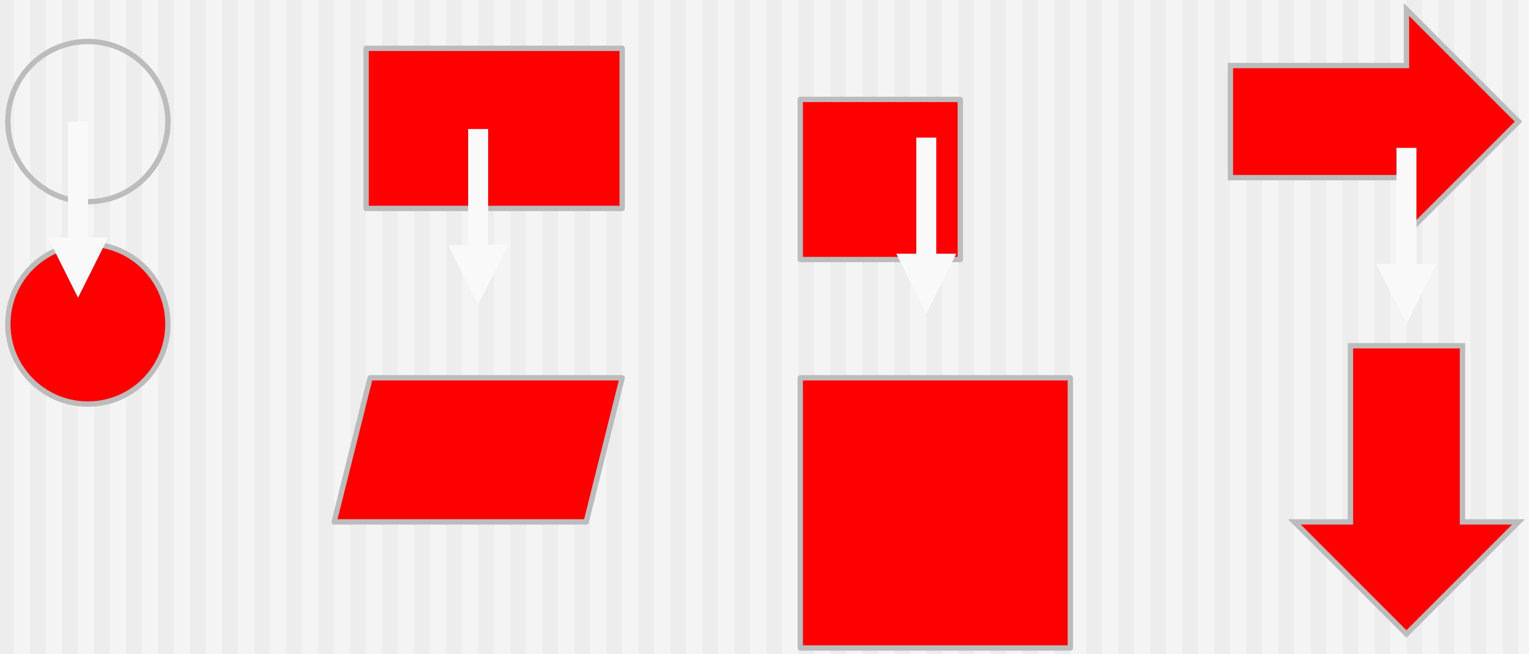
Outline

- Transformation
- Basic transformation
- Matrix representation and homogeneous coordinates
- Composite transformation
- Other transformation
- The viewing pipeline
- Viewing coordinate reference frame
- Window-to-viewport coordinate transformation
- Point clipping
- Line clipping
- Polygon clipping

Transformation

- Changing Position, shape, size, or orientation of an object is known as transformation.

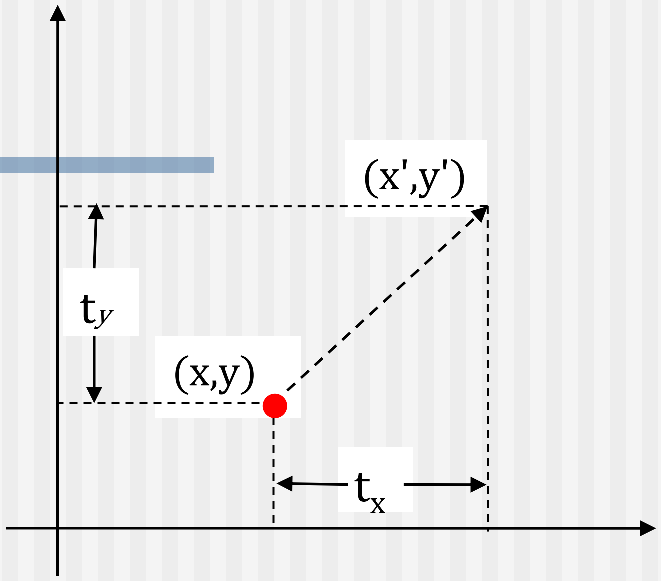


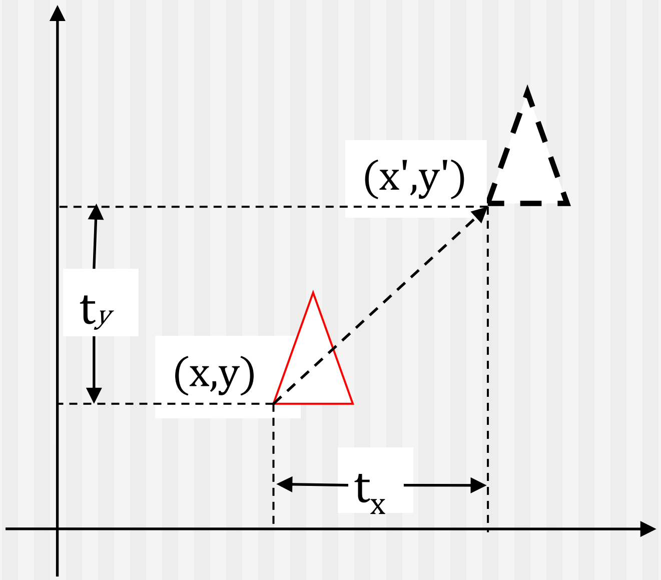
Basic Transformation

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

Translation

- Transformation that used to reposition the object along the straight line path.
- Adding translation distance $\boldsymbol{t_x}$ and $\boldsymbol{t_y}$ to the original coordinate position $(\boldsymbol{x}, \boldsymbol{y})$ respectively we move point at new position $(\boldsymbol{x'}, \boldsymbol{y'})$.
- $\boldsymbol{x' = x + t_x}$ **&** $\boldsymbol{y' = y + t_y}$
- Translation distance pair $(\boldsymbol{t_{x,} t_y})$ is called a **Translation Vector** or **Shift Vector**.
- It is rigid body transformation so we need to translate whole object.





Contd.

- We can represent it into single matrix equation in column vector as;

$$\boldsymbol{P'} = \boldsymbol{P} + \boldsymbol{T}$$

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

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

$$\boldsymbol{P'} = \boldsymbol{P} + \boldsymbol{T}$$

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

- Since column vector representation is standard mathematical notation and also many graphics package like **GKS** and **PHIGS** uses column vector we will also follow column vector representation.

Translation Example

- **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 $P' = P + T = [P] + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$

  For point (10, 10) $A' = \begin{bmatrix} 10 \\ 10 \end{bmatrix} + \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 12 \\ 11 \end{bmatrix}$

  For point (15, 15) $B' = \begin{bmatrix} 15 \\ 15 \end{bmatrix} + \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 17 \\ 16 \end{bmatrix}$

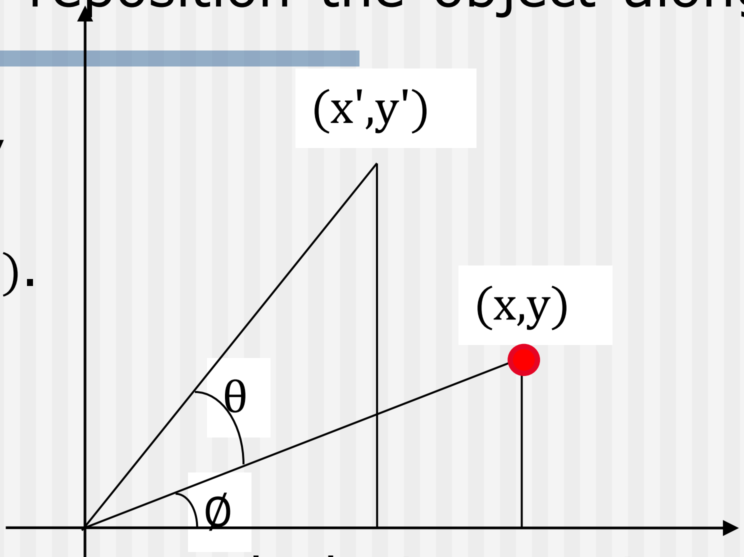
  For point (10, 10) $C' = \begin{bmatrix} 20 \\ 10 \end{bmatrix} + \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 22 \\ 11 \end{bmatrix}$

- 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 $xy - plane$.
- To generate a rotation we specify a,
  - **Rotation angle θ.**
  - **Rotation Point (Pivot Point) $(x_{r,}y_r)$.**

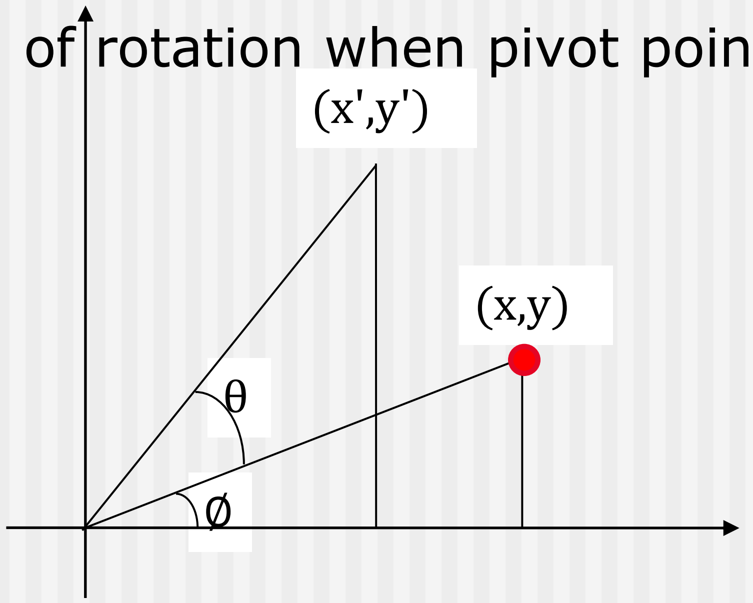


- +ve value of rotation angle defines counter clockwise rotation.
- -ve value of rotation angle defines clockwise rotation.

Rotation Equation

- We first find the equation of rotation when pivot point is at coordinate origin(0,0).
- From figure we can write.

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

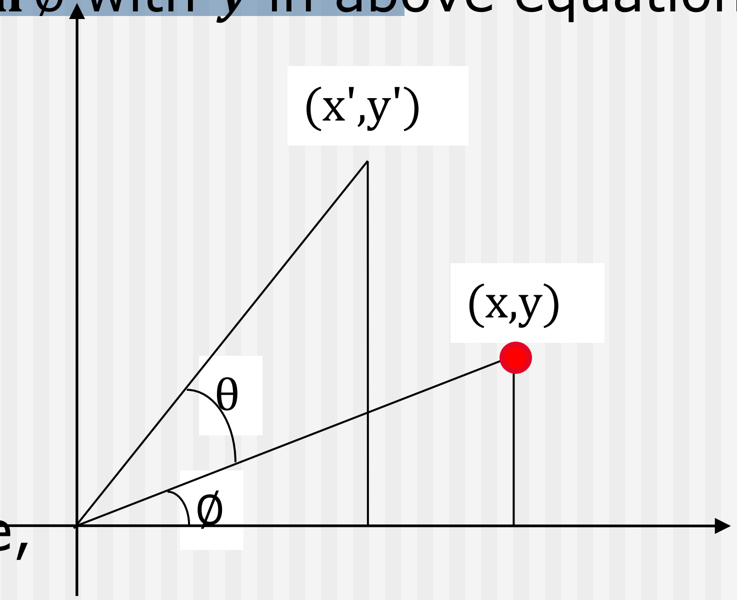


&

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

Contd.

- Now replace $\boldsymbol{r}\cos\emptyset$ with $\boldsymbol{x}$ and $\boldsymbol{r}\sin\emptyset$ with $\boldsymbol{y}$ in above equation.
- $\boldsymbol{x' = x\cos\theta - y\sin\theta}$
- $\boldsymbol{y' = x\sin\theta + y\cos\theta}$

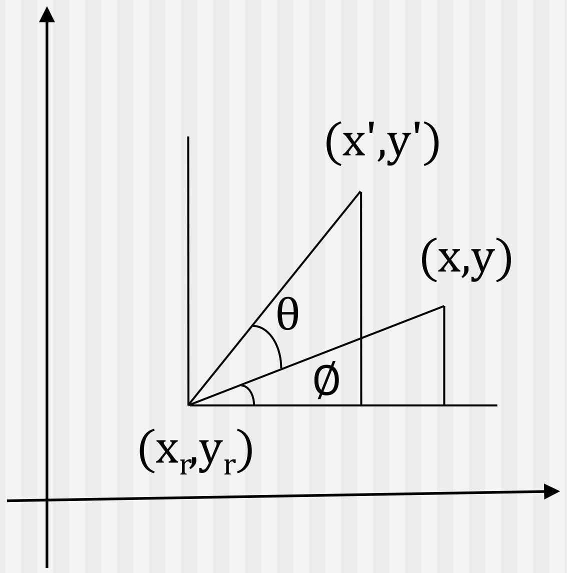


- column vector matrix equation are,
- $\boldsymbol{P' = R \cdot P}$
- $\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$

Rotation about arbitrary point

- Transformation equation for rotation of a point about pivot point $(x_{r,}y_r)$ is:
- $x' = x_r + (x - x_r)\cos\theta - (y - y_r)\sin\theta$
- $y' = y_r + (x - x_r)\sin\theta + (y - y_r)\cos\theta$
- These equations are differing from rotation about origin and its matrix representation is also different.

- Its matrix equation can be obtained by simple method.
- Rotation is also rigid body transformation so we need to rotate each point of object.



Rotation Example

- **Example**: - Locate the new position of the triangle [A (5, 4), B (8, 3), C (8, 8)] after its rotation by 90° clockwise about the origin.
- As rotation is clockwise we will take $\theta = -90^\circ$.
- $P' = R \cdot P$
- $P' = \begin{bmatrix} \cos(-90) & -\sin(-90) \\ \sin(-90) & \cos(-90) \end{bmatrix} \begin{bmatrix} 5 & 8 & 8 \\ 4 & 3 & 8 \end{bmatrix}$
- $P' = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} 5 & 8 & 8 \\ 4 & 3 & 8 \end{bmatrix}$
- $P' = \begin{bmatrix} 4 & 3 & 8 \\ -5 & -8 & -8 \end{bmatrix}$
- Final coordinates after rotation are: [A′ (4, -5), B′ (3, -8), C′ (8, -8)]

Scaling

- Transformation that used to alter the size of an object is known as scaling.
- This operation is carried out by multiplying coordinate value (x, y) with scale factors (s_x, s_y) respectively.
- Equation for scaling is given by,

  $$x' = x \cdot s_x \;\&\; y' = y \cdot s_y$$
- These equation can be represented in column vector matrix equation as:

  $$P' = S \cdot P$$

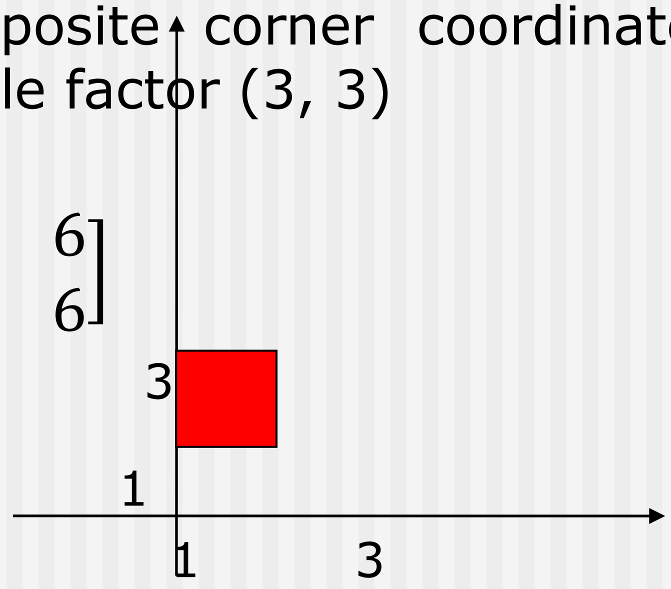
  $$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$
- Any positive value can be assigned to (s_x, s_y).

Contd.

- Normal scaling will scale as well as reposition the object.
- Scale factors less than 1 reduce the size and move object closer to origin.
- Scale factors greater than 1 enlarge the size of object and move object away from origin
- Example scale square with opposite corner coordinate point are (1, 1) and (2, 2) by scale factor (3, 3)

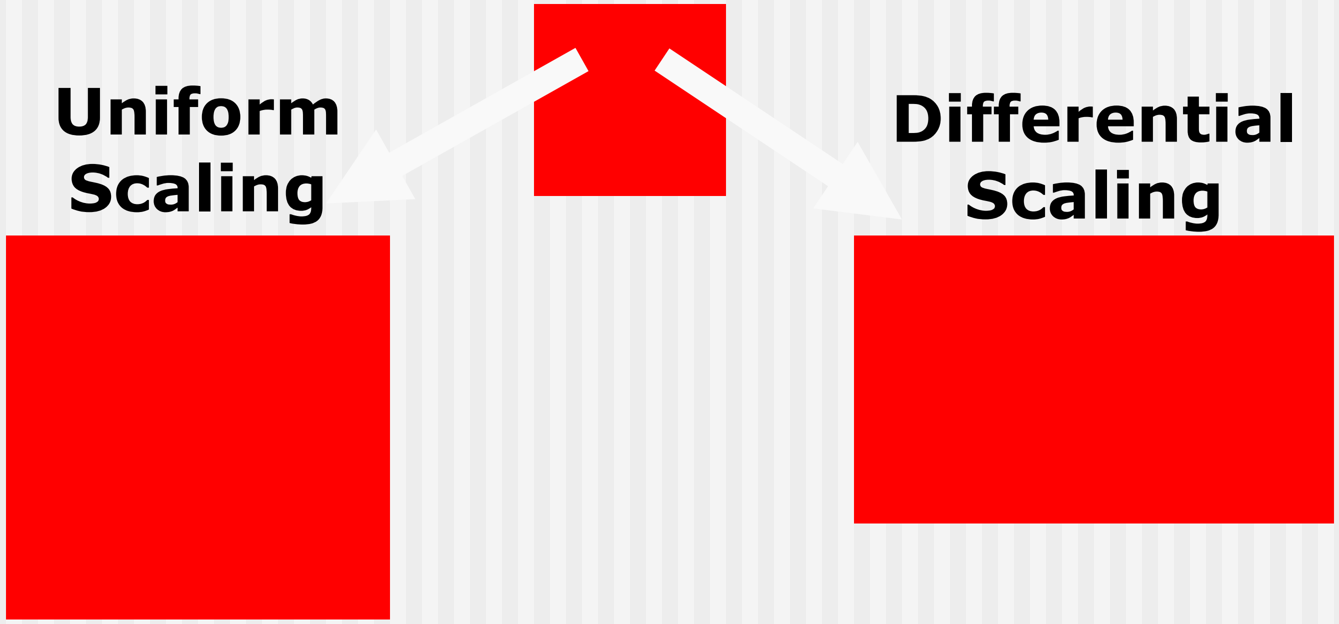
$$\boldsymbol{P'} = \boldsymbol{S} \cdot \boldsymbol{P}$$

$$\begin{bmatrix} x_1' & x_2' \\ y_1' & y_2' \end{bmatrix} = \begin{bmatrix} 3 & 0 \\ 0 & 3 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 1 & 2 \end{bmatrix} = \begin{bmatrix} 3 & 6 \\ 3 & 6 \end{bmatrix}$$



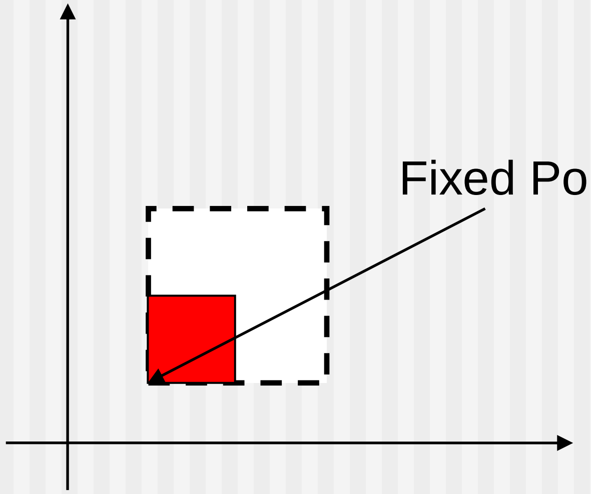
Contd.

- Same values of s_x and s_y will produce **Uniform Scaling**.
- Different values of s_x and s_y will produce **Differential (Non Uniform) Scaling**.



Fixed Point Scaling

- We can control the position of object after scaling by keeping one position fixed called **Fix point** (x_f, y_f).



Fixed Point Scaling Equation

- Equation for scaling with fixed point position as (x_f, y_f) is:
- $x' = x_f + (x - x_f)s_x$ $\quad$ $y' = y_f + (y - y_f)s_y$
- $x' = x_f + xs_x - x_f s_x$ $\quad$ $y' = y_f + ys_y - y_f s_y$
- $x' = xs_x + x_f(1 - s_x)$ $\quad$ $y' = ys_y + y_f(1 - s_y)$
- Polygons are scaled by applying scaling at coordinates and redrawing.
- Other body like circle and ellipse will scale using its defining parameters.
- For example ellipse will scale using its semi major axis, semi minor axis and center point scaling and redrawing at that position.

Scaling Example

- **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.
- 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)].
- $P' = S \cdot P$
- $P' = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} 2 & 6 & 6 & 2 \\ 2 & 2 & 6 & 6 \end{bmatrix} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} 2 & 6 & 6 & 2 \\ 2 & 2 & 6 & 6 \end{bmatrix}$
- $P' = \begin{bmatrix} 1 & 3 & 3 & 1 \\ 1 & 1 & 3 & 3 \end{bmatrix}$
- Final coordinate after scaling are:

  [A′ (1, 1), B′ (3, 1), C′ (3, 3), D′ (1, 3)]

Matrix Representation and Homogeneous Coordinates

- 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.

Contd.

- We have matrix representation of basic transformation and we can express it in the general matrix form as:

$$\boldsymbol{P'} = \boldsymbol{M_1} \cdot \boldsymbol{P} + \boldsymbol{M_2}$$

Where

$\boldsymbol{P}$ and $\boldsymbol{P'}$ are initial and final point position,

$\boldsymbol{M_1}$ contains rotation and scaling terms and

$\boldsymbol{M_2}$ contains translational terms associated with pivot point, fixed point and reposition.

Contd.

- For efficient utilization we must calculate all sequence of transformation in one step.
- For that reason we reformulate above equation to eliminate the matrix addition associated with translation terms in matrix $\boldsymbol{M_2}$.
- We can combine that thing by expanding 2X2 matrix representation into 3X3 matrices.
- It will allows us to convert all transformation into matrix multiplication.
- We need to represent vertex position $(\boldsymbol{x}, \boldsymbol{y})$ with homogeneous coordinate triple $(\boldsymbol{x_h}, \boldsymbol{y_h}, \boldsymbol{h})$.
- Where $\boldsymbol{x} = \frac{\boldsymbol{x_h}}{\boldsymbol{h}}$, $\boldsymbol{y} = \frac{\boldsymbol{y_h}}{\boldsymbol{h}}$ thus we can also write triple as

  $(\boldsymbol{h} \cdot \boldsymbol{x}, \boldsymbol{h} \cdot \boldsymbol{y}, \boldsymbol{h})$.

Contd.

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

Homogeneous Matrix for Translation

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

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

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

- Inverse of translation matrix is obtain by putting $\boldsymbol{-t_x}$ & $\boldsymbol{-t_y}$ instead of $\boldsymbol{t_x}$ & $\boldsymbol{t_y}$.

Homogeneous Matrix for Rotation

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

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

- Inverse of rotation matrix is obtained by replacing $\boldsymbol{\theta}$ by $-\boldsymbol{\theta}$.

Homogeneous Matrix for Scaling

$$P' = S_{(s_x, s_y)} \cdot P$$

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

- Inverse of scaling matrix is obtained by replacing $s_x \,\&\, s_y$ **b**y $\frac{1}{s_x}$ & $\frac{1}{s_y}$ respectively.

Composite Transformation

- In practice we need to apply more than one transformations to get desired result.
- It is time consuming to apply transformation one by one on each point of object.
- So first we multiply all matrix of required transformations which is known as **composite transformation matrix.**
- Than we use composite transformation matrix to transform object.
- For column matrix representation, we form composite transformations by multiplying matrices from right to left.

Multiple Translations

- Two successive translations are performed as:
- $P' = T(t_{x2}, t_{y2}) \cdot \{T(t_{x1}, t_{y1}) \cdot P\}$
- $P' = \{T(t_{x2}, t_{y2}) \cdot T(t_{x1}, t_{y1})\} \cdot P$
- $P' = \begin{bmatrix} 1 & 0 & t_{x2} \\ 0 & 1 & t_{y2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & t_{x1} \\ 0 & 1 & t_{y1} \\ 0 & 0 & 1 \end{bmatrix} \cdot P$

Contd.

- $P' = \begin{bmatrix} 1 & 0 & t_{x2} \\ 0 & 1 & t_{y2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & t_{x1} \\ 0 & 1 & t_{y1} \\ 0 & 0 & 1 \end{bmatrix} \cdot P$
- $P' = \begin{bmatrix} 1 & 0 & t_{x1} + t_{x2} \\ 0 & 1 & t_{y1} + t_{y2} \\ 0 & 0 & 1 \end{bmatrix} \cdot P$
- $P' = T\left(t_{x1} + t_{x2}, t_{y1} + t_{y2}\right) \cdot P$
- Here P' and P are column vector of final and initial point coordinate respectively.
- This concept can be extended for any number of successive translations.

Multiple Translations Example

- **Example:** Obtain the final coordinates after two translations on point $p(2,3)$ with translation vector $(4,3)$ and $(-1,2)$ respectively.
- $P' = T\left(t_{x1} + t_{x2}, t_{y1} + t_{y2}\right) \cdot P$
- $P' = \begin{bmatrix} 1 & 0 & t_{x1} + t_{x2} \\ 0 & 1 & t_{y1} + t_{y2} \\ 0 & 0 & 1 \end{bmatrix} \cdot P = \begin{bmatrix} 1 & 0 & 4 + (-1) \\ 0 & 1 & 3 + 2 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}$
- $P' = \begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 5 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 5 \\ 8 \\ 1 \end{bmatrix}$
- Final Coordinates after translations are $p'(5, 8)$.

Multiple Rotations

- Two successive Rotations are performed as:
- $P' = R(\theta_2) \cdot \{R(\theta_1) \cdot P\}$
- $P' = \{R(\theta_2) \cdot R(\theta_1)\} \cdot P$
- $P' = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 \\ \sin\theta_2 & \cos\theta_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot P$
- $P' = \begin{bmatrix} \cos\theta_2\cos\theta_1 - \sin\theta_2\sin\theta_1 & -\sin\theta_1\cos\theta_2 - \sin\theta_2\cos\theta_1 & 0 \\ \sin\theta_1\cos\theta_2 + \sin\theta_2\cos\theta_1 & \cos\theta_2\cos\theta_1 - \sin\theta_2\sin\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot P$
- $P' = \begin{bmatrix} \cos(\theta_1+\theta_2) & -\sin(\theta_1+\theta_2) & 0 \\ \sin(\theta_1+\theta_2) & \cos(\theta_1+\theta_2) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot P$

Contd.

- $P' = \begin{bmatrix} \cos(\theta_1+\theta_2) & -\sin(\theta_1+\theta_2) & 0 \\ \sin(\theta_1+\theta_2) & \cos(\theta_1+\theta_2) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot P$
- $P' = R(\theta_1+\theta_2) \cdot P$
- Here P' and P are column vector of final and initial point coordinate respectively.
- This concept can be extended for any number of successive rotations.

Multiple Rotations Example

- **Example:** Obtain the final coordinates after two rotations on point $p(6{,}9)$ with rotation angles are 30^o and 60^o respectively.
- $P' = R(\theta_1{+}\theta_2) \cdot P$
- $P' = \begin{bmatrix} cos(\theta_1{+}\theta_2) & -sin(\theta_1{+}\theta_2) & 0 \\ sin(\theta_1{+}\theta_2) & cos(\theta_1{+}\theta_2) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot P$
- $P' = \begin{bmatrix} cos(30+60) & -sin(30+60) & 0 \\ sin(30+60) & cos(30+60) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot P$
- $P' = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 6 \\ 9 \\ 1 \end{bmatrix} = \begin{bmatrix} -9 \\ 6 \\ 1 \end{bmatrix}$
- Final Coordinates after rotations are $p'(-9, 6)$.

Multiple Scaling

- Two successive scaling are performed as:
- $P' = S(s_{x2}, s_{y2}) \cdot \{S(s_{x1}, s_{y1}) \cdot P\}$
- $P' = \{S(s_{x2}, s_{y2}) \cdot S(s_{x1}, s_{y1})\} \cdot P$
- $P' = \begin{bmatrix} s_{x2} & 0 & 0 \\ 0 & s_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_{x1} & 0 & 0 \\ 0 & s_{y1} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot P$
- $P' = \begin{bmatrix} s_{x1} \cdot s_{x2} & 0 & 0 \\ 0 & s_{y1} \cdot s_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot P$

Contd.

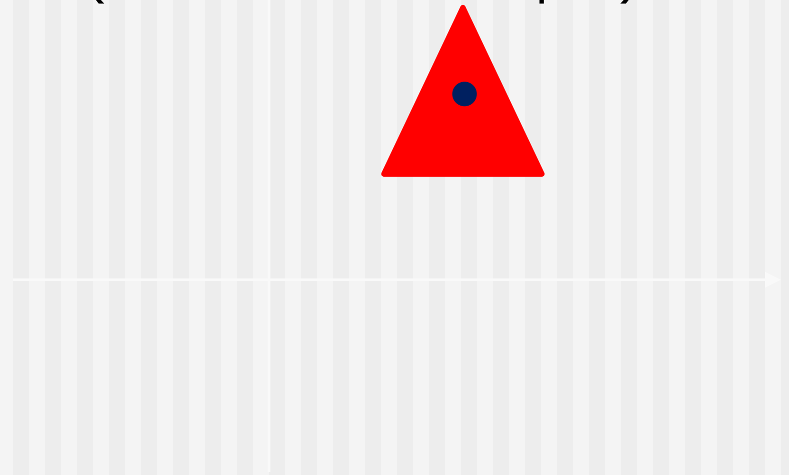
- $\boldsymbol{P'} = \begin{bmatrix} \boldsymbol{s_{x1} \cdot s_{x2}} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{s_{y1} \cdot s_{y2}} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} \end{bmatrix} \cdot \boldsymbol{P}$
- $\boldsymbol{P' = S\left(s_{x1} \cdot s_{x2}, s_{y1} \cdot s_{y2}\right) \cdot P}$
- Here $\boldsymbol{P'}$ and $\boldsymbol{P}$ are column vector of final and initial point coordinate respectively.
- This concept can be extended for any number of successive scaling.

Multiple Scaling Example

- **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.
- $P' = S(s_{x1} \cdot s_{x2}, s_{y1} \cdot s_{y2}) \cdot P$
- $P' = \begin{bmatrix} s_{x1} \cdot s_{x2} & 0 & 0 \\ 0 & s_{y1} \cdot s_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot P = \begin{bmatrix} 2 \cdot 3 & 0 & 0 \\ 0 & 2 \cdot 3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot P$
- $P' = \begin{bmatrix} 6 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 & 8 \\ 2 & 8 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 12 & 48 \\ 12 & 48 \\ 1 & 1 \end{bmatrix}$
- Final Coordinates after rotations are $p'(12,12)$ and $q'(48,48)$.

General Pivot-Point Rotation

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



General Pivot-Point Rotation Equation

- Let’s find matrix equation for this
- $\boldsymbol{P' = T(x_r, y_r) \cdot [R(\theta) \cdot \{T(-x_r, -y_r) \cdot P\}]}$
- $\boldsymbol{P' = \{T(x_r, y_r) \cdot R(\theta) \cdot T(-x_r, -y_r)\} \cdot P}$
- $\boldsymbol{P' = \begin{bmatrix} 1 & 0 & x_r \\ 0 & 1 & y_r \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_r \\ 0 & 1 & -y_r \\ 0 & 0 & 1 \end{bmatrix} \cdot P}$
- $\boldsymbol{P' = \begin{bmatrix} \cos\theta & -\sin\theta & x_r(1-\cos\theta) + y_r \sin\theta \\ \sin\theta & \cos\theta & y_r(1-\cos\theta) - x_r \sin\theta \\ 0 & 0 & 1 \end{bmatrix} \cdot P}$

Contd.

- $$P' = \begin{bmatrix} \cos\theta & -\sin\theta & x_r(1-\cos\theta) + y_r\sin\theta \\ \sin\theta & \cos\theta & y_r(1-\cos\theta) - x_r\sin\theta \\ 0 & 0 & 1 \end{bmatrix} \cdot P$$
- $P' = R(x_r, y_r, \theta) \cdot P$
- Here P' and P are column vector of final and initial point coordinate respectively and (x_r, y_r) are the coordinates of pivot-point.

General Pivot-Point Rotation Example

- **Example**: - Locate the new position of the triangle ABC [A (5, 4), B (8, 3), C (8, 8)] after its rotation by 90° clockwise about the centroid.
- Pivot point is centroid of the triangle so:
- $x_r = \frac{5+8+8}{3} = 7\,,\ y_r = \frac{4+3+8}{3} = 5$
- As rotation is clockwise we will take $\theta = -90^\circ$.
- $P' = R_{(x_r,\, y_r, \theta)} \cdot P$
- $P' = \begin{bmatrix} \cos\theta & -\sin\theta & x_r(1-\cos\theta) + y_r \sin\theta \\ \sin\theta & \cos\theta & y_r(1-\cos\theta) - x_r \sin\theta \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 5 & 8 & 8 \\ 4 & 3 & 8 \\ 1 & 1 & 1 \end{bmatrix}$

Contd.

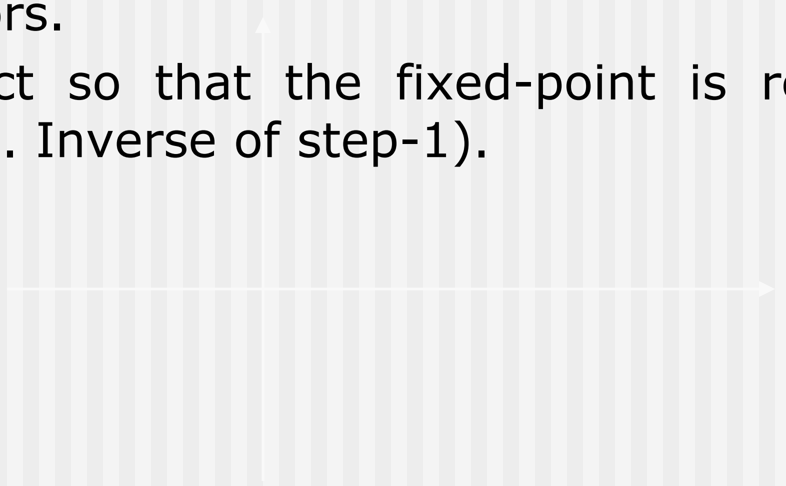
- $$P' = \begin{bmatrix} \cos(-90) & -\sin(-90) & 7(1-\cos(-90)) + 5\sin(-90) \\ \sin(-90) & \cos(-90) & 5(1-\cos(-90)) - 7\sin(-90) \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 5 & 8 & 8 \\ 4 & 3 & 8 \\ 1 & 1 & 1 \end{bmatrix}$$
- $$P' = \begin{bmatrix} 0 & 1 & 7(1-0) - 5(1) \\ -1 & 0 & 5(1-0) + 7(1) \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 5 & 8 & 8 \\ 4 & 3 & 8 \\ 1 & 1 & 1 \end{bmatrix}$$
- $$P' = \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 12 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 5 & 8 & 8 \\ 4 & 3 & 8 \\ 1 & 1 & 1 \end{bmatrix}$$

Contd.

- $P' = \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 12 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 5 & 8 & 8 \\ 4 & 3 & 8 \\ 1 & 1 & 1 \end{bmatrix}$
- $P' = \begin{bmatrix} 6 & 5 & 10 \\ 7 & 4 & 4 \\ 1 & 1 & 1 \end{bmatrix}$
- Final coordinates after rotation are [A′ (6, 7), B′ (5, 4), C′ (10, 4)].

General Fixed-Point Scaling

- For scaling object with position of one point called fixed point will remains same, we need to apply following sequence of transformation.
  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).



General Fixed-Point Scaling Equation

- Let’s find matrix equation for this
- $P' = T(x_f, y_f) \cdot [S(s_x, s_y) \cdot \{T(-x_f, -y_f) \cdot P\}]$
- $P' = \{T(x_f, y_f) \cdot S(s_x, s_y) \cdot T(-x_f, -y_f)\} \cdot P$
- $P' = \begin{bmatrix} 1 & 0 & x_f \\ 0 & 1 & y_f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_f \\ 0 & 1 & -y_f \\ 0 & 0 & 1 \end{bmatrix} \cdot P$
- $P' = \begin{bmatrix} s_x & 0 & x_f(1 - s_x) \\ 0 & s_y & y_f(1 - s_y) \\ 0 & 0 & 1 \end{bmatrix} \cdot P$

Contd.

- $\boldsymbol{P'} = \begin{bmatrix} \boldsymbol{s_x} & \mathbf{0} & \boldsymbol{x_f(1 - s_x)} \\ \mathbf{0} & \boldsymbol{s_y} & \boldsymbol{y_f(1 - s_y)} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix} \cdot \boldsymbol{P}$
- $\boldsymbol{P'} = \boldsymbol{S(x_f, y_f, s_x, s_y)} \cdot \boldsymbol{P}$
- Here $\boldsymbol{P'}$ and $\boldsymbol{P}$ are column vector of final and initial point coordinate respectively and $(\boldsymbol{x_f, y_f})$ are the coordinates of fixed-point.

General Fixed-Point Scaling Example

- **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.
- Fixed point is center of square so:
- $x_f = \frac{6+2}{2}$, $y_f = \frac{6+2}{2}$
- 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)].
- $P' = S(x_f, y_f, s_x, s_y) \cdot P$

Contd.

- $P' = S(x_f, y_f, s_x, s_y) \cdot P$
- $P' = \begin{bmatrix} s_x & 0 & x_f(1 - s_x) \\ 0 & s_y & y_f(1 - s_y) \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 6 & 6 & 2 \\ 2 & 2 & 6 & 6 \\ 1 & 1 & 1 & 1 \end{bmatrix}$
- $P' = \begin{bmatrix} 0.5 & 0 & 4(1 - 0.5) \\ 0 & 0.5 & 4(1 - 0.5) \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 6 & 6 & 2 \\ 2 & 2 & 6 & 6 \\ 1 & 1 & 1 & 1 \end{bmatrix}$
- $P' = \begin{bmatrix} 0.5 & 0 & 2 \\ 0 & 0.5 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 6 & 6 & 2 \\ 2 & 2 & 6 & 6 \\ 1 & 1 & 1 & 1 \end{bmatrix}$

Contd.

- $P' = \begin{bmatrix} 0.5 & 0 & 2 \\ 0 & 0.5 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 6 & 6 & 2 \\ 2 & 2 & 6 & 6 \\ 1 & 1 & 1 & 1 \end{bmatrix}$
- $P' = \begin{bmatrix} 3 & 5 & 5 & 3 \\ 3 & 3 & 5 & 5 \\ 1 & 1 & 1 & 1 \end{bmatrix}$
- Final coordinate after scaling are:

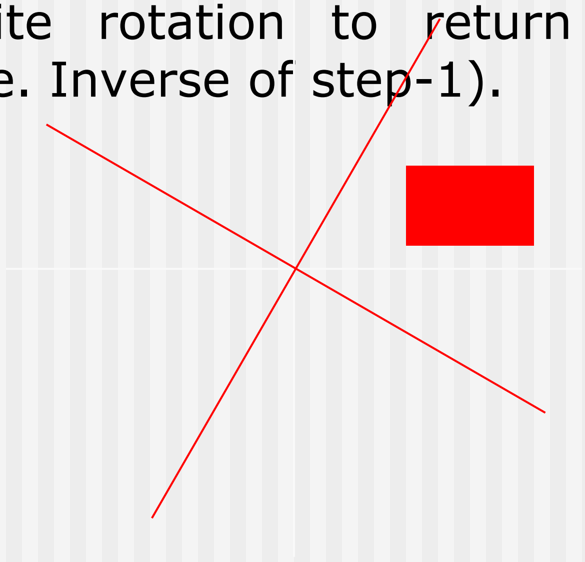
  [A′ (3, 3), B′ (5, 3), C′ (5, 5), D′ (3, 5)]

General Scaling Directions

- Parameter s_x and s_y scale the object along x and y directions.
- We can scale an object in other directions also.
- By rotating the object to align the desired scaling directions with the coordinate axes before applying the scaling transformation.

Contd.

- Suppose we apply scaling factor s_1 and s_2 in direction shown in figure than we will apply following transformations.
  1. Perform a rotation so that the direction for s_1 and s_2 coincide with x and y axes.
  2. Scale the object with specified scale factors.
  3. Perform opposite rotation to return points to their original orientations. (i.e. Inverse of step-1).



General Scaling Directions Equation

- Let’s find matrix equation for this
- $P' = R^{-1}(\theta) \cdot [S(s_1, s_2) \cdot \{R(\theta) \cdot P\}]$
- $P' = \{R^{-1}(\theta) \cdot S(s_1, s_2) \cdot R(\theta)\} \cdot P$
- $P' = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot P$
- $P' = \begin{bmatrix} s_1\cos^2\theta + s_2\sin^2\theta & (s_2 - s_1)\cos\theta\sin\theta & 0 \\ (s_2 - s_1)\cos\theta\sin\theta & s_1\sin^2\theta + s_2\cos^2\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot P$
- Here P' and P are column vector of final and initial point coordinate respectively and θ is the angle between actual scaling direction and our standard coordinate axes.

Other Transformation

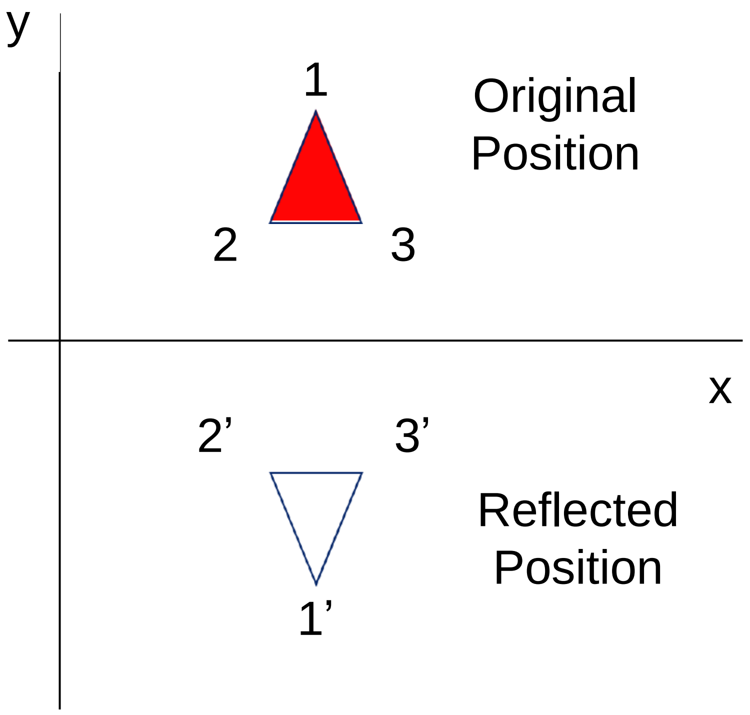
- Some package provides few additional transformations which are useful in certain applications.
- Two such transformations are:
  1. Reflection
  2. Shear.

1. Reflection

- A reflection is a transformation that produces a mirror image of an object.
- The mirror image for a two –dimensional reflection is generated relative to an **axis of reflection** by rotating the object 180^o about the reflection axis.
- Reflection gives image based on position of axis of reflection. Transformation matrix for few positions are discussed here.

Reflection About X-Axis

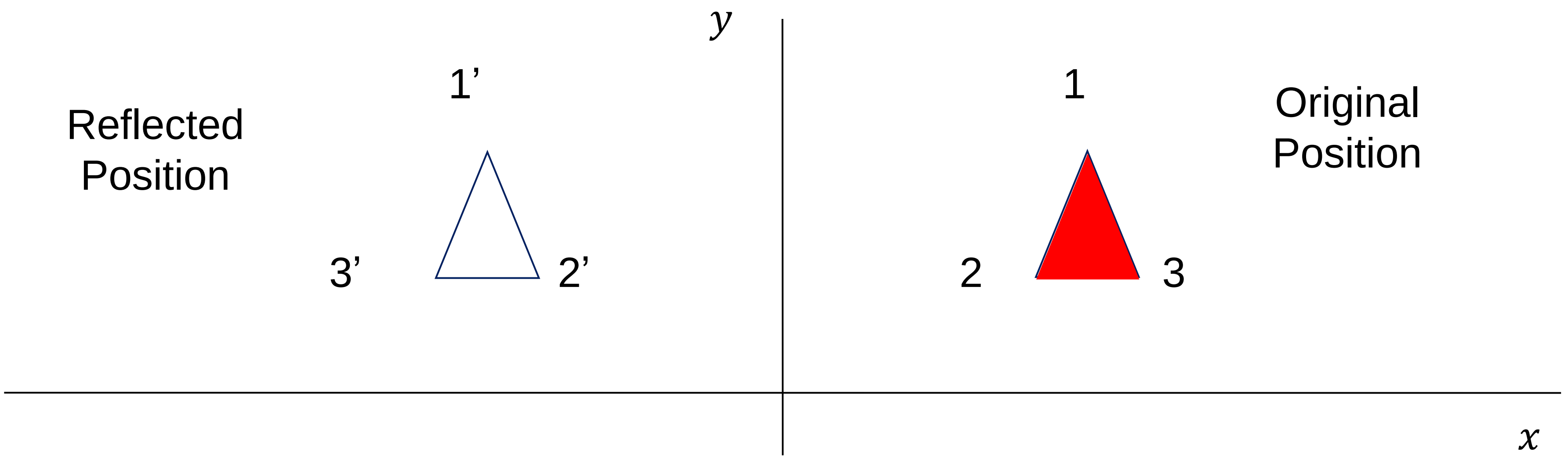
- For reflection about the line
- $\boldsymbol{y = 0\ , the\ x\ axis}$
- Transformation matrix:
- $Ref_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$



- This transformation keeps x values are same, but flips (Change the sign) y values of coordinate positions.

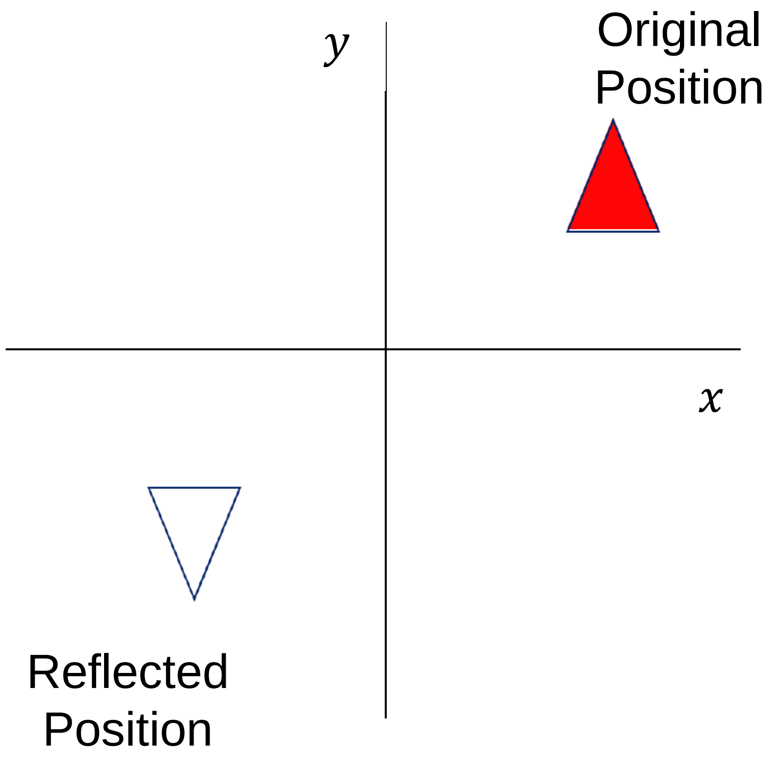
Reflection About Y-Axis

- For reflection about the line $\boldsymbol{x = 0\ , the\ y\ axis}$
- Transformation matrix: $Ref_y = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
- This transformation keeps y values are same, but flips (Change the sign) x values of coordinate positions.



Reflection About Origin

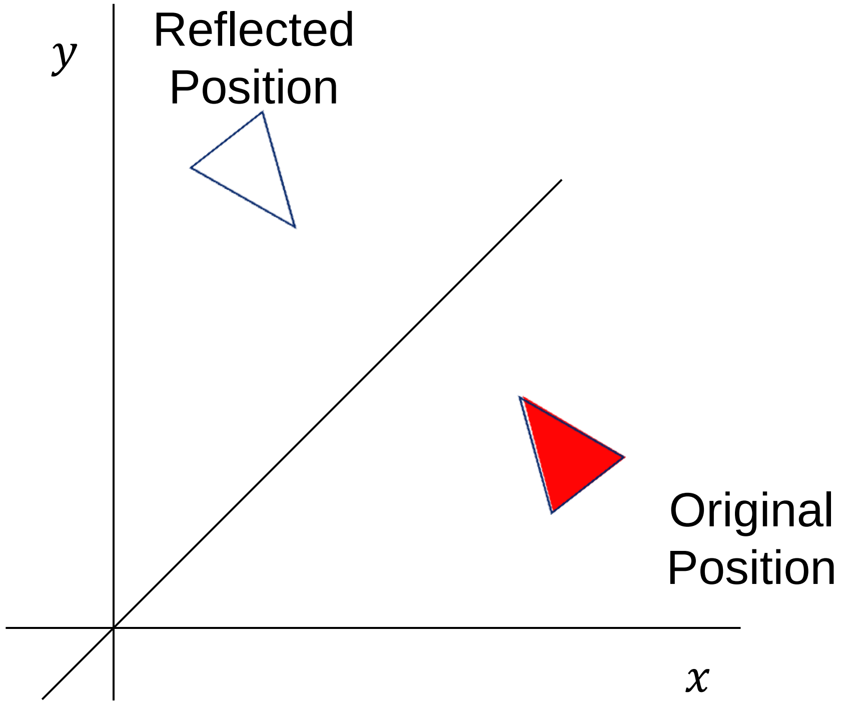
- For reflection about the $\boldsymbol{Origin}$.
- Transformation matrix:
- $Ref_{origin} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$



- This transformation flips (Change the sign) x and y both values of coordinate positions.

Reflection About X = Y Line

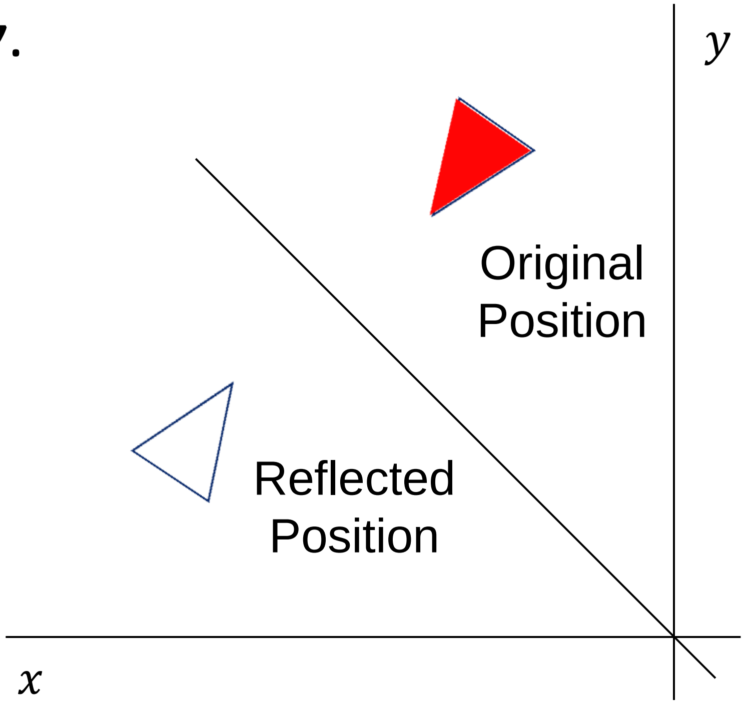
- For reflection about the line $\boldsymbol{x = y}$.
- Transformation matrix:
- $Ref_{x=y\ line} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$



- This transformation interchange x and y values of coordinate positions.

Reflection About X = - Y Line

- For reflection about the line $\boldsymbol{x} = -\boldsymbol{y}$.
- Transformation matrix:
- $Ref_{x=-y\ line} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$



- This transformation interchange x and y values of coordinate positions.

Reflection Example

- **Example**: - Find the coordinates after reflection of the triangle

  [A (10, 10), B (15, 15), C (20, 10)] about x axis.

- $$P' = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 10 & 15 & 20 \\ 10 & 15 & 10 \\ 1 & 1 & 1 \end{bmatrix}$$

- $$P' = \begin{bmatrix} 10 & 15 & 20 \\ -10 & -15 & -10 \\ 1 & 1 & 1 \end{bmatrix}$$

- Final coordinate after reflection are:

  [A′ (10, -10), B′ (15, -15), C′ (20, -10)]

2. Shear

- 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**.
- Two common shearing transformations are those that shift coordinate x values and those that shift y values.

Shear in $\mathbf{x - Direction}$

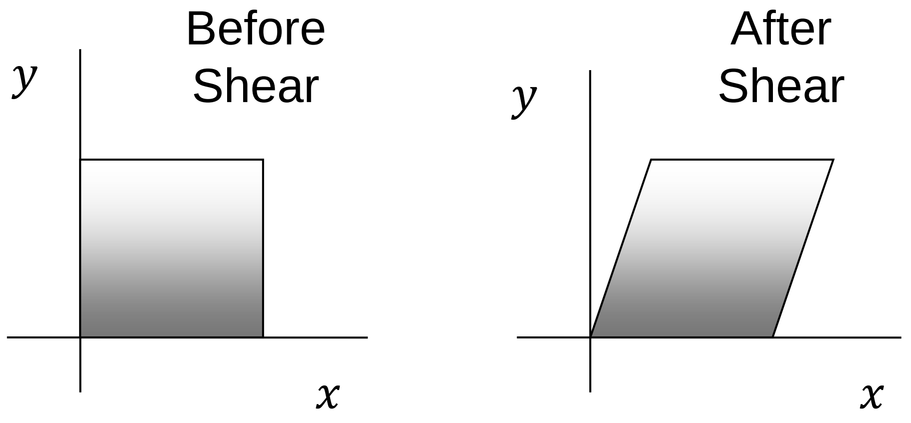
- Shear relative to $x - axis$ that is $y = 0$ line can be produced by following equation:

$$\boldsymbol{x' = x + sh_x \cdot y}\,, \qquad \boldsymbol{y' = y}$$

- Transformation matrix for that is:

$$\boldsymbol{Shear_{x-direction}} = \begin{bmatrix} \mathbf{1} & \boldsymbol{sh_x} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}$$

- Here $\boldsymbol{sh_x}$ is shear parameter.
- We can assign any real value to $\boldsymbol{sh_x}$.



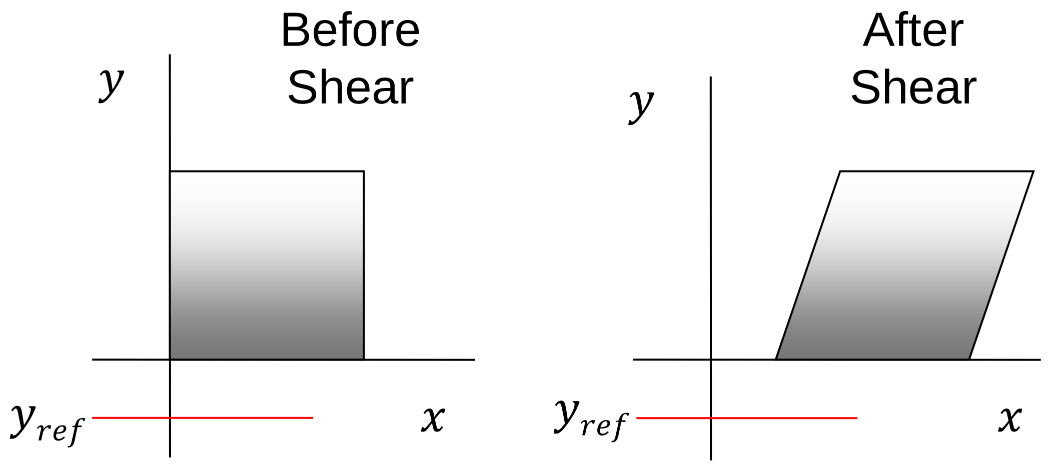
Contd.

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

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

- Transformation matrix for that is:

$$\boldsymbol{Shear_{x-direction}} = \begin{bmatrix} \mathbf{1} & \boldsymbol{sh_x} & \boldsymbol{-sh_x \cdot y_{ref}} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}$$



Shear in $\mathbf{x-Direction\ Example}$

- **Example**: - Shear the unit square in x direction with shear parameter ½ relative to line $y = -1$.
- Here $y_{ref} = -1$ and $sh_x = 0.5$
- Coordinates of unit square are: [A (0, 0), B (1, 0), C (1, 1), D (0, 1)].
- $P' = \begin{bmatrix} 1 & sh_x & -sh_x \cdot y_{ref} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$
- $P' = \begin{bmatrix} 1 & 0.5 & -0.5 \cdot (-1) \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$

Contd.

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

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

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

- Final coordinate after shear are:

  [A$'$ (0.5, 0), B$'$ (1.5, 0), C$'$ (2, 1), D$'$ (1, 1)]

Shear in $\mathbf{y - Direction}$

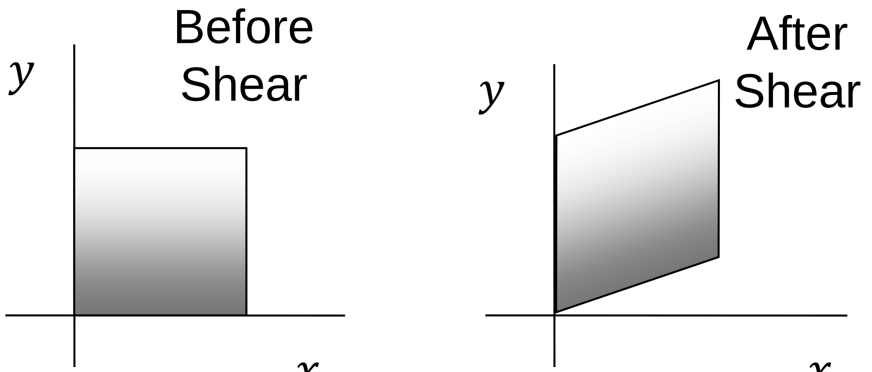
- Shear relative to $y - axis$ that is $x = 0$ line can be produced by following equation:

$$\boldsymbol{x' = x\ ,\qquad y' = y + sh_y \cdot x}$$

- Transformation matrix for that is:

$$\boldsymbol{Shear_{y-direction}} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \boldsymbol{sh_y} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}$$

- Here $\boldsymbol{sh_y}$ is shear parameter.
- We can assign any real value to $\boldsymbol{sh_y}$.



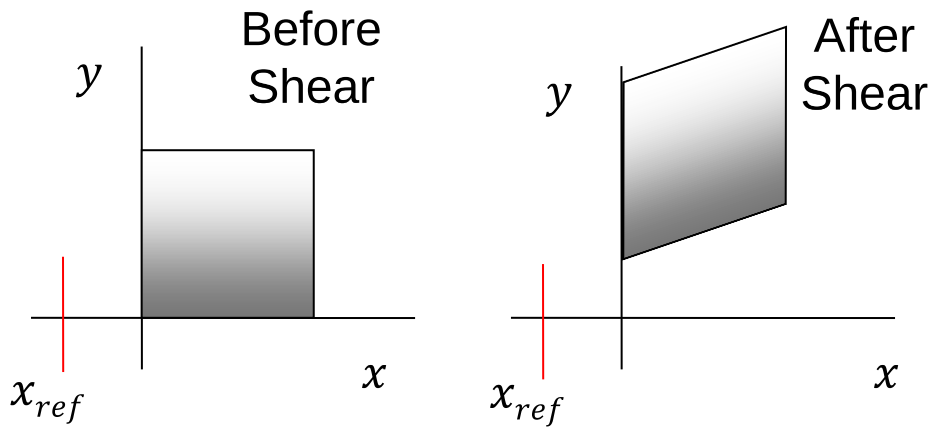
Contd.

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

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

- Transformation matrix for that is:

$$\boldsymbol{Shear_{y-direction} = \begin{bmatrix} 1 & 0 & 0 \\ sh_y & 1 & -sh_y \cdot x_{ref} \\ 0 & 0 & 1 \end{bmatrix}}$$



Shear in $\mathbf{y-Direction\ Example}$

- **Example**: - Shear the unit square in y direction with shear parameter ½ relative to line $x = -1$.
- Here $x_{ref} = -1$ and $sh_y = 0.5$
- Coordinates of unit square are [A (0, 0), B (1, 0), C (1, 1), D (0, 1)].
- $P' = \begin{bmatrix} 1 & 0 & 0 \\ sh_y & 1 & -sh_y \cdot x_{ref} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$
- $P' = \begin{bmatrix} 1 & 0 & 0 \\ 0.5 & 1 & -0.5 \cdot (-1) \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$

Contd.

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

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

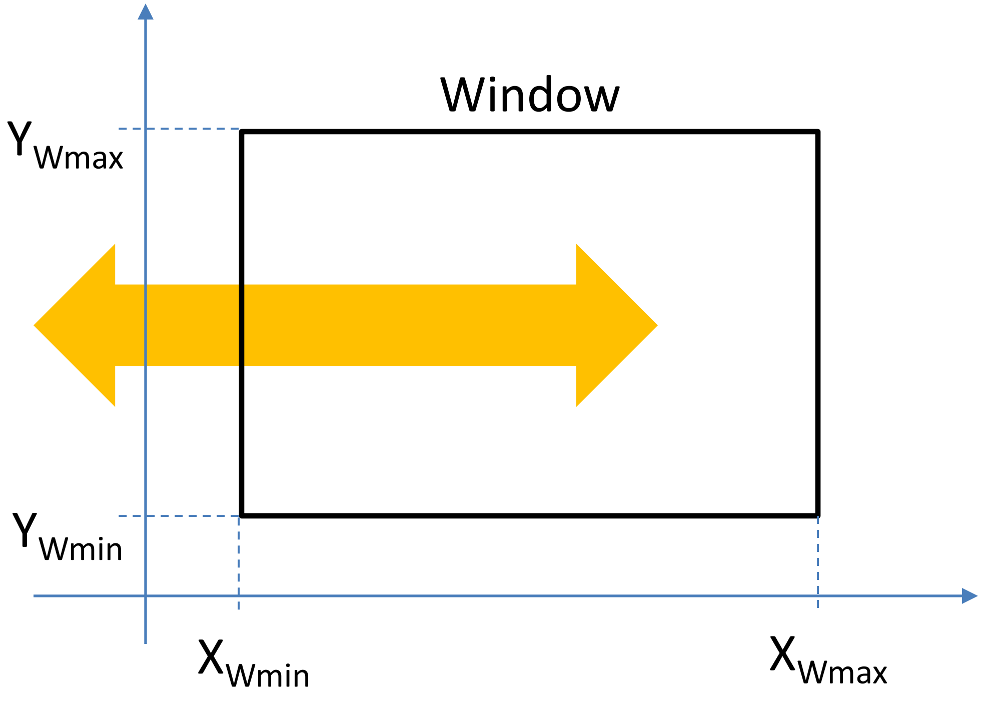
- $P' = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0.5 & 1 & 2 & 1.5 \\ 1 & 1 & 1 & 1 \end{bmatrix}$

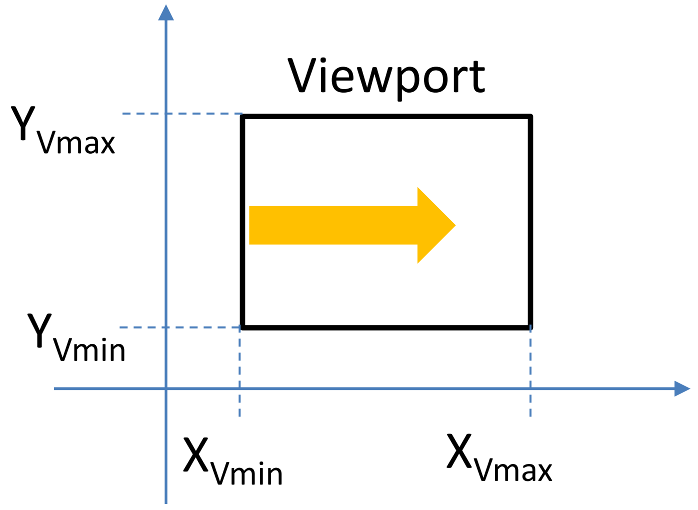
- Final coordinate after shear are:

  [A′ (0, 0.5), B′ (1, 1), C′ (1, 2), D′ (0, 1.5)]

Window and Viewport

- **Window:** Area selected in world-coordinate for display is called window. It defines what is to be viewed.
- **Viewport:** Area on a display device in which window image is display (mapped) is called viewport. It defines where to display.

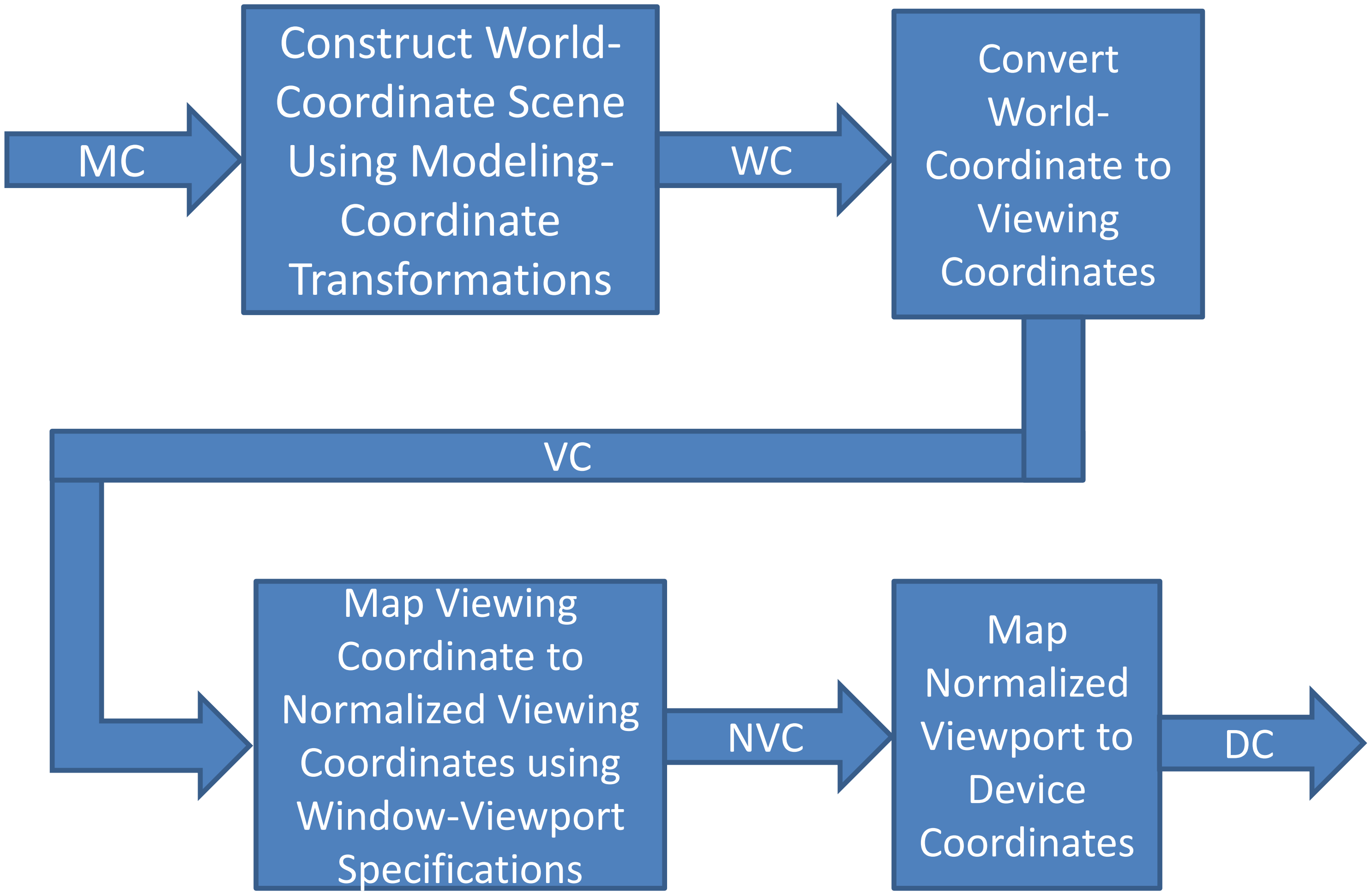




The Viewing Pipeline

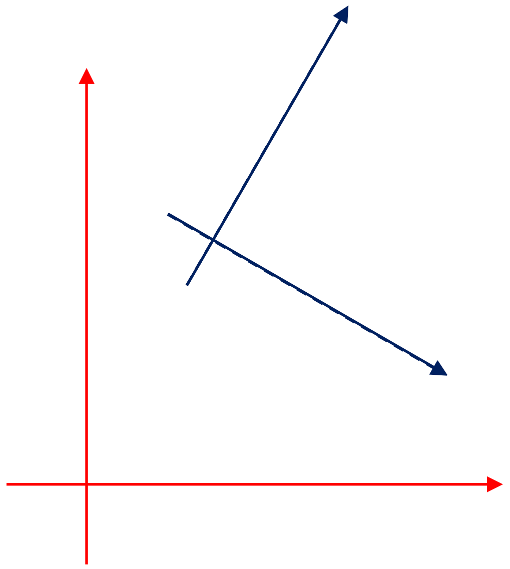
- In many case window and viewport are rectangle, also other shape may be used as window and viewport.
- In general finding device coordinates of viewport from world coordinates of window is called as **viewing transformation.**
- Sometimes we consider this viewing transformation as window-to-viewport transformation but in general it involves more steps.
- Let’s see steps involved in viewing pipeline.

Contd.


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

Viewing Coordinate Reference Frame

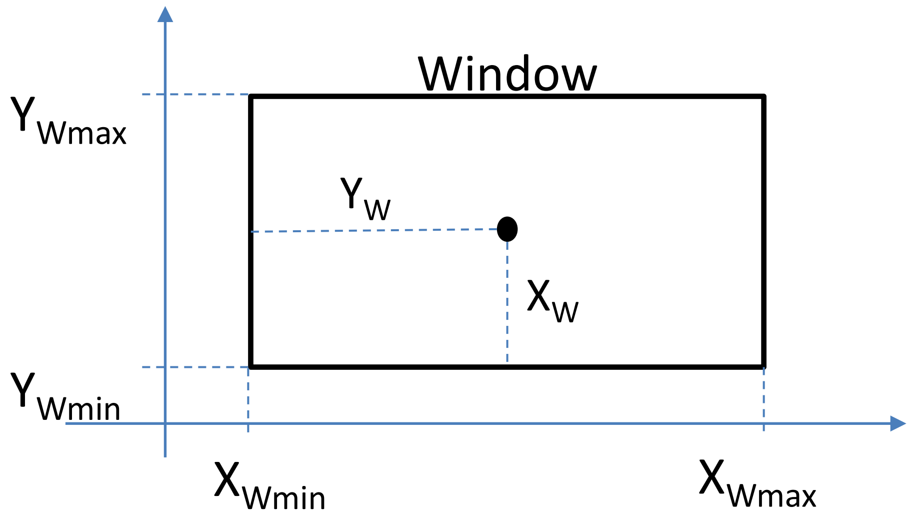
- We can obtain reference frame in any direction and at any position.
- For handling such condition
  - first of all we translate reference frame origin to standard reference frame origin.
  - Then we rotate it to align it to standard axis.

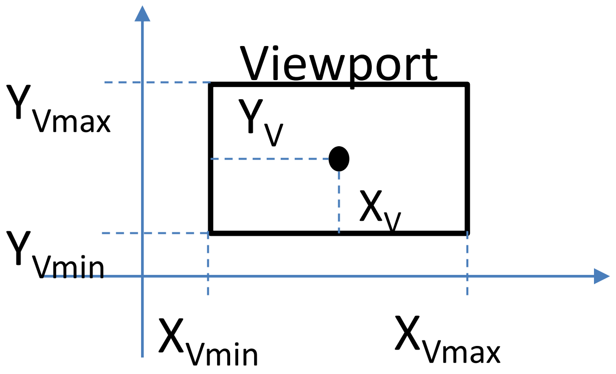


- In this way we can adjust window in any reference frame.
- It is illustrate by following transformation matrix: $M_{wc,vc} = RT$
- Where T is translation matrix and R is rotation matrix.

Window-To-Viewport Coordinate Transformation

- Mapping of window coordinate to viewport is called **window to viewport transformation.**
- We do this using transformation that maintains relative position of window coordinate into viewport.
- That means center coordinates in window must be remains at center position in viewport.





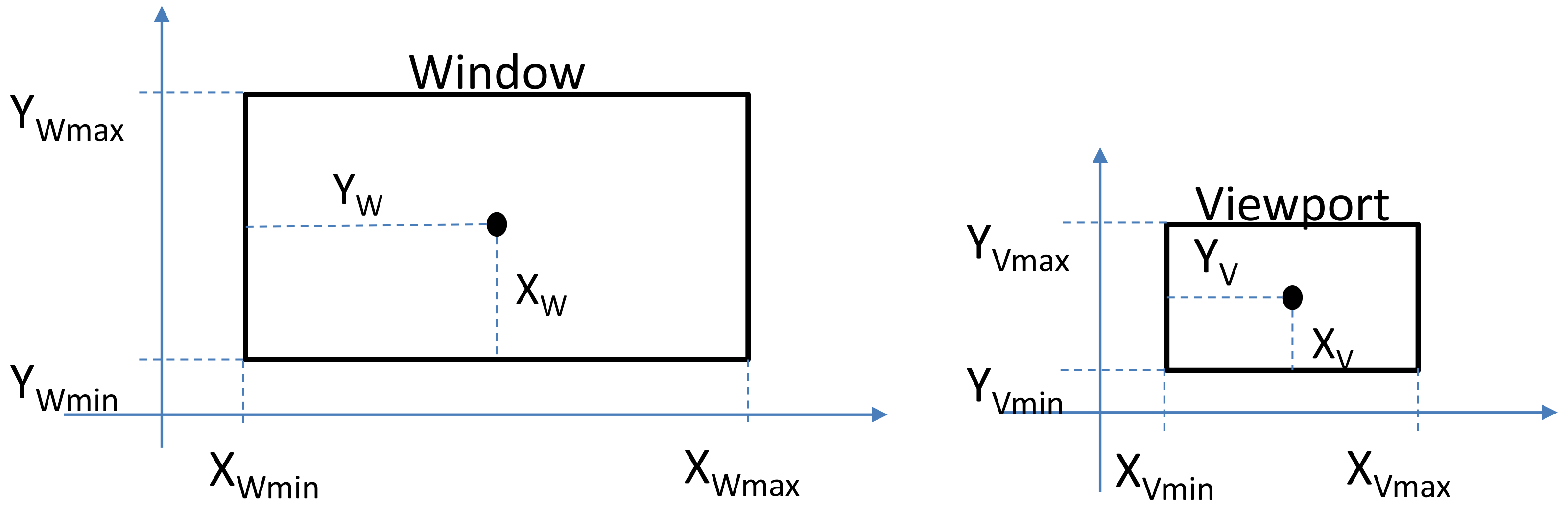
Contd.

- We find relative position by equation as follow:

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

- *Similarly*

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



Contd.

- Solving for x direction by making viewport position as subject we obtain:

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

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

$$x_v = x_{vmin} + (x_w - x_{wmin})s_x$$

- *Where*

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

Contd.

- Similarly Solving for y direction by making viewport position as subject we obtain:

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

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

$$y_v = y_{vmin} + (y_w - y_{wmin})s_y$$

- *Where*

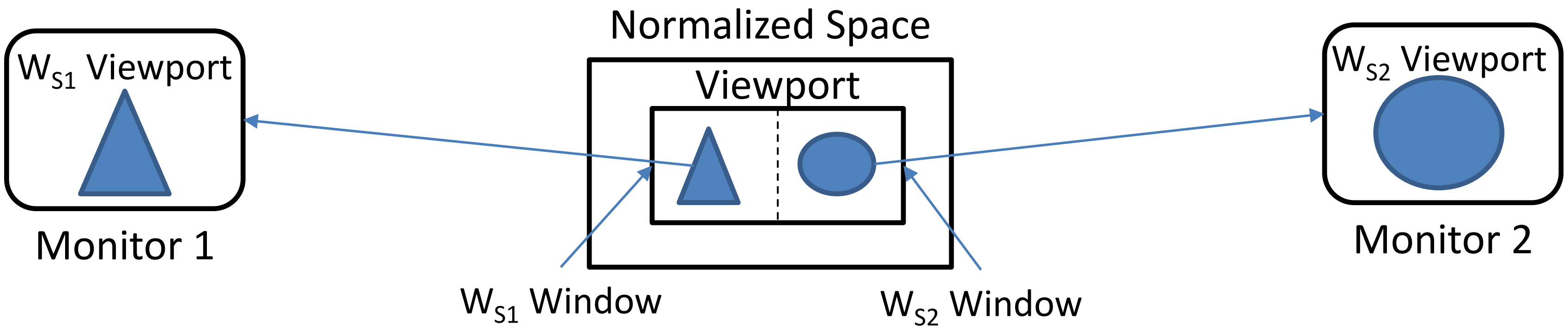
$$s_y = \frac{y_{vmax} - y_{vmin}}{y_{wmax} - y_{wmin}}$$

Contd.

- We can also map window to viewport with the set of transformation:
    - Perform a scaling transformation using a fixed-point position of $(x_{wmin}, ywmin)$ that scales the window area to the size of the viewport.
    - Translate the scaled window area to the position of the viewport.
- For maintaining relative proportions we take $(s_x = s_y)$.
- If both are not equal then we get stretched or contracted in either the x or y direction when displayed on the output device.
- Characters are handle in two different way
    - One way is simply maintain relative position like other primitive.
    - Other is to maintain standard character size even though viewport size is enlarged or reduce.

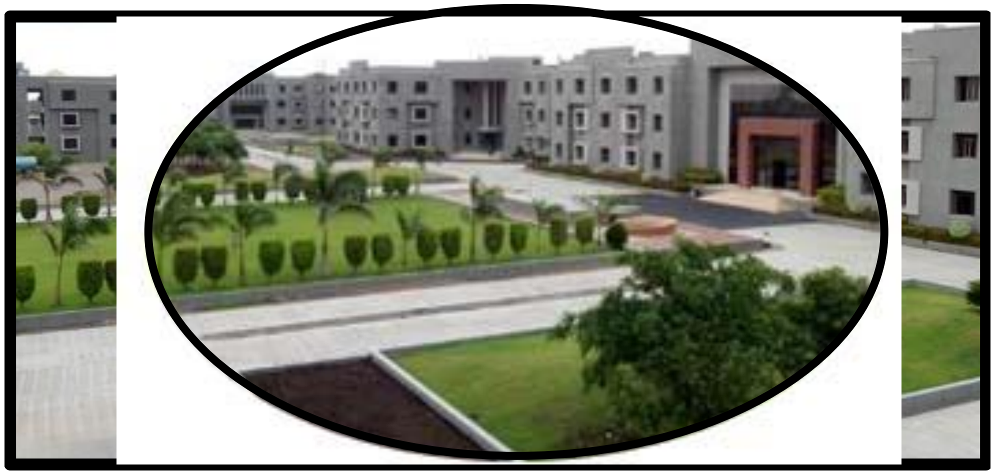
Contd.

- For each display device we can use different window to viewport transformation. This mapping is known as **workstation transformation.**
- Also we can use two different displays devices and we map different window-to-viewport on each one.



Clipping

- Any procedure that identifies those portions of a picture that are either inside or outside of a specified region of space is referred to as a **clipping algorithm**, or simply **clipping**.
- The region against which an object is to clip is called a **clip window.**
- Clip window can be general polygon or it can be curved boundary.



Application of Clipping

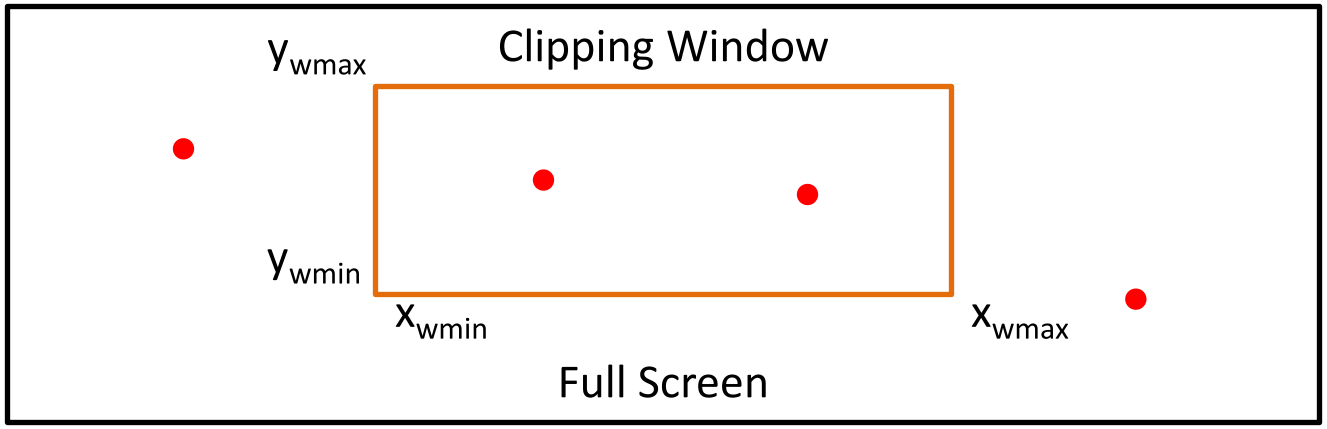
- It can be used for displaying particular part of the picture on display screen.
- Identifying visible surface in 3D views.
- Antialiasing.
- Creating objects using solid-modeling procedures.
- Displaying multiple windows on same screen.
- Drawing and painting.

Point Clipping

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

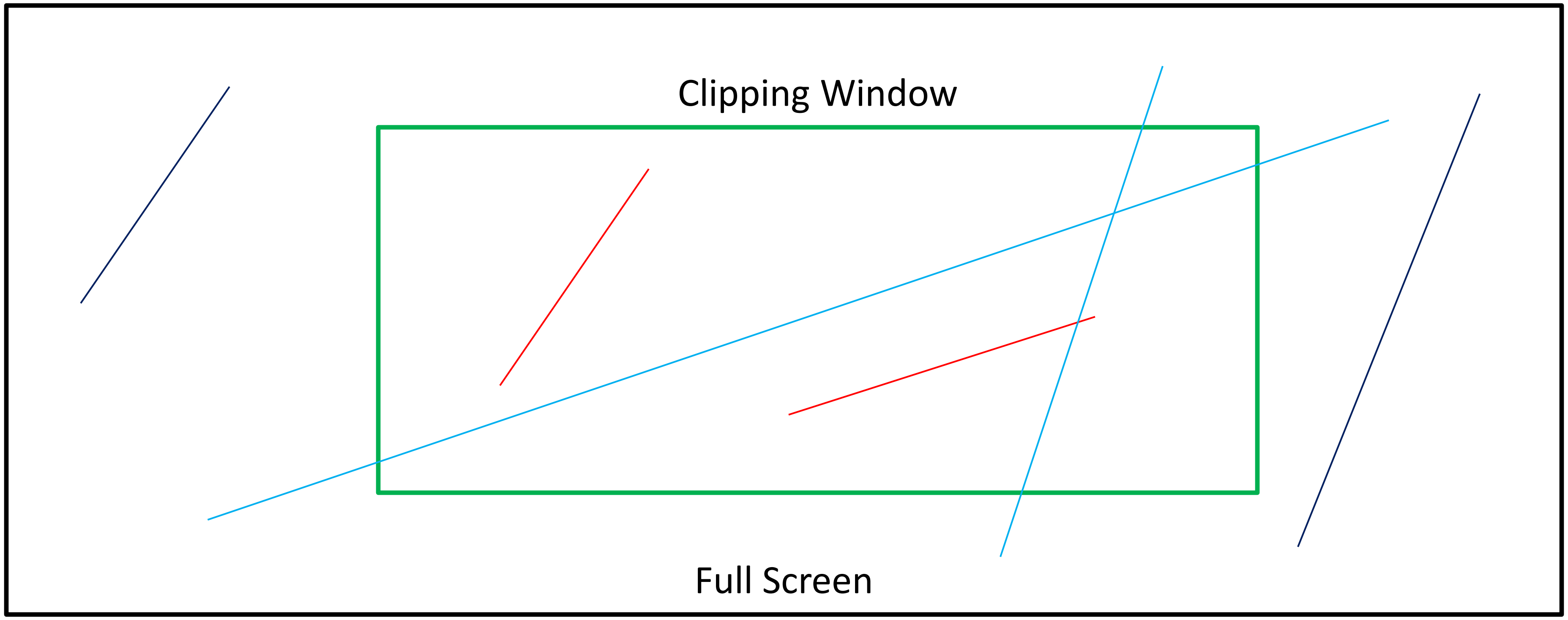
$$\boldsymbol{x_{wmin} \le x \le x_{wamx},} \qquad \boldsymbol{y_{wmin} \le y \le y_{wamx}}$$

- 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.

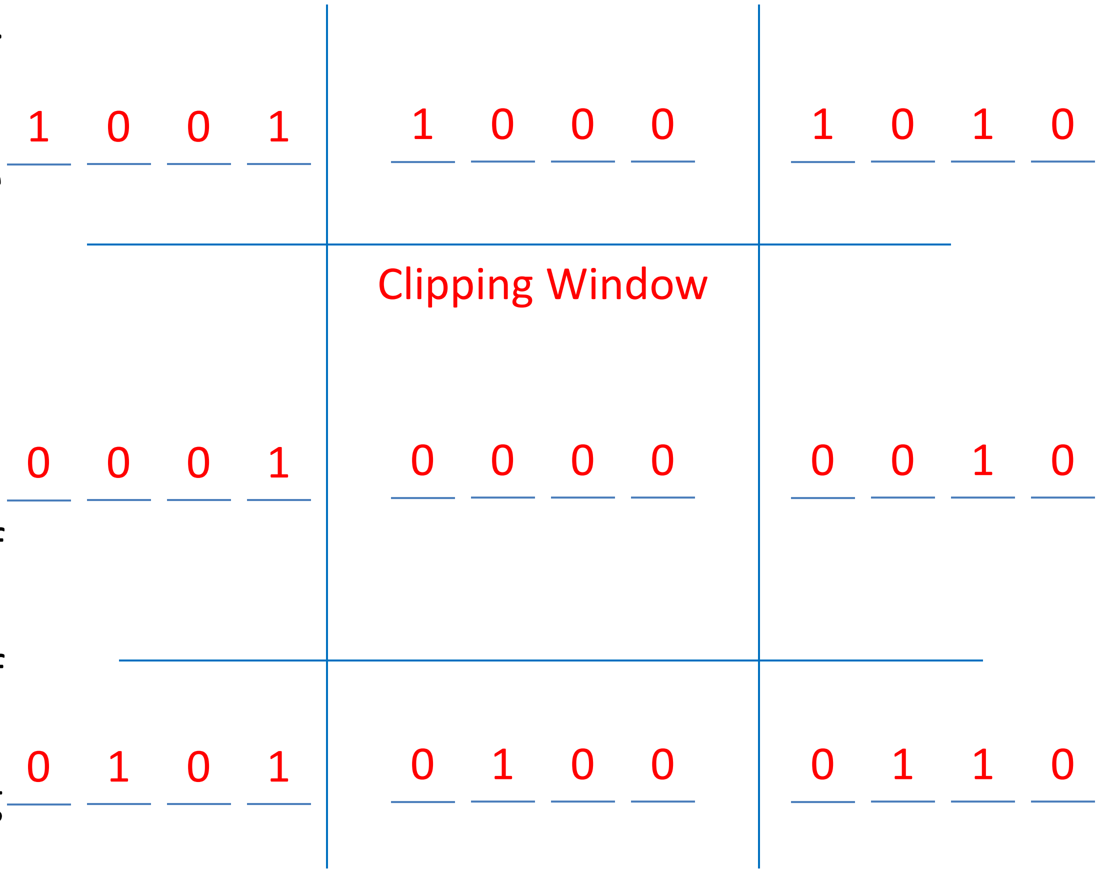


Line Clipping

- For line clipping several scientists tried different methods to solve this clipping procedure.
- Some of them we need to know in this course. Which are:
  1. Cohen-Sutherland Line Clipping (In class)
  2. **Liang-Barsky Line Clipping (Reading required)**
  3. **Nicholl-Lee-Nicholl Line Clipping (Reading required)**

Region Code in Cohen-Sutherland Line Clipping

- This is one of the oldest and most popular line-clipping procedures.
- In this we divide whole space into nine region and give 4 bit region code.
- Code is deriving by:
- Also Called autocodes
  - Set bit 1: For left side of clipping window.
  - Set bit 2: For right side of clipping window.
  - Set bit 3: For below clipping window.
  - Set bit 4: For above clipping window.



Put all other zero (Reset remaining bit) e.g 1001 (1-top 0-bottom 0-right 1 – left)

Steps Cohen-Sutherland Line Clipping

Step-1:

Assign region code to both endpoint of a line depending on the position where the line endpoint is located.

Step-2:

```
If both endpoint have code ‘0000’
        Then line is completely inside.
Otherwise
        Perform logical ending between this two codes.

        If result of logical ending is non-zero
                Line is completely outside the clipping window.
        Otherwise
                Calculate the intersection point with the boundary one by one.
                Divide the line into two parts from intersection point.
                Repeat from Step-1 for both line segments.
```

Step-3:

Draw line segment which are completely inside and eliminate other line segment which found completely outside.

Intersection points- Cohen-Sutherland Algorithm

- For intersection calculation we use line equation "$y = mx + b$".
- "x" is constant for left and right boundary which is:
  - for left "$x = x_{wmin}$"
  - for right "$x = x_{wmax}$"
- So we calculate y coordinate of intersection for this boundary by putting values of x depending on boundary is left or right in below equation.

$$\boldsymbol{y = y_1 + m(x - x_1)}$$

Contd.

- "y" coordinate is constant for top and bottom boundary which is:
  - for top "$y = y_{wmax}$"
  - for bottom "$y = y_{wmin}$"
- So we calculate x coordinate of intersection for this boundary by putting values of y depending on boundary is top or bottom in below equation.

$$\boldsymbol{x = x_1 + \frac{y - y_1}{m}}$$

Liang-Barsky Line Clipping

- Line clipping approach is given by the Liang and Barsky is faster then cohen-sutherland line clipping.
- Which is based on analysis of the parametric equation of the line which are,

$$x = x_1 + u\Delta x$$
$$y = y_1 + u\Delta y$$

- Where $0 \leq u \leq 1$,$\Delta x = x_2 - x_1$ and $\Delta y = y_2 - y_1$.

Liang-Barsky Line Clipping Algorithm

1. Read two end points of line $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$.
2. Read two corner vertices, left top and right bottom of window: (x_{wmin}, y_{wmax}) and (x_{wmax}, y_{wmin}).
3. Calculate values of parameters p_k and q_k for $k = 1, 2, 3, 4$ such that,

$$
\begin{aligned}
p_1 &= -\Delta x, & q_1 &= x_1 - x_{wmin} \\
p_2 &= \Delta x, & q_2 &= x_{wmax} - x_1 \\
p_3 &= -\Delta y, & q_3 &= y_1 - y_{wmin} \\
p_4 &= \Delta y, & q_4 &= y_{wmax} - y_1
\end{aligned}
$$

Contd.

```
4.  If $p_k = 0$ for any value of $k = 1, 2, 3, 4$ then,
        Line is parallel to $k^{th}$ boundary.

        If corresponding $q_k < 0$ then,
            Line is completely outside the boundary. Therefore, discard
            line segment and Go to Step 8.
        Otherwise
            Check line is horizontal or vertical and accordingly check
            line end points with corresponding boundaries.

            If line endpoints lie within the bounded area
                Then use them to draw line.
            Otherwise
                Use boundary coordinates to draw line.
                And go to Step 8.
```

Contd.

5. For $k = 1,2,3,4$ calculate r_k for nonzero values of p_k and q_k as follows:

$$r_k = \frac{q_k}{p_k}, for\ k = 1,2,3,4$$

6. Find $u_1\ and\ u_2$ as given below:

$$u_1 = \max\{0, r_k | where\ k\ takes\ all\ values\ for\ which\ p_k < 0\}$$

$$u_2 = \min\{1, r_k | where\ k\ takes\ all\ values\ for\ which\ p_k > 0\}$$

7. If $u_1 \leq u_2$ then

   Calculate endpoints of clipped line:

   $x_1' = x_1 + u_1 \Delta x$

   $y_1' = y_1 + u_1 \Delta y$

   $x_2' = x_1 + u_2 \Delta x$

   $y_2' = y_1 + u_2 \Delta y$

   Draw line (x_1', y_1', x_2', y_2');

8. Stop.

Advantages of **Liang-Barsky Line Clipping**

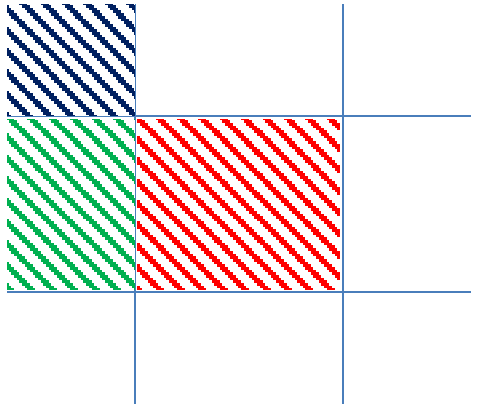
1. More efficient.
2. Only requires one division to update u_1 and u_2.
3. Window intersections of line are calculated just once.

Nicholl-Lee-Nicholl Line (NLN) Clipping

- By creating more regions around the clip window the NLN algorithm avoids multiple clipping of an individual line segment.
- In Cohen-Sutherlan line clipping sometimes multiple calculation of intersection point of a line is done before actual window boundary intersection.
- These multiple intersection calculation is avoided in NLN line clipping procedure.
- NLN line clipping perform the fewer comparisons and divisions so it is more efficient.
- But NLN line clipping cannot be extended for three dimensions.

Contd.

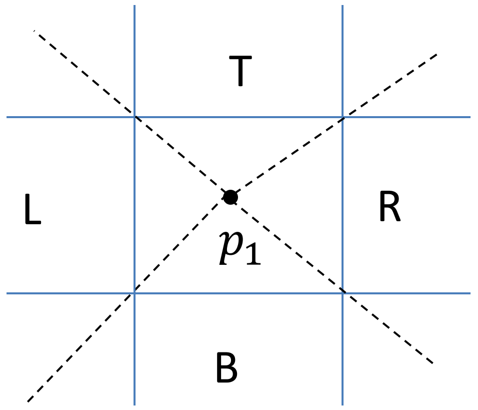
- For given line we find first point falls in which region out of nine region shown in figure.
- Only three region are considered which are.
    - Window region
    - Edge region
    - Corner region



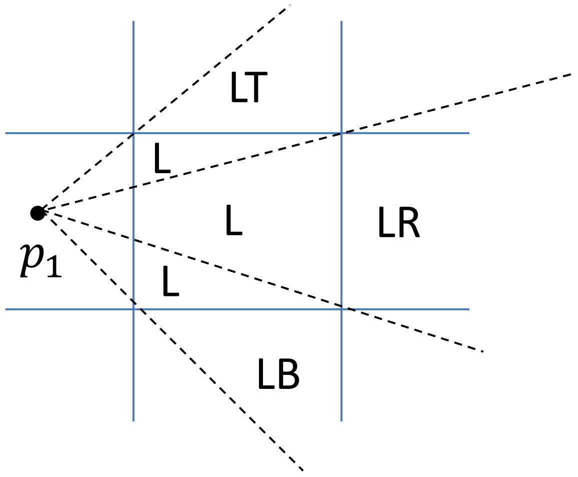
- If point falls in other region than we transfer that point in one of the three region by using transformations.
- We can also extend this procedure for all nine regions.

Dividing Region in NLN

- Based on position of first point out of three region highlighted we divide whole space in new regions.
- Regions are name in such a way that name in which region p2 falls is gives the window edge which intersects the line.
- $p_1$is in window region

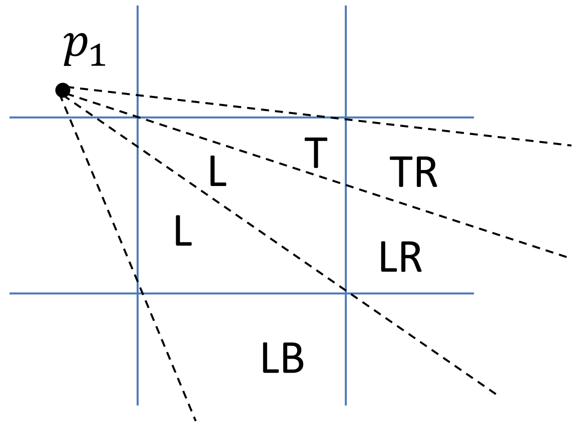


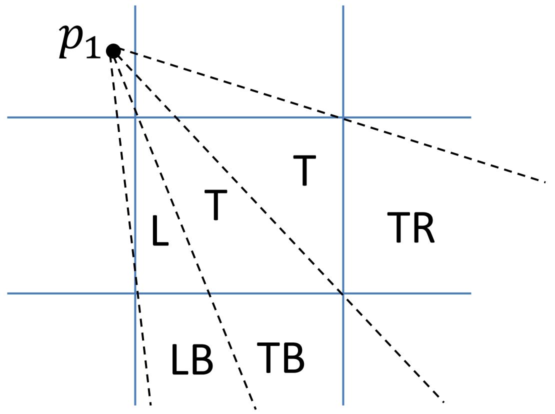
- $p_1$is in edge region



Contd.

- p_1 is in Corner region (one of the two possible sets of region can be generated)





Finding Region of Given Line in NLN

- For finding that in which region line $\boldsymbol{p_1 p_2}$ falls we compare the slope of the line to the slope of the boundaries:

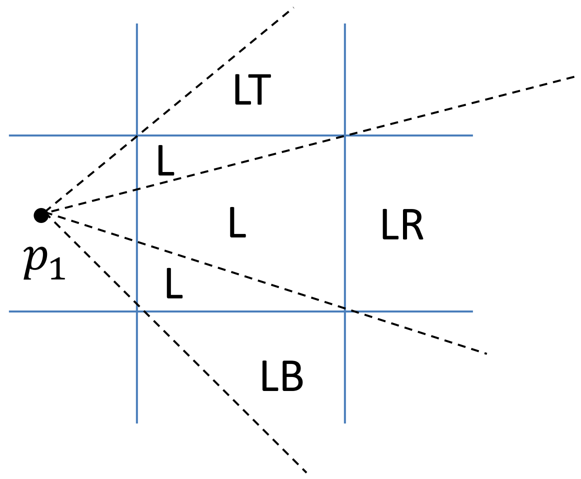
  $$\boldsymbol{slope\ \overline{p_1 p_{B1}}} < \boldsymbol{slope\ \overline{p_1 p_2}} < slope\ \boldsymbol{\overline{p_1 p_{B2}}}$$

  Where $\boldsymbol{\overline{p_1 p_{B1}}}$ and $\boldsymbol{\overline{p_1 p_{B2}}}$ are boundary lines.

- For example p1 is in edge region and for checking whether p2 is in region LT we use following equation.

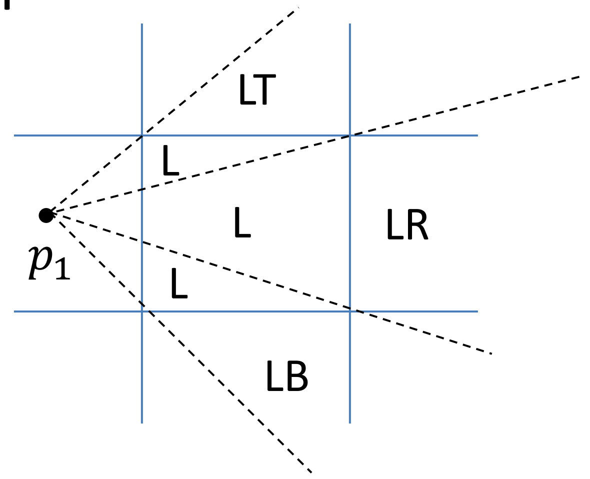
  $$\boldsymbol{slope\ \overline{p_1 p_{TR}}} < \boldsymbol{slope\ \overline{p_1 p_2}} < slope\ \boldsymbol{\overline{p_1 p_{TL}}}$$

  $$\boldsymbol{\frac{y_T - y_1}{x_R - x_1} < \frac{y_2 - y_1}{x_2 - x_1} < \frac{y_T - y_1}{x_L - x_1}}$$



Contd.

- After checking slope condition we need to check weather it crossing zero, one or two edges.
- This can be done by comparing coordinates of p_2 with coordinates of window boundary.
- For left and right boundary we compare x coordinates and for top and bottom boundary we compare y coordinates.
- If line is not fall in any defined region than clip entire line.
- Otherwise calculate intersection.



Intersection Calculation in NLN

- After finding region we calculate intersection point using parametric equation which are:

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

- For left or right boundary $x = x_l \; or \; x_r$ respectively, with $u = (x_{l/r} - x_1)/(x_2 - x_1)$, so that y can be obtain from parametric equation as below:

  $$\boldsymbol{y = y_1 + \frac{y_2 - y_1}{x_2 - x_1}(x_L - x_1)}$$

- Keep the portion which is inside and clip the rest.

Contd.

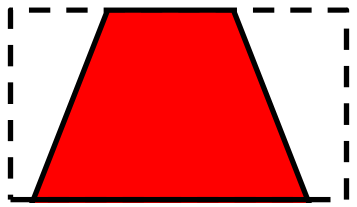
- Similarly for top or bottom boundary $y = y_t \; or \; y_b$ respectively, and $u = (y_{t/b} - y_1)/(y_2 - y_1)$, so that we can calculate x intercept as follow:

$$\boldsymbol{x = x_1 + \frac{x_2 - x_1}{y_2 - y_1}(y_T - y_1)}$$

Polygon Clipping

- For polygon clipping we need to modify the line clipping procedure.
- In line clipping we needed to consider about only line segment.
- In polygon clipping we need to consider the area and the new boundary of the polygon after clipping.
- Various algorithm available for polygon clipping are:

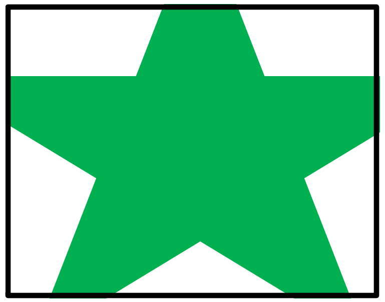
1. Sutherland-Hodgeman Polygon Clipping (inclass)
2. **Weiler-Atherton Polygon Clipping (read)**



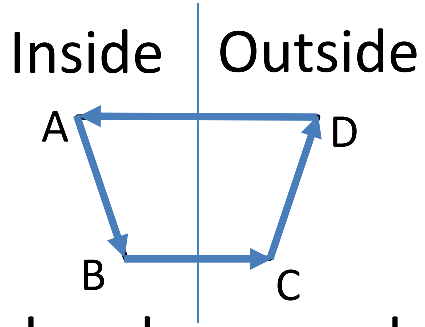
Sutherland-Hodgeman Polygon Clipping

- For correctly clip a polygon we process the polygon boundary as a whole against each window edge.
- This is done by whole polygon vertices against each clip rectangle boundary one by one.

$$\begin{tikzcd}
\text{in} \arrow[r] & \boxed{\begin{array}{c}\text{Left}\\\text{Clipper}\end{array}} \arrow[r] & \boxed{\begin{array}{c}\text{Right}\\\text{Clipper}\end{array}} \arrow[r] & \boxed{\begin{array}{c}\text{Bottom}\\\text{Clipper}\end{array}} \arrow[r] & \boxed{\begin{array}{c}\text{Top}\\\text{Clipper}\end{array}} \arrow[r] & \text{out}
\end{tikzcd}$$



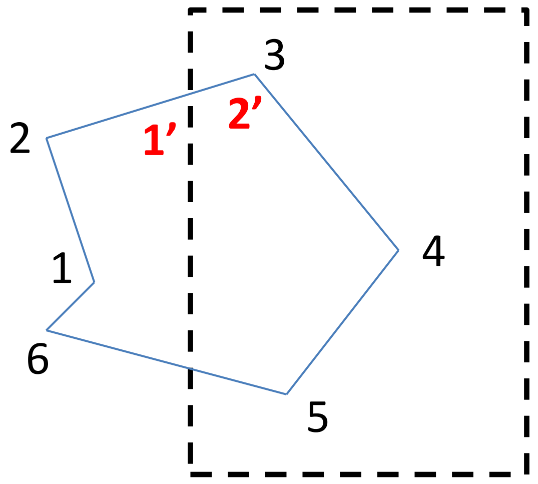
Processing Steps



- We process vertices in sequence as a closed polygon.
- Four possible cases are there.

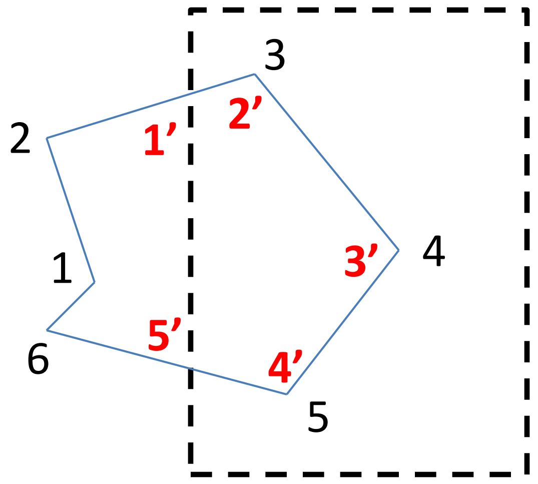
1. If both vertices are inside the window we add only second vertices to output list.
2. If first vertices is inside the boundary and second vertices is outside the boundary only the edge intersection with the window boundary is added to the output vertex list.
3. If both vertices are outside the window boundary nothing is added to window boundary.
4. first vertex is outside and second vertex is inside the boundary, then adds both intersection point with window boundary, and second vertex to the output list.

Example



- As shown in figure we clip against left boundary.
- Vertices 1 and 2 are found to be on the outside of the boundary.
- Then we move to vertex 3, which is inside, we calculate the intersection and add both intersection point and vertex 3 to output list.

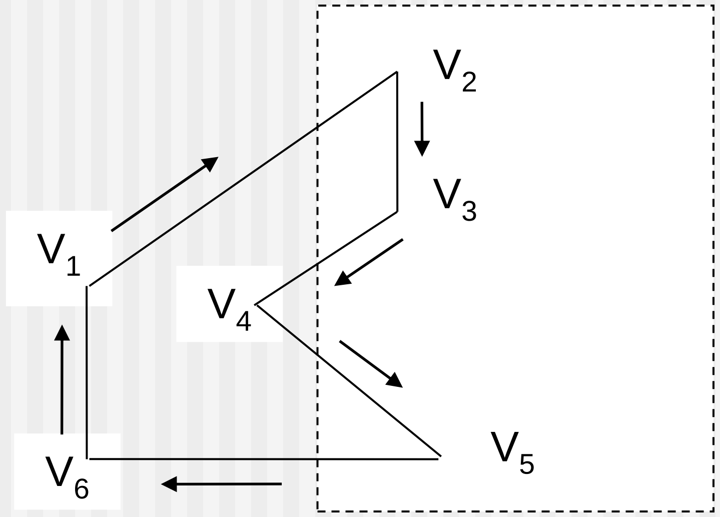
Contd.



- Then we move to vertex 4 in which vertex 3 and 4 both are inside so we add vertex 4 to output list.
- Similarly from 4 to 5 we add 5 to output list.
- From 5 to 6 we move inside to outside so we add intersection pint to output list.
- Finally 6 to 1 both vertex are outside the window so we does not add anything.

Limitation of Sutherlan-Hodgeman Algorithm

- It may not clip concave polygon properly.

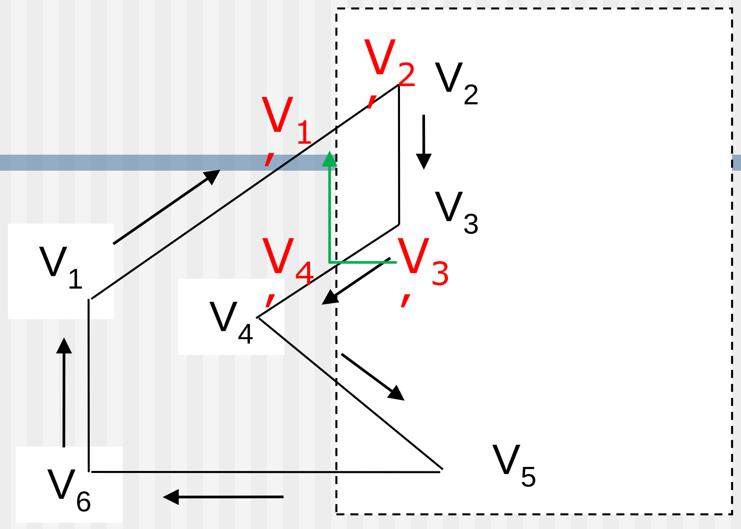


- One possible solution is to divide polygon into numbers of small convex polygon and then process one by one.
- Another approach is to use Weiler-Atherton algorithm.

Weiler-Atherton Polygon Clipping

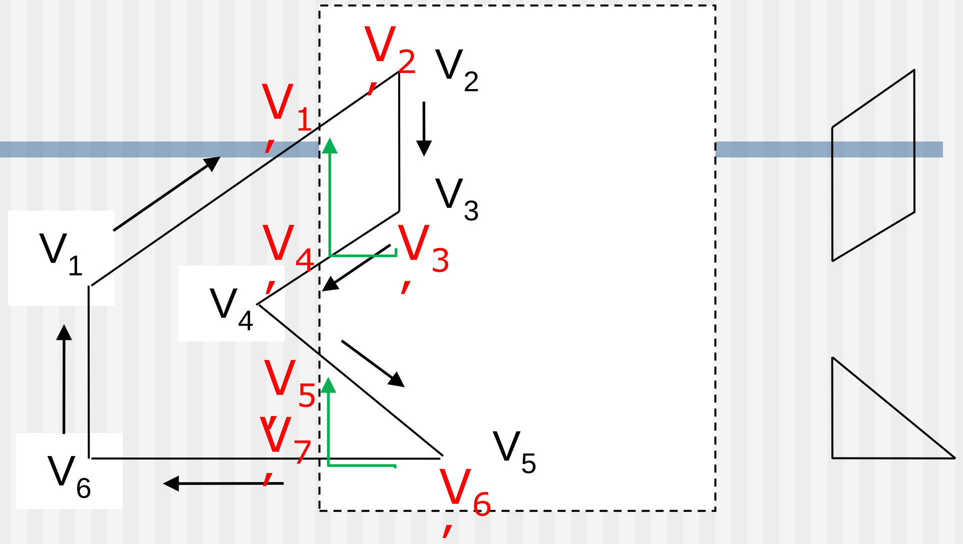
- It modifies Sutherland-Hodgeman vertex processing procedure for window boundary so that concave polygon also clip correctly.
- This can be applied for arbitrary polygon clipping regions as it is developed for visible surface identification.
- Procedure is similar to Sutherland-Hodgeman algorithm.
- Only change is sometimes need to follow the window boundaries Instead of always follow polygon boundaries.
- For clockwise processing of polygon vertices we use the following rules:
  1. For an outside to inside pair of vertices, follow the polygon boundary.
  2. For an inside to outside pair of vertices, follow the window boundary in a clockwise direction.

Example



- Start from v1 and move clockwise towards v2 and add intersection point and next point to output list by following polygon boundary,
- then from v2 to v3 we add v3 to output list.
- From v3 to v4 we calculate intersection point and add to output list and follow window boundary.

Contd.



- Similarly from v4 to v5 we add intersection point and next point and follow the polygon boundary,
- next we move v5 to v6 and add intersection point and follow the window boundary, and
- finally v6 to v1 is outside so no need to add anything.
- This way we get two separate polygon section after clipping.

Thank You