

unit 2

4

2-D Geometric Transformation

4.1 Introduction

Almost all graphics systems allow the programmer to define picture that include a variety of transformations. For example, the programmer is able to magnify a picture so that detail appears more clearly, or reduce it so that more of the picture is visible. The programmer is also able to rotate the picture so that he can see it in different angles.

In this chapter we discuss the 2D transformations.

4.2 Two Dimensional Transformations

In this section, we describe the general procedures for applying translation, rotation, and scaling parameters to reposition and resize the two dimensional objects.

4.2.1 Translation

Translation is a process of changing the position of an object in a straight-line path from one coordinate location to another. We can translate a two dimensional point by adding translation distances, t_x and t_y , to the original coordinate position (x, y) to move the point to a new position (x', y') , as shown in the Fig. 4.1.

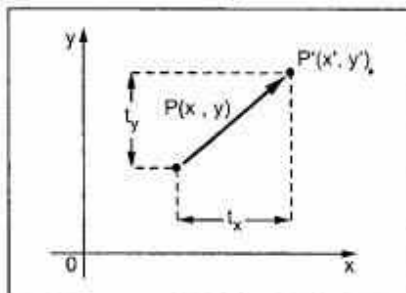


Fig. 4.1

$$x' = x + t_x \quad \dots (4.1)$$

$$y' = y + t_y \quad \dots (4.2)$$

The translation distance pair (t_x, t_y) is called a **translation vector** or **shift vector**.

It is possible to express the translation equations 4.1 and 4.2 as a single matrix equation by using column vectors to represent coordinate positions and the translation vector :

$$P = \begin{bmatrix} x \\ y \end{bmatrix} \quad P' = \begin{bmatrix} x' \\ y' \end{bmatrix} \quad T = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

This allows us to write the two dimensional translation equations in the matrix form :

$$P' = P + T \quad \dots (4.3)$$

Ex. 4.1: Translate a polygon with coordinates A (2, 5), B (7, 10) and C (10, 2) by 3 units in x direction and 4 units in y direction.

Sol.:

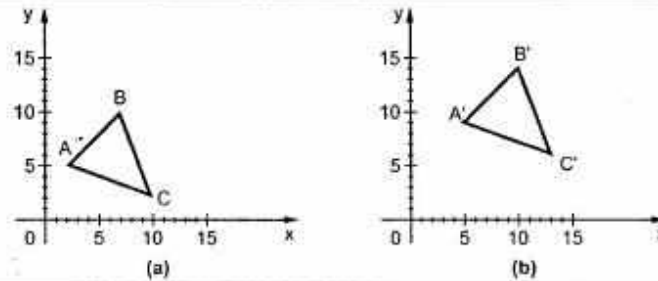


Fig. 4.2 Translation of polygon

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

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

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

4.2.2 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 of the rotation point about which the object is to be rotated.

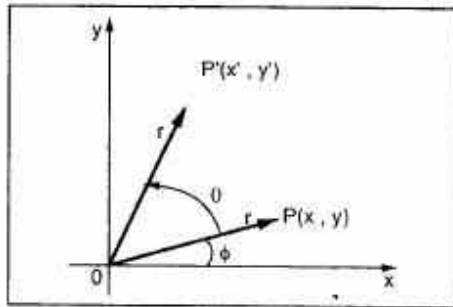


Fig. 4.3

Let us consider the rotation of the object about the origin, as shown in the Fig. 4.3.

Here, 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. Using standard trigonometric equations, we can express the transformed coordinates in terms of angles θ and ϕ as

$$\left. \begin{aligned} x' &= r \cos(\phi + \theta) = r \cos\phi \cos\theta - r \sin\phi \sin\theta \\ y' &= r \sin(\phi + \theta) = r \cos\phi \sin\theta + r \sin\phi \cos\theta \end{aligned} \right\} \dots (4.4)$$

The original coordinates of the point in polar coordinates are given as

$$\left. \begin{aligned} x &= r \cos\phi \\ y &= r \sin\phi \end{aligned} \right\} \dots (4.5)$$

Substituting equations 4.5 into 4.4, we get the transformation equations for rotating a point (x, y) through an angle θ about the origin :

$$\left. \begin{aligned} x' &= x \cos\theta - y \sin\theta \\ y' &= x \sin\theta + y \cos\theta \end{aligned} \right\} \dots (4.6)$$

The above equations can be represented in the matrix form as given below

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

$$\therefore P' = P \cdot R \dots (4.7)$$

where R is rotation matrix and it is given as

$$R = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \dots (4.8)$$

It is important to note that positive values for the rotation angle define counterclockwise rotations about the rotation point and negative values rotate objects in the clockwise sense.

For negative values of θ i.e., for clockwise rotation, the rotation matrix becomes

$$\begin{aligned} R &= \begin{bmatrix} \cos(-\theta) & \sin(-\theta) \\ -\sin(-\theta) & \cos(-\theta) \end{bmatrix} \\ &= \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \quad \because \cos(-\theta) = \cos\theta \quad \text{and} \quad \dots (4.9) \\ &\quad \sin(-\theta) = -\sin\theta \end{aligned}$$

Ex. 4.2: A point (4, 3) is rotated counterclockwise by an angle of 45° . Find the rotation matrix and the resultant point.

Sol. :

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

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

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

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

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

4.2.3 Scaling

A scaling transformation changes 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 and S_y to produce the transformed coordinates (x', y').

$$x' = x \cdot S_x$$

and $y' = y \cdot S_y$... (4.10)

Scaling factor S_x scales object in the x direction and scaling factor S_y scales object in the y direction. The equations 4.10 can be written in the matrix form as given below :

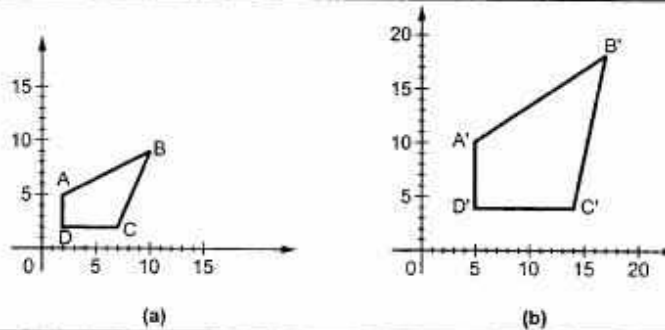


Fig. 4.4

$$[x' \ y'] = [x \ y] \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix}$$

$$= [x \cdot S_x \quad y \cdot S_y]$$

$$= P \cdot S$$

... (4.11)

Any positive numeric values are valid for scaling factors S_x and S_y . Values less than 1 reduce the size of the objects and values greater than 1 produce an enlarged object. For both S_x and S_y values equal to 1, the size of object does not change. To get uniform scaling it is necessary to assign same value for S_x and S_y . Unequal values for S_x and S_y result in a differential scaling.

Ex. 4.3 : 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

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

x y

The object matrix is :

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

$$\begin{matrix} A' \\ B' \\ C' \end{matrix} \begin{bmatrix} x'_1 & y'_1 \\ x'_2 & y'_2 \\ x'_3 & y'_3 \end{bmatrix} = \begin{bmatrix} 2 & 5 \\ 7 & 10 \\ 10 & 2 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

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

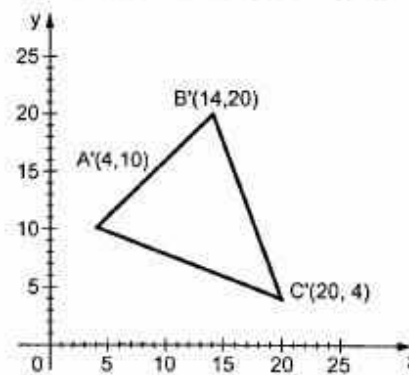
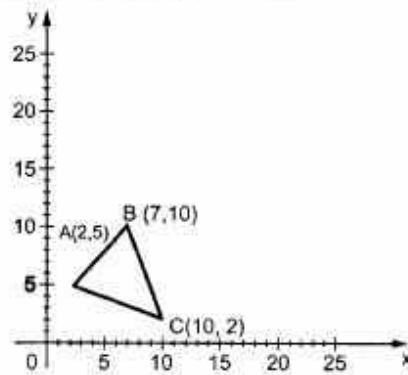


Fig. 4.5

4.3 Homogeneous Coordinates

In design and picture formation process, many times we may require to perform translation, rotations, and scaling to fit the picture components into their proper positions. In the previous section we have seen that each of the basic transformations can be expressed in the general matrix form

$$P' = P \cdot M_1 + M_2 \quad \dots (4.12)$$

For translation :

$$P' = P \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

i.e. $M_1 =$ Identity matrix

$M_2 =$ Translation vector

For rotation :

$$P' = P \cdot \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

i.e. $M_1 =$ Rotational matrix

$M_2 = 0$

For scaling :

$$P' = P \cdot \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

i.e. $M_1 =$ Scaling matrix

$M_2 = 0$

To produce a sequence of transformations with above equations, such as translation followed by rotation and then scaling, we must calculate the transformed coordinates one step at a time. First, coordinates are translated, then these translated coordinates are scaled, and finally, the scaled coordinates are rotated. But this sequential transformation process is not efficient. A more efficient approach is to combine sequence of transformations into one transformation so that the final coordinate positions are obtained directly from initial coordinates. This eliminates the calculation of intermediate coordinate values.

In order to combine sequence of transformations we have to eliminate the matrix addition associated with the translation terms in M_2 (Refer equation 4.12). To achieve this we have to represent matrix M_1 as 3×3 matrix instead of 2×2 introducing an additional dummy coordinate W . Here, points are specified by three numbers instead of two. This coordinate system is called **homogeneous coordinate system** and it allows us to express all transformation equations as matrix multiplication.

The homogeneous coordinate is represented by a triplet (X_w, Y_w, W) ,

where

$$x = \frac{X_w}{W} \quad \text{and} \quad y = \frac{Y_w}{W}$$

For two dimensional transformations, we can have the homogeneous parameter W to be any non zero value. But it is convenient to have $W = 1$. Therefore, each two dimensional position can be represented with homogeneous coordinate as $(x, y, 1)$.

Summarizing it all up, we can say that the homogeneous coordinates allow combined transformation, eliminating the calculation of intermediate coordinate values and thus save required time for transformation and memory required to store the intermediate coordinate values. Let us see the homogeneous coordinates for three basic transformations.

4.3.1 Homogeneous Coordinates for Translation

The homogeneous coordinates for translation are given as

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix} \quad \dots (4.13)$$

Therefore, we have

$$\begin{aligned} [x' \ y' \ 1] &= [x \ y \ 1] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix} \\ &= [x + t_x \ y + t_y \ 1] \end{aligned} \quad \dots (4.14)$$

4.3.2 Homogeneous Coordinates for Rotation

The homogeneous coordinates for rotation are given as

$$R = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \dots (4.15)$$

Therefore, we have

$$\begin{aligned} [x' \ y' \ 1] &= [x \ y \ 1] \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= [x \cos\theta - y \sin\theta \quad x \sin\theta + y \cos\theta \quad 1] \end{aligned} \quad \dots (4.16)$$

4.3.3 Homogeneous Coordinates for Scaling

The homogeneous coordinate for scaling are given as

$$S = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Therefore, we have

$$\begin{aligned} [x' \ y' \ 1] &= [x \ y \ 1] \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= [x \cdot S_x \ y \cdot S_y \ 1] \end{aligned} \quad \dots (4.17)$$

Note : In this text, the object matrix is written first and it is then multiplied by the required transformation matrix. If we wish to write the transformation matrix first and then the object matrix we have to take the transpose of both the matrices and post-multiply the object matrix i.e.,

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

Ex. 4.4 : Give a 3×3 homogeneous coordinate transformation matrix for each of the following translations

- Shift the image to the right 3-units
- Shift the image up 2 units
- Move the image down $\frac{1}{2}$ unit and right 1 unit
- Move the image down $\frac{2}{3}$ unit and left 4 units

Sol. : We know that homogenous coordinates for translation are

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix}$$

a) Here, $t_x = 3$ and $t_y = 0$

$$\therefore T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 3 & 0 & 1 \end{bmatrix}$$

b) Here, $t_x = 0$ and $t_y = 2$

$$\therefore T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 2 & 1 \end{bmatrix}$$

c) Here, $t_x = 1$ and $t_y = -0.5$

$$\therefore T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & -0.5 & 1 \end{bmatrix}$$

d) Here, $t_x = -4$ and $t_y = -0.66$

$$\therefore T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -4 & -0.66 & 1 \end{bmatrix}$$

Ex. 4.5: Find the transformation matrix that transforms the given square ABCD to half its size with centre still remaining at the same position. The coordinates of the square are : A(1, 1), B(3, 1), C(3, 3), D(1, 3) and centre at (2, 2). Also find the resultant coordinates of square.

Sol.: This transformation can be carried out in the following steps.

1. Translate the square so that its center coincides with the origin.
2. Scale the square with respect to the origin.
3. Translate the square back to the original position.

Thus, the overall transformation matrix is formed by multiplication of three matrices.

$$\begin{aligned} \therefore T_1 \cdot S \cdot T &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -2 & -2 & 1 \end{bmatrix} \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & 2 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & 2 & 1 \end{bmatrix} \end{aligned}$$

$$\begin{aligned}
 &= \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 1 & 1 & 1 \end{bmatrix} \\
 \begin{bmatrix} A' \\ B' \\ C' \\ D' \end{bmatrix} &= \begin{bmatrix} 1 & 1 & 1 \\ 3 & 1 & 1 \\ 3 & 3 & 1 \\ 1 & 3 & 1 \end{bmatrix} \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 1 & 1 & 1 \end{bmatrix} \\
 &= \begin{bmatrix} 1.5 & 1.5 & 1 \\ 2.5 & 1.5 & 1 \\ 2.5 & 2.5 & 1 \\ 1.5 & 2.5 & 1 \end{bmatrix}
 \end{aligned}$$

Ex. 4.6: Find a transformation of triangle A (1, 0), B (0, 1), C (1, 1) by

- Rotating 45° about the origin and then translating one unit in x and y direction.
- Translating one unit in x and y direction and then rotating 45° about the origin.

Sol.: The rotation matrix is

$$R = \begin{bmatrix} \cos 45 & \sin 45 & 0 \\ -\sin 45 & \cos 45 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ -1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{ and}$$

The translation matrix is

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

$$\begin{aligned} \text{a) } R \cdot T &= \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ -1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ -1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 1 & 1 & 1 \end{bmatrix} \end{aligned}$$

$$\begin{aligned} \therefore \begin{bmatrix} A' \\ B' \\ C' \end{bmatrix} &= \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ -1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 1 & 1 & 1 \end{bmatrix} \\ &= \begin{bmatrix} \frac{1}{\sqrt{2}}+1 & \frac{1}{\sqrt{2}}+1 & 1 \\ -\frac{1}{\sqrt{2}}+1 & \frac{1}{\sqrt{2}}+1 & 1 \\ 1 & \sqrt{2}+1 & 1 \end{bmatrix} \end{aligned}$$

$$\begin{aligned} \text{b) } T \cdot R &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ -1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ -1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & \sqrt{2} & 1 \end{bmatrix} \end{aligned}$$

$$\therefore \begin{bmatrix} A' \\ B' \\ C' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ -1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & \sqrt{2} & 1 \end{bmatrix} = \begin{bmatrix} 1/\sqrt{2} & 3/\sqrt{2} & 1 \\ -1/\sqrt{2} & 3/\sqrt{2} & 1 \\ 0 & 2\sqrt{2} & 1 \end{bmatrix}$$

In the above example, the resultant coordinates of a triangle calculated in part (a) and (b) are not same. This shows that the order in which the transformations are applied is important in the formation of combined or concatenated or composed transformations.

4.4 Composition of 2D Transformations

We have seen what is meant by combined or concatenated or composed transformations in the previous section. The basic purpose of composing transformations is to gain efficiency by applying a single composed transformation to a point, rather than applying a series of transformations, one after the other.

4.4.1 Rotation About an Arbitrary Point

To rotate an object about an arbitrary point, (x_p, y_p) we have to carry out three steps :

1. Translate point (x_p, y_p) to the origin
2. Rotate it about the origin and
3. Finally, translate the center of rotation back where it belongs (See Fig.4.6)

We have already seen that matrix multiplication is not commutative, i.e. multiplying matrix A by matrix B will not always yield the same result as multiplying matrix B by matrix A. Therefore, in obtaining composite transformation matrix, we must be careful to order the matrices so that they correspond to the order of the transformations on the object. Let us find the transformation matrices to carry out individual steps.

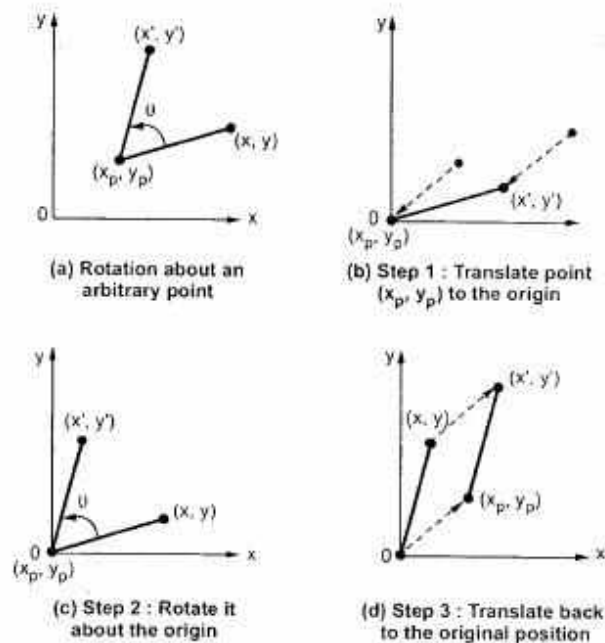


Fig. 4.6

The translation matrix to move point (x_p, y_p) to the origin is given as

$$T_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_p & -y_p & 1 \end{bmatrix}$$

The rotation matrix for counterclockwise rotation of point about the origin is given as

$$R = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The translation matrix to move the center point back to its original position is given as

$$T_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_p & y_p & 1 \end{bmatrix}$$

Therefore, the overall transformation matrix for a counterclockwise rotation by an angle θ about the point (x_p, y_p) is given as

$$\begin{aligned} T_1 \cdot R \cdot T_2 &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_p & -y_p & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_p & y_p & 1 \end{bmatrix} \\ &= \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ -x_p \cos\theta + y_p \sin\theta & -x_p \sin\theta - y_p \cos\theta & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_p & y_p & 1 \end{bmatrix} \\ &= \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ -x_p \cos\theta + y_p \sin\theta + x_p & -x_p \sin\theta - y_p \cos\theta + y_p & 1 \end{bmatrix} \dots (4.18) \end{aligned}$$

Ex. 4.7: Perform a counterclockwise 45° rotation of triangle $A(2, 3)$, $B(5, 5)$, $C(4, 3)$ about point $(1, 1)$.

Sol.: From equation 4.18 we have

$$T_1 \cdot R \cdot T_2 = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ -x_p \cos\theta + y_p \sin\theta + x_p & -x_p \sin\theta - y_p \cos\theta + y_p & 1 \end{bmatrix}$$

Here, $\theta = 45^\circ$, $x_p = 1$ and $y_p = 1$. Substituting values we get

$$T_1 \cdot R \cdot T_2 = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ -1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 1 & -\sqrt{2}+1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} A' \\ B' \\ C' \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 \\ 5 & 5 & 1 \\ 4 & 3 & 1 \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ -1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 1 & -\sqrt{2}+1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} -\frac{1}{\sqrt{2}}+1 & \frac{3}{\sqrt{2}}+1 & 1 \\ 1 & \frac{8}{\sqrt{2}}+1 & 1 \\ \frac{1}{\sqrt{2}}+1 & \frac{5}{\sqrt{2}}+1 & 1 \end{bmatrix}$$

4.5 Other Transformations

The three basic transformations of scaling, rotating, and translating are the most useful and most common. There are some other transformations which are useful in certain applications. Two such transformations are reflection and shear.

4.5.1 Reflection

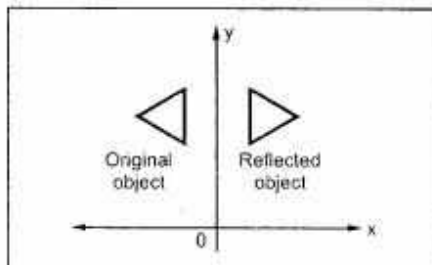


Fig. 4.7 Reflection about y axis

A reflection is a transformation that produces a mirror image of an object relative to an axis of reflection. We can choose an axis of reflection in the xy plane or perpendicular to the xy plane. The table 4.1 gives examples of some common reflections.

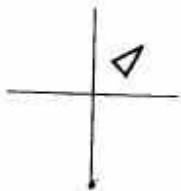


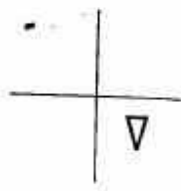
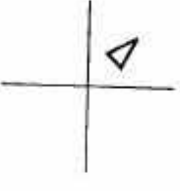

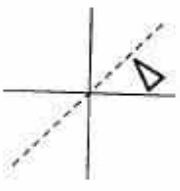
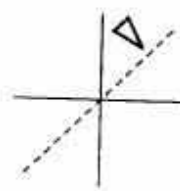

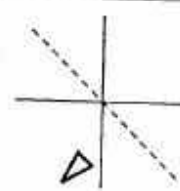
Reflection	Transformation matrix	Original image	Reflected image
Reflection about Y-axis	$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$		
Reflection about X axis	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$		
Reflection about origin	$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$		
Reflection about line $y = x$	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$		
Reflection about line $y = -x$	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$		

Table 4.1 Common reflections

4.5.2 Shear

A transformation that slants the shape of an object is called the shear transformation. Two common shearing transformations are used. One shifts x coordinate values and other shifts y coordinate values. However, in both the cases only one coordinate (x or y) changes its coordinates and other preserves its values.

4.5.2.1 X shear

The x shear preserves the y coordinates, but changes the x values which causes vertical lines to tilt right or left as shown in the Fig. 4.8. The transformation matrix for x shear is given as

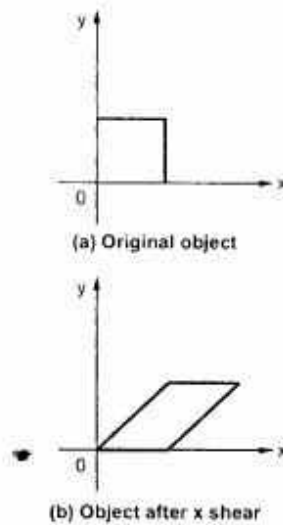


Fig. 4.8

$$X_{sh} = \begin{bmatrix} 1 & 0 & 0 \\ Sh_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\therefore \begin{aligned} x' &= x + Sh_x \cdot y & \text{and} \\ y' &= y \end{aligned} \quad \dots (4.19)$$

4.5.2.2 Y shear

The y shear preserves the x coordinates, but changes the y values which causes horizontal lines to transform into lines which slope up or down, as shown in the Fig. 4.9.

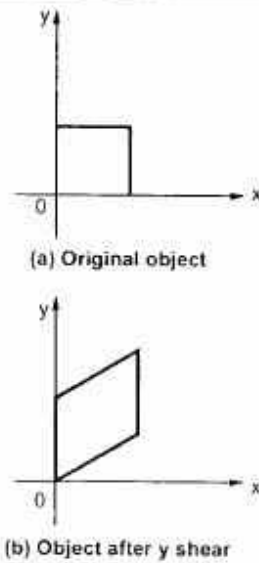


Fig. 4.9

The transformation matrix for y shear is given as

$$Y_{sh} = \begin{bmatrix} 1 & Sh_y & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\therefore x' = x \text{ and } y' = y + Sh_y \cdot x \quad \dots (4.20)$$

4.5.2.3 Shearing Relative to Other Reference Line

We can apply x shear and y shear transformations relative to other reference lines. In x shear transformation we can use y reference line and in y shear we can use x reference line. The transformation matrices for both are given below :

$$\text{x shear with y reference line: } \begin{bmatrix} 1 & 0 & 0 \\ Sh_x & 1 & 0 \\ -Sh_x \cdot y_{ref} & 0 & 1 \end{bmatrix}$$

$$\text{y shear with x reference line: } \begin{bmatrix} 1 & Sh_y & 0 \\ 0 & 1 & 0 \\ 0 & -Sh_y \cdot x_{ref} & 0 \end{bmatrix}$$

Ex. 4.8: Apply the shearing transformation to square with $A(0, 0)$, $B(1, 0)$, $C(1, 1)$ and $D(0, 1)$ as given below

- a) Shear parameter value of 0.5 relative to the line $y_{ref} = -1$
- b) Shear parameter value of 0.5 relative to the line $x_{ref} = -1$

Sol.: a) Here $Sh_x = 0.5$ and $y_{ref} = -1$

$$\begin{bmatrix} A' \\ B' \\ C' \\ D' \end{bmatrix} = \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ Sh_x & 1 & 0 \\ -Sh_x \cdot y_{ref} & 0 & 1 \end{bmatrix}$$

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

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

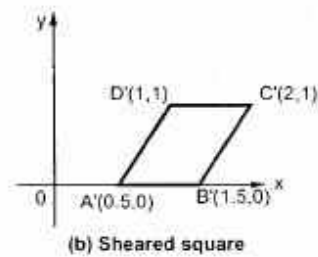
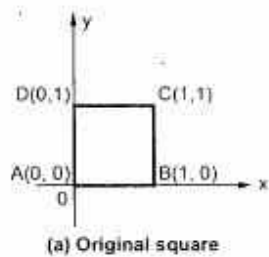


Fig. 4.10

b) Here $Sh_x = 0.5$ and $x_{ref} = -1$

$$\begin{aligned} \begin{bmatrix} A' \\ B' \\ C' \\ D' \end{bmatrix} &= \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} \begin{bmatrix} 1 & Sh_y & 0 \\ 0 & 1 & 0 \\ 0 & -Sh_y \cdot x_{ref} & 1 \end{bmatrix} \\ &= \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0.5 & 0 \\ 0 & 1 & 0 \\ 0 & 0.5 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0.5 & 1 \\ 1 & 1 & 1 \\ 1 & 2 & 1 \\ 0 & 1.5 & 1 \end{bmatrix} \end{aligned}$$

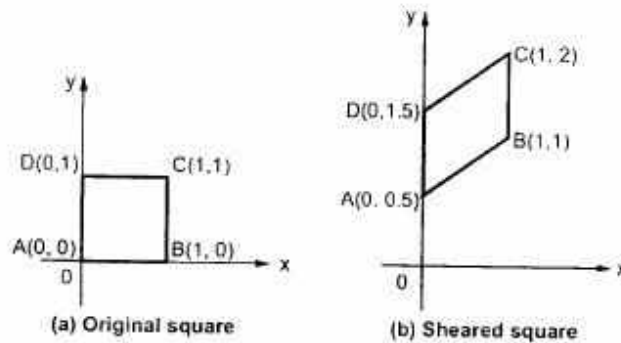


Fig. 4.11

It is important to note that shearing operations can be expressed as sequence of basic transformations. The sequence of basic transformations involve series of rotation and scaling transformations.

Ex. 4.9 : Show how shear transformation may be expressed in terms of rotation and scaling.

Sol. : The shear transformation matrix for x and y combinely can be given as

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

We have scaling matrix and rotation matrix as given below

$$S = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad R = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

If we combine scale matrix and rotation matrix we have,

$$S \cdot R = \begin{bmatrix} S_x \cos\theta & S_x \sin\theta & 0 \\ -S_y \sin\theta & S_y \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Comparing shear matrix and S · R matrix we have

$$Sh_x = -S_y \sin\theta$$

$$Sh_y = S_x \sin\theta$$

$$S_x \cos\theta = 1 \quad \text{and}$$

$$S_y \cos\theta = 1$$

$$\therefore S_x = \frac{1}{\cos\theta} \quad \text{and}$$

$$S_y = \frac{1}{\cos\theta}$$

Substituting values of S_x and S_y we get,

$$Sh_x = -\frac{1}{\cos\theta} \cdot \sin\theta = -\tan\theta$$

$$Sh_y = \frac{1}{\cos\theta} \cdot \sin\theta = \tan\theta$$

Therefore, the shear transformation matrix expressed in terms of rotation and scales is

$$\begin{bmatrix} 1 & \tan\theta & 0 \\ -\tan\theta & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \because S_x \cos\theta = S_y \cos\theta = 1$$

where θ : angle of rotation

S_x : x scale and

S_y : y scale

4.6 Inverse Transformations

When we apply any transformation to point (x, y) we get a new point (x', y') . Sometimes it may require to undo the applied transformation. In such a case we have to get original point (x, y) from the point (x', y') . This can be achieved by inverse transformation. The inverse transformation uses the matrix inverse of the transformation matrix to get the original point (x, y) . The inverse of a matrix is another matrix such that when the two are multiplied together, we get the identity matrix.

If the inverse of matrix T is T^{-1} , then

$$T T^{-1} = T^{-1} T = I \quad \dots (4.21)$$

where I is the identity matrix with all elements along the major diagonal having value 1, and all other elements having value zero.

The elements for the inverse matrix T^{-1} can be calculated from the elements of T as

$$t_{ji}^{-1} = \frac{(-1)^{i+j} \det M_{ij}}{\det T} \quad \dots (4.22)$$

where t_{ji}^{-1} is the element in the i^{th} row and j^{th} column of T^{-1} , and M_{ij} is the $(n-1)$ by $(n-1)$ submatrix obtained by deleting the j^{th} row and i^{th} column of the matrix A . The $\det M_{ij}$ and $\det T$ is the determinant of the M_{ij} and T matrices.

The determinant of a 2×2 matrix is

$$\det \begin{bmatrix} t_{11} & t_{12} \\ t_{21} & t_{22} \end{bmatrix} = t_{11} \cdot t_{22} - t_{12} \cdot t_{21} \quad \dots (4.23)$$

The determinant of a 3×3 matrix is

$$\det T = t_{11} \cdot (t_{22} t_{33} - t_{23} t_{32}) - t_{12} \cdot (t_{21} t_{33} - t_{23} t_{31}) + t_{13} \cdot (t_{21} t_{32} - t_{22} t_{31}) \quad \dots (4.24)$$

In general form, the determinant of T is given by

$$\det T_j = \sum t_{ij} (-1)^{i+j} \det M_{ij} \quad \dots (4.25)$$

where M_{ij} is the submatrix formed by deleting row i and column j from matrix T .

The inverse of the homogeneous coordinate transformation matrix can be given as

$$\begin{bmatrix} a & d & 0 \\ b & e & 0 \\ c & f & 1 \end{bmatrix}^{-1} = \frac{1}{ae - bd} \begin{bmatrix} e & -d & 0 \\ -b & a & 0 \\ bf - ce & cd - af & ae - bd \end{bmatrix}$$

It is important to note that the elements of inverse matrix T^{-1} can be calculated from the element of T as

$$t_{ij}^{-1} = \frac{(-1)^{i+j} \det M_{ji}}{\det T} \quad \dots (4.26)$$

In the above equation the term $\det T$ is in the denominator. Hence, we can obtain an inverse matrix if and only if the determinant of the matrix is nonzero.

Solved Examples

Ex. 4.10: Find out the final coordinates of a figure bounded by the coordinates (1, 1), (3, 4), (5, 7), (10, 3) when rotated about a point (8, 8) by 30° in clockwise direction and scaled by two units in x -direction and three units y direction.

Sol.: From following equation we have the transformation matrix for rotation about an arbitrary point given as

$$T_1 R T_2 = \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ -x_p \cos \theta + y_p \sin \theta + x_p & -x_p \sin \theta - y_p \cos \theta + y_p & 1 \end{bmatrix}$$

In this case, it is clockwise rotation therefore we take value of θ negative.

$$\therefore T_1 \cdot R \cdot T_2 = \begin{bmatrix} 1 & 1 & 1 \\ 3 & 4 & 1 \\ 5 & 7 & 1 \\ 10 & 3 & 1 \end{bmatrix} \begin{bmatrix} \cos(-30) & \sin(-30) & 0 \\ -\sin(-30) & \cos(-30) & 0 \\ -8 \times \cos(-30) + 8 \times \sin(-30) + 8 & -8 \times \sin(-30) - 8 \times \cos(-30) + 8 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1 & 1 \\ 3 & 4 & 1 \\ 5 & 7 & 1 \\ 10 & 3 & 1 \end{bmatrix} \begin{bmatrix} 0.866 & -0.5 & 0 \\ 0.5 & 0.866 & 0 \\ -2.928 & 5.072 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} -1.562 & 5.438 & 1 \\ 1.67 & 7.036 & 1 \\ 4.902 & 8.634 & 1 \\ 7.232 & 2.67 & 1 \end{bmatrix}$$

$$\begin{aligned} \therefore T_1 \cdot R \cdot T_2 \cdot S &= \begin{bmatrix} -1.562 & 5.438 & 1 \\ 1.67 & 7.036 & 1 \\ 4.902 & 8.634 & 1 \\ 7.232 & 2.67 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} -3.124 & 16.314 & 1 \\ 3.34 & 21.108 & 1 \\ 9.804 & 25.902 & 1 \\ 14.464 & 8.01 & 1 \end{bmatrix} \end{aligned}$$

Ex. 4.11: Show that transformation matrix for a reflection about a line $Y = X$ is equivalent to reflection to X -axis followed by counter-clockwise rotation of 90° .

Sol.: The transformation matrix for reflection about a line $Y = X$ is given as

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

The transformation matrix for reflection about x -axis and for counter clockwise rotation of 90 are given as

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} \cos(90) & \sin(90) \\ -\sin(90) & \cos(90) \end{bmatrix}$$

Hence,

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

... Proved

Ex. 4.12: Find out final transformation matrix, when point $P(x, y)$ is to be reflected about a line $y = mx + C$.

Sol.: Equation of line :

$$y = mx + C$$

slope = m y intercept = C

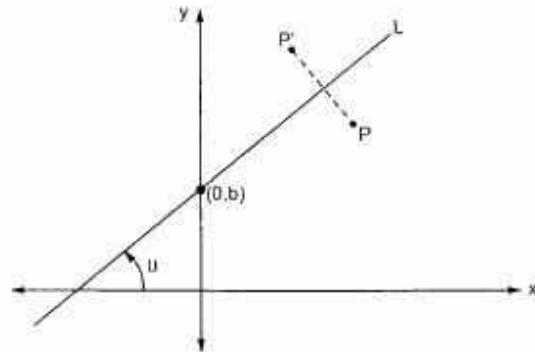


Fig. 4.12

We can relate slope m to angle θ by equation

$$m = \tan \theta$$

$$\therefore \theta = \tan^{-1} m$$

where θ is in inclination of line with respect to x axis.

Translational matrix can be given as

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

Rotational matrix to match the given line with x axis can be obtained as

$$R_{\theta} = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad [\text{Note : angle of rotation} = -\theta]$$

Reflection matrix about x axis

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

Inverse transformation matrices,

$$R_z^{-1} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad T^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & c & 1 \end{bmatrix}$$

∴ Final transformation matrix can be obtained as

$$R_T = T \cdot R_z \cdot M \cdot R_z^{-1} \cdot T^{-1}$$

As we have $\tan\theta = m$, using trigonometry we can obtain

$$\sin\theta = \frac{m}{\sqrt{m^2+1}} \quad \cos\theta = \frac{1}{\sqrt{m^2+1}}$$

$$R_T = \begin{bmatrix} \cos 2\theta & \sin 2\theta & 0 \\ \sin 2\theta & -\cos 2\theta & 0 \\ -c \sin 2\theta & c(1 + \cos 2\theta) & 1 \end{bmatrix}$$

By substituting values of $\sin\theta$ and $\cos\theta$ we have,

$$R_T = \begin{bmatrix} \frac{1-m^2}{m^2+1} & \frac{2m}{m^2+1} & 0 \\ \frac{2m}{m^2+1} & \frac{m^2-1}{m^2+1} & 0 \\ \frac{-2cm}{m^2+1} & \frac{2c}{m^2+1} & 1 \end{bmatrix}$$

Ex. 4.13: Derive the appropriate 2D transformation which reflects a figure in point (0.5, 0.5)

Sol.: Translating given point to origin

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

Now obtaining reflection of the object about origin

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

Translating point back to original position.

$$T^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.5 & 0.5 & 1 \end{bmatrix}$$

The transformation can be given as

$$R_T = T \cdot M \cdot T^{-1}$$

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

Ex. 4.14: Find out the co-ordinates of a figure bounded by $(0, 0)$ $(1, 5)$ $(6, 3)$ $(-3, -4)$ when reflected along the line whose equation is $y = 2x + 4$ and sheared by 2 units in x direction and 2 units in y direction.

Sol.: Equation of the line : $y = 2x + 4$

$$\therefore \text{slope} = 2 \text{ and } y \text{ intercept} = 4$$

$$\therefore \theta = 63.43^\circ$$

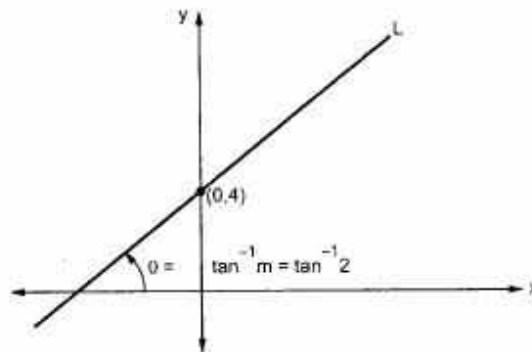


Fig. 4.13

Translational matrix can be given as

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -4 & 1 \end{bmatrix}$$

For matching of given line with x axis we have

$$R_z = \begin{bmatrix} \cos(-63.43) & \sin(-63.43) & 0 \\ -\sin(-63.43) & \cos(-63.43) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R_z = \begin{bmatrix} 0.4472 & -0.8944 & 0 \\ 0.8944 & 0.4472 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

For reflection about x axis we have

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

Inverse transformation matrices are

$$R_z^{-1} = \begin{bmatrix} \cos(-63.43) & -\sin(63.43) & 0 \\ +\sin(-63.43) & \cos(-63.43) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.4472 & 0.8944 & 0 \\ -0.8944 & 0.4472 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$T^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 4 & 1 \end{bmatrix}$$

For shearing along x axis :

$$S_x = \begin{bmatrix} 1 & 0 & 0 \\ Sh_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

For shearing along y axis

$$S_y = \begin{bmatrix} 1 & Sh_y & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The resultant transformation matrix can be obtained by

$$R_T = T \cdot R_y \cdot M \cdot R_y^{-1} \cdot T^{-1} \cdot S_x \cdot S_y$$

Final co-ordinates of the given figure can be obtained by

$$\begin{bmatrix} A' \\ B' \\ C' \\ D' \end{bmatrix} = \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} \cdot R_T$$

Calculations are left for the students as an exercise.

Ex. 4.15: Show that 2D reflection through X axis followed by 2-D reflection through the line $Y = -X$ is equivalent to a pure rotation about the origin.

Sol.: 2D reflection about X axis

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

2D reflection about $Y = -X$

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

∴ Resultant transformation matrix

$$\begin{aligned} R_T &= R_X \cdot R' \\ &= \begin{bmatrix} +1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

For pure rotation about origin we have

$$R_\theta = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ +\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

where θ is angle of rotation

put

$$\theta = 90^\circ$$

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

$$R_T = R_\theta$$

Hence the result

Ex. 4.16 Prove that successive 2D rotations are additive; i.e.

$$R(\theta_1) \cdot R(\theta_2) = R(\theta_1 + \theta_2)$$

Sol.: We can write rotation matrix $R(\theta_1)$ as

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

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

since,

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

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

Ex. 4.17 Prove that 2D rotation and scaling commute if $S_x = S_y$ or $\theta = n\pi$ for integral n and that otherwise they do not. (Dec-99)

Sol.: The matrix notation for scaling along S_x and S_y is as given below

$$S = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \text{ and}$$

The matrix notation for rotation is as given below

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

$$S \cdot R = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

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

$$= \begin{bmatrix} S_x \cos\theta & S_x \sin\theta \\ -S_x \sin\theta & S_x \cos\theta \end{bmatrix} \quad \because S_x = S_y \dots I$$

$$\text{or} \quad = \begin{bmatrix} -S_x & 0 \\ 0 & -S_y \end{bmatrix} \quad \because \theta = n\pi \text{ where } n \text{ is integer} \dots II$$

$$R \cdot S = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix}$$

$$\begin{aligned}
 &= \begin{bmatrix} S_x \cos \theta & S_y \sin \theta \\ -S_x \sin \theta & S_y \cos \theta \end{bmatrix} \\
 &= \begin{bmatrix} S_x \cos \theta & S_x \sin \theta \\ -S_x \sin \theta & S_x \cos \theta \end{bmatrix} & \because S_x = S_y \dots \text{III} \\
 \text{or} &= \begin{bmatrix} -S_x & 0 \\ 0 & -S_y \end{bmatrix} & \because \theta = n\pi \text{ where } n \text{ is integer} \dots \text{IV}
 \end{aligned}$$

From equations I and III, and equations II and IV it is proved that 2D rotation and scaling commute if $S_x = S_y$ or $\theta = n\pi$ for integral n and that otherwise they do not.

Ex. : 4.18 A circular disc of diameter 'd' is rolling down the inclined plane starting from rest as shown below. Assume there is no slip and develop the set of transformation required to produce this animation and also write a program. (Dec-96)

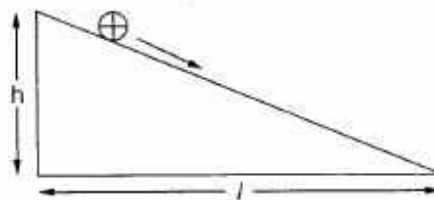


Fig. 4.14

Sol.: For rolling a circular disc of diameter d down the inclined plane starting from rest we have to consider the movement of disc in x direction and in y direction along with the rotation of disc. As length is greater than height we increment x by 1 unit and y by h/l units i.e.

$$x = x + 1$$

$$y = y + h/l$$

The increment of rolling angle can be calculated by relating of a circle with the diagonal length of inclined plane as given below

$$d\theta = \frac{(\sqrt{h^2 + l^2} / \pi d) \times 360}{1}$$

It is necessary to rotate two lines on the disc by $d\theta$ after increment of x and rotation is clockwise. The rotation matrix necessary for this purpose is

2-D Viewing and Clipping

5.1 Introduction

Typically, a graphics package allows us to specify which part of a defined picture is to be displayed and where that part is to be displayed on the display device. Furthermore, the package also provides the use of the scaling, translation and rotation techniques described in the previous chapter to generate a variety of different views of a single picture. We can generate different view of a picture by applying the appropriate scaling and translation. While doing this, we have to identify the visible part of the picture for inclusion in the display image. This selection process is not straight forward. Certain lines may lie partly inside the visible portion of the picture and partly outside. These lines cannot be omitted entirely from the display image because the image would become inaccurate. This is illustrated in Fig. 5.1. The process of selecting and viewing the picture with different views is called **windowing**, and a process which divides each element of the picture into its visible and invisible portions, allowing the invisible portion to be discarded is called **clipping**.

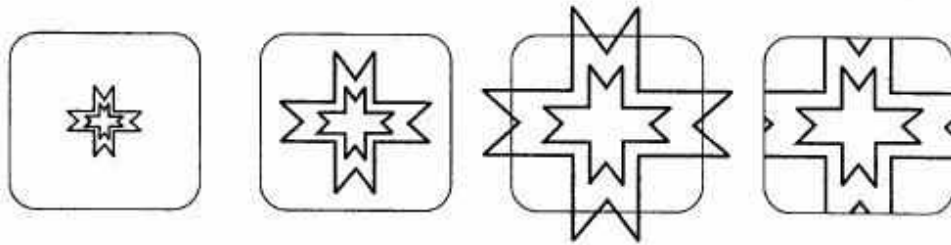


Fig. 5.1

In this chapter we are going to discuss the concepts involved in windowing and various clipping algorithms

5.2 Viewing Transformation

We know that the picture is stored in the computer memory using any convenient cartesian coordinate system, referred to as **world coordinate system (WCS)**. However, when picture is displayed on the display device it is measured in **physical device coordinate system (PDCS)** corresponding to the display device. Therefore, displaying an

image of a picture involves mapping the coordinates of the points and lines that form the picture into the appropriate physical device coordinate where the image is to be displayed. This mapping of coordinates is achieved with the use of coordinate transformation known as **viewing transformation**.

The viewing transformation which maps picture coordinates in the WCS to display coordinates in PDCS is performed by the following transformations :

- Normalization transformation (N) and
- Workstation transformation (W)

5.2.1 Normalization Transformation

We know that, different display devices may have different screen sizes as measured in pixels. Size of the screen in pixels increases as resolution of the screen increases. When picture is defined in the pixel values then it is displayed large in size on the low resolution screen while small in size on the high resolution screen as shown in the Fig. 5.2. To avoid this and to make our programs to be device independent, we have to define the picture coordinates in some units other than pixels and use the interpreter to convert these coordinates to appropriate pixel values for the particular display device. The device independent units are called the **normalized device coordinates**. In these units, the screen measures 1 unit wide and 1 unit length as shown in the Fig. 5.3. The lower left corner of the screen is the origin, and the upper-right corner is the point (1, 1). The point(0.5, 0.5) is the center of the screen no matter what the physical dimensions or resolution of the actual display device may be.

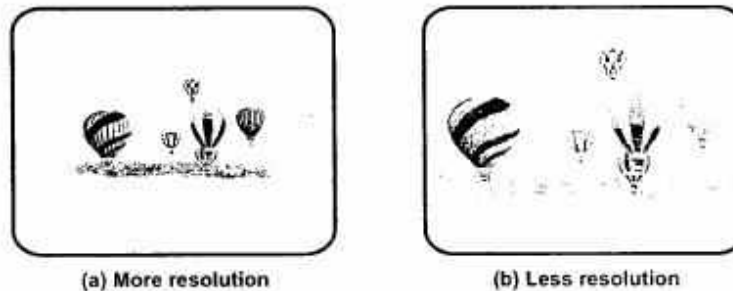


Fig. 5.2 Picture definition in pixels

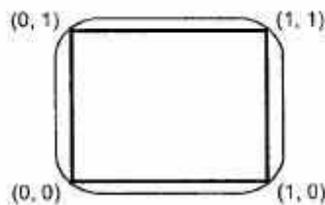


Fig. 5.3 Picture definition in normalized device coordinates

The interpreter uses a simple linear formula to convert the normalized device coordinates to the actual device coordinates.

$$x = x_n \times X_W \quad \dots (5.1)$$

$$y = y_n \times Y_H \quad \dots (5.2)$$

where

- x : Actual device x coordinate
- y : Actual device y coordinate
- x_n : Normalized x coordinate
- y_n : Normalized y coordinate
- X_W : Width of actual screen in pixels
- Y_H : Height of actual screen in pixels.

The transformation which maps the world coordinate to normalized device coordinate is called **normalization transformation**. It involves scaling of x and y, thus it is also referred to as **scaling transformation**.

5.2.2 Workstation Transformation

The transformation which maps the normalized device coordinates to physical device coordinates is called workstation transformation.

The viewing transformation is the combination of normalization transformation and workstation transformations as shown in the Fig. 5.4. It is given as

$$V = N \cdot W \quad \dots (5.3)$$

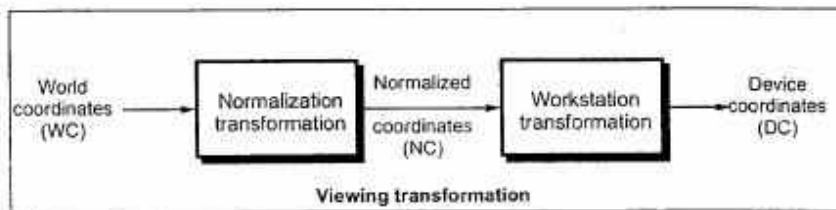


Fig. 5.4 Two dimensional viewing transformation

We know that world coordinate system (WCS) is infinite in extent and the device display area is finite. Therefore, to perform a viewing transformation we select a finite world coordinate area for display called a **window**. An area on a 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, as shown in the Fig. 5.5.

The window defined in world coordinates is first transformed into the normalized device coordinates. The normalized window is then transformed into the viewport coordinate. This window to viewport coordinate transformation is known as workstation transformation. It is achieved by performing following steps :

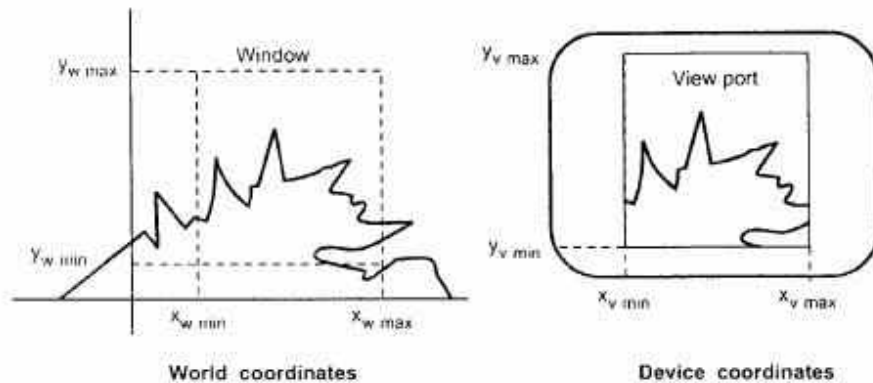


Fig. 5.5 Window and viewport

1. The object together with its window is translated until the lower left corner of the window is at the origin.
2. Object and window are scaled until the window has the dimensions of the viewport.
3. Translate the viewport to its correct position on the screen.

This is illustrated in Fig.5.6

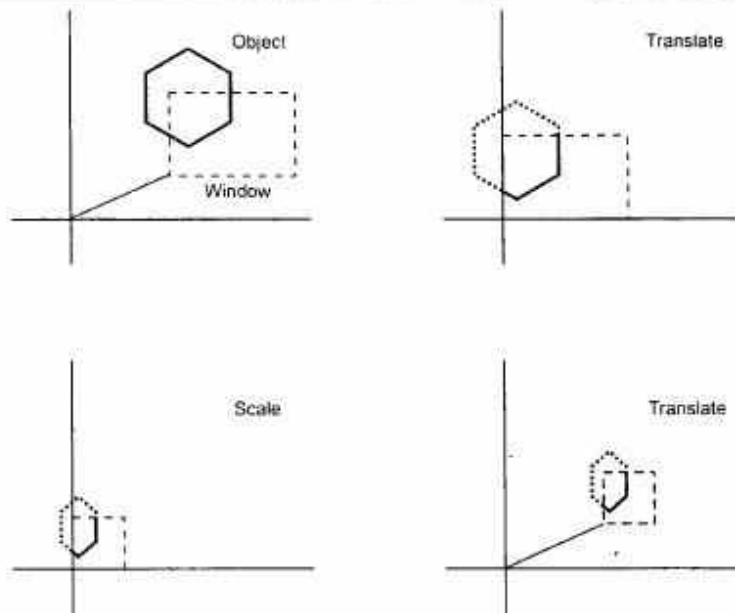


Fig. 5.6 Steps in workstation transformation

Therefore, the workstation transformation is given as

$$W = T \cdot S \cdot T^{-1} \quad \dots (5.4)$$

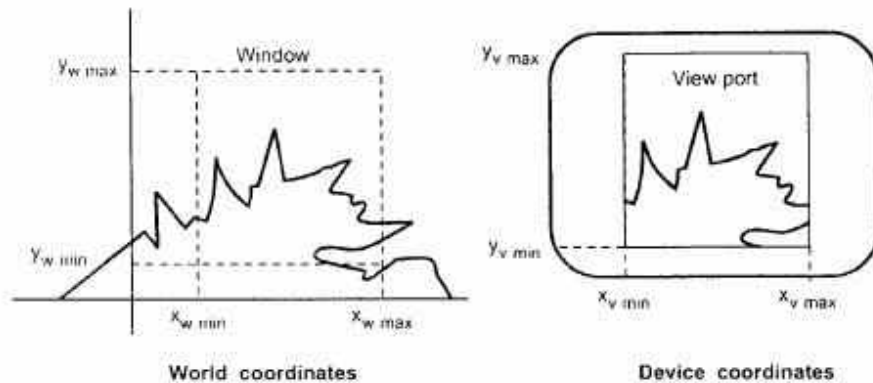


Fig. 5.5 Window and viewport

1. The object together with its window is translated until the lower left corner of the window is at the origin.
2. Object and window are scaled until the window has the dimensions of the viewport.
3. Translate the viewport to its correct position on the screen.

This is illustrated in Fig.5.6

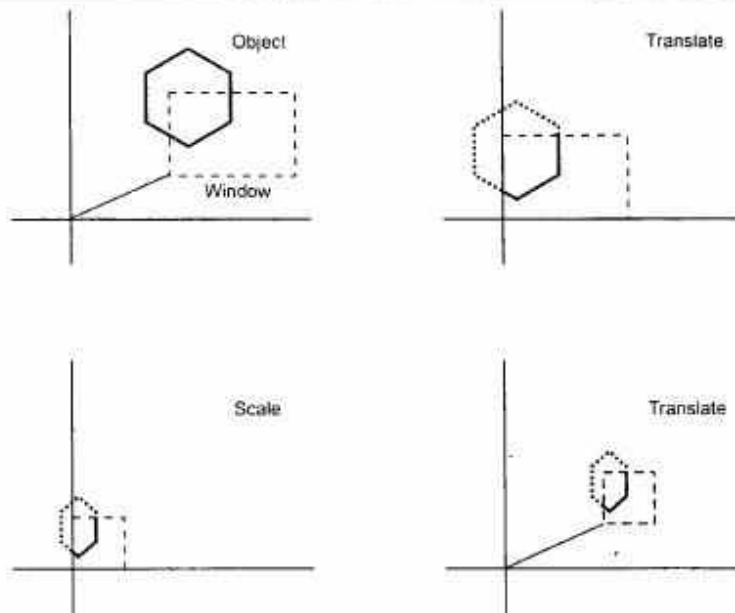


Fig. 5.6 Steps in workstation transformation

Therefore, the workstation transformation is given as

$$W = T \cdot S \cdot T^{-1} \quad \dots (5.4)$$

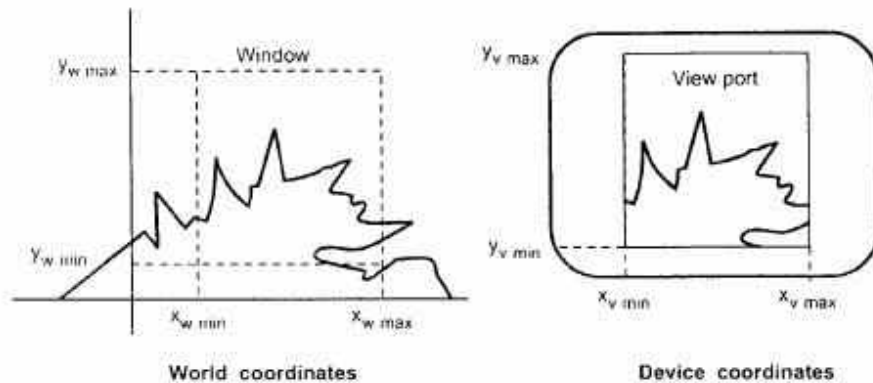


Fig. 5.5 Window and viewport

1. The object together with its window is translated until the lower left corner of the window is at the origin.
2. Object and window are scaled until the window has the dimensions of the viewport.
3. Translate the viewport to its correct position on the screen.

This is illustrated in Fig.5.6

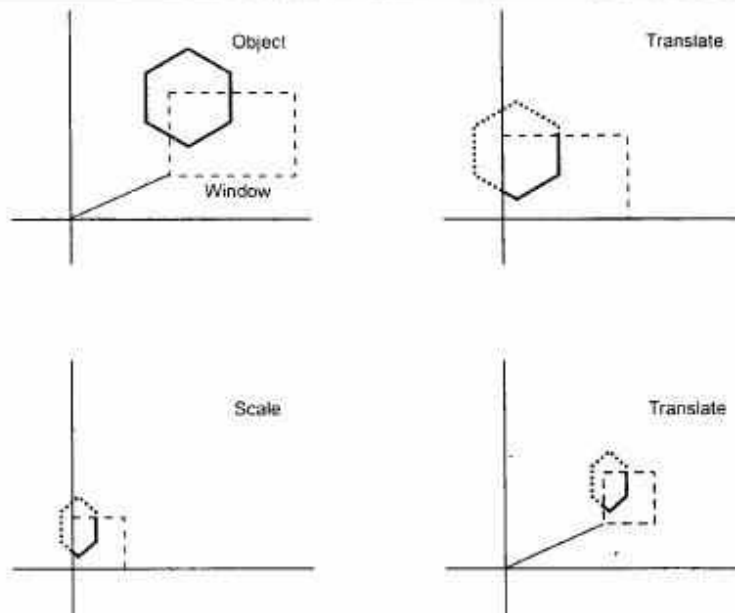


Fig. 5.6 Steps in workstation transformation

Therefore, the workstation transformation is given as

$$W = T \cdot S \cdot T^{-1} \quad \dots (5.4)$$

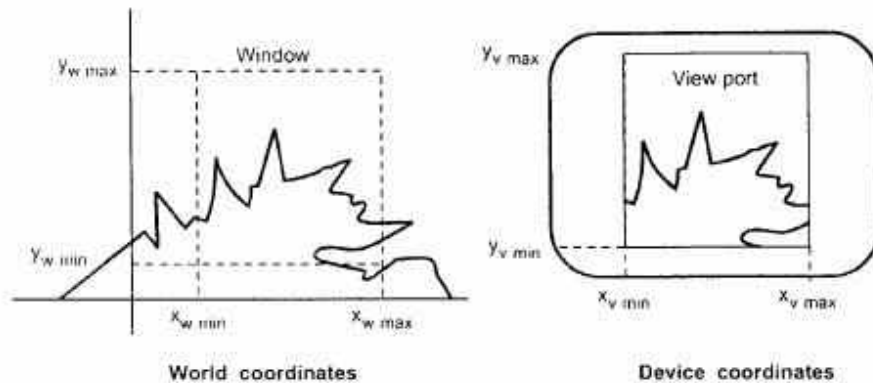


Fig. 5.5 Window and viewport

1. The object together with its window is translated until the lower left corner of the window is at the origin.
2. Object and window are scaled until the window has the dimensions of the viewport.
3. Translate the viewport to its correct position on the screen.

This is illustrated in Fig.5.6

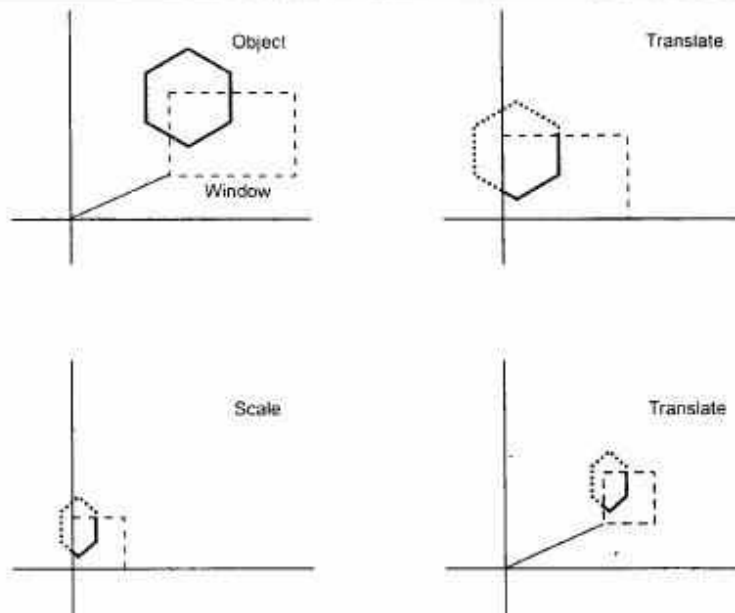


Fig. 5.6 Steps in workstation transformation

Therefore, the workstation transformation is given as

$$W = T \cdot S \cdot T^{-1} \quad \dots (5.4)$$

The transformation matrices for individual transformation are as given below :

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -X_{w \min} & -Y_{w \min} & 1 \end{bmatrix}$$

$$S = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{where} \quad S_x = \frac{x_{v \max} - x_{v \min}}{x_{w \max} - x_{w \min}}$$

$$S_y = \frac{y_{v \max} - y_{v \min}}{y_{w \max} - y_{w \min}}$$

$$T^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_{v \min} & y_{v \min} & 1 \end{bmatrix}$$

The overall transformation matrix for W is given as

$$W = T \cdot S \cdot T^{-1}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -X_{w \min} & -Y_{w \min} & 1 \end{bmatrix} \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_{v \min} & y_{v \min} & 1 \end{bmatrix}$$

$$= \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ x_{v \min} - x_{w \min} \cdot S_x & y_{v \min} - y_{w \min} \cdot S_y & 1 \end{bmatrix}$$

The Fig. 5.7 shows the complete viewing transformation.

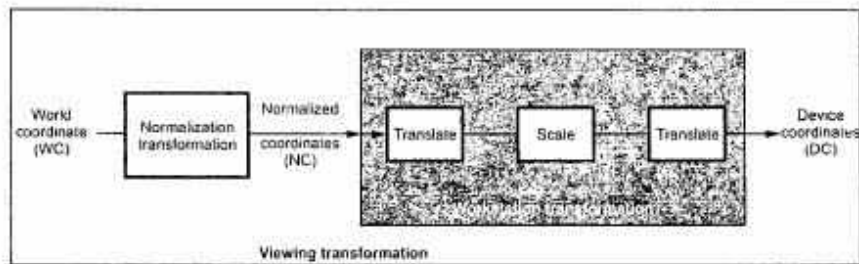


Fig. 5.7 Viewing transformation

Ex. 5.1 : Find the normalization transformation window to viewpoint, with window, lower left corner at (1, 1) and upper right corner at (3, 5) onto a viewpoint with lower left corner at (0, 0) and upper right corner at (1/2, 1/2).

Sol. : Given : Coordinates for window

$$\begin{aligned}x_{w \min} &= 1 & y_{w \min} &= 1 \\x_{w \max} &= 3 & y_{w \max} &= 5\end{aligned}$$

Coordinates for view port

$$\begin{aligned}x_{v \min} &= 0 & y_{v \min} &= 0 \\x_{v \max} &= 1/2 = 0.5 & y_{v \max} &= 1/2 = 0.5\end{aligned}$$

We know that,

$$\begin{aligned}S_x &= \frac{x_{v \max} - x_{v \min}}{x_{w \max} - x_{w \min}} \\&= \frac{0.5 - 0}{3 - 1} \\&= 0.25\end{aligned}$$

and

$$\begin{aligned}S_y &= \frac{y_{v \max} - y_{v \min}}{y_{w \max} - y_{w \min}} \\&= \frac{0.5 - 0}{5 - 1} \\&= 0.125\end{aligned}$$

We know that transformation matrix is given as

$$\begin{aligned}T \cdot S \cdot T^{-1} &= \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ x_{v \min} - x_{w \min} S_x & y_{v \min} - y_{w \min} S_y & 1 \end{bmatrix} \\&= \begin{bmatrix} 0.25 & 0 & 0 \\ 0 & 0.125 & 0 \\ 0 - (1 \times 0.25) & 0 - (1 \times 0.125) & 1 \end{bmatrix} \\&= \begin{bmatrix} 0.25 & 0 & 0 \\ 0 & 0.125 & 0 \\ -0.25 & -0.125 & 1 \end{bmatrix}\end{aligned}$$

5.3 2D Clipping

The procedure that identifies the portions of a picture that are either inside or outside of a specified region of space is referred to as clipping. The region against which an object is to be clipped is called a **clip window** or **clipping window**. It usually is in a rectangular shape, as shown in the Fig. 5.8.

The clipping algorithm determines which points, lines or portions of lines lie within the clipping window. These points, lines or portions of lines are retained for display. All others are discarded.

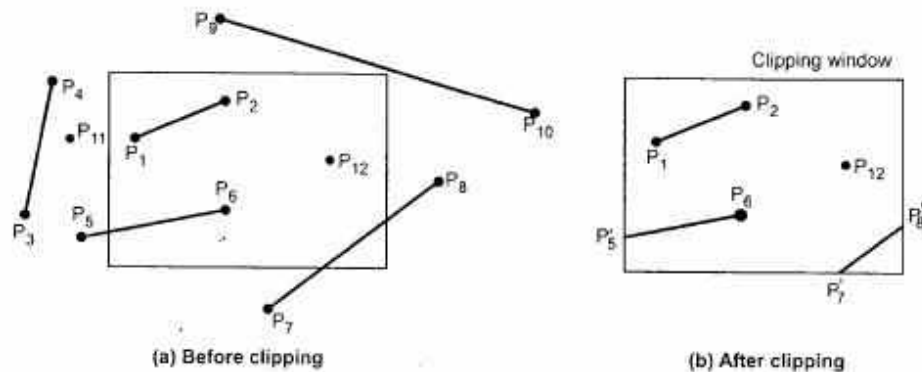


Fig. 5.8

5.3.1 Point Clipping

The points are said to be interior to the clipping window if

$$x_{w \min} \leq x \leq x_{w \max} \quad \text{and} \\ y_{w \min} \leq y \leq y_{w \max}$$

The equal sign indicates that points on the window boundary are included within the window.

5.3.2 Line Clipping

The lines are said to be interior to the clipping window and hence visible if both end points are interior to the window, e.g., line $P_1 P_2$ in Fig. 5.8. However, if both end points of a line are exterior to the window, the line is not necessarily completely exterior to the window, e.g., line $P_7 P_8$ in Fig. 5.8. If both end points of a line are completely to the right of, completely to the left of, completely above, or completely below the window, then the line is completely exterior to the window and hence invisible. For example, line $P_3 P_4$ in Fig. 5.8.

The lines which across one or more clipping boundaries require calculation of multiple intersection points to decide the visible portion of them. To minimize the intersection calculations and to increase the efficiency of the clipping algorithm, initially, completely visible and invisible lines are identified and then the intersection points are calculated for remaining lines. There are many line clipping algorithms. Let us discuss a few of them.

5.3.2.1 Sutherland and Cohen Subdivision Line Clipping Algorithm

This is one of the oldest and most popular line clipping algorithm developed by Dan Cohen and Ivan Sutherland. To speed up the processing this algorithm performs initial tests that reduce the number of intersections that must be calculated. This algorithm uses a four digit (bit) code to indicate which of nine regions contain the end point of line. The four bit codes are called **region codes** or **outcodes**. These codes identify the location of the point relative to the boundaries of the clipping rectangle as shown in the Fig. 5.9.

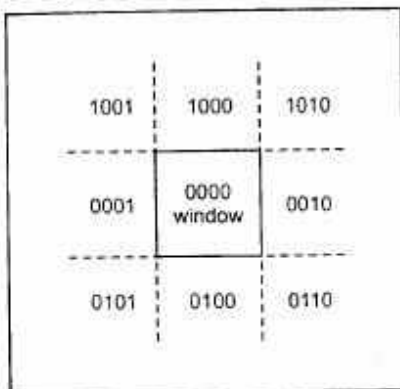


Fig. 5.9 Four-bit codes for nine regions

Each bit position in the region code is used to indicate one of the four relative coordinate positions of the point with respect to the clipping window : to the left, right, top or bottom. The rightmost bit is the first bit and the bits are set to 1 based on the following scheme :

Set Bit 1 - if the end point is to the **left** of the window

Set Bit 2 - if the end point is to the **right** of the window

Set Bit 3 - if the end point is **below** the window

Set Bit 4 - if the end point is **above** the window

Otherwise, the bit is set to zero.

Once we have established region codes for all the line endpoints, we can determine which lines are completely inside the clipping window and which are clearly outside. Any lines that are completely inside the window boundaries have a region code of 0000 for both endpoints and we trivially accept these lines. Any lines that have a 1 in the same bit position in the region codes for each endpoint are completely outside the clipping rectangle, and we trivially reject these lines. A method used to test lines for total clipping is equivalent to the logical **AND** operator. If the result of the logical **AND** operation with two end point codes is not 0000, the line is completely outside the clipping region. The lines that cannot be identified as completely inside or completely outside a clipping window by these tests are checked for intersection with the window boundaries.

Ex. 5.2: Consider the clipping window and the lines shown in Fig. 5.10. Find the region codes for each end point and identify whether the line is completely visible, partially visible or completely invisible.

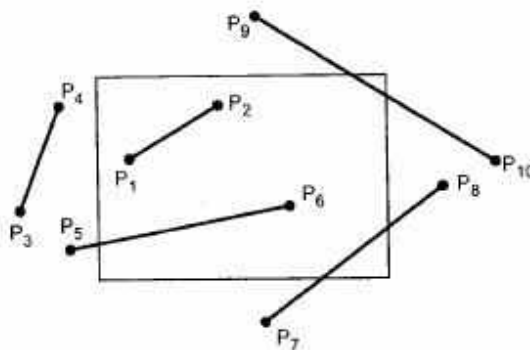


Fig. 5.10

Sol.: The Fig. 5.11 shows the clipping window and lines with region codes. These codes are tabulated and end point codes are logically ANDed to identify the visibility of the line in table 5.1.

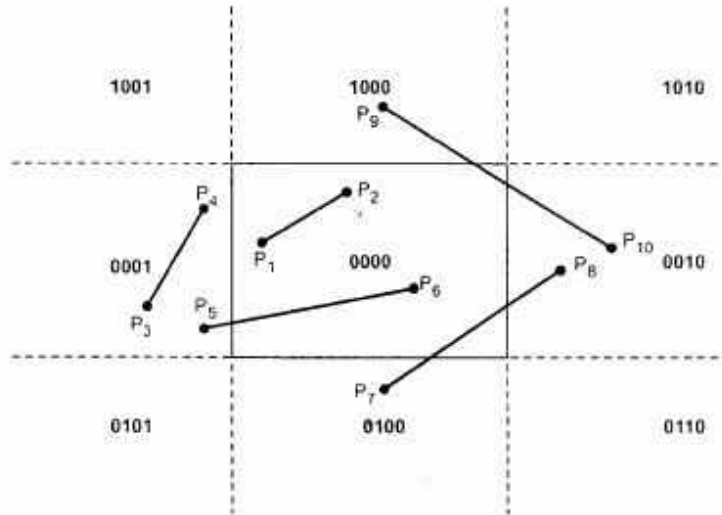


Fig. 5.11

Line	End Point Codes		Logical ANDing	Result
P ₁ P ₂	0000	0000	0000	Completely visible
P ₃ P ₄	0001	0001	0001	Completely invisible
P ₅ P ₆	0001	0000	0000	Partially visible
P ₇ P ₈	0100	0010	0000	Partially visible
P ₉ P ₁₀	1000	0010	0000	Partially visible

Table 5.1

The Sutherland - Cohen algorithm begins the clipping process for a partially visible line by comparing an outside endpoint to a clipping boundary to determine how much of the line can be discarded. Then the remaining part of the line is checked against the other boundaries, and the process is continued until either the line is totally discarded or a section is found inside the window.

This is illustrated in Fig. 5.12.

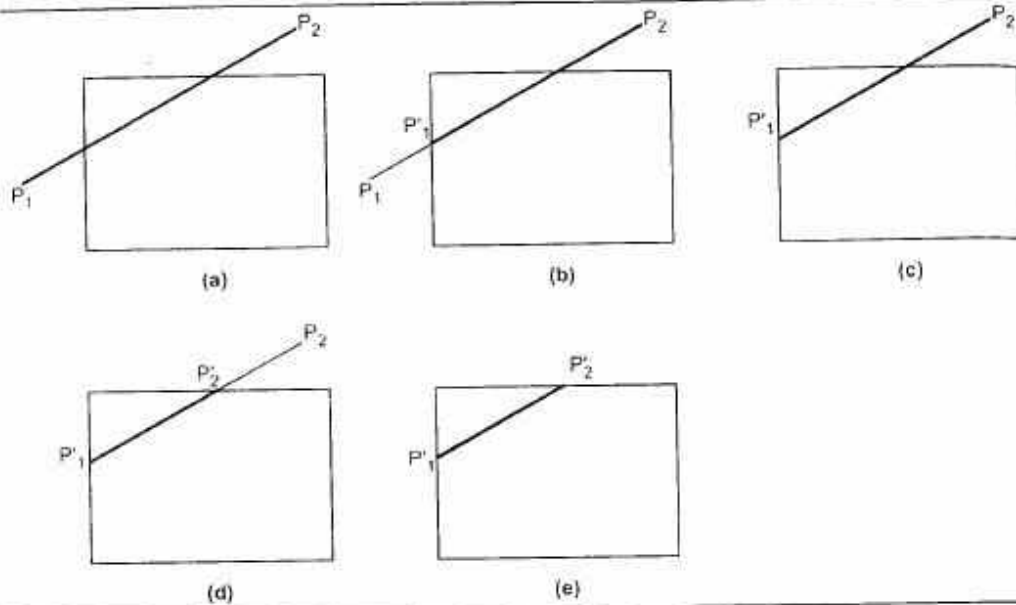


Fig. 5.12 Sutherland-Cohen subdivision line clipping

As shown in the Fig. 5.12, line $P_1 P_2$ is a partially visible and point P_1 is outside the window. Starting with point P_1 , the intersection point P'_1 is found and we get two line segments $P_1 - P'_1$ and $P'_1 - P_2$. We know that, for $P_1 - P'_1$ one end point i.e. P_1 is outside the window and thus the line segment $P_1 - P'_1$ is discarded. The line is now reduced to the section from P'_1 to P_2 . Since P_2 is outside the clip window, it is checked against the boundaries and intersection point P'_2 is found. Again the line segment is divided into two segments giving $P'_1 - P'_2$ and $P'_2 - P_2$. We know that, for $P'_2 - P_2$ one end point i.e. P_2 is outside the window and thus the line segment $P'_2 - P_2$ is discarded. The remaining line segment $P'_1 - P'_2$ is completely inside the clipping window and hence made visible.

The intersection points with a clipping boundary can be calculated using the slope-intercept form of the line equation. The equation for line passing through points $P_1 (x_1, y_1)$ and $P_2 (x_2, y_2)$ is

$$y = m(x - x_1) + y_1 \quad \text{or} \quad y = m(x - x_2) + y_2 \quad \dots (5.5)$$

where $m = \frac{y_2 - y_1}{x_2 - x_1}$ (slope of the line)

Therefore, the intersections with the clipping boundaries of the window are given as :

Left : $x_L, y = m(x_L - x_1) + y_1$; $m \neq \infty$

Right : $x_R, y = m(x_R - x_1) + y_1$; $m \neq \infty$

Top : $y_T, x = x_1 + \left(\frac{1}{m}\right)(y_T - y_1)$; $m \neq 0$

Bottom : $y_B, x = x_1 + \left(\frac{1}{m}\right)(y_B - y_1)$; $m \neq 0$

Sutherland and Cohen subdivision line clipping algorithm :

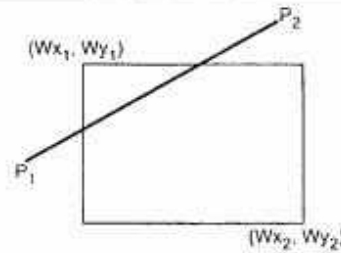


Fig. 5.13

1. Read two end points of the line say $P_1 (x_1, y_1)$ and $P_2 (x_2, y_2)$.
2. Read two corners (left-top and right-bottom) of the window, say (Wx_1, Wy_1) and (Wx_2, Wy_2) .
3. Assign the region codes for two endpoints P_1 and P_2 using following steps :
 Initialize code with bits 0000
 Set Bit 1 - if $(x < Wx_1)$
 Set Bit 2 - if $(x > Wx_2)$
 Set Bit 3 - if $(y < Wy_2)$
 Set Bit 4 - if $(y > Wy_1)$
4. Check for visibility of line $P_1 P_2$
 - a) If region codes for both endpoints P_1 and P_2 are zero then the line is completely visible. Hence draw the line and go to step 9.
 - b) If region codes for endpoints are not zero and the logical ANDing of them is also nonzero then the line is completely invisible, so reject the line and go to step 9.
 - c) If region codes for two endpoints do not satisfy the conditions in 4a) and 4b) the line is partially visible.
5. Determine the intersecting edge of the clipping window by inspecting the region codes of two endpoints.
 - a) If region codes for both the end points are non-zero, find intersection points P_1' and P_2' with boundary edges of clipping window with respect to point P_1 and point P_2 , respectively
 - b) If region code for any one end point is non zero then find intersection point P_1' or P_2' with the boundary edge of the clipping window with respect to it.
6. Divide the line segments considering intersection points.
7. Reject the line segment if any one end point of it appears outside the clipping window.
8. Draw the remaining line segments.
9. Stop.

```

k= (float)p1.x+(float)(y-p1.y)/m;
temp.x=k;
temp.y=y;
for(i=0;i<4;i++)
temp.code[i]=p1.code[i];
return(temp);
}
else
return(p1);
}

```

5.3.2.2 Midpoint Subdivision Algorithm

We have seen that, the Sutherland Cohen subdivision line clipping algorithm requires the calculation of the intersection of the line with the window edge. These calculations can be avoided by repeatedly subdividing the line at its midpoint.

Like previous algorithm, initially the line is tested for visibility. If line is completely visible it is drawn and if it is completely invisible it is rejected. If line is partially visible then it is subdivided in two equal parts. The visibility tests are then applied to each half. This subdivision process is repeated until we get completely visible and completely invisible line segments. This is illustrated in Fig. 5.14. (see on next page)

As shown in the Fig. 5.14, line P_1P_2 is partially visible. It is subdivided in two equal parts P_1P_3 and P_3P_2 (see Fig. 5.14(b)). Both the line segments are tested for visibility and found to be partially visible. Both line segments are then subdivided in two equal parts to get midpoints P_4 and P_5 (see Fig. 5.14 (c)). It is observed that line segments P_1P_4 and P_5P_2 are completely invisible and hence rejected. However, line segment P_3P_5 is completely visible and hence drawn. The remaining line segment P_4P_3 is still partially visible. It is then subdivided to get midpoint P_6 . It is observed that P_6P_3 is completely visible whereas P_4P_6 is partially visible. Thus P_6P_3 line segment is drawn and P_4P_6 line segment is further subdivided into equal parts to get midpoint P_7 . Now, it is observed that line segment P_4P_7 is completely invisible and line segment P_7P_6 is completely visible (see Fig. 5.14 (f)), and there is no further partially visible segment.

Midpoint Subdivision Algorithm :

1. Read two endpoints of the line say $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$.
2. Read two corners (left-top and right-bottom) of the window, say (Wx_1, Wy_1) and (Wx_2, Wy_2) .
3. Assign region codes for two end points using following steps :
 - Initialize code with bits 0000
 - Set Bit 1 - if $(x < Wx_1)$
 - Set Bit 2 - if $(x > Wx_2)$
 - Set Bit 3 - if $(y < Wy_1)$
 - Set Bit 4 - if $(y > Wy_2)$

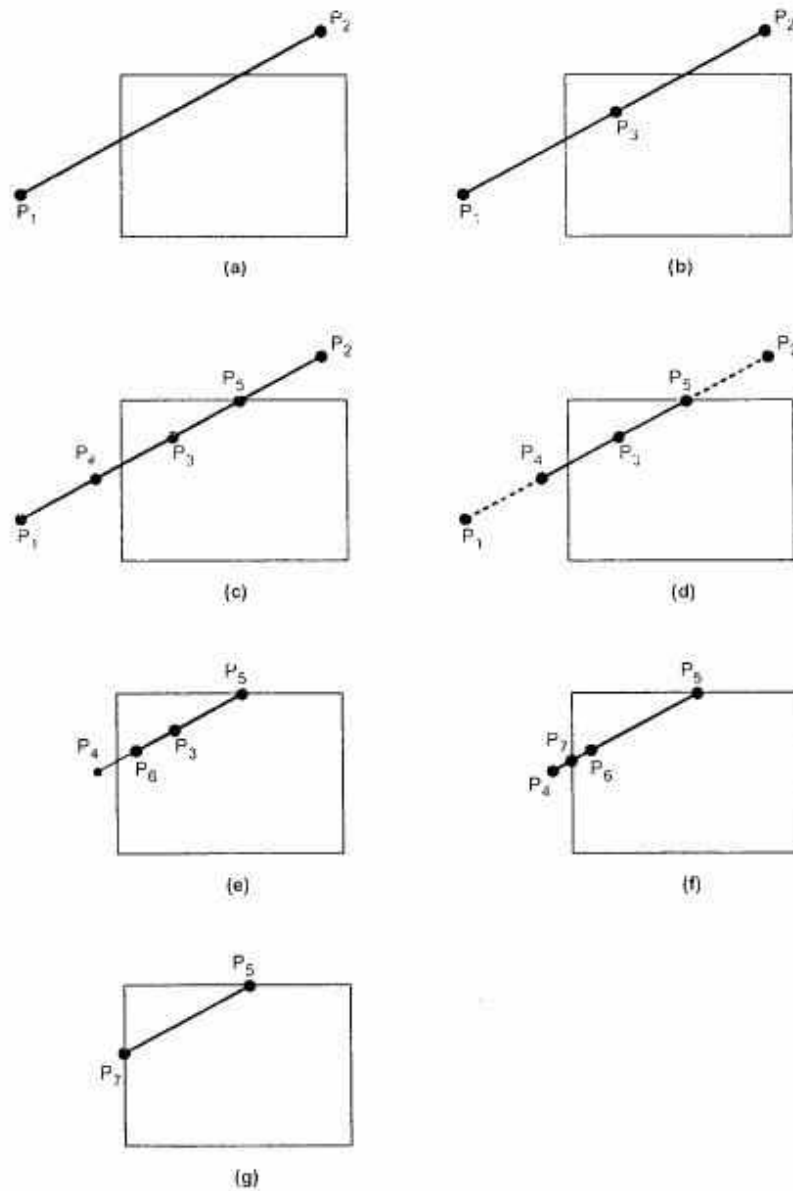


Fig. 5.14 Clipping line with midpoint subdivision algorithm

4. Check for visibility of line
 - a) If region codes for both endpoints are zero then the line is completely visible. Hence draw the line and go to step 6.
 - b) If region codes for endpoints are not zero and the logical ANDing of them is also nonzero then the line is completely invisible, so reject the line and go to step 6.
 - c) If region codes for two endpoints do not satisfy the conditions in 4a) and 4b) the line is partially visible.
5. Divide the partially visible line segment in equal parts and repeat steps 3 through 5 for both subdivided line segments until you get completely visible and completely invisible line segments.
6. Stop.

'C' code for Midpoint Subdivision Line Clipping Algorithm

(Softcopy of this program is available at vtubooks.com)

```

#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<con.h>
#include<math.h>
#include<graphics.h>
/* Defining structure for end point of line */
typedef struct coordinate
{
int x,y;
char code[4];
}PT;
void drawwindow();
void drawline (PT p1,PT p2,int cl);
PT setcode(PT p);
int visibility (PT p1,PT p2);
PT resetendpt (PT p1,PT p2);
main()
{
int gd=DETECT, gm,v;
PT p1,p2,ptemp;
initgraph (&gd,&gm, " ");
cleardevice();
printf("\n\n\t\tENTER END-POINT 1 (x,y): ");
scanf ("%d,%d",&p1.x,&p1.y);

```

Thus, the two intersection points to line P_1P_2 are [1.2 1.8] and [5 2.75] with edges V_1V_2 and V_5V_6 , respectively.

5.3.2.4 Liang-Barsky Line Clipping Algorithm

In the last section we have seen Cyrus-Beck line clipping algorithm using parametric equations. It is more efficient than Cohen-Sutherland algorithm. Liang and Barsky have developed even more efficient algorithm than Cyrus-Beck algorithm using parametric equations. These parametric equations are given as

$$x = x_1 + t\Delta x$$

$$y = y_1 + t\Delta y, \quad 0 \leq t \leq 1$$

where

$$\Delta x = x_2 - x_1 \text{ and } \Delta y = y_2 - y_1$$

The point clipping conditions (Refer section 5.3.1) for Liang-Barsky approach in the parametric form can be given as

$$x_{wmin} \leq x_1 + t\Delta x \leq x_{wmax} \text{ and}$$

$$y_{wmin} \leq y_1 + t\Delta y \leq y_{wmax}$$

Liang-Barsky express these four inequalities with two parameters p and q as follows :

$$tp_i \leq q_i \quad i = 1, 2, 3, 4$$

where parameters p and q are defined as

$$p_1 = -\Delta x, \quad q_1 = x_1 - x_{wmin}$$

$$p_2 = \Delta x, \quad q_2 = x_{wmax} - x_1$$

$$p_3 = -\Delta y, \quad q_3 = y_1 - y_{wmin}$$

$$p_4 = \Delta y, \quad q_4 = y_{wmax} - y_1$$

Following observations can be easily made from above definitions of parameters p and q.

q.

If $p_1 = 0$: Line is parallel to left clipping boundary.

If $p_2 = 0$: Line is parallel to right clipping boundary.

If $p_3 = 0$: Line is parallel to bottom clipping boundary.

If $p_4 = 0$: Line is parallel to top clipping boundary.

If $p_i = 0$, and for that value of i,

If $q_i < 0$: Line is completely outside the boundary and can be eliminated.

If $q_i \geq 0$: Line is inside the clipping boundary.

If $p_i < 0$: Line proceeds from outside to inside of the clipping boundary.

If $p_i > 0$: Line proceeds from inside to outside of the clipping boundary.

Therefore, for nonzero value of p_i , the line crosses the clipping boundary and we have to find parameter t. The parameter t for any clipping boundary i can be given as

$$t = \frac{q_i}{p_i} \quad i = 1, 2, 3, 4$$

Liang-Barsky algorithm calculates two values of parameter t : t_1 and t_2 that define that part of the line that lies within the clip rectangle. The value of t_1 is determined by checking

the rectangle edges for which the line proceeds from the outside to the inside ($p < 0$). The value of t_1 is taken as a largest value amongst various values of intersections with all edges. On the other hand, the value of t_2 is determined by checking the rectangle edges for which the line proceeds from the inside to the outside ($p > 0$). The minimum of the calculated value is taken as a value for t_2 .

Now, if $t_1 > t_2$, the line is completely outside the clipping window and it can be rejected. Otherwise the values of t_1 and t_2 are substituted in the parametric equations to get the end points of the clipped line.

Algorithm

1. Read two endpoints of the line say $p_1 (x_1, y_1)$ and $p_2 (x_2, y_2)$.
2. Read two corners (left-top and right-bottom) of the window, say $(x_{wmin}, y_{wmin}, x_{wmax}, y_{wmax})$.
3. Calculate the values of parameters p_i and q_i for $i = 1, 2, 3, 4$ such that

$$p_1 = -\Delta x \quad q_1 = x_1 - x_{wmin}$$

$$p_2 = \Delta x \quad q_2 = x_{wmax} - x_1$$

$$p_3 = -\Delta y \quad q_3 = y_1 - y_{wmin}$$

$$p_4 = \Delta y \quad q_4 = y_{wmax} - y_1$$
4. if $p_i = 0$, then
 - [The line is parallel to i^{th} boundary.
 - Now, if $q_i < 0$ then
 - [line is completely outside the boundary, hence discard the line segment and goto stop.
 -] else
 - [Check whether the line is horizontal or vertical and accordingly check the line endpoint with corresponding boundaries. If line endpoint/s lie within the bounded area then use them to draw line otherwise use boundary coordinates to draw line. Go to stop.
5. Initialise values for t_1 and t_2 as

$$t_1 = 0 \text{ and } t_2 = 1$$
6. Calculate values for q_i/p_i for $i = 1, 2, 3, 4$.
7. Select values of q_i/p_i where $p_i < 0$ and assign maximum out of them as t_1 .
8. Select values of q_i/p_i where $p_i > 0$ and assign minimum out of them as t_2 .
9. If $(t_1 < t_2)$
 - [Calculate the endpoints of the clipped line as follows :

$$xx_1 = x_1 + t_1 \Delta x$$

```

    xx2 = x1 + t2 Δx
    yy1 = y1 + t1 Δy
    yy2 = y1 + t2 Δy
    Draw line (xx1, yy1, xx2, yy2)
  |
10. Stop.

```

'C' code for Liang-Barsky Line Clipping Algorithm

(Softcopy of this program is available at vtubooks.com)

```

#include<stdio.h>
#include<graphics.h>
#include<math.h>
main()
|
int i,gd,gm;
int x1,y1,x2,y2,xmin,xmax,ymin,ymax,xx1,xx2,yy1,yy2;
float t1,t2,p[4],q[4],temp;
detectgraph(&gd,&gm);
initgraph(&gd,&gm,"");
x1=10;
y1=10;
x2=60;
y2=30;
xmin = 15;
xmax = 25;
ymin = 15;
ymax = 25;

rectangle(xmin,ymin,xmax,ymax);
p[0] = -(x2-x1);
p[1] = (x2-x1);
p[2] = -(y2-y1);
p[3] = (y2-y1);
q[0] = (x1-xmin);
q[1] = (xmax-x1);
q[2] = (y1-ymin);
q[3] = (ymax-y1);

```

```

    }
    temp = q[i]/p[i];
    if(p[i] < 0)
    {
        if(t1 <= temp)
        {
            t1 = temp;
        }
    }
    else
    {
        if(t2 > temp)
        {
            t2 = temp;
        }
    }
}
if(t1 < t2)
{
    xx1 = x1 + t1 * p[1];
    xx2 = x1 + t2 * p[1];
    yy1 = y1 + t1 * p[3];
    yy2 = y1 + t2 * p[3];
    line(xx1, yy1, xx2, yy2);
}
getch();
closegraph();
}

```

Advantages

1. It is more efficient than Cohen-Sutherland algorithm, since intersection calculations are reduced.
2. It requires only one division to update parameters t_1 and t_2 .
3. Window intersections of the line are computed only once.

Ex. 5.5 Find the clipping coordinates for a line p_1p_2 where $p_1 = (10, 10)$ and $p_2 = (60, 30)$, against window with $(x_{\min}, y_{\min}) = (15, 15)$ and $(x_{\max}, y_{\max}) = (25, 25)$.

Sol.: Here,

$$\begin{array}{ll}
 x_1 = 10 & x_{\min} = 15 \\
 y_1 = 10 & y_{\min} = 15
 \end{array}$$

$$\begin{array}{lll}
 x_2 = 60 & x_{wmax} = 25 & \\
 y_2 = 30 & y_{wmax} = 25 & \\
 p_1 = -50 & q_1 = -5 & p_1/q_1 = 0.1 \\
 p_2 = 50 & q_2 = 15 & p_2/q_2 = 0.3 \\
 p_3 = -20 & q_3 = -5 & p_3/q_3 = 0.25 \\
 p_4 = 20 & q_4 = 15 & p_4/q_4 = 0.75 \\
 t_1 = \max(0.25, 0.1) = 0.25 & & \text{since for these values } p < 0 \\
 t_2 = \min(0.3, 0.75) = 0.3 & & \text{since for these values } p > 0
 \end{array}$$

Here, $t_1 < t_2$ and the endpoints of clipped line are :

$$\begin{aligned}
 xx_1 &= x_1 + t_1 \Delta x \\
 &= 10 + 0.25 \times 50 \\
 &= 22.5 \\
 yy_1 &= y_1 + t_1 \Delta y \\
 &= 10 + 0.25 \times 20 \\
 &= 15 \\
 xx_2 &= x_1 + t_2 \Delta x \\
 &= 10 + 0.3 \times 50 \\
 &= 25 \\
 yy_2 &= y_1 + t_2 \Delta y \\
 &= 10 + 0.3 \times 20 \\
 &= 16
 \end{aligned}$$

5.4 Polygon Clipping

In the previous sections we have seen line clipping algorithms. A polygon is nothing but the collection of lines. Therefore, we might think that line clipping algorithm can be used directly for polygon clipping. However, when a closed polygon is clipped as a collection of lines with line clipping algorithm, the original closed polygon becomes one or more open polygon or discrete lines as shown in the Fig. 5.19. Thus, we need to modify the line clipping algorithm to clip polygons.

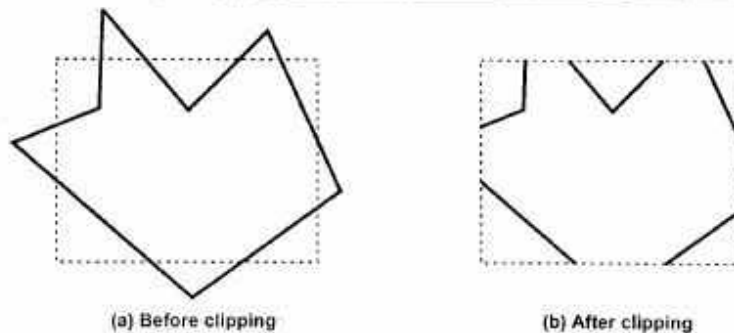


Fig. 5.19 Polygon clipping done by line clipping algorithm

We consider a polygon as a closed solid area. Hence after clipping it should remain closed. To achieve this we require an algorithm that will generate additional line segment which make the polygon as a closed area. For example, in Fig. 5.20 the lines $a - b$, $c - d$, $d - e$, $f - g$, and $h - i$ are added to polygon description to make it closed.

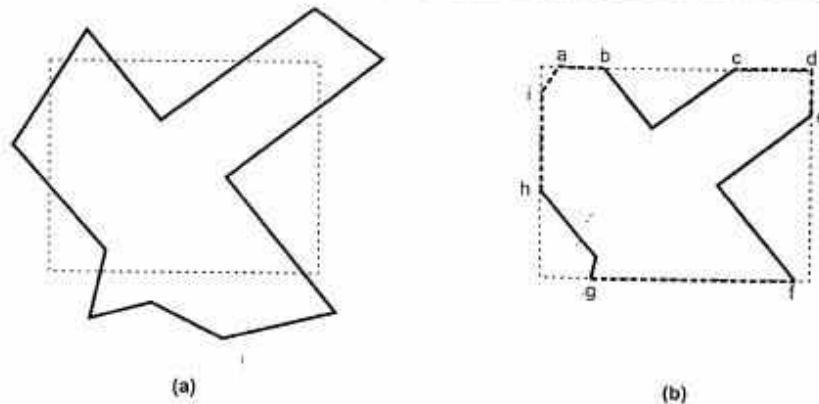


Fig. 5.20 Modifying the line clipping algorithm for polygon

Adding lines $c - d$ and $d - e$ is particularly difficult. Considerable difficulty also occurs when clipping a polygon results in several disjoint smaller polygons as shown in the Fig. 5.21. For example, the lines $a - b$, $c - d$, $d - e$ and $g - f$ are frequently included in the clipped polygon description which is not desired.

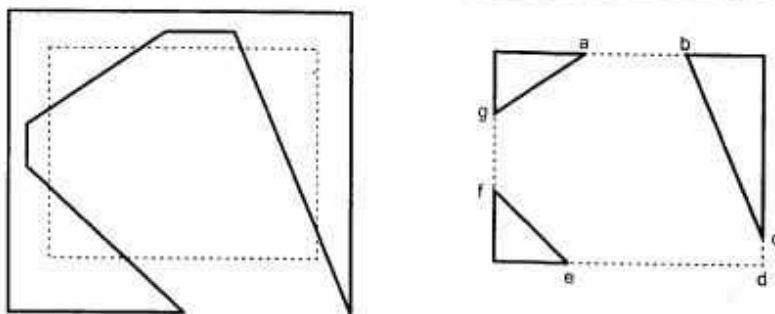


Fig. 5.21 Disjoint polygons in polygon clipping

5.5 Sutherland - Hodgeman Polygon Clipping

A polygon can be clipped by processing its boundary as a whole against each window edge. This is achieved by processing all polygon vertices against each clip rectangle boundary in turn. Beginning with the original set of polygon vertices, we could first clip the polygon against the left rectangle boundary to produce a new sequence of vertices. The new

set of vertices could then be successively passed to a right boundary clipper, a top boundary clipper and a bottom boundary clipper, as shown in Fig. 5.22. At each step a new set of polygon vertices is generated and passed to the next window boundary clipper. This is the fundamental idea used in the Sutherland - Hodgeman algorithm.

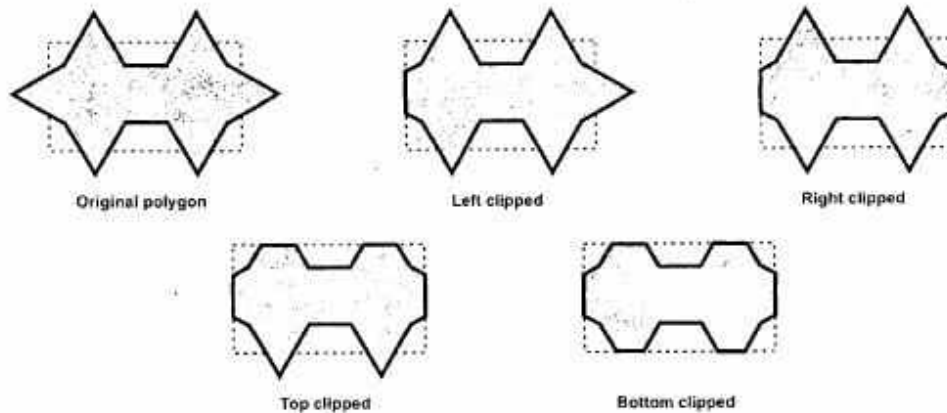


Fig. 5.22 Clipping a polygon against successive window boundaries

The output of the algorithm is a list of polygon vertices all of which are on the visible side of a clipping plane. Such each edge of the polygon is individually compared with the clipping plane. This is achieved by processing two vertices of each edge of the polygon around the clipping boundary or plane. This results in four possible relationships between the edge and the clipping boundary or plane. (See Fig. 5.23).

1. If the first vertex of the edge is outside the window boundary and the second vertex of the edge is inside then the intersection point of the polygon edge with the window boundary and the second vertex are added to the output vertex list (See Fig. 5.23 a).
2. If both vertices of the edge are inside the window boundary, only the second vertex is added to the output vertex list. (See Fig. 5.23 b).
3. If the first vertex of the edge is inside the window boundary and the second vertex of the edge is outside, only the edge intersection with the window boundary is added to the output vertex list. (See Fig. 5.23 c).
4. If both vertices of the edge are outside the window boundary, nothing is added to the output list. (See Fig. 5.23 d).

Once all vertices are processed for one clip window boundary, the output list of vertices is clipped against the next window boundary.

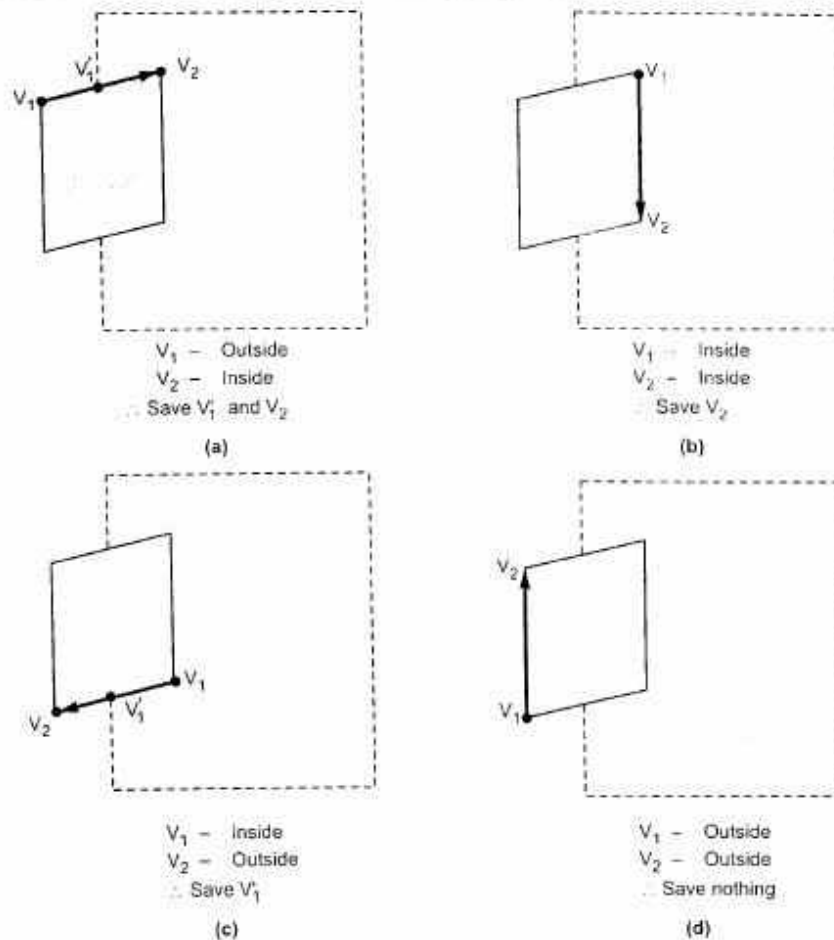


Fig. 5.23 Processing of edges of the polygon against the left window boundary

Going through above four cases we can realize that there are two key processes in this algorithm.

1. Determining the visibility of a point or vertex (Inside - Outside test) and
2. Determining the intersection of the polygon edge and the clipping plane.

One way of determining the visibility of a point or vertex is described here. Consider that two points A and B define the window boundary and point under consideration is V, then these three points define a plane. Two vectors which lie in that plane are AB and AV. If this plane is considered in the xy plane, then the vector cross product $AV \times AB$ has only a z component given by $(x_V - x_A)(y_B - y_A) - (y_V - y_A)(x_B - x_A)$. The sign of the z component decides the position of point V with respect to window boundary.

- If z is :
- Positive - Point is on the **right side** of the window boundary
 - Zero - Point is **on** the window boundary
 - Negative - Point is on the **left side** of the window boundary

Ex. 5.6 : Consider the clipping boundary as shown in the Fig. 5.24 and determine the positions of points V_1 and V_2 .

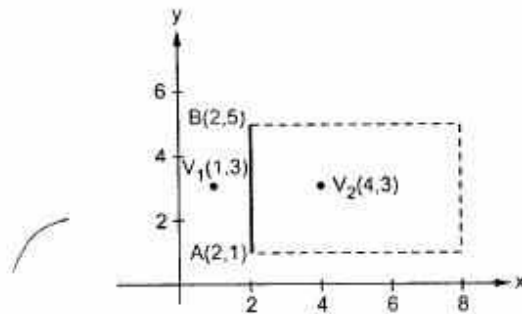


Fig. 5.24

Sol. : Using the cross product for V_1 we get,

$$\begin{aligned} & (x_V - x_A)(y_B - y_A) - (y_V - y_A)(x_B - x_A) \\ &= (1 - 2)(5 - 1) - (3 - 1)(2 - 2) \\ &= (-1)(4) - 0 \\ &= -4 \end{aligned}$$

The result of the cross product for V_1 is negative hence V_1 is on the left side of the window boundary.

$$\begin{aligned} & \text{Using the cross product for } V_2 \text{ we get, } (4 - 2)(5 - 1) - (3 - 1)(2 - 2) \\ &= (2)(4) - 0 \\ &= 8 \end{aligned}$$

The result of the cross product for V_2 is positive hence V_2 is on the right side of the window boundary.

The second key process in Sutherland - Hodgeman polygon clipping algorithm is to determine the intersection of the polygon edge and the clipping plane. Any of the line intersection (clipping) techniques discussed in the previous sections such as Cyrus-Beck or mid point subdivision can be used for this purpose.

Sutherland-Hodgeman Polygon Clipping Algorithm

1. Read coordinates of all vertices of the polygon.
2. Read coordinates of the clipping window
3. Consider the left edge of the window
4. Compare the vertices of each edge of the polygon, individually with the clipping plane

5. Save the resulting intersections and vertices in the new list of vertices according to four possible relationships between the edge and the clipping boundary discussed earlier.
6. Repeat the steps 4 and 5 for remaining edges of the clipping window. Each time the resultant list of vertices is successively passed to process the next edge of the clipping window.
7. Stop.

Ex. 5.7: For a polygon and clipping window shown in Fig. 5.25 give the list of vertices after each boundary clipping.

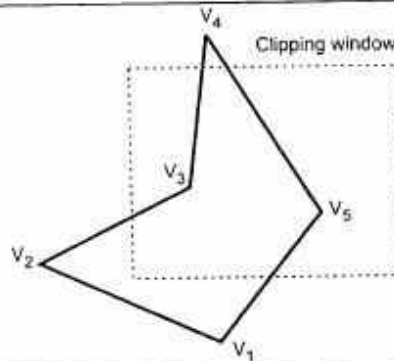


Fig. 5.25

Sol.: Original polygon vertices are V_1, V_2, V_3, V_4, V_5 . After clipping each boundary the new vertices are given in Fig. 5.26

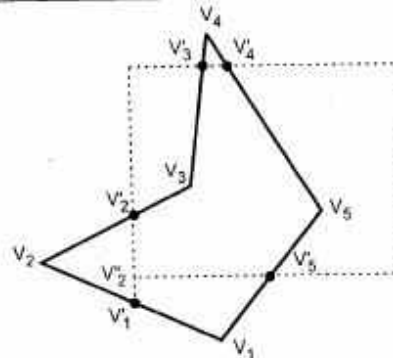


Fig. 5.26

- After left clipping : $V_1, V_1', V_2', V_3, V_4, V_5$
 After right clipping : $V_1, V_1', V_2', V_3, V_4, V_5$
 After top clipping : $V_1, V_1', V_2', V_3, V_3', V_4', V_5$
 After bottom clipping : $V_2', V_2', V_3, V_3', V_4', V_5, V_5'$

The Sutherland-Hodgeman polygon clipping algorithm clips convex polygons correctly, but in case of concave polygons, clipped polygon may be displayed with extraneous lines, as shown in Fig. 5.27.



Fig. 5.27 Clipping the can cave polygon in (a) with the Sutherland-Hodgeman algorithm produces the two connected areas in (b)

The problem of extraneous lines for concave polygons in Sutherland-Hodgeman polygon clipping algorithm can be solved by separating concave polygon into two or more convex polygons and processing each convex polygon separately.

5.6 Weiler-Atherton Algorithm

The clipping algorithms previously discussed require a convex polygon. In context of many applications, e.g., hidden surface removal, the ability to clip to concave polygon is required. A powerful but somewhat more complex clipping algorithm developed by Weiler and Atherton meets this requirement. This algorithm defines the polygon to be clipped as a **subject polygon** and the clipping region is the **clip polygon**.

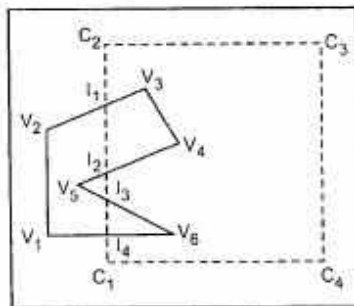


Fig. 5.28

The algorithm describes both the subject and the clip polygon by a circular list of vertices. The boundaries of the subject polygon and the clip polygon may or may not intersect. If they intersect, then the intersections occur in pairs. One of the intersections occurs when a subject polygon edge enters the inside of the clip polygon and one when it leaves. As shown in the Fig. 5.28, there are four intersection vertices I_1, I_2, I_3 and I_4 . In these intersections I_1 and I_3 are entering intersections, and I_2 and I_4 are leaving intersections. The clip polygon vertices are marked as C_1, C_2, C_3 and C_4 .

In this algorithm two separate vertices lists are made one for clip polygon and one for subject polygon including intersection points. The Table 5.3 shows these two lists for polygons shown in Fig. 5.28.

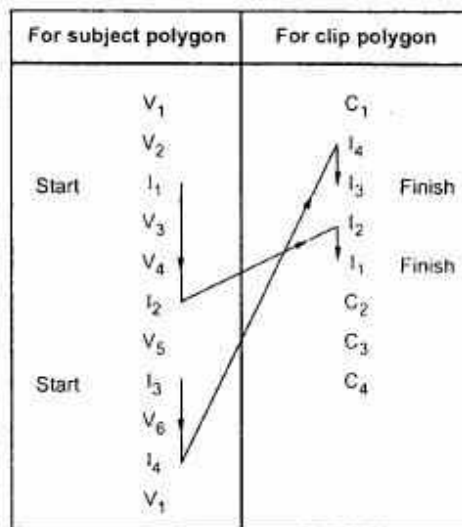


Table 5.3 List of polygon vertices

The algorithm starts at an entering intersection (I_1) and follows the subject polygon vertex list in the downward direction (i.e. I_1, V_3, V_4, I_2). At the occurrence of leaving intersection the algorithm follows the clip polygon vertex list from the leaving intersection vertex in the downward direction (i.e. I_2, I_1). At the occurrence of the entering intersection the algorithm follows the subject polygon vertex list from the entering intersection vertex. This process is repeated until we get the starting vertex. This process we have to repeat for all remaining entering intersections which are not included in the previous traversing of vertex list. In our example, entering vertex I_3 was not included in the first traversing of vertex list. Therefore, we have to go for another vertex traversal from vertex I_3 .

The above two vertex traversals gives two clipped inside polygons. There are :

I_1, V_3, V_4, I_2, I_1 and I_3, V_6, I_4, I_3

5.7 Generalized Clipping

We have seen that in Sutherland - Hodgeman polygon clipping algorithm we need separate clipping routines, one for each boundary of the clipping window. But these routines are almost identical. They differ only in their test for determining whether a point is inside or outside the boundary. It is possible to generalize these routines so that they will be exactly identical and information about the boundary is passed to the routines through their parameters. Using recursive technique the generalized routine can be 'called' for each boundary of the clipping window with a different boundary specified by its parameters. This form of algorithm allows us to have any number of boundaries to the clipping window, thus the generalized algorithm with recursive technique can be used to clip a polygon along an arbitrary convex clipping window.

5.8 Interior and Exterior Clipping

So far we have discussed only algorithms for clipping point, line and polygon to the interior of a clipping region by eliminating every thing outside the clipping region. However, it is also possible to clip a point, line or polygon to the exterior of a clipping region, i.e., the point, portion of line and polygon which lie outside the clipping region. This is referred to as **exterior clipping**.

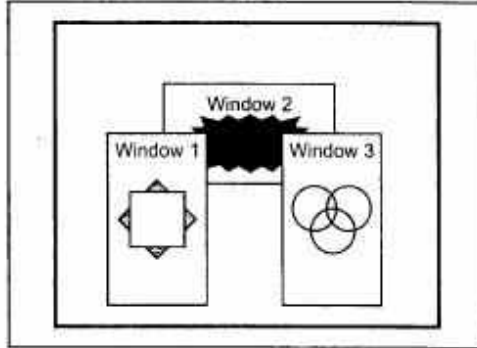


Fig. 5.29 Clipping in multiwindow environment

Exterior clipping is important in a multiwindow display environment, as shown in Fig. 5.29. The Fig. 5.29 shows the overlapping windows with window 1 and window 3 having priority over window 2. The objects within the window are clipped to the interior of that window. When other higher-priority windows such as window 1 and/or window 3 overlap these objects, the objects are also clipped to the exterior of the overlapping windows.

Solved Examples

Ex. 5.8 : Use the Cohen-Sutherland outcode algorithm to clip two lines $P_1 (40, 15) - P_2 (75, 45)$ and $P_3 (70, 20) - P_4 (100, 10)$ against a window $A (50, 10), B (80, 10), C (80, 40), D(50,40)$.

Sol. : Line 1 : $P_1 (40, 15) P_2 (75, 45) W_{x1} = 50 \quad W_{y1} = 40$
 $W_{x2} = 80 \quad W_{y2} = 10$

Point	Encode	ANDing	
P_1	0001	0000	Partially visible
P_2	0000		

$$y_1 = m(x_1 - x) + y = \frac{6}{7}(50 - 40) + 15 \quad m = \frac{45 - 15}{75 - 40} = \frac{6}{7}$$

$$= 23.57$$

$$x_1 = \frac{1}{m}(y_T - y) + x \Rightarrow \frac{7}{6}(40 - 15) + 40 = 69.16$$

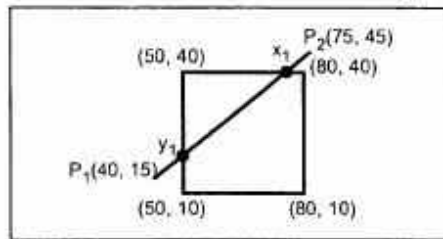


Fig. 5.30

$$y_2 = m(x_2 - x) + y$$

$$= \frac{6}{7}(80 - 40) + 15$$

$$= 49.28$$

$$x_2 = \frac{1}{m}(y_B - y) + x$$

$$= \frac{7}{6}(10 - 15) + 40$$

$$= 34.16$$

Line 2 : $P_3(70, 20)$ $P_4(100, 10)$

Point	End code	ANDing	Position
P_3	0000	0000	Partially visible
P_4	0010		

$$\text{Slope } m' = \frac{10 - 20}{100 - 70} = \frac{-10}{30} = \frac{-1}{3}$$

$$y'_1 = m(x_1 - x) + y = \frac{-1}{3}(50 - 70) + 20$$

$$= 26.66$$

$$x'_1 = \frac{1}{m}(y_1 - y) + x = -3(40 - 20) + 70$$

$$= 10$$

$$y'_2 = m(x_2 - x) + y = \frac{-1}{3}(80 - 70) + 20$$

$$= 16.66$$

$$x'_2 = \frac{1}{m}(y_2 - y) + x = -3(10 - 20) + 70$$

$$= 100$$

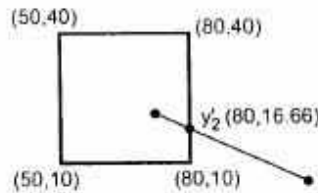


Fig. 5.31

Ex. 5.9: Find the normalization transformation window to viewport, with window, lower left corner at (1, 1) and upper right corner at (3, 5) onto a viewport, for entire normalized device screen.

Sol.: $x_{w \min} = 1$ $y_{w \min} = 1$
 $x_{w \max} = 3$ $y_{w \max} = 5$

Entire normalized screen

$$x_{v \min} = 0$$

$$x_{v \max} = 1$$

$$y_{v \min} = 0$$

$$y_{v \max} = 1$$

$$S_x = \frac{x_{v \max} - x_{v \min}}{x_{w \max} - x_{w \min}}$$

$$= \frac{1 - 0}{3 - 1} = \frac{1}{2}$$