Computer Vision Handout 1

2D Transformations

With the procedures for displaying output primitives and their attributes, we can create a variety of pictures and graphs. In many applications, there is also a need for altering or manipulating displays. Design applications and facility layouts are created by arranging the orientations and sizes of the component parts of the scene. And animations are produced by moving the "camera" or the objects in a scene along animation paths. Changes in orientation, size, and shape are accomplished with **geometric transformations** that alter the coordinate descriptions of objects. The basic geometric transformations are translation, rotation, and scaling. Other transformations that are often applied to objects include reflection and shear. We first discuss methods for performing geometric transformations and then consider how transformation functions can be incorporated into graphics packages.

5-1 BASIC TRANSFORMATIONS

Here, we first discuss general procedures for applying translation, rotation, and scaling parameters to reposition and resize two-dimensional objects. Then, in Section 5-2, we consider how transformation equations can be expressed in a more convenient matrix formulation that allows efficient combination of object transformations.

Translation

A **translation** is applied to an object by repositioning it along a straight-line path from one coordinate location to another. We translate a two-dimensional point by adding **translation distances**, t_x and t_y, to the original coordinate position (x, y) to move the point to a new position (x', y') (Fig. 5-1).

$$x' = x + t_x, \qquad y' = y + t_y \tag{5-1}$$

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

We can express the translation equations 5-1 as a single matrix equation by using column vectors to represent coordinate positions and the translation vector:

$$\mathbf{P} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \qquad \mathbf{P}' = \begin{bmatrix} x_1' \\ x_2' \end{bmatrix}, \qquad \mathbf{T} = \begin{bmatrix} t_x \\ t_y \end{bmatrix} \tag{5-2}$$

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

$$\mathbf{P}' = \mathbf{P} + \mathbf{T} \tag{5-3}$$

Sometimes matrix-transformation equations are expressed in terms of coordinate row vectors instead of column vectors. In this case, we would write the matrix representations as $\mathbf{P} = [x\ y]$ and $\mathbf{T} = [t_x\ t_y]$. Since the column-vector representation for a point is standard mathematical notation, and since many graphics packages, for example, GKS and PHIGS, also use the column-vector representation, we will follow this convention.

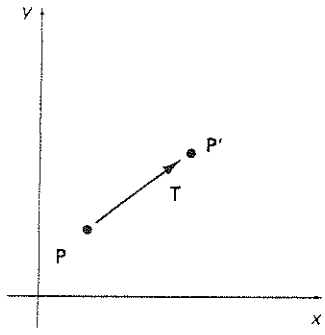


Figure 5-1
Translating a point from position P to position P' with translation vector T.

Translation is a *rigid-body transformation* that moves objects without deformation. That is, every point on the object is translated by the same amount. A straight line segment is translated by applying the transformation equation 5-3 to each of the line endpoints and redrawing the line between the new endpoint positions. Polygons are translated by adding the translation vector to the coordinate position of each vertex and regenerating the polygon using the new set of vertex coordinates and the current attribute settings. Figure 5-2 illustrates the application of a specified translation vector to move an object from one position to another.

Similar methods are used to translate curved objects. To change the position of a circle or ellipse, we translate the center coordinates and redraw the figure in the new location. We translate other curves (for example, splines) by displacing the coordinate positions defining the objects, then we reconstruct the curve paths using the translated coordinate points.

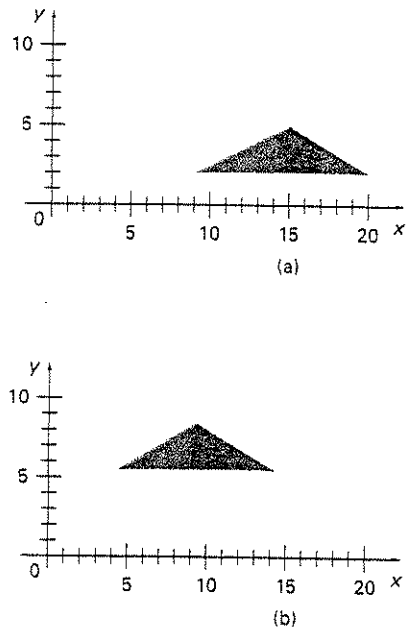


Figure 5-2
Moving a polygon from position (a) to position (b) with the translation vector (-5.50, 3.75).

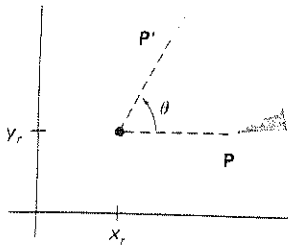


Figure 5-3
Rotation of an object through angle θ about the pivot point (x_r, y_r).

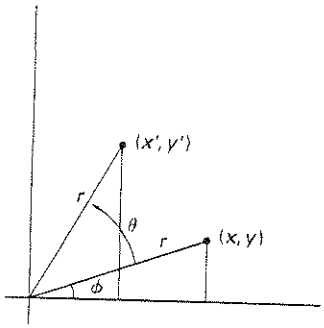


Figure 5-4
Rotation of a point from position (x, y) to position (x', y') through an angle θ relative to the coordinate origin. The original angular displacement of the point from the x axis is ϕ.

Rotation

A two-dimensional **rotation** is applied to an object by repositioning it along a circular path in the xy plane. To generate a rotation, we specify a **rotation angle** θ and the position (x_r, y_r) of the **rotation point** (or **pivot point**) about which the object is to be rotated (Fig. 5-3). Positive values for the rotation angle define counterclockwise rotations about the pivot point, as in Fig. 5-3, and negative values rotate objects in the clockwise direction. This transformation can also be described as a rotation about a **rotation axis** that is perpendicular to the xy plane and passes through the pivot point.

We first determine the transformation equations for rotation of a point position $\mathbf{P}$ when the pivot point is at the coordinate origin. The angular and coordinate relationships of the original and transformed point positions are shown in Fig. 5-4. In this figure, r is the constant distance of the point from the origin, angle ϕ is the original angular position of the point from the horizontal, and θ is the rotation angle. Using standard trigonometric identities, we can express the transformed coordinates in terms of angles θ and ϕ as

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

The original coordinates of the point in polar coordinates are

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

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

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

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

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

where the rotation matrix is

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

When coordinate positions are represented as row vectors instead of column vectors, the matrix product in rotation equation 5-7 is transposed so that the transformed row coordinate vector $[x'\ y']$ is calculated as

$$\begin{aligned} \mathbf{P}'^T &= (\mathbf{R} \cdot \mathbf{P})^T \\ &= \mathbf{P}^T \cdot \mathbf{R}^T \end{aligned}$$

where $\mathbf{P}^T = [x\ y]$, and the transpose $\mathbf{R}^T$ of matrix $\mathbf{R}$ is obtained by interchanging rows and columns. For a rotation matrix, the transpose is obtained by simply changing the sign of the sine terms.

Rotation of a point about an arbitrary pivot position is illustrated in Fig. 5-5. Using the trigonometric relationships in this figure, we can generalize Eqs. 5-6 to obtain the transformation equations for rotation of a point about any specified rotation position (x_r, y_r):

$$x' = x_r + (x - x_r)\cos\theta - (y - y_r)\sin\theta$$
$$y' = y_r + (x - x_r)\sin\theta + (y - y_r)\cos\theta \qquad (5\text{-}9)$$

These general rotation equations differ from Eqs. 5-6 by the inclusion of additive terms, as well as the multiplicative factors on the coordinate values. Thus, the matrix expression 5-7 could be modified to include pivot coordinates by matrix addition of a column vector whose elements contain the additive (translational) terms in Eqs. 5-9. There are better ways, however, to formulate such matrix equations, and we discuss in Section 5-2 a more consistent scheme for representing the transformation equations.

As with translations, rotations are rigid-body transformations that move objects without deformation. Every point on an object is rotated through the same angle. A straight line segment is rotated by applying the rotation equations 5-9 to each of the line endpoints and redrawing the line between the new endpoint positions. Polygons are rotated by displacing each vertex through the specified rotation angle and regenerating the polygon using the new vertices. Curved lines are rotated by repositioning the defining points and redrawing the curves. A circle or an ellipse, for instance, can be rotated about a noncentral axis by moving the center position through the arc that subtends the specified rotation angle. An ellipse can be rotated about its center coordinates by rotating the major and minor axes.

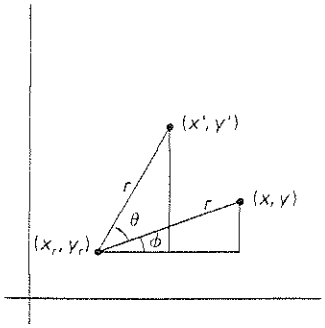


Figure 5-5
Rotating a point from position (x, y) to position (x', y') through an angle θ about rotation point (x_r, y_r).

Scaling

A **scaling** transformation alters the size of an object. This operation can be carried out for polygons by multiplying the coordinate values (x, y) of each vertex by **scaling factors** s_x and s_y to produce the transformed coordinates (x', y'):

$$x' = x \cdot s_x, \qquad y' = y \cdot s_y \qquad (5\text{-}10)$$

Scaling factor s_x scales objects in the x direction, while s_y scales in the y direction. The transformation equations 5-10 can also be written in the matrix form:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} \qquad (5\text{-}11)$$

or

$$\mathbf{P}' = \mathbf{S} \cdot \mathbf{P} \qquad (5\text{-}12)$$

where $\mathbf{S}$ is the 2 by 2 scaling matrix in Eq. 5-11.

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 pro-

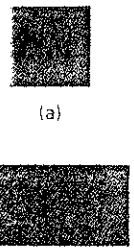


Figure 5-6
Turning a square (a) into a rectangle (b) with scaling factors $s_x = 2$ and $s_y = 1$.

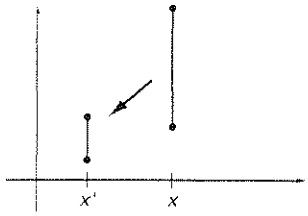


Figure 5-7
A line scaled with Eq. 5-12 using $s_x = s_y = 0.5$ is reduced in size and moved closer to the coordinate origin.

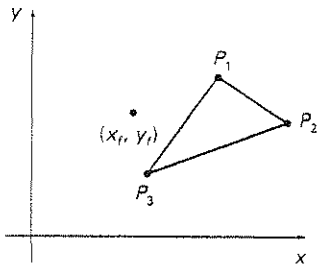


Figure 5-8
Scaling relative to a chosen fixed point (x_f, y_f). Distances from each polygon vertex to the fixed point are scaled by transformation equations 5-13.

duced 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. 5-6).

Objects transformed with Eq. 5-11 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.

We can control the location of a scaled object 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) can be chosen as one of the vertices, the object centroid, or any other position (Fig. 5-8). A polygon is then scaled relative to the fixed point by scaling the distance from each vertex to the fixed point. For a vertex with coordinates (x, y), the scaled coordinates (x', y') are calculated as

$$x' = x_f + (x - x_f)s_x, \qquad y' = y_f + (y - y_f)s_y \tag{5-13}$$

We can rewrite these scaling transformations to separate the multiplicative and additive terms:

$$\begin{aligned} x' &= x \cdot s_x + x_f(1 - s_x) \\ y' &= y \cdot s_y + y_f(1 - s_y) \end{aligned} \tag{5-14}$$

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

Including coordinates for a fixed point in the scaling equations is similar to including coordinates for a pivot point in the rotation equations. We can set up a column vector whose elements are the constant terms in Eqs. 5-14, then we add this column vector to the product $\mathbf{S} \cdot \mathbf{P}$ in Eq. 5-12. In the next section, we discuss a matrix formulation for the transformation equations that involves only matrix multiplication.

Polygons are scaled by applying transformations 5-14 to each vertex and then regenerating the polygon using the transformed vertices. Other objects are scaled by applying the scaling transformation equations to the parameters defining the objects. An ellipse in standard position is resized by scaling the semimajor and semiminor axes and redrawing the ellipse about the designated center coordinates. Uniform scaling of a circle is done by simply adjusting the radius. Then we redisplay the circle about the center coordinates using the transformed radius.

5-2
MATRIX REPRESENTATIONS AND HOMOGENEOUS COORDINATES

Many graphics applications involve sequences of geometric transformations. An animation, for example, might require an object to be translated and rotated at each increment of the motion. In design and picture construction applications,

we perform translations, rotations, and scalings to fit the picture components into their proper positions. Here we consider how the matrix representations discussed in the previous sections can be reformulated so that such transformation sequences can be efficiently processed.

We have seen in Section 5-1 that each of the basic transformations can be expressed in the general matrix form

$$\mathbf{P}' = \mathbf{M}_1 \cdot \mathbf{P} + \mathbf{M}_2 \tag{5-15}$$

with coordinate positions $\mathbf{P}$ and $\mathbf{P}'$ represented as column vectors. Matrix $\mathbf{M}_1$ is a 2 by 2 array containing multiplicative factors, and $\mathbf{M}_2$ is a two-element column matrix containing translational terms. For translation, $\mathbf{M}_1$ is the identity matrix. For rotation or scaling, $\mathbf{M}_2$ contains the translational terms associated with the pivot point or scaling fixed point. To produce a sequence of transformations with these equations, such as scaling followed by rotation then translation, we must calculate the transformed coordinates one step at a time. First, coordinate positions are scaled, then these scaled coordinates are rotated, and finally the rotated coordinates are translated. A more efficient approach would be to combine the transformations so that the final coordinate positions are obtained directly from the initial coordinates, thereby eliminating the calculation of intermediate coordinate values. To be able to do this, we need to reformulate Eq. 5-15 to eliminate the matrix addition associated with the translation terms in $\mathbf{M}_2$.

We can combine the multiplicative and translational terms for two-dimensional geometric transformations into a single matrix representation by expanding the 2 by 2 matrix representations to 3 by 3 matrices. This allows us to express all transformation equations as matrix multiplications, providing that we also expand the matrix representations for coordinate positions. To express any two-dimensional transformation as a matrix multiplication, we represent each Cartesian coordinate position (x, y) with the **homogeneous coordinate** triple (x_h, y_h, h), where

$$x = \frac{x_h}{h}, \qquad y = \frac{y_h}{h} \tag{5-16}$$

Thus, a general homogeneous coordinate representation can also be written as $(h \cdot x, h \cdot y, h)$. For two-dimensional geometric transformations, we can choose the homogeneous parameter h to be any nonzero value. Thus, there is an infinite number of equivalent homogeneous representations for each coordinate point (x, y). A convenient choice is simply to set $h = 1$. Each two-dimensional position is then represented with homogeneous coordinates $(x, y, 1)$. Other values for parameter h are needed, for example, in matrix formulations of three-dimensional viewing transformations.

The term *homogeneous coordinates* is used in mathematics to refer to the effect of this representation on Cartesian equations. When a Cartesian point (x, y) is converted to a homogeneous representation (x_h, y_h, h), equations containing x and y, such as $f(x, y) = 0$, become homogeneous equations in the three parameters x_h, y_h, and h. This just means that if each of the three parameters is replaced by any value v times that parameter, the value v can be factored out of the equations.

Expressing positions in homogeneous coordinates allows us to represent all geometric transformation equations as matrix multiplications. Coordinates are

represented with three-element column vectors, and transformation operations are written as 3 by 3 matrices. For translation, we have

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{5-17}$$

which we can write in the abbreviated form

$$\mathbf{P}' = \mathbf{T}(t_x, t_y) \cdot \mathbf{P} \tag{5-18}$$

with $\mathbf{T}(t_x, t_y)$ as the 3 by 3 translation matrix in Eq. 5-17. The inverse of the translation matrix is obtained by replacing the translation parameters t_x and t_y with their negatives: $-t_x$ and $-t_y$.

Similarly, rotation transformation equations about the coordinate origin are now written as

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{5-19}$$

or as

$$\mathbf{P}' = \mathbf{R}(\theta) \cdot \mathbf{P} \tag{5-20}$$

The rotation transformation operator $\mathbf{R}(\theta)$ is the 3 by 3 matrix in Eq. 5-19 with rotation parameter θ. We get the inverse rotation matrix when θ is replaced with $-\theta$.

Finally, a scaling transformation relative to the coordinate origin is now expressed as the matrix multiplication

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{5-21}$$

or

$$\mathbf{P}' = \mathbf{S}(s_x, s_y) \cdot \mathbf{P} \tag{5-22}$$

where $\mathbf{S}(s_x, s_y)$ is the 3 by 3 matrix in Eq. 5-21 with parameters s_x and s_y. Replacing these parameters with their multiplicative inverses ($1/s_x$ and $1/s_y$) yields the inverse scaling matrix.

Matrix representations are standard methods for implementing transformations in graphics systems. In many systems, rotation and scaling functions produce transformations with respect to the coordinate origin, as in Eqs. 5-19 and 5-21. Rotations and scalings relative to other reference positions are then handled as a succession of transformation operations. An alternate approach in a graphics package is to provide parameters in the transformation functions for the scaling fixed-point coordinates and the pivot-point coordinates. General rotation and scaling matrices that include the pivot or fixed point are then set up directly without the need to invoke a succession of transformation functions.

5-3 COMPOSITE TRANSFORMATIONS

With the matrix representations of the previous section, we can set up a matrix for any sequence of transformations as a **composite transformation matrix** by calculating the matrix product of the individual transformations. Forming products of transformation matrices is often referred to as a **concatenation**, or **composition**, of matrices. For column-matrix representation of coordinate positions, we form composite transformations by multiplying matrices in order from right to left. That is, each successive transformation matrix premultiplies the product of the preceding transformation matrices.

Translations

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

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

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

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

or

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

which demonstrates that two successive translations are additive.

Rotations

Two successive rotations applied to point $\mathbf{P}$ produce the transformed position

$$\begin{aligned}\mathbf{P}' &= \mathbf{R}(\theta_2) \cdot \{\mathbf{R}(\theta_1) \cdot \mathbf{P}\} \\ &= \{\mathbf{R}(\theta_2) \cdot \mathbf{R}(\theta_1)\} \cdot \mathbf{P}\end{aligned} \tag{5-26}$$

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

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

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

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

Scalings

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

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

or

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

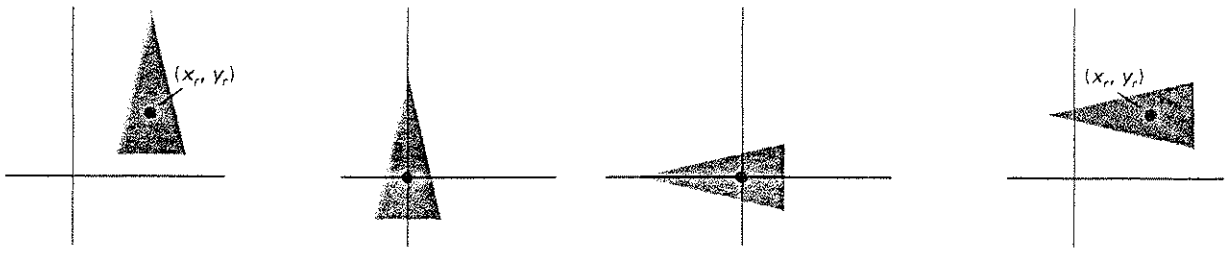
The resulting matrix in this case indicates that successive scaling operations are multiplicative. That is, if we were to triple the size of an object twice in succession, the final size would be nine times that of the original.

General Pivot-Point Rotation

With a graphics package that only provides a rotate function for revolving objects about the coordinate origin, we can generate rotations about any selected pivot point (x_r, y_r) by performing the following sequence of translate–rotate–translate operations:

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

This transformation sequence is illustrated in Fig. 5-9. The composite transforma-



(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)

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

tion matrix for this sequence is obtained with the concatenation

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

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

which can be expressed in the form

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

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

General Fixed-Point Scaling

Figure 5-10 illustrates a transformation sequence to produce scaling with respect to a selected fixed position (x_f, y_f) using a scaling function that can only scale relative to the coordinate origin.

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

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

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

or

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

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

General Scaling Directions

Parameters s_x and s_y scale objects along the x and y directions. We can scale an object in other directions by rotating the object to align the desired scaling directions with the coordinate axes before applying the scaling transformation.

Suppose we want to apply scaling factors with values specified by parameters s_1 and s_2 in the directions shown in Fig. 5-11. To accomplish the scaling with-

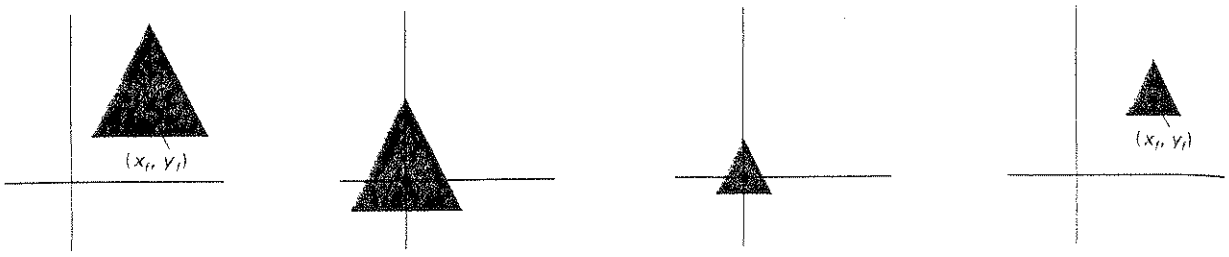


Figure 5-10
A transformation sequence for scaling an object with respect to a specified fixed position using the scaling matrix $S(s_x, s_y)$ of transformation 5-21.

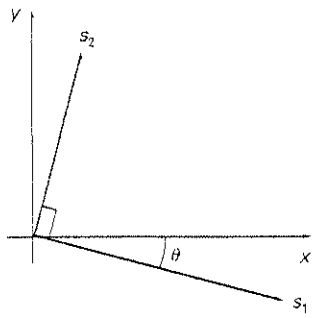


Figure 5-11
Scaling parameters s_1 and s_2 are to be applied in orthogonal directions defined by the angular displacement θ.

out changing the orientation of the object, we first perform a rotation so that the directions for s_1 and s_2 coincide with the x and y axes, respectively. Then the scaling transformation is applied, followed by an opposite rotation to return points to their original orientations. The composite matrix resulting from the product of these three transformations is

$$\mathbf{R}^{-1}(\theta) \cdot \mathbf{S}(s_1, s_2) \cdot \mathbf{R}(\theta) = \begin{bmatrix} s_1 \cos^2\theta + s_2 \sin^2\theta & (s_2 - s_1)\cos\theta \sin\theta & 0 \\ (s_2 - s_1)\cos\theta \sin\theta & s_1 \sin^2\theta + s_2 \cos^2\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5-35}$$

As an example of this scaling transformation, we turn a unit square into a parallelogram (Fig. 5-12) by stretching it along the diagonal from (0, 0) to (1, 1). We rotate the diagonal onto the y axis and double its length with the transformation parameters $\theta = 45°$, $s_1 = 1$, and $s_2 = 2$.

In Eq. 5-35, we assumed that scaling was to be performed relative to the origin. We could take this scaling operation one step further and concatenate the matrix with translation operators, so that the composite matrix would include parameters for the specification of a scaling fixed position.

Concatenation Properties

Matrix multiplication is associative. For any three matrices, **A**, **B**, and **C**, the matrix product $\mathbf{A} \cdot \mathbf{B} \cdot \mathbf{C}$ can be performed by first multiplying **A** and **B** or by first multiplying **B** and **C**:

$$\mathbf{A} \cdot \mathbf{B} \cdot \mathbf{C} = (\mathbf{A} \cdot \mathbf{B}) \cdot \mathbf{C} = \mathbf{A} \cdot (\mathbf{B} \cdot \mathbf{C}) \tag{5-36}$$

Therefore, we can evaluate matrix products using either a left-to-right or a right-to-left associative grouping.

On the other hand, transformation products may not be commutative: The matrix product $\mathbf{A} \cdot \mathbf{B}$ is not equal to $\mathbf{B} \cdot \mathbf{A}$, in general. This means that if we want

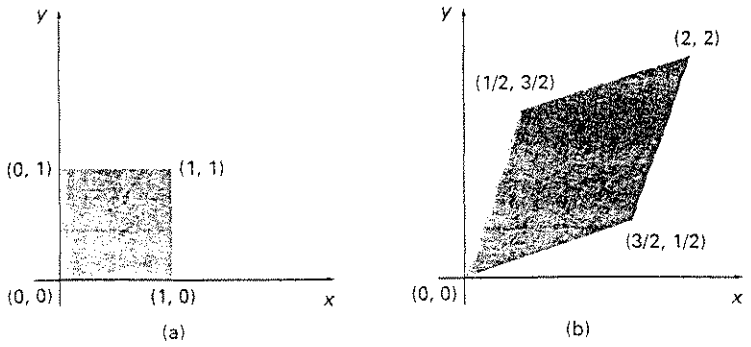


Figure 5-12
A square (a) is converted to a parallelogram (b) using the composite transformation matrix 5-35, with $s_1 = 1$, $s_2 = 2$, and $\theta = 45°$.

to translate and rotate an object, we must be careful about the order in which the composite matrix is evaluated (Fig. 5-13). For some special cases, such as a sequence of transformations all of the same kind, the multiplication of transformation matrices is commutative. As an example, two successive rotations could be performed in either order and the final position would be the same. This commutative property holds also for two successive translations or two successive scalings. Another commutative pair of operations is rotation and uniform scaling ($s_x = s_y$).

General Composite Transformations and Computational Efficiency

A general two-dimensional transformation, representing a combination of translations, rotations, and scalings, can be expressed as

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} rs_{xx} & rs_{xy} & trs_x \\ rs_{yx} & rs_{yy} & trs_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{5-37}$$

The four elements rs_{ij} are the multiplicative rotation-scaling terms in the transformation that involve only rotation angles and scaling factors. Elements trs_x and trs_y are the translational terms containing combinations of translation distances, pivot-point and fixed-point coordinates, and rotation angles and scaling parameters. For example, if an object is to be scaled and rotated about its centroid coordinates (x_c, y_c) and then translated, the values for the elements of the composite transformation matrix are

$$\begin{aligned} &\mathbf{T}(t_x, t_y) \cdot \mathbf{R}(x_c, y_c, \theta) \cdot \mathbf{S}(x_c, y_c, s_x, s_y) \\ &\quad = \begin{bmatrix} s_x \cos\theta & -s_y \sin\theta & x_c(1 - s_x \cos\theta) + y_c s_y \sin\theta + t_x \\ s_x \sin\theta & s_y \cos\theta & y_c(1 - s_y \cos\theta) - x_c s_x \sin\theta + t_y \\ 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{5-38}$$

Although matrix equation 5-37 requires nine multiplications and six additions, the explicit calculations for the transformed coordinates are

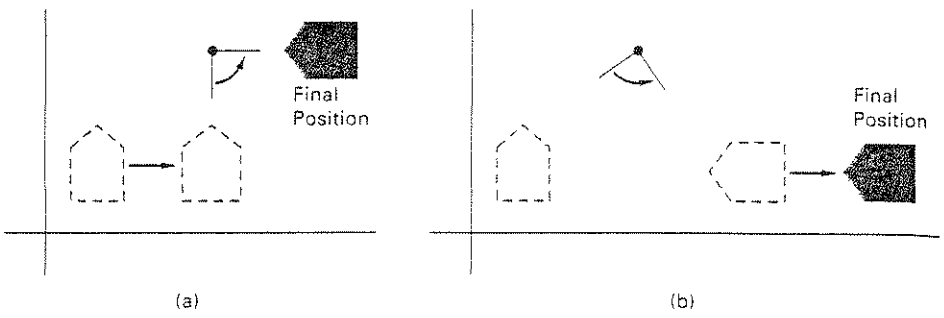


Figure 5-13
Reversing the order in which a sequence of transformations is performed may affect the transformed position of an object. In (a), an object is first translated, then rotated. In (b), the object is rotated first, then translated.

$$x' = x \cdot rs_{xx} + y \cdot rs_{xy} + trs_x, \qquad y' = x \cdot rs_{yx} + y \cdot rs_{yy} + trs_y \tag{5-39}$$

Thus, we actually only need to perform four multiplications and four additions to transform coordinate positions. This is the maximum number of computations required for any transformation sequence, once the individual matrices have been concatenated and the elements of the composite matrix evaluated. Without concatenation, the individual transformations would be applied one at a time and the number of calculations could be significantly increased. An efficient implementation for the transformation operations, therefore, is to formulate transformation matrices, concatenate any transformation sequence, and calculate transformed coordinates using Eq. 5-39. On parallel systems, direct matrix multiplications with the composite transformation matrix of Eq. 5-37 can be equally efficient.

A general **rigid-body transformation matrix**, involving only translations and rotations, can be expressed in the form

$$\begin{bmatrix} r_{xx} & r_{xy} & tr_x \\ r_{yx} & r_{yy} & tr_y \\ 0 & 0 & 1 \end{bmatrix} \tag{5-40}$$

where the four elements r_{ij} are the multiplicative rotation terms, and elements tr_x and tr_y are the translational terms. A rigid-body change in coordinate position is also sometimes referred to as a **rigid-motion** transformation. All angles and distances between coordinate positions are unchanged by the transformation. In addition, matrix 5-40 has the property that its upper-left 2-by-2 submatrix is an orthogonal matrix. This means that if we consider each row of the submatrix as a vector, then the two vectors (r_{xx}, r_{xy}) and (r_{yx}, r_{yy}) form an orthogonal set of unit vectors: Each vector has unit length

$$r_{xx}^2 + r_{xy}^2 = r_{yx}^2 + r_{yy}^2 = 1 \tag{5-41}$$

and the vectors are perpendicular (their dot product is 0):

$$r_{xx}r_{yx} + r_{xy}r_{yy} = 0 \tag{5-42}$$

Therefore, if these unit vectors are transformed by the rotation submatrix, (r_{xx}, r_{xy}) is converted to a unit vector along the x axis and (r_{yx}, r_{yy}) is transformed into a unit vector along the y axis of the coordinate system:

$$\begin{bmatrix} r_{xx} & r_{xy} & 0 \\ r_{yx} & r_{yy} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{xx} \\ r_{xy} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \tag{5-43}$$

$$\begin{bmatrix} r_{xx} & r_{xy} & 0 \\ r_{yx} & r_{yy} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{yx} \\ r_{yy} \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} \tag{5-44}$$

As an example, the following rigid-body transformation first rotates an object through an angle θ about a pivot point (x_r, y_r) and then translates:

$$\begin{aligned} &\mathrm{T}(t_x, t_y) \cdot \mathrm{R}(x_r, y_r, \theta) \\ &\quad = \begin{bmatrix} \cos\theta & -\sin\theta & x_r(1-\cos\theta) + y_r \sin\theta + t_x \\ \sin\theta & \cos\theta & y_r(1-\cos\theta) - x_r \sin\theta + t_y \\ 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{5-45}$$

Here, orthogonal unit vectors in the upper-left 2-by-2 submatrix are $(\cos\theta, -\sin\theta)$ and $(\sin\theta, \cos\theta)$, and

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\theta \\ -\sin\theta \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \tag{5-46}$$

Similarly, unit vector $(\sin\theta, \cos\theta)$ is converted by the transformation matrix in Eq. 5-46 to the unit vector $(0, 1)$ in the y direction.

The orthogonal property of rotation matrices is useful for constructing a rotation matrix when we know the final orientation of an object rather than the amount of angular rotation necessary to put the object into that position. Directions for the desired orientation of an object could be determined by the alignment of certain objects in a scene or by selected positions in the scene. Figure 5-14 shows an object that is to be aligned with the unit direction vectors $\mathbf{u}'$ and $\mathbf{v}'$. Assuming that the original object orientation, as shown in Fig. 5-14(a), is aligned with the coordinate axes, we construct the desired transformation by assigning the elements of $\mathbf{u}'$ to the first row of the rotation matrix and the elements of $\mathbf{v}'$ to the second row. This can be a convenient method for obtaining the transformation matrix for rotation within a local (or "object") coordinate system when we know the final orientation vectors. A similar transformation is the conversion of object descriptions from one coordinate system to another, and in Section 5-5, we consider how to set up transformations to accomplish this coordinate conversion.

Since rotation calculations require trignometric evaluations and several multiplications for each transformed point, computational efficiency can become an important consideration in rotation transformations. In animations and other applications that involve many repeated transformations and small rotation angles, we can use approximations and iterative calculations to reduce computa-

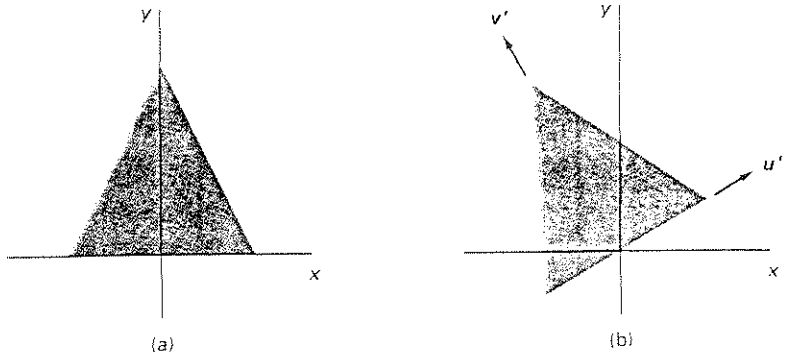


Figure 5-14
The rotation matrix for revolving an object from position (a) to position (b) can be constructed with the values of the unit orientation vectors **u'** and **v'** relative to the original orientation.

tions in the composite transformation equations. When the rotation angle is small, the trigonometric functions can be replaced with approximation values based on the first few terms of their power-series expansions. For small enough angles (less than 10°), cos θ is approximately 1 and sin θ has a value very close to the value of θ in radians. If we are rotating in small angular steps about the origin, for instance, we can set cos θ to 1 and reduce transformation calculations at each step to two multiplications and two additions for each set of coordinates to be rotated:

$$x' = x - y \sin\theta, \qquad y' = x \sin\theta + y \tag{5-47}$$

where sin θ is evaluated once for all steps, assuming the rotation angle does not change. The error introduced by this approximation at each step decreases as the rotation angle decreases. But even with small rotation angles, the accumulated error over many steps can become quite large. We can control the accumulated error by estimating the error in x' and y' at each step and resetting object positions when the error accumulation becomes too great.

Composite transformations often involve inverse matrix calculations. Transformation sequences for general scaling directions and for reflections and shears (Section 5-4), for example, can be described with inverse rotation components. As we have noted, the inverse matrix representations for the basic geometric transformations can be generated with simple procedures. An inverse translation matrix is obtained by changing the signs of the translation distances, and an inverse rotation matrix is obtained by performing a matrix transpose (or changing the sign of the sine terms). These operations are much simpler than direct inverse matrix calculations.

An implementation of composite transformations is given in the following procedure. Matrix **M** is initialized to the identity matrix. As each individual transformation is specified, it is concatenated with the total transformation matrix **M**. When all transformations have been specified, this composite transformation is applied to a given object. For this example, a polygon is scaled and rotated about a given reference point. Then the object is translated. Figure 5-15 shows the original and final positions of the polygon transformed by this sequence.

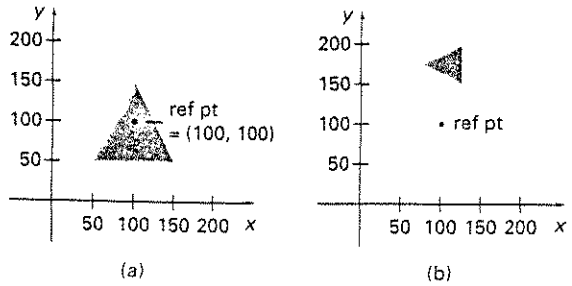


Figure 5-15
A polygon (a) is transformed into (b) by the composite operations in the following procedure.

```
#include <math.h>
#include "graphics.h"

typedef float Matrix3x3[3][3];
Matrix3x3 theMatrix;

void matrix3x3SetIdentity (Matrix3x3 m)
{
  int i,j;

  for (i=0; i<3; i++) for (j=0; j<3; j++) m[i][j] = (i == j);
}

/* Multiplies matrix a times b, putting result in b */
void matrix3x3PreMultiply (Matrix3x3 a, Matrix3x3 b)
{
  int r,c;
  Matrix3x3 tmp;

  for (r = 0; r < 3; r++)
    for (c = 0; c < 3; c++)
      tmp[r][c] =
        a[r][0]*b[0][c] + a[r][1]*b[1][c] + a[r][2]*b[2][c];

  for (r = 0; r < 3; r++)
    for (c = 0; c < 3; c++)
      b[r][c] = tmp[r][c];
}

void translate2 (int tx, int ty)
{
  Matrix3x3 m;

  matrix3x3SetIdentity (m);
  m[0][2] = tx;
  m[1][2] = ty;
  matrix3x3PreMultiply (m, theMatrix);
```

Figure 5-22 shows the original and final positions for an object transformed with this reflection matrix.

Reflections about any line $y = mx + b$ in the xy plane can be accomplished with a combination of translate-rotate-reflect transformations. In general, we first translate the line so that it passes through the origin. Then we can rotate the line onto one of the coordinate axes and reflect about that axis. Finally, we restore the line to its original position with the inverse rotation and translation transformations.

We can implement reflections with respect to the coordinate axes or coordinate origin as scaling transformations with negative scaling factors. Also, elements of the reflection matrix can be set to values other than ± 1. Values whose magnitudes are greater than 1 shift the mirror image farther from the reflection axis, and values with magnitudes less than 1 bring the mirror image closer to the reflection axis.

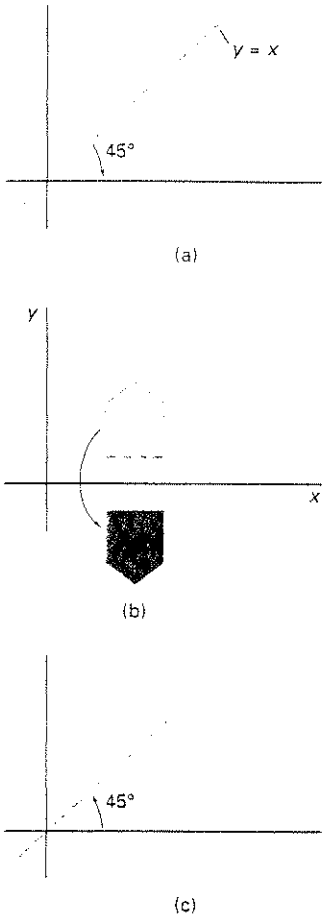


Figure 5-21
Sequence of transformations to produce reflection about the line $y = x$: (a) clockwise rotation of 45°, (b) reflection about the x axis, and (c) counterclockwise rotation by 45°.

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

$$\begin{bmatrix} 1 & sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5-53}$$

which transforms coordinate positions as

$$x' = x + sh_x \cdot y, \qquad y' = y \tag{5-54}$$

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 distance (y value) from the x axis ($y = 0$). Setting sh_x to 2, for example, changes the square in Fig. 5-23 into a parallelogram. Negative values for sh_x shift coordinate positions to the left.

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

$$\begin{bmatrix} 1 & sh_x & -sh_x \cdot y_{ref} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5-55}$$

with coordinate positions transformed as

$$x' = x + sh_x(y - y_{ref}), \qquad y' = y \tag{5-56}$$

An example of this shearing transformation is given in Fig. 5-24 for a shear parameter value of $1/2$ relative to the line $y_{ref} = -1$.

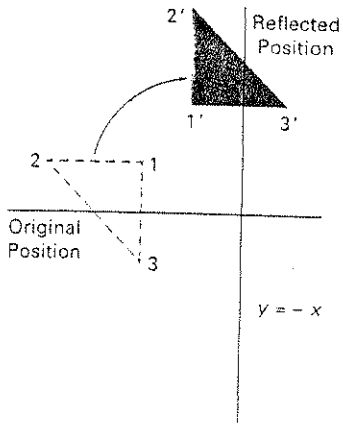


Figure 5-22
Reflection with respect to the line $y = -x$.

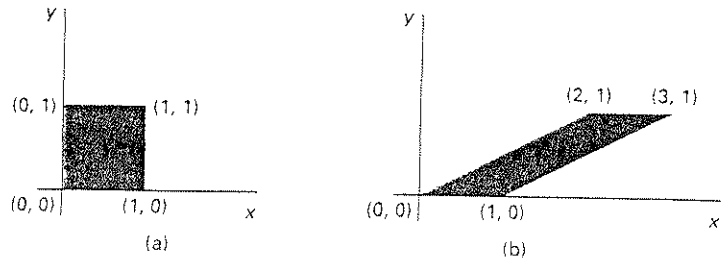


Figure 5-23
A unit square (a) is converted to a parallelogram (b) using the x-direction shear matrix 5-53 with $sh_x = 2$.

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

$$\begin{bmatrix} 1 & 0 & 0 \\ sh_y & 1 & -sh_y \cdot x_{ref} \\ 0 & 0 & 1 \end{bmatrix} \tag{5-57}$$

which generates transformed coordinate positions

$$x' = x, \qquad y' = sh_y(x - x_{ref}) + y \tag{5-58}$$

This transformation shifts a coordinate position vertically by an amount proportional to its distance from the reference line $x = x_{ref}$. Figure 5-25 illustrates the conversion of a square into a parallelogram with $sh_y = 1/2$ and $x_{ref} = -1$.

Shearing operations can be expressed as sequences of basic transformations. The x-direction shear matrix 5-53, for example, can be written as a composite transformation involving a series of rotation and scaling matrices that would scale the unit square of Fig. 5-23 along its diagonal, while maintaining the original lengths and orientations of edges parallel to the x axis. Shifts in the positions of objects relative to shearing reference lines are equivalent to translations.

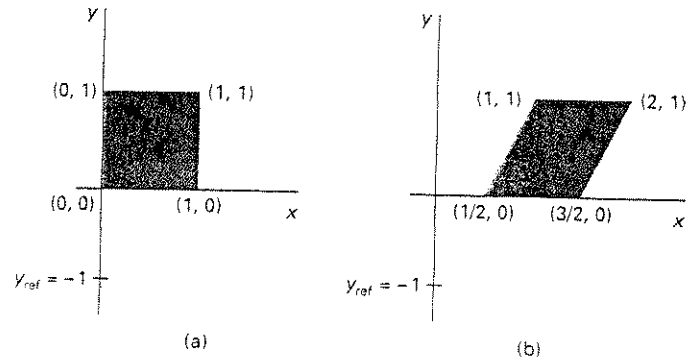


Figure 5-24
A unit square (a) is transformed to a shifted parallelogram (b) with $sh_x = 1/2$ and $y_{ref} = -1$ in the shear matrix 5-55.

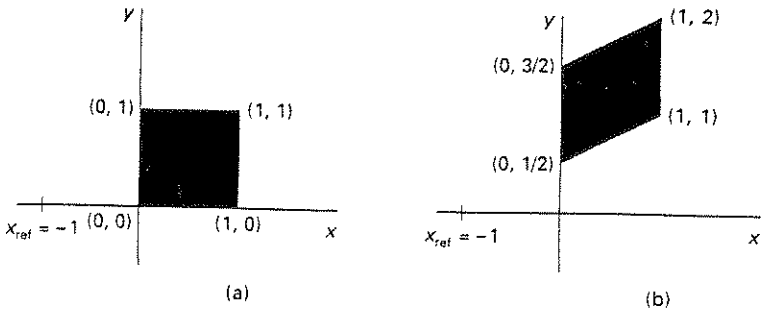


Figure 5-25
A unit square (a) is turned into a shifted parallelogram (b) with parameter values $sh_y = 1/2$ and $x_{ref} = -1$ in the y-direction using shearing transformation 5-57.

5-5 TRANSFORMATIONS BETWEEN COORDINATE SYSTEMS

Graphics applications often require the transformation of object descriptions from one coordinate system to another. Sometimes objects are described in non-Cartesian reference frames that take advantage of object symmetries. Coordinate descriptions in these systems must then be converted to Cartesian device coordinates for display. Some examples of two-dimensional non-Cartesian systems are polar coordinates, elliptical coordinates, and parabolic coordinates. In other cases, we need to transform between two Cartesian systems. For modeling and design applications, individual objects may be defined in their own local Cartesian references, and the local coordinates must then be transformed to position the objects within the overall scene coordinate system. A facility management program for office layouts, for instance, has individual coordinate reference descriptions for chairs and tables and other furniture that can be placed into a floor plan, with multiple copies of the chairs and other items in different positions. In other applications, we may simply want to reorient the coordinate reference for displaying a scene. Relationships between Cartesian reference systems and some common non-Cartesian systems are given in Appendix A. Here, we consider transformations between two Cartesian frames of reference.

Figure 5-26 shows two Cartesian systems, with the coordinate origins at (0, 0) and (x_0, y_0) and with an orientation angle θ between the x and x' axes. To transform object descriptions from xy coordinates to $x'y'$ coordinates, we need to set up a transformation that superimposes the $x'y'$ axes onto the xy axes. This is done in two steps:

1. Translate so that the origin (x_0, y_0) of the $x'y'$ system is moved to the origin of the xy system.
2. Rotate the x' axis onto the x axis.

Translation of the coordinate origin is expressed with the matrix operation

$$T(-x_0, -y_0) = \begin{bmatrix} 1 & 0 & -x_0 \\ 0 & 1 & -y_0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (5\text{-}59)$$