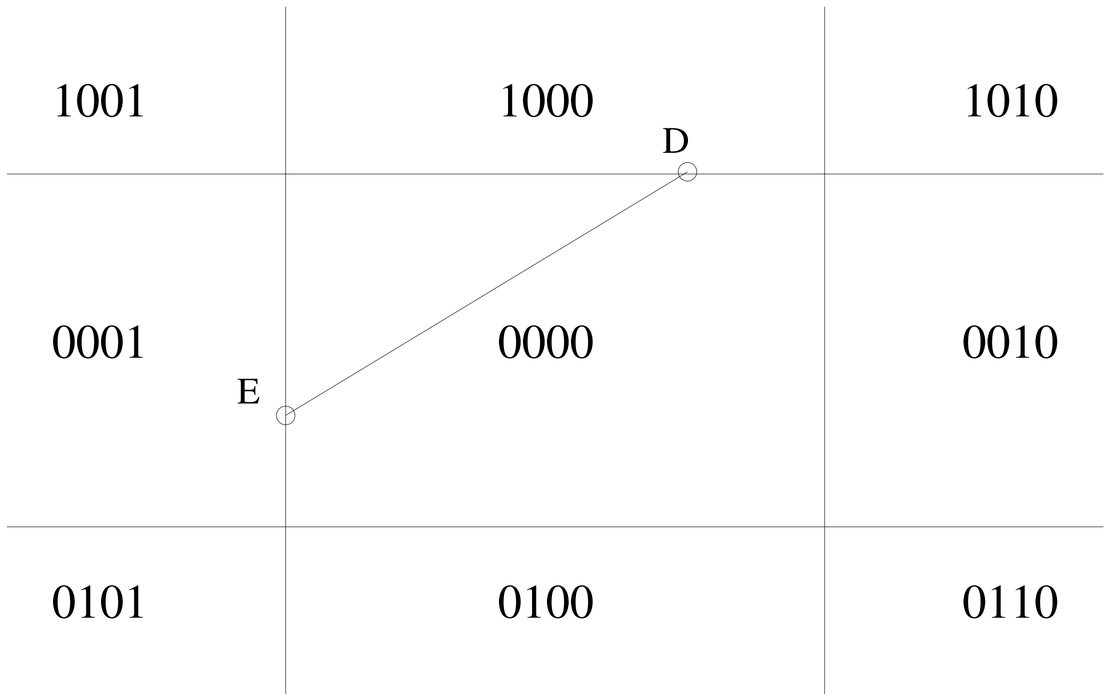
bit 4	bit 3	bit 2	bit 1
top	bottom	right	left

Use bit 1 of B (left clipping edge) to do the subdivision

Subdivide at E (need to find y coordinate of E)

$$y = m \cdot x_{\min} + b$$

Example (con't)



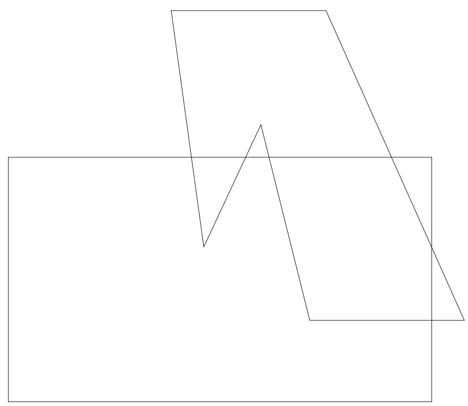
	D:	0000
/	E:	0000
		0000

bit 4	bit 3	bit 2	bit 1
top	bottom	right	left

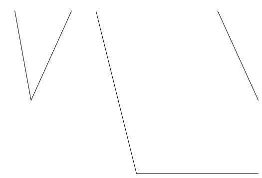
Segment ED is trivially accepted

Polygon clipping:

- Can not simply use a line clipper since it may generate a series of unconnected line segments

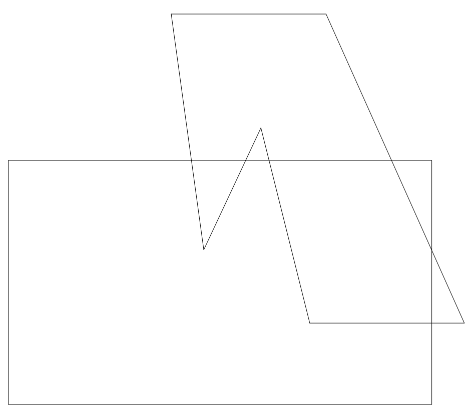


Before clipping

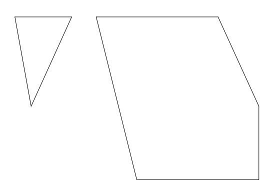


After clipping

- A polygon clipper should generate one or more closed areas



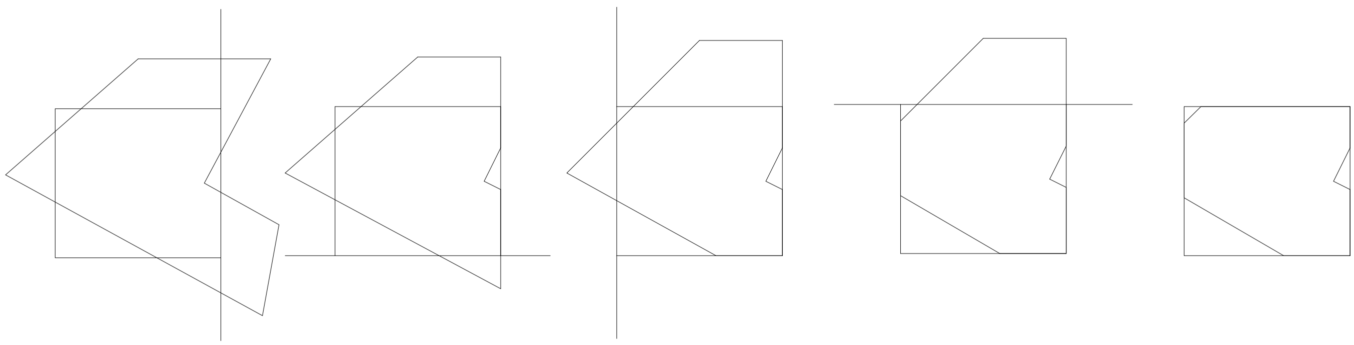
Before clipping



After clipping

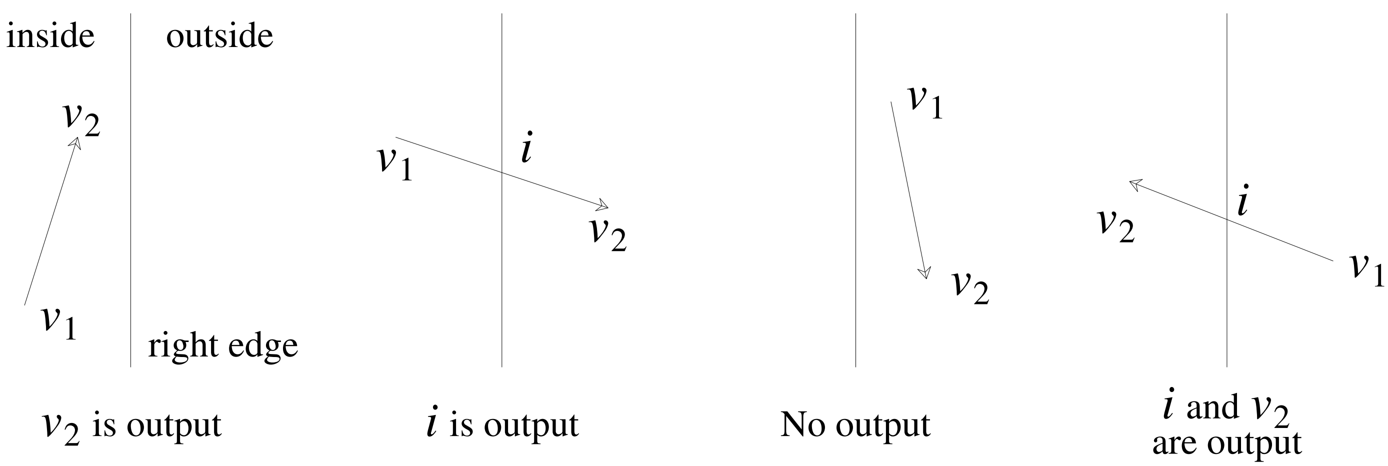
Sutherland-Hodgman Algorithm

- Polygon boundary is clipped as a whole against the four edges of the window separately



Original Polygon Clip right Clip bottom Clip left Clip top

- For each bounding edge of the window, traverse (directed) edges of the polygon and output vertices according to the following rules:



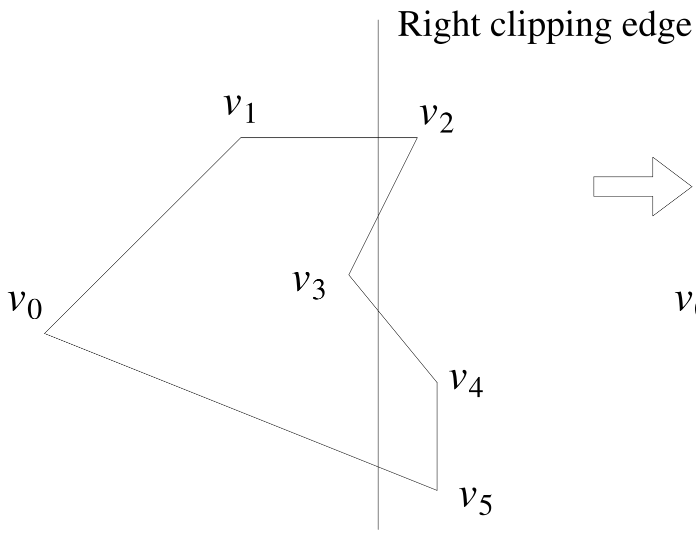
v_2 is output (a)

i is output (b)

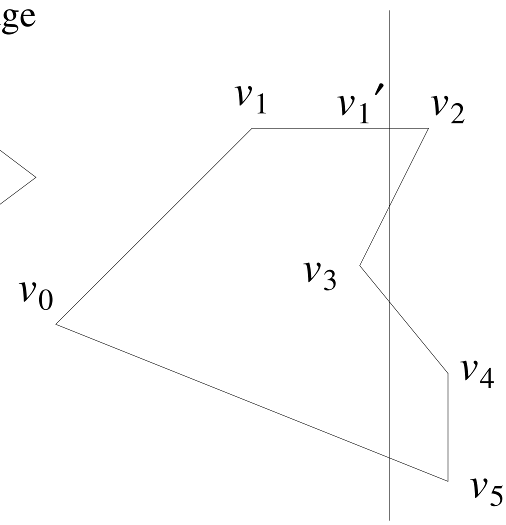
No output (c)

i and v_2 are output (d)

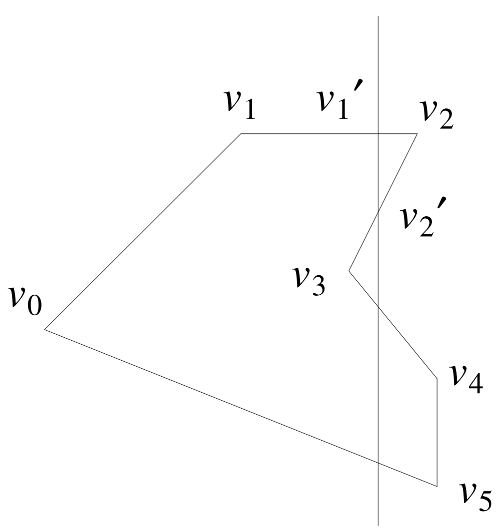
An Example
(clipping against the right edge of the window)



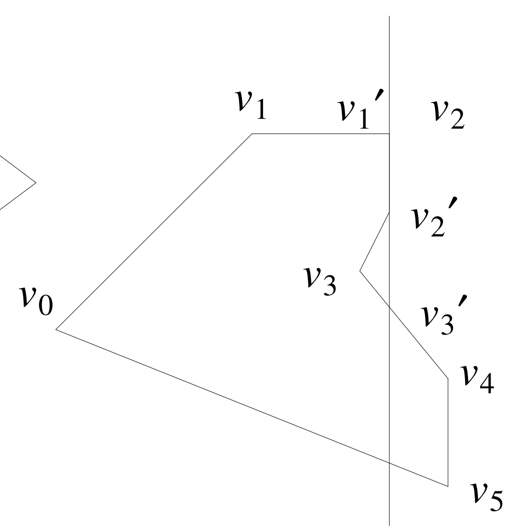
Start with $v_0 v_1$
Output: v_1



Process $v_1 v_2$
Output: $v_1 v_1'$

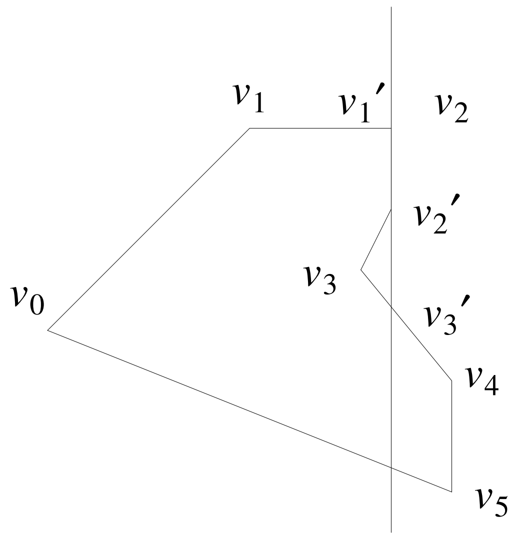


Process $v_2 v_3$
Output: $v_1 v_1' v_2' v_3$

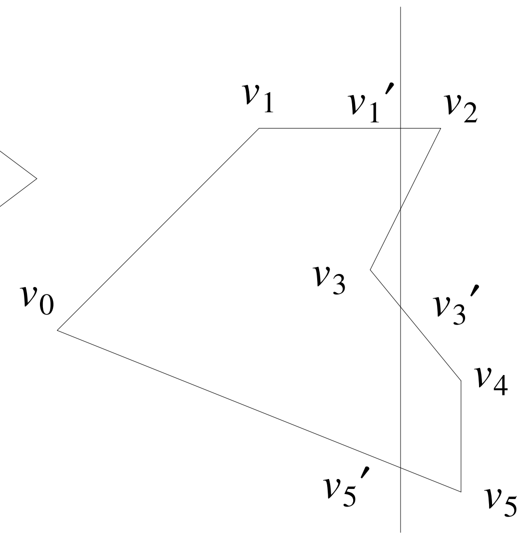


Process $v_3 v_4$
Output: $v_1 v_1' v_2' v_3 v_3'$

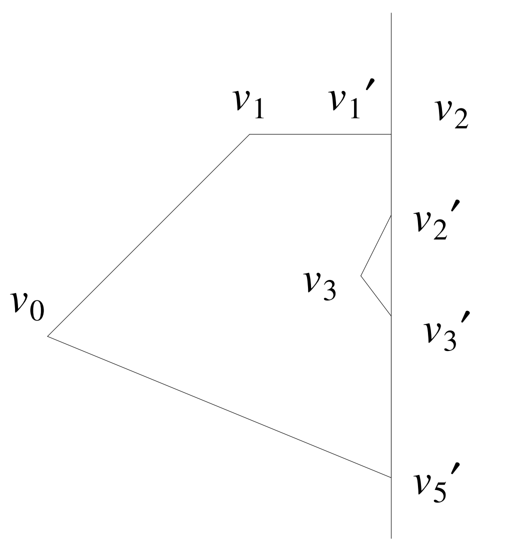
Example (con't)



Process $v_4 v_5$
Output: $v_1 v_1' v_2' v_3 v_3'$



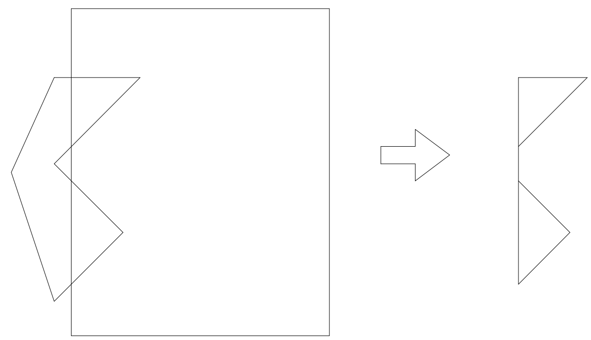
Process $v_5 v_0$
Output: $v_1 v_1' v_2' v_3 v_3' v_5' v_0$



Result: $v_1 v_1' v_2' v_3 v_3' v_5' v_0$

Disadvantage of S-H algorithm:

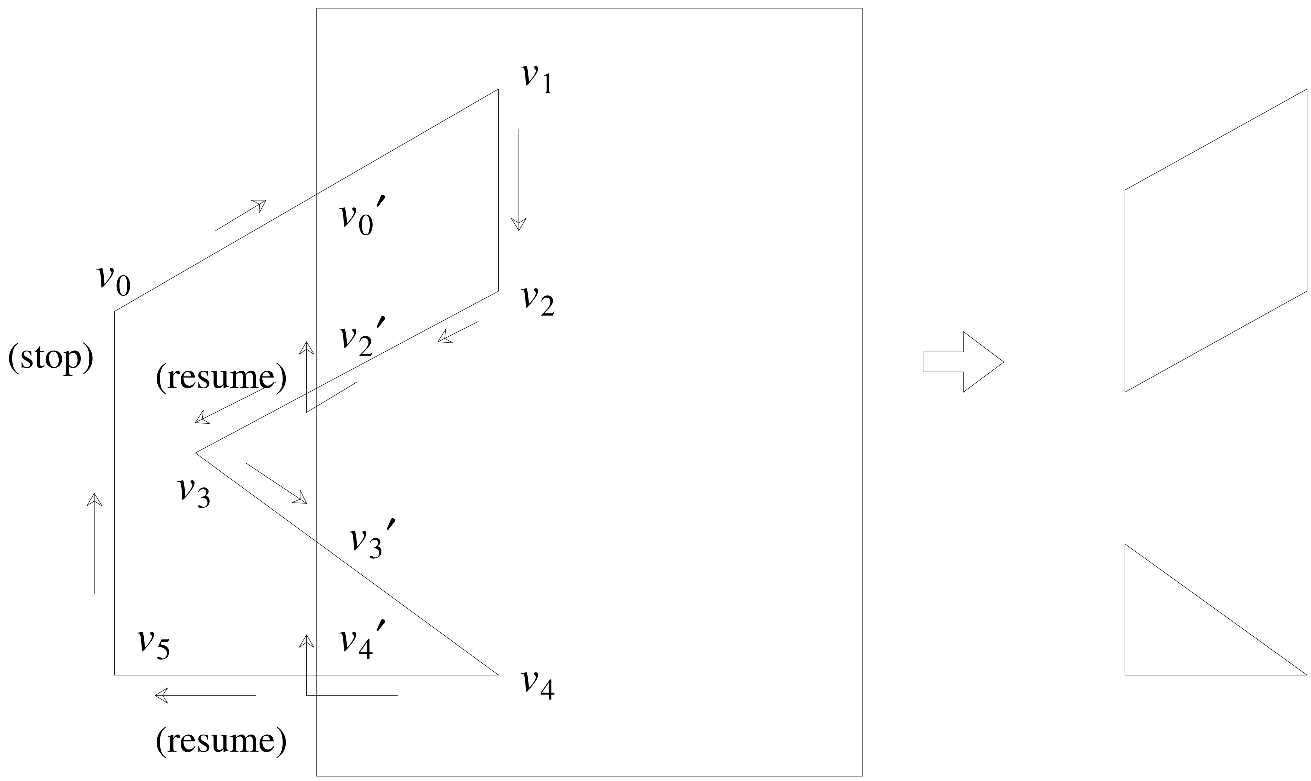
- Output is always a connected area



Remedy: using Weiler-Atherton’s approach

For clockwise processing of polygon vertices in S-H clipping algorithm:

- For an outside-to-inside pair of vertices, follow the polygon boundary
- For an inside-to-outside pair of vertices, follow the window coundary in a clockwise direction


v_1
v_0'
v_0
v_2
v_2'
(stop)
(resume)
v_3
v_3'
v_5
v_4'
v_4
(resume)

<u>Text clipping:</u>

Usually clip the bounding box (rectangle) of an individual character or the entire string

- All-or-none character-clipping

Before clipping After clipping

- All-or-none string-clipping

Before clipping After clipping