

## 2.1 COLOR AND GRAYSCALE LEVELS

Various color and intensity-level options can be made available to a user, depending on the capabilities and design objectives of a particular system. General purpose raster-scan systems, for example, usually provide a wide range of colors, while random-scan monitors typically offer only a few color choices, if any. Color options are numerically coded with values ranging from 0 through the positive integers. For CRT monitors, these color codes are then converted to intensity level settings for the electron beams. With color plotters, the codes could control ink-jet deposits or pen selections.

In a color raster system, the number of color choices available depends on the amount of storage provided per pixel in the frame buffer. Also, color-information can be stored in the frame buffer in two ways: We can store color codes directly in the frame buffer, or we can put the color codes in a separate table and use pixel values as an index into this table. With the direct storage scheme, whenever a particular color code is specified in an application program, the corresponding binary value is placed in the frame buffer for each-component pixel in the output primitives to be displayed in that color. A minimum number of colors can be provided in the scheme with 3 bits of storage per pixel, as shown in Table. Each of the three bit positions is used to control the intensity level (either on or off) of the corresponding electron gun in an RGB monitor. The leftmost bit controls the red gun, the middle bit controls the green gun, and the rightmost bit controls the blue gun. Adding more bits per pixel to the frame buffer increases the number of color choices. With 6 bits per pixel, 2 bits can be used for each gun. This allows four different intensity settings for each of the three color guns, and a total of 64 color values are available for each screen pixel. With a resolution of 1024 by 1024, a full-color (24bit per pixel) RGB system needs 3 megabytes of storage for the frame buffer. Color tables are an alternate means for providing extended color capabilities to a user without requiring large frame buffers. Lower-cost personal computer systems, in particular, often use color tables to reduce frame-buffer storage requirements.

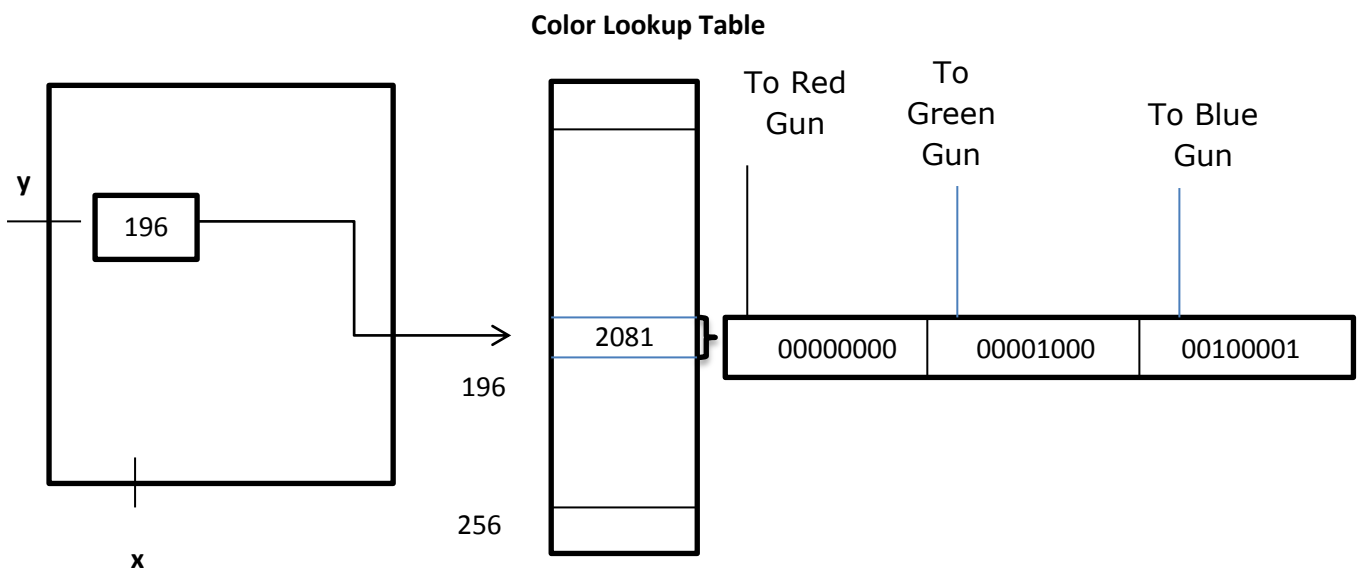
### 2.1.1 COLOR TABLES

Figure 2.1 illustrates a possible scheme for storing color values in a color lookup table (or video lookup table), where frame-buffer values are now used as indices into the color table. In this example, each pixel can reference any one of the 256 table positions, and each entry in the table uses 24 bits to specify an RGB color. For the color code 2081, a combination green-blue color is displayed for pixel location (x, y). Systems employing this particular lookup table would allow a user to select any 256 colors for simultaneous display from a palette of nearly 17 million colors. Compared to a full color system, this scheme reduces the number of simultaneous colors that can be displayed,

but it also reduces the frame buffer storage requirements to 1 megabyte. Some graphics systems provide 9 bits per pixel in the frame buffer, permitting a user to select 512 colors that could be used in each display.

Color Code	RED	GREEN	BLUE	Displayed Color
0	0	0	0	Black
1	0	0	1	Blue
2	0	1	0	Green
3	0	1	1	Cyan
4	1	0	0	Red
5	1	0	1	Magenta
6	1	1	0	Yellow
7	1	1	1	White

(Table 2.1, The eight color codes for a three-bit per pixel frame buffer)



(Fig: 2:1, Color Lookup Table)

There are several advantages in storing color codes in a lookup table. Use of a color table can provide a "reasonable" number of simultaneous colors without requiring large frame buffers. For most applications, 256 or 512 different colors are sufficient for a single picture. Also, table entries can be changed at any time, allowing a user to be able to experiment easily with different color combinations in a design, scene, or graph without changing the attribute settings for the graphics data structure. Similarly, visualization applications can store values for some physical quantity, such as energy, in

the frame buffer and use a lookup table to try out various color encodings without changing the pixel values. And in visualization and image-processing applications, color tables are a convenient means of setting color thresholds so that all pixel values above or below a specified threshold can be set to the same color. For these reasons, some systems provide both capabilities for color-code storage, so that a user can elect either to use color tables or to store color codes directly in the frame buffer.

### 2.1.2 GRAYSCALE

With monitors that have no color capability, color functions can be used in an application program to set the shades of gray, or grayscale, for displayed primitives. Numeric values over the range from 0 to 1 can be used to specify grayscale levels, which are then converted to appropriate binary codes for storage in the raster. This allows the intensity settings to be easily adapted to systems with differing grayscale capabilities.

Table lists the specifications for intensity codes for a four-level grayscale system. In this example, any intensity input value near 0.33 would be stored as the binary value 01 in the frame buffer, and pixels with this value would be displayed as dark gray. If additional bits per pixel are available in the frame buffer, the value of 0.33 would be mapped to the nearest level. With 3 bits per pixel, we can accommodate 8 gray levels; while 8 bits per pixel would give us 256 shades of gray. An alternative scheme for storing the intensity information is to convert each intensity code directly to the voltage value that produces this grayscale level on the output device in use.

Intensity Codes	Stored intensity the frame the buffer	values in (Binary code)	Displayed Gray Scale
0.0	0	00	Black
0.33	1	01	Dark Gray
0.67	2	10	Light Gray
1.0	3	11	White

(Table 2.2, Intensity codes for a four-level grayscale system)

## 2.2 2D TRANSFORMATIONS

A graphic system should allow the programmer to define pictures that include a variety of transformation. That means, he should be able to magnify a picture so that details appear more clearly or reduce it so that more of the picture is visible.

Translation, rotation, and scaling are the basic geometric transformations. Other transformations are reflection and shear. Two transformations can be combined or

concatenated to yield a single transformation with the same effect as the sequential application of the original two.

### 2.2.1 TRANSLATION

A translation is applied to an object by repositioning it from one coordinate location to another. We can translate points in the (x,y) plane to new positions by adding translation amounts to the coordinates of the points. For each point P(x, y) to be moved by  $t_x$  units parallel to the x-axis and  $t_y$  units parallel to the y axis to the new point  $p'(x', y')$ . We can write,

$$x' = x + t_x$$

$$y' = y + t_y$$

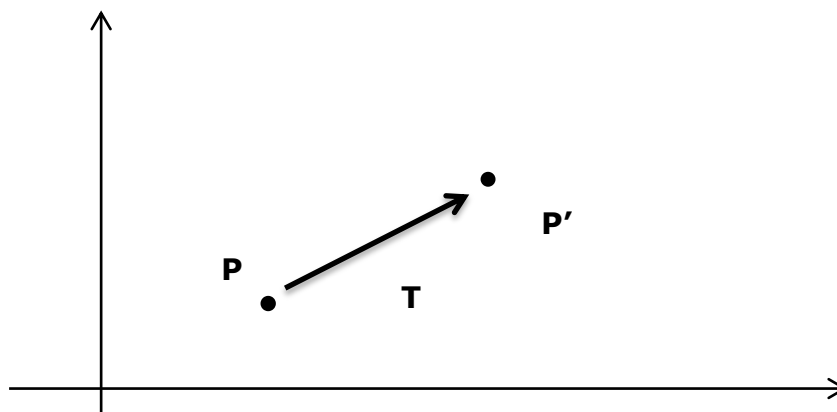
Suppose  $P = \begin{pmatrix} x \\ y \end{pmatrix}$

$$P' = \begin{pmatrix} x' \\ y' \end{pmatrix}$$

$$T = \begin{pmatrix} t_x \\ t_y \end{pmatrix}$$

$$P' = P + T$$

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



**(Fig: 2:2 Translating a point from position P to position p' with translation vector T)**

We can translate an object by applying the above eqn. to every point of the object.

For eg: consider a line defined by two points  $x(2, 1)$  ,  $y(4, 4)$

Suppose we want to translate these line 2 units to the right and 3 units up.

Then  $t_x = 2$  ,  $t_y = 3$

$$x' = x + t_x$$

$$y' = y + t_y$$

$$= x + 2$$

$$y - 3$$

Consider a triangle defined by its 3 vertices.

Suppose we want to translate it 2 units to the right and 4 units up. Then  $t_x = 2$ ,  $t_y = 4$ .

$$x' = x + 2$$

$$y' = y + 4$$

### 2.2.2 SCALING

A scaling transformation changes the size of an object. This operation can be carried out by multiplying the coordinate values  $(x, y)$  for each vertex by scaling factors  $S_x$  and  $S_y$  to produce the transformed coordinates  $(x', y')$ .

$$x' = x \cdot S_x$$

$$y' = y \cdot S_y$$

Scaling factor  $S_x$  scales the objects in the  $x$  direction, while  $S_y$  scales in the  $y$  direction. The transformation eqns can be written in the matrix form.

$$\begin{matrix} x' \\ y' \end{matrix} = \begin{matrix} S_x & 0 \\ 0 & S_y \end{matrix} \begin{matrix} x \\ y \end{matrix} \longrightarrow \mathbf{A}$$

or

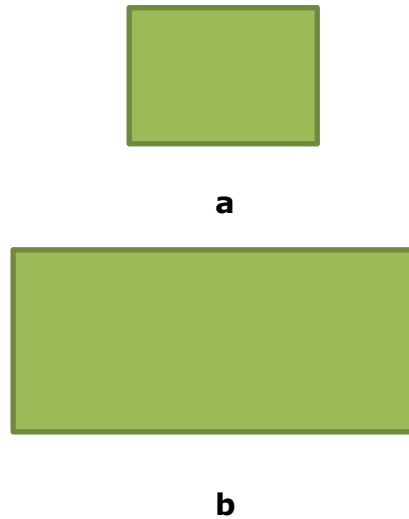
$$P' = S \cdot P$$

Suppose we want to scale the triangle below 3 units in the  $x$  direction and 2 units in the  $y$  direction. Then  $S_x = 3$ ,  $S_y = 2$ .

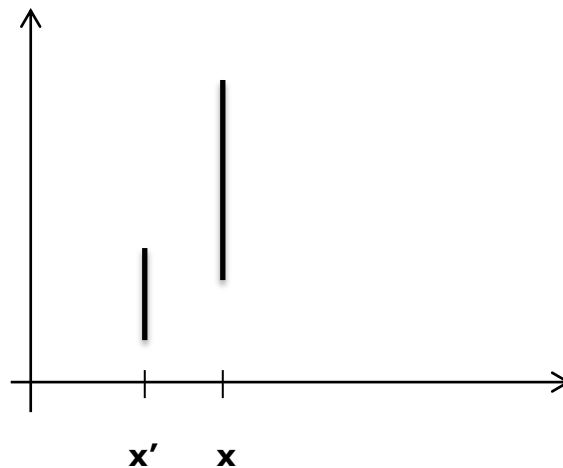
Scaling (note that scaling is non-uniform also, the triangle changes its position).

Any positive numeric values can be assigned to the scaling factors  $s_x$  and  $s_y$ . Values less than 1 reduce the size of objects; values greater than 1 produce an enlargement. Specifying a value of 1 for both  $s_x$  and  $s_y$  leaves the size of objects unchanged. When  $s_x$  and  $s_y$  are assigned the same value, a **uniform** scaling is produced that maintains relative object proportions. Unequal values for  $s_x$  and  $s_y$  result in a differential scaling that is often used in design applications, where pictures are constructed from a few basic shapes that can be adjusted by scaling and positioning transformations (**Fig. 2-3**).

Objects transformed with **Eq.A** are both scaled and repositioned. Scaling factors with values less than 1 move objects closer to the coordinate origin, while values greater than 1 move coordinate positions farther from the origin. **Figure 5-7** illustrates scaling a line by assigning the value 0.5 to both  $s_x$  and  $s_y$  in **Eq.5-11**. Both the line length and the distance from the origin are reduced by a factor of 1/2.



(Fig. 2.3 Turning a square(a) in to a rectangle(b)with scaling factors  $s_x=2$  and  $s_y=1$ )



(Fig. 2.4 A line scaled using  $s_x=s_y=0.5$  is reduced in size and moved closer to the coordinate origin)

### 2.2.3 ROTATION

A 2d 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  $\theta$  and the position  $(x_r, y_r)$  of the rotation point about which the object is to be rotated.

If we rotate an object through an angle  $\theta$  about the origin, then it can be defined as

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

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

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

$$y' \quad \sin \theta \quad \cos \theta \quad y$$

$$P' = R.P$$

Figure 2.5 shows the rotation of a triangle by  $45^\circ$ . Here rotation is about the origin.

The eqns

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

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

can be easily derived from the following figure. In the fig a point  $P(x, y)$  is transformed to point  $p'(x', y')$  by rotation  $\theta$ .

$$\cos @ = \frac{\text{adjacent side}}{\text{hyp}} = \frac{x}{r}$$

$$x = r \cdot \cos @$$

$$\sin @ = y/r$$

$$y = r \cdot \sin @$$

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

$$x' = r \cdot \cos (@ + \theta)$$

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

$$y' = r \cdot \sin (@ + \theta)$$

$$x' = r \cdot \cos (@ + \theta)$$

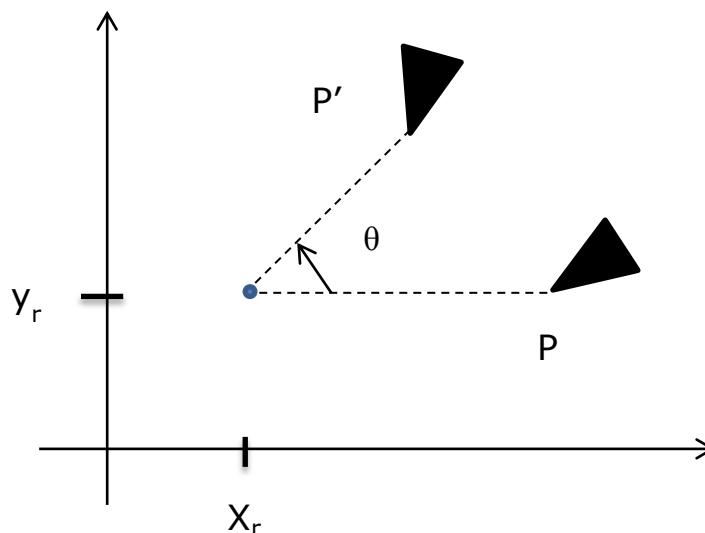
$$x' = r \cos @ \cdot \cos\theta - r \cdot \sin@ \cdot \sin\theta$$

$$y' = r \cdot \sin (@ + \theta)$$

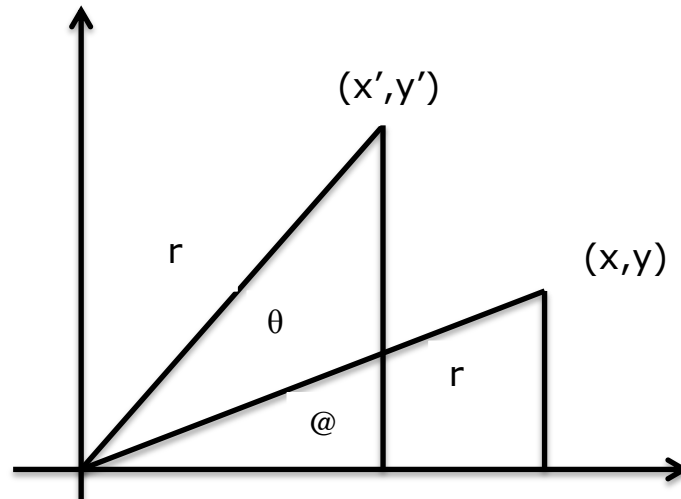
$$y' = r \cdot \sin @ \cdot \cos\theta + r \cdot \sin\theta \cdot \cos @$$

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

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



(Figure 2.5 Rotation of an object through angle  $\theta$  about the pivot point  $(x_r, y_r)$ )



(Figure 2.6 Rotation of a point from position (x,y) to position (x',y') through an angle  $\theta$  relative to the coordinate origin. The regular angular displacement of the point from the x axis is @)

## 2.2.4 MATRIX REPRESENTATION OF TRANSFORMATIONS.

For translation,

$$P' = P + T$$

$$x' = x + tx$$

$$y' = y + ty$$

This can also be written as ,

$$x' = 1 \ 0 \ tx \ . \ x$$

$$y' = 0 \ 1 \ ty \ y$$

$$1 \ 0 \ 0 \ 1 \ 1$$

For rotation,

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

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

This can be written as,

$$x' = \cos@ \ -\sin @ \ 0 \ . \ x$$

$$y' = \sin @ \ \cos @ \ 0 \ y$$

$$1 \ 0 \ 0 \ 1 \ 1$$

For scaling,

$$P' = S.P$$

This can be written as,

$$x' = Sx \ 0 \ 0 \ . \ x$$

$$y' = 0 \ Sy \ 0 \ y$$

$$1 \ 0 \ 0 \ 1 \ 1$$



Matrix representations are standard methods for implementing transformations in graphics systems.

### 2.2.5 SHEAR

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

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

The transformation matrix for an x direction shear is

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

Any real number can be assigned to shear parameter  $sh_x$ .

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

For e.g. If this cube is sheared in the x direction we will get,



### 2.2.6 REFLECTION

A reflection is a transformation that produces the mirror image of an object. The mirror image for a 2d reflection is generated relative to an axis of reflection by rotating the object  $180^\circ$  about the reflection axis.

Reflection about the line  $y=0$  the x axis is accomplished with the transformation matrix,

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

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

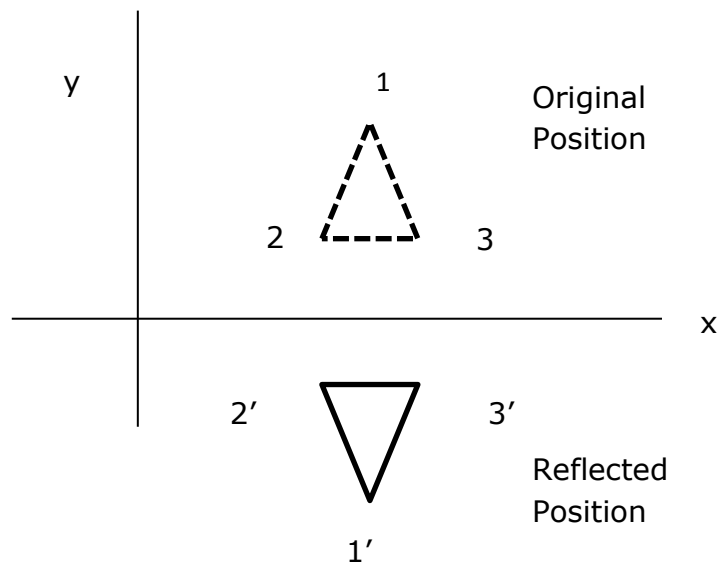
Reflection of an object about the x axis while keeping y coordinate same. The matrix for this transformation about y axis is,

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

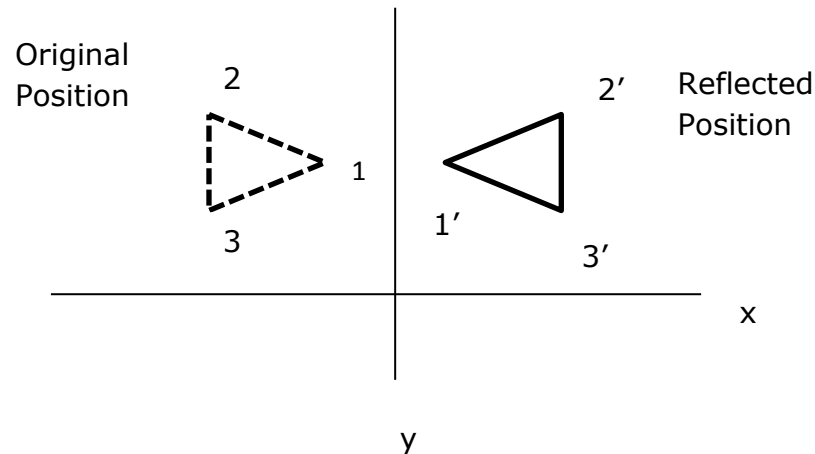
$$\begin{vmatrix} x' \\ y' \\ 1 \end{vmatrix} = \begin{vmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} x \\ y \\ 1 \end{vmatrix}$$

We flip both the x and y coordinates of a point by reflecting relative to an axis that is perpendicular to the xy plane and that passes through the coordinate origin .This transformation ,referred to as a reflection relative to the coordinate origin ,has the matrix representation:

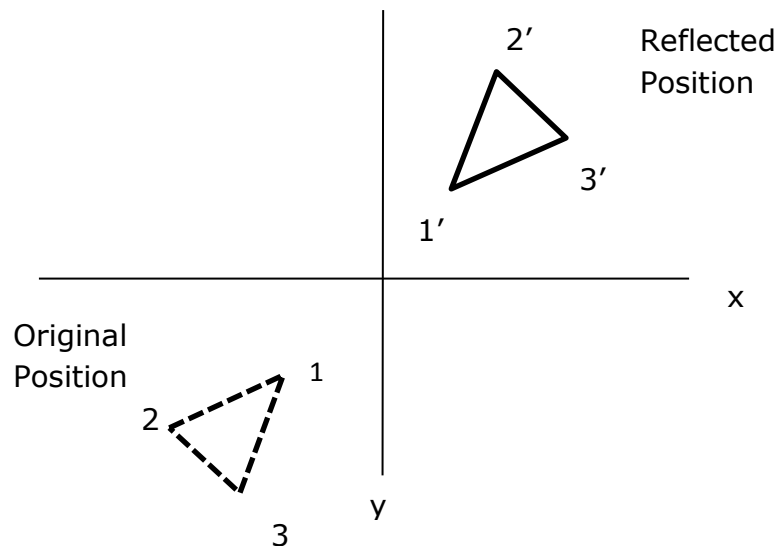
$$T_m = \begin{vmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{vmatrix}$$



**(Figure 2.7 Reflection of an object about the x axis)**



(Figure 2.8 Reflection of an object about the y axis)



(Figure 2.9 Reflection of an object relative to an axis perpendicular to the xy plane and passing through the coordinate origin)

### 2.2.7 COMPOSITE TRANSFORMATIONS

We have learnt matrix representations of transformation. We can set up a matrix for any sequence of transformations as a composite transform matrix by calculating the matrix product of the individual transform.

For e.g. suppose we want to perform rotation of an object about an arbitrary point. This can be performed by applying a sequence of three simple transforms a translation, followed by a rotation followed by another translation. For e.g. suppose we are asked to rotate the triangle through  $90^0$ .

This can be done by first translate the object to the origin. Rotate it by  $90^0$  and again translate it by  $t_x = 2, t_y = 0$ .

Translate the object to  $tx=2, ty= 0$

Here we have done one translation, one rotation , one translation again.

Complex transformation can be described as concatenation of simple ones. Suppose we wish to derive a transformation which will rotate a point through a clockwise angle  $@$  about the point  $(R_x, R_y)$ . The rotation transformation be applied to rotate points only about the origin. So we must translate points so that  $(R_x, R_y)$  becomes the origin.

$$\begin{array}{rcccccc} x' & = & x & \cdot & 1 & 0 & -R_x \\ y' & & y & & 0 & 1 & -R_y \\ 1 & & 1 & & 0 & 0 & 1 \end{array}$$

then rotation can be applied

$$\begin{array}{rcccccc} x'' = x' & \text{Cos@} & -\text{Sin@} & & 0 \\ y'' = y' & \text{Sin@} & \text{Cos@} & & 0 \\ 1 & 1 & 0 & & 0 & 1 \end{array}$$

Finally, we translate point so that the origin is returned to  $(R_x, R_y)$

$$\begin{array}{rcccccc} x''' = & x'' & 1 & 0 & R_x \\ y''' & y'' & 0 & 1 & R_y \\ 1 & 1 & 0 & 0 & 1 \end{array}$$