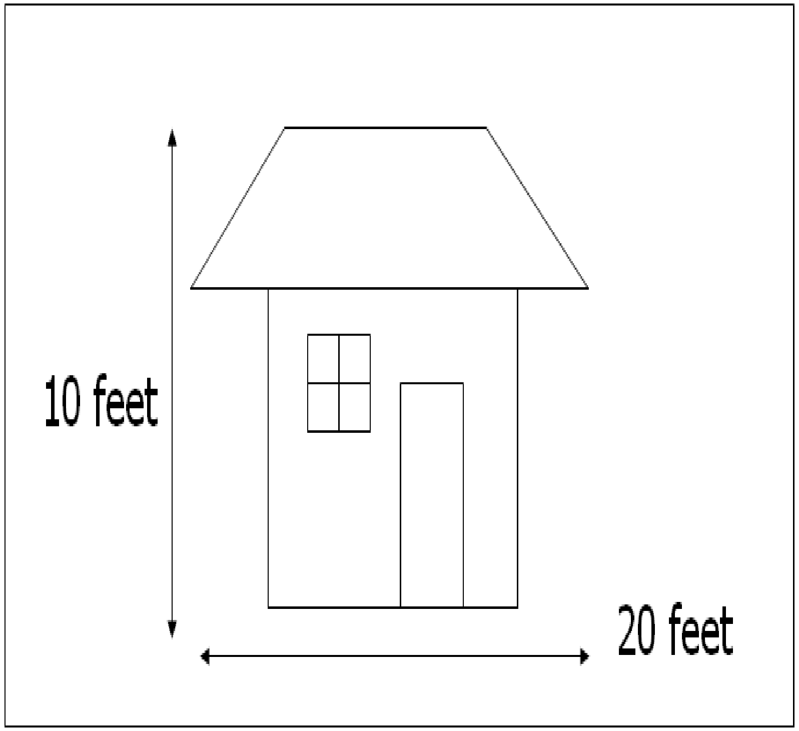
Computer Graphics Viewing

Objective

- The aim of this lesson is to make the student aware of the following concepts:
  - Window
  - Viewport
  - Window to Viewport Transformation
  - line clipping
  - Polygon Clipping

Introduction

- Every 2D graphics usually use **device coordinates**.
- If any graphics primitive lies partially or completely outside the window then the portion outside will not be drawn. It is **<u>clipped out of the image</u>**.
- In many situations we have to draw objects whose dimensions are given in natural coordinates (**World coordinates WC**).

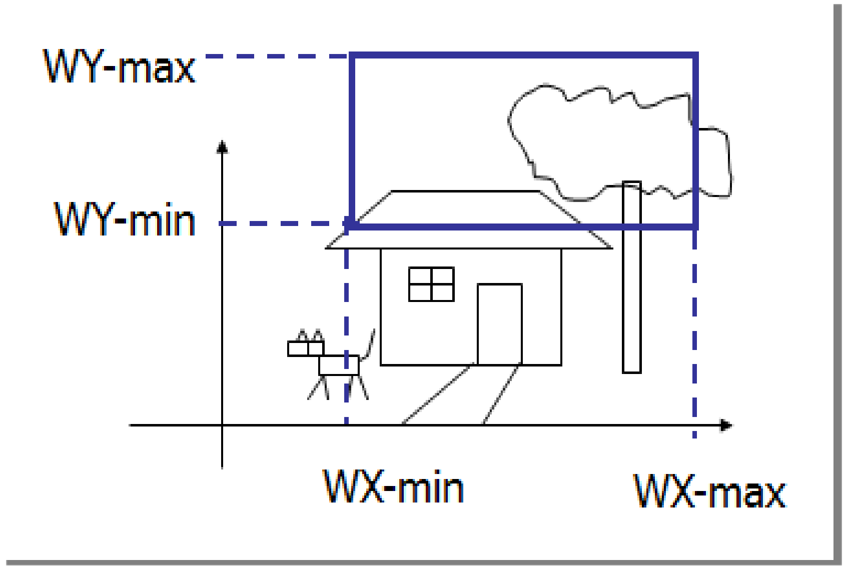


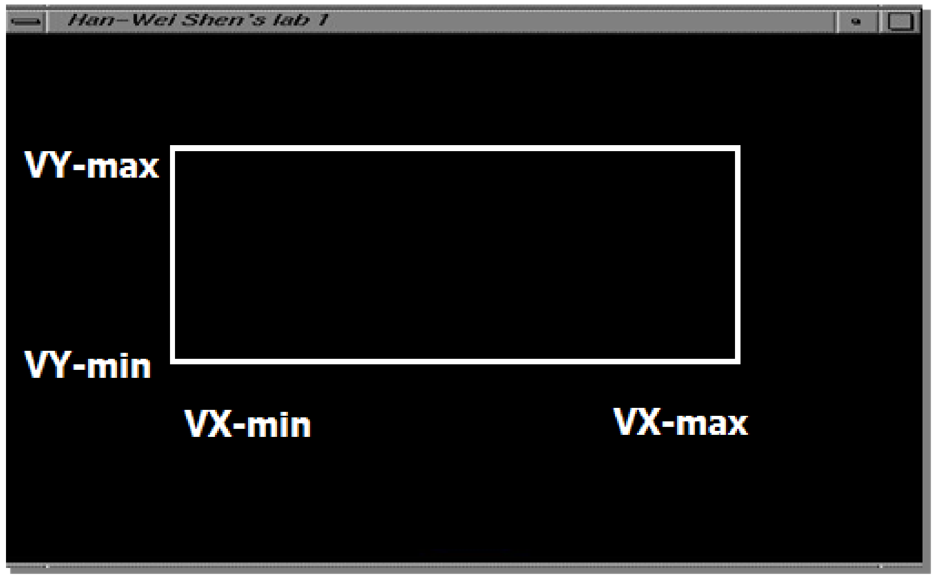
Viewing Transformation

- The **Viewing Transformation:**
  - is the process of going form a window in World coordinates to viewport in **Physical Device Coordinates (PDC).**
- Programming are required to scale from the **WC** to **device coordinates (DC)**.
- The transformation form the **WC** to **DC** is often carried out in tow steps:
  - using the **Normalisation Transformation**
  - the **Workstation Transformation**.

World Window & View Port

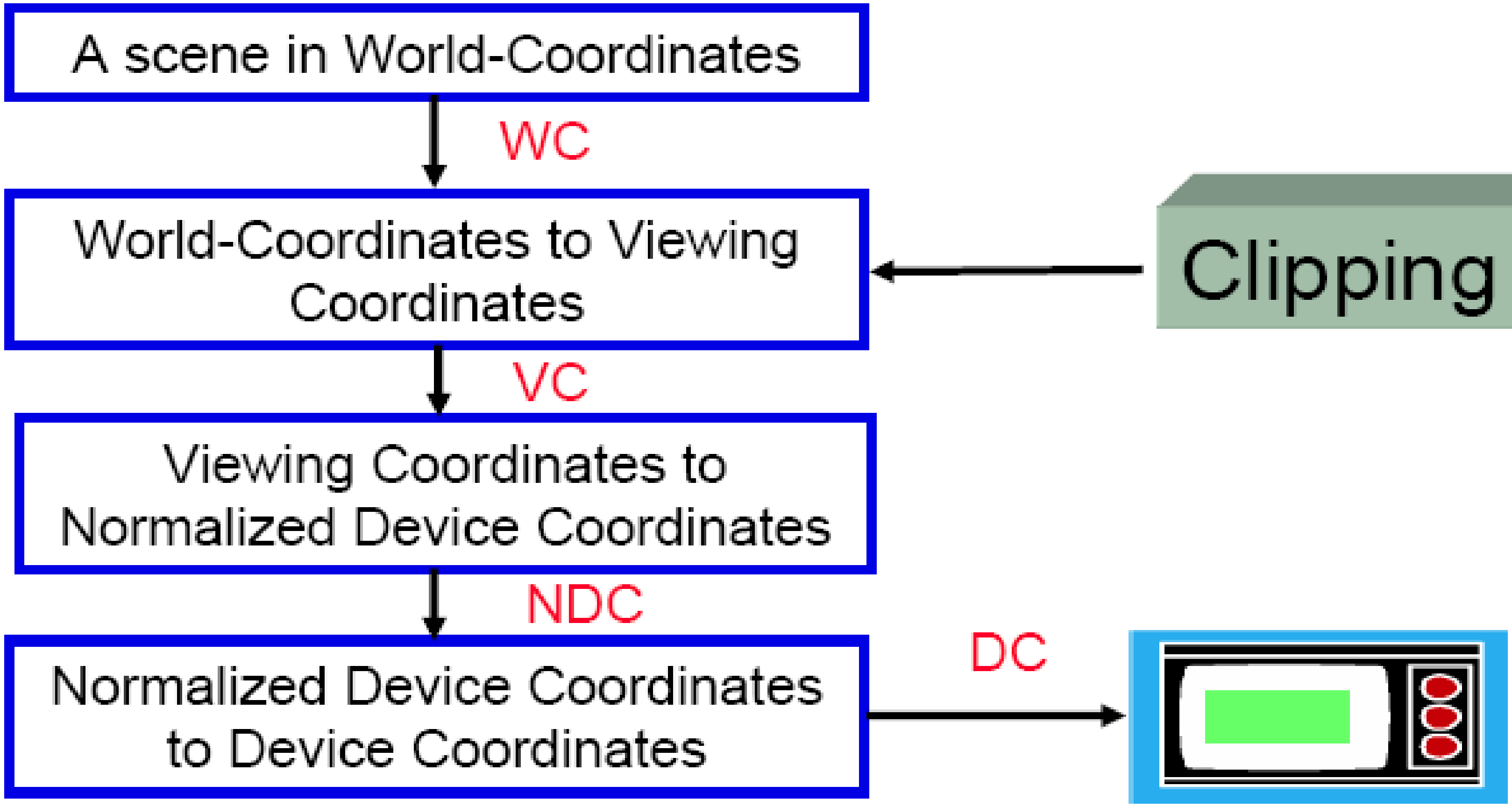
- **World window:**
  - is a rectangular region in the world that limits our view. It is specified by four world coordinates **XW-min**, **XW-max**, **YW-min** and **YW-max**
- **View Port:**
  - **is** a rectangular region in the screen that maps to our **world window into physical device coordinate,** it is specified by four normalized device coordinates **VX-min**, **VX-max**, **VY-min** and **VY-max**





Viewing Transformation in 2D

- Objects are given in *world coordinates*
- The world is viewed through a *window*
- The window is mapped onto a *device viewport*

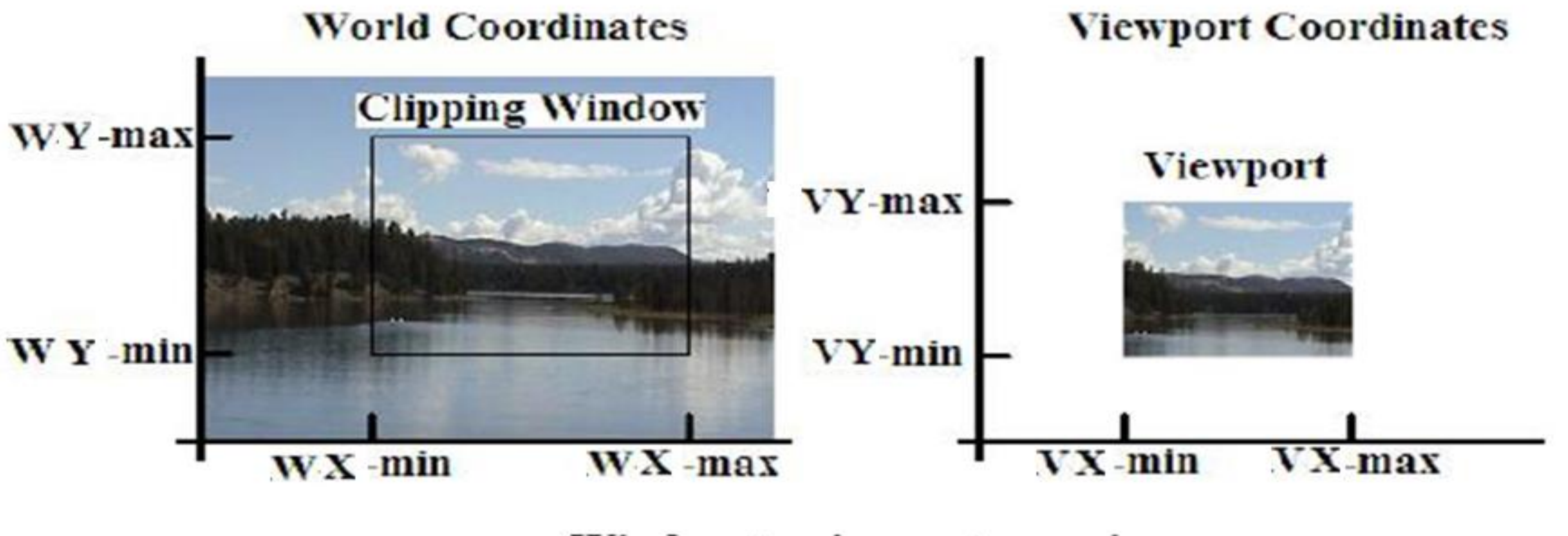

A scene in World-Coordinates
WC
World-Coordinates to Viewing Coordinates
Clipping
VC
Viewing Coordinates to Normalized Device Coordinates
NDC
Normalized Device Coordinates to Device Coordinates
DC

Window to viewport mapping

- Is the mapping process performed to convert the world coordinates of an arbitrary point in the world coordinate (WX,WY) to its corresponding normalized device coordinate (VX,VY). The objects in the world window will then be drawing in onto the view port.

$$VX = VX_{min} + \frac{(WX - WX_{min})(VX_{max} - VX_{min})}{WX_{max} - WX_{min}}$$

$$VY = VY_{min} + \frac{(WY - WY_{min})(VY_{max} - VY_{min})}{WY_{max} - WY_{min}}$$



Window-to-viewport mapping

Window to viewport mapping

- In order to maintain the same relative placement of the point in the viewport as in the window, we require Point (WX,WY) on world window to Point (VX,VY) on view port

$$\frac{WX - WX_{min}}{WX_{max} - WX_{min}} = \frac{VX - VX_{min}}{VX_{max} - VX_{min}}$$ and

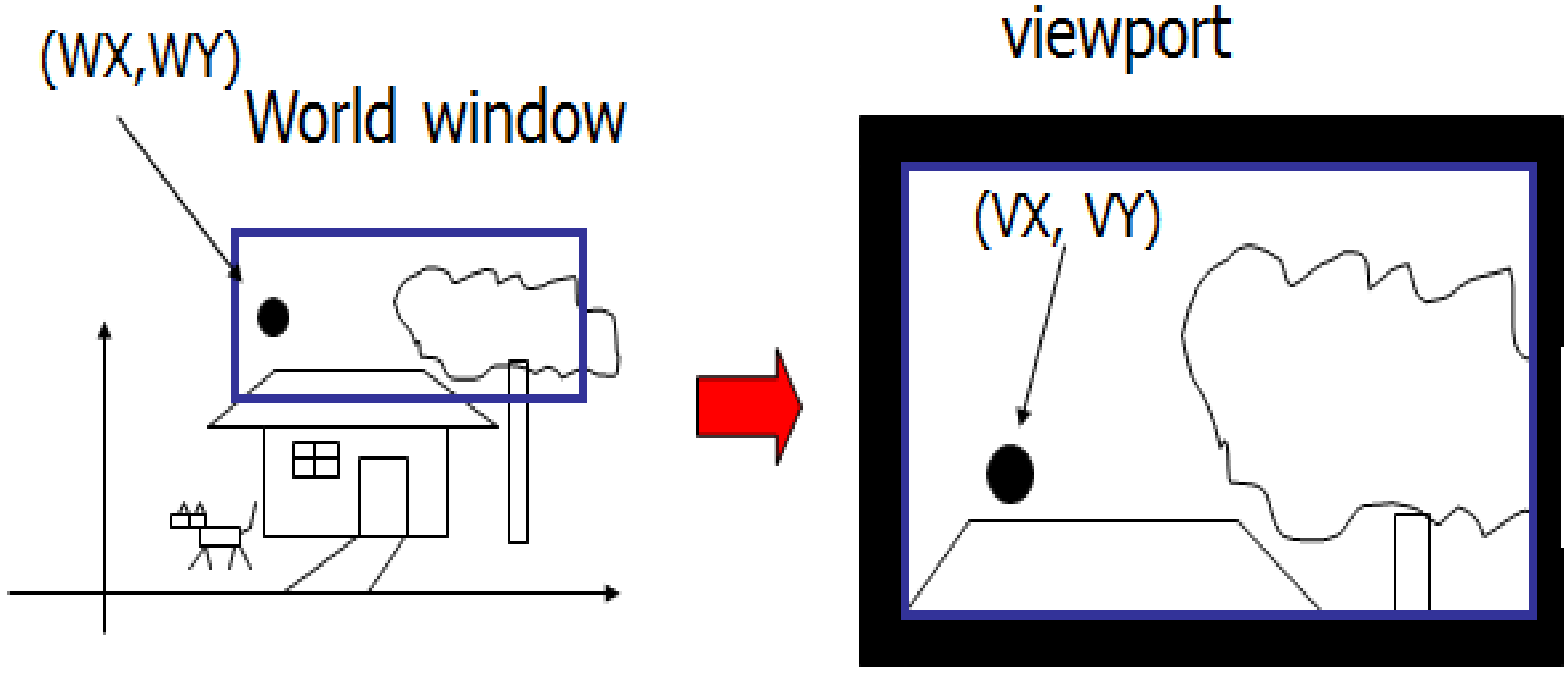
$$\frac{WY - WY_{min}}{WY_{max} - WY_{min}} = \frac{VY - VY_{min}}{VY_{max} - VY_{min}}$$ then

$$VX = VX_{min} + \frac{(WX - WX_{min})(VX_{max} - VX_{min})}{WX_{max} - WX_{min}}$$ and

$$VY = VY_{min} + \frac{(WY - WY_{min})(VY_{max} - VY_{min})}{WY_{max} - WY_{min}}$$

Window to viewport mapping

$$\begin{bmatrix} VX \\ VY \\ 1 \end{bmatrix} = N \begin{bmatrix} WX \\ WY \\ 1 \end{bmatrix}$$ where



Window to viewport mapping

- Since the eight coordinate values that define the window and the viewport are just constants.
- We can express these two formulas for computing (VX, VY) from (WX, WY) in terms of a translate-scale-translate transformation N

$$\begin{bmatrix} VX \\ VY \\ 1 \end{bmatrix} = N \begin{bmatrix} WX \\ WY \\ 1 \end{bmatrix} \quad \text{where}$$

$$N = \begin{pmatrix} 1 & 0 & VX_{min} \\ 0 & 1 & VY_{min} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{VX_{max} - VX_{min}}{WX_{max} - WX_{min}} & 0 & 0 \\ 0 & \frac{VY_{max} - VY_{min}}{WY_{max} - WY_{min}} & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & WX_{min} \\ 0 & 1 & WY_{min} \\ 0 & 0 & 1 \end{pmatrix}$$

Window to viewport mapping

Problem 1 Let $s_x = \frac{vx_{max} - vx_{min}}{wx_{max}}$ and $s_y = \frac{vy_{max} - vy_{min}}{wy_{max} - wy_{min}}$

Express window-to-viewport mapping in the form of a composite transformation matrix.

Answer

$$N = \begin{pmatrix} 1 & 0 & vx_{min} \\ 0 & 1 & vx_{min} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -wx_{min} \\ 0 & 1 & -wx_{min} \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} s_x & 0 & -s_x wx_{min} + vx_{min} \\ 0 & s_y & -s_y wy_{min} + vy_{min} \\ 0 & 0 & 1 \end{pmatrix}$$

Window to viewport mapping

- **Example 1:** Find the normalization transformation that maps a window whose lower left corner is at (1,1) and upper right corner is at (3, 5) onto a viewport that is the entire normalized device screen.
- The window parameters are WX-min=1 , WY-min=1 , WX-max=3, WY-max=5
- The View parameters are VX-min=0 , VY-min=0 , VX-max=1, VY-max=1

$$S_x = \frac{VX_{max} - VX_{min}}{WX_{max}} \text{ and } s_y = \frac{VY_{max} - VY_{min}}{WY_{max} - WY_{min}}$$

- Sx=1/2 and Sy=1/4

$$N = \begin{pmatrix} \frac{1}{2} & 0 & -\frac{1}{2} \\ 0 & \frac{1}{4} & -\frac{1}{4} \\ 0 & 0 & 1 \end{pmatrix}$$

Window to viewport mapping

- **Example 2:** Find the point B in the view port that is resulted from the transformation of point A(2,3) in the world window using the boundary of the previous example 1.

- $$\begin{bmatrix} Vx \\ Vy \\ 1 \end{bmatrix} = N \begin{bmatrix} Wx \\ Wy \\ 1 \end{bmatrix} = \begin{pmatrix} \frac{1}{2} & 0 & \frac{-1}{2} \\ 0 & \frac{1}{4} & \frac{-1}{4} \\ 0 & 0 & 1 \end{pmatrix} \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{2} \\ \frac{1}{2} \\ 1 \end{bmatrix}$$

- Then Vx=$\frac{1}{2}$ and Vy=$\frac{1}{2}$

- The required point B is $(\frac{1}{2}, \frac{1}{2})$

Window to viewport mapping

- **Example 3:** Find the normalization transformation that maps a window whose lower left corner is at (1,1) and upper right corner is at (3, 5) onto a viewport that has lower left corner at (0, 0) and upper right corner (0.5,0.5)

The window parameters are $wx_{min} = 1$, $wy_{max} = 1$, and $wy_{max} = 5$.

The viewport parameters are

$vx_{min} = 0$, $vx_{max} = \frac{1}{2}$, $vy_{min} = 0$, $vy_{max} = \frac{1}{2}$. The $s_x = \frac{1}{4}$, $s_y = \frac{1}{8}$, and

$$N = \begin{pmatrix} \frac{1}{4} & 0 & -\frac{1}{4} \\ 0 & \frac{1}{8} & -\frac{1}{8} \\ 0 & 0 & 1 \end{pmatrix}$$

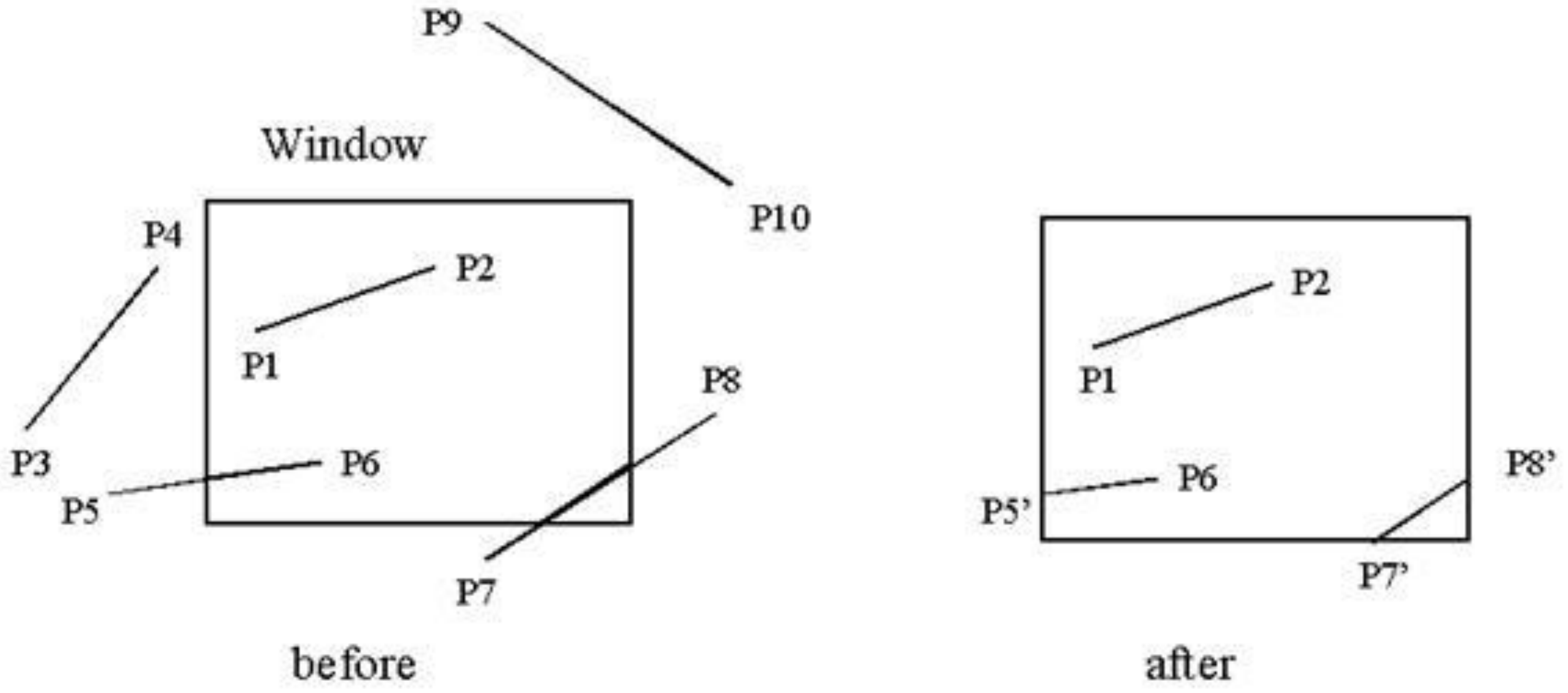
clipping

- Objects in the scene may be:
  - ➢ completely inside the window
  - ➢ completely outside the window
  - ➢ partially visible through the window.
- The **clipping operation** eliminates objects or portions of objects that are not visible through the window to ensure the proper construction of the corresponding image. It may occur in the **WC** or **DC** space, where the window is used to clip the objects; it may also occur in the **NDC** space.
- **Point Clipping**:
  - A point (X,Y) is considered inside the window when the following inequalities all evaluate to true.

    $WX_{min} \le X \le WX_{max}$ and $WY_{min} \le Y \le WYmax$
  - Where WX_{min}, WX_{max}, WY_{min} and WY_{max} define the clipping window.

Line clipping Procedure

- Check a given line segment to determine whether it is:
  - **lies completely inside** the clipping window
  - **lies completely outside** the window
  - **Lies Partially**: perform intersection calculations with one or more clipping boundaries



Cohen-Sutherland line clipping

- **Divide the line clipping process into two phases:**
  1. Identify those lines which intersect the clipping window and so need to be clipped.
  2. perform the clipping.

- **All lines fall into one of the following clipping categories:**
  1. **Visible** – both endpoints of the line within the window
  2. **Not visible** – the line definitely lies outside the window. This will occur if the line from (x1,y1,) to (x2,y2) satisfies any one of the following four inequalities:

$$x_{1,}x_2 > x_{\max} \qquad y_{1,}y_2 > y\,\mathrm{max}$$
$$x_{1,}x_2 < x_{\min} \qquad y_1, y_2 < y\,\mathrm{min}$$

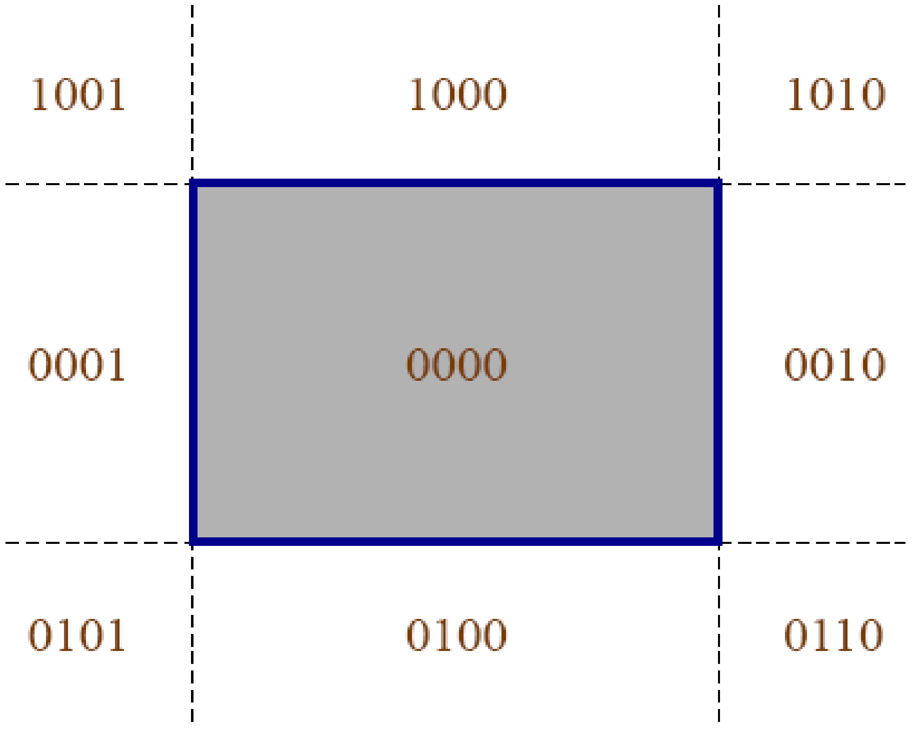
  3. **Clipping candidate** – the line is in neither visible and not Visible

Cohen-Sutherland line clipping

- Assign a 4-bit region code to each endpoint of the line, the code is determined according to which of the following nine-regions of the plan the end-points lies in.

Top – Bottom – Right – Left

- Assign a 4-bit code to each endpoint c_0, c_1 based on its position:
  - 1st bit (1000): if $y > y_{max}$
  - 2nd bit (0100): if $y < y_{min}$
  - 3rd bit (0010): if $x > x_{max}$
  - 4th bit (0001): if $x < x_{min}$

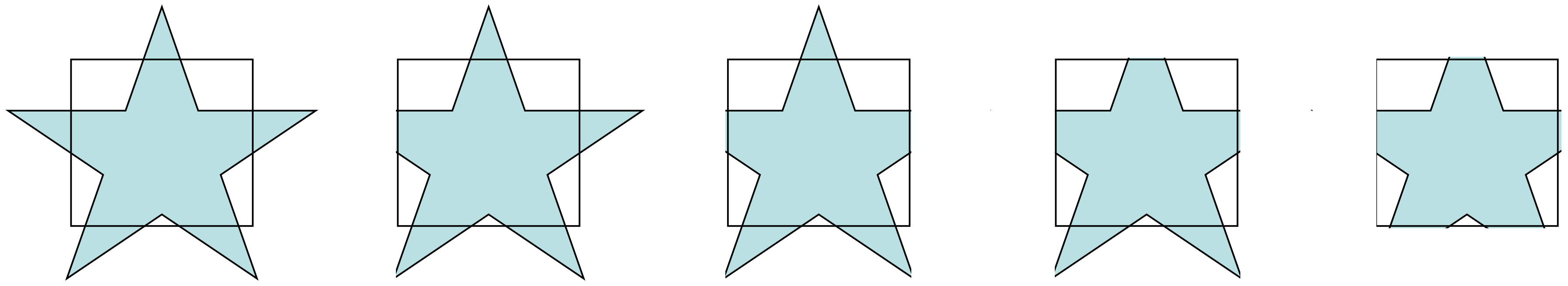


- Test using bitwise functions

```
if $c_0 \mid c_1 = 0000$
    accept (draw)
else if $c_0 \; \& \; c_1 \neq 0000$
    reject (don't draw)
else clip and retest
```

Sutherland-Hodgman Polygon Clipping Algorithm

- A technique for clipping areas developed by Sutherland & Hodgman
- Put simply the polygon is clipped by comparing it against each boundary in turn.



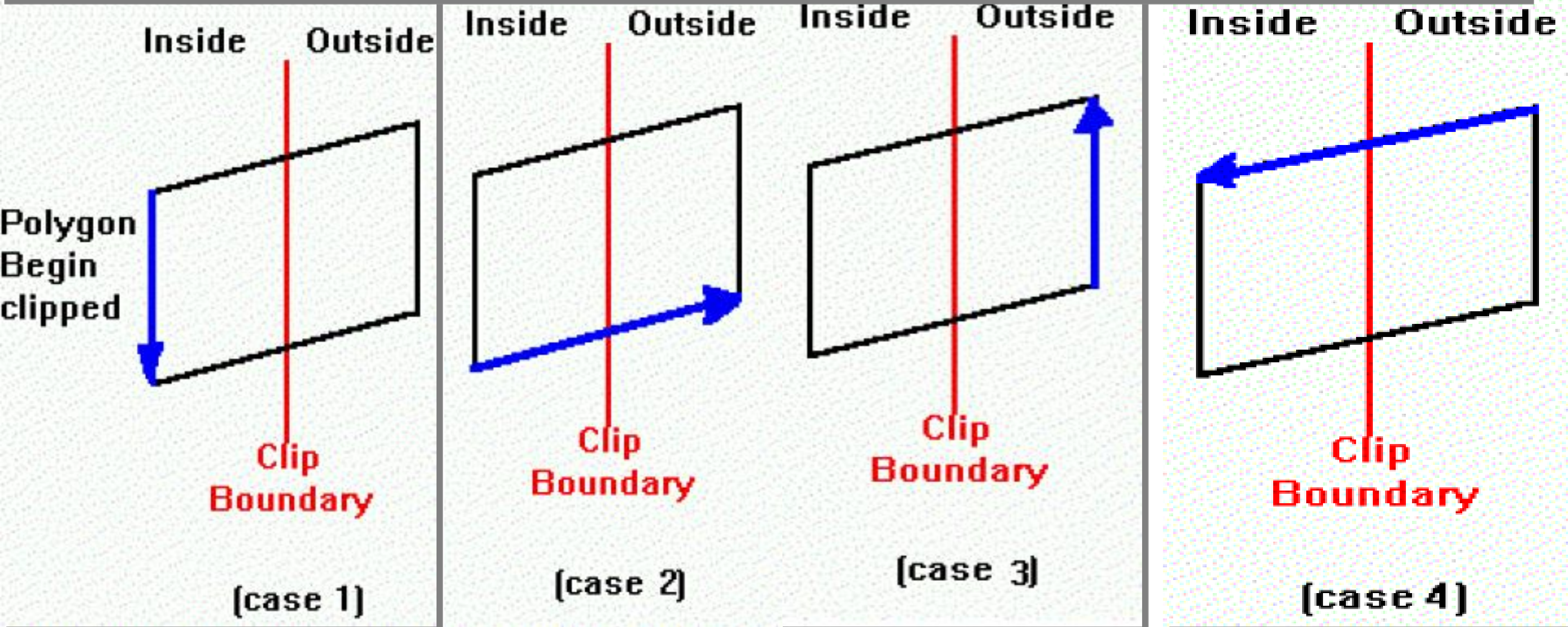
Sutherland-Hodgman Polygon Clipping Algorithm

- **Input**: list of polygon vertices in order.
- **Output**: list of clipped polygon vertices consisting of old vertices (maybe) and new vertices (maybe)
- Sutherland-Hodgman does four tests on every edge of the polygon:
  - Inside – Inside (I-I)
  - Inside – Outside (I-O)
  - Outside –Outside (O-O)
  - Outside -Inside(O-I)
- Output co-ordinate list is created by doing these tests on every edge of poly.

Four Cases of polygon clipping against one edge

The clip boundary determines a visible and invisible region. The edges from vertex i to vertex i+1 can be one of four types:

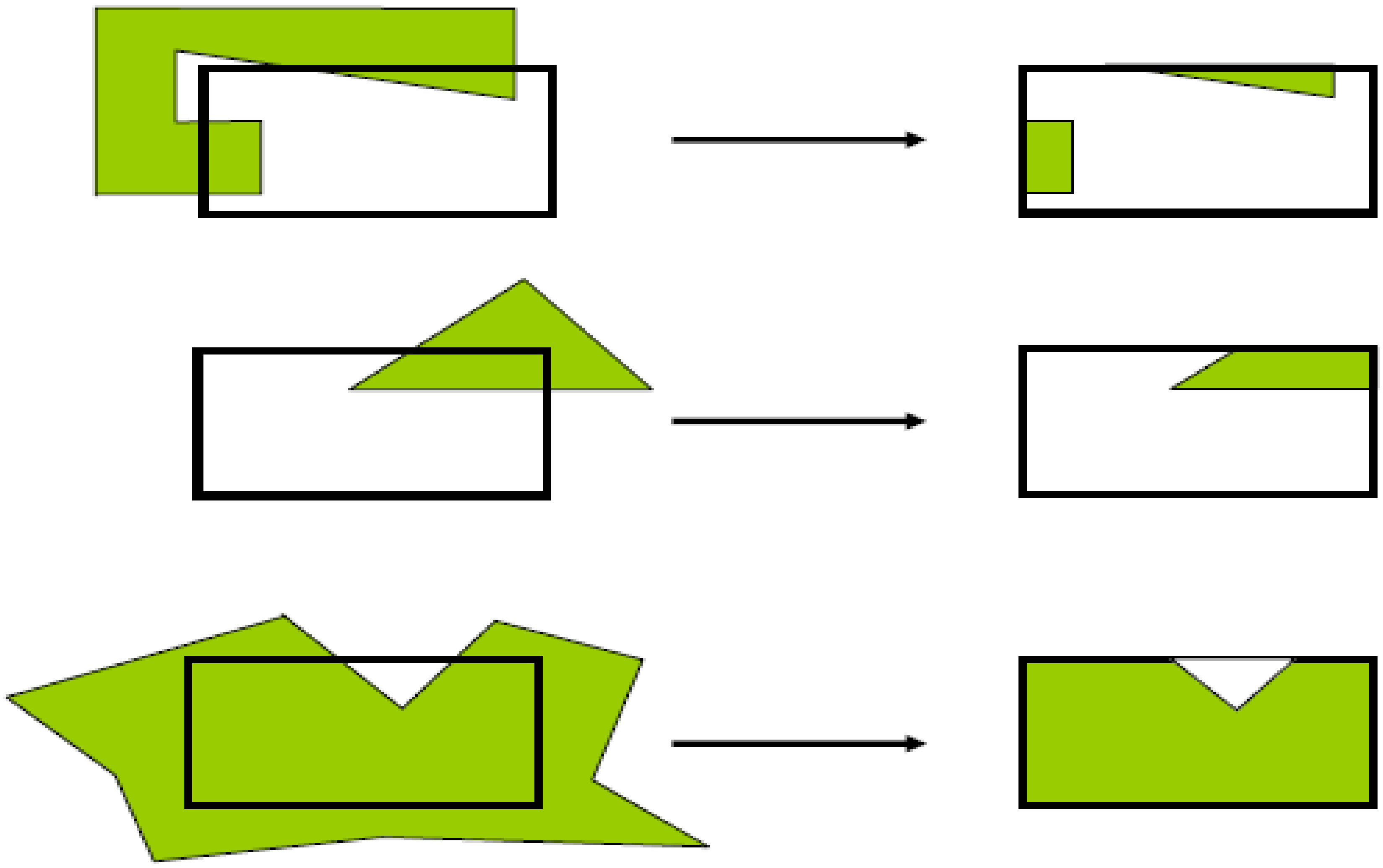
- Case 1 : Wholly inside visible region - save endpoint
- Case 2 : Exit visible region - save the intersection
- Case 3 : Wholly outside visible region - save nothing
- Case 4 : Enter visible region - save intersection and endpoint



Sutherland-Hodgman Polygon Clipping Algorithm

1. Read polygon vertices
2. Read window coordinates
3. For every edge of window do
4. Check every edge of the polygon to do 4 tests
    1. **Save the resultant vertices and the intersections in the output list.**
    2. **The resultant set of vertices is then sent for checking against next boundary.**
5. Draw polygon using output-list.
6. Stop.

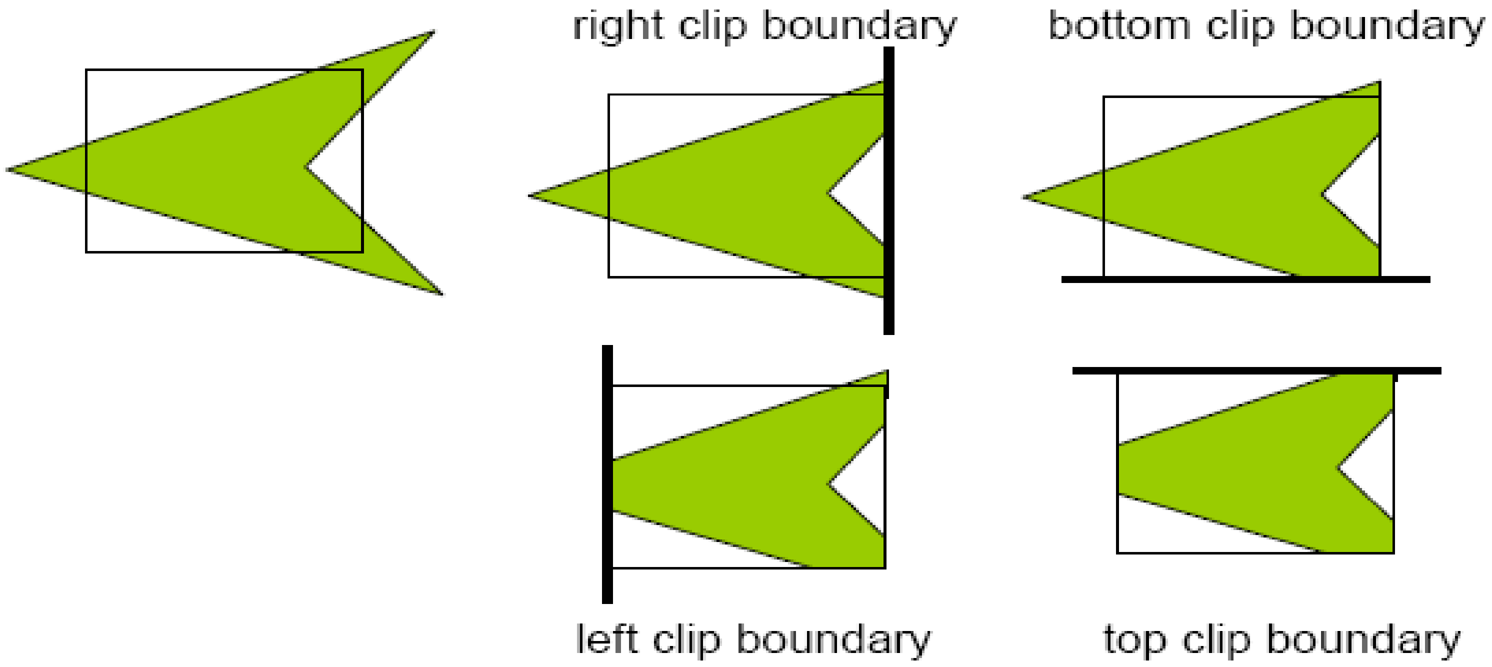
Sutherland-Hodgman Polygon-Clipping Algorithm



Sutherland-Hodgman Polygon-Clipping Algorithm

Idea: Clip a polygon by successively clipping against each (infinite) clip edge

After each clipping a new set of vertices is produced.

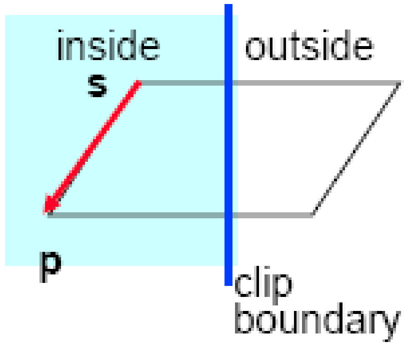


Sutherland-Hodgman Polygon-Clipping Algorithm

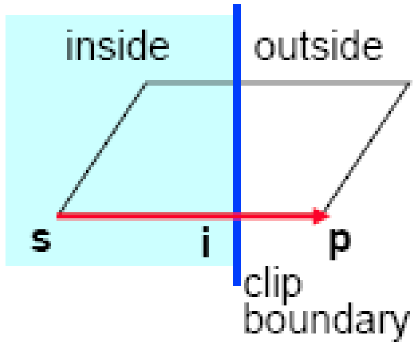
For each clip edge - scan the polygon and consider the relation between successive vertices of the polygon

Each iteration adds 0, 1 or 2 new vertices

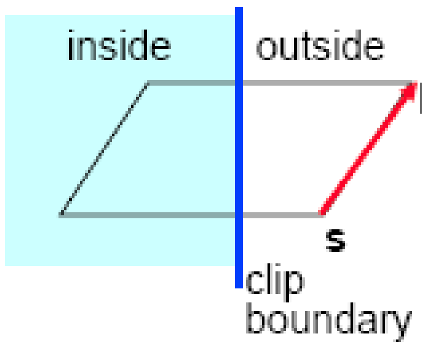
Assume vertex **s** has been dealt with, vertex **p** follows:



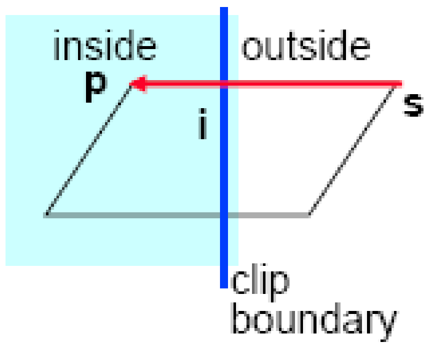
p added to output list



i added to output list

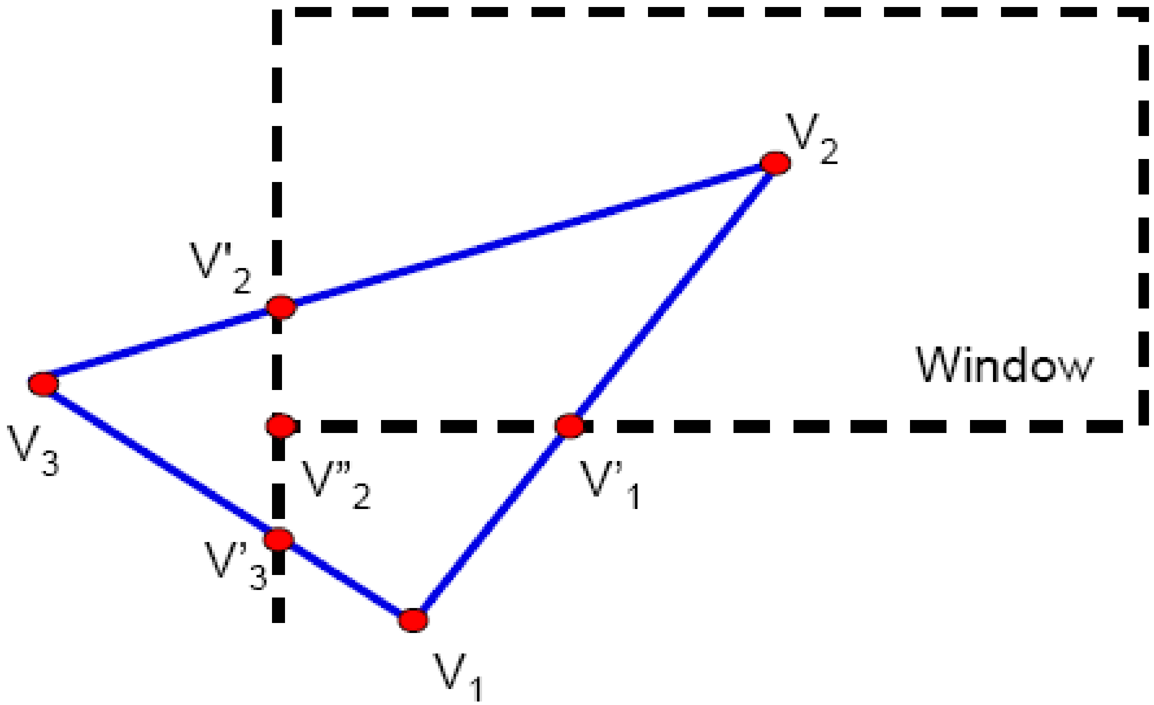


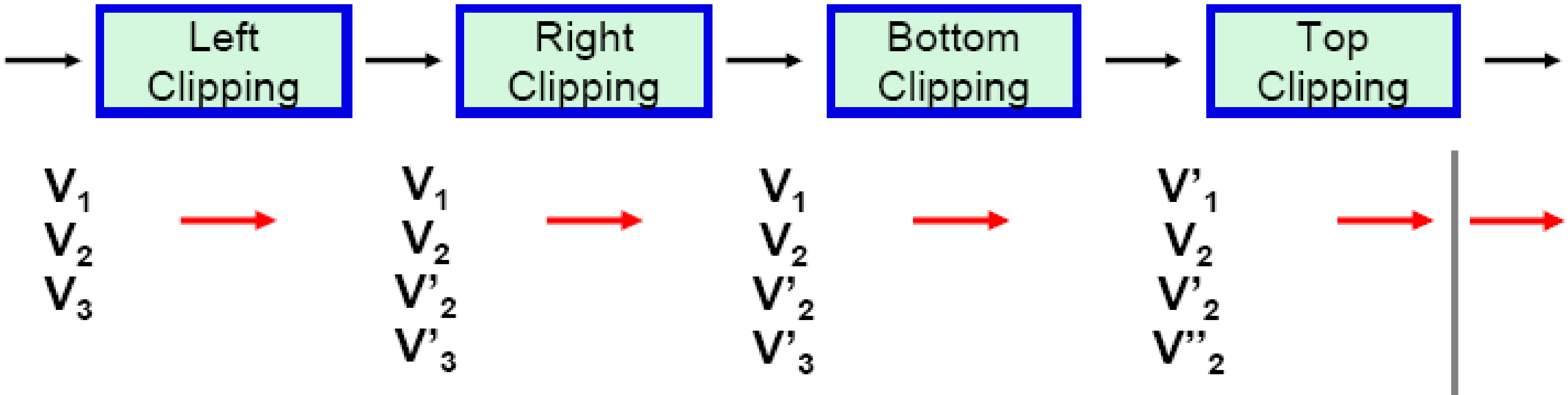
no output



i and **p** added to output list

Sutherland-Hodgman Polygon-Clipping Algorithm

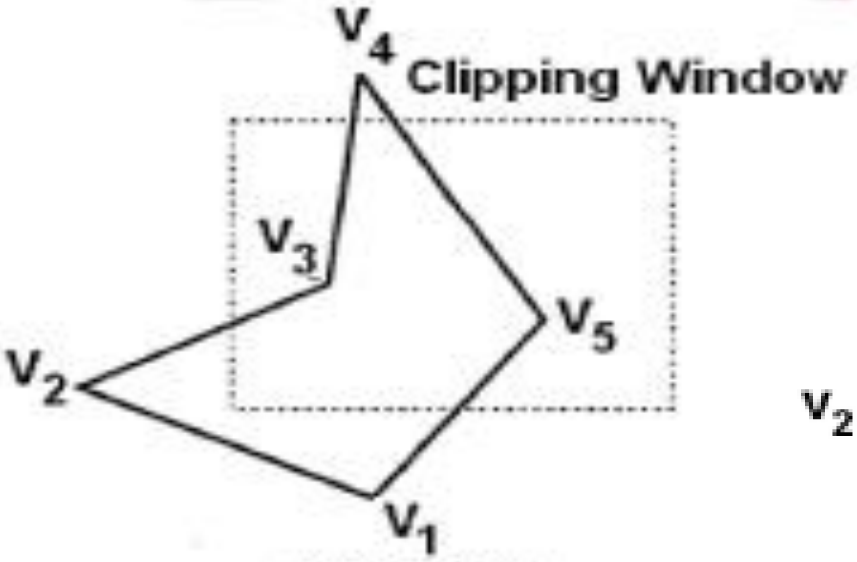




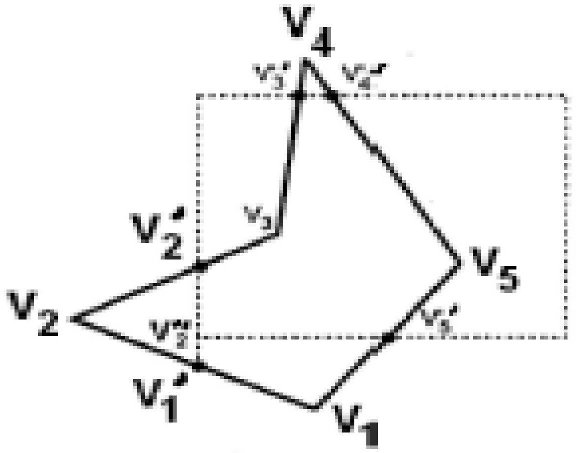
V_1, V_2, V_3 → V_1, V_2, V'_2, V'_3 → V_1, V_2, V'_2, V'_3 → V'_1, V_2, V'_2, V''_2 →

Example:

For a polygon and clipping window shown in Fig M4.20, give the list of vertices after each boundary clipping.



(Fig M4.20)



(Fig M4.21)

Solution:

Original polygon vertices are V_1 V_2 V_3 V_4 V_5 . After clipping each boundary the new vertices are given in fig M4.21:

After left clipping	:	V_1, V_1', V_2', V_3, V_4, V_5
After right clipping	:	V_1, V_1', V_2', V_3, V_4, V_5
After top clipping	:	V_1, V_1', V_2', V_3, V_3',V_4', V_5
After bottom clipping	:	V_2'', V_2', V_3, V_3', V_4', V_5, V_5'