Trinity College of Engineering & Research
Department of Computer Engineering

Computer Graphics

UNIT 2 Polygon, Windowing and Clipping

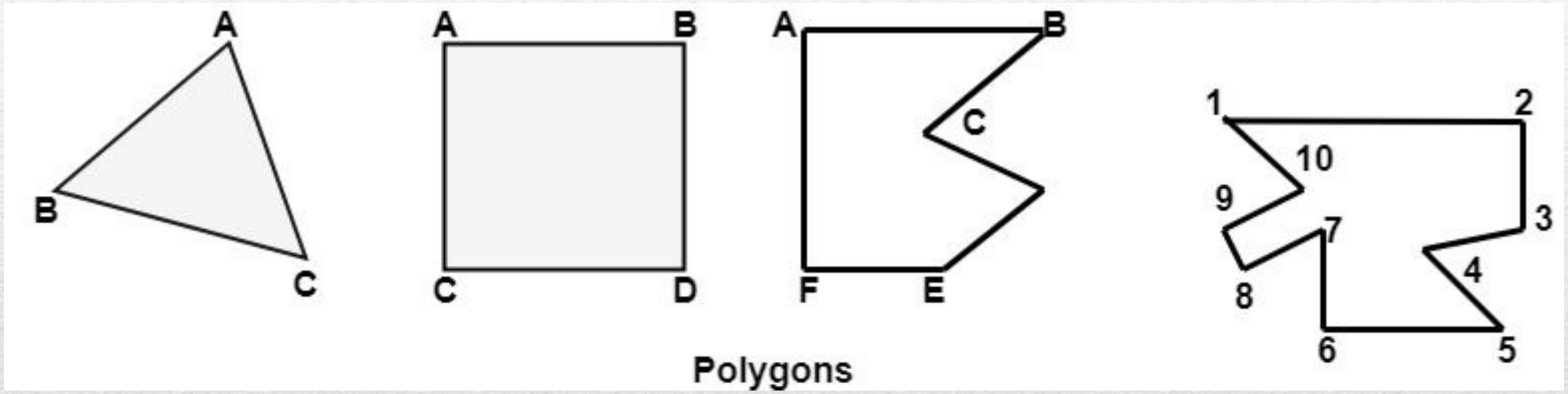
Prof. Prasad I. Bhosle

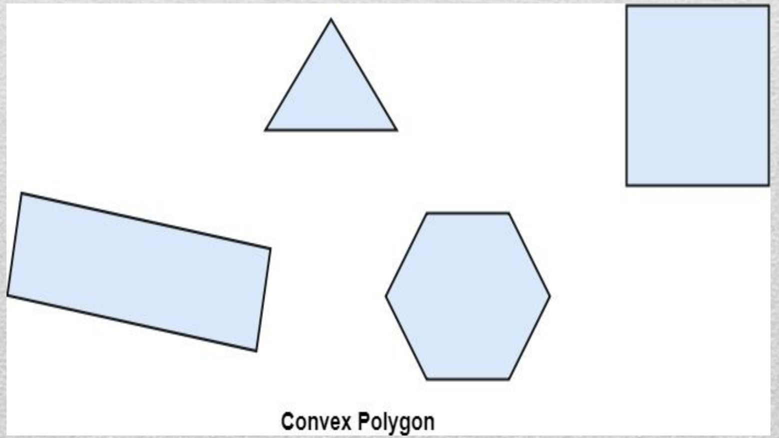
UNIT2 Polygon, Windowing and Clipping

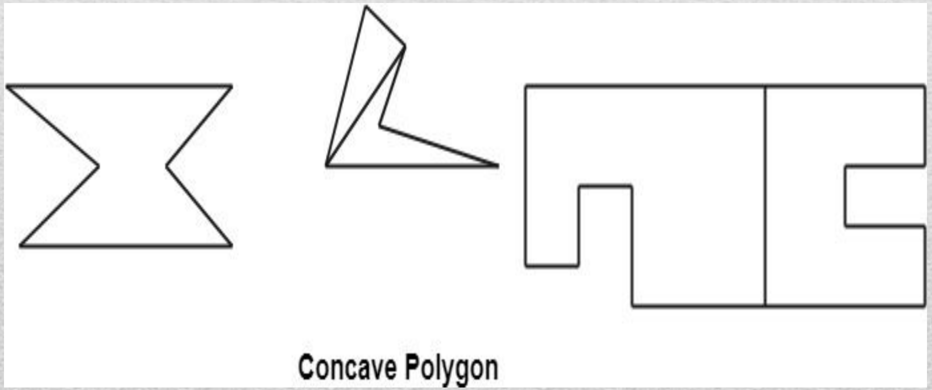
2.1. Polygon:

It is primitive which is closed in nature. It is formed using a collection of lines. It is also called as many-sided figure.

The lines combined to form polygon are called sides or edges. The lines are obtained by combining two vertices.







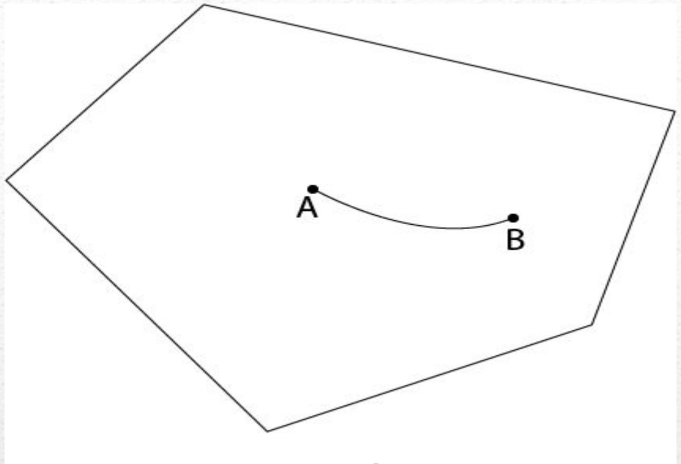
2.1 Polygon:

Types of Polygons

Concave

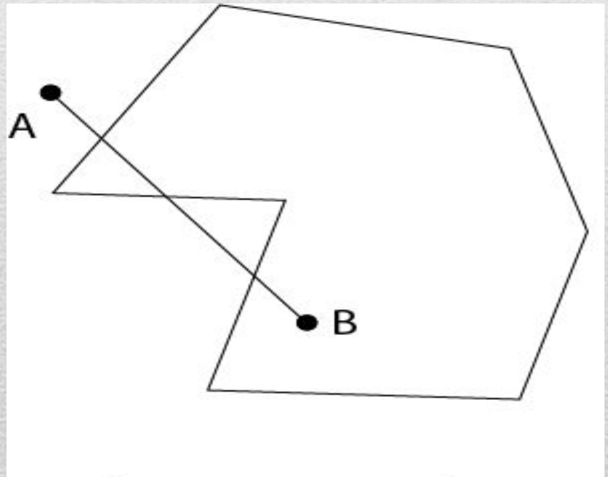
Convex

- A polygon is called **convex** of line joining any two interior points of the polygon lies inside the polygon.



Convex polygon

- A non-convex polygon is said to be **concave**. A concave polygon has one interior angle greater than 180°. So that it can be clipped into similar polygons.



Concave polygon

2.2 Polygon Filling

2.2.1 Scan Line Algorithm

This algorithm works by intersecting scanline with polygon edges and fills the polygon between pairs of intersections.

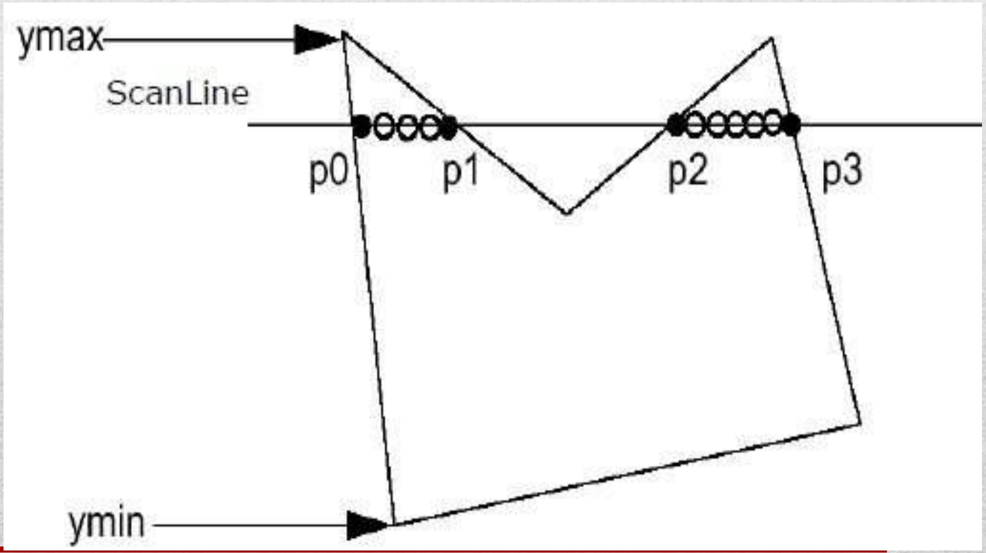
The following steps depict how this algorithm works.

Step 1 − Find out the Ymin and Ymax from the given polygon.

Step 2 − ScanLine intersects with each edge of the polygon from Ymin to Ymax. Name each intersection point of the polygon. As per the figure shown above, they are named as p0, p1, p2, p3.

Step 3 − Sort the intersection point in the increasing order of X coordinate i.e. p0,p1, p1,p2, and p2,p3.

Step 4 − Fill all those pair of coordinates that are inside polygons and ignore the alternate pairs.

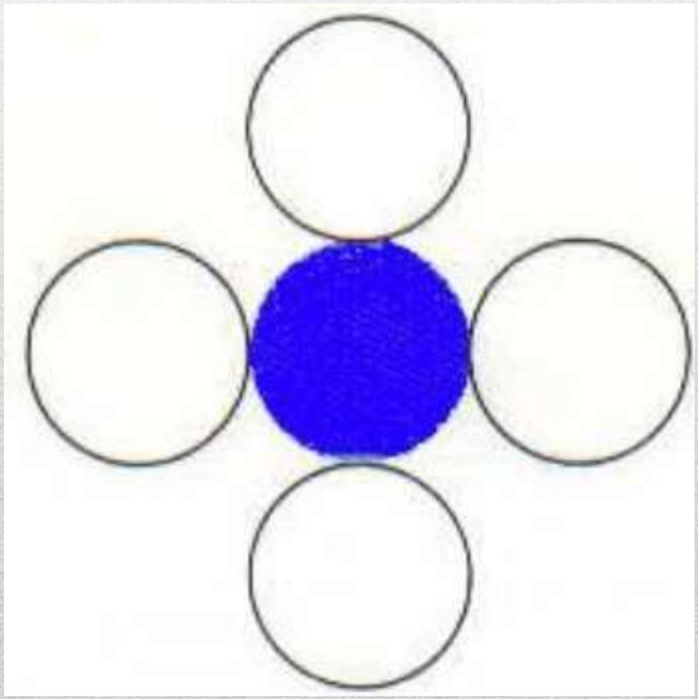


2.2.3 Flood Fill Algorithm

- This algorithm picks a point (seed point) inside an object and starts to fill until it hits the boundary of the object.
- The color of the boundary and the color that we fill should be different for this algorithm to work.
- The boundary fill algorithm can be implemented by 4-connected pixels or 8-connected pixels.

4-Connected Polygon

- In this technique 4-connected pixels are used as shown in the figure.
- We are putting the pixels above, below, to the right, and to the left side of the current pixels and this process will continue until we find a boundary with different color.



Algorithm

Step 1 − Initialize the value of seed point seedx,seedy, fcolor and dcol.
Step 2 − Define the boundary values of the polygon.
Step 3 − Check if the current seed point is of default color, then repeat the steps 4 and 5 till the boundary pixels reached.

If getpixel(x, y) = dcol then repeat step 4 and 5

Step 4 − Change the default color with the fill color at the seed point.

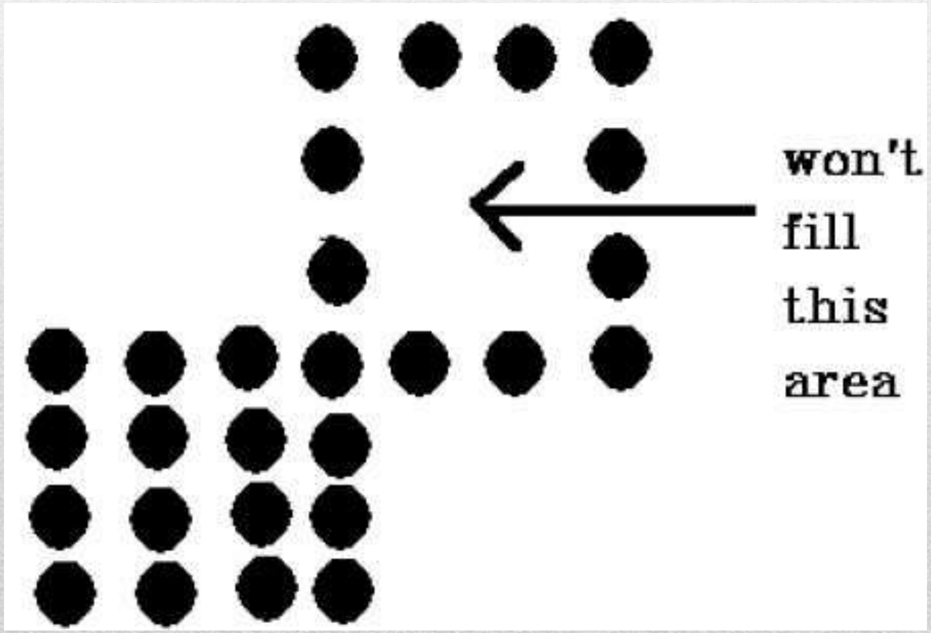
```
setPixel(seedx, seedy, fcol)
```

Step 5- Recursively follow the procedure with four neighborhood points.

```
FloodFill (seedx – 1, seedy, fcol, dcol)
FloodFill (seedx + 1, seedy, fcol, dcol)
FloodFill (seedx, seedy - 1, fcol, dcol)
FloodFill (seedx , seedy + 1, fcol, dcol)
```

Step 6- Exit

- There is a problem with this technique. Consider the case as shown below where we tried to fill the entire region. Here, the image is filled only partially. In such cases, 4-connected pixels technique cannot be used.

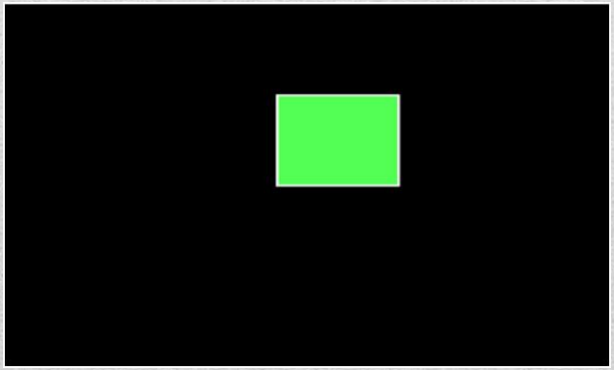


Program1: To implement 4-connected flood fill algorithm:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<dos.h>
void flood(int,int,int,int);
void main()
{
    int gd=DETECT,gm;
    initgraph(&gd,&gm,"C:/TURBOC3/bgi");
    rectangle(50,50,250,250);
    flood(55,55,10,0);
    getch();
}
```

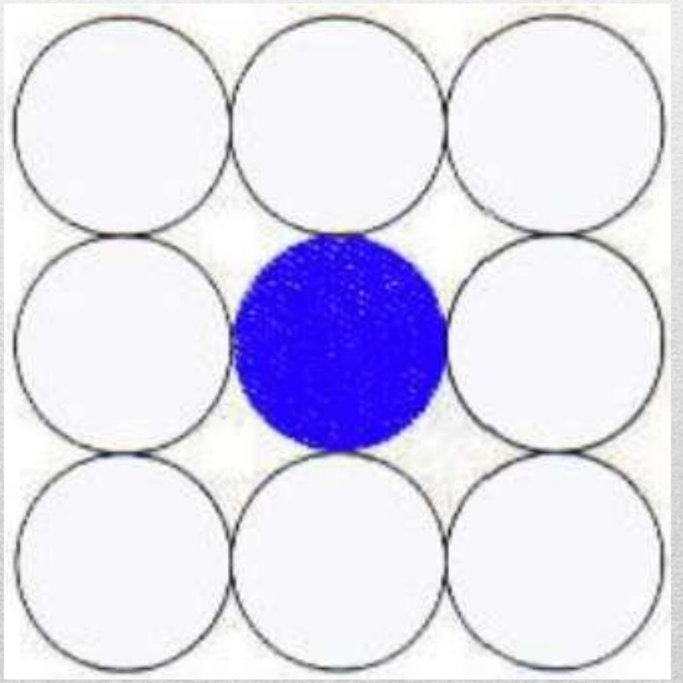
```
void flood(intx,inty,intfillColor, intdefault
Color)
{
    if(getpixel(x,y)==defaultColor)
    {
        delay(1);
        putpixel(x,y,fillColor);
        flood(x+1,y,fillColor,defaultColor);
        flood(x-1,y,fillColor,defaultColor);
        flood(x,y+1,fillColor,defaultColor);
        flood(x,y-1,fillColor,defaultColor);
    }
}
```

Out put of program1



8-Connected Polygon

- In this technique 8-connected pixels are used as shown in the figure.
- We are putting pixels above, below, right and left side of the current pixels as we were doing in 4-connected technique.
- In addition to this, we are also putting pixels in diagonals so that entire area of the current pixel is covered. This process will continue until we find a boundary with different color.



Algorithm

Step 1 − Initialize the value of seed point seedx,seedyseedx,seedy, fcolor and dcol.
Step 2 − Define the boundary values of the polygon.
Step 3 − Check if the current seed point is of default color then repeat the steps 4 and 5 till the boundary pixels reached

```
If getpixel(x,y) = dcol then repeat step 4 and 5
```

Step 4 − Change the default color with the fill color at the seed point.

```
setPixel(seedx, seedy, fcol)
```

Step 5 − Recursively follow the procedure with four neighbourhood points

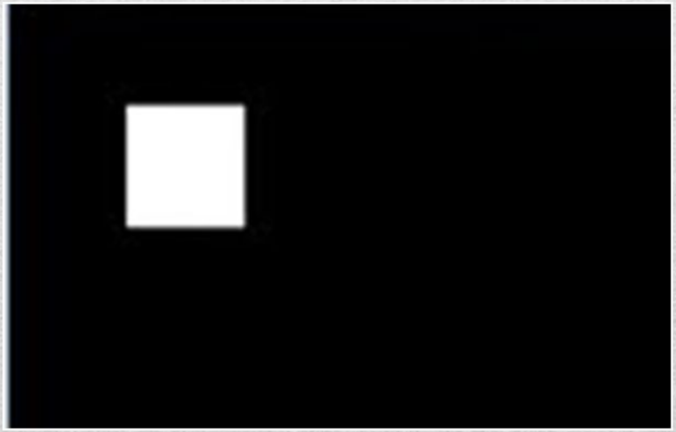
```
    FloodFill (seedx – 1, seedy, fcol, dcol)
    FloodFill (seedx + 1, seedy, fcol, dcol)
    FloodFill (seedx, seedy - 1, fcol, dcol)
    FloodFill (seedx, seedy + 1, fcol, dcol)
    FloodFill (seedx – 1, seedy + 1, fcol, dcol)
    FloodFill (seedx + 1, seedy + 1, fcol, dcol)
    FloodFill (seedx + 1, seedy - 1, fcol, dcol)
    FloodFill (seedx – 1, seedy - 1, fcol, dcol)
```

Step 6 − Exit

Program2: To implement 8-connected flood fill algorithm:

```
#include<stdio.h>
#include<graphics.h>
#include<dos.h>
#include<conio.h>
void floodfill(intx,inty,intold,intnewcol)
{               int current;
            current=getpixel(x,y);
            if(current==old)
            {         delay(5);
                      putpixel(x,y,newcol);
                      floodfill(x+1,y,old,newcol);
                      floodfill(x-1,y,old,newcol);
                      floodfill(x,y+1,old,newcol);
                      floodfill(x,y-1,old,newcol);
                      floodfill(x+1,y+1,old,newcol);
                     floodfill(x-1,y+1,old,newcol);
                    floodfill(x+1,y-1,old,newcol);
                    floodfill(x-1,y-1,old,newcol);
            }
}
```

```
void main()
{
            intgd=DETECT,gm;
            initgraph(&gd,&gm,"C:\\TUR
BOC3\\BGI");
            rectangle(50,50,150,150);
            floodfill(70,70,0,15);
            getch();
            closegraph();
}
```



Click here : https://youtu.be/o2ATh_Ss7zI?t=2

2.3 Windowing and Clipping Concept:

Introduction:

- The method of selecting and enlarging a portion of a drawing is called windowing.
- The area chosen for this display is called a window.
- The window is selected by world-coordinate.
- **Viewport:** An area on display device to which a window is mapped [where it is to displayed].
- The window is an area in object space. It encloses the object. After the user selects this,
- The space is mapped on the whole area of the viewport.

2.3 Window to viewport transformation or windowing transformation:

The mapping of a part of a world-coordinate scene to device coordinates is referred to as a viewing transformation.

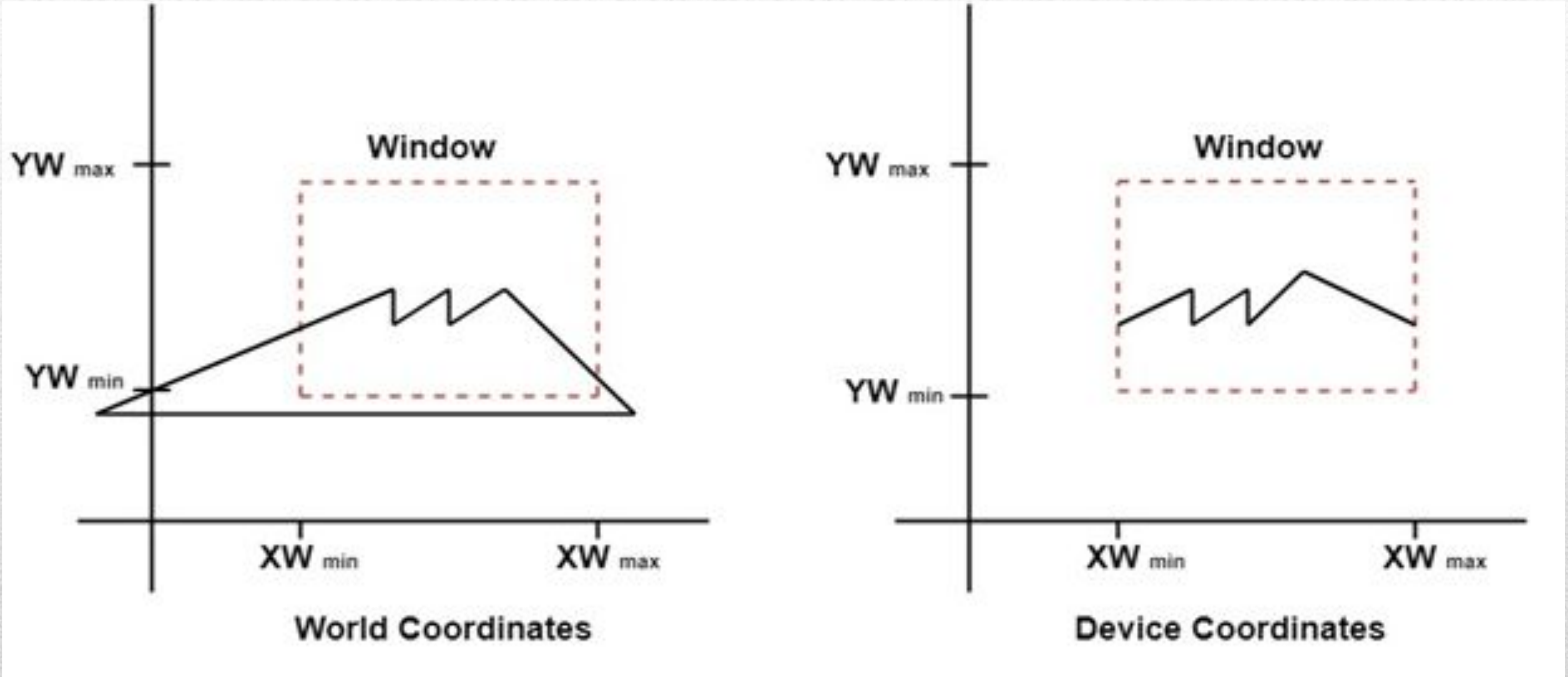
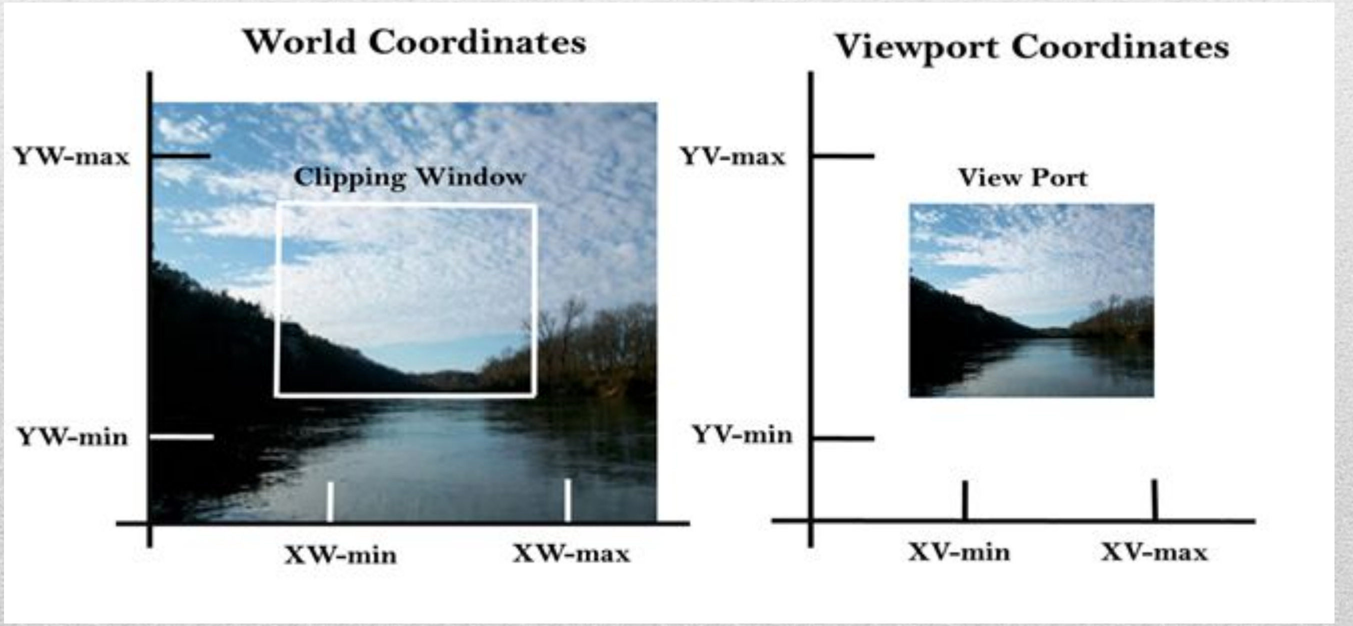


Fig: A viewing transformation using standard rectangles for the window and viewport.

2.3 Window to viewport transformation or windowing transformation:

Consider one example:

- Once object description has been transmitted to the viewing reference frame, we choose the window extends in viewing coordinates and selects the viewport limits in normalized coordinates.
- Object descriptions are then transferred to normalized device coordinates:
- We do this thing using a transformation that maintains the same relative placement of an object in normalized space as they had in viewing coordinates.



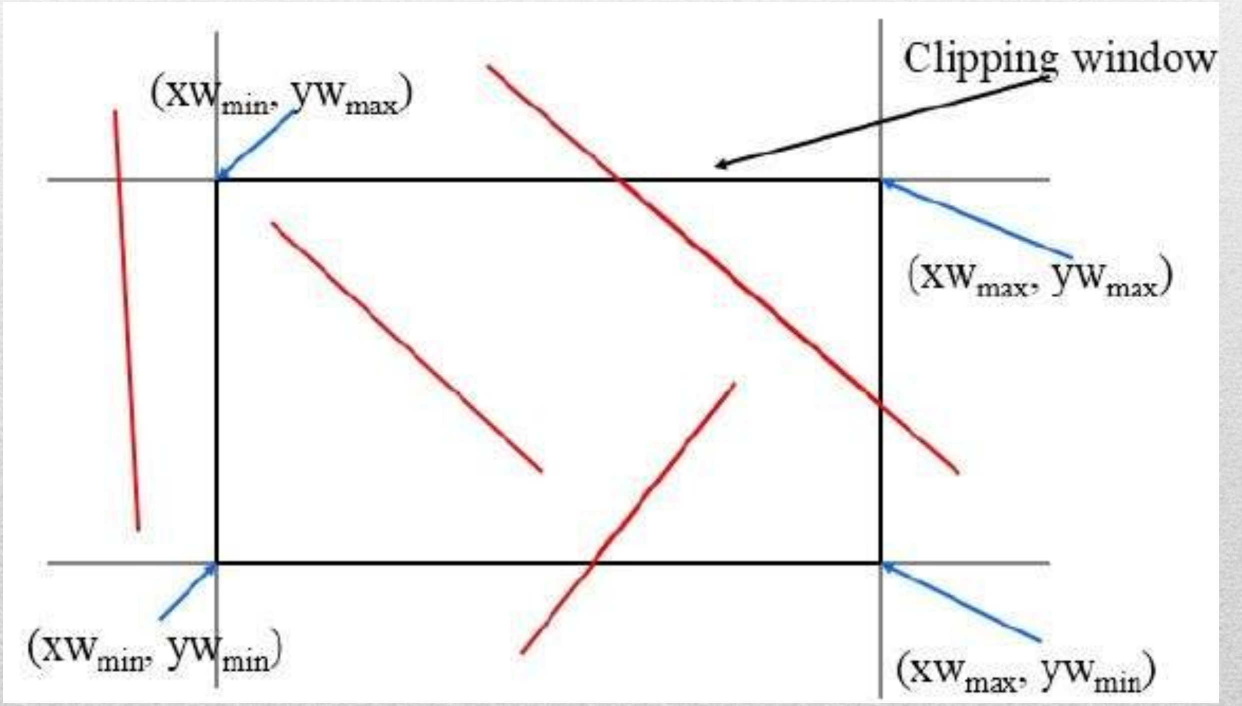
2.3. Line Clipping

- The primary use of clipping in computer graphics is to remove objects, lines, or line segments that are outside the viewing plane.
- In line clipping, we will cut the portion of line which is outside of window and keep only the portion that is inside the window.
- It is performed by using the line clipping algorithm. The line clipping algorithms are:
  1. Cohen Sutherland Line Clipping Algorithm
  2. Sutherland Hodgeman Polygon clipping algorithm,
  3. Weiler Atherton Polygon Clipping algorithm.

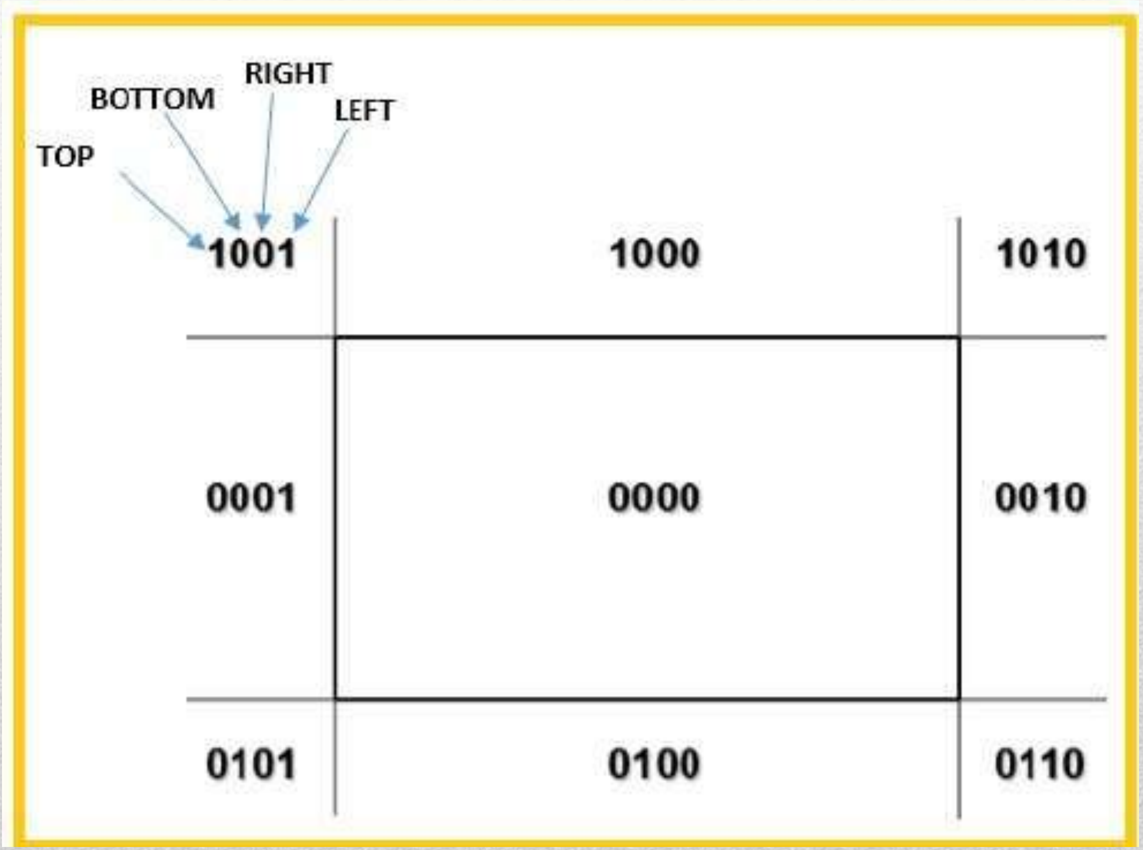
2.3 Line Clipping

2.3.1 Cohen Sutherland Line Clipping Algorithm:

- In the algorithm, first of all, it is detected whether line lies inside the screen or it is outside the screen.
- This algorithm uses the clipping window as shown in the following figure. The minimum coordinate for the clipping region is (XWmin,YWmin) and the maximum coordinate for the clipping region is (XWmax,YWmax).



2.3.1 Cohen Sutherland Line Clipping Algorithm:
We will use 4-bits to divide the entire region. These 4 bits represent the Top, Bottom, Right, and Left of the region as shown in the following figure. Here, the **TOP** and **LEFT** bit is set to 1 because it is the **TOP-LEFT** corner.



2.3.Line Clipping

2.3.1 Cohen Sutherland Line Clipping Algorithm:

There are 3 possibilities for the line −

1. Line can be completely inside the window This line should be accepted This line.
2. Line can be completely outside of the window This line will be completely removed from the region,
3. Line can be partially inside the window We will find intersection point and draw only that portion of line that is inside region.

Algorithm

```
Step 1 − Assign a region code for each end points.
Step 2 − If both endpoints have a region code 0000 then accept this line.(OR operation)
Step 3 − Else, perform the logical AND operation for both region codes.
    Step 3.1 − If the result is not 0000, then reject the line.
    Step 3.2 − Else you need clipping.
    Step 3.2.1 − Choose an endpoint of the line that is outside the window.
    Step 3.2.2 − Find the intersection point at the window boundary base on region code.
    Step 3.2.3 − Replace endpoint with the intersection point and update the region code.
    Step 3.2.4 − Repeat step 2 until we find a clipped line either trivially accepted or trivially rejected.
Step 4 − Repeat step 1 for other lines.
```

Algorithm of Cohen Sutherland Line Clipping:

Step1: Calculate positions of both endpoints of the line

Step2: Perform OR operation on both of these end-points

Step3: If the OR operation gives 0000

```
       Then
              line is considered to be visible
       else
         Perform AND operation on both endpoints
      If And ≠ 0000
         then the line is invisible
        else
      And=0000
    Line is considered the clipped case.
```

Algorithm of Cohen Sutherland Line Clipping

Step4: If a line is clipped case, find an intersection with boundaries of the window

$m=(y_2-y_1)/(x_2-x_1)$

(a) If bit 1 is "1" line intersects with left boundary of rectangle window

$y_3=y_1+m(X_{wmin}-X_1)$

where X_{wmin} is the minimum value of X co-ordinate of window

(b) If bit 2 is "1" line intersect with right boundary

$y_3=y_1+m(X_{wmax}-X_1)$

where X_{wmax} is maximum value of X co-ordinate of the window

(c) If bit 3 is "1" line intersects with bottom boundary

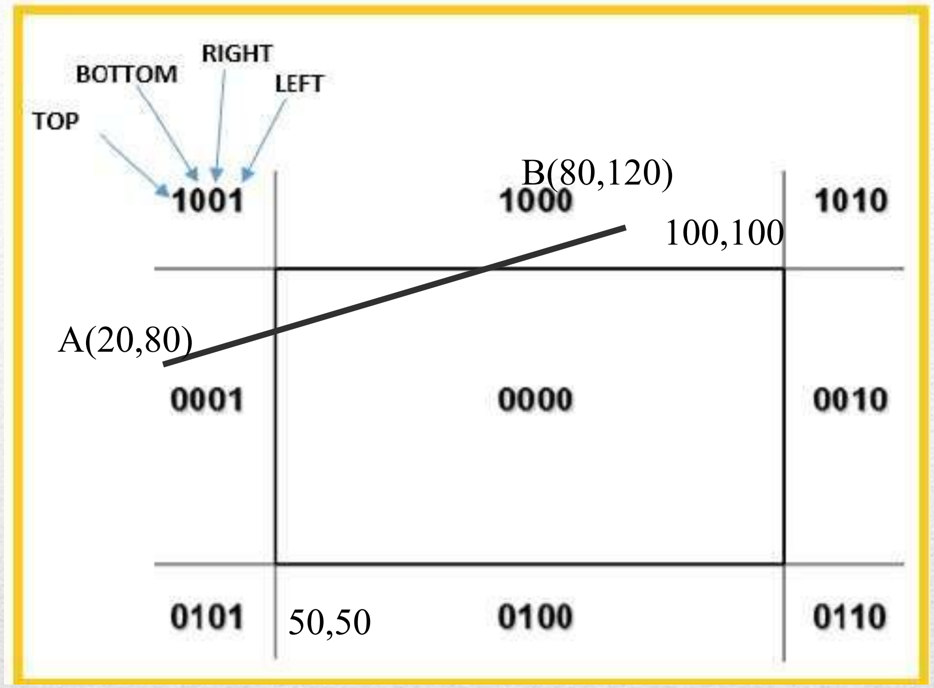
$X_3=X_1+(y_{wmin}-y_1)/m$

y_{wmin} is the minimum value of Y co-ordinate of the window

(d) If bit 4 is "1" line intersects with the top boundary

$X_3=X_1+(y_{wmax}-y_1)/m$

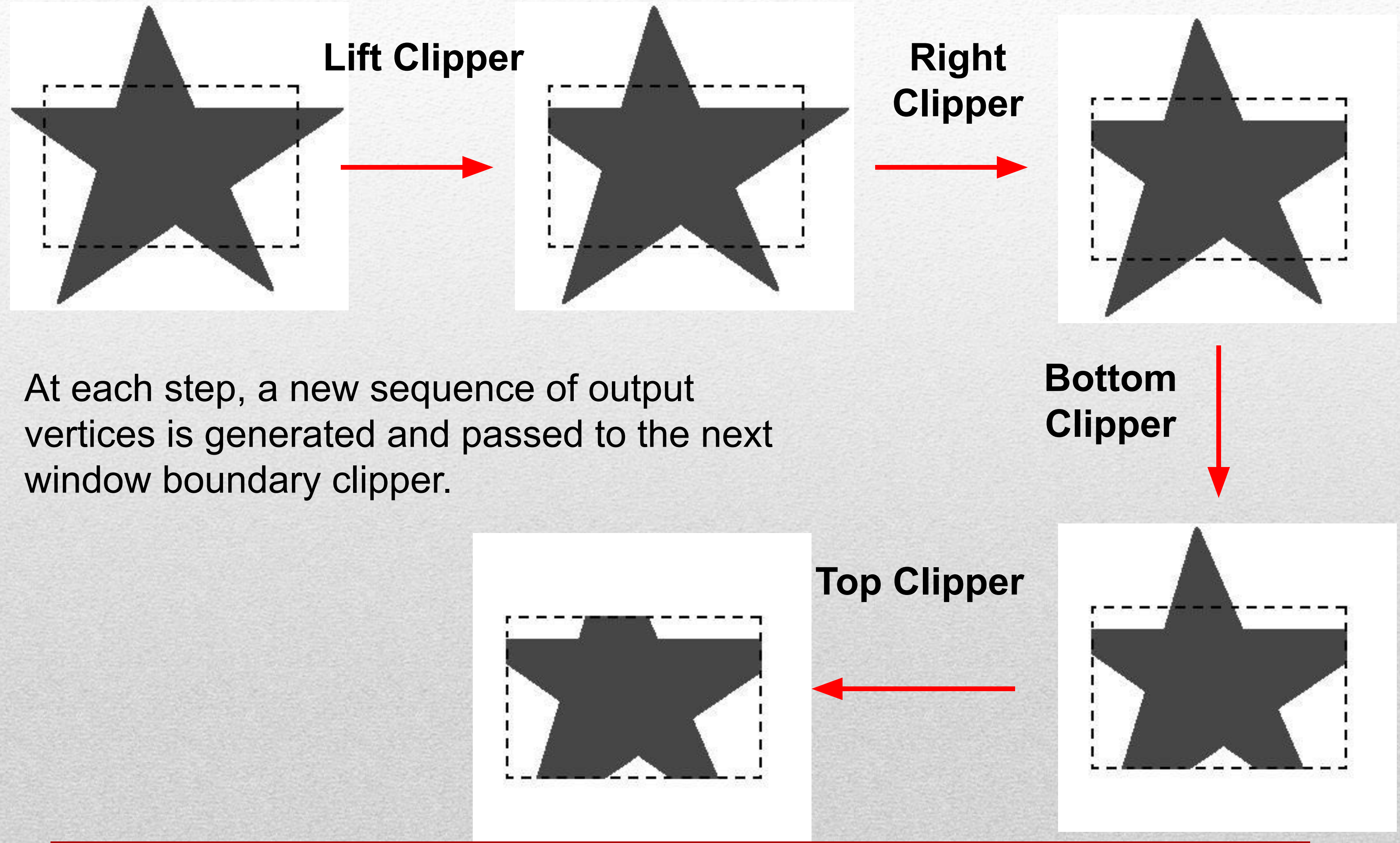
y_{wmax} is the maximum value of Y co-ordinate of the window


BOTTOM
RIGHT
LEFT
TOP
1001
B(80,120)
1000
1010
100,100
A(20,80)
0001
0000
0010
0101
50,50
0100
0110

2.3.2 Sutherland-Hodgman Polygon Clipping

- Clip a polygon by processing the polygon boundary as a whole against each window edge.
- Processing all polygon vertices against each clip rectangle boundary in turn.
- Beginning with the initial set of polygon vertices, we could first clip the polygon against the **left** rectangle boundary to produce a new sequence of vertices.
- The new set of vertices could be successively passed to a **right** boundary clipper, a **bottom** boundary clipper, and a **top** boundary clipper, a right boundary clipper.

2.3.2Sutherland-Hodgman Polygon Clipping



At each step, a new sequence of output vertices is generated and passed to the next window boundary clipper.

2.3.2 Sutherland-Hodgman Polygon Clipping

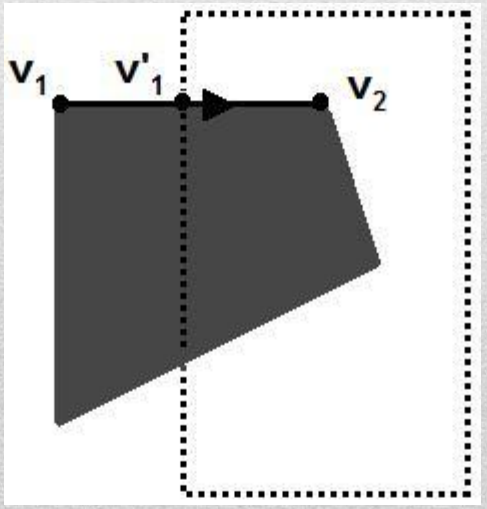
There are **four** possible **cases** when processing vertices in sequence around the perimeter of a polygon.

As **each pair** of **adjacent polygon vertices** is passed to a next window boundary clipper, we make the following tests:

2.3.2 Sutherland-Hodgman Polygon Clipping

1. **If** the **first vertex** is **outside** the window boundary and the **second vertex** is **inside**

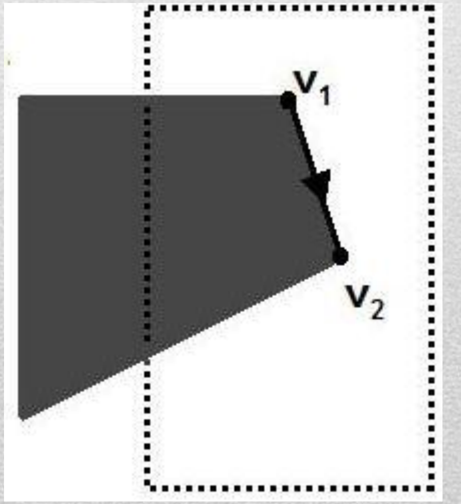
Then , both the **intersection point** of the polygon edge with the window boundary and the **second vertex** are added to the output vertex list.



4.3.1 Sutherland-Hodgman Polygon Clipping

2. **If** both input vertices are **inside** the window boundary.

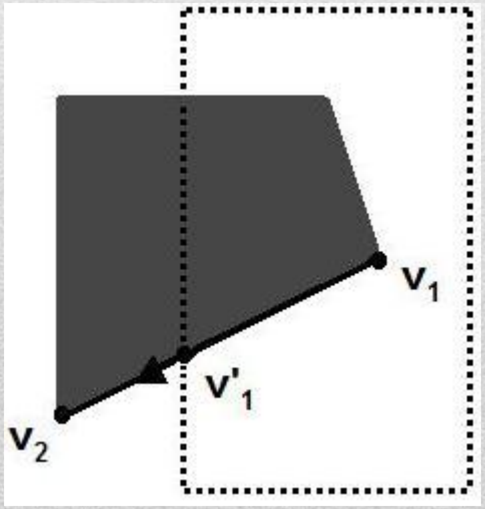
Then, only the **second vertex** is added to the output vertex list.



2.3.2 Sutherland-Hodgman Polygon Clipping

3. **If** the **first vertex** is **inside** the window boundary and the **second vertex** is **outside**.

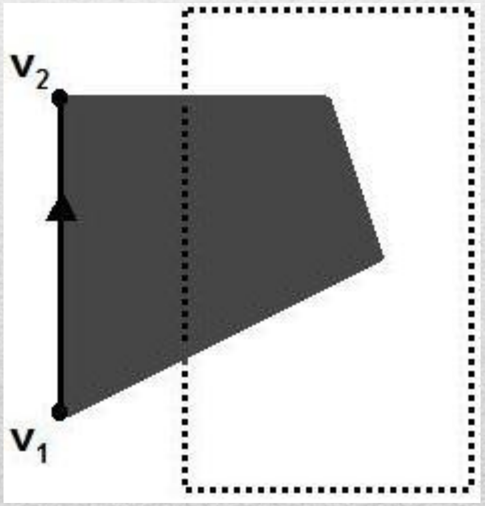
Then, only the edge intersection with the window boundary is added to the output vertex list.



2.3.2 Sutherland-Hodgman Polygon Clipping

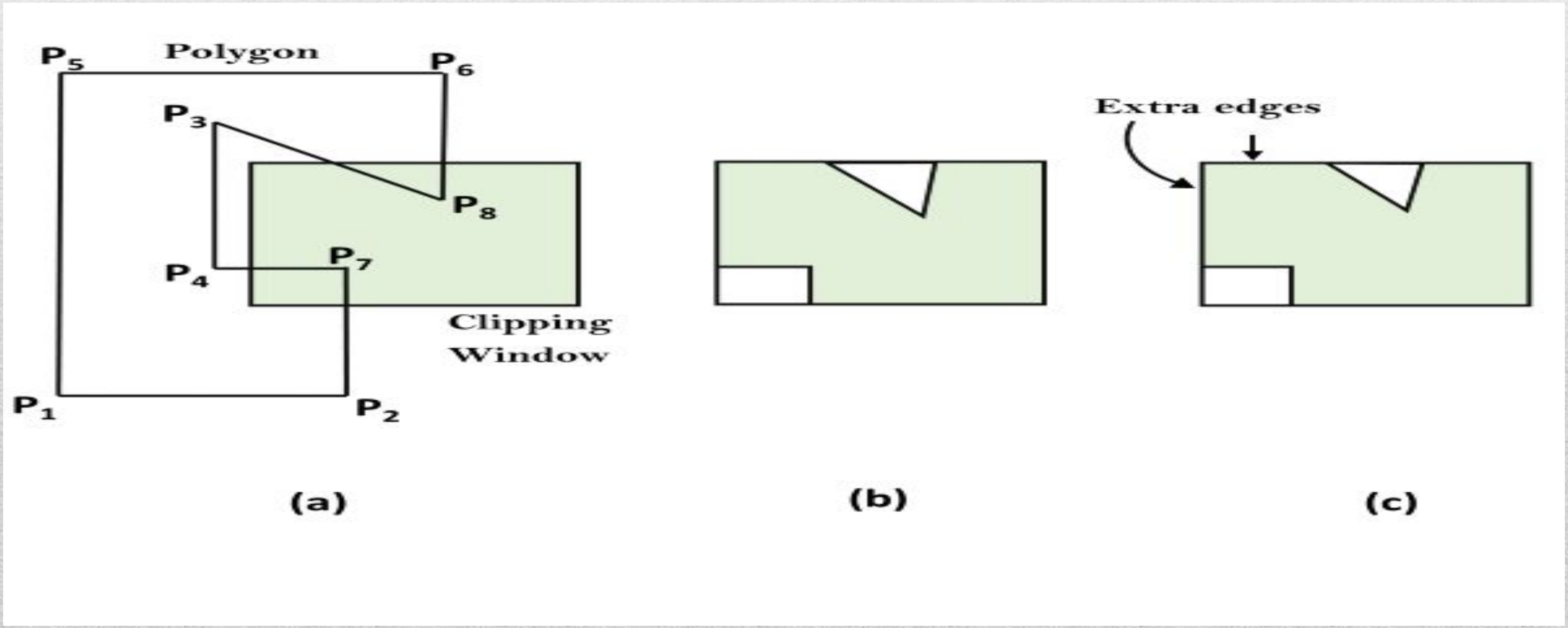
4. **If both** input vertices are **outside** the window boundary.

Then, nothing is added to the output vertex list.



Weiler-Atherton Polygon Clipping:

- When the clipped polygons have two or more separate sections, then it is the concave polygon handled by this algorithm.
- The vertex-processing procedures for window boundaries are modified so that concave polygon is displayed.
- Let the clipping window be initially called clip polygon and the polygon to be clipped the subject polygon.
- We start with an arbitrary vertex of the subject polygon and trace around its border in the clockwise direction until an intersection with the clip polygon is encountered:

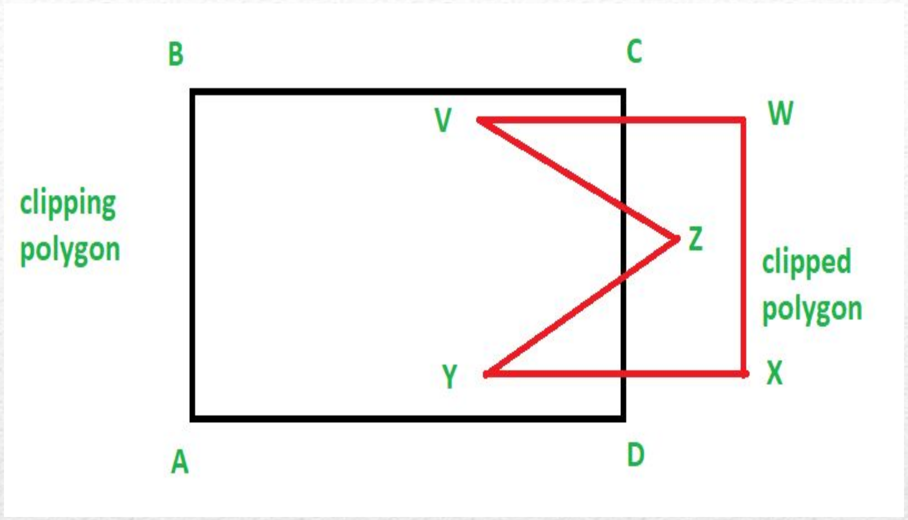


1. If the edge enters the clip polygon, record the intersection point and continue to trace the subject polygon.

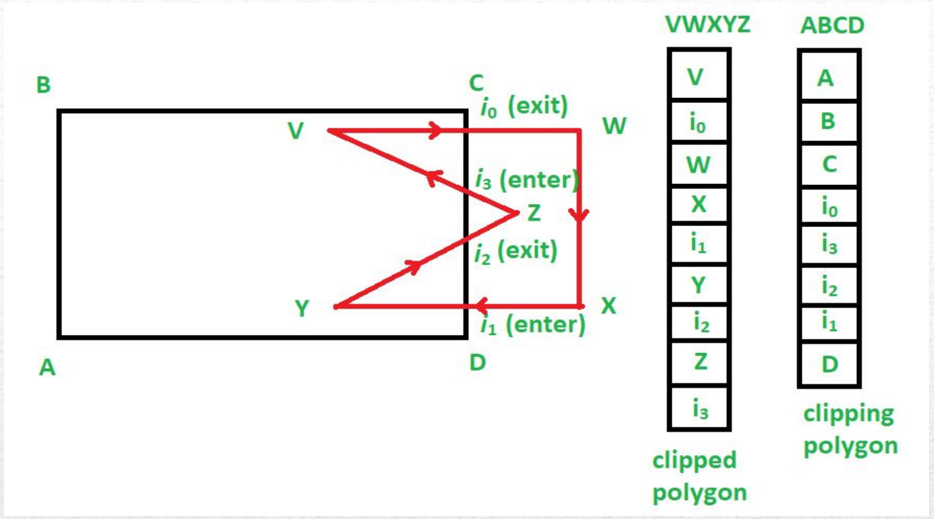
2. . If the edge leaves the clip polygon, record the intersection point and make a right turn to follow the clip polygon in the same manner (i.e., treat the clip polygon as subject polygon and the subject polygon as clip polygon and proceed as before).

- Whenever our path of traversal forms a sub-polygon we output the sub-polygon as part of the overall result.
- We then continue to trace the rest of the original subject polygon from a recorded intersection point that marks the beginning of a not-yet traced edge or portion of an edge.
- The algorithm terminates when the entire border of the original subject polygon has been traced exactly once

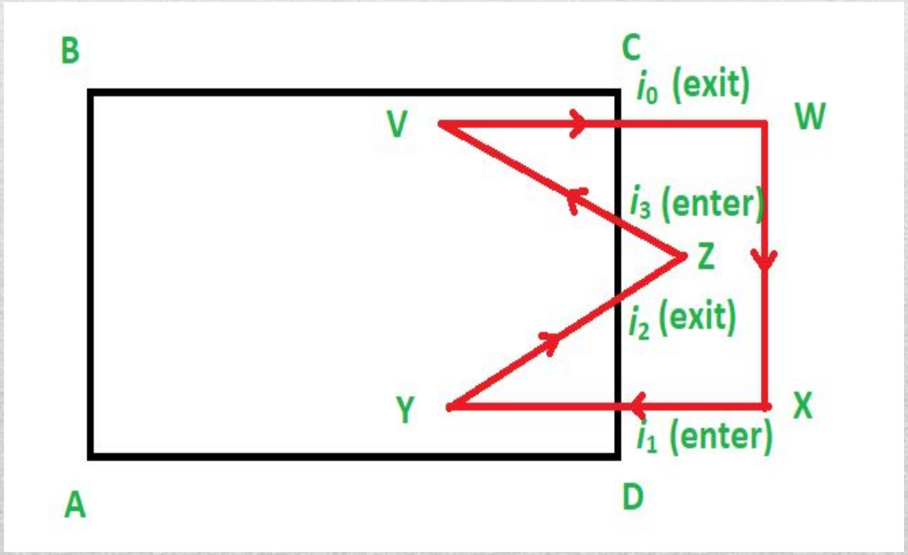
UNIT 2 WINDOWING & CLIPPING

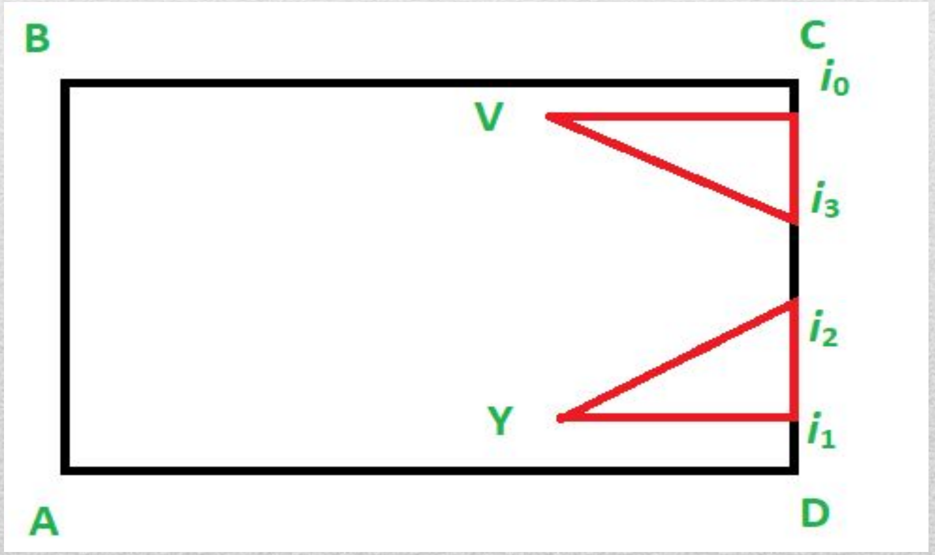


A



C





B

D