# Introduction to Computer Graphics
## 5. Clipping

I-Chen Lin
National Chiao Tung University

# Objectives

▶ Introduce basic implementation strategies

▶ 2D Clipping
  ▶ Lines
  ▶ Polygons

▶ Clipping in 3D

# Clipping

盡量避免float計算

**判斷是否在frame buffer裡** : floating point->if/else
->bitwise operation(AND, OR)與prediction效率有關

▶ 2D against clipping window

▶ 3D against clipping volume

▶ Easy for line segments polygons 線與框框的交點

▶ Hard for curves and text

    ▶ Convert to <u>lines</u> or <u>polygons</u> first



frame buffer

多餘的東西不該寫到記憶體裡

# Clipping 2D Line Segments

▶ Brute force approach: compute intersections with all sides of clipping window 和每個邊做相交，一條線要做四次intersection

   ▶ Inefficient: one division per intersection 應減少float computation
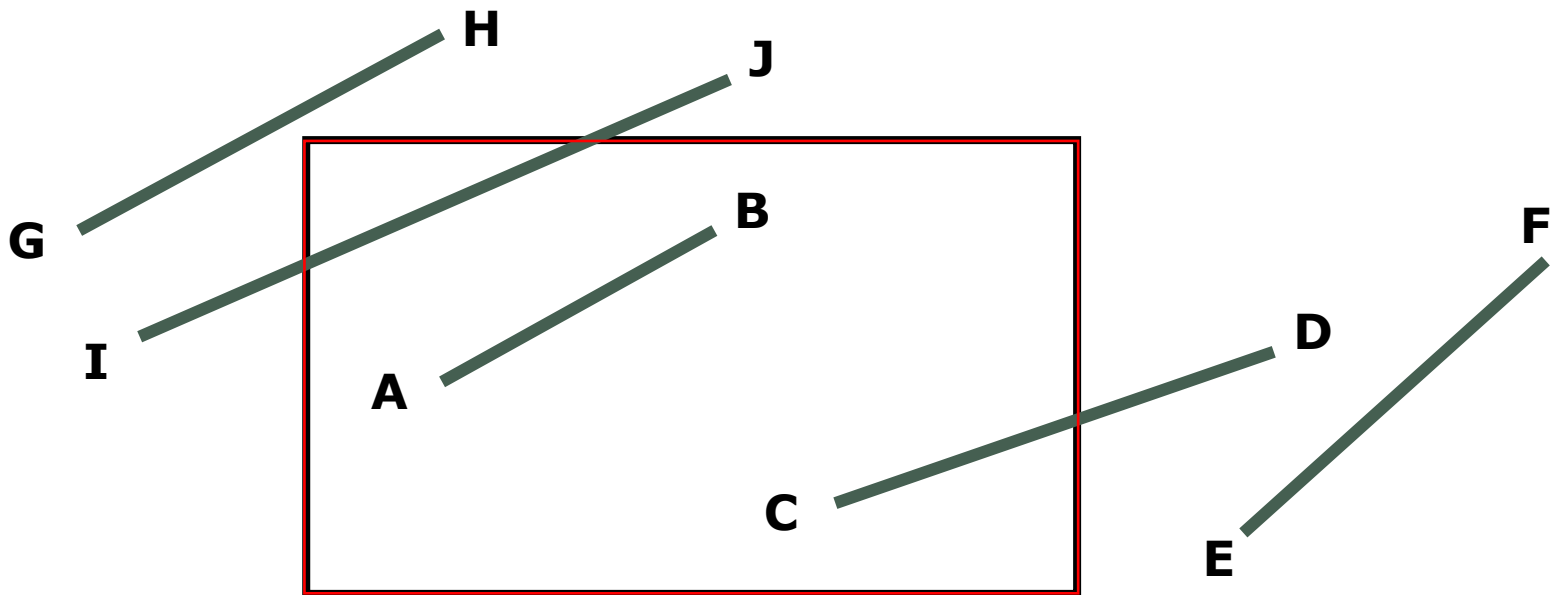
# Cohen-Sutherland Algorithm

▶ Idea: eliminate as many cases as possible without computing intersections  有些可以用if/else來避開float computation

▶ Start with four lines that determine the sides of the clipping window
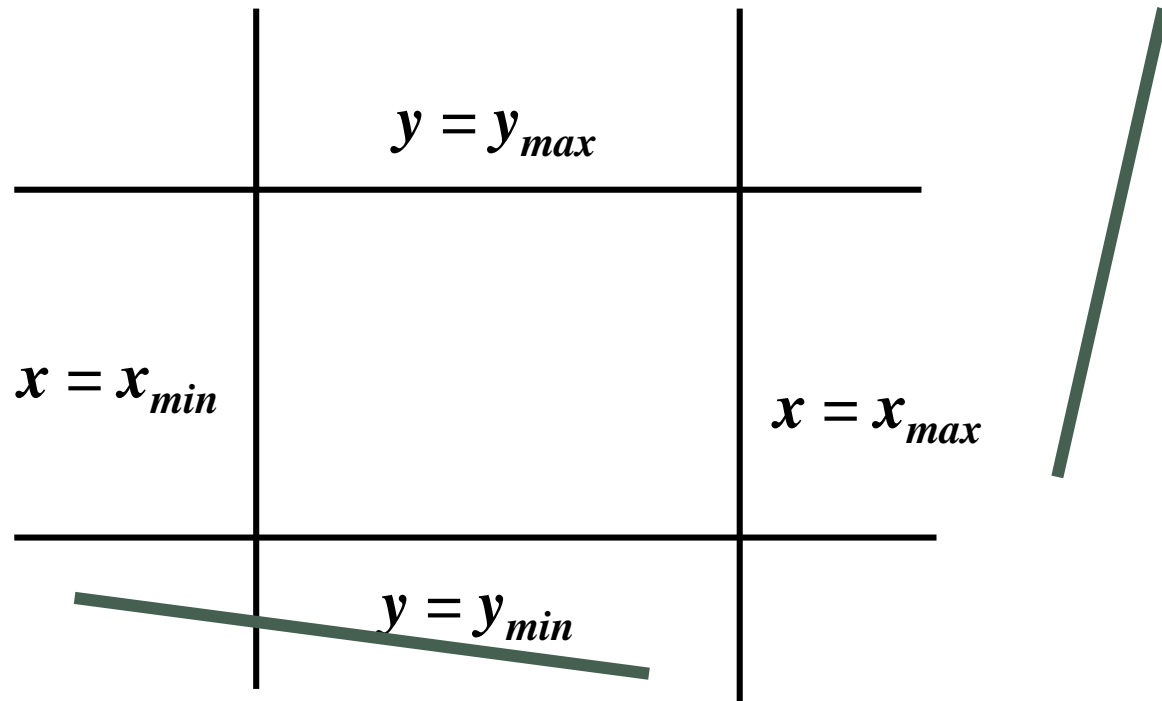
$$y = y_{max}$$

$$x = x_{min}$$

$$x = x_{max}$$

$$y = y_{min}$$
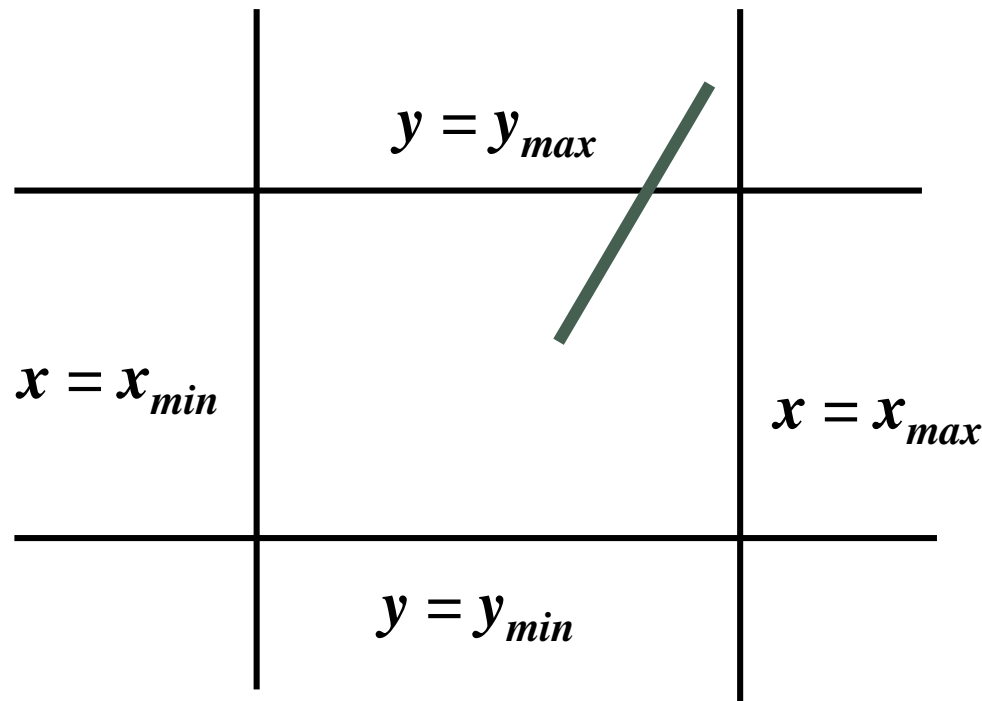
# Case 1 完全不需要做intersection

▶ Case 1: both endpoints of a line segment inside all four lines

  ▶ Draw (accept) the line segment as is

$$y = y_{max}$$

$$x = x_{min}$$
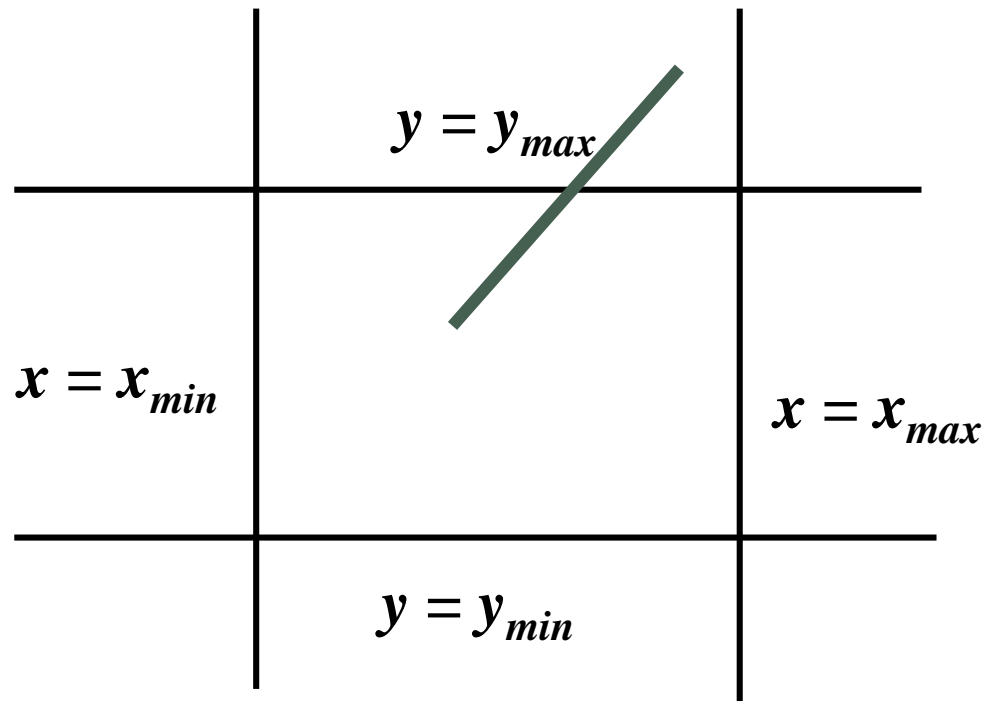
$$x = x_{max}$$

$$y = y_{min}$$
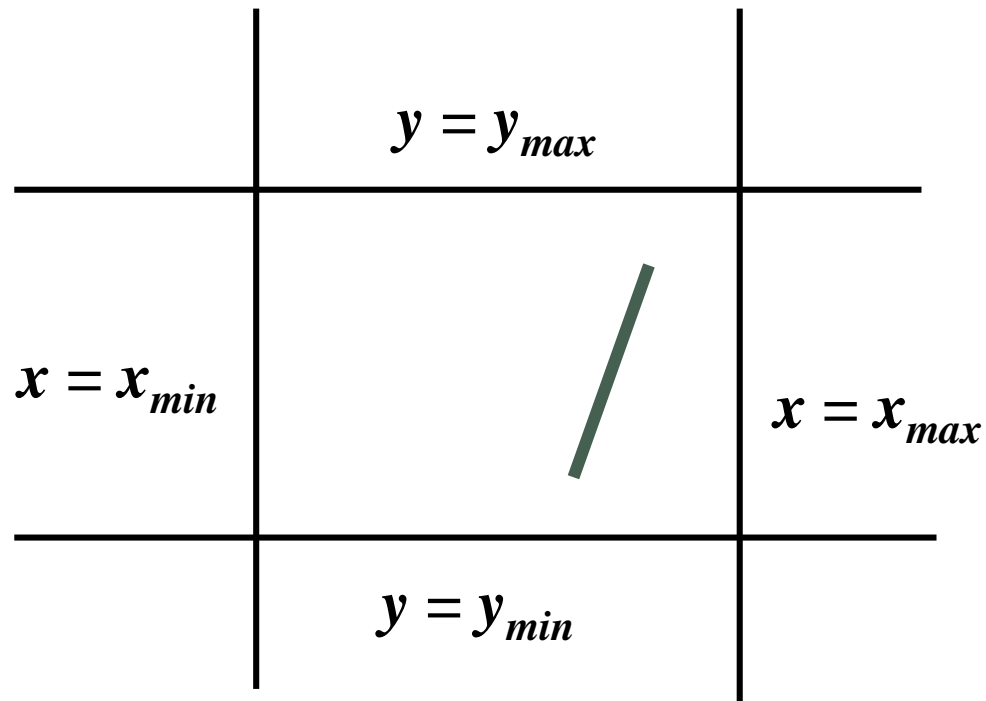
# Case 2 完全在某一個平面或某一條線外面

▶ Case 2: both endpoints outside all lines and on same side of a line

　　▶ Discard (reject) the line segment

$$y = y_{max}$$

$$x = x_{min}$$

$$x = x_{max}$$

$$y = y_{min}$$

# Case 3

對於某邊，這條線一個點在裡面一個點在外面=>算一次交點

▶ One endpoint inside, one outside

　▶ Must do at least one intersection

$$y = y_{max}$$

$$x = x_{min}$$

$$x = x_{max}$$

$$y = y_{min}$$

▶ Both outside

  ▶ May have part inside

  ▶ Must do at least one intersection

做一次intersection：
交點無意義，整條線直接丟掉

$$y = y_{max}$$

$$x = x_{min}$$

做兩次intersection

$$x = x_{max}$$

$$y = y_{min}$$
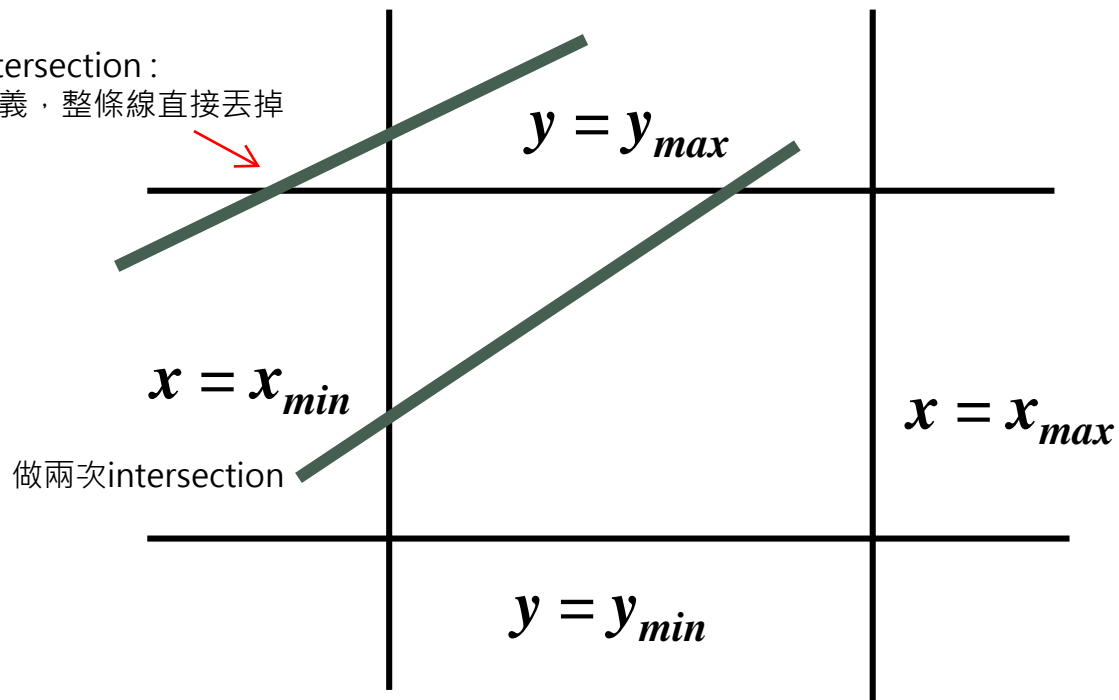
# Defining Outcodes

- ► For each endpoint, define an outcode
  - ► [b0 b1 b2 b3 ]
- ► Outcodes divide space into 9 regions
- ► Computation of outcode requires at most 4 subtractions

| 1001 | 1000 | 1010 |
|------|------|------|
| 0001 | 0000 | 0010 |
| 0101 | 0100 | 0110 |

b0 = 1 if y > ymax, 0 otherwise

b1 = 1 if y < ymin, 0 otherwise

b2 = 1 if x > xmax, 0 otherwise

b3 = 1 if x < xmin, 0 otherwise

# Using Outcodes <small>bitwise operation</small>

兩個點進來會自帶outcode

▶ AB: <u>outcode(A) = outcode(B) = 0</u>

    ▶ Accept the line segment
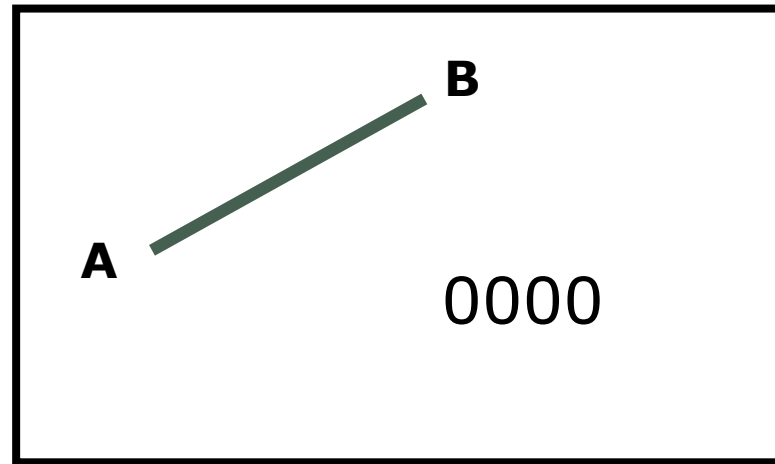
# Using Outcodes

▶ CD: <u>outcode (C) = 0, outcode(D) ≠ 0</u>

   ▶ Compute <u>intersection</u>

   ▶ Location of 1 in outcode(D) determines which edge to intersect with

   ▶ Note <u>if there were a segment from C to a point in a region with 2 ones in outcode, we might have to do two intersections</u>

1010

0000          D

              0010

   C

0110
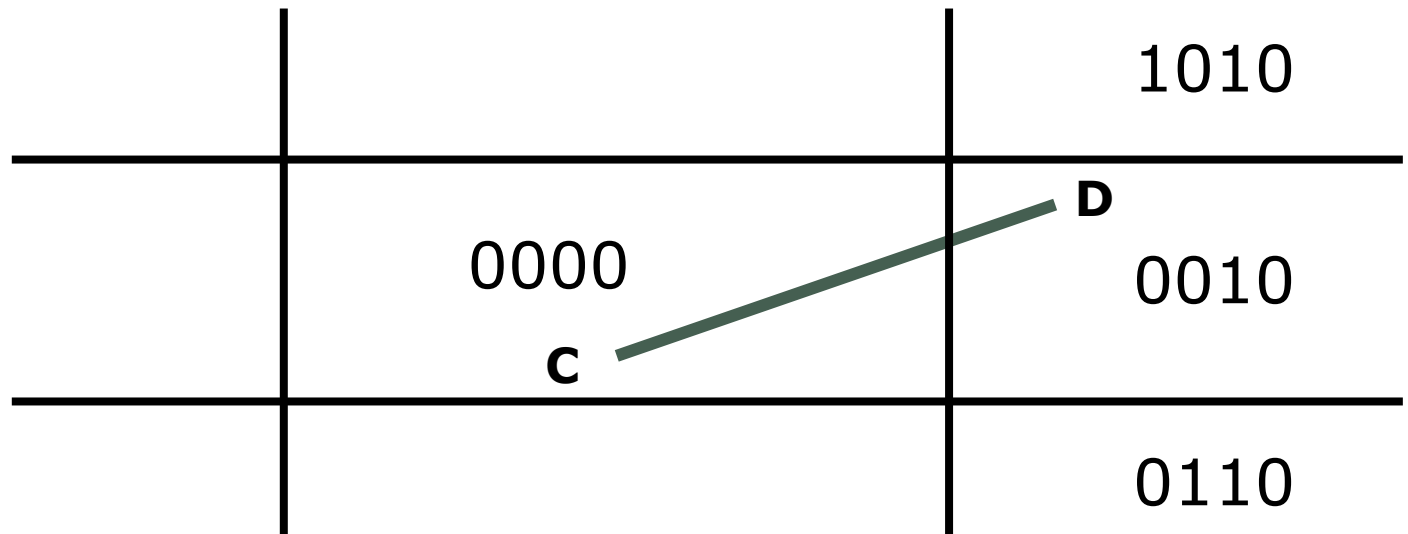
# Using Outcodes

▶ EF: outcode(E) logically ANDed with outcode(F) (bitwise) ≠ 0

  ▶ Both outcodes have a 1 bit in the same place

  ▶ The line segment is outside of corresponding side of clipping window  會留下某一個數值(AND起來是0010) : 剛好對某一條線會全部在外面(不是0000)
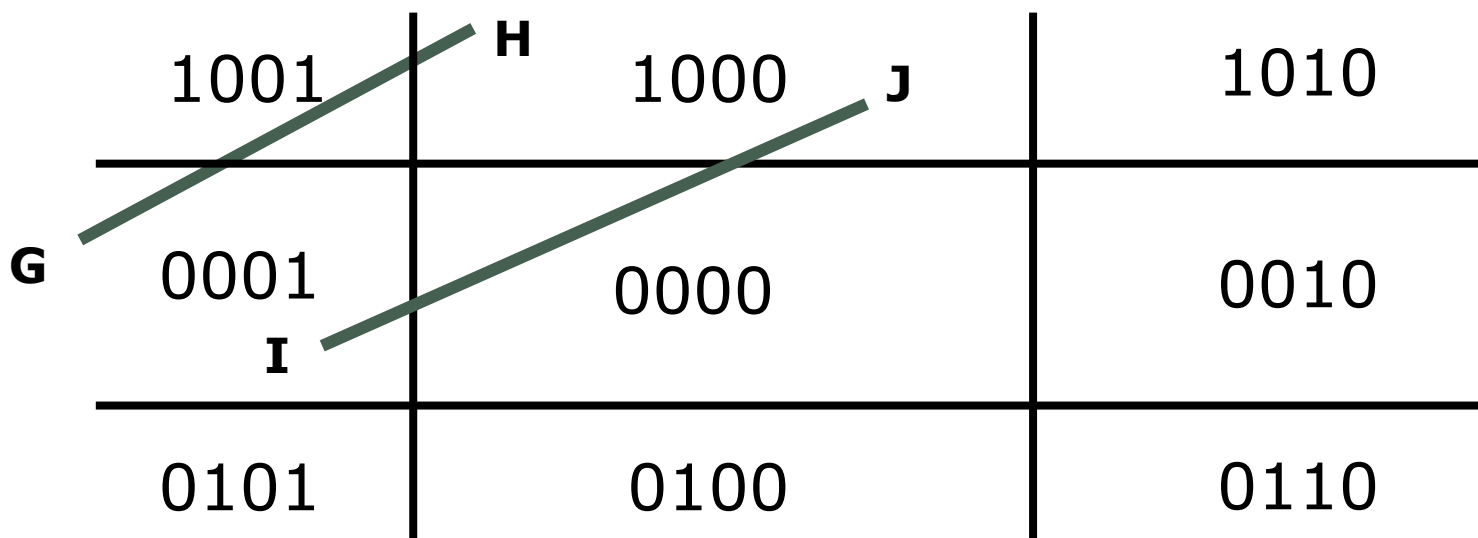
  ▶ reject
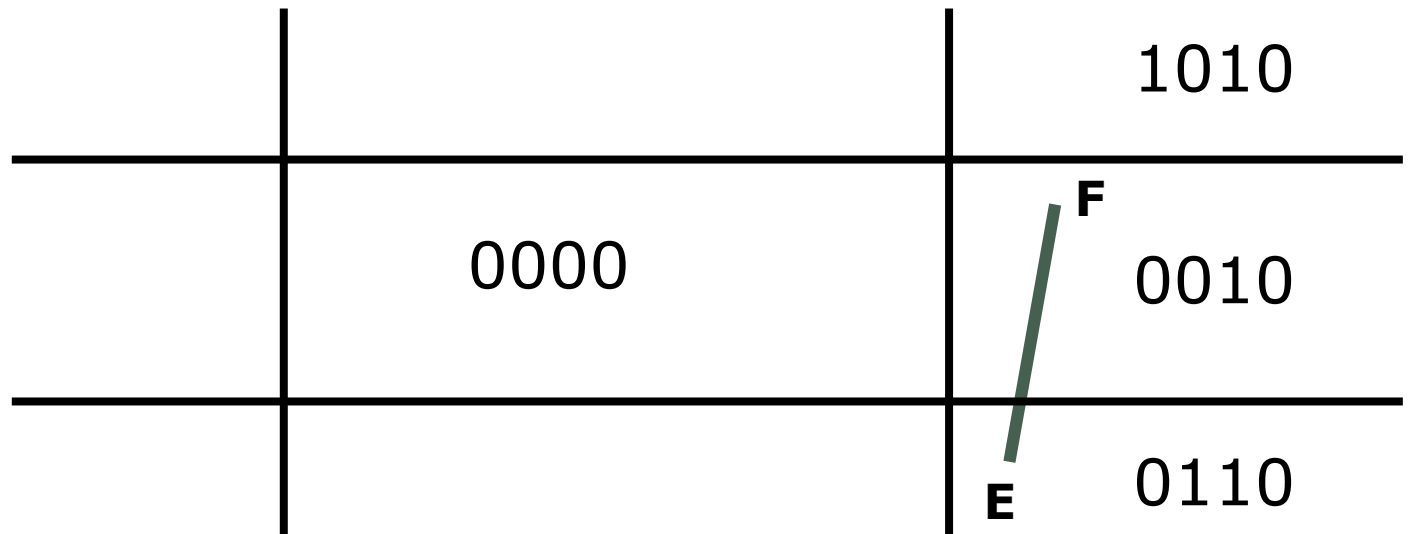
# Using Outcodes

檢查其中一個交點來判斷是哪一種

▶ GH and IJ: same outcodes, neither zero but logical AND yields zero 雖然AND之後是0000，但兩條都不是0000

   ▶ Shorten line segment by intersecting with one of sides of window

   ▶ Compute outcode of intersection (newendpoint of shortened line segment)

   ▶ Re-execute algorithm

| 1001 | **H** 1000 | **J** 1010 |
|------|------|------|
| **G** 0001 | 0000 | 0010 |
| **I** 0101 | 0100 | 0110 |

# Efficiency

▶ In many applications, the clipping window is small relative to the size of the entire data base

  ▶ Most line segments can be eliminated based on their outcodes.

▶ Inefficiency when code has to be re-executed for line segments that must be shortened in more than one step

# Polygon Clipping

▶ Not as simple as line segment clipping

 ▶ Clipping a line segment yields at most one line segment

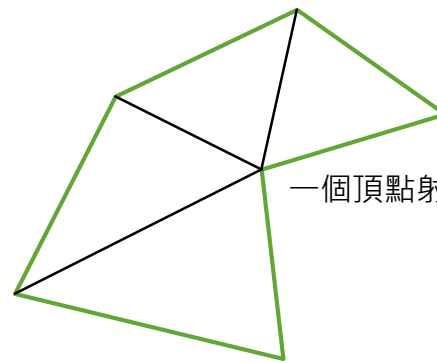 ▶ Clipping a polygon can yield multiple polygons

concave : 凹多邊形
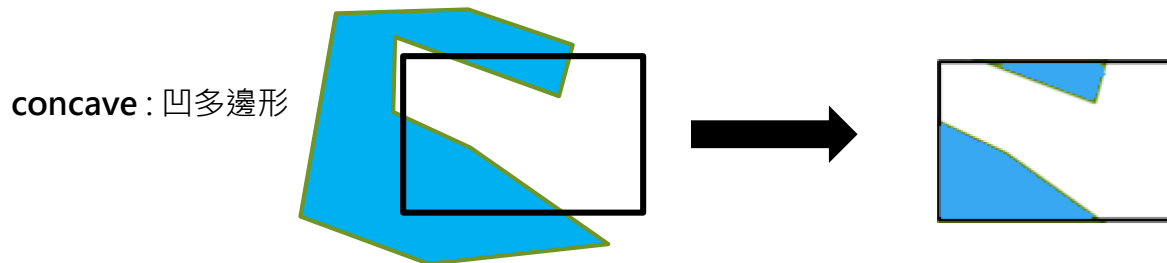
▶ However, clipping a convex polygon can yield at most one other polygon

# Tessellation and Convexity

凹邊形要先打斷

▶ One strategy is to replace nonconvex (concave) polygons with a set of triangular polygons (a tessellation)

▶ Also makes fill easier
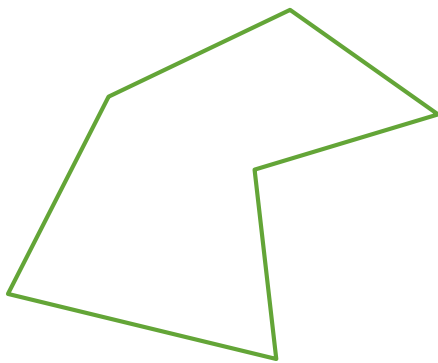
一個頂點射線出去分割凹多邊形

# Clipping as a Black Box pipeline

▶ Consider line segment clipping as a process that takes in two vertices and produces either no vertices or the vertices of a clipped line segment.

# Pipeline Clipping of Line Segments
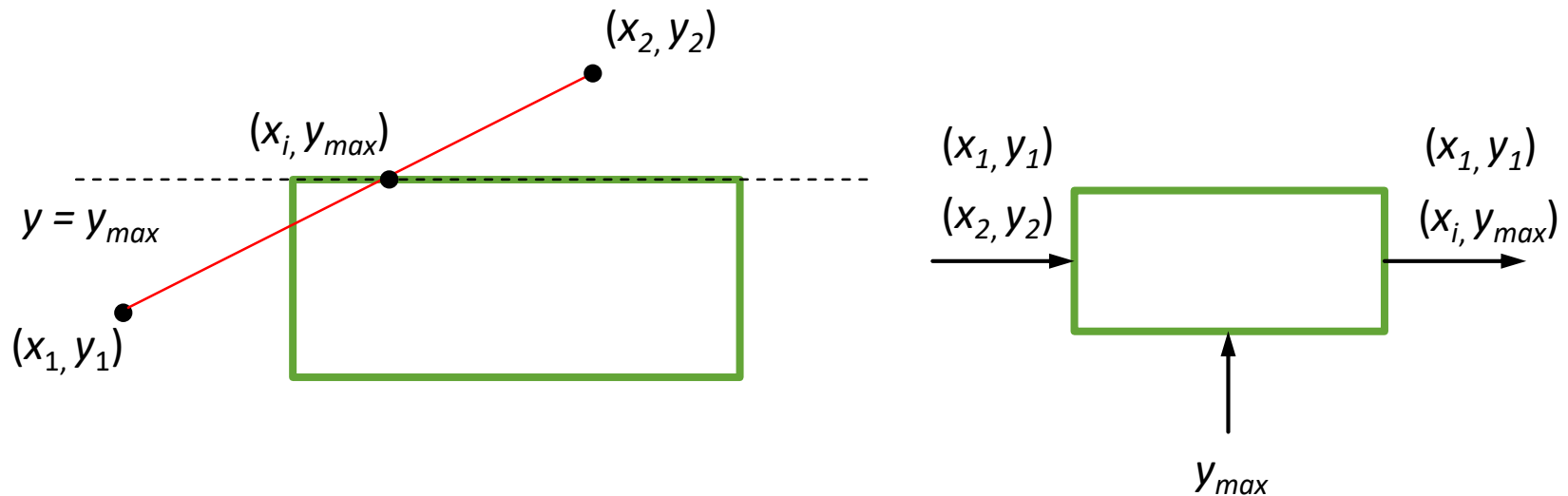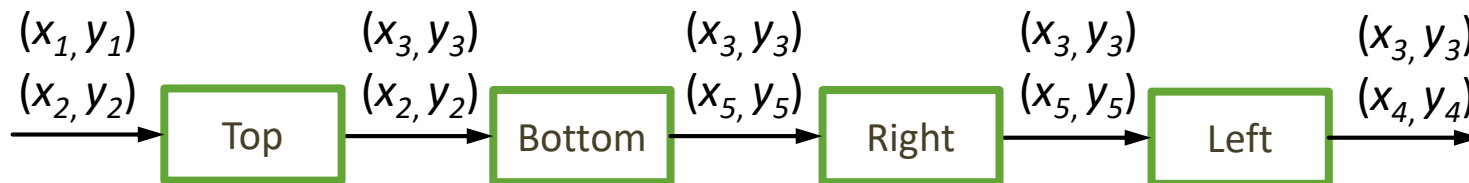
▶ Clipping against each side of window is independent of other sides

   ▶ Can use four independent clippers in a pipeline 分成四個stage，在每一階段砍一刀
   可能會做多餘的事情，在數量多的時候還算有效率

$(x_1, y_1)$

$(x_3, y_3)$

$(x_4, y_4)$

$(x_5, y_5)$

$(x_2, y_2)$

$(x_1, y_1)$        $(x_3, y_3)$        $(x_3, y_3)$        $(x_3, y_3)$        $(x_3, y_3)$
$(x_2, y_2)$        $(x_2, y_2)$        $(x_5, y_5)$        $(x_5, y_5)$        $(x_4, y_4)$

→ | Top | → | Bottom | → | Right | → | Left | →
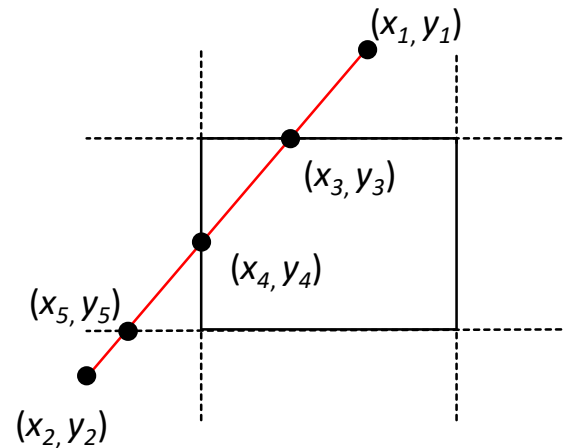
# Pipeline Clipping of Polygons

- ▶ Sutherland-Hodgman algorithm
- ▶ Strategy used in SGI Geometry Engine
- ▶ Small increase in latency
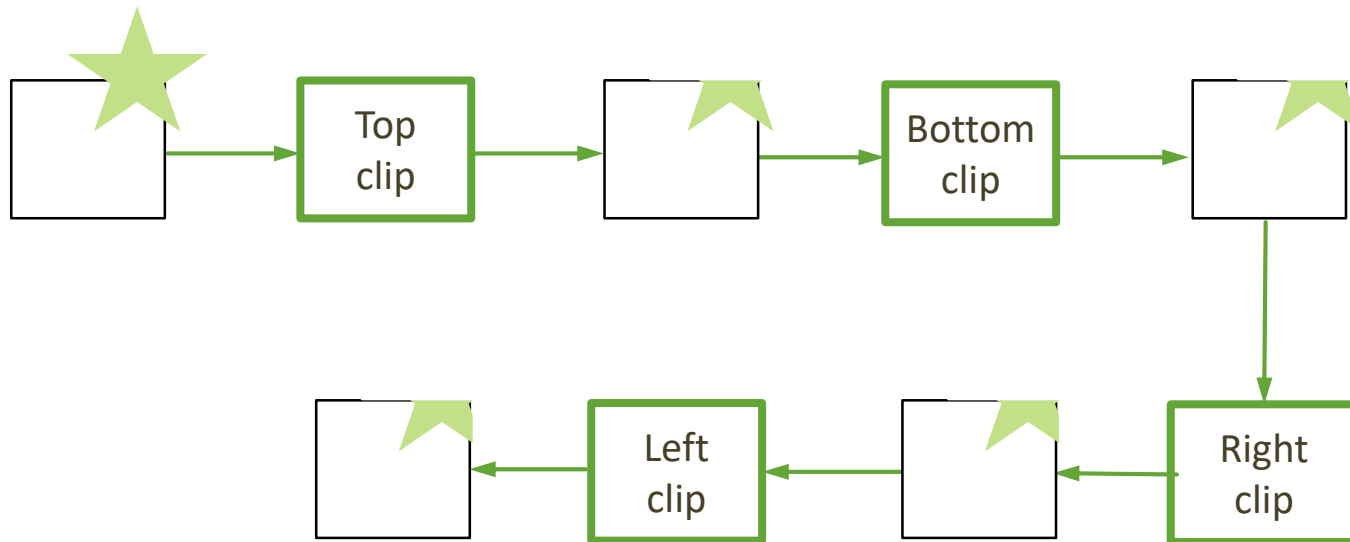
# Cohen-Sutherland Method in 3D

► Use 6-bit outcodes

   ► When needed, clip line segments against planes
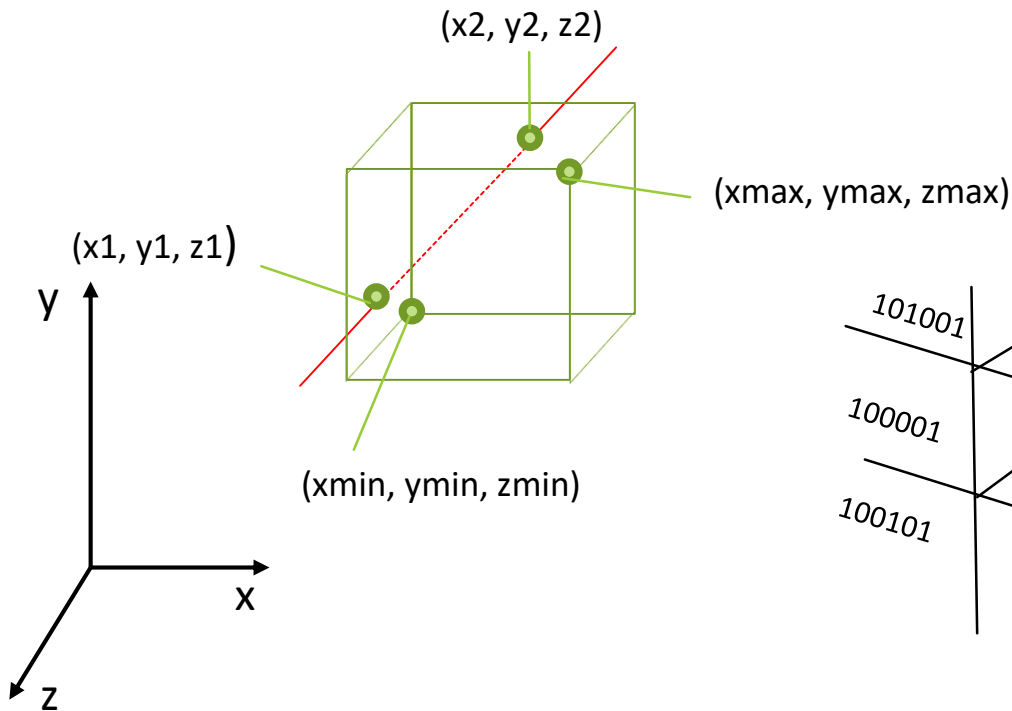
# Cohen-Sutherland Method in 3D  clipping做在normalized space

Check for outcodes:

$$-1 \le x_p \le 1, \ -1 \le y_p \le 1, \ -1 \le z_p \le 1$$

Since

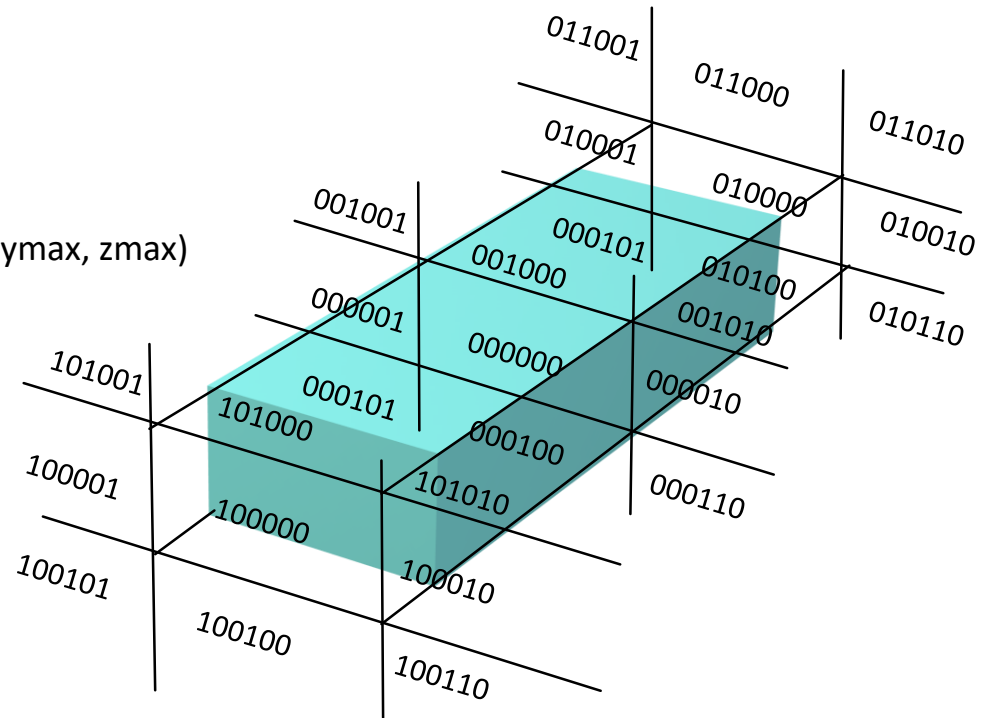$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \Rightarrow \cdots \Rightarrow \begin{bmatrix} x_h \\ y_h \\ z_h \\ h \end{bmatrix} \Rightarrow \cdots \Rightarrow \begin{bmatrix} x_h/h \\ y_h/h \\ z_h/h \\ 1 \end{bmatrix} = \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix}$$

SRT...     Divide h

To avoid unnecessary float division, We can check

$$-h \le x_h \le h, \ -h \le y_h \le h, \ -h \le z_h \le h$$

# Cohen-Sutherland Method in 3D

▶ If outcode(A)==outcode(B)==0

    ▶ Accept the whole line segment.

▶ If(outcode(A) and outcode(B))!=0

    ▶ Reject the line segment.

▶ Other cases

    ▶ Calculate an intersection (according to outcode bits)

    ▶ Then check outcode again

▶ Note: use <u>parametric forms</u>
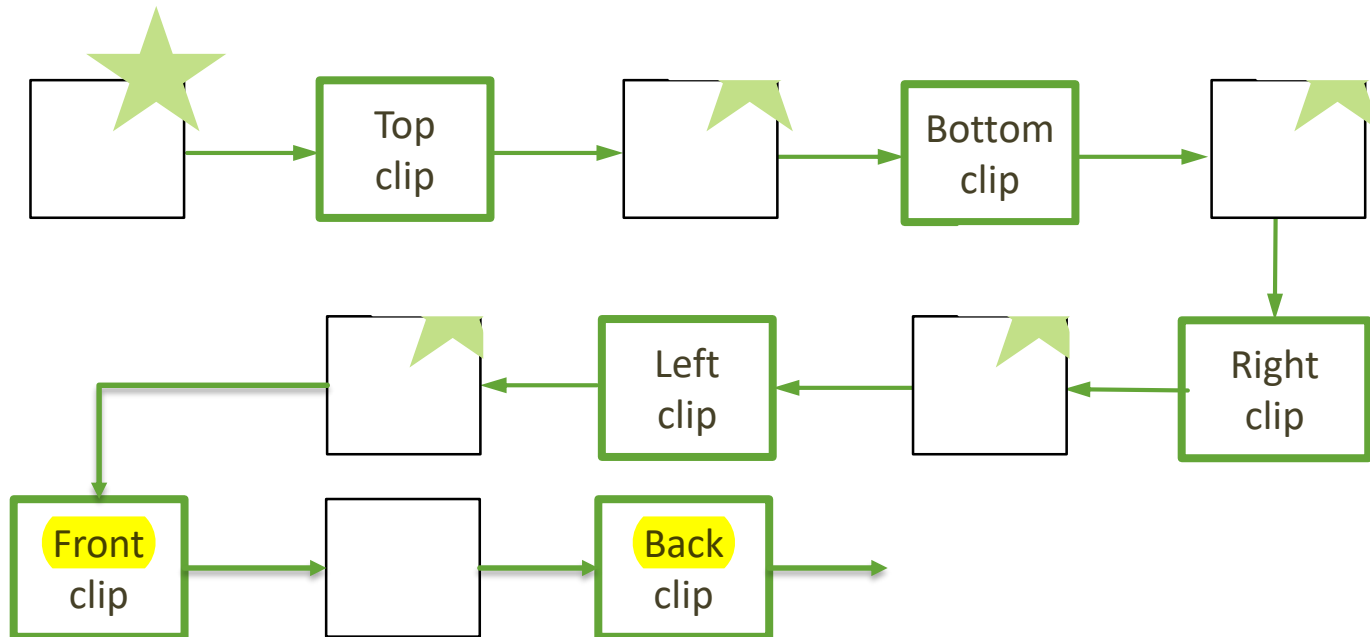
$$x_h = x_{ha} + (x_{hb} - x_{ha})u$$

$$y_h = y_{ha} + (y_{hb} - y_{ha})u$$

$$z_h = z_{ha} + (z_{hb} - z_{ha})u$$

$$h = h_a + (h_b - h_a)u$$

# Polygon Clipping in 3D

▶ Similar to 2D clipping

  ▶ Bounding box

  ▶ Clipping with each clipping plane

  ▶ Etc…..

```
┌──────┐      ┌──────┐      ┌──────┐      ┌──────┐      ┌──────┐
│      │ ───▶ │ Top  │ ───▶ │      │ ───▶ │Bottom│ ───▶ │      │
│      │      │ clip │      │      │      │ clip │      │      │
└──────┘      └──────┘      └──────┘      └──────┘      └──────┘
                                                             │
                                                             ▼
┌──────┐      ┌──────┐      ┌──────┐      ┌──────┐      ┌──────┐
│      │ ◀─── │ Left │ ◀─── │      │ ◀─── │Right │
│      │      │ clip │      │      │      │ clip │
└──────┘      └──────┘      └──────┘      └──────┘
    │
    ▼
┌──────┐      ┌──────┐      ┌──────┐
│Front │ ───▶ │      │ ───▶ │ Back │ ───▶
│ clip │      │      │      │ clip │
└──────┘      └──────┘      └──────┘
```
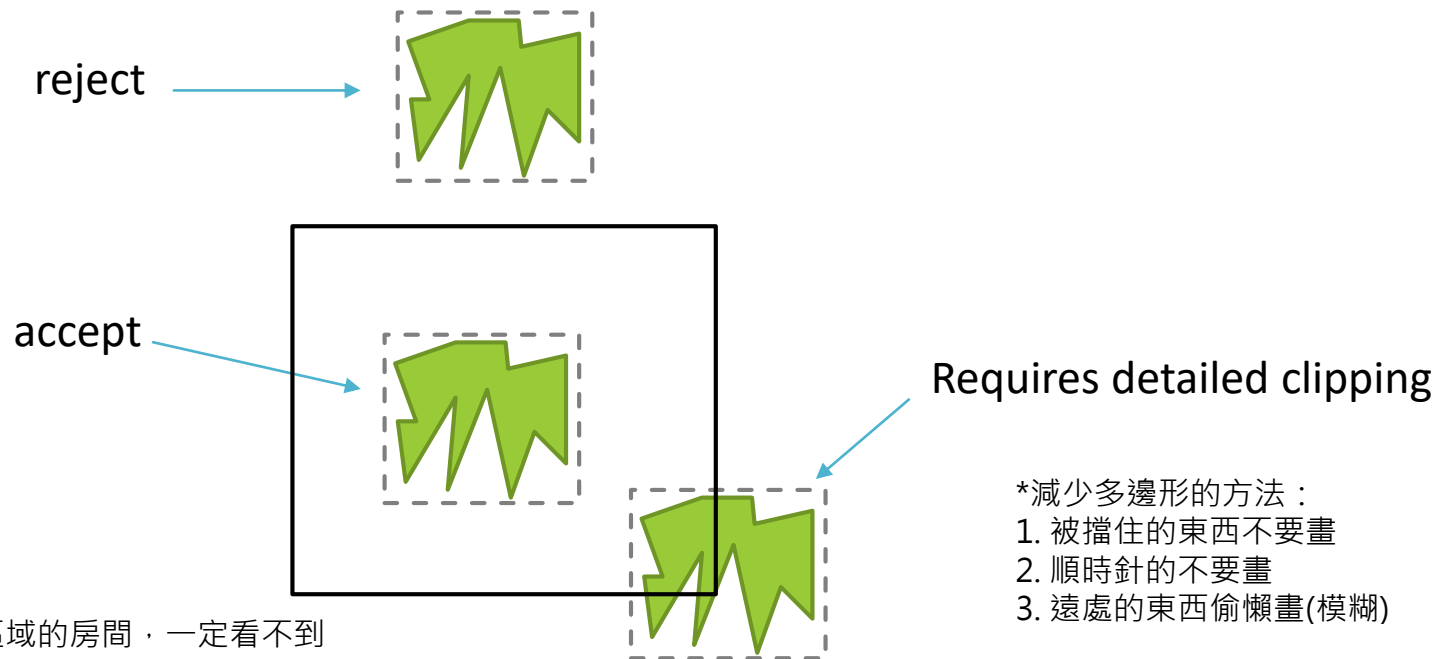
# Bounding Boxes

*把物件用一個多邊形框住(不會真正畫出來)，兩個立體去做intersection
=>可以直接用min/max踢掉frame box以外的物件

▶ Rather than doing clipping on a complex polygon, we can use an <mark>axis-aligned</mark> bounding box or extent

  ▶ Smallest rectangle aligned with axes that encloses the polygon

  ▶ Simple to compute: max and min of x and y

reject

accept

Requires detailed clipping

*減少多邊形的方法：
1. 被擋住的東西不要畫
2. 順時針的不要畫
3. 遠處的東西偷懶畫(模糊)

*地下城->不渲染不同區域的房間，一定看不到