TRANSFORMATIONS IN OPEN GL

INSTRUCTOR: Sabina Irum

TRANSFORMATIONS

Why use transformations?

- Create object in convenient coordinates
- Reuse basic shape multiple times

Learn how to carry out transformations in

- **OpenGL**

- Rotation
- Translation
- Scaling

- **Introduce OpenGL matrix modes**

– Viewing transformation(Projection)

Used for positioning and aiming a camera.

Modeling transformation.(Model-view)

Used for positioning and orienting the model.

OPENGL MATRICES

- • **Matrix Mode(glMatrixMode)**

ModelViewMatrix (GL_MODELVIEW)

- Model related operations: glBegin, glEnd,
- glTranslate, glRotate, glScale, gluLookAt…

Projection Matrix (GL_PROJECTION)

- Setup projection matrix: glViewport,
- gluPerspective/glOrtho/glFrustum…

CURRENT TRANSFORMATION MATRIX (CTM)

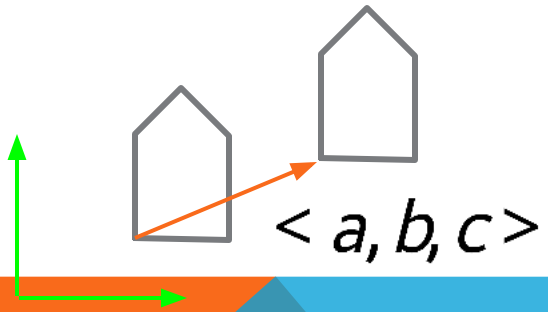
- **Conceptually there is a 4 x 4 homogeneous coordinate matrix, the *current transformation matrix (CTM) that is part of the state and is* applied to all vertices that pass down the pipeline**
- **The CTM is defined in the user program and loaded into a transformation unit**

CTM is manipulated by doing the following to the object.

- Translating the object.
- Scaling the object.
- Rotating the object.

TRANSFORMATIONS

- **Translation**



- Translation:

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

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

- Translate(a,b,c)

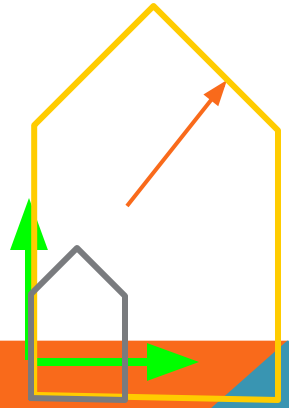
$$\begin{vmatrix} x' \\ y' \\ Z' \\ 1 \end{vmatrix} = \begin{vmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{vmatrix} * \begin{vmatrix} x \\ y \\ z \\ 1 \end{vmatrix}$$

glTranslatef(a,b,c);
glTranslated(a,b,c);

TRANSFORMATIONS

- **Scaling**

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

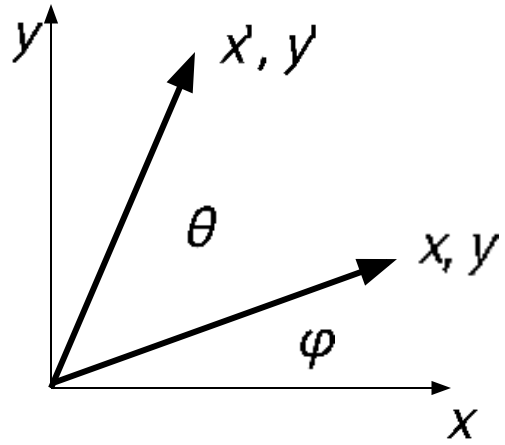


scale(a,b, c)

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} a & & & \\ & b & & \\ & & c & \\ & & & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

glScalef(a,b,c);
glScaled(a,b, c);

ROTATIONS (2D)



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

$$\cos(\varphi+\theta) = \cos\varphi\cos\theta - \sin\varphi\sin\theta$$
$$\sin(\varphi+\theta) = \cos\varphi\sin\theta + \sin\varphi\cos\theta$$

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

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

ROTATIONS 2D

So in matrix notation

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

ROTATIONS (3D)

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

OPENGL ROTATION MATRIX

$$\begin{vmatrix} x' \\ y' \\ Z' \\ 1 \end{vmatrix} = M * \begin{vmatrix} x \\ y \\ z \\ 1 \end{vmatrix}$$

glRotate*(a, 1, 0, 0):

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

glRotate*(a, 0, 1, 0):

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

glRotate*(a, 0, 0, 1):

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

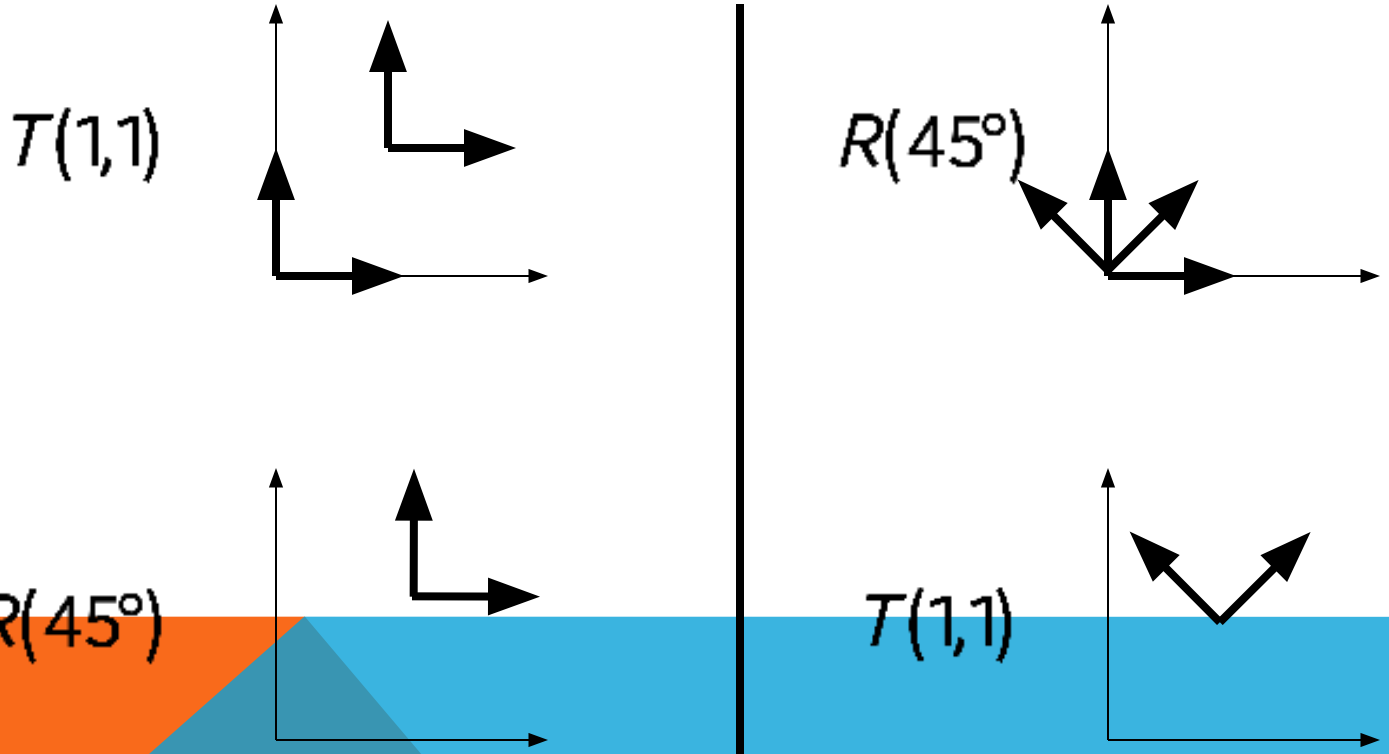
2D TRANSFORMATIONS

Shears:

$$SH_x(a) = \begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix} \qquad P' = SH_x(a) \cdot P = \ldots = \begin{bmatrix} x + ay \\ y \end{bmatrix}$$

$$SH_y(b) = \begin{bmatrix} 1 & 0 \\ b & 1 \end{bmatrix}$$

COMBINING TRANSLATION & ROTATION



EXAMPLE REVISIT

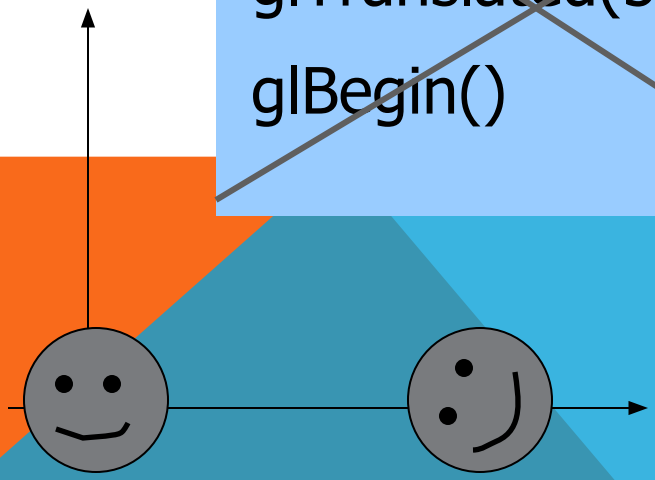
We want rotation and then translation

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 transform the object in a fixed global coordinate system,

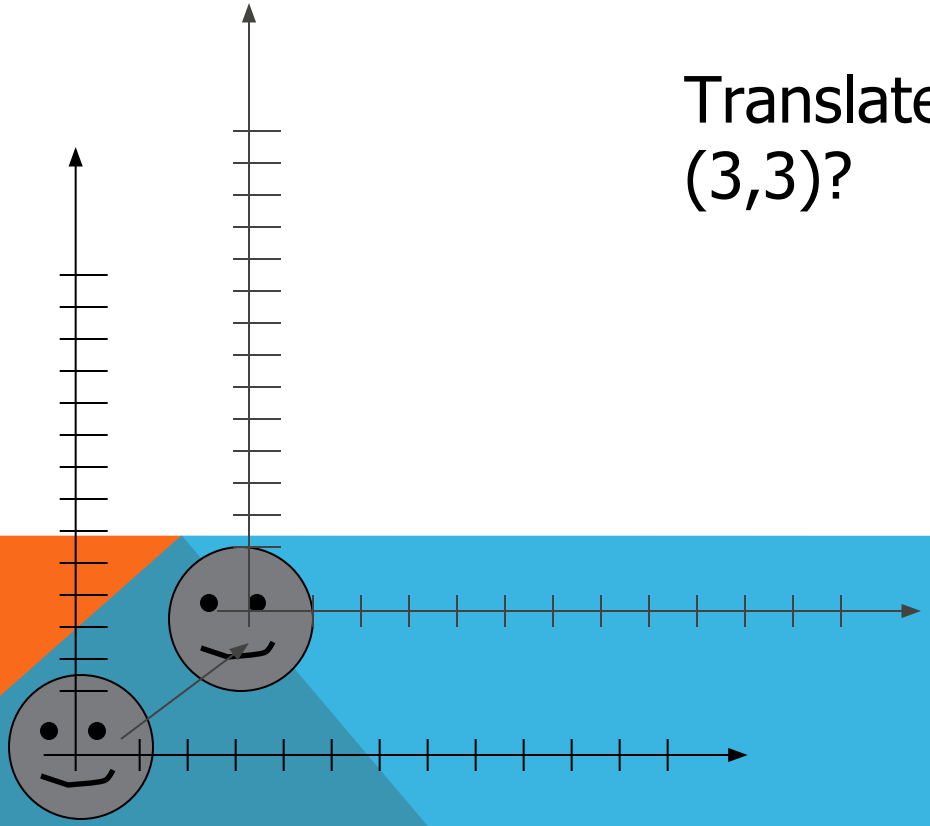
- you should think of transforming an object as moving (transforming) its local coordinate system.

OpenGL Transformation

When using OpenGL, we need to think of object transformations as moving (transforming) an object's local coordinate frame.

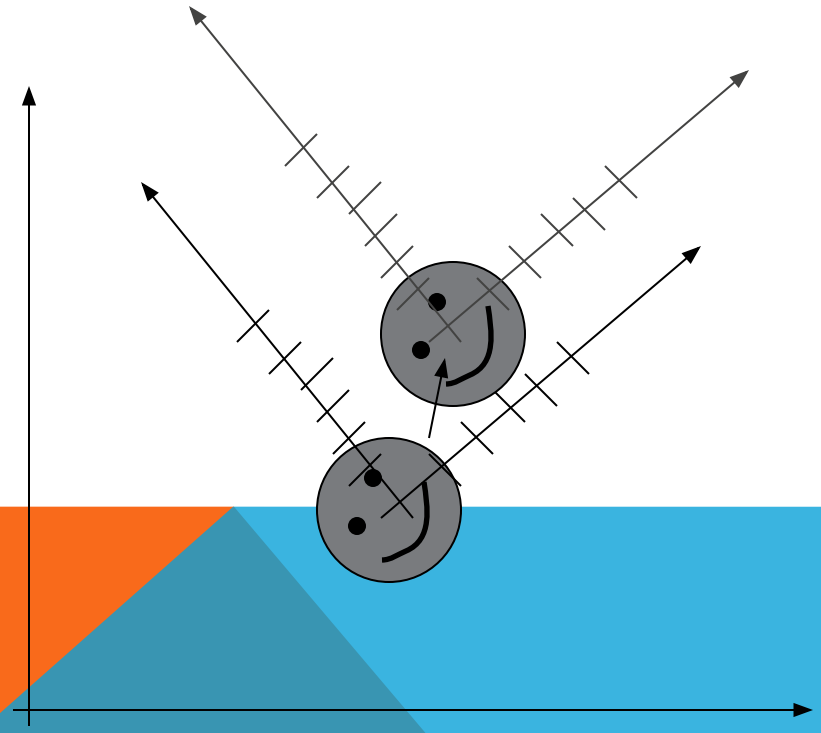
All the transformations are performed relative to the current coordinate frame origin and axes.

TRANSLATE COORDINATE FRAME



Translate
(3,3)?

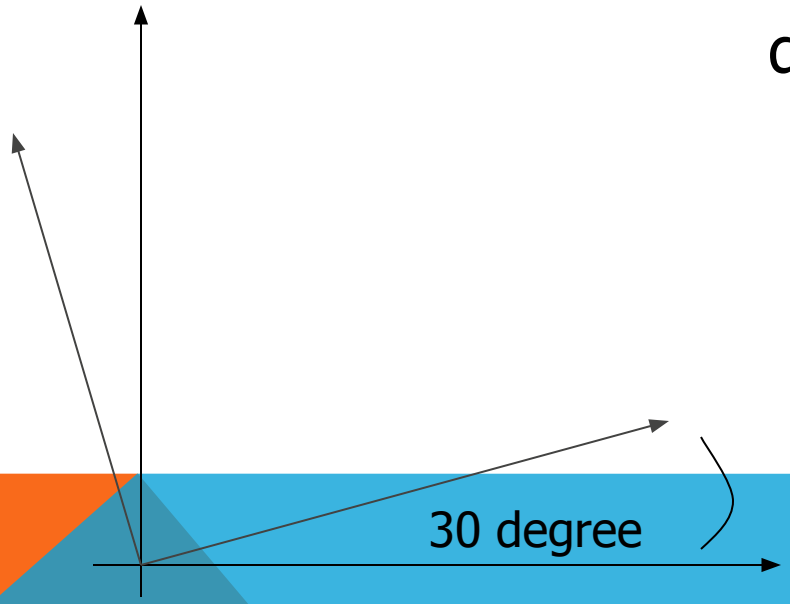
TRANSLATE COORDINATE FRAME (2)



Translate (3,3)?

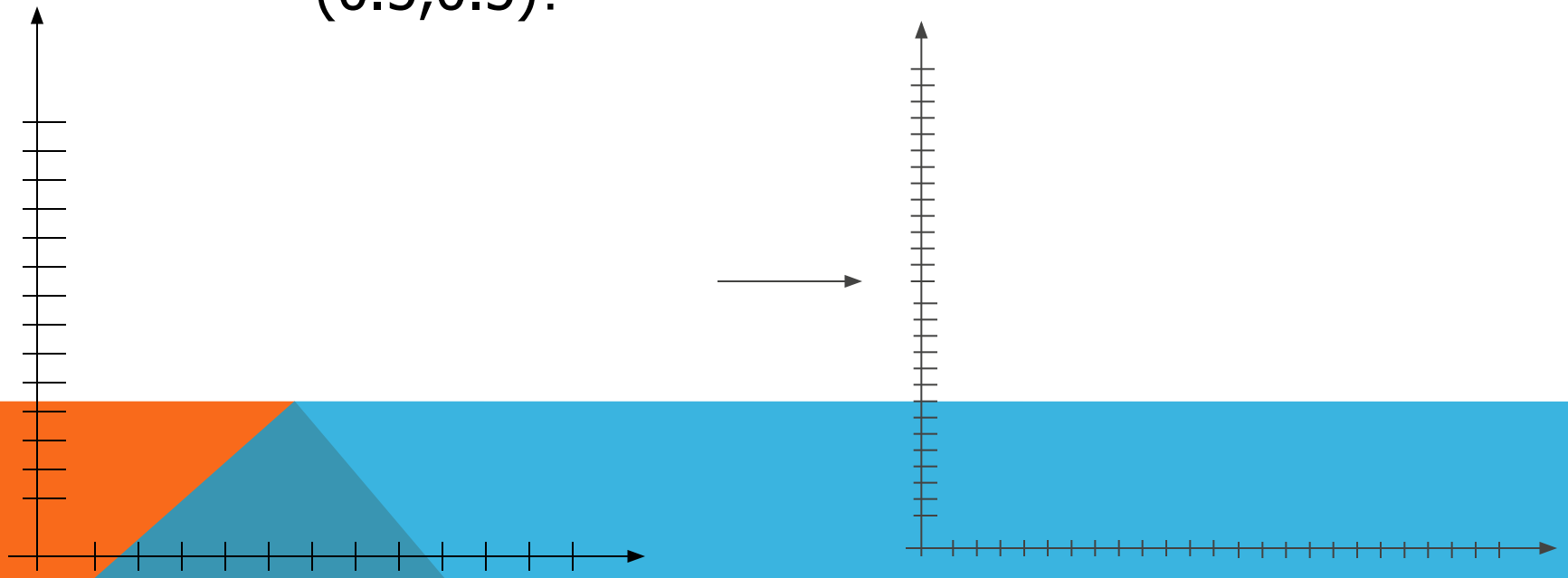
ROTATE COORDINATE FRAME

Rotate 30 degree?

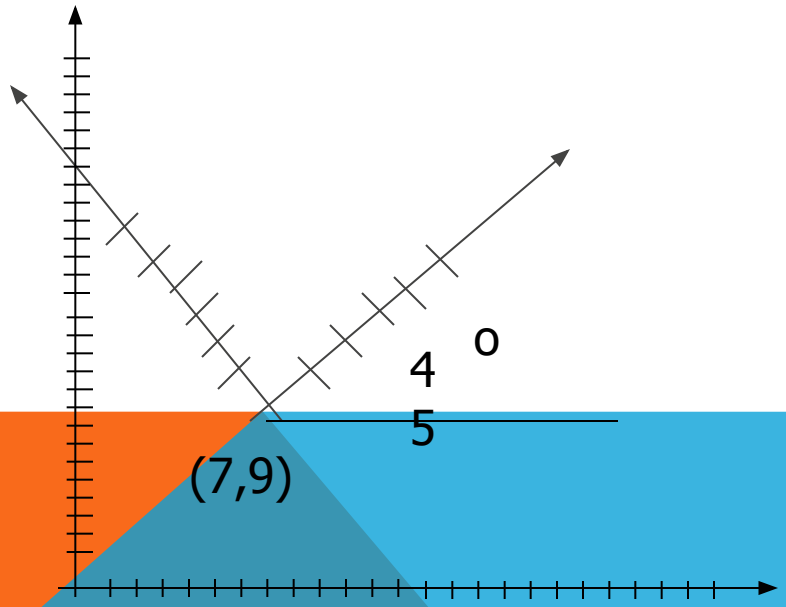


SCALE COORDINATE FRAME

Scale
(0.5,0.5)?



COMPOSE TRANSFORMATIONS

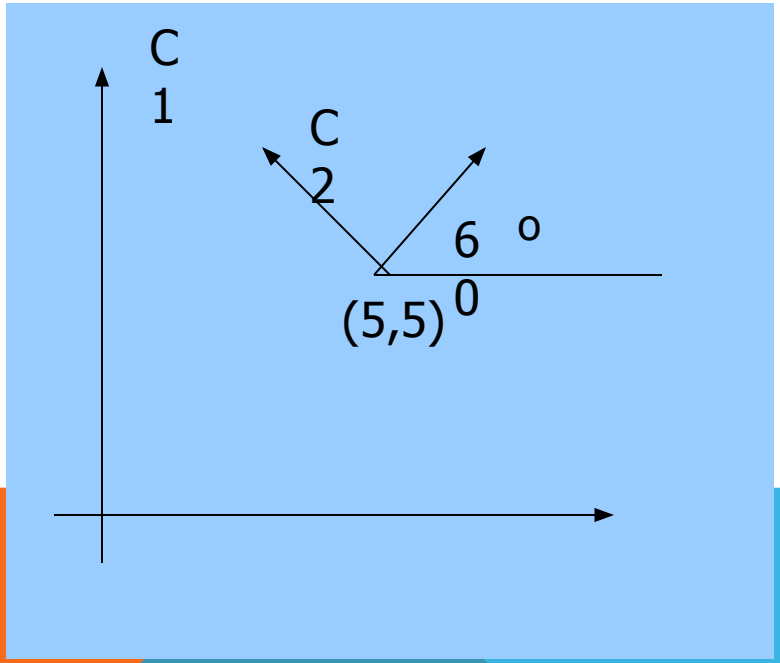


Transformations?

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)
???

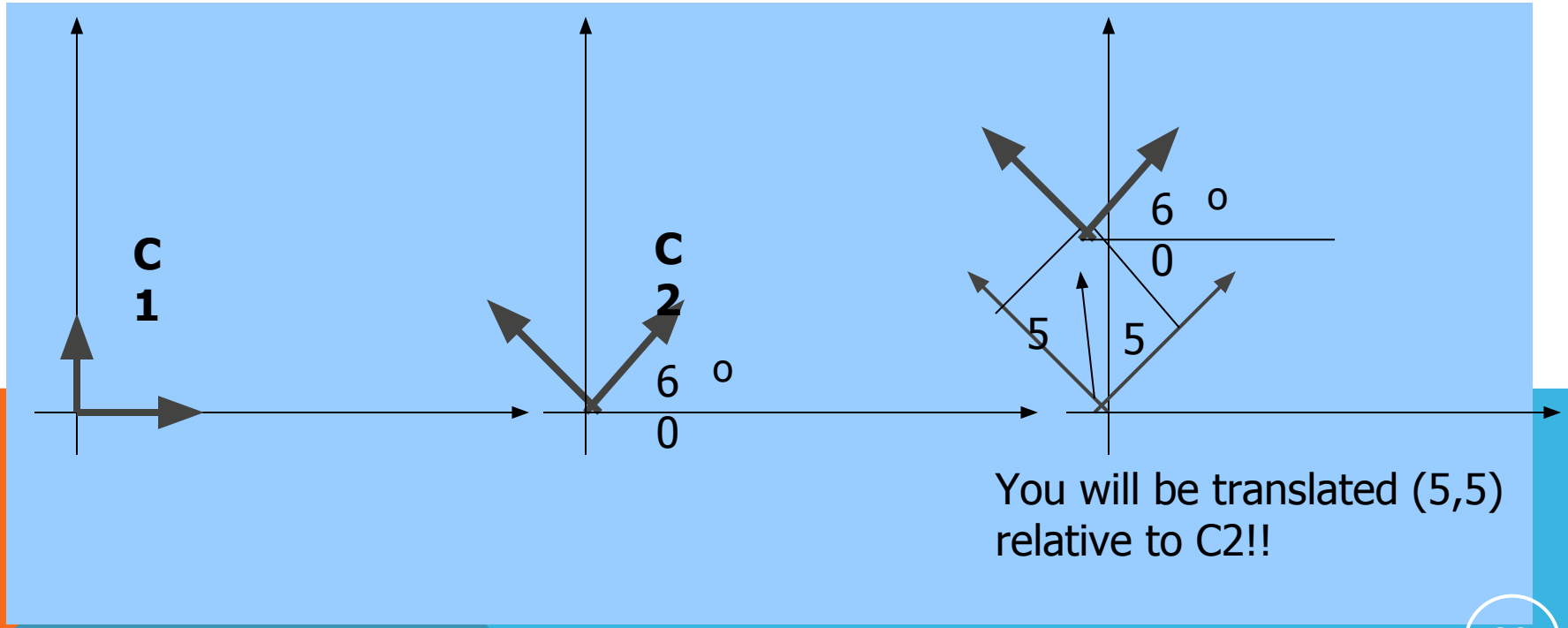
Answer: Translate(5,5) and then

Rotate (60)

Another example (cont'd)

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

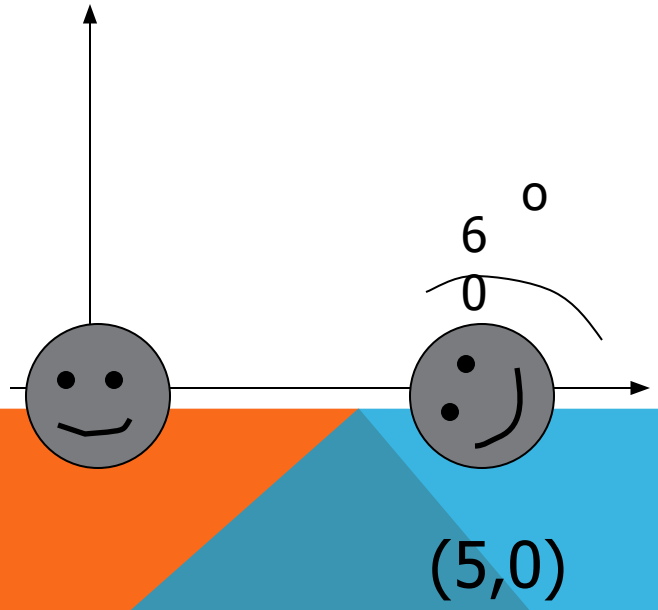
…



TRANSFORM OBJECTS

- Does coordinate frame transformation have anything to do with object transformation?
  - Yes,
    - you can view transformation as paste the object to a local coordinate frame
    - and move that coordinate frame.

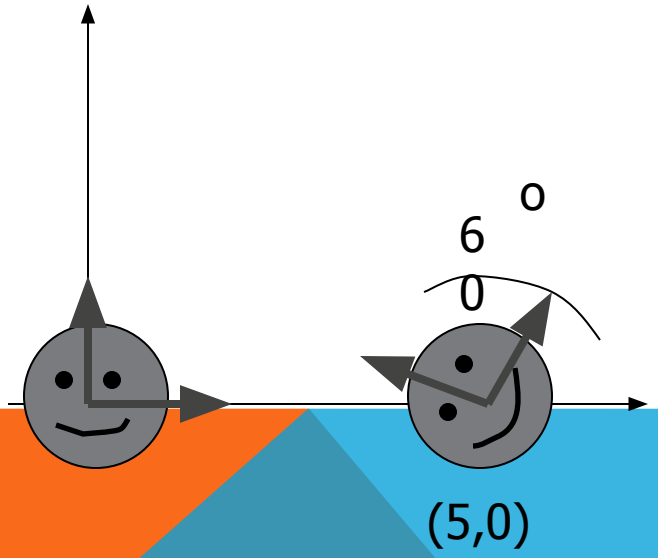
Example



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

1) Rotate (60^{o})
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^{o})

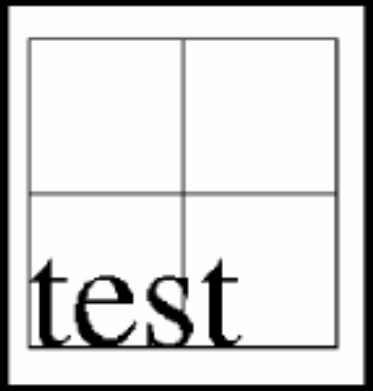
Exact the opposite order compared to the previous slide!!

PUT IT ALL TOGETHER

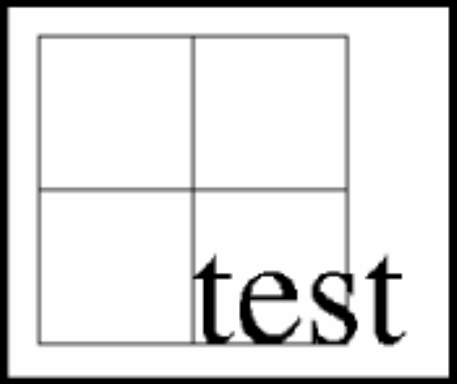
- **When you use OpenGL …**
- **Think of transformation as moving coordinate frames.**
- **Call OpenGL transformation functions in that order.**
- **OpenGL will actually perform the transformations in the reverse order.**
- **Everything will be just right!!!**

2D Transformations in Postscript, 1

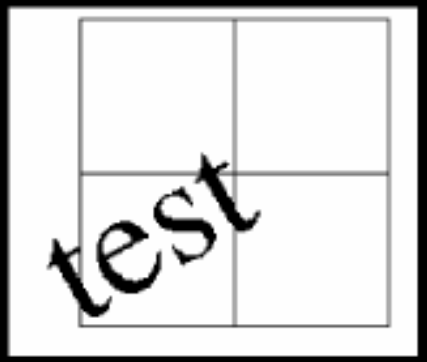
```
0 0 moveto
(test) show
```




```
1 0 translate
0 0 moveto
(test) show
```

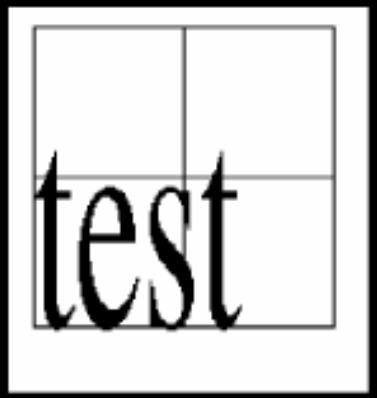



```
30 rotate
0 0 moveto
(test) show
```

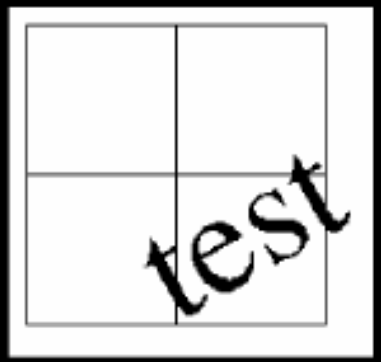


2D Transformations in Postscript, 2

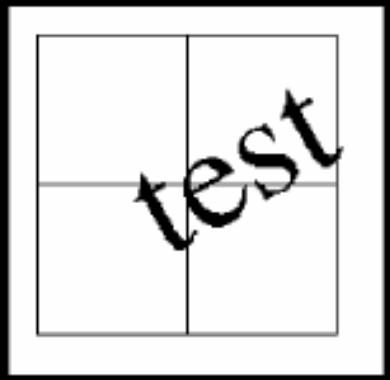
```
1 2 scale
0 0 moveto
(test) show
```




```
1 0 translate
30 rotate
0 0 moveto
(test) show
```




```
30 rotate
1 0 translate
0 0 moveto
(test) show
```



2D Transformations in Postscript, 3

```
30 rotate
1 2 scale
0 0 moveto
(test) show
```

```
1 2 scale
30 rotate
0 0 moveto
(test) show
```

```
-1 1 scale
0 0 moveto
(test) show
```

SUMMARIZING TRANSFORMATIONS

Translation

- *glTranslatef(dx, dy, dz)*
- can be used to shift an object in space

Rotation

- *glRotatef(theta, vx, vy, vz);*
- Note that *theta* is in degrees, and (vx, vy, vz) define the axis of rotation

- **scaling**

- *glScalef(sx, sy, sz)*

Each has a float (f) and double (d) format (e.g., glScaled)

OPENGL STACK MATRIX
PUSH AND POP MATRIX STACK

- glTranslate(5,0,0)
- Draw_base();
- glPushMatrix();
- glRotate(75, 0,1,0);
- Draw_left_hammer();
- glPopMatrix();
- glRotate(-75, 0,1,0);
- Draw_right_hammer();