Unit 4
Two-Dimensional Transformations

Introduction to Transformation

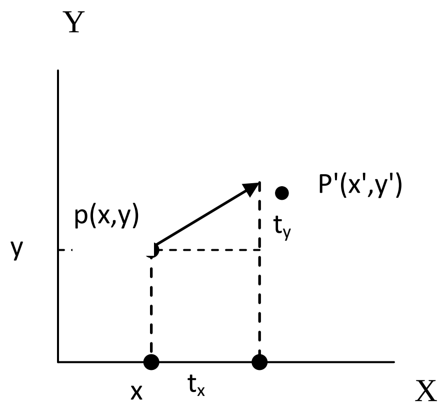
In computer graphics, transformations of 2D objects are essential to many graphics applications. The transformations are used directly by application programs and within many graphics subroutines in application programs. Many applications use the geometric transformations to change the position, orientation, and size or shape of the objects in drawing. Rotation, Translation and scaling are three major transformations that are extensively used by all most all graphical packages or graphical subroutines in applications. Other than these, reflection and shearing transformations are also used by some graphical packages.

<u>2D Translation</u>

A translation is applied to an object by re-positioning it along a straight line path from one co-ordinate location to another. We translate a two-dimensional point by adding translation distances, t_x, t_y to the respective co-ordinate values of original co-ordinate position (x, y) to move the point to a new position (x', y') as:

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

The translation distance pair (t_x, t_y) is known as translation vector or shift vector. We can express translation equations as matrix representations as



$$P = \begin{bmatrix} x \\ y \end{bmatrix} \qquad P' = \begin{bmatrix} x' \\ y' \end{bmatrix} \qquad T = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

$$\therefore P' = P + T$$

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

Sometimes, matrix transformations are represented by co-ordinate rows vector instead of column vectors as.

$$P = (x, y) \qquad T = (t_x, t_y) \qquad P' = P + T.$$

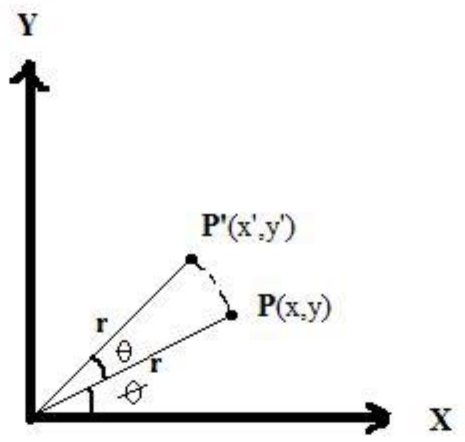
For translation of any object in Cartesian plane, we transform the distinct co-ordinates by the translation vector and re-draw image at the new transformed location.

2D Rotation

The 2D rotation is applied to re-position the object along a circular path in XY-plane. To generate rotation, we specify a rotation angle θ, through which the co-ordinates are to be rotated. Rotation can be made by angle θ either clockwise or anticlockwise direction. Besides the angle of rotation θ, there should be a pivot point through which the object is to be rotated. The positive θ rotates object in anti-clockwise direction and the negative value of θ rotates the object in clock-wise direction.

A line perpendicular to rotating plane and passing through pivot point is called axis of rotation.

Let P(x,y) is a point in XY-plane which is to be rotated with angle θ. Also let OP = r (As in figure below) is constant distance from origin. Let r makes angle ϕ with positive X – direction as shown in figure.



When OP is rotated through an angle θ taking origin as pivot point for rotation, then OP' makes an angle $\theta + \phi$ with X-axis.

Now ,

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

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

But, $r\cos\phi = x, r\sin\phi = y$ therefore we get

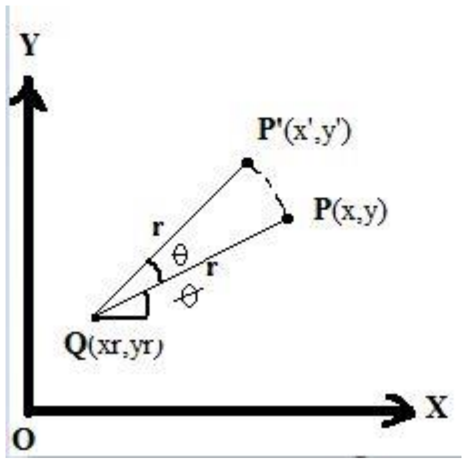
$x' = x\cos\theta - y\sin\theta$ ------------1

$y' = x\sin\theta + y\cos\theta$ ------------2 which are equation for rotation of (x,y) with angle θ and taking pivot as origin.

Rotation from any arbitrary pivot point (x_r, y_r)

- Let Q(x_r,y_r) is pivot point for rotation.
- P(x,y) is co-ordinate of point to be rotated by angle θ.
- Let ϕ is the angle made by QP with X-direction. θ.

Then angle made by QP' with X-direction is $\theta + \phi$



Hence,

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

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

or $x'-x_r=r\cos\phi\cos\theta-r\sin\phi\sin\theta$

since $r\cos\phi=x-x_r, rsiin\phi=y-y_r$

$$x'=x_r+(x-x_r)\cos\theta-(y-y_r)\sin\theta \ldots\ldots\ldots\ldots(1)$$

Similarly,

$\sin(\phi+\theta)=(y'-y_r)/r$

or $r\sin(\phi+\theta)=(y'-y_r)$

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

since $r\cos\phi=x-x_r, rsiin\phi=y-y_r$

$$y'=y_r+(x-x_r)\sin\theta+(y-y_r)\cos\theta \ldots\ldots\ldots\ldots(2)$$

These equations (1) and (2) are the equations for rotation of a point (x,y) with angle θ taking pivot point (x_r,y_r).

The rotation about pivot point (x_r,y_r) can be achieve by sequence of translation, rotation about origin and reverse translation.

- Translate the point (x_r,y_r) and P(x,y) by translation vector ($-x_r$,$-y_r$) which translates the pivot to origin and P(x,y) to ($x-x_r$,$y-y_r$).
- Now apply the rotation equations when pivot is at origin to rotate the translated point ($x-x_r$,$y-y_r$) as:

  $$x_1=(x-x_r)\cos\theta-(y-y_r)\sin\theta$$

  $$y_1=(x-x_r)\sin\theta+(y-y_r)\cos\theta$$

- Re-translate the rotated point (x_1,y_1) with translation vector (x_r,y_r) which is reverse translation to original translation. Finally we get the equation after successive transformation as

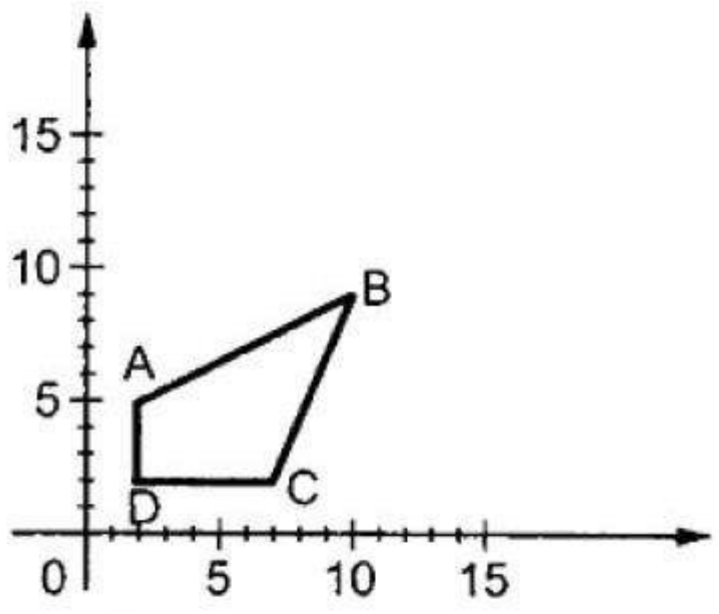
  $$x'=x_r+(x-x_r)\cos\theta-(y-y_r)\sin\theta \ldots\ldots\ldots\ldots(1)$$

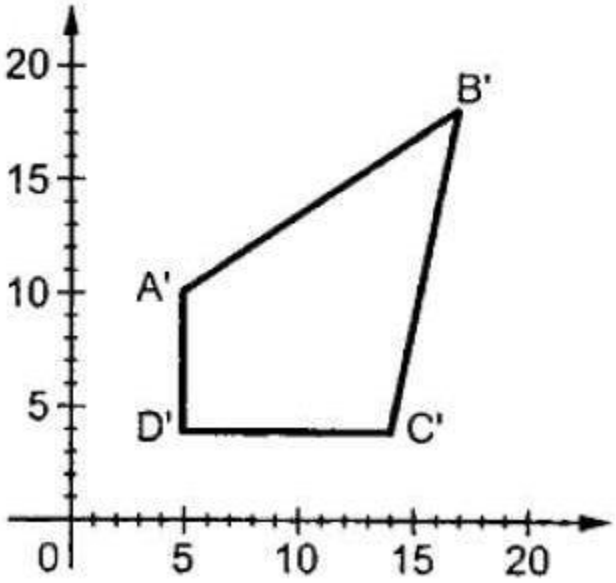
  $$y'=y_r+(x-x_r)\sin\theta+(y-y_r)\cos\theta \ldots\ldots\ldots\ldots(2)$$ which are actually the equations for rotation of (x,y) from the pivot point (x_r,y_r)

2D Scaling

A scaling transformation alters the size of the object. This operation can be carried out for polygon by multiplying the co-ordinate values (x,y) of each vertex by scaling factor s_x and s_y to produce transformed co-ordinates (x', y').

i.e. $x' = x.s_x$ and $y' = y.s_y$





Scaling factor s_x scales object in x- direction and s_y scales in y- direction. If the scaling factor is less than 1, the size of object is decreased and if it is greater than 1 the size of object is increased. The scaling factor = 1 for both direction does not change the size of the object. If both scaling factors have same value then the scaling is known as uniform scaling. If the value of s_x and s_y are different, then the scaling is known as differential scaling. The differential scaling is mostly used in the graphical package to change the shape of the object.

The matrix equation for scaling is:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} . \begin{bmatrix} x \\ y \end{bmatrix} \quad \text{i.e. } P' = S.P$$

<u>Homogeneous co-ordinate representation of 2D Transformation</u>

- The homogeneous co-ordinate system provide a uniform frame-work for handling different geometric transformations, simply as multiplication of matrices.
- Its extension to 3D is straight forward which also helps to produce perspective projections by use of matrix multiplication. We simply add a third co-ordinate to 2D point i.e.
  (x,y) =(x_h,y_h,h) where x = x_h/h , y = y_h/h where h is 1 usually for 2D case.
- By using this homogeneous co-ordinate system, a 2D point would be (x,y,1). The matrix formulation for 2D translation for T(t_x,t_y) is :

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \text{By which we can get}$$

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

For Rotation: R(θ) about origin the homogeneous matrix equation will be

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \text{which gives the}$$

$x' = x\cos\theta - y\sin\theta$ ------------1

$y' = x\sin\theta + y\cos\theta$ ------------2 which are equation for rotation of (x,y) with angle θ and taking pivot as origin.

<u>Rotation about an arbitrary fixed pivot point (x_r,y_r)</u>

For a fixed pivot point rotation, we can apply composite transformation as

1. Translate fixed point (x_r,y_r) to the co-ordinate origin by T(-x_r,-y_r).
2. Rotate with angle θ → R(θ) .
3. Translate back to original position by T(x_r,y_r)

This composite transformation is represented as:

P' = T(x_r,y_r). R(θ). T(-x_r,-y_r).P

Which can be represented in matrix equation using homogeneous co-ordinate system as

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_r \\ 0 & 1 & y_r \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_r \\ 0 & 1 & -y_r \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$ Expanding this equation we get

$$x' = x_r + (x - x_r)\cos\theta - (y - y_r)\sin\theta \ldots\ldots\ldots\ldots(1)$$

$$y' = y_r + (x - x_r)\sin\theta + (y - y_r)\cos\theta \ldots\ldots\ldots\ldots(2)$$ Which are actually the equations for rotation of (x,y) from the pivot point (x_r,y_r)

<u>Scaling with scaling factors</u> (s_x, s_y)

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$ which exactly gives the equations

$x' = x.s_x$ and $y' = y.s_y$

<u>Fixed point scaling:</u>

Objects transformed with standard scaling equation are scaled as well as re-positioned. Scaling factor with value less than 1 moves the object closer to the origin whereas value of scaling factor greater than 1 moves object away from origin.

- To control the location of scaled object, we can choose the position called fixed point. Let co-ordinates of fixed point = (x_f,y_f). It can be any point on the object.
- A polygon is then scaled relative to (x_f,y_f) by scaling the distance from each vertex to the fixed point.
- For a vertex with co-ordinate (x,y), the scaled co-ordinates (x',y') are calculated as:

$$x' = x_f + (x - x_f)s_x$$
$$y' = y_f + (y - y_f)s_y$$ or equivalently,

$$x' = x.s_x + (1 - s_x)x_f$$
$$y' = y.s_y + (1 - s_y)y_f$$

Where $(1-s_x)x_f$ *and* $(1-s_y)y_f$ are constant for all points in object.

To represent fixed point scaling using matrix equations in homogeneous co-ordinate system, we can use composite transformation as in fixed point rotation.

1. Translate object to the origin so that (x_f, y_f) lies at origin by T(-x_f, -y_f).
2. Scale the object with (s_x, s_y)
3. Re- translate the object back to its original position so that fixed point (x_f, y_f) moves to its original position. In this case translation vector is T(x_f, y_f).

$$\therefore P' = \left[T(x_f, y_f).S(s_x, s_y).T(-x_f, -y_f)\right]P$$

The homogeneous matrix equation for fixed point scaling is

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_f \\ 0 & 1 & y_f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_f \\ 0 & 1 & -y_f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & x_f(1-s_x) \\ 0 & s_y & y_f(1-s_y) \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Directive scaling

Standard and fixed point scaling scales object along x and y axis only. Directive scaling scales the object in any direction.

Let S_1 and S_2 are given directions for scaling at angle ϴ from co-ordinate axes as in figure below

1. First perform the rotation so that directions S_1 and S_2 concide with x and y – axes.
2. Then the scaling transformation is applied to scale the object by given scaling factors (s_1,s_2).
3. Re-apply the rotation in opposite direction to return to their original orientation.

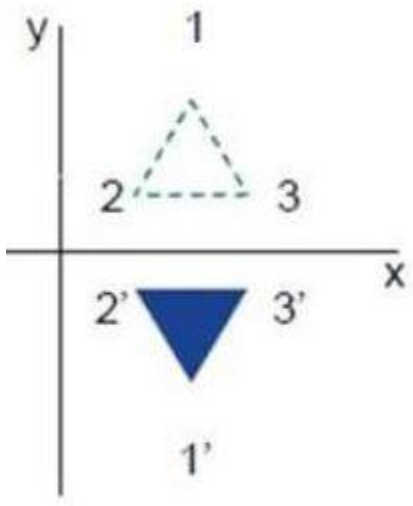
   For any point P in object, the directive scaling position P' is given by following composite transformation.

   $P' = R^{-1}(\theta).S(s_1, s_2).R(\theta).P$ for which the homogeneous matrix equation is

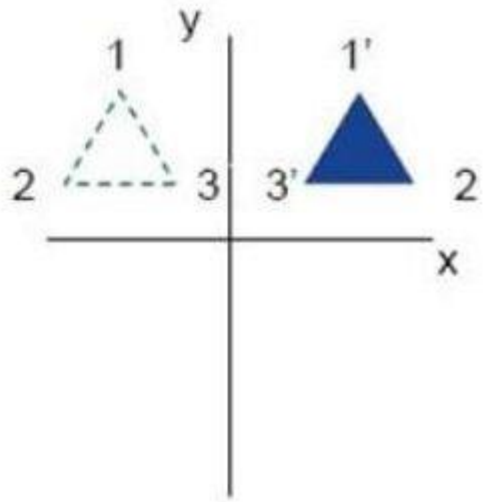
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} s_1 & 0 & 0 \\ 0 & s_2 & 0 \\ 0 & 0 & 1 \end{Bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Reflection

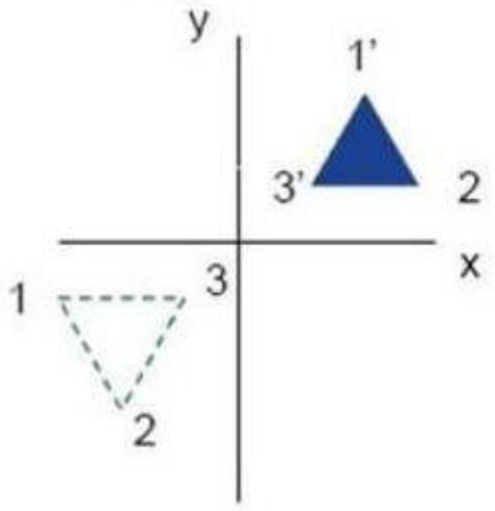
A reflection is a transformation that produces a mirror image of an object. In 2D-transformation, reflection is generated relative to an axis of reflection. The reflection of an object to a relative axis of reflection is same as 180^o rotation about the reflection axis.



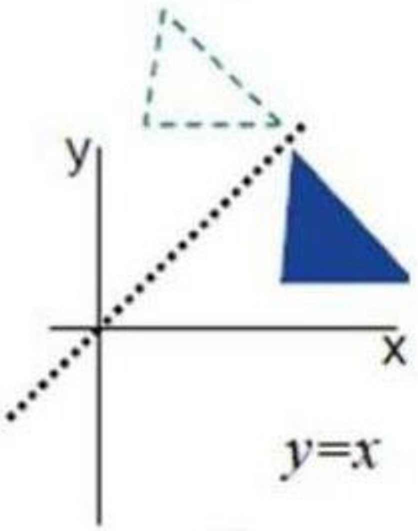
(a)



(b)



(c)



(d)

1. Reflection about X-axis: The line representing x-axis is y = 0. The reflection of a point P(x,y) on x-axis, changes the y-coordinate sign, i.e. Reflection about x-axis, the reflected position of P(x,y) will be P'(x,-y). Hence, reflection in x-axis is accomplished with transformation equation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

gives the reflection of a point.

To reflect an 2D object, reflect each distinct points of object by above equation, then joining the points with straight line, redraws the image for reflected image.

2. Reflection about Y-axis: The line representing y-axis is x = 0. The reflection of a point P(x,y) on y-axis changes the sign of x-coordinate. i.e. P(x,y) changes to P'(-x,y).

Hence reflection of a point on y-axis is obtained by following matrix equation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

For any 2D object, reflect each point and re-draw image joining the reflected images of all distinct points.

3. Reflection on an arbitrary axis: The reflection on any arbitrary axis of reflection can be achieved by sequence of rotation and co-ordinate axes reflection matrices.
    - First, rotate the arbitrary axis of reflection so that the axis of reflection and one of the co-ordinate axis coincide.
    - Reflect the image on the co-ordinate axis to which the axis of reflection coincides.
    - Rotate the axis of reflection back to its original position.

   For example, consider a line y = x for axis of reflection, the possible sequences of transformation for reflection of 2D object are

   i. Rotation (line coincides with horizontal axis)
   ii. Reflection about x-axis
   iii. Re-rotation

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

4. Reflection about origin: The reflection on the line perpendicular to xy-plane and passing through flips x and y co-ordinates both. So sign of x and y co-ordinate value changes. The equivalent matrix equation for the point is:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

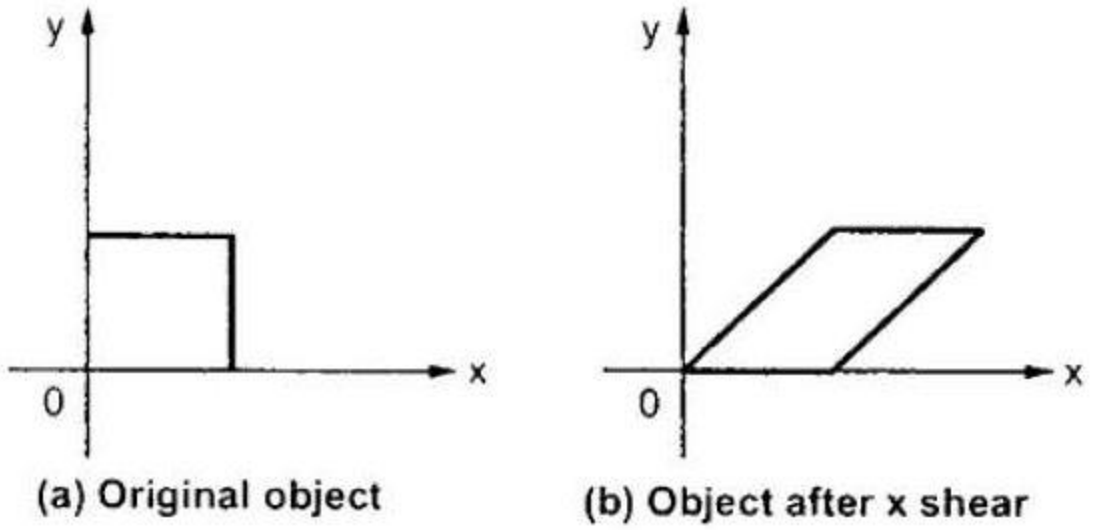
Q. Show the steps for reflection about y = -x. Also, find out the composite matrix.
Q. Show the steps for reflection about y = mx + c. Also, find out the composite matrix.
Q. Justify that a reflection matrix is inverse of itself.

Shearing

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 is called shear.

X-direction Shear: An x-direction shear relative to x-axis is produced with transformation matrix equation.

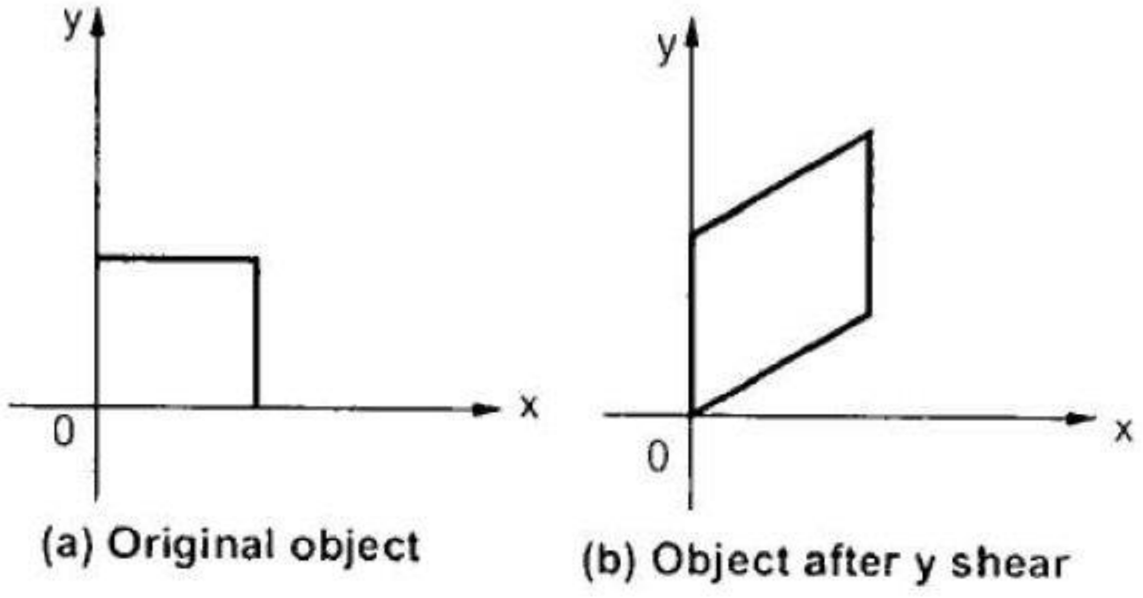
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$ which transforms $x' = x + y.sh_x$ and y' = y



A unit square transformed to a parallelogram using x-direction shear with $sh_x = 2$.

Y-direction shear: A y-direction shear relative to y-axis is produced by following transformation equations.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$ which transforms $x' = x$ and $y' = x.sh_y + y$

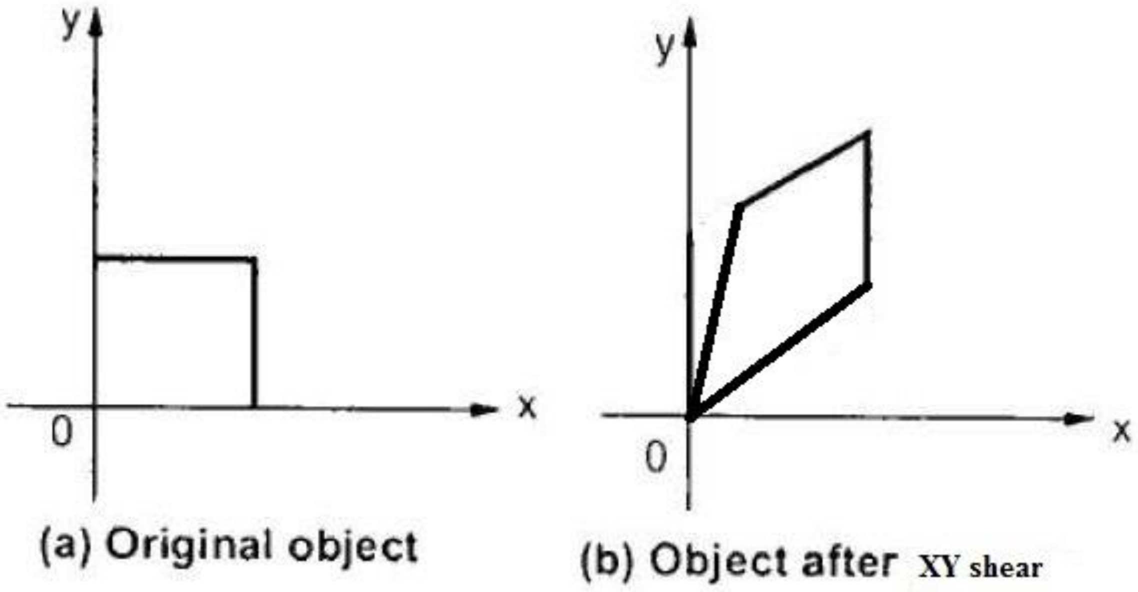


XY Shear: Shearing in both directions is given by

$x' = x + sh_x*y$

$y' = x*sh_y + y$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & sh_x & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



(a) Original object (b) Object after XY shear

Successive and Composite Transformations

If a transformation of the plane T1 is followed by a second plane transformation T2, then the result itself may be represented by a single transformation T which is the composition of T1 and T2 taken in that order. This is written as T = T1·T2.

Composite transformation can be achieved by concatenation of transformation matrices to obtain a combined transformation matrix.

A combined matrix −

[T][X] = [X] [T1] [T2] [T3] [T4] …. [Tn]

Where [Ti] is any combination of

- Translation
- Scaling
- Shearing
- Rotation
- Reflection

The change in the order of transformation would lead to different results, as in general matrix multiplication is not cumulative, that is [A] . [B] ≠ [B] . [A] and the order of multiplication. The basic purpose of composing transformations is to gain efficiency by applying a single composed transformation to a point, rather than applying a series of transformation, one after another.

For example, to rotate an object about an arbitrary point (X_p, Y_p), we have to carry out three steps –

- Translate point (X_p, Y_p) to the origin.
- Rotate it about the origin.
- Finally, translate the center of rotation back where it belonged.

- We'll discuss more in class.

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

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

The composite transformation matrix for this sequence of transformation is:

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

This shows that two successive translations are additive.

*** Rotation:**

$$\begin{bmatrix}\cos\theta_1 & -\sin\theta_1 & 0\\ \sin\theta_1 & \cos\theta_1 & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\cos\theta_2 & -\sin\theta_2 & 0\\ \sin\theta_2 & \cos\theta_2 & 0\\ 0 & 0 & 1\end{bmatrix} = \begin{bmatrix}\cos(\theta_1+\theta_2) & -\sin(\theta_1+\theta_2) & 0\\ \sin(\theta_1+\theta_2) & \cos(\theta_1+\theta_2) & 0\\ 0 & 0 & 1\end{bmatrix}$$

This shows that successive rotations are additive.

***Scaling:**

$$\begin{bmatrix}s_{x1} & 0 & 0\\ 0 & s_{y1} & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}s_{x2} & 0 & 0\\ 0 & s_{y2} & 0\\ 0 & 0 & 1\end{bmatrix} = \begin{bmatrix}s_{x1}.s_{x2} & 0 & 0\\ 0 & s_{y1}.s_{y2} & 0\\ 0 & 0 & 1\end{bmatrix}$$

This shows that successive scalings are multiplicative.

Q. Reflect the point P(0,3) with respect to y=x+2.
Q. Find the co-ordinates of object bound by (0,0), (1,5), (6,3) when reflected from line y= 2x + 4.
Q. Find the transformation matrix that transforms the square to half its size with its centre remaining same. The square is of (1,1), (1,3), (3,3) and (3,1).
Q. After rotating a triangle with vertex A(0,0), B(1,7) and C(9,2) in 60^0 anticlockwise about point (10,10). What will be the new vertex?
Q. Show that the transformation matrix

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

about the line y=x is equivalent to a reflection relative to x-axis followed by a counter-clockwise rotation of 90^0.

Viewing

Two Dimensional viewing is the formal mechanism for displaying views of a picture on an output device. Typically, a graphics package allows a user to specify which part of a defined picture is to be displayed and where that part is to be placed on the display device. Any convenient Cartesian coordinate system, referred to as the world-coordinate reference frame, can be used to define the picture. For a two-dimensional picture, a view is selected by specifying a subarea of the total picture area. The picture parts within the selected areas are then mapped onto specified areas of the device coordinates. Transformations from world to device co ordinates involve translation, rotation, and scaling operations, as well as procedures for deleting those parts of the picture that are outside the limits of a selected display area.

Co-ordinate Representation

- With few exceptions, general packages are designed to be used with Cartesian co-ordinate system.
- If it is other co-ordinate system, it must be converted.

Local co-ordinate

- The shape of individual object can be constructed with in a scene within separate co-ordinate reference frame is called *local co-ordinate* or *modeling co-ordinate* or *master co-ordinate*.

World co-ordinate

- Once individual objects have been specified (identified), those objects are placed into appropriate position within scene using the reference frame and this reference frame is called *world co-ordinate*.

Device co-ordinate

- When the world co-ordinate description of the scene is transferred to one or more output device reference frame for display, the display co-ordinate systems are referred to as *device co-ordinate* or screen co-ordinate in the case of video monitor.

Viewing co-ordinate

- Used to define window in the world co-ordinate plane with any possible orientation.

Normalized Device co-ordinate

- The coordinates are in range 0 to 1. Generally, a graphical system first converts world co-ordinate position to normalized device co-ordinates before final conversion to specified device co-ordinates. This makes the system independent of the various devices that might be used at a particular workstation.

Window: A world-coordinate area selected for display
Viewport: An area on a display device to which a window is mapped
"The window defines ***what*** is to be viewed; the viewport defines ***where*** it is to be displayed"

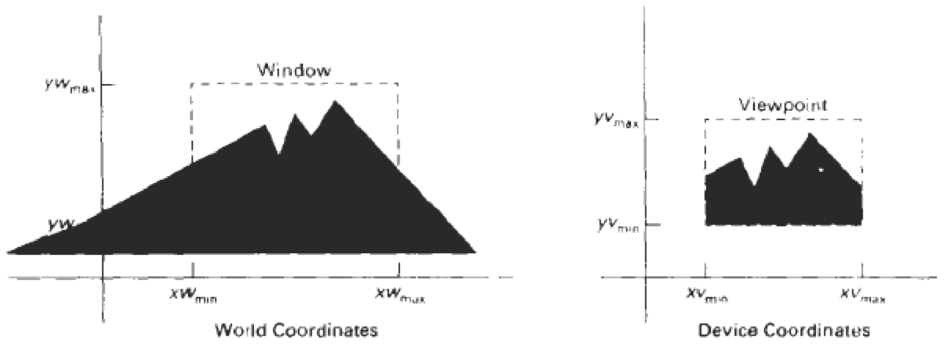
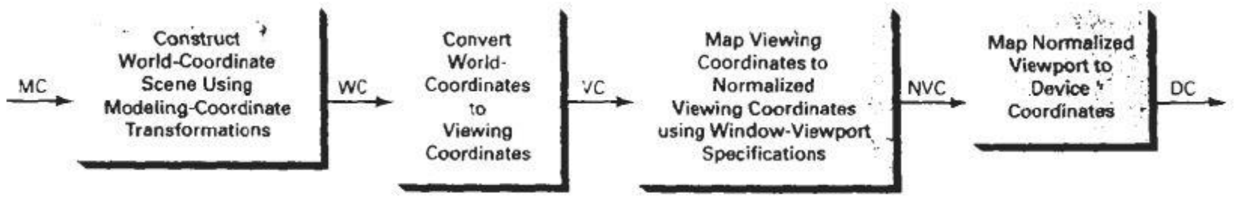
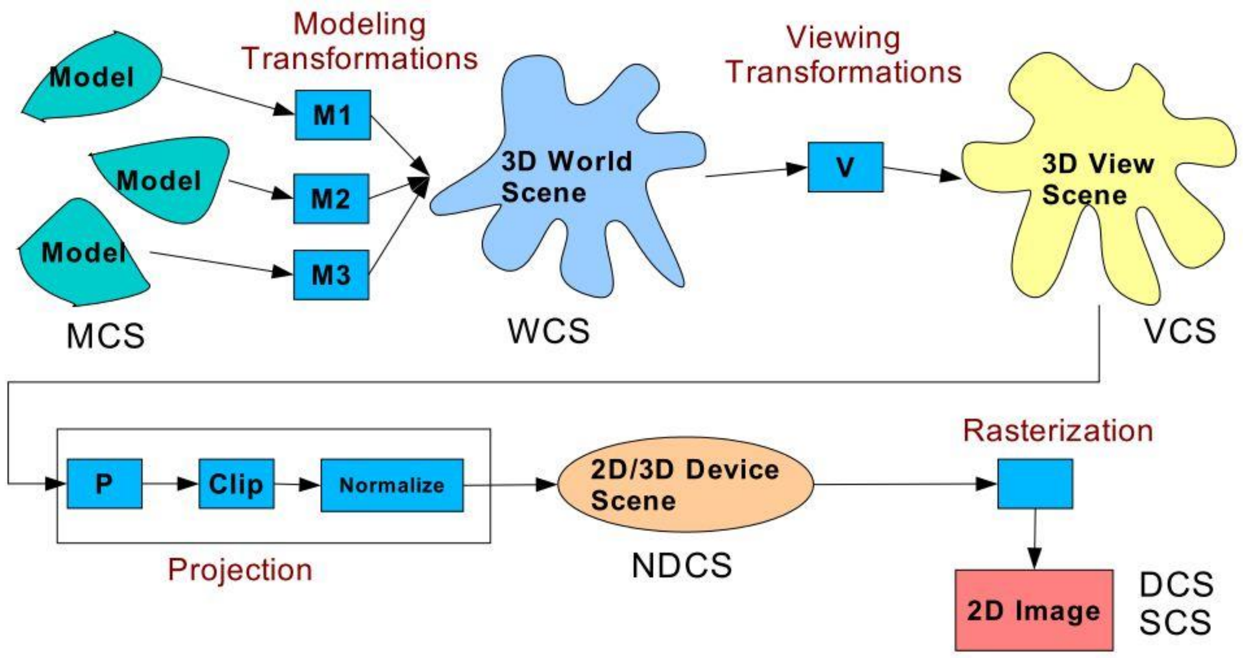


Fig: A viewing transformation using standard rectangles for the window and viewport



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

Fig: 2D viewing pipeline



Window-to-Viewport Coordinate Transformation

(Windowing 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.

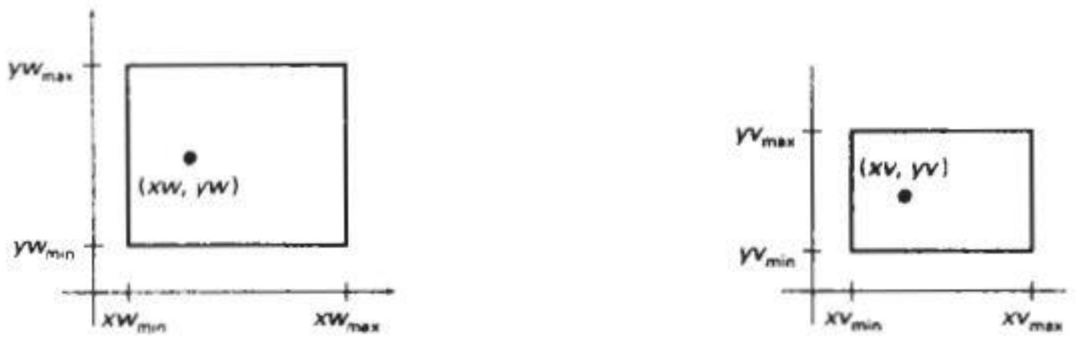


Fig: A point at position (xw,yw) in a designated window is mapped to viewport coordinates (xv,yv) so that relative positions in the two areas are the same.

- Window and viewport are rectangular in standard position with the rectangular edge parallel to the co-ordinate axis.
- The mapping of a part of a world co-ordinate scene to device co-ordinate is referred to as a *viewing transformation*.
- Sometimes 2D viewing transformation is simply referred to as *window-to-viewport* or *windowing transformation*.

To maintain the same relative placement in the viewport as in the window, we require that:

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

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

Solving these equations 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

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

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

Equations above 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.

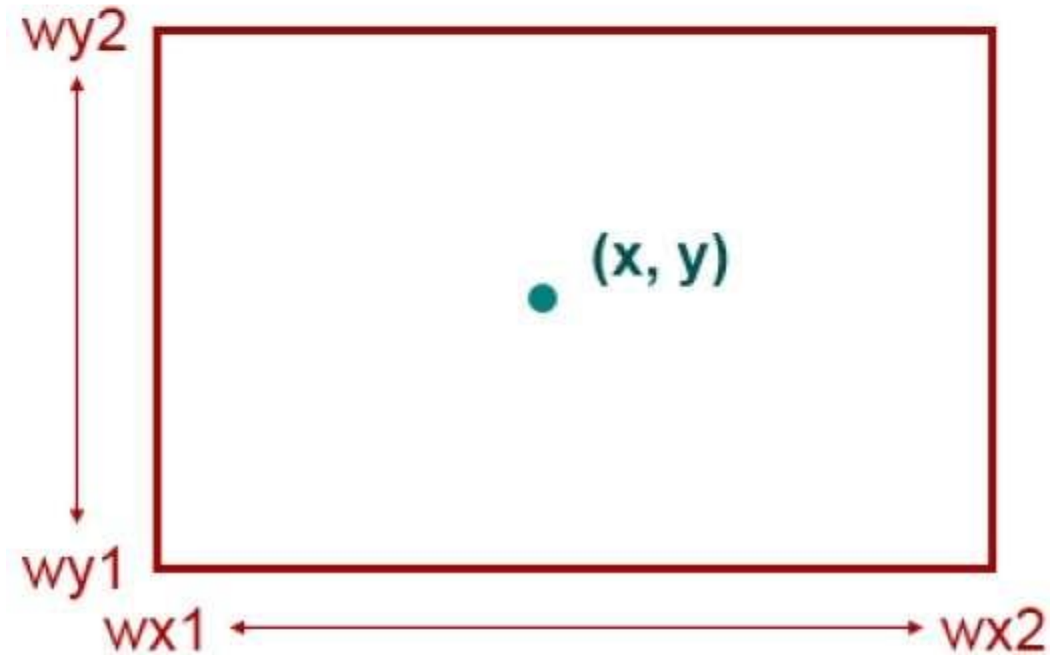
Q. Find out the composite matrix for window-to-viewport transformation.
Q. given a window of the following dimension:
(xw_{min}, yw_{min}) = (10, 10) & (xw_{max}, yw_{max}) = (50, 50)
(xv_{min}, yv_{min}) = (5, 5) & (xv_{max}, yv_{max}) = (20, 20)
How do you trsnsform a point (10, 25) in a window to a viewport?

Clipping

The primary use of clipping in computer graphics is to remove objects, lines, or line segments that are outside the viewing pane. The viewing transformation is insensitive to the position of points relative to the viewing volume − especially those points behind the viewer − and it is necessary to remove these points before generating the view.

Point Clipping

Clipping a point from a given window is very easy. Consider the following figure, where the rectangle indicates the window. Point clipping tells us whether the given point (X, Y) is within the given window or not; and decides whether we will use the minimum and maximum coordinates of the window.
The X-coordinate of the given point is inside the window, if X lies in between $Wx1 \le X \le Wx2$.
Same way, Y coordinate of the given point is inside the window, if Y lies in between $Wy1 \le Y \le Wy2$.

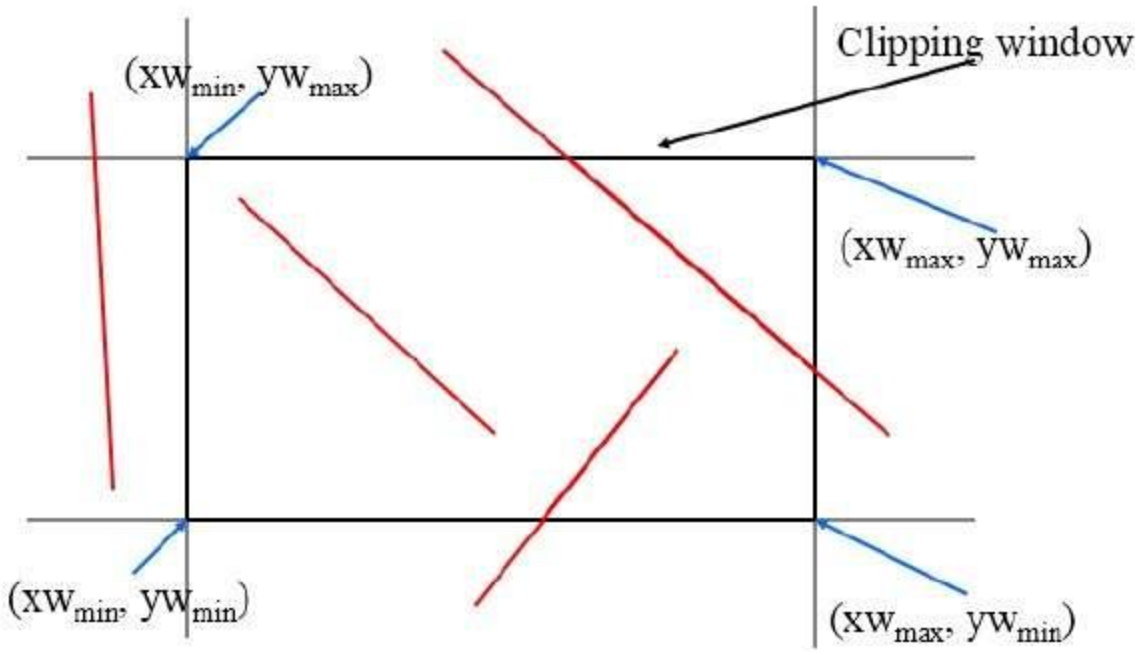


Line Clipping

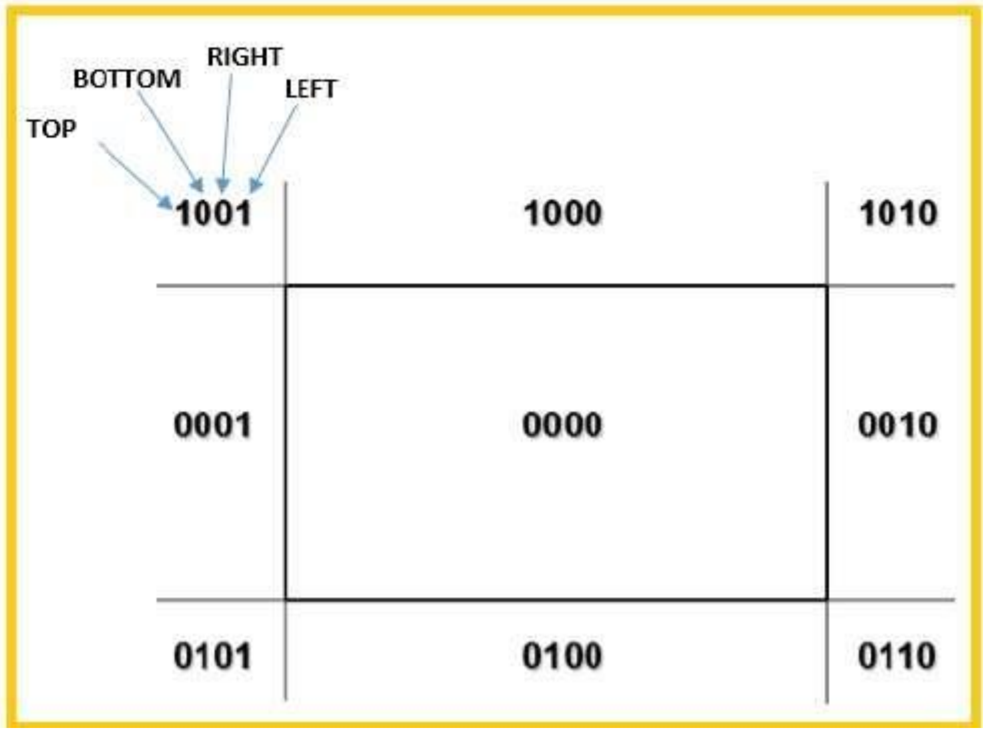
The concept of line clipping is same as point clipping. In line clipping, we will cut the portion of line which is outside of window and keep only the portion that is inside the window.

Cohen-Sutherland Line Clipping Algorithm

This algorithm uses the clipping window as shown in the following figure. The minimum coordinate for the clipping region is (XWmin,YWmin)(XWmin,YWmin) and the maximum coordinate for the clipping region is (XWmax,YWmax)(XWmax,YWmax).



We will use 4-bits to divide the entire region. These 4 bits represent TBRL (the Top, Bottom, Right, and Left) of the region as shown in the following figure. Here, the **TOP** and **LEFT** bit is set to 1 because it is the **TOP-LEFT** corner.



There are 3 possibilities for the line −

- Line can be completely inside the window (This line will be completely removed from the region).

- Line can be completely outside of the window (This line will be completely removed from the region).
- Line can be partially inside the window (We will find intersection point and draw only that portion of line that is inside region).

Algorithm

1. Assign 4-bit REGION Code [TBRL] for end-points of line.
   → Bit 1 is set to 1 if $x<xw_{min}$, else set to 0. (L)
   → Bit 2 is set to 1 if $x>xw_{max}$, else set to 0. (R)
   → Bit 3 is set to 1 if $y<yw_{min}$, else set to 0. (B)
   → Bit 4 is set to 1 if $y>yw_{max}$, else set to 0. (T)
2. Determine whether the line is completely inside or outside of the window using tests.
   a) If end-point has region code 0000, the point is completely inside.
   b) If logical AND of end-points is not 0000, the line is completely outside.
   c) If logical AND of end-points is 0000, the line is not completely outside.
3. If 2(c) exists, we need to find the intersection with window boundary.
   Here, $m = \frac{y_2-y_1}{x_2-x_1}$
   i) If bit 1 is 1, line intersects with left boundary.
   So, $y_i = y_1 + m(x - x_1)$ where $x=xw_{min}$.
   ii) If bit 2 is 1, line intersects with right boundary.
   So, $y_i = y_1 + m(x - x_1)$ where $x=xw_{max}$.
   iii) If bit 3 is 1, line intersects with bottom boundary.
   So, $x_i = x_1 + \frac{1}{m}(y - y_1)$ where $y=yw_{min}$.
   iv) If bit 4 is 1, line intersects with top boundary.
   So, $x_i = x_1 + \frac{1}{m}(y - y_1)$ where $y=yw_{max}$.
   Here, x_i & y_i are x & y intercepts for that line. Update that.
4. Repeat (1) to (3) for every points till completely accepted.

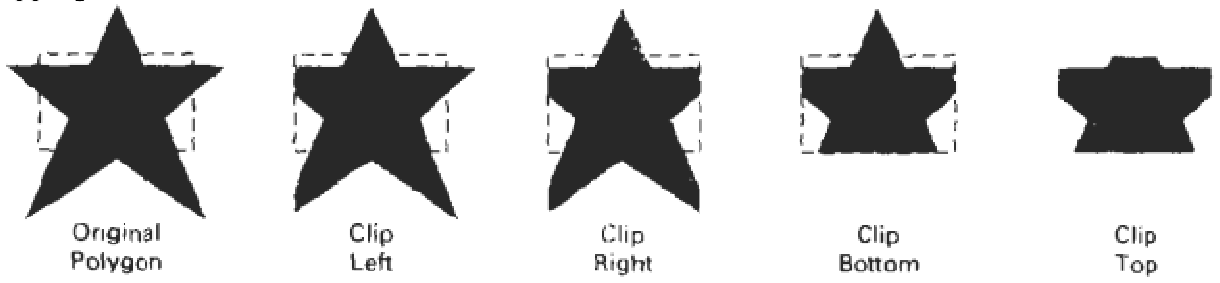
Q. Use this clipping method to clip a line starting from A(-13, 5) and ending at B(17, 11) against the window having its lower corner at (-8, -4) and upper right corner at (12, 8).

Polygon Clipping (Sutherland-Hodgman Algorithm)

A polygon can also be clipped by specifying the clipping window. Sutherland Hodgeman polygon clipping algorithm is used for polygon clipping. In this algorithm, all the vertices of the polygon are clipped against each edge of the clipping window.

First, the polygon is clipped against the left edge of the polygon window to get new vertices of the polygon. These new vertices are used to clip the polygon against right edge, top edge, bottom edge, of the clipping window as shown in the following figure.

While processing an edge of a polygon with clipping window, an intersection point is found if edge is not completely inside clipping window and then, a partial edge from the intersection point to the outside edge is clipped. The following figures show left, right, bottom and top edge clippings −



There are four possible cases when processing vertices in sequence around the perimeter of a polygon. As each pair of adjacent polygon vertices is passed to a window boundary clipper, we make the following tests: (1) If the first vertex is outside the window boundary and the second vertex is inside, both the intersection point of the polygon edge with the window boundary and the second vertex are added to the *output 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 and 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 vertex list*. These four cases are illustrated in figure below for successive pairs of polygon vertices. Once all vertices have been processed for one clip window boundary, the output 11st of vertices is clipped against the next window boundary.

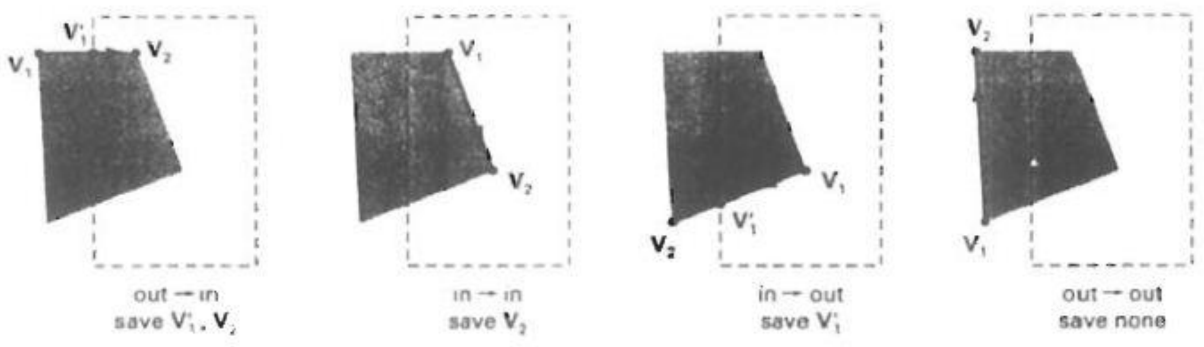


Fig: Successive processing of pairs of polygon vertices against the left window boundary.

Text Clipping

Various techniques are used to provide text clipping in a computer graphics. It depends on the methods used to generate characters and the requirements of a particular application. There are three methods for text clipping which are listed below −

- All or none string clipping
- All or none character clipping
- Text clipping

The following figure shows all or none string clipping −

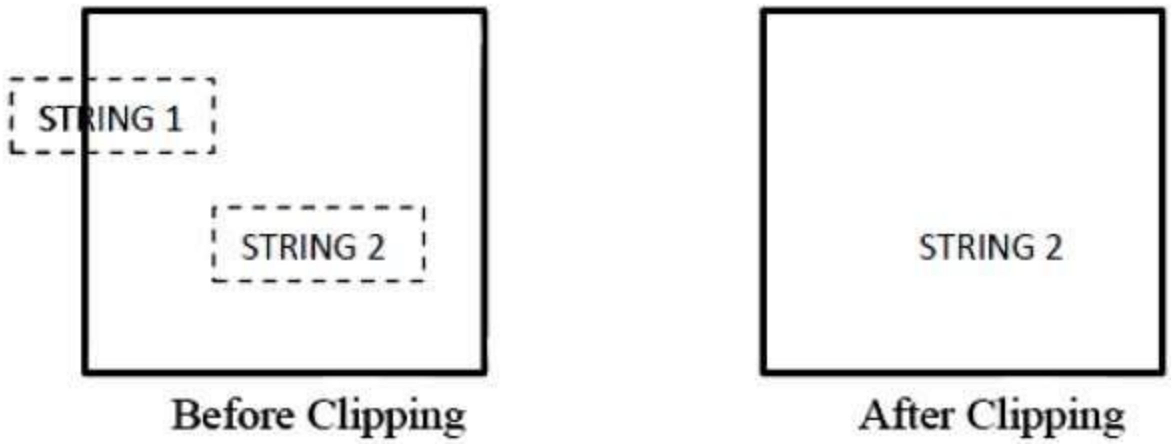


Fig: All or none string clipping

In all or none string clipping method, either we keep the entire string or we reject entire string based on the clipping window. As shown in the above figure, STRING2 is entirely inside the clipping window so we keep it and STRING1 being only partially inside the window, we reject.

The following figure shows all or none character clipping −

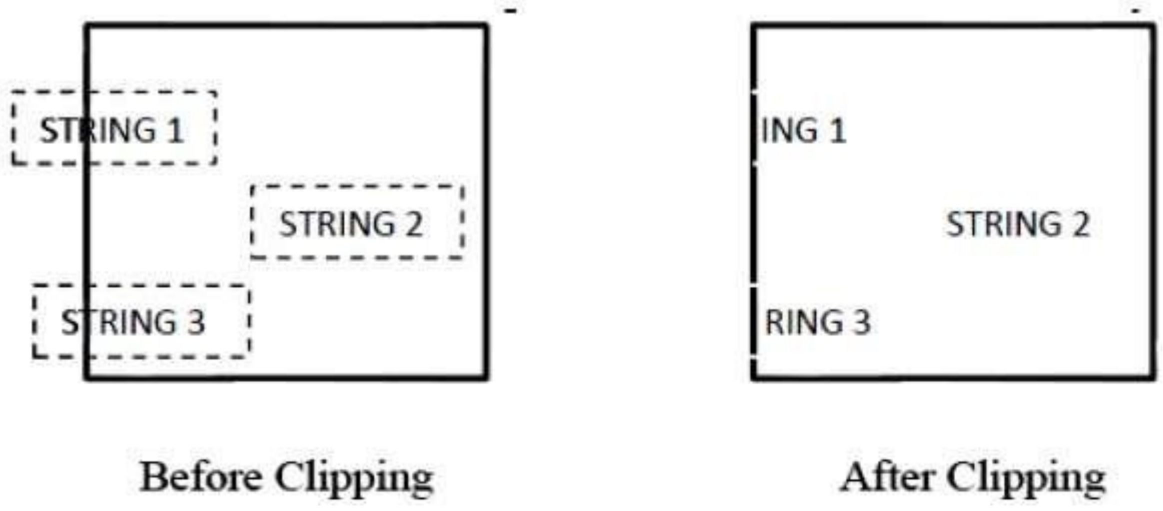


Fig: All or none character clipping

This clipping method is based on characters rather than entire string. In this method, if the string is entirely inside the clipping window, then we keep it. If it is partially outside the window, then

- You reject only the portion of the string being outside
- If the character is on the boundary of the clipping window, then we discard that entire character and keep the rest string.

The following figure shows text clipping −

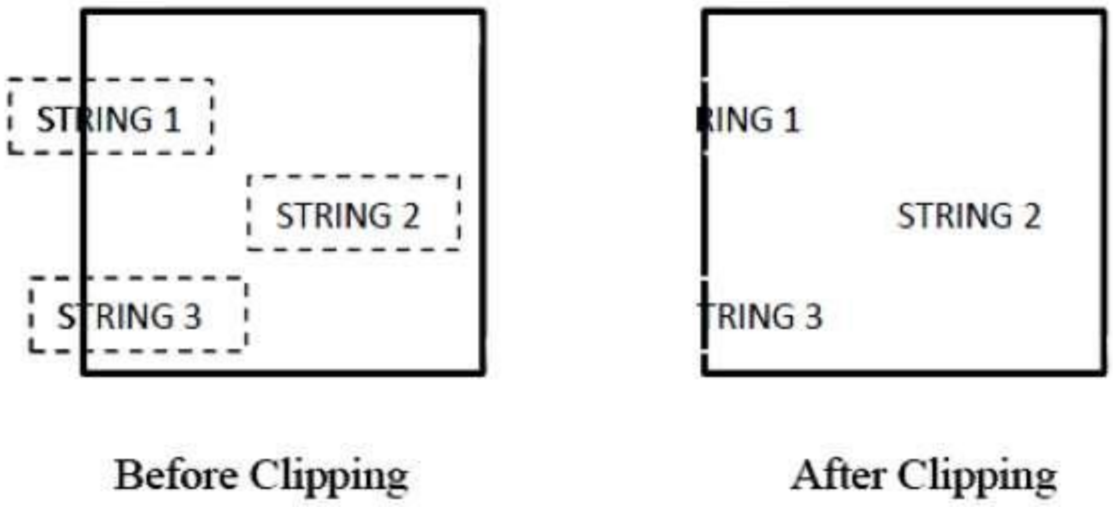


Fig: Text clipping

This clipping method is based on characters rather than the entire string. In this method, if the string is entirely inside the clipping window, then we keep it. If it is partially outside the window, then

- You reject only the portion of string being outside.
- If the character is on the boundary of the clipping window, then we discard only that portion of character that is outside of the clipping window.