Module - 2(Filled Area Primitives and transformations)
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.

AREA FILLING OR POLYGON FILLING ALGORITHMS

- Fill-Area algorithms are used to fill the interior of a polygonal shape.
- Many algorithms perform fill operations by first identifying the interior points, given the polygon boundary.

- Boundary fill algorithm
- Flood fill algorithm
- Scan line filling 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.

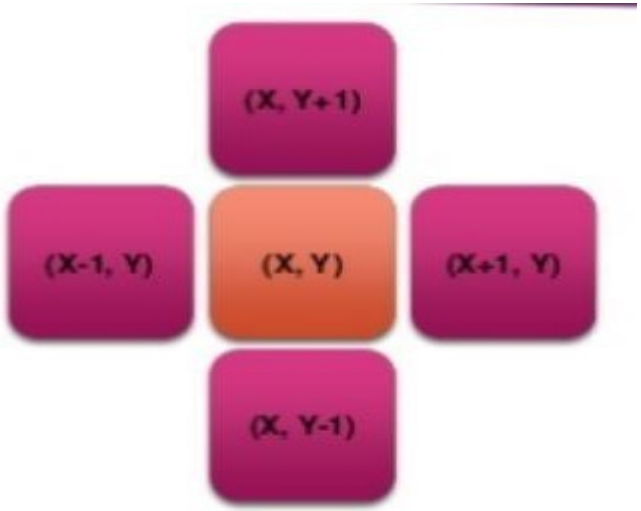
- This algorithm works **only if** the color with which the region has to be filled and the color of the boundary of the region are different.

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

- 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

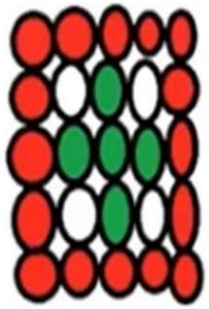


Boundary Fill Algorithm (Code)

```
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);
  }
}
```

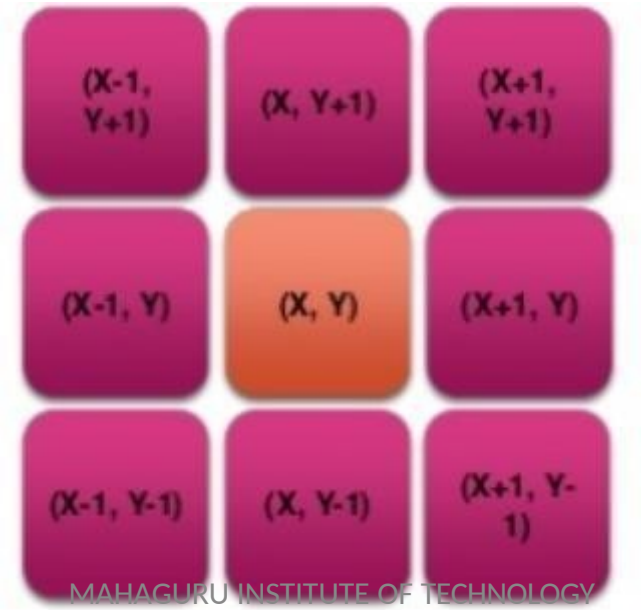
Problem in 4-connected

- There is a problem with this technique that 4-connected pixels technique cannot fill all the pixels.



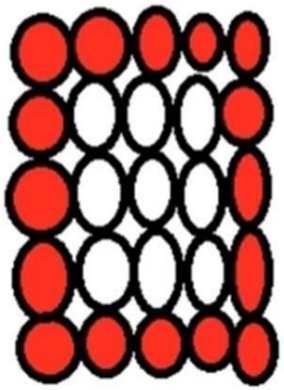
8-connected pixels

- 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.

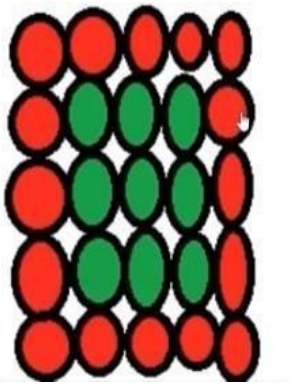


```
void boundaryFill8(int x, int y, int fill_color,int boundary_color)
{
    if(getpixel(x, y) != boundary_color &&
       getpixel(x, y) != fill_color)
    {
        putpixel(x, y, fill_color);
        boundaryFill8(x + 1, y, fill_color, boundary_color);
        boundaryFill8(x, y + 1, fill_color,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

8-connected pixel filling

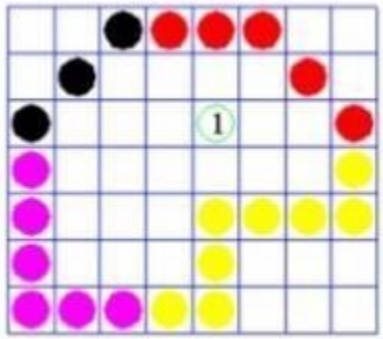


8-connected pixel filling



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.
- 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 positions until all interior points have been repainted.


1

```
Void floodFill4 (int x, int y, int fillColor, int oldColor)
{
    if (getpixel (x, y) == oldcolor)
    {
        setcolor (fillcolor);
        setpixel (x, y);
        floodFill4 (x+1, y, fillColor, oldColor);
        floodFill4 (x-1, y, fillColor, oldColor);
        floodFill4 (x, y+1, fillColor, oldColor);
        floodFill4 (x, y-1, fillColor, oldColor);
    }
}
```

Flood Fill Algorithm	Boundary Fill Algorithm
Flood fill colors an entire area in an enclosed figure through interconnected pixels using a single color	Here area gets colored with pixels of a chosen color as boundary this giving the technique its name
So, Flood Fill is one in which all connected pixels of a selected color get replaced by a fill color.	Boundary Fill is very similar with the difference being the program stopping when a given color boundary is found.
A flood fill may use an unpredictable amount of memory to finish because it isn't known how many sub-fills will be spawned	Boundary fill is usually more complicated but it is a linear algorithm and doesn't require recursion
Time Consuming	It is less time Consuming

SCAN LINE POLYGON FILLING 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 color.
- 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 x = 24

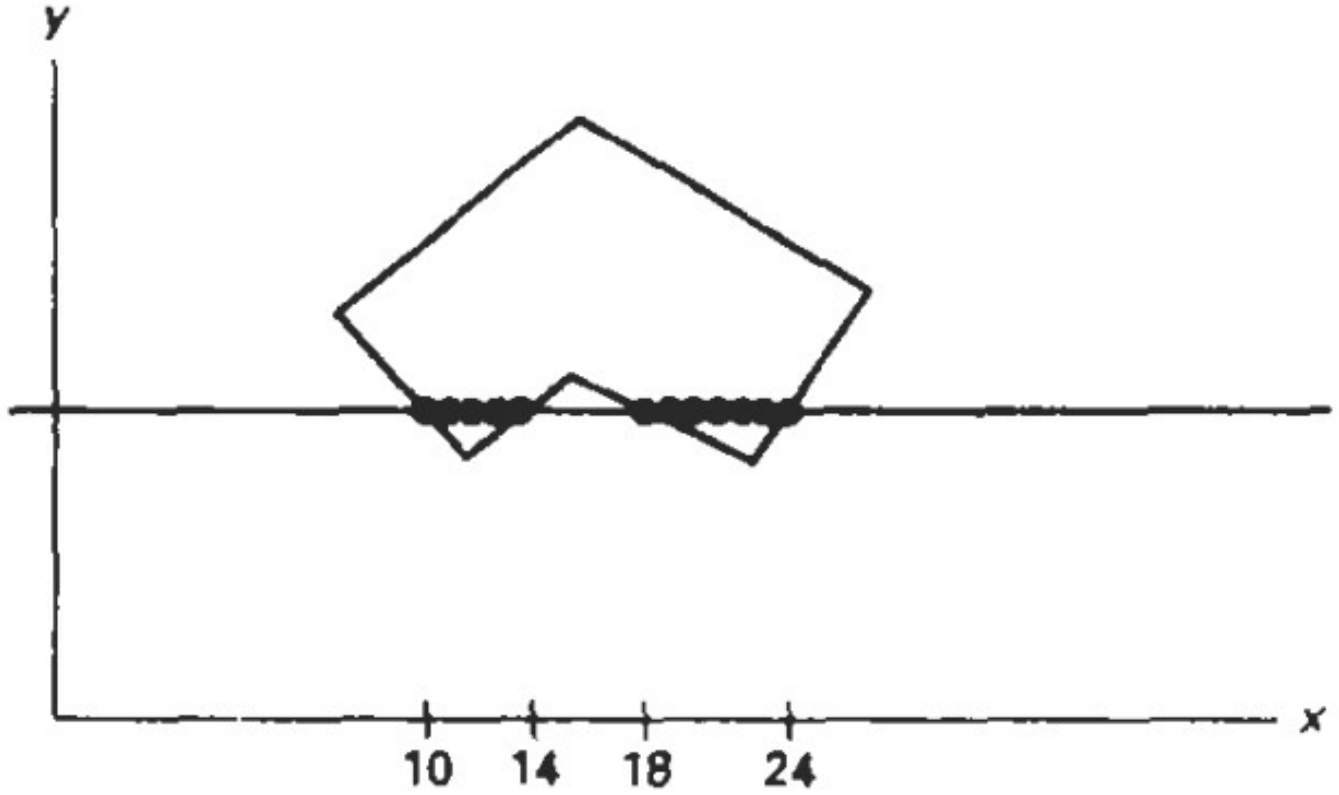


Figure 3-35
Interior pixels along a scan line passing through a polygon area.

- 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 3-36 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.
- Intersection points along scan line y' correctly identify the interior pixel spans.
- But with scan line y, we need to do some additional processing to determine the correct interior points

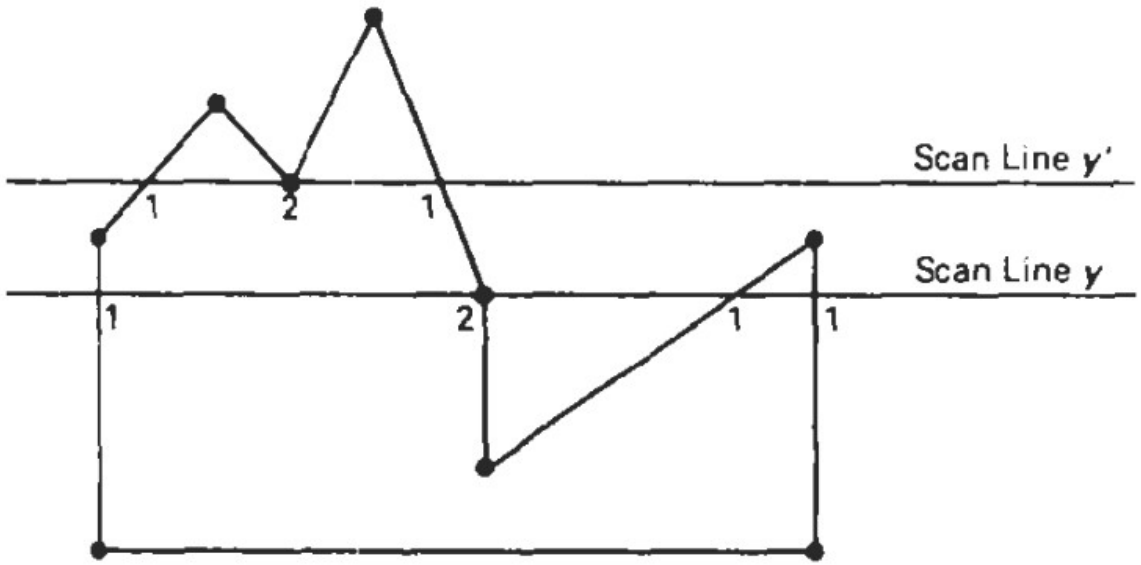


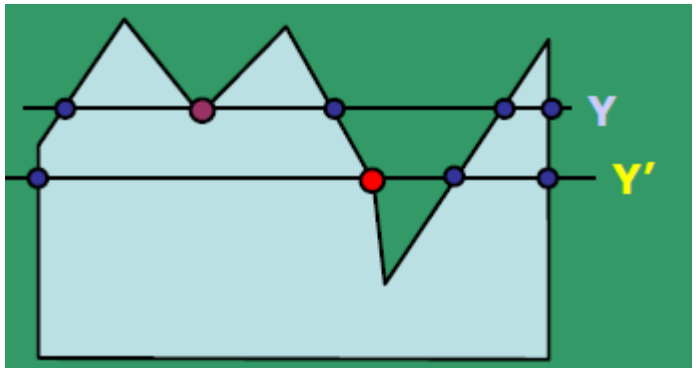
Figure 3-36
Intersection points along scan lines that intersect polygon vertices. Scan line y generates an odd number of intersections, but scan line y' generates an even number of intersections that can be paired to identify correctly the interior pixel spans.

- For scan line y, the two intersecting edges sharing a vertex are on opposite sides of the scan line
- Thus, the vertices that require additional processing are those that have connecting edges on opposite sides of the scan line.
- But for scan line y', the two intersecting edges are both above the scan line.

Special cases of polygon

- If both lines intersecting at the vertex are on the same side of the scan line, consider it as two points.
- If lines intersecting at the vertex are at opposite sides of the scan line, consider it as only one point.

- For Y, the edges at the vertex are on the same side of the scan line.
- Whereas for Y', the edges are on either/both sides of the vertex.
- In this case, we require additional processing.



- To identify the vertices are :

- Traverse along the polygon boundary clockwise (or counter-clockwise) and Observe the *relative change in Y-value* of the edges on either side of the vertex (i.e. as we move from one edge to another).

- Check the condition:

- If end-point Y values of two consecutive edges monotonically increase or decrease, count the middle vertex as a single intersection point for the scan line passing through it.

- Otherwise, 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 edge to split those vertices that should be counted as one intersection.
- process nonhorizontal edges around the polygon boundary in the order specified, either clockwise or counterclockwise.
- As we process each edge, we can check to determine whether that edge and the next nonhorizontal 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 3-37 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 1s decreased by 1, as in Fig. 3-37(a).
- When the endpoint y values are monotonically decreasing, as in Fig. 3-37(b),
- we decrease they coordinate of the upper endpoint of the edge following the current edge.

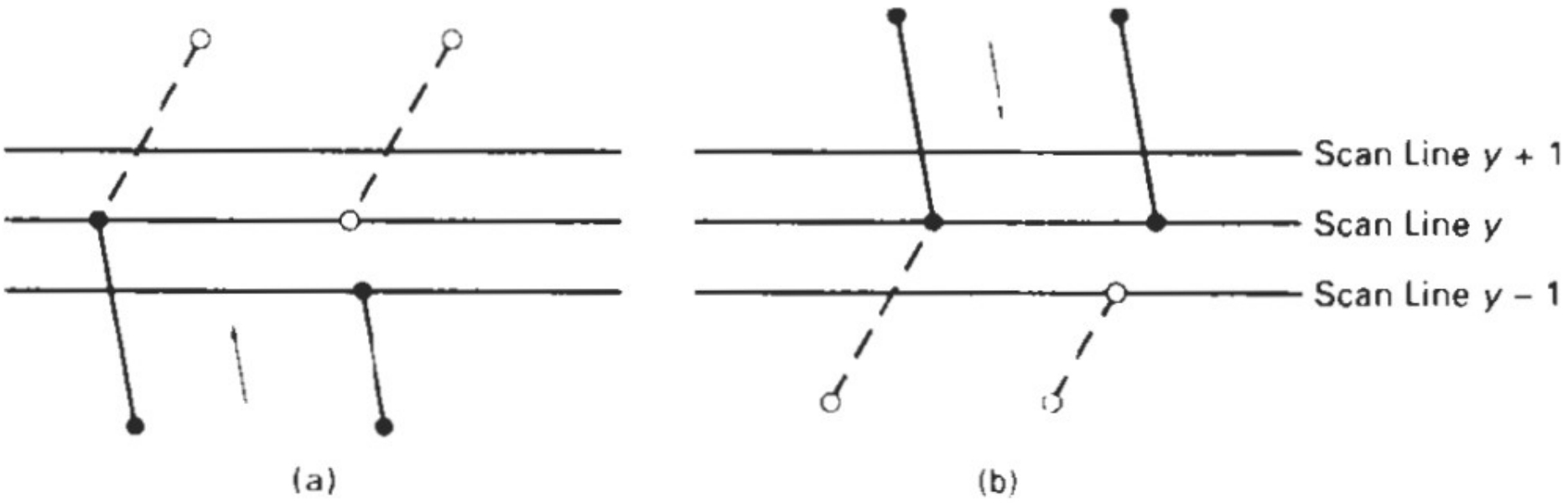


Figure 3-37
Adjusting endpoint y values for a polygon, as we process edges in order around the polygon perimeter. The edge currently being processed is indicated as a solid line. In (a), the y coordinate of the upper endpoint of the current edge is decreased by 1. In (b), the y coordinate of the upper endpoint of the next edge is decreased by 1.

- Calculations performed in scan-conversion and other graphics algorithms typically take advantage of various coherence properties of a scene that is to be displayed.

- 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, we can 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.

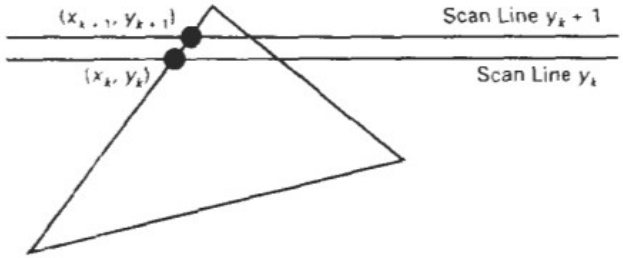


Figure 3-38
Two successive scan lines intersecting a polygon boundary.

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

$$m = \frac{y_{k+1} - y_k}{x_{k+1} - x_k} \tag{3-58}$$

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

$$y_{k+1} - y_k = 1 \tag{3-59}$$

the x-intersection value xi,, on the upper scan line can be determined from the x-intersection value xk 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

IMPLEMENTATION OF INTEGER OPERATION

- In a sequential fill algorithm, the increment of x values by the amount l/m along an edge can be accomplished with integer operations by recalling that thc slope m is the ratio of two integers:

$$m = \frac{\Delta y}{\Delta x}$$

- where dx and dy are the differences between the edge endpoint x and y coordinate values. Thus, incremental calculations of x intercepts along an edge for successive scan lines can be expressed as

$$x_{k+1} = x_k + \frac{\Delta x}{\Delta y}$$

- Using this equation we can perform integer evaluation of the x intercepts by initializing a counter to 0,
- Then incrementing the counter by the value of dx each time we move up to a new scan line.
- Whenever the counter value becomes equal to or greater than dy,
- we increment the current x intersection value by 1 and decrease the counter by the value dy.
- This procedure is equivalent to maintaining integer and fractional parts for x intercepts and incrementing the fractional part until we reach the next integer value.

- As an example of integer incrementing, suppose we have an edge with slope rn = 7/3.
- dx =3,dy=7
- At the initial scan line, we set the counter to 0 and the counter increment to 3.
- As we move up to the next three scan lines along this edge, the counter is successively assigned the values 3, 6, and 9

Scan line	**Counter**	**(x,y)**	**Description**
0	0	1,1	Initial points
1	0+3=3	1,2	Counter<dy thus x same y =y+1
2	3+3=6	1,3	Counter<dy thus x same y =y+1
3	6+3=9	2,4	Counter>dy thus x=x+1,y=y+1, counter=9-7=2
4	2	2,5	Counter<dy thus x same y =y+1

- We can round to the nearest pixel x-intersection value, instead of truncating to obtain integer positions, by modifying the edge-intersection algorithm so that the increment is compared to dy/2

- Initialize the counter to 0
- incrementing the counter with the value 2dx
- at each step and comparing the increment to dy
- When the increment is greater than or equal to dy, we increase the x value by 1 and decrement the counter by the value of 2dy.

Scan line	**Counter**	**(x,y)**	**Description**
0	0	1,1	Initial points
1	0+6=6	1,2	Counter<dy thus x same y =y+1
2	6+6=12	2,3	Counter>dy so x and y increases Counter=12-14=-2
3	-2+6=4	2,4	Counter<dy thus x same y =y+1
4	4+6=10	2,5	Counter>dy so x and y increases Counter=10-14=-4

- To efficiently perform a polygon fill, first store the polygon boundary in a sorted edge table that contains all the information necessary to process the scan lines efficiently.

- Only nonhorizontal edges are entered into the sorted edge table. As the edges are processed, also shorten certain edges to resolve the vertex intersection question.

- Each entry in the table for a particular scan line contains maximum y value for that edge, the x-intercept value (at the lower vertex) for the edge, and the inverse slope of the edge.

- For each scan line, the edges are in sorted order from left to nght.

- Figure 3-39 shows a polygon and the associated sorted edge table

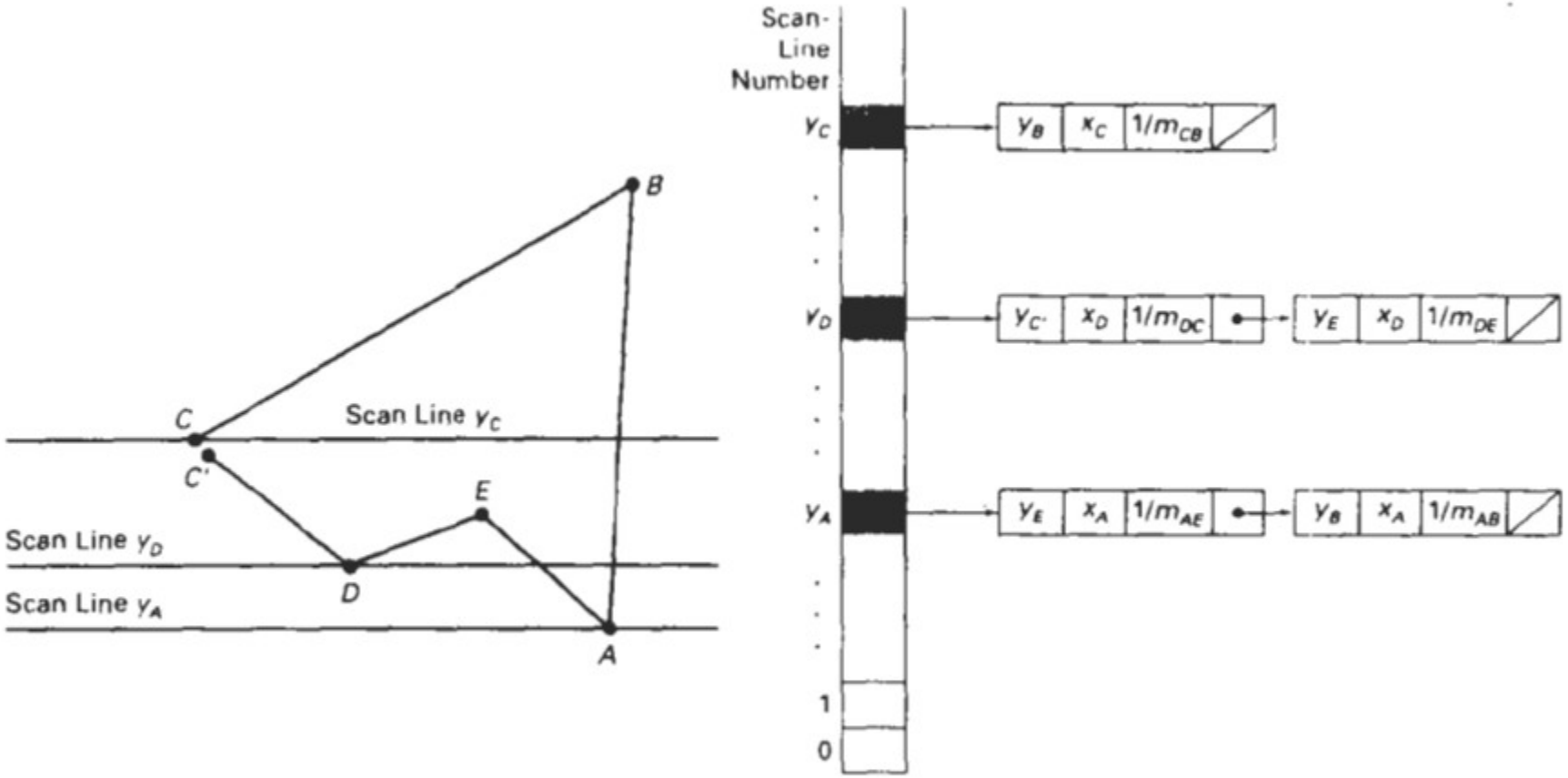


Figure 3-39
A polygon and its sorted edge table, with edge $\overline{DC}$ shortened by one unit in the y direction.

- Next, we process the scan lines from the bottom of the polygon to its top, Producing an active edge list for each scan line crossing the polygon boundaries.
- The active edge list for a scan line contains all edges crossed by that scan line, with iterative coherence calculations used to obtain the edge intersections

Definition

- Polygon is a closed figure with many vertices and edges (line segments), and at each vertex exactly two edges meet and no edge crosses the other.

Types of Polygon:

- Regular polygons
  - No. of egdes equal to no. of angles.
- Convex polygons
  - Line generated by taking two points in the polygon must lie within the polygon.

- Concave polygons
  - Line generated by taking two points in the polygon may lie outside the polygon.

Polygon Inside/Outside Test

- This test is required to identify whether a point is inside or outside the polygon. This test is mostly used for identify the points in hollow polgons.

Two types of methods are there for this test:

I. Even-Odd Method,
II. Winding Number Method.

1/2

27

Even-Odd method:

1. Draw a line from any position P to a distant point outside the coordinate extents of the object and counting the number of edge crossings along the scan line.

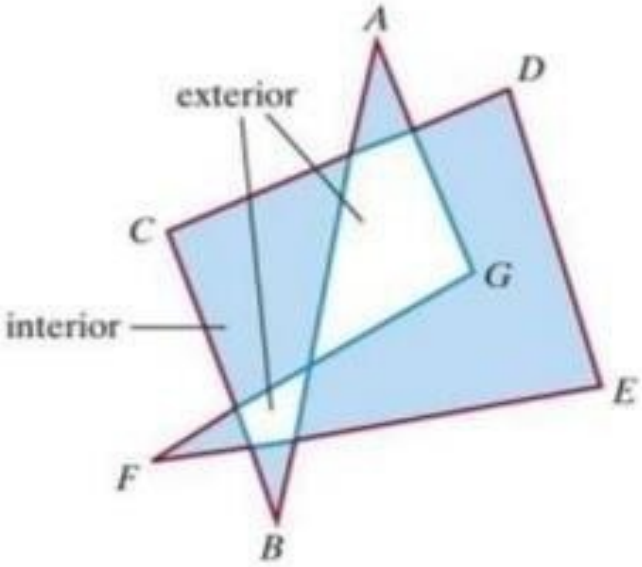
2. If the number of polygon edges crossed by this line is **odd** then

   P is an **interior** point.

   Else

   P is an **exterior** point

In some situations the logic described above doesn't hold, when a scan line passes through a vertex having two intersecting edges then if both the edges lies on the same side of the scan line the intersection is considered to be even.
Otherwise the intersection is considered to be odd i.e. if the two edges are on the opposite side of the scan line.



Odd-Even Rule

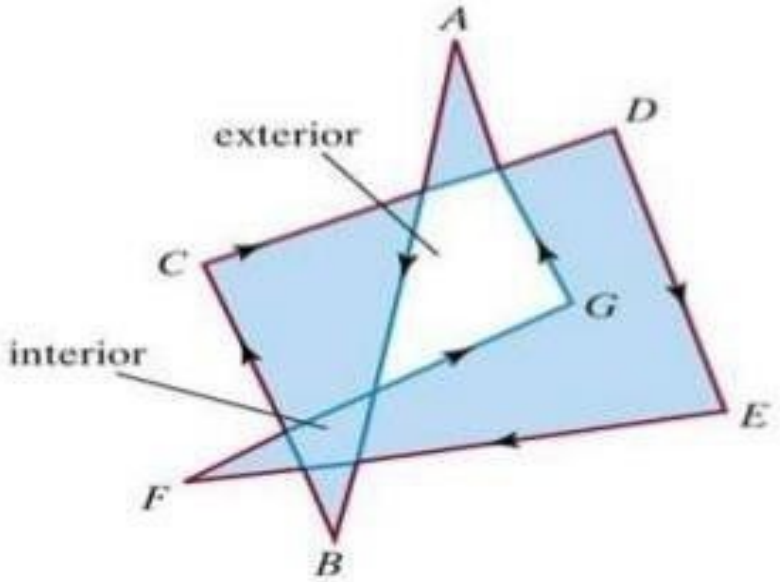
29

Nonzero Winding Number Rule :

- Another method of finding whether a point is inside or outside of the polygon. In this every point has a winding number, and the interior points of a two-dimensional object are defined to be those that have a nonzero value for the winding number.

1. Initializing the winding number to 0.
2. Imagine a line drawn from any position P to a distant point beyond the coordinate extents of the object.
3. Count the number of edges that cross the line in each direction. We add 1 to the winding number every time we intersect a polygon edge that crosses the line from right to left, and we subtract 1 every time we intersect an edge that crosses from left to right.

4. If the winding number is **nonzero**, then
   P is defined to be an **interior** point
   Else
   P is taken to be an **exterior** point.



Nonzero Winding-Number Rule

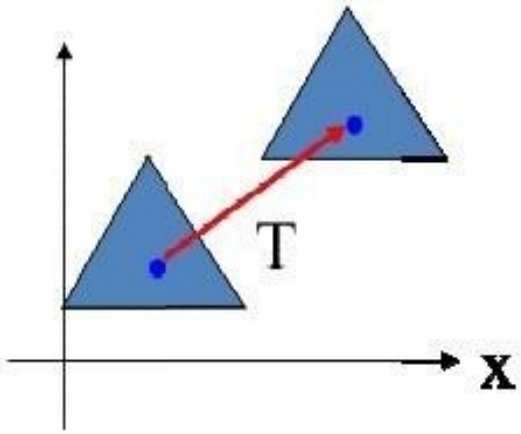
Two dimensional transformations

Basic geometric transformations:

- **Translation**
- **Rotation**
- **Scaling**

Translation

- A translation is applied to an object by repositioning it along a straight-line path from one coordinate location to another.
- We can translate a point in 2D by adding translation distances, (tx, ty) to the original coordinate (X, Y) to move the point to a new position (X', Y').



$$X' = X + tx$$

$$Y' = Y + ty$$

- The pair (tx, ty) is called the translation vector or shift vector.
- The above equations can also be represented using the column vectors.

$$P = \begin{bmatrix} X \\ Y \end{bmatrix} \qquad P' = \begin{bmatrix} X' \\ Y' \end{bmatrix} \qquad T = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

- So, the two-dimensional translation equations in the matrix form:

$$P' = P + T$$

- Polygons are translated by adding the translation vector to the coordinate position of each vertex and regenerating the polygon using the new set of vertex coordinates and the current attribute settings.

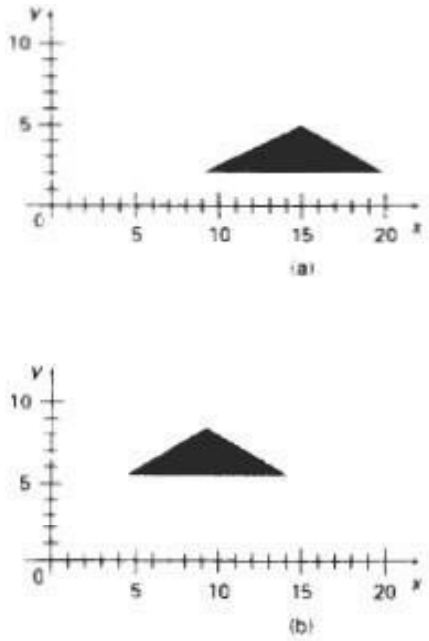


Figure 5-2
Moving a polygon from position (a) to position (b) with the translation vector (-5.50, 3.75).

- Similar methods are used to translate curved objects. To change the position of a circle or ellipse, we translate the center coordinates and redraw the figure in the new location.

Problems

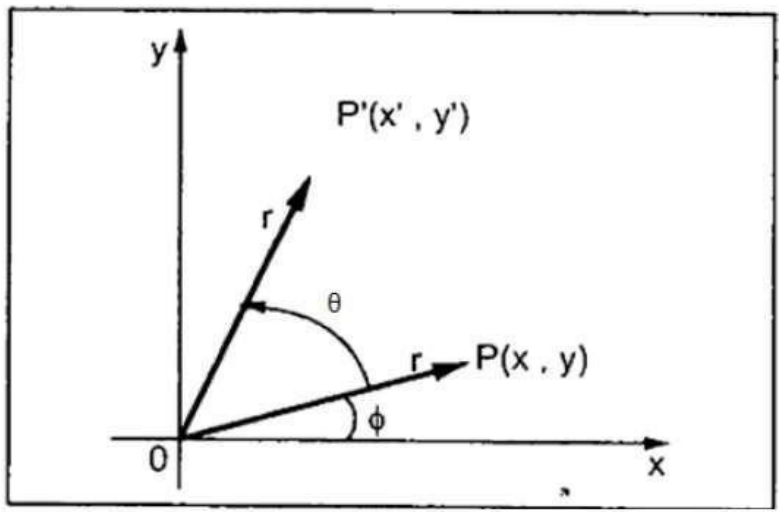
Two dimensional transformations

Basic geometric transformations:

- **Translation**
- **Rotation**
- **Scaling**

Rotation

- In rotation, we rotate the object at particular angle θ (theta) from its origin.
- To generate a rotation, we specify a rotation angle θ and the position (x_r, y_r) of the rotation point (or pivot point) about which the object is to be rotated



- Suppose we want to rotate point P at the angle θ. After rotating it to a new location, we will get a new point P′(X′,Y′).
- In this figure, r is the constant distance of the point from the origin, angle ϕ is the original angular position of the point from the horizontal, and θ is the rotation angle.
- Positive values for the rotation angle define counter - clockwise rotationsabout the pivot point and negative rotate objects in the clockwise direction.

$$x' = r\cos(\phi + \theta) = r\cos\phi\cos\theta - r\sin\phi\sin\theta$$
$$y' = r\sin(\phi + \theta) = r\cos\phi\sin\theta + r\sin\phi\cos\theta \tag{5-4}$$

The original coordinates of the point in polar coordinates are

$$x = r\cos\phi, \qquad y = r\sin\phi \tag{5-5}$$

Substituting expressions 5-5 into 5-4, we obtain the transformation equations for rotating a point at position (x, y) through an angle θ about the origin:

$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta \tag{5-6}$$

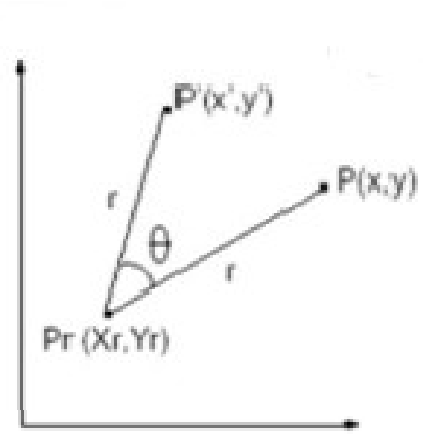
With the column-vector representations 5-2 for coordinate positions, we can write the rotation equations in the matrix form:

$$\mathbf{P}' = \mathbf{R} \cdot \mathbf{P} \tag{5-7}$$

where the rotation matrix is

$$\mathbf{R} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{5-8}$$

Rotation about a fixed reference point



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

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

- Rotations are rigid-body transformations that move objects without deformation.
- Every point on an object is rotated through the same angle.
- A **straight line** segment is rotated by applying the rotation equations to each of the **line endpoints** and redrawing the line between the new endpoint positions.
- **Polygons** are rotated by displacing each **vertex** through the specified rotation angle and regenerating the polygon using the new vertices.
- **Curved lines** are rotated by repositioning the **defining points** and redrawing the curves.
- A **circle** or an **ellipse** can be rotated about a noncentral axis by moving the **center** position through the arc that subtends the specified rotation angle.
- An **ellipse** can be rotated about its **center** coordinates by rotating the major and minor axes**.**

Two dimensional transformations

Basic geometric transformations:

- **Translation**
- **Rotation**
- **Scaling**

Scaling

- A scaling transformation alters the size of an object.
- This operation can be carried out for polygons by multiplying the coordinate values ***(x, y)*** of each vertex by scaling factors S_x

$$\mathbf{x' = x \,.\, S_x}$$

$$\mathbf{y' = y \,.\, S_y}$$

- Scaling factor S_x, scales objects in the x direction, while S_y scales in the y direction.

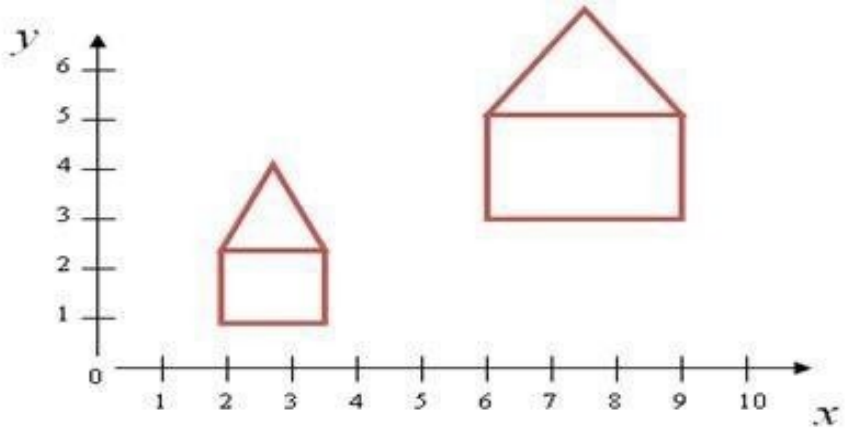
The above equations can also be represented in matrix form as below –

$$\begin{pmatrix} X' \\ Y' \end{pmatrix} = \begin{pmatrix} X \\ Y \end{pmatrix} \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix}$$

OR

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

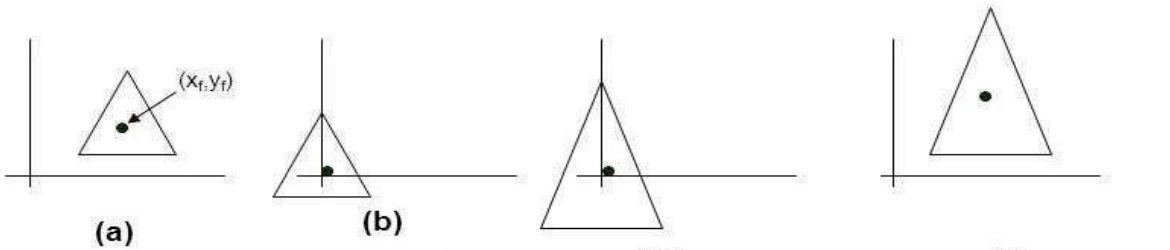
Where S is the scaling matrix. The scaling process is shown in the following figure.



- If S_x & S_y >1, the size of the object will increase. Also objects move away from origin
- If S_x & S_y <1, the size of the object is reduced. Also objects move towards origin
- If S_x & S_y =1, the size of the object will remain unchanged. Also there will be no change in location.
- If S_x & S_y have same values, it is called **uniform scaling**.
- If S_x & S_y have different values, it is called **differential scaling**.

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

- Translate object so that the fixed point coincides with the coordinate origin
- Scale the object with respect to the coordinate origin
- Use the inverse translation of step 1 to return the object to its original position



(a) Original Position of Object and Fixed point

(b) Translation of object so that fixed point (x_f, y_f) is at origin

(c) scaling was about origin

(d) Translation of the object so that the Fixed point is returned to position (x_f, y_f)

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

Homogeneous co-ordinates

- We can represent the point by 3 numbers instead of 2 numbers, which is called Homogenous Coordinate system.
- To express any two-dimensional transformation as a matrix multiplication, we represent each Cartesian coordinate position (x, y) with the homogeneous coordinate triple (x_h, y_h, h) where

$$x = \frac{x_h}{h} \qquad y = \frac{y_h}{h}$$

- 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 two dimensional position is thenrepresented with homogeneous coordinates *(x, y, 1)*.

- Expressing positions in homogeneous coordinates allows us to represent all geometric transformation equations as matrix multiplications.
- **Advantages of using homogenous coordinate:** We can perform all transformations using matrix/vector multiplications. This allows us to pre-multiply all the matrices together.

Translation

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

Rotation

- About the coordinate origin:

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

Scaling

- Scaling transformation relative to the coordinate origin:

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

COMPOSITE TRANSFORMATIONS

- 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.

Translation

- When two successive translations are applied to a point with

  translation vectors (tx1,ty1) and (tx2,ty2), then

$$\begin{bmatrix} 1 & 0 & t_{x_2} \\ 0 & 1 & t_{y_2} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & t_{x_1} \\ 0 & 1 & t_{y_1} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_{x_1} + t_{x_2} \\ 0 & 1 & t_{y_1} + t_{y_2} \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{P}' = \mathbf{T}(t_{x_2}, t_{y_2}) \cdot \mathbf{T}(t_{x_1}, t_{y_1}) \cdot \mathbf{P} = \mathbf{T}(t_{x_1} + t_{x_2}, t_{y_1} + t_{y_2}) \cdot \mathbf{P}$$

- This demonstrates that successive translations are additive in nature.

Rotations

- When two successive rotations are applied to a point .

we can write rotation matrix $R(\theta_1)$ as

$$R(\theta_1) = \begin{bmatrix} \cos\theta_1 & \sin\theta_1 \\ -\sin\theta_1 & \cos\theta_1 \end{bmatrix} \text{ and } R(\theta_2) = \begin{bmatrix} \cos\theta_2 & \sin\theta_2 \\ -\sin\theta_2 & \cos\theta_2 \end{bmatrix}$$

$$R(\theta_1) \cdot R(\theta_2) = \begin{bmatrix} \cos\theta_1 & \sin\theta_1 \\ -\sin\theta_1 & \cos\theta_1 \end{bmatrix} \times \begin{bmatrix} \cos\theta_2 & \sin\theta_2 \\ -\sin\theta_2 & \cos\theta_2 \end{bmatrix}$$

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

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

since,

$$\cos(\theta_1 + \theta_2) = \cos\theta_1 \cos\theta_2 - \sin\theta_1 \sin\theta_2$$

$$\sin(\theta_1 + \theta_2) = \cos\theta_1 \sin\theta_2 + \sin\theta_1 \cos\theta_2$$

- This demonstrates that successive rotations are additive in nature.

Scaling

- When two successive scaling are applied to a point, with vectors (S_{x1}, S_{y1}) and (S_{x2}, S_{y2})

$$\begin{vmatrix} S_{x2} & 0 & 0 \\ 0 & S_{y2} & 0 \\ 0 & 0 & 1 \end{vmatrix} \cdot \begin{vmatrix} S_{x1} & 0 & 0 \\ 0 & S_{y1} & 0 \\ 0 & 0 & 1 \end{vmatrix} = \begin{vmatrix} S_{x1} \cdot S_{x2} & 0 & 0 \\ 0 & S_{y1} \cdot S_{y2} & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

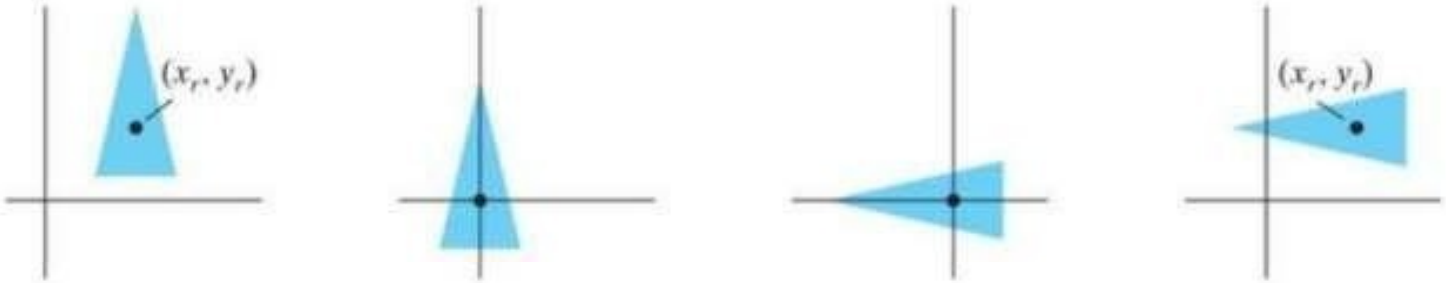
$$S(S_{x2}, S_{y2}) \cdot S(S_{x1}, S_{y1}) = S(S_{x1} \cdot S_{x2}, S_{y1} \cdot S_{y2})$$

- This demonstrates that successive scalings are multiplicative in nature.

Rotation with respect to a fixed reference point

- We can generate rotations about any selected pivot point (xr, yr) by performing the following sequence of translate- rotate-translate operations:

1. Translate the object so that the pivot-point position is moved to the coordinate origin.
2. Rotate the object about the coordinate origin.
3. Translate the object so that the pivot point is returned to its original position.



(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 the Pivot Point is Returned to Position (x_r, y_r)

Note the order of operations:

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

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

$$\mathbf{T}(x_r, y_r) \cdot \mathbf{R}(\theta) \cdot \mathbf{T}(-x_r, -y_r) = \mathbf{R}(x_r, y_r, \theta)$$

Scaling with respect to a fixed reference point

Transformation sequence to produce scaling with respect to a selected fixed position (xf, yf) using a scaling function that can only scale relative to the coordinate origin.

1. Translate object so that the fixed point coincides with the coordinate origin.
2. Scale the object with respect to the coordinate origin.
3. Use the inverse translation of step 1 to return the object to its original position.

(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 the Fixed Point is Returned to Position (x_f, y_f)

- Concatenating the matrices for these three operations produces the required scaling matrix:

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

$$\mathbf{T}(x_f, y_f) \cdot \mathbf{S}(s_x, s_y) \cdot \mathbf{T}(-x_f, -y_f) = \mathbf{S}(x_f, y_f, s_x, s_y)$$

Matrix concatenation properties

- Matrix multiplication is associative

$$\mathbf{M_3.M_2.M_1 = (M_3.M_2).M_1 = M_3.(M_2.M_1)}$$

- Multiplication is not commutative.
  - Unless the sequence of transformations are all of the same kind.
  - $\mathbf{M_2.M_1}$ not equal to $\mathbf{M_1.M_2}$ in general.
- This means that if we want to translate and rotate an object, we must be careful about the order in which the composite matrix is evaluated

- For some special cases, such as a sequence of transformations all of the same kind, the multiplication of transformation

matrices is commutative.

  - Two successive rotations could be performed in either

    order and the final position would be the same.

  - Two successive translations or two successive scaling

    are commutative.

  - Another commutative pair of operations is rotation and

    uniform scaling ($S_x = S_y$).

OTHER TRANSFORMATIONS

- **Reflection**
- **Shear**

REFLECTION

- A reflection is a transformation that produces a mirror image of an object.
- In other words, we can say that it is a rotation operation with 180° about the reflection axis.
- In reflection transformation, the size of the object does not change.

Reflection about the x axis (y = 0)

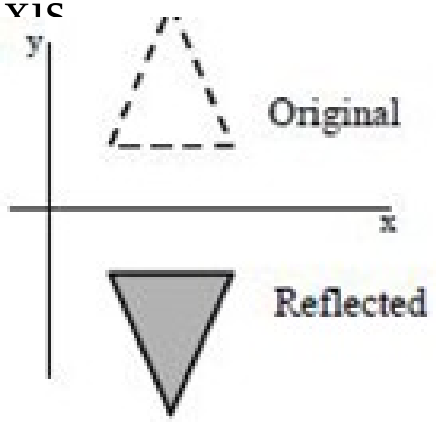
- Change the sign of y coordinate but x coordinate remains

  same.

$$x' = x \quad y' = -y$$

- Equivalent to 180° rotation about x axis

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



Reflection about the y axis (x = 0)

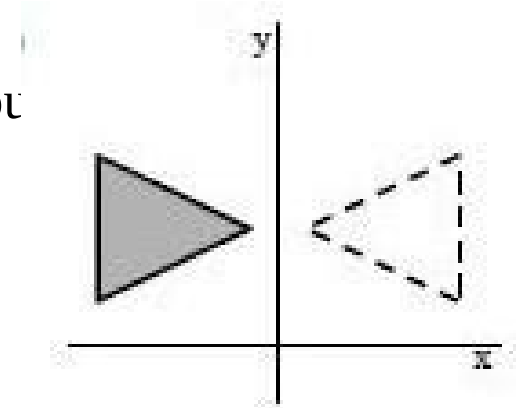
- Change the sign of x coordinate but y coordinate remains same.

$$x' = -x$$

$$y' = y$$

- Equivalent to 180 – rotation abou

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



Reflection relative to the coordinate origin

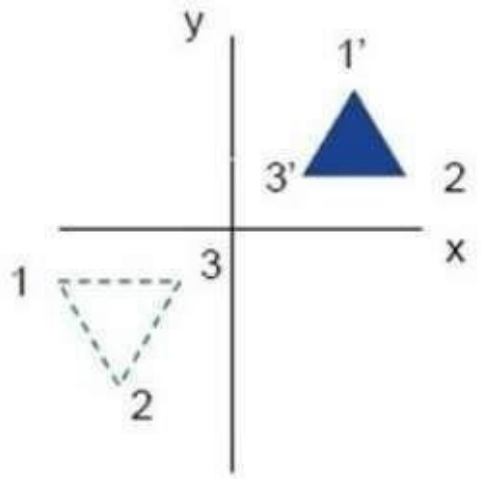
- Reflection relative to an axis that is perpendicular to the xy plane and that passes through the coordinate origin.
- Change the sign of both x coordinate and y coordinate.

$$x' = -x$$

$$y' = -y$$

- Equivalent to 180° rotation about o

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



Reflection along an axis passing through origin and perpendicular to xy plane

We flip both the x and y coordinates of a point by reflecting relative to an axis that is perpendicular to the xy plane and that passes through the coordinate origin.

this transformation, referred to as a reflection relative to the coordinate origin, has the matrix representation:

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

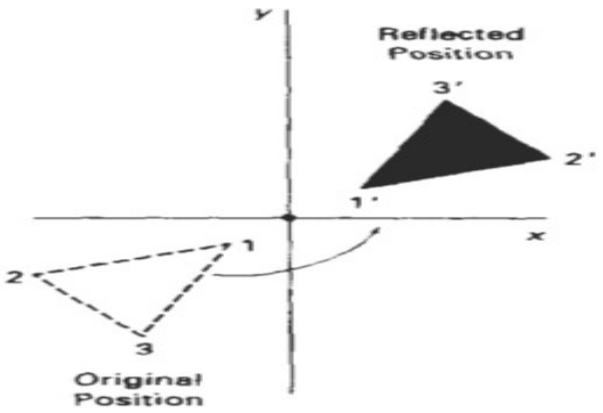


Figure 5-18
Reflection of an object relative to an axis perpendicular to the *xy* plane and passing through the coordinate origin.

Reflection about arbitrary fixed Point

This reflection is the same as a 180" rotation in the xy plane using the reflection point as the pivot point.

If we chose the reflection axis as the diagonal line y = x the reflection matrix is

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

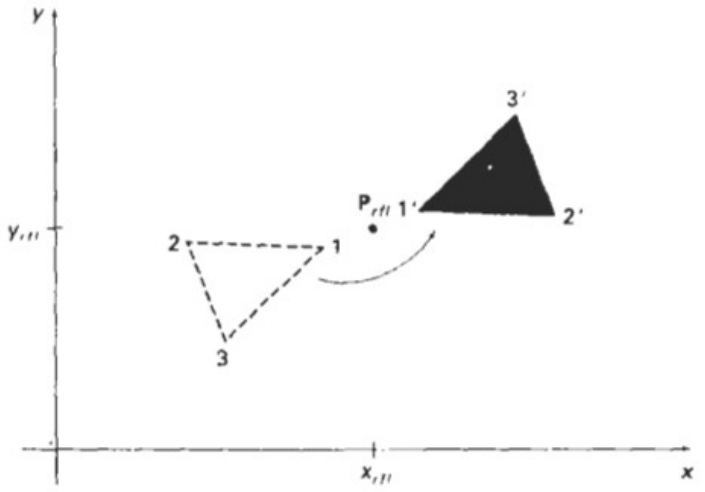


Figure 5-19
Reflection of an object relative to an axis perpendicular to the xy plane and passing through point $\mathbf{P}_{ref}$.

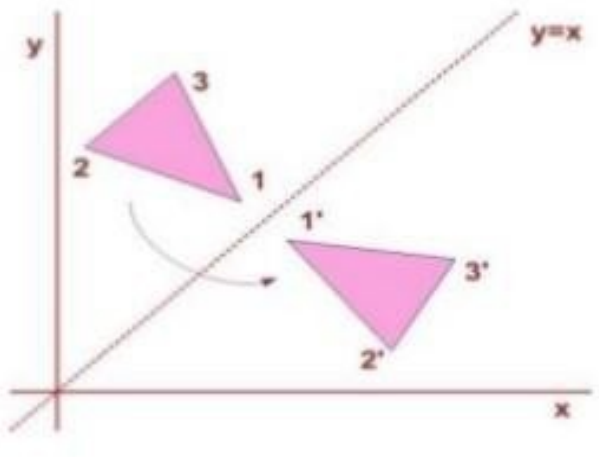
Reflection with respect to diagonal line , line y = x

¬ can be obtained by performing following sequence of

transformations:

1. Clock wise rotation with 45 ≤
2. Reflection with respect to x axis
3. Inverse rotation with an angle

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



Reflection about the diagonal y = -x

- can be obtained by performing following sequence of transformations:

1. Clockwise rotation by 45°
2. Reflection about the y axis
3. Counterclockwise rotation by 45°

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

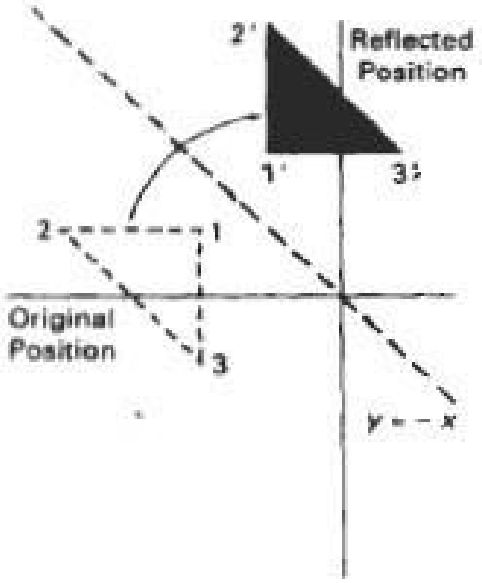


Figure 5-22
Reflection with respect to the line $y = -x$.

SHEAR

A transformation that distorts the **shape** of an object such that the transformed shape appears as if the object were composed of internal layers that had been caused to slide over each other is called shear

There are two shear transformations **X-Shear** and **Y-Shear**.

One shifts X coordinates values and other shifts Y coordinate

- However, in both the cases only one coordinate changes its coordinates and other preserves its values.

- Shearing is also termed as **Skewing**.

X direction shear:

- An x-direction shear relative to the x axis is produced with the

transformation matrix

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

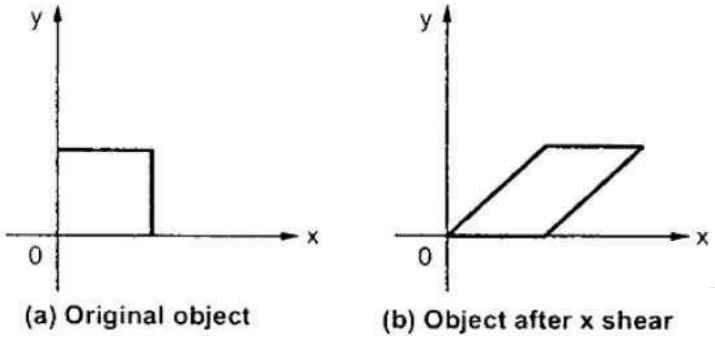
with coordinate positions trans for m ed as

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

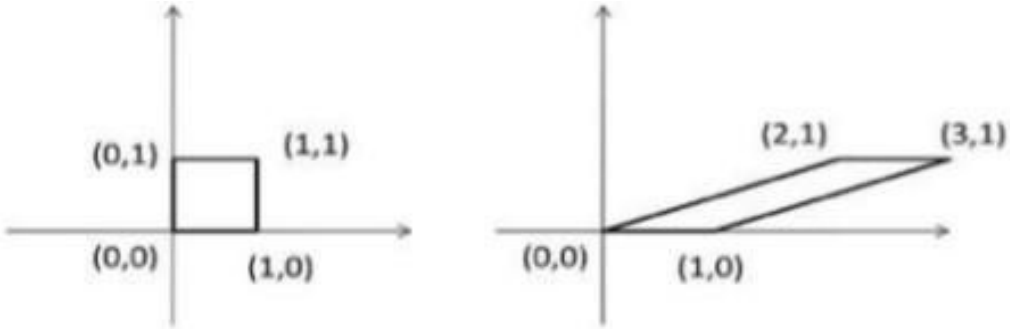
$$y' = y$$

- Negative values for sh_x shift coor dinate positi ons to the left.

- The X-Shear preserves the Y coordinate and changes are made to X coordinates, which causes the vertical lines to tilt right or left as shown in below figure.



- A unit square (a) is converted to a parallelogram (b) using the *x direction* shear matrix with $sh_x = 2$.



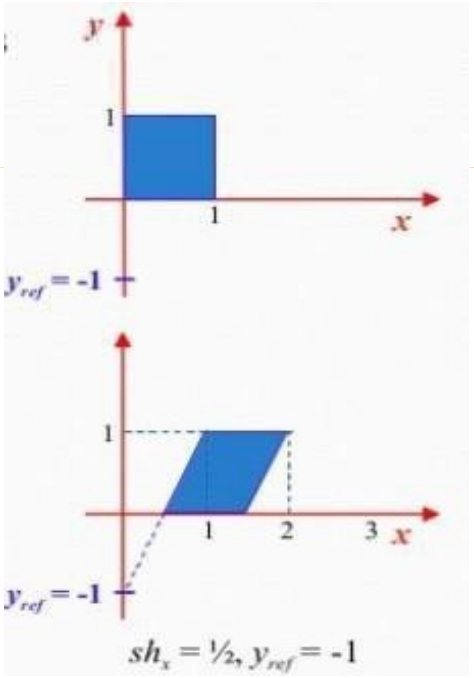
x-direction shears relative to other reference lines

$$\begin{pmatrix} 1 & sh_x & -sh_x.y_{ref} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

with coordinate positions transformed as

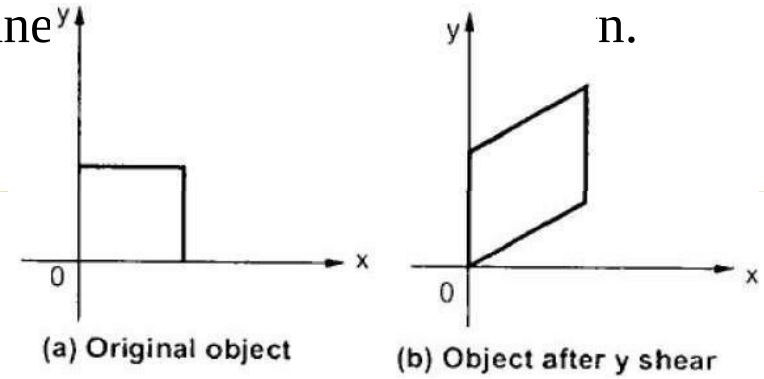
$$x' = x + sh_x(y - y_{ref})$$
$$y' = y$$

- An example of this shearing transformation for a shear parameter value of 1/2 relative to the line $y_{ref} = -1$.



Y-Shear

- The Y-Shear preserves the X coordinates and changes the Y coordinates which causes the horizontal lines to transform into line n.



- The Y-Shear can be represented in matrix form as:

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

$$x' = x$$

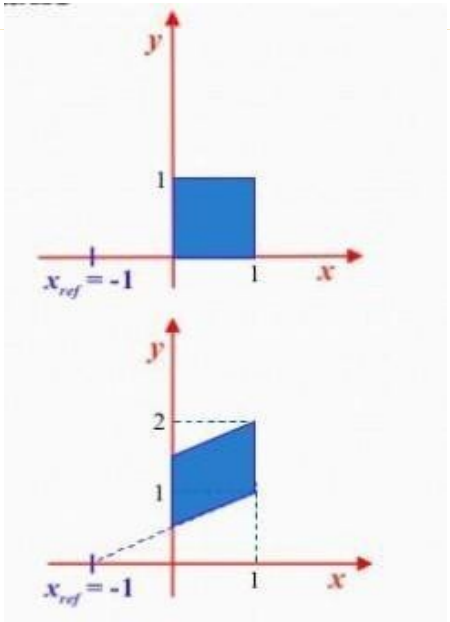
$$\begin{pmatrix} 1 & 0 & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- A y-direction shear relative to the line $x = x_{ref}$ is generated with the transformation matrix

$$\begin{pmatrix} 1 & 0 & 0 \\ sh_y & 1 & -sh_y.x_{ref} \\ 0 & 0 & 1 \end{pmatrix}$$

which generates transformed coordinate

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



Exampl e:

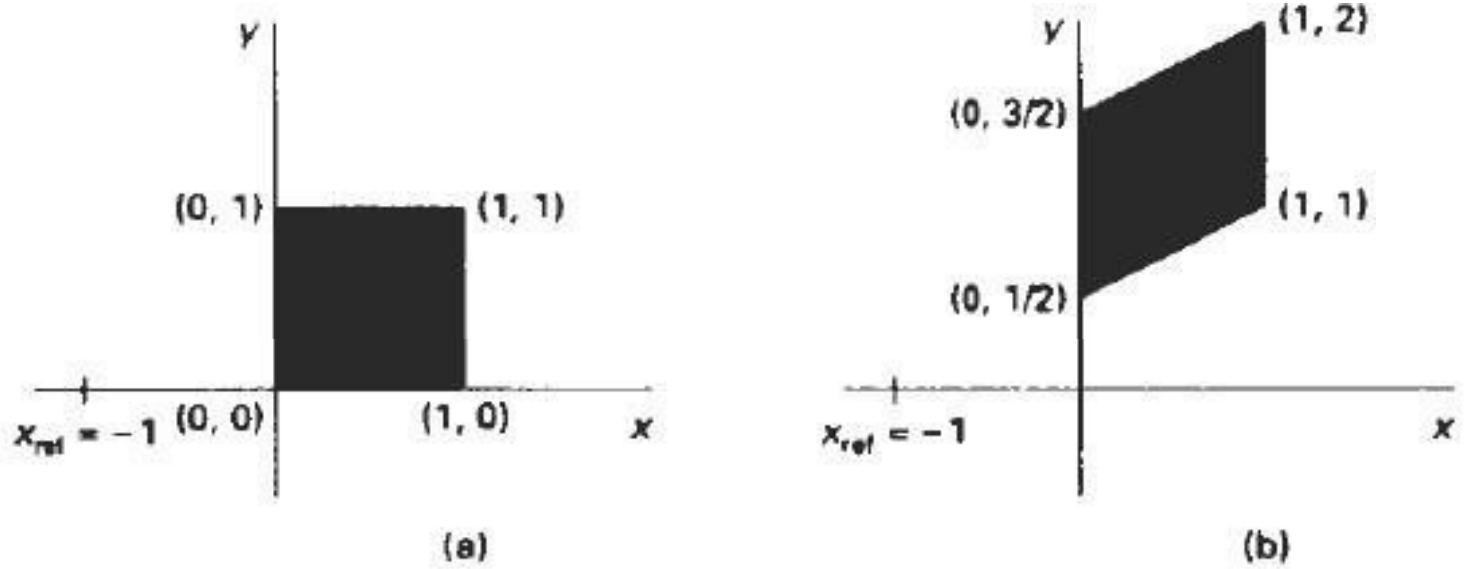


Figure 5-25
A unit square (a) is turned into a shifted parallelogram (b) with parameter values $sh_y = 1/2$ and $x_{ref} = -1$ in the y-direction using shearing transformation 5-57.

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

Solution:

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

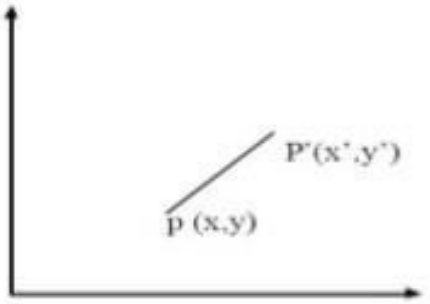
$T_x = 3$

$T_y = 3$

$X' = x + t_x$

$Y' = y + t_y$

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



- 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:

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

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

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

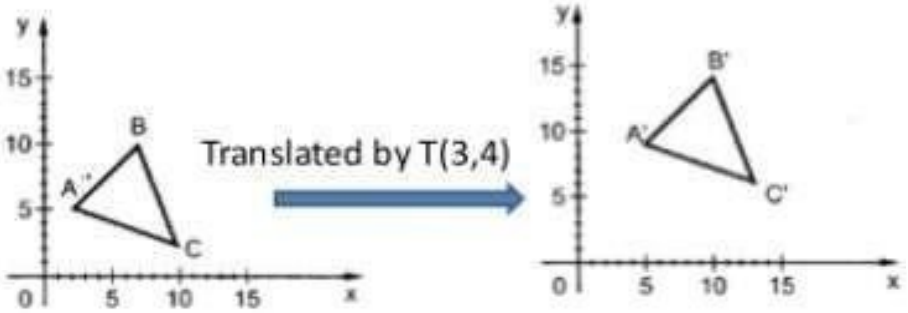


Fig: Before translation

Fig: After translation

- **Example:** A point (4,3) is rotated counterclockwise by an angle 45 degree. Find the rotation matrix and the resultant point.
- **Solution:**

$$R = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} = \begin{bmatrix} \cos 45^\circ & \sin 45^\circ \\ -\sin 45^\circ & \cos 45^\circ \end{bmatrix}$$

$$= \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ -1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix}$$

$$\therefore \quad P' = [4 \;\; 3] \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ -1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix}$$

$$= \left[4/\sqrt{2} - 3/\sqrt{2} \quad 4/\sqrt{2} + 3/\sqrt{2}\right]$$

$$= \left[1/\sqrt{2} \quad 7/\sqrt{2}\right]$$

- **Example1:** scale the polygon with coordinates A(2,5), B(7,10) and C(10,2) by two units in x-direction and two units in y-direction.

Sol. : Here $S_x = 2$ and $S_y = 2$. Therefore, transformation matrix is given as

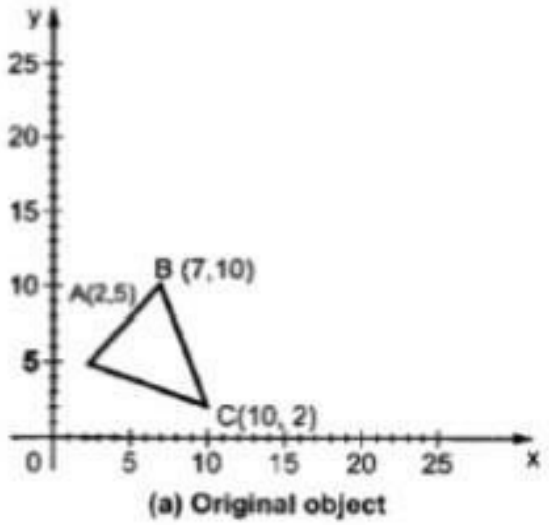
$$S = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

The object matrix is :

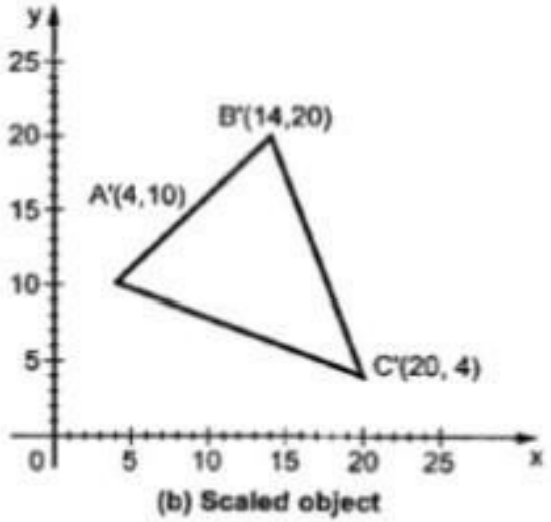
$$\begin{matrix} & x & y \\ A & 2 & 5 \\ B & 7 & 10 \\ C & 10 & 2 \end{matrix}$$

$$\therefore \begin{matrix} A' \\ B' \\ C' \end{bmatrix} \begin{matrix} x_1' & y_1' \\ x_2' & y_2' \\ x_3' & y_3' \end{matrix} = \begin{bmatrix} 2 & 5 \\ 7 & 10 \\ 10 & 2 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

$$= \begin{bmatrix} 4 & 10 \\ 14 & 20 \\ 20 & 4 \end{bmatrix}$$



(a) Original object



(b) Scaled object

- 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)

Answers:

i. Rotation about the origin

Given:

Triangle A(0,0), B(10,10) and C(50,20)

Rotation θ = 45°

$P' = R * P$

$$P' = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} Ax & Bx & Cx \\ Ay & By & Cy \end{bmatrix}$$

$$P' = \begin{bmatrix} \cos 45 & -\sin 45 \\ \sin 45 & \cos 45 \end{bmatrix} \begin{bmatrix} 0 & 10 & 50 \\ 0 & 10 & 20 \end{bmatrix}$$

$$P' = \begin{bmatrix} 0.707 & -0.707 \\ 0.707 & 0.707 \end{bmatrix} \begin{bmatrix} 0 & 10 & 50 \\ 0 & 10 & 20 \end{bmatrix}$$

$$P' = \begin{bmatrix} 0 & 0 & 21.21 \\ 0 & 14.14 & 49.49 \end{bmatrix}$$

After rotation about origin, the new points of triangle are

A(0,0), B(0,14), C(21,49)

Rotation about arbitrary point

Arbitrary Point (-10,-10)

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

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

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

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

$$P' = \begin{bmatrix} \cos 45 & -\sin 45 & -10(1-\cos 45) - 10\sin 45 \\ 0 & 14.14 & 49.49 \end{bmatrix} \begin{bmatrix} 0 & 10 & 50 \\ 0 & 10 & 20 \\ 0 & 0 & 1 \end{bmatrix}$$

$$P' = \begin{bmatrix} \cos 45 & -\sin 45 & -10(1-\cos 45) - 10\sin 45 \\ \sin 45 & \cos 45 & -10(1-\cos 45) + 10\sin 45 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 10 & 50 \\ 0 & 10 & 20 \\ 0 & 0 & 1 \end{bmatrix}$$

$$P' = \begin{bmatrix} 0.707 & -0.707 & -10(1-0.707) - 10 * 0.707 \\ 0.707 & 0.707 & -10(1-0.707) + 10 * 0.707 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 10 & 50 \\ 0 & 10 & 20 \\ 0 & 0 & 1 \end{bmatrix}$$

$$P' = \begin{bmatrix} 0.707 & -0.707 & -10 \\ 0.707 & 0.707 & 4.14 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 10 & 50 \\ 0 & 10 & 20 \\ 0 & 0 & 1 \end{bmatrix}$$

$$P' = \begin{bmatrix} 0 & 0 & 11.21 \\ 0 & 14.14 & 53.63 \\ 0 & 0 & 1 \end{bmatrix}$$

- After rotation about arbitrary point, the new points are A(0,0), B(0,14), C(11,54)

- 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.

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

$$P' = \begin{bmatrix} 2 & 0 & 6(1-2) \\ 0 & 3 & 0(1-3) \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 6 & 6 \\ 0 & 5 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$P' = \begin{bmatrix} 0 & 12 & 6 \\ 0 & 15 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The new coordinates after scaling in A(0,0), B (12, 15), C (6, 0)

3D transformations

- Methods for geometric transformations and object modeling in three dimensions are
- extended from two-dimensional methods by including considerations for the z coordinate.

TRANSLATION

In a three-dimensional homogeneous coordinate representation, a point is translated (Fig. 11-1) from position $\mathbf{P} = (x, y, z)$ to position $\mathbf{P}' = (x', y', z')$ with the matrix operation

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{11-1}$$

or

$$\mathbf{P}' = \mathbf{T} \cdot \mathbf{P} \tag{11-2}$$

Parameters t_x, t_y, *and* t_z, specifying translation distances for the coordinate directions x, y, and z, are assigned any real values. The matrix representation in Eq. 11-1 is equivalent to the three equations

$$x' = x + t_x, \qquad y' = y + t_y, \qquad z' = z + t_z \tag{11-3}$$

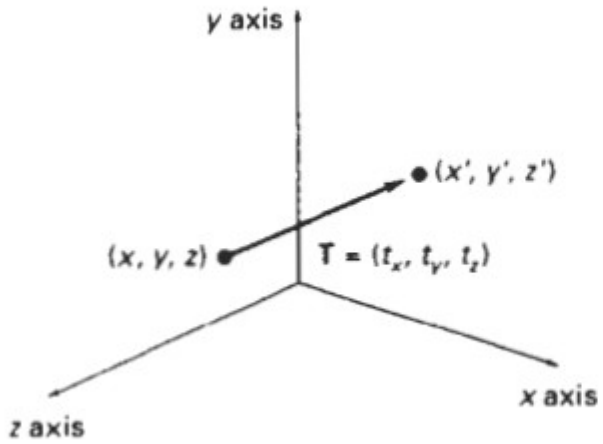


Figure 11-1
Translating a point with translation vector $\mathbf{T} = (t_x, t_y, t_z)$.

Rotation

To generate a rotation transformation for an object, we require an axis of rotation (about which the object is to be rotated) and the amount of angular rotation.

A three-dimensional rotation can be specified around any line in space.

positive rotation angles produce counterclockwise rotations about a coordinate axis, if we are looking along the positive half of the axis toward the coordinate origin and negative rotation angles produce clockwise rotation

Rotation along Z axis

$$
\begin{aligned}
x' &= x\cos\theta - y\sin\theta \\
y' &= x\sin\theta + y\cos\theta \\
z' &= z
\end{aligned}
\tag{11-4}
$$

Parameter θ specifies the rotation angle. In homogeneous coordinate form, the three-dimensional z-axis rotation equations are expressed as

$$
\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{11-5}
$$

$$
\mathbf{P}' = \mathbf{R}_z(\theta) \cdot \mathbf{P}
$$

Rotation along x axis

x-axis rotation:

$$\begin{aligned} y' &= y\cos\theta - z\sin\theta \\ z' &= y\sin\theta + z\cos\theta \\ x' &= x \end{aligned} \tag{11-8}$$

which can be written in the homogeneous coordinate form

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{11-9}$$

$$\mathbf{P}' = \mathbf{R}_x(\theta) \cdot \mathbf{P}$$

Rotation along Y axis

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

The matrix representation for y-axis rotation is

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

or

$$\mathbf{P}' = \mathbf{R}_y(\theta) \cdot \mathbf{P}$$

In the special case where an object is to be rotated about an axis that is parallel to one of the coordinate axes,.

we can attain the desired rotation with the following transformation sequence

1. Translate the object so that the rotation axis coincides with the parallel coordinate axis.
2. Perform the specified rotation about that axis.
3. Translate the object so that the rotation axis is moved back to its original position.

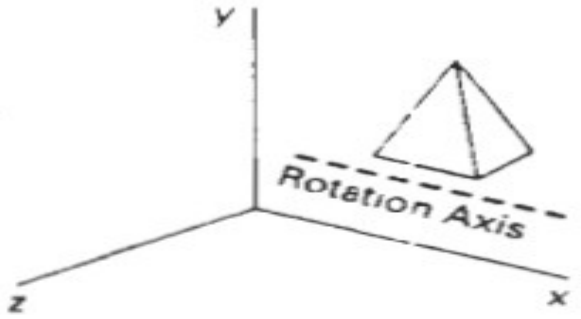
The steps in this sequence are illustrated in Fig. 11-8. Any coordinate position **P** on the object in this figure is transformed with the sequence shown as

$$\mathbf{P}' = \mathbf{T}^{-1} \cdot \mathbf{R}_x(\theta) \cdot \mathbf{T} \cdot \mathbf{P}$$

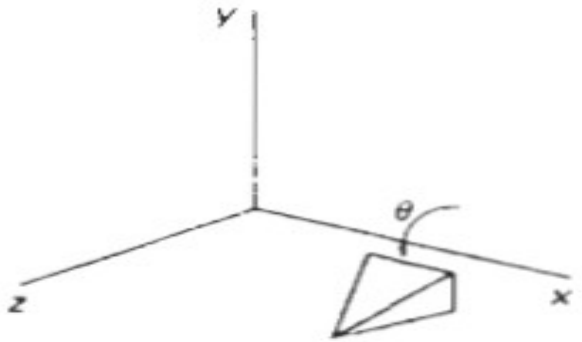
where the composite matrix for the transformation is

$$\mathbf{R}(\theta) = \mathbf{T}^{-1} \cdot \mathbf{R}_x(\theta) \cdot \mathbf{T}$$

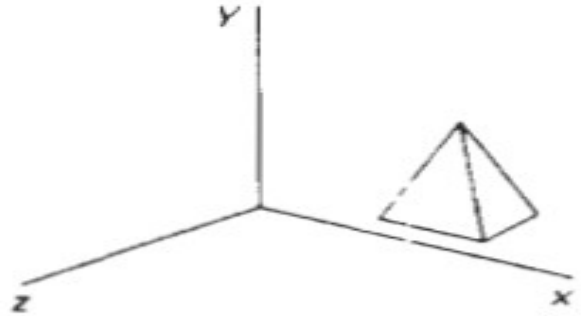
which is of the same form as the two-dimensional transformation sequence for rotation about an arbitrary pivot point.



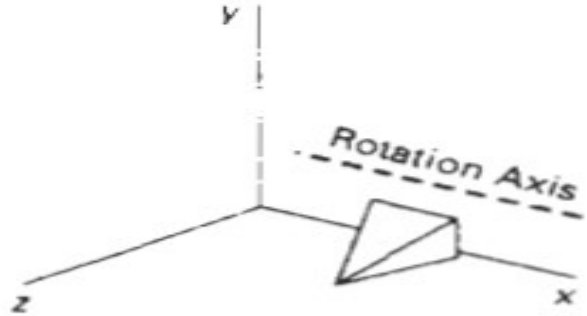
(a)
Original Position of Object



(c)
Rotate Object Through Angle θ



(b)
Translate Rotation Axis onto x Axis



(d)
Translate Rotation
Axis to Original Position

Figure 11-8
Sequence of transformations for rotating an object about an axis that is parallel to the x axis

When an object is to be rotated about an axis that is not parallel to one of the coordinate axes, we need to perform some additional transformations

1. Translate the object so that the rotation axis passes through the coordinate origin.
2. Rotate the object so that the axis of rotation coincides with one of the coordinate axes.
3. Perform the specified rotation about that coordinate axis.
4. Apply inverse rotations to bring the rotation axis back to its original orientation.
5. Apply the inverse translation to bring the rotation axis back to its original position.

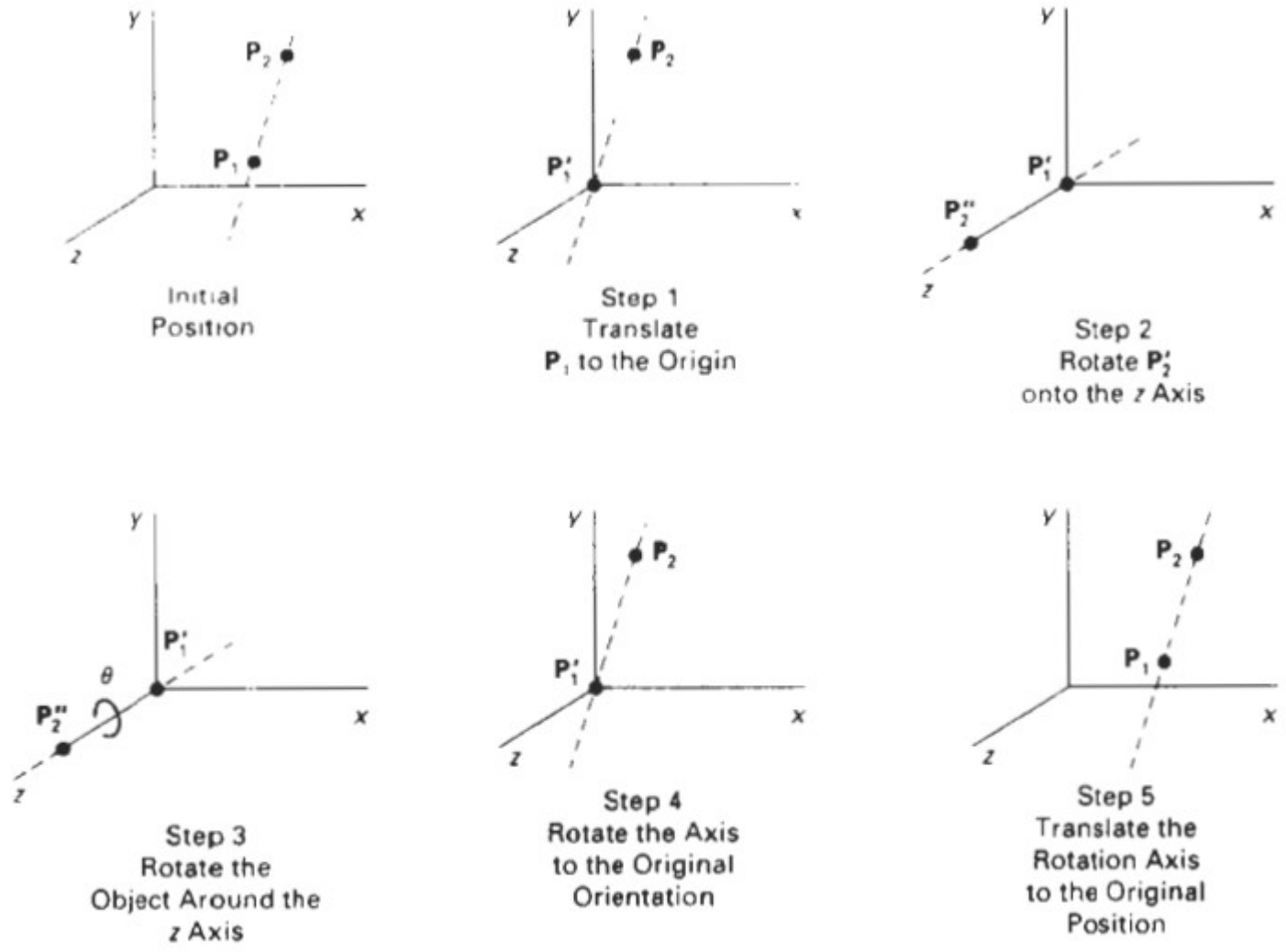


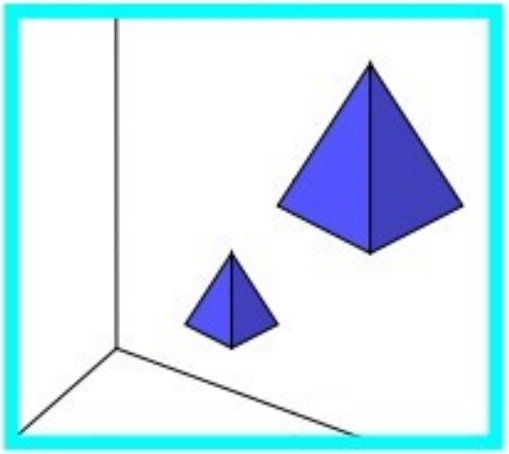
Figure 11-9
Five transformation steps for obtaining a composite matrix for rotation about an arbitrary axis, with the rotation axis projected onto the z axis.

3D Scaling

Scaling about origin: Changes the size of the object and repositions the object relative to the coordinate origin

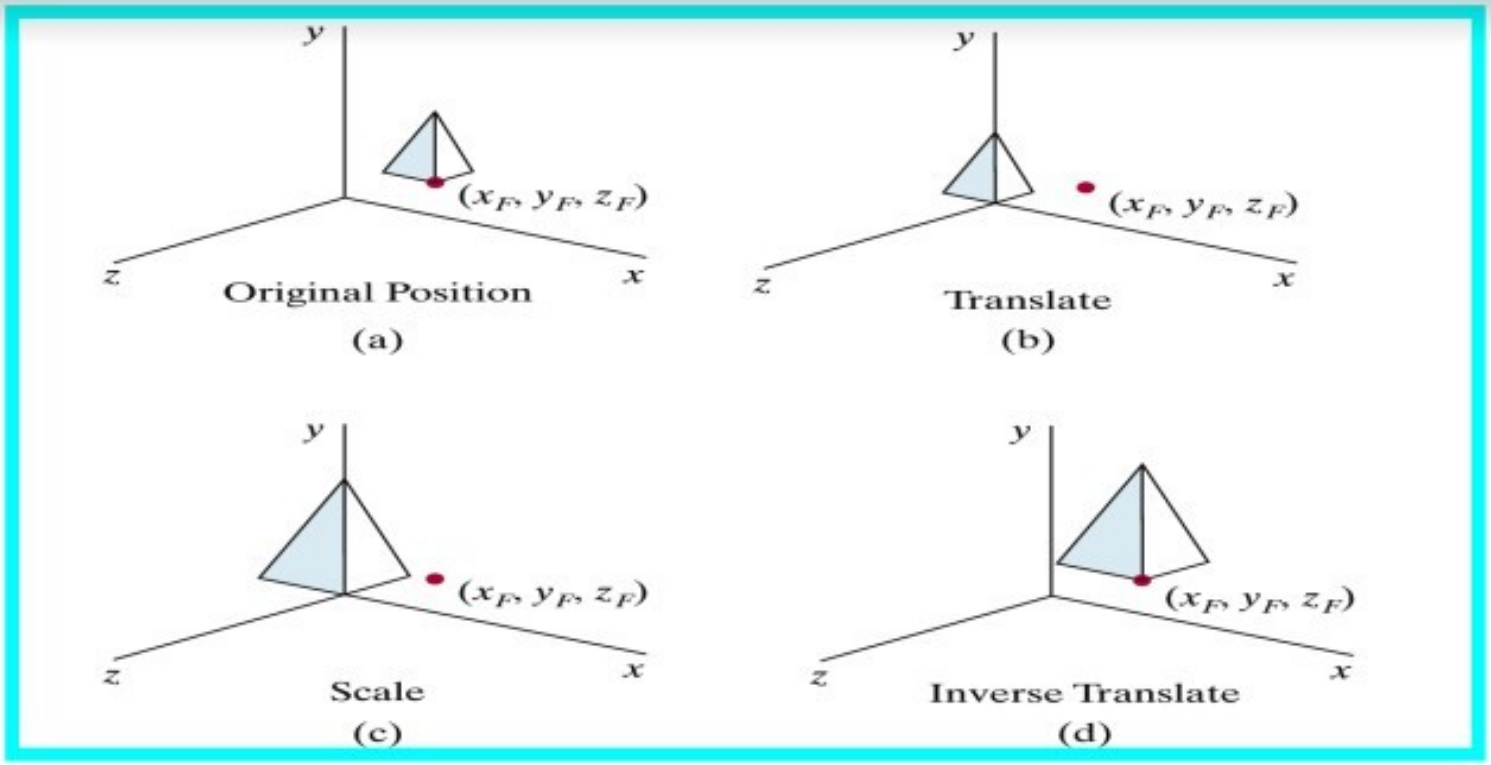
$$x' = x \cdot sx$$
$$y' = y \cdot sy$$
$$z' = z \cdot sz$$

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



Scaling about any fixed point:

- Translate object so that the fixed point coincides with the coordinate origin
- Scale the object with respect to the coordinate origin
- Use the inverse translation of step 1 to return the object to its original position



$$\mathbf{T}(x_f, y_f, z_f) \cdot \mathbf{S}(s_x, s_y, s_z) \cdot \mathbf{T}(-x_f, -y_f, -z_f) = \begin{bmatrix} s_x & 0 & 0 & (1-s_x)x_f \\ 0 & s_y & 0 & (1-s_y)y_f \\ 0 & 0 & s_z & (1-s_z)z_f \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

OTHER TRANSFORMATIONS

In addition to translation, rotation, and scaling, there are various additional transformations that are often useful in three-dimensional graphics applications.

Two of these are reflection and shear.

Reflections

A three-dimensional reflection can be performed relative to a selected reflection axis or with respect to a selected reflection plane.

. Reflections relative to a given axis are equivalent to 180 " rotations about that axis. Reflections with respect to a plane are equivalent to 180' rotations in four-dimensional space

The matrix representation for this reflection of points relative to the xy plane (z axis) is

3D Reflections

About an axis: equivalent to 180˚rotation about that axis

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

reflection wrt yz-plane

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

reflection wrt xz-plane

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

reflection wrt xy-plane

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

reflection wrt the origin

SHEAR

Shearing transformations can be used to modify object shapes.
They are also useful in three-dimensional viewing for obtaining general projection

X axis shear:

The effect of this transformation matrix is to alter x values by an amount that is proportional to the shearing factors Sh_{x1} and Sh_{x2} value, while leaving the y and z coordinates unchanged.

$$\begin{bmatrix} 1 & sh_{x_1} & sh_{x_2} & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ sh_{y_1} & 1 & sh_{y_2} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ sh_{z_1} & sh_{z_2} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

along x-axis, along y-axis, along z-axis