By: Mukunda Paudel
paudelmuku@gmail.com

Chapter-4 Two Dimensional Geometric Transformation

What is Transformation?

- Transformation is changing of Position, shape, size, or orientation of an object on display.

Two Types

1. Basic Transformation
2. Other Transformation

Basic Transformation

- Basic transformation includes three transformations
  - ✓ **Translation**,
  - ✓ **Rotation**, and
  - ✓ **Scaling**.
- These three transformations are known as basic transformation because with combination of these three transformations we can obtain any transformation.

Translation

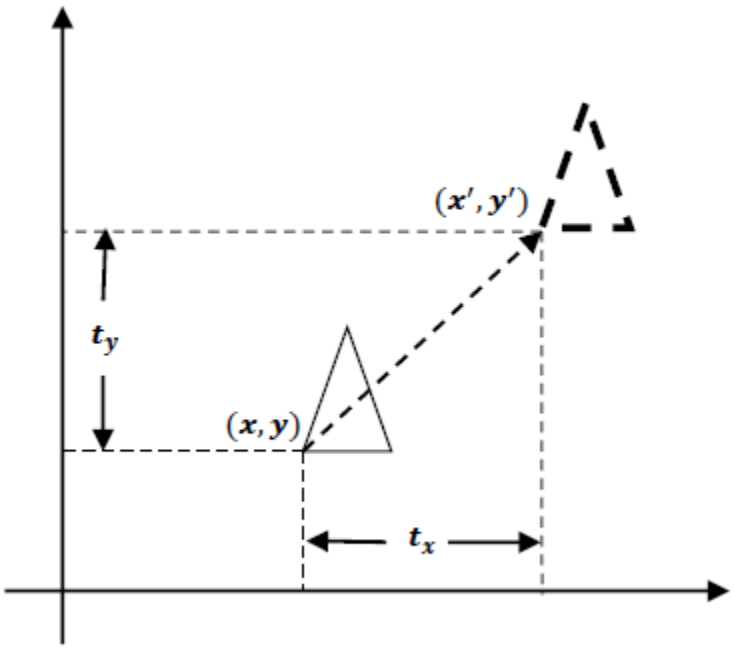


Figure: Translation of a vertex p (x, y) to P' (x', y') of triangle.

- ❖ It is a transformation that used to reposition the object along the straight line path from one coordinate location to another.
- ❖ It is rigid body transformation so we need to translate whole object.
- ❖ We translate two dimensional point by adding translation distance t_x and t_y to the original coordinate position (x, y) to move at new position (x', y') as:

$$x' = x + t_x$$
$$\& \quad y' = y + t_y$$

- ❖ Translation distance pair (t_x, t_y) is called a **Translation Vector** or **Shift Vector**.
- ❖ We can represent it into single matrix equation in column vector as

$$P' = P + T$$

Matrix form

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

- ❖ We can also represent it in row vector form as:

$$P' = P + T$$

$$[x' \; y'] = [x \; y] + [t_x \; t_y]$$

- ❖ Since column vector representation is standard mathematical notation and since many graphics package like **GKS (**Graphics Kernel System**)** and **PHIGS (**Programmer Hierarchical Graphics System,**)** uses column vector we will also follow column vector representation.

Example: - Translate the triangle [A (10, 10), B (15, 15), C (20, 10)] 2 unit in x direction and 1 unit in y-direction.

We know that

$$P' = P + T$$

$$P' = [P] + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

For point (10, 10)

$$A' = \begin{bmatrix} 10 \\ 10 \end{bmatrix} + \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

$$A' = \begin{bmatrix} 12 \\ 11 \end{bmatrix}$$

For point (15, 15)

$$B' = \begin{bmatrix} 15 \\ 15 \end{bmatrix} + \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

$$B' = \begin{bmatrix} 17 \\ 16 \end{bmatrix}$$

For point (10, 10)

$$C' = \begin{bmatrix} 20 \\ 10 \end{bmatrix} + \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

$$C' = \begin{bmatrix} 22 \\ 11 \end{bmatrix}$$

Hence, Final coordinates after translation are [A' (12, 11), B' (17, 16), C' (22, 11)].

Rotation

- It is a transformation that used to reposition the object along the circular path in the XY - plane.
- I.e. move the object from one point to another point with an angle but the distance of that object / point from the origin should be same.

Key Notes:

- To generate a rotation we specify **a rotation angle → $\boldsymbol{\theta}$** and the position of the **Rotation Point** (**Pivot Point**) ($\boldsymbol{x_r}$, $\boldsymbol{y_r}$) about which the object is to be rotated.

- Positive value of rotation angle defines **counter clockwise rotation** (CCW) / **Anti-Clockwise** and negative value of rotation angle defines **clockwise** rotation.

- Line perpendicular to rotating plane and passing through pivot point is called **axis of rotation**.

Rotation when pivot point is at coordinate origin (0, 0).

- Consider a point P(x, y) is original point and r is the constant distance from origin and ∅ is the original angular displacement from x-axis.
- Rotate P (x, y) with and angle $\boldsymbol{\theta}$ in anticlockwise direction and point obtained after rotation is P' (x', y')

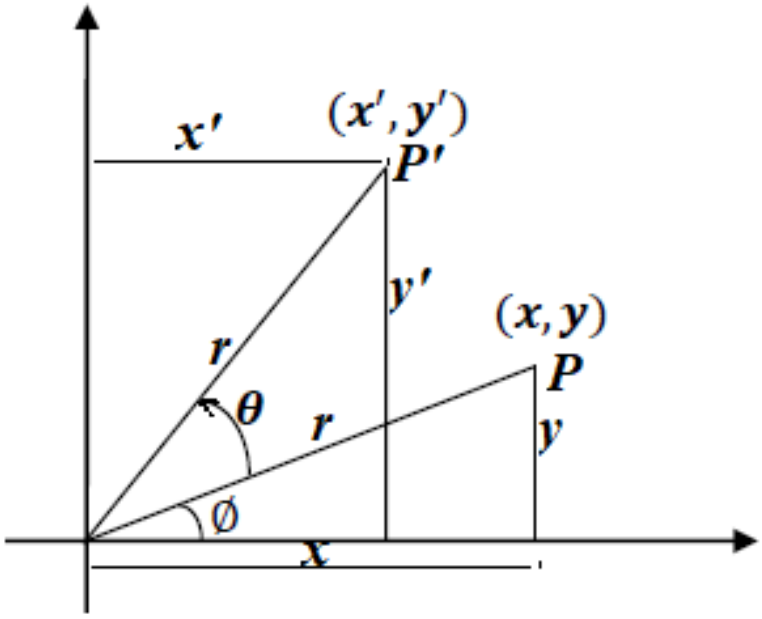


Figure: Rotation at origin (0,0)

From figure we can write.

Cos ∅ = b / h = x / r

$$x = r \cos \emptyset$$

Sin ∅ = p / h = y / r

$$y = r \sin \emptyset$$

Similarly,

Cos (θ + ∅) = x' / r

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

Now replace $r\cos\emptyset$ with x and $r\sin\emptyset$ with y in above equation.

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

And

Sin (θ+∅) = y' / r

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

Now replace $r\cos\emptyset$ with x and $r\sin\emptyset$ with y in above equation.

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

Therefore,

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

We can write it in the form of column vector matrix equation as;

$$P' = R \cdot P$$

Matrix Form

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

If the Rotation is Clockwise then,

θ is negative

We know,

Cos (-$\boldsymbol{\theta}$) = cos $\boldsymbol{\theta}$ and

Sin (-$\boldsymbol{\theta}$) = -sin $\boldsymbol{\theta}$

Example:

1. Apply Rotation on point (4, 3) and angle is 45 degree.
2. Locate the new position of the triangle [A (5, 4), B (8, 3), C (8, 8)] after its rotation by
   i. 90 degree clockwise about the origin.
   ii. 90 degree CCW at origin

Solution of I.

Here,

Given vertices of triangle are A (5, 4), B (8, 3) C (8, 8)

And, Rotation is clockwise hence we take $\theta = -90°$.

We know,

$P' = R \cdot P$

Let's write given vertices into matrix form P=

$$\begin{bmatrix} 5 & 8 & 8 \\ 4 & 3 & 8 \end{bmatrix}$$

Then, P'=R.P

$$P' = \begin{bmatrix} \cos(-90) & -\sin(-90) \\ \sin(-90) & \cos(-90) \end{bmatrix} \begin{bmatrix} 5 & 8 & 8 \\ 4 & 3 & 8 \end{bmatrix}$$

$$P' = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} 5 & 8 & 8 \\ 4 & 3 & 8 \end{bmatrix}$$

$$P' = \begin{bmatrix} 4 & 3 & 8 \\ -5 & -8 & -8 \end{bmatrix}$$

Therefore, Final coordinates after rotation are [A' (4, -5), B' (3, -8), C' (8, -8)].

Scaling

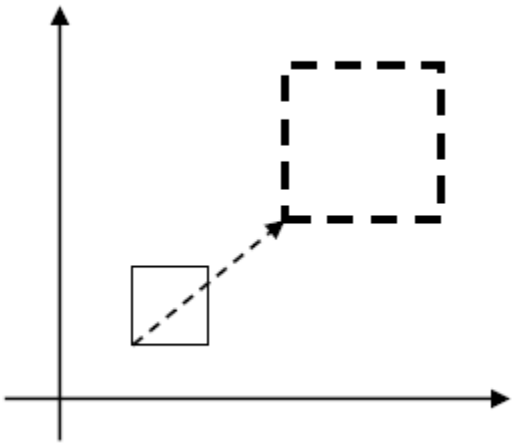


Figure: Scaling

- It is a transformation that used to alter the size of an object.
- This operation is carried out by multiplying coordinate value ($\boldsymbol{x}$, $\boldsymbol{y}$) with scaling factor ($\boldsymbol{sx}$, $\boldsymbol{sy}$) respectively.

So, equation for scaling is given by:

$$\boldsymbol{x' = x \cdot s_x}$$
$$\boldsymbol{y' = y \cdot s_y}$$

- These equation can be represented in column vector matrix equation as:

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

Matrix Representation

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

Any positive value can be assigned to ($\boldsymbol{s_x}$, $\boldsymbol{s_y}$).

- Values of (s_x, s_y) less than 1 reduce the size while values greater than 1 enlarge the size of object, and
- Object remains unchanged when values of both factor is 1.

- ❖ Same values of s_x and s_y will produce **Uniform Scaling**. And different values of s_x and s_y will produce **Differential Scaling**.
- ❖ Objects transformed with above equation are both scale and repositioned.
- ❖ Scaling factor with value less than 1 will move object closer to origin, while scaling factor with value greater than 1 will move object away from origin.

Example:

1. Scale an object about origin with 3 vertices (4,4), (3,2) and (5,2) along x axis and y axis by 2 unit .
2. Consider the square with left-bottom corner at (2, 2) and right-top corner at (6, 6). Apply the transformation which makes its size half.

   Solution:

$$P' = S \cdot P$$

$$P' = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} 2 & 6 & 6 & 2 \\ 2 & 2 & 6 & 6 \end{bmatrix}$$

$$P' = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} 2 & 6 & 6 & 2 \\ 2 & 2 & 6 & 6 \end{bmatrix}$$

$$P' = \begin{bmatrix} 1 & 3 & 3 & 1 \\ 1 & 1 & 3 & 3 \end{bmatrix}$$

Hence, Final coordinate after scaling are [A' (1, 1), B' (3, 1), C' (3, 3), D' (1, 3)].

Matrix Representation and homogeneous coordinates

Why Homogeneous coordinate system?

- ✓ **Homogeneous coordinates** a coordinate system that algebraically treats all points in the projective plane (both Euclidean and ideal) equally.
- ✓ For example, the standard homogeneous coordinates $[p_1,p_2,p_3]$ of a point P in the projective plane are of the form $[x,y,1]$ if P is a point in the Euclidean plane $z=1$ whose Cartesian coordinates are $(x,y,1)$, or are of the form $[a,b,0]$ if P is the ideal point – the point at infinity – associated to all lines in the Euclidean plane $z=1$ with direction numbers $a,b,0$.
- ✓ Homogeneous coordinates are so called because they treat Euclidean and ideal points in the same way.
- ✓ Homogeneous coordinates are widely used in computer graphics because they enable affine and projective transformations to be described as matrix manipulations in a coherent way.

- Many graphics application involves sequence of geometric transformations.
- For example in design and picture construction application we perform Translation, Rotation, and scaling to fit the picture components into their proper positions.
- For efficient processing we will reformulate transformation sequences.
- We have matrix representation of basic transformation and we can express it in the general matrix form as:

$$\boldsymbol{P' = M1 \cdot P + M2}$$

Where,

$\boldsymbol{P}$ and $\boldsymbol{P'}$ are initial and final point position,
$\boldsymbol{M1}$ contains rotation and scaling terms and
$\boldsymbol{M2}$ contains translational terms associated with pivot point, fixed point and reposition.

- For efficient utilization we must calculate all sequence of transformation in one step and for that reason we reformulate above equation to eliminate the matrix addition associated with translation terms in matrix $\boldsymbol{M2}$.
- We can combine that thing by expanding 2X2 matrix representation into 3X3 matrices.
- It will allows us to convert all transformation into matrix multiplication but we need to represent vertex position $(\boldsymbol{x}, \boldsymbol{y})$ with homogeneous coordinate triple $(\boldsymbol{x_h}, \boldsymbol{y_h}, \boldsymbol{h})$ Where $\boldsymbol{x} = \boldsymbol{x_h / h}$,

$$\boldsymbol{y} = \boldsymbol{y_h / h}$$

Thus, we can also write triple as $(\boldsymbol{h \cdot x}, \boldsymbol{h \cdot y}, \boldsymbol{h})$.

- For two dimensional geometric transformation we can take value of $\boldsymbol{h}$ is any positive number so we can get infinite homogeneous representation for coordinate value $(\boldsymbol{x}, \boldsymbol{y})$.
- But convenient choice is set $\boldsymbol{h} = \boldsymbol{1}$ as it is multiplicative identity, than $(\boldsymbol{x}, \boldsymbol{y})$ is represented as $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{1})$.
- Expressing coordinates in homogeneous coordinates form allows us to represent all geometric transformation equations as matrix multiplication.

In short,

- ✓ To convert a 2×2 matrix to 3×3 matrix, we have to add an extra dummy coordinate. So, we can represent the point by 3 numbers instead of 2 numbers, which is called Homogenous Coordinate system.

- ✓ In this system, we can represent all the transformation equations in matrix multiplication.

- ✓ Any Cartesian point P(X, Y) can be converted to homogenous coordinates by P' (x_h, y_h, h). The 'h' is normally set to 1. If the value of 'h' is more the one value then all the co-ordinate values are scaled by this value.

Let's see each representation with $\boldsymbol{h} = \mathbf{1}$

Translation

$$P' = T_{(t_x,t_y)} \cdot P$$

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

NOTE: - Inverse of translation matrix is obtain by putting $-t_x$ & $-t_y$ instead of t_x & t_y.

Rotation:

$$P' = R_{(\theta)} \cdot P$$

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

NOTE: - Inverse of rotation matrix is obtained by replacing θ by $-\theta$.

Scaling

$$P' = S(s_x,s_y) \cdot P$$

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

NOTE: - Inverse of scaling matrix is obtained by replacing s_x & s_y by 1 / s_x & 1 / s_y respectively

Composite Transformation:

- ❖ When more than one transformation are applied for performing a task then such transformation is called composite transformation.
- ❖ Forming product of transformation matrix is referred as concatenation or composition of matrices.
- ❖ Basic purpose of composing transformation is to gain efficiency by applying a single composed transformation to appoint, rather than applying series of transformation one after another.
- ❖ Composite transformation indicates combination of different basic transformation in sequence to obtain desire result or combination could be a sequence of two or more successive transformation (e.g. two successive translation, two successive rotation or two or more successive scaling)

Translations

If Two successive transformation vectors **(t_{x1}, t_{y1}) and (t_{x2}, t_{y2})** are applied to a point **P**, the final position or transformed location **P'** is calculated as:

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

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

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

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

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

- ❖ Here P' and P are column vector of final and initial point coordinate respectively.

- ❖ This shows that **two successive translation are additive**.
- ❖ This concept can be extended for any number of successive translations

Example: Obtain the final coordinates after two translations on point P (2, 3) with translation vector (4, 3) and (−1, 2) respectively.

We know,

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

$$P' = \begin{bmatrix} 1 & 0 & t_{x1} + t_{x2} \\ 0 & 1 & t_{y1} + t_{y2} \\ 0 & 0 & 1 \end{bmatrix} \cdot P = \begin{bmatrix} 1 & 0 & 4 + (-1) \\ 0 & 1 & 3 + 2 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}$$

$$P' = \begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 5 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 5 \\ 8 \\ 1 \end{bmatrix}$$

Hence, Final Coordinates after translations are P' (5, 8).

Rotations

Two successive Rotations are performed as:

$$\boldsymbol{P' = R(\theta_2) \cdot \{R(\theta_1) \cdot P\}}$$
$$\boldsymbol{P' = \{R(\theta_2) \cdot R(\theta_1)\} \cdot P}$$

$$\boldsymbol{P' = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 \\ \sin\theta_2 & \cos\theta_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot P}$$

$$\boldsymbol{P' = \begin{bmatrix} \cos\theta_2\cos\theta_1 - \sin\theta_2\sin\theta_1 & -\sin\theta_1\cos\theta_2 - \sin\theta_2\cos\theta_1 & 0 \\ \sin\theta_1\cos\theta_2 + \sin\theta_2\cos\theta_1 & \cos\theta_2\cos\theta_1 - \sin\theta_2\sin\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot P}$$

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

Where $\boldsymbol{P'}$ and $\boldsymbol{P}$ are column vector of final and initial point coordinate respectively.

- This shows that **<u>two successive rotation are additive</u>**.
- This concept can be extended for any number of successive rotations.

Example: Obtain the final coordinates after two rotations on point P (6, 9) with rotation angles are 30 and 60 degree respectively.

We have,

$$P' = R\,(\theta_1 + \theta_2) \cdot P$$

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

$$P' = \begin{bmatrix} cos(30 + 60) & -sin(30 + 60) & 0 \\ sin(30 + 60) & cos(30 + 60) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot P$$

$$P' = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 6 \\ 9 \\ 1 \end{bmatrix} = \begin{bmatrix} -9 \\ 6 \\ 1 \end{bmatrix}$$

Hence, Final Coordinates after rotations are P' (−9, 6).

Scaling

Two successive scaling are performed as:

$$\boldsymbol{P' = S(s_{x2}, s_{y2}) \cdot \{S(s_{x1}, s_{y1}) \cdot P\}}$$
$$\boldsymbol{P' = \{S(s_{x2}, s_{y2}) \cdot S(s_{x1}, s_{y1})\} \cdot P}$$

$$\boldsymbol{P' = \begin{bmatrix} s_{x2} & 0 & 0 \\ 0 & s_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_{x1} & 0 & 0 \\ 0 & s_{y1} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot P}$$

$$\boldsymbol{P' = \begin{bmatrix} s_{x1} \cdot s_{x2} & 0 & 0 \\ 0 & s_{y1} \cdot s_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot P}$$

$$\boldsymbol{P' = S(s_{x1} \cdot s_{x2},\ s_{y1} \cdot s_{y2}) \cdot P}$$

Where, $\boldsymbol{P'}$ and $\boldsymbol{P}$ are column vector of final and initial point coordinate respectively.

- ❖ This shows that **<u>two successive scaling are multiplicative</u>**.
- ❖ This concept can be extended for any number of successive scaling.

Example: Obtain the final coordinates after two scaling on line PQ [P (2, 2), Q (8, 8)] with scaling factors are (2, 2) and (3, 3) respectively.

$$P' = S\ (s_{x1} \cdot s_{x2},\ s_{y1} \cdot s_{y2}) \cdot P$$

$$P' = \begin{bmatrix} s_{x1} \cdot s_{x2} & 0 & 0 \\ 0 & s_{y1} \cdot s_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot P = \begin{bmatrix} 2 \cdot 3 & 0 & 0 \\ 0 & 2 \cdot 3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot P$$

$$P' = \begin{bmatrix} 6 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 & 8 \\ 2 & 8 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 12 & 48 \\ 12 & 48 \\ 1 & 1 \end{bmatrix}$$

Hence, Final Coordinates after rotations are P' (12, 12) and Q' (48, 48).

Fixed Point Rotation

- ❖ Also called *<u>Rotation of a point about an arbitrary pivot position</u>*

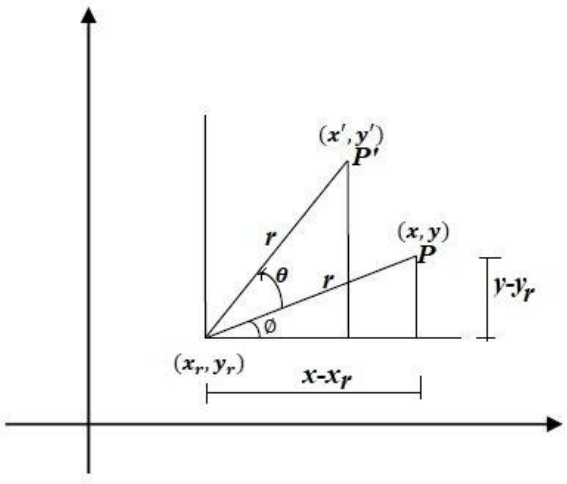


Figure: Rotation about an arbitrary pivot point

- Transformation equation for rotation of a point about pivot point (x_r, y_r) is calculated in the similar way as rotation about an origin and they are:

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

- These equations are differing from rotation about origin and its matrix representation is also different.

For rotating object about arbitrary point (called pivot point) we need to apply following sequence of transformation.

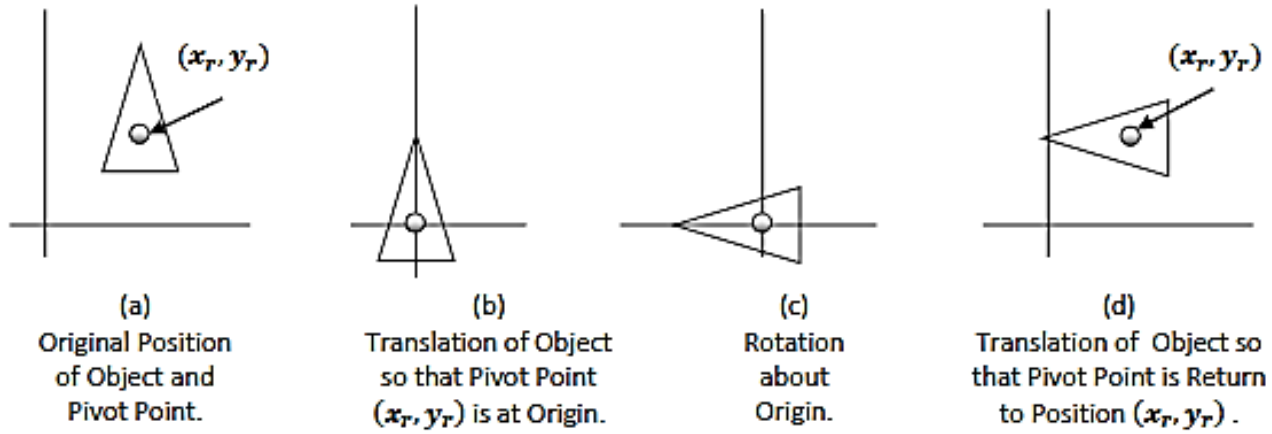


Figure: General Fixed point Rotation

Let P (x, y) is rotated to new position P' (x', y') by an angle θ about pivot point (x_r, y_r) then P' (x', y') can be calculated as:

DO

1. Translate the object so that the pivot-point coincides with the coordinate origin.
2. Rotate the object about the coordinate origin with specified angle.
3. Translate the object so that the pivot-point is returned to its original position (i.e. Inverse of step-1).

Composite Matrix $(CM) = T.R_{\theta}.T^{-1}$

$$CM = T_{(x_r, y_r)}(R_{\theta})\ T_{(-x_r, -y_r)}$$

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

$$= \begin{bmatrix} \cos\theta & -\sin\theta & x_r(1-\cos\theta) + y_r\sin\theta \\ \sin\theta & \cos\theta & y_r(1-\cos\theta) - x_r\sin\theta \\ 0 & 0 & 1 \end{bmatrix}$$

Therefore,

P'= CM*P , Which is required expression

Where P' and P are column vector of final and initial point coordinate respectively and (x_r, y_r) are the coordinates of pivot-point.

Example: - Locate the new position of the triangle [A (5, 4), B (8, 3), C (8, 8)] after its rotation by 90^0 Clockwise about the centroid.

Solution:

Pivot point is centroid of the triangle so:

$$x_r = \frac{5+8+8}{3} = 7\,, \qquad y_r = \frac{4+3+8}{3} = 5$$

As rotation is clockwise we will take $\theta = -90°$.

$P' = R(x_r, y_r, \theta) \cdot P$

We have,

Composite Matrix $(CM) = T.R_{\theta}.T^{-1}$

$$= T_{(x_r, y_r)}(R_{\theta})\ T_{(-x_r, -y_r)}$$

$$= \begin{bmatrix} 1 & 0 & 7 \\ 0 & 1 & 5 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(-90) & -\sin(-90) & 0 \\ \sin(-90) & \cos(-90) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -7 \\ 0 & 1 & -5 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 12 \\ 0 & 0 & 1 \end{bmatrix}$$

Then, P'= CM * P

$$= \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 12 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 5 & 8 & 8 \\ 4 & 3 & 8 \\ 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 6 & 5 & 10 \\ 7 & 4 & 4 \\ 1 & 1 & 1 \end{bmatrix}$$

Hence, final coordinates after rotation are A' (6, 7), B' (5, 4), C' (10, 4)

Or directly you can solve as:

$P' = R(x_r, y_r, \theta) \cdot P$

$$P' = \begin{bmatrix} \cos\theta & -\sin\theta & x_r(1-\cos\theta) + y_r \sin\theta \\ \sin\theta & \cos\theta & y_r(1-\cos\theta) - x_r \sin\theta \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 5 & 8 & 8 \\ 4 & 3 & 8 \\ 1 & 1 & 1 \end{bmatrix}$$

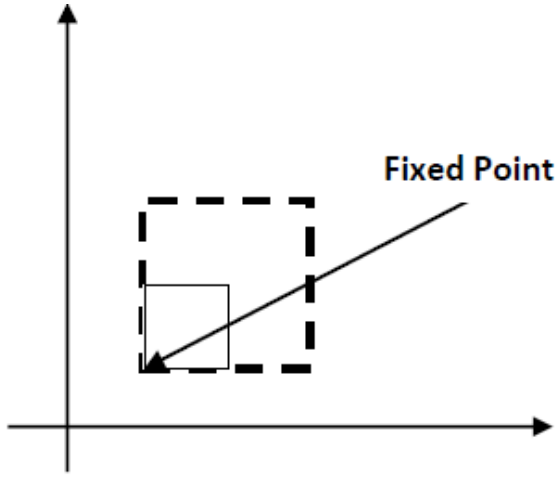
$$P' = \begin{bmatrix} \cos(-90) & -\sin(-90) & 7(1-\cos(-90)) + 5\sin(-90) \\ \sin(-90) & \cos(-90) & 5(1-\cos(-90)) - 7\sin(-90) \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 5 & 8 & 8 \\ 4 & 3 & 8 \\ 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 6 & 5 & 10 \\ 7 & 4 & 4 \\ 1 & 1 & 1 \end{bmatrix}$$

final coordinates after rotation are A' (6, 7), B' (5, 4), C' (10, 4)

Fixed point scaling:

We can control the position of object after scaling by keeping one position fixed called **Fix point** (x_f, y_f) that point will remain unchanged after the scaling transformation



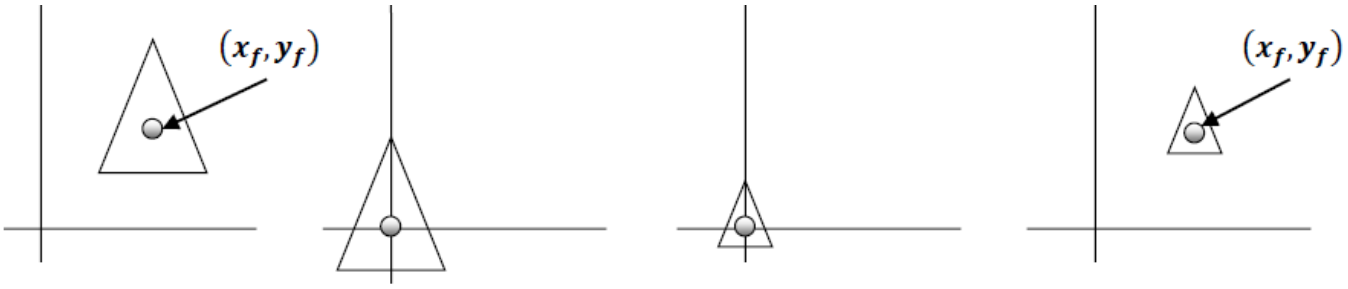
Equation for scaling with fixed point position as (xf, yf) is:

$x' = x_f + (x - x_f)sx$ $\quad$ $y' = y_f + (y - y_f)sy$

$x' = x_f + xsx - x_fsx$ $\quad$ $y' = y_f + ysy - y_fsy$

$x' = xsx + x_f(1 - sx)$ $\quad$ $y' = ysy + y_f(1 - sy)$

For scaling object about arbitrary point (called pivot point) we need to apply following sequence of transformation



(a) Original Position of Object and Fixed Point

(b) Translate Object so that Fixed Point (x_f, y_f) is at Origin

(c) Scale Object with Respect to Origin

(d) Translate Object so that Fixed Point is Return to Position (x_f, y_f) .

Let P (x, y) is Scaled to new position P' (x', y') about pivot point (x_f, y_f) then P' (x', y') can be calculated as:

1. Translate the object so that the fixed-point coincides with the coordinate origin.
2. Scale the object with respect to the coordinate origin with specified scale factors.
3. Translate the object so that the fixed-point is returned to its original position (i.e. Inverse of step-1).

Composite Matrix (CM) = **T. S. T^{-1}**

$$= \boldsymbol{T}(x_f, y_f) \cdot \boldsymbol{S}(sx, sy) \cdot \boldsymbol{T}(-x_f, -y_f)$$

$$\begin{bmatrix} 1 & 0 & x_f \\ 0 & 1 & y_f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_f \\ 0 & 1 & -y_f \\ 0 & 0 & 1 \end{bmatrix}$$

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

Therefore,

P'= CM *P

Here $\boldsymbol{P'}$ and $\boldsymbol{P}$ are column vector of final and initial point coordinate respectively and (x_f, y_f) are the coordinates of fixed-point.

$X_f, y_f = (x_1+x_2 / 2 , y_1+y_2 / 2)$

Example: - Consider square with left-bottom corner at (2, 2) and right-top corner at (6, 6) apply the Transformation which makes its size half such that its center remains same.

As we want size half so value of scale factor are $s_x = 0.5$, $s_y = 0.5$ and Coordinates of square are [A (2, 2), B (6, 2), C (6, 6), D (2, 6)].

Fixed point is the center of square so:

$x_f, y_f = (x_1+x_2) / 2$ and $(y_1+y_2) / 2$ (calculate the mid-point of diagonal)

$= (4, 4)$

$P' = S(x_f , y_f , s_x, s_y) \cdot P$

$$P' = \begin{bmatrix} s_x & 0 & x_f(1-s_x) \\ 0 & s_y & y_f(1-s_y) \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 6 & 6 & 2 \\ 2 & 2 & 6 & 6 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$P' = \begin{bmatrix} 0.5 & 0 & 4(1-0.5) \\ 0 & 0.5 & 4(1-0.5) \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 6 & 6 & 2 \\ 2 & 2 & 6 & 6 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$P' = \begin{bmatrix} 0.5 & 0 & 2 \\ 0 & 0.5 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 6 & 6 & 2 \\ 2 & 2 & 6 & 6 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$P' = \begin{bmatrix} 3 & 5 & 5 & 3 \\ 3 & 3 & 5 & 5 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Hence, Final coordinate after scaling are [A' (3, 3), B' (5, 3), C' (5, 5), D' (3, 5)]

OTHER TRANSFORMATION:

- ✓ Two types

1. Shear
2. Reflection

Shear

- ❖ A transformation that distorts the shape of an object is called **shear**.
- ❖ Two common shearing transformations are:

  - X-shear
  - Y-shear

X-Shear

- ❖ Changes are made to the x coordinate and preserves the y coordinates, which causes the vertical line to tilt right or left as shown in figure below:

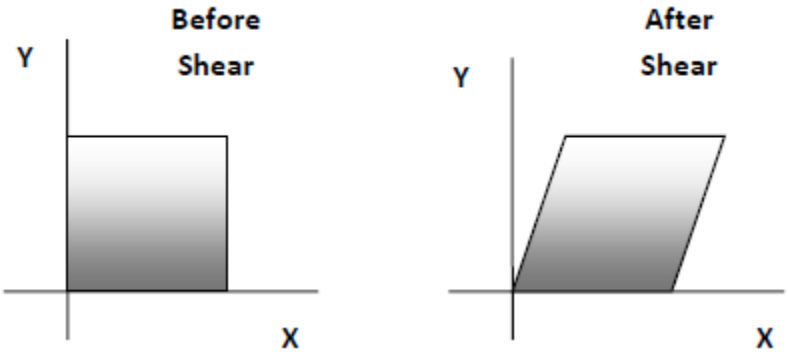
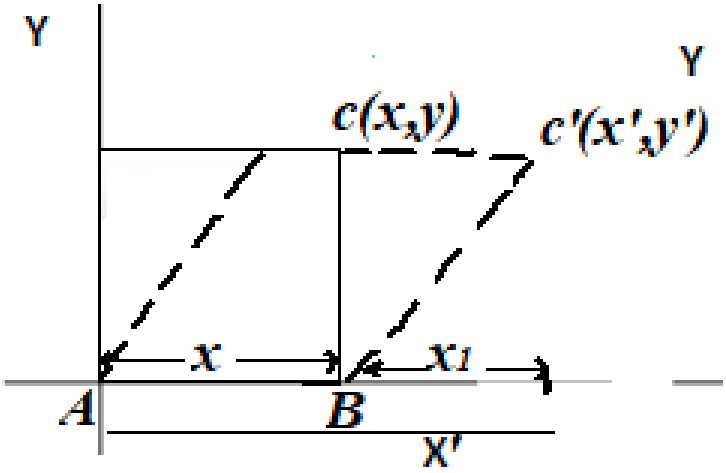


Figure: Share in X-direction



Shearing towards x direction / relative to x axis is given by

Y' = y
X' = x + x_1, **where x_1 α y** (it means coordinate position is shifted horizontally by an amount proportional to its distance y value from the axis)
And $x_1 = Sh_x .y$ Where, **Sh_x** is shearing constant and is any real number.

Therefore,

$x' = x + shx \cdot y$,
$y' = y$

Representation in matrix form,

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

Hence, **P' = Sh_x * P**

Y-Shear

- Changes are made to the y coordinate and preserves the x coordinates, which causes the horizontal line to transform into which slopes up or down as shown in figure below:

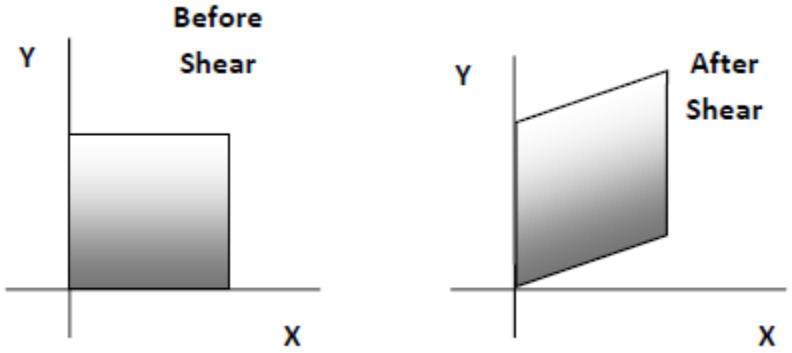


Figure: Y-share

Shearing towards y direction / relative to y axis is given by

X' = x,
Y' = y + Sh_y . x

Representation in matrix form,

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

Hence, **P' = Sh_y * P**

Shear in both direction:

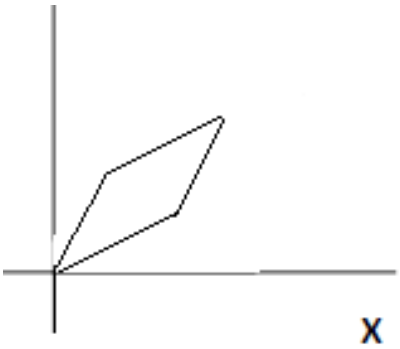


Figure: Shearing towards both (x and y) direction

- No co-ordinates preserved, both changed

$$X' = x + Sh_x \cdot y$$

$$Y' = y + Sh_y \cdot x$$

Representing in matrix form,

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

X-direction Shearing relative to other reference line:

We can generate $x - direction$ shear relative to other reference line $y = y_{ref}$ with following equation:

$\boldsymbol{x' = x + shx \cdot (y - yref)}$,

$\boldsymbol{y' = y}$

Matrix form,

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & Shx & -Shx \,.\, y\,ref \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Y-direction Shearing relative to other reference line:

We can generate $y - direction$ shear relative to other reference line $x = xref$ with following equation:

$\boldsymbol{x' = x}$,

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

In matrix form,

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

Numerical:

Problem-01: Shear the unit square in x direction with shear parameter ½ relative to line $y = -1$.

Solution:

Here $yref = -1$ and $shx = 0.5$

Coordinates of unit square are [A (0, 0), B (1, 0), C (1, 1), D (0, 1)]

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

$$P' = \begin{bmatrix} 1 & 0.5 & -0.5 \cdot (-1) \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$P' = \begin{bmatrix} 1 & 0.5 & 0.5 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$P' = \begin{bmatrix} 0.5 & 1.5 & 2 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Hence, Final coordinate after shear are [A' (0.5, 0), B' (1.5, 0), C' (2, 1), D' (1, 1)]

Problem-02: Shear the unit square in y direction with shear parameter ½ relative to line $x = -1$.

Solution:

Here, $xref = -1$ and

$shy = 0.5$

Coordinates of unit square are [A (0, 0), B (1, 0), C (1, 1), D (0, 1)].

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

$$P' = \begin{bmatrix} 1 & 0 & 0 \\ 0.5 & 1 & -0.5 \cdot (-1) \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$P' = \begin{bmatrix} 1 & 0 & 0 \\ 0.5 & 1 & 0.5 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$P' = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0.5 & 1 & 2 & 1.5 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Hence, Final coordinate after share are [A' (0, 0.5), B' (1, 1), C' (1, 2), D' (0, 1.5)]

Reflection:

- Transformation that produces a mirror image of an object is reflection

  OR

  Providing a mirror image about an axis of an object is called reflection
- Reflection is a rotation operation with 180^0

  OR

  The mirror image for a two –dimensional reflection is generated relative to an ***axis of reflection*** by rotating the object 180o about the reflection axis
- In reflection size of object do not change.
- Basically, are of 3 types
  - About x-axis
  - About y-axis
  - About both (x, y) axis

Reflection about $the\ x\ axis$ (i.e. $y = 0$ line)

- In this Reflection
  - X- coordinate position same
  - Y- Coordinate position flip (change the sign)

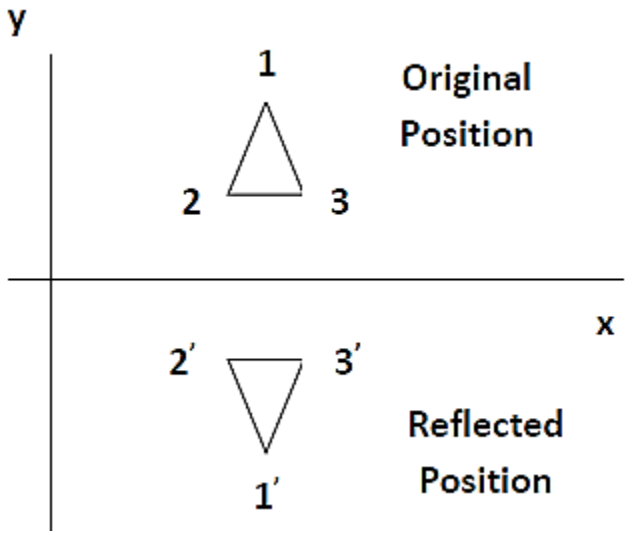


Figure: Reflection about x - axis.

- Matrix Representation:

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

Hence, **P' = R_{fx} . P**

Reflection about the *the y axis* (i.e. line $x = 0$)

- In this Reflection
  - X- Coordinate position flip (change the sign)
  - Y- coordinate position same

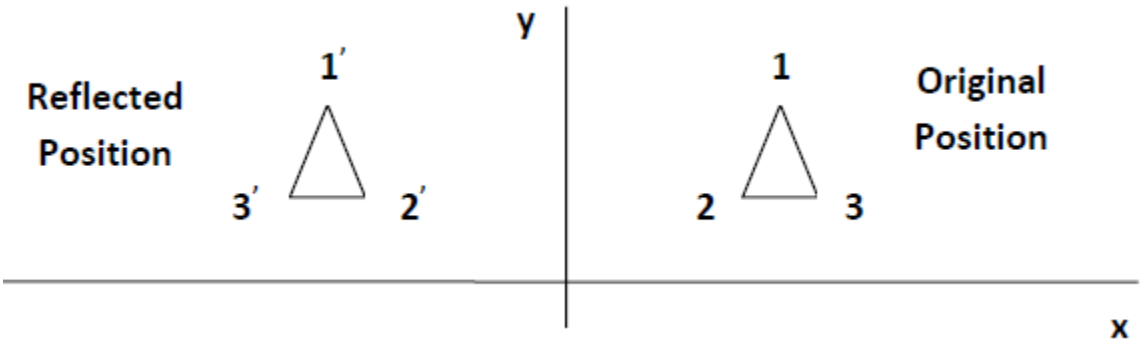


Figure: Reflection about y-axis

In matrix form,

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

Hence, P' = R_{fy} . P

Reflection about the *Origin*.

- In this Reflection
  - X- Coordinate position flip (change the sign)
  - Y- coordinate position flip
  - i.e. both x and y coordinate flip

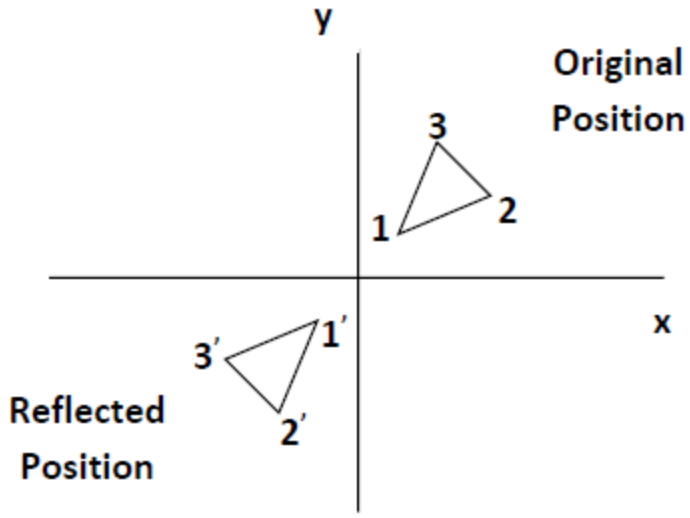


Figure: Reflection about Origin

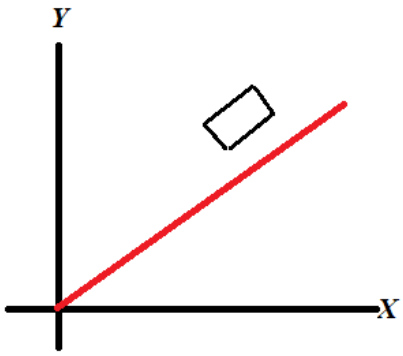
Matrix form,

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

Reflection on any arbitrary axis:

- Any chosen axis within the coordinate system is arbitrary axis.
- Some arbitrary axis and reflection on them are as follows:

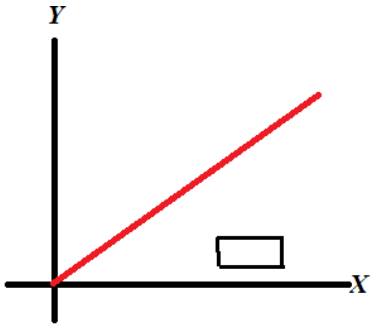
Reflection about line y = x (i.e. θ = 45^0)



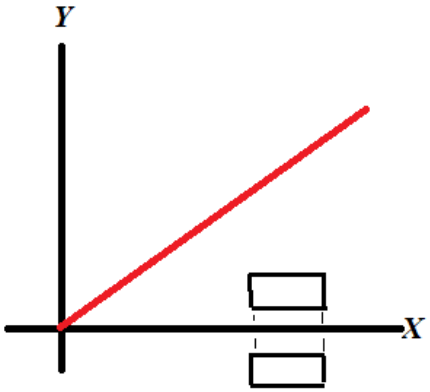
Do:

1. Rotate about origin in Clockwise direction by 45^0

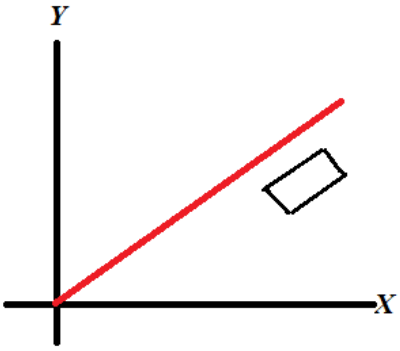
   (This rotates the line y = x to x- axis)



2. Take Reflection against X-axis



3. Rotate in anticlockwise direction by same angle



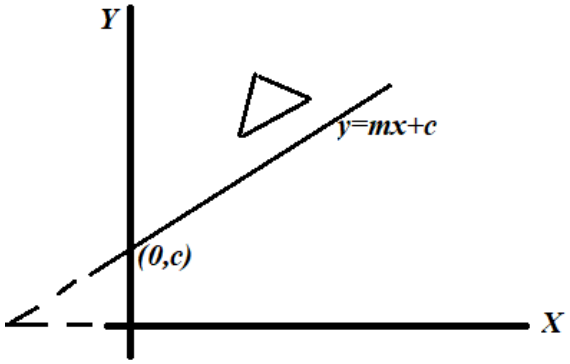
Therefore,

$R_{f(y = x)} = R_{(\theta)} * R_{fx} * R_{(\theta)}$

Reflection about y = mx + b

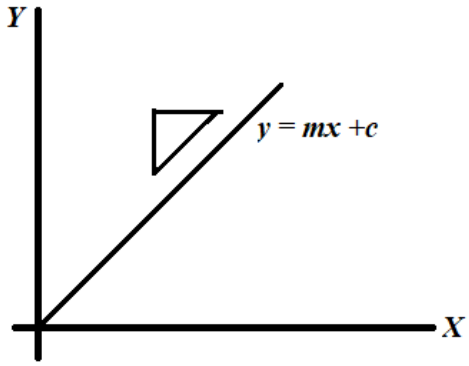
Here, initial position of object at

y= mx + c

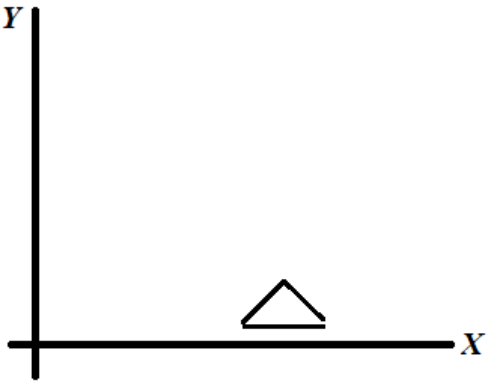


Do:

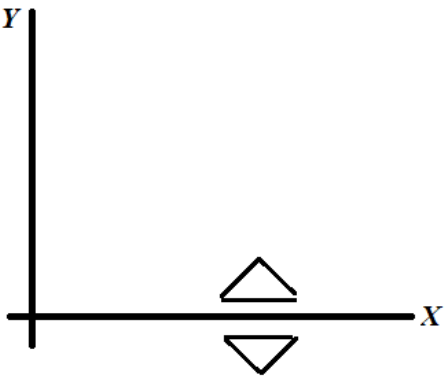
1. Translate the line and object so that the line passes through origin.



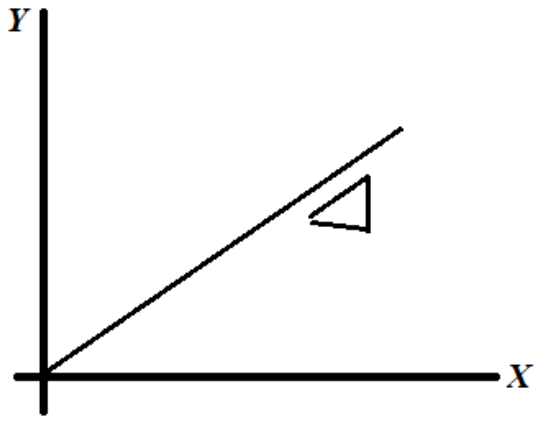
2. Rotate the line and object about origin until the line coincides with one of the coordinate axis



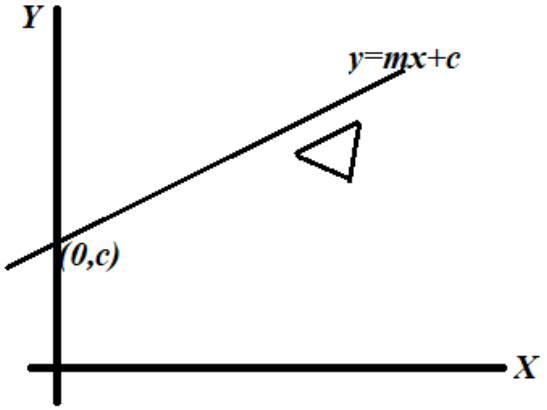
3. Reflect the object through about that axis



4. Apply the Inverse Rotation



5. Translate back to original Position



Therefore,

Composite Transformation Matrix for the Reflection is:

$CM = T_{(0,c)} . R_{\theta} . R_x . R_{-\theta} . T_{(0,-c)}$

Here, line y=mx+c

Slope (m) = tan θ

We have,

$\cos^2 \theta = 1 / (\tan^2 \theta + 1) = 1/ (m^2 + 1)$

$$\cos \theta = \frac{1}{\sqrt{m^2+1}}$$

Similarly,

$\text{Sin}^2\,\theta + \cos^2\,\theta = 1$

$\text{Sin}^2\,\theta = 1 - \cos^2\,\theta = 1 - \frac{1}{m^2+1} = \frac{m^2+1-1}{m^2+1}$

$\text{Sin}\,\theta = \frac{m}{\sqrt{m^2+1}}$

Hence, CM = $\mathbf{T_{(0,c)} \cdot R_{\theta} \cdot R_x \cdot R_{-\theta} \cdot T_{(0,-c)}}$

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

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

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

$$= \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & c\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\frac{1}{\sqrt{(m^2+1)}} & \frac{-m}{\sqrt{(m^2+1)}} & 0\\ \frac{-m}{\sqrt{(m^2+1)}} & \frac{1}{\sqrt{(m^2+1)}} & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\frac{1}{\sqrt{(m^2+1)}} & \frac{m}{\sqrt{(m^2+1)}} & \frac{-cm}{\sqrt{(m^2+1)}}\\ \frac{m}{\sqrt{(m^2+1)}} & \frac{-1}{\sqrt{(m^2+1)}} & \frac{c}{\sqrt{(m^2+1)}}\\ 0 & 0 & 1\end{bmatrix}$$

$$= \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & c\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\frac{1-m^2}{m^2+1} & \frac{2m}{m^2+1} & \frac{-2cm}{m^2+1}\\ \frac{2m}{m^2+1} & \frac{m^2-1}{m^2+1} & \frac{c-cm^2}{m^2+1}\\ 0 & 0 & 1\end{bmatrix}$$

$$= \begin{bmatrix}\frac{1-m^2}{m^2+1} & \frac{2m}{m^2+1} & \frac{-2mc}{m^2+1}\\ \frac{2m}{m^2+1} & \frac{m^2-1}{m^2+1} & \frac{2c}{m^2+1}\\ 0 & 0 & 1\end{bmatrix}$$

Hence, **P' = CM * P**

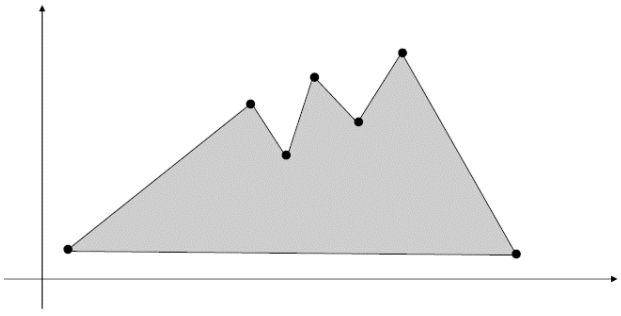
Numerical Session:

2 Dimensional Viewing

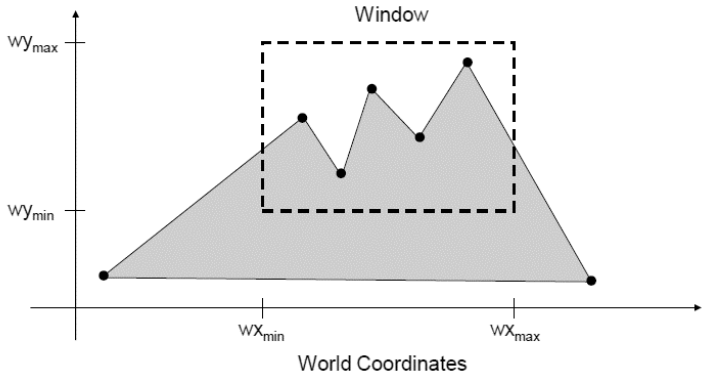
- Two Dimensional viewing is the formal mechanism for displaying views of a picture on an output device.
- Typically, a graphics package allows a user to specify which part of a defined picture is to be displayed and where that part is to be placed on the display device.
- Any convenient Cartesian coordinate 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 coordinates 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.

Windowing Concept:

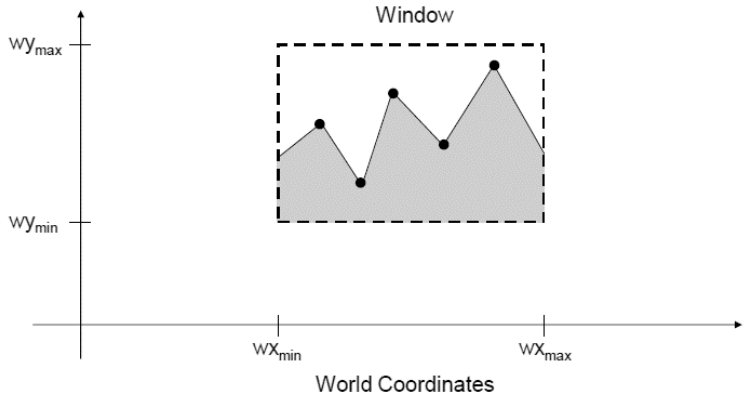
- A scene is made up of a collection of objects specified in world coordinates



- When we display a scene only those objects within a particular window are displayed



- Because drawing things to a display takes time we *clip* everything outside the window



Key Notes:

- 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**. In other words we can say that viewport is part of computer screen. It defines where to display.
- In many case window and viewport are rectangle, also other shape may be used as window and viewport.
- In general finding device coordinates of viewport from word coordinates of window is called as **viewing transformation.**
- The window defines what is to be viewed, the viewport defines where it is to be displayed.
- In computer graphics terminology, the term window originally referred to an area of a picture that is selected for viewing.
- Windows and viewport are rectangular in standard position, with the rectangular edge parallel to co-ordinate axis.

- ❖ The world window specifies which part of the world should be drawn. Whichever part lies inside the window should be drawn and whichever part lies outside should be clipped away and not drawn.
- ❖ OpenGL does the clipping automatically
- ❖ A mapping between world window and viewport is established by OpenGL
- ❖ OpenGL converts our coordinates to screen coordinates when we set up a screen window and viewport

Multiple Coordinates System

In a typical graphics program, we may need to deal with a number of different coordinate systems, and a good part of the work (and the cause of many headaches) is the conversion of coordinates from one system to another. Here is a list of some of the coordinate systems you may encounter.

1. Modelling coordinate/local/master coordinate (MC)

The shape of the individual object can be constructed in a scene within separate coordinate reference frame, is called Model Coordinate System.

It is used to define coordinates that are used to construct the shape of individual parts (objects) of a 2D scene.

2. World coordinate system (WC)

A Model Coordinate System is the unique coordinate space of the model. Two distinct models, each with their own coordinate systems can't interact with each other. There needs to be a universal coordinate system that allows each model to interact with each other. This universal system is called World Coordinate System. For interaction to occur, the coordinate system of each model is transformed into the World Coordinate System

Also known as the "universe" or sometimes "model" coordinate system. This is the base reference system for the overall model, (generally in 3D), to which all other model coordinates relate.

OR,

Once individual objects have been identified or specified, those objects are placed into appropriate position within scene using reference frame is called world coordinate. It contains many objects with one dimension.

3. Viewing coordinate system (VC)

Used to define window in the world coordinate plane with any possible orientation, viewing some object or item at a time
Viewing co-ordinate system are used to define particular view of a 2D scene. Translation, scaling, and rotation of the window will generate a different view of the scene. For a 2-D picture, a view is selected by specifying a sub area (window) of the total picture area.

4. Device coordinate (DC)

Device co-ordinate are used to define coordinates in an output device. They are integers within the range (0, 0) to (x_{max}, y_{max}) for a particular output device.

OR,

When the world coordinate description of the scene is transformed to one or more output device reference frame for display, these coordinate system are referred to a device coordinate or screen coordinate in the case of video monitor.

5. Device coordinate/normalize device coordinate (NDC)

NVC‘s are used to make the viewing process independent of the output device (monitor, mobile, hard copy devices). Normally, the value of NVC is 0 to 1.

OR,

The coordinate are in range 0 to 1.

Generally a graphical system, first convert, world coordinate position to normalize device coordinate before final conversion to specified device coordinated. This makes the system independent of the various devices that might be used at a particular workstation.

Window to Viewport Transformation

Window to Viewport Transformation is the process of transforming a 2D world-coordinate objects to device coordinates. Objects inside the world or clipping window are mapped to the viewport which is the area on the screen where world coordinates are mapped to be displayed.

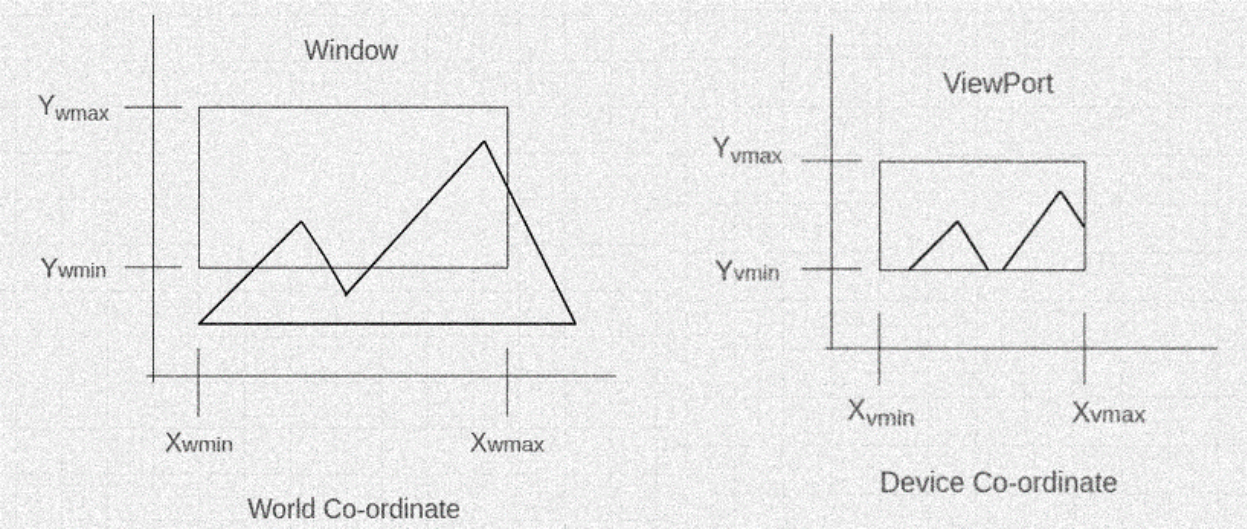


Figure: Mapping of picture section falling under rectangular Window onto a designated rectangular view port (i.e. viewing transformation)

It may be possible that the size of the Viewport is much smaller or greater than the Window. In these cases, we have to increase or decrease the size of the Window according to the Viewport.

Here,

World coordinate – It is the Cartesian coordinate w.r.t which we define the diagram, like X_{wmin}, X_{wmax}, Y_{wmin}, Y_{wmax}

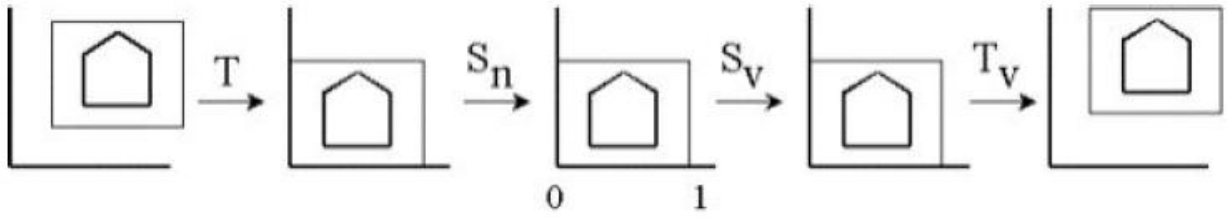
Device Coordinate –It is the screen coordinate where the objects is to be displayed, like X_{vmin}, X_{vmax}, Y_{vmin}, Y_{vmax}

Procedure for to transform a window to the view port we have to perform the following steps:

Step1: The object together with its window is translated until the lower left corner of the window is at the origin

Step2: The object and window are scaled until the window has the dimensions of the view port

Step3: Again translate to move the view port to its correct position on the screen (Setup Window,Translate window, Scale to normalize, Scale to view port, Translate to View port)



Application of windows to viewport transformation

i. By changing the position of the view port, we can view objects at different positions on the display area of an output device.
ii. ii. By varying the size of view ports, we can change size of displayed objects.
iii. iii. Zooming effects can be obtained by successively mapping different-sized windows on a fixed-sized view port
iv. iv. Panning effects (Horizontal Scrolling) are produced by moving a fixed-size window across the various objects in a scene.

Viewing Transformation pipeline

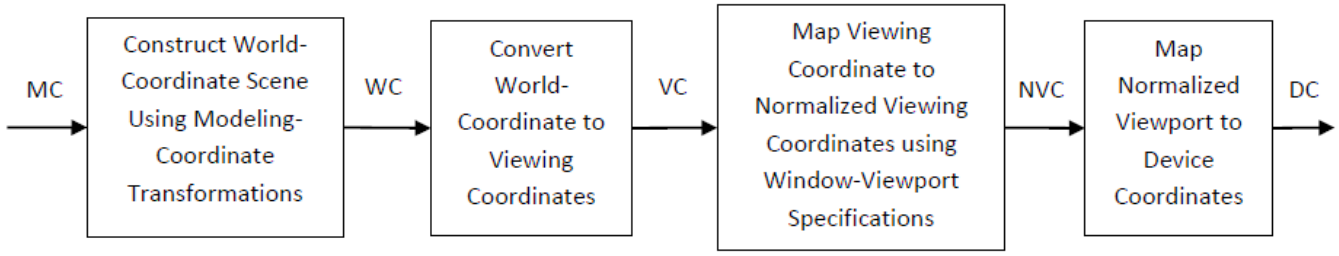


Figure: 2D viewing Pipeline

As shown in figure above first of all construct world coordinate scene using modeling coordinate transformation.

- ❖ After this, convert viewing coordinates from world coordinates using window to viewport transformation.
- ❖ Then map the viewing coordinate to normalized viewing coordinate in which we obtain values in between 0 to 1.

- At last, convert the normalized viewing coordinate to device coordinate using device driver software which provide device specification.
- Finally device coordinate is used to display image on display screen.
- By changing the viewport position on screen we can see image at different place on the screen.
- By changing the size of the window and viewport we can obtain zoom in and zoom out effect as per requirement.
- Fixed size viewport and small size window gives zoom in effect, and fixed size viewport and larger window gives zoom out effect.
- View ports are generally defines with the unit square so that graphics package are more device independent which we call as normalized viewing coordinate.

Window-To-Viewport Coordinate Transformation (mapping)

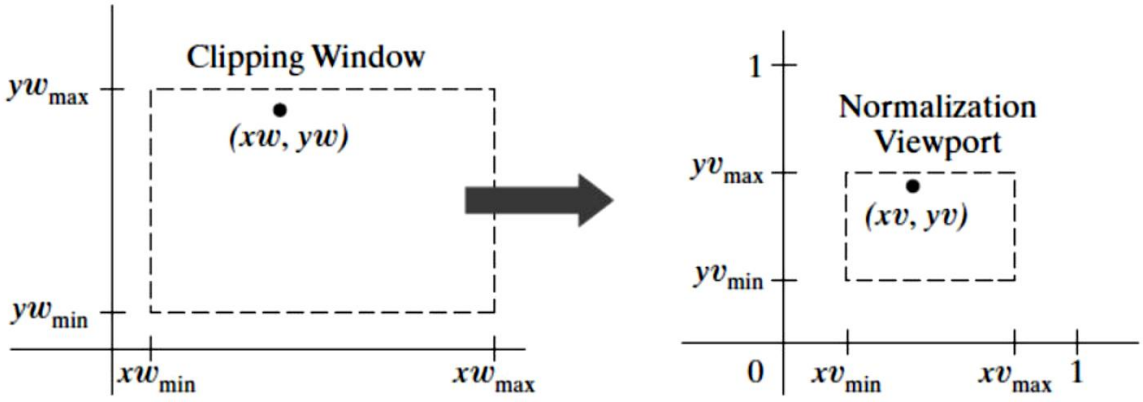


Figure: Windows to viewport mapping

- **A** point at position (x_w, y_w) in the designated window is mapped to viewport coordinate position (x_v, y_v) to maintain the same relative placement in the viewport as in the window
- A window can be specified by four world coordinates: x_{wmin}, x_{wmax}, Y_{wmin} and Y_{wmax}.
- Similarly a view port can be described by four device coordinates: x_{vmin}, x_{vmax}, Y_{vmin} and Y_{vmax}

To maintain the same relative placement in the viewport (or to calculate the point (x_V, y_V)) as in window the following proportional ratios must be equal.

Now, the relative position of the object in window and viewport are same

Then, for X-coordinate,

$$\frac{x_v - x_{vmin}}{x_{vmax} - x_{vmin}} = \frac{x_w - x_{wmin}}{x_{wmax} - x_{wmin}}$$

And, for Y- coordinate

$$\frac{y_v - y_{vmin}}{y_{vmax} - y_{vmin}} = \frac{y_w - y_{wmin}}{y_{wmax} - y_{wmin}}$$

Solving these or after calculating X and Y coordinate, we get.

$$x_v = x_{vmin} + (x_w - x_{wmin})s_x$$
$$y_v = y_{vmin} + (yw - y_{wmin})s_y$$

Where, s_x is scaling factor of x coordinate and s_y is scaling factor of y coordinate and determined by,

$$s_x = \frac{x_{vmax} - x_{vmin}}{x_{wmax} - x_{wmin}}$$
$$s_y = \frac{y_{vmax} - y_{vmin}}{y_{wmax} - y_{wmin}}$$

Workstation transformation.

- Number of display device can be used in application and for each we can use different window to viewport transformation. This mapping is called the **workstation transformation**
- As shown in following figure two different displays devices are used and we map different window-to-viewport on each one.

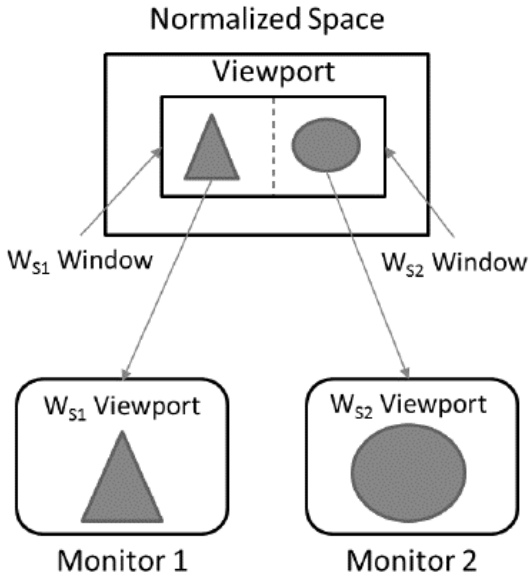


Figure: Workstation Transformation

Numerical

Problem-01: Window port is given by (100,100,300,300) and view port is given by (50, 50,150,150).Convert the window port coordinate (200,200) to the view port coordinate.

Solution:

$(x_{wmin}, y_{wmin}) = (100, 100)$

$(x_{wmax}, y_{wmax}) = (300, 300)$

$(x_{vmin}, y_{vmin}) = (50, 50)$

$(x_{vmax}, y_{vmax}) = (150, 150)$

$(x_w, y_w) = (200,200)$

$(x_v, y_v) = ?$

We know,

$$s_x = \frac{x_{vmax} - x_{vmin}}{x_{wmax} - x_{wmin}}$$

= (150- 50) / (300 - 100) = 0.5

$$s_y = \frac{y_{vmax} - y_{vmin}}{y_{wmax} - y_{wmin}}$$

= (150- 50) / (300 - 100) = 0.5

The equations for mapping window co-ordinate to view port coordinate is given by
$x_v = x_{vmin} + (x_w - x_{wmin})Sx$
$yv = y_{vmin} + (y_w - y_{wmin})Sy$

Hence,

$x_v = 50 + (200 - 100) * 0.5 = 100$
$y_v = 50 + (200 - 100) * 0.5 = 100$

The transformed view port coordinate is (100,100).

Problem-02

Given Coordinate for world coordinate representation are $\mathbf{X_{wmin} = 20}$ $\mathbf{X_{wmax} = 80,}$ $\mathbf{Y_{wmin} = 40,}$ $\mathbf{Y_{wmax} = 80}$ and coordinate of viewport are $\mathbf{X_{vmin} = 30,}$ $\mathbf{X_{vmax} = 60,}$ $\mathbf{Y_{vmin} = 40,}$ $\mathbf{Y_{vmax} = 60}$. A point (X_w, Y_w) be (30, 80) on the window. Calculate same point on viewport.

Solution:

Calculate scaling factor of x coordinate S_x and scaling factor of y coordinate S_y as:

S_x = (60 - 30) / (80 - 20) = 30 / 60
S_y = (60 - 40) / (80 - 40) = 20 / 40

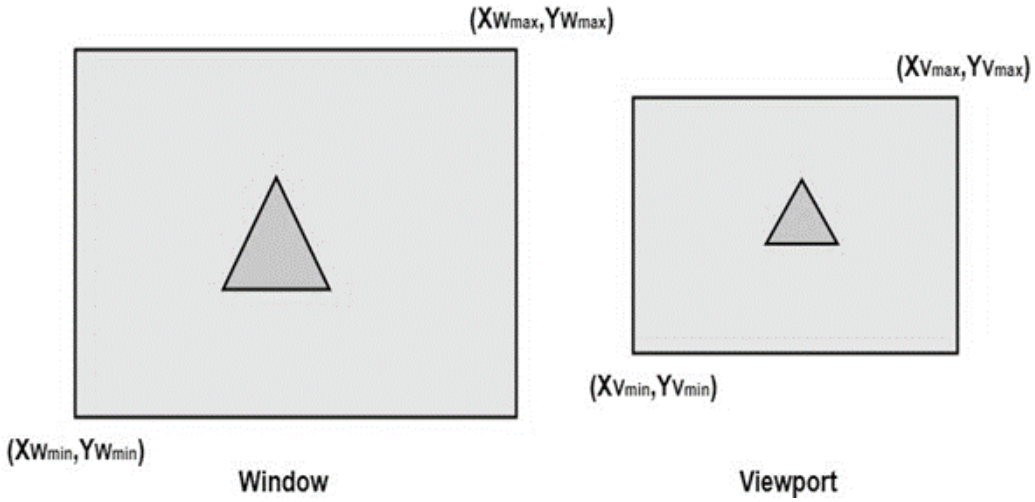
So, now calculate the point on viewport (X_v, Y_v).

X_v = 30 + (30 - 20) * (30 / 60) = 35

$Y_v = 40 + (80 - 40) * (20 / 40) = 60$

So, the point on window (X_w, Y_w) = (30, 80) will be **(X_v, Y_v) = (35, 60)** on viewport

Matrix Representation viewing Transformation



Step1: Translate window to origin

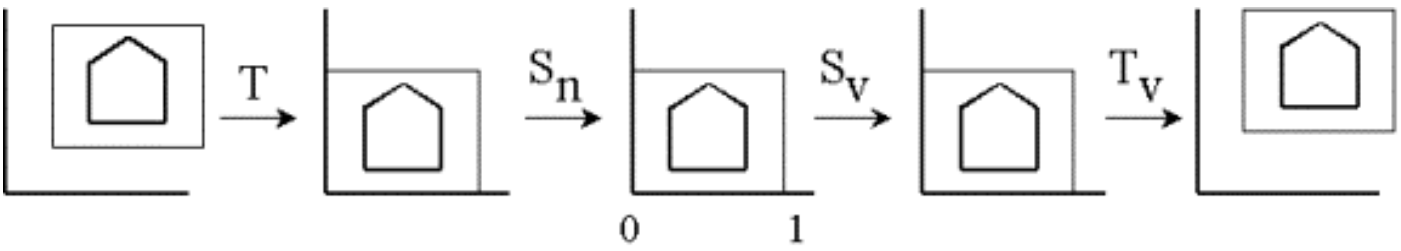
Step2: Scaling of the window to match its size to the viewport

$S_x = (Xy_{max} - Xv_{min}) / (Xw_{max} - Xw_{min})$

$S_y = (Yv_{max} - Yv_{min}) / (Yw_{max} - Yw_{min})$

Step3: Again translate viewport to its correct position on screen. (i.e. inverse of step 1)

Above three steps can be represented in matrix form:



View CM=T * S * T^{-1}

I.e. Use Transformation as:

1. Set up window
2. Translate window
3. Scale to normalize
4. Scale to viewport
5. Translate to Viewport

Clipping:

- Any Procedure that identifies those portions of a picture that are either inside or outside of a specified region of a space is referred to as a **Clipping algorithm** or simply **clipping**.
- The region against which an object is to clip is called a **Clip Window.**
- Clipping is necessary to remove those portions of the objects which are not necessary for further operations.
- Clipping excludes unwanted graphics from the screen.
- Depending on the application, the clip window can be a general polygon or it can even have curved boundaries. But Rectangular clip regions are preferred and standard.
- For the viewing transformation, we want to display only those picture parts that are within the window area. Everything outside the window is discarded.
- Clipping algorithm can be applied in World Coordinates, so that only the contents of the window interior are mapped to Device coordinates.
- So there are three cases
    1. The object may be completely outside the viewing area defined by the window port.
    2. The object may be seen partially in the window port.
    3. The object may be seen completely in the window port
- For case i and ii, clipping operation is necessary but not for case iii.

Application of Clipping

- Extracting part of a defined scene for viewing.
- Identifying visible surfaces in three-dimensional views.
- Creating objects using solid-modeling procedures.
- Displaying a multi window environment.
- Anti-aliasing line segments or object boundaries.

- Drawing and painting operations that allow parts of a picture to be selected for copying, moving erasing, or duplicating.

Types of Clipping (based on different output primitives)

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

Point Clipping:

- In point clipping we eliminate those points which are outside the clipping window and draw points which are inside the clipping window.
- Let's consider clipping window is rectangular boundary with edge (x_{wmin},x_{wmax},y_{wmin},y_{wmax}).
- So for finding whether given point is inside or outside the clipping window we use following inequality:

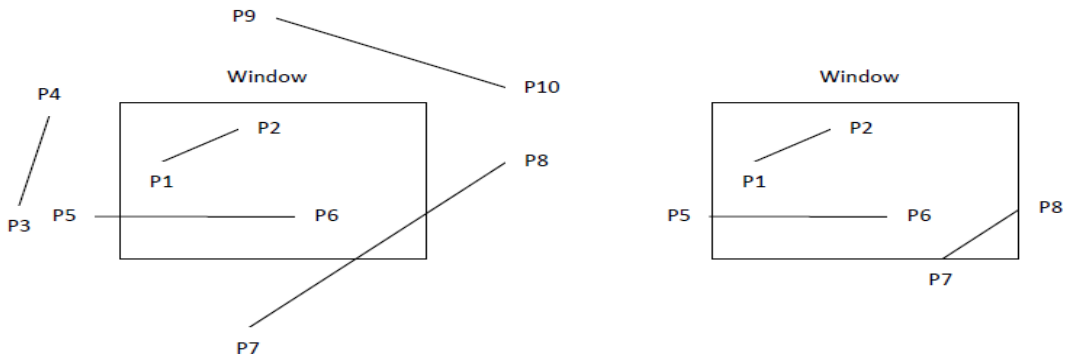
$$xwmin \leq x \leq xwamx$$
$$ywmin \leq y \leq ywamx$$

- If above both inequality is satisfied then the point is inside otherwise the point is outside the clipping window.

Line Clipping

Line clipping involves several possible cases.

1. Completely inside the clipping window.
2. Completely outside the clipping window.
3. partially inside and partially outside the clipping window.

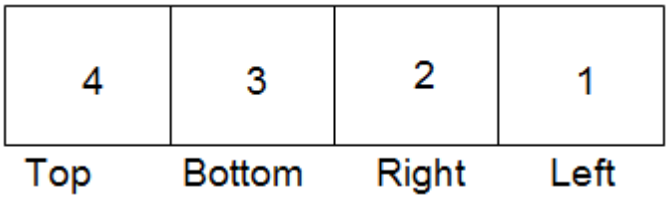


- Line which is completely inside is display completely. Line which is completely outside is eliminated from display. And for partially inside line we need to calculate intersection with window boundary and find which part is inside the clipping boundary and which part is eliminated.

Some of clipping procedure are discuss below.

Cohen-Sutherland Line Clipping

- One of the oldest and most popular line-clipping procedure.
- Danny Cohen and Ivan Sutherland
- Can be used only on a rectangular window
- World space is divide 9 different regions based on the windows boundaries.
- Middle region is the clipping window.
- Each region is assigned a unique 4-bit code, called **region code.**
- Region codes indicate the position of the regions with respect to the window



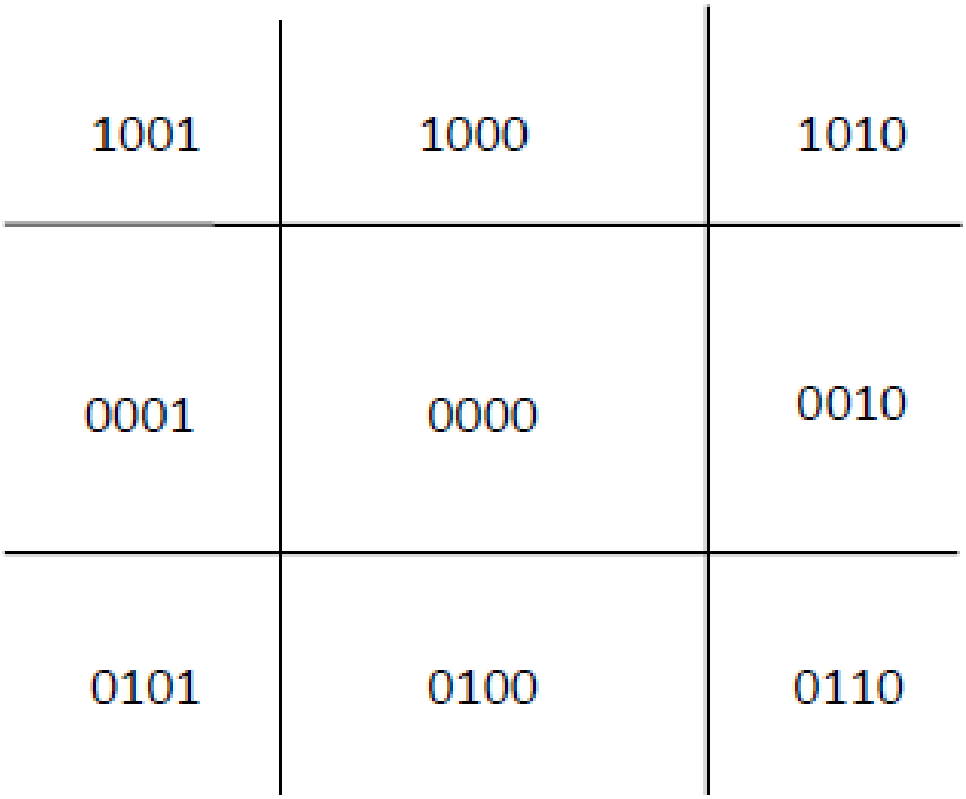
TBRL

- Any point inside the clipping window has a region code 0000.

The rightmost bit is the first bit. The bits are set to 1 based on the following scheme.

1) First bit Set- If the end point is to the left of the window.
2) Second bit Set- If the end point is to the right of the window.
3) Third bit Set- If the end point is to the below the window.
4) Fourth bit Set- If the end point is to the above the window.

Otherwise the bit is set to Zero.

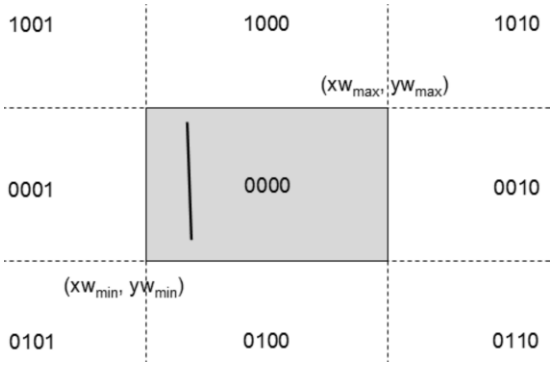


- It is obvious that, if both end point codes are zero, then both ends of the line lie inside the window & the line is visible.

Algorithm:

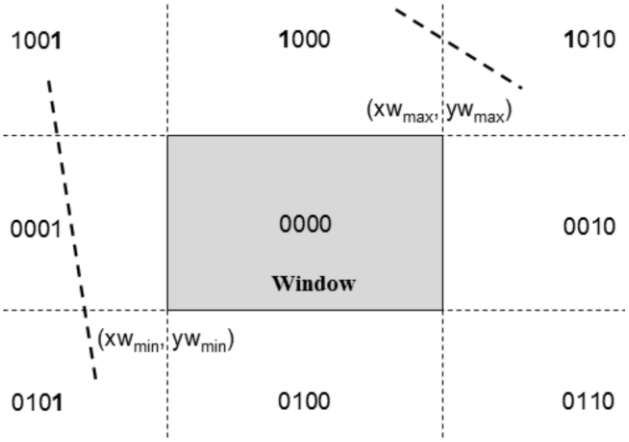
1. Given a line segment with end points $P_1=(x_1,y_1)$ and $P_2(x_2,y_2)$, compute 4 bit region code for each end point.
2. To clip a line, first we have to find out which regions its two endpoints lie in.

Case I:
If both end point code is 0000, then the line segment is completely inside the window. I.e. if bitwise (logical) OR of the codes yields 0000, then the line segment is accepted for display.



Case II:

If the two end point region code both have a 1 in the same bit position, then the line segment lies completely outside the window, so discarded. I.e. if bitwise (logical) AND of the codes not equal to 0000, then the line segment is rejected.



Case III:

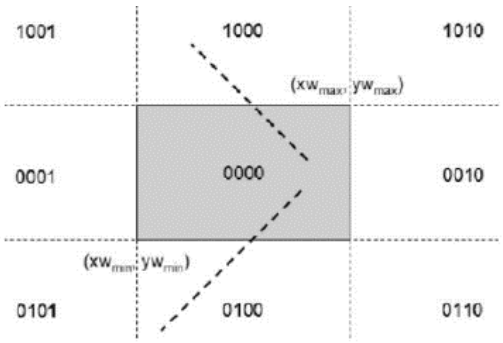
If lines cannot be identified as completely inside or outside we have to do some more calculations.

Here we find the intersection points with a clipping boundary using the slope intercept form of the line equation

Here, to find the visible surface, the intersection points on the boundary of window can be determined as:

$y - y_1 = m\,(x - x_1)$

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



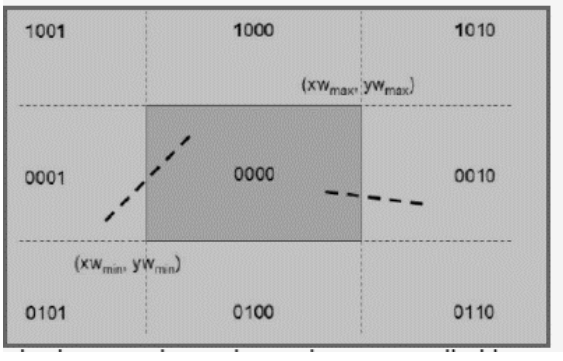
i. If the intersection is with horizontal boundary: $x = x_1 + \frac{y - y_1}{m}$

Where y is set to either y_{wmin} or y_{wmax}.

ii. If the intersection is with vertical boundary :

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

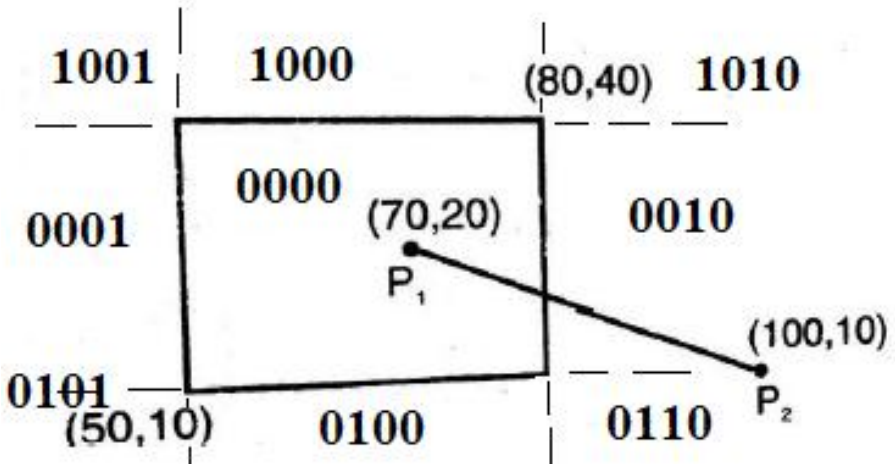
Where x is set to either x_{wmin} or x_{wmax}



3. Assign a new four-bit code to the intersection point and repeat until either cas1 or case2 are satisfied.

Example: Use the Cohen –Sutherland algorithm to clip the line P_1 (70, 20) and P_2 (100, 10) against a window lower left hand corner (50, 10) and upper right hand corner (80, 40)**.**

Solution:



Step-1: Assign 4 bit binary code to the two end point

$P_1 = 0000$ $P_2 = 0010$

Step-2: Calculate bitwise **OR:**

P_1 **|** $P_2 = 0000$ **|** $0010 = 0010$

Since P_1 **|** P_2! = 0000, hence the two point doesn't lies completely inside the window.

Step-3: Finding bitwise **AND**:

P_1 & $P_2 = 0000$ & $0010 = 0000$

Since P_1 & $P_2 = 0000$, hence line is partially visible.

Step- 3.1: Now, to find the intersection of P_1 and P_2 with the boundary of Window,

We have,

$P_1(x_1, y_1) = (70, 20)$
$P_2(x_2, y_2) = (100, 10)$

Slope m = (10-20) / (100-70) = -1/3

We have to find the intersection with right edge of window,

Here, x=80
y=?

We have

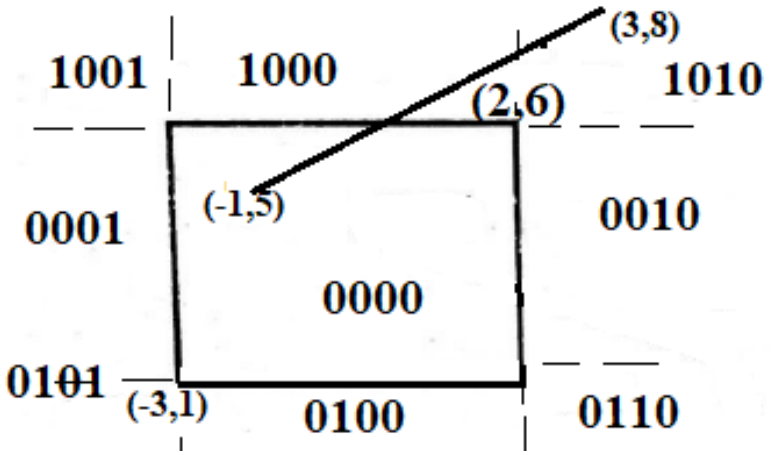
$y = y_2 + m(x-x_2)$
$= 10 + (-1/3)(80-100)$
$= 10 + 6.67$
$= 16.67$

Thus, the intersection point P3 = (80, 16.66), so discarding the line segment that lie outside the boundary i.e. P_3P_2, we get new line P_1P_3 with co-ordinate P_1 (70, 20) and P_3 (80, 16.67)

Problem: clip a line M (-1, 5) and P (3, 8) using Cohen Sutherland line clipping algorithm having window coordinates (-3, 1) and (2, 6). (Classwork)

Answer: point of intersection (1/3, 6) hence (-1, 5) to (1/3, 6) visible

Solution:



From figure,

Region code for two endpoint are $p_1 = 0000$ and $p_2 = 1010$

Step-1: Calculate logical OR

$P_1 \mid p_2 = 0000 \mid 1010 = 1010$
$P_1 \mid p_2 ! = 0000$ therefore, calculate logical AND

Step-2: calculate P_1 & p_2

$P_1 \& p_2 = 0000 \& 1010 = 0000$ i.e. line is partially visible

Step-3: given line intersects in horizontal boundary therefore,

$X = x_1 + (y-y_1) / m$ where, $m = (8-5) (/ 3 +1) = ¾$

Now, $x = (-1) + (6-5) / (3/4) = 1 / 3$

Thus, the intersection point P3 = (1/3, 6) so discarding the line segment that lie outside the boundary i.e. P_3P_2, we get new line P_1P_3 with co-ordinate P_1 (-1, 5) and P_3 (1/3, 6)

Polygon Clipping

- For polygon clipping we need to modify the line clipping procedure because in line clipping we need to consider about only line segment while in polygon clipping we need to consider the area and the new boundary of the polygon after clipping.

Sutherland-Hodgeman Polygon Clipping Algorithm

- Sutherland–Hodgeman algorithm is used for clipping polygons.
- A single polygon can actually be split into multiple polygons.
- The algorithm clips a polygon against all edges of the clipping region in turn.

Process:

- You can take any order but LEFT, RIGHT, BOTTOM, TOP is General.

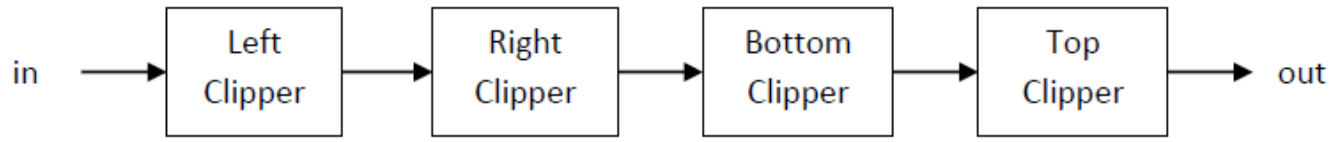


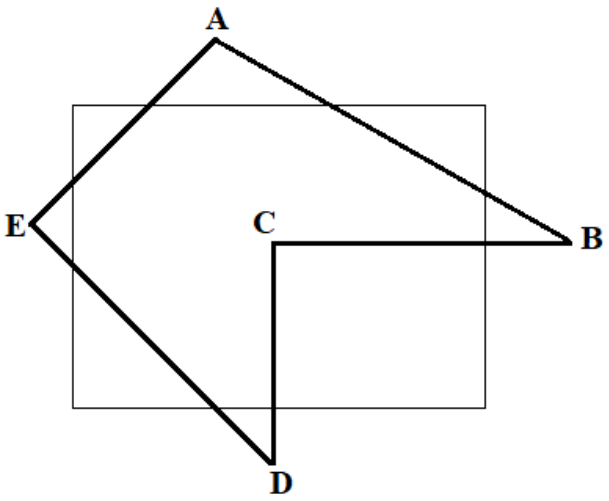
Figure: Processing the vertices of the polygon through boundary clipper

There are four possible cases for any given edge of given polygon against clipping edge.

Cases	Action	Output
1. **Both** Vertices are **inside**	Only the second vertex is added to the output list	Second Vertex
2. **First** vertex is **outside** while **second** vertex is **inside**	Both the point of intersection of the edge with the clip boundary and the second vertex are added to the output list	Second Vertex and Intersection vertex
3. **First** vertex is **inside** while **second** vertex is **outside**	Only the point of intersection of the edge with the clip boundary is added to the output list	Intersection point / Vertex
4. **Both** vertices are **outside**	No vertices are added to the output list	None

Example:

Clip the given polygon using Sutherland-Hodgeman polygon clipping algorithm



Let's proceed algorithm

Solution: on board