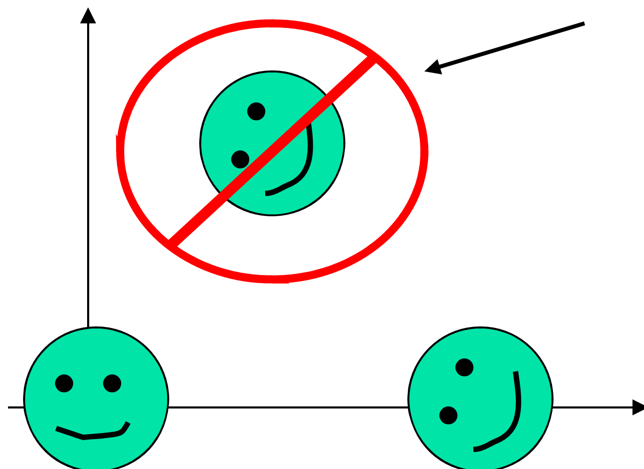
Composing Transformations

- Typically you need a sequence of transformations to position your objects
  - e.g., a combination of rotations and translations
- The order you apply transformations matters!
  - e.g. rotation and translations are not commutative



Translate (5,0) and then Rotate 60 degree

OR

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

Rotate and then translate !!

Composing Transformations - Notation

- Below we will use the following convention to explain transformations

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{pmatrix} m_{11} & m_{12} & m_{13} & 0 \\ m_{21} & m_{22} & m_{23} & 0 \\ m_{31} & m_{32} & m_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Matrix applied to left of vector

Column vector as a point

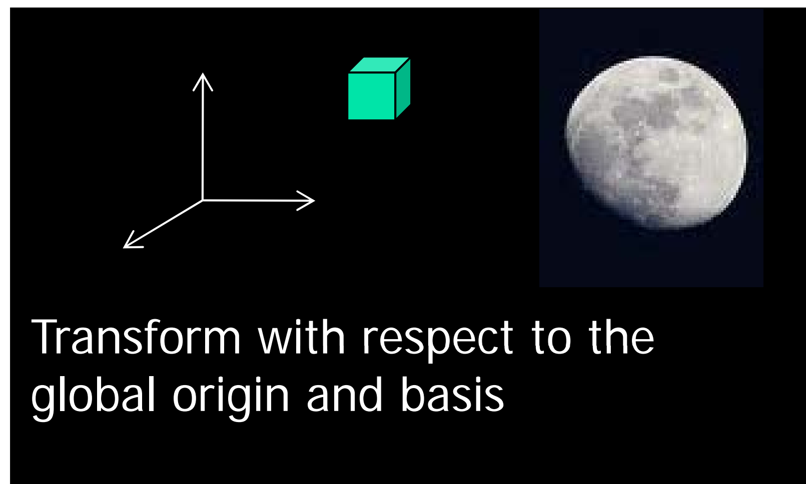
I am not concerned with how the matrix/vector is stored here – just focused on mathematics (but for your information, OpenGL fixed function pipeline stores matrices in column major order, i.e., m[0][0] = $m_{11,}$ m[0][1] = $m_{21,}$ m[0][2] = $m_{31,}$ …)

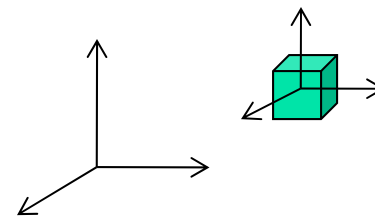
Composing Transformations - Concatenation

- There are two ways to concatenate transformation matrices
  - Pre- and Post-multiplication
- Pre-multiplication is to multiply the new matrix (B) to the left of the existing matrix (A) to get the result (C)
  - C = B * A
- Post-multiplication is to multiply the new matrix (B) to the right of the existing matrix (B)
  - C = A * B
- Which one you choose depends on what you do
  - OpenGL fixed function pipeline uses **post-multiplication**. I will explain why (and this is what we will use)

Two ways to think of a sequence of transformations

- How you think of it determines how you should concatenate the transformation matrices together (pre- or post-multiplication)
- Both will work but sometimes one is more convenient than the other

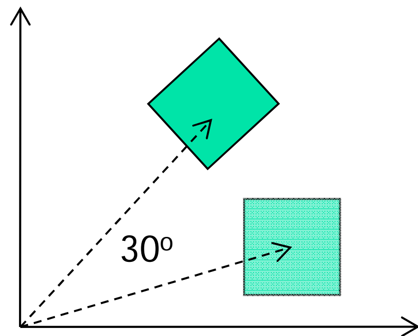




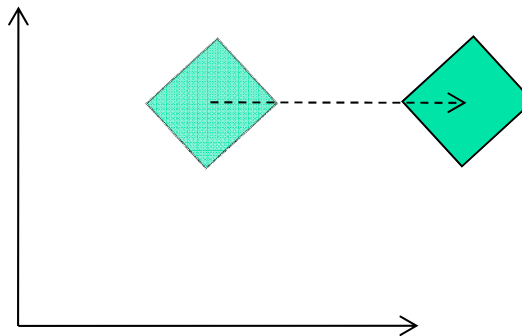
Transform with respect to the local Origin and basis

Think of transformations with respect to the global (world) coordinate system

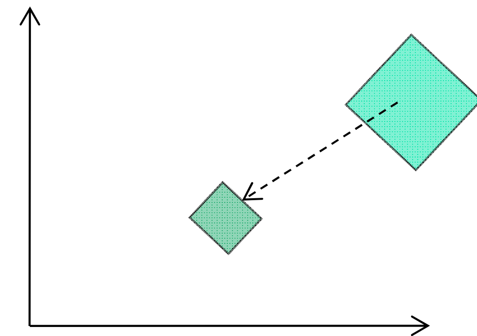
- Everything you do is relative to the global origin and the basis



Step 1 (R):
Rotated by 30°



Step 2 (T):
Translated by (2,0)

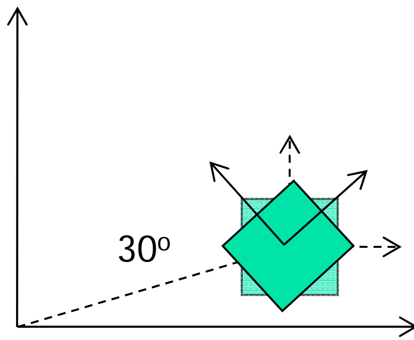


Step 3 (S):
Scaled by (0.5,0.5)

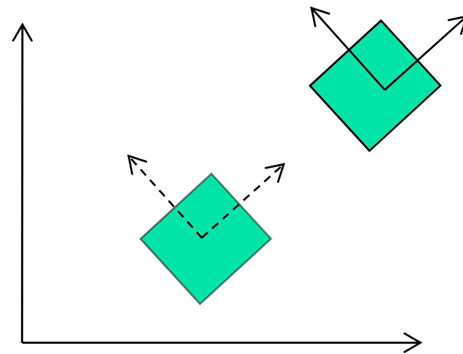
- In this case, you should pre-multiply the matrices together $v' = M \times v = S \times T \times R \times v$

Think of transformations as transforming the local coordinate frame

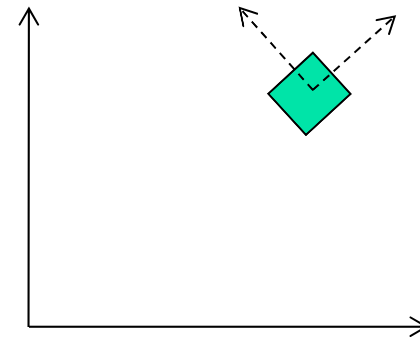
- Every thing you do is affecting the position and the orientation of the local coordinate frame



Step 1 (R):
Rotated by 30°



Step 2 (T):
Translated by (2,0)



Step 3 (S):
Scaled by (0.5,0.5)

- In this case, you should post-multiply the matrices

$v' = M \times v = R \times T \times S \times v$

Which way should I think?

- Of course, both ways will work so it is up to you
- Two methods will give you the transformation sequence in the opposite order
- It is generally much easier to control the object if you think of the transformations as moving the local coordinate frames
  - This is how the OpenGL fixed function pipelined does things!!
  - In other words, OpenGL fixed function pipeline use post-multiplication

OpenGL Post-Multiplication

- OpenGL post-multiplies each new transformation matrix

  $M = M \times M_{new}$

- Example: perform translation, then rotation

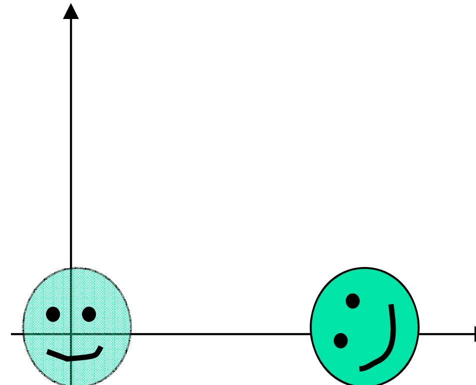
  0) M = Identity

  1) translation T(tx,ty,0) -> M = M x T(tx,ty,0)

  2) rotation $R(\theta)$ -> $M = M \times R(\theta)$

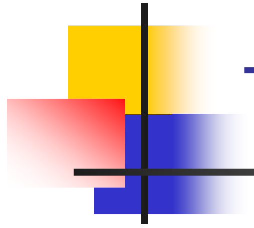
  3) Now, transform a point P -> P' = M x P

  $= T(tx, ty, 0) \times R(\theta) \times P$

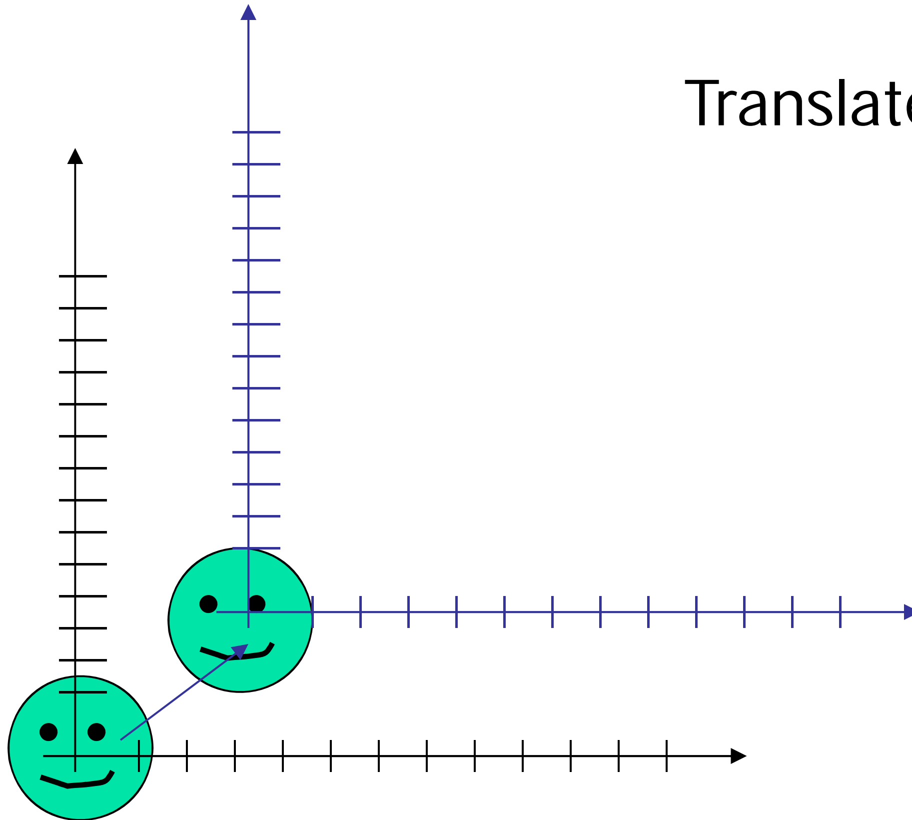


OpenGL Transformation

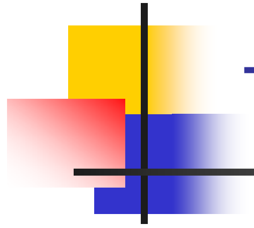
- When use OpenGL, you need to think of object transformations as moving its local coordinate frame
- All the transformations are performed relative to the current coordinate frame origin and basis



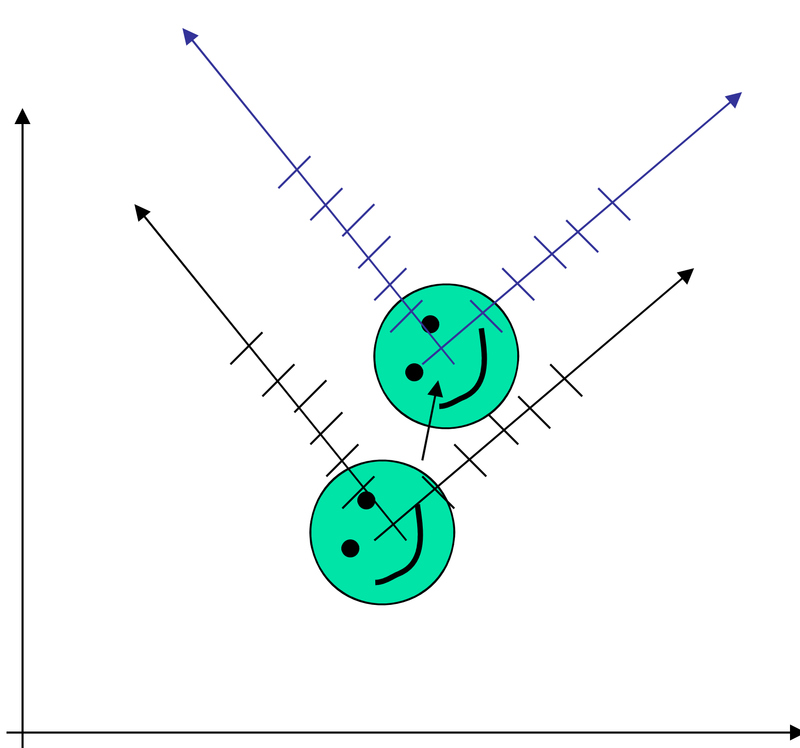
Translate Coordinate Frame



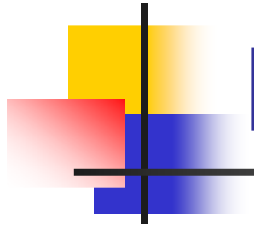
Translate (3,3)?



Translate Coordinate Frame (2)

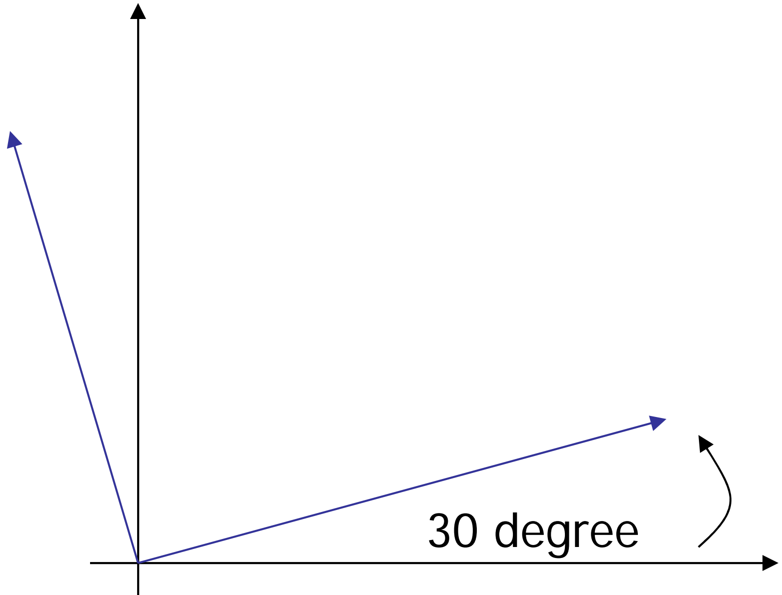


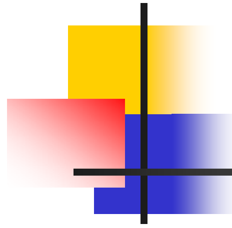
Translate (3,3)?



Rotate Coordinate Frame

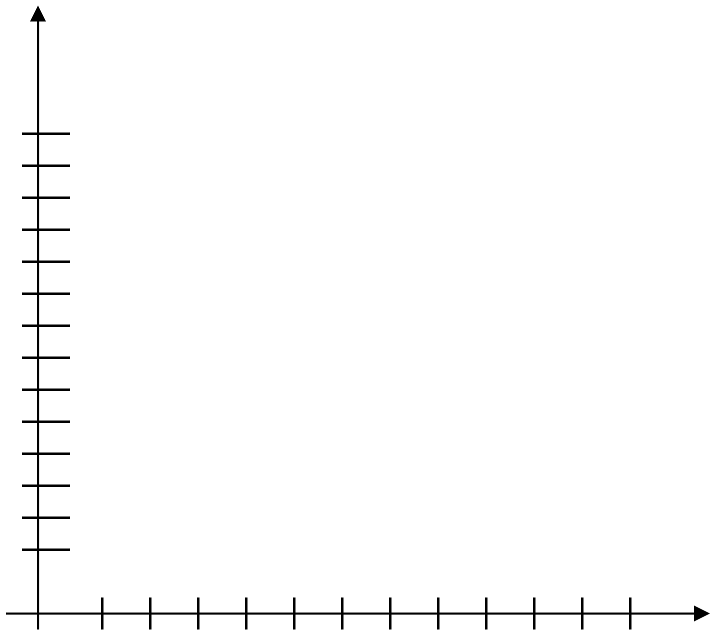
Rotate 30 degree?



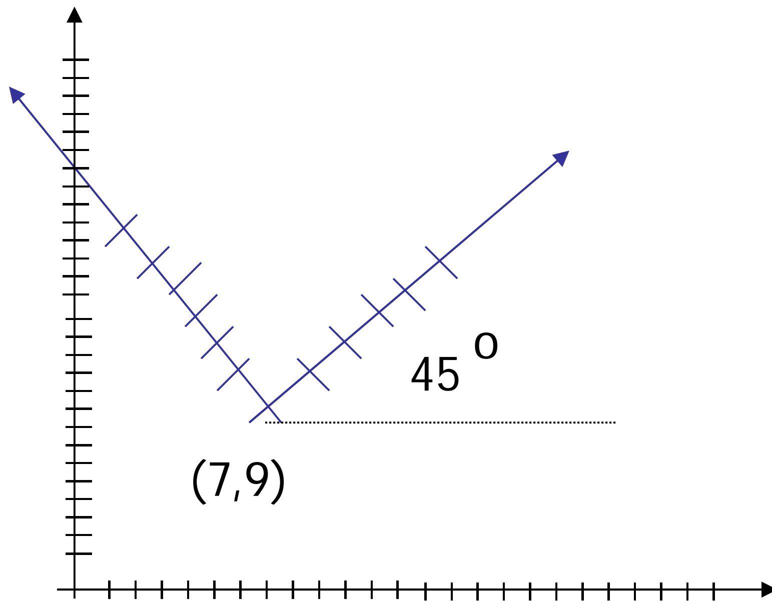


Scale Coordinate Frame

Scale (0.5,0.5)?



Compose Transformations

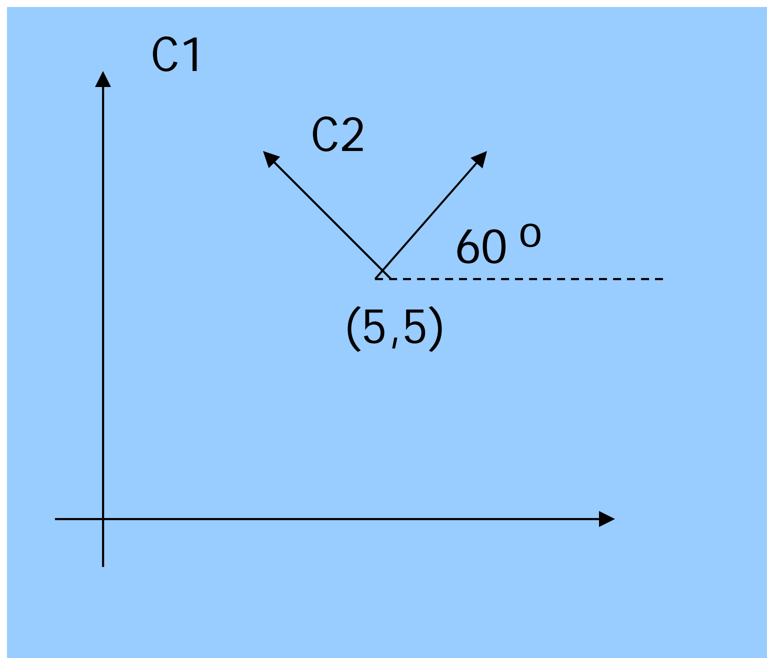


Transformations?

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)

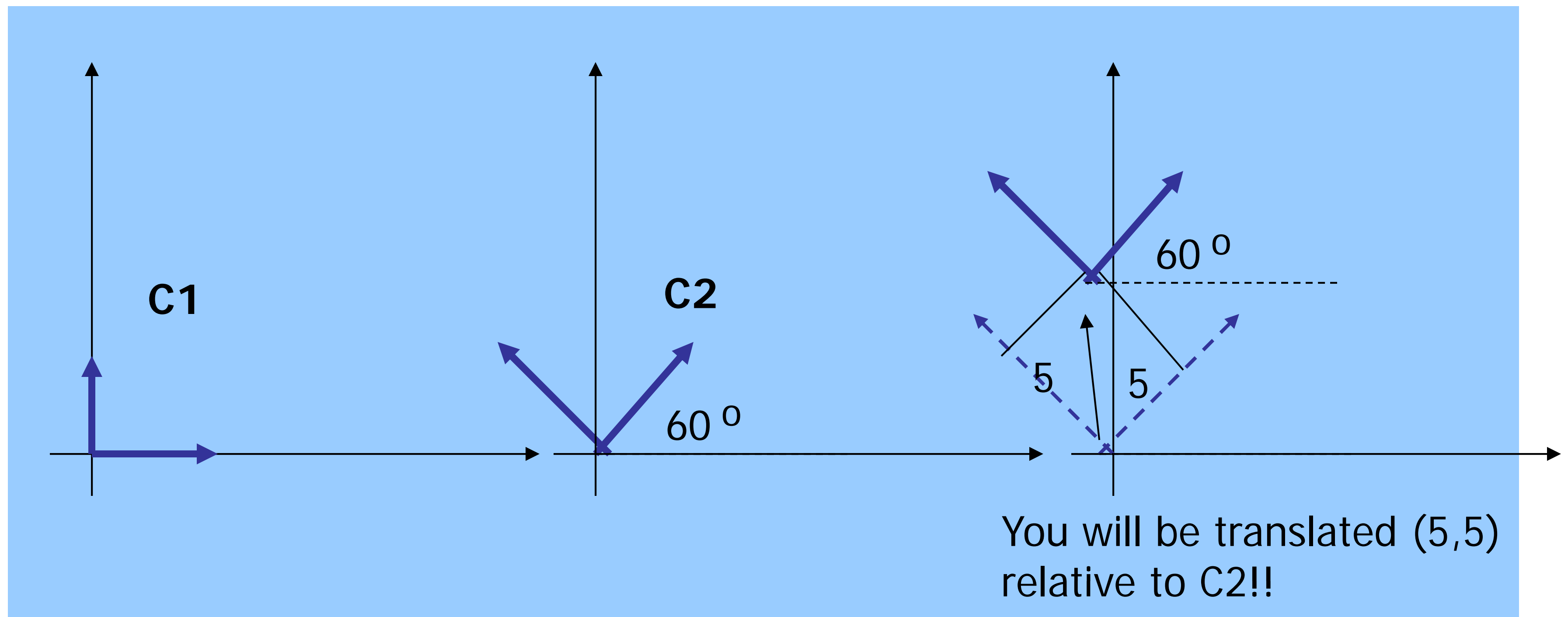
OR

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

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

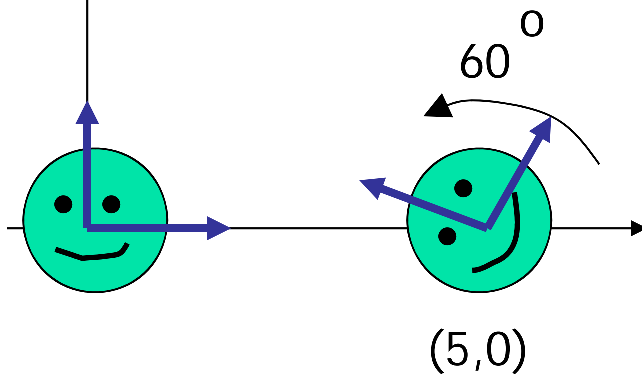
Another example (cont'd)

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



- What does coordinate frame transformation 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

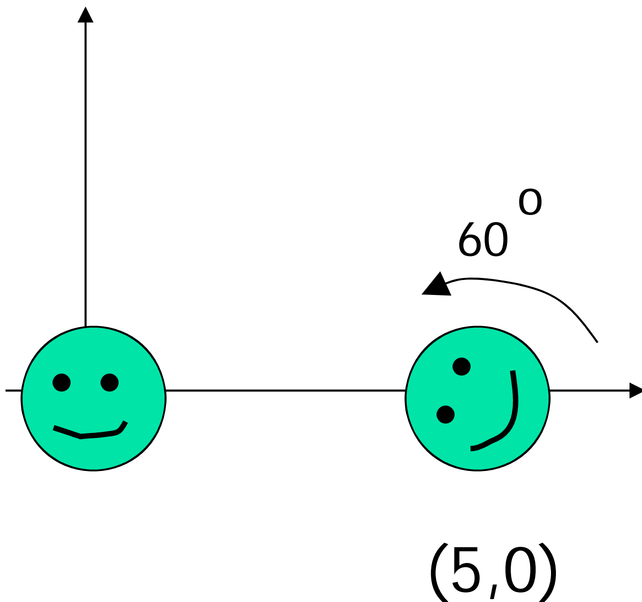
Example



Think of transformations as moving the local coordinate frame as Well as the object

1) Translate (5,0)
2) Rotate (60°)

If you think the other way



Transformation as moving the object relative to the origin of a global world coordinate frame

1) Rotate (60°)
2) Translate (5,0)

Exact the opposite order

When you use OpenGL …

- Think of transformation as moving coordinate frames
- Call OpenGL transformation functions in that order
- OpenGL does post-multiplication of matrices
- The accumulated matrix will be multiplied to your object vertices