# CST 304 COMPUTER GRAPHICS AND IMAGE PROCESSING

MODULE 2

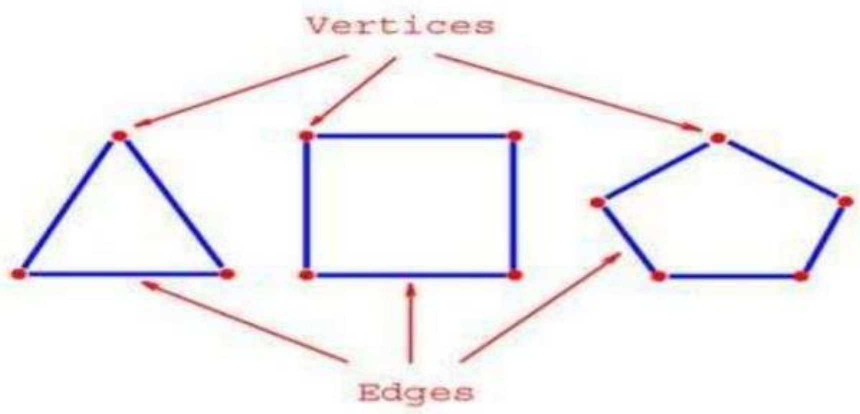
## SYLLABUS

- Filled Area Primitives
- Scan line polygon filling, Boundary filling and flood filling.
- Two dimensional transformations-Translation, Rotation, Scaling, Reflection and Shearing.
- Composite transformations
- Matrix representations and homogeneous coordinates.
- Basic 3D transformations.

# Polygon filling algorithm

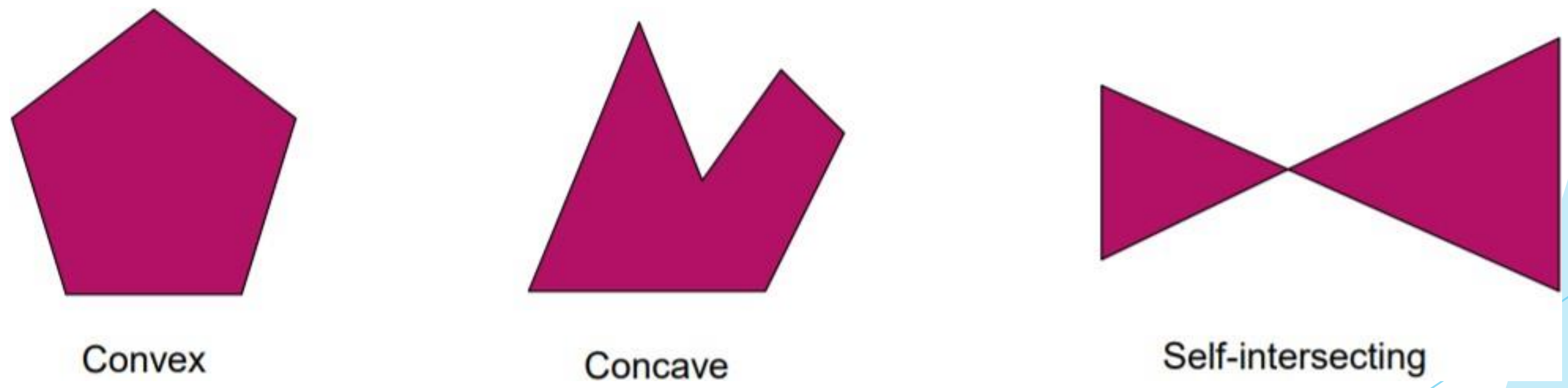
**Polygon**

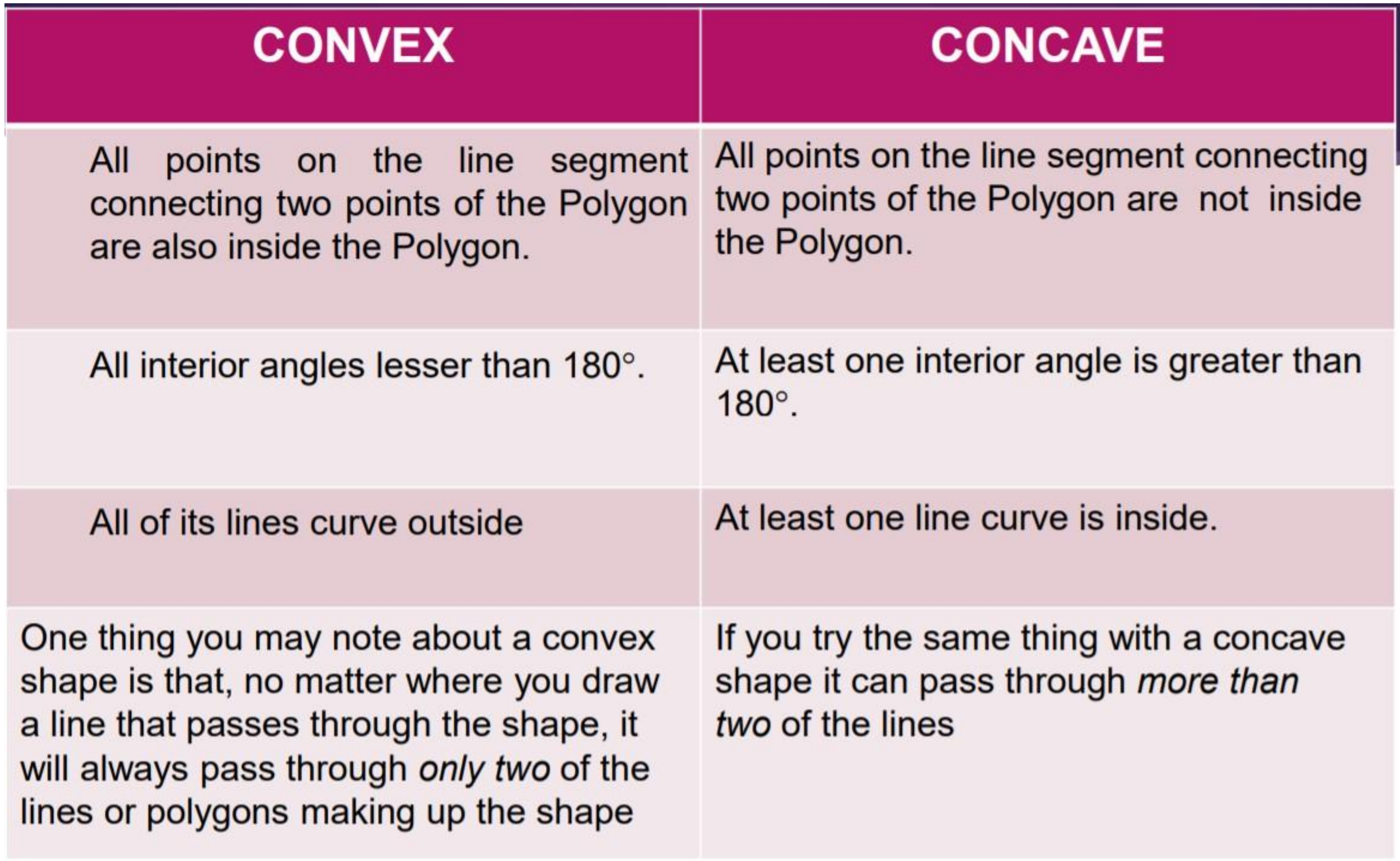
- A closed figure represented by a collection of more than 2 line segments connected end to end.
- The line segments are known as “Edge” of the Polygon and make up the Polygon boundary.
- Endpoints of the edges are known as “Vertex” of the Polygon.



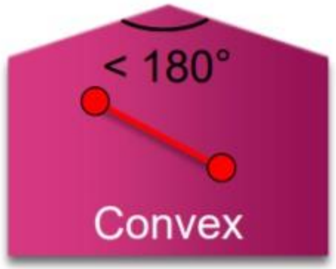
## Types Of Polygon

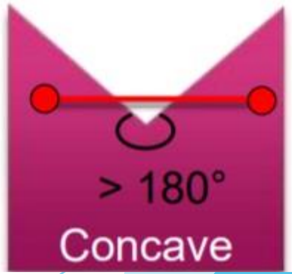
- Simple Convex
- Simple Concave
- Non-simple : self-intersecting



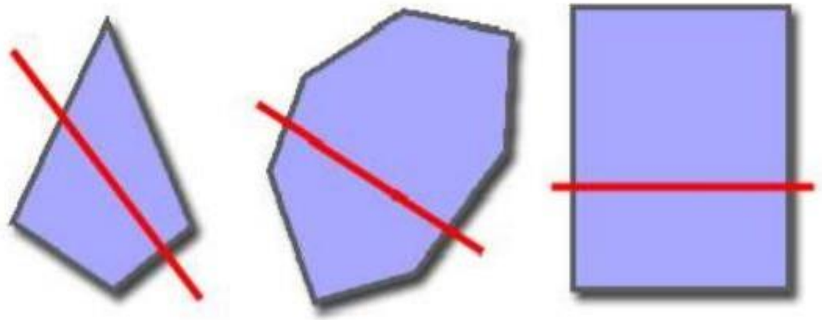


| CONVEX | CONCAVE |
|---|---|
| All points on the line segment connecting two points of the Polygon are also inside the Polygon. | All points on the line segment connecting two points of the Polygon are not inside the Polygon. |
| All interior angles lesser than 180°. | At least one interior angle is greater than 180°. |
| All of its lines curve outside | At least one line curve is inside. |
| One thing you may note about a convex shape is that, no matter where you draw a line that passes through the shape, it will always pass through *only two* of the lines or polygons making up the shape | If you try the same thing with a concave shape it can pass through *more than two* of the lines |

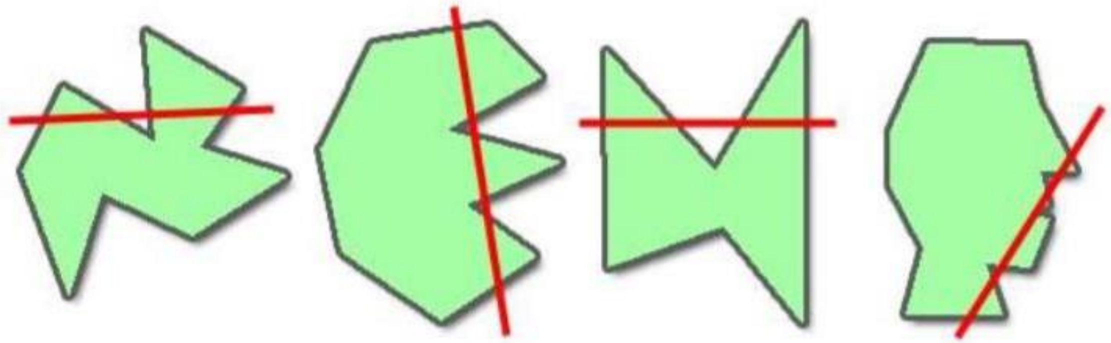




- **CONVEX POLYGON:**



- **CONCAVE POLYGON:**



## Polygon Filling

- Filling a Polygon is the process of coloring every pixel that comes inside the Polygon region.

**Techniques:**

- Boundary Fill Method
- Flood Fill Method
- Scan – Line Fill Method

## Boundary Fill Method

- Boundary Fill Algorithm starts at a pixel inside the polygon to be filled and paints the interior proceeding outwards towards the boundary.

- This algorithm works **only if** the color with which the region has to be filled and the color of the boundary of the region are different.

- If the boundary is of one single color, this approach proceeds outwards pixel by pixel until it hits the boundary of the region.

## Explanation

1. Draw a closed region using a boundary color.
2. Select an interior point (x,y). (any point that lies inside the polygon)
3. Get the color of the pixel (x,y)

Color= getpixel(x,y)

If color not equal to boundary color and if that pixel is not already colored

Then fill it with fill color.

- ***Boundary Fill Algorithm is recursive in nature.*** It takes an interior point(x, y), a fill color, and a boundary color as the input.

- The algorithm starts by checking the color of (x, y).

- If it's color is not equal to the fill color and the boundary color, then it is painted with the fill color and the function is called for all the neighbours of (x, y).

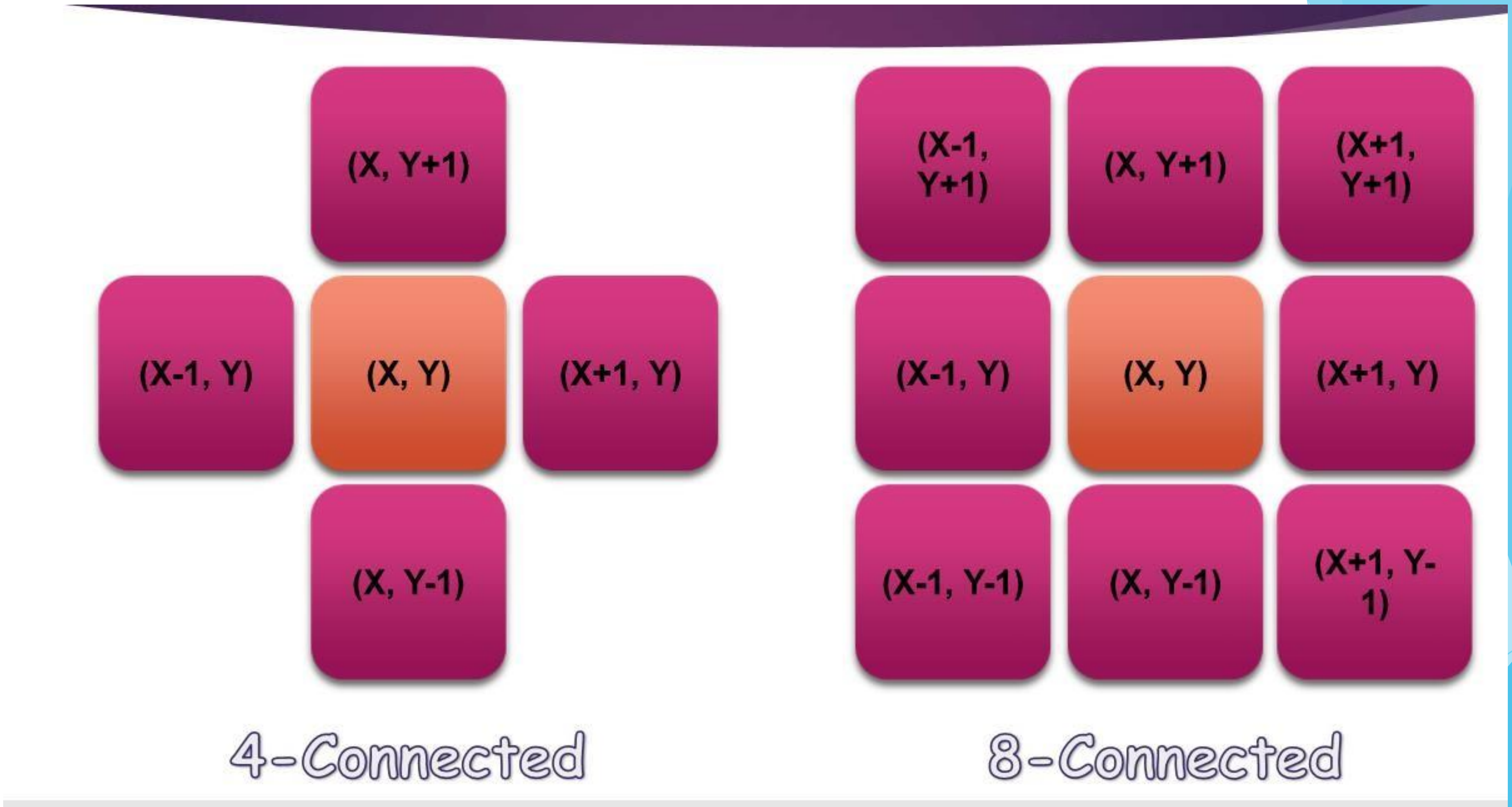
- If a point is found to be of fill color or of boundary color, the function does not call its neighbours and returns.

- This process continues until all points up to the boundary color for the region have been tested.

- The boundary fill algorithm can be implemented by 4-connected pixels or 8-connected pixels.

- **4-connected pixels :** After painting a pixel, the function is called for four neighboring points.

- These are the pixel positions that are right, left, above and below the current pixel. Areas filled by this method are called 4-connected.


(X, Y+1)
(X-1, Y)
(X, Y)
(X+1, Y)
(X, Y-1)
4-Connected
(X-1, Y+1)
(X, Y+1)
(X+1, Y+1)
(X-1, Y)
(X, Y)
(X+1, Y)
(X-1, Y-1)
(X, Y-1)
(X+1, Y-1)
8-Connected

## 4-Connected

```
void boundaryFill4(int x, int y, int fill_color,int boundary_color)
  { if(getpixel(x, y) != boundary_color && getpixel(x, y) != fill_color)
       { putpixel(x, y, fill_color);
         boundaryFill4(x + 1, y, fill_color, boundary_color);
          boundaryFill4(x, y + 1, fill_color, boundary_color);
          boundaryFill4(x - 1, y, fill_color, boundary_color);
          boundaryFill4(x, y - 1, fill_color, boundary_color);
       }
  }
```

- **8-connected pixels :** More complex figures are filled using this approach.
- The pixels to be tested are the 8 neighboring pixels, the pixel on the right, left, above, below and the 4 diagonal pixels.
- Areas filled by this method are called 8-connected.

```
void boundaryFill8(int x, int y, int fill_color,int boundary_color)
    { if(getpixel(x, y) != boundary_color && getpixel(x, y) != fill_color)
        { putpixel(x, y, fill_color); boundaryFill8(x + 1, y, fill_color,
         boundary_color);
         boundaryFill8(x, y + 1, fill_color, boundary_color);
         boundaryFill8(x - 1, y, fill_color, boundary_color);
         boundaryFill8(x, y - 1, fill_color, boundary_color);
         boundaryFill8(x - 1, y - 1, fill_color, boundary_color);
         boundaryFill8(x - 1, y + 1, fill_color, boundary_color);
         boundaryFill8(x + 1, y - 1, fill_color, boundary_color);
         boundaryFill8(x + 1, y + 1, fill_color, boundary_color);
         }
    }
```

## Flood Fill Algorithm

- In this method, a point or seed which is inside region is selected. This point is called a **seed point.**

- Then four connected approaches or eight connected approaches is used to fill with specified color.

- The flood fill algorithm has many characters similar to boundary fill.

- But this method is more suitable for filling multiple colors boundary.

- When boundary is of many colors and interior is to be filled with one color we use this algorithm.

- In fill algorithm, we start from a specified interior point (x, y) and reassign all  pixel values are currently set to a given interior color with the desired color.

- Using either a 4-connected or 8-connected approaches, we then step through pixel

## Explanation

- Draw a closed region and fill the interior region with a color.
- Select an interior point (x,y)
- Get the color of interior pixel
- Color = getpixel(x,y)
- If color= old color
- Setpixel(x,y)=new color
- Continue until all pixels inside the region are replaced using new color.

## 4-connected

```
Procedure floodfill (x, y,fill_ color, old_color: integer)
    If (getpixel (x, y)=old_color)
   {
    setpixel (x, y, fill_color);
    fill (x+1, y, fill_color, old_color);
     fill (x-1, y, fill_color, old_color);
    fill (x, y+1, fill_color, old_color);
    fill (x, y-1, fill_color, old_color);
     }
}
```

## 8-connected

```
Procedure floodfill (x, y,fill_ color, old_color: integer) If (getpixel (x,
   y)=old_color)
  {
   setpixel (x, y, fill_color);
   fill (x+1, y, fill_color, old_color); fill (x-1, y, fill_color,
    old_color);
   fill (x, y+1, fill_color, old_color);  fill (x, y-1, fill_color,
   old_color);  fill (x-1, y-1, fill_color, old_color);  fill (x-1,
   y+1, fill_color, old_color);  fill (x+1, y-1, fill_color,
   old_color);  fill (x+1, y+1, fill_color, old_color).

    }
}
```

## Disadvantage:

- Very slow algorithm
- May be fail for large polygons
- Initial pixel required more knowledge about surrounding pixels.

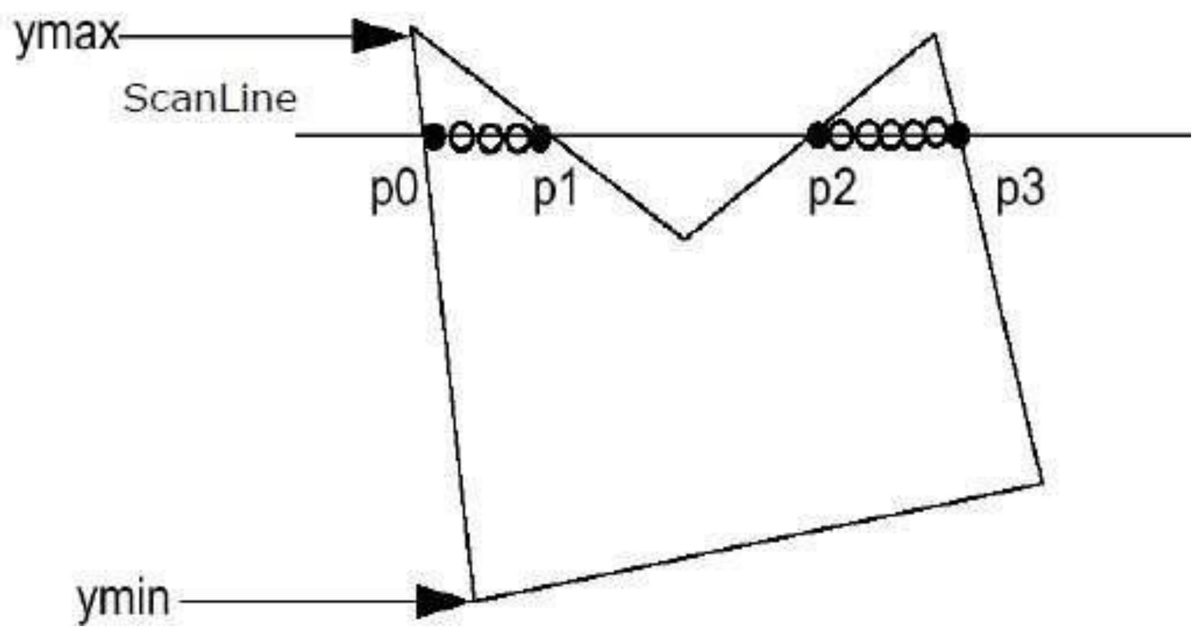
## Scan Line Polygon Fill Algorithm

- Scan line algorithm is a process of filling regions of a polygon that are geometrically defined by the coordinates of vertices of this polygon graph.
- This algorithm is specially used for the region filling just like the boundary-fill and flood-fill algorithm.
- The specialty of this algorithm is that it scans lines at a time rather than scans a pixel.
- This property makes it faster than others.

For a scan line polygon filling there are 3 steps to perform in the following order.

1. Find the intersections of the scan line with all edges of the polygon.
2. Sort the intersections by increasing x-coordinates ie, from left to right
3. Make pairs of the intersections and fill in color within all pixels inside the pair.

## Example



- The polygon edges are being intersected with the scanline by Scan Line Algorithm. The polygon is filled with colours in between the intersecting pairs. The practical working of the algorithm is shown below:

**Step 1** − Find out the Ymin and Ymax from the given polygon.

**Step 2** – From each edge of the polygon from Ymin to Ymax, all the egdes are intersected by Scanline. Each of the points of intersection are named as p0, p1, p2, p3.

**Step 3** − Sort the intersection point in the increasing order of X coordinate i.e. (p0, p1), (p1, p2), and (p2, p3).

**Step 4** − Fill all those pair of coordinates that are inside polygons and ignore the alternate pairs.

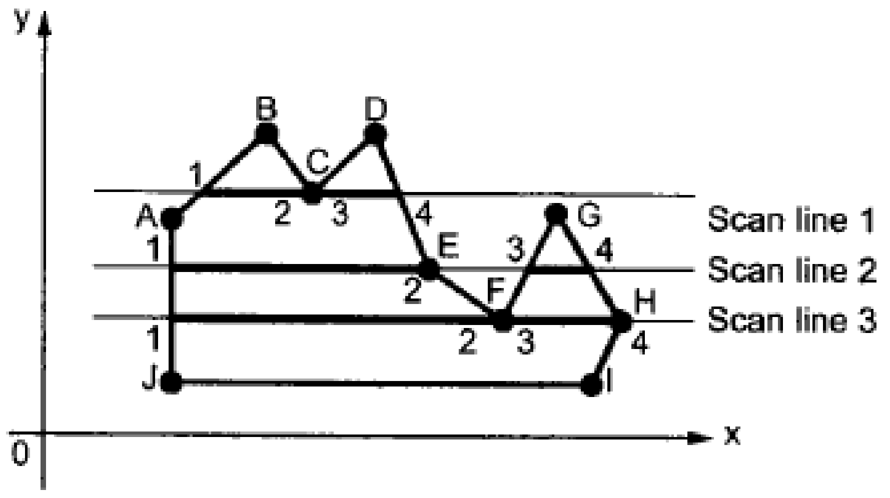


**Fig. (b) Intersection points along the scan line that intersect polygon vertices**

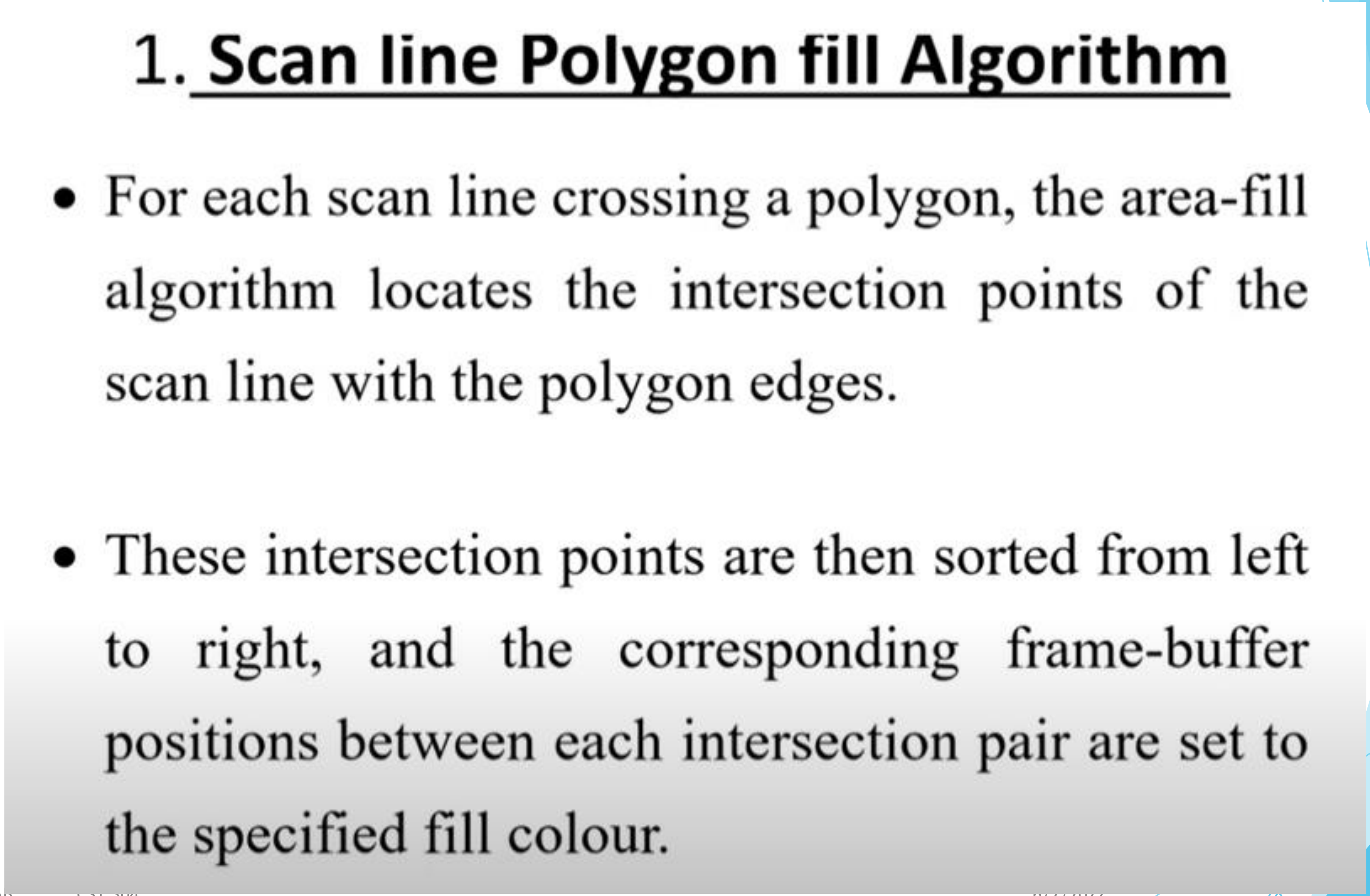
- The slope of the polygon boundary line can be expressed in terms of the scan line intersection coordinates
- $m=(y_{k+1} - y_k)/(x_{k+1} - x_k)$
- Since the change in y coordinates between the two scan lines is simply

  $y_{k+1} - y_k = 1$

The intersection value $x_{k+1}$ , on the upper scan line can be determined from the x-intersection value $x_k$ on the preceding scan line as,

$$\mathbf{x_{k+1} = x_k + 1/m}$$

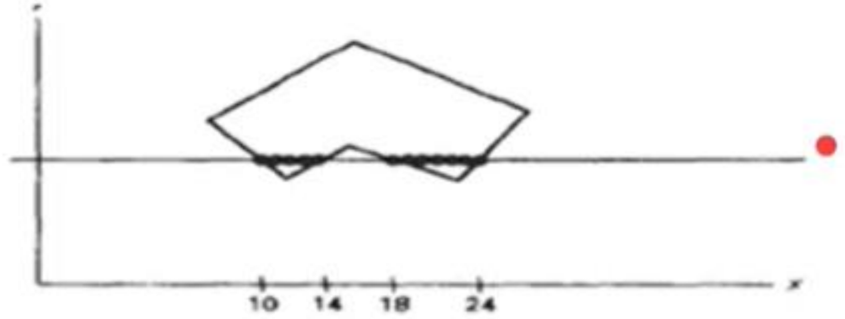
each successive intercept can thus be calculated by adding the inverse of the slope and rounding to the nearest integer.



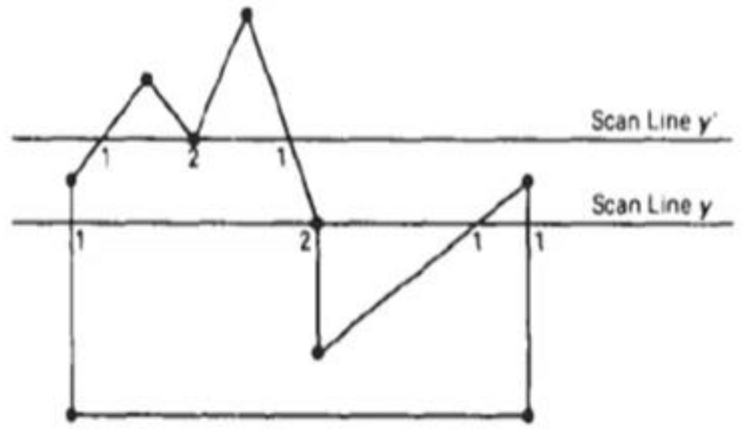
## 1. Scan line Polygon fill Algorithm

- For each scan line crossing a polygon, the area-fill algorithm locates the intersection points of the scan line with the polygon edges.

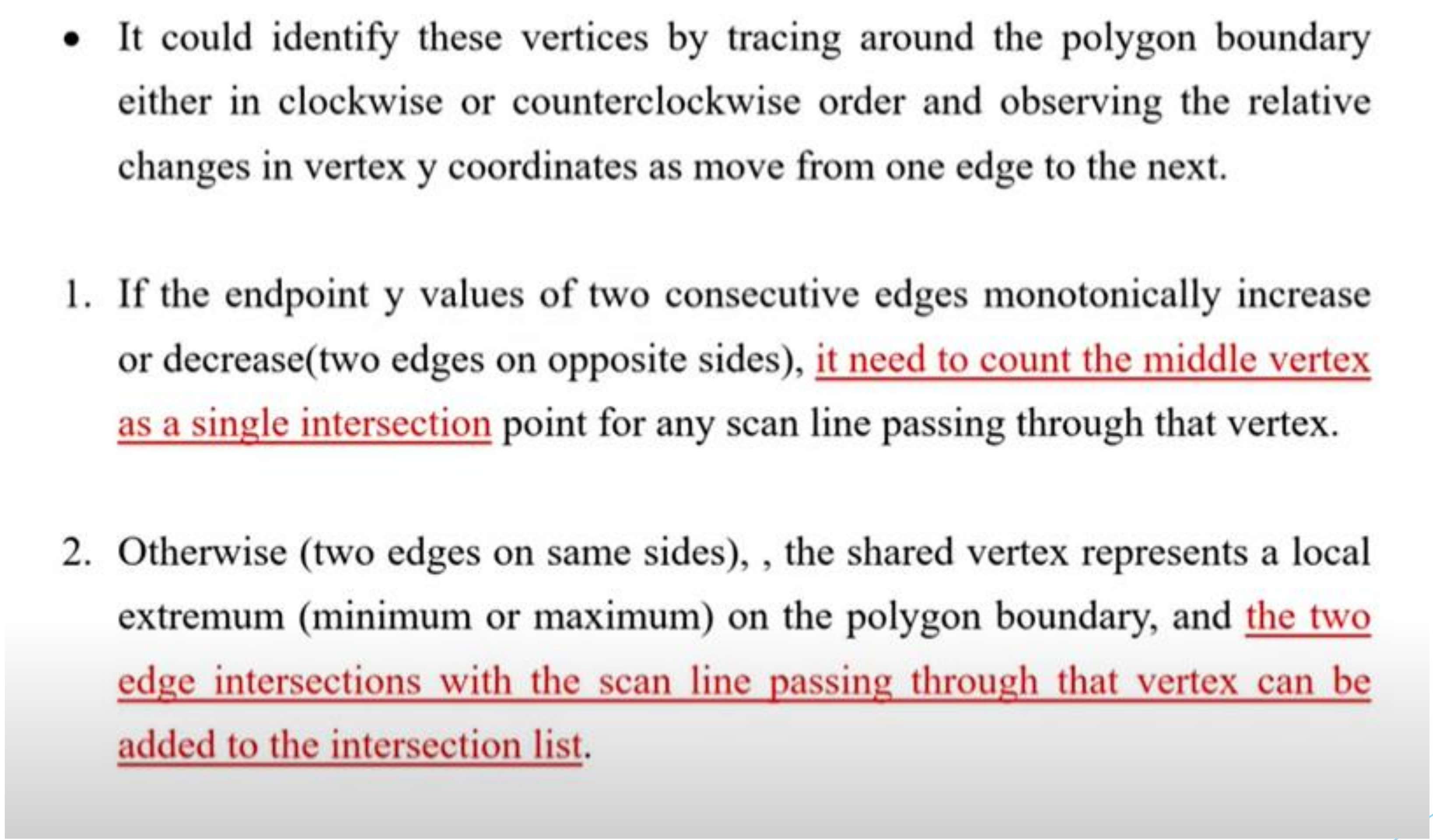
- These intersection points are then sorted from left to right, and the corresponding frame-buffer positions between each intersection pair are set to the specified fill colour.



- In the example ,the four pixel intersection positions with the polygon boundaries define two stretches of interior pixels from $x = 10$ to $x = 14$ and from $x = 18$ to $\mathbf{x} = 24$.
- Some scan-line intersections at polygon vertices require special handling. A scan line passing through a vertex intersects two edges at that position, adding two points to the list of intersections for the scan line.



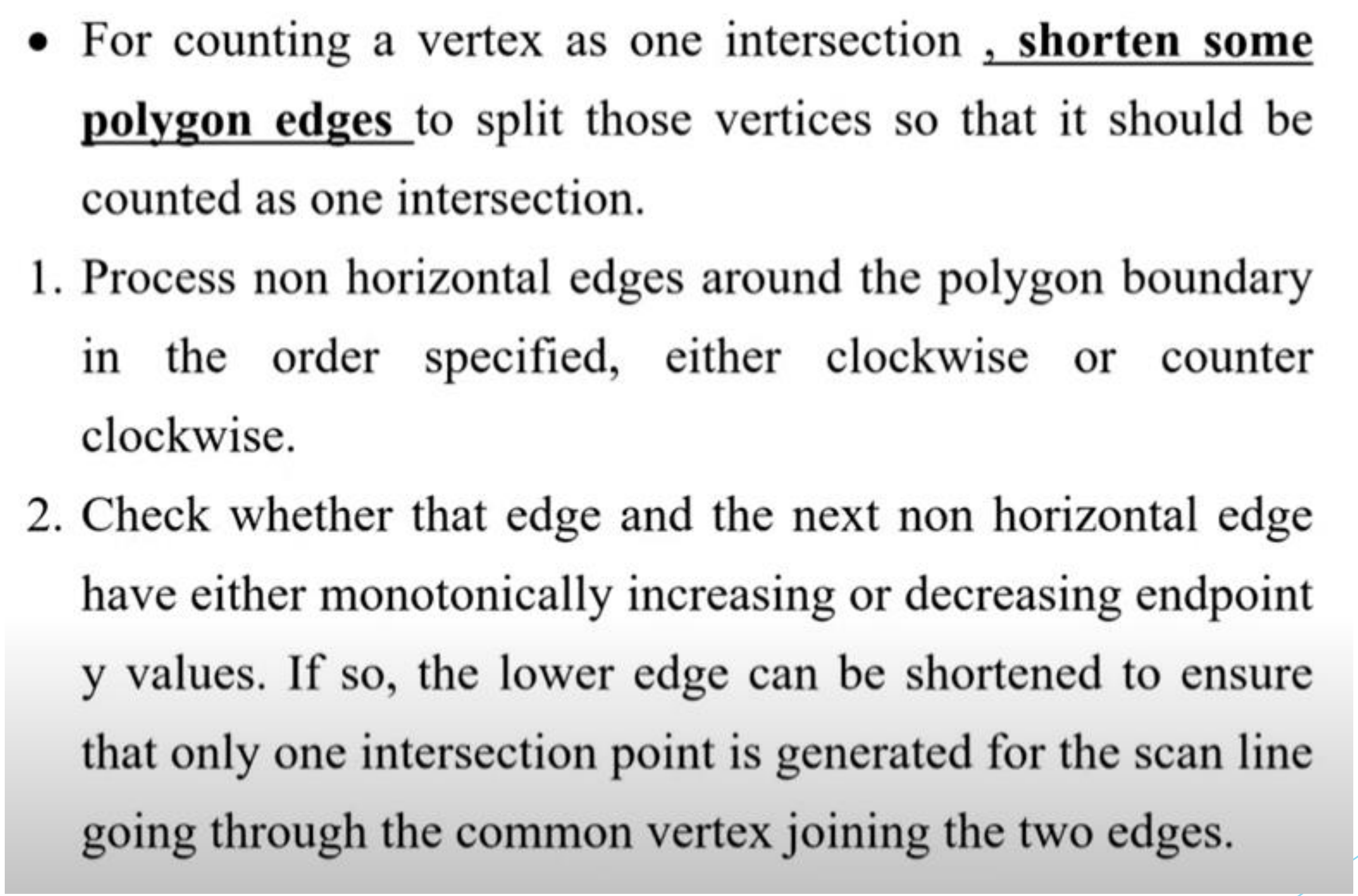
- Figure shows two scan lines at positions y and y' that intersect edge endpoints.
- Scan line y intersects five polygon edges.
- Scan line y', however, intersects an even number of edges although it also passes through a vertex.
- But with scan line y, it need to do some additional processing to determine the correct interior points.
- For scan line y, the two intersecting edges sharing a vertex are on opposite sides of the scan line. But for scan line **y'**, the two intersecting edges are both above the scan line.Thus, the vertices that require additional processing are those that have connecting edges on opposite sides of the scan line.



- It could identify these vertices by tracing around the polygon boundary either in clockwise or counterclockwise order and observing the relative changes in vertex y coordinates as move from one edge to the next.

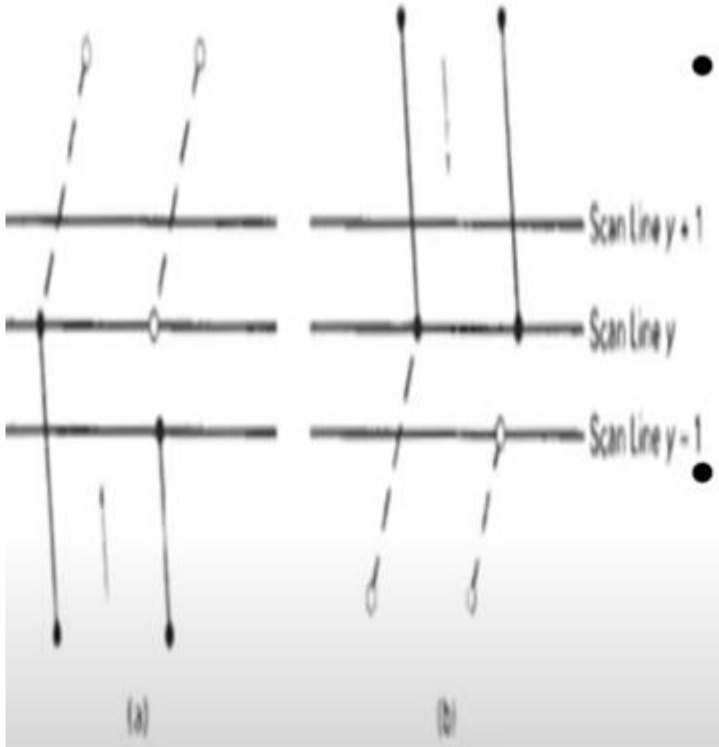
1. If the endpoint y values of two consecutive edges monotonically increase or decrease(two edges on opposite sides), it need to count the middle vertex as a single intersection point for any scan line passing through that vertex.

2. Otherwise (two edges on same sides), , the shared vertex represents a local extremum (minimum or maximum) on the polygon boundary, and the two edge intersections with the scan line passing through that vertex can be added to the intersection list.

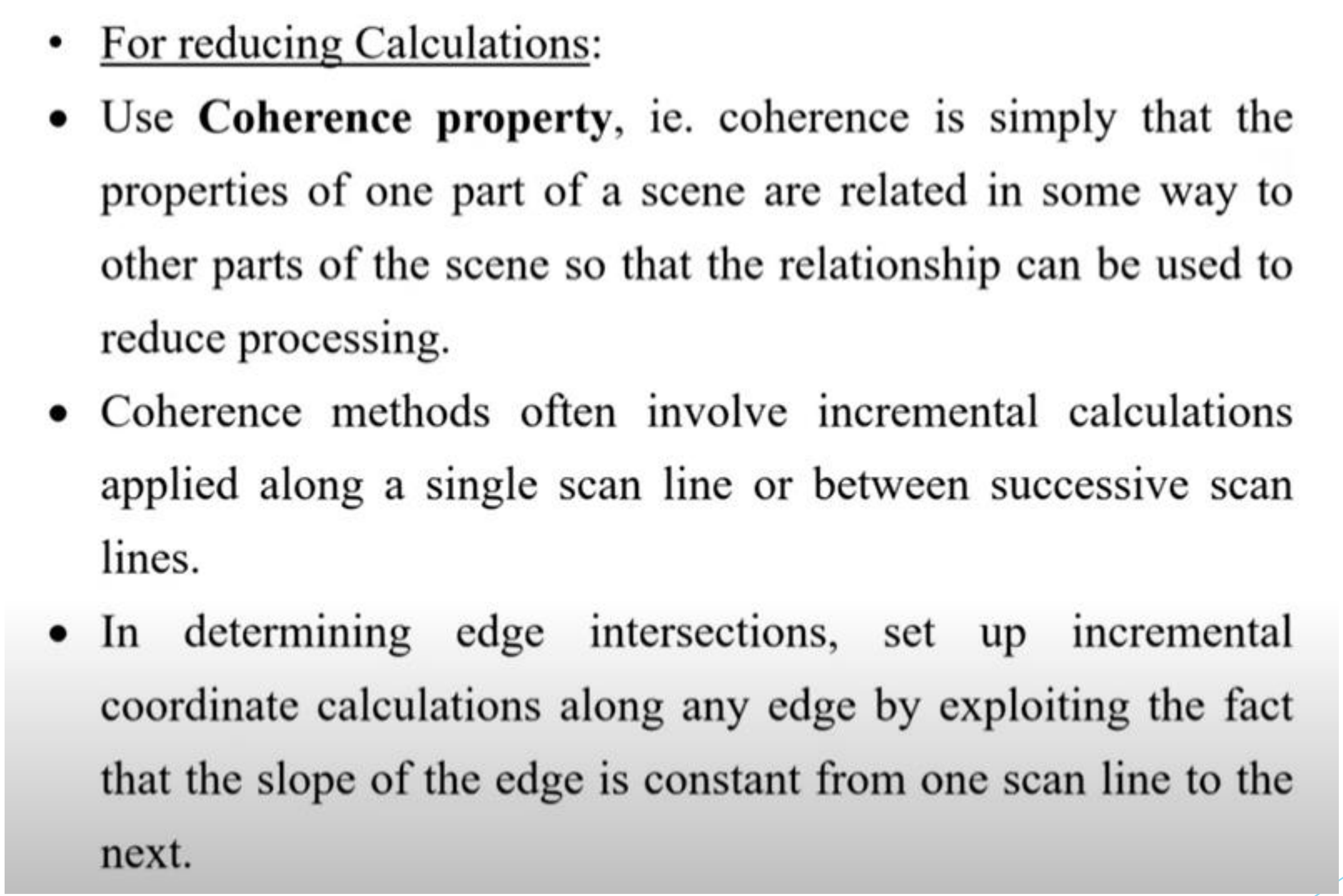


- For counting a vertex as one intersection **, shorten some polygon edges** to split those vertices so that it should be counted as one intersection.

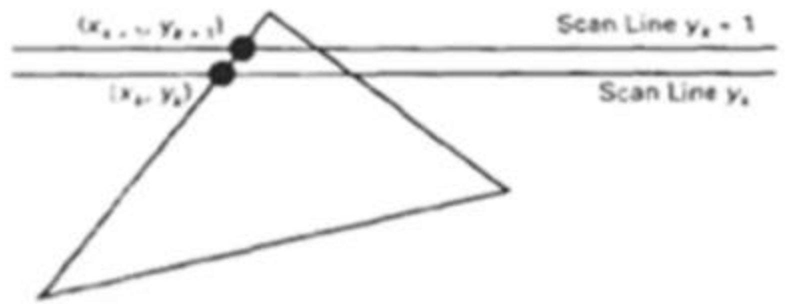
1. Process non horizontal edges around the polygon boundary in the order specified, either clockwise or counter clockwise.
2. Check whether that edge and the next non horizontal edge have either monotonically increasing or decreasing endpoint y values. If so, the lower edge can be shortened to ensure that only one intersection point is generated for the scan line going through the common vertex joining the two edges.



- Figure illustrates shortening of an edge.
- When the endpoint y coordinates of the two edges are increasing, the y value of the upper endpoint for the current edge decreased by 1.
- When the endpoint y values are monotonically decreasing, decrease the y coordinate of the upper endpoint of the edge following the current edge.



- For reducing Calculations:
- Use **Coherence property**, ie. coherence is simply that the properties of one part of a scene are related in some way to other parts of the scene so that the relationship can be used to reduce processing.
- Coherence methods often involve incremental calculations applied along a single scan line or between successive scan lines.
- In determining edge intersections, set up incremental coordinate calculations along any edge by exploiting the fact that the slope of the edge is constant from one scan line to the next.



- The slope of this polygon boundary line can be expressed in terms of the scan-line intersection coordinates:

$$m = \frac{y_{k+1} - y_k}{x_{k+1} - x_k}$$

Since the change in y coordinates between the two scan lines is simply

$$y_{k+1} - y_k = 1$$

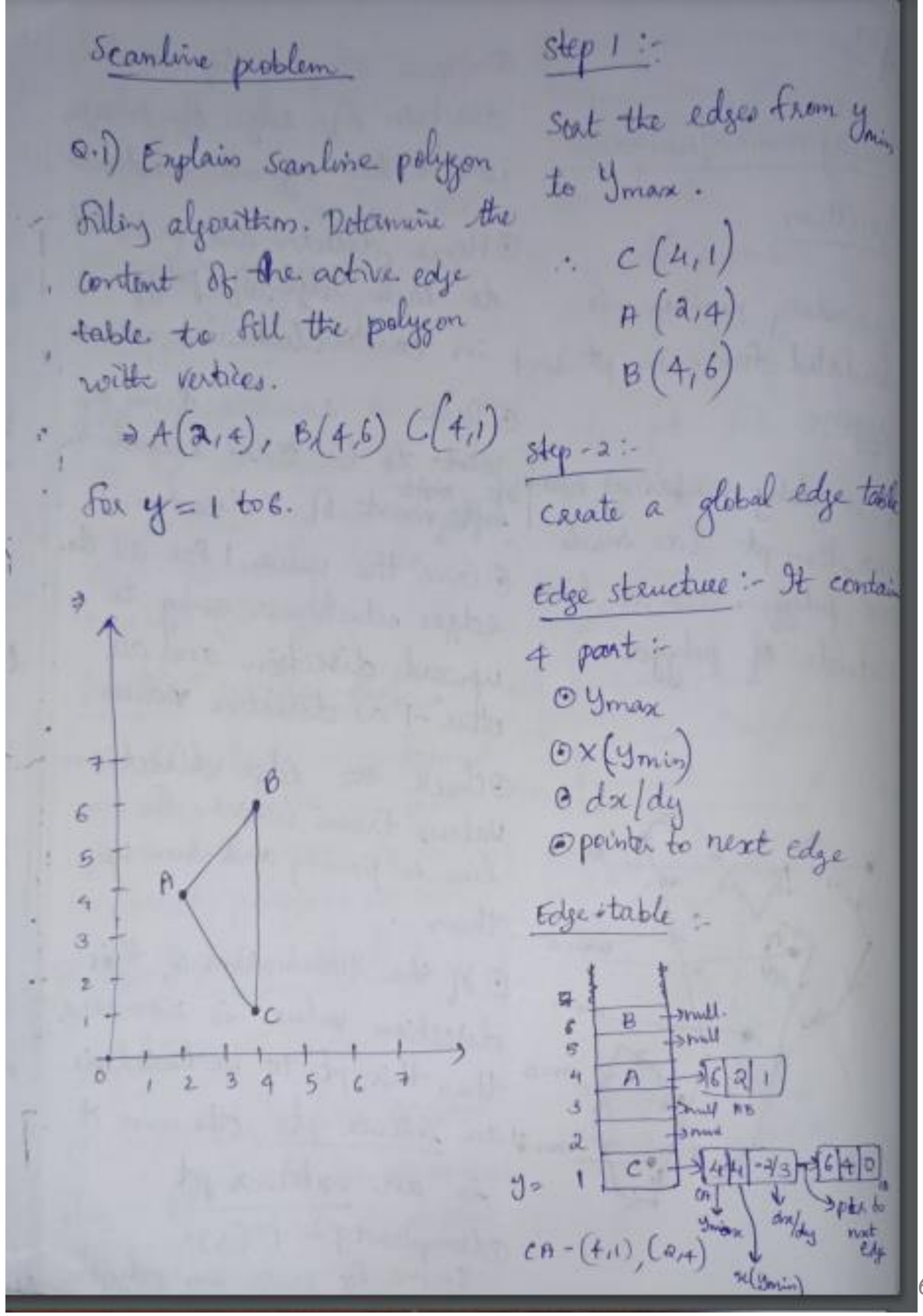
the x-intersection value $\mathbf{x_{k+1}}$ on the upper scan line can be determined from the x-intersection value $\mathbf{x_k}$ on the preceding scan line as

$$x_{k+1} = x_k + \frac{1}{m}$$

Each successive **x** intercept can thus be calculated by adding the inverse of the slope and rounding to the nearest integer.

## Problem

- Explain scan line polygon filling algorithm. Determine the content of the active edge table to fill the polygon with vertices A( 2,4) B(4,6) C(4,1) for y= 1 to 6.



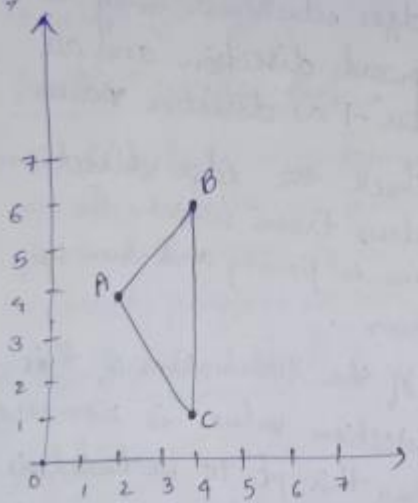
Scanline problem

Q.1) Explain Scanline polygon filling algorithm. Determine the content of the active edge table to fill the polygon with vertices.

⇒ A(2,4), B(4,6) C(4,1)

For y = 1 to 6.

⇒



Step 1 :-

Sort the edges from $y_{min}$ to $y_{max}$.

∴ C (4,1)
A (2,4)
B (4,6)

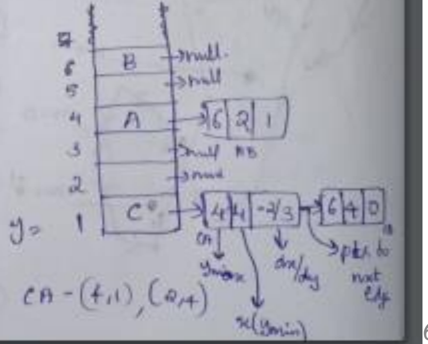
Step - 2 :-

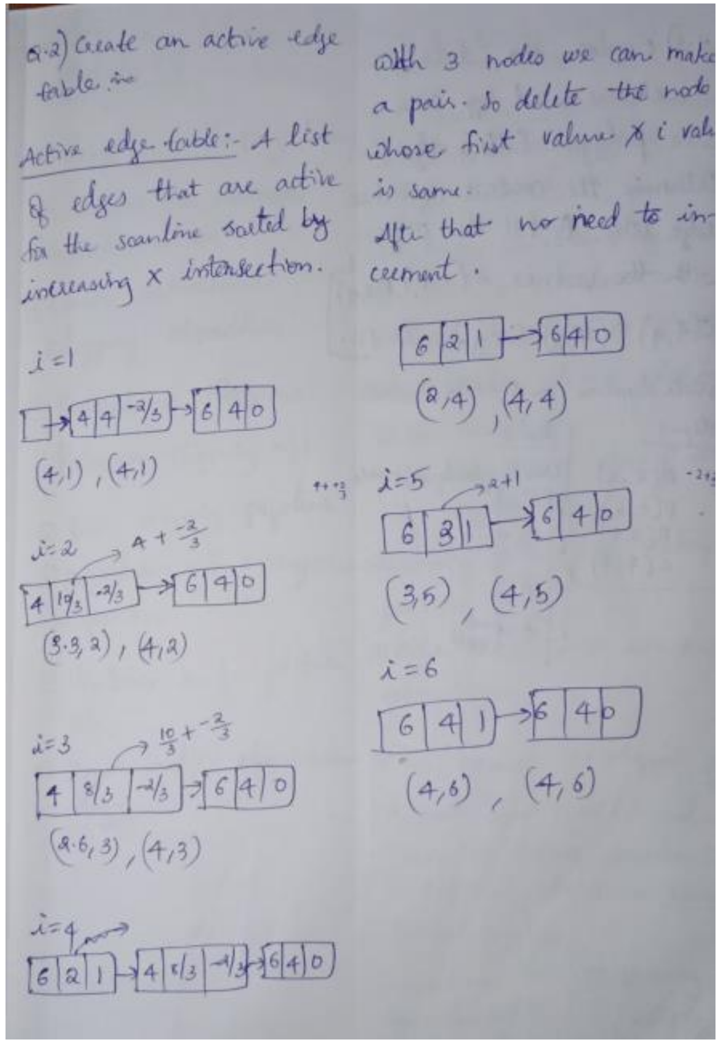
Create a global edge table

Edge structure :- It contain 4 part :-

- $y_{max}$
- $x(y_{min})$
- $dx/dy$
- pointer to next edge

Edge table :-





Q.2) Create an active edge table.

Active edge table:- A list of edges that are active for the scanline sorted by increasing x intersection.

$i=1$

$\boxed{\phantom{0}} \rightarrow \boxed{4 \mid 4 \mid -2/3} \rightarrow \boxed{6 \mid 4 \mid 0}$

(4,1), (4,1)

$i=2$ $\quad 4 + \frac{-2}{3}$

$\boxed{4 \mid 10/3 \mid -2/3} \rightarrow \boxed{6 \mid 4 \mid 0}$

(3.3, 2), (4,2)

$i=3$ $\quad \frac{10}{3} + \frac{-2}{3}$

$\boxed{4 \mid 8/3 \mid -2/3} \rightarrow \boxed{6 \mid 4 \mid 0}$

(2.6, 3), (4,3)

$i=4$

$\boxed{6 \mid 2 \mid 1} \rightarrow \boxed{4 \mid 6/3 \mid -2/3} \rightarrow \boxed{6 \mid 4 \mid 0}$

With 3 nodes we can make a pair. So delete the node whose first value & i value is same.
After that no need to increment.

$\boxed{6 \mid 2 \mid 1} \rightarrow \boxed{6 \mid 4 \mid 0}$

(2,4), (4,4)

$i=5$ $\quad 2+1$

$\boxed{6 \mid 3 \mid 1} \rightarrow \boxed{6 \mid 4 \mid 0}$

(3,5), (4,5)

$i=6$

$\boxed{6 \mid 4 \mid 1} \rightarrow \boxed{6 \mid 4 \mid 0}$

(4,6), (4,6)

| i | First point | Second point |
|---|---|---|
| 1 | (4,1) | (4,1) |
| 2 | (3.3,2) | (4,2) |
| 3 | (2.6,3) | (4,3) |
| 4 | (2,4) | (4,4) |
| 5 | (3,5) | (4,5) |
| 6 | (4,6) | (4,6) |

For each scan line from i= 1 to 6 fill color in between each pair of intersections.

## Inside Outside test

- In Computer Graphics, Inside Outside is performed to test whether a given point lies inside of a closed polygon or not.
- Mainly, there are **two methods** to determine a point is interior/exterior to polygon:

1. Even-Odd / Odd-Even Rule or Odd Parity Rule
2. Winding Number Method

## Even-Odd Rule / Odd Parity Rule

- It is also known as crossing number and ray casting algorithm.
- The algorithm follows a basic observation that if a ray coming from infinity crosses through border of polygon, then it goes from outside to inside and inside to outside alternately.
- For every two crossings, point lies outside of polygon.

## Algorithm:

- Construct a line segment from point to be examined to point outside of a polygon.
- Count the number of intersections of line segment with polygon boundaries.
- If Odd number of intersection, then Point lies inside of Polygon.
- Else, Point lies outside of polygon.

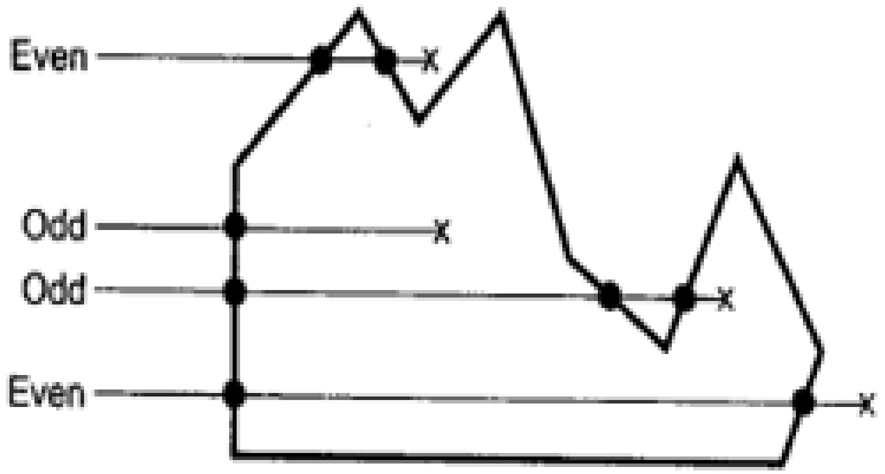


Fig. (a)

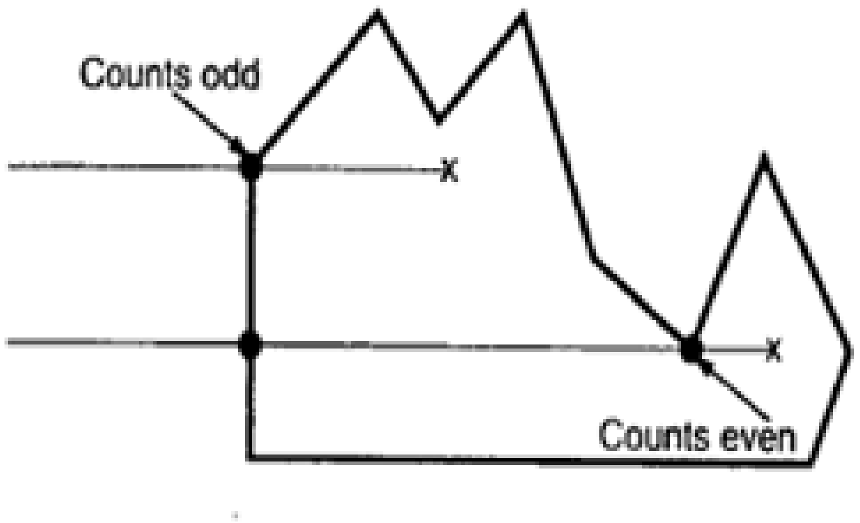


Fig. (b)

- This test fails in case line segment intersects at vertex point. To handle it, few modifications are made. Look at other end points of two line segments of polygon.
- If end points lie  at same side of constructed line segment, then even number of intersection is considered for that intersection point.
- If end points lie at opposite side of it, then odd number of intersection is considered.
- **Time Complexity:**
- O(S) where S is the number of sides in the polygon

## Winding Number / Non-Zero Algorithm

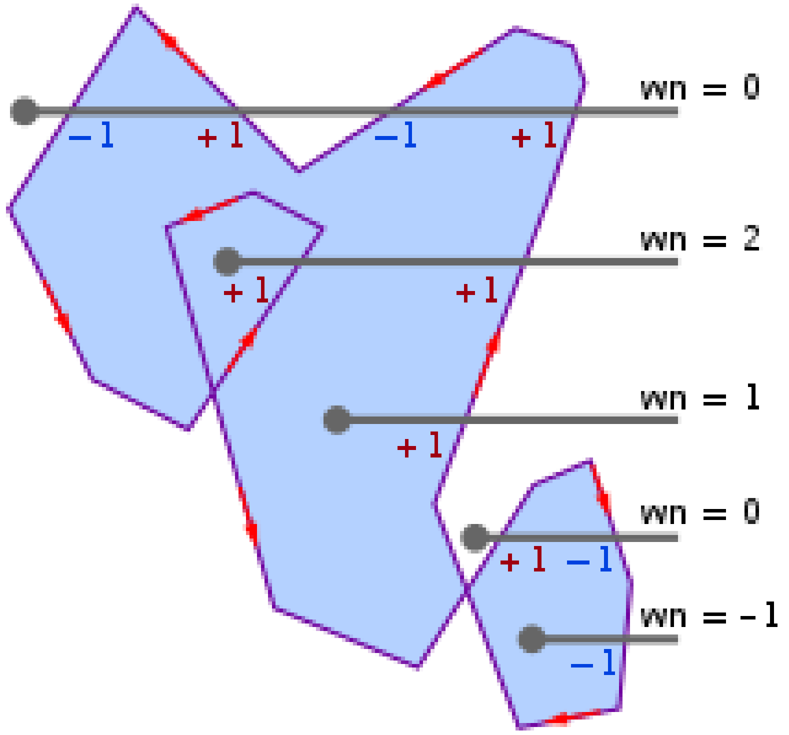
- Alternative algorithm to perform test is Winding Number algorithm.
- A winding Number is calculated for given point with respect to polygon.
- If winding number is non-zero, then point lies inside the polygon. Else, it lies outside of polygon.

## Calculation of Winding Number

- Conceptually, to check a point P, construct a line segment starting from P to point on boundary.
- Treat line segment to be elastic pinned at P. Stretch other end of elastic around the polygon for one complete cycle.
- Check how many times elastic has been wounded around point P.
- If count is non-zero, then point lies inside of polygon. Else, outside of polygon.

- Another way to score up winding number is to assign a score for each intersection with boundary of polygon and sum these numbers.
- The **score** is given by considering direction of edge of polygon with respect to line segment constructed.
- Hence, directions are assigned to each edge of polygon in **counter-clock** manner.
- If side of edge starts from below of constructed line, then **score -1** is given.
- If edge starts from above of constructed line then **score 1** is given.
- **Time Complexity:**
- O(S) where S is the number of sides in the polygon.

## Example



- **Example**
  For a given figure,

**1.** For a top-most point, Winding number = (-1) + (1) + (-1) + (1) = 0 , lies outside.

**2.** For bottom-most point, Winding number = -1 , lies inside.

## Two dimensional transformations

- The geometrical changes of an object from a current state to modified state.
- Changes in orientation, size and shape are accomplished with geometric transformations that alter the coordinate descriptions of an object.

**<u>2 ways</u>**

**Object Transformation**

- Alter the coordinates descriptions an object
- Translation, rotation, scaling etc.
- Coordinate system unchanged

**Coordinate transformation**

- Produce a different coordinate system

**<u>The basic geometric transformations are,</u>**

- Translation
- Rotation
- Scaling

**<u>Other transformations</u>**

- Reflection
- Shear

# Translation

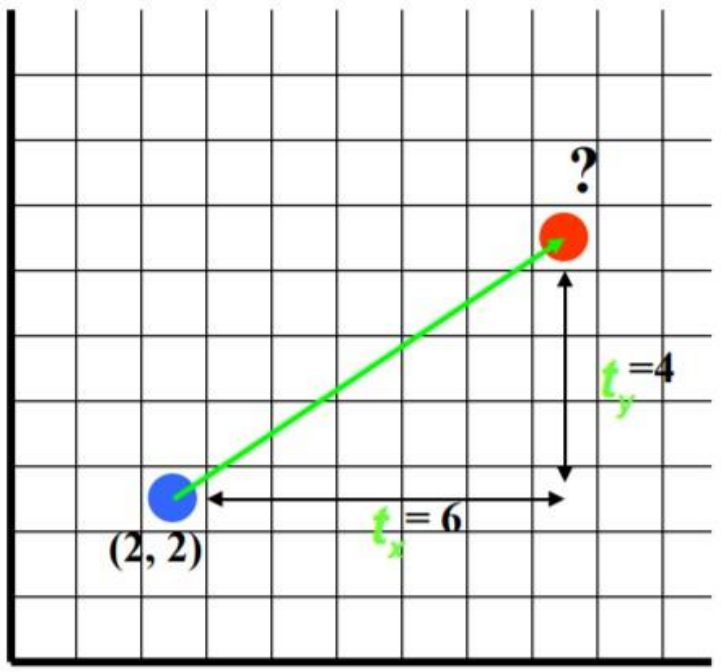
- A translation moves all points in an object along the same straight-line path to new positions.
- The path is represented by a vector, called the translation or shift vector.
- We can write the components:

$$p'_x = p_x + t_x$$
$$p'_y = p_y + t_y$$

- or in matrix form:

$$P' = P + T$$

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



- We translate a 2D point by adding translation distance $t_x$ and $t_y$ to the original position (x,y) to move the point to a new position (x',y')

$\mathbf{x' = x + t_x}$

$\mathbf{y' = y + t_y}$

- Translation is a rigid body transformation that moves objects without deformation , that is every point on the object is translated by the same amount.

## Contd….

- Example1: Translate the given point (2,5) by translation vector (3,3)

Solution:

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

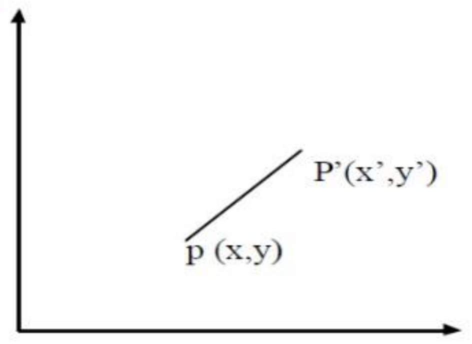
$T_x = 3$

$T_y = 3$

$X' = x + t_x$

$Y' = y + t_y$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 2 \\ 5 \end{bmatrix} + \begin{bmatrix} 3 \\ 3 \end{bmatrix} = \begin{bmatrix} 5 \\ 8 \end{bmatrix}$$



- Example2: Translate a polygon with coordinates A(2,5), B(7,10) and C(10,2) by 3 units in x direction and 4 unit in y direction.
- Solution:

$$A' = A + T = \begin{bmatrix} 2 \\ 5 \end{bmatrix} + \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 5 \\ 9 \end{bmatrix}$$

$$B' = B + T = \begin{bmatrix} 7 \\ 10 \end{bmatrix} + \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 10 \\ 14 \end{bmatrix}$$

$$C' = C + T = \begin{bmatrix} 10 \\ 2 \end{bmatrix} + \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 13 \\ 6 \end{bmatrix}$$

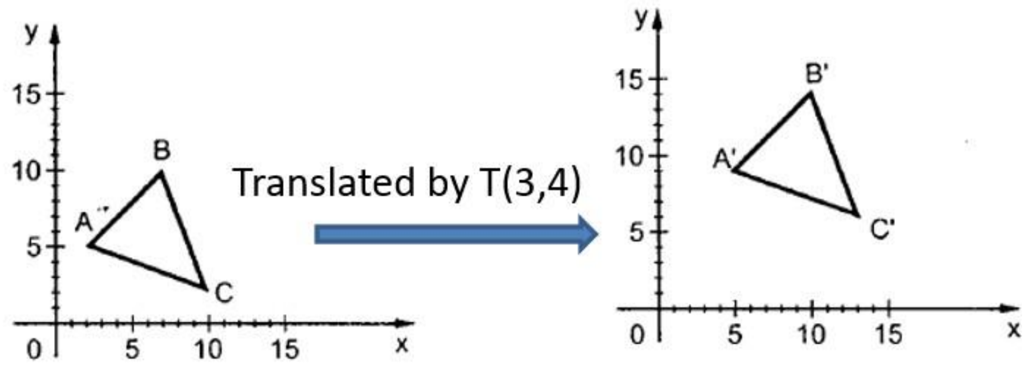


Fig: Before translation Fig: After translation

## Move an object with points (5,5) , (10,5) , (7.5,10) with a distance (10,10)

P' = P+T

P = [ (5,5) , (10,5) , (7.5,10)]  , first draw the object before translation

T=(10,10)

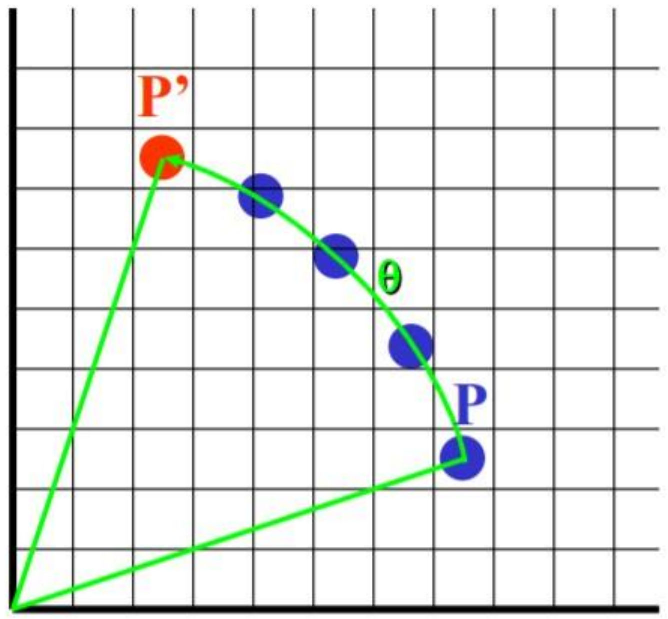
So calculate P' for all points

P'= [(15,15), (20,15), (17.5,20)]

Draw the translated object

## Rotation

- A rotation repositions all points in an object along a circular path in the plane centered at the pivot point.
- First, we'll assume the pivot is at the origin.



## Rotation

- Review Trigonometry

=> $\cos\phi = x/r$ , $\sin\phi = y/r$

- $x = r.\cos\phi$, $y = r.\sin\phi$

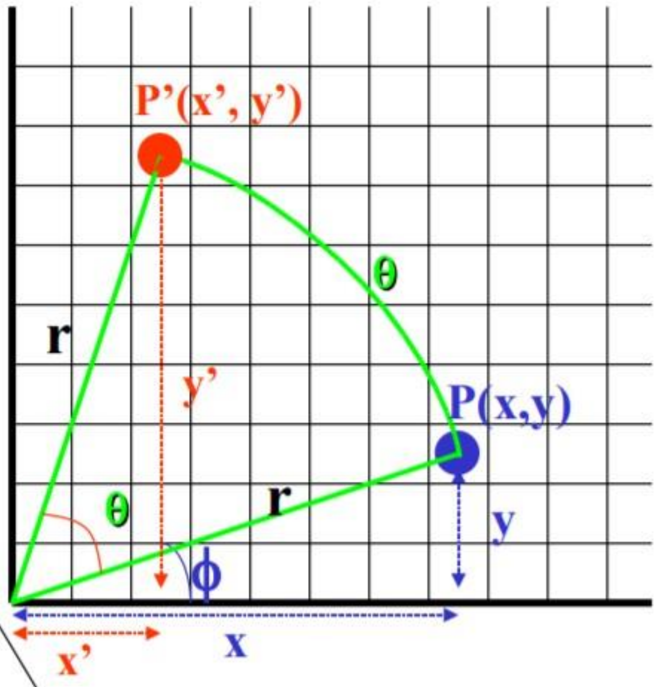
=> $\cos(\phi + \theta) = x'/r$

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

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

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

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



Identity of Trigonometry

## Rotation

- We can write the components:

$$p'_x = p_x \cos\theta - p_y \sin\theta$$
$$p'_y = p_x \sin\theta + p_y \cos\theta$$

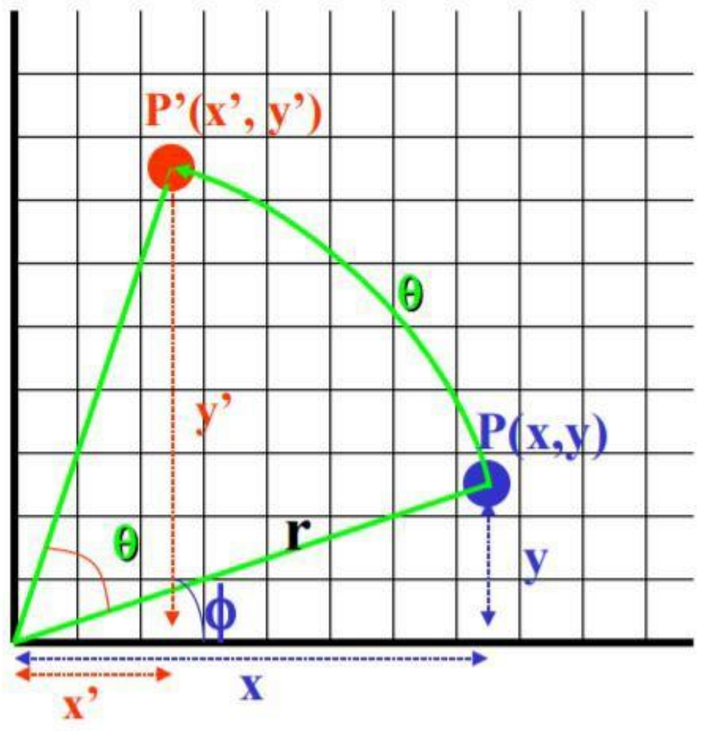
- or in matrix form:

$$P' = R \cdot P$$

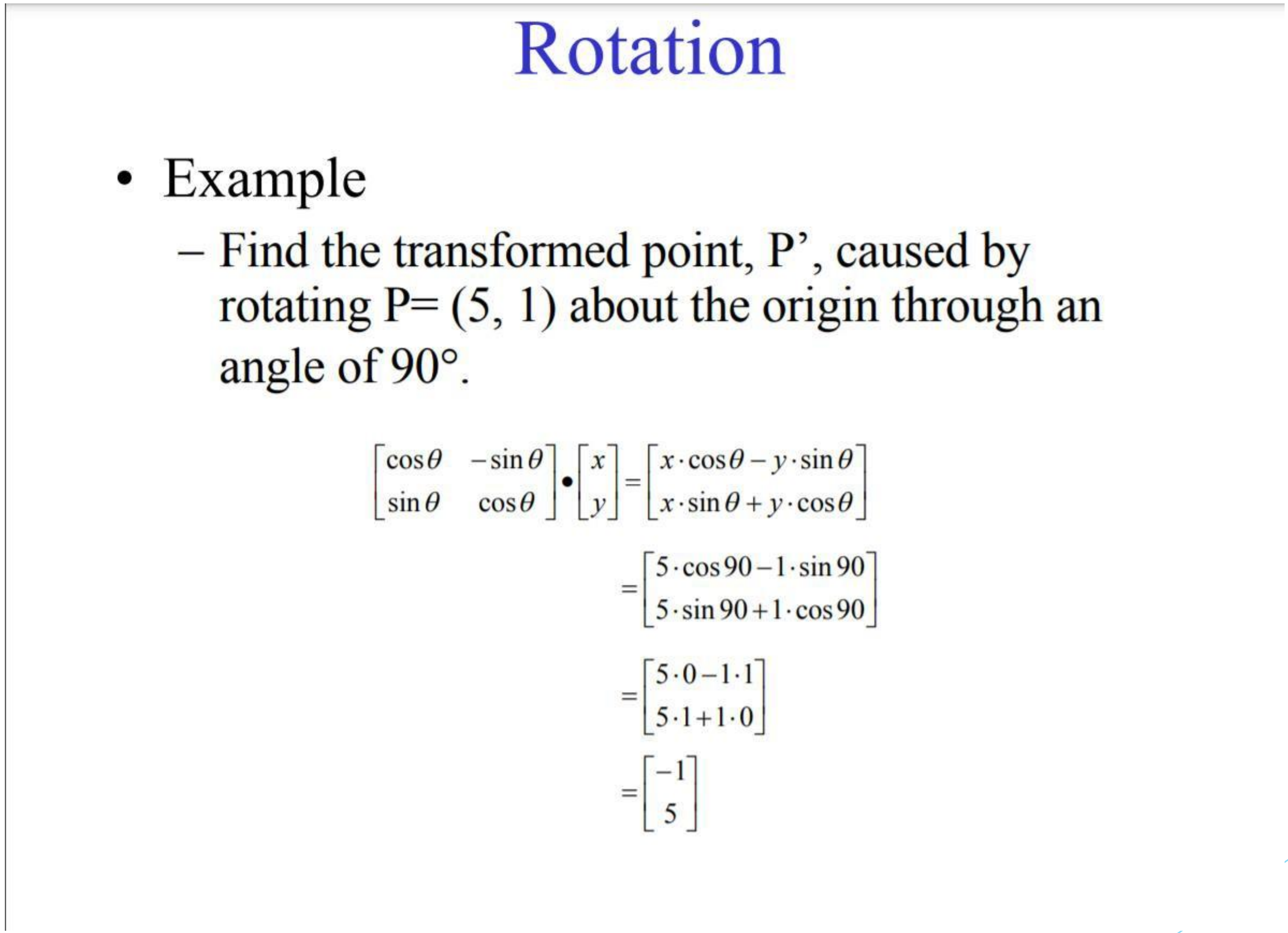
∀ θ can be clockwise (-ve) or counterclockwise (+ve as our example).

- Rotation matrix

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



- Clock wise ($\theta$ become negative)
- So R= $\begin{matrix} \cos(-\boldsymbol{\theta}) & -\sin(-\boldsymbol{\theta}) \\ \sin(-\boldsymbol{\theta}) & \cos(-\boldsymbol{\theta}) \end{matrix}$
- $\cos(-\boldsymbol{\theta}) = \cos \boldsymbol{\theta}$
- $\sin(-\boldsymbol{\theta}) = -\sin(\boldsymbol{\theta})$
- P'= R.P
- $= \begin{matrix} \cos(\boldsymbol{\theta}) & \sin(\boldsymbol{\theta}) \\ -\sin(\boldsymbol{\theta}) & \cos(\boldsymbol{\theta}) \end{matrix} . \begin{matrix} X \\ Y \end{matrix}$
- X' $= X\cos(\boldsymbol{\theta}) + Y \sin(\boldsymbol{\theta})$
- Y'$= -X\sin(\boldsymbol{\theta}) + Y\cos(\boldsymbol{\theta})$



## Rotation

- Example
  - Find the transformed point, P', caused by rotating P= (5, 1) about the origin through an angle of 90°.

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

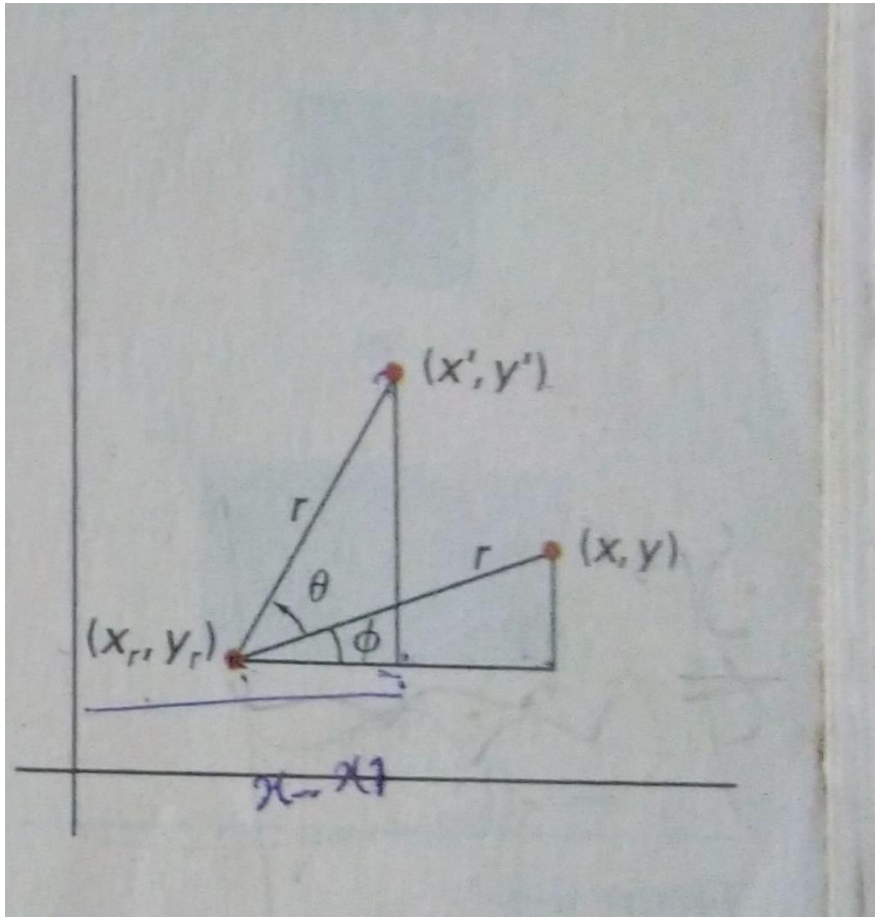
$$=\begin{bmatrix}5\cdot\cos 90-1\cdot\sin 90\\ 5\cdot\sin 90+1\cdot\cos 90\end{bmatrix}$$

$$=\begin{bmatrix}5\cdot 0-1\cdot 1\\ 5\cdot 1+1\cdot 0\end{bmatrix}$$

$$=\begin{bmatrix}-1\\ 5\end{bmatrix}$$

- **Homework:**
  - **Example2:** Rotate the triangle ABC having coordinates A(1,2), B(2,3) and C(4,5) by $60^0$ about the origin.
  - **Example3:** A point (4,3) is rotated clockwise by an angle -45 degree. Find the rotation matrix and the resultant point.
  - **Example4:** Rotate the triangle with vertices A(5,8), B(12,10) and C(10,10) about origin with angle 90 degree in anticlockwise direction.

## Rotation about a fixed point ($x_r$ , $y_r$)



$(x' - x_r) = \cos(\phi + \boldsymbol{\theta}).r$

$= \mathbf{r\cos}\ \phi\cos\ \boldsymbol{\theta} \mathbf{-\ r\sin}\phi\sin\ \boldsymbol{\theta}$

$= \mathbf{x}\cos\ \boldsymbol{\theta}\ \mathbf{-y}\sin\ \boldsymbol{\theta}$

$= (\mathbf{x} - x_r)\cos\ \boldsymbol{\theta} - (\mathbf{y} - y_r)\sin\ \boldsymbol{\theta}$ where $y = \mathbf{y} - y_r$ ; $x = \mathbf{x} - x_r$

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

***Find* y'** **?????**

**Rotations are rigid body transformations that move objects without deformations.**

## Scaling

- Scaling changes the size of an object and involves two scale factors, $S_x$ and $S_y$ for the x- and y- coordinates respectively.
- Scales are about the origin.
- We can write the components:

$$p'_x = s_x \bullet p_x$$

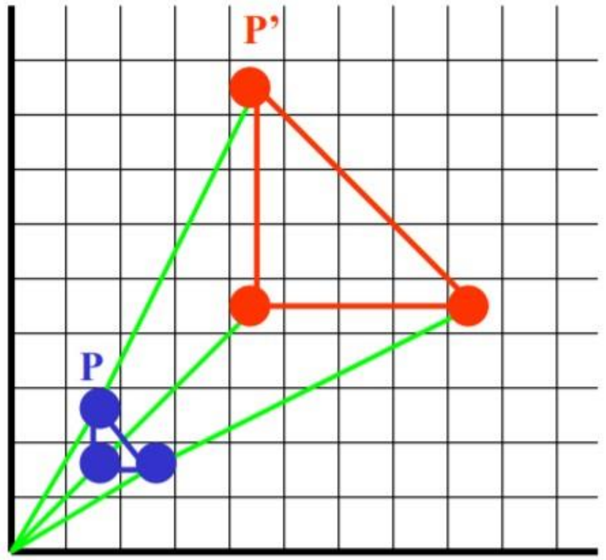
$$p'_y = s_y \bullet p_y$$

or in matrix form:

$$P' = S \bullet P$$

Scale matrix as:

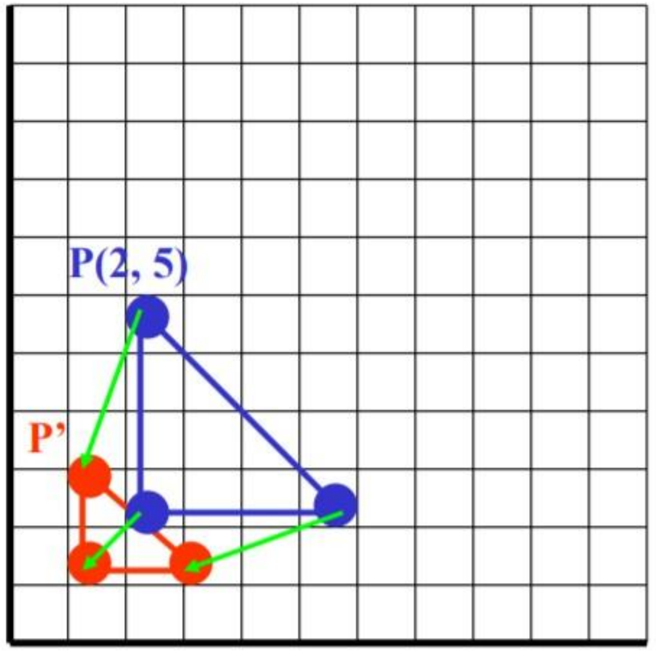
$$S = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$$



## Scaling

- If the scale factors are in between 0 and 1 ➔ the points will be moved closer to the origin ➔ the object will be smaller.

- Example :
  - P(2, 5), Sx = 0.5, Sy = 0.5
  - Find P' ?



## Scaling

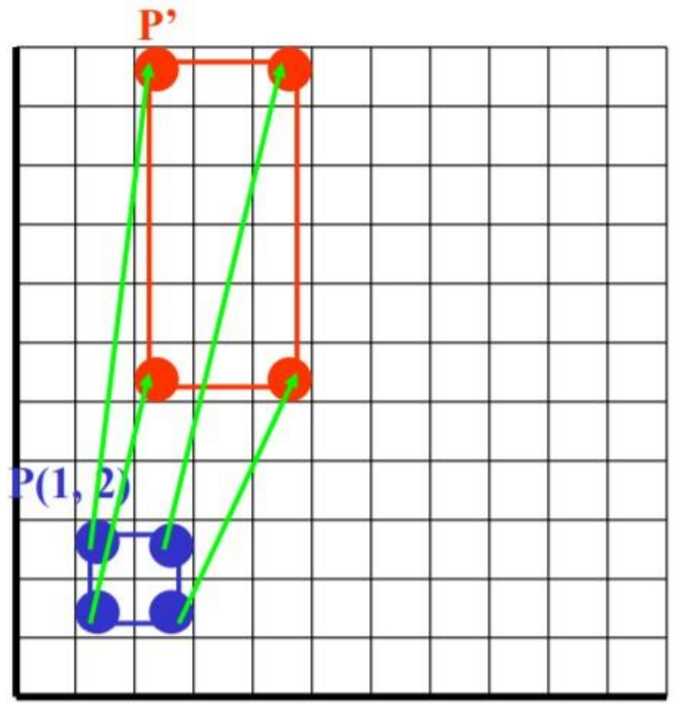
- If the scale factors are the same, $S_x = S_y$ ➔ uniform scaling
- Only change in size (as previous example)

- If $S_x \neq S_y$ ➔ differential scaling.
- Change in size and shape
- Example : square → rectangle
  - P(1, 3), $S_x = 2$, $S_y = 5$ , P' ?

What does scaling by 1 do?

What is that matrix called?

What does scaling by a negative value do?



## Problems

- A Square object with the coordinate points P (1, 4), Q (4, 4), R (4, 1), T (1,1). Apply the scaling factor 3 on the X-axis and 4 on the Y-axis. Find out the new coordinates of the square?
- Given a square object with coordinate points A(0, 3), B(3, 3), C(3, 0), D(0, 0). Apply the scaling parameter 2 towards X axis and 3 towards Y axis for the given point (4,6) and obtain the new coordinates of the object.
- A triangle ABC with coordinates A(0,0), B(6,5), C(6,0) is scaled with scaling factors Sx=2 and Sy=3 about the vertex C(6,0). Find the transformed coordinate points. 4M
- Perform a 45 degree rotation of a triangle ABC having the vertices at A(0,0) B(10,10) and C(50,20)
- i. About the origin
- ii. About an arbitrary point P(-10,-10) 6M

## Scaling with respect to a fixed point ($x_f$ , $y_f$ )

- A polygon is then scaled relative to the fixed point by scaling the distance from each vertex to the fixed point.
- For a vertex with coordinates (x,y) , the scaled coordinates (x', y') are calculated as

$$x' = x_f + (x - x_f).s_x$$

$$y' = y_f + (y - y_f).s_y$$

- Rewrite these scaling transformations to separate multiplicative and additive terms

$$\mathbf{x' = xs_x + x_f (1-s_x)}$$

$$\mathbf{y' = ys_y + y_f (1-s_y)}$$

- Where the additive terms $x_f (1-s_x)$ and $y_f (1-s_y)$ constants for all points in the object.

- Perform the following transformations on a point (6, 4).

  i) Translate by tx = −2 and ty = 4

  ii) then, Scale by sx = 2 and sy = 1

  iii) and Rotate by 90o in clockwise direction. Determine the final coordinates of the transformed point.

- Given a triangle A(20,10) B(80,20) C(50,70). Find the co-ordinates of vertices after each of the following transformation.

  Rotation of the triangle ABC about vertex A in clockwise direction for an angle 90 degree.

## Matrix representations and homogenous coordinates

- Many graphics applications involve sequences of geometric transformations.
- An animation, for example, might require an object to be translated and rotated at each increment of the motion.
- Here we reformulated the matrix representation of basic transformation . So that the transformation sequences can be efficiently processed.
- Each of the basic transformations can be expressed in the general matrix form

  **$P' = M_1 . P + M_2$**
- with coordinate positions P and P' represented as column vectors.
- Matrix **$M_1$** is a 2 by 2 array containing multiplicative factors, **$M_2$** and is a two-element column matrix containing translational terms.
- For translation, **$M_1$** is the identity matrix.
- For rotation or scaling, **$M_2$** contains the translational terms associated with the pivot point or scaling fixed point.

- To produce a sequence of transformations with these equations, such as scaling followed by rotation then translation, we must calculate the transformed coordinates one step at a time.
- First, coordinate positions are scaled, then these scaled coordinates are rotated, and finally the rotated coordinates are translated.
- We can combine the multiplicative and translational terms for 2D geometric transformations into a single matrix representation by expanding the 2 by 2 matrix representations to 3 by 3 matrices.
- This allows us to express all transformation equations as matrix multiplications.
- we represent each Cartesian coordinate position (x,y) with the homogeneous coordinate triple ($\mathbf{x_h}$, $\mathbf{y_h}$, h), where

$$\mathbf{x = x_h/h}$$

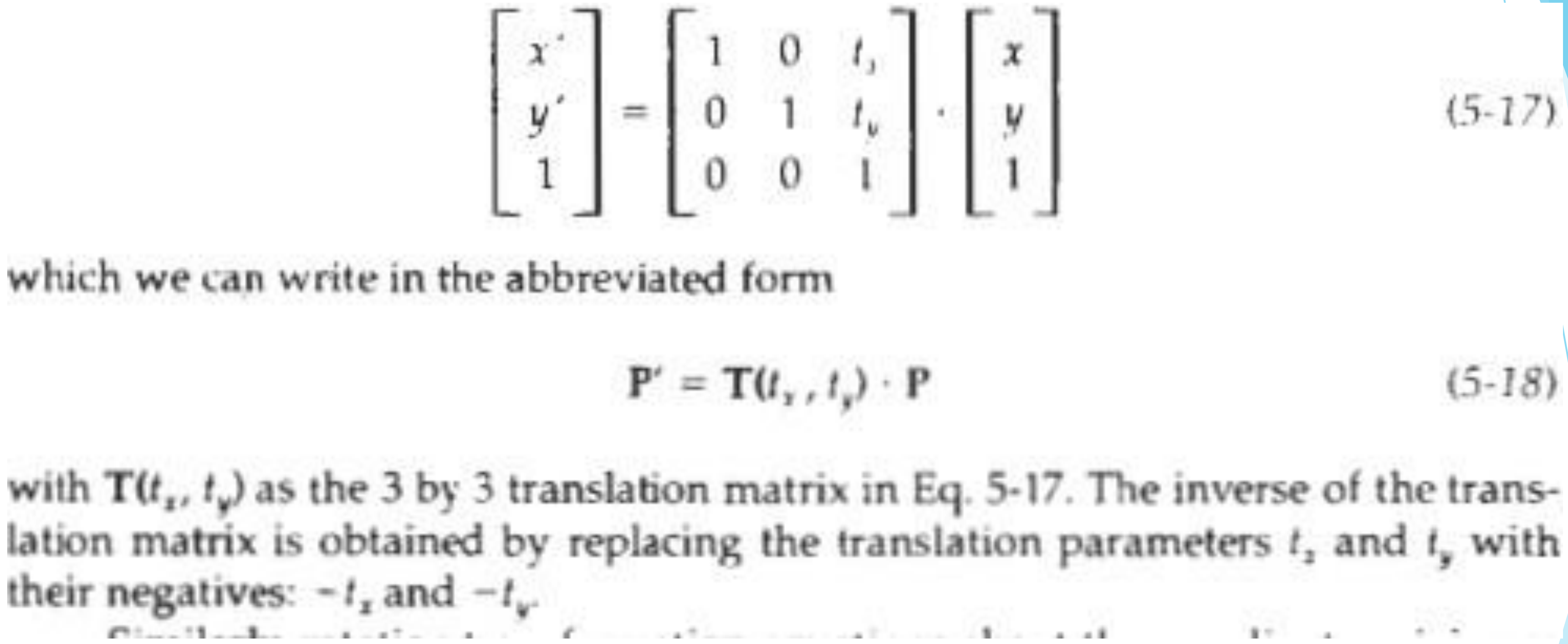
$$\mathbf{y = y_h/h}$$

- A general homogenous coordinate representation can also be written as (**x.h** , **y.h**, h).

- For two-dimensional geometric transformations, we can choose the homogenous parameter h to be any nonzero value.

- A convenient choice is simply to set h = 1.

- Each 2D position is then represented with homogeneous coordinates (x, y, 1).



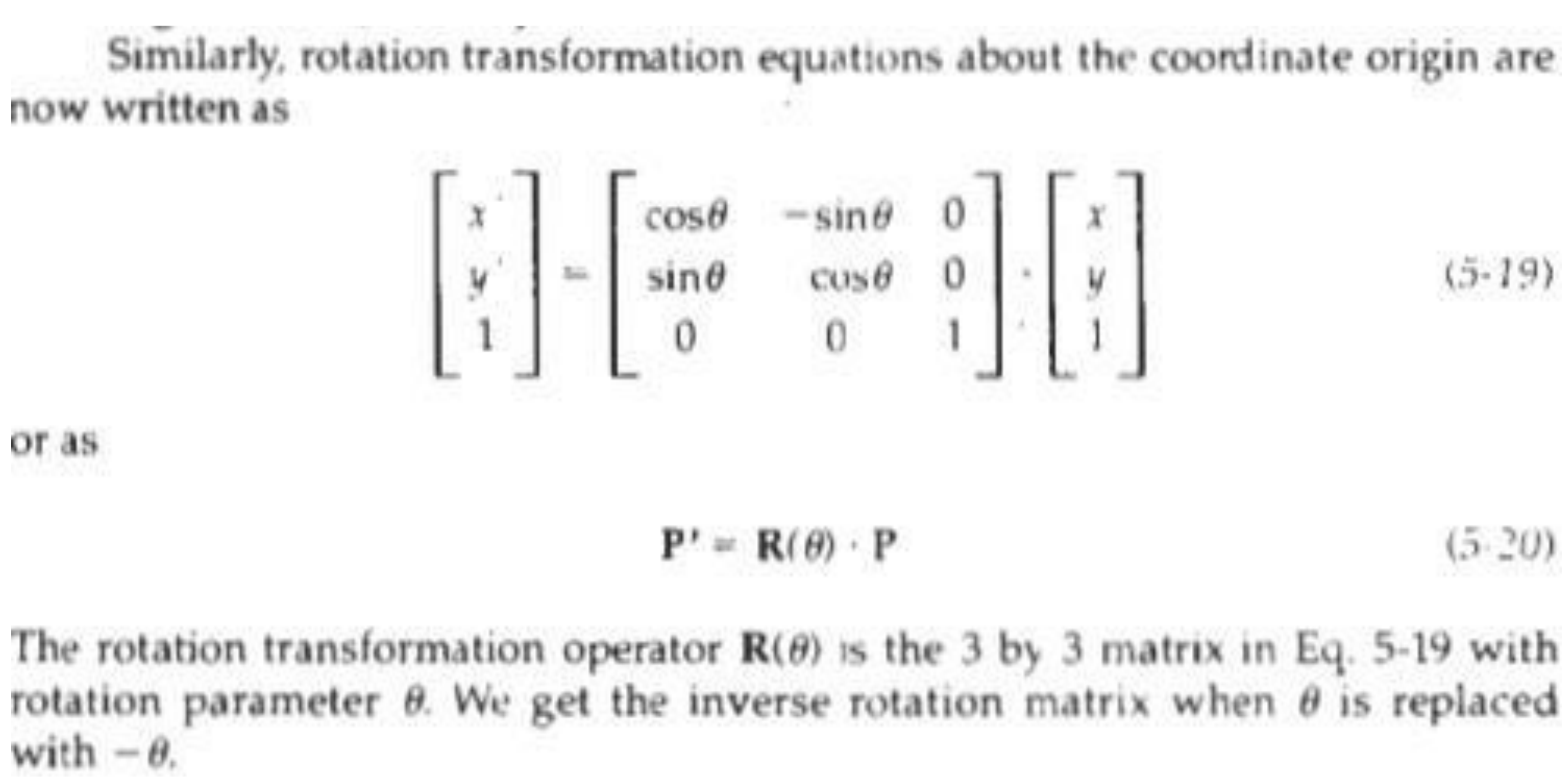
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{5-17}$$

which we can write in the abbreviated form

$$\mathbf{P}' = \mathbf{T}(t_x, t_y) \cdot \mathbf{P} \tag{5-18}$$

with $\mathbf{T}(t_x, t_y)$ as the 3 by 3 translation matrix in Eq. 5-17. The inverse of the translation matrix is obtained by replacing the translation parameters $t_x$ and $t_y$ with their negatives: $-t_x$ and $-t_y$.

[illegible]



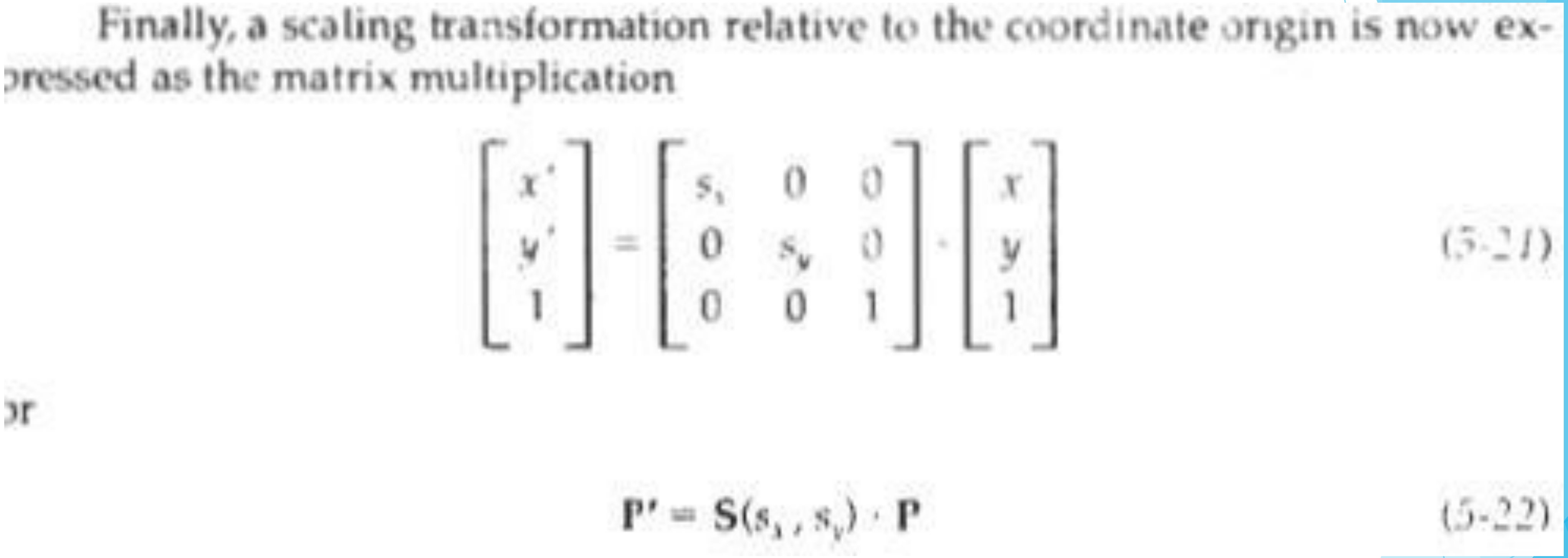
Similarly, rotation transformation equations about the coordinate origin are now written as

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{5-19}$$

or as

$$\mathbf{P}' = \mathbf{R}(\theta) \cdot \mathbf{P} \tag{5-20}$$

The rotation transformation operator $\mathbf{R}(\theta)$ is the 3 by 3 matrix in Eq. 5-19 with rotation parameter $\theta$. We get the inverse rotation matrix when $\theta$ is replaced with $-\theta$.



Finally, a scaling transformation relative to the coordinate origin is now ex-
pressed as the matrix multiplication

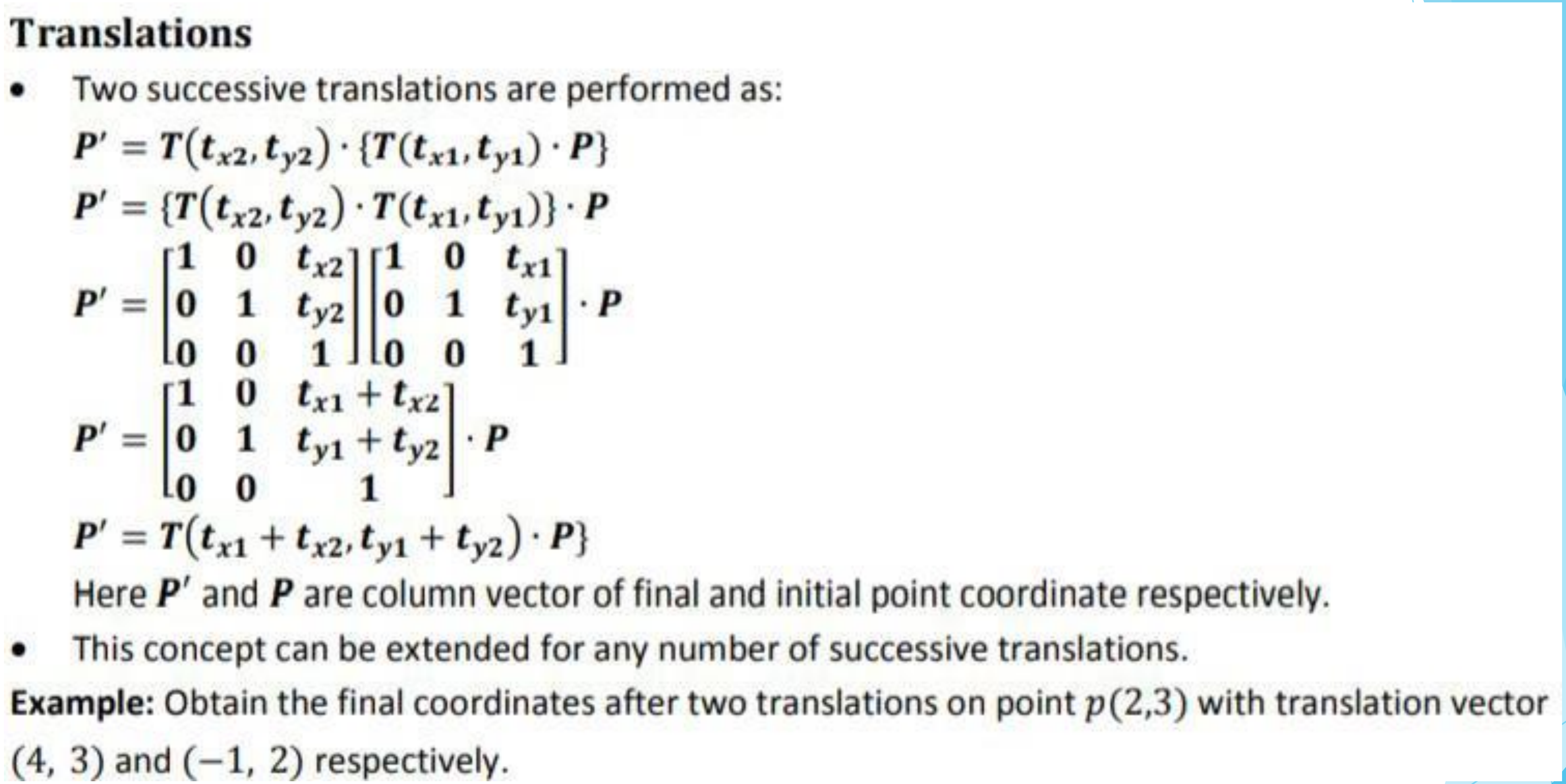
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad (5\text{-}21)$$

or

$$\mathbf{P}' = \mathbf{S}(s_x, s_y) \cdot \mathbf{P} \qquad (5\text{-}22)$$

## Composite transformation

- we can set up a matrix for any sequence of transformations as a composite transformation matrix by calculating the matrix product of the individual transformations.
- Forming products of transformation matrices is often referred to as a **concatenation, or composition, of matrices.**
- For column matrix representation of coordinate positions, we form composite transformations by multiplying matrices in order from right to left.
- That is, each successive transformation matrix premultiplies the product of the preceding transformation matrices.



## Translations

- Two successive translations are performed as:

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

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

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

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

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

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

- This 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.

## Rotations

- Two successive Rotations are performed as:

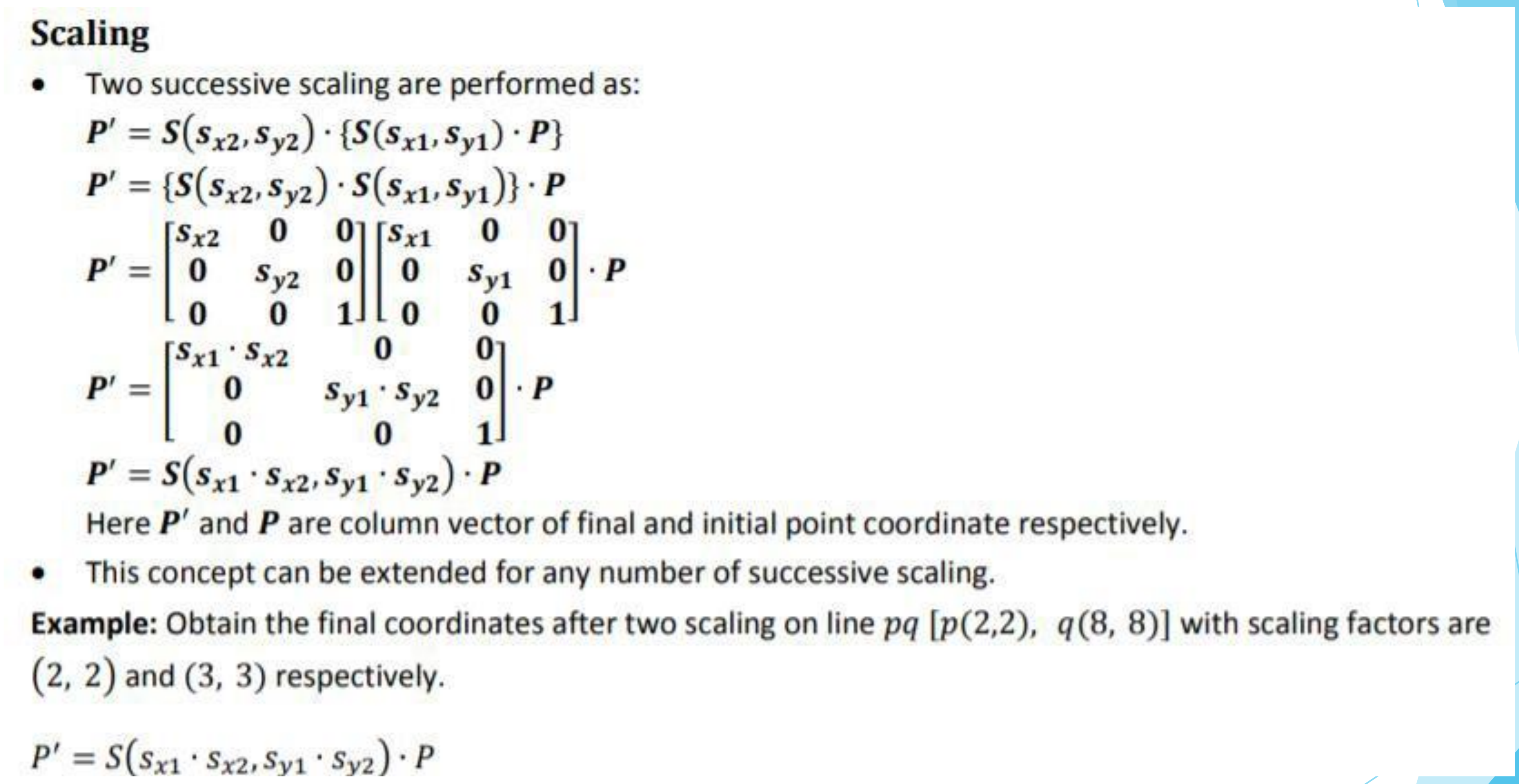
$$P' = R(\theta_2) \cdot \{R(\theta_1) \cdot P\}$$
$$P' = \{R(\theta_2) \cdot R(\theta_1)\} \cdot P$$
$$P' = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 \\ \sin\theta_2 & \cos\theta_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 \\ \sin\theta_1 & \cos\theta_1 &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

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

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.

**Example:** Obtain the final coordinates after two rotations on point $p(6,9)$ with rotation angles are $30^o$ and $60^o$ respectively.



## Scaling

- Two successive scaling are performed as:

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

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

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

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

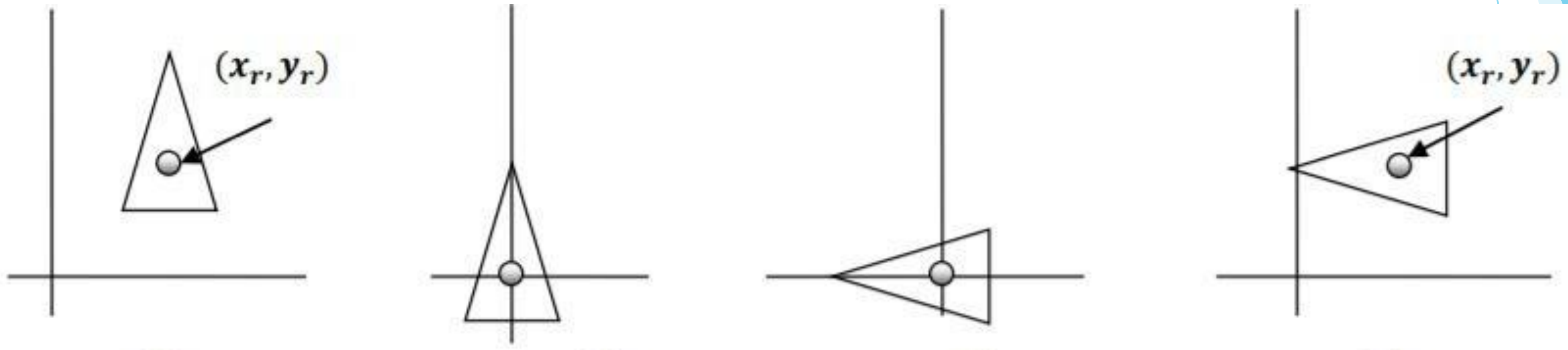
  $$\boldsymbol{P'} = \boldsymbol{S}(\boldsymbol{s_{x1}} \cdot \boldsymbol{s_{x2}}, \boldsymbol{s_{y1}} \cdot \boldsymbol{s_{y2}}) \cdot \boldsymbol{P}$$

  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.

**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.

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

# General pivot point rotation



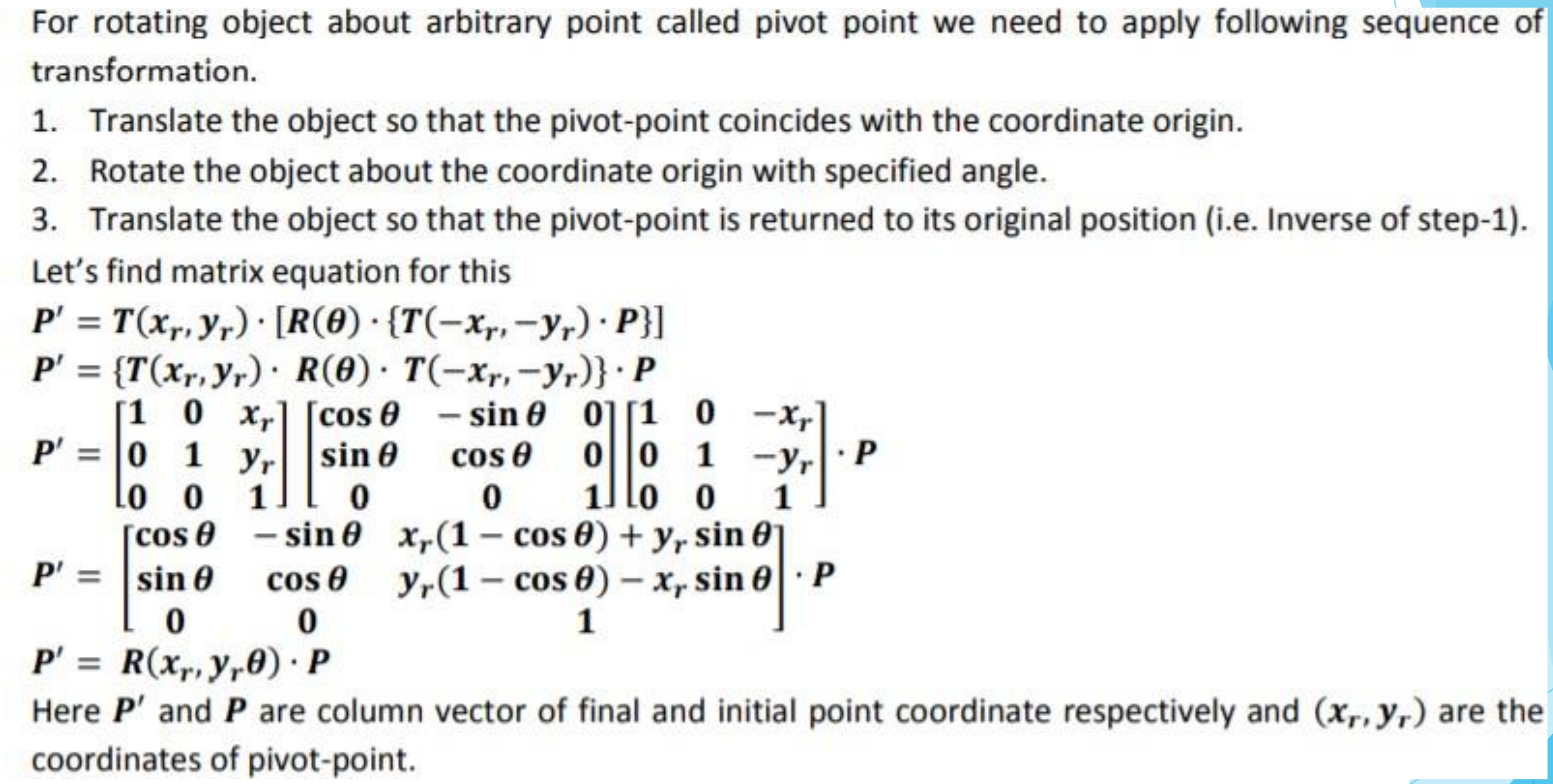
(a) Original Position of Object and Pivot Point.

(b) Translation of Object so that Pivot Point $(x_r, y_r)$ is at Origin.

(c) Rotation about Origin.

(d) Translation of Object so that Pivot Point is Return to Position $(x_r, y_r)$ .

Fig. 3.6: - 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).
- Let's find matrix equation for this

$$\boldsymbol{P'} = \boldsymbol{T}(\boldsymbol{x_r}, \boldsymbol{y_r}) \cdot [\boldsymbol{R}(\boldsymbol{\theta}) \cdot \{\boldsymbol{T}(-\boldsymbol{x_r}, -\boldsymbol{y_r}) \cdot \boldsymbol{P}\}]$$

$$\boldsymbol{P'} = \{\boldsymbol{T}(\boldsymbol{x_r}, \boldsymbol{y_r}) \cdot \boldsymbol{R}(\boldsymbol{\theta}) \cdot \boldsymbol{T}(-\boldsymbol{x_r}, -\boldsymbol{y_r})\} \cdot \boldsymbol{P}$$

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

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

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

Here $\boldsymbol{P'}$ and $\boldsymbol{P}$ are column vector of final and initial point coordinate respectively and $(\boldsymbol{x_r}, \boldsymbol{y_r})$ 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° clockwise about the centroid.

  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 $\theta = -90^\circ$.

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

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

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

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

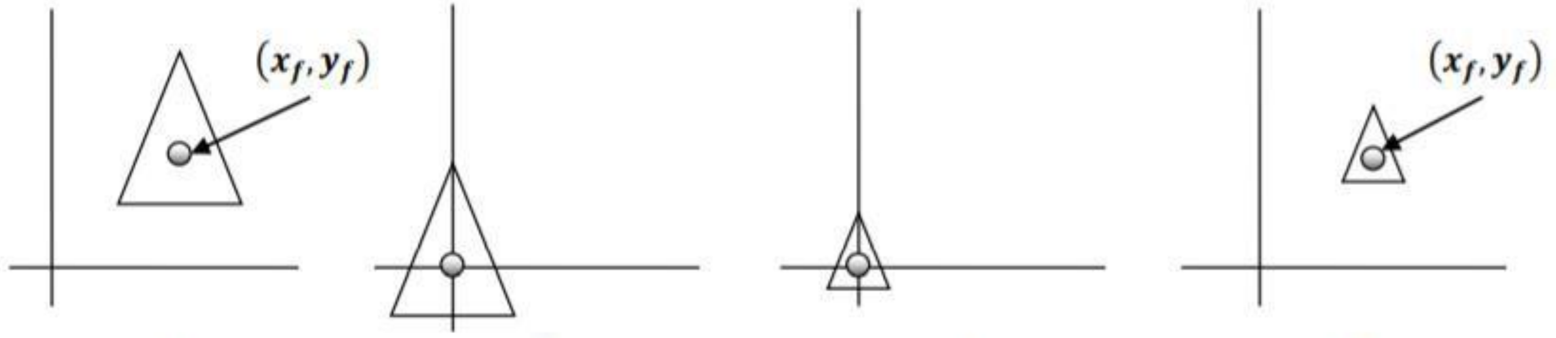
  $$P' = \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 12 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 5 & 8 & 8 \\ 4 & 3 & 8 \\ 1 & 1 & 1 \end{bmatrix}$$

  $$P' = \begin{bmatrix} 11 & 13 & 18 \\ 7 & 4 & 4 \\ 1 & 1 & 1 \end{bmatrix}$$

- Final coordinates after rotation are [A′ (11, 7), B′ (13, 4), C′ (18, 4)].

# General fixed point scaling

**General Fixed-Point Scaling**



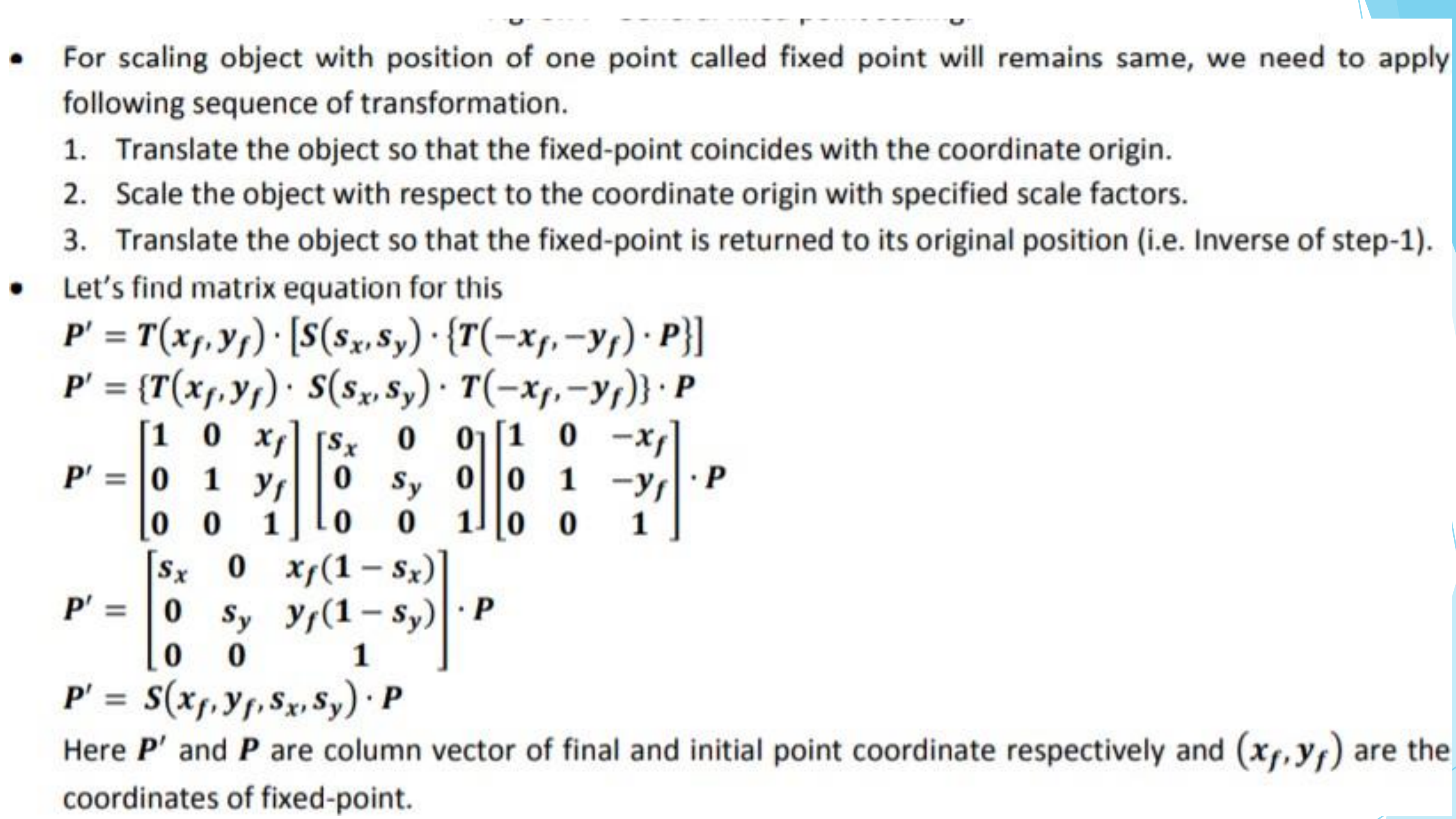
(a)
Original Position of Object and Fixed Point

(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 $(x_f, y_f)$ .

Fig. 3.7: - 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).
- Let's find matrix equation for this

$$P' = T(x_f, y_f) \cdot [S(s_x, s_y) \cdot \{T(-x_f, -y_f) \cdot P\}]$$

$$P' = \{T(x_f, y_f) \cdot S(s_x, s_y) \cdot T(-x_f, -y_f)\} \cdot P$$

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

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

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

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

- **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:

  $$x_f = 2 + \frac{6-2}{2}, \quad y_f = 2 + \frac{6-2}{2}$$

  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)].

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

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

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

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

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

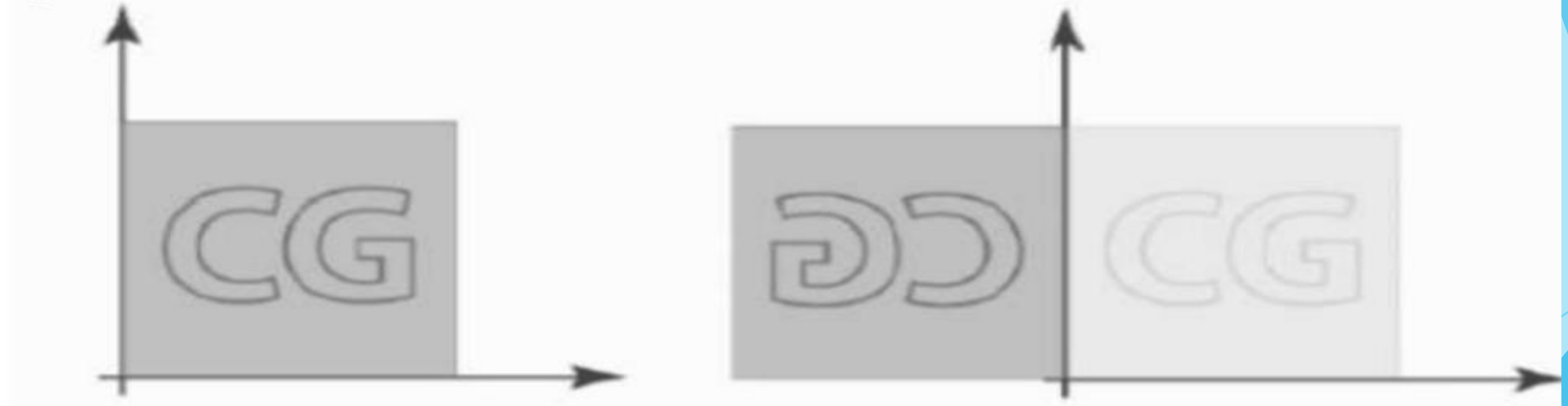
- Final coordinate after scaling are [A′ (3, 3), B′ (5, 3), C′ (5, 5), D′ (3, 5)]

# Reflection transformation

- Definition –

  Reflection is the mirror image of original object. In other words, we can say that it is a rotation operation with 180°.

  In reflection transformation, the size of the object does not change.



  Basically the mirror image of any image for 2D reflection is generated with respect to the "Axis of Reflection". For that we need to rotate main object 180 Degrees about the reflection axis.

## Reflection

- **Reflection with respect to the axis**
  - x

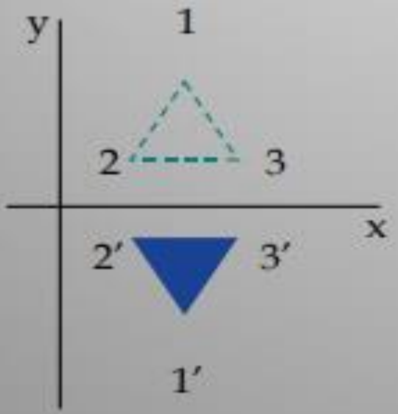
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

  - y

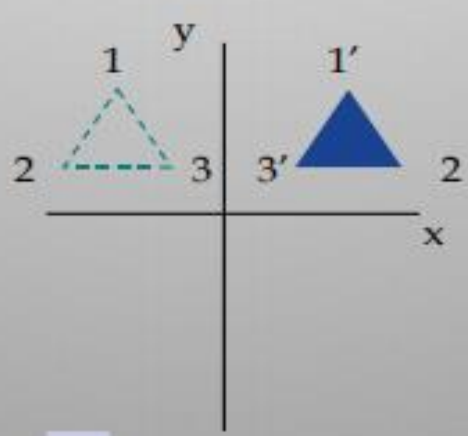
$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

  - xy

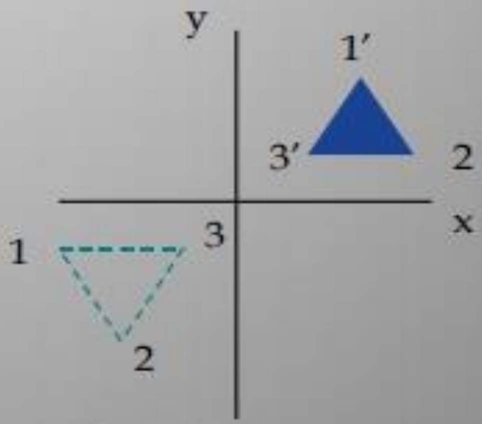
$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



*x*



*y*

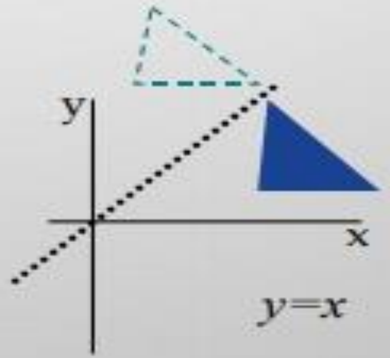


*xy*

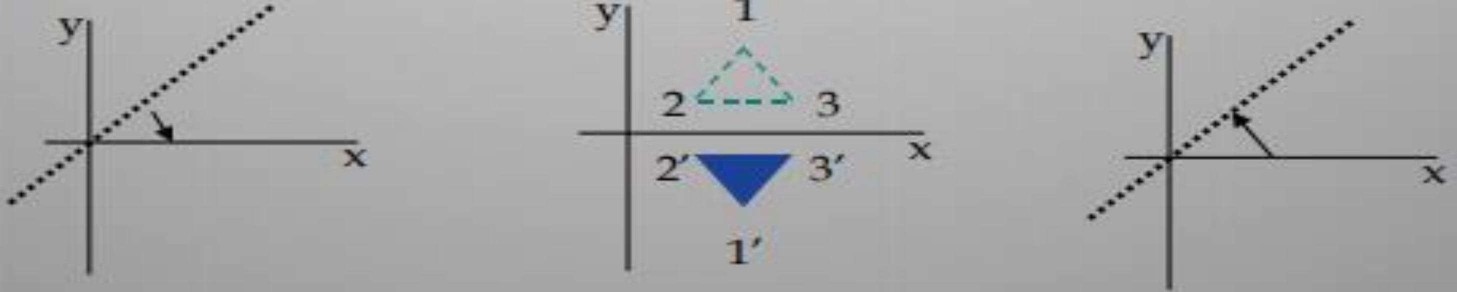
## Reflection

- Reflection with respect to a Line

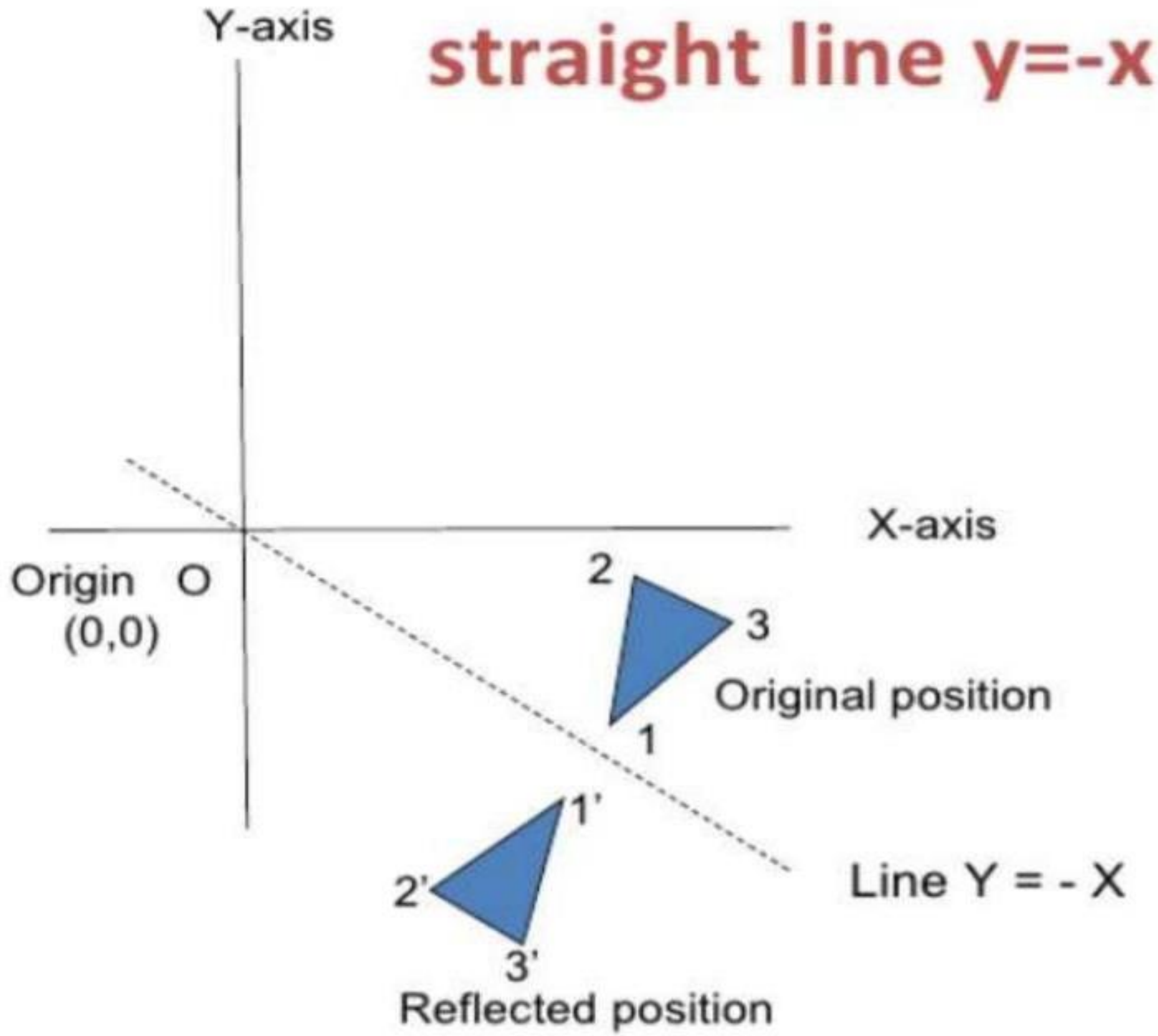
$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



- Clockwise rotation of 45° → Reflection about the x axis → Counterclockwise rotation of 45°

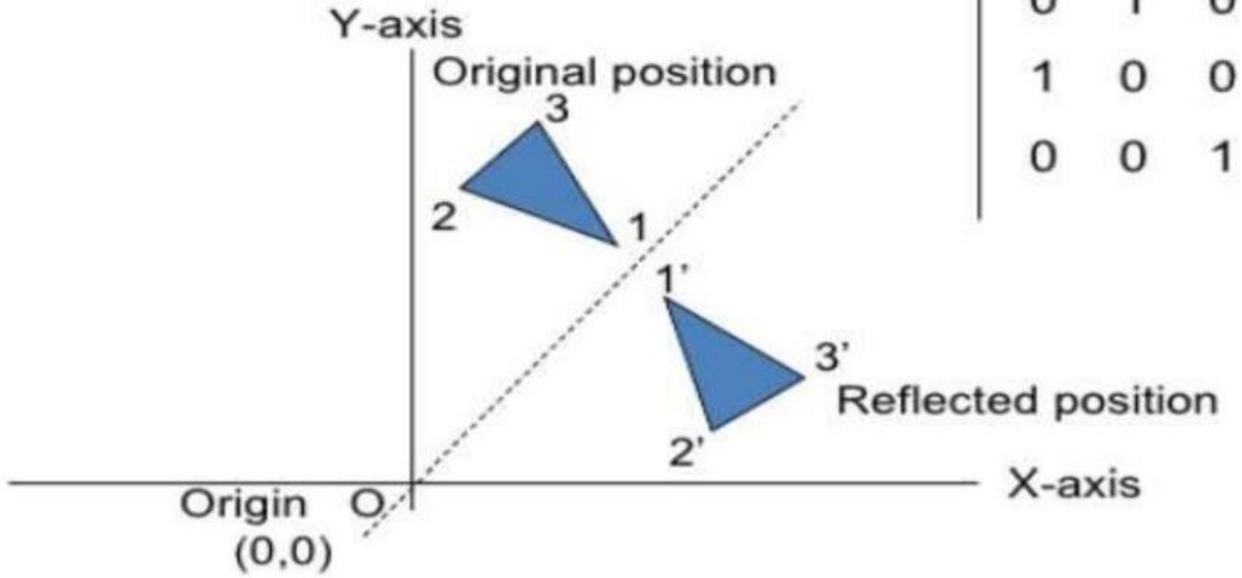


## Reflection of an object w.r.t the straight line y=-x



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

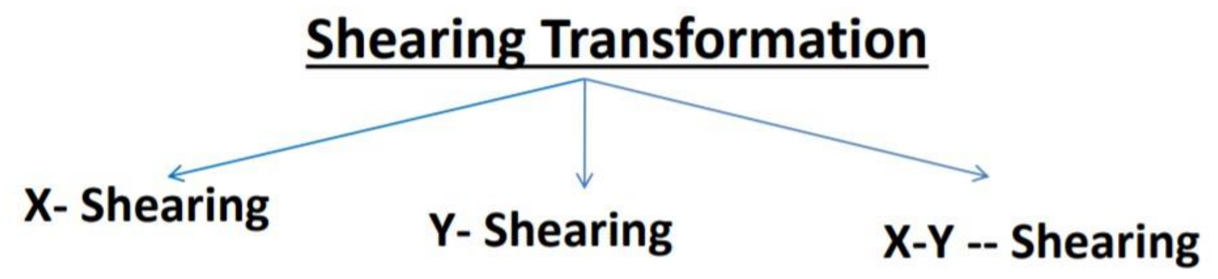
## Reflection of an object w.r.t the straight line y=x



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

## Shearing

- Shearing is also known as **Skewing**.
- It is a transformation that **slants the shape** of an object.



## Different types

- **X- Shearing relative to the  x axis**
- **X- Shearing relative to the other reference line**
- Y- Shearing relative to y axis
- **Y- Shearing relative to the other reference line**
- XY shearing

**X- Shearing relative to the  x axis**

X-Shear preserves the Y coordinate and changes are made to X coordinates.

**Y- Shearing relative to y axis**

Y-Shear preserves the X coordinate and changes are made to Y coordinates.
X shear

**X-Y - Shearing**

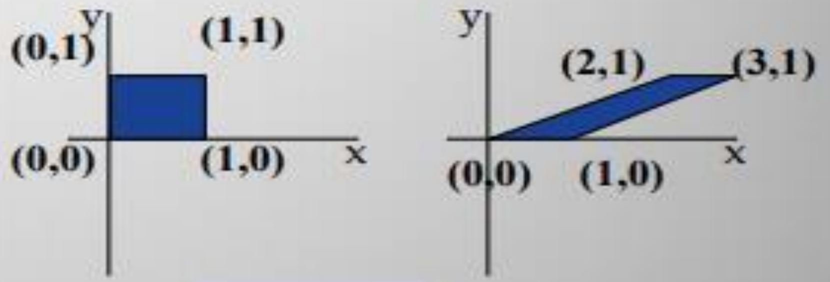
Here, both co – ordinates changes.

## Shear

- **Converted to a parallelogram**

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

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

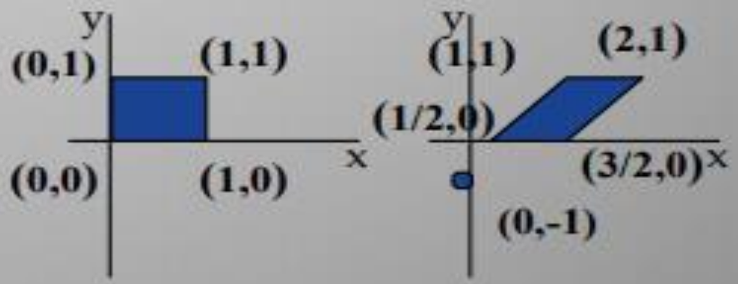


$x\ (Sh_x=2)$

- **Transformed to a shifted parallelogram**
(Y = Yref)

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

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



$(Sh_x=1/2,\ y_{ref}=-1)$

$$\begin{bmatrix} X_{new} \\ Y_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & Sh_y & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \\ 1 \end{bmatrix}$$

**Shearing Matrix**

**(In Y axis)**

**(Homogeneous Coordinates Representation)**

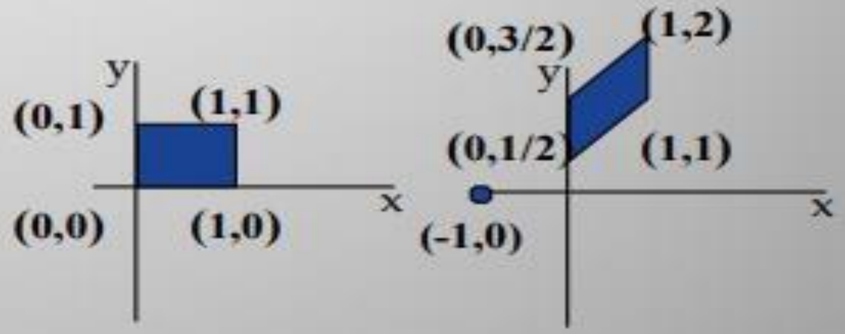
## Shear

- **Transformed to a shifted parallelogram**

  (X = Xref)

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

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



$(Sh_y = 1/2,\ x_{ref} = -1)$

## <u>Sample problems</u>

- Given a triangle with coordinate points A(3, 4), B(6, 4), C(5, 6). Apply the **reflection on the X axis** and obtain the new coordinates of the object.

- Given a triangle with coordinate points A(3, 4), B(6, 4), C(5, 6). Apply the **reflection on the Y axis** and obtain the new coordinates of the object.

- Given a triangle A(20,10) B(80,20) C(50,70). Find the co-ordinates of vertices after each of the following transformation.

(a) Reflection about the line x=y.

(b) Rotation of the triangle ABC about vertex A in clockwise direction for an angle 90 degree.

- Given a triangle with points (1, 1), (0, 0) and (1, 0). Apply shear parameter 2 on X axis and 2 on Y axis and find out the new coordinates of the object.

- Perform shearing for the given unit square A(0,0) B(1,0) C(1,1) D(0,1) relative to the reference line Yref =- 2 and shx=1/2.

- Flip the given quadrilateral A(10,8) B(22,8) C(34,17) D(10,27) about the origin and then zoom it to twice its size. Find the new position of the quadrilateral.

- Show that transformation matrix for a reflection about the line y=x is equivalent to a reflection relative to the x axis followed by a counter clockwise rotation of 90 degree.

- A triangle ABC with coordinates A(0,0), B(6,5), C(6,0) is scaled with scaling factors Sx=2 and Sy=3 about the vertex C(6,0). Find the transformed coordinate points.

- Perform a 45 degree rotation of a triangle ABC having the vertices at A(0,0) B(10,10) and C(50,20)

i. About the origin

ii. About an arbitrary point P(-10,-10)

- Show that transformation matrix for a reflection about the line y=x is equivalent to a reflection relative to the x axis followed by a counter clockwise rotation of 90 degree.

- Perform the following transformations on a point (6, 4).
- Translate by tx = −2 and ty = 4
- then, Scale by sx = 2 and sy = 1
- and Rotate by 90 in clockwise direction. Determine the final coordinates of the transformed point.

- Prove that the multiplication of 2D transformation matrices for two successive rotations is commutative.

- Show that the composition of two successive rotations are additive i.e. R(ϴ1).

R(ϴ2) = R(ϴ1+ ϴ2).

- Consider a triangle at (2,2), (10,2), (2,10). Perform the following 2D transformations in succession and find the resultant vertices

(i) Scale with respect to (2,2) by scaling factors (2,2) respectively along x and y directions.

(ii) Rotate by 90 counter clockwise direction.

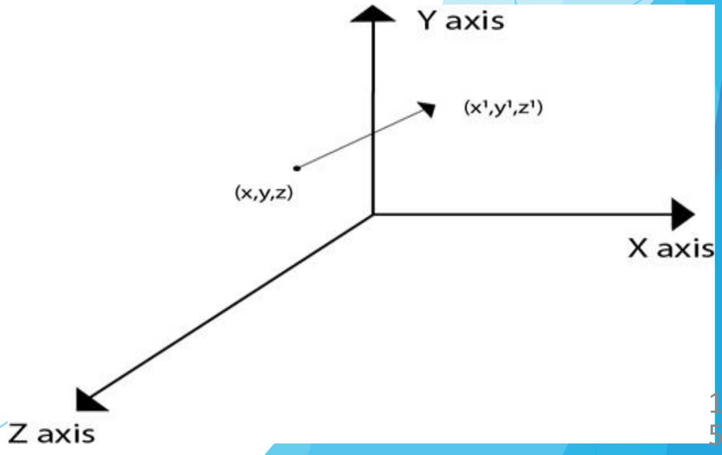
- Show that two successive reflections about either of the coordinate axes is equivalent to a single rotation about the coordinate origin.
- Determine a sequence of basic transformations that are equivalent to the x-direction shearing matrix.
- Reflect a triangle ABC about the line 3x-4y+8=0. The position vector of the coordinate ABC is given as A(4,1), B(5,2) and C(4,3).
- Describe the transformation which reflects a 2-D object about a line L which has a y-intercept(0,b) and an angle of intersection theta degree w.r.t. to the x-axis.

## Basic 3D Transformations

- Methods for geometric transformations and object modelling in 3D are extended from 2D methods by including the considerations for the z coordinates.
- 3D Transformations take place in a three dimensional plane. 3D Transformations are important and a bit more complex than 2D Transformations.
- 3D translation
- 3D rotation
- 3D scaling
- 3D reflection
- 3D shearing

# 3D Translation

- It is the movement of an object from one position to another position.
- Translation is done using translation vectors. There are three vectors in 3D instead of two.
- These vectors are in x, y, and z directions. Translation in the x-direction is represented using $T_x$. The translation is y-direction is represented using $T_y$. The translation in the z- direction is represented using $T_z$.
- If P is a point having co-ordinates in three directions (x, y, z) is translated, then after translation its coordinates will be (x' y' z') after translation. $T_x$ $T_y$ $T_z$ are translation vectors in x, y, and z directions respectively.



$$x' = x + T_x$$
$$y' = y + T_y$$
$$z' = z + T_z$$

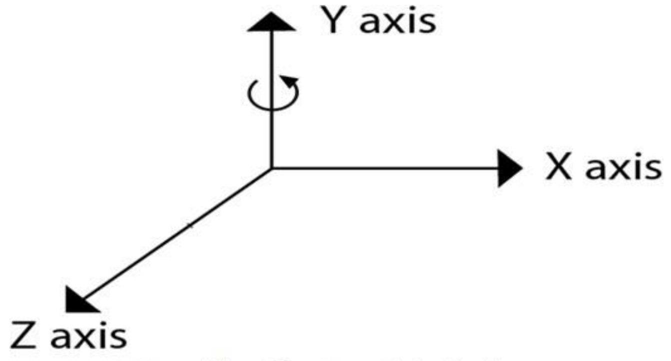
- In matrix form

$$P' = T \cdot P$$
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

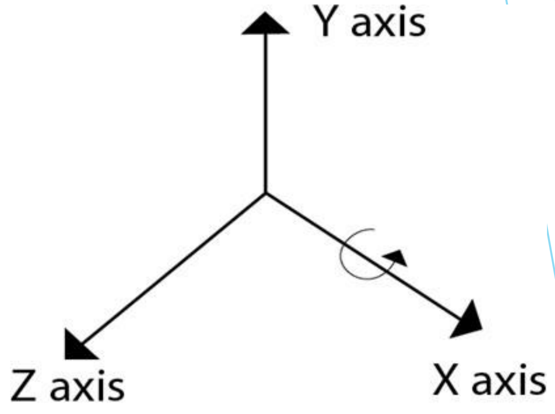
- Translate the given point P(10,10,10) with a translation vector (10,20,5)

## 3D Rotation

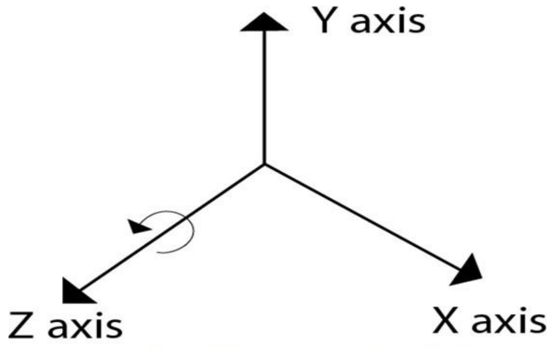
- It is moving of an object about an angle.
- Movement can be anticlockwise or clockwise.
- 3D rotation is complex as compared to the 2D rotation.
- For 2D we describe the angle of rotation, but for a 3D angle of rotation and axis of rotation are required. The axis can be either x or y or z.



rotation object Y axis anticlockwise



rotation object X axis anticlockwise



rotation object Z axis anticlockwise

## Z-axis rotation

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

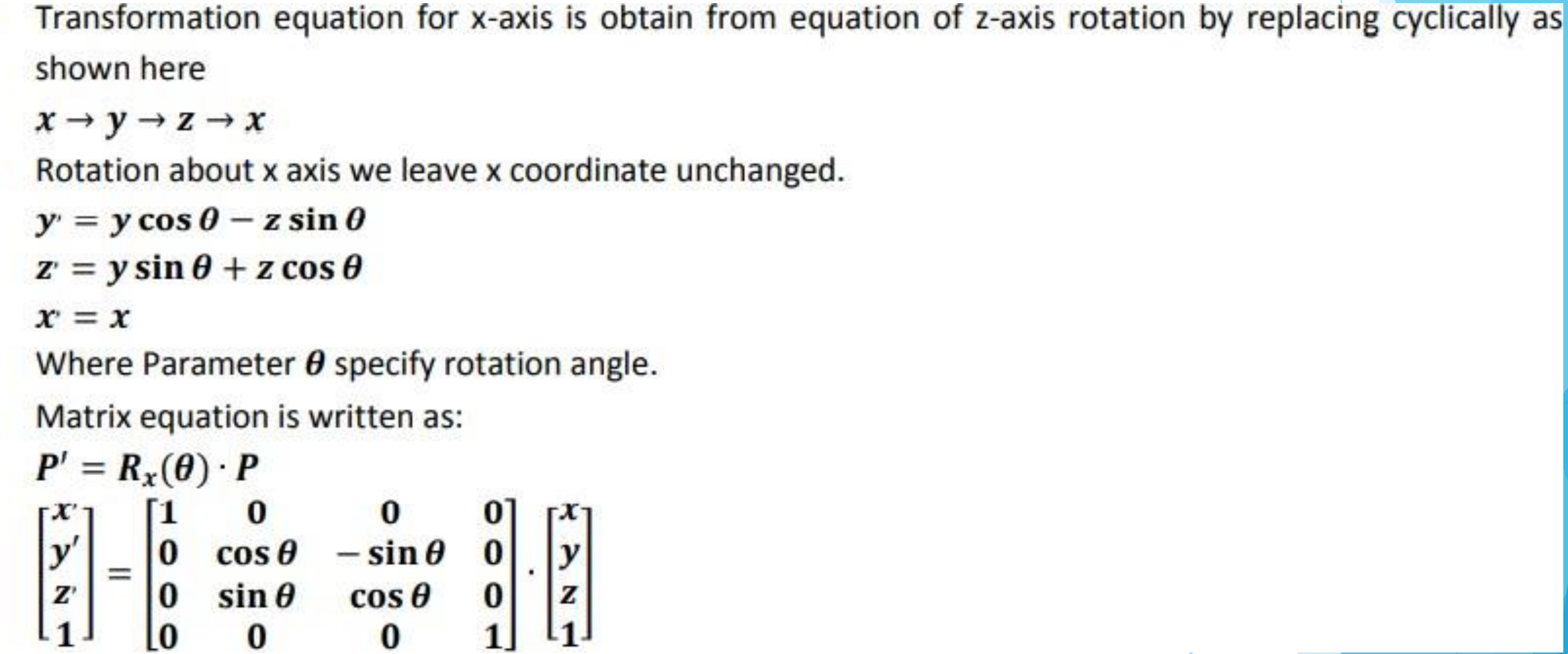
Where Parameter $\theta$ specify rotation angle.

Matrix equation is written as:

$$P' = R_z(\theta)\cdot P$$

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

# X-axis rotation



- Transformation equation for x-axis is obtain from equation of z-axis rotation by replacing cyclically as shown here

  $x \rightarrow y \rightarrow z \rightarrow x$
- Rotation about x axis we leave x coordinate unchanged.

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

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

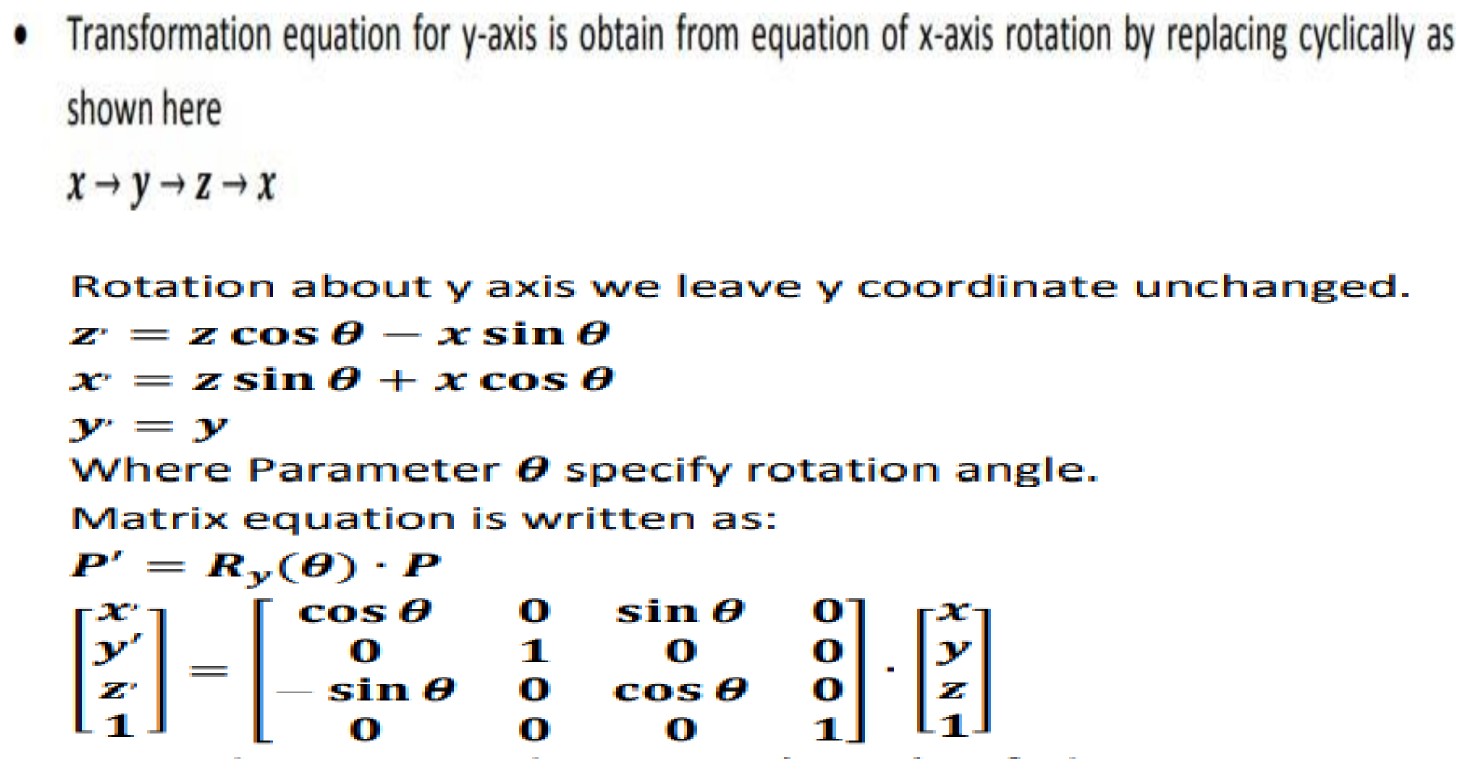
  $x' = x$

  Where Parameter $\theta$ specify rotation angle.
- Matrix equation is written as:

  $P' = R_x(\theta) \cdot P$

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

## Y-axis rotation



- Transformation equation for y-axis is obtain from equation of x-axis rotation by replacing cyclically as shown here

$x \rightarrow y \rightarrow z \rightarrow x$

Rotation about y axis we leave y coordinate unchanged.

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

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

$y' = y$

Where Parameter $\theta$ specify rotation angle.

Matrix equation is written as:

$P' = R_y(\theta) \cdot P$

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

Rotate a point P(5,5,5) 90 degree about z-axis

## Scaling

- Scaling is used to change the size of an object. The size can be increased or decreased. The scaling three factors are required $S_x$ $S_y$ and $S_z$.
- $S_x$=Scaling factor in x- direction
  $S_y$=Scaling factor in y-direction
  $S_z$=Scaling factor in z-direction

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

- Scale the line AB with endpoint (10,20,10) and (20,30,30) respectively with scaling factors (3,2,4)

# Fixed point scaling

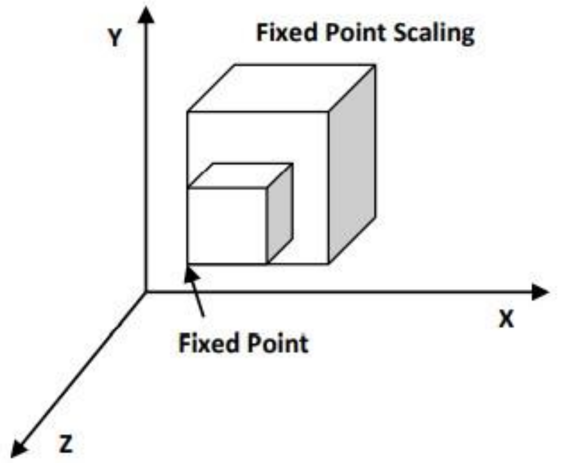


Fig. 5.7: - 3D Fixed point scaling.

- Fixed point scaling is used when we require scaling of object but particular point must be at its original position.
- Fixed point scaling matrix can be obtained in three step procedure.
  1. Translate the fixed point to the origin.
  2. Scale the object relative to the coordinate origin using coordinate axes scaling.
  3. Translate the fixed point back to its original position.
- Let's see its equation.

$$P' = T(x_f, y_f, z_f) \cdot S(s_x, s_y, s_z) \cdot T(-x_f, -y_f, -z_f) \cdot P$$

$$P' = \begin{bmatrix} 1 & 0 & 0 & x_f \\ 0 & 1 & 0 & y_f \\ 0 & 0 & 1 & z_f \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & -x_f \\ 0 & 1 & 0 & -y_f \\ 0 & 0 & 1 & {=}z_f \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot P$$

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

## Other Transformations

### Reflections

- Reflection means mirror image produced when mirror is placed at require position.
- When mirror is placed in XY-plane we obtain coordinates of image by just changing the sign of z coordinate.
- Transformation matrix for reflection about XY-plane is given below.

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

- Similarly Transformation matrix for reflection about YZ-plane is.

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

- Similarly Transformation matrix for reflection about XZ-plane is.

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

## Shears

- Shearing transformation can be used to modify object shapes.
- They are also useful in 3D viewing for obtaining general projection transformations.
- Here we use shear parameter **‘a’** and **‘b’**
- Shear matrix for Z-axis is given below

$$SH_z = \begin{bmatrix} 1 & 0 & a & 0 \\ 0 & 1 & b & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Similarly Shear matrix for X-axis is.

$$SH_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ a & 1 & 0 & 0 \\ b & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Similarly Shear matrix for X-axis is.

$$SH_y = \begin{bmatrix} 1 & a & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & b & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# University questions..

- What are the steps for general 3D rotation if the rotation axis is not parallel to any one of the principal axis. The rotation axis is defined by the points P1(x1,y1,z1) and P2(x2,y2,z2). Write down the composite matrix representation.

- Given a 3D triangle with points (0, 0, 0), (1, 1, 2) and (1, 1, 3). Apply shear parameter 2 on X axis, 2 on Y axis and 3 on Z axis and find out the new coordinates of the object.

- Given a 3D triangle with coordinate points A(3, 4, 1), B(6, 4, 2), C(5, 6, and Apply the reflection on the XY plane and find out the new coordinates of the object.

- Given a 3D triangle with coordinate points A(3, 4, 1), B(6, 4, 2), C(5, 6, 3). Apply the  reflection on the XZ plane and find out the new coordinates of the object.

- Given a 3D object with coordinate points A(0, 3, 3), B(3, 3, 6), C(3, 0, 1), D(0, 0, 0).  Apply the scaling parameter 2 towards X axis, 3 towards Y axis and 3 towards Z axis  and obtain the new coordinates of the object.

- Given a homogeneous point (1, 2, 3). Apply rotation 90 degree towards X, Y and Z axis and find out the new coordinate points.

- Given a 3D object with coordinate points A(0, 3, 1), B(3, 3, 2), C(3, 0, 0), D(0, 0, 0). Apply the translation with the distance 1 towards X axis, 1 towards Y axis and 2 towards Z axis and obtain the new coordinates of the object.

- What are the steps for general 3D rotation if the rotation axis is not parallel to any one of the principal axis. The rotation axis is defined by the points P1(x1,y1,z1) and P2(x2,y2,z2). Write down the composite matrix representation.

- Describe the steps involved in scaling a 3D object with respect to a fixed point (xf, yf, zf). Derive the composite transformation matrix.

- A rectangular parallelepiped is unit distance on Z-axis, 2 units on X-axis and 3 units on Y-axis. Determine the new coordinates of the parallelepiped when it is rotated counter clockwise about X-axis by an angle of 45 and Magnify the triangle ABC with A(0, 0), B(1, 1) and C(5, 2) to twice its size while keeping C(5, 2) fixed.

- Write the 3D translation matrix for moving an object by -2 units, -4 units and -6 units respectively in x, y and z directions.