MODULE-2

2D Geometric Transformations

Operations that are applied to the geometric description of an object to change its position, orientation, or size are called **geometric transformations.** Sometimes geometric transformation operations are also referred to as **modeling transformations**.

Basic Two-Dimensional Geometric Transformations

The geometric-transformation functions that are available in all graphics packages are those for **translation, rotation, and scaling**. Other useful transformation routines that are sometimes included in a package are **reflection** and **shearing** operations.

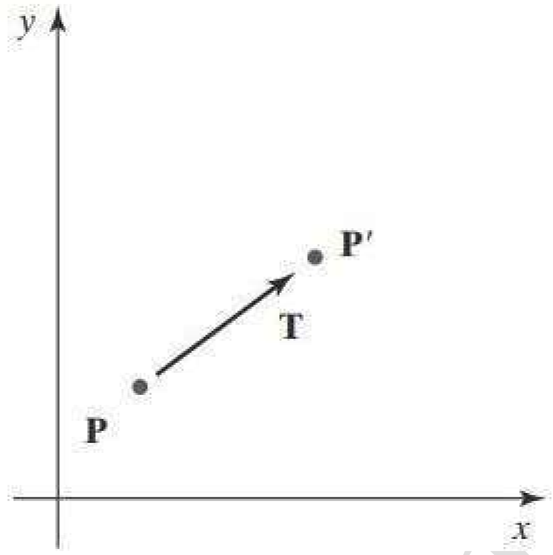
Two-Dimensional Translation

Translation on single coordinate point is performed by adding offsets to its coordinates to generate a new coordinate position. The original point position is moved along a straight line path to its new location.

To translate a two-dimensional position, we add **translation distances** t_x and t_y to the original coordinates (x,y) to obtain the new coordinate position (x′,y′) as shown

$$x' = x + t_x, \qquad y' = y + t_y \tag{1}$$

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



Translating a point from position P to position P using a translation vector T.

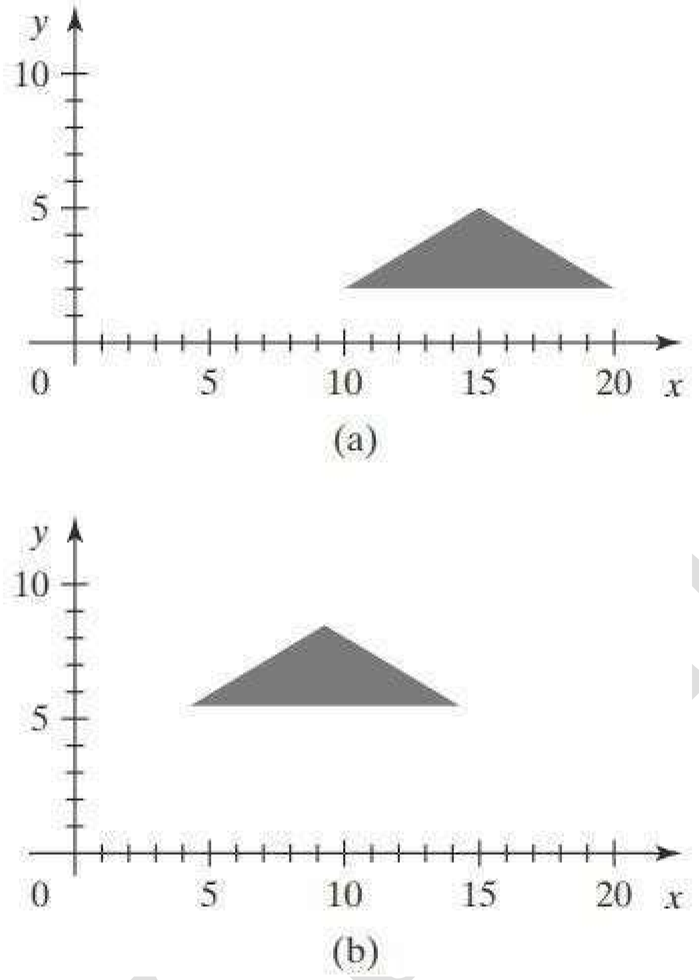
We can express Equations 1 as a single matrix equation by using the following column vectors to represent coordinate positions and the translation vector:

$$\mathbf{P} = \begin{bmatrix} x \\ y \end{bmatrix}, \qquad \mathbf{P}' = \begin{bmatrix} x' \\ y' \end{bmatrix}, \qquad \mathbf{T} = \begin{bmatrix} t_x \\ t_y \end{bmatrix} \qquad (2)$$

The two-dimensional translation equations can be written in matrix forms as:

$$\mathbf{P}' = \mathbf{P} + \mathbf{T} \qquad (3)$$

Translation is a rigid-body transformation that moves objects without deformation. That is, every point on the object is translated by the same amount. Below figure illustrates the application of a specified translation vector to move an object from one position to another.



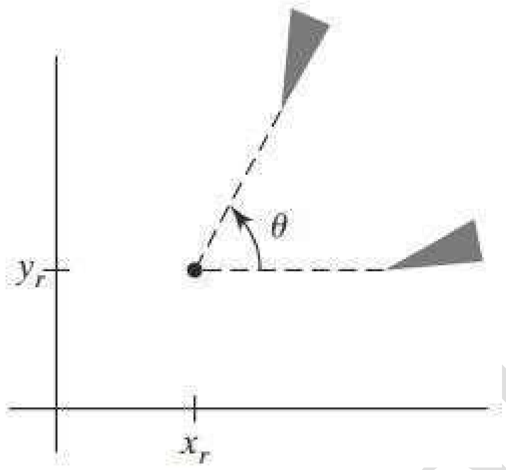
Moving a polygon from position (a) to position (b) with thetranslation vector (-5.50, 3.75)

Two-Dimensional Rotation

Rotation transformation of an object is generated by specifying a rotation axis and a rotation angle. All points of the object are then transformed to new positions by rotating the points through the specified angle about the rotation axis.

A two-dimensional rotation of an object is obtained by repositioning the object along a circular path in the xy plane. Parameters for the two-dimensional rotation are the rotation angle θ and a position(x_r,y_r), called the rotation point (or pivot point), about which the object is to be rotated.

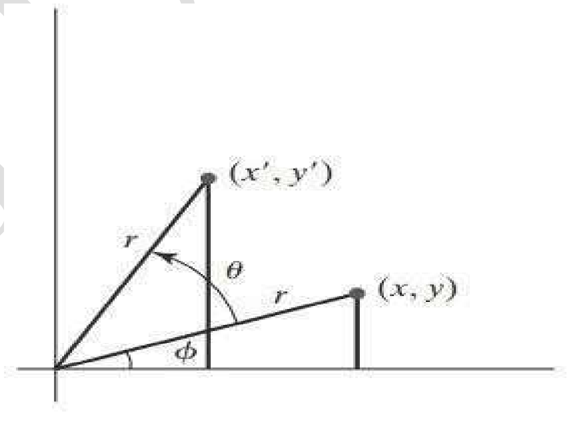
A positive value for the angle θ defines a counterclockwise rotation about the pivot point. A negative value for the angle θ rotates objects in the clockwise direction.



Rotation of an object through angle θ about the pivot point (x_r ,y_r)

1) **Determination of the transformation equations for rotation of a point position P when the pivot point is at the coordinate origin.**

The angular and coordinate relationships of the original and transformed point positions are shown



Rotation of a point from position (x,y) to position (x′, y′) through an angle θ relative to the coordinate origin. The original angular displacement of the point from the xaxis is ϕ

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 $\boldsymbol{\theta}$ is the rotation angle.

Using standard trigonometric identities, transformed coordinates can be expressed in terms of angles of θ and ϕ

$$x' = r\cos(\phi+\theta) = r\cos\phi\cos\theta - r\sin\phi\sin\theta \qquad (4)$$

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

The original coordinates of the point in the polar coordinates are

$$x = r\cos\phi \quad y = r\sin\phi \qquad (5)$$

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

$$x' = x\cos\theta - y\sin\theta \qquad (6)$$

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

The rotation equation in matrix form is written as

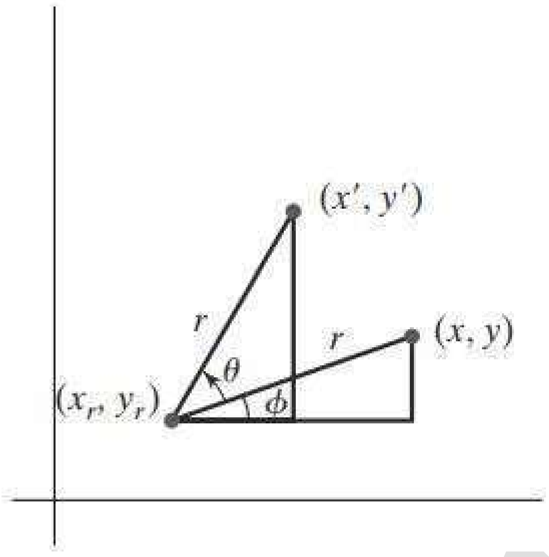
$$P' = RP \qquad (7)$$

where the rotation matrix is

$$\mathbf{R} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \qquad (8)$$

2) **<u>Determination of the transformation equations for rotation of a point position P when the pivot point is at (x_r,y_r).</u>**

Rotation of a point about an arbitrary pivot position is illustrated



Rotating a point from position(x,y) to position (x′,y′) through an angle θ about rotation point (x_r,y_r)

Using the trigonometric relationships indicated by the two right triangles in this figure, we can generalize Equations 6 to obtain the transformation equations for rotation of a point about any specified rotation position (x_r, y_r):

$$x' = x_r + (x - x_r)\cos\theta - (y - y_r)\sin\theta \qquad \textbf{(9)}$$

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

Two-Dimensional Scaling

To alter the size of an object, **scaling** transformation is used. A two dimensional scaling operation is performed by multiplying object positions (x,y) by scaling factors s_x and s_y to produce the transformed coordinates(x',y').

$$x' = x \cdot s_x, \qquad y' = y \cdot s_y \qquad (10)$$

Scaling factor s_x scales an object in the x direction, while s_y scales in the y direction. The basic two-dimensional scaling equations 10 can also be written in the following matrix form:

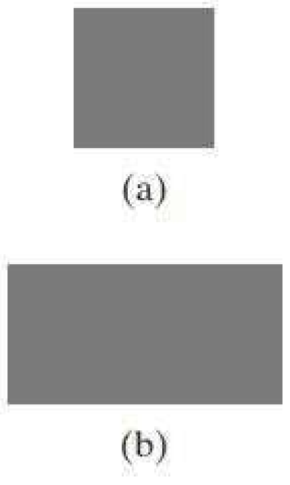
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} \tag{11}$$

or

P'=S.P (12)

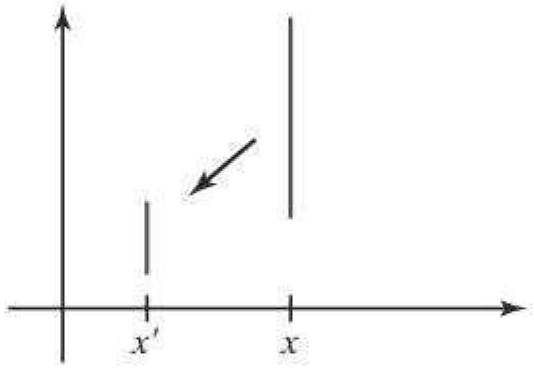
where **S** is the 2 × 2 scaling matrix in Equation 11

Any positive 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 enlargements. 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, which maintains relative object proportions. Unequal values for s_x and s_y result in a **differential scaling**.



Turning a square (a) into a rectangle (b) with scaling factors s_x = 2 ands_y = 1.

Figure below illustrates scaling of a line by assigning the value 0.5 to both s_x and s_y.Both the line length and the distance from the origin are reduced by a factor of ½.



A line scaled with Equation 12 using $s_x = s_y = 0.5$ is reduced in size and moved closer to the coordinate origin

The location of a scaled object is controlled by choosing a position, called the **fixed point,** that is to remain unchanged after the scaling transformation. Coordinates for the fixed point, (x_f, y_f), are often chosen at some object position, such as its centroid but any other spatial position can be selected.

Objects are now resized by scaling the distances between object points and the point For a coordinate position (x, y), the scaled coordinates (x′, y′) are then calculated from the following relationships:

$$x' - x_f = (x - x_f)s_x \qquad (13)$$

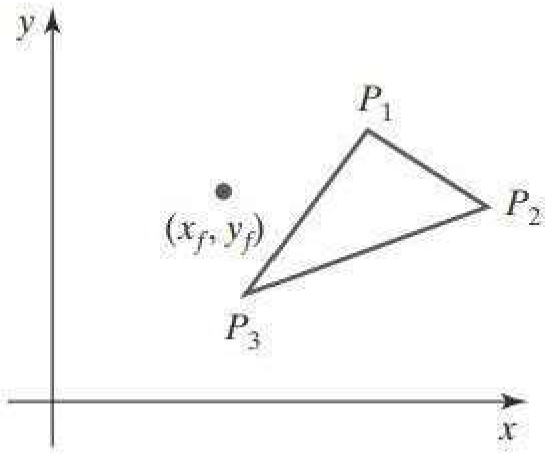
$$y' - y_f = (y - y_f)s_y$$

The above equation can be rewritten to separate the multiplicative and additive terms as

$$x' = x.s_x + x_f(1 - s_x) \qquad (14)$$

$$y' = y.s_y + y_f(1 - s_y)$$

The additive terms $x_f(1-s_x)$ and $y_f(1-s_y)$ are constants for all points in the object.



Scaling relative to a chosen fixed point(x_f , y_f). The distance from each polygonvertex to the fixed point is scaled by Equations 13

Matrix Representations and Homogeneous Coordinates

Each of the three basic two-dimensional transformations (translation, rotation, and scaling) can be expressed in the general matrix form.

$P' = M_1 . P + M_2$ (15)

With coordinate positions **P** and **P'** represented as column vectors. Matrix **M_1** is a 2 × 2 array containing multiplicative factors, and **M_2** is a two-element column matrix containing translational terms.

For translation, **M_1** is the identity matrix. For rotation or scaling, **M_2** contains the translational terms associated with the pivot point or scaling fixed point.

Homogeneous Coordinates

Multiplicative and translational terms for a two-dimensional geometric transformation can be combined into a single matrix, if representations are expanded to 3 × 3 matrices. The third column of a transformation matrix can be used for the translation terms, and all transformation equations can be expressed as matrix multiplications.

The matrix representation for a two-dimensional coordinate position is expanded to a three-element column matrix. Each two dimensional coordinate-position representation(x,y) is expanded to a three-element representation (x_h,y_h,h) called **homogeneous coordinates.**

The homogeneous parameter **h** is a nonzero value such that

$\mathbf{x=x_h/h}$ **(16)**

$\mathbf{y=y_h/h}$

A general two-dimensional homogeneous coordinate representation could also be written as

(h· x, h· y, h).

A convenient choice is simply to set h = 1. Each two-dimensional position is then represented with homogeneous coordinates (x,y,1). The term homogeneous coordinates is used in mathematics to refer to the effect of this representation on Cartesian equations.

Expressing positions in homogeneous coordinates allows us to represent all geometric transformation equations as matrix multiplications, which is the standard method used in graphics systems.

Two-Dimensional Translation Matrix

The homogeneous-coordinate for translation is given by

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \textbf{(17)}$$

This translation operation can be written in the abbreviated form

$\mathbf{P'=T(t_x,t_y).P}$ **(18)**

with $\mathbf{T(t_x, t_y)}$ as the 3 × 3 translation matrix in Equation 17

Two-Dimensional Rotation Matrix

Two-dimensional rotation transformation equations about the coordinate origin can be expressed in the matrix form as

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

(19)

or as

P'=R(θ).P (20)

The rotation transformation operator **R(θ)** is the 3 × 3 matrix with rotation parameter **θ**.

Two-Dimensional Scaling Matrix

A scaling transformation relative to the coordinate origin can be expressed as the matrix multiplication.

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

(21)

$P'=S(s_x,s_y).P$ (22)

The scaling operator **S**(s_x,s_y) is the 3x3 matrix with parameters s_x and s_y.

Inverse Transformations

For translation, inverse matrix is obtained by negating the translation distances. If the two dimensional translation distances are t_x and t_y , then inverse translation matrix is

$$\mathbf{T}^{-1} = \begin{bmatrix} 1 & 0 & -t_x \\ 0 & 1 & -t_y \\ 0 & 0 & 1 \end{bmatrix}$$ **(23)**

An inverse rotation is accomplished by replacing the rotation angle by its negative. A two-dimensional rotation through an angle **θ** about the coordinate origin has the inverse transformation matrix

$$\mathbf{R}^{-1} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$ **(24)**

Negative values for rotation angles generate rotations in a clockwise direction.

The inverse matrix for any scaling transformation is obtained by replacing the scaling parameters with their reciprocals. For two-dimensional scaling with parameters s_x and s_y applied relative to the coordinate origin, the inverse transformation matrix is

$$\mathbf{S}^{-1} = \begin{bmatrix} \frac{1}{s_x} & 0 & 0 \\ 0 & \frac{1}{s_y} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$ **(25)**

The inverse matrix generates an opposite scaling transformation.

Two-Dimensional Composite Transformations

Forming products of transformation matrices is referred to as a **concatenation,** or **composition,** of matrices. If two transformations are applied to point position P, the transformed location would be calculated as:

$P' = M_2 . M_1 . P$

$P' = M . P$ (26)

The coordinate position is transformed using the composite matrix **M**, rather than applying the individual transformations **M_1** and then **M_2**.

Composite Two-Dimensional Translations

If two successive translation vectors (t_{1x}, t_{1y}) and (t_{2x}, t_{2y}) are applied to a two dimensional coordinate position **P.** The final transformed location **P′** is calculated as

$P' = T(t_{2x}, t_{2y}) . \{T(t_{1x}, t_{1y}) . P\}$

$P' = \{T(t_{2x}, t_{2y}) . T(t_{1x}, t_{1y})\} . P$ (27)

where P and P′ are represented as three-element, homogeneous-coordinate column vectors.

The composite transformation matrix for this sequence of translations is

$$\begin{bmatrix} 1 & 0 & t_{2x} \\ 0 & 1 & t_{2y} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & t_{1x} \\ 0 & 1 & t_{1y} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_{1x} + t_{2x} \\ 0 & 1 & t_{1y} + t_{2y} \\ 0 & 0 & 1 \end{bmatrix} \quad \textbf{(28)}$$

or

$T(t_{2x}, t_{2y}) . T(t_{1x}, t_{1y}) = T(t_{1x} + t_{2x}, t_{1y} + t_{2y})$ (29)

Composite Two-Dimensional Rotations

Two successive rotations applied to a point **P** produce the transformed position

$P' = R(\theta_2) . \{R(\theta_1) . P\}$

$= \{R(\theta_2) . \{R(\theta_1) . P\}$ **(30)**

By multiplying the two rotation matrices, it can be verified that two successive rotations are additive.

$R(\theta_2).R(\theta_1)=R(\theta_1+\theta_2)$ (31)

The final rotated coordinates of a point can be calculated with the composite rotation matrix as

$P'=R(\theta_1+\theta_2).P$ (32)

Composite Two-Dimensional Scalings

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

$$\begin{bmatrix} s_{2x} & 0 & 0 \\ 0 & s_{2y} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_{1x} & 0 & 0 \\ 0 & s_{1y} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_{1x} \cdot s_{2x} & 0 & 0 \\ 0 & s_{1y} \cdot s_{2y} & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (33)$$

or

$S(s_{2x},s_{2y}).\ S(s_{1x},s_{1y})=S(s_{1x}.s_{2x},s_{1y}.s_{2y})$ (34)

General Two-Dimensional Pivot-Point Rotation

Two-dimensional rotation about any other pivot point (x_r,y_r) can be generated by performing the following sequence of translate-rotate-translate operations:

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

The composite transformation matrix for this sequence is obtained with the concatenation

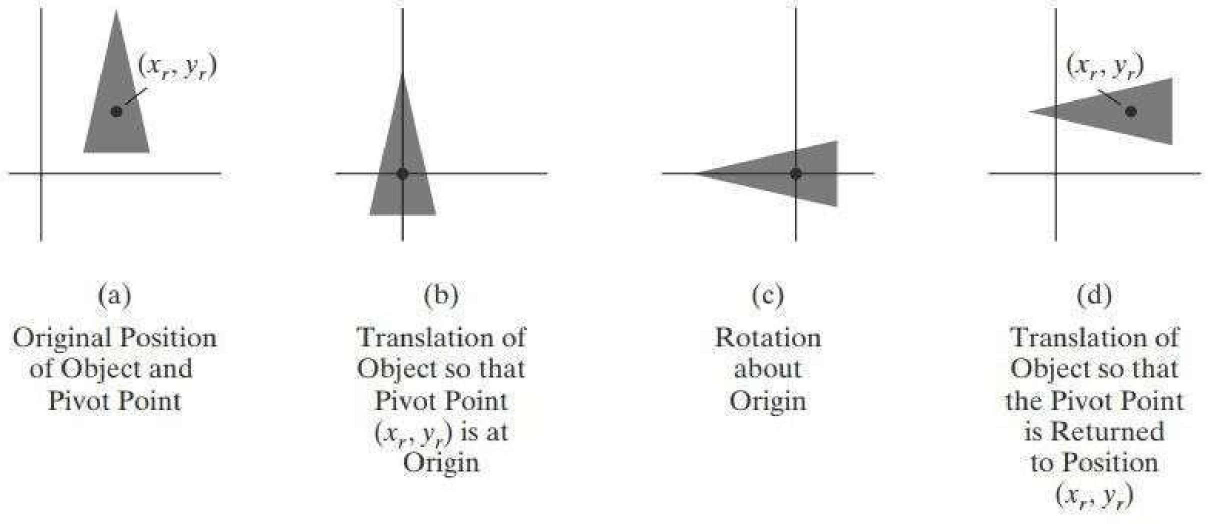
$$\begin{bmatrix} 1 & 0 & x_r \\ 0 & 1 & y_r \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_r \\ 0 & 1 & -y_r \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\theta & -\sin\theta & x_r(1-\cos\theta) + y_r\sin\theta \\ \sin\theta & \cos\theta & y_r(1-\cos\theta) - x_r\sin\theta \\ 0 & 0 & 1 \end{bmatrix} \quad \textbf{(35)}$$

which can be expressed in form

$T(x_r,y_r).R(\theta).T(-x_r,-y_r)=R(x_r,y_r,\theta)$ (36)

where $\mathbf{T}(-x_r,-y_r) = \mathbf{T}^{-1}(x_r,y_r)$


(x_r, y_r)
(x_r, y_r)
(a)
Original Position of Object and Pivot Point
(b)
Translation of Object so that Pivot Point (x_r, y_r) is at Origin
(c)
Rotation about Origin
(d)
Translation of Object so that the Pivot Point is Returned to Position (x_r, y_r)

A transformation sequence for rotating an object about a specified pivot pointusing the rotation matrix R(θ)

General Two-Dimensional Fixed-Point Scaling

To produce a two-dimensional scaling with respect to a selected fixed position (x_f,y_f), following sequence is followed.

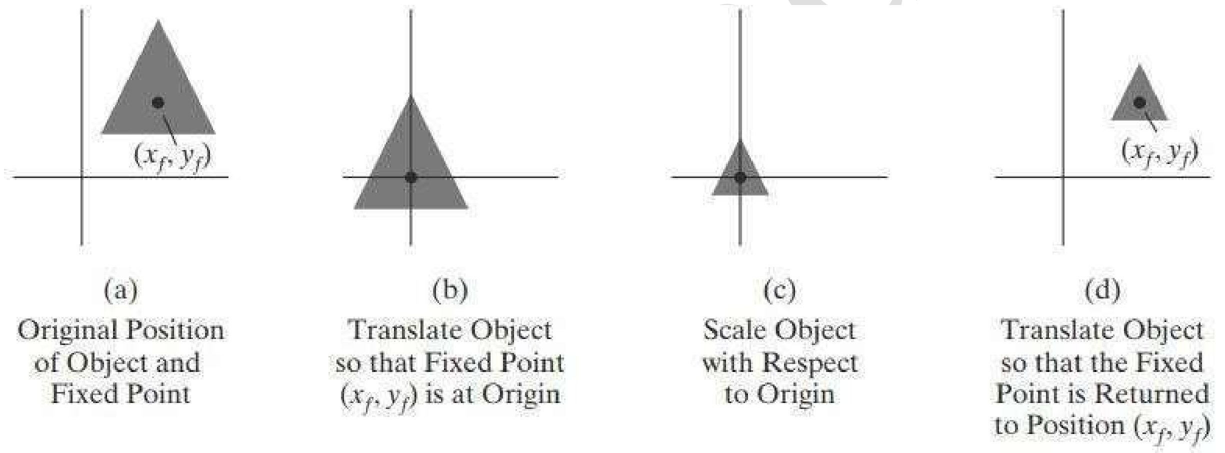
1. Translate the object so that the fixed point coincides with the coordinate origin.
2. Scale the object with respect to the coordinate origin.
3. Use the inverse of the translation in step (1) to return the object to its original position.

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

$$\begin{bmatrix} 1 & 0 & x_f \\ 0 & 1 & y_f \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_f \\ 0 & 1 & -y_f \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & x_f(1-s_x) \\ 0 & s_y & y_f(1-s_y) \\ 0 & 0 & 1 \end{bmatrix} \quad \textbf{(37)}$$

or

$$\mathbf{T(x_f,y_f).S(s_x,s_y).T(-x_f,-y_f)=S(x_f,y_f,s_x,s_y)} \quad \textbf{(38)}$$


(x_f, y_f)
(a)
Original Position of Object and Fixed Point
(b)
Translate Object so that Fixed Point (x_f, y_f) is at Origin
(c)
Scale Object with Respect to Origin
(d)
Translate Object so that the Fixed Point is Returned to Position (x_f, y_f)

A transformation sequence for scaling an object with respect to a specified fixed position using the scaling matrix S(s_x, s_y)

Other Two-Dimensional Transformations

Basic transformations such as translation, rotation, and scaling are standard components of graphics libraries. Some packages provide a few additional transformations that are useful in certain applications.

Two such transformations are reflection and shear.

1. Reflection and
2. Shear.

Reflection

A transformation that produces a mirror image of an object is called a **reflection.**

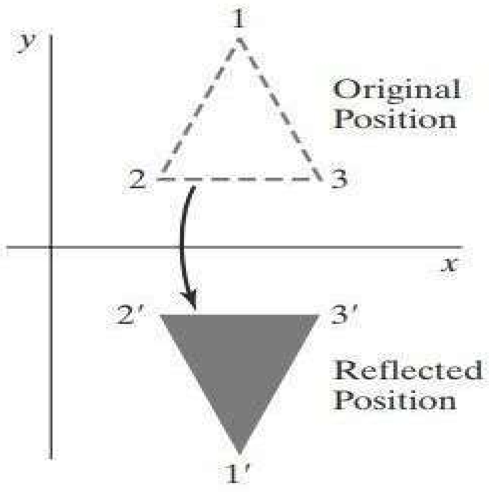
For a two-dimensional reflection, this image is generated relative to an **axis of reflection** by rotating the object 180◦ about the reflection axis.

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

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \textbf{(39)}$$

This transformation retains x values, but **"**flips**"** the y values of coordinate positions.

The resulting orientation of an object after it has been reflected about the x axis is showni

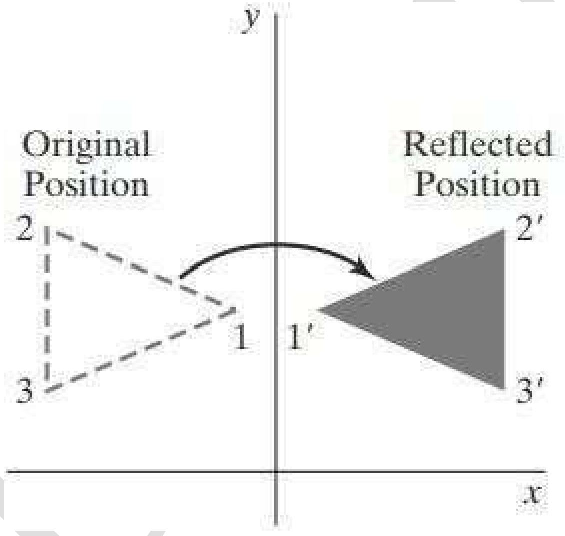


Reflection of an object about the x axis

A reflection about the line $x = 0$ (the y axis) flips x coordinates while keeping y coordinates the same. The matrix for this transformation is

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \textbf{(40)}$$

Below illustrates the change in position of an object that has been reflected aboutthe line $x = 0$.

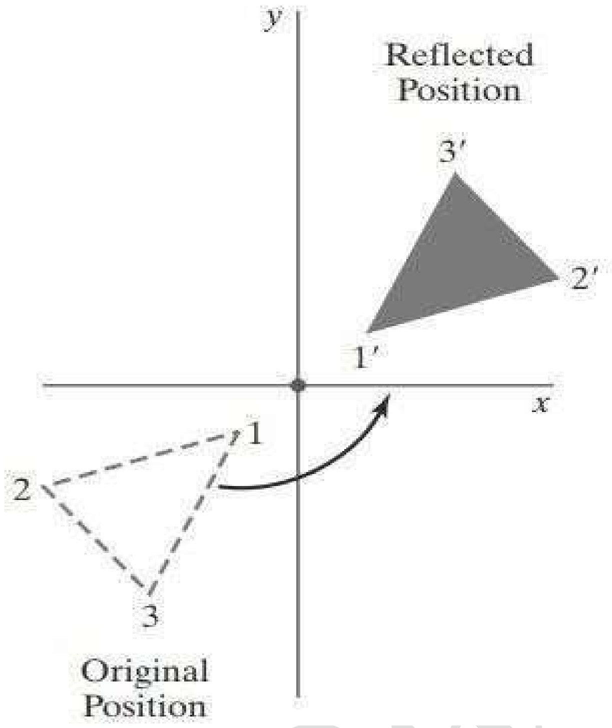


Reflection of an object about the y axis.

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. The matrix representation for this reflection is

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \textbf{(41)}$$

An example of reflection about the origin is shown here

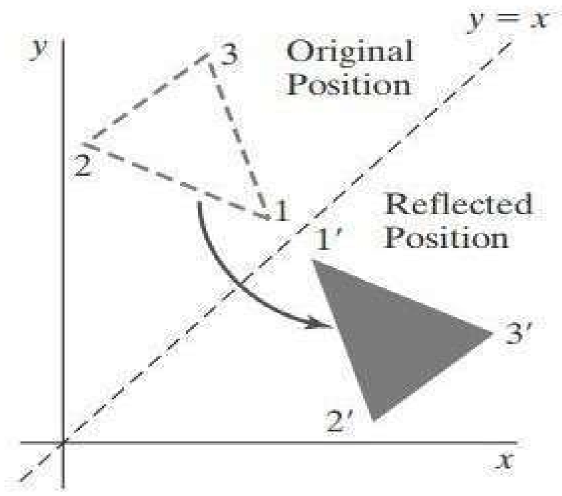


Reflection of an object relative to the coordinate origin. This transformationcan be accomplished with a rotation in the x y plane about the coordinate origin.

If we choose the reflection axis as the diagonal line *y* = *x* , the reflection matrix is

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

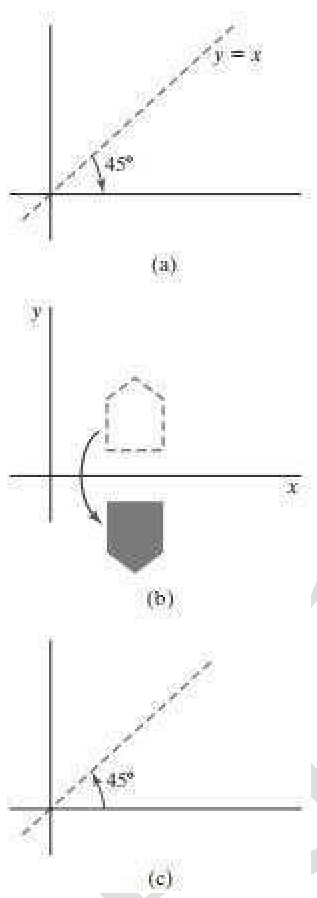
(42)



Reflection of an object with respect to the line y = x .

To obtain a transformation matrix for reflection about the diagonal y = x, concatenate matrices for the transformation sequence:

1. Clockwise rotation by 45◦,
2. Reflection about the x axis
3. Counterclockwise rotation by 45◦



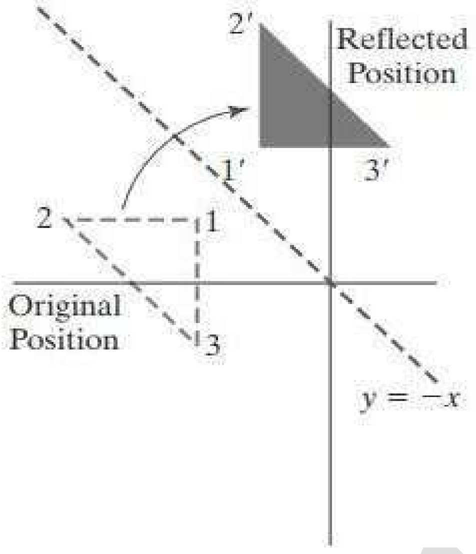
Sequence of transformations to produce a reflection about the line y = x : A clockwise rotation of 45◦ (a), a reflection about the x axis (b), and a counterclockwise rotation by 45◦ (c)

To obtain a transformation matrix for reflection about the diagonal ***y = -x***, we could concatenate matrices for the transformation sequence:

1. Clockwise rotation by 45◦,
2. Reflection about the *y* axis
3. Counterclockwise rotation by 45◦.

The resulting transformation matrix is

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



Reflection with respect to the line y = -x .

Shear

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

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

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

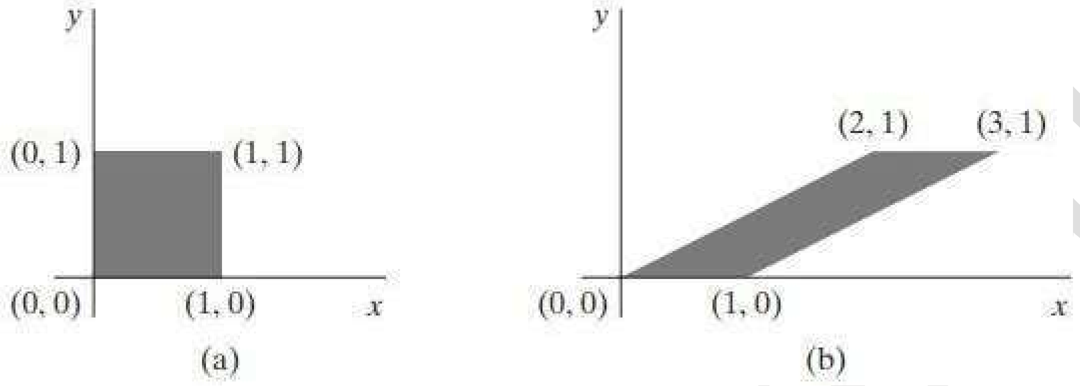
$$\begin{bmatrix} 1 & sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$ **(44)**

which transforms coordinate positions as

$x' = x + sh_x . y$ **(45)**

$y' = y$

Any real number can be assigned to the shear parameter sh_x. A coordinate position (x, y) is then shifted horizontally by an amount proportional to its perpendicular distance (y value) from the x axis. Setting parameter sh_x to the value 2, for example, changes the square in Figure below into a parallelogram. Negative values for sh_x shift coordinate positions to the left.



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

We can generate *x*-direction shears relative to other reference lines with

$$\begin{bmatrix} 1 & sh_x & -sh_x \cdot y_{\text{ref}} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \textbf{(46)}$$

Coordinate positions are transformed as

$$x' = x + sh_x(y - y_{\text{ref}}) \qquad \textbf{(47)}$$

$$y' = y$$

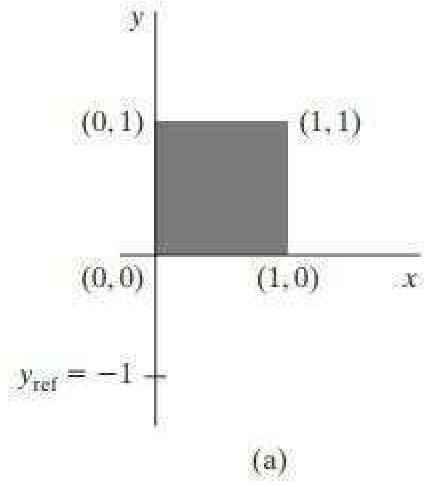
A *y*-direction shear relative to the line $x = x_{\text{ref}}$ is generated with the transformation matrix

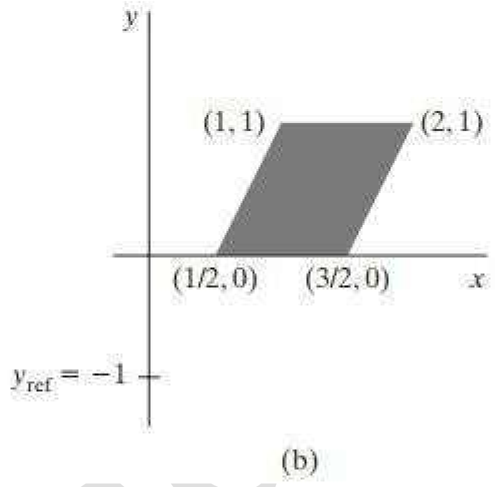
$$\begin{bmatrix} 1 & 0 & 0 \\ sh_y & 1 & -sh_y \cdot x_{\text{ref}} \\ 0 & 0 & 1 \end{bmatrix} \qquad \textbf{(48)}$$

which generates transformed coordinates as

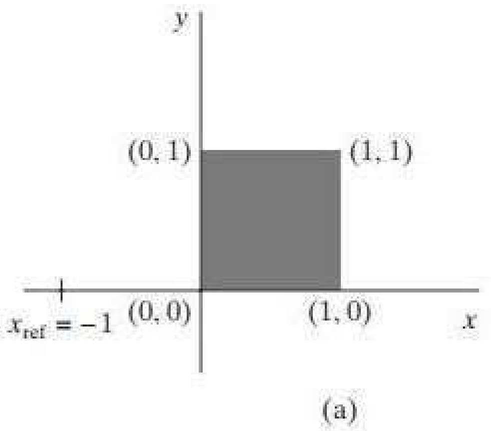
$$x' = x \qquad (49)$$

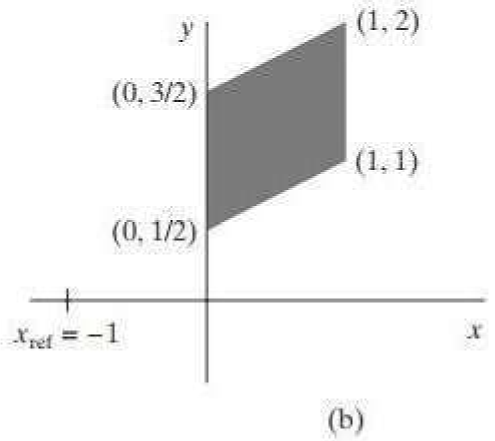
$$y' = y + sh_y(x - x_{ref})$$





A unit square (a) is transformed to a shifted parallelogram (b) with $sh_x = 0.5$and $y_{ref} = -1$ in the shear matrix 46.





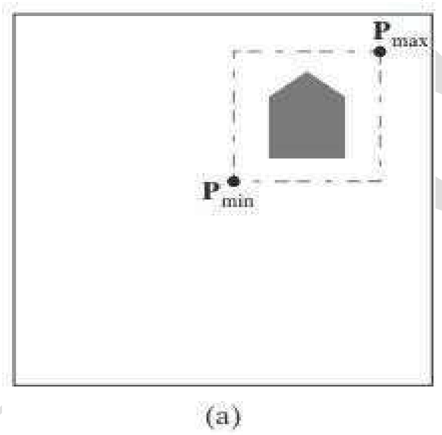
A unit square (a) is turned into a shifted parallelogram (b) with parametervalues $sh_y = 0.5$ and $x_{ref} = -1$ in the y -direction shearing transformation 48.

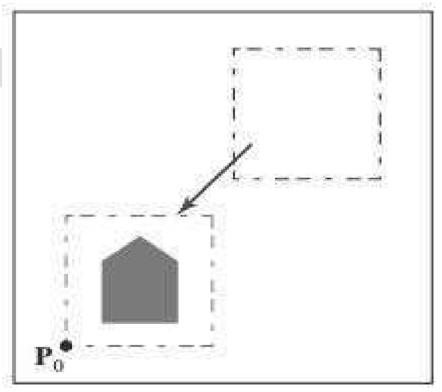
Raster Methods for Geometric Transformations

Raster systems store picture information as color patterns in the frame buffer. Therefore, some simple object transformations can be carried out rapidly by manipulating an array of pixel values. Few arithmetic operations are needed, so the pixel transformations are particularly efficient.

Functions that manipulate rectangular pixel arrays are called **raster operations** and moving a block of pixel values from one position to another is termed a block transfer, a bitblt, or a pixblt.

Figure below illustrates a two-dimensional translation implemented as a block transfer ofa refresh-buffer area





Translating an object from screen position (a) to the destination position shown in (b) by moving a rectangular block of pixel values. Coordinate positions P_{min} andP_{max} specify the limits of the rectangular block to be moved, and P_0 is the destination reference position

Rotations in 90-degree increments are accomplished easily by rearranging the elements of a pixel array. We can rotate a two-dimensional object or pattern 90◦ counterclockwise by reversing the pixel values in each row of the array, then interchanging rows and columns

A 180◦ rotation is obtained by reversing the order of the elements in each row of the array, then reversing the order of the rows.

Figure below demonstrates the array manipulations that can be used to rotate a pixel block by 90◦ and by 180◦.

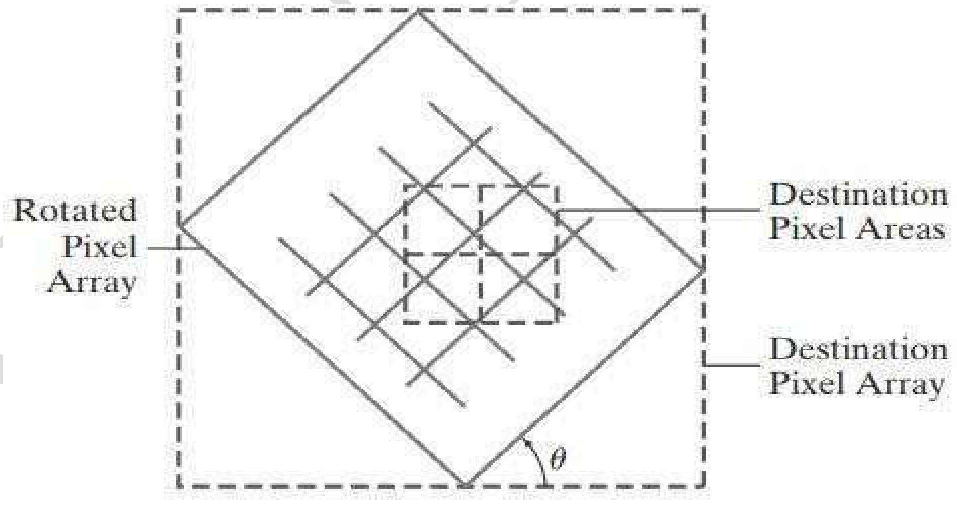
$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix} \qquad \begin{bmatrix} 3 & 6 & 9 & 12 \\ 2 & 5 & 8 & 11 \\ 1 & 4 & 7 & 10 \end{bmatrix} \qquad \begin{bmatrix} 12 & 11 & 10 \\ 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix}$$

(a) (b) (c)

Rotating an array of pixel values. The original array is shown in (a), the positions of the array elements after a 90◦ counterclockwise rotation are shown in (b), andthe positions of the array elements after a 180◦ rotation are shown in (c).

For array rotations that are not multiples of 90◦, we need to do some extra processing.

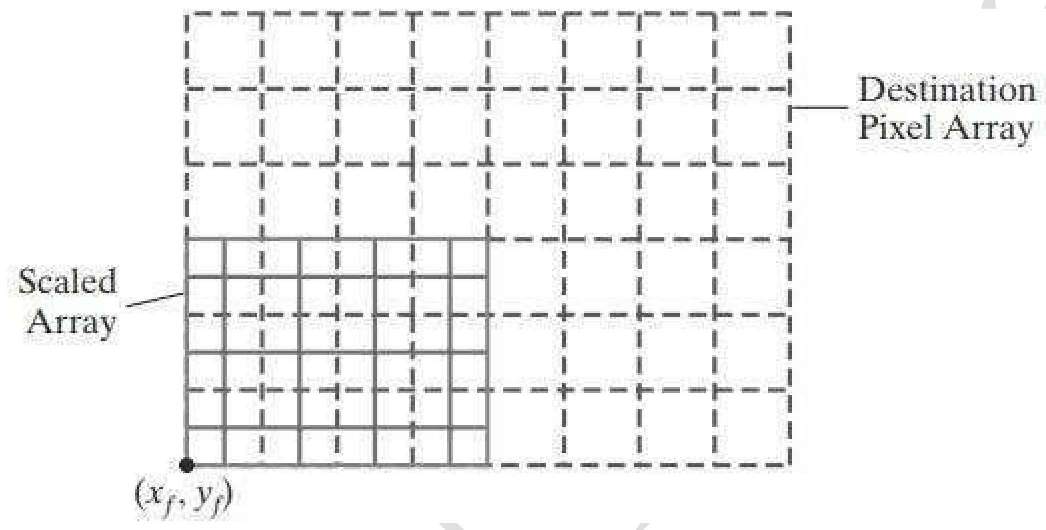
The general procedure is illustrated in Figure below



A raster rotation for a rectangular block of pixels can be accomplished bymapping the destination pixel areas onto the rotated block.

Each destination pixel area is mapped onto the rotated array and the amount of overlap with the rotated pixel areas is calculated. A color for a destination pixel can then be computed by averaging the colors of the overlapped source pixels, weighted by their percentage of area overlap.

Similar methods can be used to scale a block of pixels. Pixel areas in the original block are scaled, using specified values for s_x and s_y, and then mapped onto a set of destination pixels. The color of each destination pixel is then assigned according to its area of overlap with the scaled pixel areas. (Figure below)



Mapping destination pixel areas onto a scaled array of pixel values. Scalingfactors $s_x = s_y = 0.5$ are applied relative to fixed point (x_f, y_f).

OpenGL Raster Transformations

A translation of a rectangular array of pixel-color values from one buffer area to another can be accomplished in OpenGL as the following copy operation:

glCopyPixels (xmin, ymin, width, height, GL_COLOR);

The first four parameters in this function give the location and dimensions of the pixel block; and the OpenGL symbolic constant **GL_COLOR** specifies that it is color values are to be copied

A block of RGB color values in a buffer can be saved in an array with the function

glReadPixels (xmin, ymin, width, height, GL_RGB, GL_UNSIGNED_BYTE, colorArray);
If color-table indices are stored at the pixel positions, we replace the constant GL RGB with GL_COLOR_INDEX.
To rotate the color values, we rearrange the rows and columns of the color array, as described in the previous section. Then we put the rotated array back in the buffer with
glDrawPixels (width, height, GL_RGB, GL_UNSIGNED_BYTE, colorArray);

A two-dimensional scaling transformation can be performed as a raster operation in OpenGL by specifying scaling factors and then invoking either **glCopyPixels** or **glDrawPixels**.
For the raster operations, we set the scaling factors with
glPixelZoom (sx, sy);
We can also combine raster transformations with logical operations to produce various effects with the *exclusive or* operator

OpenGL Functions for Two-Dimensional Geometric Transformations

In the core library of OpenGL, a separate function is available for each of the basic geometric transformations. OpenGL is designed as a three-dimensional graphics application programming interface (API), all transformations are specified in three dimensions. Internally, all coordinate positions are represented as four-element column vectors, and all transformations are represented using 4 × 4 matrices.

To perform a translation, we invoke the translation routine and set the components for the three-dimensional translation vector.

In the rotation function, we specify the angle and the orientation for a rotation axis that intersects the coordinate origin. A scaling function is used to set the three coordinate scaling factors relative to the coordinate origin. In each case, the transformation routine sets up a 4 × 4 matrix that is applied to the coordinates of objects that are referenced after the transformation call

Basic OpenGL Geometric Transformations

A 4× 4 translation matrix is constructed with the following routine:

glTranslate* (tx, ty, tz);

Translation parameters **tx**, **ty**, and **tz** can be assigned any real-number values, and the single suffix code to be affixed to this function is either **f** (float) or **d** (double).

For two-dimensional applications, we set **tz** = 0.0; and a two-dimensional position is represented as a four-element column matrix with the *z* component equal to 0.0.

Example: glTranslatef (25.0, -10.0, 0.0);

Using above statement defined coordinate positions is translated 25 units in the x direction and -10 units in the y direction.

A 4 × 4 rotation matrix is generated with

glRotate* (theta, vx, vy, vz);

where the vector **v** = (**vx**, **vy**, **vz**) can have any floating-point values for its components.

This vector defines the orientation for a rotation axis that passes through the coordinate origin.

If **v** is not specified as a unit vector, then it is normalized automatically before the elements of the rotation matrix are computed.

The suffix code can be either **f** or **d**, and parameter **theta** is to be assigned a rotation angle in degree.

For example, the statement: **glRotatef (90.0, 0.0, 0.0, 1.0);**

sets up the matrix for a 90◦ rotation about the *z* axis.

We obtain a 4 × 4 scaling matrix with respect to the coordinate origin with the following routine:

glScale* (sx, sy, sz);

The suffix code is again either **f** or **d**, and the scaling parameters can be assigned any real-number values.

Scaling in a two-dimensional system involves changes in the *x* and *y* dimensions, so a typical two-dimensional scaling operation has a *z* scaling factor of 1.0

Example: glScalef (2.0, -3.0, 1.0);

The above statement produces a matrix that scales by a factor of 2 in the *x* direction, scales by a factor of 3 in the *y* direction, and reflects with respect to the *x* axis:

OpenGL Matrix Operations

The glMatrixMode routine is used to set the projection mode which designates the matrix that is to be used for the projection transformation.

modelview mode is specified with the following statement

glMatrixMode (GL_MODELVIEW);

which designates the 4×4 modelview matrix as the **current matrix**

Two other modes that can be set with the **glMatrixMode** function are the ***texture mode*** and the ***color mode***.

The texture matrix is used for mapping texture patterns to surfaces, and the color matrix is used to convert from one color model to another. The default argument for the **glMatrixMode** function is **GL_MODELVIEW**.

Identity matrix is assigned to the current matrix using following function:

glLoadIdentity( );

Other values can be assigned to the elements of the current matrix using

glLoadMatrix* (elements16);

A single-subscripted, 16-element array of floating-point values is specified with parameter **elements16**, and a suffix code of either **f** or **d** is used to designate the data type. The elements in this array must be specified in *column-major* order

To illustrate this ordering, we initialize the modelview matrix with the following code:

```
glMatrixMode (GL_MODELVIEW);
GLfloat elems [16];
GLint k;
 for (k = 0; k < 16; k++)
     elems [k] = float (k);
 glLoadMatrixf (elems);
```

which produces the matrix

$$\mathbf{M} = \begin{bmatrix} 0.0 & 4.0 & 8.0 & 12.0 \\ 1.0 & 5.0 & 9.0 & 13.0 \\ 2.0 & 6.0 & 10.0 & 14.0 \\ 3.0 & 7.0 & 11.0 & 15.0 \end{bmatrix}$$

A specified matrix can be concatenated with the current matrix as follows:

glMultMatrix* (otherElements16);

The suffix code is either **f** or **d**, and parameter **otherElements16** is a 16-element, single-subscripted array that lists the elements of some other matrix in column-major order. Assuming that the current matrix is the modelview matrix, which we designate as **M**, then the updated modelview matrix is computed as

M = M· M'

M′ represents the matrix whose elements are specified by parameter **otherElements16** in the preceding **glMultMatrix** statement.

The **glMultMatrix** function can also be used to set up any transformation sequence with individually defined matrices.

For example,

glMatrixMode (GL_MODELVIEW);

glLoadIdentity ( ); // Set current matrix to the identity.

glMultMatrixf (elemsM2); // Postmultiply identity with matrix M2.

glMultMatrixf (elemsM1); // Postmultiply M2 with matrix M1.

produces the following current modelview matrix:

$\mathbf{M} = \mathbf{M}_2 \cdot \mathbf{M}_1$

Three-Dimensional Geometric Transformations

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

A three-dimensional position, expressed in homogeneous coordinates, is represented as a four-element column vector.

Three-Dimensional Translation

A position **P** = (x, y, z) in three-dimensional space is translated to a location **P′**=(x′,y′,z′) by adding translation distances t_x, t_y, and t_z to the Cartesian coordinates of **P.**

$$x' = x + t_x$$

$$y' = y + t_y$$

$$z' = z + t_z$$

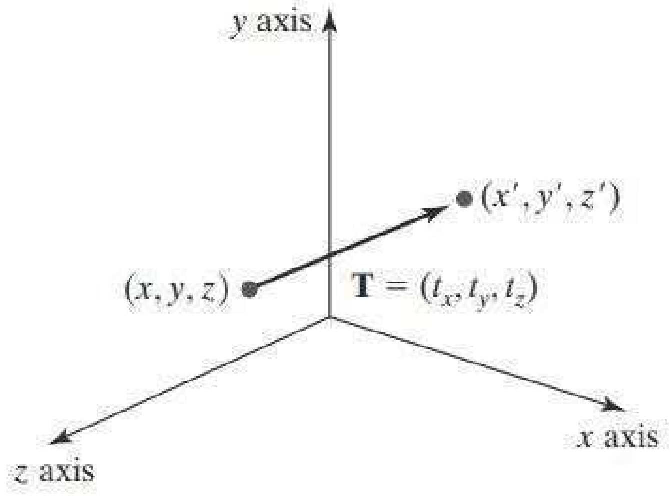
Three-dimensional translation operations can be represented in matrix form.

The coordinate positions, **P** and **P′** , are represented in homogeneous coordinates with four-element column matrices, and the translation operator **T** is a 4 × 4 matrix:

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

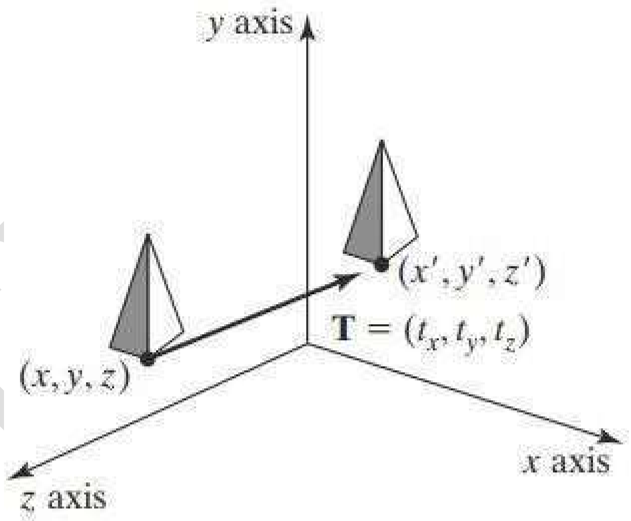
or

P’=T.P



Moving a coordinate position with translation vector T = (t_x, t_y, t_z)

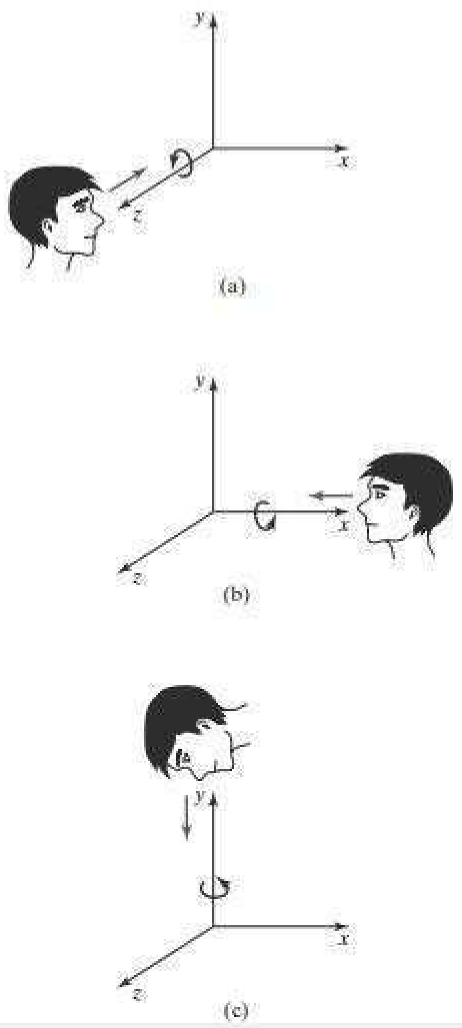
An inverse of a three-dimensional translation matrix is obtained by negating the translation distances t_x, t_y, and t_z



Shifting the position of a three-dimensional object using translation vector T

Three-Dimensional Rotation

By convention, positive rotation angles produce counterclockwise rotations about a coordinate axis. Positive rotations about a coordinate axis are counterclockwise, when looking along the positive half of the axis toward the origin.



Positive rotations about a coordinate axis are counterclockwise, when lookingalong the positive half of the axis toward the origin.

Three-Dimensional Coordinate-Axis Rotations

Three dimensional z-axis rotation equation is as follows:

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

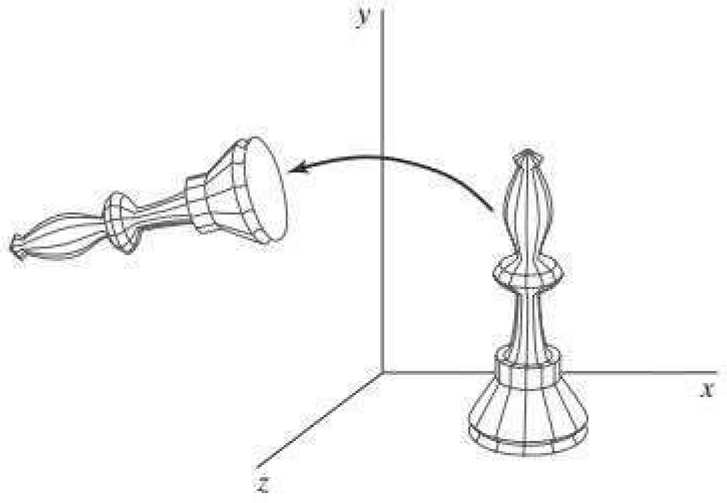
$y'=x\sin\theta+y\cos\theta$ **(50)**

$z'=z$

Parameter θ specifies the rotation angle about the z axis, and z-coordinate values are unchanged by this transformation. In homogeneous-coordinate form, the three-dimensional *z*-axis rotation equations are:

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

$$\mathbf{P' = R_z(\theta) . P}$$



Rotation of an object about the z axis

Figure illustrates rotation of an object about z-axis.

Transformation equations for rotations about the other two coordinate axes can be obtained with a cyclic permutation of the coordinate parameters x, y, and z in equation 50.

x → y → z → x (51)

To obtain the x-axis and y-axis rotation transformations, cyclically replace x with y, y with z, and z with x.

By substituting permutations 51 in equation 50, equations for x-axis rotation is obtained.

The equation for x axis rotation is:

$$y' = y\cos\theta - z\sin\theta \qquad (52)$$

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

$$x' = x$$

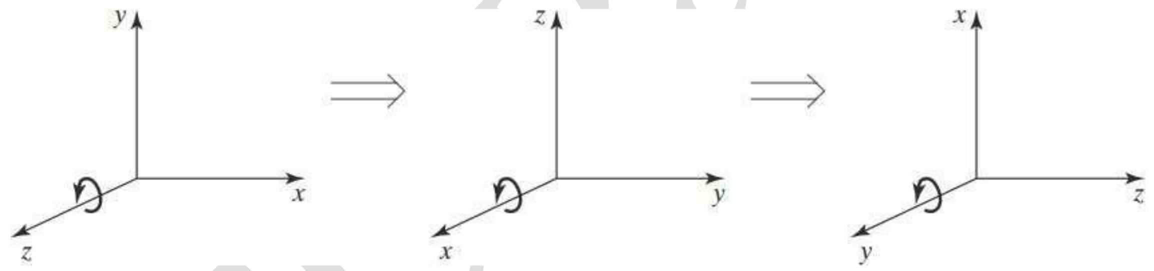
A cyclic permutation of coordinates in Equations 52 gives us the transformation equations for a *y*-axis rotation:

The equation for y- axis rotation is:

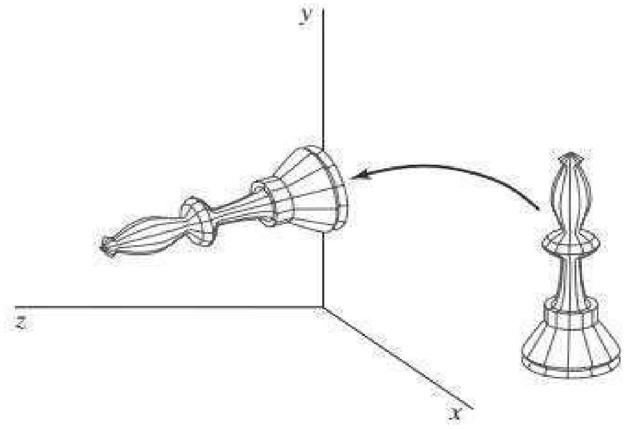
$$z' = z\cos\theta - x\sin\theta \qquad (53)$$

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

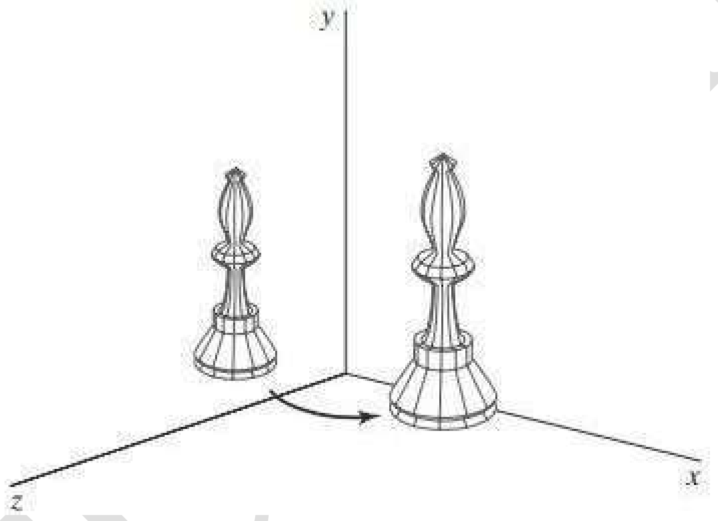
$$y' = y$$



Cyclic permutation of the Cartesian-coordinate axes to produce the three setsof coordinate-axis rotation equations



Rotation of an object about the x axis



Rotation of an object about the y axis

Negative values for rotation angles generate rotations in a clockwise direction, and the identity matrix is produced when we multiply any rotation matrix by its inverse.

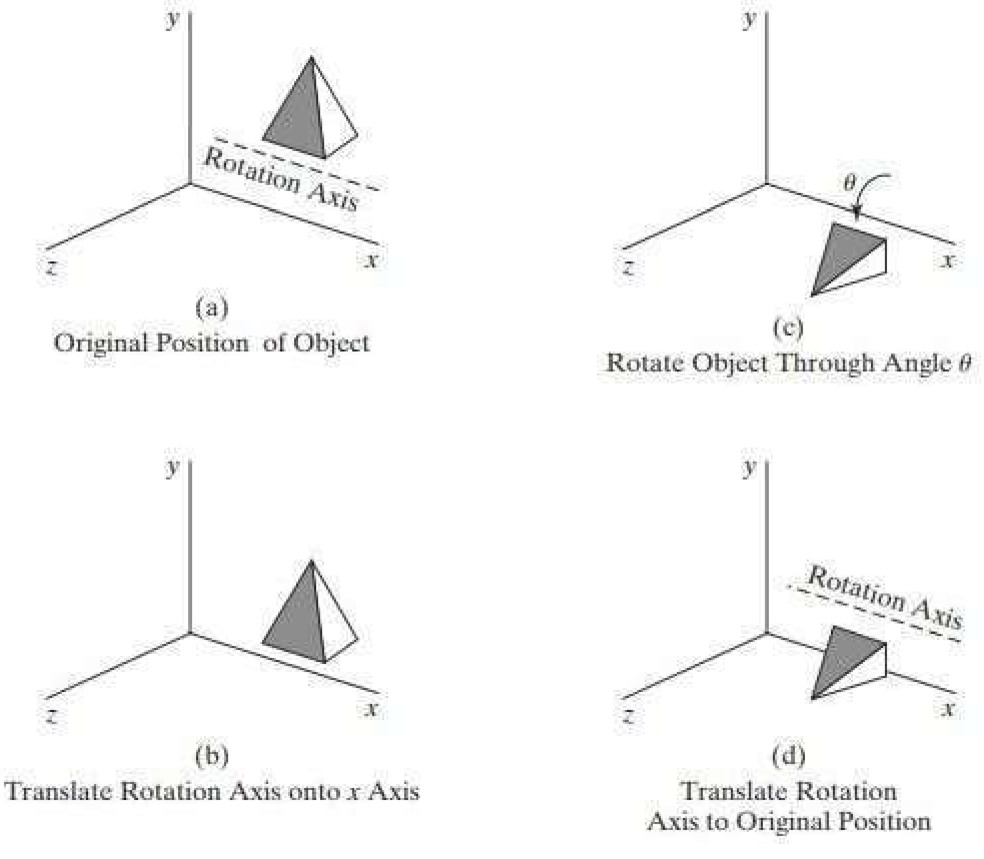
i.e $\mathbf{RR^{-1}=I}$

General Three-Dimensional Rotations

A rotation matrix for any axis that does not coincide with a coordinate axis can be set up as a composite transformation involving combinations of translations and the coordinate axis rotations.

The following transformation sequence is used:

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


y
Rotation Axis
z
x
(a)
Original Position of Object
y
θ
z
x
(c)
Rotate Object Through Angle θ
y
z
x
(b)
Translate Rotation Axis onto x Axis
y
Rotation Axis
z
x
(d)
Translate Rotation
Axis to Original Position

Sequence of transformations for rotating an object about an axis that isparallel to the x axis.

A coordinate position **P** is transformed with the sequence

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

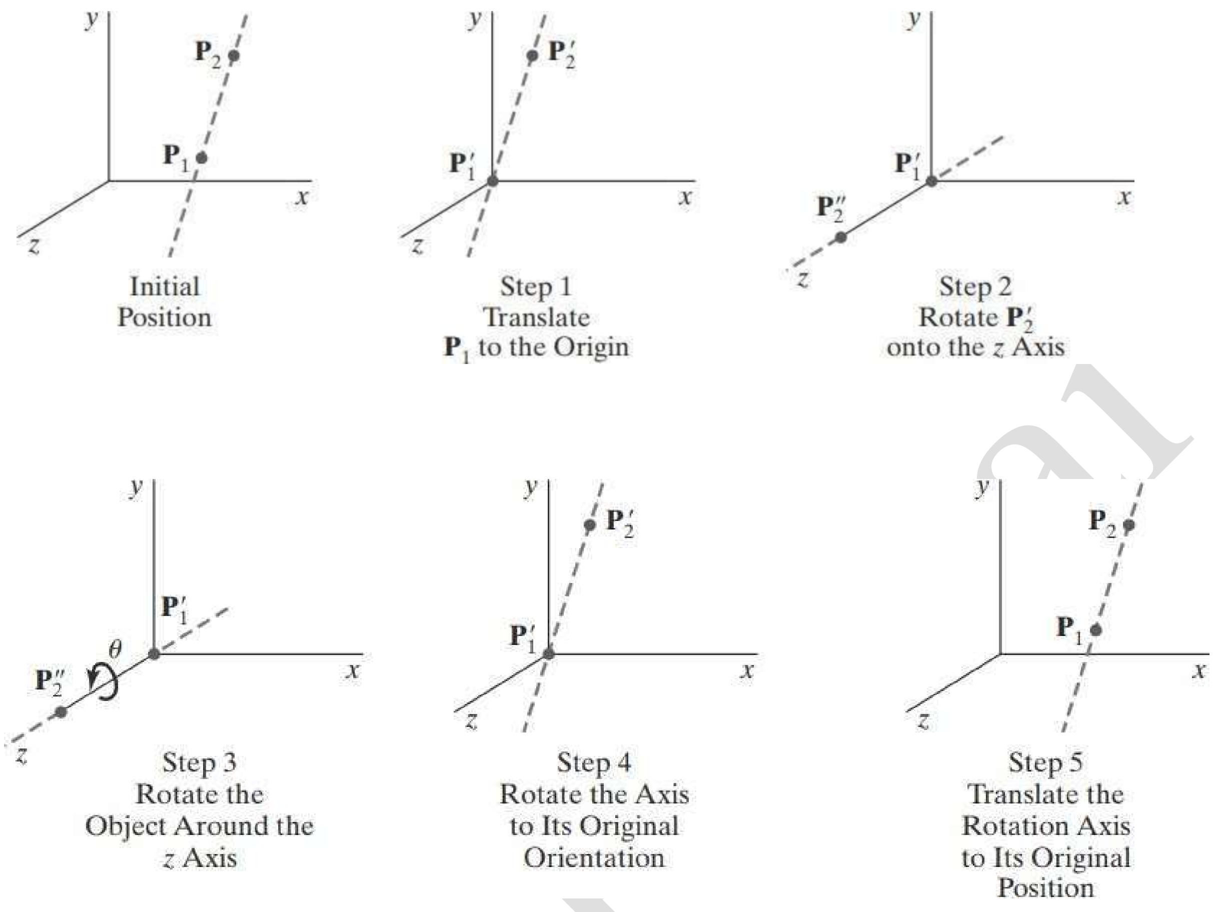
where the composite rotation matrix for the transformation is

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

When an object is to be rotated about an axis that is not parallel to one of the coordinate axes, some additional transformations has to be performed.

The required rotation can be accomplished in five steps:

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


y
x
z
P₂
P₁
Initial Position
P'₂
P'₁
Step 1 Translate P₁ to the Origin
P''₂
Step 2 Rotate P'₂ onto the z Axis
θ
Step 3 Rotate the Object Around the z Axis
Step 4 Rotate the Axis to Its Original Orientation
Step 5 Translate the Rotation Axis to Its Original Position

Five transformation steps for obtaining a composite matrix for rotation aboutan arbitrary axis, with the rotation axis projected onto the z axis.

Three-Dimensional Scaling

The matrix expression for the three-dimensional scaling transformation of a position **P** =(x, y, z) is given by

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

(54)

The three-dimensional scaling transformation for a point position can be represented as

$P' = S.P$

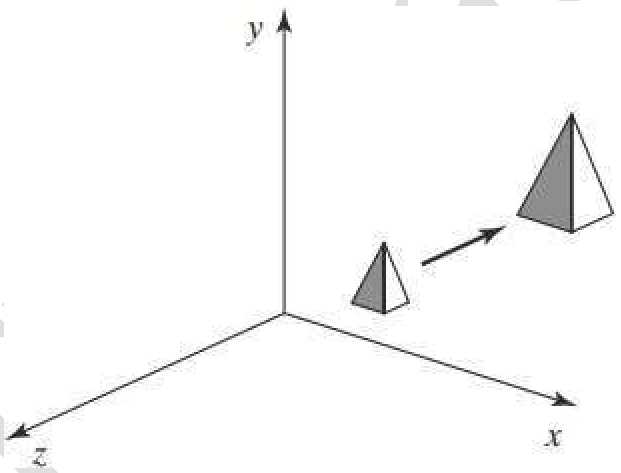
where scaling parameters s_x, s_y, and s_z are assigned any positive values

Explicit expressions for the scaling transformation relative to the origin are

$x' = x \cdot s_x$, $y' = y \cdot s_y$, $z' = z \cdot s_z$

Scaling an object with transformation (54) changes the position of object relative to the coordinate origin. A parameter value greater than 1 move a point farther from the origin. A parameter value less than 1 move a point closer to the origin.

Uniform scaling is performed when $s_x = s_y = s_z$. If the scaling parameters are not all equal, relative dimensions of a transformed object are changed.



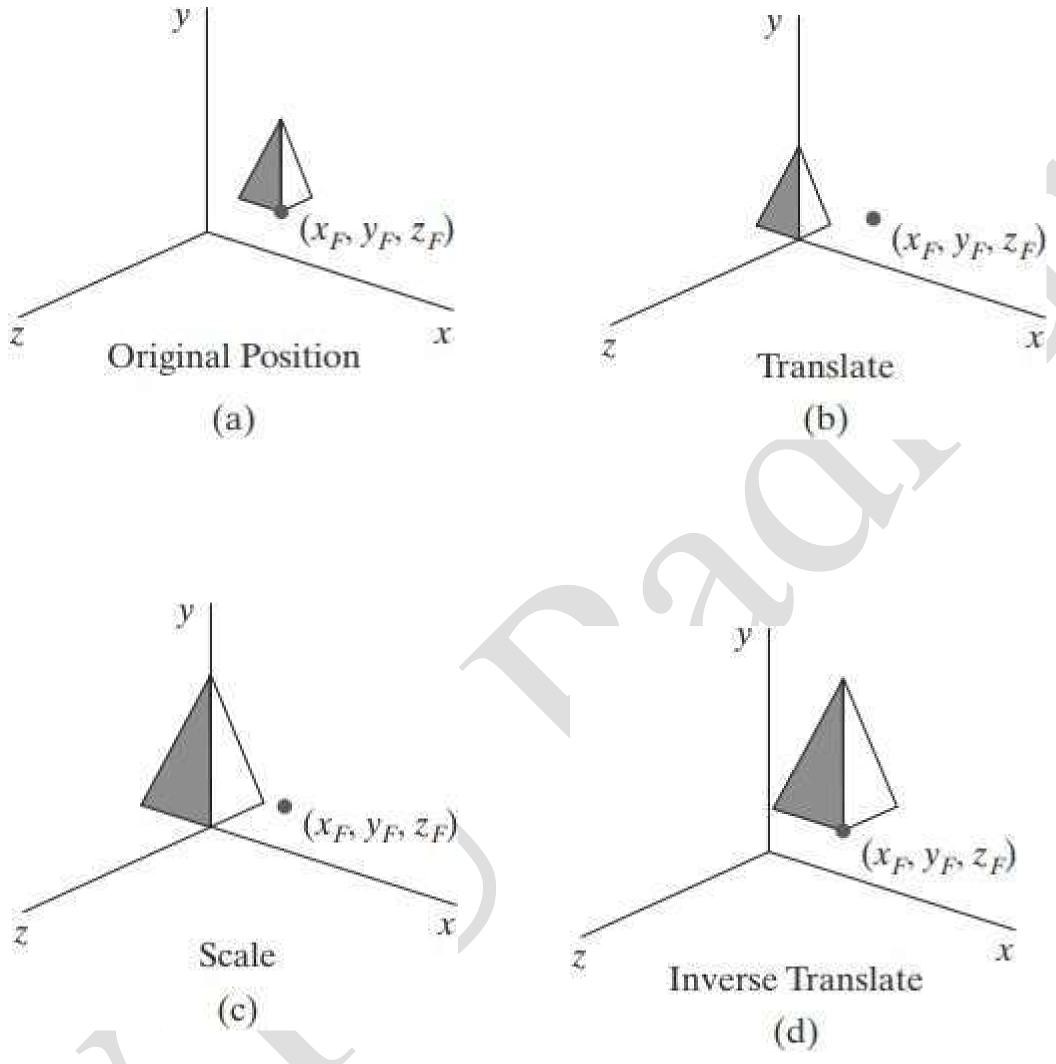
Doubling the size of an object with transformation 54 also moves the objectfarther from the origin. Scaling parameter is set to 2.

Scaling transformation with respect to any selected fixed position (x_f,y_f,z_f) can be constructed using the following transformation sequence:

1. Translate the fixed point to the origin

2. Apply the scaling transformation relative to the coordinate origin

3. Translate the fixed point back to its original position.



A sequence of transformations for scaling an object relative to a selected fixedpoint

The matrix representation for an arbitrary fixed-point scaling can be expressed as the concatenation of translate-scale-translate transformations:

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

Composite Three-Dimensional Transformations

Composite three dimensional transformation can be formed by multiplying the matrix representations for the individual operations in the transformation sequence. Transformation sequence can be implemented by concatenating the individual matrices from right to left or from left to right, depending on the order in which the matrix representations are specified. Rightmost term in a matrix product is always the first transformation to be applied to an object. Leftmost term is always the last transformation. Coordinate positions are represented as four-element column vectors which are premultiplied by the composite 4 X 4 transformation matrix.

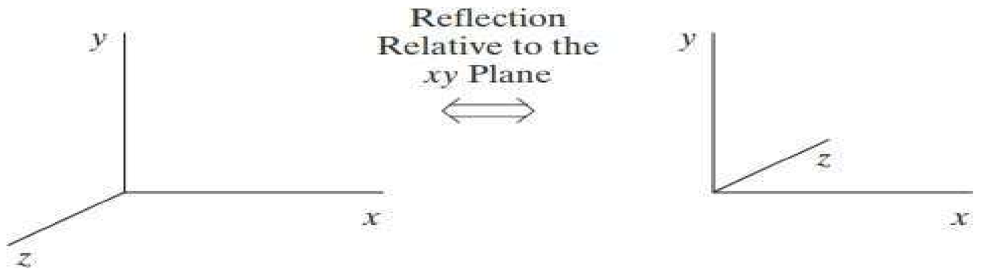
Other Three-Dimensional Transformations

Three-Dimensional Reflections

A reflection in a three-dimensional space can be performed relative to a selected reflection axis or with respect to a reflection plane. In general, three dimensional-reflection matrices are set up similarly to those for two dimensions. Reflection relative to a given axis is equivalent to 180^0 rotations about that axis.

When the reflection plane is a coordinate plane (xy, xz, or yz), transformation can be thought as a 180◦ rotation in four dimensional space with a conversion between a left-handed frame and a right-handed frame.

An example of a reflection that converts coordinate specifications from a right handed system to a left-handed system is shown in Figure below.



Conversion of coordinate specifications between a right-handed and a left-handed system can be carried out with the reflection transformation 55

The matrix representation for this reflection relative to the xy plane is

$$M_{\text{zreflect}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \textbf{(55)}$$

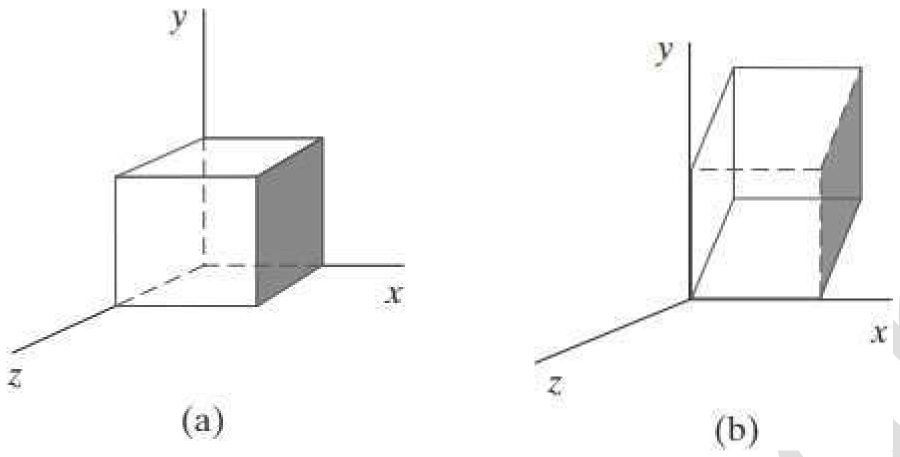
In this transformation, sign of z coordinates changes, but the values of x and y coordinates remains unchanged.

Three-Dimensional Shears

These transformations can be used to modify object shapes. For three-dimensional, shears can be generated relative to the z axis. A general z-axis shearing transformation relative to a selected reference position is produced with the following matrix:

$$M_{\text{zshear}} = \begin{bmatrix} 1 & 0 & \text{sh}_{zx} & -\text{sh}_{zx} \cdot z_{\text{ref}} \\ 0 & 1 & \text{sh}_{zy} & -\text{sh}_{zy} \cdot z_{\text{ref}} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \textbf{(56)}$$

Shearing parameter sh_{zx} and sh_{zy} can be assigned any real value. Transformation matrix alters the values for the x and y coordinates by an amount that is proportional to distance from z_{ref} . The z coordinate value remain unchanged. Plane areas that are perpendicular to the z axis are shifted by an amount equal to $z-z_{ref}$.



A unit cube (a) is sheared relative to the origin (b) by Matrix 56, with $sh_{zx} = sh_{zy} = 1$. Reference position $z_{ref} = 0$

OpenGL Geometric-Transformation Functions

OpenGL Matrix Stacks

glMatrixMode specify which matrix is the current matrix.

There are four modes:

1. Modelview
2. Projection
3. Texture
4. Color

For each mode, OpenGL maintains a matrix stack. Initially, each stack contains only the identity matrix. At any time during the processing of a scene, the top matrix on each stack is called the "current matrix" for that mode. After we specify the viewing and geometric transformations, the top of the **modelview matrix stack** is the 4 × 4 composite matrix that combines the viewing transformations and the various geometric transformations that we want to apply to a scene.

OpenGL supports a modelview stack depth of at least 32.

glGetIntegerv(GL_MAX_MODELVIEW_STACK_DEPTH, stackSize);

The above function determine the number of positions available in the modelview stack for a particular implementation of OpenGL. It returns a single integer value to array **stackSize**

We can also find out how many matrices are currently in the stack with

glGetIntegerv (GL_MODELVIEW_STACK_DEPTH, numMats);

Other OpenGL symbolic constants are

1. GL_MAX_PROJECTION_STACK_DEPTH
2. GL_MAX_TEXTURE_STACK_DEPTH
3. GL_MAX_COLOR_STACK_DEPTH

There are two functions available in OpenGL for processing the matrices in a stack

glPushMatrix( );

Copy the current matrix at the top of the active stack and store that copy in the second stack position.

glPopMatrix( );

Destroys the matrix at the top of the stack, and the second matrix in the stack becomes the current matrix.