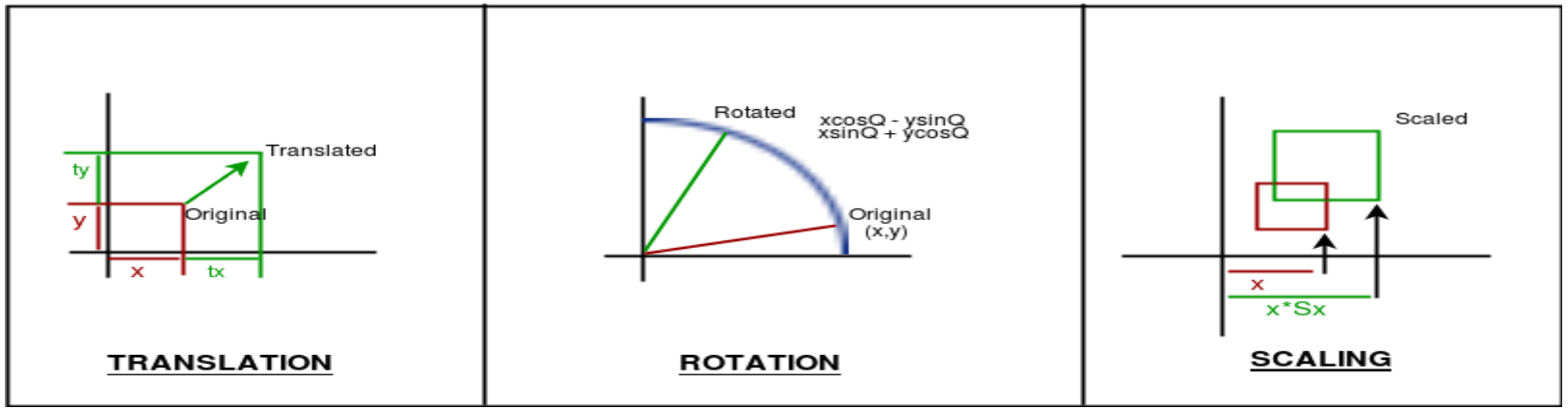
Computer Graphics
&
Multimedia Techniques

Unit-3

Two Dimensional Graphics

Transformations:

- **Changes in orientation, size, and shape are accomplished with geometric transformations that alter the coordinate descriptions of objects**.
- The basic geometric transformations are **translation, rotation, and scaling**. Other transformations that are often applied to objects include **reflection and shear**.

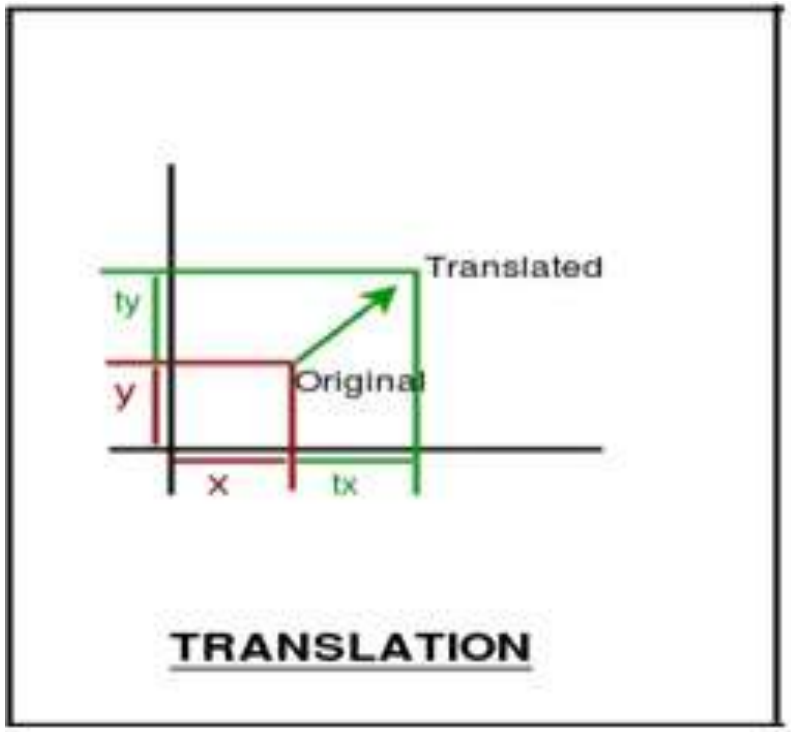


Translation:

- **A** translation is applied to an object by repositioning it along a straight-line path from one coordinate location to another. We translate a two-dimensional point by adding translation distances, **tx,** and **ty**, to the original coordinate position **(x, y)** to move the point to a new position ***(x ′ , y')***

$$x' = x + tx,$$

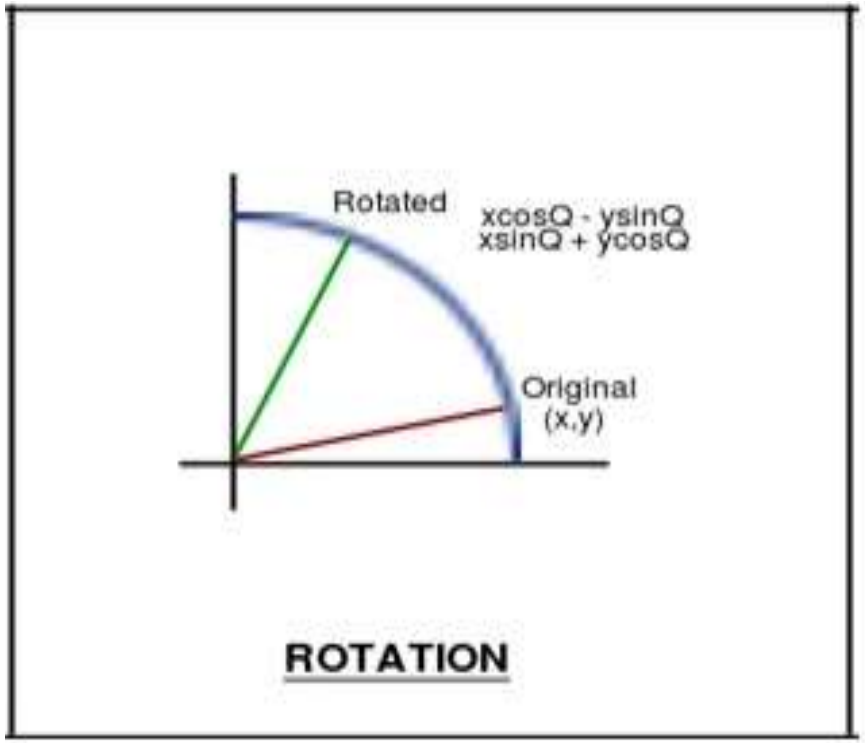
$$y' = y + ty,$$



Rotation:

A two-dimensional rotation is applied to an object by repositioning it along a circular path in the **xy** plane. To generate a rotation, we specify a rotation angle θ and the position ***(x, y,)*** of the rotation point (or pivot point) about which the object is to be rotated.

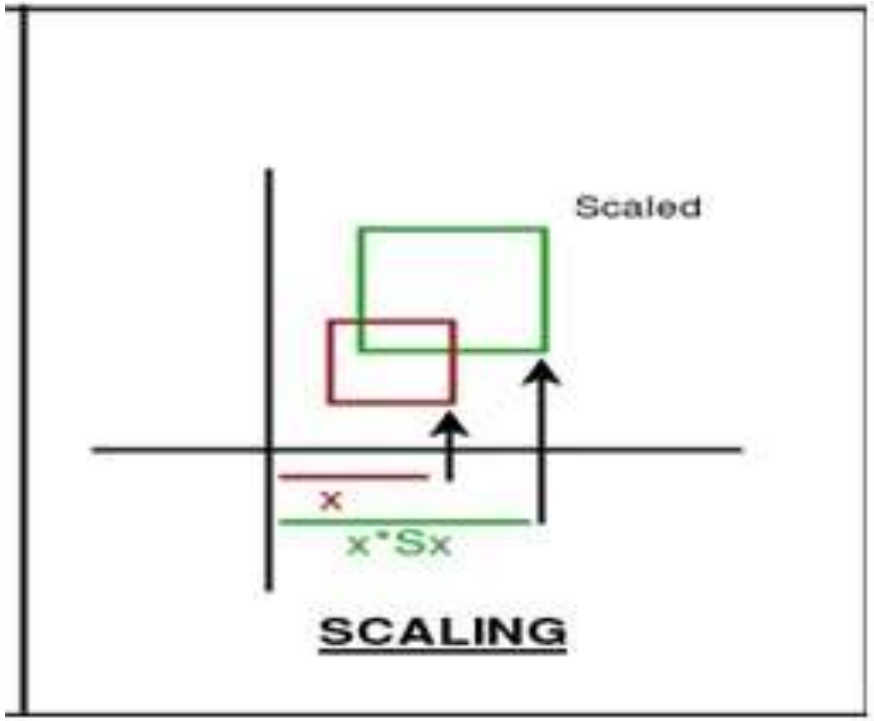
$$x' = x \cos\theta - y \sin\theta$$
$$y' = x \sin\theta + y \cos\theta$$



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 **Sx**, and **Sy**, to produce the transformed coordinates **(x', y').**

$$x' = x \cdot Sx$$
$$y' = y \cdot Sy$$



Matrix representation & homogeneous coordinates

Translation:

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

$$p' = T(t_x, t_y) \cdot p$$

Matrix representation & homogeneous coordinates

- Rotation:

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

$$p' = R(\theta) \cdot p$$

Matrix representation & homogeneous coordinates

- Scaling

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

$$p' = s(s_x, s_y) \cdot p$$

Composite transformations

- A number of transformations or sequence of transformations can be combined into single one called as composition. The resulting matrix is called as composite matrix. The process of combining is called as concatenation.
- Translation-translation
- Rotation-rotation
- Scaling-scaling
- Pivot point rotation
- Fixed point scaling

Translation-translation

Translations

If two successive translation vectors (t_{x1}, t_{y1}) and (t_{x2}, t_{y2}) are applied to a coordinate position **P**, the final transformed location **P'** is calculated as

$$\begin{aligned}\mathbf{P}' &= T(t_{x2}, t_{y2}) \cdot \{\mathbf{T}(t_{x1}, t_{y1}) \cdot \mathbf{P}\} \\ &= \{\mathbf{T}(t_{x2}, t_{y2}) \cdot \mathbf{T}(t_{x1}, t_{y1})\} \cdot \mathbf{P}\end{aligned} \tag{5-23}$$

where **P** and **P'** are represented as homogeneous-coordinate column vectors. We can verify this result by calculating the matrix product for the two associative groupings. Also, the composite transformation matrix for this sequence of translations is

$$\begin{bmatrix} 1 & 0 & t_{x2} \\ 0 & 1 & t_{y2} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & t_{x1} \\ 0 & 1 & t_{y1} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_{x1} + t_{x2} \\ 0 & 1 & t_{y1} + t_{y2} \\ 0 & 0 & 1 \end{bmatrix} \tag{5-24}$$

or

$$\mathbf{T}(t_{x2}, t_{y2}) \cdot \mathbf{T}(t_{x1}, t_{y1}) = \mathbf{T}(t_{x1} + t_{x2}, t_{y1} + t_{y2}) \tag{5-25}$$

which demonstrates that two successive translations are additive.

Rotation-rotation

Rotations

Two successive rotations applied to point P produce the transformed position

$$\mathbf{P}' = \mathbf{R}(\theta_2) \cdot \{\mathbf{R}(\theta_1) \cdot \mathbf{P}$$
$$= \{\mathbf{R}(\theta_2) \cdot \mathbf{R}(\theta_1)\} \cdot \mathbf{P} \qquad (5\text{-}26)$$

By multiplying the two rotation matrices, we can verify that two successive rotations are additive:

$$\mathbf{R}(\theta_2) \cdot \mathbf{R}(\theta_1) = \mathbf{R}(\theta_1 + \theta_2) \qquad (5\text{-}27)$$

so that the final rotated coordinates can be calculated with the composite rotation matrix as

$$\mathbf{P}' = \mathbf{R}(\theta_1 + \theta_2) \cdot \mathbf{P} \qquad (5\text{-}28)$$

Scaling-scaling

Scalings

Concatenating transformation matrices for two successive scaling operations produces the following composite scaling matrix:

$$\begin{bmatrix} s_{x2} & 0 & 0 \\ 0 & s_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_{x1} & 0 & 0 \\ 0 & s_{y1} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_{x1} \cdot s_{x2} & 0 & 0 \\ 0 & s_{y1} \cdot s_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (5\text{-}29)$$

or

$$\mathbf{S}(s_{x2}, s_{y2}) \cdot \mathbf{S}(s_{x1}, s_{y1}) = \mathbf{S}(s_{x1} \cdot s_{x2}, s_{y1} \cdot s_{y2}) \quad (5\text{-}30)$$

The resulting matrix in this case indicates that successive scaling operations are multiplicative. That is, if we were to triple the size of an object twice in succession, the final size would be nine times that of the original.

General Pivot-Point Rotation

General Pivot-Point Rotation

With a graphics package that only provides a rotate function for revolving objects about the coordinate origin, we can generate rotations about any selected pivot point (x_r, y_r) 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.

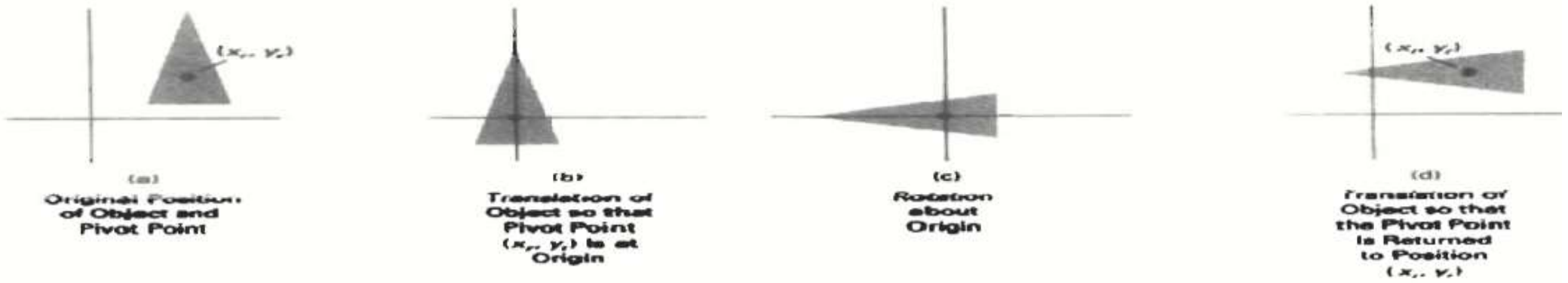


Figure 5-9
A transformation sequence for rotating an object about a specified pivot point using the rotation matrix $\mathbf{R}(\theta)$ of transformation 5-19.

General Pivot-Point Rotation

tion matrix for this sequence is obtained with the concatenation

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

$$= \begin{bmatrix} \cos\theta & -\sin\theta & x_r(1-\cos\theta) + y_r \sin\theta \\ \sin\theta & \cos\theta & y_r(1-\cos\theta) - x_r \sin\theta \\ 0 & 0 & 1 \end{bmatrix} \qquad (5\text{-}31)$$

which can be expressed in the form

$$\mathbf{T}(x_r, y_r) \cdot \mathbf{R}(\theta) \cdot \mathbf{T}(-x_r, -y_r) = \mathbf{R}(x_r, y_r, \theta) \qquad (5\text{-}32)$$

where $\mathbf{T}(-x_r, -y_r) = \mathbf{T}^{-1}(x_r, y_r)$. In general, a rotate function can be set up to accept parameters for pivot-point coordinates, as well as the rotation angle, and to generate automatically the rotation matrix of Eq. 5-31.

Fixed point scaling

General Fixed-Point Scaling

Figure 5-10 illustrates a transformation sequence to produce scaling with respect to a selected fixed position (x_f, y_f) 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.

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

$$\begin{bmatrix} 1 & 0 & x_f \\ 0 & 1 & y_f \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_f \\ 0 & 1 & -y_f \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & x_f(1-s_x) \\ 0 & s_y & y_f(1-s_y) \\ 0 & 0 & 1 \end{bmatrix} \quad (5\text{-}33)$$

or

$$\mathbf{T}(x_f, y_f) \cdot \mathbf{S}(s_x, s_y) \cdot \mathbf{T}(-x_f, -y_f) = \mathbf{S}(x_f, y_f, s_x, s_y) \quad (5\text{-}34)$$

This transformation is automatically generated on systems that provide a scale function that accepts coordinates for the fixed point.

Fixed point scaling

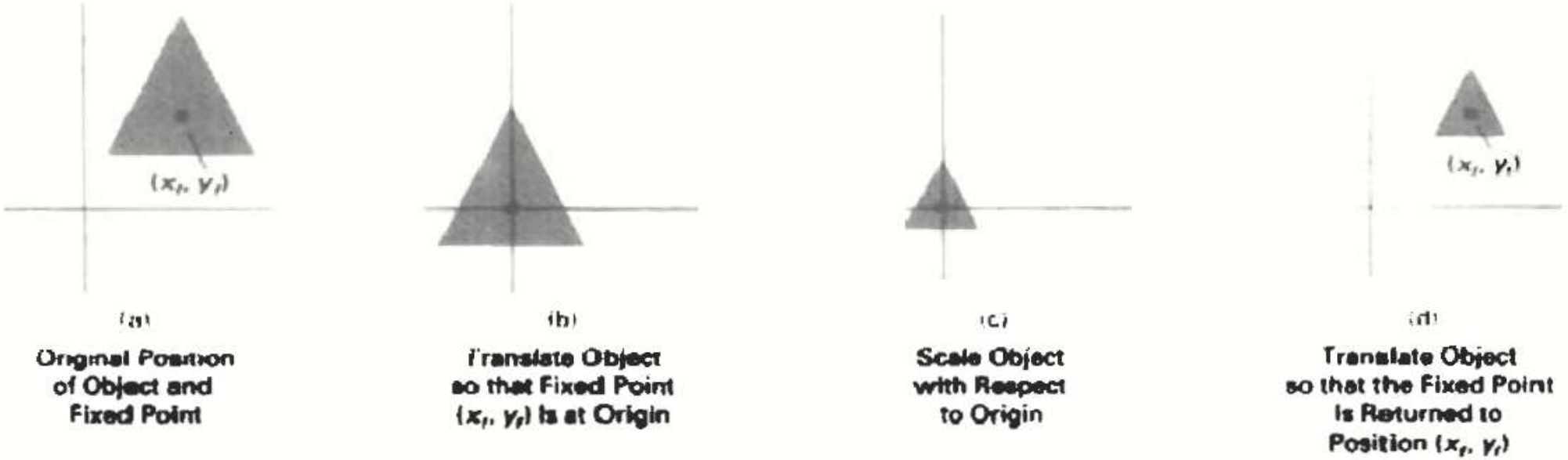


Figure 5-10
A transformation sequence for scaling an object with respect to a specified fixed position using the scaling matrix $S(s_x, s_y)$ of transformation 5-21.

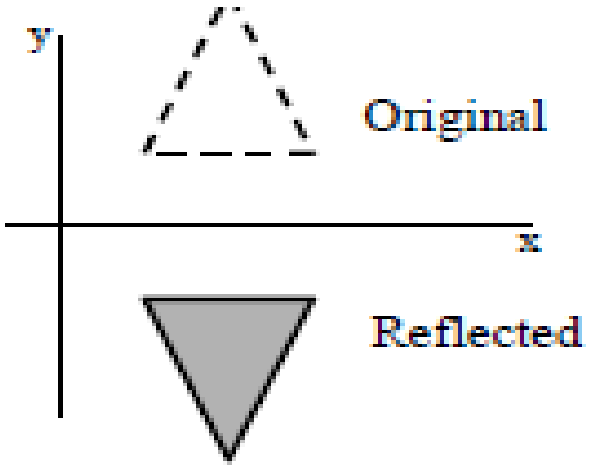
Reflection

- **A** reflection is a transformation that produces a mirror image of an object.
- The mirror image for a two-dimensional reflection is generated relative to an **axis** of reflection by rotating the object 180° about the reflection axis.
- We can choose an axis of reflection in the *xy* plane or perpendicular to the *xy* plane.
- When the reflection axis is a line in the *xy* plane, the rotation path about this axis is in a plane perpendicular to the *xy* plane.
- For reflection axes that are perpendicular to the *xy* plane, the rotation path is in the *xy* plane.

Reflection

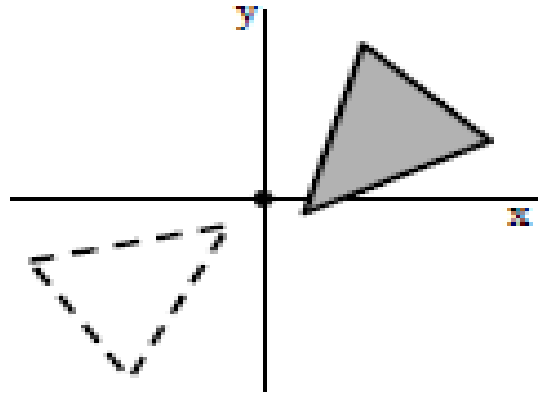
x-axis ($y = 0$)

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



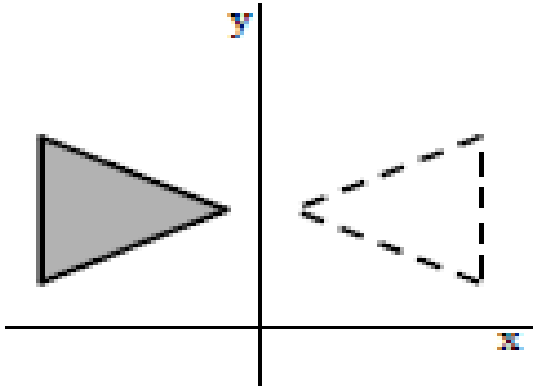
xy-plane

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



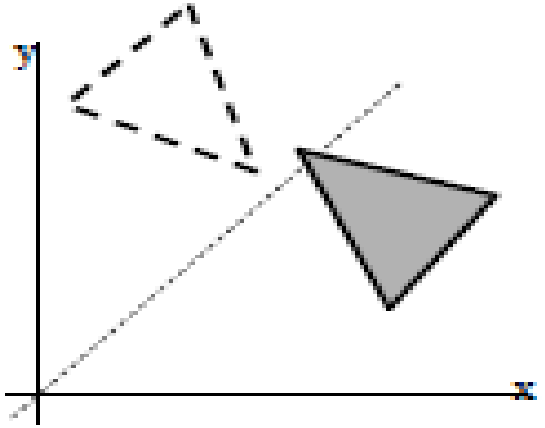
y-axis ($x = 0$)

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



xy-plane

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



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

Shear

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

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

which transforms coordinate positions as

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

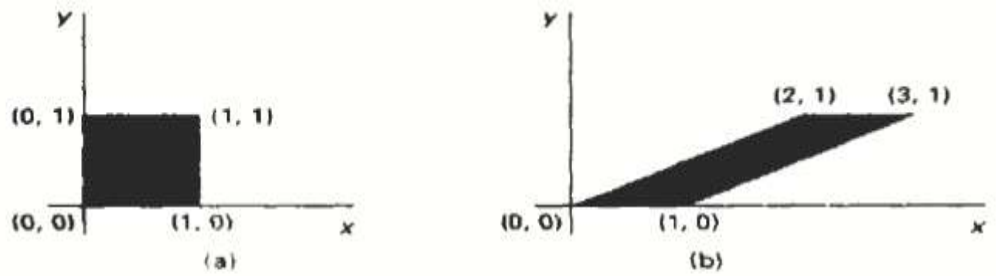


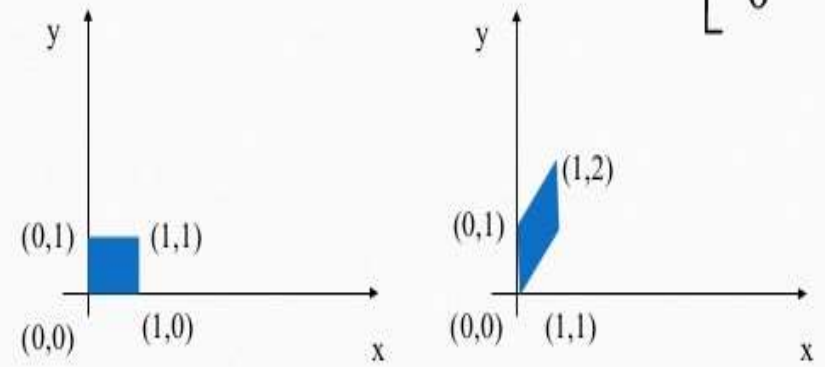
Figure 5-23
A unit square (a) is converted to a parallelogram (b) using the x-direction shear matrix 5-53 with $sh_x = 2$.

- **An y-direction** shear relative to the y axis is produced with the transformation matrix

$$x' = x$$

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

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



THE VIEWING PIPELINE

- A world-coordinate area selected for display is called a window.
- An area on a display device to which a window is mapped is called a viewport.
- The window defines ***what*** is to be viewed; the viewport defines ***where*** it is to be displayed.

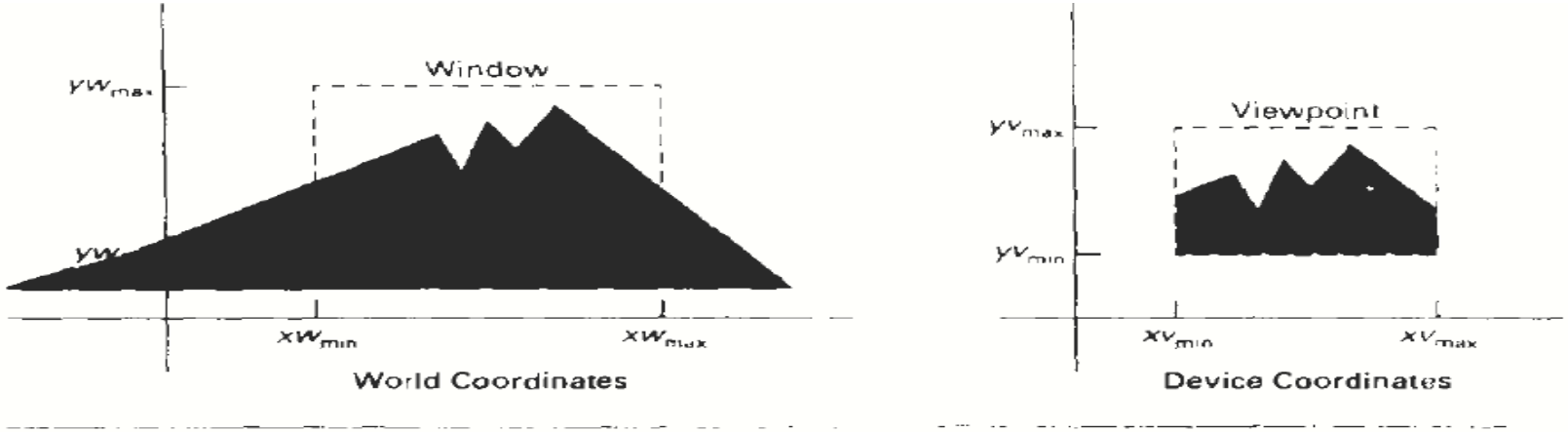
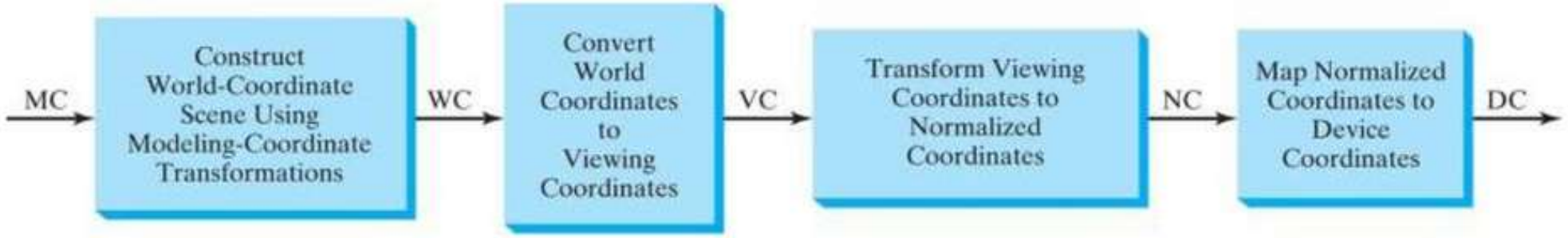


Figure 6-1
A viewing transformation using standard rectangles for the window and viewport.

THE VIEWING PIPELINE

- the mapping of a part of a world-coordinate scene to device coordinates is referred to as a viewing transformation.
- Sometimes the two-dimensional viewing transformation is simply referred to as the ***window-to-viewport transformation*** or the ***windowing transformation.***



WINDOW-TO-VIEWPORT COORDINATE TRANSFORMATION

- Object descriptions are transferred to normalized device coordinates:
- We do this thing using a transformation that maintains the same relative placement of an object in normalized space as they had in viewing coordinates.
- If a coordinate position is at the center of the viewing window, It will display at the center of the viewport.

Figure 6-5
A point at position (xw, yw) in a designated window is mapped to viewport coordinates (xv, yv) so that relative positions in the two areas are the same.

WINDOW-TO-VIEWPORT COORDINATE TRANSFORMATION

- In order to maintain the same relative placement of the point in the viewport as in the window, we require:

$$\frac{xv - xv_{min}}{xv_{max} - xv_{min}} = \frac{xw - xw_{min}}{xw_{max} - xw_{min}}$$

$$\frac{yv - yv_{min}}{yv_{max} - yv_{min}} = \frac{yw - yw_{min}}{yw_{max} - yw_{min}}$$

WINDOW-TO-VIEWPORT COORDINATE TRANSFORMATION

Solving these expressions for the viewport position (xv, yv), we have

$$xv = xv_{min} + (xw - xw_{min})sx$$

$$yv = yv_{min} + (yw - yw_{min})sy$$

where the scaling factors are

$$sx = \frac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}}$$

$$sy = \frac{yv_{max} - yv_{min}}{yw_{max} - yw_{min}}$$

WINDOW-TO-VIEWPORT COORDINATE TRANSFORMATION

- This conversion is performed with the following sequence of transformations:
  1. Perform a scaling transformation using a fixed point position (xw_{min},yw_{min}) that scales the window area to the size of the viewport.
  2. Translate the scaled window area to the position of the viewport. Relative proportions of objects are maintained if the scaling factors are the same (sx=sy).

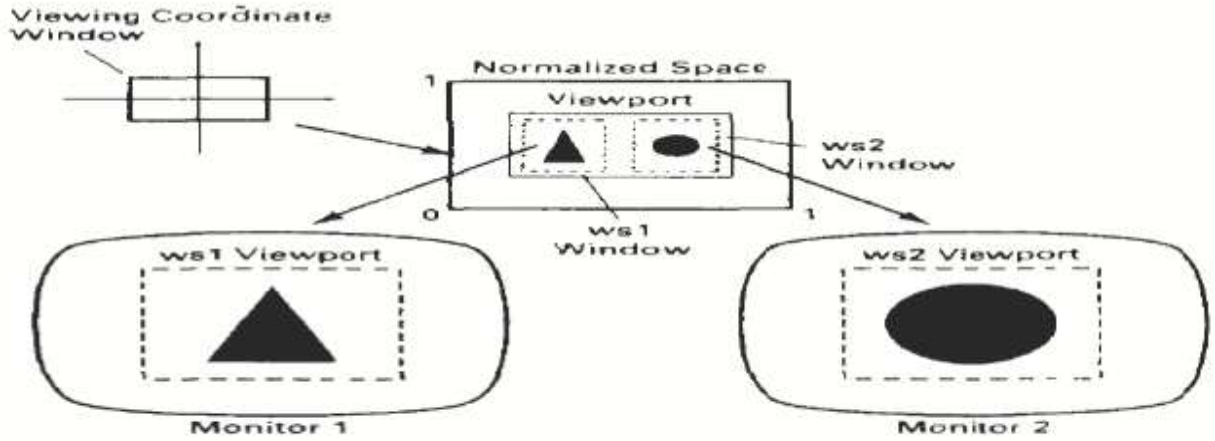


Figure 6-6
Mapping selected parts of a scene in normalized coordinates to different video monitors with workstation transformations.

Clipping

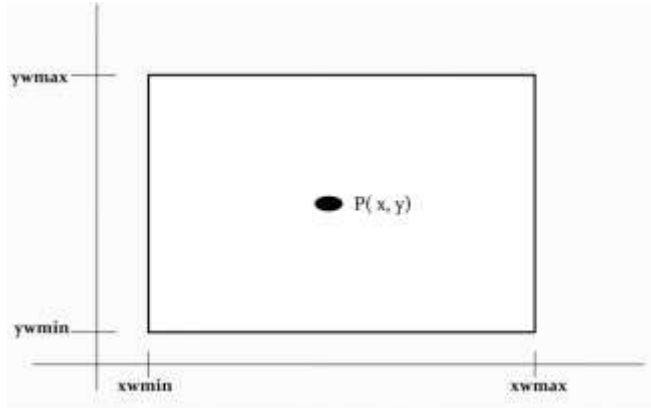
- Generally, any procedure that identifies those portions of a picture that are either inside or outside of a specified region of space is referred to as a clipping algorithm, or simply clipping.
- The region against which an object is to clipped is called a clip window.
- Applications of clipping include extracting part of a defined scene for viewing, identifying visible surfaces in three-dimensional views and antialiasing line segments or object boundaries;
- In the following sections, we consider algorithms for clipping the following primitive types
  - **Point clipping**
  - **Line clipping (cohen-sutherland, liang- bersky and NLN algorithms)**
  - **Polygon clipping (Sutherland Hodgeman)**

Point clipping

- Assuming that the clip window is a rectangle in standard position, we save a point P = ***(x, y)*** for display if the following inequalities are satisfied:

$$xw_{min} \leq x \leq xw_{max}$$

$$yw_{min} \leq y \leq yw_{max}$$



- where the *edges of* the clip window **(XWmin, XWmax, Ywmin, YWmax*)*** can be either the world-coordinate window boundaries or viewport boundaries. If any one of these four inequalities is not satisfied, the point is clipped (not saved for display).

Line Clipping

- line clipping is the process of removing lines or portions of lines outside an area of interest. Typically, any line or part thereof which is outside of the viewing area is removed.

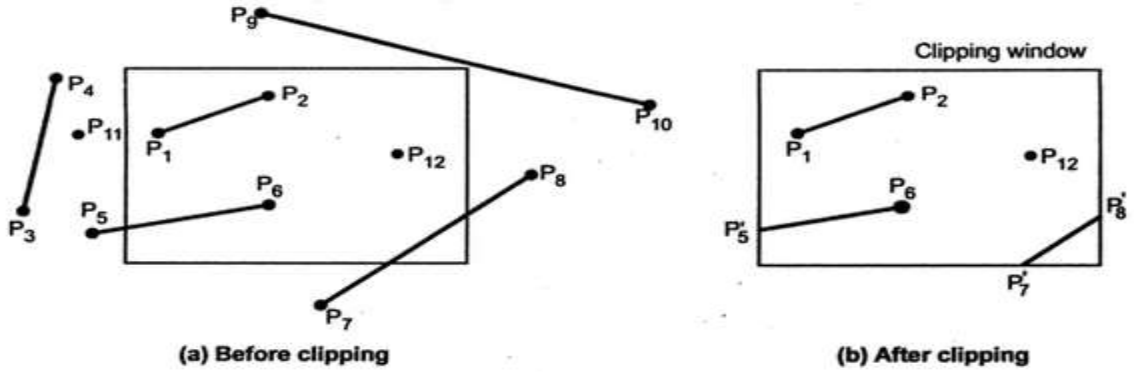
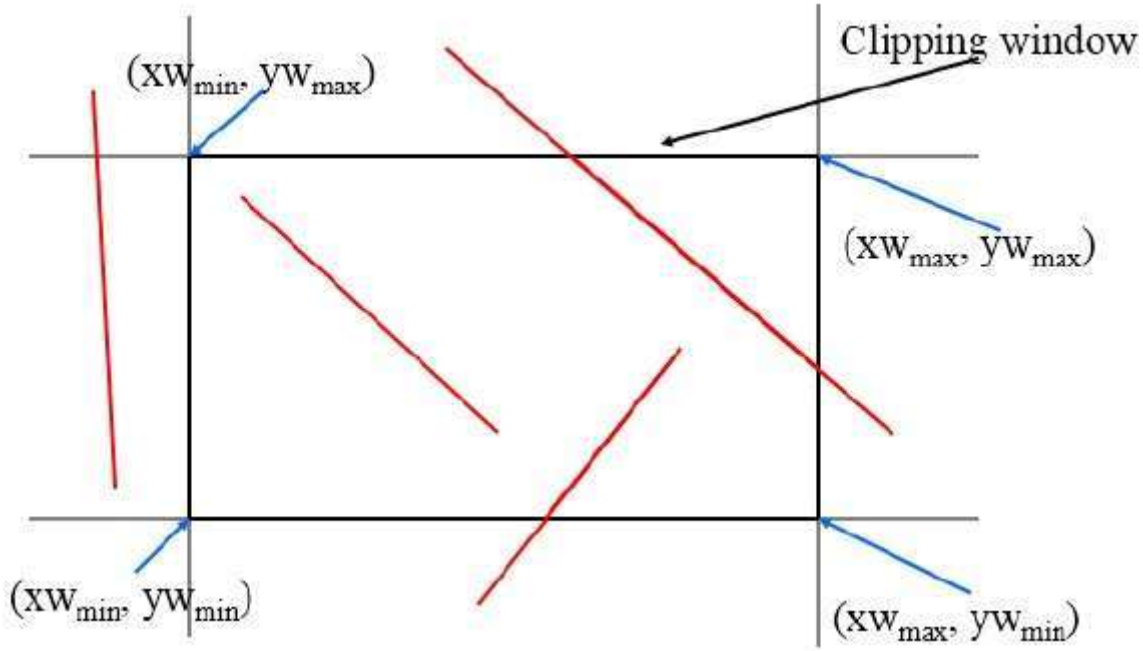


Fig. (e)

- There are two common algorithms for line clipping:
  - **Cohen–Sutherland and Liang–Barsky.**

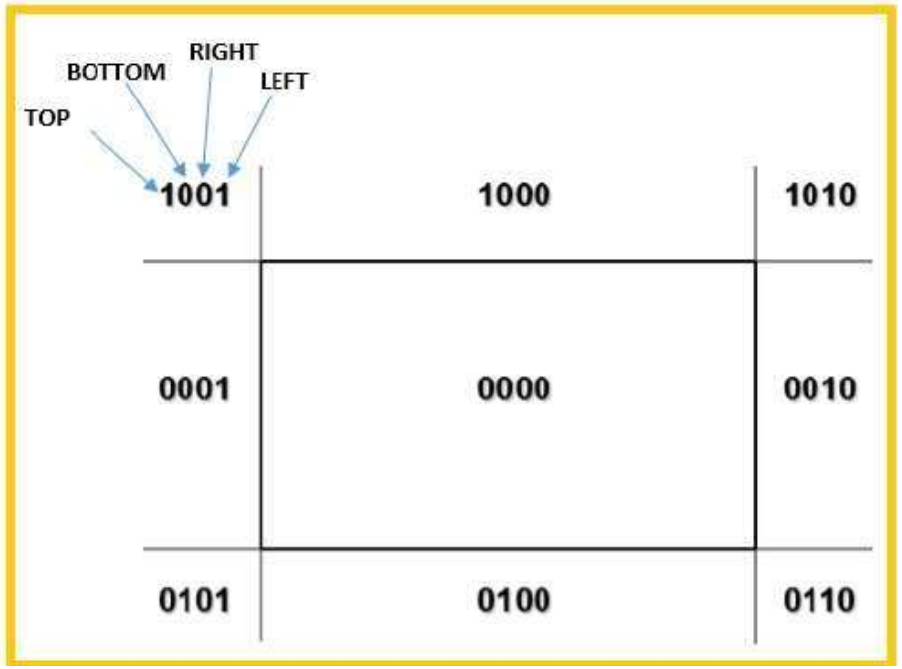
Cohen-Sutherland line clipping

This algorithm uses the clipping window as shown in the following figure. The minimum coordinate for the clipping region is (XWmin,Ywmin) and the maximum coordinate for the clipping region is (XWmax,YWmax).



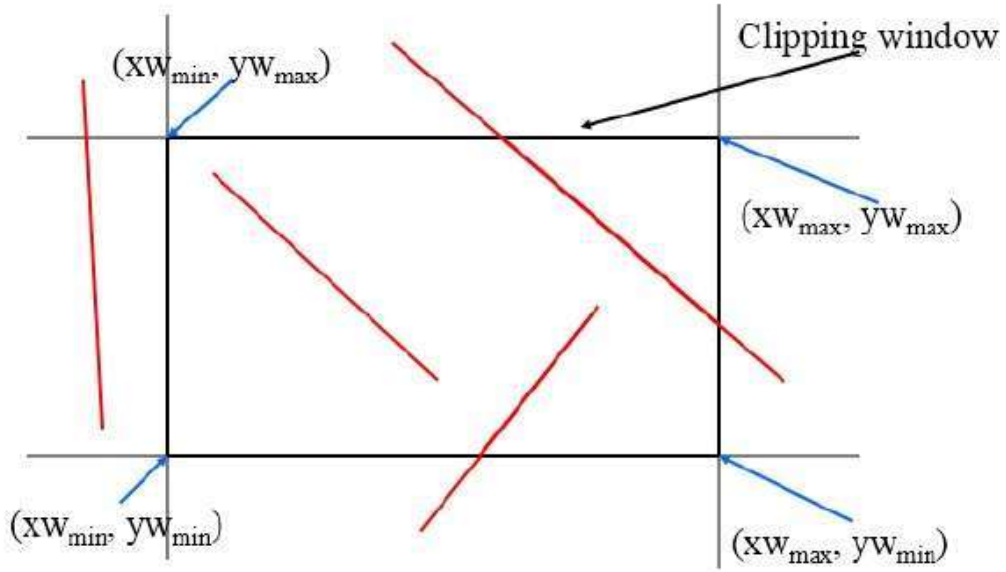
Cohen-Sutherland line clipping

- We will use 4-bits to divide the entire region. These 4 bits represent the Top, Bottom, Right, and Left of the region as shown in the following figure. Here, the **TOP** and **LEFT** bit is set to 1 because it is the **TOP-LEFT** corner.



Cohen-Sutherland line clipping

- There are 3 possibilities for the line –
  - Line can be completely inside the window
  - Line can be completely outside of the window
  - Line can be partially inside the window



Cohen-Sutherland line clipping

Algorithm

Step 1 − Assign a region code for each endpoints.

Step 2 − If both endpoints have a region code **0000** then accept this line.

Step 3 − Else, perform the logical **AND**operation for both region codes.

Step 3.1 − If the result is not **0000,** then reject the line.

Step 3.2 − Else you need clipping.

Step 3.2.1 − Choose an endpoint of the line that is outside the window.

Step 3.2.2 − Find the intersection point at the window boundary $baseonregioncode$.

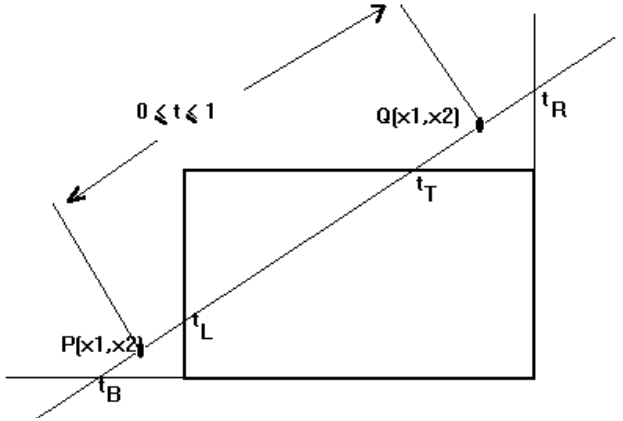
Step 3.2.3 − Replace endpoint with the intersection point and update the region code.

Step 3.2.4 − Repeat step 2 until we find a clipped line either trivially accepted or trivially rejected.

Step 4 − Repeat step 1 for other lines.

Liang-Bersky line clipping

- Liang and Barsky have created an algorithm that uses floating-point arithmetic but finds the appropriate end points with at most four computations.
- This algorithm uses the parametric equations for a line and solves four inequalities to find the range of the parameter for which the line is in the viewport.



Let $P(x_1,y_1)$, $Q(x_2,y_2)$ be the line which we want to study. The **parametric equation of the line segment** from gives x-values and y-values for every point in terms of a **parameter t**that ranges from 0 to 1. The equations are

$$x = x_1 + (x_2 - x_1)*t = x_1 + dx*t \quad \text{and} \quad y = y_1 + (y_2 - y_1)*t = y_1 + dy*t$$

We can see that when t = 0, the point computed is P(x1,y1); and when t = 1, the point computed is Q(x2,y2).

Liang-Bersky line clipping

Algorithm

1. Set $t_{min} = 0$ and $t_{max} = 1$

2. Calculate the values of tL, tR, tT, and tB (tvalues).
   - if $t < t_{min}$ or $t > t_{max}$ ignore it and go to the next edge
   - otherwise classify the **t**value as entering or exiting value (using inner product to classify)
   - if **t** is entering value set $t_{min} = t$; if **t** is exiting value set $t_{max} = t$

3. If $t_{min} < t_{max}$ then **draw a line** from (x1 + dx*tmin, y1 + dy*tmin) to (x1 + dx*tmax, y1 + dy*tmax)

4. If the line crosses over the window, you will see (x1 + dx*tmin, y1 + dy*tmin) and (x1 + dx*tmax, y1 + dy*tmax) are intersection between line and edge.

Nicholl–Lee–Nicholl (NLN) Algorithm

- Remove extra calculation of cohen-Sutherland and Liang-bersky line clipping algorithm.
- The trade-off is that NLN algorithm can only be applied to two-dimensional clipping.
- Steps
  - For a line with endpoints P1 and P2, we first determine the position of point P1 for the nine possible regions relative to the clipping rectangle.
  - Only three regions needed to be considered as shown in figure, if P1 lies in any one of the other six regions, we can move it to one of the three regions using a symmetric transformation. For example, the region directly above the clip window can be transformed to the region left of the clip window using a reflection about the line y=-x, or we could use a 90° counter clock wise rotation.

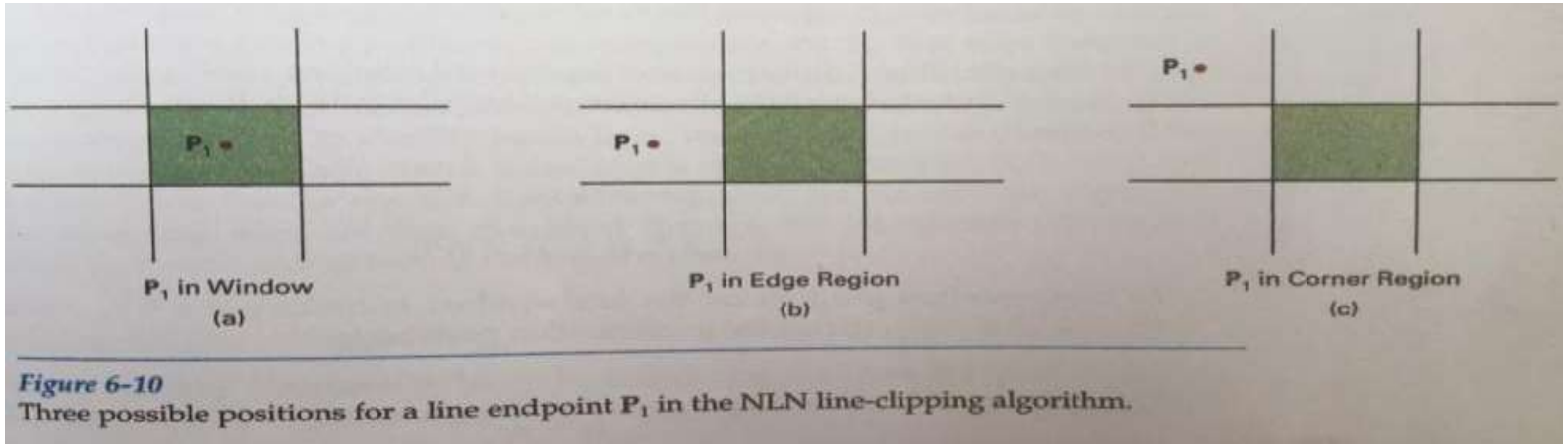


Figure 6-10
Three possible positions for a line endpoint P_1 in the NLN line-clipping algorithm.

Nicholl–Lee–Nicholl (NLN) Algorithm

- Next we determine the position of P2 relative to P1. We create new regions in the plane, depending on the location of P1.
- Boundaries of the new regions are half-infinite line segments that start a the position of P1 and pass through the window corners. If P1 is inside the clip window and P2 is out side, we can set up the four regions shown in figure 6-11.
- The intersection with the appropriate window boundary is then carried out, depending on which one of the four regions (L, T, R or B) contains P2. of course, if both the P1 and P2 are inside the clipping rectangle, we simply save the entire line.

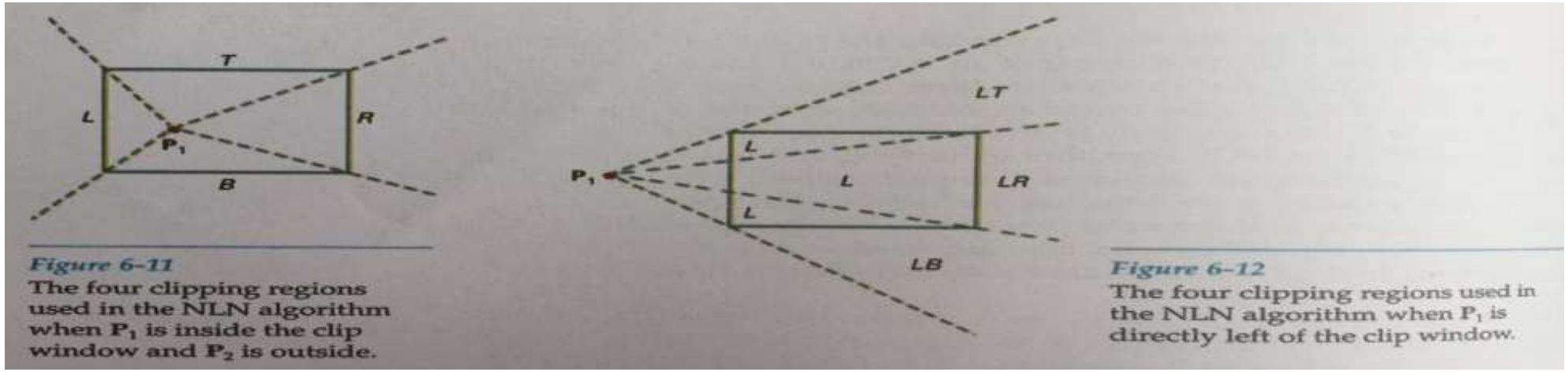


Figure 6-11
The four clipping regions used in the NLN algorithm when P_1 is inside the clip window and P_2 is outside.

Figure 6-12
The four clipping regions used in the NLN algorithm when P_1 is directly left of the clip window.

Nicholl–Lee–Nicholl (NLN) Algorithm

- If P1 is in the region to the left of the window, we can set up the four regions, L, LT, LR and LB, shown in figure 6-12.
- These four regions determine a unique boundary for the line segment. For instance, if P2 is in region L, we can clip at the left boundary and save the line segments from intersection point to P2.
- But if P2 is in region LT, we save the line segment from the left window boundary to the top boundary. If P2 is not in any of the four regions, L, LT, LR or LB, the entire line is clipped.

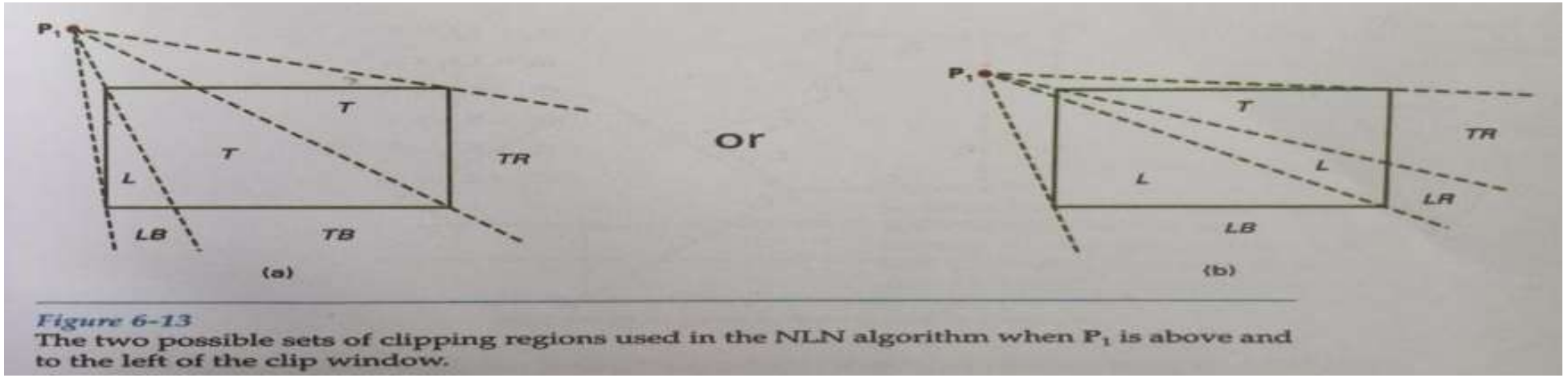
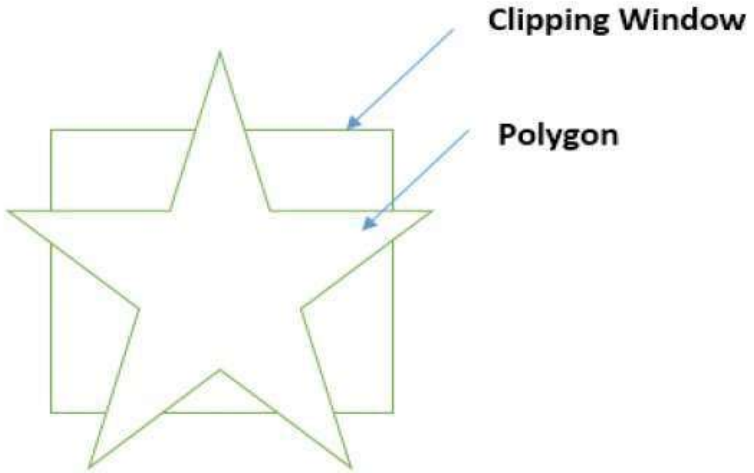


Figure 6-13
The two possible sets of clipping regions used in the NLN algorithm when P_1 is above and to the left of the clip window.

Polygon Clipping (Sutherland Hodgeman)

- A polygon can also be clipped by specifying the clipping window. Sutherland Hodgeman polygon clipping algorithm is used for polygon clipping. In this algorithm, all the vertices of the polygon are clipped against each edge of the clipping window.
- First the polygon is clipped against the left edge of the polygon window to get new vertices of the polygon. These new vertices are used to clip the polygon against right edge, top edge, bottom edge, of the clipping window as shown in the following figure



Polygon Clipping (Sutherland Hodgeman)

- While processing an edge of a polygon with clipping window, an intersection point is found if edge is not completely inside clipping window and the a partial edge from the intersection point to the outside edge is clipped. The following figures show left, right, top and bottom edge clippings.

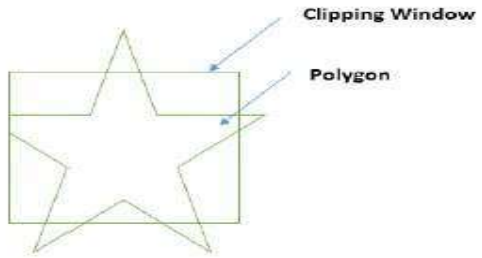


Figure: Clipping Left Edge

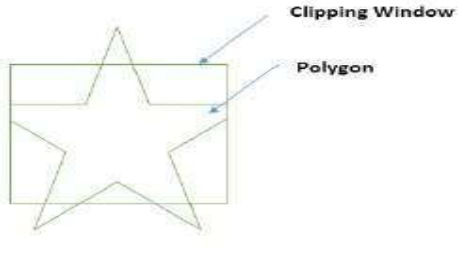


Figure: Clipping Right Edge

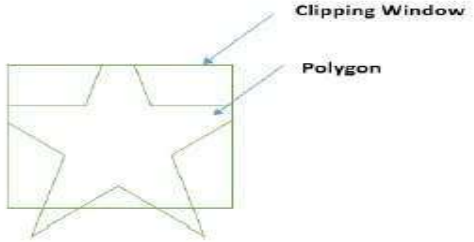


Figure: Clipping Top Edge

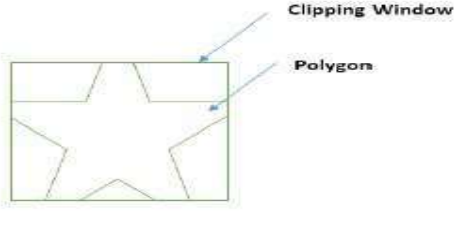


Figure: Clipping Bottom Edge


Thanks!