CMPE 466 COMPUTER GRAPHICS

Chapter 7
2D Geometric Transformations

Instructor: D. Arifler

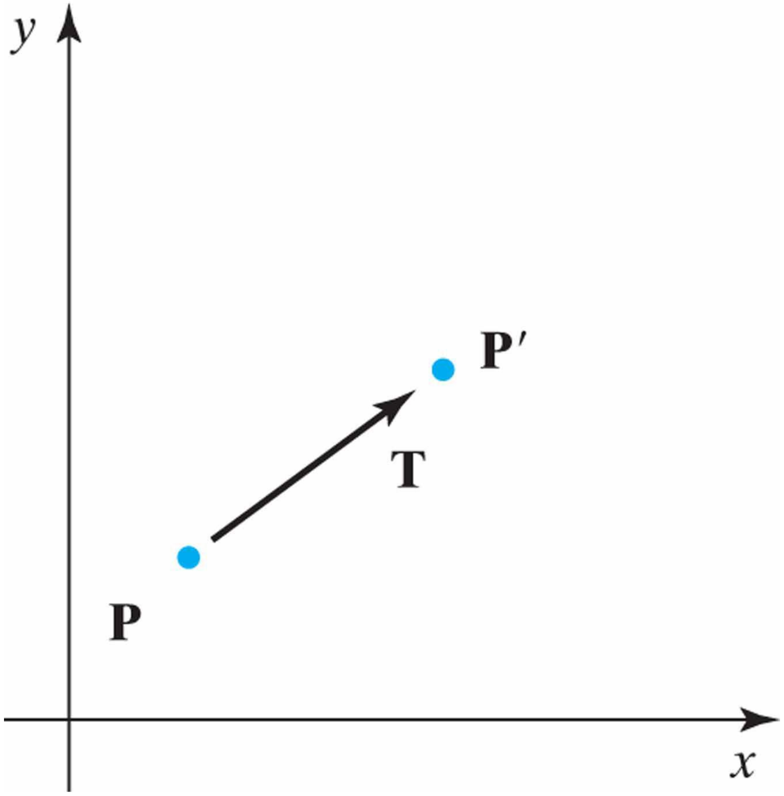
Material based on
- *Computer Graphics with OpenGL®,* Fourth Edition by Donald Hearn, M. Pauline Baker, and Warren R. Carithers
- *Fundamentals of Computer Graphics*, Third Edition by by Peter Shirley and Steve Marschner
- *Computer Graphics* by F. S. Hill

Basic geometric transformations

- Translation
- Rotation
- Scaling

2D translation

Figure 7-1 Translating a point from position **P** to position **P**' using a translation vector **T**.



2D translation equations

$$x' = x + t_x, \qquad y' = y + t_y$$

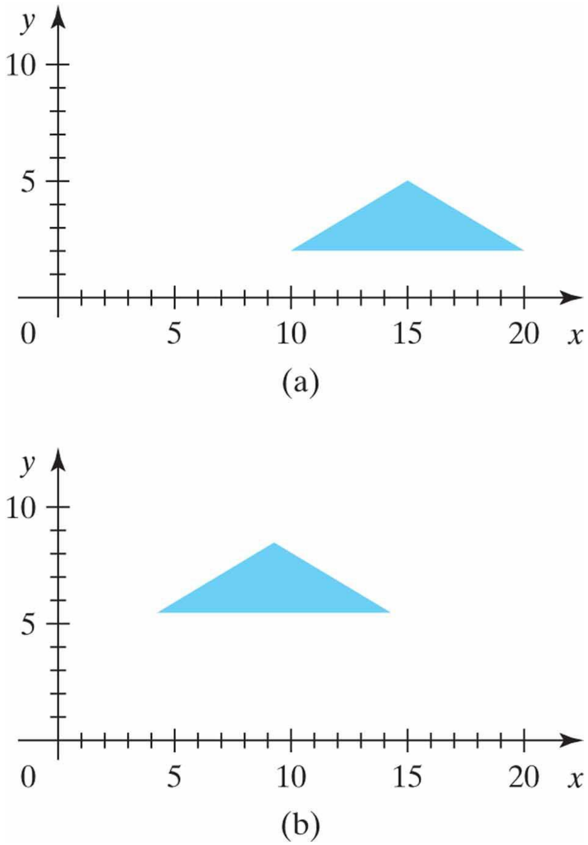
$$\mathbf{P} = \begin{bmatrix} x \\ y \end{bmatrix}, \qquad \mathbf{P}' = \begin{bmatrix} x' \\ y' \end{bmatrix}, \qquad \mathbf{T} = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

$$\mathbf{P}' = \mathbf{P} + \mathbf{T}$$

Translation is a rigid-body transformation: Objects are moved without deformation.

2D translation example

Figure 7-2 Moving a polygon from position (a) to position (b) with the translation vector (−5.50, 3.75).



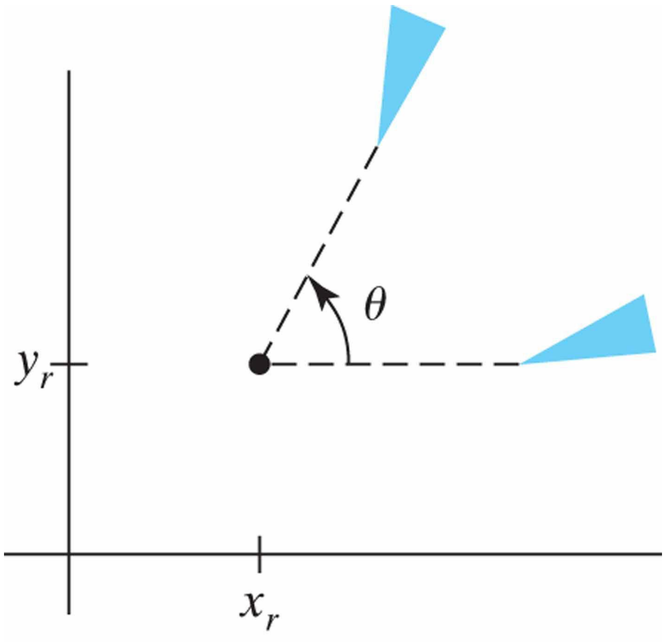
2D translation example program

```
class wcPt2D {
   public:
      GLfloat x, y;
};

void translatePolygon (wcPt2D * verts, GLint nVerts, GLfloat tx, GLfloat ty)
{
   GLint k;

   for (k = 0; k < nVerts; k++) {
      verts [k].x = verts [k].x + tx;
      verts [k].y = verts [k].y + ty;
   }
   glBegin (GL_POLYGON);
      for (k = 0; k < nVerts; k++)
         glVertex2f (verts [k].x, verts [k].y);
   glEnd ( );
}
```

2D rotation

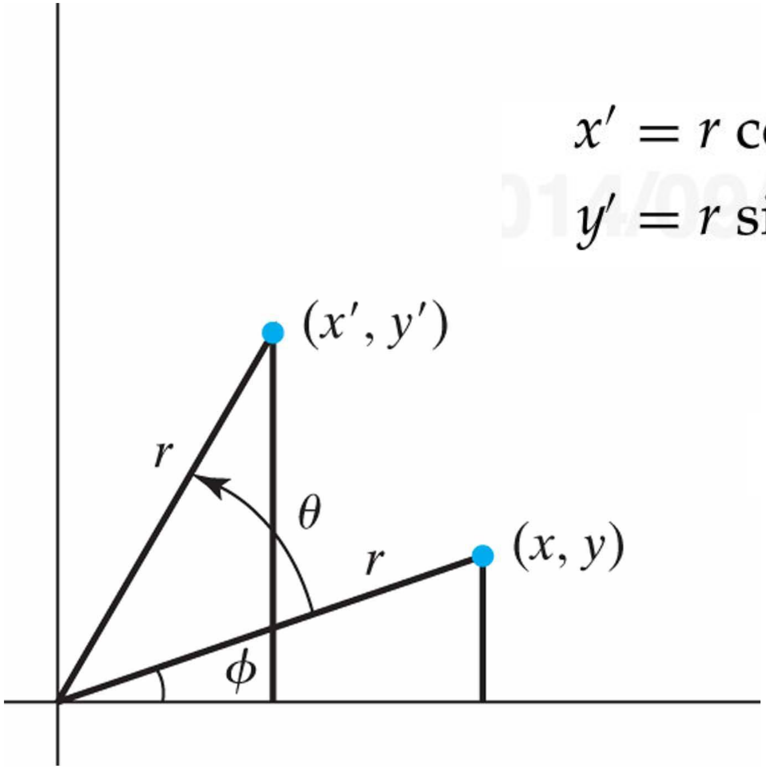
- All points of the object are transformed to new positions by rotating the points through a specified rotation angle about the rotation axis (in 2D, rotation pivot or pivot point)

Figure 7-3 Rotation of an object through angle *θ* about the pivot point *(x_r, y_r)*.



2D rotation

Figure 7-4 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*.



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

Setting

$$x = r\cos\phi, \qquad y = r\sin\phi$$

we have

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

2D rotation in matrix form

Equations can be compactly expressed in matrix form:

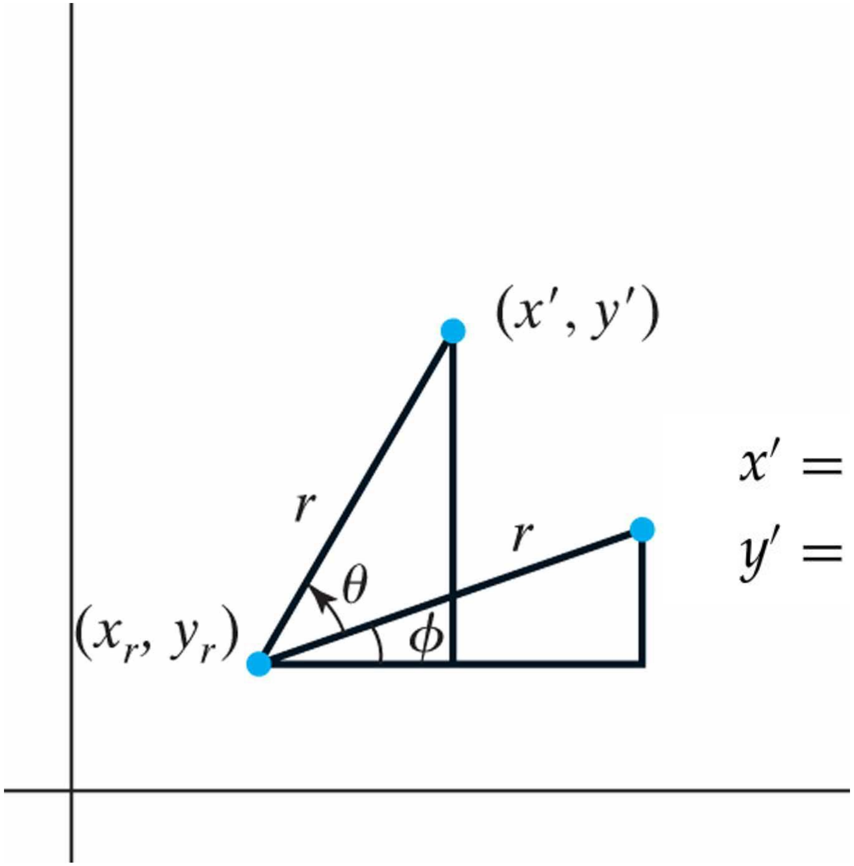
$$\mathbf{P}' = \mathbf{R} \cdot \mathbf{P}$$

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

Rotation is a rigid-body transformation: Objects are moved without deformation.

Rotation about a general pivot point

Figure 7-5 Rotating a point from position (x, y) to position (x', y') through an angle θ about rotation point (x_r, y_r).



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

2D rotation example

```
class wcPt2D {
   public:
      GLfloat x, y;
};

void rotatePolygon (wcPt2D * verts, GLint nVerts, wcPt2D pivPt,
                      GLdouble theta)
{
   wcPt2D * vertsRot;    // Make necessary allocations!!
   GLint k;

   for (k = 0; k < nVerts; k++) {
      vertsRot [k].x = pivPt.x + (verts [k].x - pivPt.x) * cos (theta)
                        - (verts [k].y - pivPt.y) * sin (theta);
      vertsRot [k].y = pivPt.y + (verts [k].x - pivPt.x) * sin (theta)
                        + (verts [k].y - pivPt.y) * cos (theta);
   }
   glBegin {GL_POLYGON};
      for (k = 0; k < nVerts; k++)
         glVertex2f (vertsRot [k].x, vertsRot [k].y);
   glEnd ( );
}
```

2D scaling

$$x' = x \cdot s_x, \qquad y' = y \cdot s_y$$

s_x and s_y are scaling factors
s_x scales an object in x direction
s_y scales an object in y direction

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

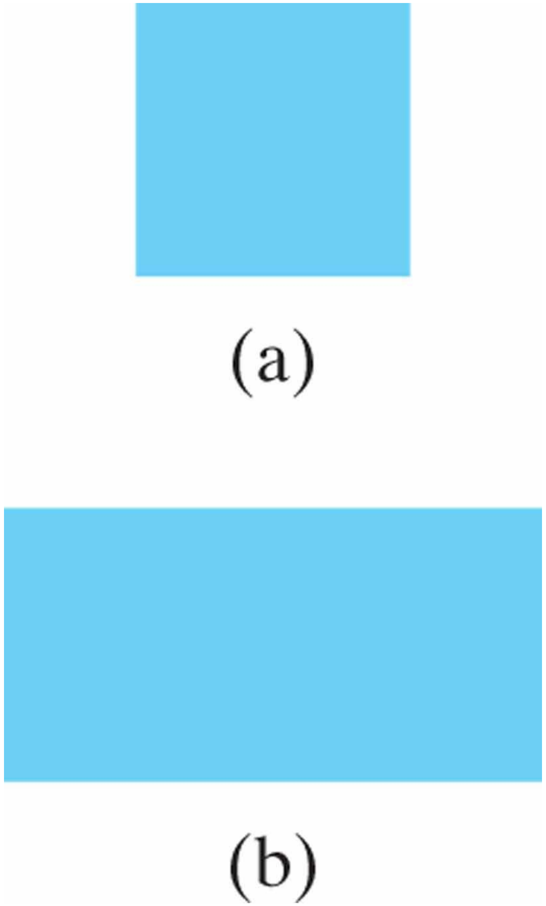
$$\mathbf{P}' = \mathbf{S} \cdot \mathbf{P}$$

If $s_x=s_y$, we have uniform scaling: Object proportions are maintained.
If $s_x \neq s_y$, we have differential scaling.
Negative scaling factors resizes and reflects the object about one or more of the coordinate axes.

2D scaling

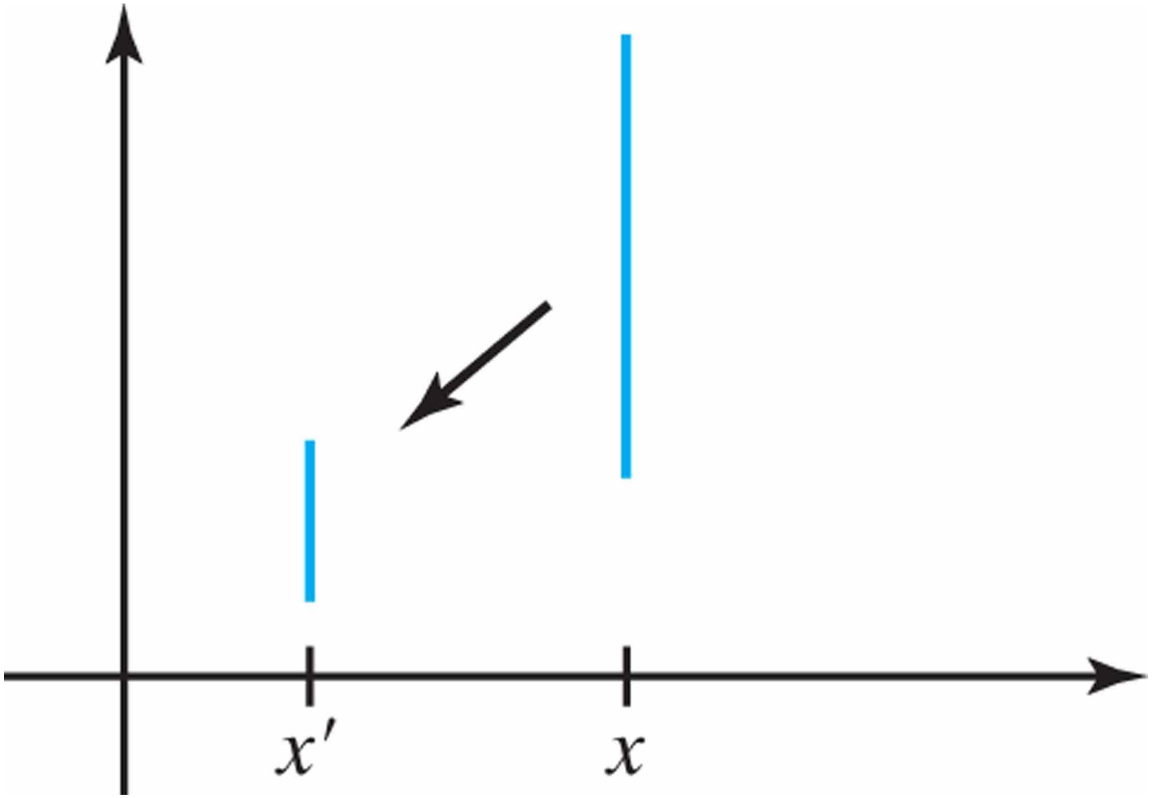
Figure 7-6 Turning a square (a) into a rectangle (b) with scaling factors $s_x = 2$ and $s_y = 1$.

Scaling factors greater than 1 produce enlargements.



2D scaling

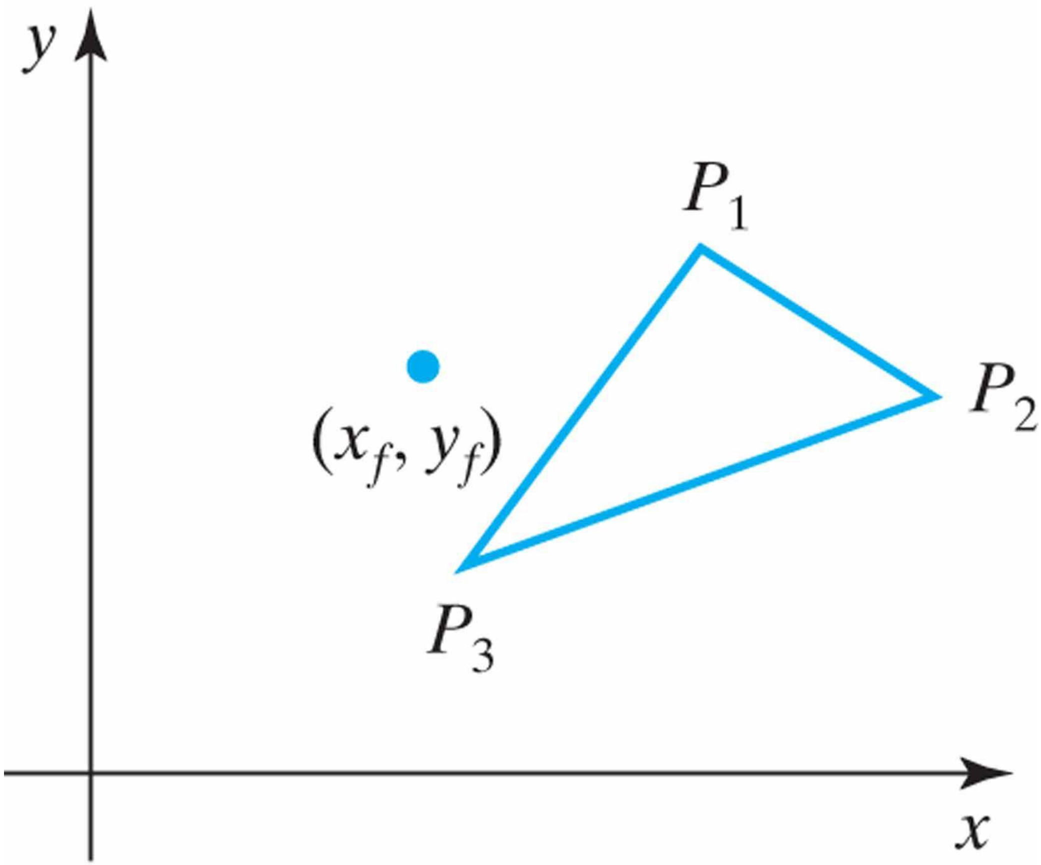
Figure 7-7 A line scaled with Equation 7-12 using $s_x = s_y = 0.5$ is reduced in size and moved closer to the coordinate origin.



Positive scaling values less than 1 reduce the size of objects.

Scaling relative to a fixed point

Figure 7-8 Scaling relative to a chosen fixed point (x_f, y_f). The distance from each polygon vertex to the fixed point is scaled by Equations 7-13.



Fixed point remains unchanged after the scaling transformation.

2D scaling relative to a fixed point

$$x' - x_f = (x - x_f)s_x, \qquad y' - y_f = (y - y_f)s_y$$

$$x' = x \cdot s_x + x_f(1 - s_x)$$
$$y' = y \cdot s_y + y_f(1 - s_y)$$

2D scaling example

```
class wcPt2D {
   public:
      GLfloat x, y;
};

void scalePolygon (wcPt2D * verts, GLint nVerts, wcPt2D fixedPt,
                            GLfloat sx, GLfloat sy)
{
   wcPt2D vertsNew;  // Make necessary allocations!!
   GLint k;

   for (k = 0; k < nVerts; k++) {
     vertsNew [k].x = verts [k].x * sx + fixedPt.x * (1 - sx);
     vertsNew [k].y = verts [k].y * sy + fixedPt.y * (1 - sy);
   }
   glBegin {GL_POLYGON};
      for (k = 0; k < nVerts; k++)
         glVertex2f (vertsNew [k].x, vertsNew [k].y);
   glEnd ( );
}
```

Matrix representations and homogeneous coordinates

- Multiplicative and translational terms for a 2D transformation can be combined into a single matrix
- This expands representations to 3x3 matrices
  - Third column is used for translation terms
- Result: All transformation equations can be expressed as matrix multiplications
- Homogeneous coordinates: (x_h, y_h, h)
  - Carry out operations on points and vectors “homogeneously”
  - h: Non-zero homogeneous parameter such that

$$x = \frac{x_h}{h}, \qquad y = \frac{y_h}{h}$$

  - We can also write: (hx, hy, h)
  - h=1 is a convenient choice so that we have (x, y, 1)
  - Other values of h are useful in 3D viewing transformations

2D translation matrix

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

$$\mathbf{P}' = \mathbf{T}(t_x, t_y) \cdot \mathbf{P}$$

2D rotation matrix

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

$$\mathbf{P}' = \mathbf{R}(\theta) \cdot \mathbf{P}$$

2D scaling matrix

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

$$\mathbf{P}' = \mathbf{S}(s_x, s_y) \cdot \mathbf{P}$$

Inverse transformations

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

Inverse translation

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

Inverse rotation

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

Inverse scaling

Composite transformations

- Composite transformation matrix is formed by calculating the product of individual transformations

$$\begin{aligned} \mathbf{P}' &= \mathbf{M}_2 \cdot \mathbf{M}_1 \cdot \mathbf{P} \\ &= \mathbf{M} \cdot \mathbf{P} \end{aligned}$$

- Successive translations (additive)

$$\begin{aligned} \mathbf{P}' &= \mathbf{T}(t_{2x}, t_{2y}) \cdot \{\mathbf{T}(t_{1x}, t_{1y}) \cdot \mathbf{P}\} \\ &= \{\mathbf{T}(t_{2x}, t_{2y}) \cdot \mathbf{T}(t_{1x}, t_{1y})\} \cdot \mathbf{P} \end{aligned}$$

$$\begin{bmatrix} 1 & 0 & t_{2x} \\ 0 & 1 & t_{2y} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & t_{1x} \\ 0 & 1 & t_{1y} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_{1x} + t_{2x} \\ 0 & 1 & t_{1y} + t_{2y} \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{T}(t_{2x}, t_{2y}) \cdot \mathbf{T}(t_{1x}, t_{1y}) = \mathbf{T}(t_{1x} + t_{2x}, t_{1y} + t_{2y})$$

Composite transformations

- Successive rotations (additive)

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

$$\mathbf{R}(\theta_2) \cdot \mathbf{R}(\theta_1) = \mathbf{R}(\theta_1 + \theta_2)$$

$$\mathbf{P}' = \mathbf{R}(\theta_1 + \theta_2) \cdot \mathbf{P}$$

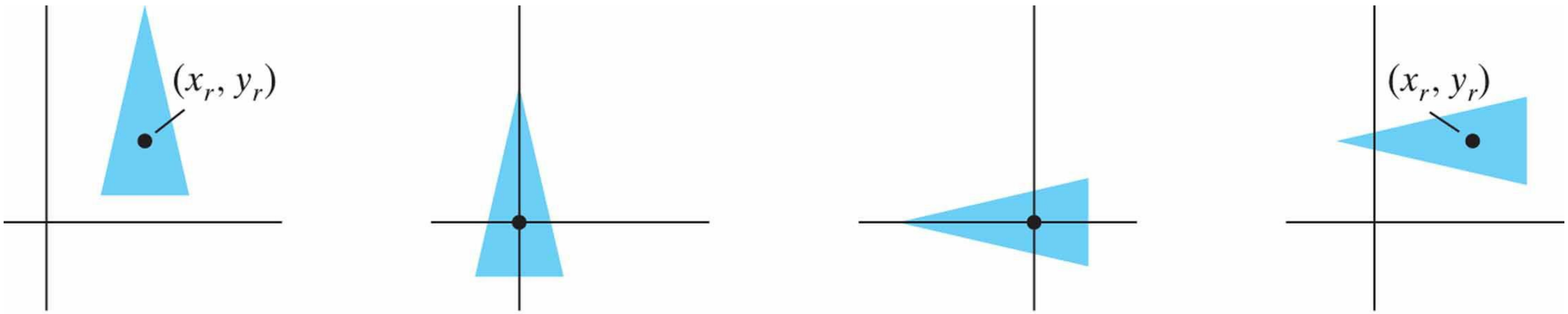
- Successive scaling (multiplicative)

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

$$\mathbf{S}(s_{2x}, s_{2y}) \cdot \mathbf{S}(s_{1x}, s_{1y}) = \mathbf{S}(s_{1x} \cdot s_{2x}, \quad s_{1y} \cdot s_{2y})$$

2D pivot-point rotation

Figure 7-9 A transformation sequence for rotating an object about a specified pivot point using the rotation matrix **R**(θ) of transformation 7-19.



(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)

2D pivot-point rotation

Note the order of operations:

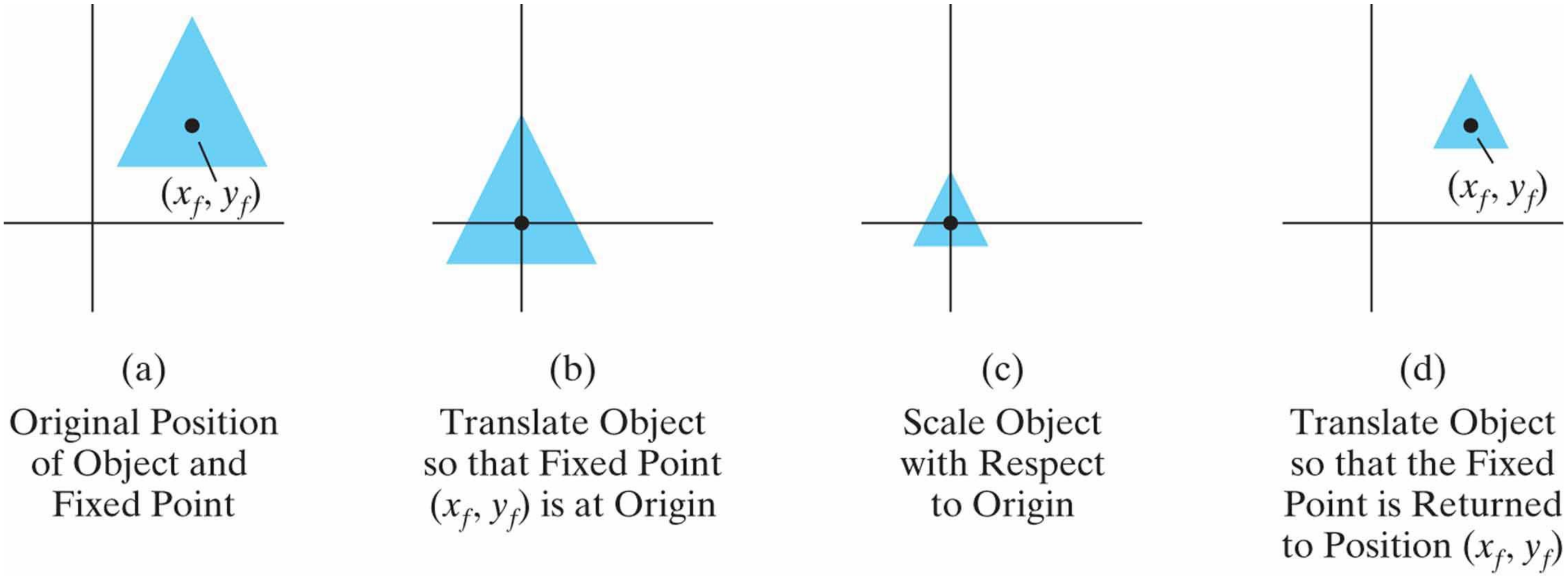
$$\begin{bmatrix} 1 & 0 & x_r \\ 0 & 1 & y_r \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_r \\ 0 & 1 & -y_r \\ 0 & 0 & 1 \end{bmatrix}$$

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

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

2D fixed-point scaling

Figure 7-10 A transformation sequence for scaling an object with respect to a specified fixed position using the scaling matrix $\mathbf{S}(s_x, s_y)$ of transformation 7-21.



2D fixed-point scaling

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

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

Matrix concatenation properties

- Multiplication is associative

$$\mathbf{M}_3 \cdot \mathbf{M}_2 \cdot \mathbf{M}_1 = (\mathbf{M}_3 \cdot \mathbf{M}_2) \cdot \mathbf{M}_1 = \mathbf{M}_3 \cdot (\mathbf{M}_2 \cdot \mathbf{M}_1)$$

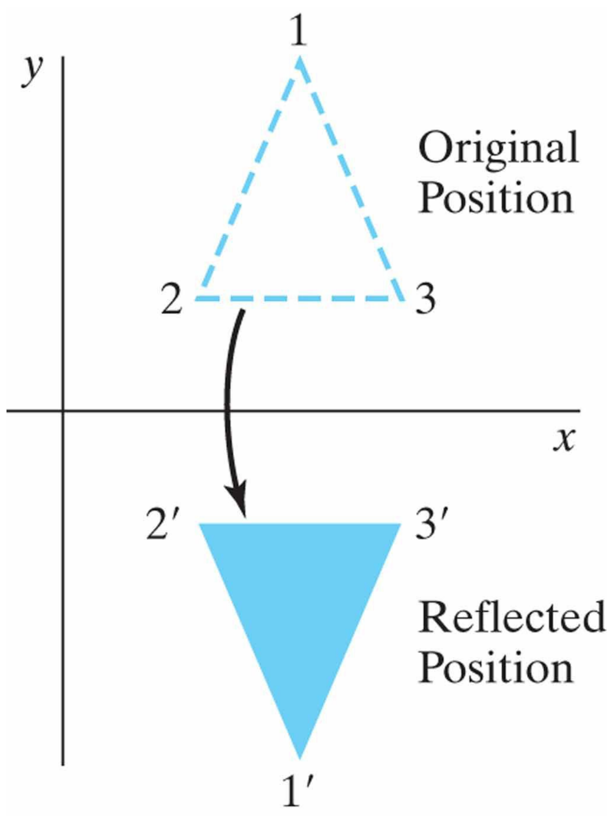
- Multiplication is NOT commutative
  - Unless the sequence of transformations are all of the same kind
  - M_2M_1 is not equal to M_1M_2 in general

Computational efficiency

- Formulation of a concatenated matrix may be more efficient
  - Requires fewer multiply/add operations
- Rotation calculations require trigonometric evaluations
  - In animations with small-angle rotations, approximations (e.g. power series) and iterative calculations can reduce complexity

Other transformations: reflection

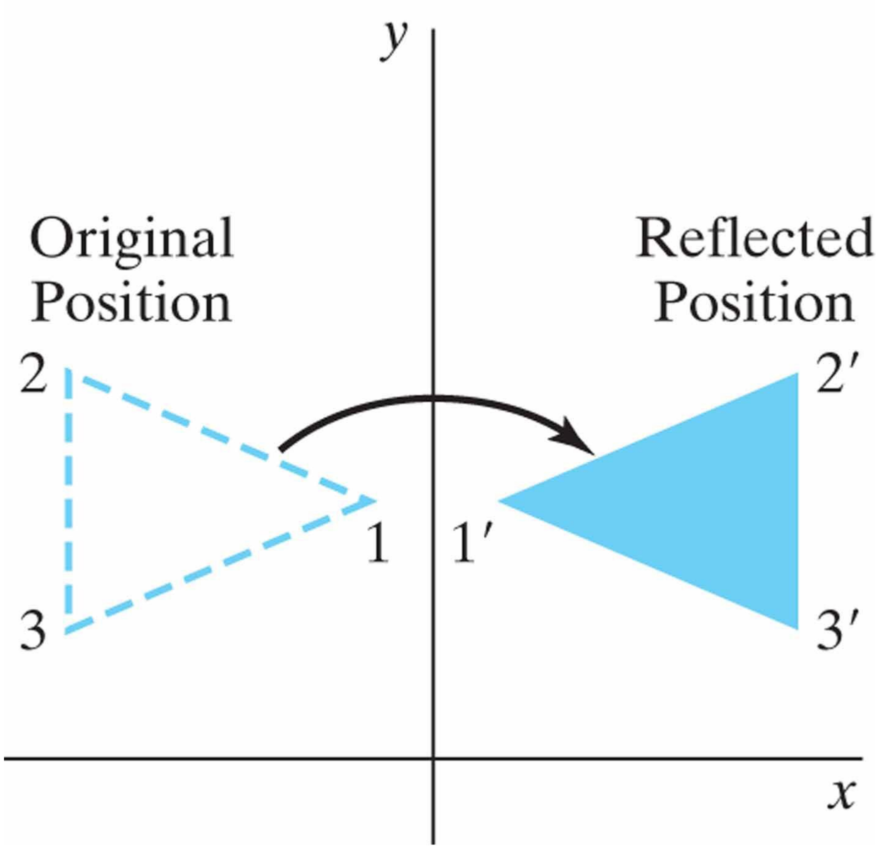
Figure 7-16 Reflection of an object about the *x* axis.



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

Reflection

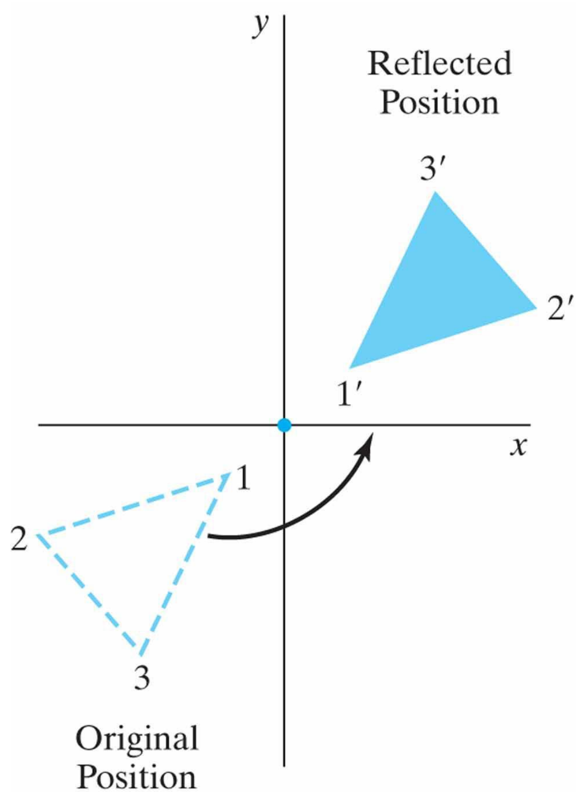
Figure 7-17 Reflection of an object about the *y* axis.



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

Reflection

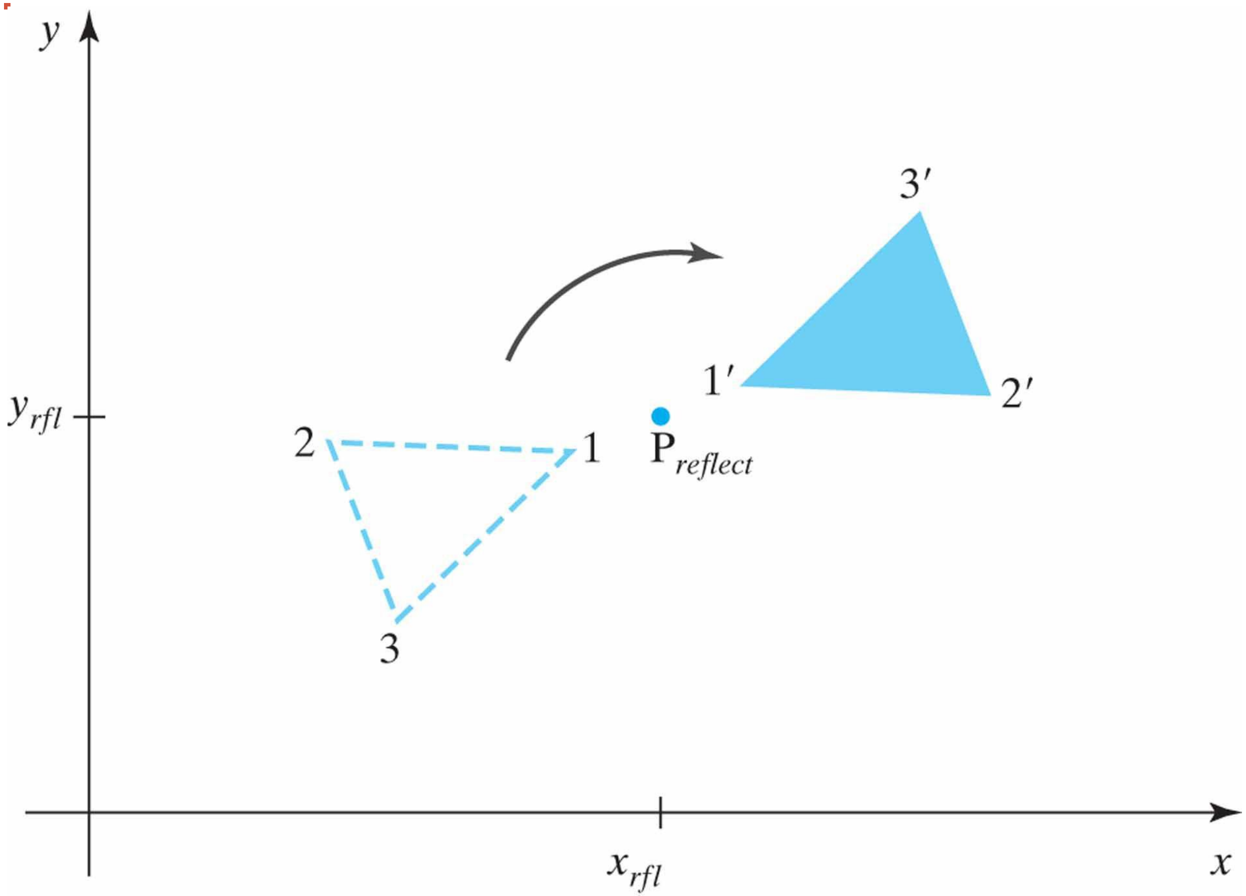
Figure 7-18 Reflection of an object relative to the coordinate origin. This transformation can be accomplished with a rotation in the *xy* plane about the coordinate origin.



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

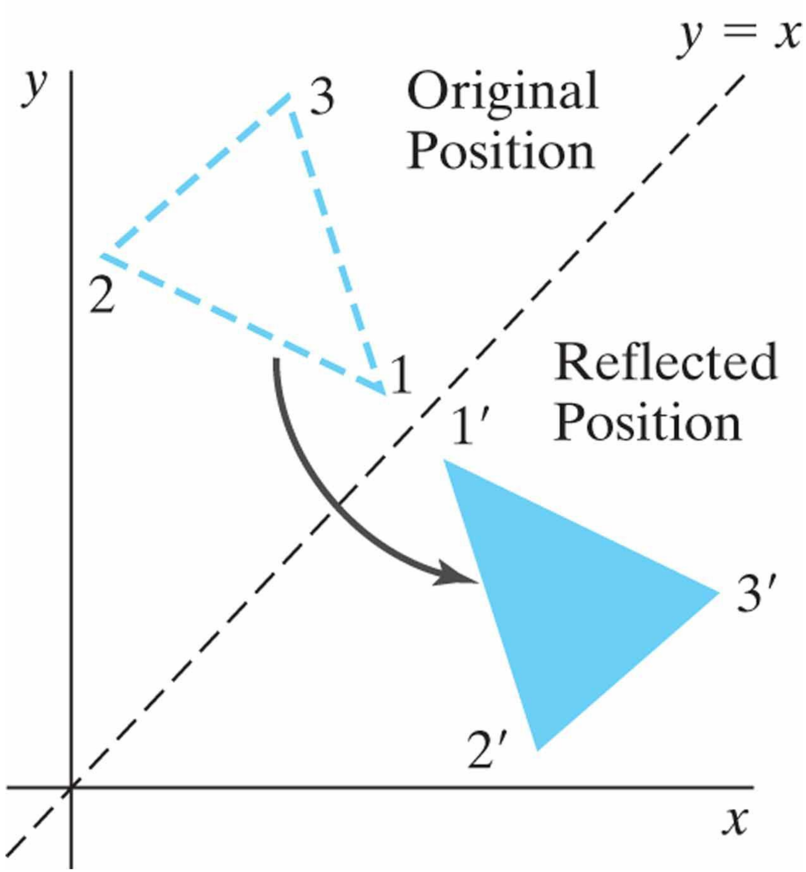
Reflection

Figure 7-19 Reflection of an object relative to an axis perpendicular to the *xy* plane and passing through point $P_{reflect}$.



Reflection

Figure 7-20 Reflection of an object with respect to the line *y* = *x* .

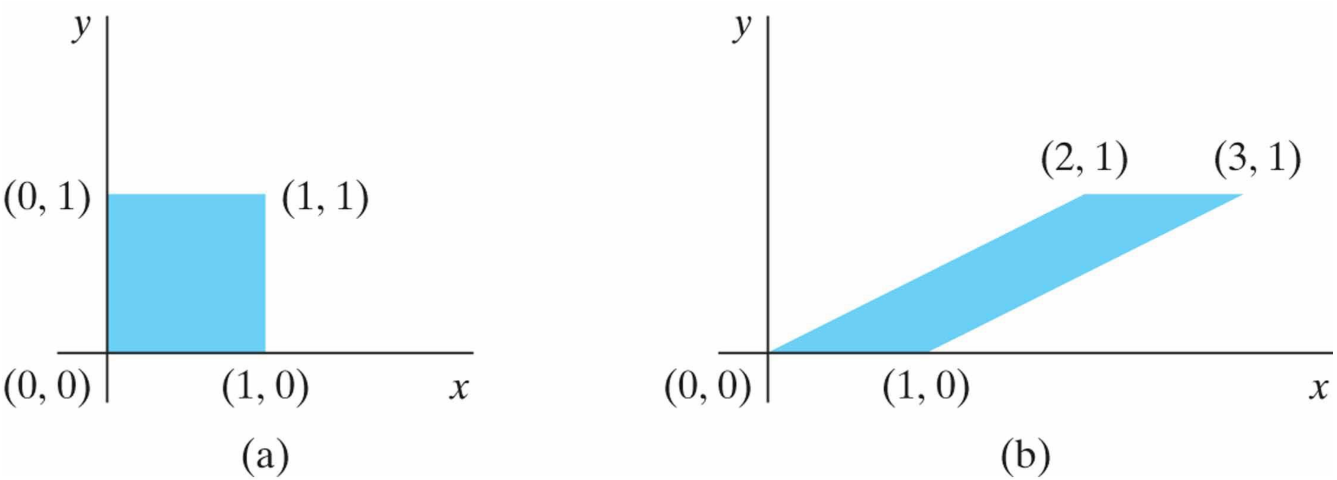


$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Other transformations: shear

- 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

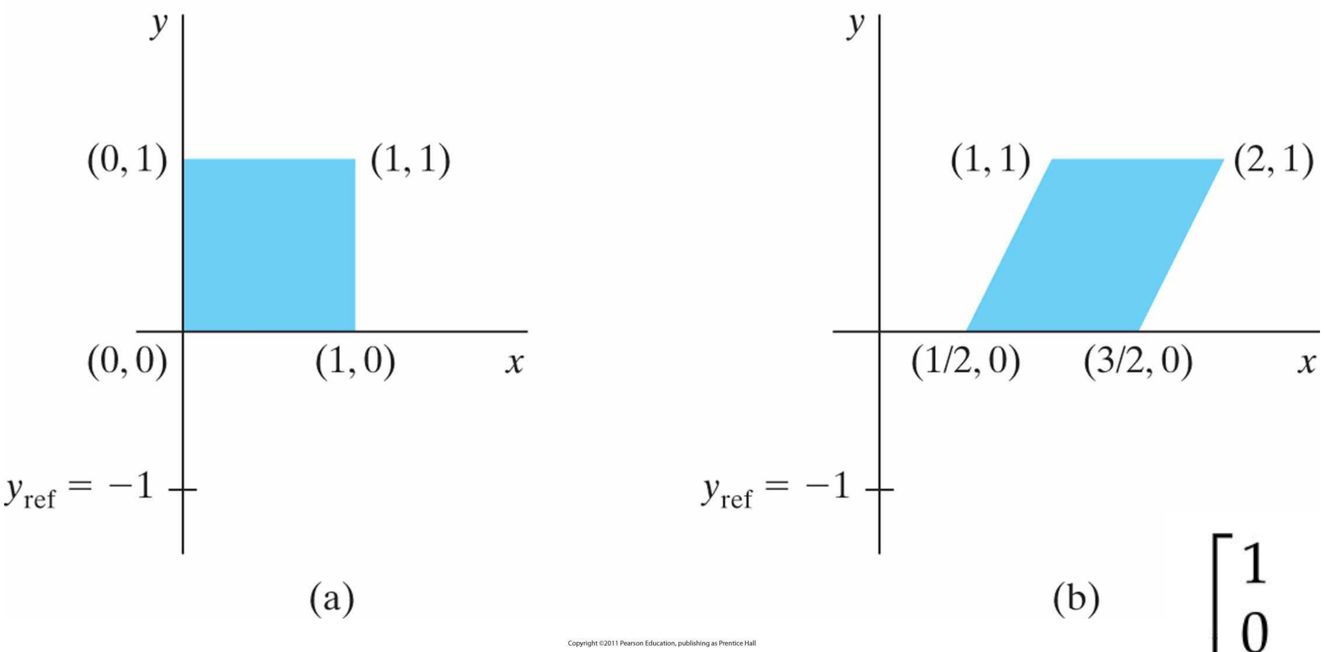
Figure 7-23 A unit square (a) is converted to a parallelogram (b) using the *x* -direction shear matrix 7-57 with $sh_x = 2$.



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

Shear

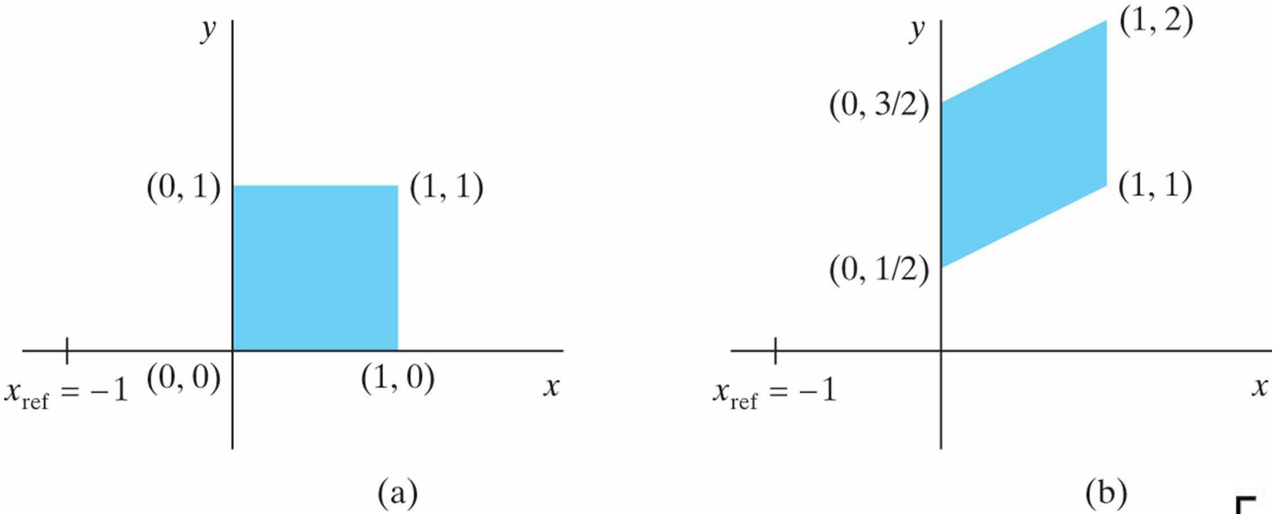
Figure 7-24 A unit square (a) is transformed to a shifted parallelogram (b) with sh_x = 0.5 and y_{ref} = −1 in the shear matrix 7-59.



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

Shear

Figure 7-25 A unit square (a) is turned into a shifted parallelogram (b) with parameter values $sh_y = 0.5$ and $x_{ref} = -1$ in the y -direction shearing transformation 7-61.



$$\begin{bmatrix} 1 & 0 & 0 \\ sh_y & 1 & -sh_y \cdot x_{\text{ref}} \\ 0 & 0 & 1 \end{bmatrix}$$

Transformations between 2D coordinate systems

Figure 7-30 A Cartesian *x′ y′* system positioned at (x_0, y_0) with orientation θ in an *xy* Cartesian system.

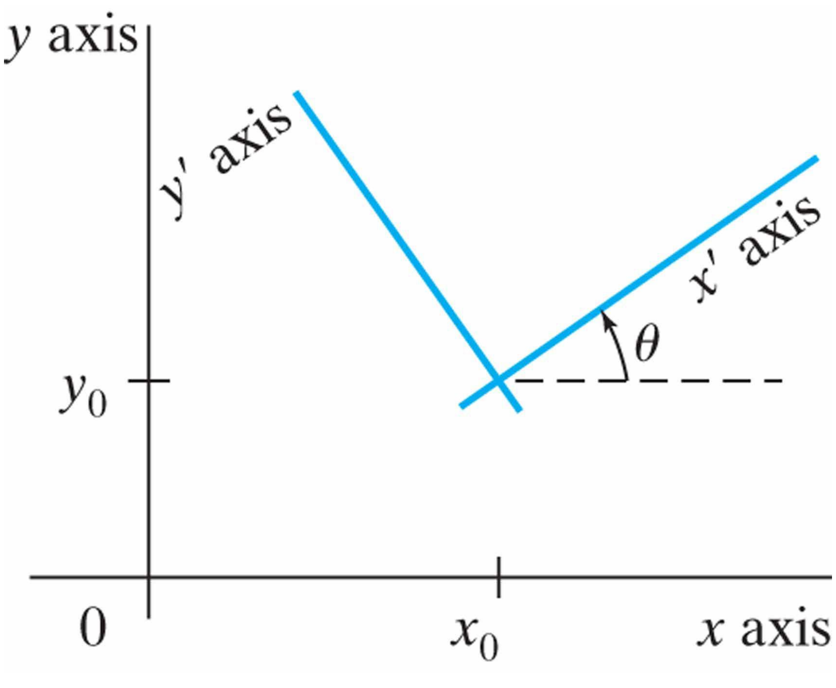
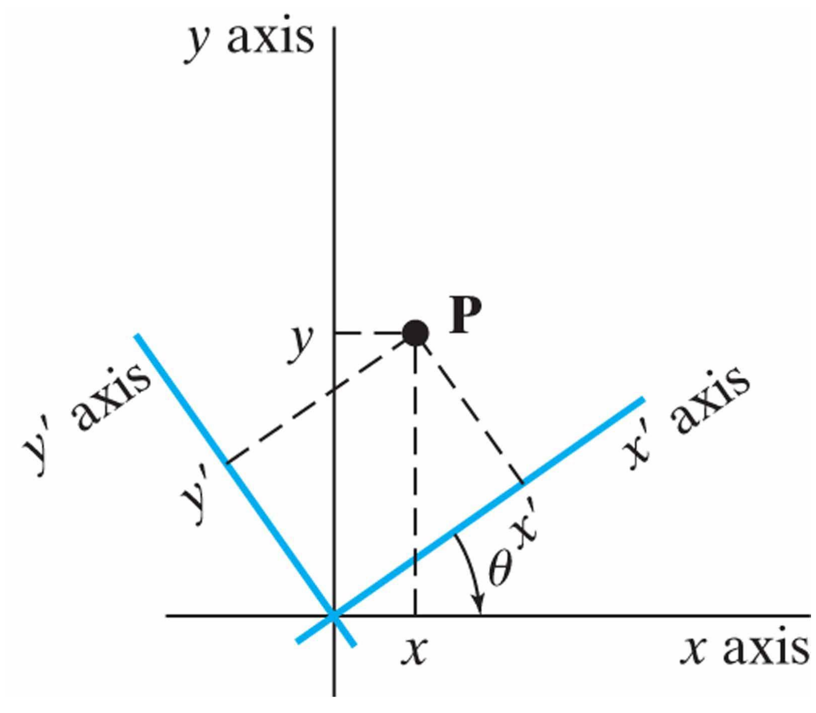
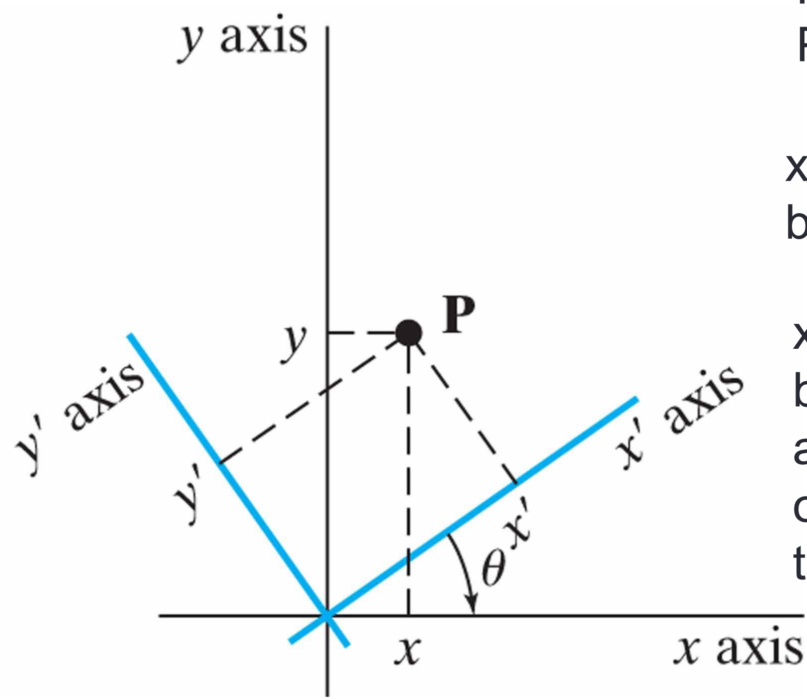


Figure 7-31 Position of the reference frames shown in Figure 7-30 after translating the origin of the *x′ y′* system to the coordinate origin of the *xy* system.



Transform object descriptions from xy coordinates to x’y’ coordinates

Transformations



P=(x,y) in system xy
P=(x',y') in system x'y'

x'y' system can be obtained by rotation of xy by Θ counter-clockwise

xy system can be obtained by rotation of x'y' by Θ clockwise. For this, you can also assign the elements of x' to the first row of the rotation matrix and the elements of y' to the second row.

Example: Transform from xy to x'y' frame:

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

Transformations between coordinate systems

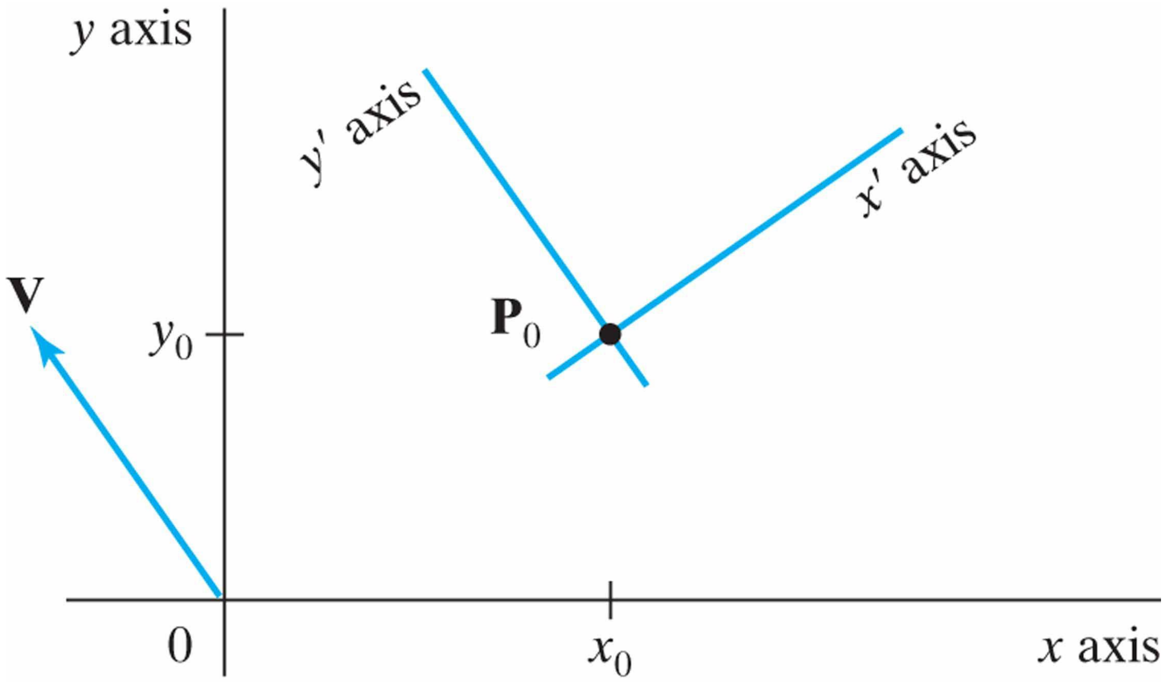
$$\mathbf{T}(-x_0, -y_0) = \begin{bmatrix} 1 & 0 & -x_0 \\ 0 & 1 & -y_0 \\ 0 & 0 & 1 \end{bmatrix}$$

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

$$\mathbf{M}_{xy,x'y'} = \mathbf{R}(-\theta) \cdot \mathbf{T}(-x_0, -y_0)$$

Alternative method

Figure 7-32 Cartesian system $x'\,y'$ with origin at P0 = (x_0, y_0) and y' axis parallel to vector **V**.



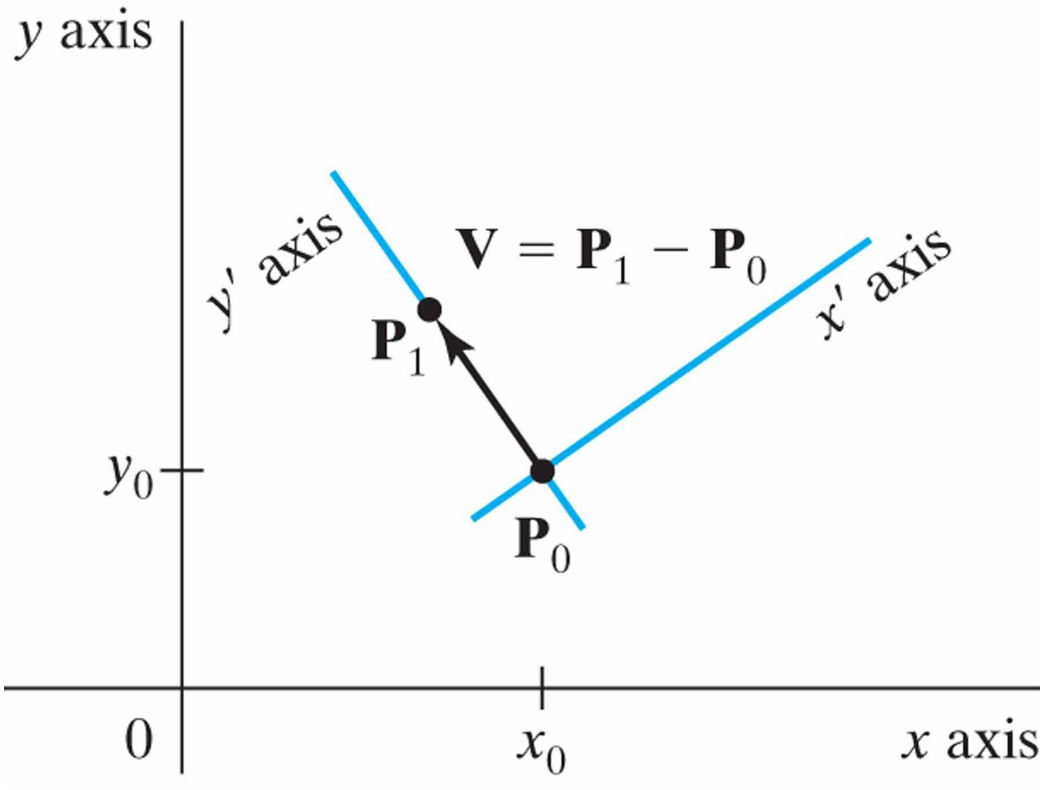
$$\mathbf{v} = \frac{\mathbf{V}}{|\mathbf{V}|} = (v_x, v_y)$$

$$\mathbf{u} = (v_y, -v_x) = (u_x, u_y)$$

$$R = \begin{bmatrix} u_x & u_y & 0 \\ v_x & v_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Transformations

Figure 7-33 A Cartesian x ′y′ system defined with two coordinate positions, P_0 and P_1, within an *xy* reference frame.



$$\mathbf{v} = \frac{\mathbf{P}_1 - \mathbf{P}_0}{|\mathbf{P}_1 - \mathbf{P}_0|}$$

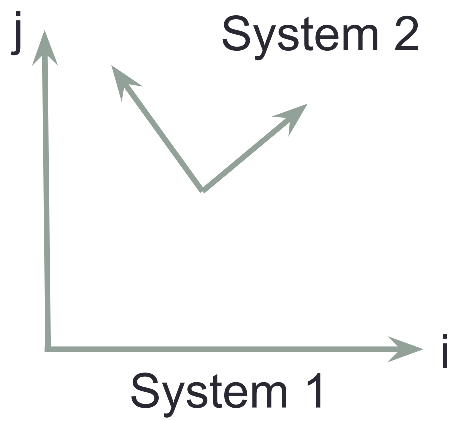
Example: Rotating points vs. rotating coordinate systems

- Consider the following transformation:
  - Rotation of points through 30° about point $v=(-2, 3)^T$
    - Translate the point through vector $-v=(2, -3)^T$
    - Rotate about origin through 30°
    - Translate the point back through $v=(-2, 3)^T$
  - Hence the composite transformation is:

$$M = \begin{pmatrix} 1 & 0 & -2 \\ 0 & 1 & 3 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos 30 & -\sin 30 & 0 \\ \sin 30 & \cos 30 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & -3 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0.866 & -0.5 & 1.232 \\ 0.5 & 0.866 & 1.402 \\ 0 & 0 & 1 \end{pmatrix}$$

Example continued

- You may think of this as mapping the origin and i and j axes into system 2
- The columns of the matrix in the previous slide reveal the transformed coordinate system



Example continued

- Now consider the point $P=(x^{(2)},y^{(2)},1)^T$ in System 2
- What are the coordinates of this point expressed in terms of the original System 1?
- The answer is MP
- For example, $(1, 2, 1)^T$ in System 2 lies at $(1.098, 3.634, 1)^T$ in System 1
- Now, consider the point $P=(x^{(1)},y^{(1)},1)^T$ in System 1
- What are the coordinates of this point expressed in terms of System 2?
- The answer is $M^{-1}P$

OpenGL matrix operations

- glMatrixMode (GL_MODELVIEW)
  - Designates the matrix that is to be used for projection transformation (current matrix)
- glLoadIdentity ( )
  - Assigns the identity matrix to the current matrix
- Note: OpenGL stores matrices in column-major order
  - Reference to a matrix element m_{jk} in OpenGL is a reference to the element in column j and row k
  - glMultMatrix* ( ) post-multiplies the current matrix
- In OpenGL, the transformation specified last is the one applied first

```
glMatrixMode (GL_MODELVIEW);

glLoadIdentity ( );         // Set current matrix to the identity.
glMultMatrixf (elemsM2);    // Postmultiply identity with matrix M2.
glMultMatrixf (elemsM1);    // Postmultiply M2 with matrix M1.
```

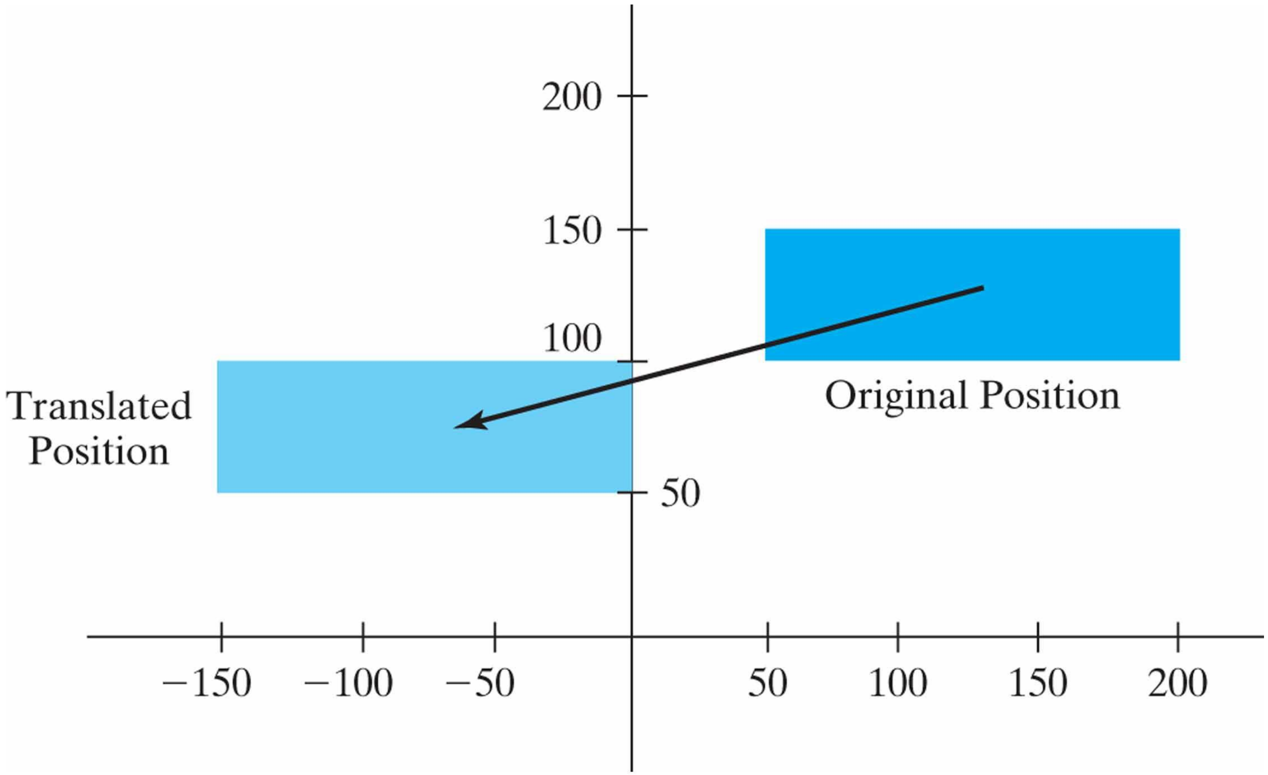
$$\mathbf{M} = \mathbf{M}_2 \cdot \mathbf{M}_1$$

OpenGL transformation example

```
glMatrixMode (GL_MODELVIEW);

glColor3f (0.0, 0.0, 1.0);
glRecti (50, 100, 200, 150);         // Display blue rectangle.

glColor3f (1.0, 0.0, 0.0);
glTranslatef (-200.0, -50.0, 0.0); // Set translation parameters.
glRecti (50, 100, 200, 150);         // Display red, translated rectangle.

glLoadIdentity ( );                  // Reset current matrix to identity.
glRotatef (90.0, 0.0, 0.0, 1.0);    // Set 90-deg. rotation about z axis.
glRecti (50, 100, 200, 150);         // Display red, rotated rectangle.

glLoadIdentity ( );                  // Reset current matrix to identity.
glScalef (-0.5, 1.0, 1.0);          // Set scale-reflection parameters.
glRecti (50, 100, 200, 150);         // Display red, transformed rectangle.
```

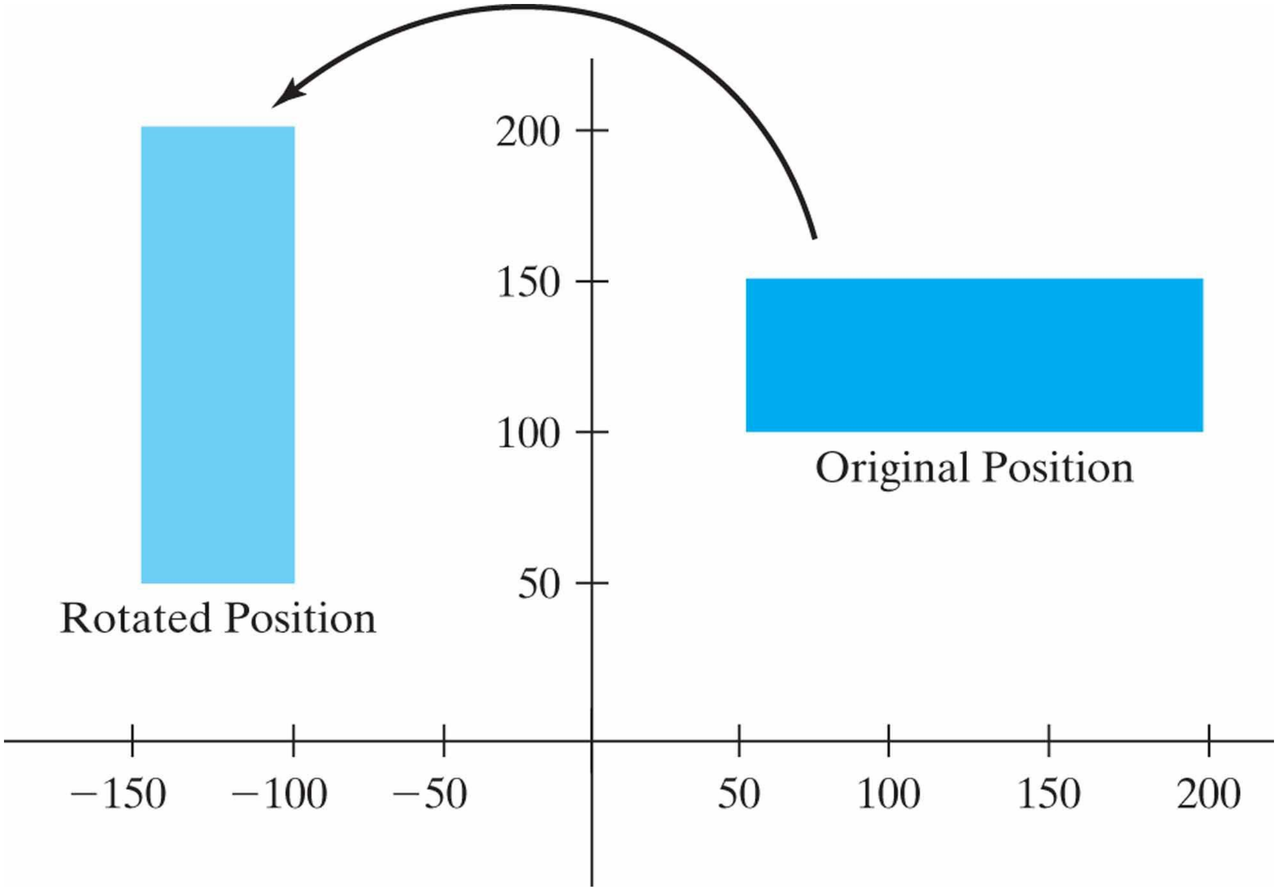
OpenGL transformation example

Figure 7-34 Translating a rectangle using the OpenGL function glTranslatef (−200.0, −50.0, 0.0).



OpenGL transformation example

Figure 7-35 Rotating a rectangle about the *z* axis using the OpenGL function glRotatef (90.0, 0.0, 0.0, 1.0).



OpenGL transformation example

Figure 7-36 Scaling and reflecting a rectangle using the OpenGL function glScalef (−0.5, 1.0, 1.0).