Watch out!!

- 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(θ) -> M = M x R(θ)

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

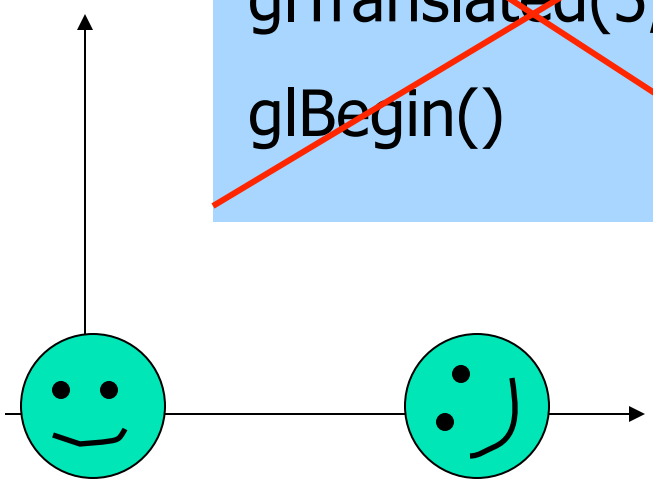
  = ~~T(tx, ty, 0) x R(θ) x P~~ Wrong!!!

Example Revisit

- We want rotation first and then translation
- Generate wrong results if you do:

~~glRotated(60,0,0,1);~~
~~glTranslated(5,0,0);~~
~~glBegin()~~

```
glTranslated(5,0,0);
glRotate(60,0,0,1);
glBegin()
...
```



You need to specify the transformation in the opposite order!!

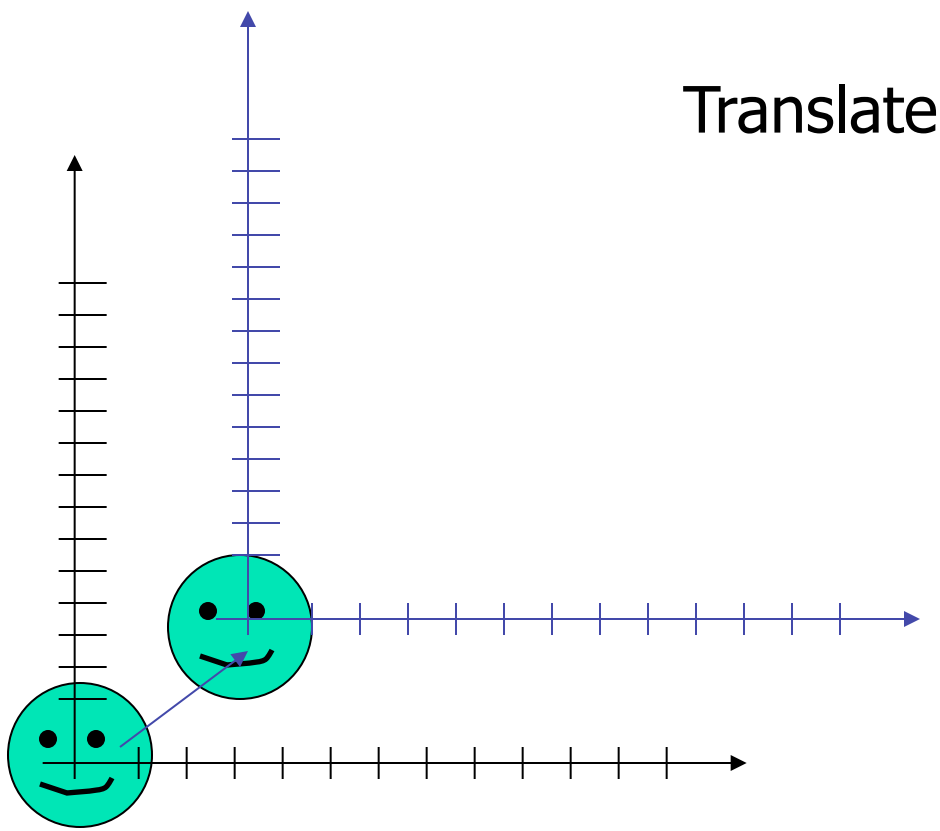
How Strange …

- OpenGL has its reason …
- It wants you to think of transformation in a different way
- Instead of thinking of transformation as moving the object in a fixed global coordinate system, you should think of transforming the object as moving (transforming) its local coordinate frame

OpenGL Transformation

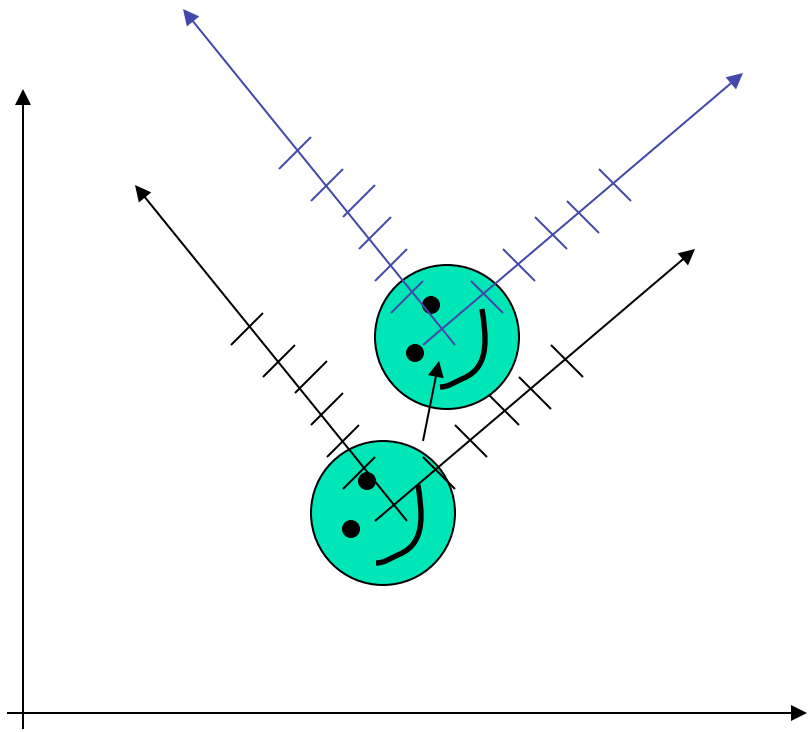
- When use OpenGL, we need to think of object transformations as moving (transforming) its local coordinate frame
- All the transformations are performed relative to the current local coordinate frame origin and axes

Translate Coordinate Frame



Translate (3,3)?

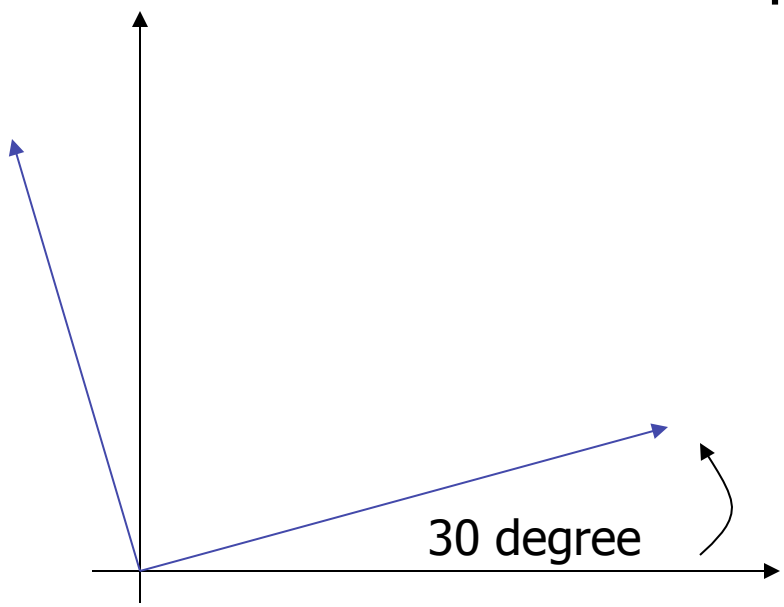
Translate Coordinate Frame (2)



Translate (3,3)?

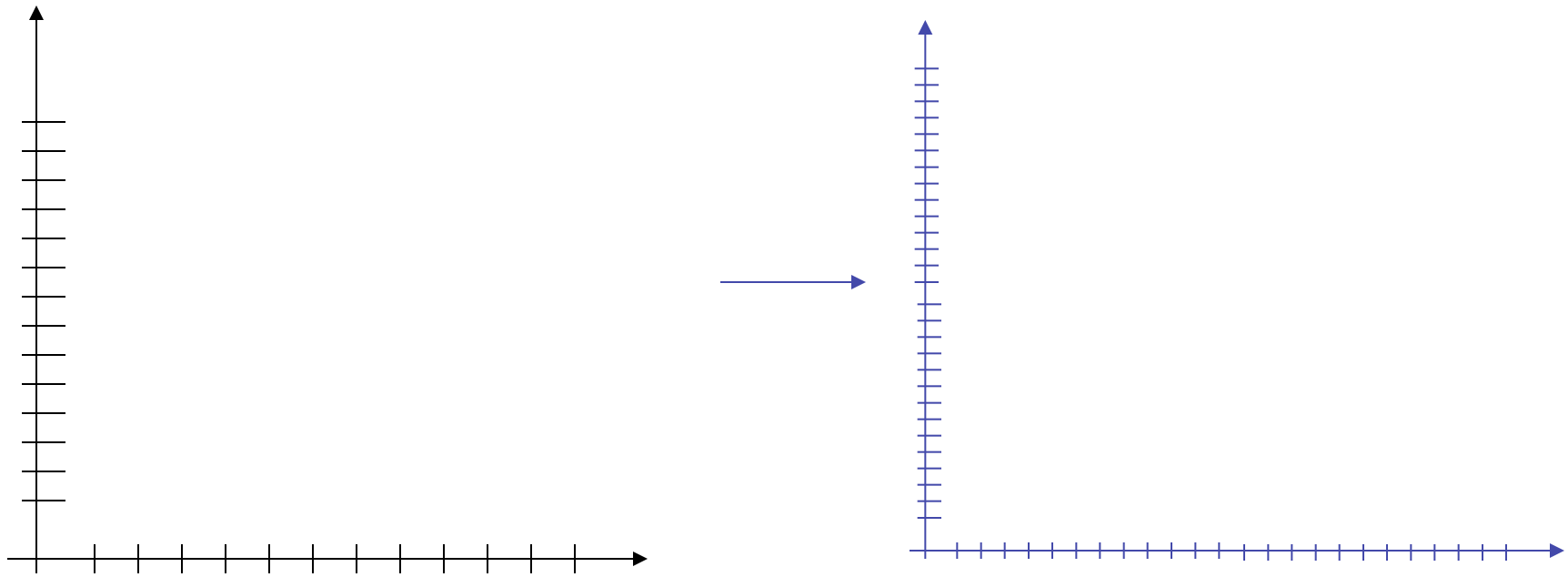
Rotate Coordinate Frame

Rotate 30 degree?

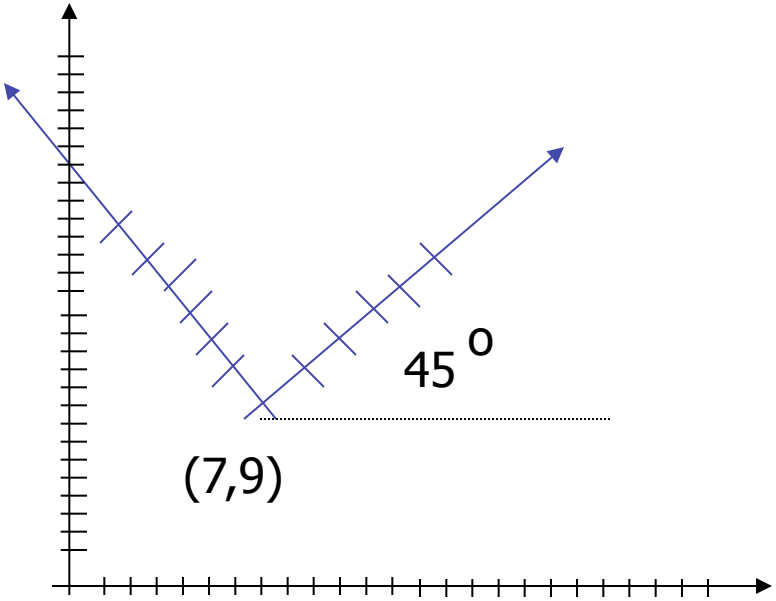


Scale Coordinate Frame

Scale (0.5,0.5)?



Compose Transformations

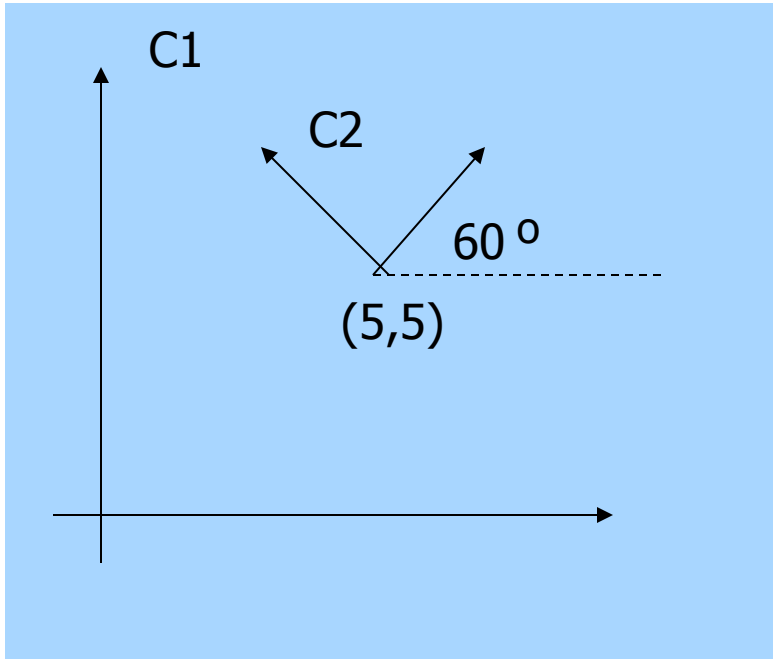


Transformations?

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)

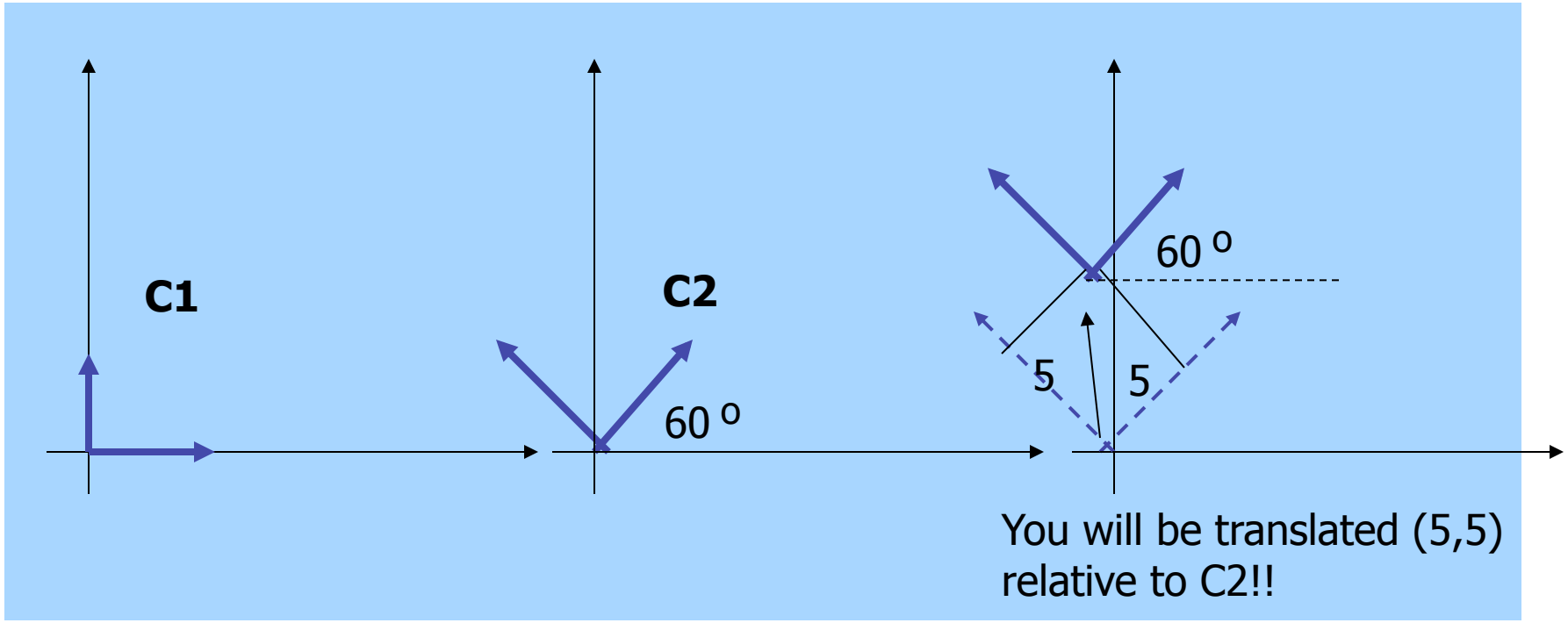
OR

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

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

Another example (cont'd)

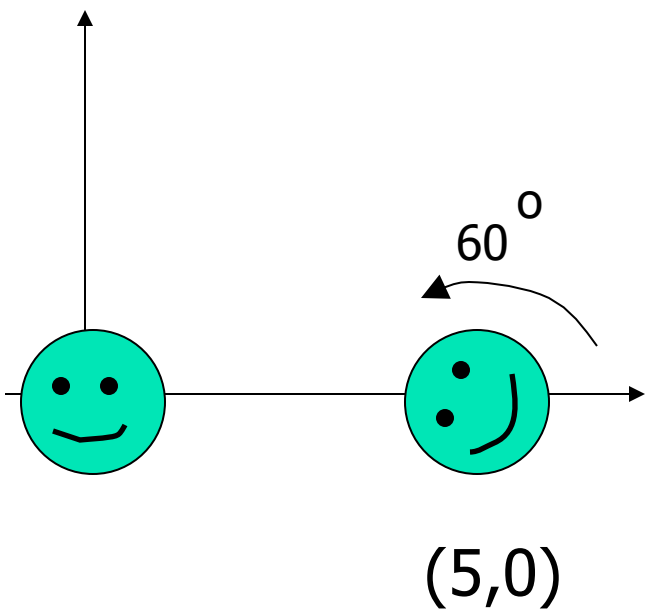
If you Rotate(60) and then Translate(5,5) …



Transform Objects

- What does coordinate frame transformation have anything to do with object transformation?
  - You can view object transformation as fixing the object to the local coordinate frame and then move that coordinate frame

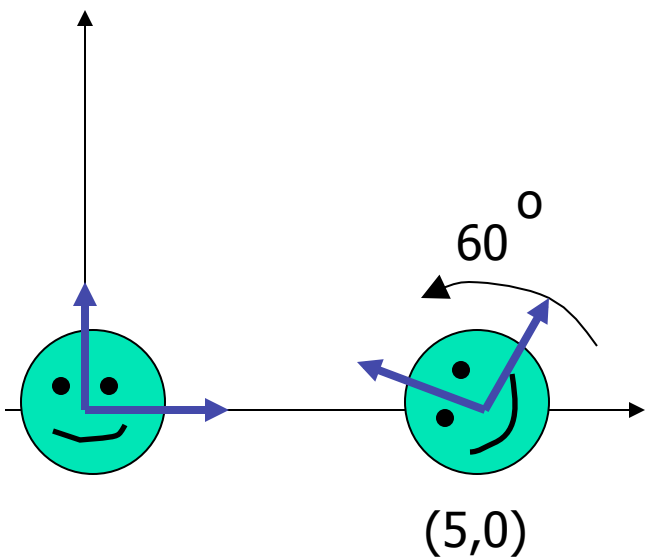
Example



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

1) Rotate (60°)
2) Translate (5,0)

Example (cont'd)



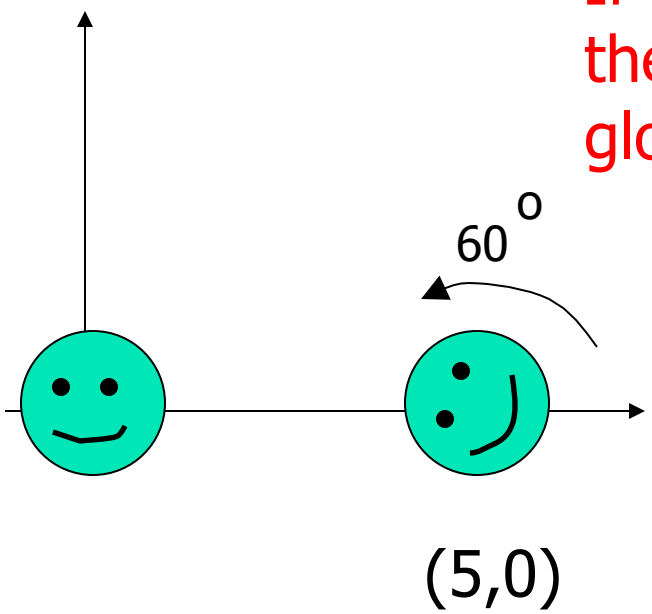
If you think of transformations as moving the local coordinate frame

1) Translate (5,0)
2) Rotate (60°)

Exact the opposite order compared to the previous slide!!

So ...

If you think of transformations as moving the object relative to the origin of a global world coordinate frame

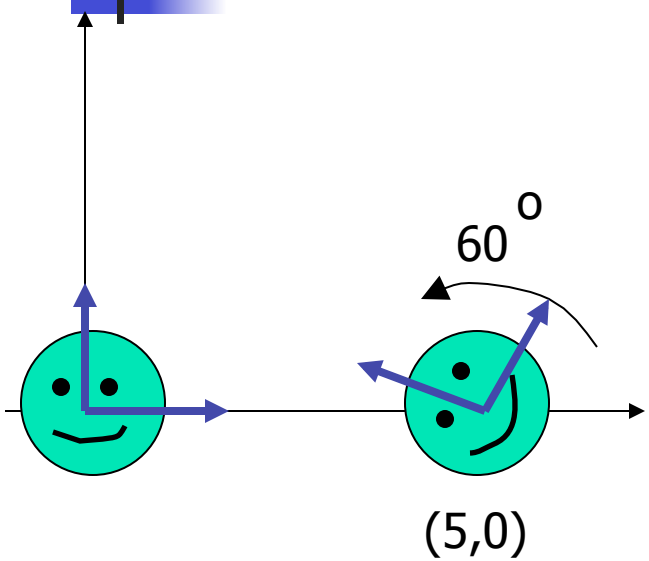


1) Rotate (60°) - M_R
2) Translate (5,0) - M_T

$P' = M_T \times M_R \times P$ is the Correct multiplication order

However, OpenGL will do $M_R \times M_T \times P$ if you call glRotate () first, and then glTranslate() because OpenGL does postmultiplication

So … (cont'd)



If you think of transformations as moving the coordinate frame

1) Translate (5,0) - M_T
2) Rotate (60°) - M_R

If you think in terms of moving coordinate frames, you will want to perform Translate first, and then Rotate (I.e., call glTranslate() first and then glRotate())

OpenGL will do $M_T \times M_R \times P$ -> The correct multiplication order!!!

Put it all together

When you use OpenGL …

- Think of transformation as moving the local coordinate frame
- Call OpenGL transformation functions according to that order
- OpenGL will actually perform the transformations in reverse order
- Everything will be just right!!!