RV College of Engineering®

Go, change the world

Geometric Transformations

UNIT-III

Geometric Transformations:

2-D geometrical transformations: Unit (III)

- ✓ Translation, Scaling, Rotation,
- ✓ reflection and shear transformations,
- ✓ matrix representations and homogeneous coordinates,
- ✓ composite transforms,
- ✓ transformations between coordinate systems.

3-D Geometric transformations: Unit (IV)

- ✓ Translation, rotation, scaling,

Geometric Transformations :

Transformations

- The geometrical changes of an object from a current state to modified state.
- ***Need*** of transformations - To manipulate the initially created object and to display the modified object without having to redraw it.

Geometric Transformations :

Transformations

- 2 ways to achieve:
  - Object Transformation
    - ❑ Alter the coordinates descriptions an object
    - ❑ Translation, rotation, scaling etc.
    - ❑ Coordinate system unchanged
  - Coordinate transformation
    - ❑ Produce a different coordinate system

Geometric Transformations :

- Sometimes also called ***modeling transformations***
  - **Geometric transformations**: Changing an object’s position (translation), orientation (rotation) or size (scaling)
  - Modeling transformations: Constructing a scene or hierarchical description of a complex object
- Others transformations: reflection and shearing operations

Geometric Transformations :

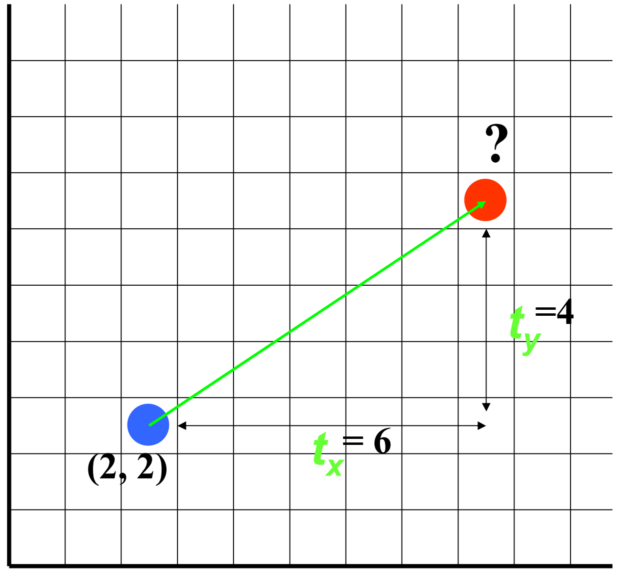
- **2D Translation**
- **A translation moves all points in an object along the same straight-line path to new positions.**
- **The path is represented by a vector, called the translation or shift vector.**
- **We can write the components:**

$$p'_x = p_x + t_x$$
$$p'_y = p_y + t_y$$

or in matrix form:

$$P' = P + T$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

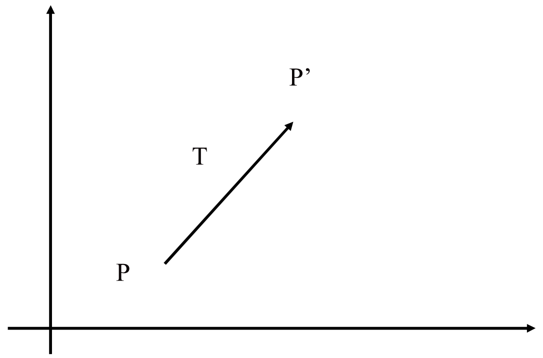


Basic 2D Geometric Transformations:

- **2D Translation**

  - $x' = x + t_x \, , \; y' = y + t_y$

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



  - P'=P+T
  - Translation moves the object without deformation ***(rigid-body transformation)***

Basic 2D Geometric Transformations (cont.)

- 2D Translation
  - To move a line segment, apply the transformation equation to each of the two line endpoints and redraw the line between new endpoints
  - To move a polygon, apply the transformation equation to coordinates of each vertex and regenerate the polygon using the new set of vertex coordinates

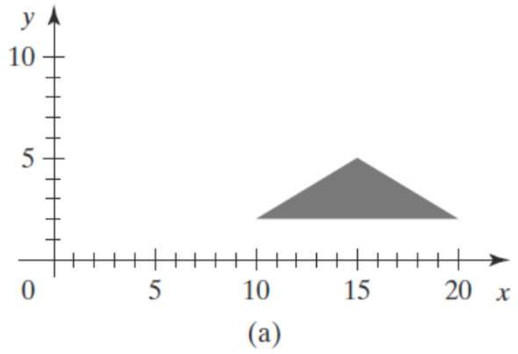
2D Translation Routine

```
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 Translation Routine



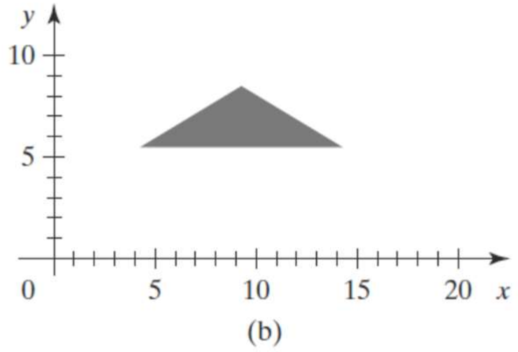


FIGURE 2
Moving a polygon from position (a) to position (b) with the translation vector (−5.50, 3.75).

Basic 2D Geometric Transformations (cont.)

- 2D Translation (Other Objects)
  - Similar methods are used to translate other objects.
  - To ***change the position of a circle or ellipse***
    - translate the center coordinates and redraw the figure in the new location.
  - For a ***spline curve***
    - translate the points that define the curve path and then reconstruct the curve sections between the new coordinate positions.

Basic 2D Geometric Transformations (cont.)

- 2D Rotation
  - Rotation axis
  - Rotation angle
  - rotation point or pivot point (x_r, y_r)

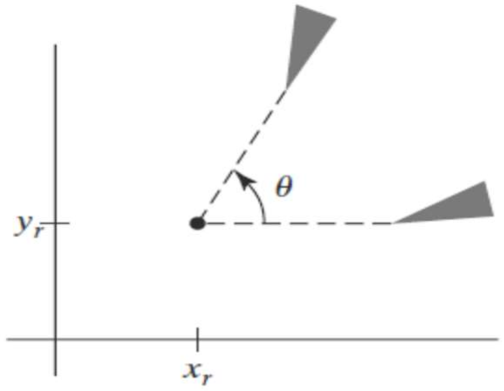
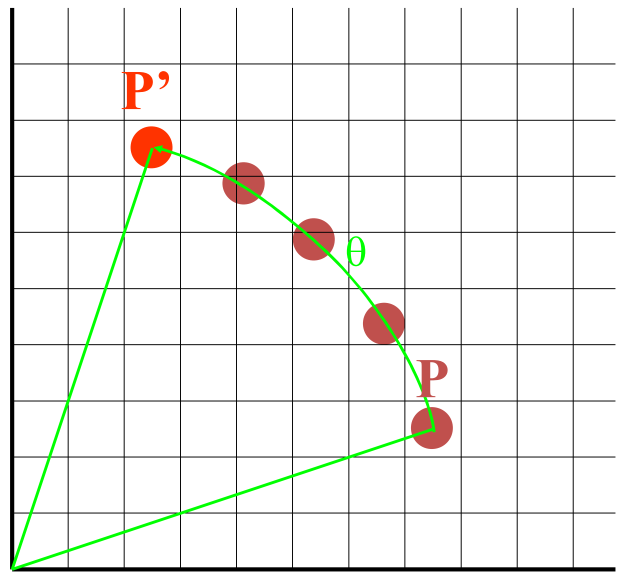


FIGURE 3
Rotation of an object through angle θ about the pivot point (x_r, y_r).

Rotation

A rotation repositions all points in an object along a circular path in the plane centered at the pivot point.

First, we'll assume the pivot is at the origin.



Rotation

- Review Trigonometry

$\Rightarrow \cos\phi = x/r$, $\sin\phi = y/r$

- $x = r.\cos\phi$, $y = r.\sin\phi$

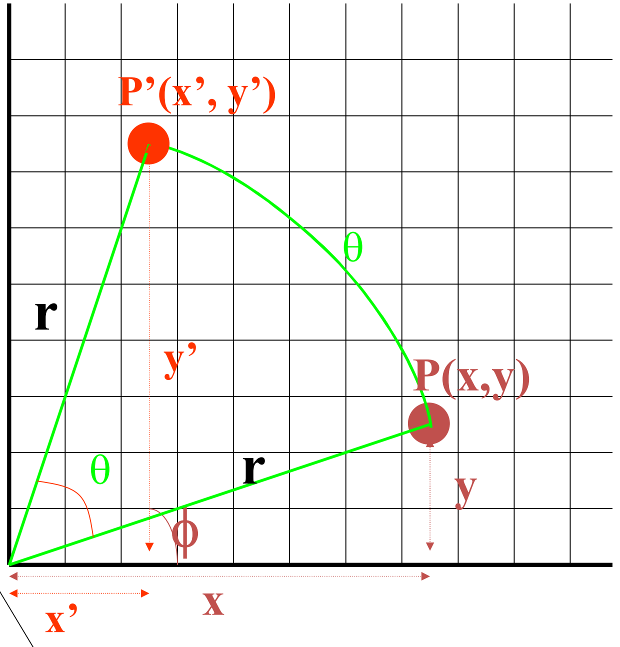
$\Rightarrow \cos(\phi+\theta) = x'/r$

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

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

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

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



Identity of Trigonometry

Rotation

- We can write the components:

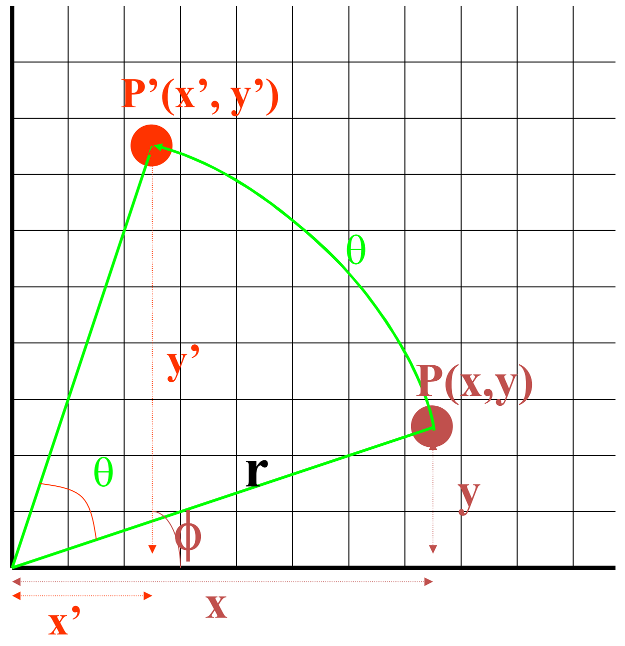
$$p'_x = p_x \cos\theta - p_y \sin\theta$$
$$p'_y = p_x \sin\theta + p_y \cos\theta$$

- or in matrix form:

$$P' = R \bullet P$$

- θ can be clockwise (-ve) or counterclockwise (+ve as our example).
- Rotation matrix

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



Basic 2D Geometric Transformations (cont.)

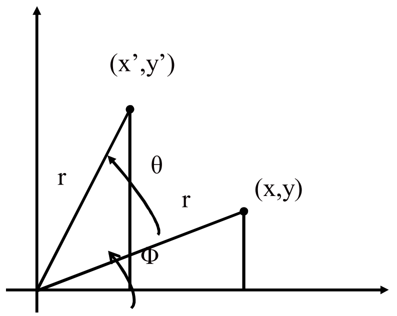
- **2D Rotation**
  - If θ is positive → counterclockwise rotation
  - If θ is negative → clockwise rotation
  - Remember:
    - $cos(a + b) = cos\ a\ cos\ b - sin\ a\ sin\ b$
    - $cos(a - b) = cos\ a\ sin\ b + sin\ a\ cos\ b$

Basic 2D Geometric Transformations (cont.)

2D Rotation

- At first, suppose the pivot point is at the origin

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



- x = r cos Φ, y = r sin Φ
- x'=x cos θ - y sin θ

  y'=x sin θ + y cos θ

Basic 2D Geometric Transformations (cont.)

- **2D Rotation**
  - P’=R·P

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

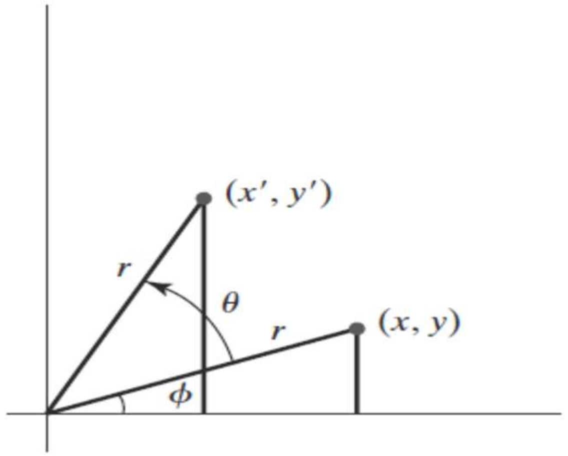


FIGURE 4
Rotation of a point from position (x, y) to position (x', y') through an angle θ relative to the coordinate origin. The original angular displacement of the point from the x axis is ϕ.

Basic 2D Geometric Transformations (cont.)

- **2D Rotation**
  - Rotation of a point about any specified position (x_r,y_r)

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

    $y' = y_r + (x - x_r)\sin\theta + (y - y_r)\cos\theta$
  - Rotations also move objects without deformation
  - A line is rotated by applying the rotation formula to each of the endpoints and redrawing the line between the new end points
  - A polygon is rotated by applying the rotation formula to each of the vertices and redrawing the polygon using new vertex coordinates

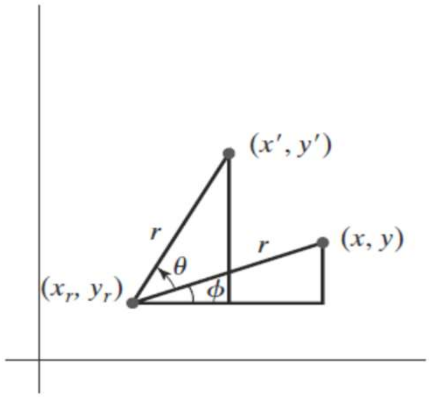


FIGURE 5 Rotating a point from position (x, y) to position (x', y') through an angle θ about rotation point (x_r, y_r).

2D Rotation Routine

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

  void rotatePolygon (wcPt2D * verts, GLint nVerts, wcPt2D pivPt, GLdouble theta)
  {
    wcPt2D * vertsRot;
    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 ( );
  }
```

Basic 2D Geometric Transformations (cont.)

- 2D Rotation (Other Objects)
  - ***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 Equations to each of the two line endpoints and redrawing the line between the new endpoint positions.
  - A ***polygon is rotated*** by displacing each vertex using the specified rotation angle and then regenerating the polygon using the new vertices.

Basic 2D Geometric Transformations (cont.)

- 2D Rotation (Other Objects)
  - ***rotate a curve*** by repositioning the defining points for the curve and then redrawing it.
  - ***A circle or an ellipse,*** for instance,
  - can be rotated about a noncentral pivot point by moving the center position through the arc that subtends the specified rotation angle.
  - In addition, we could ***rotate an ellipse*** about its center coordinates simply by rotating the major and minor axes.

Rotation

Example

Find the transformed point, P’, caused by rotating P= (5, 1) about the origin through an angle of 90°.

$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \bullet \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x\cdot\cos\theta - y\cdot\sin\theta \\ x\cdot\sin\theta + y\cdot\cos\theta \end{bmatrix}$$

$$= \begin{bmatrix} 5\cdot\cos 90 - 1\cdot\sin 90 \\ 5\cdot\sin 90 + 1\cdot\cos 90 \end{bmatrix}$$

$$= \begin{bmatrix} 5\cdot 0 - 1\cdot 1 \\ 5\cdot 1 + 1\cdot 0 \end{bmatrix}$$

$$= \begin{bmatrix} -1 \\ 5 \end{bmatrix}$$

Basic 2D Geometric Transformations (cont.)

2D Scaling

- Scaling is used to alter the size of an object
- Simple 2D scaling is performed by multiplying object positions (x, y) by scaling factors s_x and s_y

  $x' = x \cdot s_x$

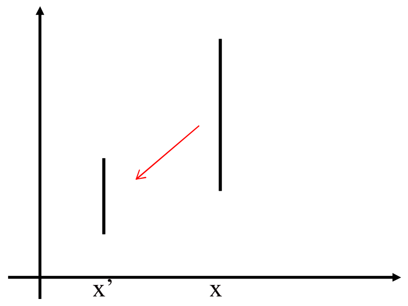
  $y' = y \cdot s_x$

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

or $P' = S \cdot P$

Basic 2D Geometric Transformations (cont.)

- **2D Scaling**
  - Any positive value can be used as scaling factor
    - Values less than 1 reduce the size of the object
    - Values greater than 1 enlarge the object
    - If scaling factor is 1 then the object stays unchanged
    - If $s_x = s_y$, we call it <u>uniform scaling</u>
    - If scaling factor <1, then the object moves closer to the origin and If scaling factor >1, then the object moves farther from the origin



Basic 2D Geometric Transformations (cont.)

2D Scaling

- Scaling changes the size of an object and involves two scale factors, S_x and S_y for the x- and y- coordinates respectively.
- Scales are about the origin.
- We can write the components:

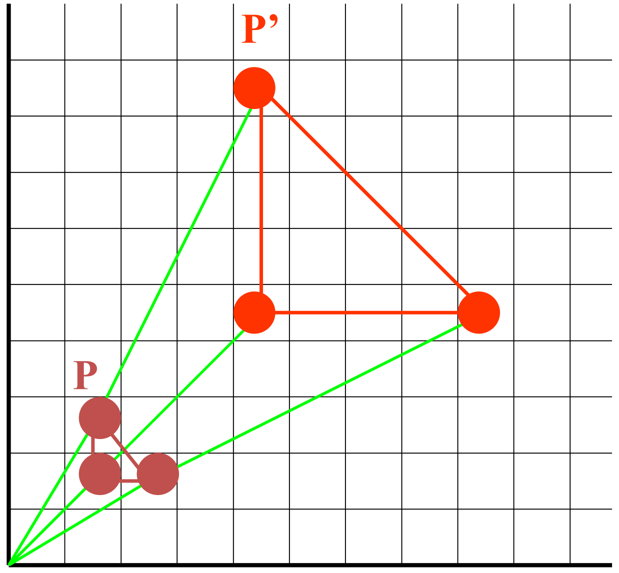
$$p'_x = s_x \bullet p_x$$
$$p'_y = s_y \bullet p_y$$

or in matrix form:

$$P' = S \bullet P$$

Scale matrix as:

$$S = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$$

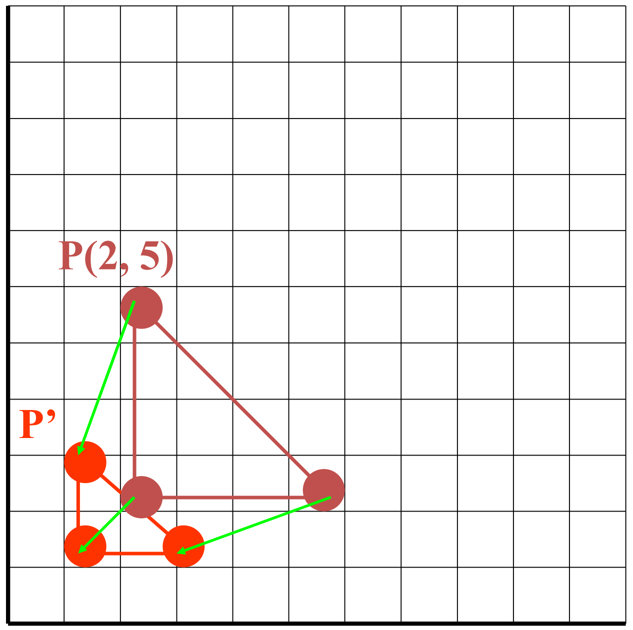


Basic 2D Geometric Transformations (cont.)

- **2D Scaling**

•If the scale factors are in between 0 and 1 ➔ the points will be moved closer to the origin ➔ the object will be smaller.

- Example :
  - •P(2, 5), Sx = 0.5, Sy = 0.5
  - •Find P' ?



Basic 2D Geometric Transformations (cont.)

▪ **2D Scaling**

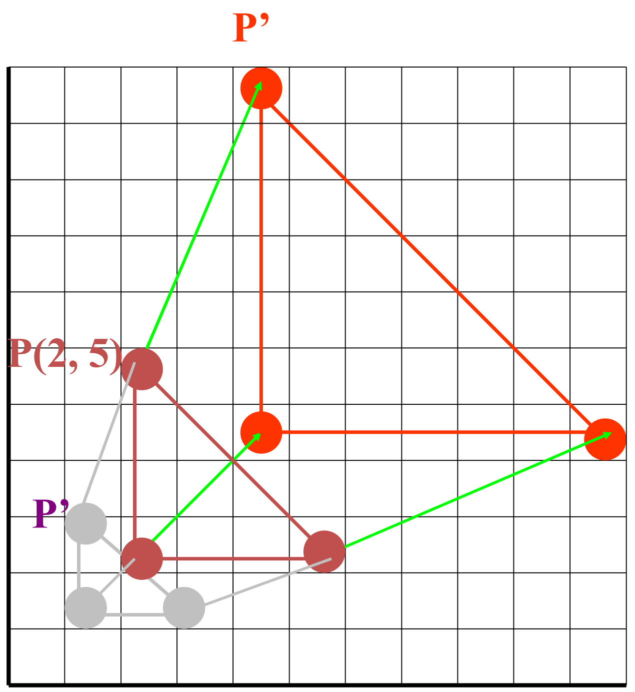
•If the scale factors are in between 0 and 1 ➔ the points will be moved closer to the origin ➔ the object will be smaller.

- Example :
  - P(2, 5), Sx = 0.5, Sy = 0.5
  - Find P' ?

- If the scale factors are larger than 1 ➔ the points will be moved away from the origin ➔ the object will be larger.

•Example :

- P(2, 5), Sx = 2, Sy = 2
- Find P' ?



Basic 2D Geometric Transformations (cont.)

- **2D Scaling**

•If the scale factors are the same, $S_x = S_y$ ➔ uniform scaling

• Only change in size (as previous example)

•If $S_x \neq S_y$ ➔ differential scaling.

•Change in size and shape

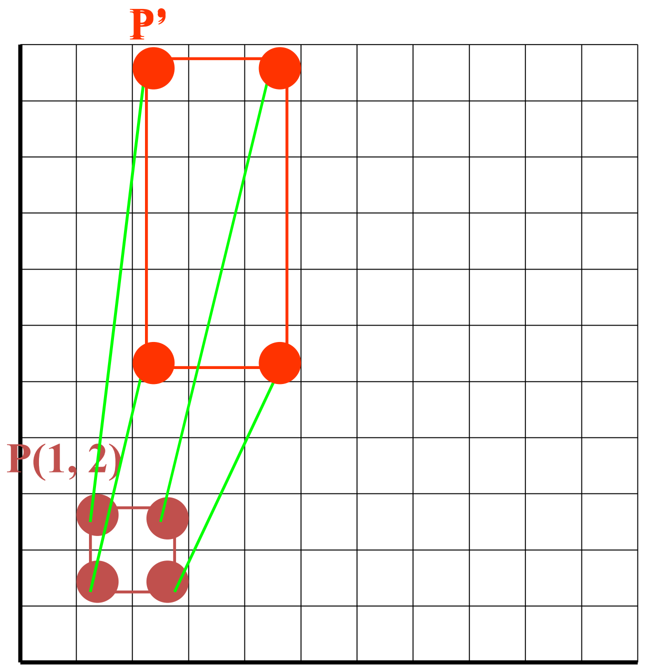
•Example : square → rectangle

- P(1, 3), $S_x = 2$, $S_y = 5$, P' ?

What does scaling by 1 do?

What is that matrix called?

What does scaling by a negative value do?



Basic 2D Geometric Transformations (cont.)

- **2D Scaling**
  - Why does scaling also reposition object?
  - Answer: See the matrix (multiplication)
  - $\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} * \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x * s_x + y * 0 \\ x * 0 + y * s_y \end{bmatrix}$

Basic 2D Geometric Transformations (cont.)

- **2D Scaling**
  - We can control the location of the scaled object by choosing a position called the **fixed point (x_f,y_f)**

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

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

    $y' = y \cdot s_y + y_f(1 - s_y)$
  - Polygons are scaled by applying the above formula to each vertex, then regenerating the polygon using the transformed vertices

2D Scaling Routine

```
 class wcPt2D {
      public:
         GLfloat x, y;
   };
   void scalePolygon (wcPt2D * verts, GLint nVerts, wcPt2D fixedPt, GLfloat sx,
GLfloat sy)
   {
      wcPt2D vertsNew;
      GLint k;

      for (k = 0; k < n; 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 < n; k++)
            glVertex2v (vertsNew [k].x, vertsNew [k].y);
      glEnd ( );
   }
```

Matrix Representations and Homogeneous Coordinates

- Many graphics applications involve sequences of geometric transformations
  - Animations
  - Design and picture construction applications
- We will now consider matrix representations of these operations
  - Sequences of transformations can be efficiently processed using matrices

Matrix Representations and Homogeneous Coordinates (cont..)

- $P' = M_1 \cdot P + M_2$
  - P and P' are column vectors
  - M_1 is a 2 by 2 array containing multiplicative factors
  - M_2 is a 2 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

Matrix Representations and Homogeneous Coordinates (cont..)

- To produce a sequence of operations, such as scaling followed by rotation then translation, we could calculate the transformed coordinates one step at a time
- A more efficient approach is to combine transformations, without calculating intermediate coordinate values

Matrix Representations and Homogeneous Coordinates (cont..)

- Multiplicative and translational terms for a 2D geometric transformation can be combined into a single matrix if we expand the representations to 3 by 3 matrices
  - We can use the third column for translation terms, and all transformation equations can be expressed as matrix multiplications

Matrix Representations and Homogeneous Coordinates (cont..)

- Expand each 2D coordinate *(x,y)* to three element representation *(x_h,y_h,h)* called **homogeneous coordinates**
- h is the **homogeneous parameter** such that
  - $x = x_h/h, \qquad y = y_h/h,$
- → infinite homogeneous representations for a point
- A convenient choice is to choose h = 1

Matrix Representations and Homogeneous Coordinates (cont..)

- 2D Translation Matrix

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

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

Matrix Representations and Homogeneous Coordinates (cont..)

- 2D Rotation Matrix

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

or, **P' = R**(θ)·**P**

Matrix Representations and Homogeneous Coordinates (cont..)

- 2D Scaling Matrix

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

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

Inverse Transformations

- 2D Inverse Translation Matrix

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

- By the way:

$$T^{-1} * T = I$$

Inverse Transformations (cont.)

- 2D Inverse Rotation Matrix

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

- And also: $R^{-1} * R = I$

Inverse Transformations (cont.)

- 2D Inverse Rotation Matrix:
  - If θ is negative → clockwise
  - In

$$R^{-1} * R = I$$

  - Only sine function is affected
  - Therefore we can say

$$R^{-1} = R^{T}$$

Inverse Transformations (cont.)

- 2D Inverse Scaling Matrix

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

- Of course:

$$S^{-1} * S = I$$

2D Composite Transformations

- We can setup a sequence of transformations as a **composite transformation matrix** by calculating the product of the individual transformations
- $P' = M_2 \cdot M_1 \cdot P$

  $= M \cdot P$

2D Composite Transformations (Cont..)

- Composite 2D Translations
  - If two successive translation are applied to a point P, then the final transformed location P' is calculated as:

$$\mathbf{P}' = \mathbf{T}(t_{x2}, t_{y2}) \cdot \mathbf{T}(t_{x1}, t_{y1}) \cdot \mathbf{P} = \mathbf{T}(t_{x_1} + t_{x_2}, t_{y_1} + t_{y_2}) \cdot \mathbf{P}$$

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

2D Composite Transformations (Cont..)

- Composite 2D Rotations

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

$$\begin{bmatrix} \cos\Theta_2 & -\sin\Theta_2 & 0 \\ \sin\Theta_2 & \cos\Theta_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\Theta_1 & -\sin\Theta_1 & 0 \\ \sin\Theta_1 & \cos\Theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos(\Theta_1 + \Theta_2) & -\sin(\Theta_1 + \Theta_2) & 0 \\ \sin(\Theta_1 + \Theta_2) & \cos(\Theta_1 + \Theta_2) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2D Composite Transformations (Cont..)

- Composite 2D Scaling

$$\mathbf{S}(s_{x_2}, s_{y_2}) \cdot \mathbf{S}(s_{x_1}, s_{y_1}) = \mathbf{S}(s_{x_1} \cdot s_{x_2}, s_{y_1} \cdot s_{y_2})$$

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

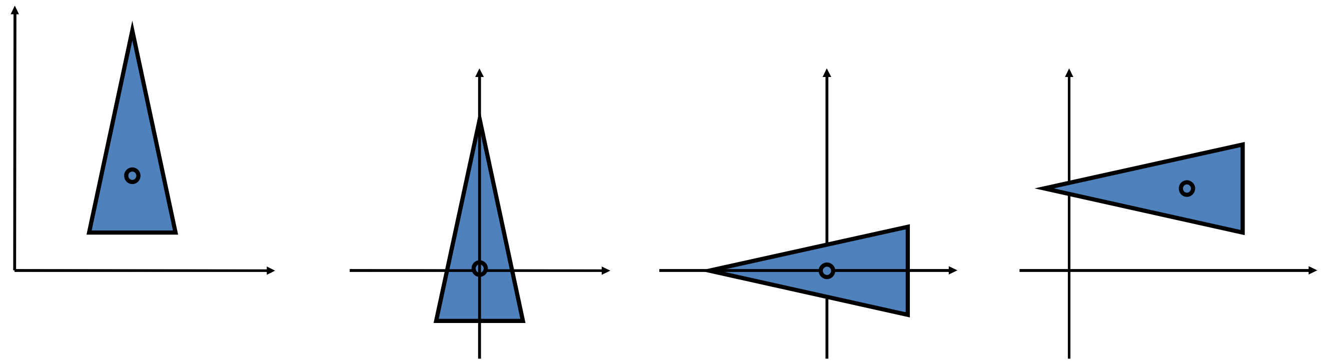
2D Composite Transformations (Cont..)

Note:

- Successive translations are additive
- Successive scalings are multiplicative
  - For example: If we triple the size of an object twice, the final size is nine (9) times the original

General Pivot Point Rotation

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



2D Composite Transformations (Cont..)

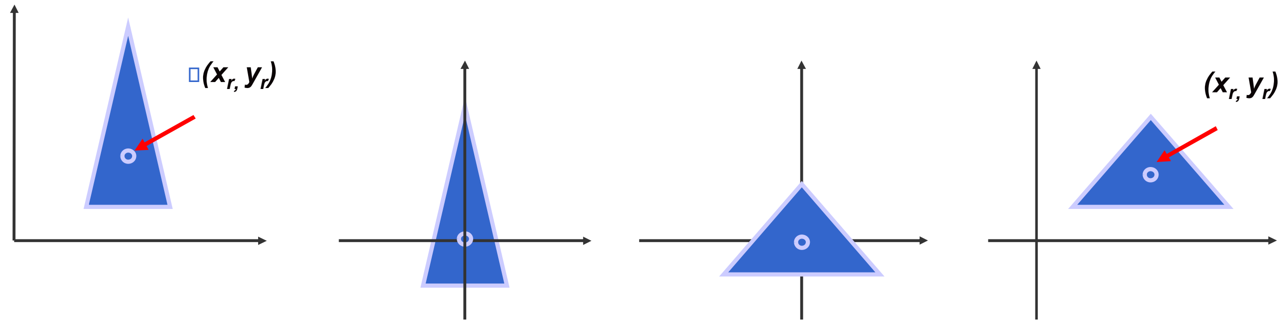
- General 2D Pivot-Point Rotation

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

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

General Fixed Point Scaling

- Steps:
  1. Translate the object so that the fixed point coincides with the coordinate origin.
  2. Scale the object about the origin.
  3. Translate the object so that the pivot point is returned to its original position.



General Fixed Point Scaling (Cont..)

- General 2D Fixed-Point Scaling:

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

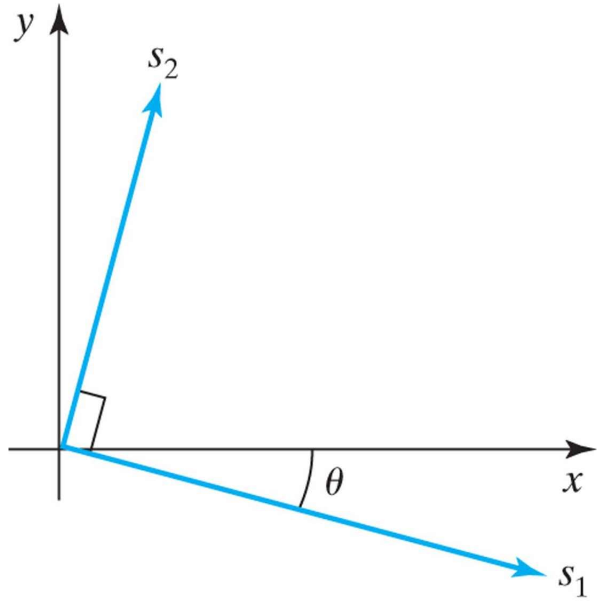
$$\mathbf{T}(x_f, y_f)\cdot\mathbf{S}(s_x, s_y)\cdot\mathbf{T}(-x_f, -y_f)=\mathbf{S}(x_f, y_f, s_x, s_y)$$

2D Composite Transformations (cont.)

- General 2D scaling directions:
  - Above: scaling parameters were along *x* and *y* directions
  - What about arbitrary directions?
  - Answer: See next slides

General 2D Scaling Directions

Scaling parameters s_1 and s_2 along orthogonal directions defined by the angular displacement θ.



General 2D Scaling Directions (Cont..)

- General procedure:
  1. Rotate so that directions coincides with *x* and *y* axes
  2. Apply scaling transformation $S(s_1, s_2)$
  3. Rotate back
- The composite matrix:

$$R^{-1}(\Theta) * S(s_1, s_2) * R(\Theta) = \begin{bmatrix} s_1 \cos^2\Theta + s_2 \sin^2\Theta & (s_2 - s_1)\cos\Theta \sin\Theta & 0 \\ (s_2 - s_1)\cos\Theta \sin\Theta & s_1 \sin^2\Theta + s_2 \cos^2\Theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

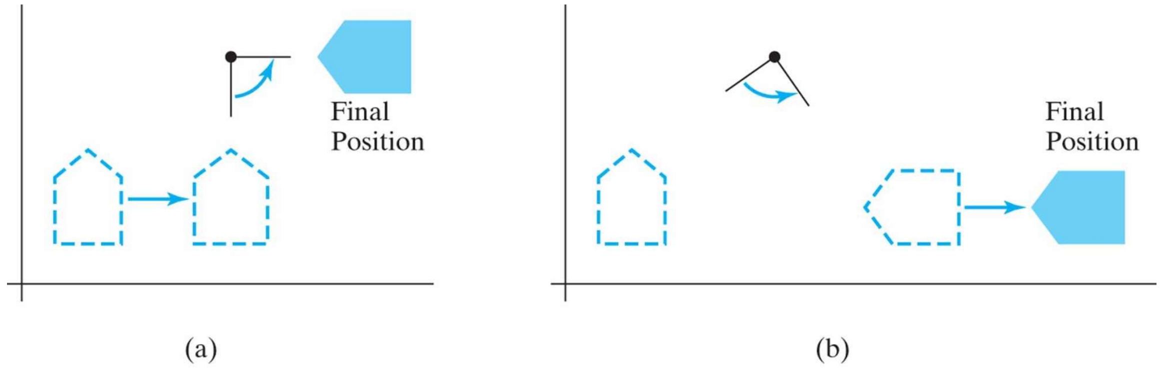
2D Composite Transformations (cont.)

- Matrix Concatenation Properties:
  - Matrix multiplication is associative !
    - $M_3 \cdot M_2 \cdot M_1 = (M_3 \cdot M_2) \cdot M_1 = M_3 \cdot (M_2 \cdot M_1)$
    - A composite matrix can be created by multiplying left-to-right (premultiplication) or right-to-left (postmultiplication)
  - Matrix multiplication is ***not*** commutative !
    - $M_2 \cdot M_1 \neq M_1 \cdot M_2$

2D Composite Transformations (cont.)

- Matrix Concatenation Properties:
  - But:
    - Two successive rotations
    - Two successive translations
    - Two successive scalings
  - ***are*** commutative!
- Prove

Reversing the ordering



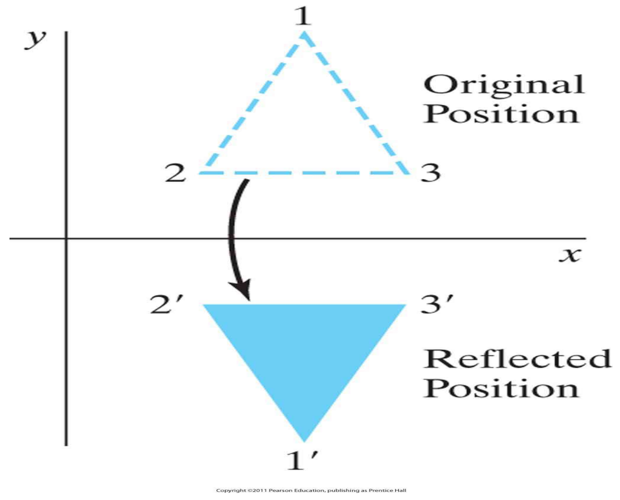
in which a sequence of transformations is performed may affect the transformed position of an object.

In (a), an object is first translated in the x direction, then rotated counterclockwise through an angle of 45° .

In (b), the object is first rotated 45° counterclockwise, then translated in the *x* direction

Other 2D Transformations

- **Reflection**
  - Transformation that produces a mirror image of an object


y
1
Original Position
2
3
x
2′
3′
Reflected Position
1′
Copyright ©2011 Pearson Education, publishing as Prentice Hall

Other 2D Transformations(Cont..)

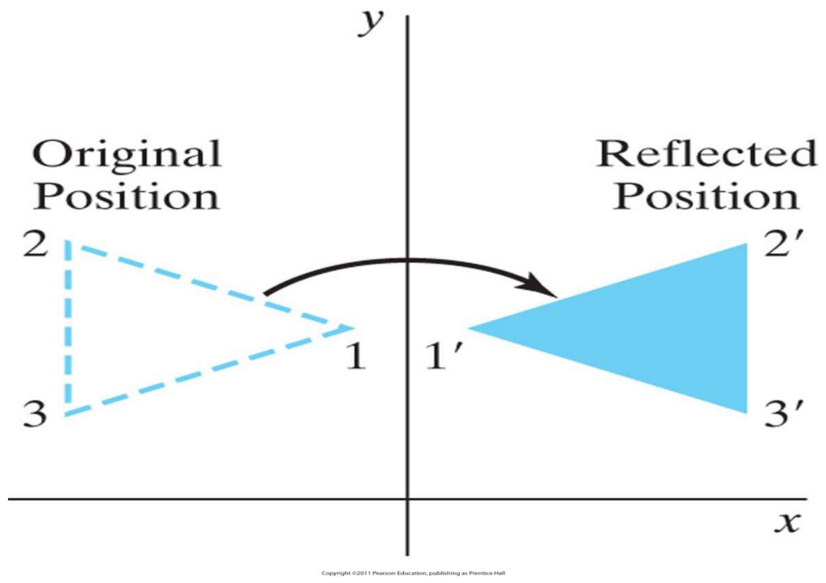
- **Reflection**
  - Image is generated relative to an axis of reflection by rotating the object 180° about the reflection axis
  - Reflection about the line y=0 (the x axis) (previous slide)

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

Other 2D Transformations(Cont..)

- **Reflection**
  - Reflection about the line x=0 (the y axis)

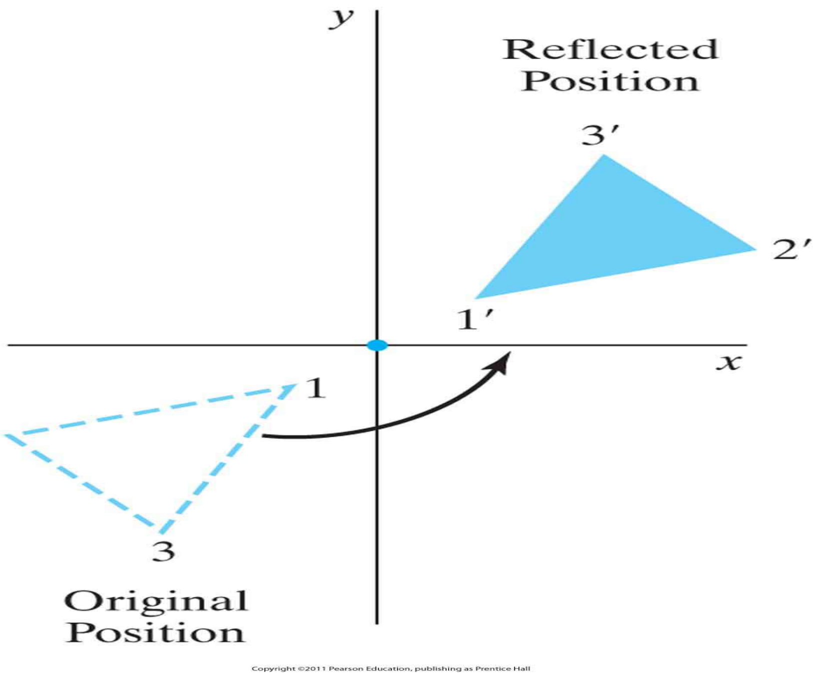
$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$


y
Original Position
Reflected Position
2
2′
1
1′
3
3′
x
Copyright ©2011 Pearson Education, publishing as Prentice Hall

Other 2D Transformations(Cont..)

- **Reflection about the origin**

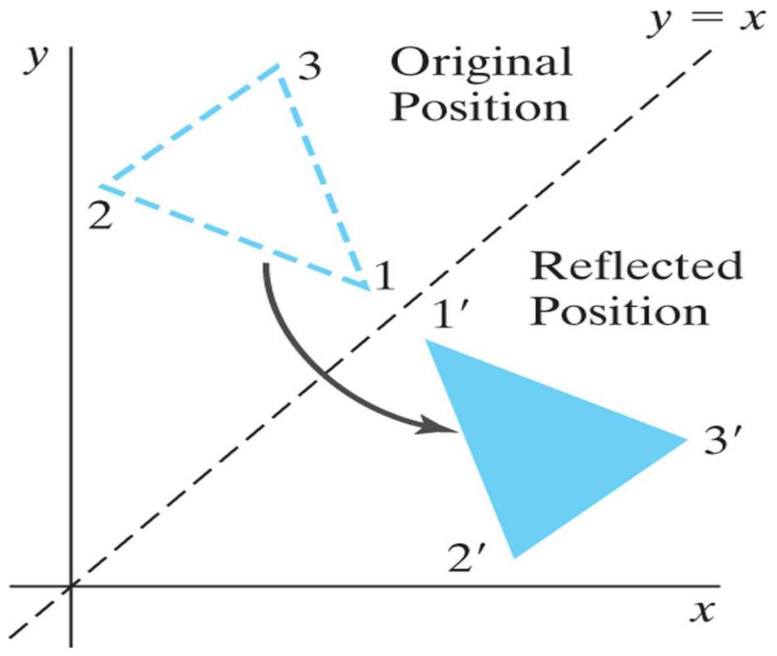
$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$


y
Reflected Position
3′
2′
1′
x
1
2
3
Original Position
Copyright ©2011 Pearson Education, publishing as Prentice Hall

Other 2D Transformations(Cont..)

- **Reflection about the line *y=x***

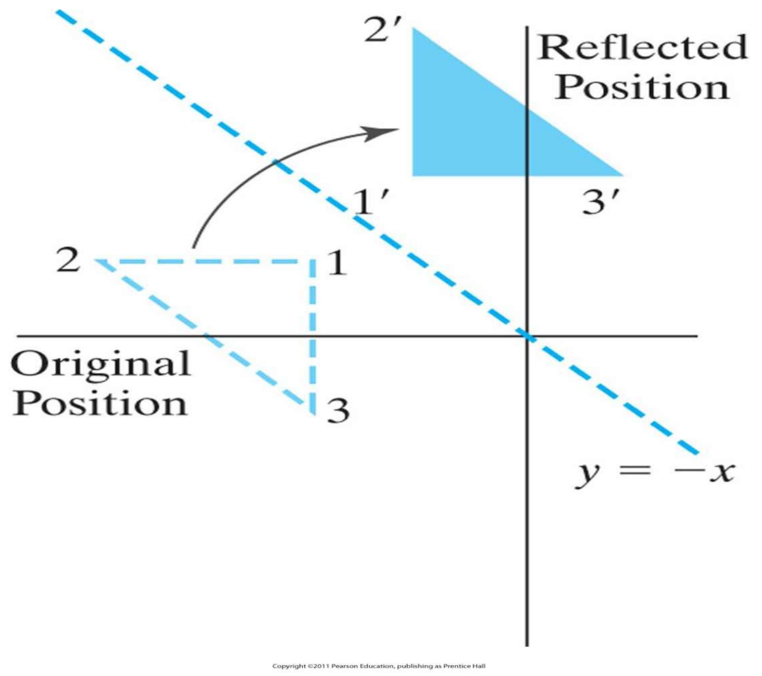
$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



Other 2D Transformations(Cont..)

- **Reflection about the line *y=-x***

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


2′
Reflected Position
1′
3′
2
1
Original Position
3
$y = -x$
Copyright ©2011 Pearson Education, publishing as Prentice Hall

Other 2D Transformations(Contd..)

- **Reflection**
  - Describe the transformation M_L which reflects an object about a line L.

- Let line L in Fig(RHS) have a y intercept (0, *b)* and an angle of inclination Θ ° (with respect to the x axis).
- We reduce the description to known transformations:

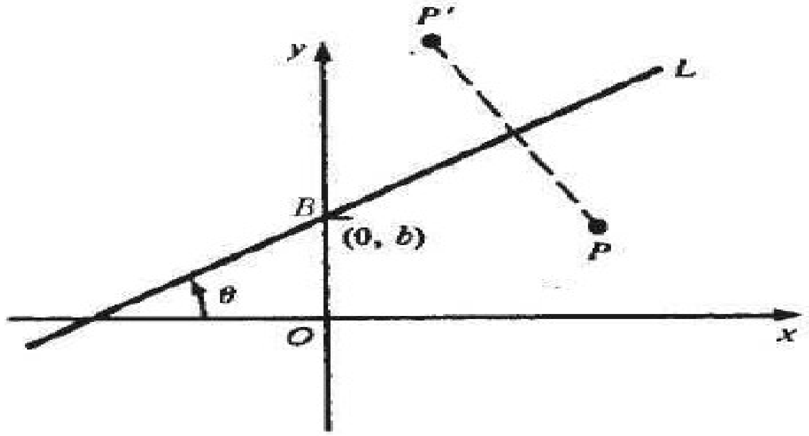
Other 2D Transformations(Contd..)

1. Translate the intersection point B to the origin.
2. Rotate by $-\theta^\circ$ so that line L aligns with the x axis.
3. Mirror-reflect about the x axis.
4. Rotate back by θ°.
5. Translate B back to $(0, b)$.

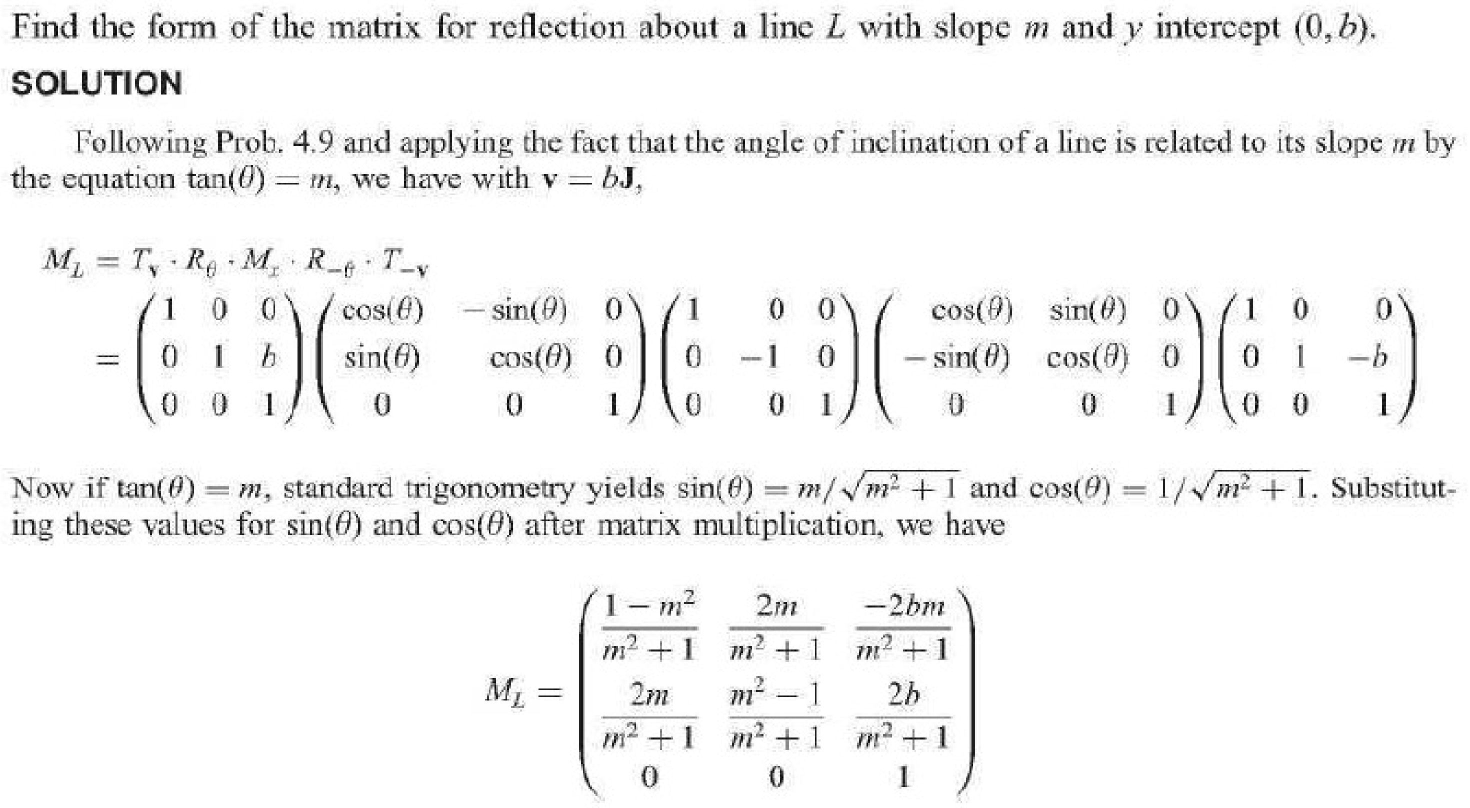
In transformation notation, we have

$$M_L = T_{\mathbf{v}} \cdot R_\theta \cdot M_x \cdot R_{-\theta} \cdot T_{-\mathbf{v}}$$

where $\mathbf{v} = b\mathbf{J}$.



Other 2D Transformations (Contd..)



Find the form of the matrix for reflection about a line L with slope m and y intercept $(0, b)$.

SOLUTION

Following Prob. 4.9 and applying the fact that the angle of inclination of a line is related to its slope m by the equation $\tan(\theta) = m$, we have with $\mathbf{v} = b\mathbf{J}$,

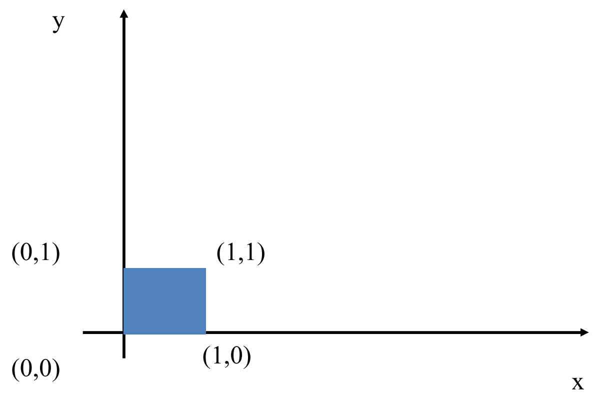
$$\begin{aligned} M_L &= T_{\mathbf{v}} \cdot R_\theta \cdot M_x \cdot R_{-\theta} \cdot T_{-\mathbf{v}} \\ &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & b \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & -b \\ 0 & 0 & 1 \end{pmatrix} \end{aligned}$$

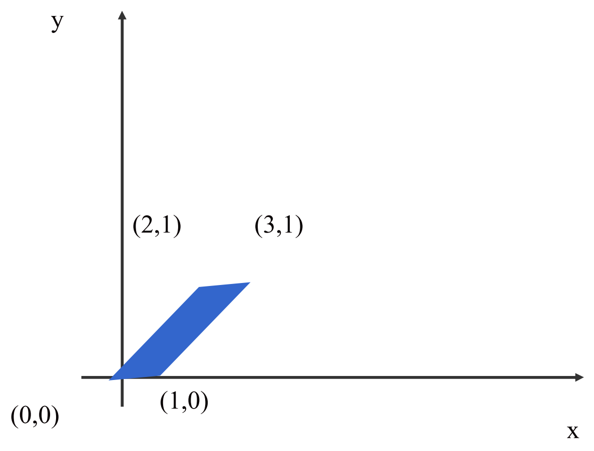
Now if $\tan(\theta) = m$, standard trigonometry yields $\sin(\theta) = m/\sqrt{m^2+1}$ and $\cos(\theta) = 1/\sqrt{m^2+1}$. Substituting these values for $\sin(\theta)$ and $\cos(\theta)$ after matrix multiplication, we have

$$M_L = \begin{pmatrix} \dfrac{1-m^2}{m^2+1} & \dfrac{2m}{m^2+1} & \dfrac{-2bm}{m^2+1} \\ \dfrac{2m}{m^2+1} & \dfrac{m^2-1}{m^2+1} & \dfrac{2b}{m^2+1} \\ 0 & 0 & 1 \end{pmatrix}$$

Other 2D Transformations(Cont..)

- **Shear**
  - Transformation that distorts the shape of an object such that the transformed shape appears as the object was composed of internal layers that had been caused to slide over each other.





Other 2D Transformations(Cont..)

- **Shear**
  - An x-direction shear relative to the x axis

$$\begin{bmatrix} 1 & sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \begin{aligned} x' &= x + sh_x \cdot y \\ y' &= y \end{aligned}$$

  - An y-direction shear relative to the y axis

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

Other 2D Transformations(Cont..)

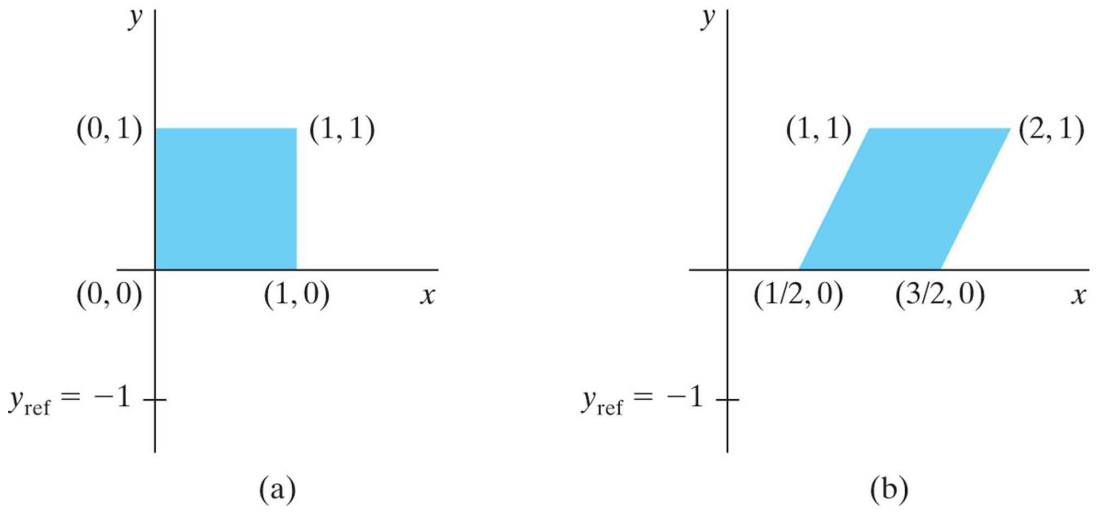
- **Shear**
  - x-direction shear relative to other reference lines

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

$$x' = x + sh_x * (y - y_{ref})$$

$$y' = y$$

Example



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

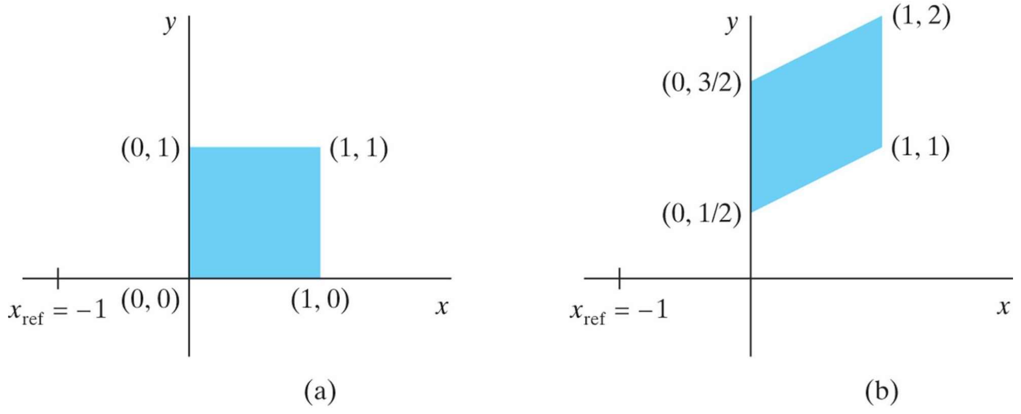
Other 2D Transformations(Cont..)

- **Shear**
  - y-direction shear relative to the line $x = x_{ref}$

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

$$x' = x$$
$$y' = x + sh_y * \left(x - x_{ref}\right)$$

Example



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 (previous slide)

Transformation Between Coordinate Systems

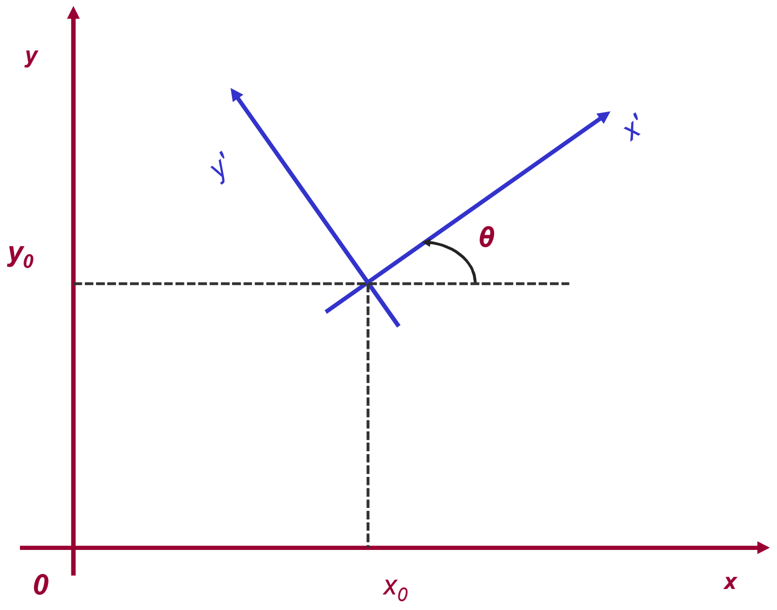
- Individual objects may be defined in their local cartesian reference system.
- The local coordinates must be transformed to position the objects within the scene coordinate system.

Steps for coordinate transformation

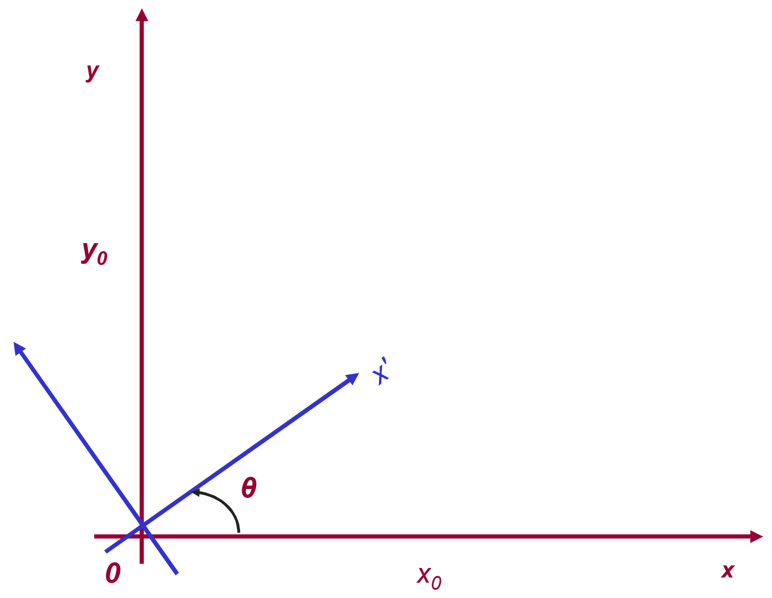
1. Translate so that the origin (x_0, y_0) of the x'-y' system is moved to the origin of the x-y system.

2.Rotate the x' axis on to the axis x.

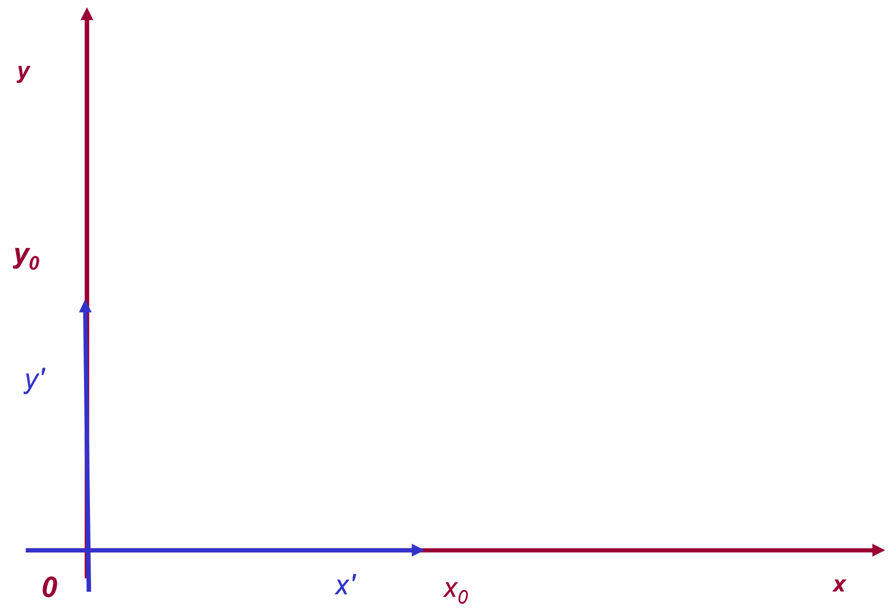
Transformation Between Coordinate Systems (contd..)



Transformation Between Coordinate Systems (contd..)



Transformation Between Coordinate Systems (contd..)



Transformation Between Coordinate Systems (contd..)

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

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

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

Transformation Between Coordinate Systems (contd..)

An alternative method:

-Specify a vector **V** that indicates the direction for the positive y′ axis. Let

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

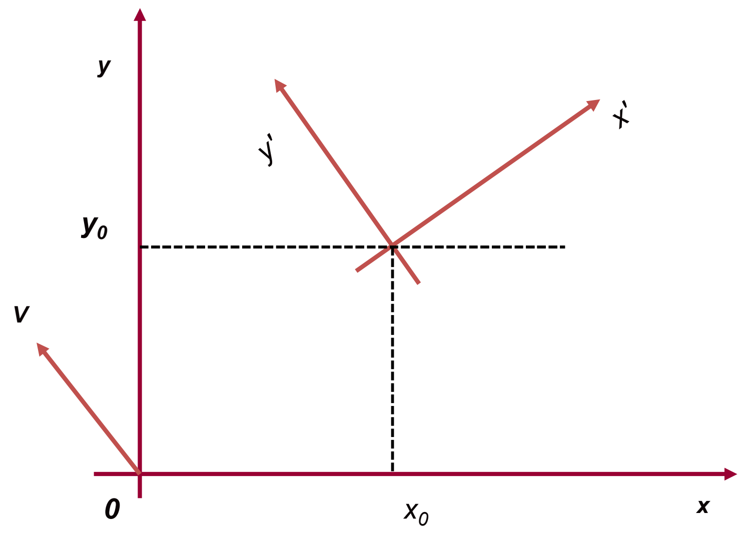
- Obtain the unit vector **u**=(u_x ,u_y) along the x′ axis by rotating **v** 90^0 clockwise.

Transformation Between Coordinate Systems (contd..)

- Elements of any rotation matrix can be expressed as elements of orthogonal unit vectors. That is, the rotation matrix can be written as

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

Transformation Between Coordinate Systems (contd..)



Transformation in 2D ***(Summary)***

2D Translation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \quad \mathbf{P}' = \mathbf{T}(t_x, t_y) \cdot \mathbf{P}$$

2D Rotation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \quad \mathbf{P}' = \mathbf{R}(\theta) \cdot \mathbf{P}$$

2D Scaling

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \quad \mathbf{P}' = \mathbf{S}(S_x, S_y) \cdot \mathbf{P}$$

Transformation in 2D *(Summary)—contd..*

Inverse transformations:

$$\mathbf{T}^{-1}=\begin{bmatrix}1 & 0 & -t_x\\ 0 & 1 & -t_y\\ 0 & 0 & 1\end{bmatrix},\ \mathbf{R}^{-1}=\begin{bmatrix}\cos\theta & \sin\theta & 0\\ -\sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix},\ \mathbf{S}^{-1}=\begin{bmatrix}1/S_x & 0 & 0\\ 0 & 1/S_y & 0\\ 0 & 0 & 1\end{bmatrix}$$

Composite transformations:

$$\mathbf{P}'=\mathbf{M}_2\left(\mathbf{M}_1\cdot\mathbf{P}\right)=\left(\mathbf{M}_2\cdot\mathbf{M}_1\right)\cdot\mathbf{P}=\mathbf{M}\cdot\mathbf{P}$$

$$\mathbf{P}'=\mathbf{T}\left(t_{2x},t_{2y}\right)\left\{\mathbf{T}\left(t_{1x},t_{1y}\right)\cdot\mathbf{P}\right\}=\left\{\mathbf{T}\left(t_{2x},t_{2y}\right)\cdot\mathbf{T}\left(t_{1x},t_{1y}\right)\right\}\cdot\mathbf{P}$$

Composite translations:

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

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

Transformation in 2D ***(Summary)-- contd..***

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

Composite Rotations:

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

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

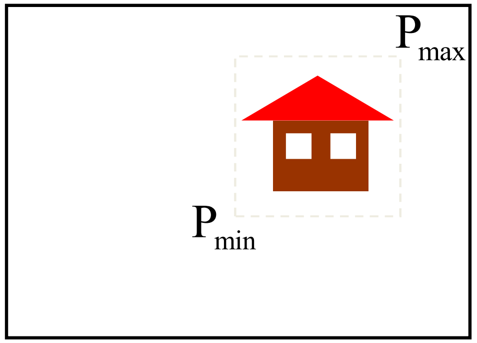
$$\begin{bmatrix} S_{2x} & 0 & 0 \\ 0 & S_{2y} & 0 \\ 0 & 0 & 1 \end{bmatrix}\cdot\begin{bmatrix} S_{1x} & 0 & 0 \\ 0 & S_{1y} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} S_{1x}\cdot S_{2x} & 0 & 0 \\ 0 & S_{1y}\cdot S_{2y} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

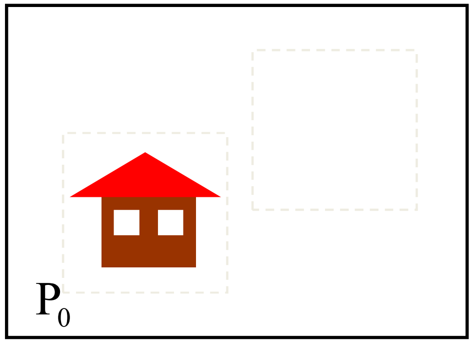
Composite Scaling:

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

Geometric Transformations by Rasterization

- The transformed shape needs to be filled.
    - A whole scan-line filling is usually in order.
- However, simple transformations can save new filling by manipulating blocks in the frame buffer.



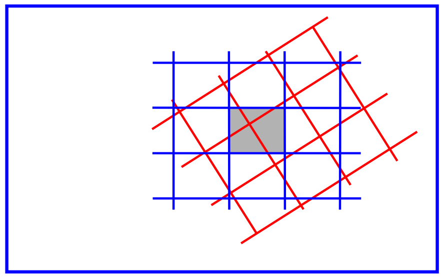


Translation:

Move block of pixels of frame buffer into new destination.

Geometric Transformations by Rasterization

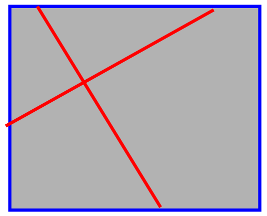
$$\begin{tikzcd}
\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix} \arrow[r, leftrightarrow, "90^\circ \text{ counterclockwise rotation}"] & \begin{bmatrix} 3 & 6 & 9 & 12 \\ 2 & 5 & 8 & 11 \\ 1 & 4 & 7 & 10 \end{bmatrix} & \begin{bmatrix} 12 & 11 & 10 \\ 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix} \arrow[ll, leftrightarrow, bend left, "180^\circ \text{ rotation}"]
\end{tikzcd}$$



Rotated pixel block

Destination pixel array

RGB of destination pixel can be determined by averaging rotated ones (as antialiasing)



OpenGL Geometric Transformation Functions

- A separate function is available for each of the basic geometric transformations

AND

- All transformations are specified in **three** dimensions
- Why?
- Answer: Remember; OpenGL was developed as 3D library
- But how to perform 2D transformations?
- Answer: Set $z = 0$

OpenGL Geometric Transformation Functions(Cont..)

- Translation
  - glTranslate* (tx, ty, tz);
    - * is either f or d
    - tx, ty and tz are any real number
    - For 2D, set tz=0.0
- Rotation
  - glRotate* (theta, vx, vy, vz);
    - * is either f or d
    - theta is rotation angle in degrees (internally converted to radian)
    - Vector v=(vx, vy, vz) defines the orientation for a rotation axis that passes through the coordinate origin
    - For 2D, set vz=1.0 and vx=vy=0.0

OpenGL Geometric Transformation Functions(Cont..)

- Scaling
  - glScale* (sx, sy, sz);
    - * is either f or d
    - sx, sy and sz are any real number
    - Negative values generate **reflection**
    - Zero values can cause error because inverse matrix cannot be calculated
- All routines construct a 4x4 transformation matrix
- OpenGL uses composite matrices
  - Be careful with the order

OpenGL Matrix Operations

- glMatrixMode(.);
  - Projection Mode: Determines how the scene is projected onto the screen
  - Modelview Mode: Used for storing and combining geometric transformations
  - Texture Mode: Used for mapping texture patterns to surfaces
  - Color Mode: Used to convert from one color mode to another

OpenGL Matrix Operations

- Modelview matrix, used to store and combine geometric transformations
  - glMatrixMode(GL_MODELVIEW);
- A call to a transformation routine generates a matrix that is multiplied by the current matrix
- To assign the identity matrix to the current matrix
  - glLoadIdentity();

OpenGL Matrix Operations

- Alternatively:
  - glLoadMatrix* (elements16);
  - To assign other values to the elements of the current matrix
  - In column-major order:
    - First four elements in first column
    - Second four elements in second column
    - Third four elements in third column
    - Fourth four elements in fourth column

OpenGL Matrix Operations

- Concatenating a specified matrix with current matrix:
  - glMultMatrix* (otherElements16);
  - Current matrix is postmultiplied (right-to-left) by the specified matrix

Note:

- Matrix notation m_{jk} means:
  - In OpenGL: $j \rightarrow$ column, $k \rightarrow$ row
  - In mathematics: $j \rightarrow$ row, $k \rightarrow$ column

OpenGL Matrix Stacks

- OpenGL maintains a matrix stack for transformations
- Initially the model view stack contains only the identity matrix

OpenGL Transformation Routines

- For example, assume we want to do in the following order:
  - translate by +2, -3, +4,
  - rotate by 45^0 around axis formed between origin and 1, 1, 1
  - scale with respect to the origin by 2 in each direction.
- Our code would be

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();                    //start with identity
glScalef(2.0,2.0,2.0);   //Note: Start with the LAST operation
glRotatef(45.0,1.0,1.0,1.0);
glTranslatef(2.0,-3.0, 4.0); //End with the FIRST operation
```

OpenGL Transformation Functions

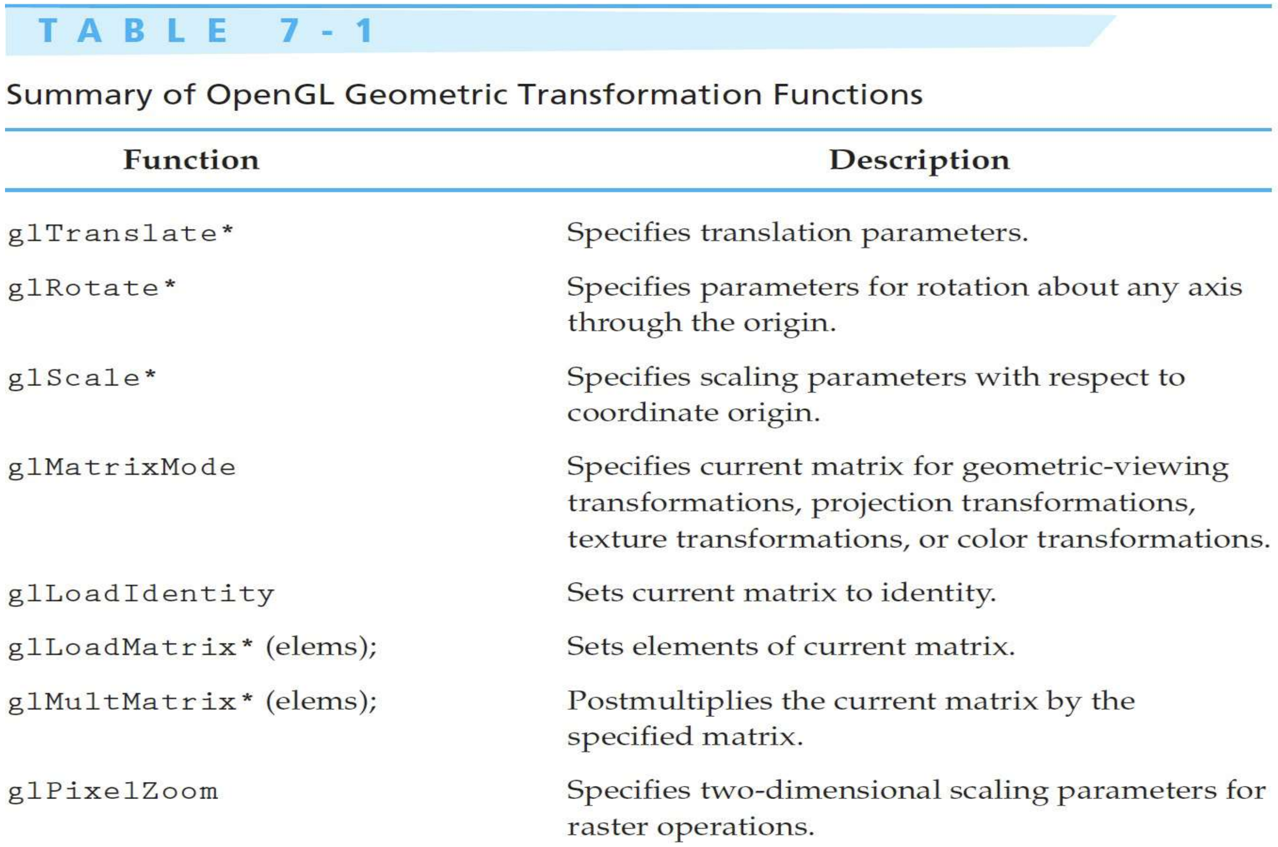


TABLE 7-1

Summary of OpenGL Geometric Transformation Functions

Function	Description
`glTranslate*`	Specifies translation parameters.
`glRotate*`	Specifies parameters for rotation about any axis through the origin.
`glScale*`	Specifies scaling parameters with respect to coordinate origin.
`glMatrixMode`	Specifies current matrix for geometric-viewing transformations, projection transformations, texture transformations, or color transformations.
`glLoadIdentity`	Sets current matrix to identity.
`glLoadMatrix*` (elems);	Sets elements of current matrix.
`glMultMatrix*` (elems);	Postmultiplies the current matrix by the specified matrix.
`glPixelZoom`	Specifies two-dimensional scaling parameters for raster operations.

Thank You