Module - 3

2-D Transformations

Transformation

Transformation are the operations applied to geometrical description of an object to change its position, orientation or size and are called geometric transformation

Basic geometric transformations:

Translation
Scaling
Rotation

Other transformations :-

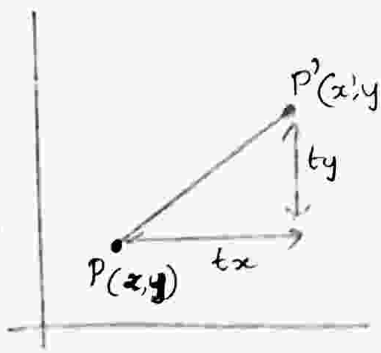
Reflection
Shear

Translation

It is applied to the object by representing it along a straight line path from one co-ordinate location to another.

A translation moves all points in an object along the same straight-line path to new positions.

The path is represented by a vector called the translation vector or shift vector.



Translation of point

In point P the coordinates are (x, y).

In point P' the coordinates are (x', y')

$x' = x + t_x$

$y' = y + t_y$

ie, $\boxed{P' = P + T}$

In matrix form,

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

Translation of polygon:-

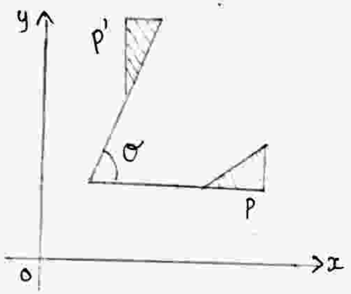
Here translation vectors ~~are~~ is added to all vertices.

Translation of circle/curve:-

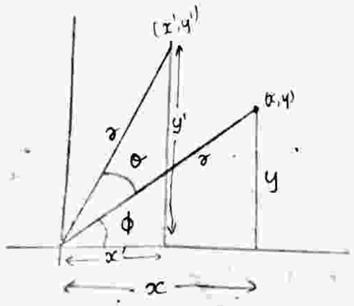
Here, move the center of original circle/curve, then redraw the circle using radius.

Rotation

It is applied ~~to~~ along a circular path in ~~an~~ an xy plane. For rotation, we specify an angle θ called rotation angle.



First position of Δ is P. We are rotating (move in a circular path in X-Y plane) then it moves to P' with coordinate (x', y'). It is moved with angle θ



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 ϕ

$$x' = r\cos(\phi + \theta)$$
$$= r\cos\phi \cos\theta - r\sin\phi\sin\theta$$

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

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

$\boxed{P' = R \cdot P}$ where,

$$R = \begin{pmatrix}\cos\theta & -\sin\theta \\ \sin\theta & \cos\theta\end{pmatrix}, \quad P = \begin{bmatrix} x \\ y\end{bmatrix}, \quad P' = \begin{bmatrix} x' \\ y'\end{bmatrix}$$

$$\begin{bmatrix} x' \\ y'\end{bmatrix} = \begin{bmatrix}\cos\theta & -\sin\theta \\ \sin\theta & \cos\theta\end{bmatrix} \times \begin{bmatrix} x \\ y\end{bmatrix}$$

$$= \begin{bmatrix} x\cos\theta + -y\sin\theta \\ x\sin\theta + y\cos\theta\end{bmatrix}$$

$$\begin{bmatrix} x' \\ y'\end{bmatrix} = \begin{bmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta\end{bmatrix}$$

Scaling

It alters the size of an object and involves two scale factors S_x and S_y for the x- and y- coordinates respectively

It is usually carried out by multiplying the vertex values by scaling factors.

$$x' = x \cdot S_x \qquad y' = y \cdot S_y$$

S_x – scales in x direction

S_y – scales in y direction

matrix form :-

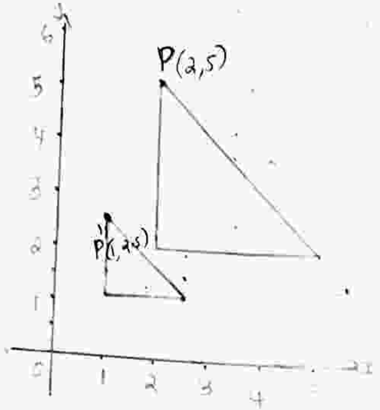
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

ie,

$$\boxed{P' = S \cdot P}$$

* If scaling factors $(S_x, S_y) < 1$
  → Reduce the size of object, moves obj closer to the coordinate origin
* If scaling factors $(S_x, S_y) > 1$
  → Increase the size of object, moves obj farther from the origin
* If scaling factors $(S_x, S_y) = 1$
  → Uniform scaling.

<u>Example</u>



- $P(2,5)$

  $S_x = 0.5$

  $S_y = 0.5$

⇒ As $(S_x, S_y) < 1$,
reduce the size of object & move the object close to the coordinate origin

$(x, y) = (2, 5)$

$x' = x \cdot S_x$ $\quad$ $y' = y \times S_y$

$= 2 \times 0.5 = \underline{1}$ $\quad$ $= 5 \times 0.5 = \underline{2.5}$

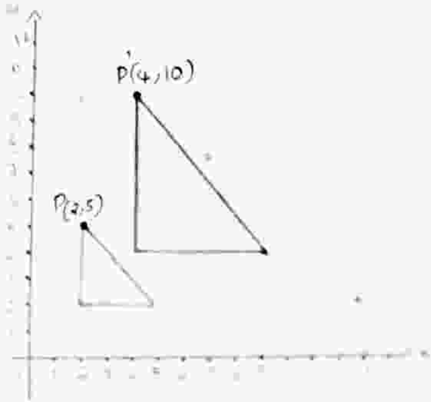
- $P(2,5)$,

  $S_x = 2$

  $S_y = 2$

⇒ As $(S_x, S_y) > 1$,

Increase the size of object & moves object farther from the origin.



$(x, y) = (2, 5)$

$x' = x \times S_x$ $\quad$ $y' = y \times S_y$

$= 2 \times 2$ $\quad$ $= 5 \times 2$

$= 4$ $\quad$ $= 10$

Note:

If $S_x \neq S_y$, then it is differential scaling

→ Change in size and shape happens

eg: square → Rectangle

$P(1, 3)$, $S_x = 2$ $\quad$ $S_y = 5$

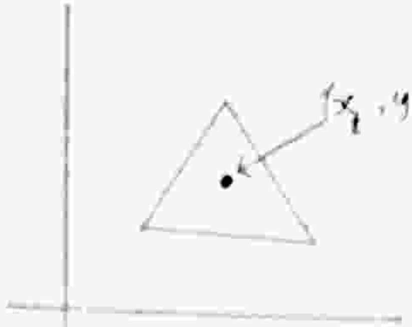
$x' = x \times S_x$ $\quad$ $y' = y \times S_y$

$= 1 \times 2$ $\quad$ $= 3 \times 5$

$= 2$ $\quad$ $= 15$

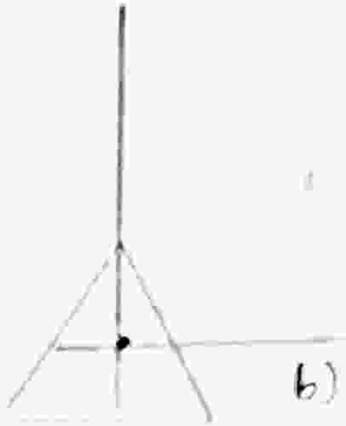
General <u>pivot point</u> Rotation

- Translate the object so that pivot-position is moved to the coordinate origin
- Rotate the object about the coordinate origin
- Translate the object so that the pivot point is returned to its original position



(a)

original position of object and pivot point



b)

Translation of object so that pivot point (x_t, y_t) is at origin

(c)

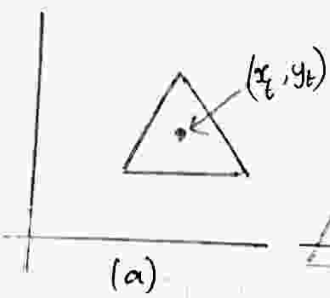
Rotation was about origin

(d)

Translation of the object so that the pivot point is returned to position (x_t, y_t)

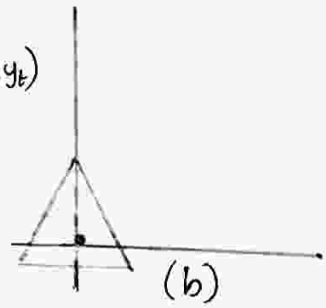
General fixed point scaling

- Translate object so that the fixed point coincides with the coordinate origin.
- Scale the object with respect to the coordinate origin.
- Use the inverse translation ~~tor~~ of step 1 to return the object to its original position.



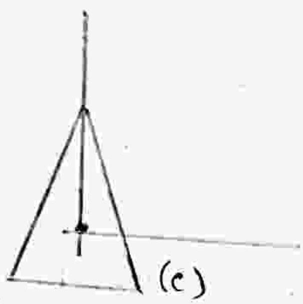
(a)

Original position of object and fixed point



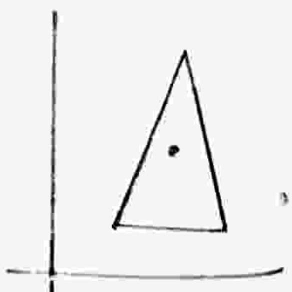
(b)

Translation of object so that fixed point (x_f, y_f) at origin



(c)

Scaling was about origin



(d)

Translation of the object so that the fixed point is returned to position

Composite Transformations

(A) Translations

If two successive translation vectors (t_{x1},t_{y1}) and (t_{x2},t_{y2}) are applied to a coordinate position P, the final transformed location P' is calculated as: -

$$P' = T(t_{x2},t_{y2}) \cdot \{T(t_{x1},t_{y1}) \cdot P\}$$
$$= \{T(t_{x2},t_{y2}) \cdot T(t_{x1},t_{y1})\} \cdot P$$

Where P and P' are represented as homogeneous-coordinate column vectors. We can verify this result by calculating the matrix product for the two associative groupings. Also, the composite transformation matrix for this sequence of transformations is: -

$$\begin{vmatrix} 1 & 0 & t_{x2} \\ 0 & 1 & t_{y2} \\ 0 & 0 & 1 \end{vmatrix} \cdot \begin{vmatrix} 1 & 0 & t_{x1} \\ 0 & 1 & t_{y1} \\ 0 & 0 & 1 \end{vmatrix} = \begin{vmatrix} 1 & 0 & t_{x1}+t_{x2} \\ 0 & 1 & t_{y1}+t_{y2} \\ 0 & 0 & 1 \end{vmatrix}$$

Or, $T(t_{x2},t_{y2}) \cdot T(t_{x1},t_{y1}) = T(t_{x1}+t_{x2},\ t_{y1}+t_{y2})$

Which demonstrate that two successive translations are additive

(B) Rotations

Two successive rotations applied to point P produce the transformed position: -

$$P' = R(\Theta_2) \cdot \{R(\Theta_1) \cdot P\}$$
$$= \{R(\Theta_2) \cdot R(\Theta_1)\} \cdot P$$

By multiplication the two rotation matrices, we can verify that two successive rotations are additive:

$$R(\Theta_2) \cdot R(\Theta_1) = R(\Theta_1 + \Theta_2)$$

So that the final rotated coordinates can be calculated with the composite rotation matrix as: -

$$P' = R(\Theta_1 + \Theta_2) \cdot P$$

(C) Scaling

Concatenating transformation matrices for two successive scaling operations produces the following composite scaling matrix: -

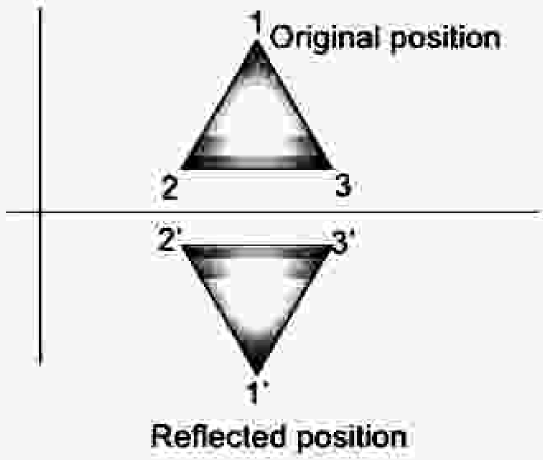
$$\begin{vmatrix} S_{x2} & 0 & 0 \\ 0 & S_{y2} & 0 \\ 0 & 0 & 1 \end{vmatrix} \cdot \begin{vmatrix} S_{x1} & 0 & 0 \\ 0 & S_{y1} & 0 \\ 0 & 0 & 1 \end{vmatrix} = \begin{vmatrix} S_{x1} \cdot S_{x2} & 0 & 0 \\ 0 & S_{y1} \cdot S_{y2} & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

Or, $S(S_{x2}, S_{y2}) \cdot S(S_{x1}, S_{y1}) = S(S_{x1} \cdot S_{x2}, S_{y1} \cdot S_{y2})$

The resulting matrix in this case indicates that successive scaling operations are multiplicative.

Other transformations

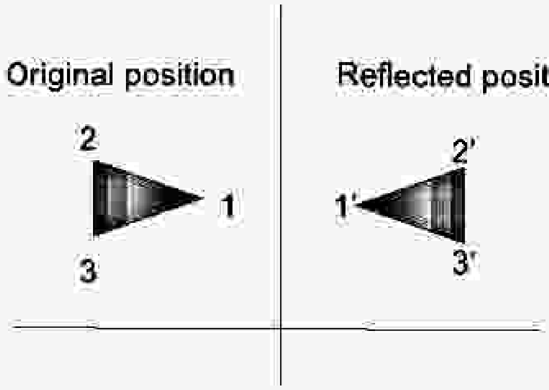
- **Reflection** is a transformation that produces a mirror image of an object. It is obtained by rotating the object by 180 deg about the reflection axis



Reflection about the line y=0, the X- axis , is accomplished with the transformation matrix

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

Reflection

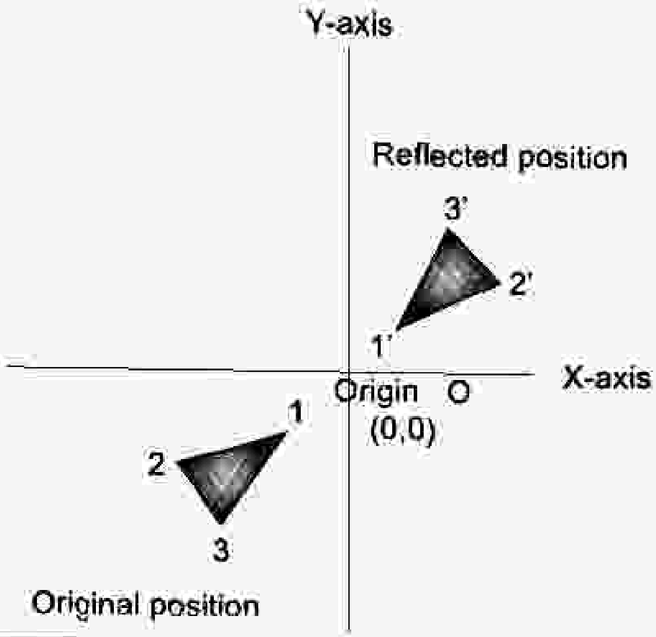


Reflection about the line x=0, the Y- axis , is accomplished with the transformation matrix

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

Reflection

Reflection of an object relative to an axis perpendicular to the xy plane and passing through the coordinate origin



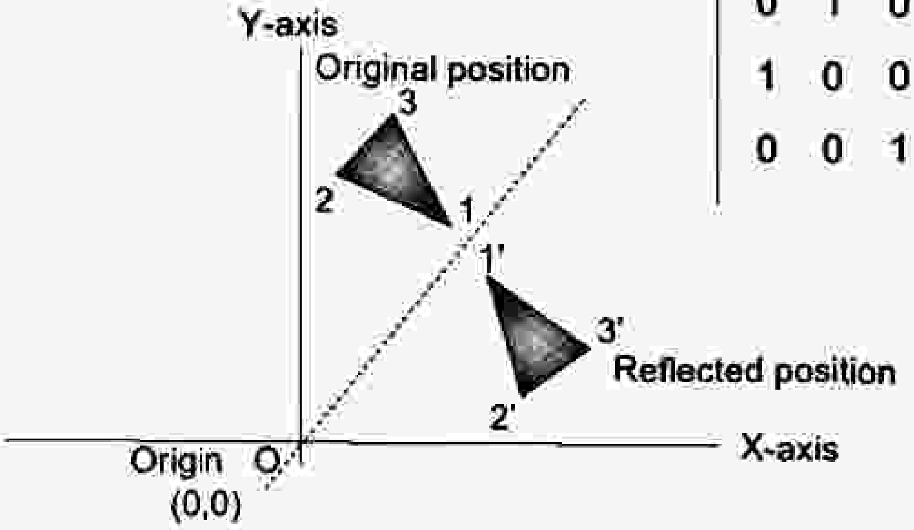
$$\begin{vmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

The above reflection matrix is the rotation matrix with angle=180 degree.

This can be generalized to any reflection point in the xy plane. This reflection is the same as a 180 degree rotation in the xy plane using the reflection point

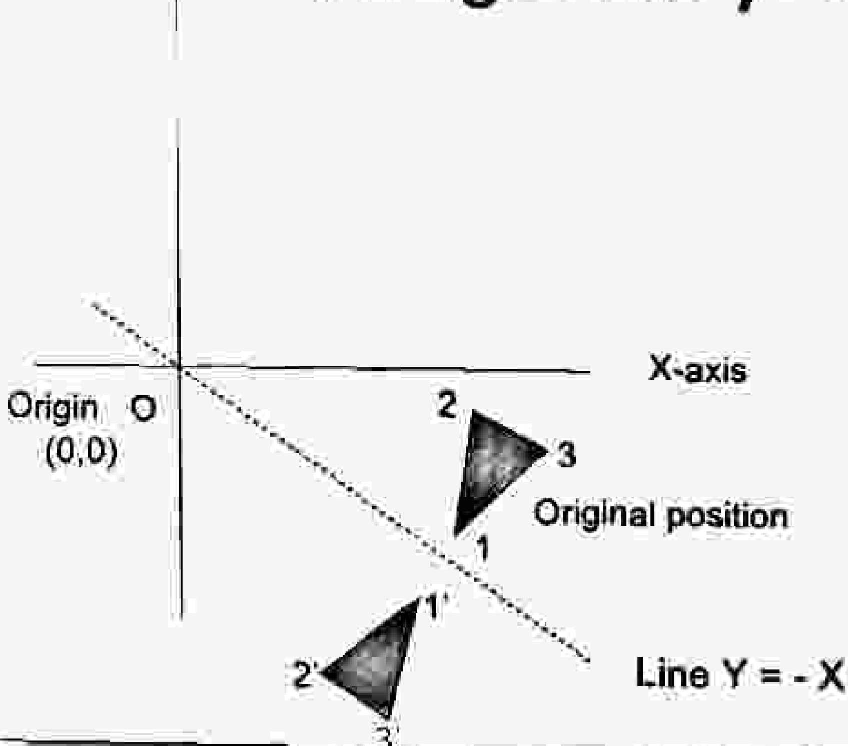
Reflection of an object w.r.t the straight line y=x

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

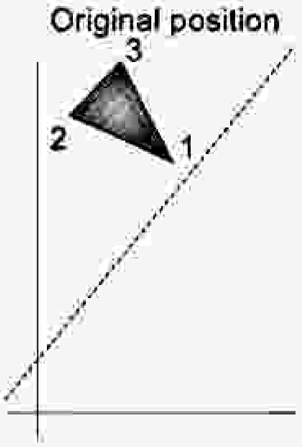


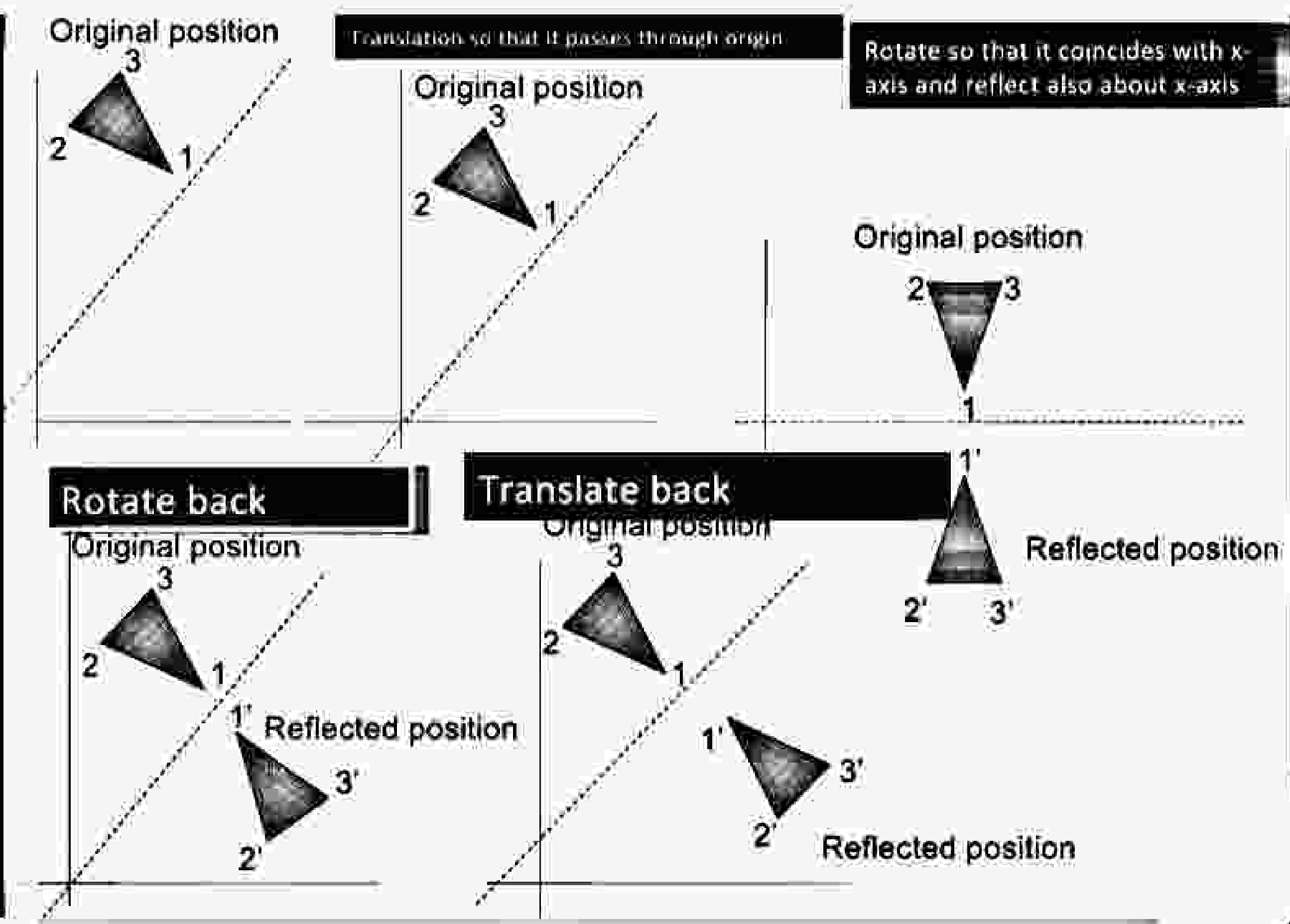
Reflection of an object w.r.t the straight line y=-x

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



Reflection of an arbitrary axis
y=mx+b

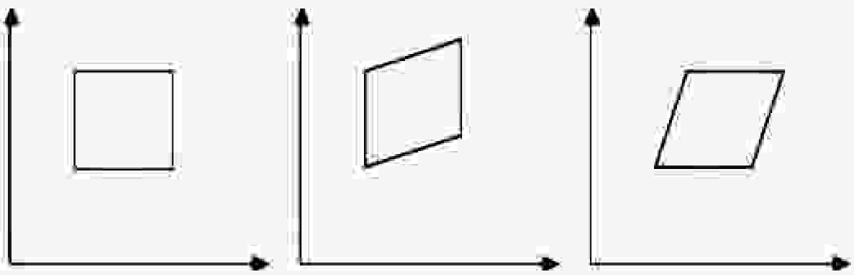




Shear Transformations

- Shear is a transformation that distorts the shape of an object such that the transformed shape appears as if the object were composed of internal layers that had been caused to slide over each other
- Two common shearing transformations are those that shift coordinate x values and those that shift y values

Shears

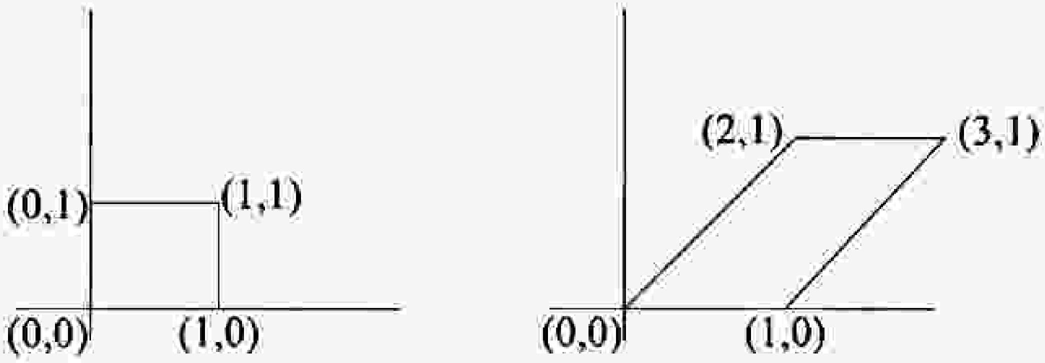


Original Data | y Shear | x Shear

$$\begin{bmatrix} 1 & 0 & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

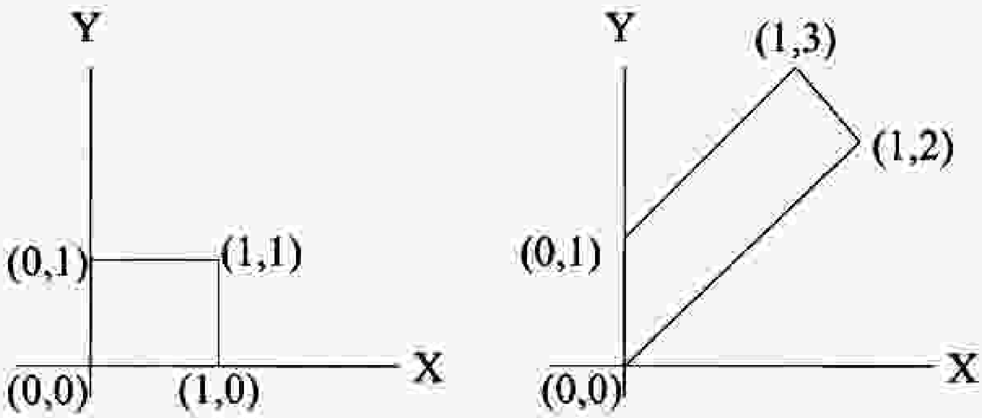
An X- direction Shear

For example, $Sh_x=2$



An Y- direction Shear

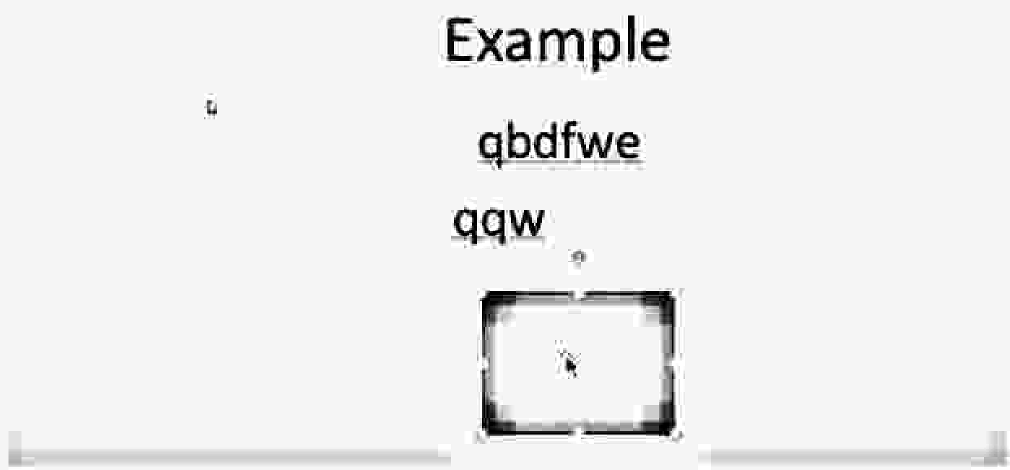
For example, $Sh_y=2$



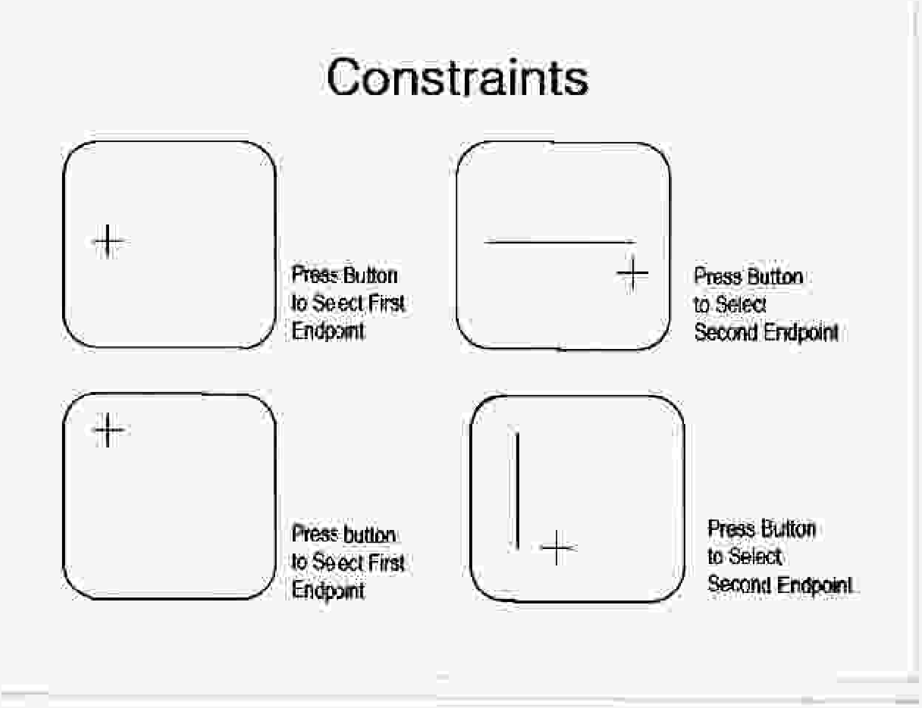
Interactive picture construction techniques, Computer Graphics

Interactive picture- construction methods are commonly used in variety of applications, including design and painting packages. These methods provide user with the capability to position objects, to constrain fig. to predefined orientations or alignments, to sketch fig., and to drag objects around the screen. Grids, gravity fields, and rubber band methods are used to aid in positioning and other picture construction operations. The several techniques used for interactive picture construction that are incorporated into graphics packages are:

(1) Basic positioning methods:- coordinate values supplied by locator input are often used with positioning methods to specify a location for displaying an object or a character string. Coordinate positions are selected interactively with a pointing device, usually by positioning the screen cursor.



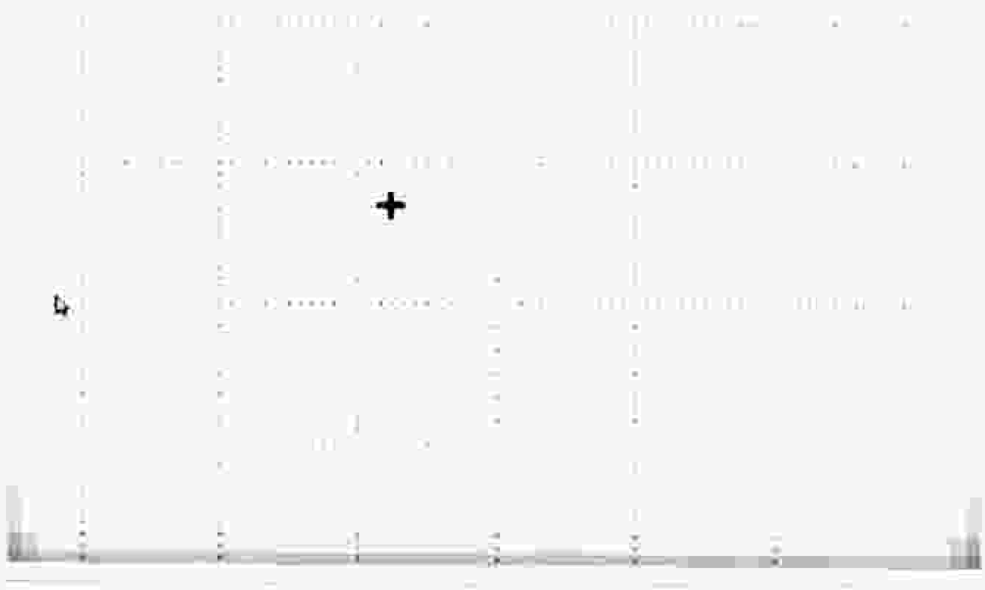
(2) constraints:-A constraint is a rule for altering input coordinates values to produce a specified orientation or alignment of the displayed coordinates. the most common constraint is a horizontal or vertical alignment of straight lines.



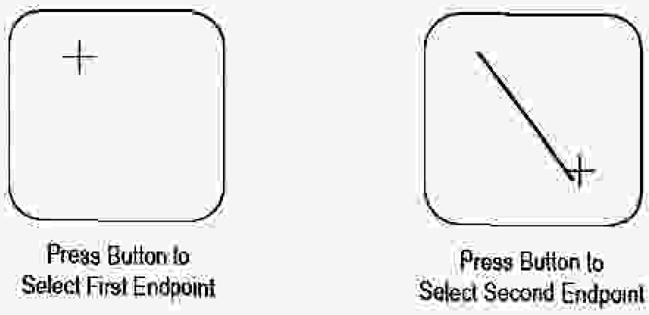
Example:

Vertical line Horizontal line

(3) Grids:- Another kind of constraint is a grid of rectangular lines displayed in some part of the screen area. When a grid is used, any input coordinate position is rounded to the nearest intersection of two grid lines.

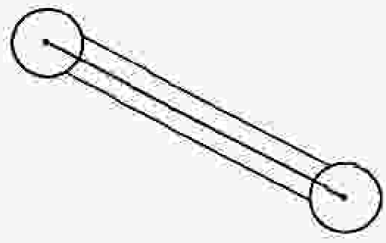


Grids



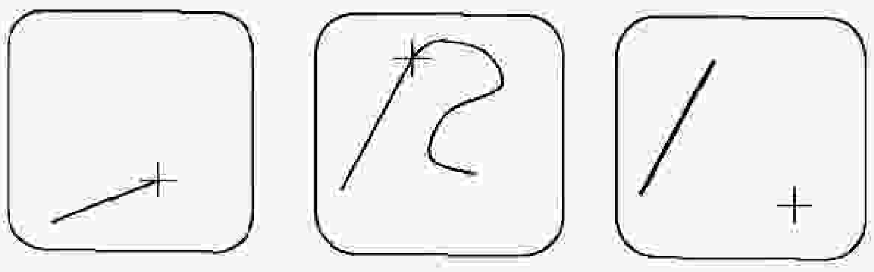
(4) Gravity field:- When it is needed to connect lines at positions between endpoints, the graphics packages convert any input position near a line to a position on the line. The conversion is accomplished by creating a gravity area around the line. Any related position within the gravity field of line is moved to the nearest position on the line. It illustrated with a shaded boundary around the line.

Gravity



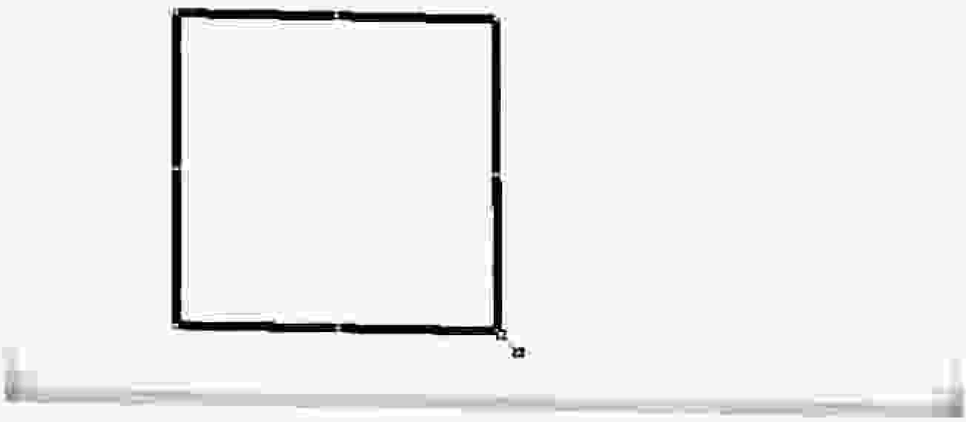
(5) Rubber Band Methods:- Straight lines can be constructed and positioned using rubber band methods which stretch out a line from a starting position as the screen cursor.

Rubber-Band

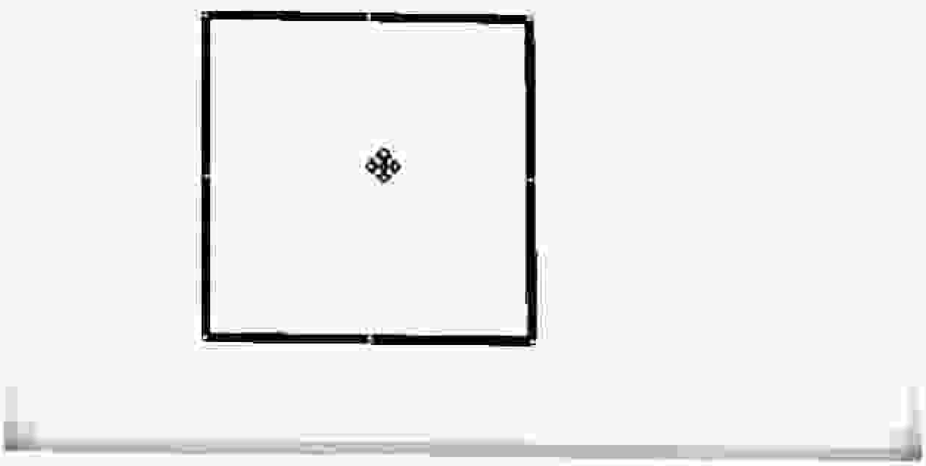


(6) Dragging:- This methods move object into position by dragging them with the screen cursor.

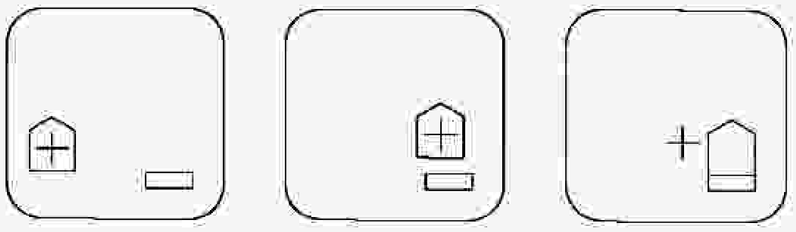
Example:

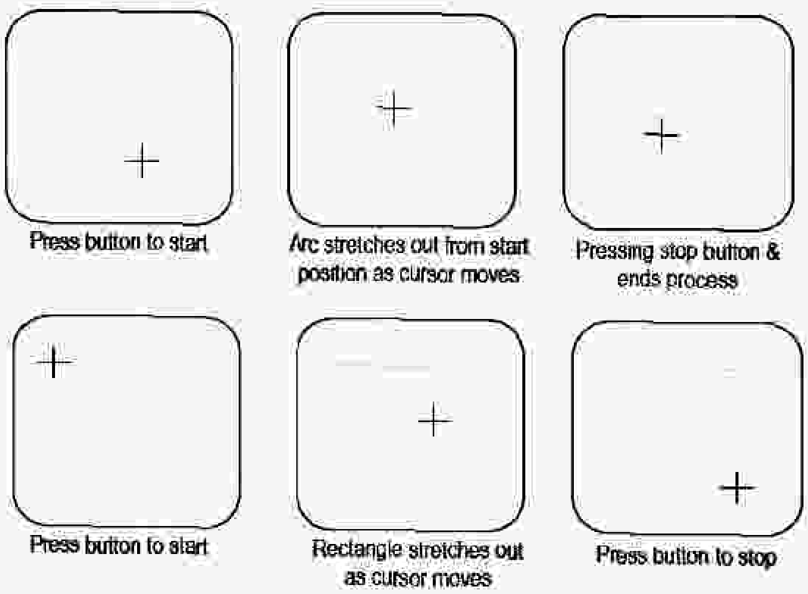


Example:



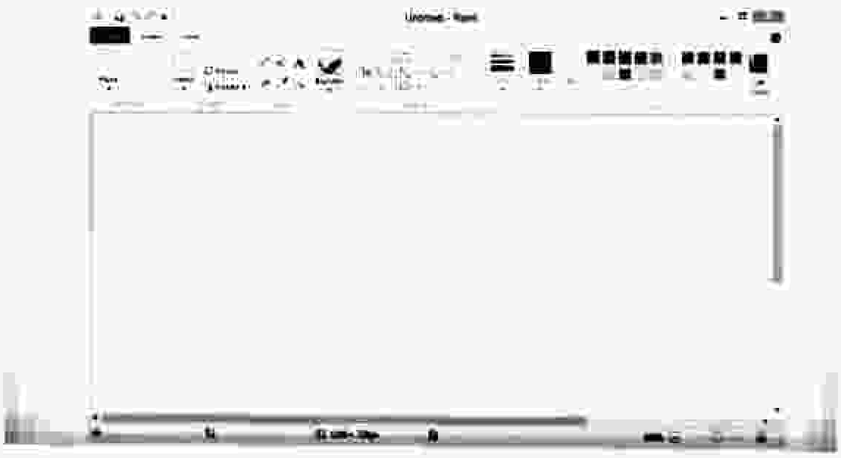
Dragging





(7) Painting and Drawing:- Cursor drawing options can be provided using standard curve shapes such as circular arcs and splices, or with freehand sketching procedures. Line widths, line styles and other attribute options are also commonly found in painting and drawing packages.

Example



Viewing pipeline

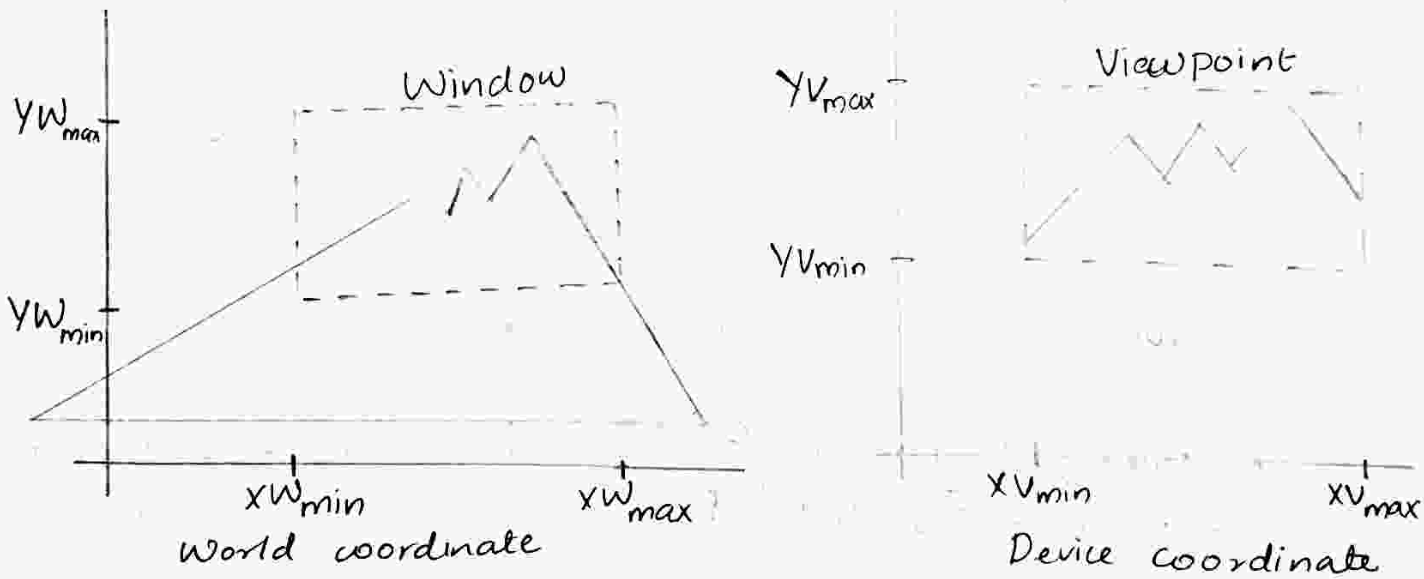
* Window :- An area on a display device / what is to be displayed.

* Viewport :- where it is to be displayed

* 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 mapping of a part of a world coordinate scene to device coordinates is called viewing transformation.

* The 2-Dimensional viewing transformation is also called window-two-viewport transformations or the windowing transformation.



Viewing transformation

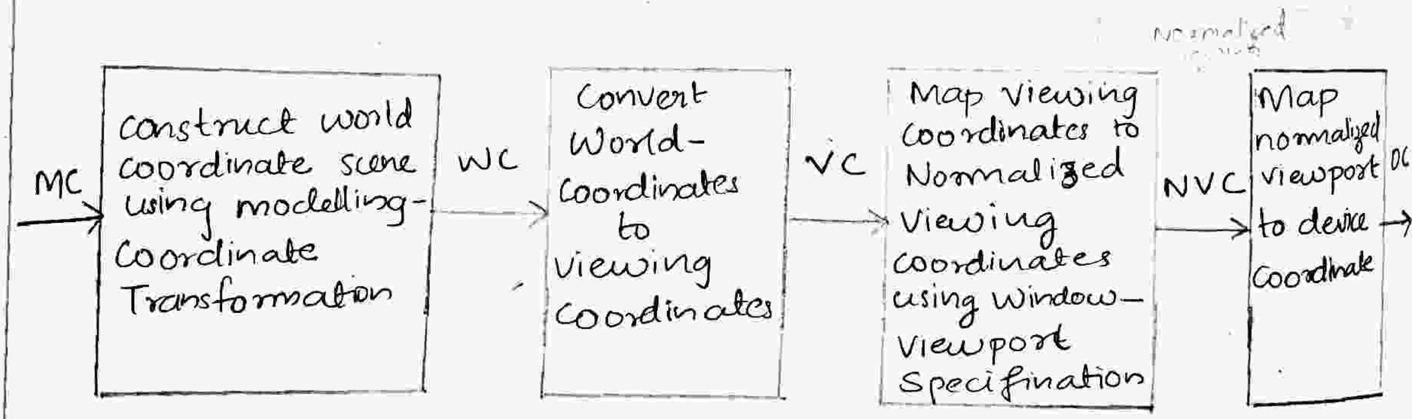
* We construct the scene in world coordinate using the output primitives and attributes.

* Next, to ~~of~~ obtain a particular orientation for the window we can set up a 2-Dimensional viewing coordinating system in the world coordinate plate & define a window in the viewing coordinate system.

* The viewing coordinate reference frame is used to provide a method for setting up arbitarary orientations for rectangular windows.

* Once the viewing reference frame is establisled we can transform descriptions in world coordinates to viewing coordinates.

* Then, define a viewport in normalized coordinates.
* At the final step, all parts of the picture that lie outside the viewport are clipped, and the content of the viewport are transferred to device coordinate.



Window to viewport coordinate transformation

A point at position (x_w, y_w) in the window is mapped into position (x_v, y_v) in the associated viewport

$$\frac{xv - xv_{min}}{xv_{max} - xv_{min}} = \frac{xw - xw_{min}}{xw_{max} - xw_{min}} \quad \text{—(1)}$$

$$\frac{yv - yv_{min}}{yv_{max} - yv_{min}} = \frac{yw - yw_{min}}{yw_{max} - yw_{min}} \quad \text{—(2)}$$

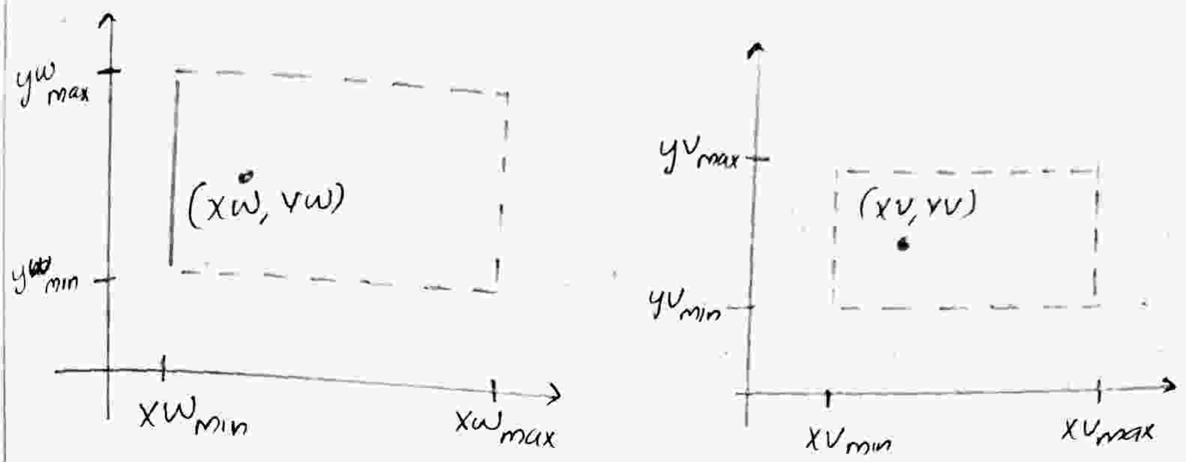


fig: A point at position (xw, yw) in a designated window is mapped to view port coordinate (xv, yv) so that relative positions in the two areas are the same.

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

$$xv = \left[\frac{xw - xw_{min}}{xw_{max} - xw_{min}} \times xv_{max} - xv_{min}\right] + xv_{min}$$

$$xv = xv_{min} + \left[(xw - xw_{min})\left[\frac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}}\right]\right] \quad -③$$

Solving ③

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

$$\left[\because \frac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}} = sx\right]$$

where the scaling factors are,

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

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

This conversion is performed with the following sequence of transformation

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 proportion of objects are maintained, if the scaling factors are the same $(sx = sy)$

* Character strings can be handled in 2 ways:

1. Maintain a constant character size, even though the viewport area may be enlarged or reduced relative to the window.

2. Character formed with line segments, the mapping to the viewport can be carried out as a sequence of line transformation.

* Any number of output devices can be opened in a particular application and another window to viewport transformation can be performed for each open output device. This mapping is called workstation transformation.

Clipping Operation

* Any procedure that identifies those position of a picture that are either inside or outside of a specified region of space is called a clipping algorithm or clipping.

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

Primitive types

* Point clipping
* Line clipping / straight line segments
* Area clipping / polygons
* Curve clipping
* Text clipping

Point clipping

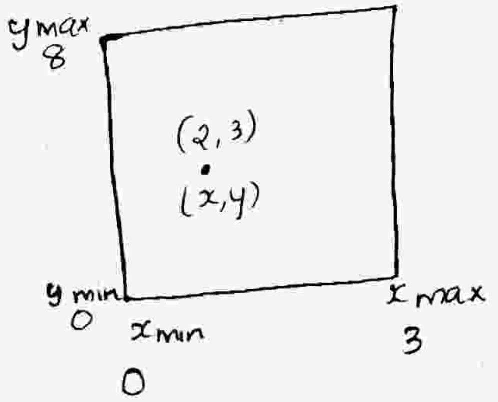
Clip window is a rectangle
Consider a point, $P = (x, y)$ for display, if the following inequalities are satisfied:

$$xw_{min} \leq x \leq xw_{max}$$

$$yw_{min} \leq y \leq yw_{max}$$

When the edges of the clip window (xw_{min}, xw_{max}, yw_{min}, yw_{max}) can be either the word coordinate window boundaries ~~coordinates~~ or viewport boundaries.

If any one of these inequalities is not satisfied, the point is clipped (not ~~satisfied~~ saved for display)



Line clipping

A line clipping procedure involves:

1. Test a given line segment to determine, whether it lies completely inside the clipping window.
2. If it doesnot determine whether it lies completely

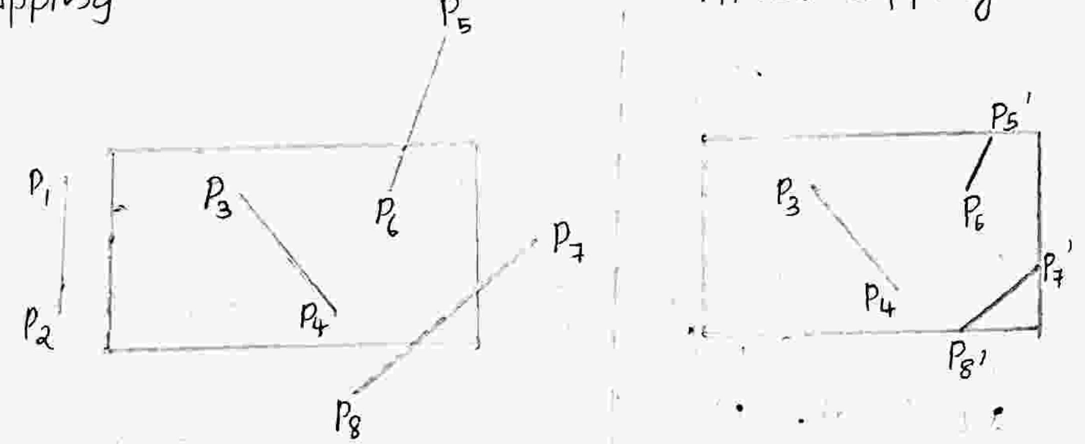
outside the window.

③ If we cannot identify a line as completely inside or completely outside, perform intersection calculation with one or more clipping.

④ A line with both the end points inside all clipping boundaries is saved.

⑤ A line with both ~~on~~ the end points outside, any one of the clip boundaries is outside the window. All other lines cross one or more clipping boundaries, may require calculation of multiple intersection points.

Before clipping | After clipping



Clipping algorithms are used to efficiently identify outside lines and reduce intersection calculations.

- For a line segment with end points (x_1, y_1) and (x_2, y_2) and ~~one~~ one or both end points outside the clipping rectangle, the parametric representation.

$$x = x_1 + u(x_2 - x_1)$$

$$y = y_1 + u(y_2 - y_1) \qquad 0 \leq u \leq 1$$

- If the value of u is outside the range 0 to 1 then the line does not enter the interior of the window.
- If the value ~~of the range~~ 'u' is within the

range from 0 to 1, the line segment does cross into the clipping area.

Cohen - Sutherland line clipping

This is one of the oldest and most popular line-clipping procedures. Every line ending point in a picture is assigned a 4-digit binary code, called a region code, that identifies the location of the point relative to the boundaries of the clipping rectangle. Regions are set up in reference to the boundaries as shown in the figure below.

1001	1000	1010
0001	0000 window	0010
0101	0100	0110

fig: "Binary region codes assigned to line endpoints according to relative positions with respect to the clipping rectangle".

~~Regions are set up in reference to the boundaries of the clipping~~

Each bit position in the region code is used to indicate one of the four relative co-ordinate positions of the point with respect to the clip window; to the left, right, top or bottom.

bit 1: left
bit 2: right
bit 3: below
bit 4: above

If a point is within the clipping rectangle, the region code is 0000. A point that is below and to the left of the rectangle has a region code of 0101.

Region-code bit values can be determined with the following 2 steps: (1) Calculate differences between endpoint coordinates and clipping boundaries. (2) Use the resultant sign bit of each difference calculation to set the corresponding value in the region code.

Any lines that are completely contained within the window boundaries have a region code of 0000 for both end points. Any lines that have a 1 in the same bit positions in the region codes for each endpoints and a code of 0101 for the other end-points. A method that can be used to test lines for total clipping is to perform the logical and operation with both region codes. If the result is not 0000, the line is completely outside the clipping region.

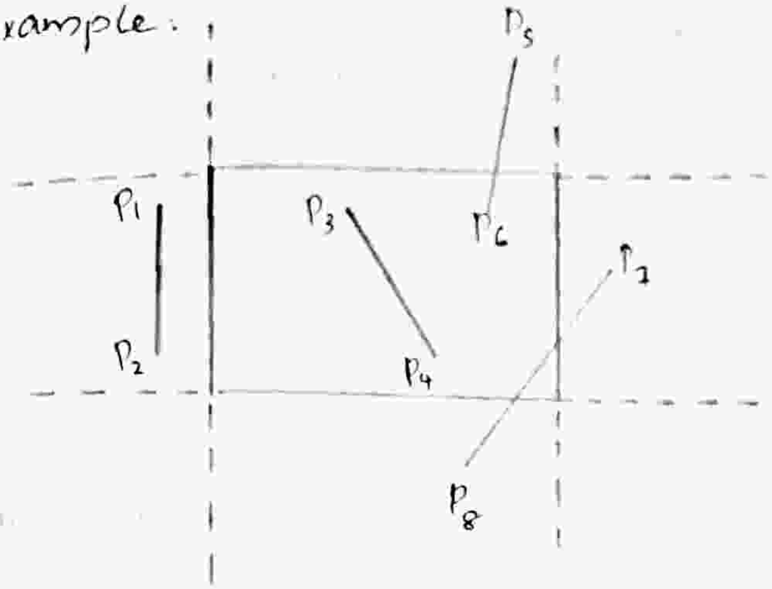
Intersection points with a clipping boundary can be calculated using the slope-intercept form of the line equation. For a line with endpoint coordinates (x_1, y_1) and (x_2, y_2) the y coordinate of the intersection point with a vertical boundary can be obtained with the calculation

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

when the x value is set either to xw_{min} or to xw_{max} and the slope of the line is calculated as $m = (y_2 - y_1)/(x_2 - x_1)$. Similarly, the intersection with a horizontal boundary, the x coordinate can be calculated as,

$$x = x_1 + \frac{y - y_1}{m}$$ with y set either to yw_{min} or to yw_{max}

Example.



- P_1 & P_2

P_1 - 0001 (non-zero value)
P_2 - 0001 (non-zero value)

AND - 0001 (non zero)

Our output of AND operation is also non-zero, so it means that the line is completely outside the window: so we have to clip it (reject).

- P_3 & P_4

P_3 - 0000 (zero)
P_4 - 0000 (zero)

AND - 0000 (zero value)

Our output of AND operation is zero and the 2 input values are also zero that means the line is completely inside the window. so we accept it.

- P_5 & P_6

P_5 - 1000 (non-zero)
P_6 - 0000 (zero)

AND - 0000

If one is zero & other is a non-zero value, we perfom partial clipping. ie, the line have some portion inside the window and some portion outside the window. Then we consider the intersection of points of the line ie, here it is P_5'

- P_5' and P_6

P_5' - 0000
P_6 - 0000

AND - 0000

Now (P_5', P_6) is inside the window. So we take it

- P_7, P_8

P_7 - 0010	P_7' - 0000
P_8 - 0100	P_8' - 0000
AND - 0000	

P_7 - 0010	P_8 - 0100
P_7' - 0000	P_8' - 0000
AND - 0000	AND - 0000
Here we apply partial clipping	Here also we apple partial clipping

Now consider P_7' and P_8'.

P_7' - 0000
P_8' - 0000

AND - 0000

We accept the line (P_7', P_8') bz all the 3 values are zero.

∴ After line clipping we got the lines inside the window. They are,

→ (P_3, P_4)
→ (P_5', P_6)
→ (P_7', P_8)

Sutherland- Hodgeman polygon clipping

Begining with the initial set of polygon vertices, first clip the polygon against the left rectangle boundary to produce a new sequence of vertices The new set of vertices could then be succesively passed to a right boundary clipper, a bottom boundary clipper can a top boundary clipper at each step, a new sequence of output vertices is generated and passed to the next window boundary clipper. There are 4 possible cases when processing vertices in sequence:

~~when processing vertices in sequence~~

1. If the first vertex is outside the window boundary and the 2nd vertex is inside, both the intersection point of the polygon edge with the window boundary and the second vertex are added to the o/p vertex list.

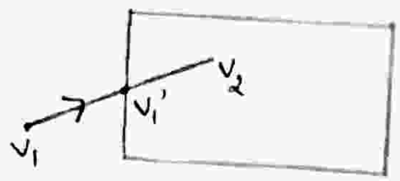
2. If both input vertices are inside the window boundary only the second vertex is added to the output vertex list

3. If the first vertex is inside the window boundary & the second vertex is outside, only the edge intersection with the window boundary is added to the output vertex list.

4. If both input vertices are outside the window boundary nothing is added to the output list.

Once all vertices have been processed for one clip window boundary the output list of vertices is clipped against the next window boundary

1st Case

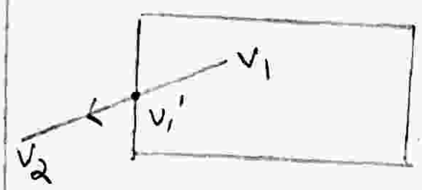


Here the line is from outside to inside so we consider the intersection point as V_1', V_2

ie, Out - in

V_1', V_2

<u>2nd case</u>

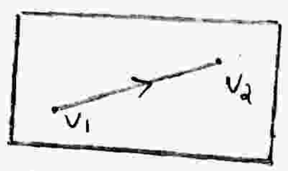


in-out

then poin is,

V_1'

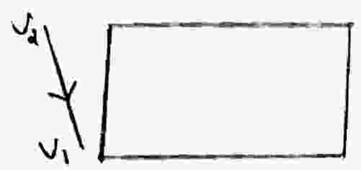
<u>3rd case</u>



in- in

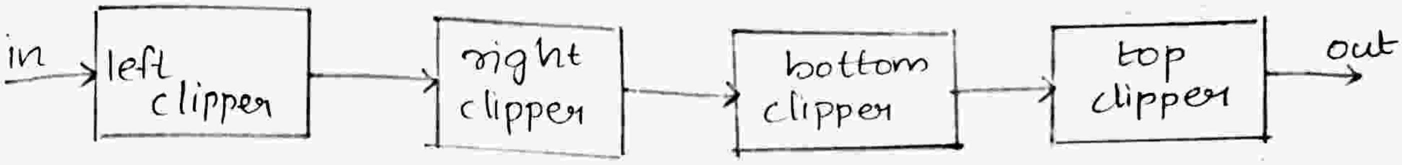
point - V_2

<u>4th case</u>



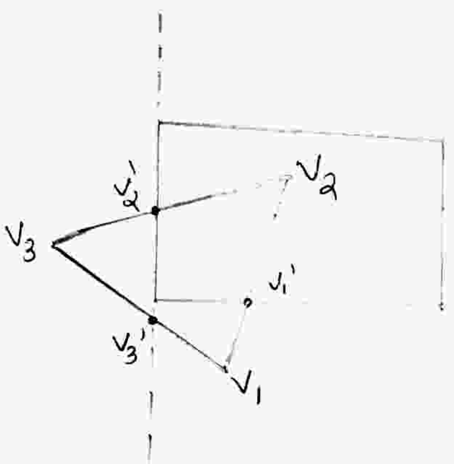
out-out

then point is NIL



Example

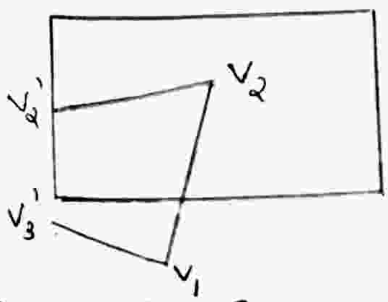
In polygon clipping we need to consider ~~to~~ each cases & each steps.



left clip

		O/P
V_1V_2	in – in	V_2
V_2V_3	in – out	V_2'
V_3V_1	out – in	V_3', V_1

Output of left clip (input of right clip)

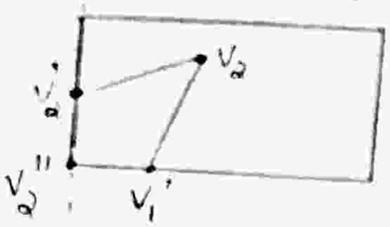


Right clip (completely inside the window)

V_1V_2
V_2V_2'
$V_2'V_3$
$V_3'V_1$

output of right clip & input of bottom clip is the above fig

Bottom clip (output)

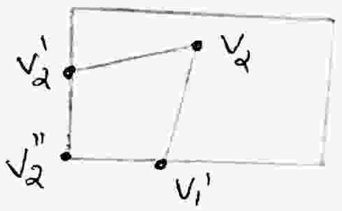


$V_1 V_2$	out - in	$V_1' V_2$	
$V_2 V_2'$	in - in	V_2	
$V_2' V_3'$	in - out	V_2'	V_2''
$V_3' V_1$	out - in	V_2''	nil

Top clip

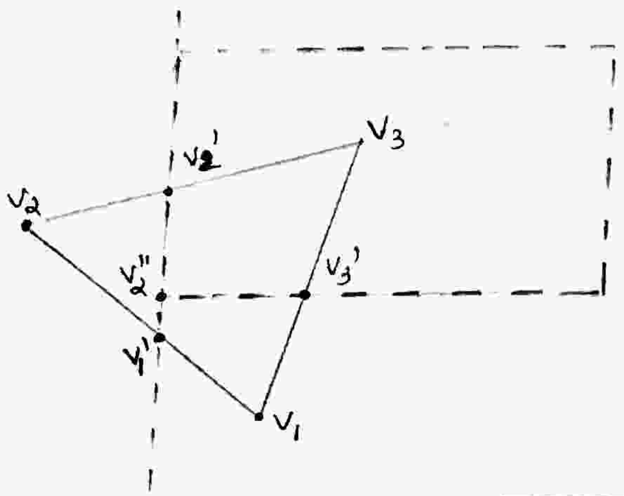
No change (all the vertices are inside the window)

Output



OR

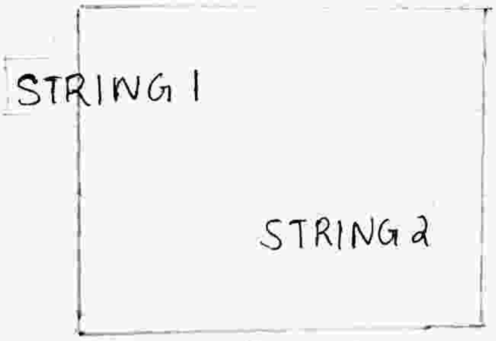
A polygon overlapping a rectangular clip window



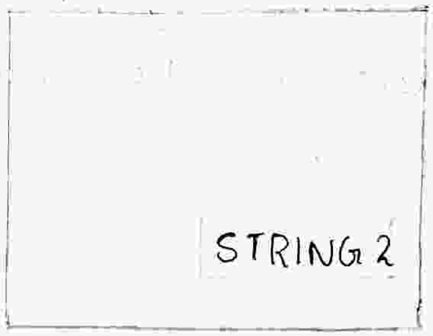
V_1 →	V_1 →	V_1		
V_2 →	V_1' →	V_1'		
V_3 →	(V_2', V_3) →	(V_2', V_3) →	(V_2'', V_2')	(V_2'', V_2')
		→	V_3	V_3
		→	V_3'	V_3'

Text clipping

- The simplest method for processing the character strings relative to a window boundary is to use "all-or-none-string-clipping" stratedgy.
- If all of the string is inside a clip window, we keep it. Otherwise the string is discarded. The boundary positions of the rectangle are compared to the window boundaries and the string is rejected if there is any overlap
- This clipping method is considered as the fastest text clipping..

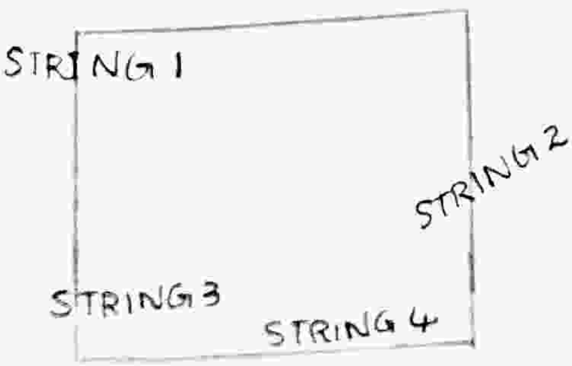


Before Clipping

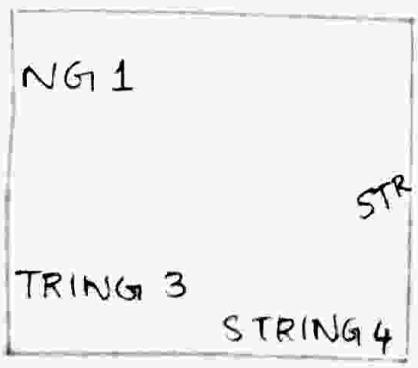


After clipping.

- An alternative method to rejecting an entire character string that overlaps a window boundary is to use all-or-none character-clipping stratedgy.
- In this method, discard only those characters that are not completely inside the window. Any character that either overlap or is outside the window boundary is clipped.



Before clipping



After clipping

- Next method for handling text clipping is to clip, the components of individual character.
- If an individual character overlaps a clip window boundary, clip off the parts of the character that are outside the window.
- Outline character fond, foomed with line segments can be processed in this way using a line clipping algorithm
- Charatters defined with bit maps would be clipped by comparing, the relative position of the individual pixels in the charadter grid patterns to the clipping boundaries.