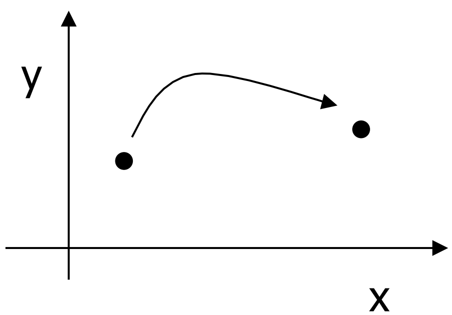
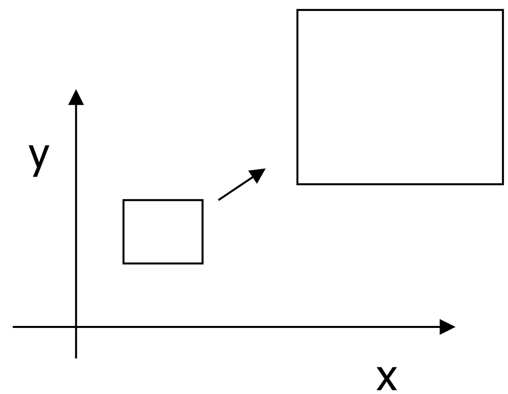
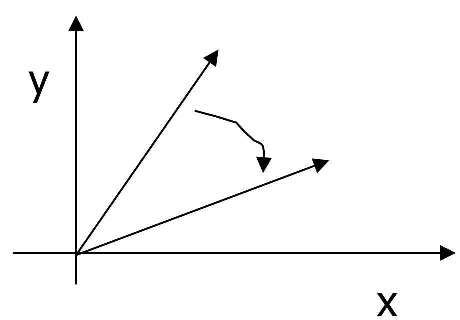
2D Transformations







2D Transformation

- Given a 2D object, transformation is to change the object's
  - Position (translation)
  - Size (scaling)
  - Orientation (rotation)
  - Shapes (shear)
- Apply a sequence of matrix multiplication to the object vertices

Point representation

- We can use a column vector (a 2x1 matrix) to represent a 2D point $\begin{vmatrix} x \\ y \end{vmatrix}$
- A general form of *linear* transformation can be written as:

$$x' = ax + by + c$$

$$y' = dx + ey + f$$

OR

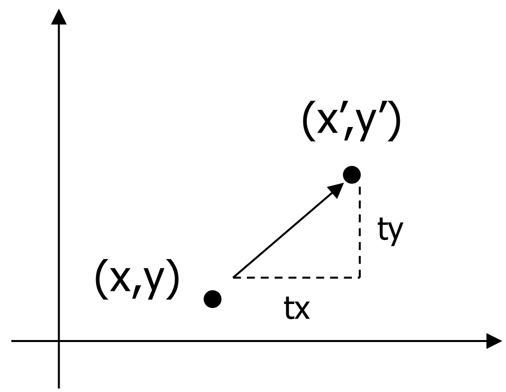
$$\begin{vmatrix} X' \\ Y' \\ 1 \end{vmatrix} = \begin{vmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{vmatrix} * \begin{vmatrix} x \\ y \\ 1 \end{vmatrix}$$

Translation

- Re-position a point along a straight line
- Given a point (x,y), and the translation distance (tx,ty)

The new point: (x′, y′)

$$x' = x + tx$$
$$y' = y + ty$$



OR $P' = P + T$ where $P' = \begin{vmatrix} x' \\ y' \end{vmatrix}$ $p = \begin{vmatrix} x \\ y \end{vmatrix}$ $T = \begin{vmatrix} tx \\ ty \end{vmatrix}$

3x3 2D Translation Matrix

$$\begin{vmatrix} x' \\ y' \end{vmatrix} = \begin{vmatrix} x \\ y \end{vmatrix} + \begin{vmatrix} tx \\ ty \end{vmatrix}$$

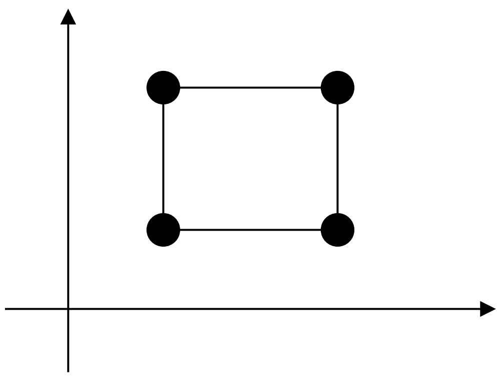
Use 3 x 1 vector

$$\begin{vmatrix} x' \\ y' \\ 1 \end{vmatrix} = \begin{vmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{vmatrix} * \begin{vmatrix} x \\ y \\ 1 \end{vmatrix}$$

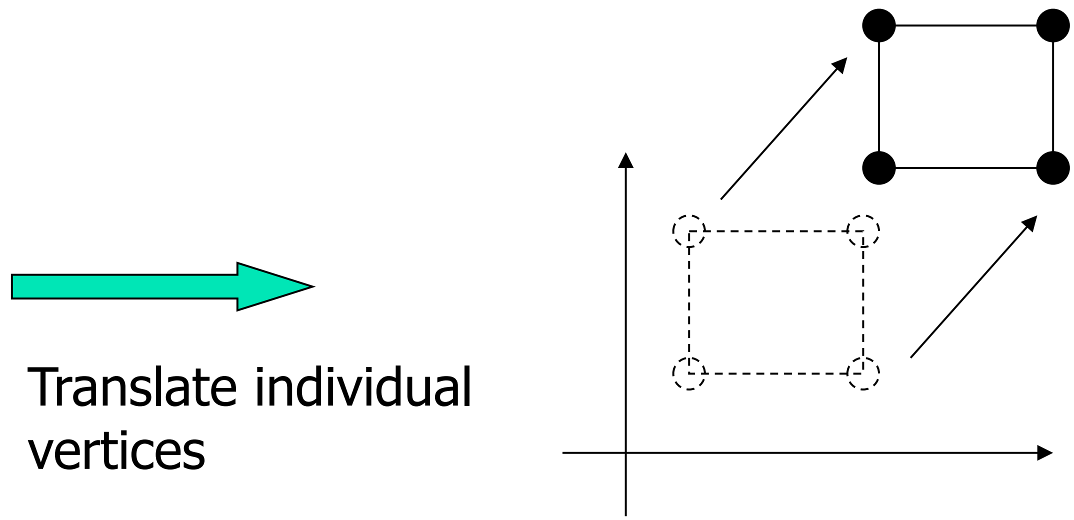
- Note that now it becomes a matrix-vector multiplication

Translation

- How to translate an object with multiple vertices?

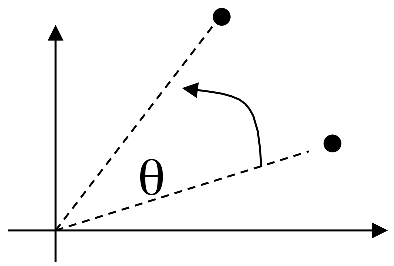


Translate individual vertices

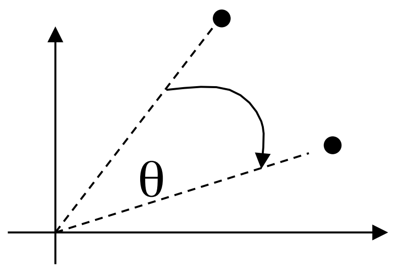


2D Rotation

- Default rotation center: Origin (0,0)



$\theta > 0$: Rotate counter clockwise



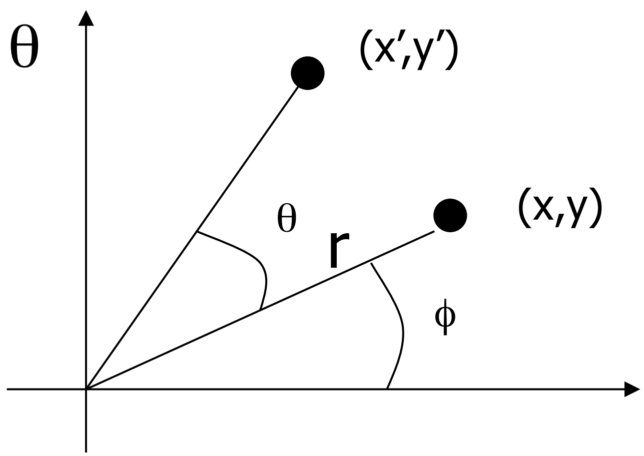
$\theta < 0$: Rotate clockwise

Rotation

(x,y) -> Rotate *about the origin* by θ

$\longrightarrow$ (x′, y′)

How to compute (x′, y′) ?



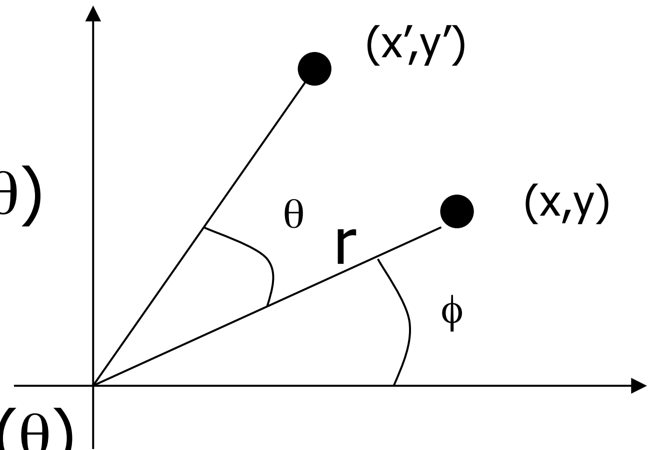
$x = r\cos(\phi) \quad y = r\sin(\phi)$

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

Rotation

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

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



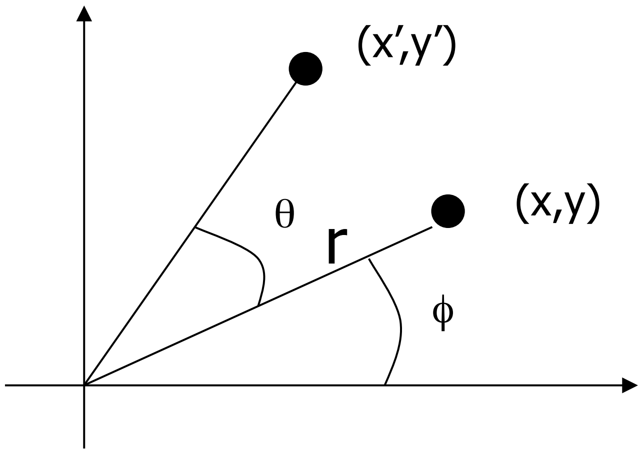
$$x' = r\cos(\phi + \theta)$$
$$= r\cos(\phi)\cos(\theta) - r\sin(\phi)\sin(\theta)$$
$$= x\cos(\theta) - y\sin(\theta)$$

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

Rotation

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

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



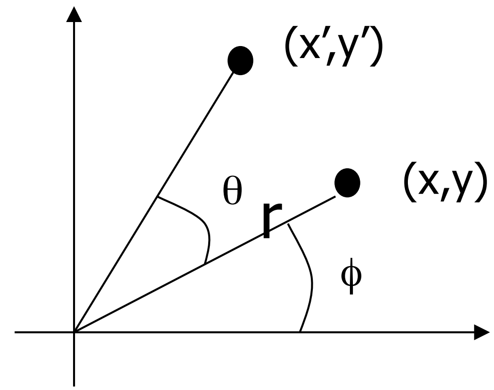
Matrix form?

$$\begin{vmatrix} x' \\ y' \end{vmatrix} = \begin{vmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{vmatrix} \begin{vmatrix} x \\ y \end{vmatrix}$$

3 x 3?

3x3 2D Rotation Matrix

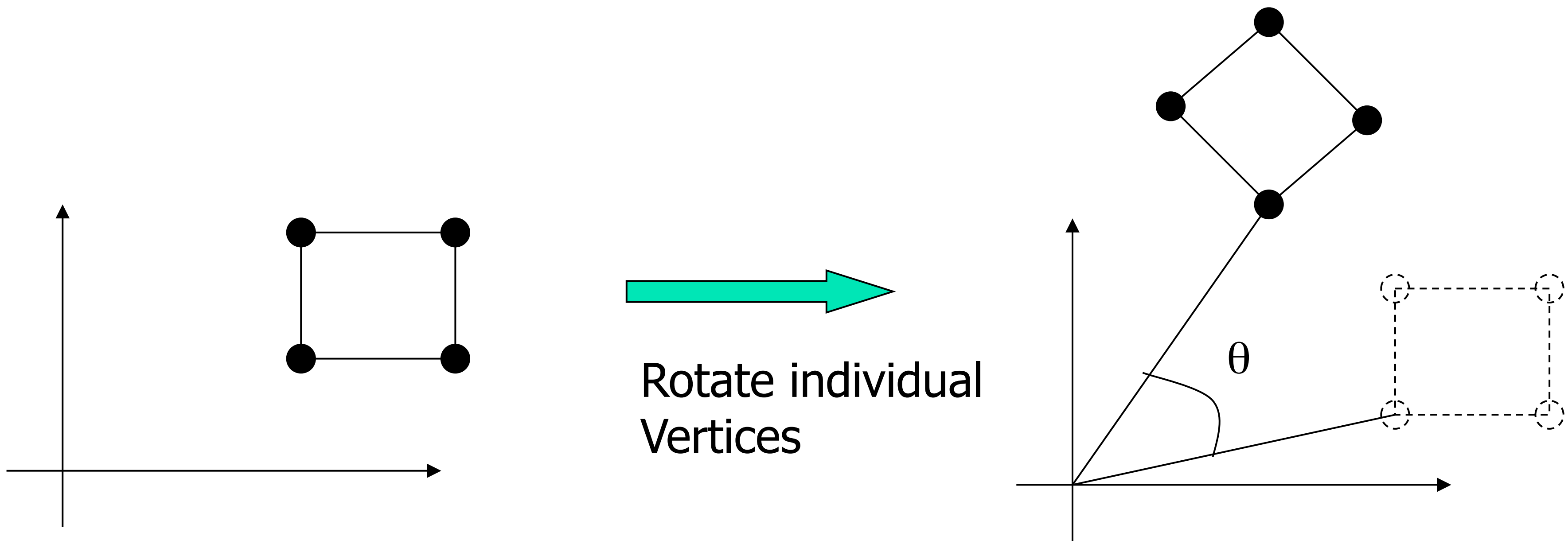
$$\begin{vmatrix} x' \\ y' \end{vmatrix} = \begin{vmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{vmatrix} \begin{vmatrix} x \\ y \end{vmatrix}$$



$$\begin{vmatrix} x' \\ y' \\ 1 \end{vmatrix} = \begin{vmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} x \\ y \\ 1 \end{vmatrix}$$

Rotation

- How to rotate an object with multiple vertices?

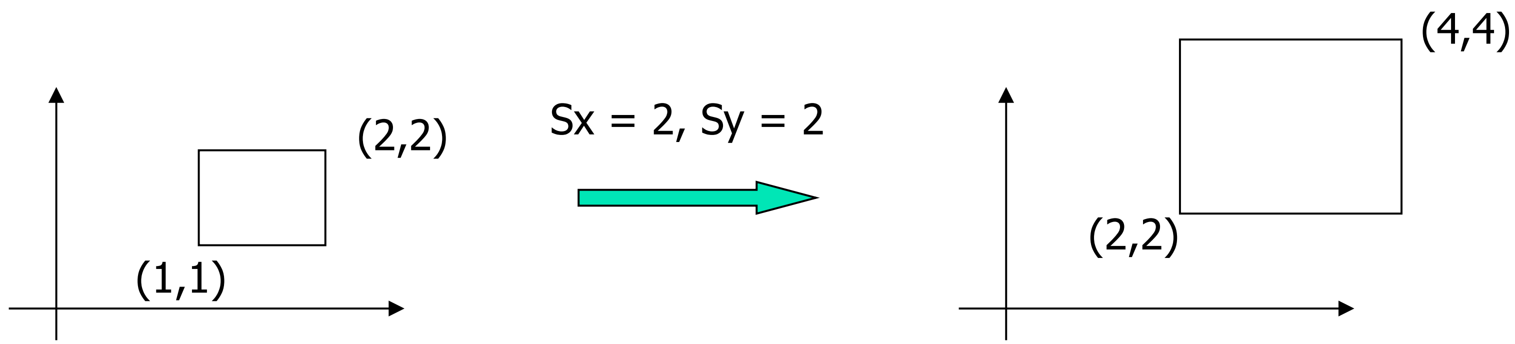


2D Scaling

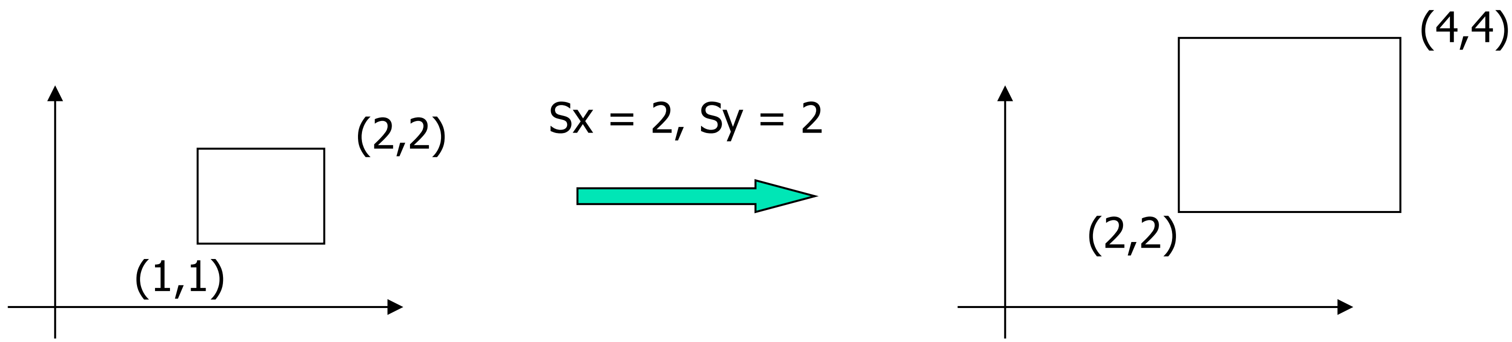
Scale: Alter the size of an object by a scaling factor (Sx, Sy), i.e.

$$x' = x \cdot Sx$$
$$y' = y \cdot Sy$$

$$\begin{vmatrix} x' \\ y' \end{vmatrix} = \begin{vmatrix} Sx & 0 \\ 0 & Sy \end{vmatrix} \begin{vmatrix} x \\ y \end{vmatrix}$$



2D Scaling



- Not only the object size is changed, it also moved!!
- Usually this is an undesirable effect
- We will discuss later (soon) how to fix it

3x3 2D Scaling Matrix

$$\begin{vmatrix} x' \\ y' \end{vmatrix} = \begin{vmatrix} Sx & 0 \\ 0 & Sy \end{vmatrix} \quad \begin{vmatrix} x \\ y \end{vmatrix}$$

$$\Downarrow$$

$$\begin{vmatrix} x' \\ y' \\ 1 \end{vmatrix} = \begin{vmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{vmatrix} * \begin{vmatrix} x \\ y \\ 1 \end{vmatrix}$$

Put it all together

- Translation: $\begin{vmatrix} x' \\ y' \end{vmatrix} = \begin{vmatrix} x \\ y \end{vmatrix} + \begin{vmatrix} tx \\ ty \end{vmatrix}$

- Rotation: $\begin{vmatrix} x' \\ y' \end{vmatrix} = \begin{vmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{vmatrix} * \begin{vmatrix} x \\ y \end{vmatrix}$

- Scaling: $\begin{vmatrix} x' \\ y' \end{vmatrix} = \begin{vmatrix} Sx & 0 \\ 0 & Sy \end{vmatrix} * \begin{vmatrix} x \\ y \end{vmatrix}$

Or, 3x3 Matrix representations

- Translation: $$\begin{vmatrix} x' \\ y' \\ 1 \end{vmatrix} = \begin{vmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{vmatrix} * \begin{vmatrix} x \\ y \\ 1 \end{vmatrix}$$
- Rotation: $$\begin{vmatrix} x' \\ y' \\ 1 \end{vmatrix} = \begin{vmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{vmatrix} * \begin{vmatrix} x \\ y \\ 1 \end{vmatrix}$$
- Scaling: $$\begin{vmatrix} x' \\ y' \\ 1 \end{vmatrix} = \begin{vmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{vmatrix} * \begin{vmatrix} x \\ y \\ 1 \end{vmatrix}$$

Why use 3x3 matrices?

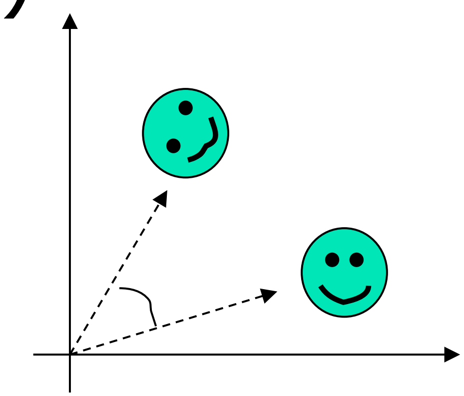
Why use 3x3 matrices?

- So that we can perform all transformations using matrix/vector multiplications
- This allows us to *pre-multiply* all the matrices together
- The point (x,y) needs to be represented as (x,y,1) -> this is called Homogeneous coordinates!

Rotation Revisit

- The standard rotation matrix is used to rotate about the origin (0,0)

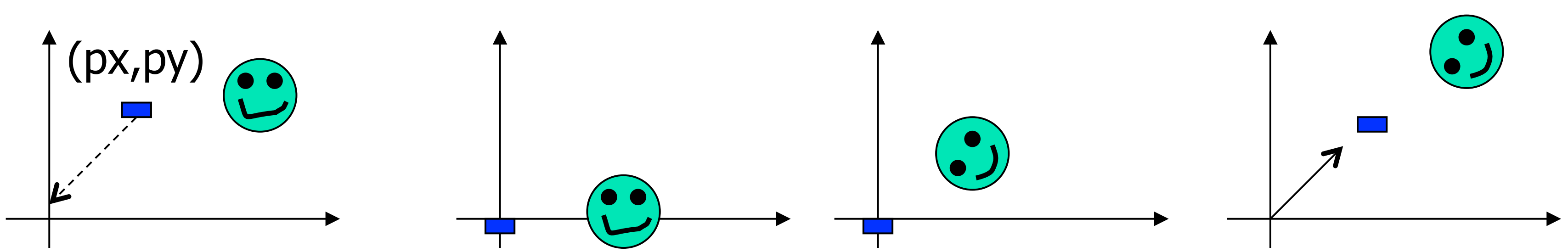
$$\begin{matrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{matrix}$$



- What if I want to rotate about an arbitrary center?

Arbitrary Rotation Center

- To rotate about an arbitrary point P (px,py) by θ:
  - Translate the object so that P will coincide with the origin: T(-px, -py)
  - Rotate the object: R(θ)
  - Translate the object back: T(px,py)



Arbitrary Rotation Center

- Translate the object so that P will coincide with the origin: T(-px, -py)
- Rotate the object: R(θ)
- Translate the object back: T(px,py)

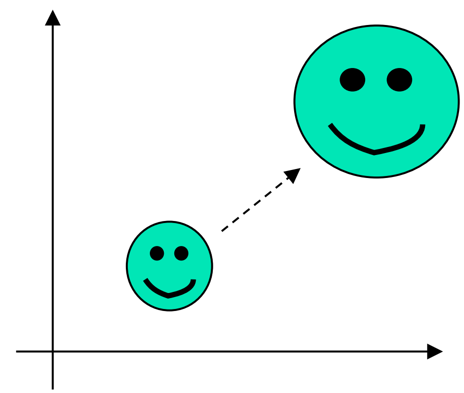
- Put in matrix form: T(px,py) R(θ) T(-px, -py) * P

$$
\begin{vmatrix} x' \\ y' \\ 1 \end{vmatrix} = \begin{vmatrix} 1 & 0 & px \\ 0 & 1 & py \\ 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} 1 & 0 & -px \\ 0 & 1 & -py \\ 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} x \\ y \\ 1 \end{vmatrix}
$$

Scaling Revisit

- The standard scaling matrix will only anchor at (0,0)

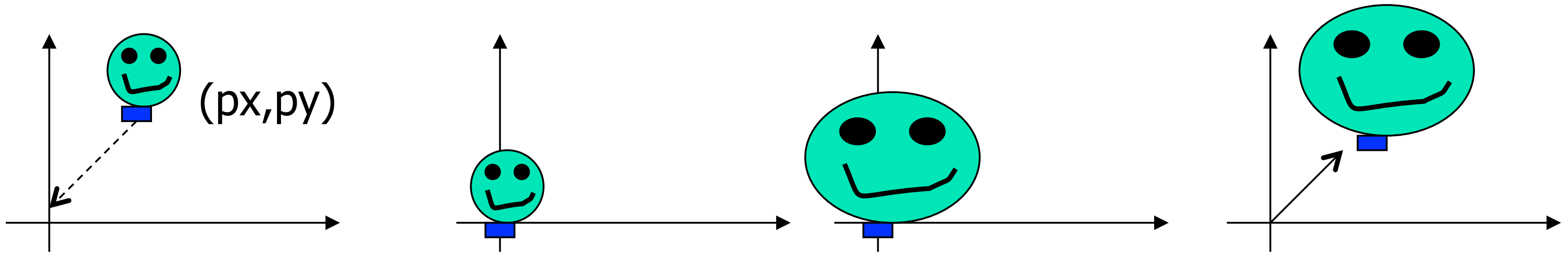
$$\begin{matrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{matrix}$$



- What if I want to scale about an arbitrary pivot point?

Arbitrary Scaling Pivot

- To scale about an arbitrary pivot point P (px,py):
  - Translate the object so that P will coincide with the origin: T(-px, -py)
  - Rotate the object: S(sx, sy)
  - Translate the object back: T(px,py)



Affine Transformation

- Translation, Scaling, Rotation, Shearing are all affine transformation
- Affine transformation – transformed point P' (x',y') is a linear combination of the original point P (x,y), i.e.

$$\begin{vmatrix} x' \\ y' \\ 1 \end{vmatrix} = \begin{vmatrix} m11 & m12 & m13 \\ m21 & m22 & m23 \\ 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} x \\ y \\ 1 \end{vmatrix}$$

- Any 2D affine transformation can be decomposed into a rotation, followed by a scaling, followed by a shearing, and followed by a translation.

  Affine matrix = translation x shearing x scaling x rotation

Composing Transformation

- Composing Transformation – the process of applying several transformation in succession to form one overall transformation
- If we apply transform a point P using M1 matrix first, and then transform using M2, and then M3, then we have:

$$(M3 \times (M2 \times (M1 \times P))) = M3 \times M2 \times M1 \times P$$

(pre-multiply) $M3 \times M2 \times M1 \rightarrow M$

Composing Transformation

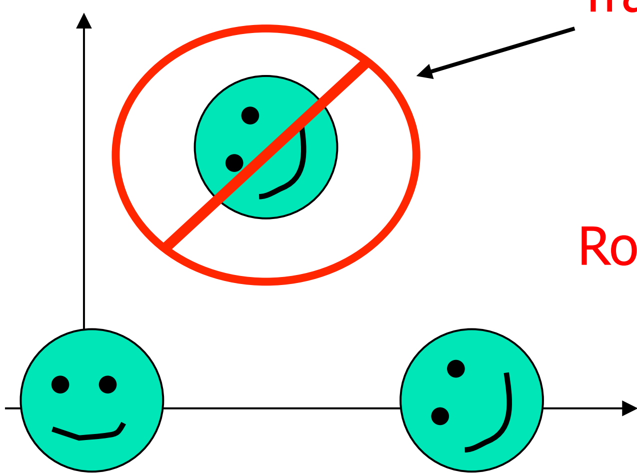
- Matrix multiplication is associative

  M3 x M2 x M1 = (M3 x M2) x M1 = M3 x (M2 x M1)
- Transformation products may not be commutative   A x B != B x A
- Some cases where A x B = B x A

A	B
translation	translation
scaling	scaling
rotation	rotation
uniform scaling (sx = sy)	rotation

Transformation order matters!

- Example: rotation and translation are not commutative



Translate (5,0) and then Rotate 60 degree

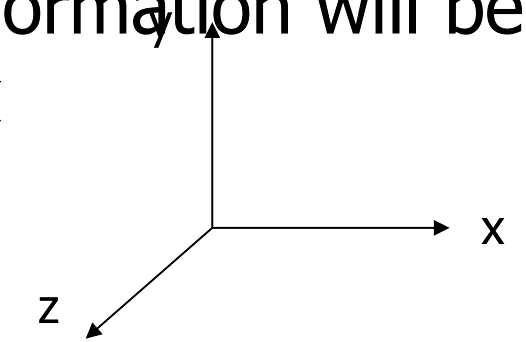
OR

Rotate 60 degree and then translate (5,0)??

Rotate and then translate !!

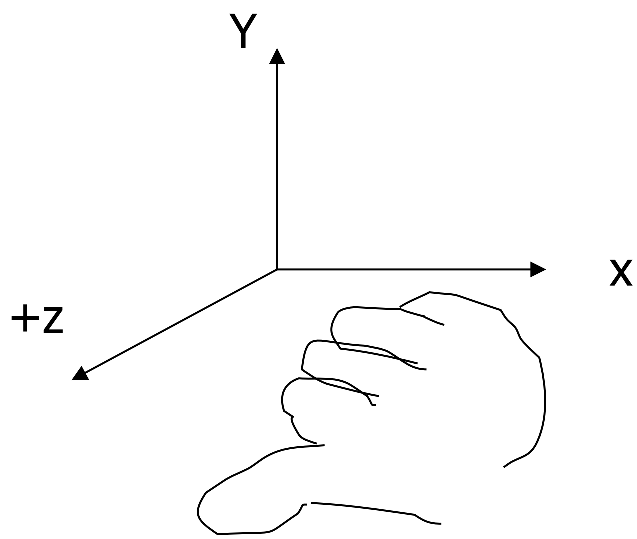
Three-Dimensional Graphics

- A 3D point (x,y,z) – x,y, and Z coordinates
- We will still use column vectors to represent points
- Homogeneous coordinates of a 3D point (x,y,z,1)
- Transformation will be performed using 4x4 matrix

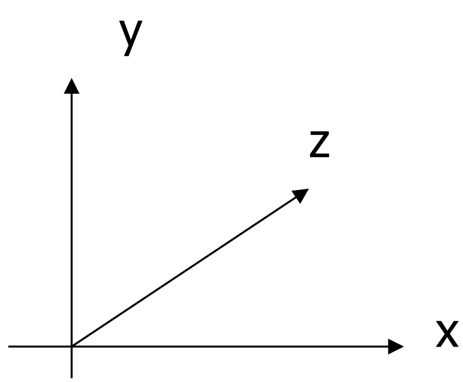


Right hand coordinate system

- $X \times Y = Z$; $Y \times Z = X$; $Z \times X = Y$;



Right hand coordinate system



Left hand coordinate system
Not used in this class and
Not in OpenGL

3D transformation

- Very similar to 2D transformation
- Translation

  $x' = x + tx;\ y' = y + ty;\ z' = z + tz$

$$\begin{vmatrix} X' \\ Y' \\ Z' \\ 1 \end{vmatrix} = \begin{vmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} X \\ Y \\ Z \\ 1 \end{vmatrix}$$

homogeneous coordinates

3D transformation

- Scaling

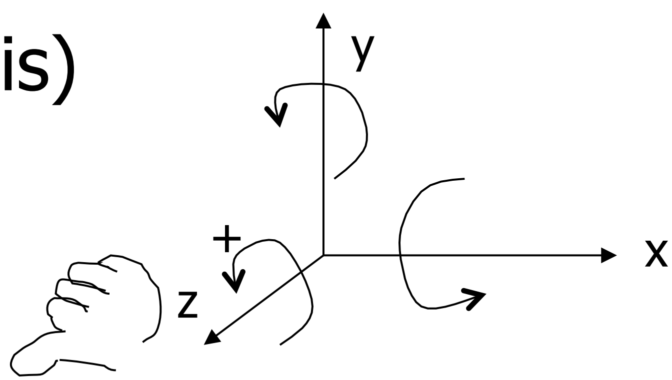
  X' = X * Sx; Y' = Y * Sy; Z' = Z * Sz

$$\begin{vmatrix} X' \\ Y' \\ Z' \\ 1 \end{vmatrix} = \begin{vmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} X \\ Y \\ Z \\ 1 \end{vmatrix}$$

3D transformation

- 3D rotation is done around a rotation **axis**
- Fundamental rotations – rotate about x, y, or z axes
- Counter-clockwise rotation is referred to as positive rotation (when you

  look down negative axis)

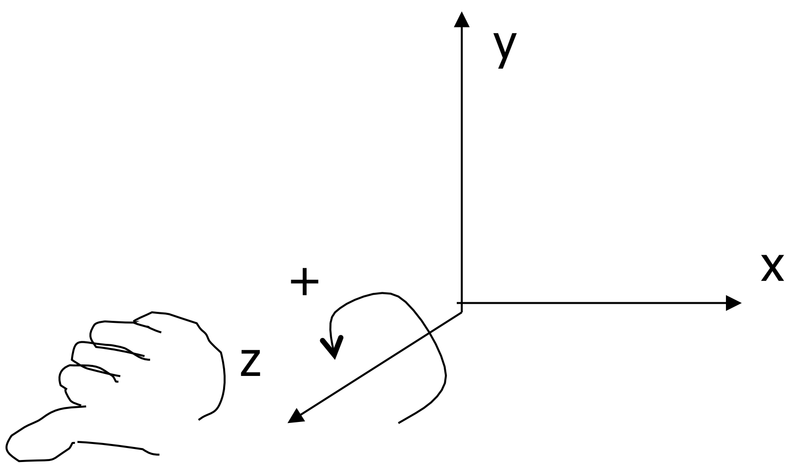


3D transformation

- Rotation about Z – similar to 2D rotation

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

$$\begin{vmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$



- OpenGL - glRotatef(θ, 0,0,1)

3D transformation

- Rotation about y (z -> y, y -> x, x->z)

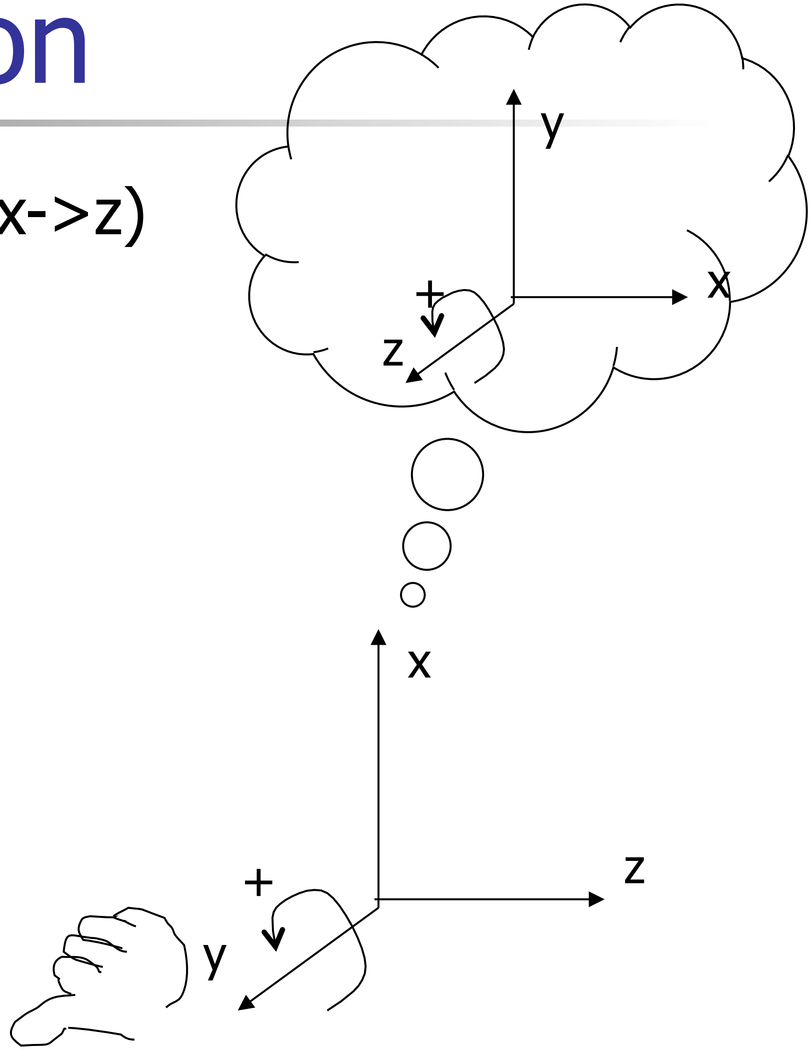
$$z' = z\cos(\theta) - x\sin(\theta)$$

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

$$y' = y$$

$$\begin{vmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

- OpenGL - glRotatef(θ, 0,1,0)



3D transformation

- Rotation about x (z -> x, y -> z, x->y)

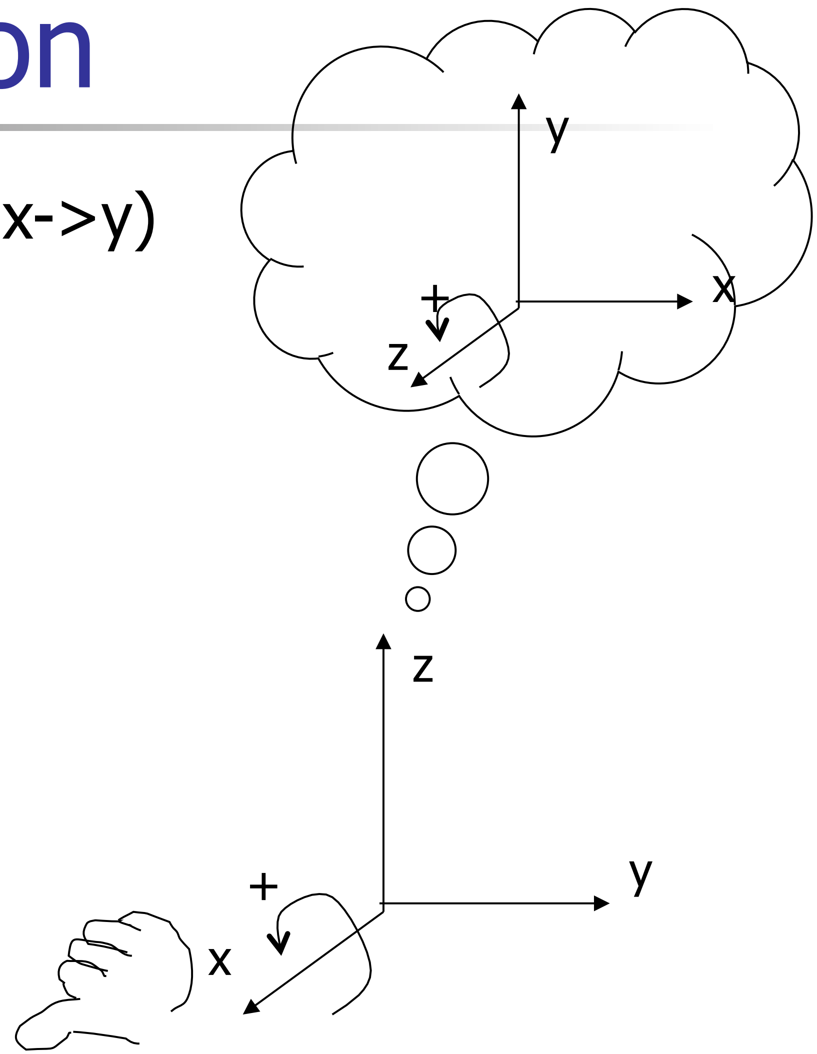
$$y' = y\cos(\theta) - z\sin(\theta)$$

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

$$x' = x$$

$$\begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

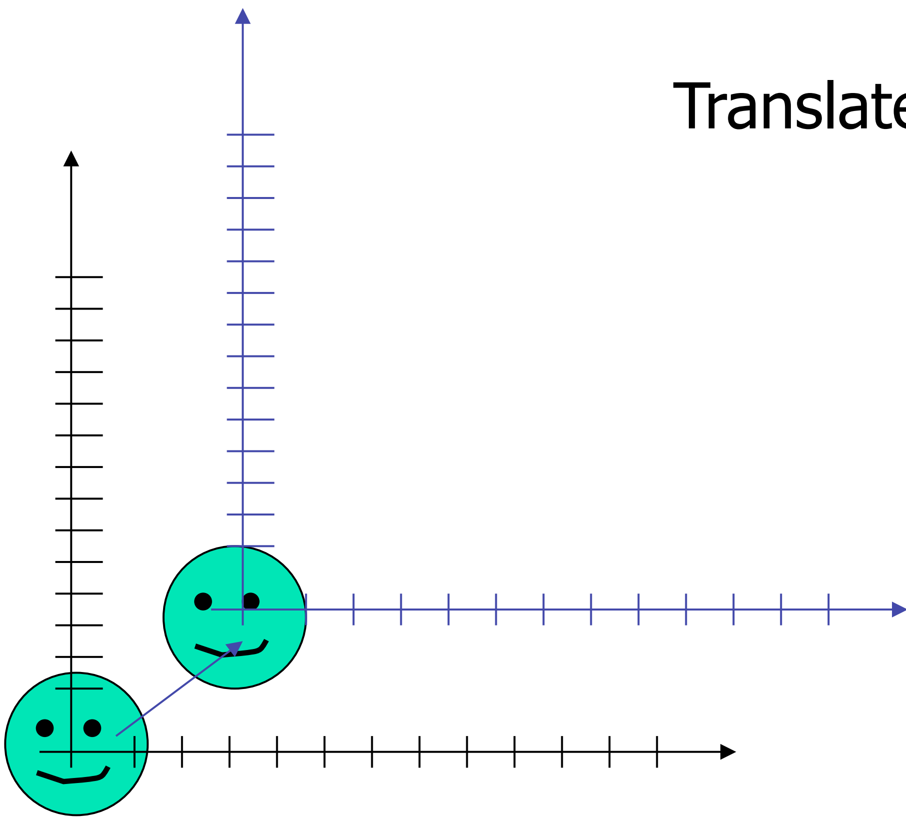
- OpenGL - glRotatef(θ, 1,0,0)



Composing Transformation

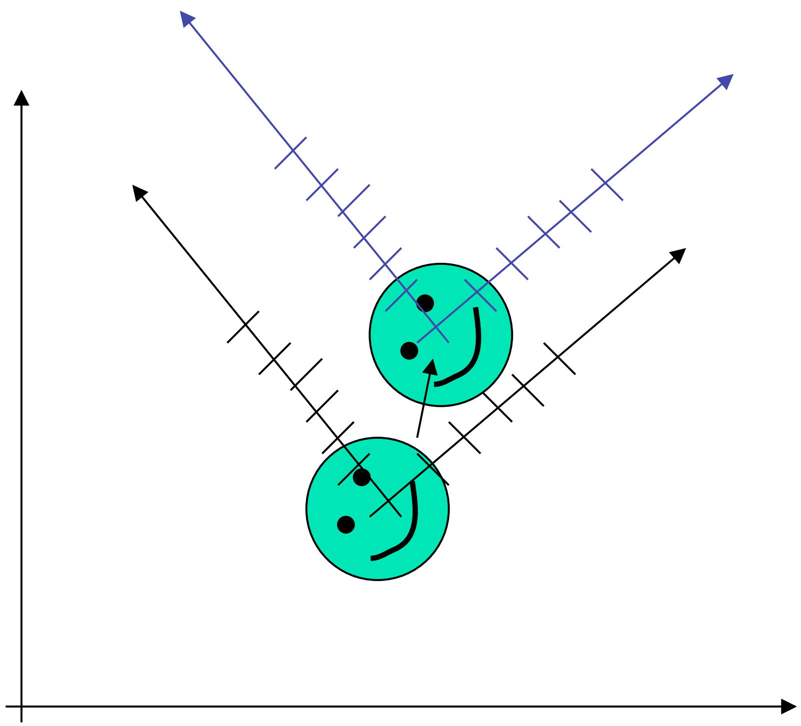
- You can think of object transformations as moving (transforming) its local coordinate frame
- All the transformations are performed relative to the current coordinate frame origin and axes

Translate Coordinate Frame



Translate (3,3)?

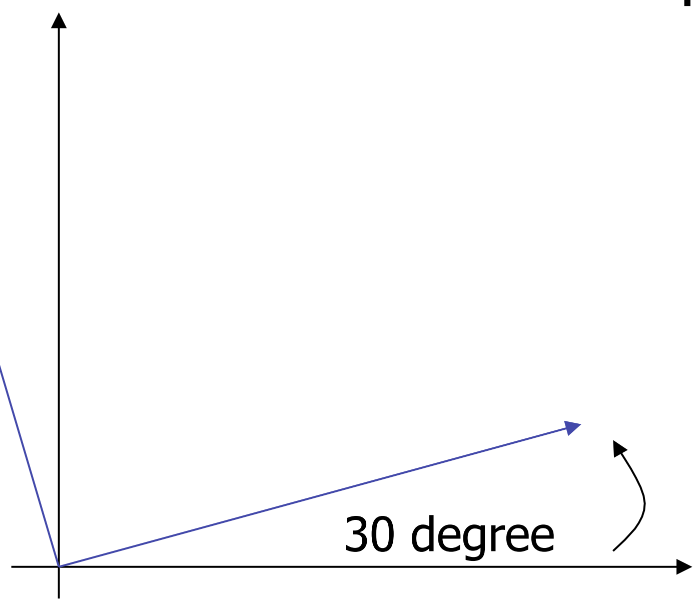
Translate Coordinate Frame (2)



Translate (3,3)?

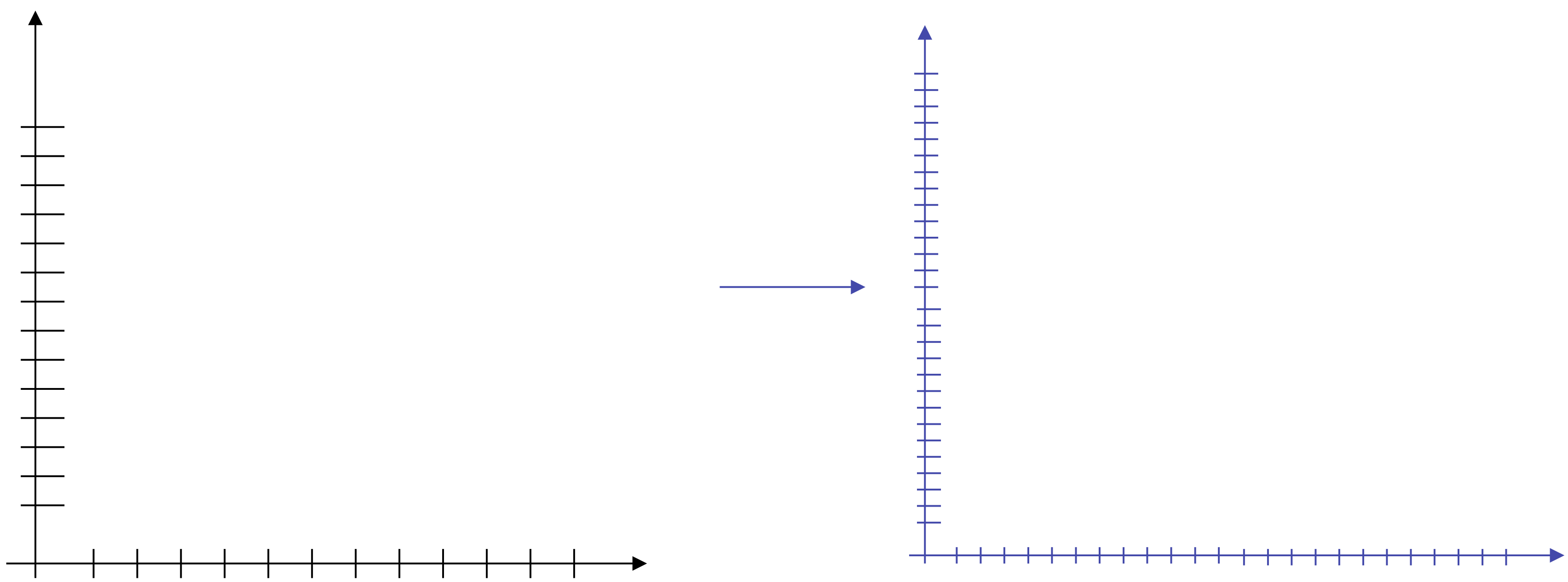
Rotate Coordinate Frame

Rotate 30 degree?

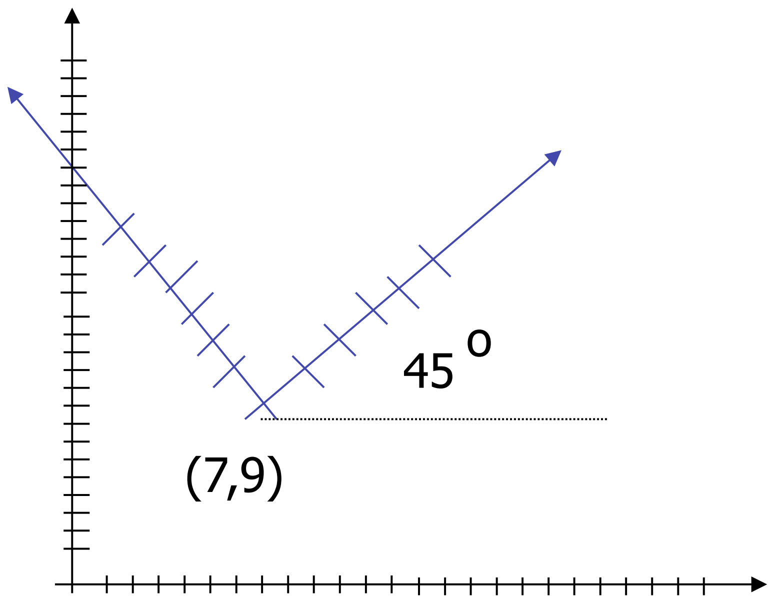


Scale Coordinate Frame

Scale (0.5,0.5)?



Compose Transformations

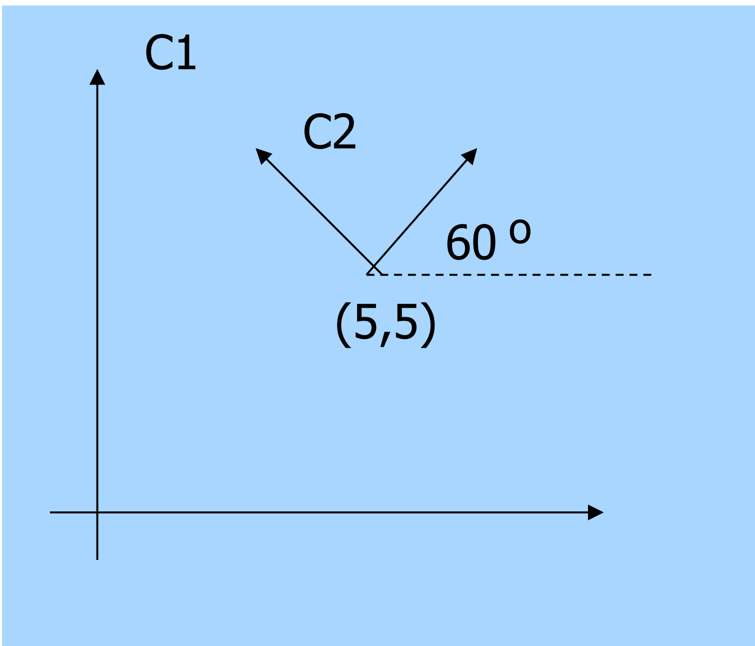


Transformations?

Answer:

1. Translate(7,9)
2. Rotate 45
3. Scale (2,2)

Another example



How do you transform from C1 to C2?

Translate (5,5) and then Rotate (60)

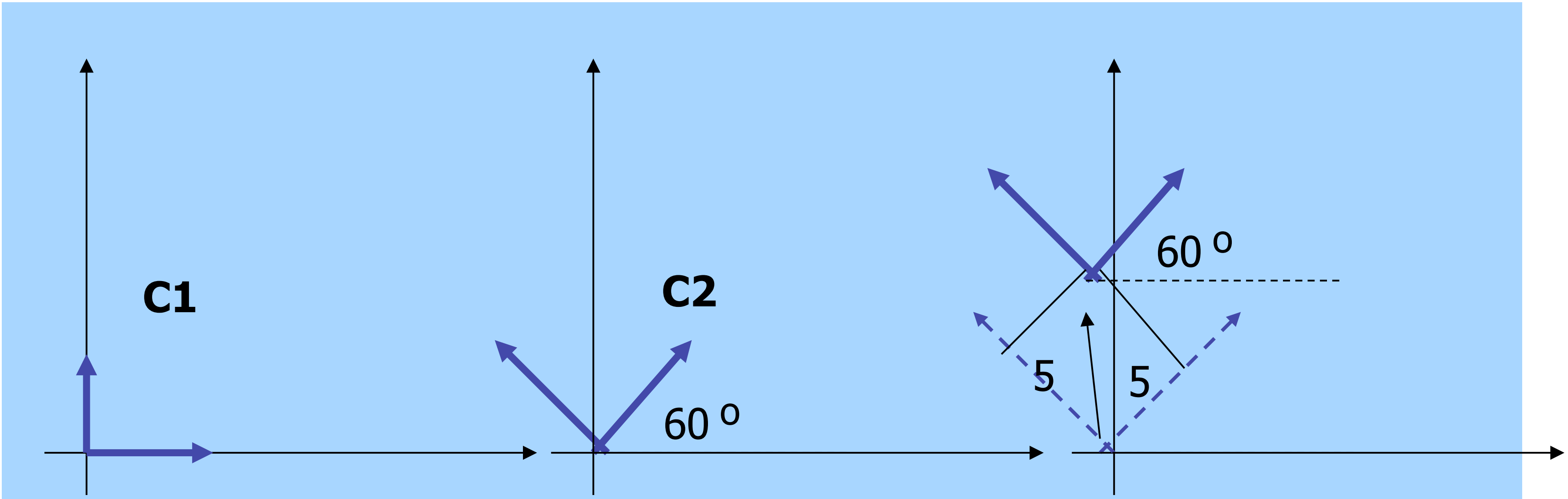
OR

Rotate (60) and then Translate (5,5) ???

Answer: Translate(5,5) and then Rotate (60)

Another example (cont'd)

If you Rotate(60) and then Translate(5,5) …

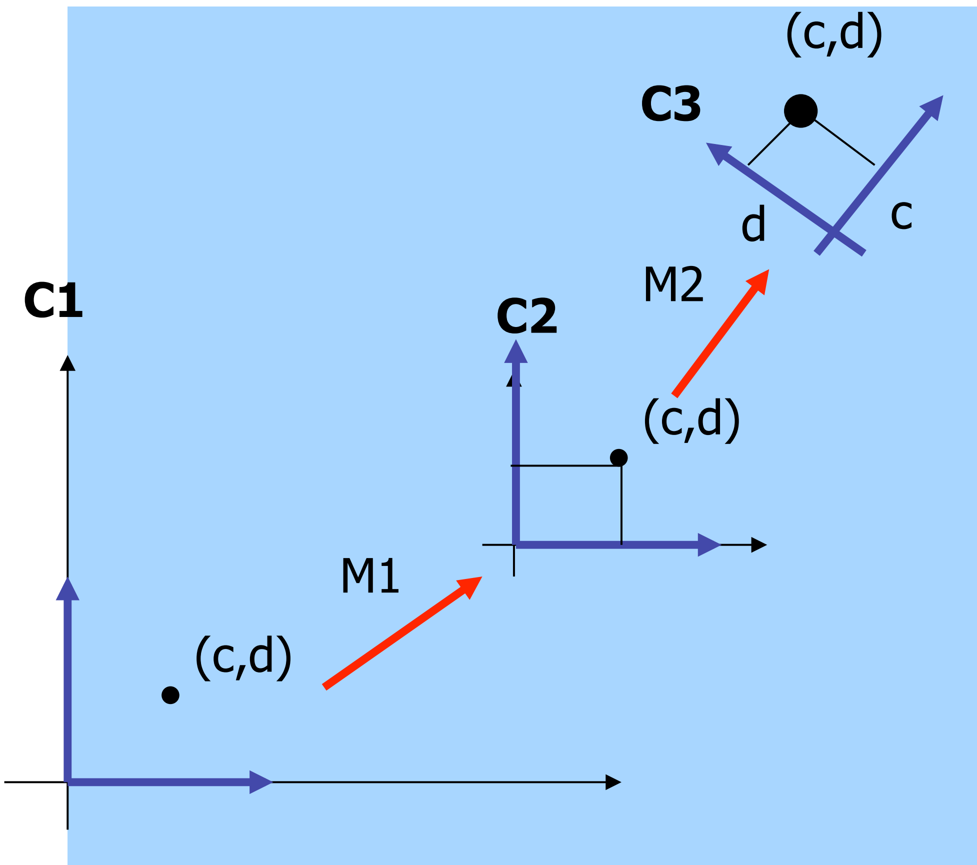


You will be translated (5,5) relative to C2!!

Transform Objects

- What does moving coordinate frames have anything to do with object transformation?
  - You can view transformation as to tie the object to a local coordinate frame and move that coordinate frame

Compose transformation



Multiply the matrix from left to right

M1 (move C1 to C2)
M2 (move C2 to C3′, without rotation)
M3 (rotate C3′ to C3)

P’s final coordinates =

M1 x M2 x M3 x P