# UNIT -05

# Graphic Operations

Clipping and Windowing

## 5.1 Introduction

In very basic two dimensional 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 clipped out of the image. In many situations we have to draw objects whose dimensions are given in units completely incompatible with the screen coordinates system. Programming in device coordinates is not very convenient since the programmer has to do any required scaling from the coordinates natural to the application to device coordinates. This has led to two dimensional packages being developed which allow the application programmer to work directly in the coordiate system which is natural to the application. These user coordinates are usually called **World Coordinates (WC).** The packages then coverts the coordinates to **Device Coordinates (DC)** automatically. The transformation form the WC to DC is often carried out in tow steps. First using the **Normalisation Transformation** and then the **Workstation Transformation**. The **Viewing Transformation** is the process of going form a window in World coordinates to viewport in **Physical Device Coordinates (PDC).**

## 5.2 Window-to-Viewport Mapping

A window is specified by four world coordinates : $wx_{min}$, $wx_{max}$, $wy_{min}$,and $wy_{max}$ (see Fig. 5.1) Similarly, a viewport is described by four normalized device coordinates: $vx_{min}$,$vx_{max}$,$vy_{min}$, and $vy_{max}$. The objective of window – to – viewport mapping is to convert the world coordinates (wx, wy) of an arbitrary point to its corresponding normalized device coordinates (vx,vy). In order to maintain the same relative placement of the point in the viewport as in the window, we require:

$$\frac{wx - wx_{\min}}{wx_{\max} - wx_{\min}} = \frac{vx - vx_{\min}}{vx_{\max} - vx_{\min}} \quad \text{and} \quad \frac{wy - wy_{\min}}{wy_{\max} - wy_{\min}} = \frac{vy - vy_{\min}}{vy_{\max} - vy_{\min}}$$

Thus

$$\begin{cases} vx = \dfrac{vx_{\max} - vx_{\min}}{wx_{\max} - wx_{\min}}\left(wx - wx_{\min}\right) + vx_{\min} \\ vy = \dfrac{vy_{\max} - vy_{\min}}{wy_{\max} - wy_{\min}}\left(wy - wy_{\min}\right) + vy_{\min} \end{cases}$$

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

$$\begin{pmatrix} vx \\ vy \\ 1 \end{pmatrix} = N. \begin{pmatrix} wx \\ wy \\ 1 \end{pmatrix}$$

where

$$N = \begin{pmatrix} 1 & 0 & vx_{min} \\ 0 & 1 & vy_{min} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{vx_{max} - vx_{min}}{wx_{max} - wx_{min}} & 0 & 0 \\ 0 & \frac{vy_{max} - vy_{min}}{wy_{max} - wy_{min}} & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -wx_{min} \\ 0 & 1 & -wy_{min} \\ 0 & 0 & 1 \end{pmatrix}$$

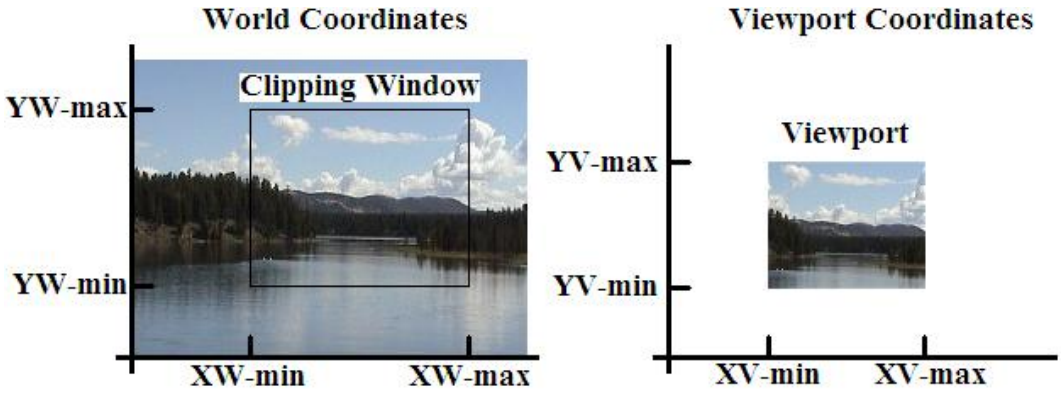



**Fig. 5.1: Window-to-viewport mapping**

Note that geometric distortions occur (e.g. squares in the window become rectangles in the viewport) whenever the two scalling constants differ.

### 5.3 Two – Dimensional Viewing and Clipping

Much like what we see in real life through a small window on the wall or the viewfinder of a camera, a Computer-generated image often depicts a partial view of a large scene. Objects are placed into the scene by modeling transformations to a master coordinate system, commonly referred to as the world coordinate system (WCS). A rectangular window with its edge parallel to the axes of the WCS is used to select the portion of the scene for which an image is to be generated (see Fig. 5.2). Sometimes an additional coordinate system called the viewing coordinate system is introduced to simulate the effect of moving and / or tilting the camera.

On the other hand, an image representing a view often becomes part of a larger image, like a photo on an album page, which models a computer monitor's display area. Since album pages vary and monitor sizes differ from one system to another, we want to introduce a device-independent tool to describe the display area. This tool is called the normalized device coordinate system (NDCS) in which a unit (1 x 1) square whose lower left corner is at the origin of the coordinate system defines the display area of a virtual display device. A rectangular viewport with its edges parallel to the axes of the NDCS is used to specify a sub-region of the display area that embodies the image.

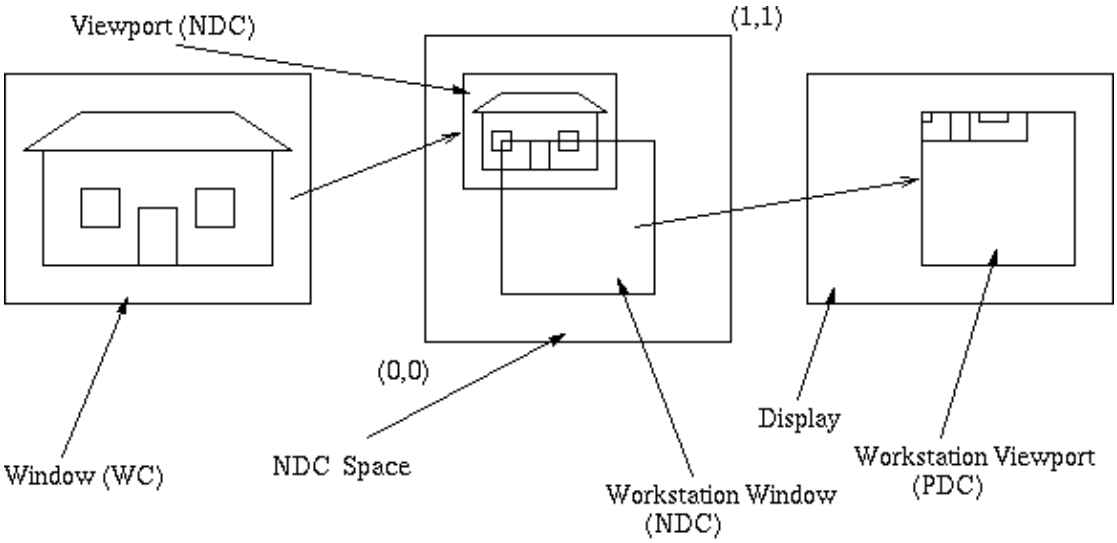



**Fig. 5.2: Viewing transformation**

The process that converts object coordinates in WCS to normalized device coordinate is called window–to– viewport mapping or normalization transformation. The process that maps normalized device coordinates to Physical Device Co-ordinates (PDC) / image coordinates is called work, station transformation, which is essentially a second window-to-viewport mapping., with a workstation window in the normalized device coordinate system and a workstation viewport in the device coordinate window in the normalized device coordinate system and a workstation viewport in the device coordinate system. Collectively, these two coordinate mapping operations are referred to as viewing transformation.

Workstation transformation is dependent on the resolution of the display device/frame buffer. When the whole display area of the virtual device is mapped to a physical

device that does not have a 1/1 aspect ratio, it may be mapped to a square sub-region (see fig. 5.2) so as to avoid introducing unwanted geometric distortion.

Along with the convenience and flexibility of using a window to specify a localized view comes the need for clipping, since objects in the scene may be completely inside the window, completely outside the window, or 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.

Note that clipping may occur in the world coordinate or viewing coordinate space, where the window is used to clip the objects; it may also occur in the normalized device coordinate space, where the viewport/workstation window is used to clip. In either case we refer to the window or the viewport/workstation window as the clipping window.

### 5.4 Point Clipping

Point clipping is essentially the evaluation of the following inequalities:

$$x_{min} \leq x \leq x_{max} \quad \text{and} \quad y_{min} \leq y \leq y_{max}$$

Where $x_{min}$, $x_{max}$, $y_{min}$ and $y_{max}$ define the clipping window. A point (x,y) is considered inside the window when the inequalities all evaluate to true.

### 5.5 Line Clipping

Lines that do not intersect the clipping window are either completely inside the window or completely outside the window. On the other hand, a line that intersects the clipping window is divided by the intersection point(s) into segments that are either inside or outside the window. The following algorithms provide efficient ways to decide the spatial relationship between an arbitrary line and the clipping window and to find intersection point(s).

#### 5.5.1 The Cohen-Sutherland Algorithm

In this algorithm we divide the line clipping process into two phases: (1) identify those lines which intersect the clipping window and so need to be clipped and (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 $(x_1,y_1,)$ to $(x_2,y_2)$ satisfies any one of the following four inequalities:

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

3. Clipping candidate – the line is in neither category 1 and 2

In fig. 5.3, line $l_1$ is in category 1 (visible); lines $l_2$ and $l_3$ are in category 2 (not visible) ; and lines $l_4$ and $l_5$ are in category 3 (clipping candidate).

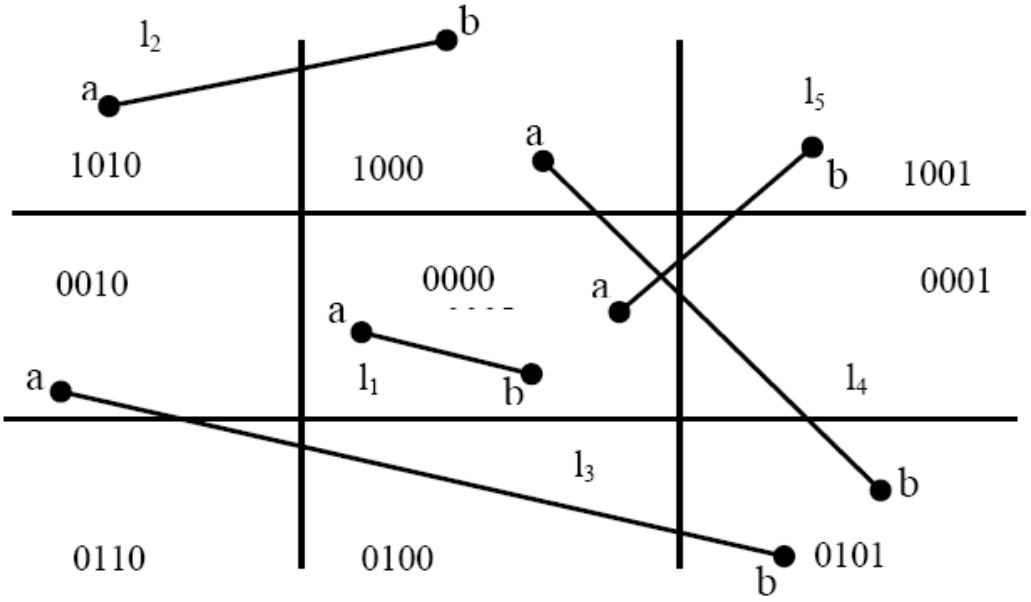
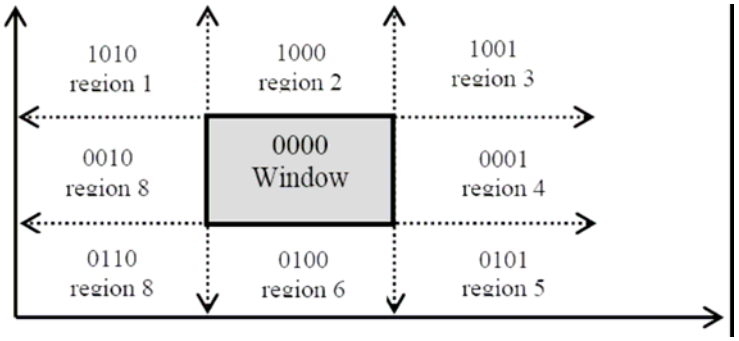



**Figure 5.3**

The algorithm employs an efficient procedure for finding the category of a line. It proceeds in two steps:

1. 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 plane the endpoint lies in



Starting from the leftmost bit, each bit of the code is set to true (1) or false (0) according to the scheme

Bit 1 ≡ endpoint is above the window = sign ($y - y_{max}$)

Bit 2 ≡ endpoint is below the window = sign ($y_{min}$-y)

Bit 3 ≡ endpoint is to the right of the window = sign ($x - x_{max}$)

Bit 4 ≡ endpoint is to the left of the window = sign ($x_{min} - x$)

We use the convention that sign (a)=1 if a is positive, 0 otherwise. Of course, a point with code 0000 is inside the window.

2. The line is visible if both region codes are 0000, and not visible if the bitwise logical AND of the codes is not 0000, and a candidate for clipping if the bitwise logical AND of the region codes is 0000.

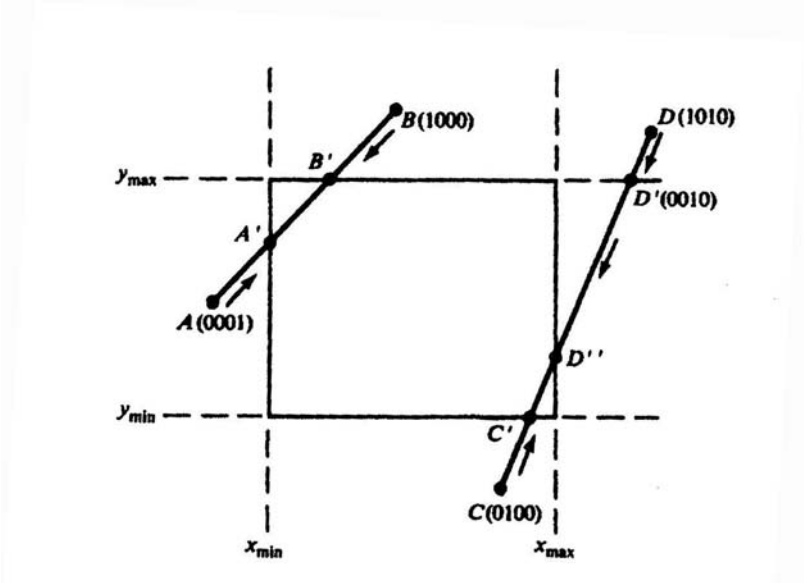



**Figure 5.4**

For a line in category 3 we proceed to find the intersection point of the line with one of the boundaries of the clipping window, or to be exact, with the infinite extension of one of the boundaries. We choose an endpoint of the line, say ($x_1$, $y_1$), that is outside the window, i.e., whose region code is not 0000. We then select an extended boundary line by observing that those boundary lines that are candidates for intersection are of ones for which the chosen endpoint must be "pushed across" so as to change a "1" in its code to a "0" (see Fig. 5.4). This means:

If bit 1 is 1, intersect with line $y = y_{max}$.

If bit 2 is 1, intersect with line $y = y_{min}$.

If bit 3 is 1, intersect with line $x = x_{max}$.

If bit 4 is 1, intersect with line $y = y_{min}$.

Consider line CD is Fig.5.4. If endpoint C is chosen, then the bottom boundary line $y=y_{min}$ is selected for computing intersection. On the other hand, if endpoint D is chosen, then either the top boundary line $y=y_{max}$ or the right boundary line $x = x_{max}$ is used. The coordinates of the intersection point are

$$\begin{cases} x_i = x_{\min} or\ x_{\max} \\ y_i = y_1 + m(x_i - x_1) \end{cases}$$ if the boundary line is vertical

or

$$\begin{cases} x_i = x_1 + (y_{i-} y_1)/m \\ y_i = y_{\min} or\ y_{\max} \end{cases}$$ if the boundary line is horizontal

where $m = (y_2 – y_1)/(x_2 – x_1)$ is the slope of the line.

Now we replace endpoint $(x_1, y_1)$ with the intersection point $(x_1, y_i)$ effectively eliminating the portion of the original line that is on the outside of the selected window boundary. The new endpoint is then assigned an updated region code and the clipped line re-categorized and handled in the same way. This iterative process terminates when we finally reach a clipped line that belongs to either category 1 (visible) or category 2 (not visible).

### 5.5.2 Midpoint Subdivision

An alternative way to process a line in category 3 is based on binary search. The line is divided at its midpoint into two shorter line segments. The clipping categories of the two new line segments are then determined by their region codes. Each segment in category 3 is divided again into shorter segments and categorized. This bisection and categorization process continues until each line segment that spans across a window boundary (hence encompasses an intersection point) reaches a threshold for line size

and all other segments are either in category 1 (visible) or in category 2 (invisible) . The midpoint coordinates $(x_{m,}y_{m)})$ of a line joining $(x_{1,}y_1)$ and $(x_{2,}y_2)$ are given by

$$x_m = \frac{x_1 + x_2}{2} \qquad y_m = \frac{y_1 + y_2}{2}$$

The example in fig.5.5 illustrates how midpoint subdivision is used to zoom in onto the two intersection points $I_i$ and $I_2$ with 10 bisections. The process continues until we reach two line segments that are, say, pixel – sized . i.e. mapped to one single pixel each in the image space. If the maximum number of pixels in a line is M , this method will yield a pixel-sized line segment in N subdivisions, where $2^N$ = M or N = $\log_2$ M. For instance, when M = 1024 we need at most N = $\log_2$ 1024 = 10 subdivisions.

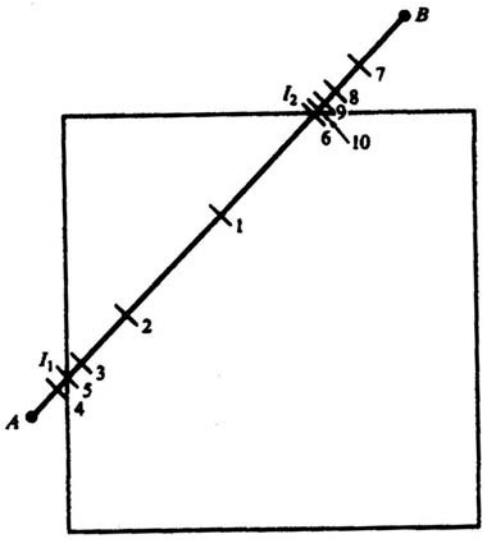



**Figure 5.5**

**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**

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

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

**Problem 2** 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) a viewport that is the entire normalized device screen and (b) a viewport that has lower left corner at (0, 0) and upper right corner $\left(\frac{1}{2}, \frac{1}{2}\right)$.

**Answer**

From problem 1, we need to identify the appropriate parameters

(a) The window parameters are $wx_{min} = 1$, $wy_{max} = 1$, and $wy_{max} = 5$. The viewport parameters are $vx_{min} = 0$, $vx_{max} = 1$, $vy_{min = 0, \text{ and }} vy_{max} = 1$. Then $s_x = \frac{1}{2}$, $s_y = \frac{1}{4}$, and

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

(b) The window parameters are the same as in (a). The viewport parameters are not $vx_{min} = 0$, $vx_{max} = \frac{1}{2}$, $vy_{min} = 0$, $vy_{max} = \frac{1}{2}$. The $s_{x =} \frac{1}{4}$, $s_y = \frac{1}{8}$, and

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

**Problem 3** Find a normalization transformation from the window whose lower left corner is at (0,0) and upper right corner is at (4,3) onto the normalized device screen so that aspect ratios are preserved.

**Answer**

The window aspect ratio is $a_w = \frac{4}{3}$. Unless otherwise indicated , we shall choose a viewport that is as large as possible with respect to the normalized device screen. To this end, we choose the x extent from 0 to 1 and the y extent from 0 to $\frac{3}{4}$. So

$$a_v = \frac{1}{\frac{3}{4}} = \frac{4}{3}$$

As in Prob. 1 , with parameters $wx_{min} = 0$, $wx_{max} = 4$, $wy_{max} = 0$, $wy_{min} = 3$, and $vx_{max} = 1$, $vy_{min} = 0$, $vy_{max} = \frac{3}{4}$. ¾

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

**5.6 Area Clipping**

An algorithm that clips a polygon must deal with many different cases. The case is particularly note worthy in that the concave polygon is clipped into two separate polygons. All in all, the task of clipping seems rather complex. Each edge of the polygon must be tested against each edge of the clip rectangle; new edges must be added, and existing edges must be discarded, retained, or divided. Multiple polygons may result from clipping a single polygon. We need an organized way to deal with all these cases.

The following example illustrate a simple case of polygon clipping (figure5.6).

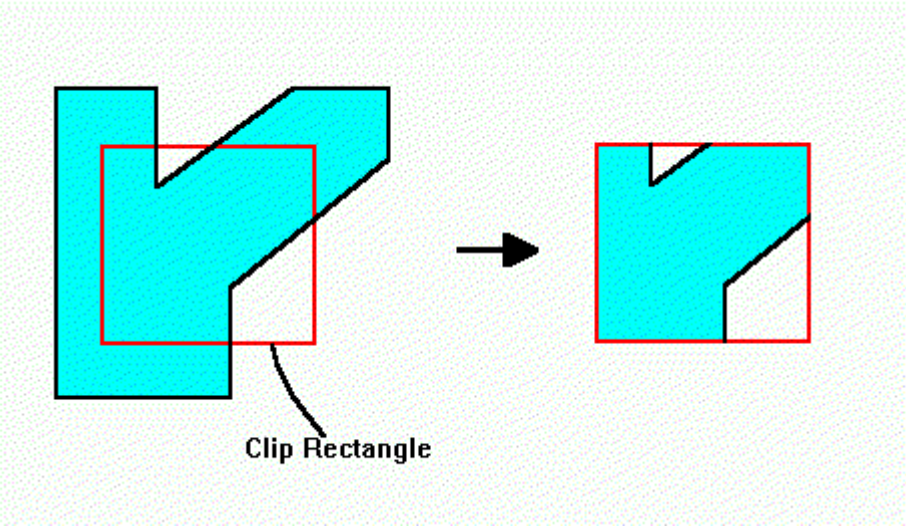



**Figure 5.6**

Sutherland and Hodgman's polygon-clipping algorithm uses a divide-and-conquer strategy: It solves a series of simple and identical problems that, when combined, solve the overall problem. The simple problem is to clip a polygon against a single infinite clip edge. Four clip edges, each defining one boundary of the clip rectangle, successively clip a polygon against a clip rectangle.

Note the difference between this strategy for a polygon and the Cohen-Sutherland algorithm for clipping a line: The polygon clipper clips against four edges in succession, whereas the line clipper tests the outcode to see which edge is crossed, and clips only when necessary.

### 5.6.1 Sutherland-Hodgman's polygon-clipping algorithm

- Polygons can be clipped against each edge of the window one at a time. Windows/edge intersections, if any, are easy to find since the X or Y coordinates are already known.
- Vertices which are kept after clipping against one window edge are saved for clipping against the remaining edges.
- Note that the number of vertices usually changes and will often increases.
- We are using the Divide and Conquer approach.

**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

**Case 1 : Wholly inside the visible region, save the endpoint (Fig. 5.7).**

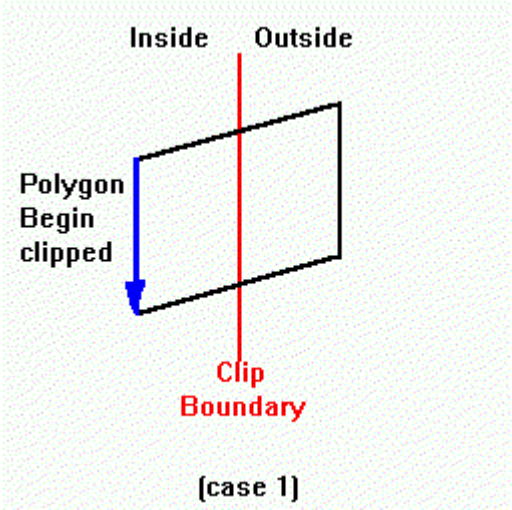



**Figure 5.7**

Case 2 : Exit visible region, save the intersection Figure 5.8

.

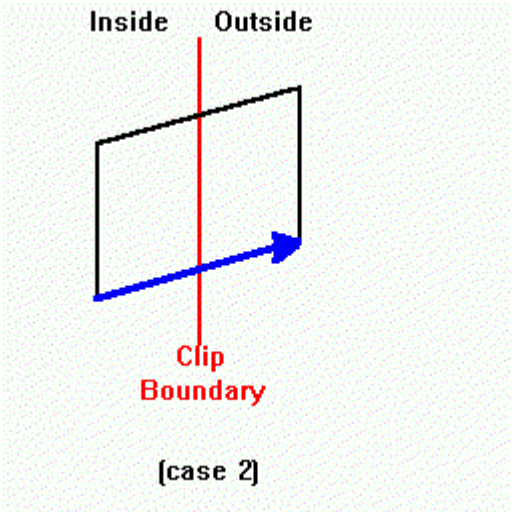



**Figure 5.8**

Case 3 : Wholly outside visible region, save nothing (Figure 5.9)

.

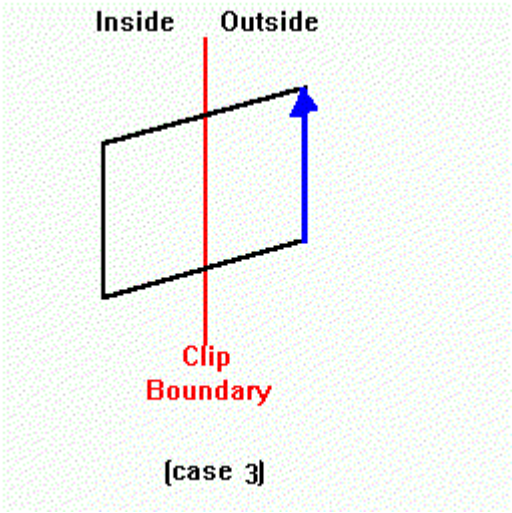



**Figure 5.9**

Case 4 : Enter visible region, save intersection and endpoint(Figure 5.10)

.

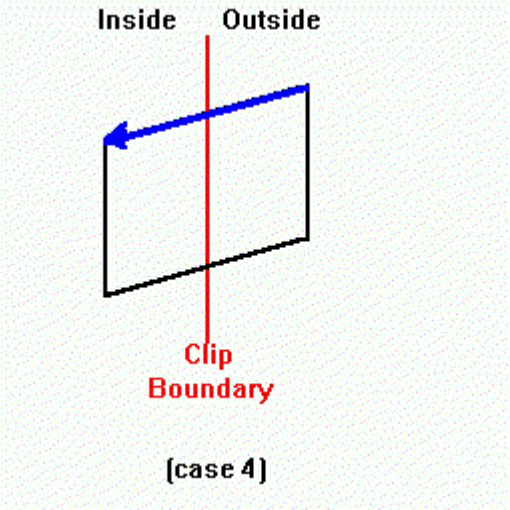



**Figure 5.10**

Because clipping against one edge is independent of all others, it is possible to arrange the clipping stages in a pipeline. The input polygon is clipped against one edge and any points that are kept are passed on as input to the next stage of the pipeline. This way four polygons can be at different stages of the clipping process simultaneously. This is often implemented in hardware.

### 5.7 Text Clipping

For text clipping various techniques are available in graphic packages. The clipping technique used will depend upon the application requirement and how characters produced. Normally we have many simple methods for clipping text. One method for processing character strings relative to a window boundary is to use the **all or none sting clipping** strategy shown in figure 5.11 below. If all of the string is discarded. This procedure is implemented by considering a boundary rectangle around the text pattern.

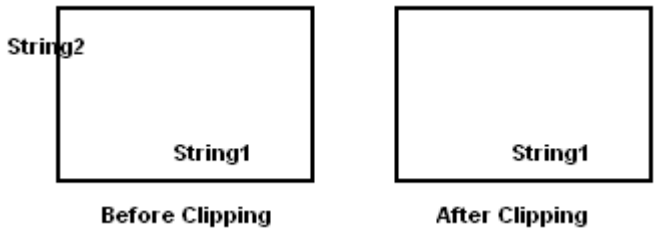



**Figure 5.11**

The boundary positions of the rectangle are the  compared to the window boundaries, and the string is rejected if there is any overlap. This method produces fastest text clipping.

### 5.8 Window-To-Viewport Coordinate Transformation

Once object descriptions have been transferred to the viewing reference frame, we choose the window extents in viewing coordinates and select the viewport limits in normalized coordinates. Object descriptions are then transferred to normalized device coordinates. We do this using a transformation that maintains the same relative placement of objects in normalized space as they had in viewing coordinates. If a coordinate position is at the center of the viewing window, for instance, it will be displayed at the center of the viewport.

Figure 5.12 illustrates the window-to-viewport mapping. A point at position (xw, yw) in the window is mapped into position (xv, yv) in the associated view-port. To maintain the same relative placement in the viewport as in the window, we require that

$$\frac{xv - xv_{min}}{xv_{max} - xv_{min}} = \frac{xw - xw_{min}}{xw_{max} - xw_{min}}$$

$$\frac{yv - yv_{min}}{yv_{max} - yv_{min}} = \frac{yw - yw_{min}}{yw_{max} - yw_{min}}$$

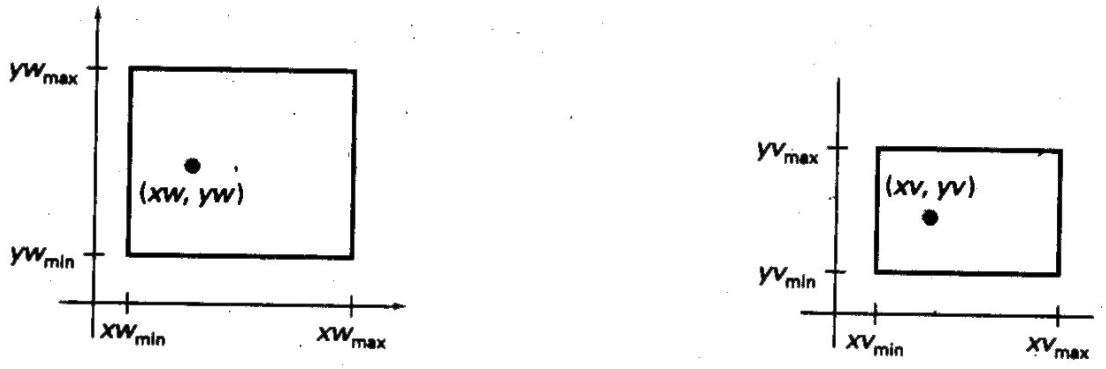



**Figure 5.12: A point at position (xm, yw) in a designated window is mapped to viewport coordinates**
**(xv, yv) so that relative positions in the two areas are the same.**

Solving these expressions for the viewport position (xv, yv), we have

$xv = xv_{min} + (xw - xw_{min})sx$

$yv = yv_{min} + (yw - yw_{min})sy$

where the scaling factors are

$$sx = \frac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}}$$

$$sy = \frac{yv_{max} - yv_{min}}{yw_{max} - yw_{min}}$$

Above equations can also be derived with a set of transformations that converts the window area into the viewport area. This conversion is performed with the following sequence of transformations:

1. Perform a scaling transformation using a fixed-point position of ($xw_{min}$, $yw_{min}$) that scales the window area to the size of the viewport.
2. Translate the scaled window area to the position of the viewport.

Relative proportions of objects are maintained if the scaling factors are the same (sx = sy). Otherwise, world objects will be stretched or contracted in either x or y direction when displayed on the output device.

Character strings can be handled in two ways when they are mapped to a viewport. The simplest mapping maintains a constant character size, even though the viewport

area may be enlarged or reduced relative to the window. This method would be employed when text is formed with standard character fonts that cannot be changed. In systems that allow .for changes in character size, string definitions can be windowed the same as other primitives. For characters formed with line segments, the mapping to the viewport can be carried out as a sequence of line transformations.

From normalized coordinates, object descriptions are mapped to the viewport display devices. Any number of output devices can be open in a particular application, and another window-to-viewport transformation can be performed for each open output device. This mapping, called the workstation transformation, is accomplished by selecting a window area in normalized space and a viewport area in the coordinates of the display device. With the workstation transformation, we gain some additional control over the positioning of parts of a scene on individual output devices. As illustrated in Fig. 5.13, we can use work station transformations to partition a view so that different parts of normalized space can be displayed on different output devices.

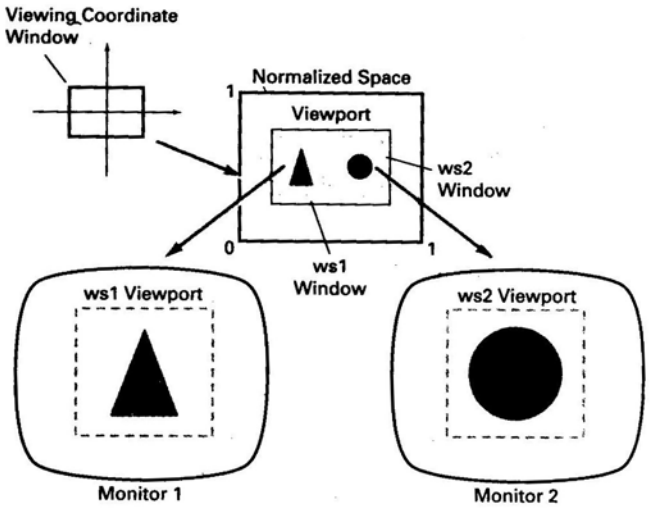



**Figure 5.13: Mapping selected parts of a scene in normalized coordinated to different video monitors with workstation transformations.**

### 5.9 Exterior and Interior Clipping

So far, we have considered only procedures for clipping a picture to the interior of a region by eliminating everything outside the clipping region. What is saved by these procedures is inside the region. In some cases, we want to do the reverse, that is, we want to clip a picture to the exterior of a specified region. The picture parts to be saved are those that are outside the region. This is referred to as exterior clipping.

A typical example of the application of exterior clipping is in multiple-window systems. To correctly display the screen windows, we often need to apply both internal and external clipping. Objects within a window are clipped to the interior of that window. When other higher-priority windows overlap these objects, the objects are also clipped to the exterior of the overlapping windows.

Exterior clipping is used also in other applications that require overlapping pictures. Examples here include the design of page layouts in advertising or publishing applications or for adding labels or design patterns to a picture. The technique can also be used for combining graphs, maps, or schematics. For these applications, we can use exterior clipping to provide a space for an insert into a larger picture.

Procedures for clipping objects to the interior of concave polygon windows can also make use of external clipping. A figure 5.14 shows a line $\overline{P_1P_2}$ that is to be clipped to the interior of a concave window with vertices $V_1V_2V_3V_4V_5$. Line $\overline{P_1P_2}$ can be clipped in two passes: (1) First, $\overline{P_1P_2}$ is clipped to the interior of the convex polygon $V_1V_2V_3V_4$ to yield the clipped segment $\overline{P'_1P'_2}$. (2) Then an external clip of $\overline{P'_1P'_2}$ is performed against the convex polygon $V_1V_5V_4$ to yield the final clipped line segment $\overline{P''_1P'_2}$.

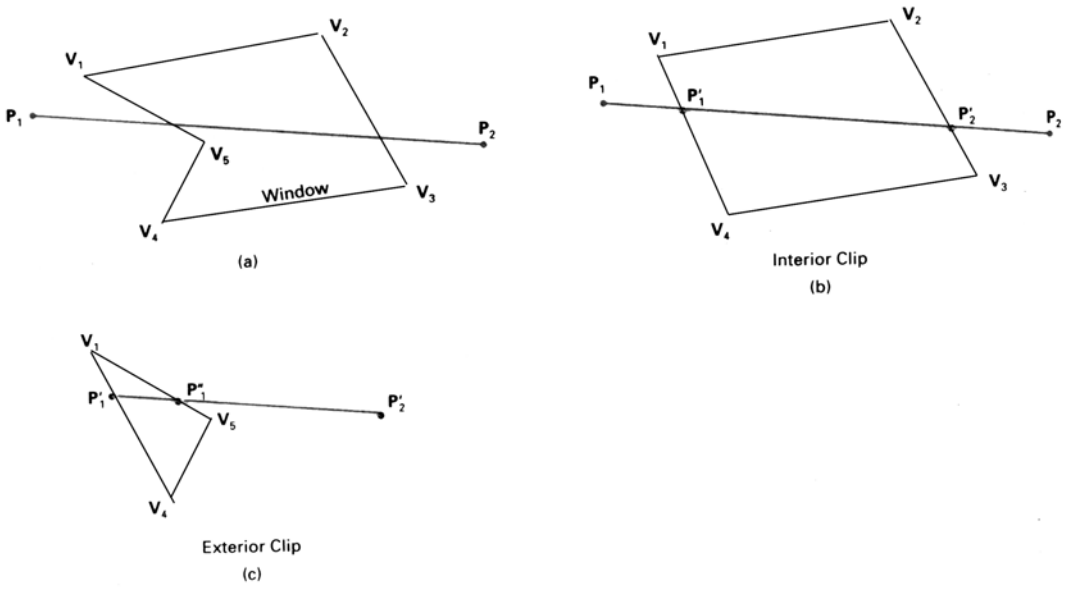



**Figure 5.14: Clipping line $\overline{P_1P_2}$ to the interior of a concave polygon with vertices $V_1V_2V_3V_4V_5$ (a), using convex polygons $V_1V_2V_3V_4$ (b) and $V_1V_5V_4$ (c), to produce the clipped line $P''_1P'_2$**

## 5.10 Summary

- The viewing transformation is used to transform a part of graphical scene to a portion of the screen.
- The Normalised Device Coordinates describe the coordinates system natural
- A window is a rectangular surrounding the object or a part of it that we wish to draw on the screen.
- A viewport beings the window content to the screen, so, that window coordinates are based on world coordinates system.
- Multiplying the basic matrix transformations can do complex transformations
- Shear transformation distorts an object by scaling one coordinate using the other in such a way as if the object were composed of internal layers that has been caused to slide over each other.

## 5.11 Key Words

Window, Viewport, Clipping

## 5.12 Self Assessment Questions (SAQ)

1. Explain The Cohen-Sutherland Line-Clipping Algorithm.
2. Explain some uses of clipping.
3. What is clipping?
4. Explain clipping in a raster world?
5. What do you mean by viewing transformation?
6. Find the workstation transformation that maps the normalized device screen onto a physical device whose x extent is 0 to 199 and y extent is 0 to 639 where origin is located at the lower left corner.

## 5.13 References/Suggested Readings


1. Computer Graphics (C Version), Donald Hearn and M. Pauline Baker, Prentice Hall,
2. Computer Graphics , Second Edition , by Pradeep K. Bhatia , I.K .International Publisher.
3. Advanced Animation and Rendering Techniques, Theory and Practice, Alan Watt and Mark Watt , ACM Press/Addison-Wesley
4. Graphics Gems I-V, various authors, Academic Press
5. Computer Graphics, Plastok, TMH
6. Principles of Interactive Computer Graphics, Newman, TMH