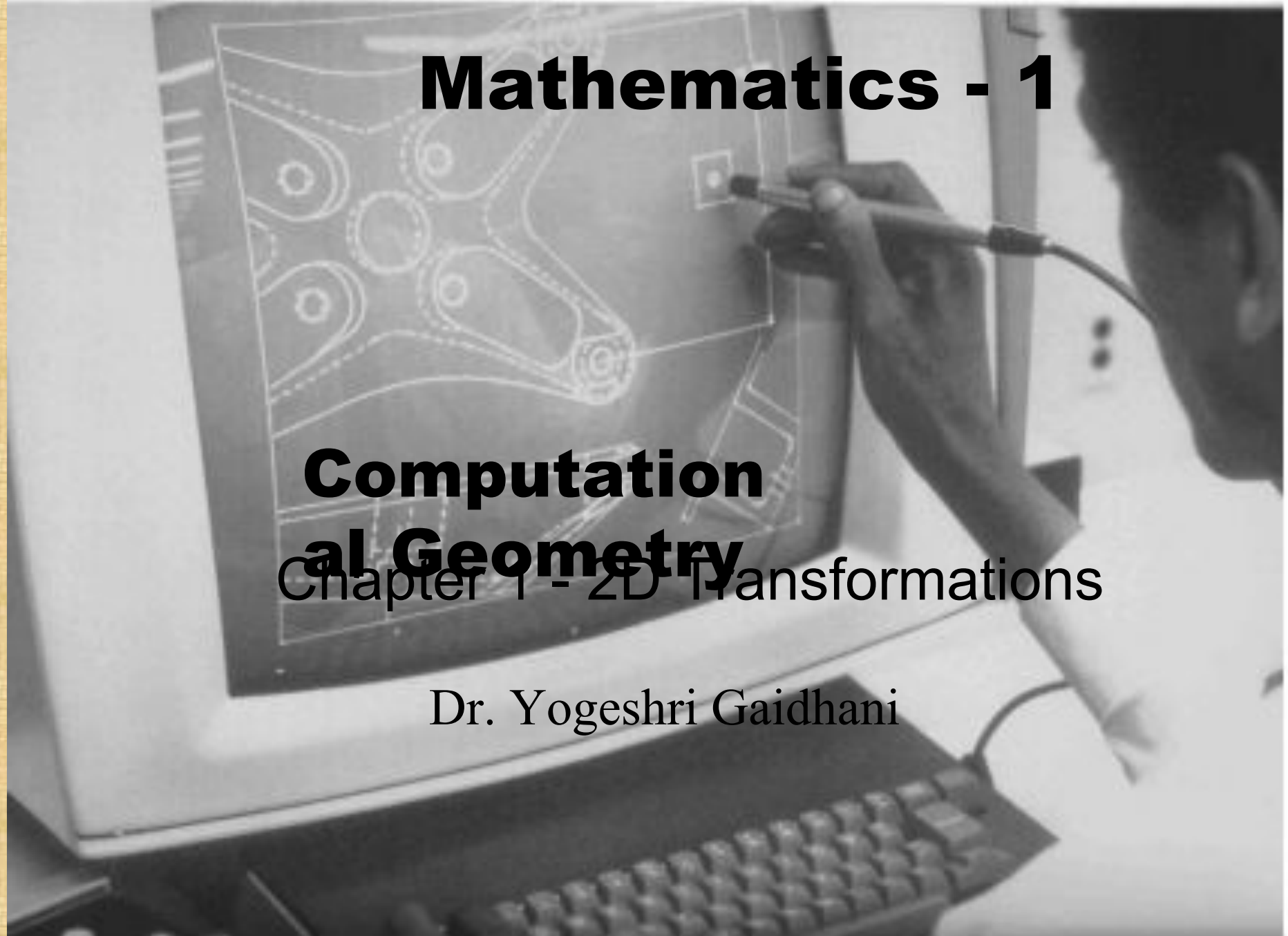# Mathematics - 1

# Computational Geometry

## Chapter 1 - 2D Transformations

Dr. Yogeshri Gaidhani

# Objectives

- Basic 2D Transformations (rigid-body transformations):
  - Translation
  - Rotation
  - Scaling
- Homogenenous Representations and Coordinates
- 2D Composite Transformations

# **Objectives (cont.)**

- Other Transformations:
  - Reflection
  - Shearing
- Raster Methods for Transformations and OpenGL (For your ref. NOT part of syllabus)
- Transformations between 2D Coordinate Systems and OpenGL (For your ref. NOT part of syllabus)

# 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

# Lets try to understand what we want to do -

- Computational Geometry – Mathematical elements of computer graphics
- Animation – One frame – Drawing curves
- Simplest curves – conic sections, line segment
- Line segment – points
- Basic unit – point

# Lets try to understand what we want to do -

- A point – P=[x  y] – a matrix
- Transformations like rotation, translation, resizing are also represented using matrices
- Transformation matrix acts on object matrix → Transformed matrix (object)
- 2 D Transformation matrix is a 2 x 2 matrix

$$T= \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

- P' = PT = [x y] $\begin{bmatrix} a & b \end{bmatrix}$ = [ax+cy  bx+dy]

# Lets try to understand what we want to do -

- A segment is defined by two end points, say A and B

- Let A = [x1, y1] , B=[x2, y2]

- Matrix of segment AB, $X = \begin{bmatrix} A \\ B \end{bmatrix} = \begin{bmatrix} x1 & y1 \\ x2 & y2 \end{bmatrix}$

- Similarly $\triangle$ ABC, $X = \begin{bmatrix} A \\ B \\ C \end{bmatrix} = \begin{bmatrix} x1 & y1 \\ x2 & y2 \\ x3 & y3 \end{bmatrix}$

# Lets try to understand what we want to do -

- Every polygon is a matrix
- Every curve can be approximated by sequence of points
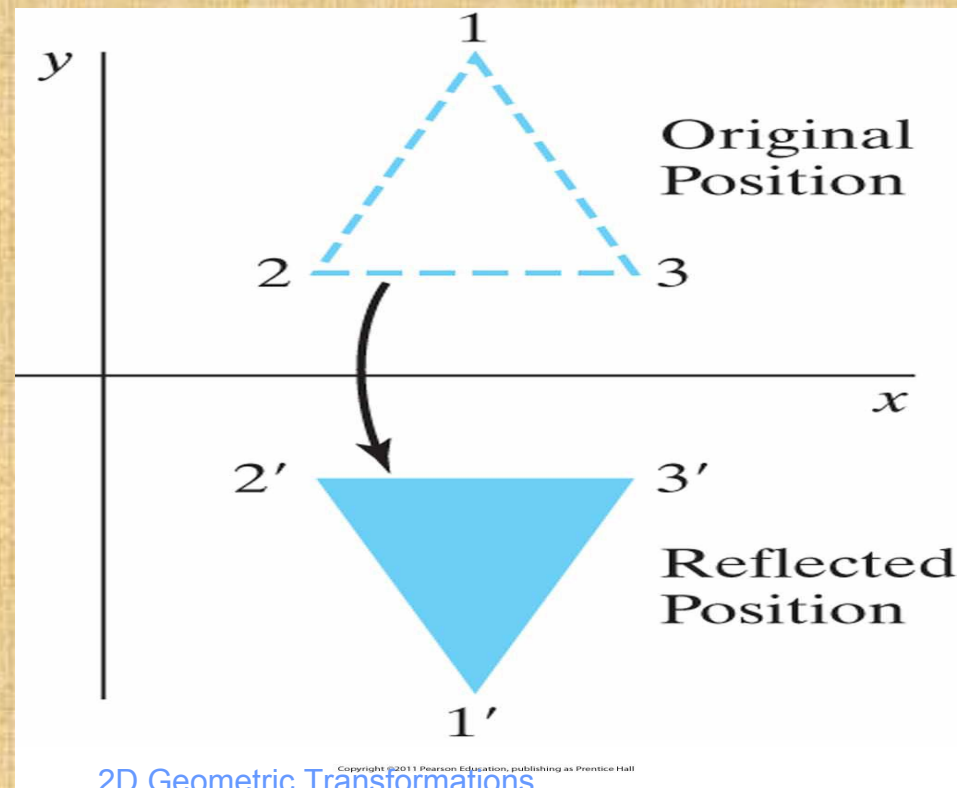- **In short, here everything is a matrix.**

# Identity Transformation

- $P = [x\ y]$, $T= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

- $P' = PT = P$

# Reflection

- Reflection through X-axis
  - Transformation that produces a mirror image of an object through X-axis
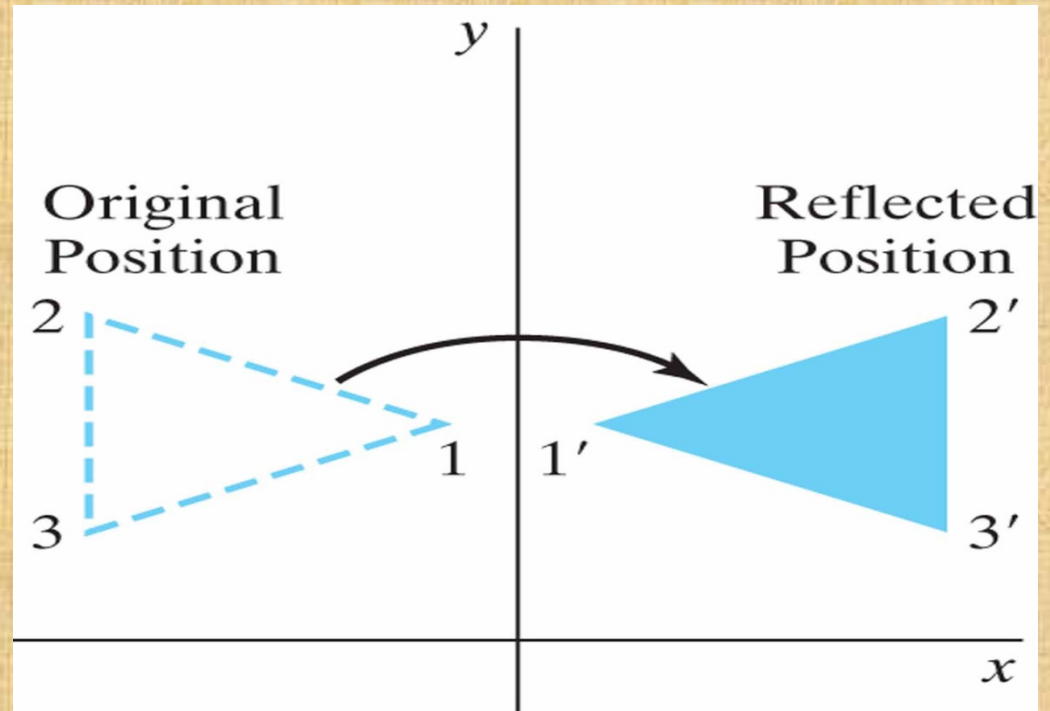
# Reflection through X-axis

- ☐ Reflection about the line y=0 (the x axis) (previous slide).
- ☐ Y coordinate of a point changes the sign and X coordinate remains unchanged.

- ☐ $T = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$

- ☐ Let P= [3   4]. P' = PT = [3  -4]

# Reflection through Y-axis

- Reflection through the y axis
  - Reflection about the line x=0 → x coordinate changes the sign



Original Position
Reflected Position

2
1    1'
3
2'
3'

y

x

Copyright ©2011 Pearson Education, publishing as Prentice Hall

# Reflection about the origin

■ Reflection about the origin ⮕ Both x and y coordinate change the signs

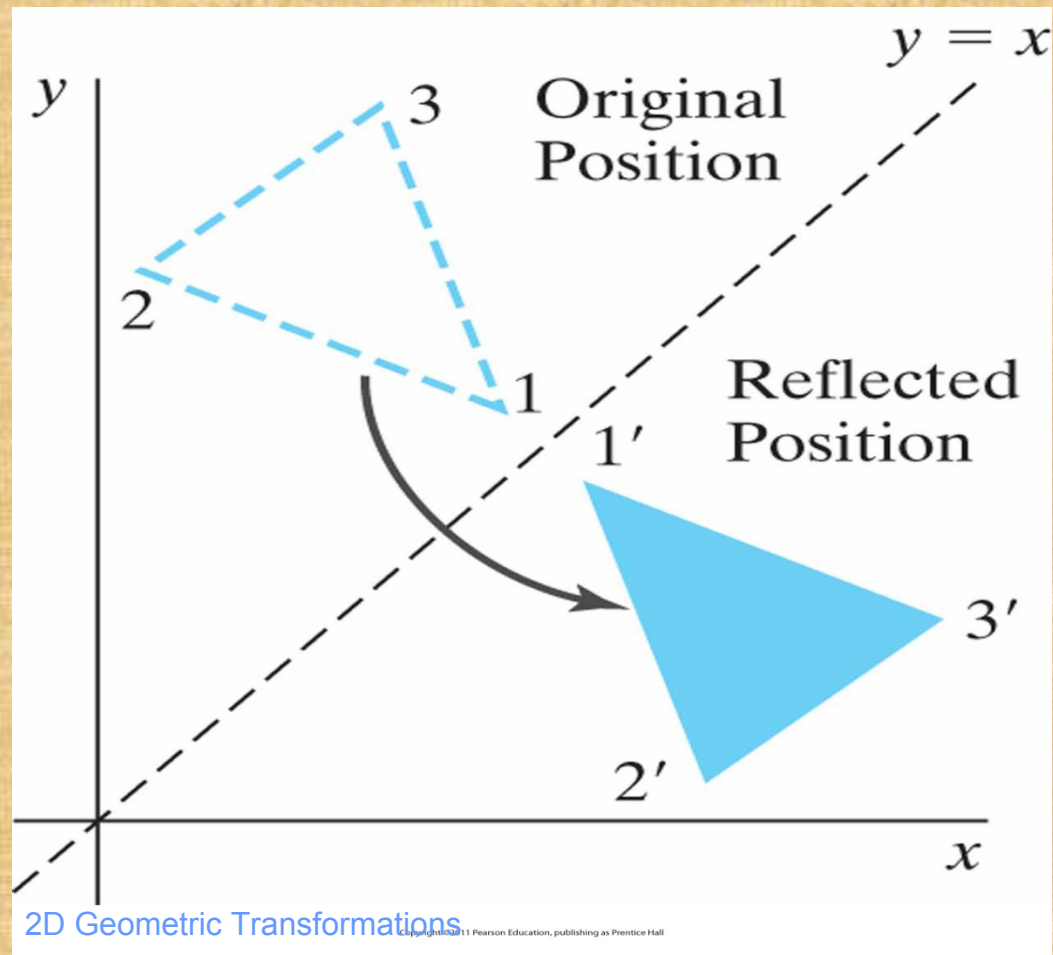$$\begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$$

P= [ 1.2    -3.5]

P' = PT
    = [-1.2    3.5]



*y*

Reflected Position

3'

2'

1'

*x*

1

2

3

Original Position

Copyright ©2011 Pearson Education, publishing as Prentice Hall

# Reflection about the line y=x

- Reflection about the line *y=x*

Copyright ©2011 Pearson Education, publishing as Prentice Hall
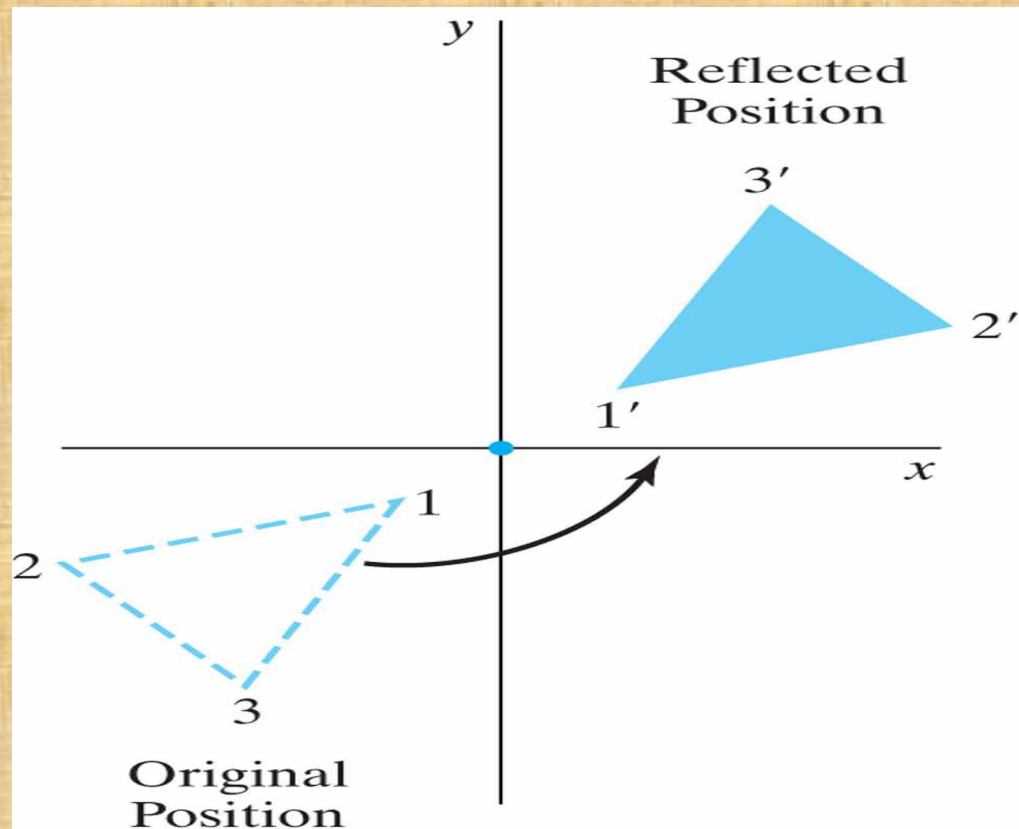
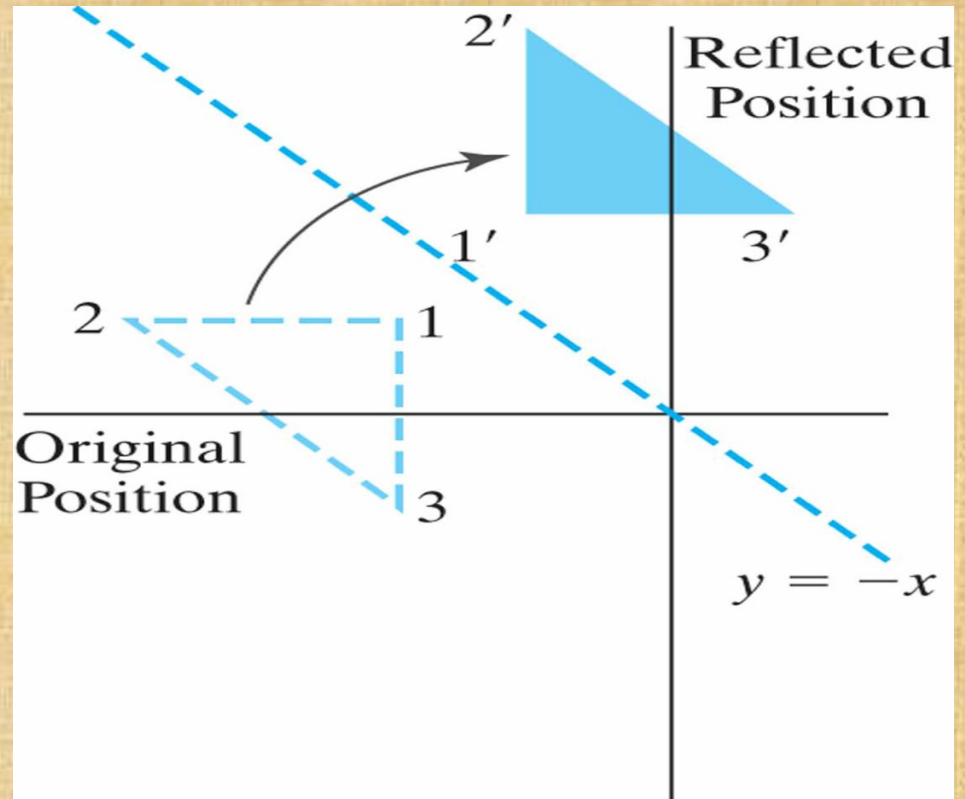Reflect line seg AB through the line y=x, where A= [1 2], B= [0.1 3].

Soln:

$$X= \text{seg AB} = \begin{bmatrix} 1 & 2 \\ 0.1 & 3 \end{bmatrix}$$

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

$$X' = XT = \begin{bmatrix} 2 & 1 \\ 3 & 0.1 \end{bmatrix}$$

# Reflection about the line y=-x

- Reflection about the line $y=-x$ → x and y coordinates swap with sign changes

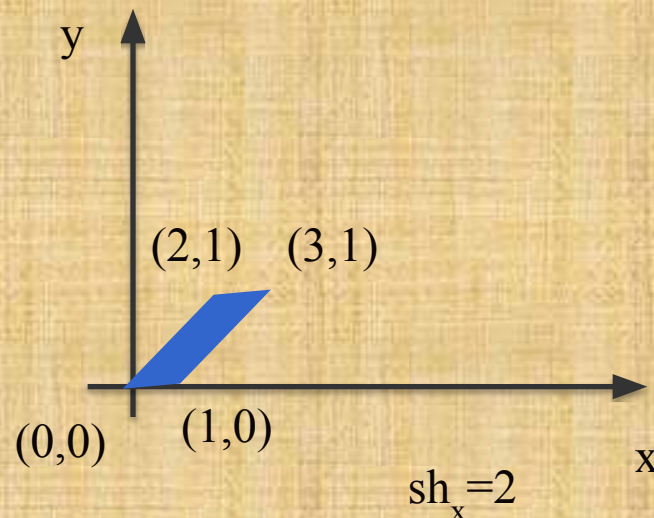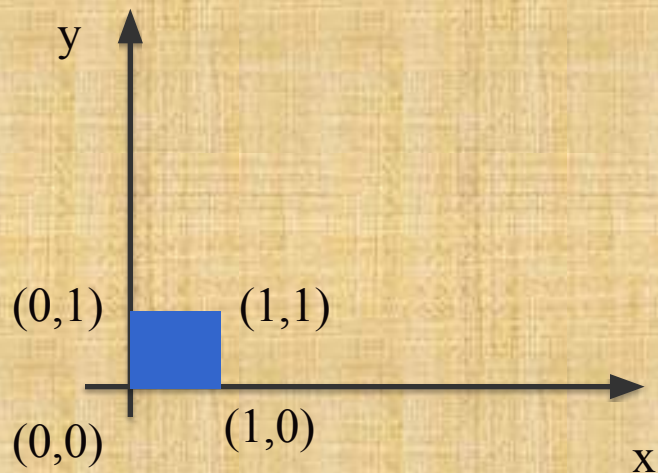Copyright ©2011 Pearson Education, publishing as Prentice Hall

# **Shear**

- ## 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

# Shear

- An y-direction shear
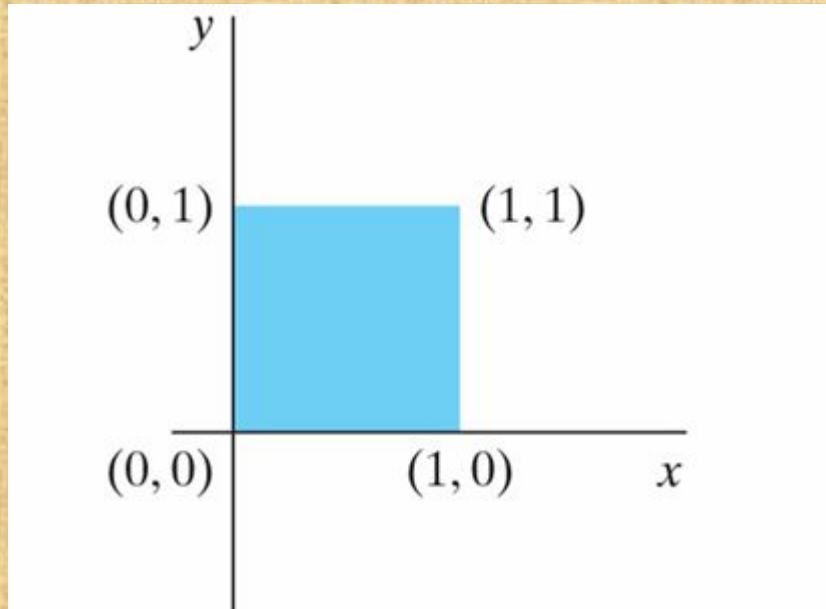
- $$\begin{bmatrix} 1 & b \\ 0 & 1 \end{bmatrix}$$

  $[x', y'] = [x, y]\ T$

  $x' = x$

  $y' = b\ x + y$

- An x-direction shear

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

  $[x', y'] = [x, y]\ T$

  $x' = x + c \cdot y$

  $y' = y$

# **Example**

A unit square is transformed to a shifted parallelogram using shear in x-direction by 0.5 units. Find the transformed object.



$$T = \begin{bmatrix} 1 & 0 \\ 0.5 & 1 \end{bmatrix}, X = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}$$

Find X' = XT

$$X' = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1.5 & 1 \\ 0.5 & 1 \end{bmatrix}$$
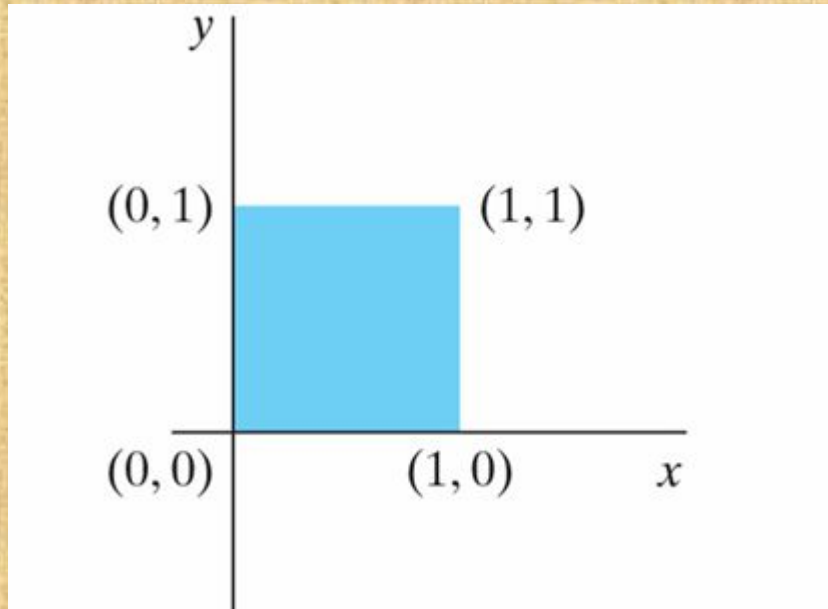
# **Example**

A unit square is transformed to a shifted parallelogram using shear in Y-direction by 0.35 units. Find the transformed object.



$$T = \begin{bmatrix} 1 & 0.35 \\ 0 & 1 \end{bmatrix}, X = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}$$

Find X' = XT

$$X' = \begin{bmatrix} 0 & 0 \\ 1 & 0.35 \\ 1 & 1.35 \\ 0 & 1 \end{bmatrix}$$

# 2D Scaling

- 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 a and d

  $x' = x \cdot a$

  $y' = y \cdot d$

  - $T = \begin{bmatrix} a & 0 \\ 0 & d \end{bmatrix}$    $[x', y'] = [x, y]\ T = [ax, dy]$

  or $P' = P\ T$

# 2D Scaling

- Any positive value can be used as scaling factor
    - Value less than 1 reduces the size of the object
    - Value greater than 1 enlarges the object
    - If scaling factor is 1 then the object stays unchanged
    - If $a = d$ , we call it <u>uniform scaling</u> i.e. **scaling in X dirn is same as scaling in y dirn then it is called uniform scaling.**
    - 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

x'      x
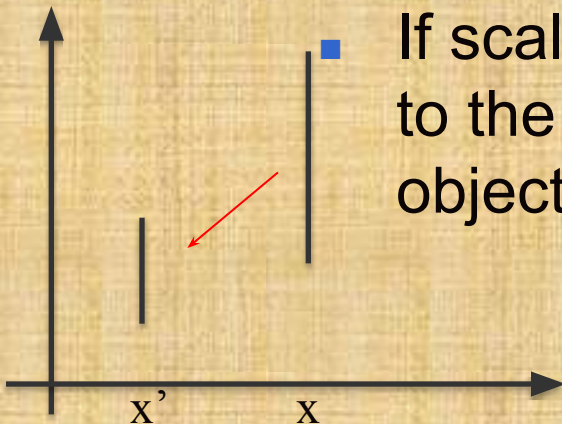
# Basic 2D Geometric Transformations (cont.)

- ## 2D Scaling

  - Why does scaling also reposition object?
  - Answer: See the matrix (multiplication)
  - Still no clue?
  - $$[x' \quad y'] = [x \quad y] \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} = [x\,s_x \quad y s_y]$$

# Write the transformation for the following.

- Scaling in x-dirn by 0.7 units.

- $T = \begin{bmatrix} 0.7 & 0 \\ 0 & 1 \end{bmatrix}$

- Uniform scaling by 3 units.

- $T = \begin{bmatrix} 3 & 0 \\ 0 & 3 \end{bmatrix}$

# 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 ( );
 }
```
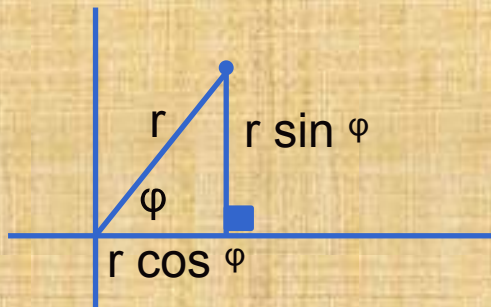
# Basic 2D Rotation

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

$y_r$

$\theta$

$x_r$
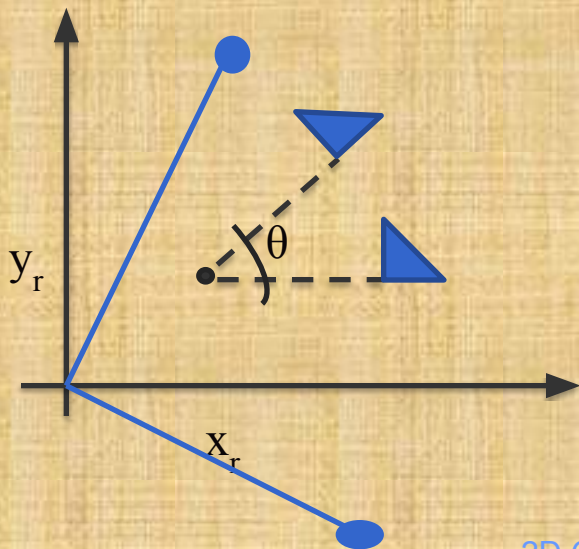
# Basic 2D Rotation

- 2D Rotation
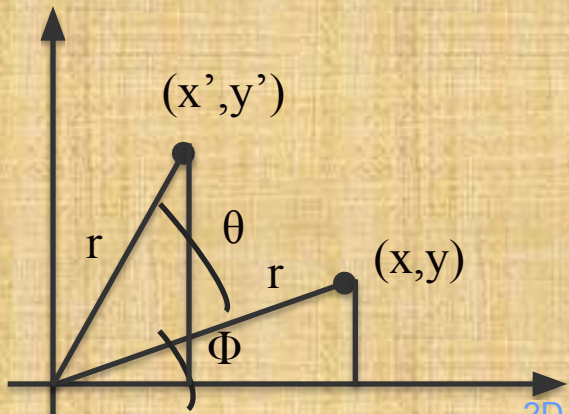  - If θ is positive ⟶ counterclockwise rotation
  - If θ is negative ⟶ clockwise rotation
  - Remember:
    - *cos(a + b) = cos a cos b - sin a sin b*
    - *cos(a - b) = cos a sin b + sin a cos b*
  - Polar coordinates of a point (x, y) = (r cos φ, r sin φ)



r

r sin φ

φ

r cos φ

# Basic 2D Rotation

- 2D Rotation

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

  ❑ $x' = r\cos(\theta+\Phi) = r\cos\theta\cos\Phi - r\sin\theta\sin\Phi$

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

  ❑ $x = r\cos\Phi,\ y = r\sin\Phi$

  ❑ $x' = x\cos\theta - y\sin\theta$

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

$$[x' \quad y'] = [x \quad y]\begin{bmatrix} \cos\Theta & \sin\Theta \\ -\sin\Theta & \cos\Theta \end{bmatrix}$$

(x',y')

r

θ

r

(x,y)

Φ

# Basic 2D Rotation

- ## 2D Rotation
  - P'=P R

$$R = \begin{bmatrix} \cos\Theta & \sin\Theta \\ -\sin\Theta & \cos\Theta \end{bmatrix}$$
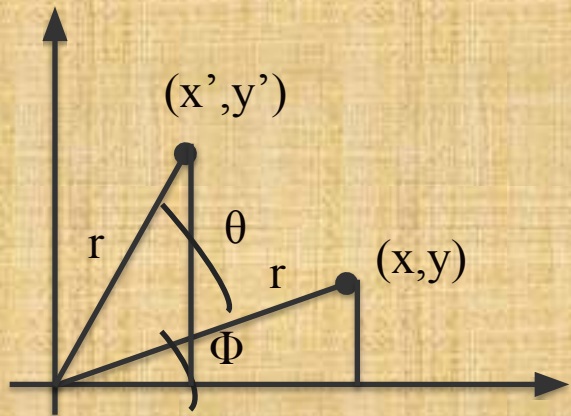
# Rotate the line seg AB in anticlockwise direction by 30°. A=[1,1], B=[0,5]

- $X = \begin{bmatrix} A \\ B \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 5 \end{bmatrix}$

- $T = \begin{bmatrix} \cos(30) & \sin(30) \\ -\sin(30) & \cos(30) \end{bmatrix}$

  $= \begin{bmatrix} \sqrt{3}/2 & 0.5 \\ -0.5 & \sqrt{3}/2 \end{bmatrix}$

- X' = XT

- $X' = \begin{bmatrix} & \\ & \end{bmatrix}$

# Rotate the triangle ABC in clockwise direction by 45°. A=[1,1], B=[0,5], C=[2,0]

- 

- $X = \begin{bmatrix} 1 & 1 \\ 0 & 5 \\ 2 & 0 \end{bmatrix}$

- $T = \begin{bmatrix} \cos(45) & \sin(-45) \\ -\sin(-45) & \cos(45) \end{bmatrix}$

  $= \begin{bmatrix} 0.7071 & -0.7071 \\ 0.7071 & 0.7071 \end{bmatrix}$

- X' = XT

- $X' = \begin{bmatrix} & \\ & \end{bmatrix}$

# Basic 2D Rotation

- 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

# 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 ( );
}
```

# 2D Translation
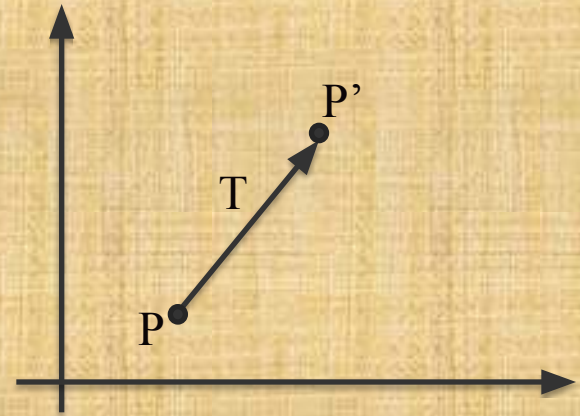
■ **Origin is invariant under 2x2 transformation matrix**

   ❑ [0, 0] T = [0, 0] for all T

   ❑ Translation means -

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

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

   ❑ P'=P+T

   ❑ **Translation moves the object without deformation** (rigid-body transformation)

# 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.)

- 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 row 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,** $h \neq 0$, $x_h = xh$, $y_h = yh$
- h is the **homogeneous parameter** such that *x = x_h/h, y = y_h/h,*
- □ infinite homogeneous representations for a point
- A convenient choice is to choose h = 1

# Homogeneous coordinates

- Example
- Let P be a point with physical coordinates [1,2].
- For h=1, P=[1, 2, 1]
- For h=2, P=[2, 4, 2]          homogeneous
- For h=0.5, P=[0.5, 1, 0.5]      coordinates

- Find physical coordinates of the point [6, 8, 2].
- [6, 8, 2] ~ [6/2, 8/2, 2/2]~ [3,4,1] = [3,4]

# Matrix Representations and Homogeneous Coordinates (cont.)

- 2D Translation Matrix

- $T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ p & q & 1 \end{bmatrix}$

- p= Translation in x-dirn
- q= Translation in y-dirn

- $X = [x, y] \sim [x, y, 1]$
- $X' = XT$

  $= [x+p \quad y+q \quad 1]$

- $X' = [x+p, y+q]$
- If X is origin, X'=[p,q].
- Thus the origin is shifted to (p, q).

# Matrix Representations and Homogeneous Coordinates (cont.)

■ 2D Rotation Matrix

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

or, **P'** = **P** . **R**($\theta$)

# Matrix Representations and Homogeneous Coordinates (cont.)

■ 2D Scaling Matrix

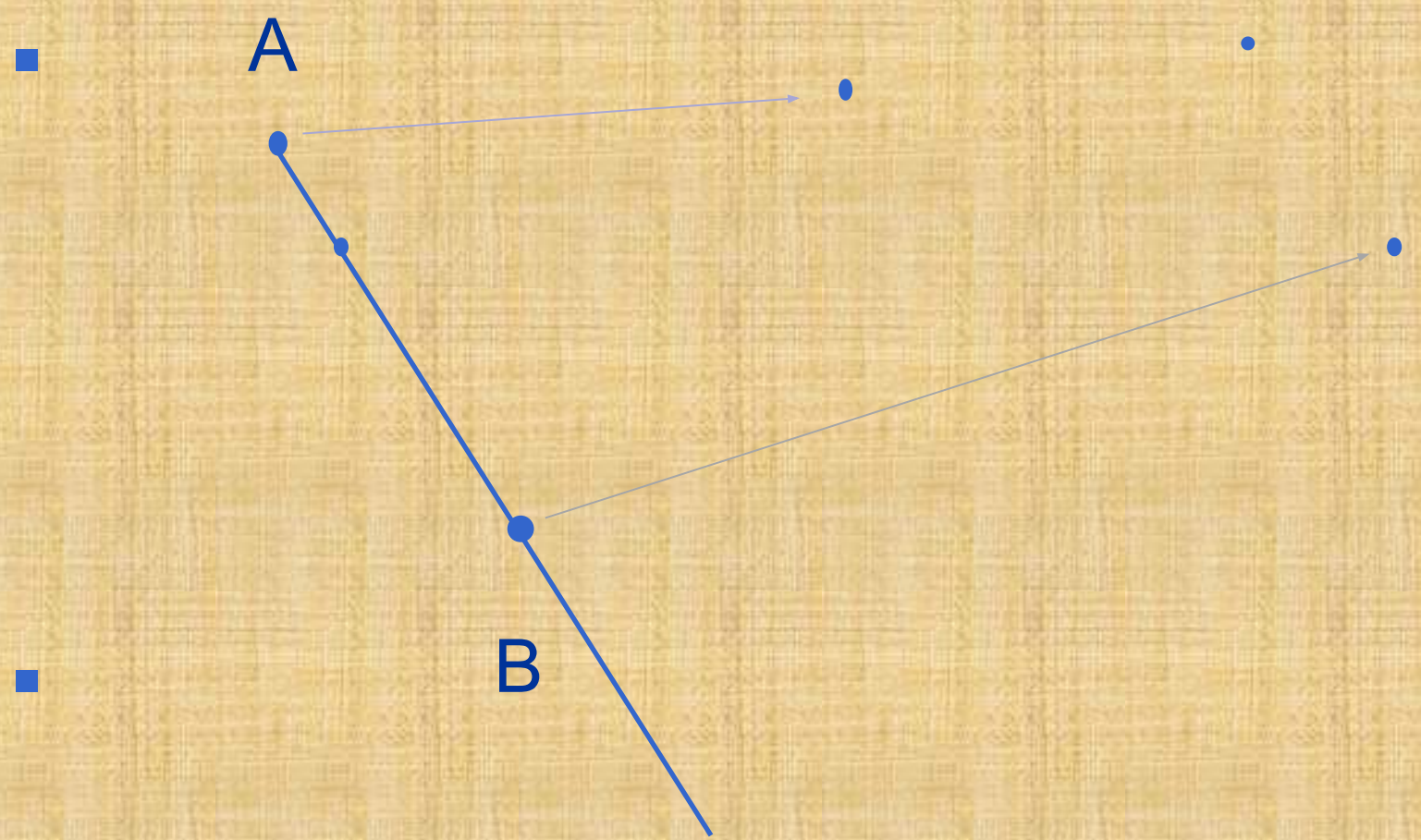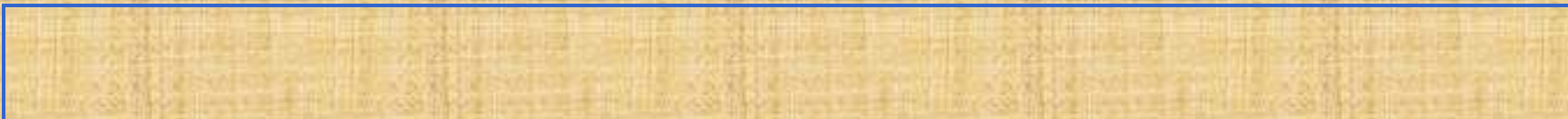$$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

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

# 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 ( );
}
```
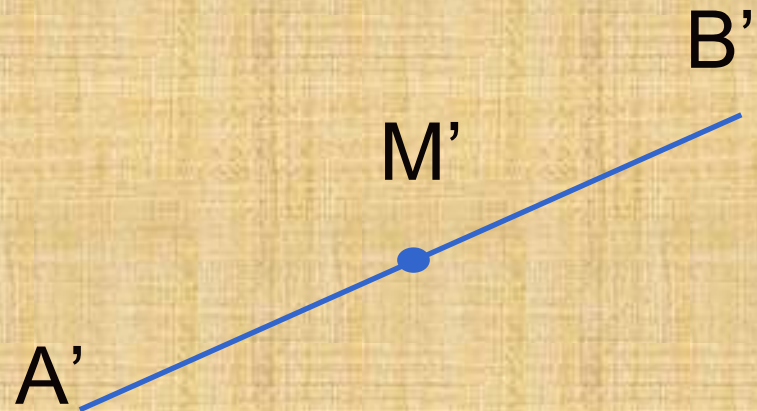
A

B

# Mid point transformation

- If a 2x2 transformation T is applied on a line segment AB, then the mid point of AB is mapped to the mid point of transformed segment A'B'.

A     M     B                         B'

                                    M'

**M' = M T**                   A'

If seg AB is transformed to seg A'B' using transformation T, then find the midpoint of A'B',

- 

- M= midpoint of AB = [3, 2.5]
- Mid point of [x1, y1], [x2, y2] = [(x1+x2)/2, (y1+y2)/2]
- M'= MT = $[3 \quad 2.5] \begin{bmatrix} 2 & 3 \\ -1 & 1 \end{bmatrix}$ = [3.5,  11.5]

# Proof of mid point transformation

- A=[x1,y1], B=[x2,y2] and T = [a     b   // c      d]
- Mid point of AB = M = [(x1+x2)/2, (y1+y2)/2]
- A' = AT = [ax1+cy1, bx1+dy1],
- B' = BT = [ax2+cy2, bx2+dy2]
- M'= MT = [a(x1+x2)/2+c(y1+y2)/2,
                    b(x1+x2)/2+d(y1+y2)/2]
-    Mid point of A'B'= [(a(x1+x2)+c(y1+y2))/2, (b(b1+x2)+d(y1+y2))/2] = M'

# Change of slope

- If a 2x2 transformation T= $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$ is applied on a line segment AB with slope m, then the slope of transformed line segment A'B' is $m' = \dfrac{b+dm}{a+cm}$ .

Find slope of transformed line if 2x-y=3 is transformed using $T=\begin{matrix} 2 & -1 \\ 2 & 4 \end{matrix}$ .

# A=[1, 1], B=[2, 0]. T is applied on seg AB. Find slope of A'B'.

- T=$\begin{bmatrix} 1 & -1 \\ 2 & 6 \end{bmatrix}$

- Slope of AB = m = $\frac{y2-y1}{x2-x1} = \frac{0-1}{2-1} = -1$

- $m' = \frac{b+dm}{a+cm}$ = 7
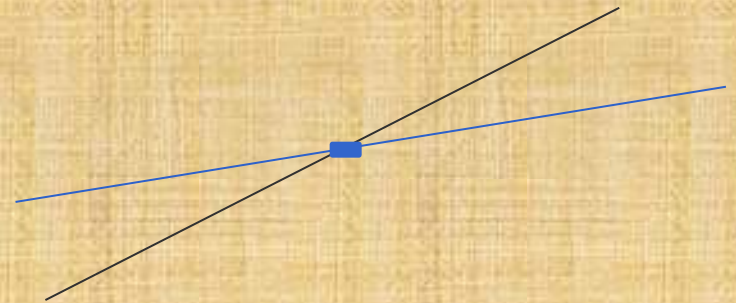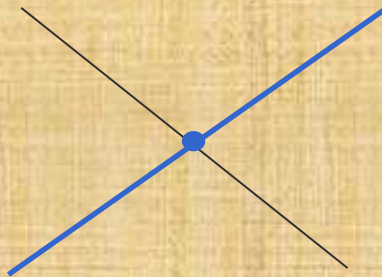
# Transformation on parallel lines

- $m' = \dfrac{b+dm}{a+cm}$

- If two lines L1 and L2 have same slope m and the same transformation T is applied on L1 and L2, then they will have same slope $m' = \dfrac{b+dm}{a+cm}$.

- L1' || L2'

# Transformation on intersecting lines

- If L1 and L2 intersect at a point P and the same transformation T is applied on both L1 and L2, then …?

# Pure rotation / Pure reflection

- A transformation T is called a pure rotation if det(T)=1.

- Ex. T= $\begin{matrix} 1 & 1 \\ 2 & 3 \end{matrix}$ = $\det(T) = 1$

- T is pure rotation.

- A transformation T is called a pure reflection if det(T)=-1.

- T= $\begin{matrix} 3 & 2 \\ -1 & -1 \end{matrix}$ . $det(T) = -1$. T is a pure reflection

# Inverse Transformations

- 2D Inverse Translation Matrix

$$T^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -tx & -ty & 1 \end{bmatrix}$$
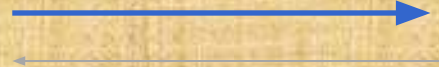
- By the way:

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

# 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

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

  - Is that true?
  - Proof: It's up to you ☺

# Inverse Transformations (cont.)

- 2D Inverse Scaling Matrix

$$S^{-1}$$

$$= \begin{bmatrix} \dfrac{1}{s_x} & 0 & 0 \\ 0 & \dfrac{1}{s_y} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- 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

- $P'=(P.M_1).M_2 =P(M1.M2)$
  $=P.M$

- M – composite / combined / concatenated transformation matrix

# Find combined trans matrix for – scaling in x-dirn by 2 units, shear in y-dirn by 1.7 units, reflection through the line x=y.

- Scaling in x-dirn by 2 units. $T1 = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$

- Shear in y-dirn by 1.7 units $T2 = \begin{bmatrix} 1 & 1.7 \\ 0 & 1 \end{bmatrix}$

- Reflection through x=y $T3 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$

- T = T1.T2.T3

  $= \begin{matrix} 3.4 & 2 \\ 1 & 0 \end{matrix}$

# Find the concatenated trans matrix – rotation about origin through 90. translation in y dirn by 3 units. Uniform scaling by 2 units

- Rotation about origin 90 T1

$$= \begin{bmatrix} c & s & 0 \\ -s & c & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Translation in y-dirn by 3

$$\text{units } T2 = \begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 3 & 1 \end{matrix}$$

- Uniform scaling by 2 units

$$T3 = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- T=T1. T2. T3

# 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

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

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ p0 & q0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ p1 & q1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ p0 + p1 & q0 + q1 & 1 \end{bmatrix}$$

# 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.)

- Don't forget:
- 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
  - 9 times?
  - Why?
  - Proof: Again up to you ☺
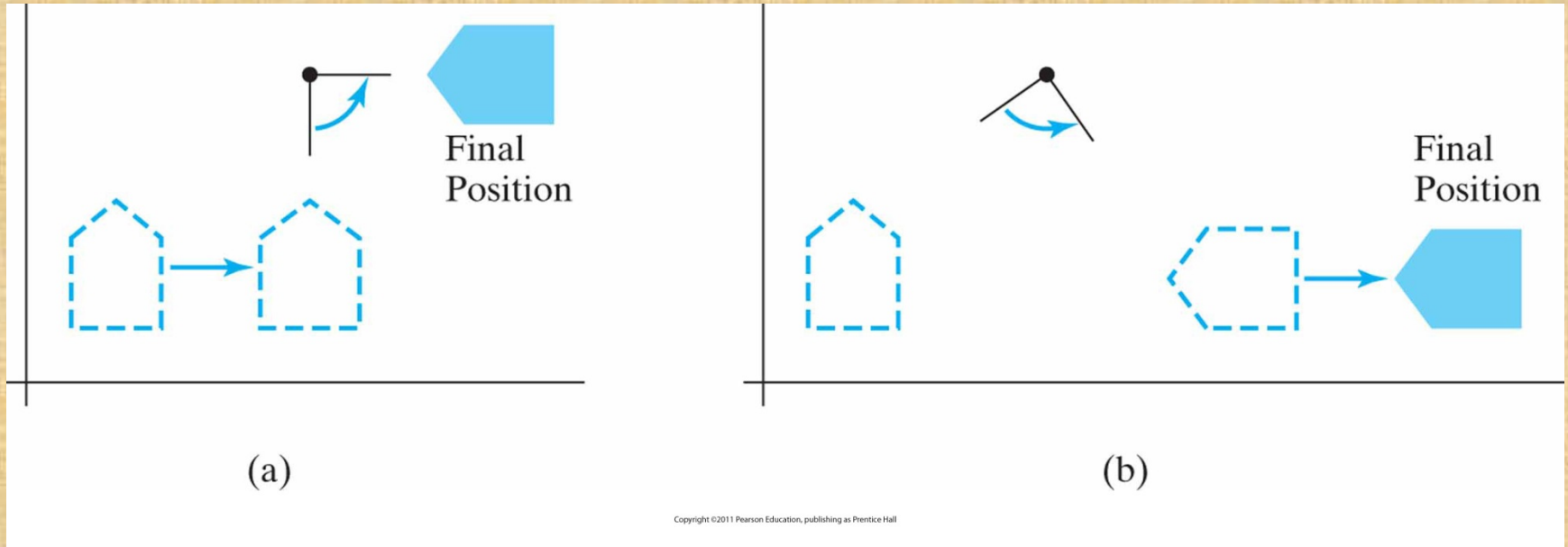
# 2D Composite Transformations (cont.)

- 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 multiplicating 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!
- Why?
- Proof: You got it: Up to you ☺  ☺

# Reversing the order



(a)

(b)

Copyright ©2011 Pearson Education, publishing as Prentice Hall

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

# General Pivot Point Rotation / Rotation about an arbitrary point

■   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.

# General Pivot Point Rotation / Rotation about an arbitrary point
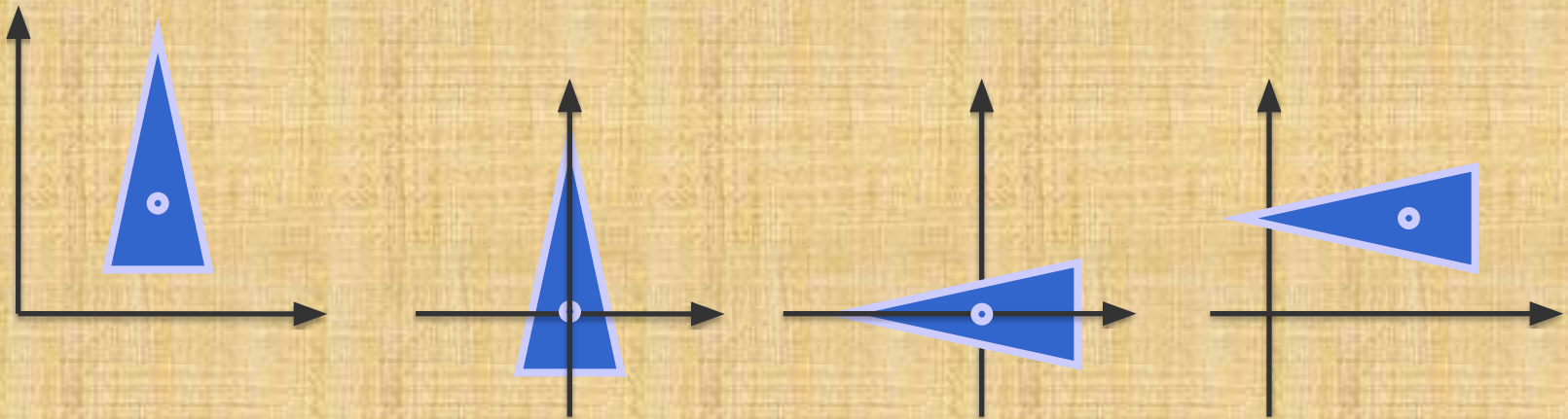
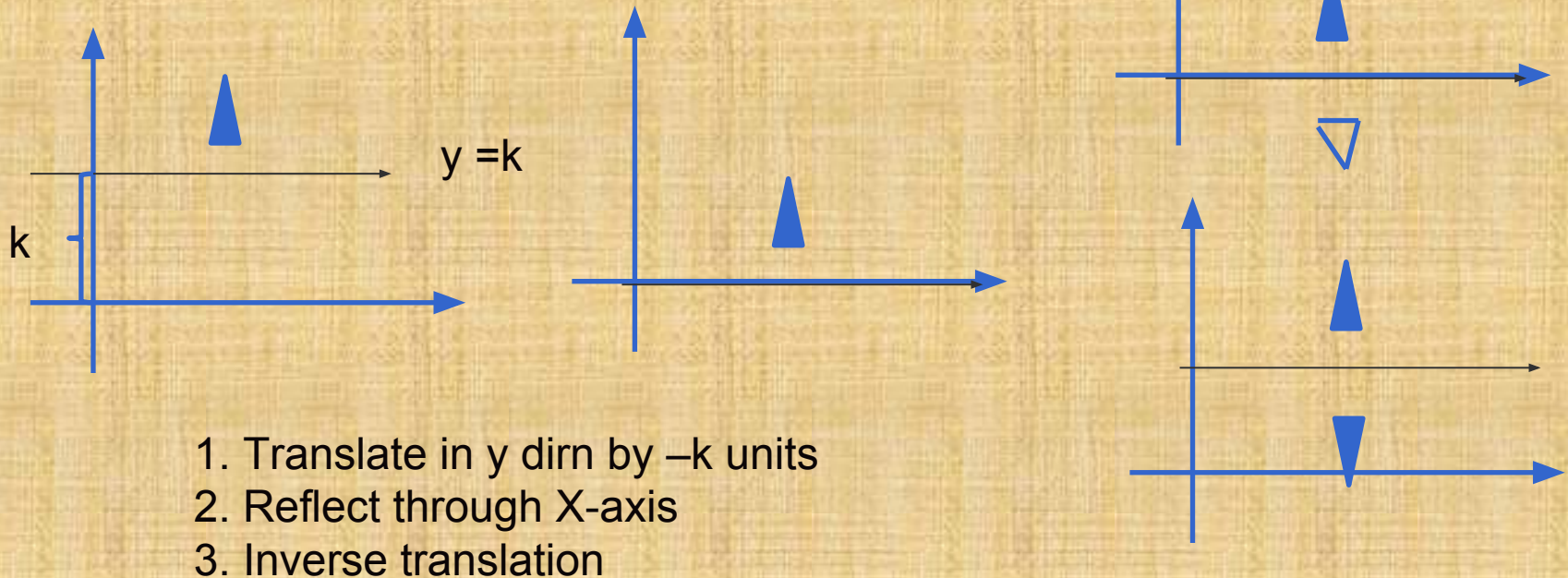# 2D Composite Transformations To rotate about point (a, b)

- General 2D Pivot-Point Rotation

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -a & -b & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\Theta & \sin\Theta & 0 \\ -\sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ a & b & 1 \end{bmatrix}$$

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

# Reflection through a line parallel to X-axis

y =k

k

1. Translate in y dirn by –k units
2. Reflect through X-axis
3. Inverse translation

# Reflection through a line parallel to X-axis, y=k

- 
- Step 1 : T1 = $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -k & 1 \end{bmatrix}$
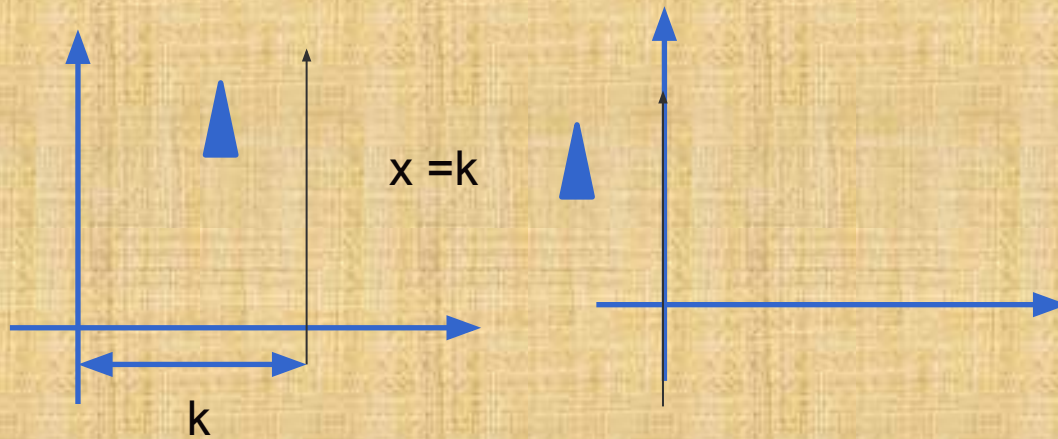
- Step 2 : T2 = $\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

- Step 3 : T3 = $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & k & 1 \end{bmatrix}$

Step 4 : T= T1. T2. T3

$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 2k & 1 \end{bmatrix}$

# Reflection through a line parallel to Y-axis, x=k

x =k

k

1. Translate in X  dirn by –k units
2. Reflect through Y-axis
3. Inverse translation (Translate
in X dirn by k units)

# Reflection through a line parallel to Y-axis, x=k

- 
- Step 1 : T1 = $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -k & 0 & 1 \end{bmatrix}$

- Step 2 : T2 = $\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

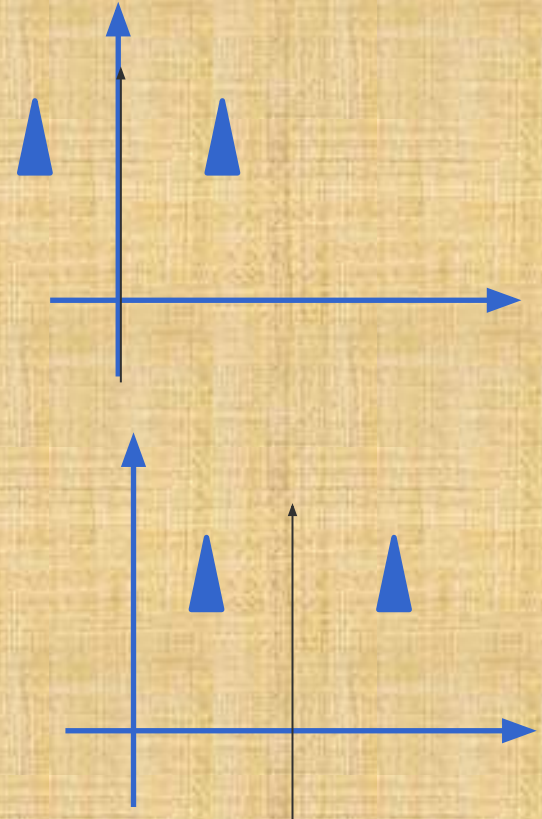- Step 3 : T3 = $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ k & 0 & 1 \end{bmatrix}$

Step 4 : T= T1. T2. T3

$$= \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 2k & 0 & 1 \end{bmatrix}$$

# Reflect the seg AB through y=3, A= [1,-2], B=[4,9]

- 
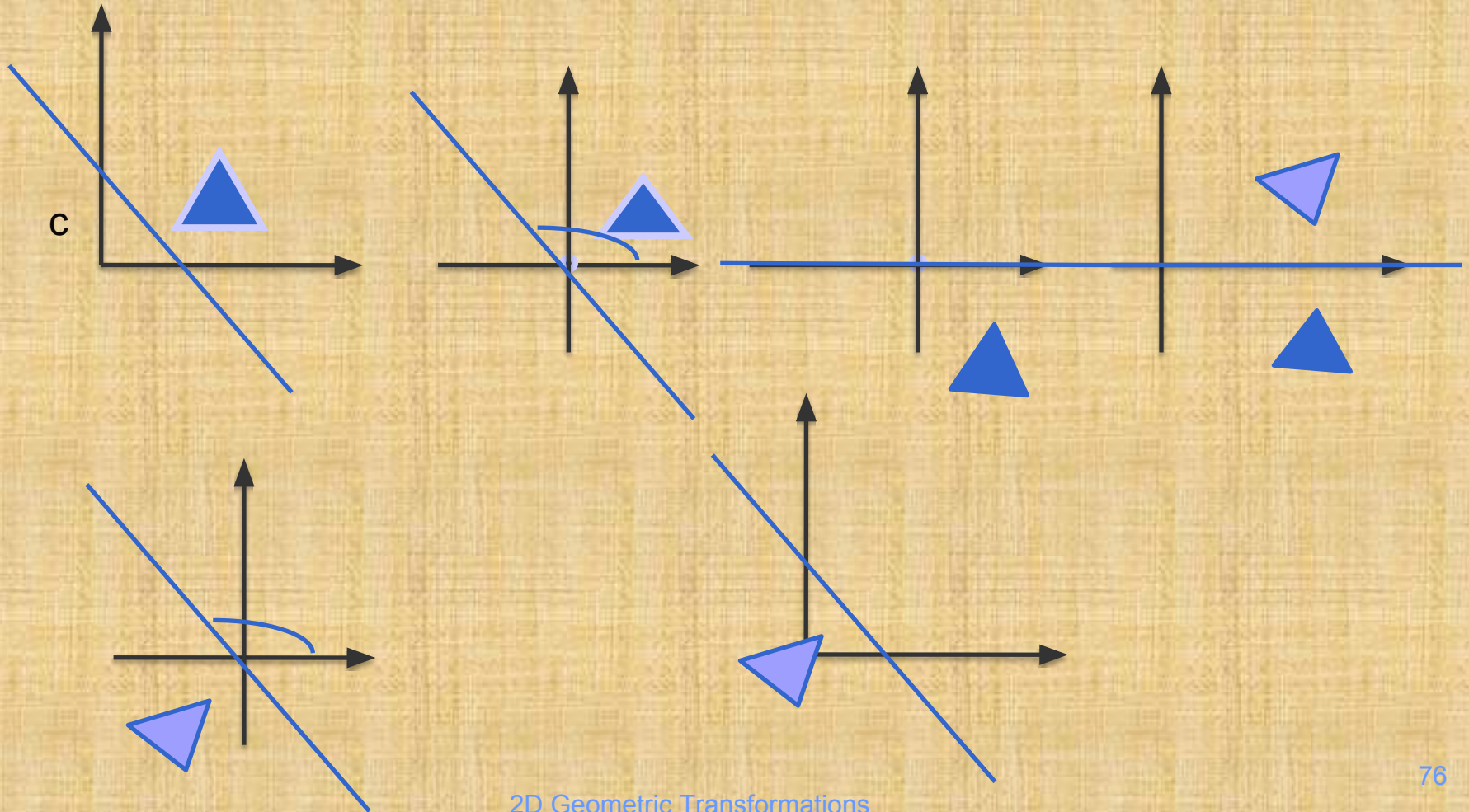- T= $\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 6 & 1 \end{bmatrix}$

- X' = XT

- $= \begin{matrix} 1 & -2 & 1 \\ 4 & 9 & 1 \end{matrix} \, T$

# Reflection through an arbitrary line L : y = mx + c

- Translate in y direction by –c units so that the line L passes though the origin.

- Find the angle of inclination $\theta = tan^{-1}(m)$

- Rotate about origin through angle $\theta$ in clockwise direction so that L coincides with X-axis.

- Reflect the object through X-axis.

- Inverse rotation

- Inverse translation

# Reflection through an arbitrary line

# Reflection through an arbitrary line

- T1= $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -c & 1 \end{bmatrix}$

- T2= $\begin{bmatrix} c(\theta) & -s(\theta) & 0 \\ s(\theta) & c(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$

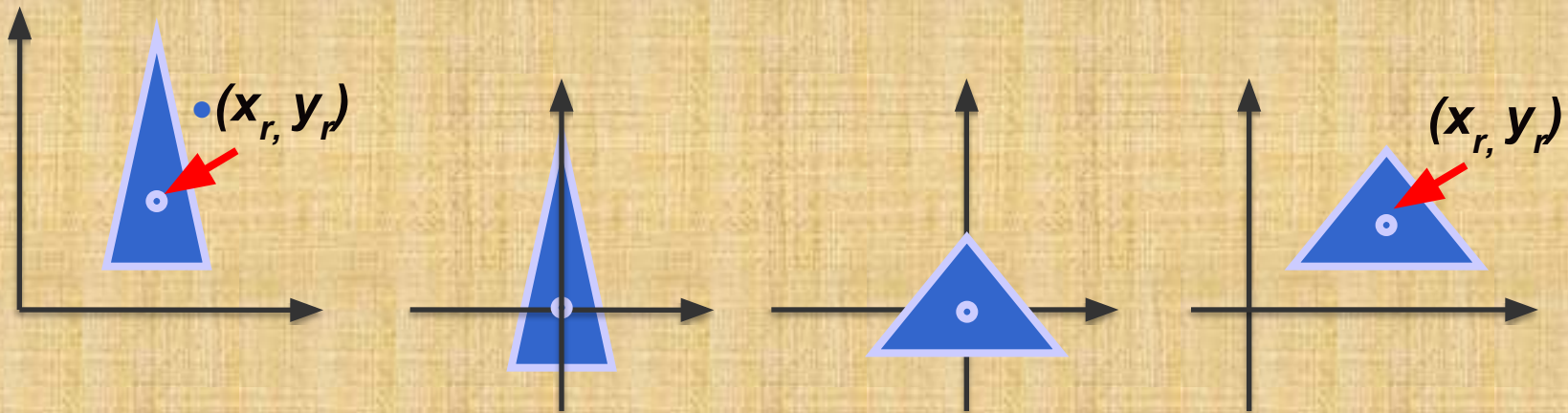- T3= $\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

- T4= $\begin{bmatrix} c(\theta) & s(\theta) & 0 \\ -s(\theta) & c(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$

- T5 = $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & c & 1 \end{bmatrix}$

- T=T1.T2.T3.T4.T5
- X'=X.T

# General Fixed Point Scaling (cont.) (Not in the syllabus)

# General Fixed Point Scaling (cont.) (Not in the syllabus)

- General 2D Fixed-Point Scaling:

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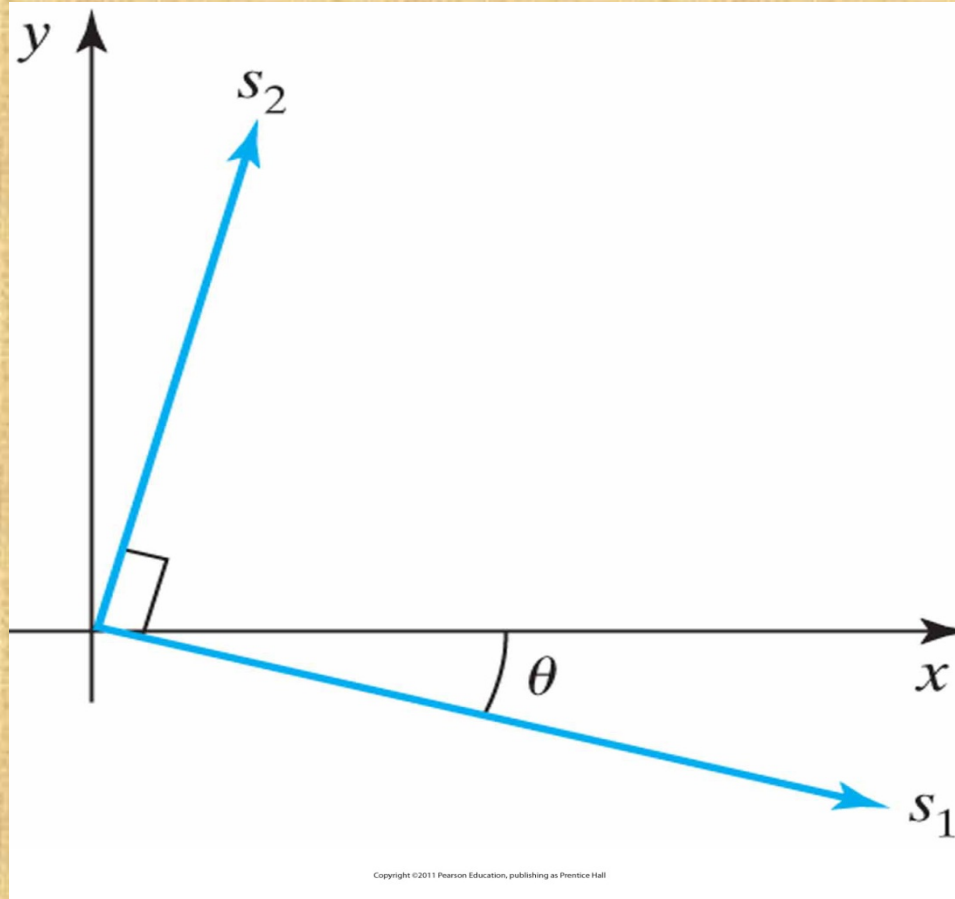
$$\mathbf{T}(x_f, y_f) \cdot \mathbf{S}(s_x, s_y) \cdot \mathbf{T}(-x_f, -y_f) = \mathbf{S}(x_f, y_f, s_x, s_y)$$

# 2D Composite Transformations (cont.) (Not in the syllabus)

- 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 (Not in the syllabus)



Copyright ©2011 Pearson Education, publishing as Prentice Hall

Scaling parameters $s_1$ and $s_2$ along orthogonal directions defined by the angular displacement $\theta$.

# General 2D Scaling Directions (cont.) (Not in the syllabus)

- **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 multiplicating left-to-right (premultiplication) or right-to-left (postmultiplication)
  - Matrix multiplication is *not* commutative !
    - $M_2 \cdot M_1 \neq M_1 \cdot M_2$
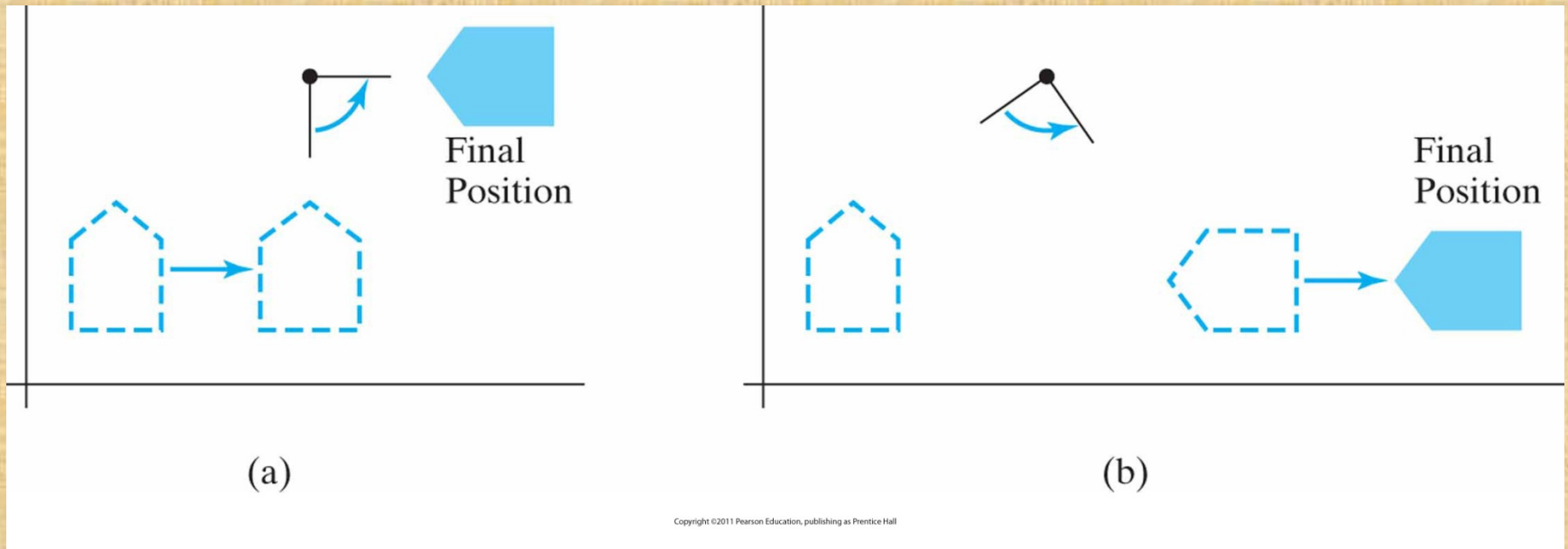
# 2D Composite Transformations (cont.)

- Matrix Concatenation Properties:
    - But:
        - ❑ Two successive rotations
        - ❑ Two successive translations
        - ❑ Two successive scalings
    - *are* commutative!
- Why?
- Proof: You got it: Up to you ☺  ☺

# Reversing the order



Final Position

Final Position

(a)

(b)
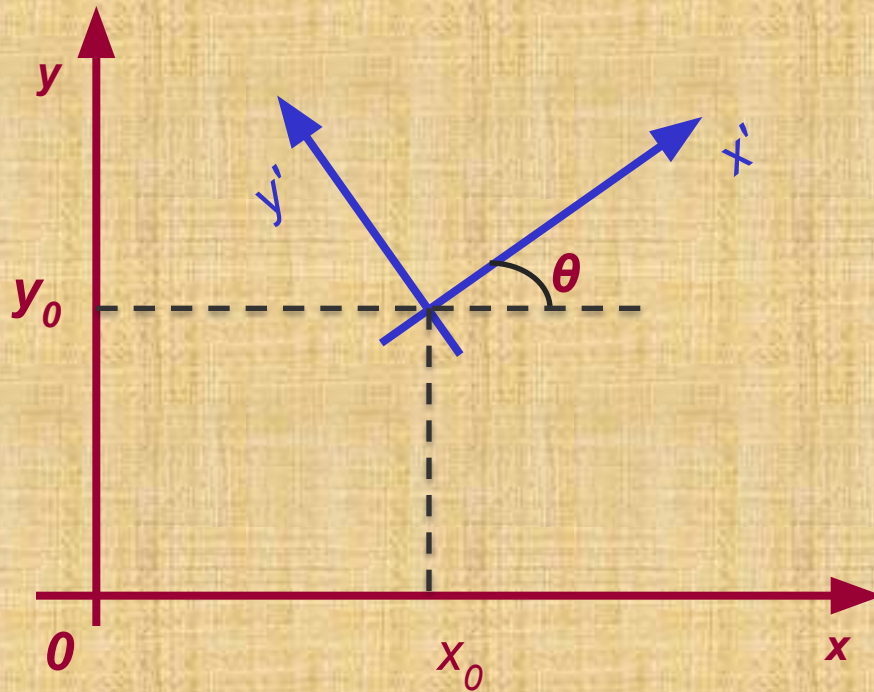
in which a sequence of transformations is performed may affect the transformed position of an object.
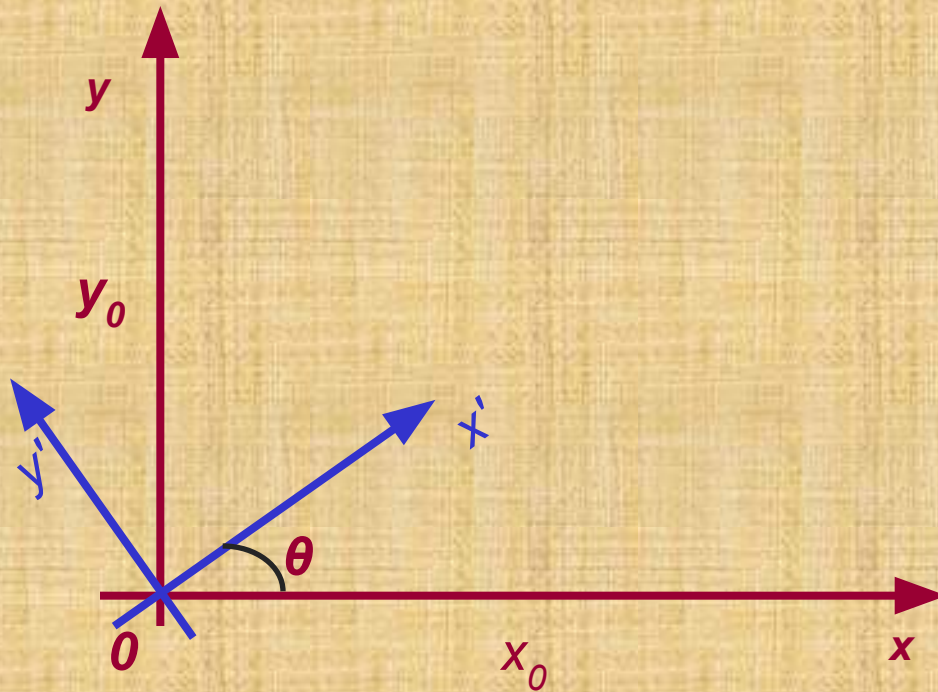In (a), an object is first translated in the x direction, then rotated counterclockwise through an angle of 45°.
In (b), the object is first rotated 45° counterclockwise, then translated in the *x* direction

# Transformation Between Coordinate Systems (cont.)

# Transformation Between Coordinate Systems (cont.)

# Transformation Between Coordinate Systems (cont.)
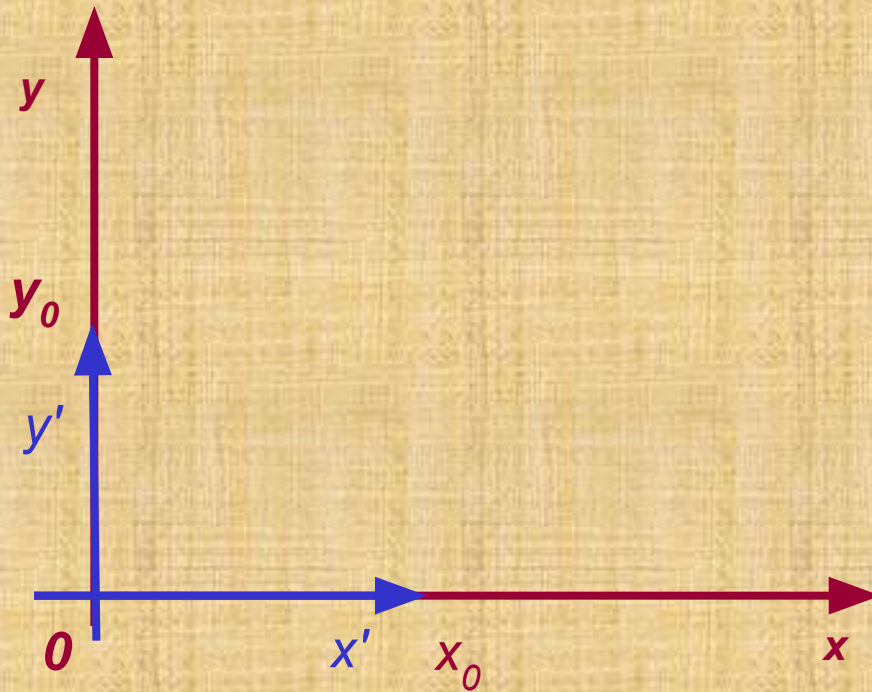
# OpenGL Geometric Transformation Functions

- 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 $z = 0$

# Basic OpenGL Geometric Transformations

- 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

# Basic OpenGL Geometric Transformations (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 (cont.)

- 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 (cont.)

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

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

# OpenGL Matrix Stacks

- OpenGL maintains a matrix stack for transformations

- Initially the modelview stack contains only the identity matrix

- More about it:
  - Coming soon

# 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. |

2D Geometric Transformations

# Next Lecture

3D Geometric Transformations

# References

- Donald Hearn, M. Pauline Baker, Warren R. Carithers, "Computer Graphics with OpenGL, 4th Edition"; Pearson, 2011

- Sumanta Guha, "Computer Graphics Through OpenGL: From Theory To Experiments", CRC Press, 2010

- Edward Angel, "Interactive Computer Graphics. A Top-Down Approach using OpenGL", Addison-Wesley, 2005