

# Computer-Graphics-Module-2-Important-Topics

 For more notes visit

<https://rtpnotes.vercel.app>

- Computer-Graphics-Module-2-Important-Topics
  - 1. Flood fill Algorithm
    - What is flood fill algorithm?
    - Steps of the Algorithm
    - Pseudocode of the algorithm
    - Disadvantages
  - 2. Boundary fill Algorithm
    - Boundary fill algorithm
    - Four connected Pixel Approach
    - Boundary fill algorithm
  - 3. Scanline polygon fill
    - What is the Scanline Algorithm?
    - Steps of the Scanline Algorithm
    - Special Case of Polygon
    - Example Problem
      - Find the active edge table for filling the given triangle
        - Step 1: Sort the edges from Ymin to Ymax
        - Step 2: Form Global Edge Table (Sorted Edge Table)
        - Step 3: Active Edge Table
  - 4. 2D transformations and Problems
    - Types of Transformations
    - Translation
      - Example
    - Scaling

- What is scaling
- Scaling Factor
- Uniform Scaling:
- Coordinate Transformation:
- Matrix Form:
- Example
- Rotation
  - Example
- Homogenous Coordinates
- Reflection
- Shearing
  - X-Shear
  - Y-Shear
  - With Reference Points
- 5. Composite Transformation
  - Arbitrary Rotation Center
    - Matrix multiplication properties
    - Transformation in matrix form
  - Arbitrary Scaling Pivot

## ***1. Flood fill Algorithm***

### **What is flood fill algorithm?**

- The Flood Fill algorithm is used to fill a region of connected pixels with a specific color, starting from a seed point. This is similar to the "paint bucket" tool in many graphics software programs.
- When boundary is of many colors and interior is to be filled with one color we use this algorithm

### **Steps of the Algorithm**

- **Select a Seed Point:** Choose a starting point inside the region you want to fill. This is called the seed point.

- **Check the Color:** At this point, check if the current pixel has the old color (the color you want to replace). If it does, change it to the new fill color.
- **Expand to Neighboring Pixels:** Using either a 4-connected approach (up, down, left, right) or an 8-connected approach (also including the four diagonals), move to the neighboring pixels and repeat the process.

## Pseudocode of the algorithm

```

Procedure floodFill(x, y, fill_color, old_color: integer)
    // Check if the current pixel (x, y) has the old color that needs to be
    // replaced
    if (getPixel(x, y) = old_color) {
        // Set the color of the current pixel (x, y) to the new fill color
        setPixel(x, y, fill_color)

        // Recursively apply flood fill to the right neighboring pixel
        floodFill(x + 1, y, fill_color, old_color) // move right

        // Recursively apply flood fill to the left neighboring pixel
        floodFill(x - 1, y, fill_color, old_color) // move left

        // Recursively apply flood fill to the pixel below
        floodFill(x, y + 1, fill_color, old_color) // move down

        // Recursively apply flood fill to the pixel above
        floodFill(x, y - 1, fill_color, old_color) // move up
    }

```

## Disadvantages

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



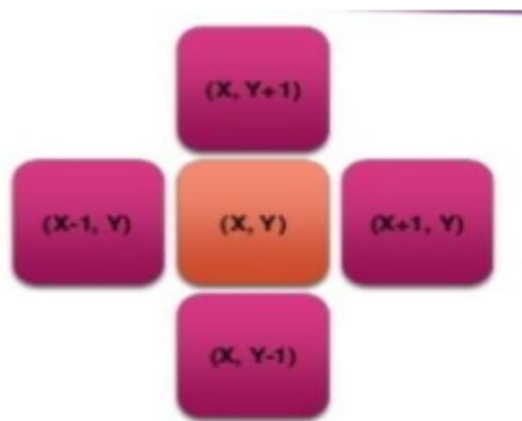
## 2. Boundary fill Algorithm

### Boundary fill algorithm

- **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
- Inputs are
  - interior point(x, y)
  - a fill color
  - a boundary color
- 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
    - 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**.

### Four connected Pixel Approach

- 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



### Boundary fill algorithm

```

void boundaryFill(int x, int y, int fillColor, int borderColor){
    getPixel(x,y,color);
    if ((color != borderColor) && (color != fillColor)){
        setPixel(x,y);
        boundaryFill(x+1,y,fillColor,borderColor);
        boundaryFill(x-1,y,fillColor,borderColor);
        boundaryFill(x,y+1,fillColor,borderColor);
        boundaryFill(x,y-1,fillColor,borderColor);
    }
}

```



### 3. Scanline polygon fill

#### What is the Scanline Algorithm?

The Scanline algorithm is used to fill polygons by drawing horizontal lines (scanlines) across the shape. It works by finding intersections of these scanlines with the edges of the polygon and filling the area between pairs of intersections.

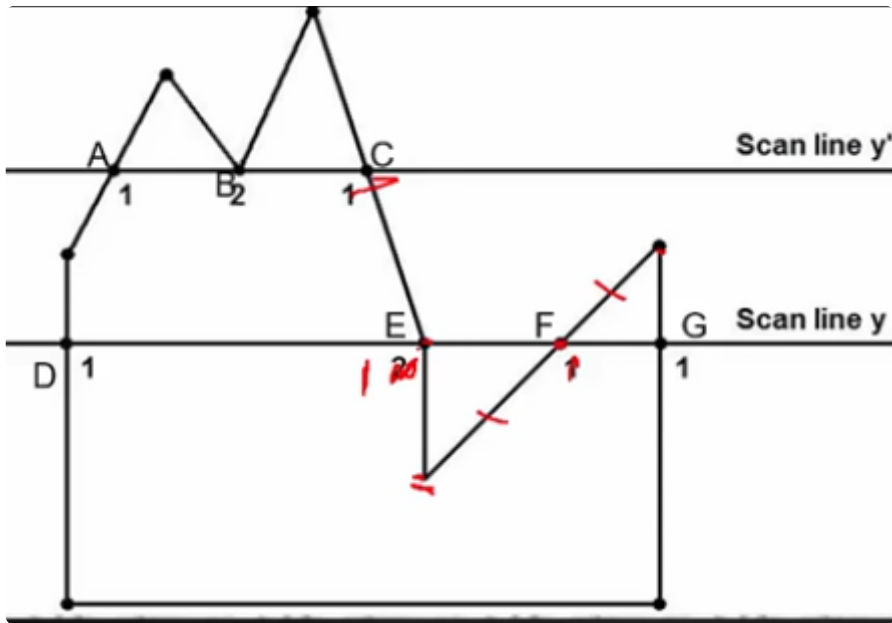
#### Steps of the Scanline Algorithm

- **Find Ymin and Ymax:** Identify the minimum and maximum Y-coordinates of the polygon. These values define the vertical range over which the scanlines will be drawn.
- **Intersect Scanlines with Polygon Edges:** For each horizontal line (scanline) from Ymin to Ymax, find where the scanline intersects the edges of the polygon. Each intersection point is noted.
- **Name Intersection Points:** Label each intersection point. For example, if a scanline intersects the polygon edges at points A and B, name them p0, p1, etc.
- **Sort Intersection Points:** For each scanline, sort the intersection points by their X-coordinates in increasing order. This ensures the points are processed from left to right.
- **Fill Between Pairs of Intersections:** For each pair of intersection points, fill the horizontal line between them. Ignore the alternate pairs to correctly fill the inside of the polygon while leaving the outside unfilled.

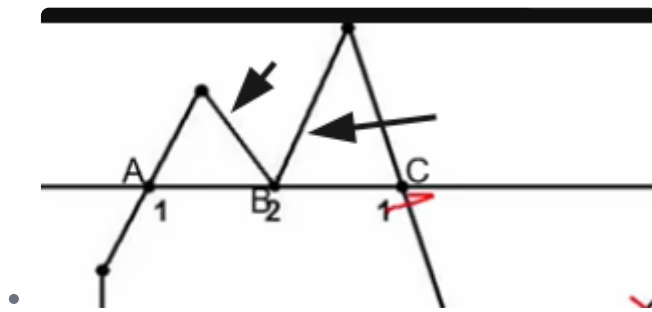
#### Special Case of Polygon

- Case 1: If both lines intersecting at the vertex are on the same side of the scanline, consider it as two points
- Case 2: If lines intersecting at the vertex are at opposite sides of the scanline, consider it as only one point

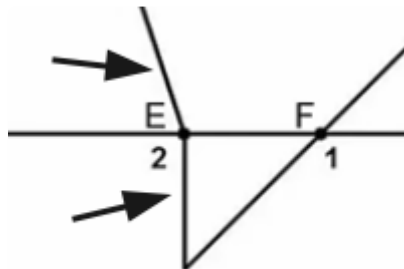
Lets check an example to understand this better



- Here there is a Polygon
- There are two scan lines
  - Scan line  $y'$
  - Scan line  $y$
- **Scan Line  $Y'$** 
  - Scan line  $y'$  intersects the following points
    - A, B, C
  - But here B is a vertex
  - As per **Case 1**
    - If both lines intersecting at the vertex are on the same side of the scanline, consider it as two points
    - There Both intersecting Vertex B is on one side



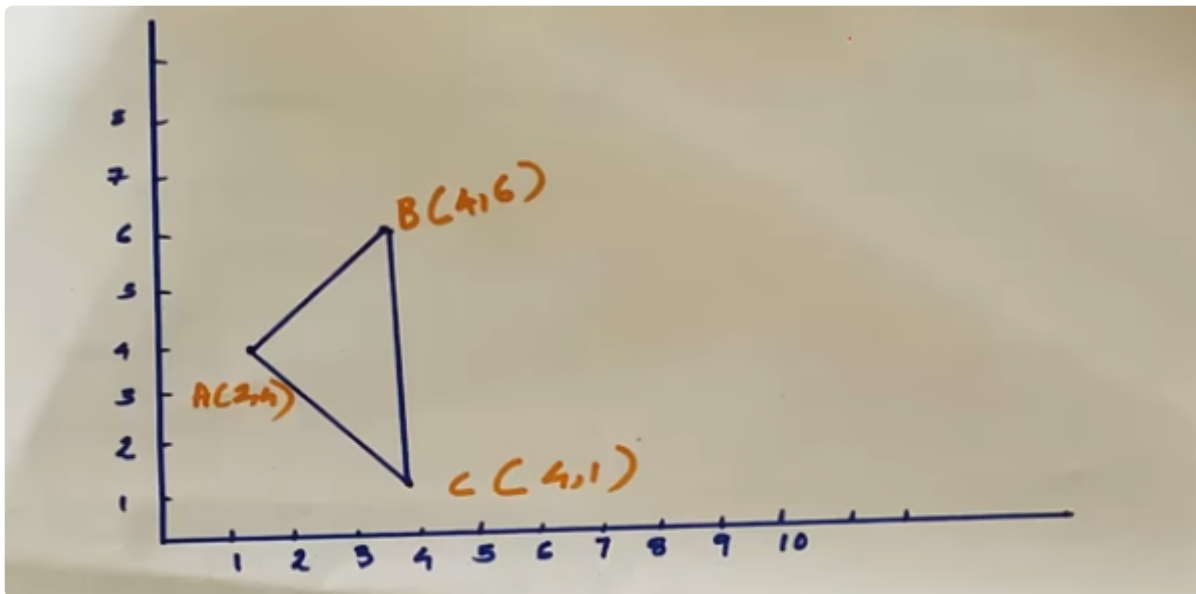
- So we are considering it as 2 Points
- Taking B two times
- {A,B,B,C}
- We get two pairs, (A,B) and (B,C)
- **Scan Line Y**
  - Here the intersection points are
    - D,E,F,G
    - Here E is a vertex
    - As per **Case 2**
      - If lines intersecting at the vertex are at opposite sides of the scanline, consider it as only one point
      - Here Both lines intersecting E is on opposite sides



- So we are considering it as a single point
- {D,E,F,G}
- Pairs are (D,E) and (F,G)

## Example Problem

Find the active edge table for filling the given triangle



### Step 1: Sort the edges from Ymin to Ymax

- Given A(2,4), B(4,6), C(4,1)
- Sorted list is
  - C(4,1), A(2,4), B(4,6)

### Step 2: Form Global Edge Table (Sorted Edge Table)

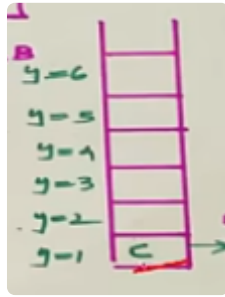
- What are Ymin and Ymax
  - We can see from the figure
    - The triangles minimum Y value is 1
    - The triangles maximum Y value is 6
- Lets make a table from Y = 1 to Y = 6

$y_{max}$	$y=6$	
	$y=5$	
	$y=4$	
	$y=3$	
	$y=2$	
$y_{min}$	$y=1$	

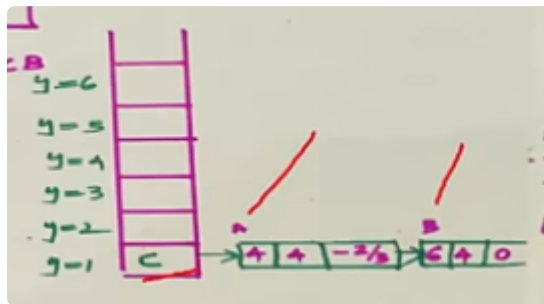
- Lets go through Each Y one by one
  - Starting with Ymin (Y = 1), From the figure you can see
    - At Y = 1 we have vertex C
    - Vertex C is connected to A
    - Edge CA



- Vertex C is connected to B
- Edge CB
  - Lets insert C into the table

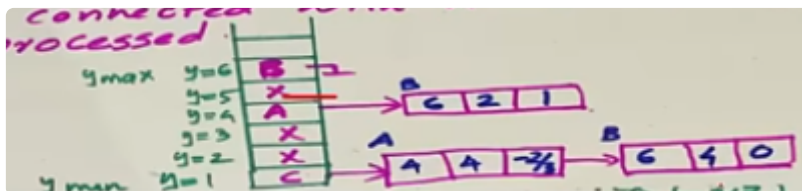


- Now, we need to show the connection to A and B using linkedlist



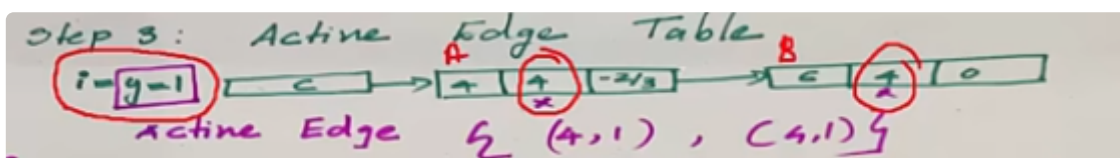
- You can see there are 3 fields in the linked list
  - First field = yMax
  - Second Field = x(yMin)
  - Third Field =  $1/\text{slope}$
- C-A
  - Between C and A, the highest Y value is 4
    - yMax = 4
  - Between C and A Lowest Y value is 1
    - yMin = 1
    - yMin is at C(4,1)
    - So its corresponding X value = 4
    - X = 4
  - Slope
    - From C to A
    - C coordinate  $x_1, y_1 = 4, 1$
    - A coordinate  $x_2, y_2 = 2, 4$
    - Slope =  $(y_2 - y_1) / (x_2 - x_1)$
    - Slope =  $4 - 1 / 2 - 4 = 3 / -2$

- $1/\text{slope} = -2/3$
- 3 fields Value
  - First field  $y_{\text{Max}} = 4$
  - Second Field  $X(y_{\text{Min}}) = 4 \times 1 = 4$
  - Third Field  $= 1/\text{slope} = -2/3$
- C-B
  - Do similar steps
  - You will get  $(6,4,0)$
- Lets check  $Y = 2$ 
  - There is no vertex with  $Y = 2$ , so discarding
  - Mark it on the Table as X
- Similarly for  $Y = 3$
- Lets Check  $Y = 4$ 
  - We have Vertex A
  - Vertex A is connected to C and B
  - Connection to C we already created linked list, We only need to handle B
  - Do similar steps as before as we will get  $(6,2,1)$
- $Y = 5$ , marking as X
- $Y = 6$ 
  - We have Vertex B
  - Its connected to A and C
  - There are already linked list for B-A and B-C connection, so no need to make
- Our table will now look like this



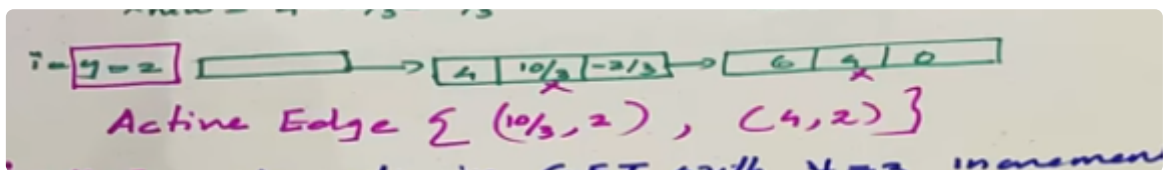
### Step 3: Active Edge Table

- Based on the global edge table we created earlier, we need to make the active edge table

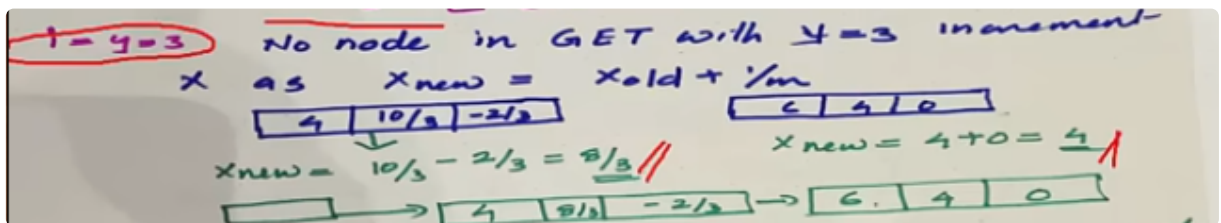


- $Y = 1$ 
  - Take first linked list element
    - Take the Middle value (take it as X)
      - Here the middle value in the linked list is 4
    - And take the current Y value of C
      - Y value of C is 1
    - We get (4,1)
  - Take the second linkedlist element
    - Take the middle value (Take it as X field)
      - Here the middle value is 4
    - Take Current value of Y
      - Y value of C is 1
    - We get (4,1)
  - So the Active Edge for  $Y = 1$  is  $\{(4,1), (4,1)\}$
  - So, Paint on the vertex (4,1)
- $Y = 2$ 
  - For  $Y = 2$ , for global edge table, its null
  - In the case of null
    - For  $Y = 1$ , modify the X field
    - Add slope to X and update
    - And create a new node out of that
  - New node - 1
    - First field ( $Y_{max}$ ) = Same = 4
    - Second Field = Previous X + previous slope (3rd field)
      - For  $Y = 1$ , X value is 4, and slope value is  $-2/3$
      - $4 + -2/3 = 10/3$
    - Third field (slope) = same
    - New node (4,10/3,-2/3)
  - New Node - 2
    - First Field ( $Y_{max}$ ) = same = 6
    - Second field = previous X + Previous slope
      - For  $Y = 1$ , X value is 4 and Slope is 0
      - $4 + 0 = 4$

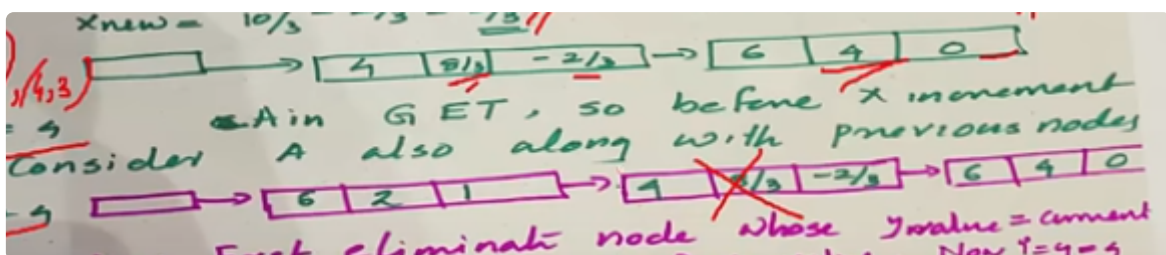
- Third field (slope) = same
- New node (6,4,0)
- Lets create Active Edge based on this
  - Node 1
    - Current Y value = 2
    - X value =  $10/3$
  - Node 3
    - Current Y value = 2
    - X value = 4
  - Active Edge =  $\{(10/3, 2), (4, 2)\}$



- Fill the vertices
  - $10/3, 2$  and  $4, 2$
- $Y = 3$ 
  - Here also its null in global table

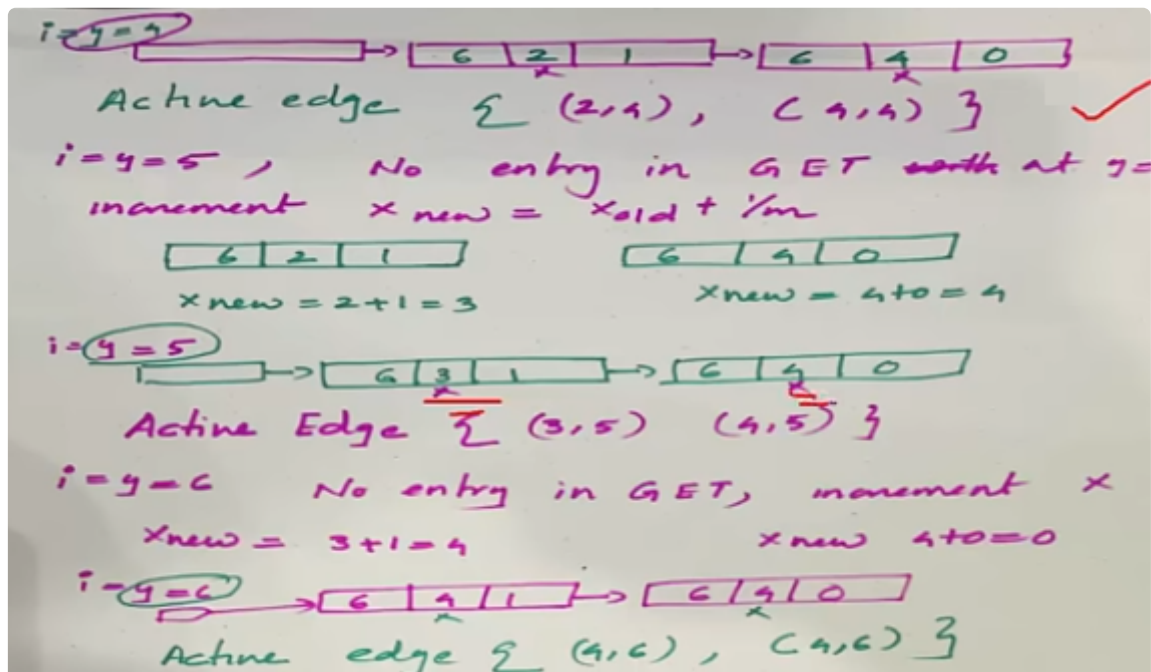


- The Active edge is  $\{(8/3, 3), (4, 3)\}$
- $Y = 4$ 
  - Here its not null, There is a node A
  - Take the Nodes from  $Y = 3$  and add them to the existing node
  - We need to eliminate one
  - Compare  $Y = 3$  and  $Y = 4$



- Attempt 1

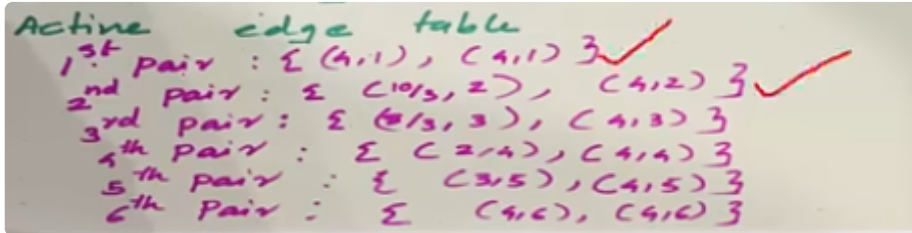
- In  $Y = 3$ , First node First value = 4
- In  $Y = 4$ , First node first value = 6
- Doesn't Match
- Attempt 2
  - In  $Y = 3$ , First node First value = 4
  - In  $Y = 4$ , Second node first value = 4
  - The value is matching, Removing 2nd node from  $Y = 4$
  - Keeping the remaining 2
- Active Edge
  - We have 2 nodes
    - 6,2,1
      - $Y = 4$
      - X value is 2
      - $(X, Y) = 2, 4$
    - 6,4,0
      - $Y = 4$
      - X value is 4
      - $(X, Y) = 4, 4$
  - Active Edge =  $\{(2, 4), (4, 4)\}$
- Repeat Same steps for  $Y = 5$  and  $Y = 6$



- $Y = 5$

- Active Edge =  $\{ (3,5), (4,5) \}$
- $Y = 6$
- Active Edge =  $\{ (4,6), (4,6) \}$

We will get this Active Edge table at the end



Active edge table

1 <sup>st</sup> pair	: $\{ (4,1), (4,1) \}$ ✓
2 <sup>nd</sup> pair	: $\{ (10/3, 2), (4,2) \}$ ✓
3 <sup>rd</sup> pair	: $\{ (4/3, 3), (4,3) \}$
4 <sup>th</sup> pair	: $\{ (2,4), (4,4) \}$
5 <sup>th</sup> pair	: $\{ (3,5), (4,5) \}$
6 <sup>th</sup> pair	: $\{ (4,6), (4,6) \}$



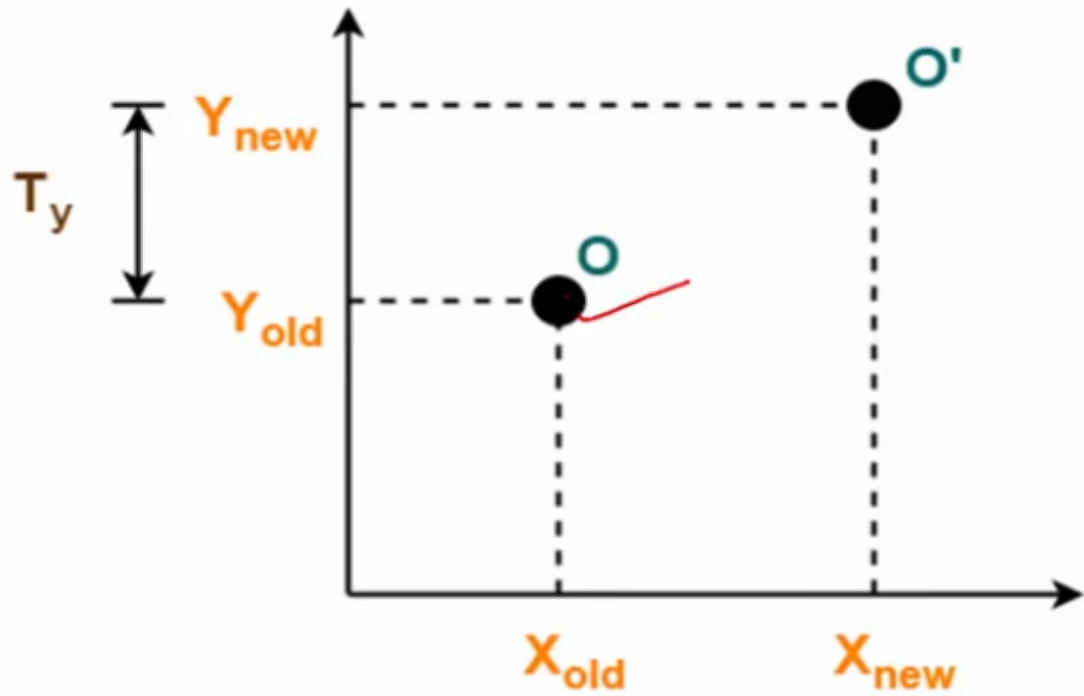
## 4. 2D transformations and Problems

### Types of Transformations

- Translation
- Scaling
- Rotation
- Reflection
- Shearing

### Translation

- 2D Translation is the process of moving an object from one position to another in a 2 dimensional plane
- Initial Coordinates of the object O ( $X_{old}$ ,  $Y_{old}$ )
- New coordinates of the object O after translation = ( $X_{new}$ ,  $Y_{new}$ )
- Translation vector or shift vector = ( $T_x$ ,  $T_y$ )



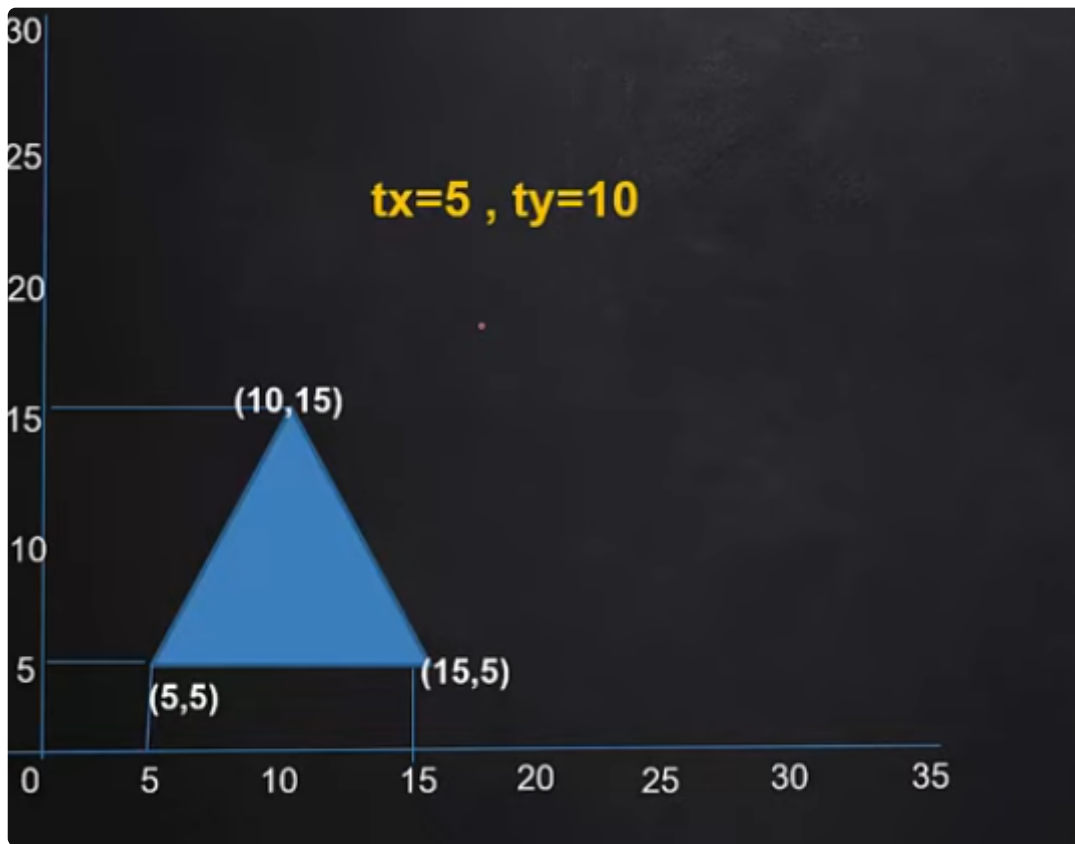
This translation is achieved by adding the translation coordinates to the old coordinates of the object as

- $X_{new} = X_{old} + T_x$
- $Y_{new} = Y_{old} + T_y$
- In Matrix form, the above translation equations may be represented as

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \end{bmatrix}$$

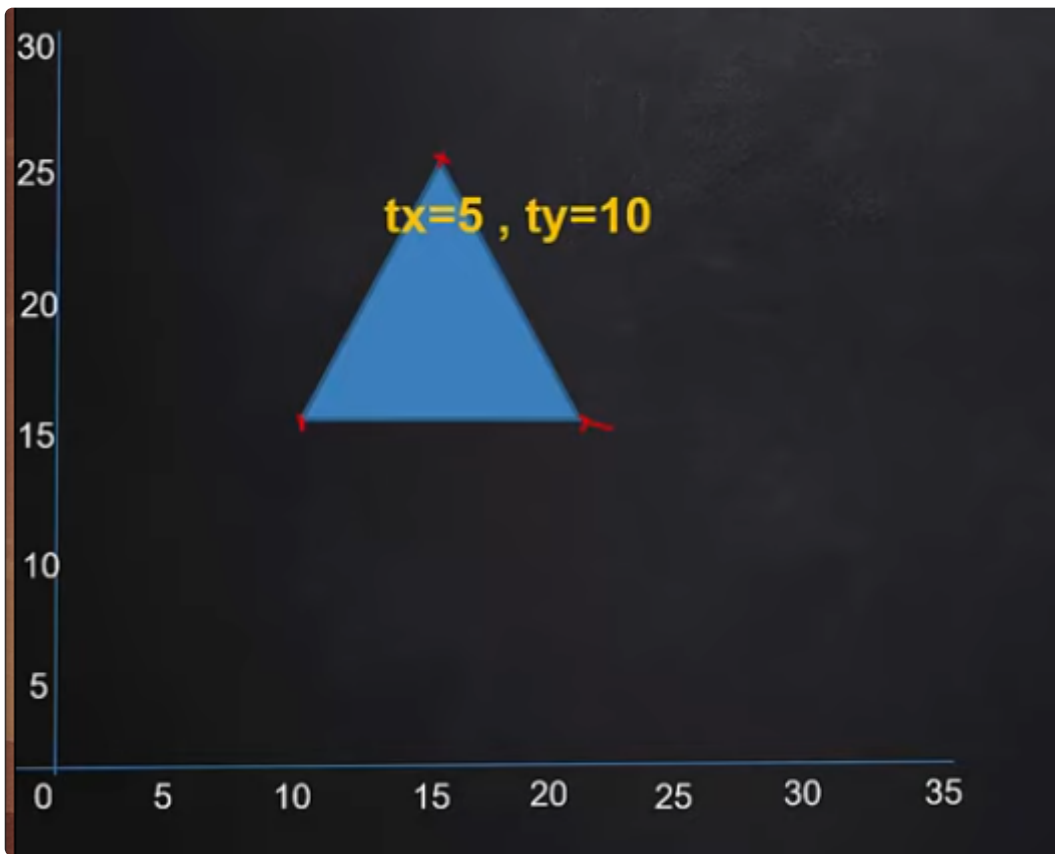
**Translation Matrix**

**Example**



- We need to translate this triangle 5 units in X and 10 units in Y
- Take each vertices
  - $(10,15) \rightarrow (15,25)$ 
    - $10 = 10 + tx = 10 + 5 = 15$
    - $15 = 15 + ty = 15 + 10 = 25$
  - $(5,5) \rightarrow (10,15)$
  - $(15,5) \rightarrow (20,15)$





## Scaling

### What is scaling

- In computer graphics, scaling is a process of modifying or altering the size of objects
- Scaling may be used to increase or reduce the size of the object
- Scaling subjects the coordinate points of the original object to change

### Scaling Factor

- Scaling factor determines whether the object size is to be increased or reduced
- If scaling factor  $> 1$  then the object size is increased
- If scaling factor  $< 1$ , then the object size is reduced
- Specifying a value of 1 for both  $S_x$  and  $S_y$  leaves the size of objects unchanged

### Uniform Scaling:

- When  $S_x$  and  $S_y$  are assigned the same value, uniform scaling is produced that maintains relative object proportions
- When  $S_x$  and  $S_y$  are assigned the value  $1/2$  length and distance from origin are reduced by half

## Coordinate Transformation:

- Initial coordinates of the object O = (Xold, Yold)
- After scaling, the new position is (Xnew, Ynew).
- The scaling factor for the X axis is Sx, and for the Y axis is Sy.
- The new coordinates are calculated as:
  - $X_{new} = X_{old} \times S_x$
  - $Y_{new} = Y_{old} \times S_y$

## Matrix Form:

- In Matrix form. the scaling equations may be represented as

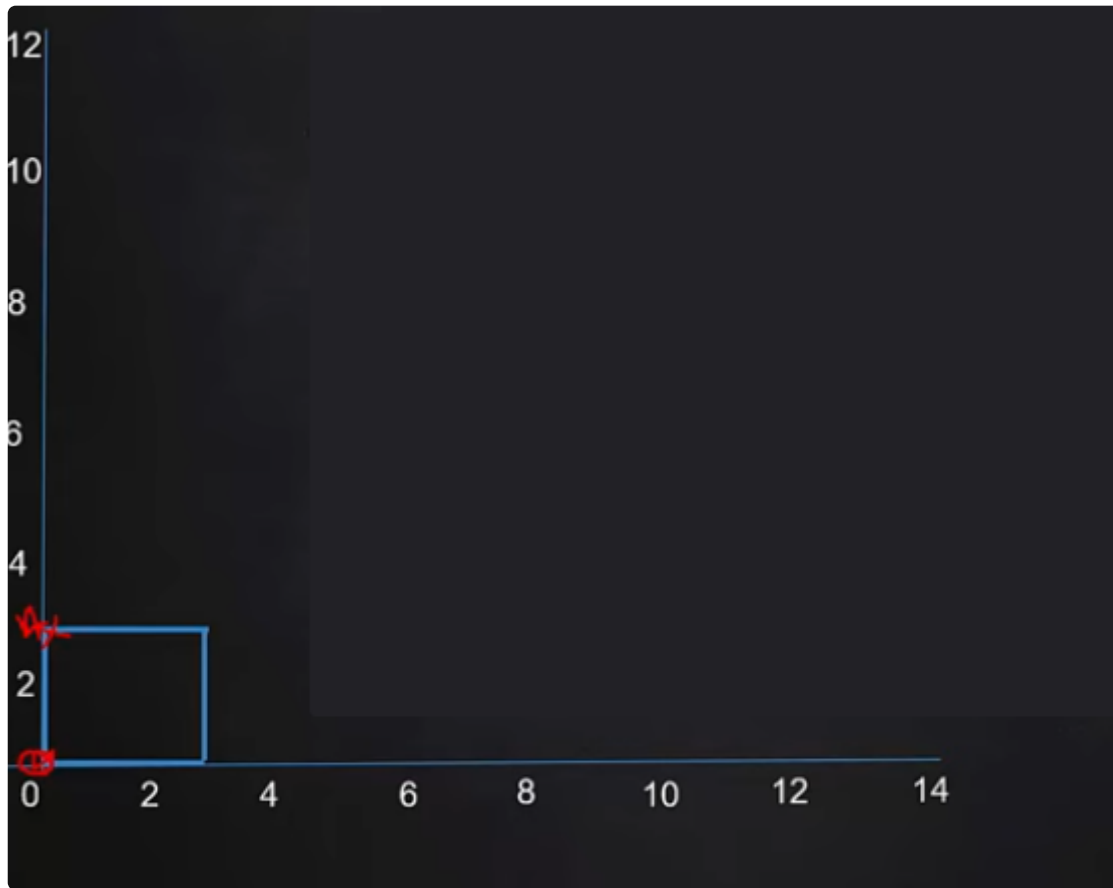
$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix}$$

**Scaling Matrix**

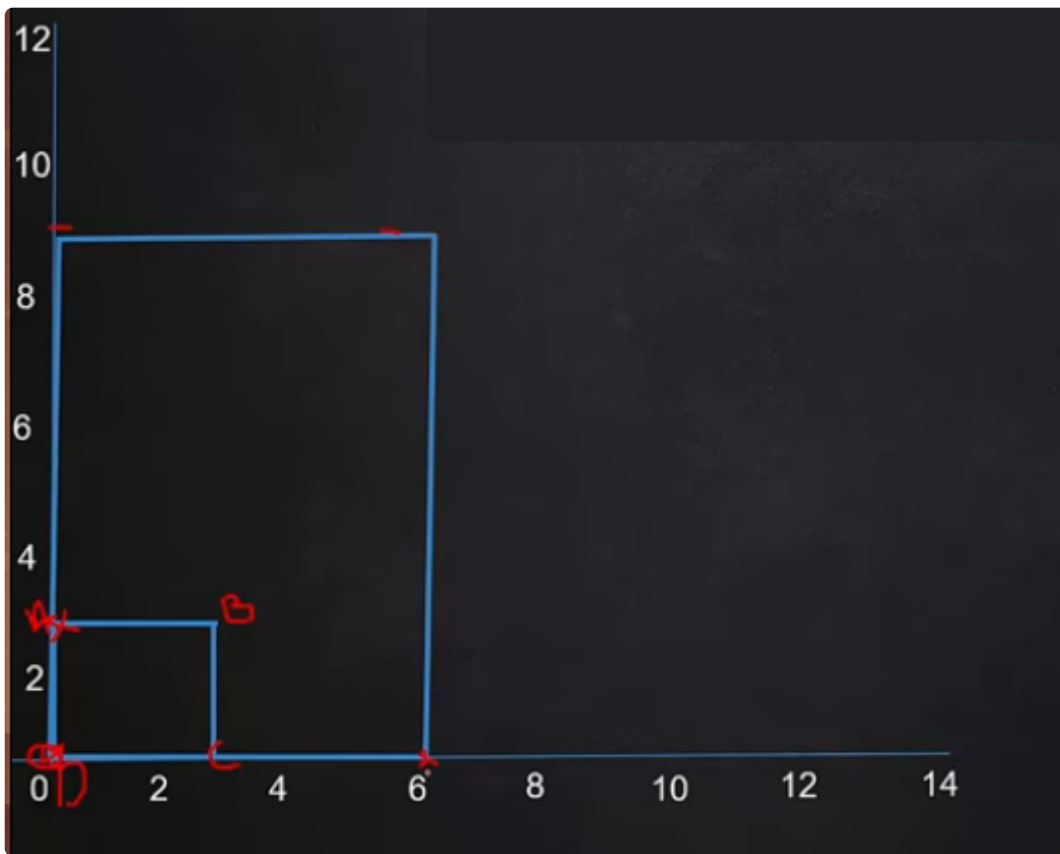
## Example

**A Square with coordinates A(0,3), B(3,3), C(3,0),D(0,0)**

- Scaling Factor along X axis = 2
- Scaling Factor along Y axis = 3

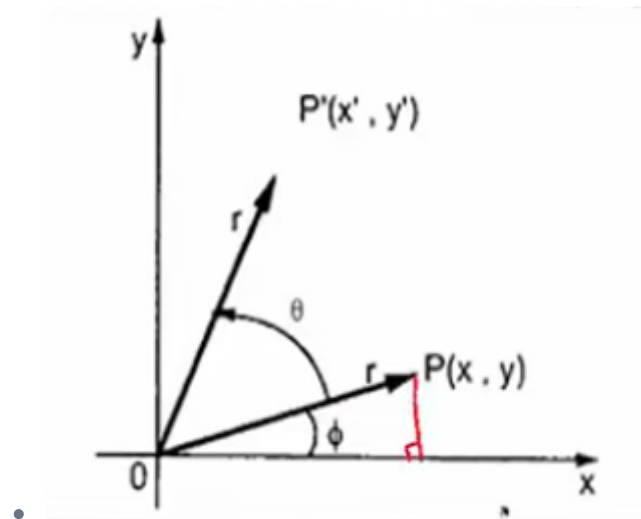


- $(0,3) \rightarrow (0,9)$ 
  - Scaling factor  $X = 2$
  - Scaling factor  $Y = 3$
  - $X_{\text{new}} = X_{\text{value}} \times \text{Scaling factor } X = 0 \times 2 = 0$
  - $Y_{\text{new}} = Y_{\text{value}} \times \text{Scaling factor } Y = 3 \times 3 = 9$
- $(3,3) \rightarrow (6,9)$ 
  - $X_{\text{new}} = 3 \times 2 = 6$
  - $Y_{\text{new}} = 3 \times 3 = 9$
- $(3,0) \rightarrow (6,0)$ 
  - $X_{\text{new}} = 3 \times 2 = 6$
  - $Y_{\text{new}} = 0 \times 3 = 0$
- $(0,9) \rightarrow (0,0)$ 
  - $X_{\text{new}} = 0 \times 2 = 0$
  - $Y_{\text{new}} = 0 \times 3 = 0$



## Rotation

- 2D Rotation is the process of rotating an object with respect to an angle in a two dimensional plane



- Using standard trigonometry, the original coordinate of point P XY can be represented as

$$X = r \cos \phi \dots\dots (1)$$

$$Y = r \sin \phi \dots\dots (2)$$

- We can represent the coordinate after rotating X'Y' as

$$x' = r \cos (\phi + \theta) = r \cos \phi \cos \theta - r \sin \phi \sin \theta \dots\dots (3)$$

$$y' = r \sin (\phi + \theta) = r \cos \phi \sin \theta + r \sin \phi \cos \theta \dots\dots (4)$$

- When using equations 1,2,3,4 we get

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

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

$$\begin{vmatrix} x' \\ y' \end{vmatrix} = \begin{vmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{vmatrix} \begin{vmatrix} x \\ y \end{vmatrix}$$

### Example

Given a triangle with corner coordinates (0,0), (1,0) and (1,1), Rotate the triangle by 90 degree anticlockwise direction and find out the new coordinates

- Lets take each coordinate one by one

$$(0,0) \rightarrow (0,0)$$

$$\begin{aligned} 0 \cos 90 - 0 \sin 90 &= 0 \\ 0 \sin 90 + 0 \cos 90 &= 0 \end{aligned}$$

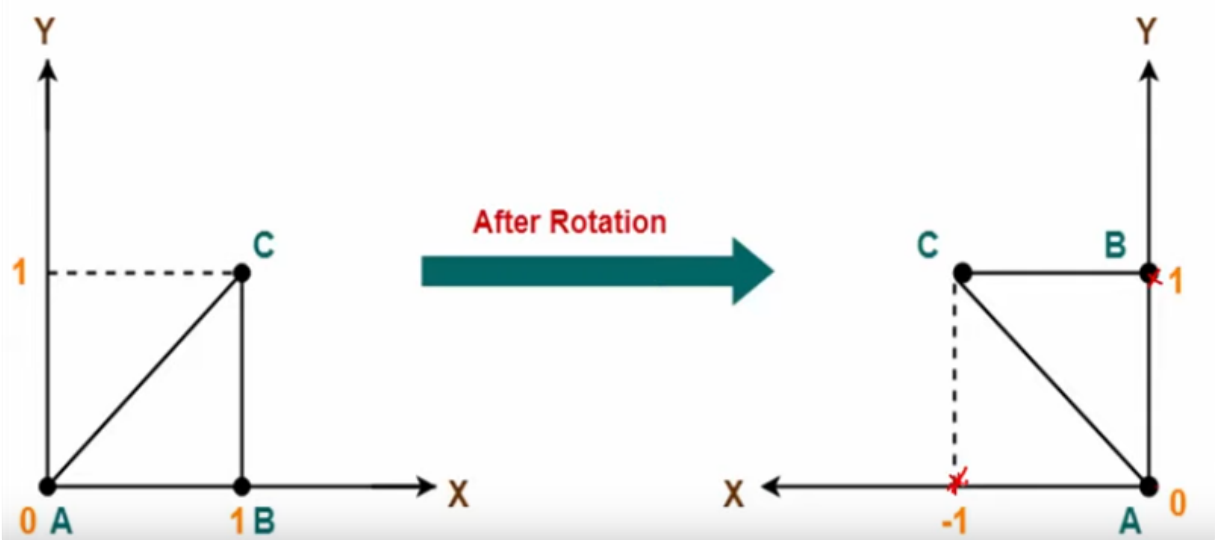
$$(1,0) \rightarrow (0,1)$$

$$\begin{aligned} 1 \cos 90 - 0 \sin 90 &= 0 \\ 1 \sin 90 + 0 \cos 90 &= 1 \end{aligned}$$

$$(1,1) \rightarrow (-1,1)$$

$$\begin{aligned} 1 \cos 90 - 1 \sin 90 &= -1 \\ 1 \sin 90 + 1 \cos 90 &= 1 \end{aligned}$$

- Before and after rotation



## Homogenous Coordinates

- In homogenous coordinate system, 2 dimensional coordinate positions are represented by triple coordinates (x,y,1)
- We can perform all transformations using matrix multiplications

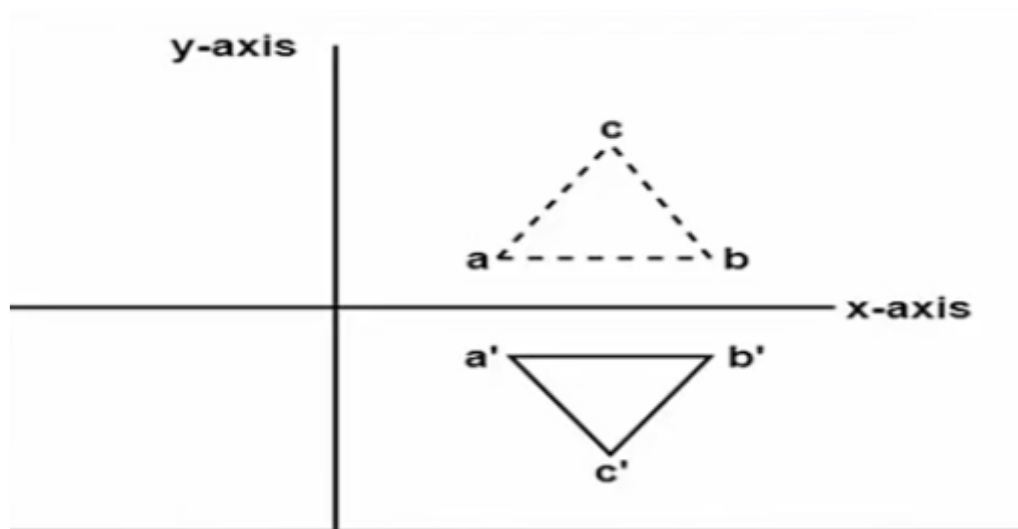
Translation: 
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Rotation: 
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Scaling: 
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

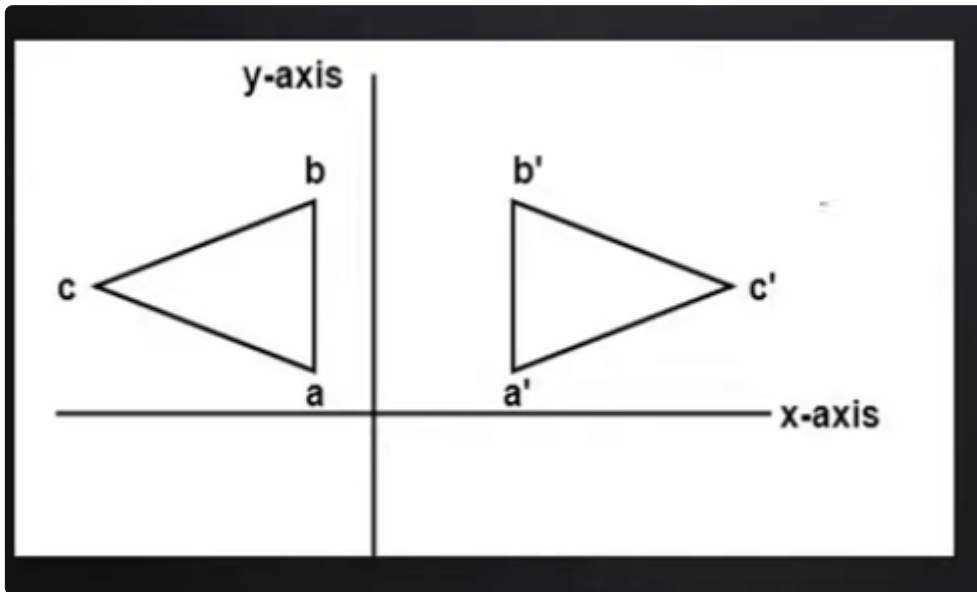
## Reflection

- It is a transformation that produces a mirror image of an object. The mirror image can be either about the x axis or y axis, **The Object is rotated by 180**
- Reflection about x-axis



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

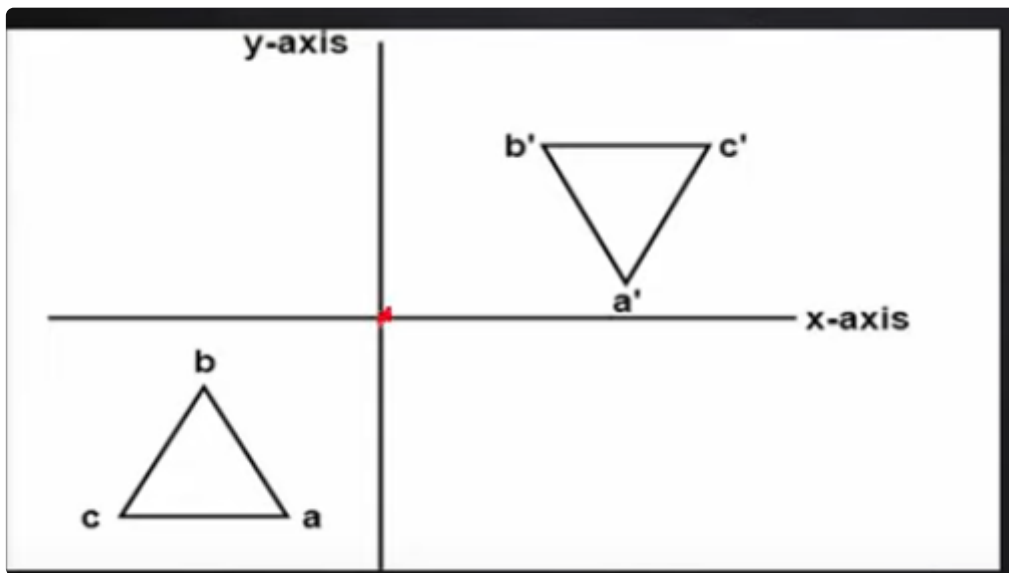
- Here -1 means, Reversing Y direction
- Reflection about y-axis



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

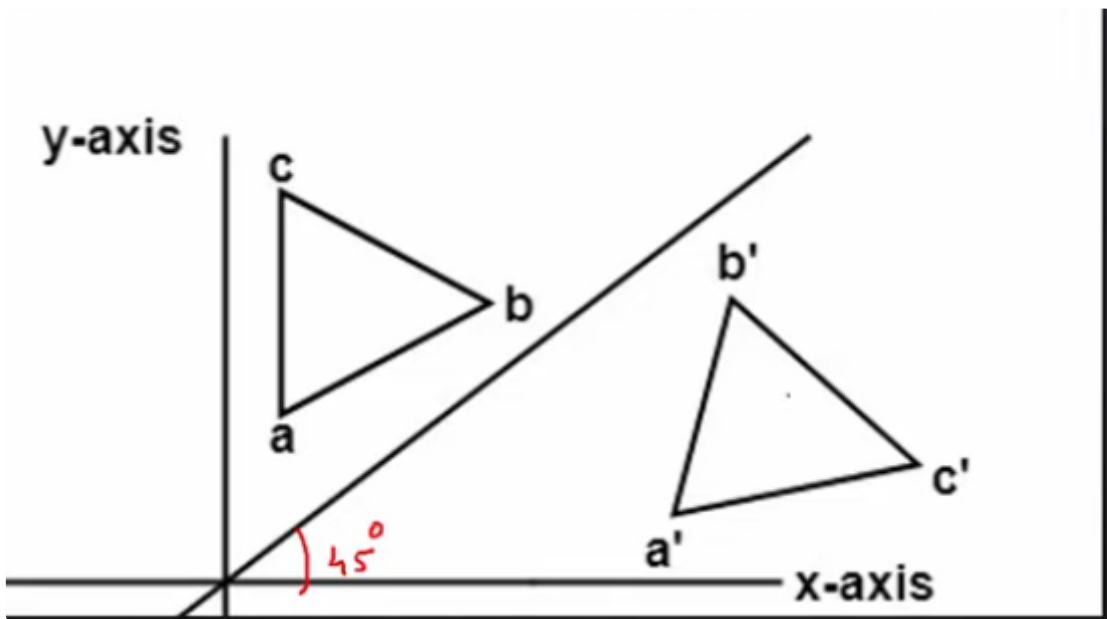
- Reflection about an axis perpendicular to xy plane and passing through origin





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

- Reflection about line  $y = x$



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

- Reflection about line  $y = -x$

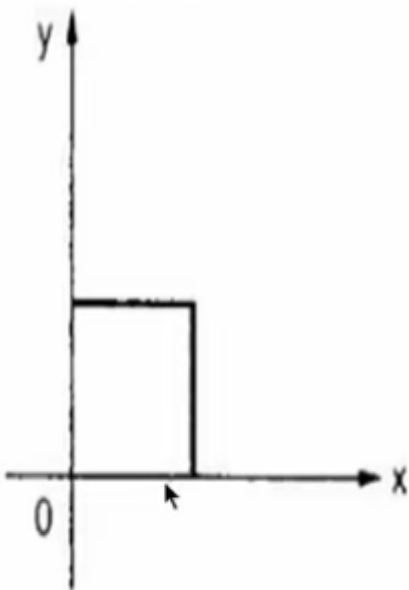
$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

## Shearing

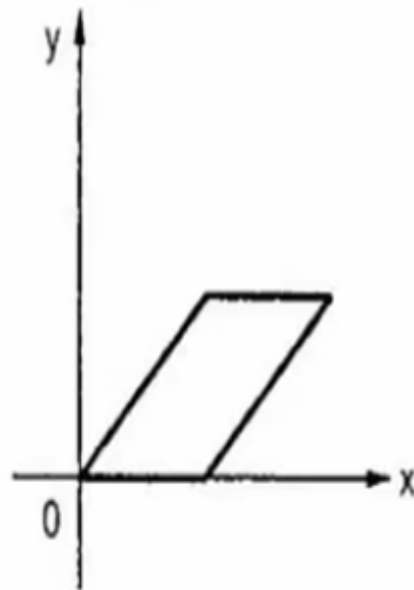
- A transformation that slants the shape of an object is called the shear transformation. There are two shear transformations X-Shear and Y-Shear.
- One shifts X coordinates values and other shifts Y coordinate values
- However, In both the cases only one coordinate changes its coordinates and other preserves its values
- Shearing is also termed as Skewing

### X-Shear

- The X-shear preserves the Y coordinate and the changes are made to X coordinates, Which causes the vertical lines to tilt right or left



(a) Original object



(b) Object after x shear

- The formula used

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

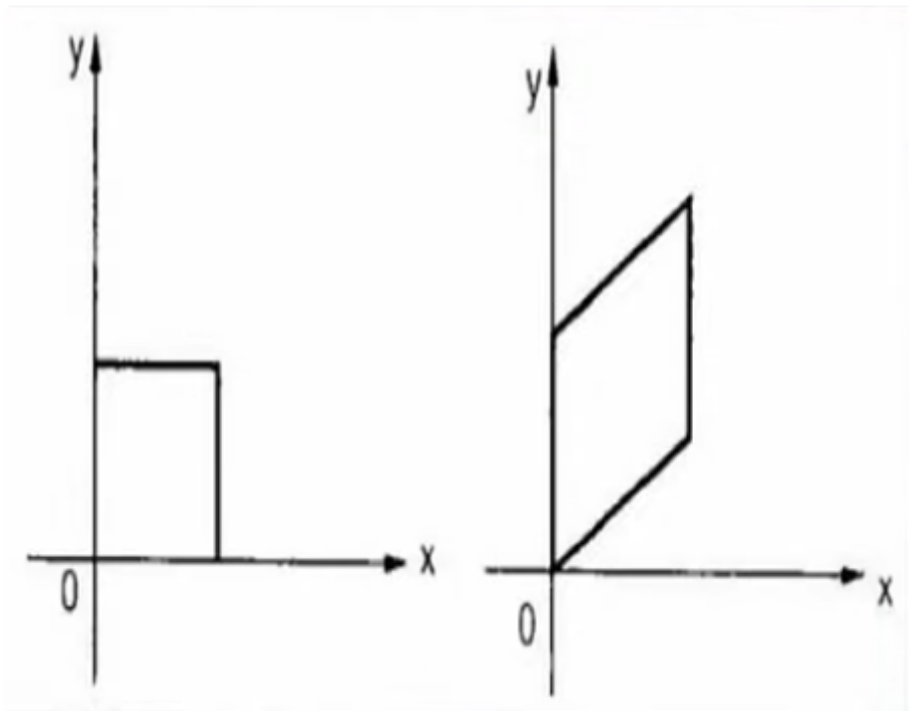
$$y' = y$$

- Here Sh is Shear factor

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

## Y-Shear

- The Y-Shear preserves the X Coordinates and changes the Y coordinates



- The formula used

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

$$x' = x$$

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

### With Reference Points

- X-Shear

## X Shear wrt a reference point $y_{ref}$

$$\begin{bmatrix} X' \\ Y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & shx & -shx \cdot y_{ref} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

$$X' = X + shx \cdot Y - shx \cdot y_{ref}$$

$$X + shx(Y - y_{ref})$$

- Y-Shear

## Y Shear wrt a reference point $x_{ref}$

$$\begin{bmatrix} X' \\ Y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ shy & 1 & -shy \cdot x_{ref} \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

## 5. Composite Transformation

- Composite Transformation, is the process of applying several transformations in succession to form one overall transformation

### Arbitrary Rotation Center

- To rotate about an arbitrary point  $P(px, py)$  by
  - 1. Translate the object so that  $P$  will coincide with the origin  $T(-px, -py)$
  - Rotate the object
  - Translate the object back  $T(px, py)$

### Matrix multiplication properties

- Matrix multiplication is associative
  - $M3 \times M2 \times M1 = (M3 \times M2) \times M1 = M3 \times (M2 \times M1)$
- Transformation products may not be commutative  $A \times B \neq B \times A$

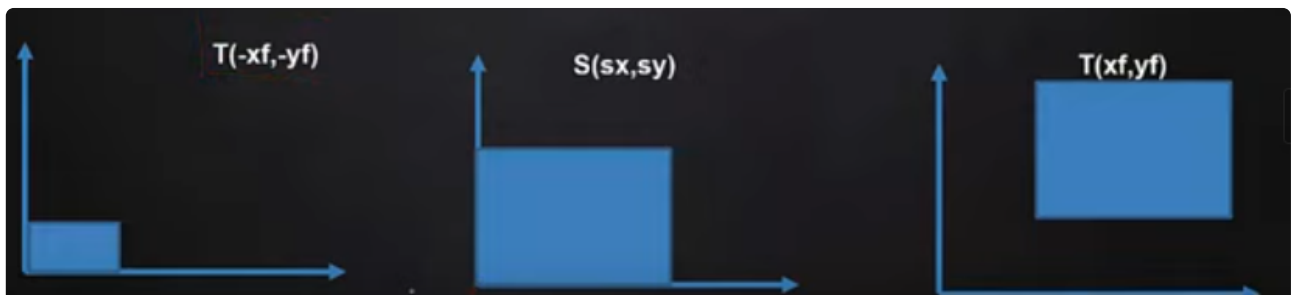
### Transformation in matrix form

■ Put in matrix form:  $T(px, py)$   $R(\theta)$   $T(-px, -py)$  \*  $P$

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

### Arbitrary Scaling Pivot

- To scale about an arbitrary pivot point  $P(xf, yf)$
- Translate the object so that  $P$  will coincide with the origin  $T(-xf, -yf)$
- Scale the object  $S(sx, sy)$
- Translate the object back:  $T(xf, yf)$



- The matrix can be written as

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

- First matrix for translation
- Second one for scaling
- Third one from translating back to original position

