UNIT – 3

Two– dimensional Geometric Transformations, Viewing & Clipping

2-D geometric Transformations: Translation, Rotation, Scaling, Reflection & Shear (with example)
Viewing Pipeline, Window-to-Viewport transformation
Point Clipping
Line clipping (without program)
Cohen Sutherland line clipping algorithm
Polygon Clipping(without program) Text clipping, Exterior Clipping

Geometric Transformation :

Definition: "Transformation means change in coordinate representation of an object made by changing orientation, size or shape of an object."

Types of Geometric Transformation :

- Translation
- Rotation
- Scaling
- Reflection
- Shear

1. Translation :

"A translation is applied to an object by repositioning it along a straight-line path from one coordinate location to another."

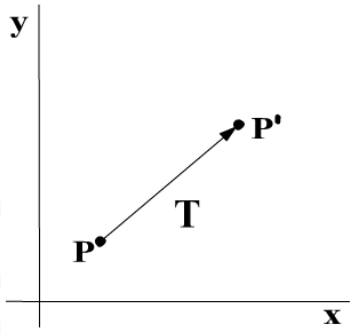


Fig (1) : Translating a point from position P to position P' with translate vector T.

We translate the object by changing the original co-ordinates P (x , y) with the translation of t_x and t_y. Thus, the new co-ordinates P' (x' , y')are

$$x' = x + t_x$$
$$y' = y + t_y,$$

The pair (t_x , t_y) is called a translation vector or shift vector.

We can express the translation equations as a single matrix equation by using column vectors to represent coordinate positions and the translation vector:

$$P = \begin{pmatrix} X \\ Y \end{pmatrix} \qquad P' = \begin{pmatrix} X' \\ Y' \end{pmatrix} \qquad T = \begin{pmatrix} t_x \\ t_y \end{pmatrix}$$

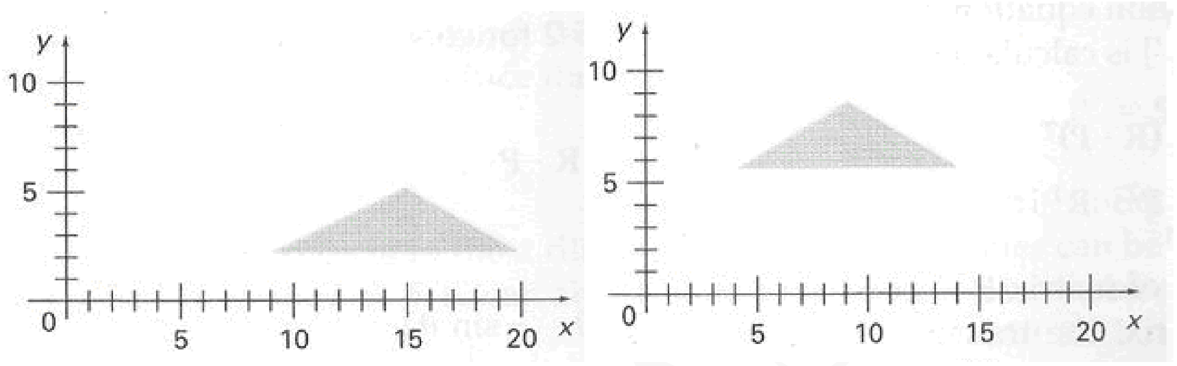
This allows us to write the two-dimensional translation equations in the matrix form:

$$P' = P + T$$

Similar to above, instead of column matrix, we can also consider the row matrix given by

$P = [\ X\ ,\ Y\ ]$ $P' = [\ X'\ ,\ Y'\ ]$ and $T = [\ t_x\ ,\ t_y\ ]$

The example of translation with the value of $t_x = -5.50$ and $t_y = 3.75$, is as below:



(a) (b)

Fig (2) : Moving a polygon from position (a) to position (b) with the translation vector (-5.50 , 3.75).

2. Rotation :

"A two-dimensional rotation is applied to an object by repositioning it along a circular path in the xy plane."

For rotating the object, we specify a rotation angle θ and the position (x_r, y_r) of the rotation point (or pivot point) about which the object is to be rotated (Fig(3)).

Positive values for the rotation angle define counterclockwise rotations about the pivot point, as in Fig (3)., and negative values rotate objects in the clockwise direction.
This transformation can also be described as a rotation about a rotation axis that is perpendicular to the xy plane and passes through the pivot point.

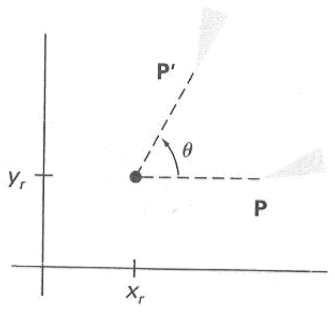


Fig (3) : Rotation of an object through angle θ about the pivot point (x_r , y_r)

We first determine the transformation equations for rotation of a point position P when the pivot point is at the coordinate origin. The angular and coordinate relationships of the original and transformed point positions are shown in Fig. (4).

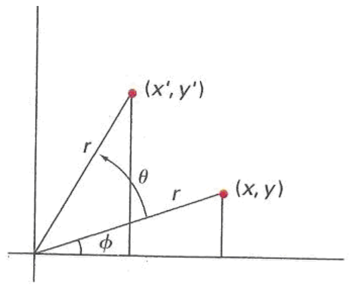


Fig (4) : Rotation of a point from position *(x,* y) to position *(x',* y ') through an angle θ relative to the coordinate origin. The original angular displacement of the point from the *x* axis is Ø.

In this Fig 4, r is the constant distance of the point from the origin, angle Ø is the original angular position of the point from the horizontal, and θ is the rotation angle. Using standard trigonometric identities, we can express the transformed coordinates in terms of angles θ and Ø as

$$x' = r\cos(\varnothing + \theta) = r\cos\varnothing\cos\theta + r\sin\varnothing\sin\theta$$
$$y' = r\sin(\varnothing + \theta) = r\cos\varnothing\sin\theta + r\sin\varnothing\cos\theta$$

------------(1)

The original coordinates of the point in polar coordinates are

$$x = r\cos\varnothing$$
$$y = r\sin\varnothing$$

------------(2)

Using eq (1) and (2), we get

$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$

------------(3)

We can write the rotation equations in the matrix form:

$$P' = R \cdot P$$ ------------(4)

Where the rotation matrix is

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

When coordinate positions are represented as row vectors instead of column vectors, the matrix product in rotation equation (4) is transposed so that the transformed row coordinate vector [x' , y'] is calculated as

$$P'^T = (R \cdot P)^T \qquad \text{------------(5)}$$
$$P'^T = R^T \cdot P^T \qquad \text{------------(6)}$$

Rotation of a point about an arbitrary pivot point.

$$x' = x_r + (x - x_r) \cos\theta - (y - y_r) \sin\theta$$
$$y' = y_r + (\underline{x - x_r}) \sin\theta + (y - y_r) \cos\theta$$

------------(1)

<u>3. Scaling :</u>

"A scaling transformation alters the size of an object." This operation can be carried out for polygons by multiplying the coordinate values (x, y) of each vertex by scaling factors s_x and s_y to produce the transformed coordinates (x' , y'):

$$x' = x \cdot s_x \quad , \quad y' = y \cdot s_y$$

Scaling factor s_x scales objects in the x direction, while s_y scales in the y direction. The transformation equations can also be written in the matrix form:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

$$P' = S \cdot P$$

Scaling related to a fix point (x_f, y_f) for coordinate (x , y), the scale coordinate can be represented as following:

$$x' = x_f + (x - x_f) \cdot s_x$$
$$y' = y_f + (y - y_f) \cdot s_y$$

Types of scaling:
Uniform and Differential :

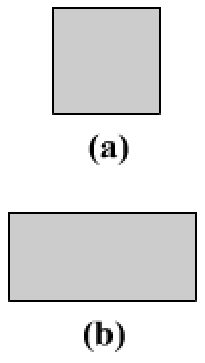
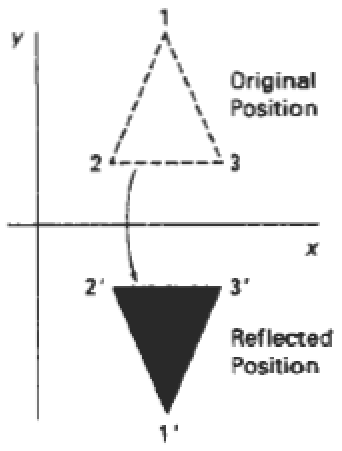


Fig (4) : Turning a square (a) into a rectangle (b) with scaling factors $S_x = 2$ and $S_y = 1$.

4. Reflection:

- A reflection is a transformation that produces a mirror image of an object.
- The mirror image for a two-dimensional reflection is generated relative to an axis of reflection by rotating the object 180° about the reflection axis.
- We can choose an axis of reflection in the xy plane or perpendicular to the xy plane. When the reflection axis is a line in the xy plane, the rotation path about this axis is in a plane perpendicular to the xy plane.
- For reflection axes that are perpendicular to the xy plane, the rotation path is in the xy plane. Following are examples of some common reflections.

Reflection about the line y = 0, the x axis.



Reflection

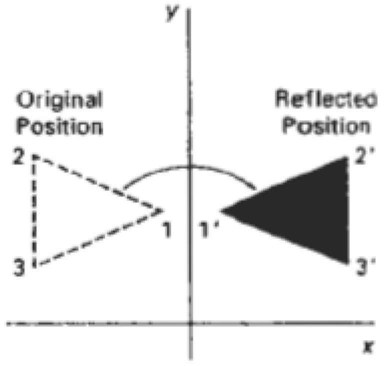
$$\begin{vmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

Fig (5) : Reflection of an object about the x axis.

This transformation keeps x value same but flips the y values of coordinate positions.

Reflection about line x = 0, the y axis :

A reflection about the y axis flips x coordinates while keeping y coordinates the same. The transformation matrix is :



Reflection

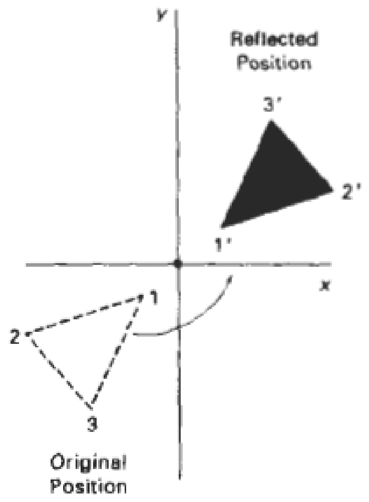
$$\begin{vmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

Fig (6) : Reflection of an object about the y axis.

This transformation keeps y value same but flips the x values.

Reflection along axis perpendicular to (x , y) plain and passes to the origin.

We flip both the x and y coordinates of a point by reflecting relative to an axis that is perpendicular to the xy plane and that passes through the coordinate origin. This transformation, referred to as a reflection relative to the coordinate origin, has the matrix representation:



Reflection

$$\begin{vmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

Fig (7) : Reflection of an object relative to an axis perpendicular to the xy plane and passing through the coordinate origin.

An example of reflection about the origin is shown in Fig. The reflection matrix is the rotation matrix R(θ) with θ = 180°. We are simply rotating the object in the xy plane half a revolution about the origin.

Reflection along diagonal line y = x

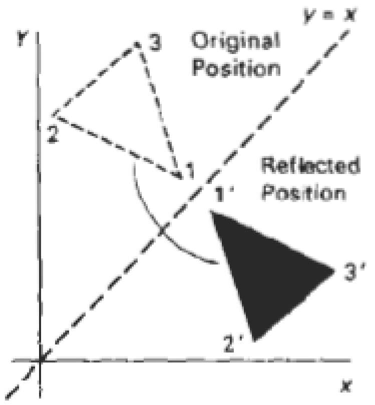


Fig (8) : Reflection of an object with respect to the line y = *x.*

This rotation is carried out in three steps:

Step 1 : Rotation line y = x by 45° degree angle in clock-wise direction which rotates line y = x on the x axis.
Step 2 : Perform reflection with respect to x axis.
Step 3 : Rotate back the line y = x by 45 degree in anti-clock wise direction.

Then final reflection transformation matrix will be.

Reflection

$$\begin{vmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

Reflection along diagonal line y = -x

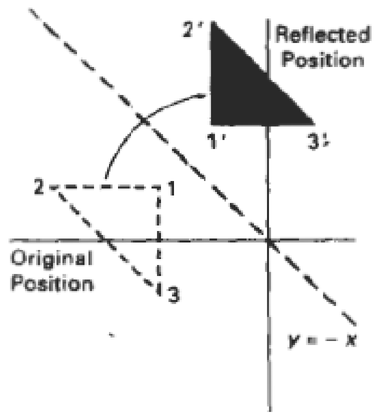


Fig (9) : Reflection with respect to the line y = -x.

This rotation is carried out in three steps:

Step 1 : Rotation line y = -x by 45 degree angle in clock-wise direction.
Step 2 : perform reflection with respect to y axis.
Step 3 : Rotate back the line y = -x by 45 degree in anti-clock wise direction.

Then final reflection transformation matrix will be.

Reflection

$$\begin{vmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

5. Shear

A transformation that distorts the shape of an object such that the transformed shape appears as if the object where composed of internal layers that had been caused to slide over each other is called a shear.

Two common shearing transformations are :

1. Shift Coordinate x values and
2. Shift Coordinate y values.

An x-direction shear relative to the x axis is produced with the transformation matrix

$$\begin{bmatrix} 1 & Sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

And transform co-ordinates are obtain by using following equations.

$$x' = x + sh_x \cdot y$$
$$y' = y$$

Any real number can be assigned to the shear parameter sh_x. A coordinate position (x, y) is then shifted horizontally by an amount proportional to its distance (y value) from the x axis (y = 0).

For example, Setting sh_x to 2, changes the square in Fig. into a parallelogram. Negative values for sh_x shift coordinate positions to the left.

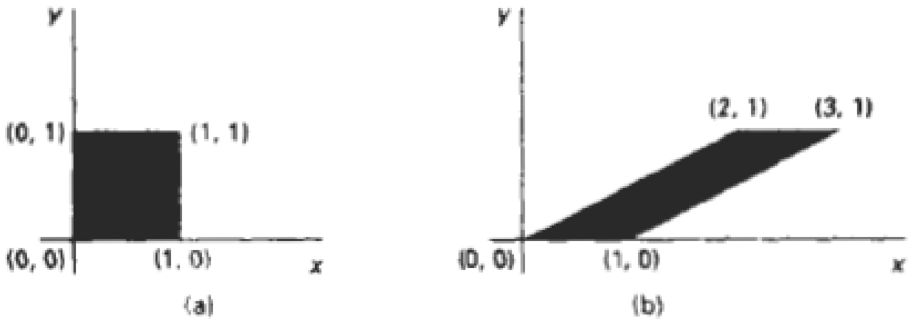


Fig (10)

We can generate x-direction shears relative to other reference lines with

$$\begin{bmatrix} 1 & sh_x & -sh_x \cdot y_{ref} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

with coordinate positions transformed as

$$x' = x + sh_x \cdot (y - y_{ref})$$
$$y' = y$$

An example of this shearing transformation is given In Fig. for a shear parameter value of 1 /2 relative to the line $y_{ref} = -1$.

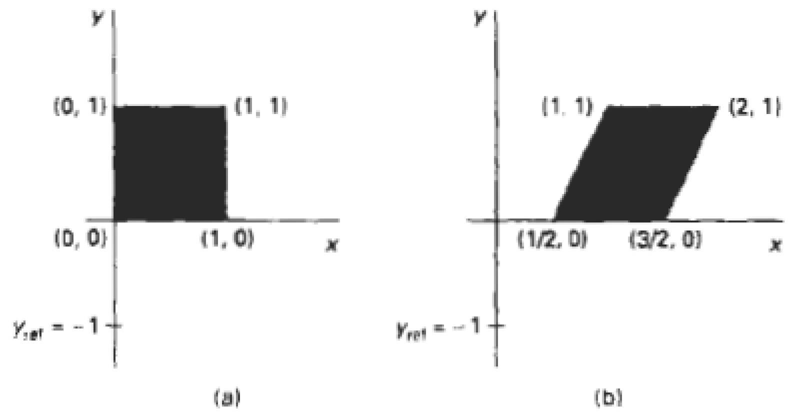


Fig (11)

A y-direction shear relative to the line x = x_{ref} is generated with the transformation matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ sh_y & 1 & -sh_y \cdot x_{ref} \\ 0 & 0 & 1 \end{bmatrix}$$

which generates transformed coordinate positions

$$x' = x$$
$$y' = sh_y \cdot (x - x_{ref}) + y$$

This transformation shifts coordinate position vertically by an amount proportional to its distance from the reference line x = x_{ref}. Figure illustrates the conversion of a square into a parallelogram with shy = 1/2 and x_{ref} = -1.

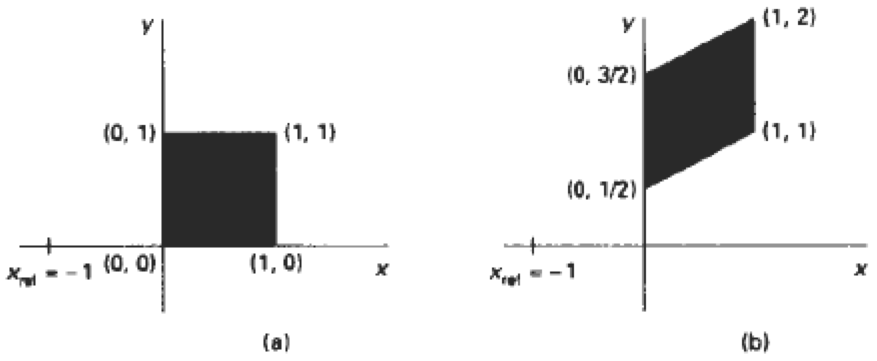


Fig (12)

VIEWING PIPELINE

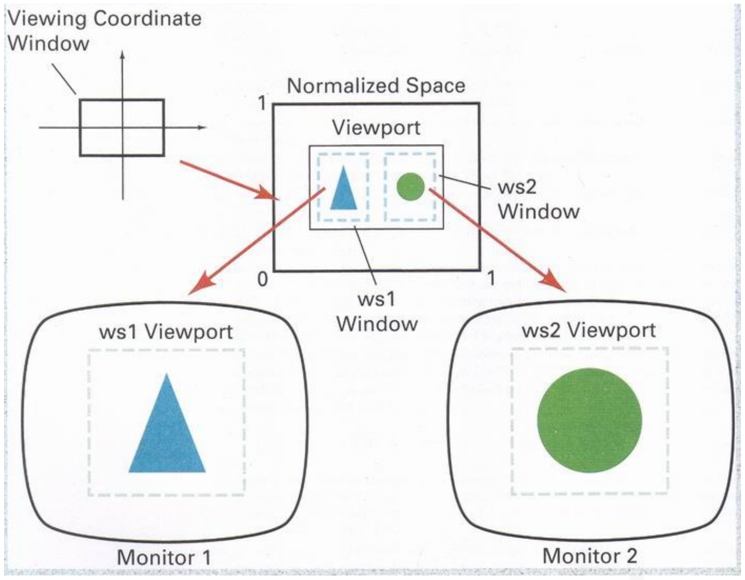


Fig (13)

- A world-coordinate area selected for display is called a window.
- An area on a display device to which a window is mapped is called a viewport.
  The window defines *what* is to be viewed;
  the viewport defines *where* it is to be displayed.
- The mapping of a part of a world-coordinate scene to device coordinates is referred to as a viewing transformation/.

Sometimes the two-dimensional viewing transformation is simply referred to as the *window-to-viewport transformation* or the *windowing transformation.*

Figure (14) shows the mapping of a picture section that falls within a rectangular window onto a designated & angular viewport.

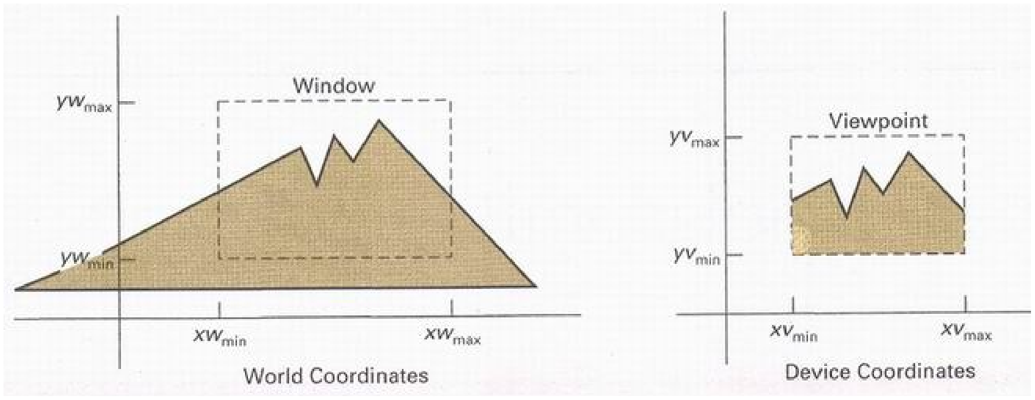


Fig (14) : A viewing transformation using standard rectangular the window and viewport.

The viewing transformation is done by different steps as follows:

1. Construct the scene in world co-ordinate by using the output primitive and attributes.

2. Obtain the orientation for window and define window in viewing co-ordinate.

3. Transform the world co-ordinate to viewing co-ordinate.

4. Define the normalized co-ordinate.

5. The viewport is transferred to the device which we require.

The viewing pipeline can be represented as below:

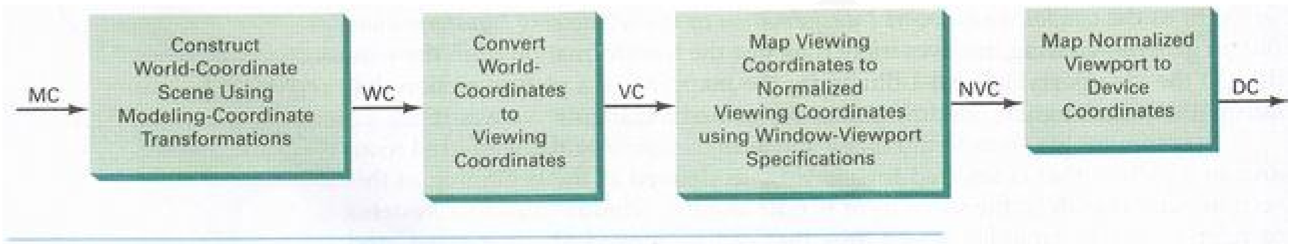

MC
Construct World-Coordinate Scene Using Modeling-Coordinate Transformations
WC
Convert World-Coordinates to Viewing Coordinates
VC
Map Viewing Coordinates to Normalized Viewing Coordinates using Window-Viewport Specifications
NVC
Map Normalized Viewport to Device Coordinates
DC

Figure 6-2
The two-dimensional viewing-transformation pipeline.

Figure (15) shows a rotated viewing-coordinate reference frame and the mapping to normalized coordinates.

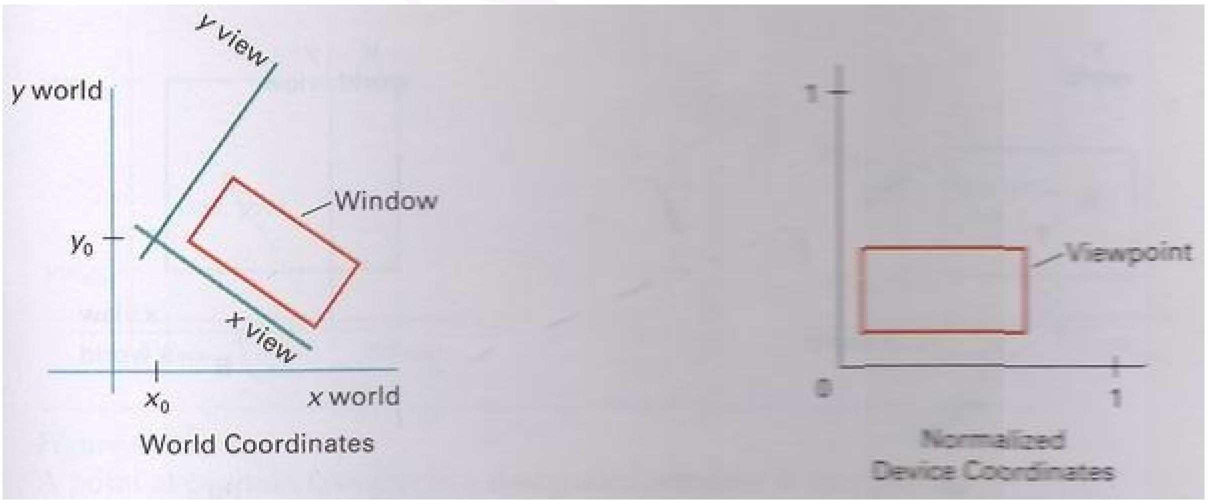
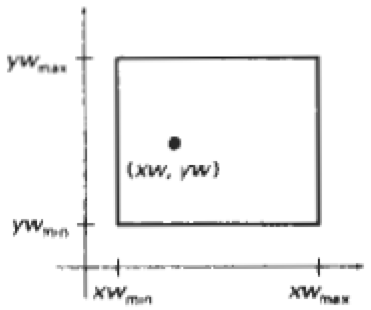


Fig (15) : Setting up a rotated world window in viewing coordinates and the corresponding normalized-coordinate viewport.

WINDOW-TO-VIEWPORT COORDINATE TRANSFORMATION



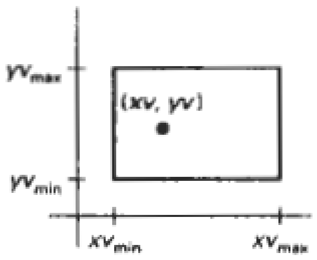


Fig (16) : A point at position (*xw,yw*) in a designated window is mapped to viewport coordinates (*xv, yv*) so that relative positions in the two areas are the same.

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 (Fig. 15). Object descriptions are then transferred to normalized device 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 (16) shows the window-to-viewport mapping. A point at position *(xw , yw)* in the window is mapped into position *(xv,* yv) in the associated viewport. To maintain the same relative placement in the viewport as in the window, we require that

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

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

------ (1)

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

$$xv = xv_{min} + (xw - xw_{min})\, sx$$
$$yv = yv_{min} + (yw - yw_{min})\, sy$$

------ (2)

Where the scaling factors are

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

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

------ (3)

Equations (2) 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 the *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.

CLIPPING OPERATIONS

Any procedure that identifies those portions of a picture that are either inside or outside of a specified region of space is referred to as a **clipping algorithm**, or **simply clipping**.

The region against which an object is to clipped is called a **clip window.**

The clipping operation can be performed on different objects. Thus it can be divided as:

- Point Clipping
- Line Clipping (straight-line segments)
- Area Clipping (polygons)
- Curve Clipping
- Text Clipping

Applications of clipping

- Include extracting part of a defined scene for viewing
- Identifying visible surfaces in three-dimensiona1 views
- Antialiasing line *seg*ments or object boundaries
- Creating objects using solid-modeling procedures
- Displaying a multi window environment
- Drawing and painting operations that allow parts of a picture to be selected for copying, moving, erasing, or duplicating.

POINT CLIPPING

Assuming that the clip window is a rectangle in standard position, we save a point P = (*x* , *y*) for display if the following inequalities are satisfied:

$$xw_{\min} <= x <= xw_{\max}$$
$$yw_{\min} <= y <= yw_{\max}$$

Where the *edges of* the clip window ($xw_{\min}$, $xw_{\max}$, $yw_{\min}$, $yw_{\max}$) can be either the world-coordinate window boundaries or viewport boundaries. If any one of these four inequalities is not satisfied, the point is clipped (not saved for display).

Although point clipping is applied less often than line or polygon clipping, some applications may require a point-clipping procedure. For example, point clipping can be applied to scenes involving explosions or sea foam that are modeled with particles (points) distributed in some region of the scene.

LINE CLIPPING

Figure shows possible relationships between line positions and a standard rectangular clipping region. First, we can test a given line segment to determine whether it lies completely inside the clipping window. If it does not, we try to determine whether it lies completely outside the window. Finally, if we cannot identify a line as completely inside or completely outside, we must perform intersection calculations with one or more clipping boundaries. We process lines through the "inside-outside'' tests by checking the line endpoints. A line with both endpoints inside all clipping boundaries, such as the line from P_1 to P_2 is saved. A line with both endpoints outside any one of the clip boundaries (line P_3P_4 in Fig) is outside the window. All other lines cross one or more clipping boundaries, and may require calculation of multiple intersection points. To minimize calculations, we try to devise clipping algorithms that can efficiently identify outside lines and reduce intersection calculations.

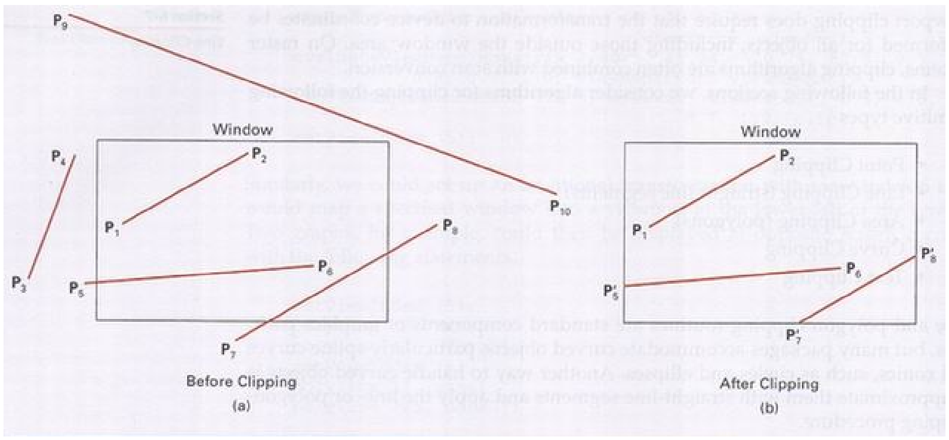


Fig (17) : Line clipping against a rectangular clip window.

For a line segment with endpoints (x_1, y_1) and (x_2, y_2) and one or both endpoints outside the clipping rectangle,

The clipping operation can be decided from the values

$$x = x_1 + u\ (x_2 - x_1)$$
$$y = y_1 + u\ (y_2 - y_1)$$

$$0 <= u <= 1$$

If the value of u is outside the range 0 to 1 then, line is outside.
If the value of u is within the range of 0 to 1 then, it crosses the clipping area.

Cohen-Sutherland Line Clipping

This is the oldest and the most popular line clipping procedure.

In this procedure both the endpoints of line are assigned a four digit binary code, known as Region Code. This code identifies the location of point with respect to the clip window.

In this,

Bit 4	Bit 3	Bit 2	Bit 1
Above	Below	Right	Left

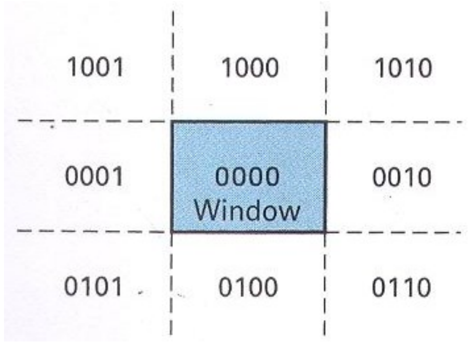


Fig (18) : Binary region codes assigned to line endpoints according to relative position with respect to the clipping rectangle.

If the value of bit is 1, it indicates the true condition e.g. 0101 means the point is Bottom and Left side of the Rectangle.

Bit values in the region code are determined by comparing end point coordinate values (x,y) to clip boundaries.

Bit 1 (L) : is set to 1 when $x < xw_{min}$
Bit 2 (R) : is set to 1 when $x > xw_{max}$
Bit 3 (B) : is set to 1 when $y < yw_{min}$
Bit 4 (A) : is set to 1 when $y > yw_{max}$

For languages where bit manipulation is possible, region code bit values are determined in following steps :

1. Calculate difference between endpoint coordinates and clipping boundaries.
2. Use the resultant sign bit of each difference to set corresponding value in region code.
   - Bit 1 (L) : is sign bit of $x - xw_{min}$
   - Bit 2 (R) : is sign bit of $xw_{max} - x$
   - Bit 3 (B) : is sign bit of $y - yw_{min}$
   - Bit 4 (A) : is sign bit of $yw_{max} - y$

Once the region code is established for all line end points we can conclude following:

1. If the region code of both endpoints is 0000, then the line is completely inside.
2. If both the endpoints contain a 1 in same bit position, then the line is completely outside. This can also be checked by logical AND operation with both region codes. If the result is not 0000 then, line is completely outside.
3. If the line is not completely inside or outside, then we have to find out the intersection point of line with clip window.

The intersection points can be calculated with the help of slope m of a line which has endpoints (x_1 , y_1) & (x_2 , y_2).

$$y = y_1 + m (x - x_1)$$

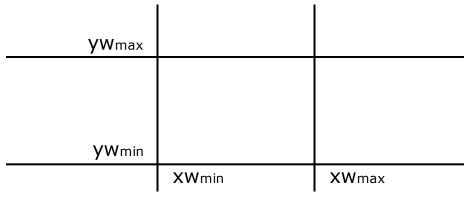
$$x = x_1 + (y - y_1) / m$$

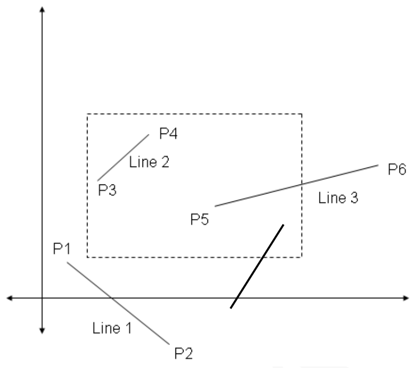
The equation of slope is given as,

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

The values of x will be xw_{min} or xw_{max} and that for y will by yw_{min} or yw_{max}

Bit values in the region code can be determined by comparing line endpoint with a clip window.





Line No.	End point 1	End point 1	Result of clipping
1	P1 (Outside)	P2(Outside)	Reject line
2	P3 (Inside)	P4 (Inside)	Save P3 and p4
3	P5 (Inside)	P6(Outside)	Save p5 and save intersection point p5’
4	P7 (Outside)	P8 (Inside)	Save intersection point p7’ and p8
5	P9 (outside) Line Intersect with clip window on both sides	P10(outside)	Save intersection points on both the sides of clip window P9’ and p10’

POLYGON CLIPPING

Polygon clipping operation is carried out by modifying line clipping algorithms. We can consider the polygon as a set of lines and thus clip the selected region. The only problem in this is that the closed area of the polygon cannot be decided.

A polygon boundary processed with a line clipper may be displayed as a series of unconnected line segments (Fig. 19), depending on the orientation of the polygon to the clipping window. What we really want to display is a bounded area after clipping, as in Fig. 20. For polygon clipping, we require an algorithm that will generate one or more closed areas that are then scan converted for the appropriate area fill. The output of a polygon clipper should be a sequence of vertices that defines the clipped polygon boundaries.

Example,

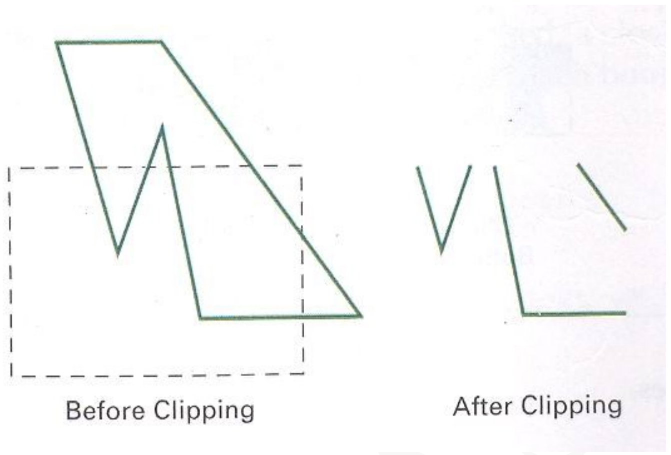


Fig (19) : Display of a polygon processed by a line-clipping algorithm.

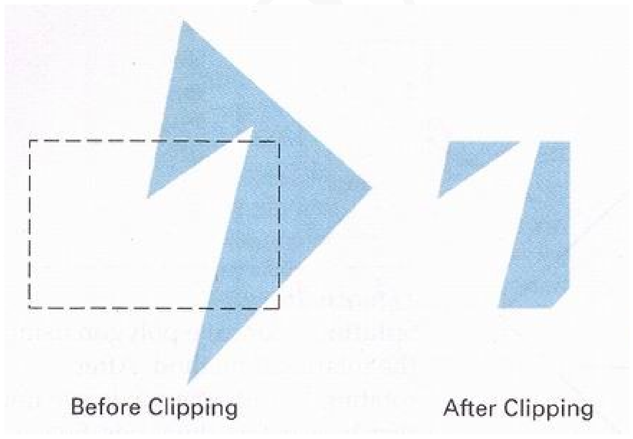


Fig (20) : Display of a correctly clipped polygon.

<u>Sutherland-Hodgeman Polygon Clipping</u>

We can correctly clip a polygon by processing the polygon boundary as a whole against each window edge. This could be accomplished by 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 then be successively passed to a right boundary clipper, a bottom boundary clipper, and a top boundary clipper, as in Fig. 21.

This procedure is divided into four steps: **left clipping, right clipping, bottom clipping** and **top clipping**.
The operation of polygon clipping by this method is as shown below:

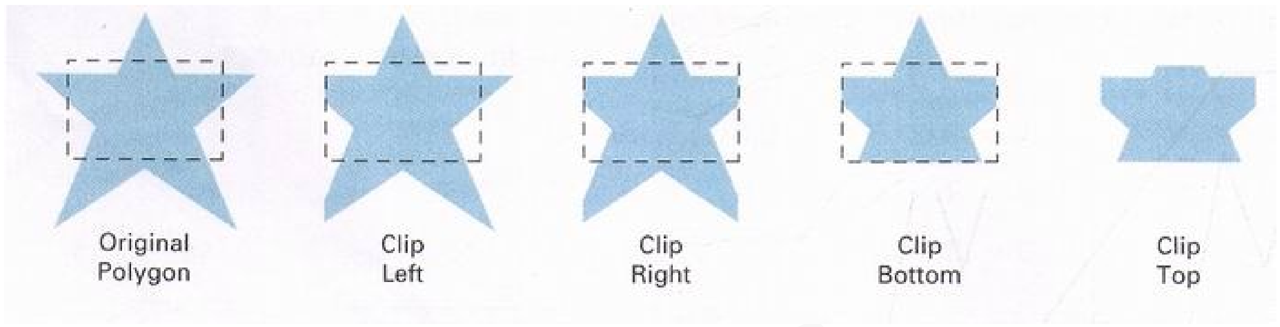


Fig (21) : Clipping a polygon against successive window boundaries.

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

There are four possible cases when processing vertices as each pair of adjacent polygon vertices is passed to a window boundary clipper, we make the following tests:

First Vertex	**Second Vertex**	**Save and Add to output vertex list**
Outside	Inside	the intersection point of the polygon edge with the window boundary and the second vertex .
Inside	Inside	only the second vertex
Inside	Outside	the edge intersection with the window boundary.
Outside	Outside	nothing

These four cases are illustrated in Fig. 22 for successive pairs of polygon vertices. Once all vertices have been processed for one clip window boundary, the output list of vertices is clipped against the next window boundary.

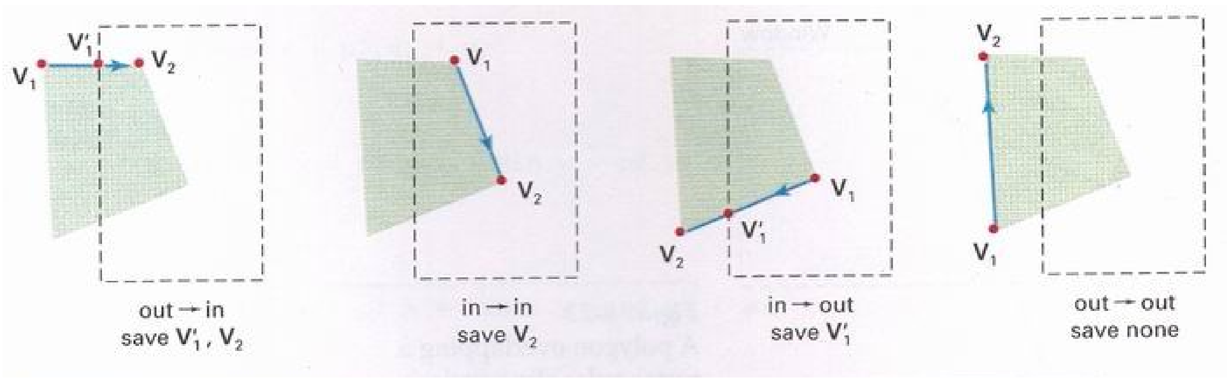
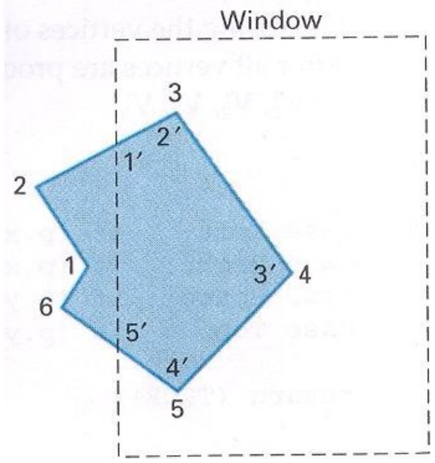


Fig (22) : Successive processing of pairs of polygon vertices against the left window boundaries

The example of polygon clipping below shows the original vertices and new vertices represented in terms of " ՝ "



We shows this method by processing the area in Fig above against the left window boundary. Vertices 1 and 2 are found to be on the outside of the boundary. Moving along to vertex 3, which is inside, we calculate the intersection and save both the intersection point and vertex 3. Vertices 4 and 5 are determined to be inside, and they also are saved. The sixth and final vertex is outside, so we find and save the intersection point. Using the five saved points, we would repeat the process for the next window boundary.

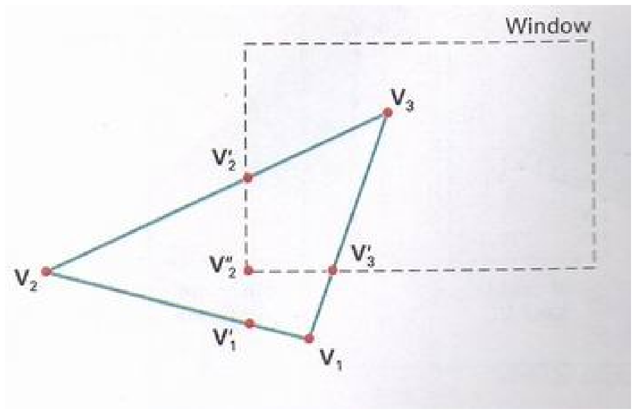


Fig (23) : A polygon overlapping a rectangle clip window.

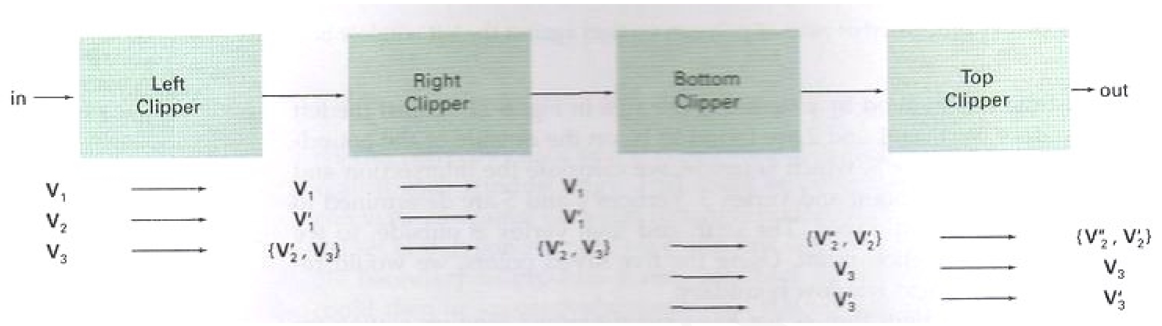


Fig (24) : Processing the vertices of the polygon in Fig. 23 through a boundary-clipping pipeline. After all vertices are processed through the pipeline, the vertex list for the clipped polygon is { V_2'' , V_2' , V_3 , V_3' }.

TEXT CLIPPING

There are several techniques that can be used to provide text clipping in a graphics package. The clipping technique used will depend on the methods used to generate characters and the requirements of a particular application.

1. All or none String
2. All or none Character
3. Clipping against individual Character

1. All or none String.

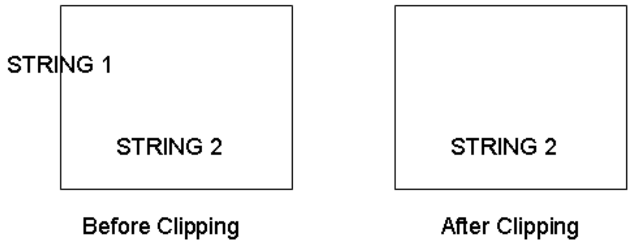


Fig (28) : Text clipping using a bounding rectangle about the entire string

If all of the string is inside a clip window, we save the complete string. Otherwise, the string is discarded. This procedure is implemented by considering a bounding rectangle around the text pattern. The boundary positions of the rectangle are then compared to the window boundaries, and the string is rejected if there is any overlap. This method produces the fastest text clipping.

1. All or none Character.

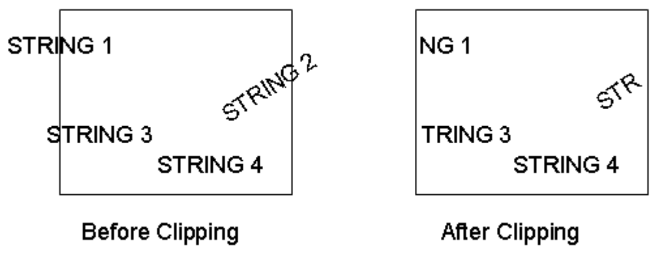


Fig (29) : Text clipping using a bounding rectangle about individual characters.

We discard only those characters that are not completely inside the window. In this case, the boundary limits of individual characters are compared to the window. Any character that either overlaps or is outside a window boundary is clipped.

3. Clipping against individual Character

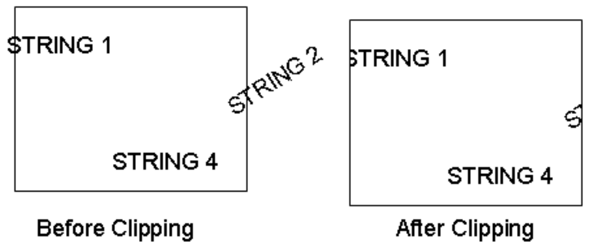


Fig (30) : Text clipping performed on the components of individual characters.

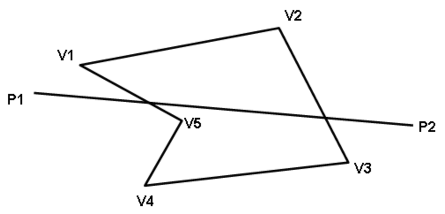
A final method for handling text clipping is to clip the components of individual characters. We now treat characters in the same way that we treated lines. If an individual character overlaps a clip window boundary, we clip off the parts of the character that are outside the window. Outline character fonts formed with line segments can be processed in this way using a line clipping algorithm. Characters defined with bit maps would be clipped by comparing the relative position of the individual pixels in the character grid patterns to the clipping boundaries.

EXTERIOR CLIPPING

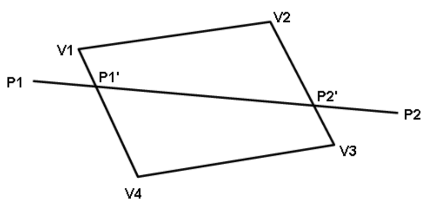
When we want to clip a picture to the exterior of a specified region. The picture parts to be saved are those that are outside the region. This is referred to as exterior clipping.

A typical example of the application of exterior clipping is in multiple window systems.

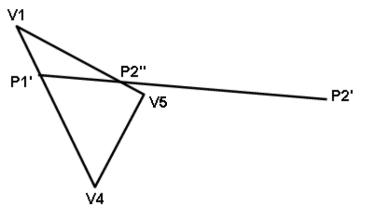
In this case, Objects within a window are clipped to the interior of that window. When other higher-priority windows overlap these objects, the objects are also clipped to the exterior of the overlapping windows.



WINDOW



INTERIOR



EXTERIOR