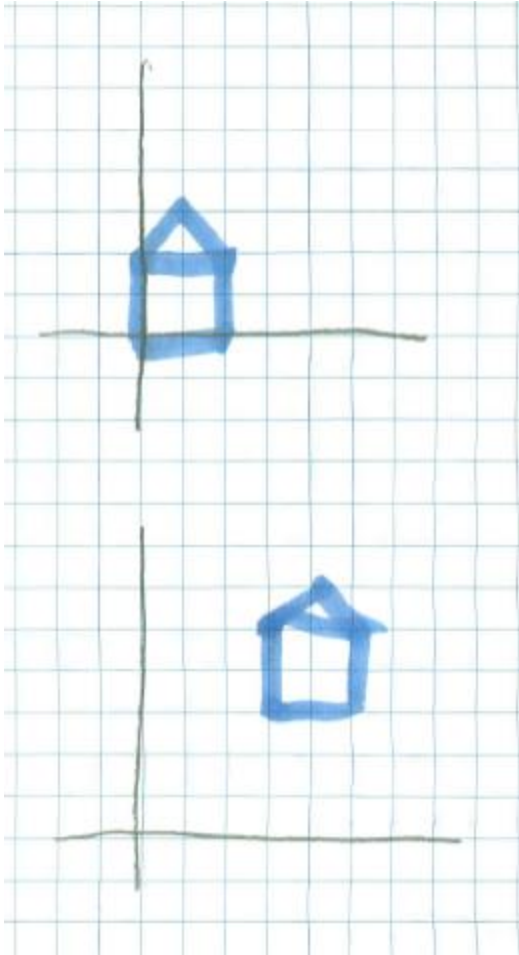


# CSE160 – Oct 15, 2020

- How can I move objects?
- Transforms in 2D
- Matrix: Transforms in 2D
- 2D homogeneous coordinates
- Matrix order matters
- 3D transforms
- Hierarchical transforms
- Assignment 1
- Administrative
- Q&A

How can I move objects?

# Transformations

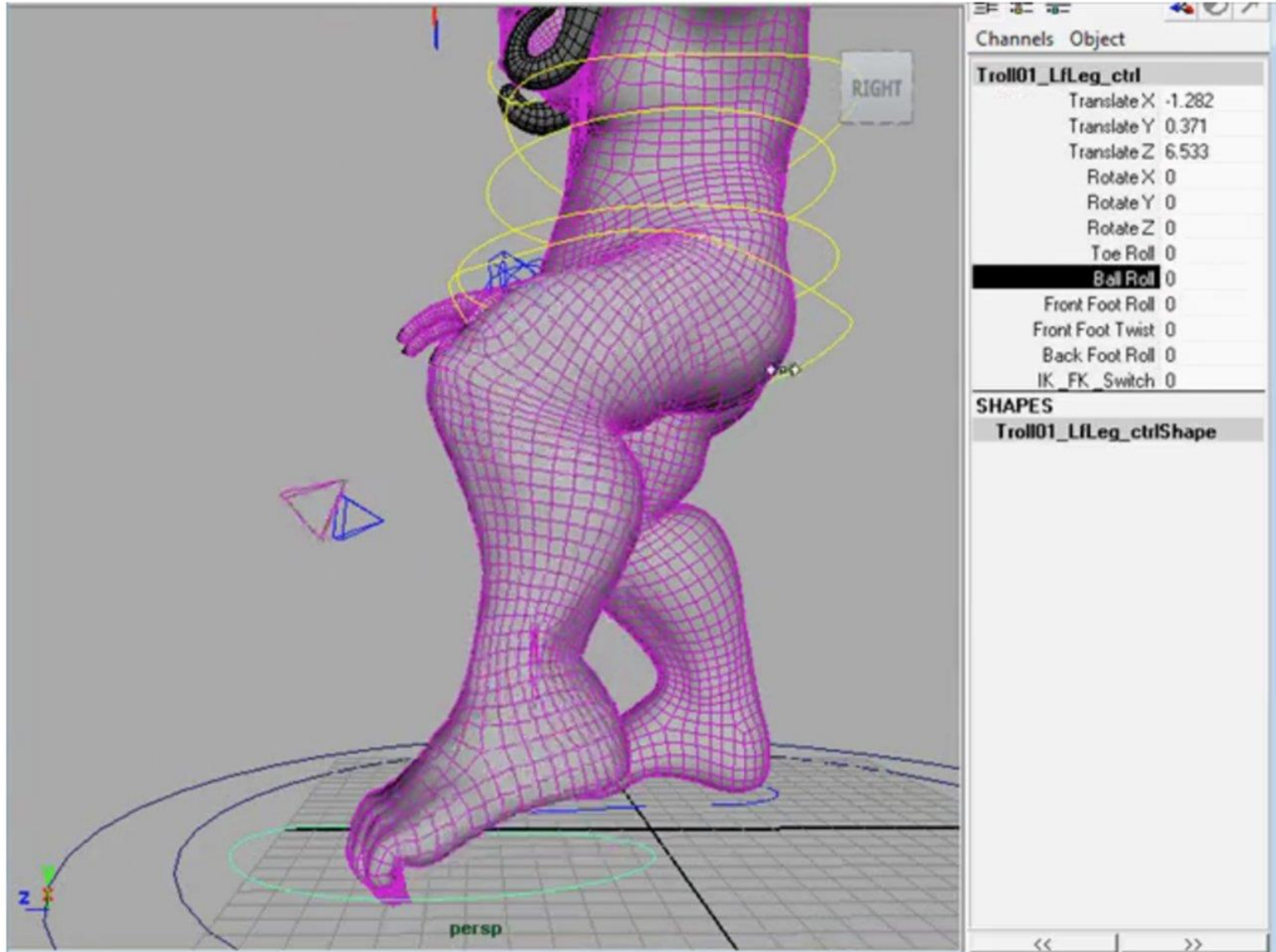


```
void drawHouse() {  
    glBegin(GL_QUADS);  
        glVertex2d(0,0);  
        glVertex2d(0,1);  
        glVertex2d(1,1);  
        glVertex2d(1,0);  
    glEnd();  
    // .... Lots more stuff  
}
```

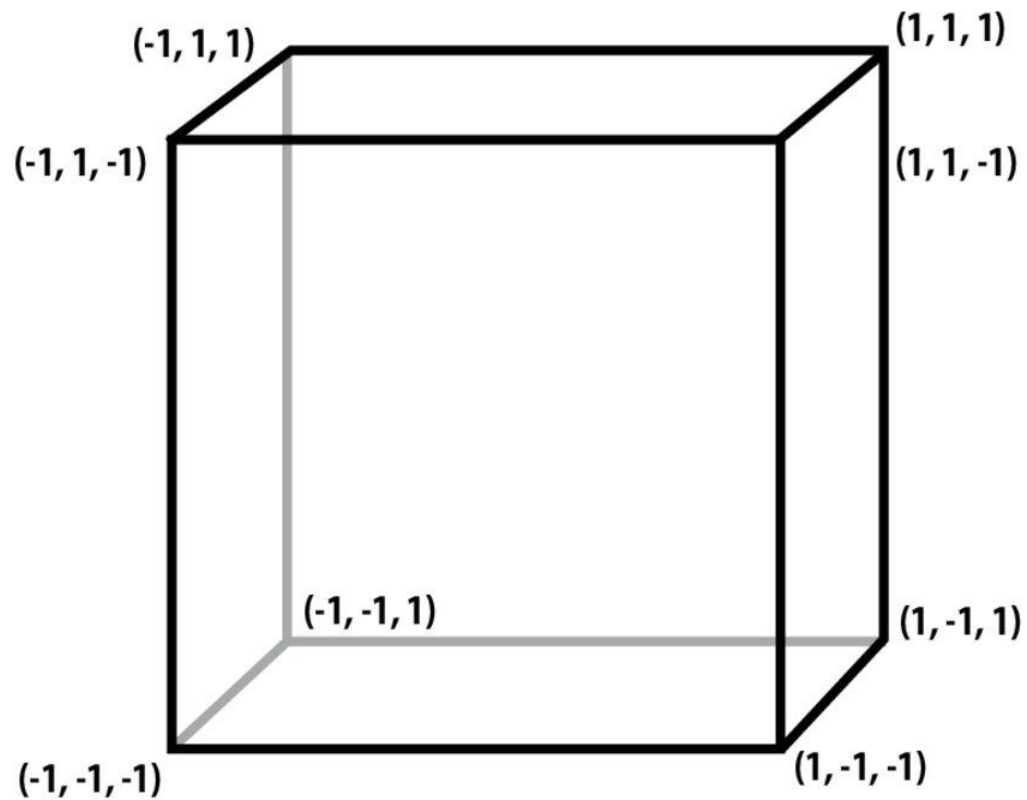
```
void main() {  
    // Draw house at origin  
    drawHouse();
```

```
    // Draw house somewhere else  
    ????  
}
```

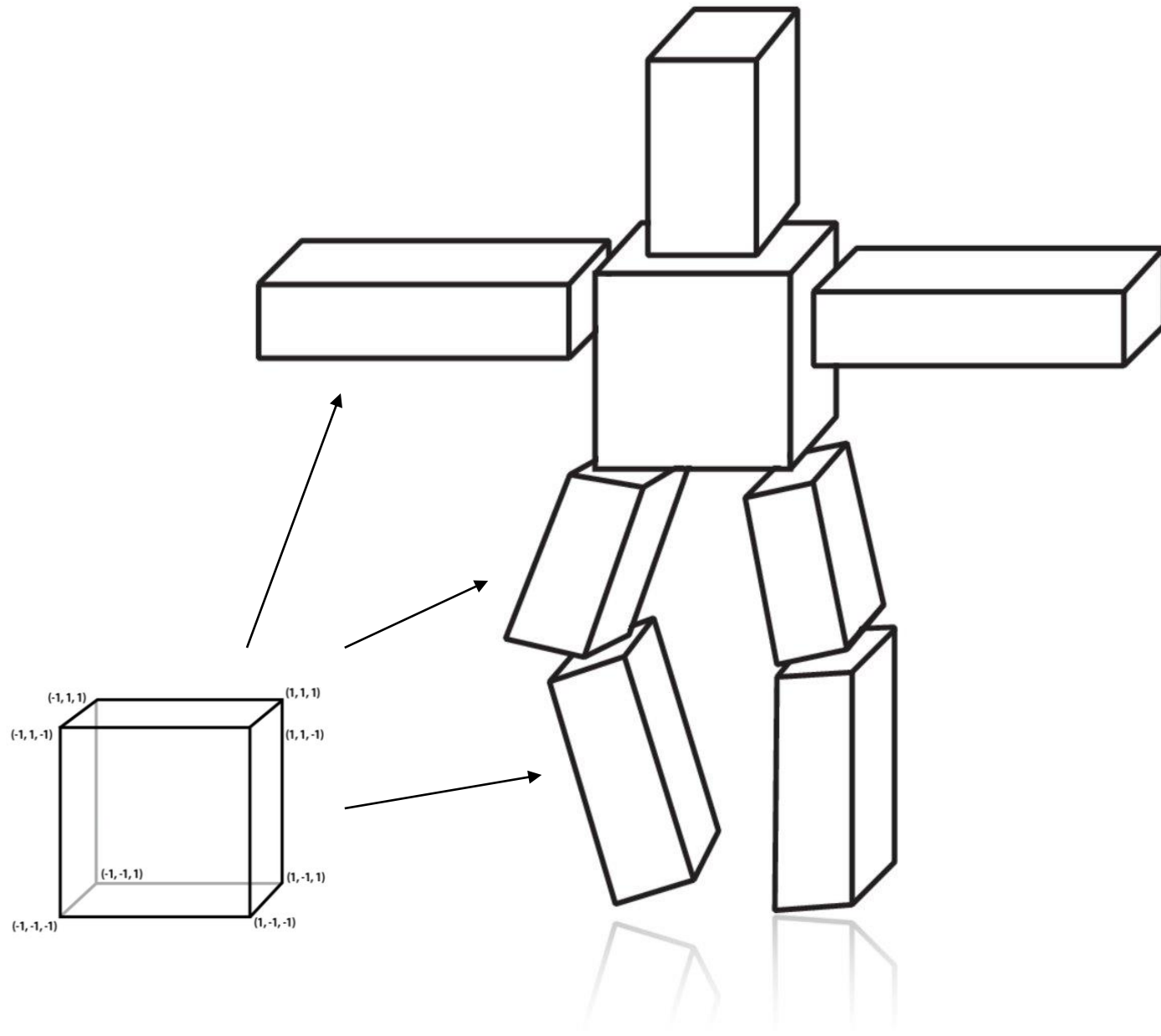
# Transformations in character rigging



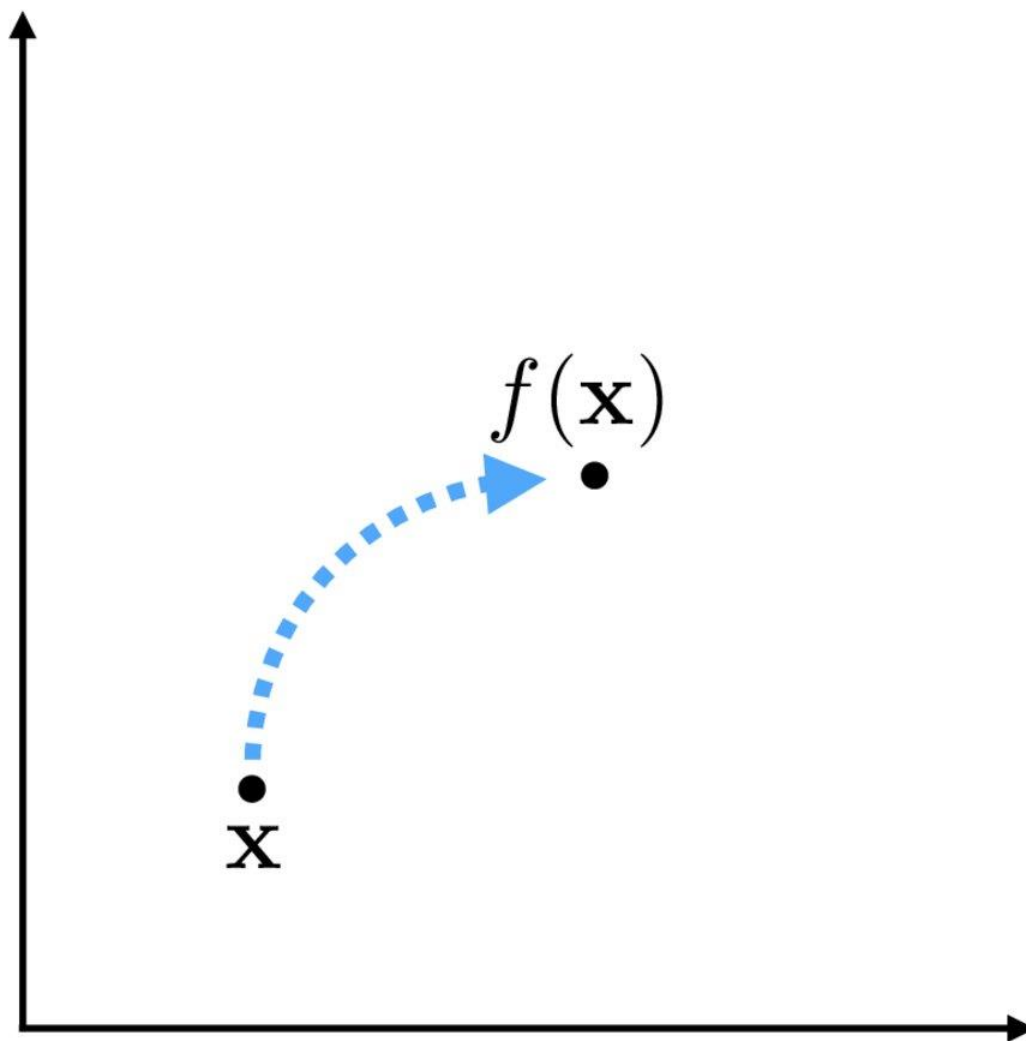
# Cube



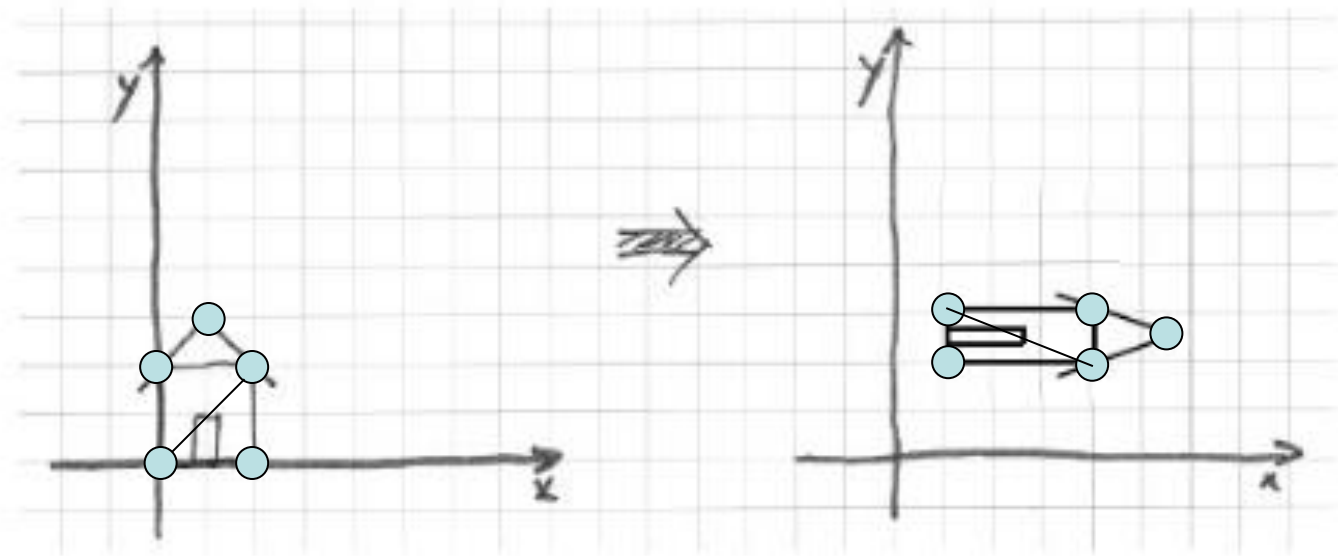
# Consider drawing a cube person



**Basic idea:  $f$  transforms  $\mathbf{x}$  to  $f(\mathbf{x})$**



# Exercise: Find the function()



Vertex buffer array

0,0  
2,2  
2,0  
0,2  
1,3  
2,2  
....

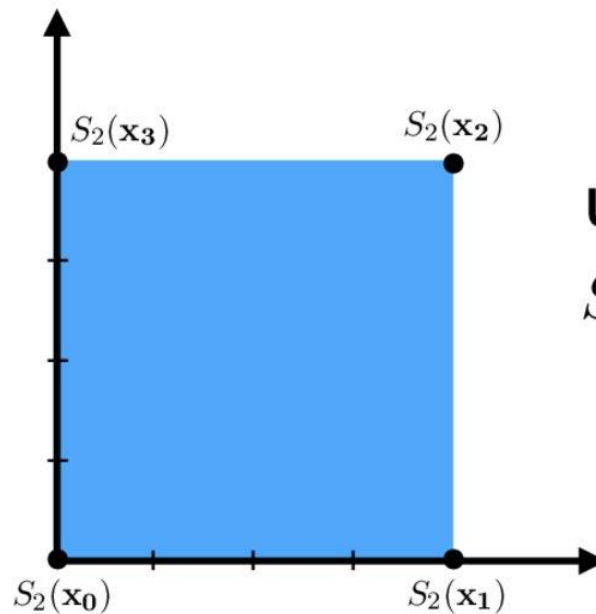
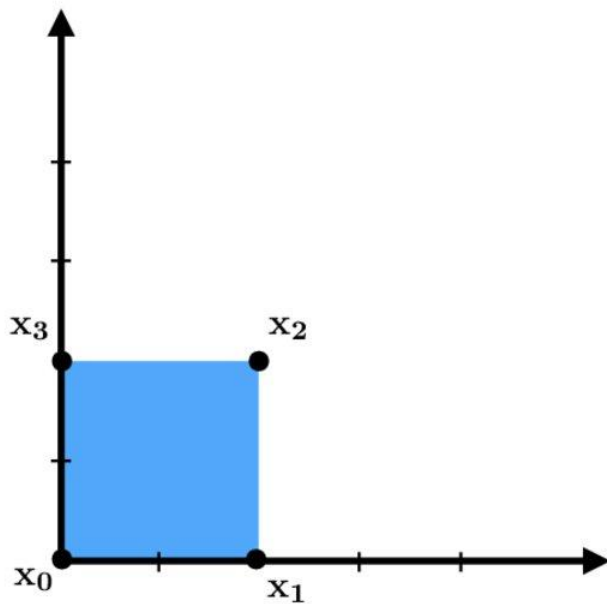
→ **f()** →

Vertex buffer array

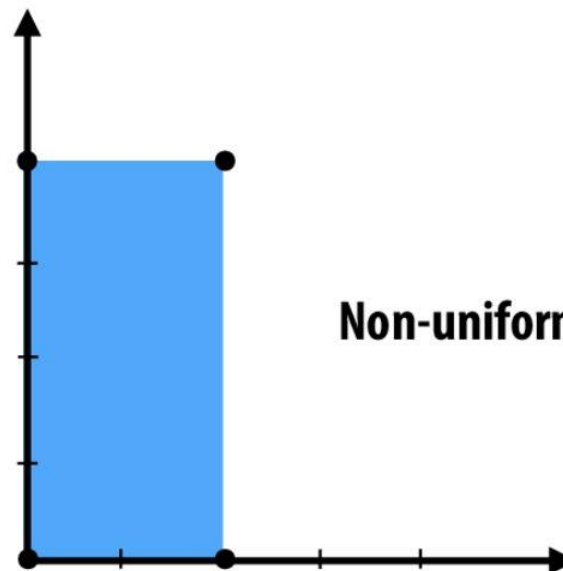
1,3  
4,2  
1,2  
4,3  
5.5, 2.5  
4,2  
....

# Transforms in 2D

# Scale

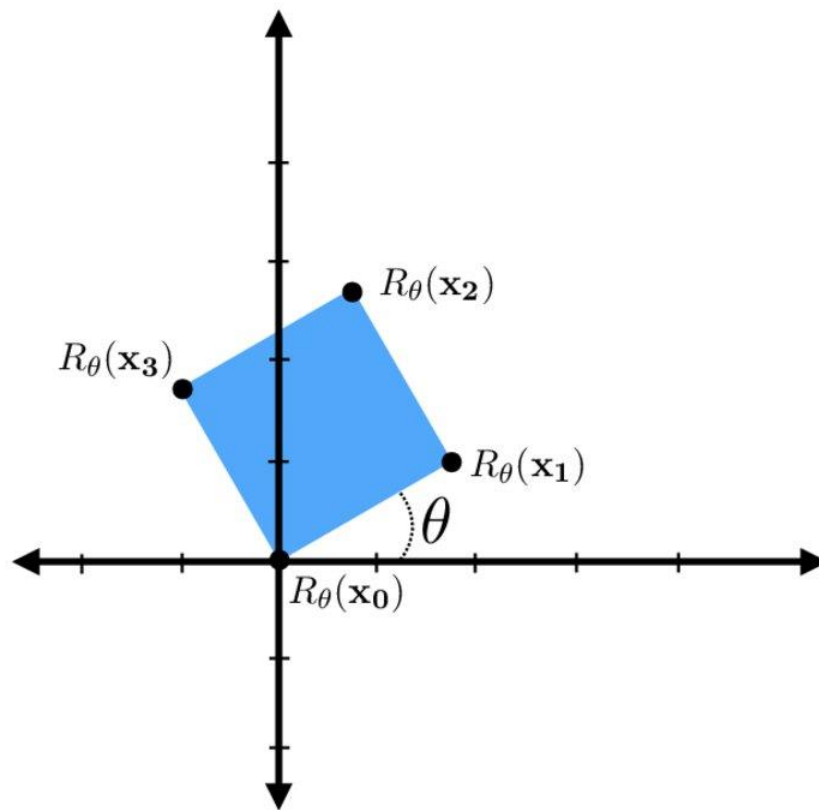
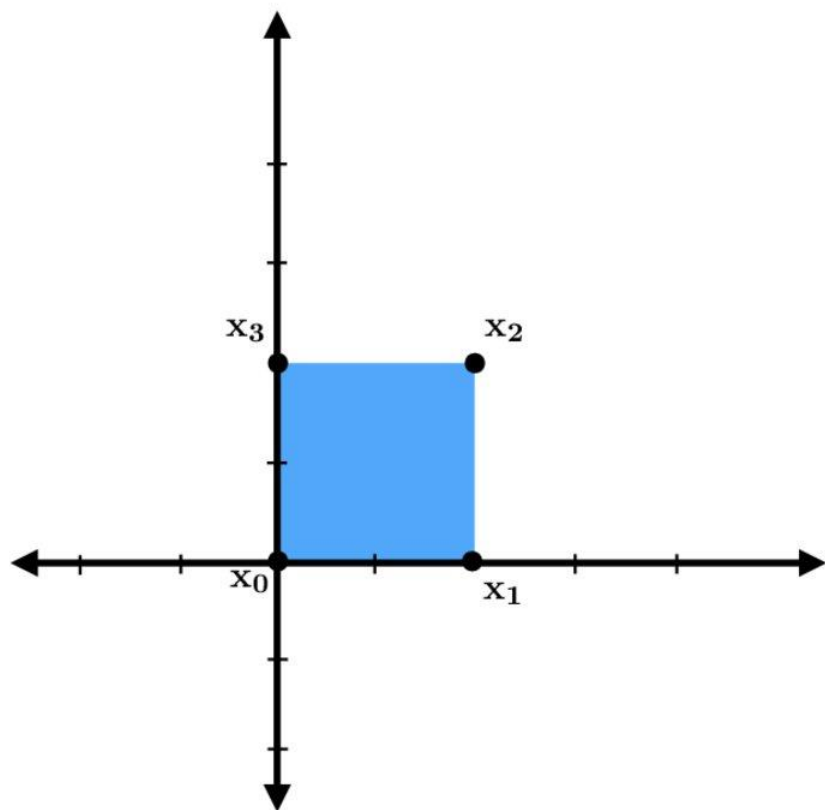


**Uniform scale:**  
 $S_a(\mathbf{x}) = a\mathbf{x}$



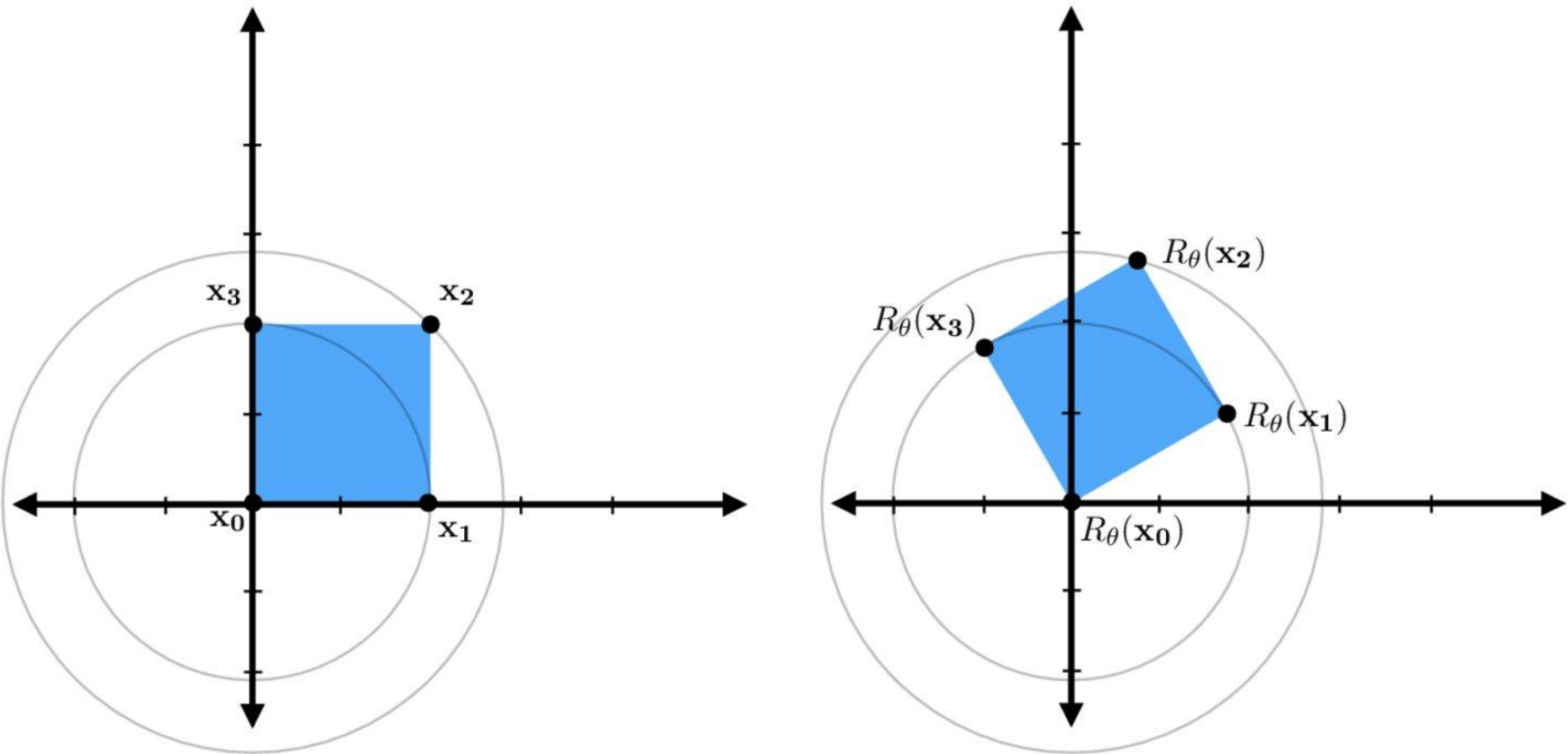
**Non-uniform scale??**

# Rotation



$R_\theta$  = rotate counter-clockwise by  $\theta$

# Rotation as circular motion

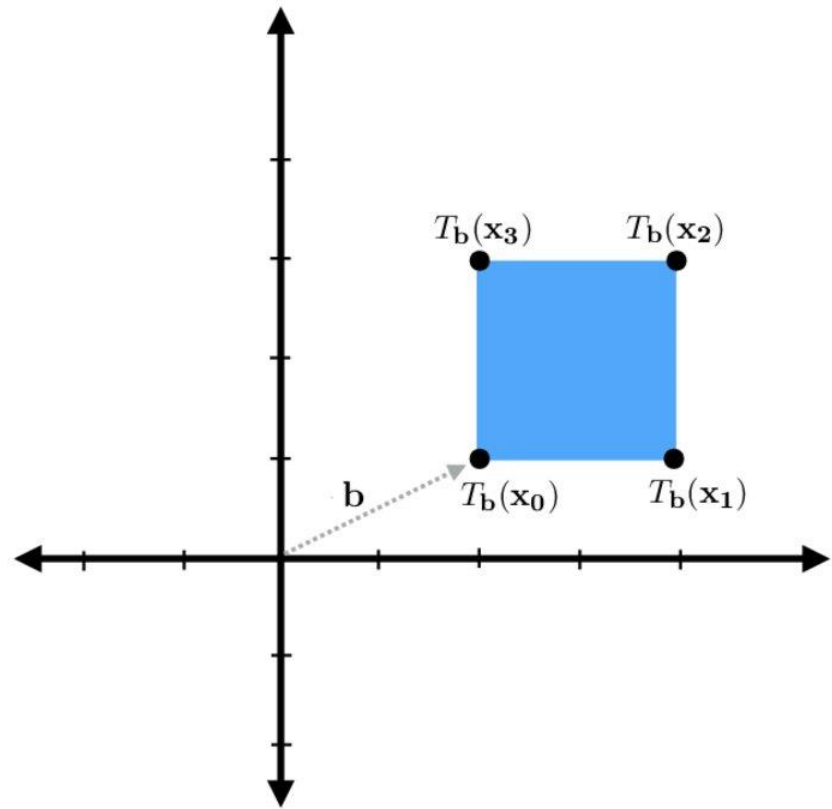
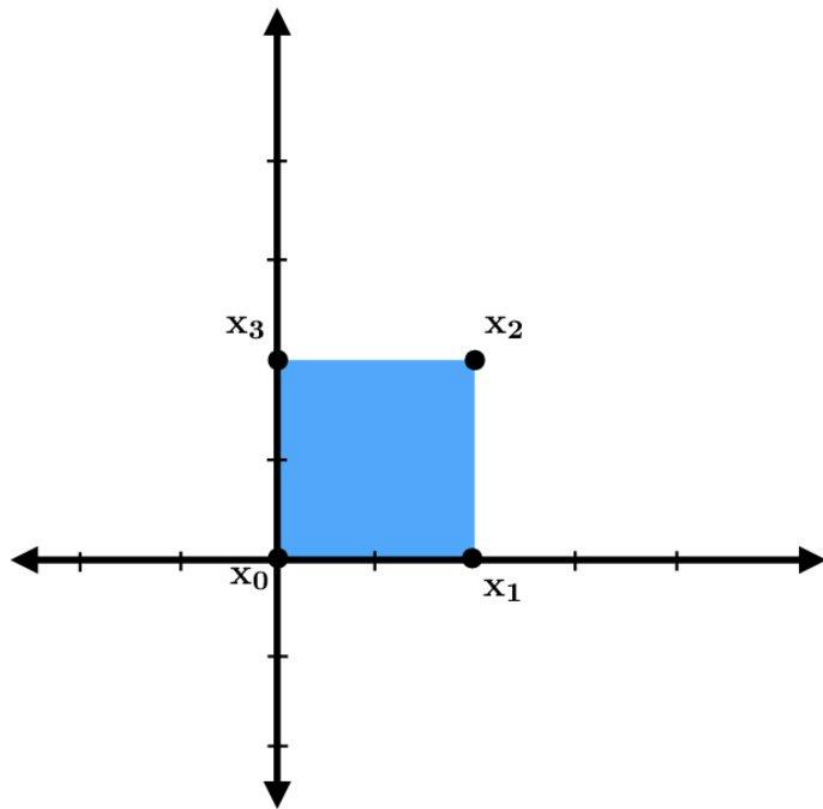


$R_\theta$  = rotate counter-clockwise by  $\theta$

As angle changes, points move along *circular* trajectories.

Hence, rotations preserve length of vectors:  $|R_\theta(\mathbf{x})| = |\mathbf{x}|$

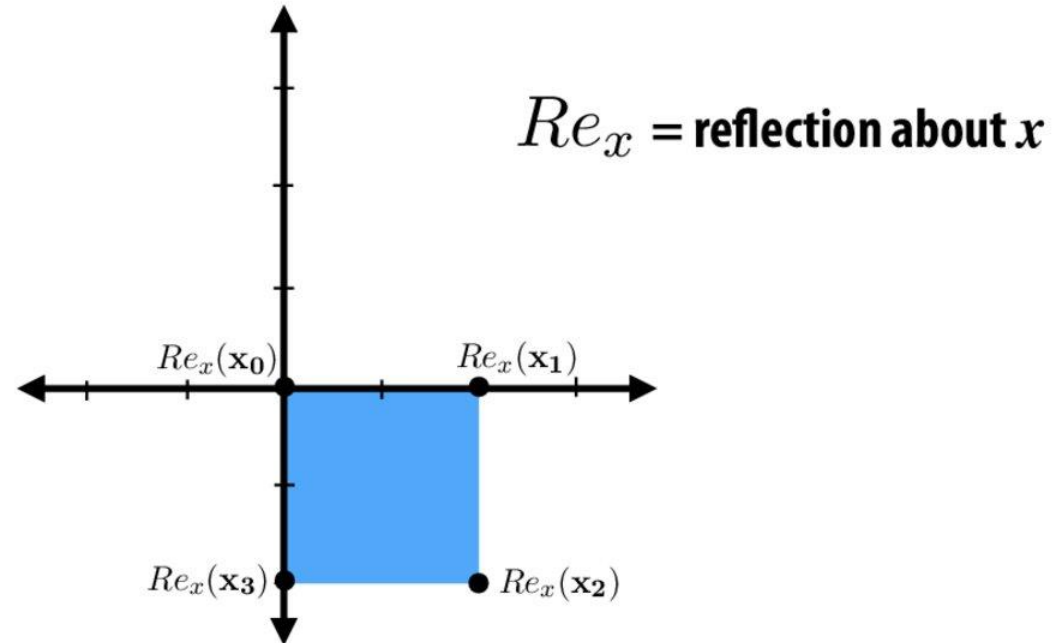
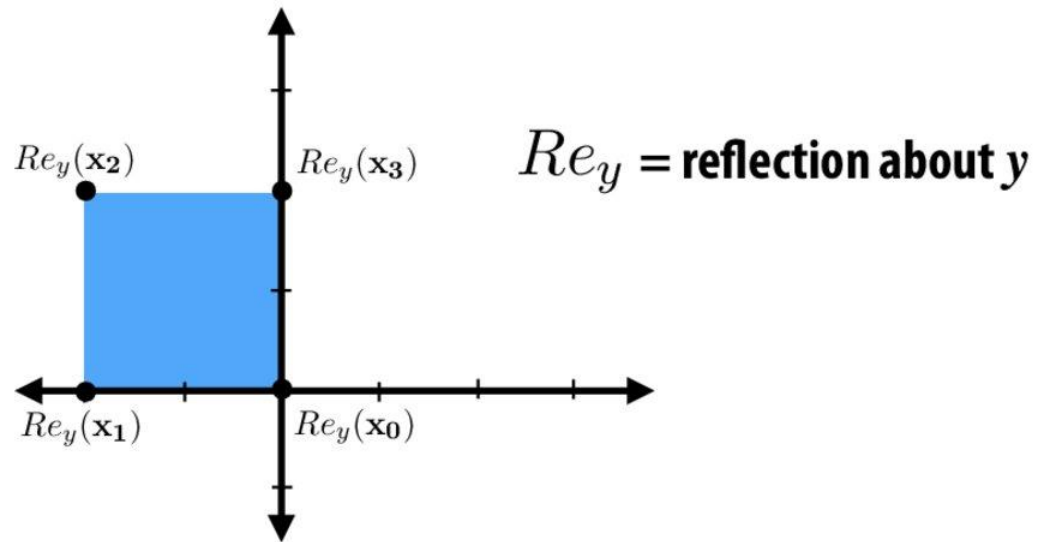
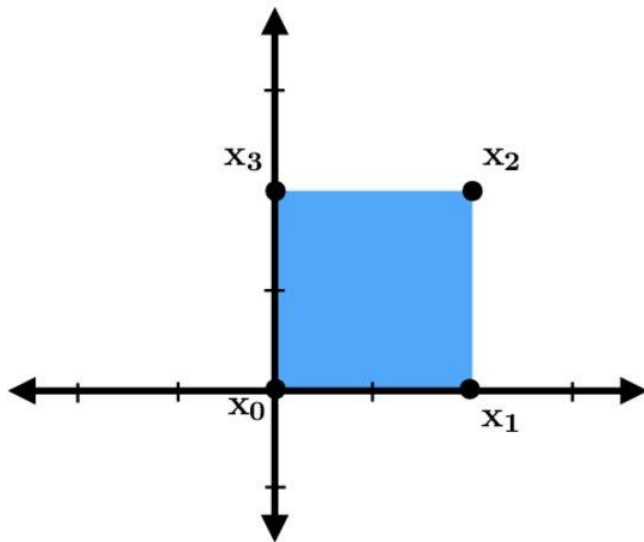
# Translation



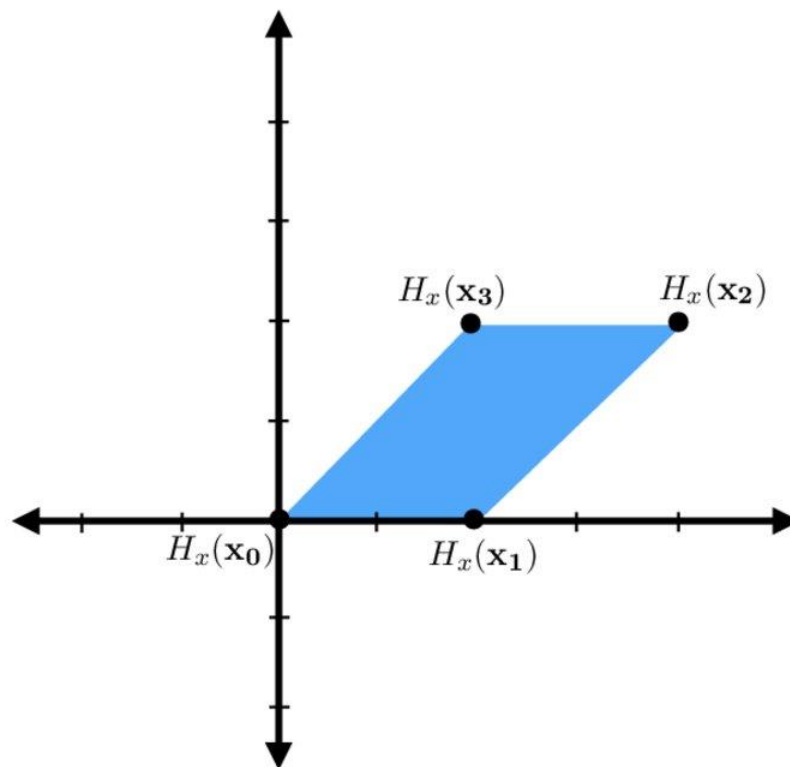
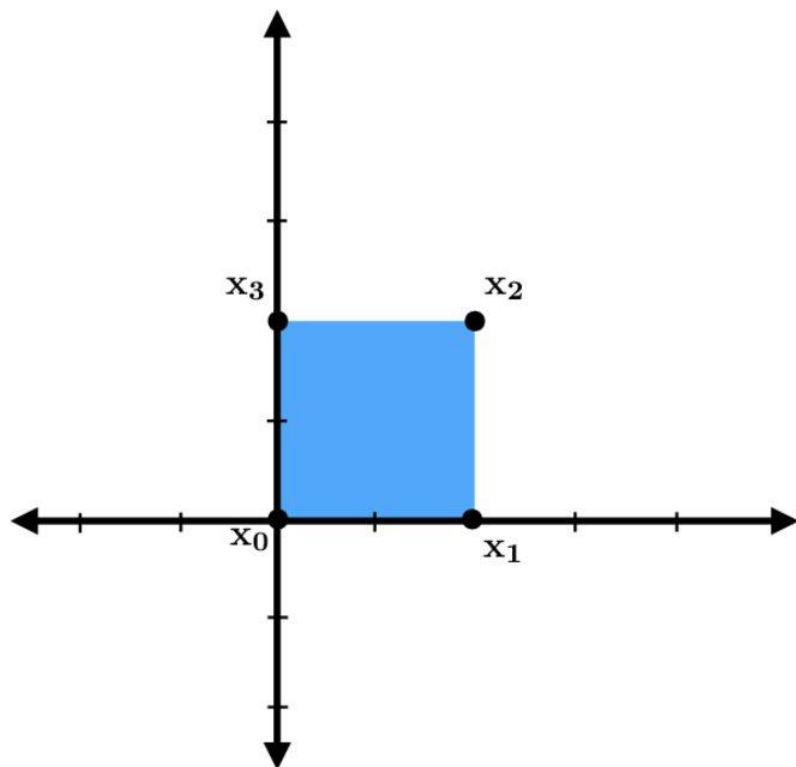
$T_b$  — “translate by  $b$ ”

$$T_b(x) = x + b$$

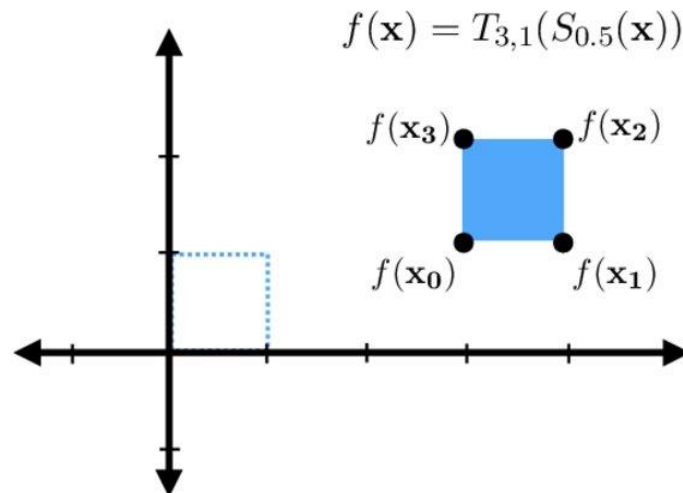
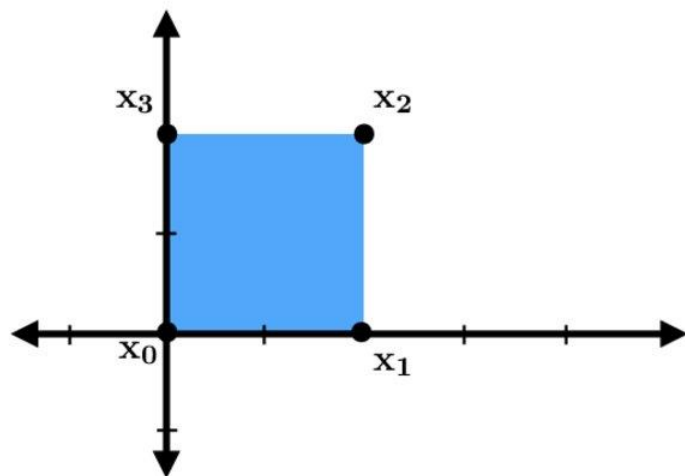
# Reflection



# Shear (in $x$ direction)



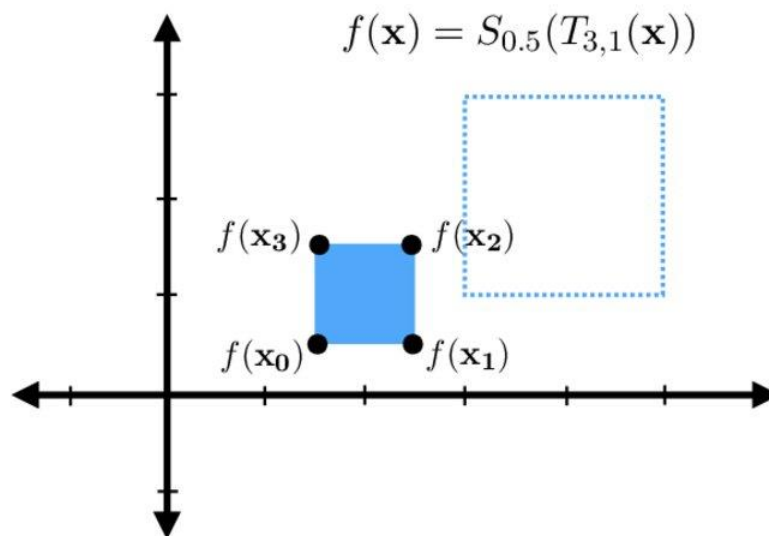
# Compose basic transformations to construct more complicated ones



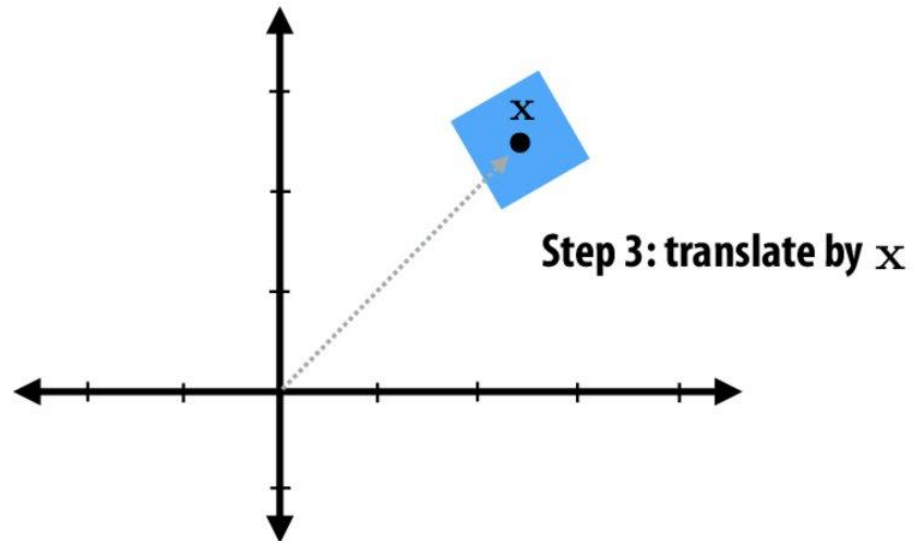
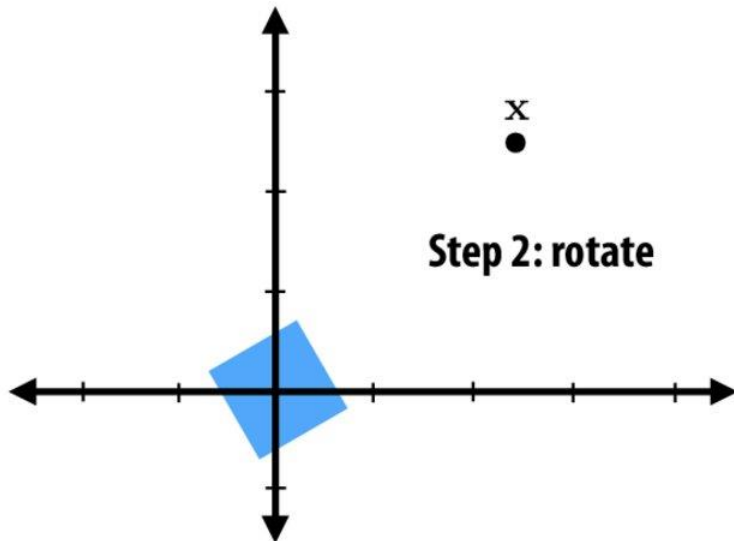
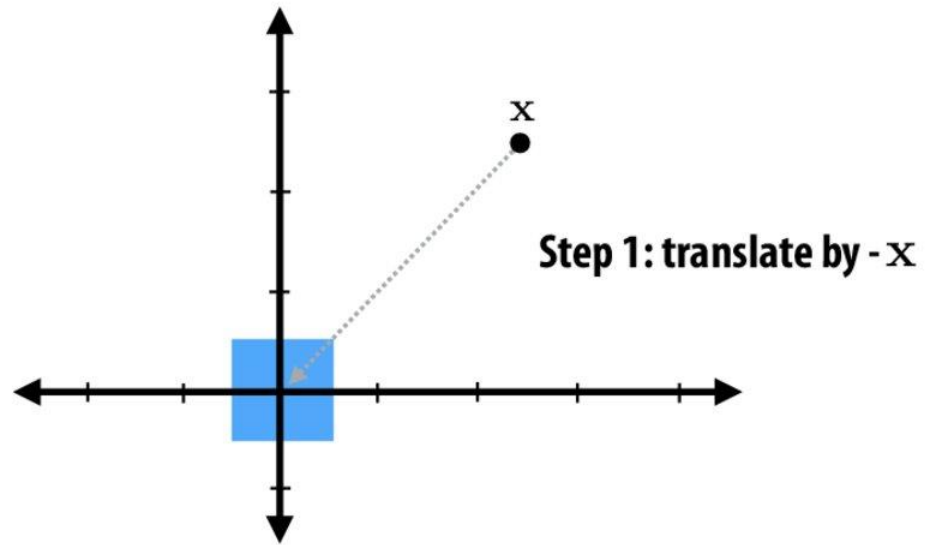
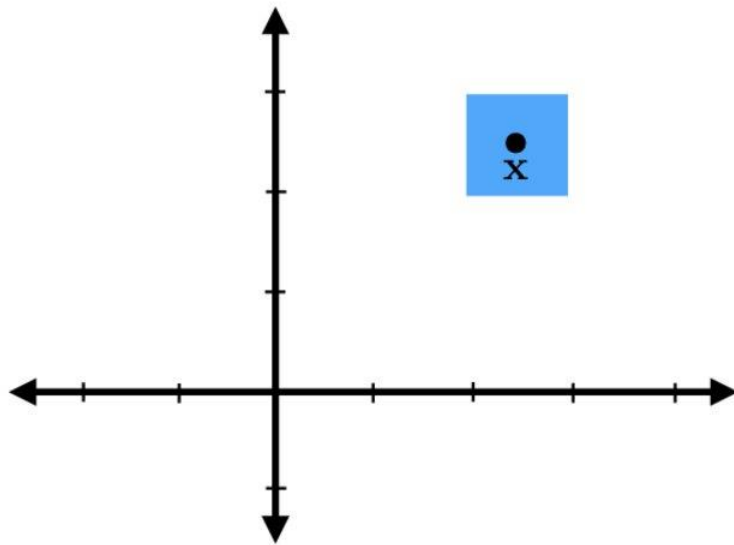
**Note: order of composition matters**

**Top-right: scale, then translate**

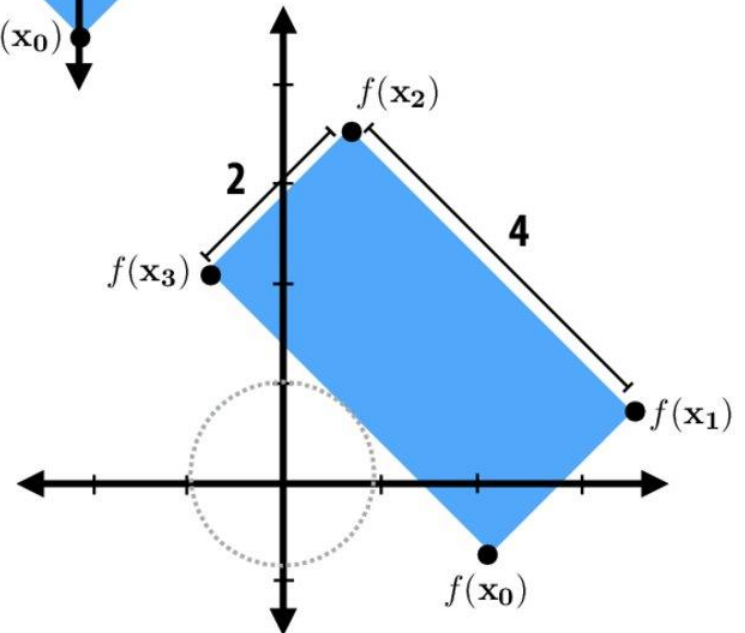
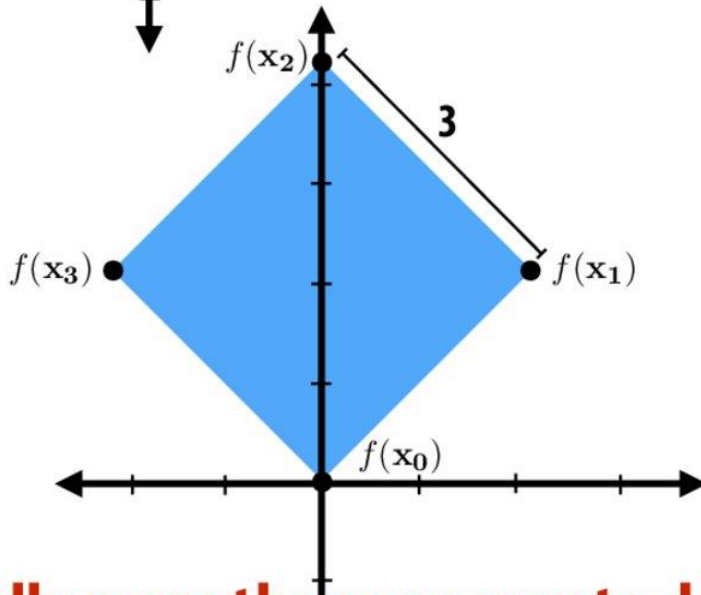
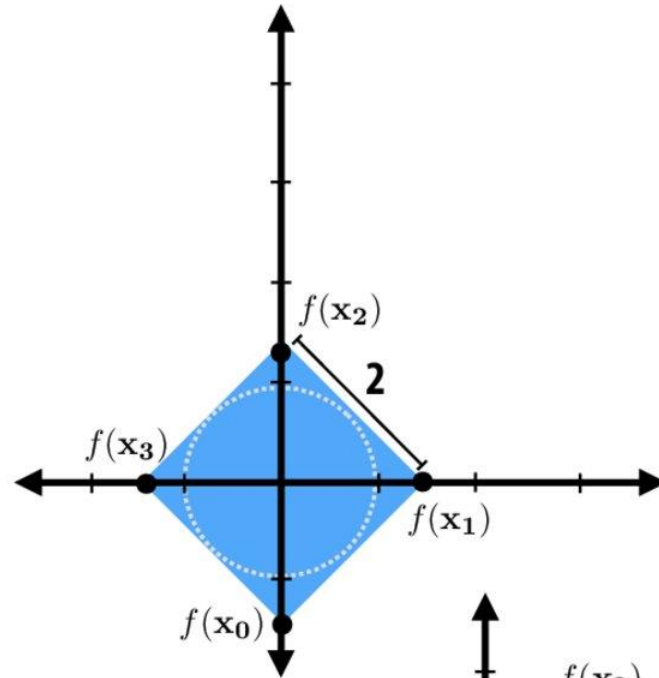
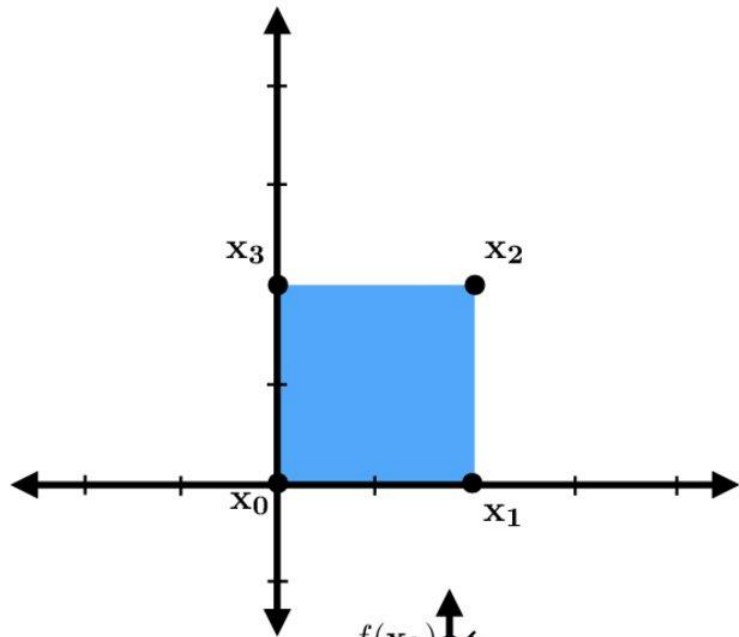
**Bottom-right: translate, then scale**



# Common task: rotate about a point $x$



# How would you perform these transformations?



Usually more than one way to do it!

# **Matrix: Transforms in 2D**

# Scaling in 2D

## Scaling

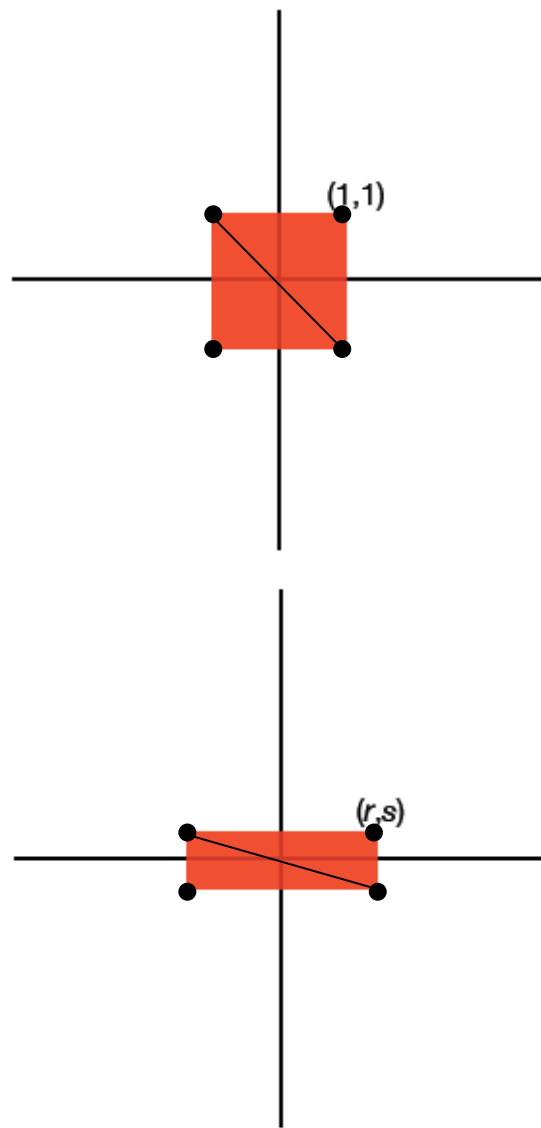
- Multiply each coordinate by a constant amount:

$$\begin{aligned}x' &= rx \\ y' &= sy\end{aligned}$$

- Use matrix notation for compactness:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} r & 0 \\ 0 & s \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$
$$\mathbf{p}' = \mathbf{S}\mathbf{p}$$

- Scaling is a linear transformation



# Translation in 2D

## Translation

- Offset each coordinate by constant amount:

$$x' = x + \Delta x$$

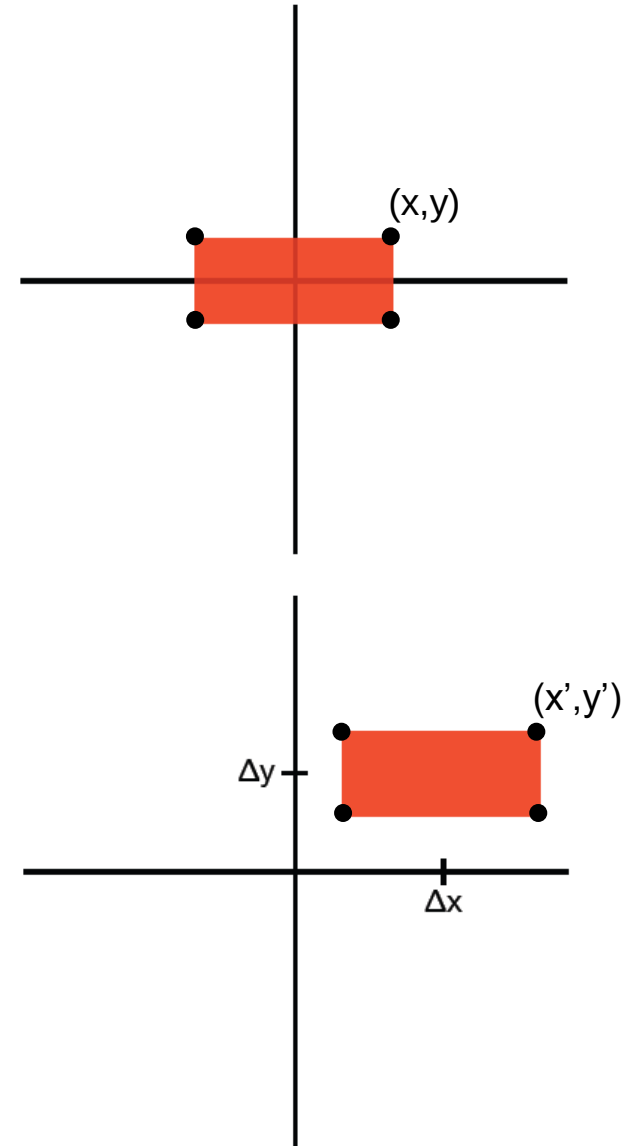
$$y' = y + \Delta y$$

- Use vector notation for compactness:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}$$

$$\mathbf{p}' = \mathbf{p} + \mathbf{d}$$

- Translation is an affine transformation, but not a linear transformation



# Rotation in 2D

## Rotation

- Rotate about the origin by some angle:

$$\begin{aligned}x &= \rho \cos \phi & x' &= \rho \cos(\phi + \theta) \\ y &= \rho \sin \phi & y' &= \rho \sin(\phi + \theta)\end{aligned}$$

- Use polar coordinates to solve for new positions:

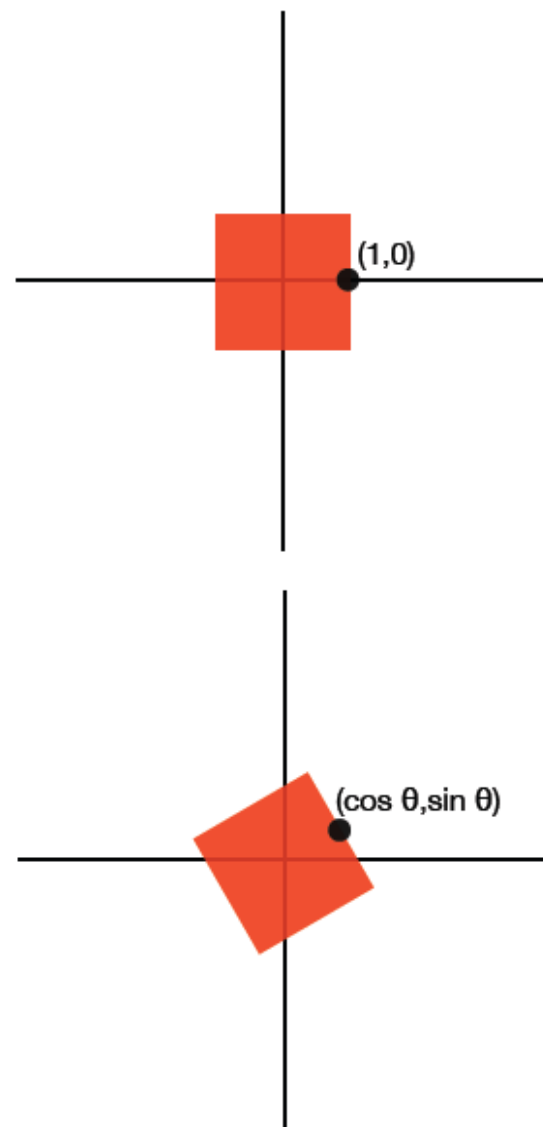
$$\begin{aligned}x' &= x \cos \theta - y \sin \theta \\ y' &= x \sin \theta + y \cos \theta\end{aligned}$$

- Use matrix notation for compactness:

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

$$\mathbf{p}' = \mathbf{R}\mathbf{p}$$

- Rotation is a linear transformation

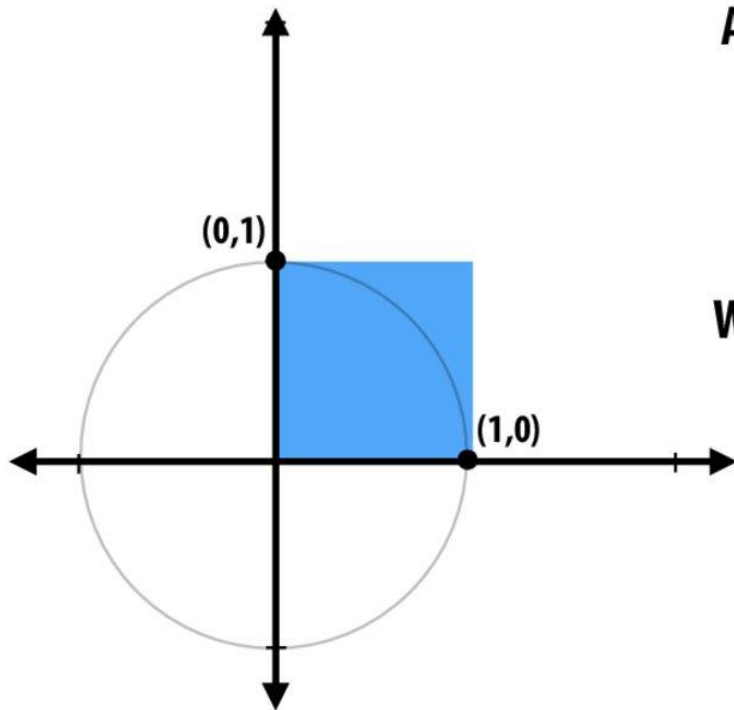


# Rotation matrix (2D)

**Question:** what happens to  $(1, 0)$  and  $(0, 1)$  after rotation by  $\theta$ ?

**Reminder:** rotation moves points along circular trajectories.

**(Recall that  $\cos \theta$  and  $\sin \theta$  are the coordinates of a point on the unit circle.)**



**Answer:**

$$R_{\theta}(1, 0) = (\cos(\theta), \sin(\theta))$$

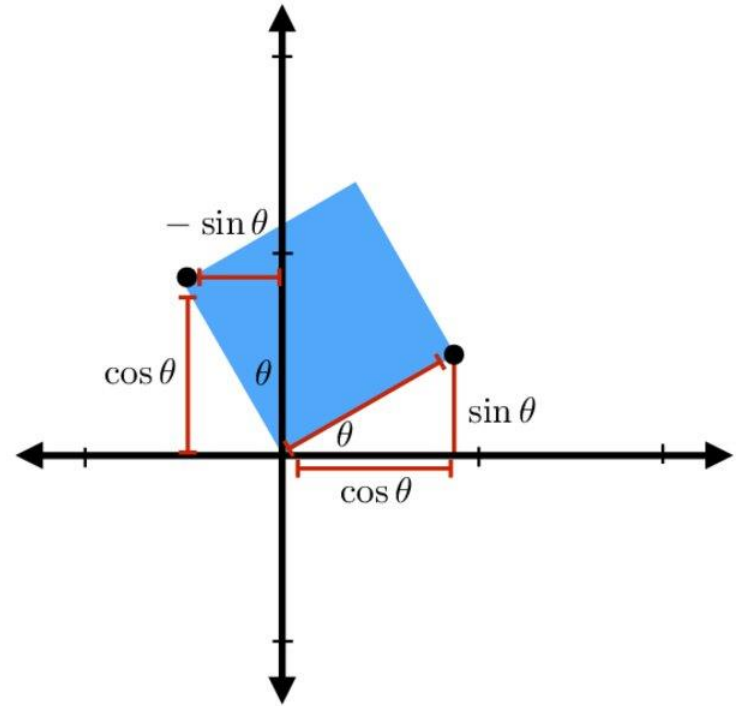
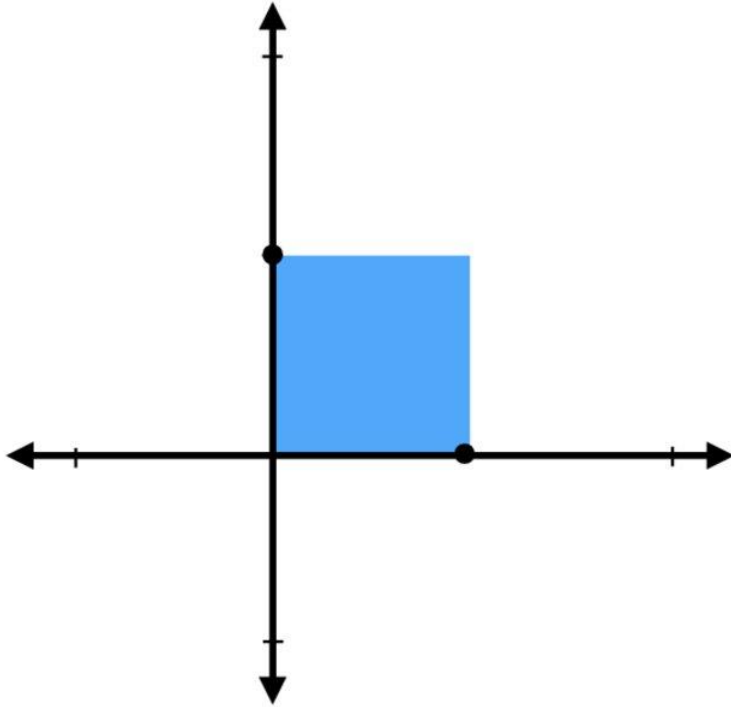
$$R_{\theta}(0, 1) = (\cos(\theta + \pi/2), \sin(\theta + \pi/2))$$

**Which means the matrix must look like:**

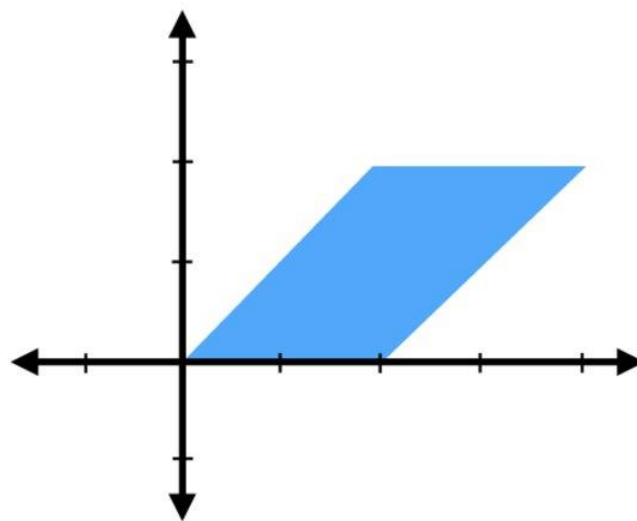
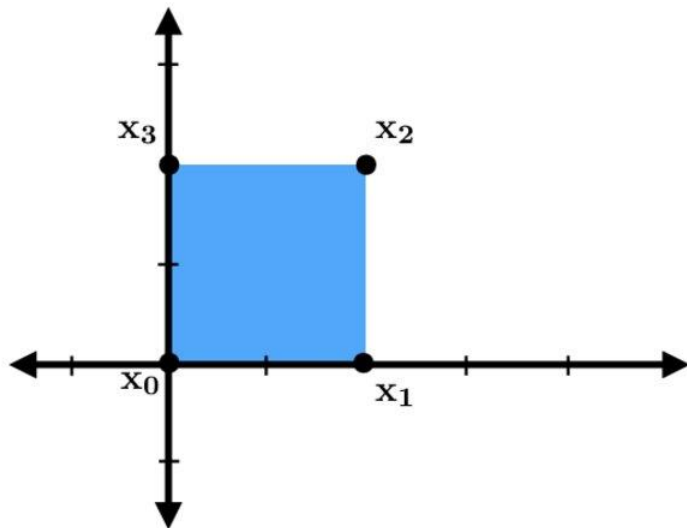
$$\begin{aligned} R_{\theta} &= \begin{bmatrix} \cos(\theta) & \cos(\theta + \pi/2) \\ \sin(\theta) & \sin(\theta + \pi/2) \end{bmatrix} \\ &= \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \end{aligned}$$

# Rotation matrix (2D): another way...

$$\mathbf{R}_\theta = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}$$



# Shear

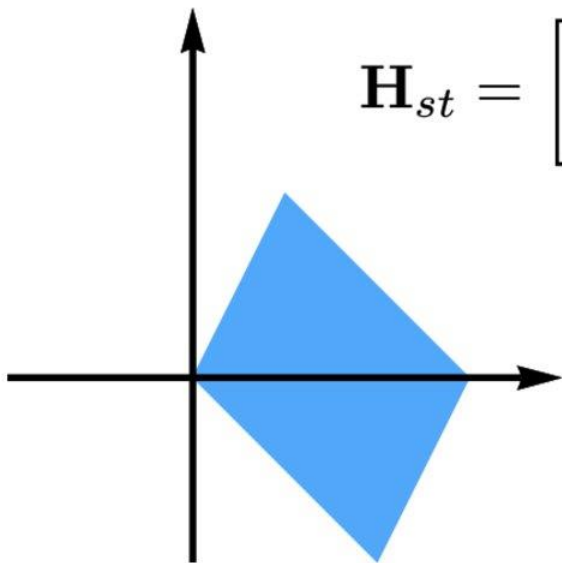


**Shear in x:**

$$\mathbf{H}_{xs} = \begin{bmatrix} 1 & s \\ 0 & 1 \end{bmatrix}$$

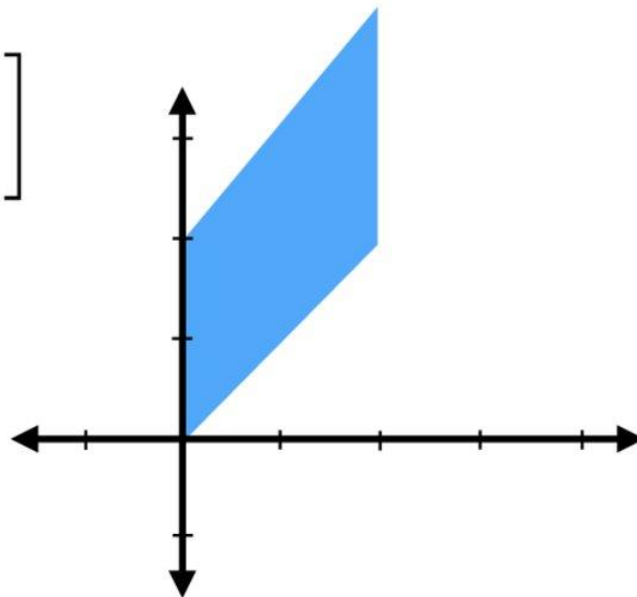
**Arbitrary shear:**

$$\mathbf{H}_{st} = \begin{bmatrix} 1 & s \\ t & 1 \end{bmatrix}$$



**Shear in y:**

$$\mathbf{H}_{ys} = \begin{bmatrix} 1 & 0 \\ s & 1 \end{bmatrix}$$



# Fundamental Transformations

---

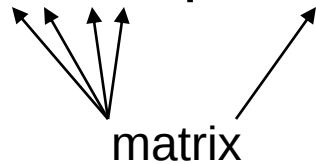
|             |                                         |
|-------------|-----------------------------------------|
| Translation | $\mathbf{p}' = \mathbf{p} + \mathbf{d}$ |
| Scaling     | $\mathbf{p}' = \mathbf{S}\mathbf{p}$    |
| Rotation    | $\mathbf{p}' = \mathbf{R}\mathbf{p}$    |

## Affine Combinations

- Any affine transformation can be represented as a sequence of these three fundamental transformations
- Because translation isn't a linear transformation, it can't be represented as an  $n \times n$  matrix in  $\mathbb{R}^n$
- This prevents us from writing affine transformations as concise sequences of  $n \times n$  matrix multiplications...

# How do we compose transformations?

- Translate then rotate then translate then scale?
- $S(R(p+T)+T)$
- Not very clean or compressible, homogeneous coordinates are a clever trick to turn this into:
- $STRT^*p = M^*p$



2D *homogeneous* coordinates

# 2D homogeneous coordinates (2D-H)

**Idea: represent 2D points with THREE values (“homogeneous coordinates”)**

**So the point  $(x, y)$  is represented as the 3-vector:  $\begin{bmatrix} x & y & 1 \end{bmatrix}^T$**

**And transformations are represented a 3x3 matrices that transform these vectors.**

**Recover final 2D coordinates by dividing by “extra” (third) coordinate**

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} \Rightarrow \begin{bmatrix} x/w \\ y/w \end{bmatrix}$$

**(More on this later...)**

# Example: scale and rotation in 2D-H coords

- For transformations that are already linear, not much changes:

$$\mathbf{S}_s = \begin{bmatrix} \mathbf{S}_x & 0 & 0 \\ 0 & \mathbf{S}_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{R}_\theta = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Notice that the last row/column doesn't do anything interesting. E.g., for scaling:

$$\mathbf{S}_s \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{S}_x x \\ \mathbf{S}_y y \\ 1 \end{bmatrix}$$

Now we divide by the 3rd coordinate to get our final 2D coordinates (not too exciting!)

$$\begin{bmatrix} \mathbf{S}_x x \\ \mathbf{S}_y y \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} \mathbf{S}_x x / 1 \\ \mathbf{S}_y y / 1 \end{bmatrix} = \begin{bmatrix} \mathbf{S}_x x \\ \mathbf{S}_y y \end{bmatrix}$$

(Will get more interesting when we talk about *perspective*...)

# Translation in 2D homogeneous coordinates

Translation expressed as 3x3 matrix multiplication:

$$\mathbf{T}_b = \begin{bmatrix} 1 & 0 & \mathbf{b}_x \\ 0 & 1 & \mathbf{b}_y \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{T}_b \mathbf{x} = \begin{bmatrix} 1 & 0 & \mathbf{b}_x \\ 0 & 1 & \mathbf{b}_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{x}_x \\ \mathbf{x}_y \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{x}_x + \mathbf{b}_x \\ \mathbf{x}_y + \mathbf{b}_y \\ 1 \end{bmatrix} \quad \text{(remember: just a linear combination of columns!)}$$

**Cool:** homogeneous coordinates let us encode translations as *linear* transformations!

# Transformations in Homogenous Coordinates

---

## Scaling

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} r & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} rx \\ sy \\ 1 \end{bmatrix}$$

## Rotation

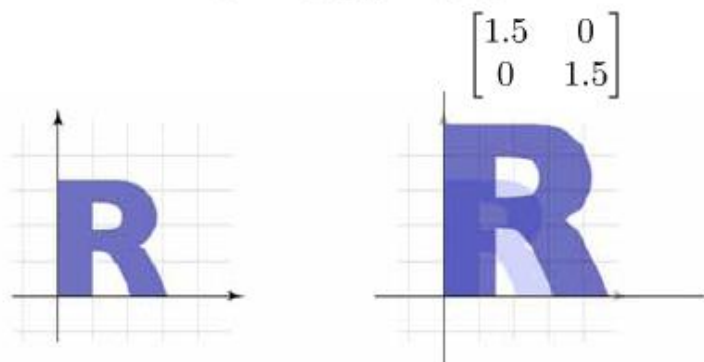
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x \cos \theta - y \sin \theta \\ x \sin \theta + y \cos \theta \\ 1 \end{bmatrix}$$

## Translation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + \Delta x \\ y + \Delta y \\ 1 \end{bmatrix}$$

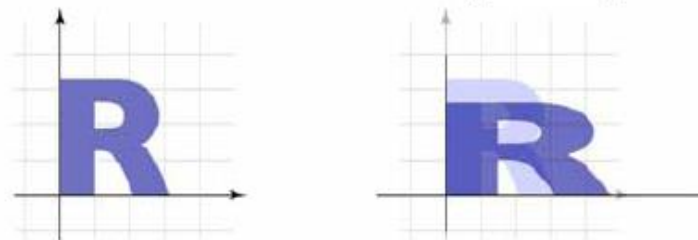
## Linear transformation gallery

- Uniform scale  $\begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} sx \\ sy \end{bmatrix}$



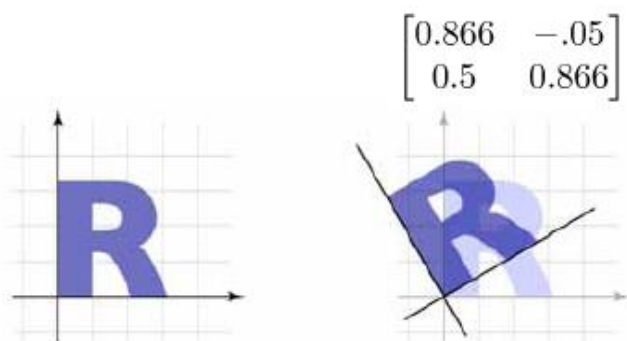
## Linear transformation gallery

- Nonuniform scale  $\begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} s_x x \\ s_y y \end{bmatrix}$



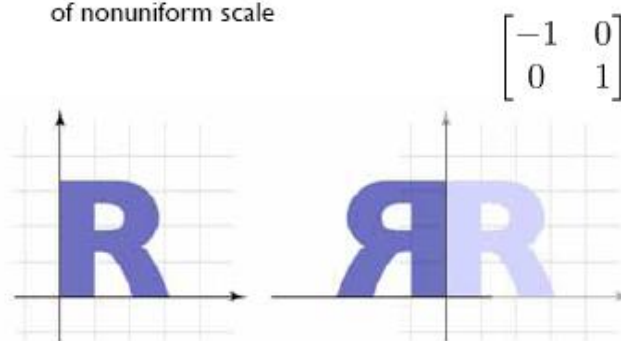
## Linear transformation gallery

- Rotation  $\begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x \cos \theta - y \sin \theta \\ x \sin \theta + y \cos \theta \end{bmatrix}$



## Linear transformation gallery

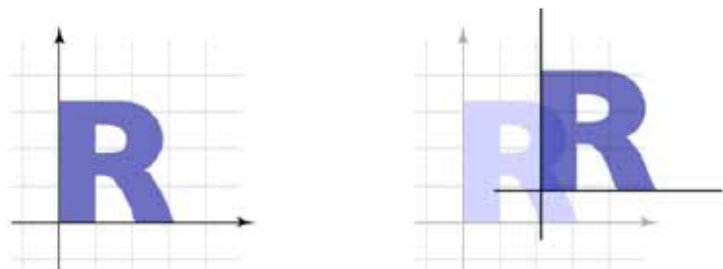
- Reflection
  - can consider it a special case of nonuniform scale



## Affine transformation gallery

- Translation

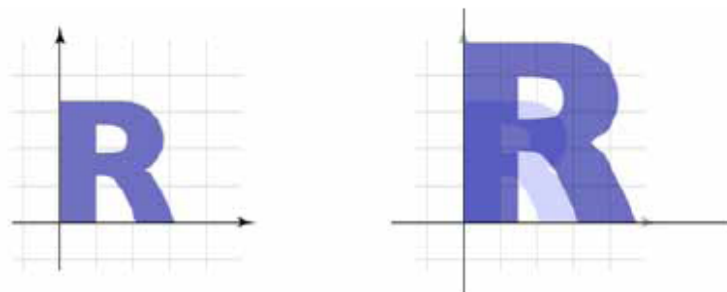
$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 2.15 \\ 0 & 1 & 0.85 \\ 0 & 0 & 1 \end{bmatrix}$$



## Affine transformation gallery

- Uniform scale

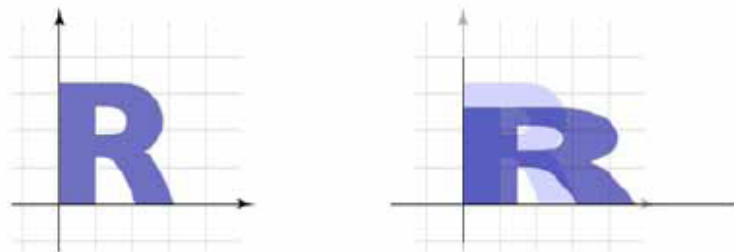
$$\begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1.5 & 0 & 0 \\ 0 & 1.5 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



## Affine transformation gallery

- Nonuniform scale

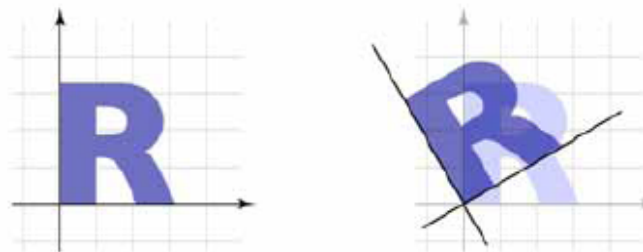
$$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1.5 & 0 & 0 \\ 0 & 0.8 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



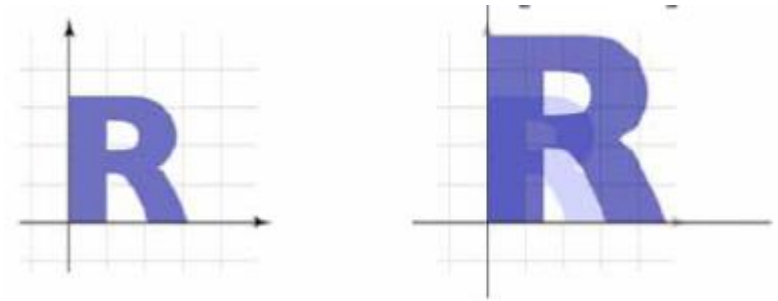
## Affine transformation gallery

- Rotation

$$\begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 0.866 & -0.5 & 0 \\ 0.5 & 0.866 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



# Q on 2D transforms



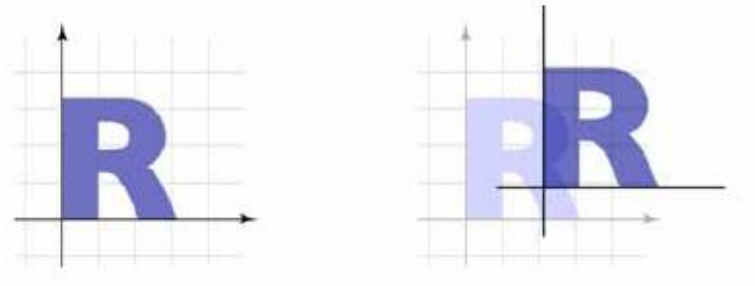
(A) 
$$\begin{bmatrix} 1 & 0 & 2.15 \\ 0 & 1 & 0.85 \\ 0 & 0 & 1 \end{bmatrix}$$

(B) 
$$\begin{bmatrix} 1.5 & 0 & 0 \\ 0 & 1.5 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(C) 
$$\begin{bmatrix} 1.5 & 0 & 0 \\ 0 & 0.8 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(D) 
$$\begin{bmatrix} 0.866 & -0.5 & 0 \\ 0.5 & 0.866 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

# Q on 2D transforms

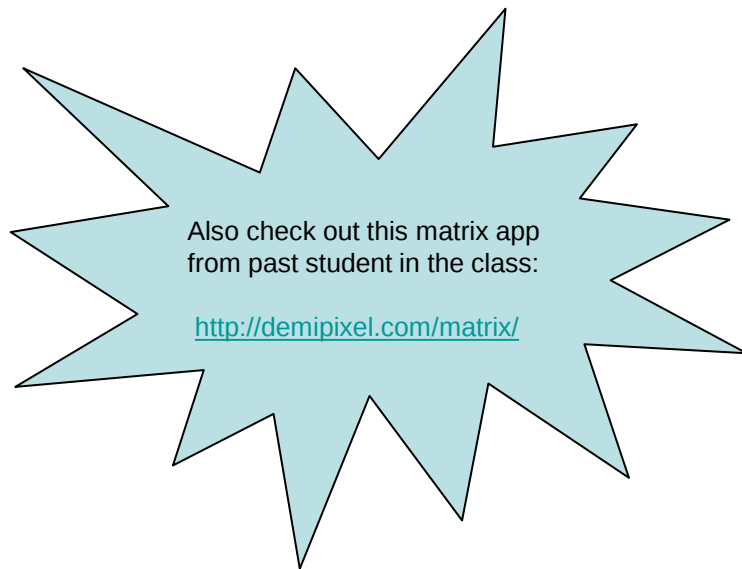


(A) 
$$\begin{bmatrix} 1 & 0 & 2.15 \\ 0 & 1 & 0.85 \\ 0 & 0 & 1 \end{bmatrix}$$

(B) 
$$\begin{bmatrix} 1.5 & 0 & 0 \\ 0 & 1.5 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

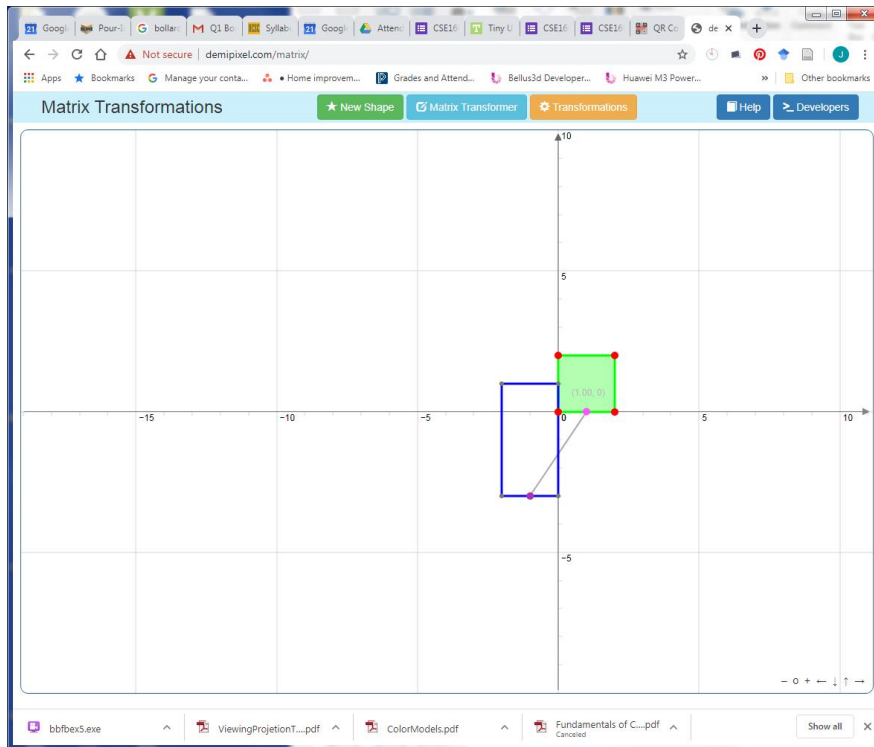
(C) 
$$\begin{bmatrix} 1.5 & 0 & 0 \\ 0 & 0.8 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(D) 
$$\begin{bmatrix} 0.866 & -0.5 & 0 \\ 0.5 & 0.866 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



# Participation Survey

<http://tiny.cc/160-1015>



## Participation Apr 14

Form description

This form is automatically collecting email addresses for UC Santa Cruz users. [Change settings](#)

I was in class Apr 14

- ☐ Yes
- ☐ No

I have finished Lab A1

- ☐ Yes
- ☐ No, but I will pretty soon
- ☐ No, and I am really stuck
- ☐ Other...

We have tried Breakout Rooms in Zoom for discussing short questions in class. ("Which answer is right? ABCDE?") Should we do this?

Suggestions: [Add all](#) | [Yes](#) | [No](#) | [Maybe](#)

- ☐ Yes, its good to discuss with classmates in smaller group
- ☐ No, technical problems were too high, so not worth it
- ☐ Other...

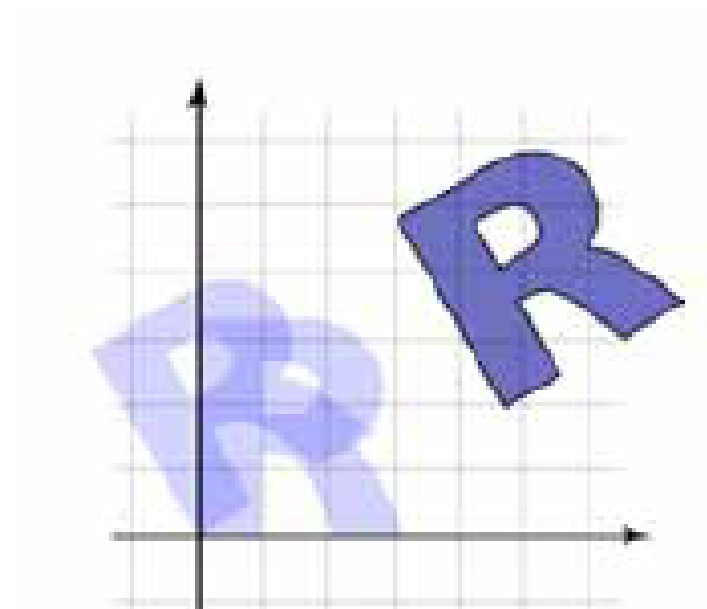
We have tried both [Slido](#) and Zoom Chat for question asking in class. Which do you prefer?

- ☐ Slido (separate web page with upvote interface)

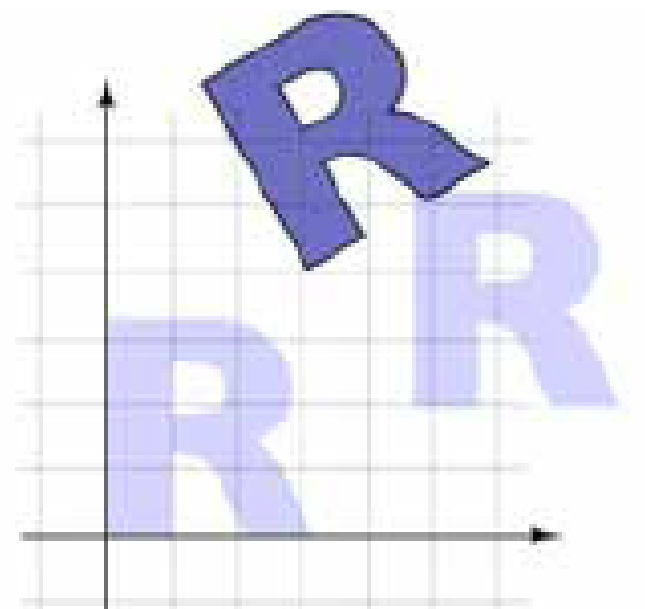
Matrix order matters

# Composite affine transformations

- In general **not** commutative: order matters!

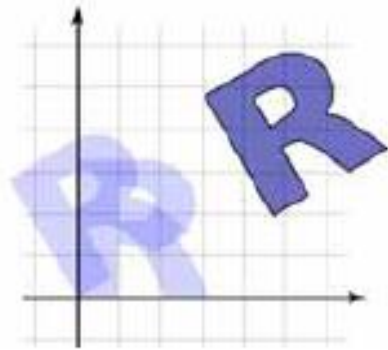


rotate, then translate

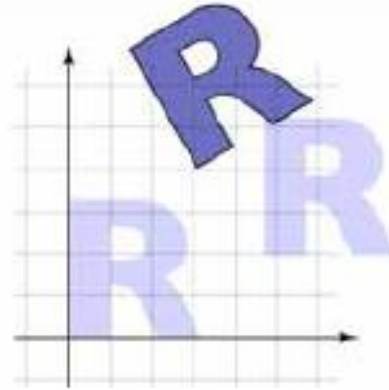


translate, then rotate

# Multiply matrices in which order?



rotate, then translate



first  
second  
translate, then rotate

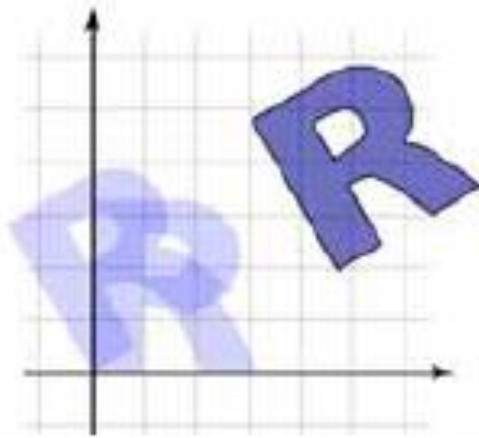
$$p' = [R] [T] p$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

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

# OpenGL Right multiplies matrices

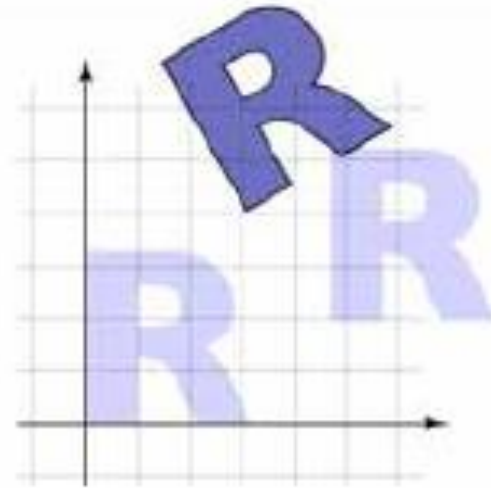


first  
rotate, then translate

$$p' = [T][R]p$$

```
Matrix M;  
M.translate();  
M.rotate();  
draw();
```

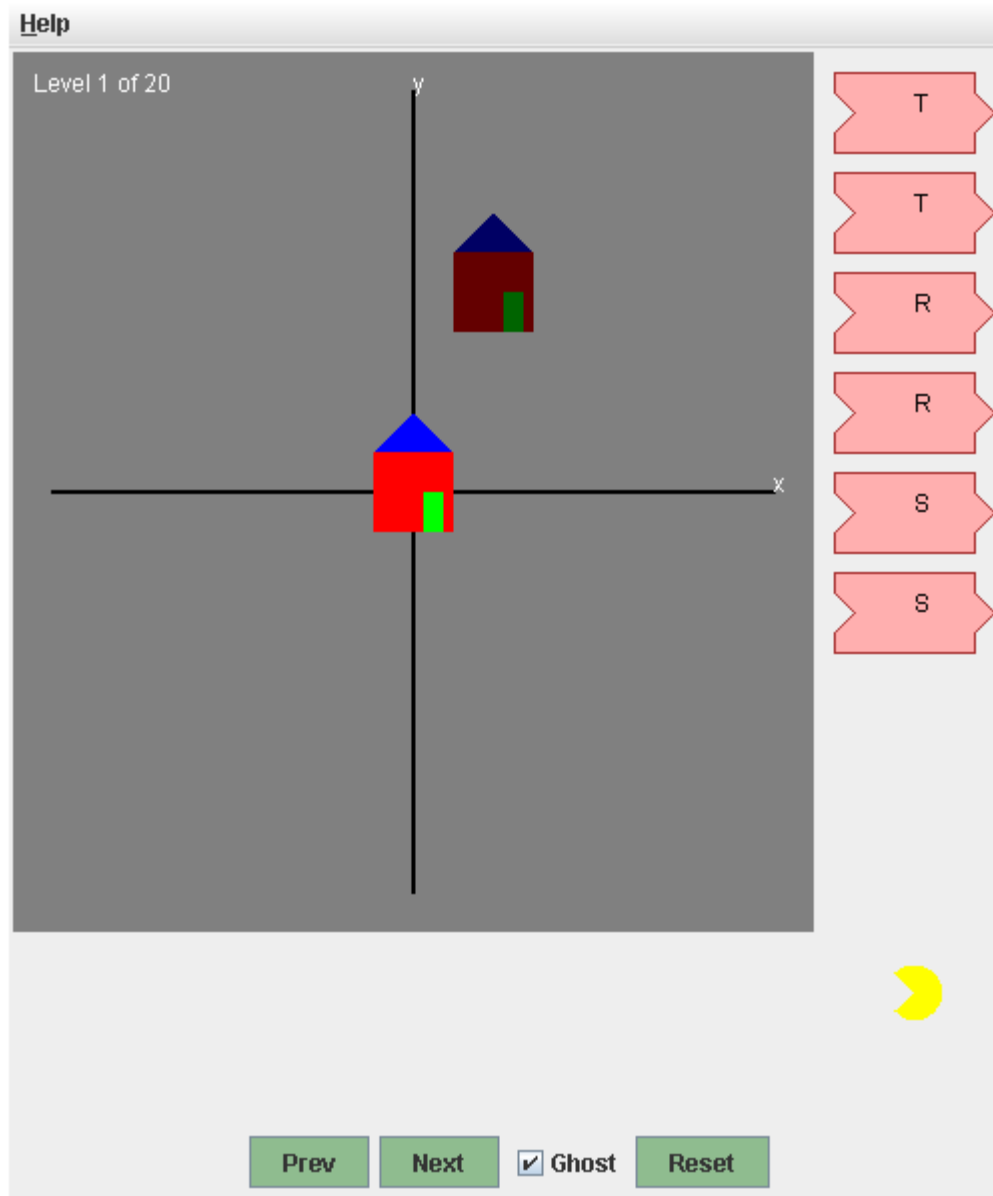
second  
first



translate, then rotate

$$p' = [R][T]p$$

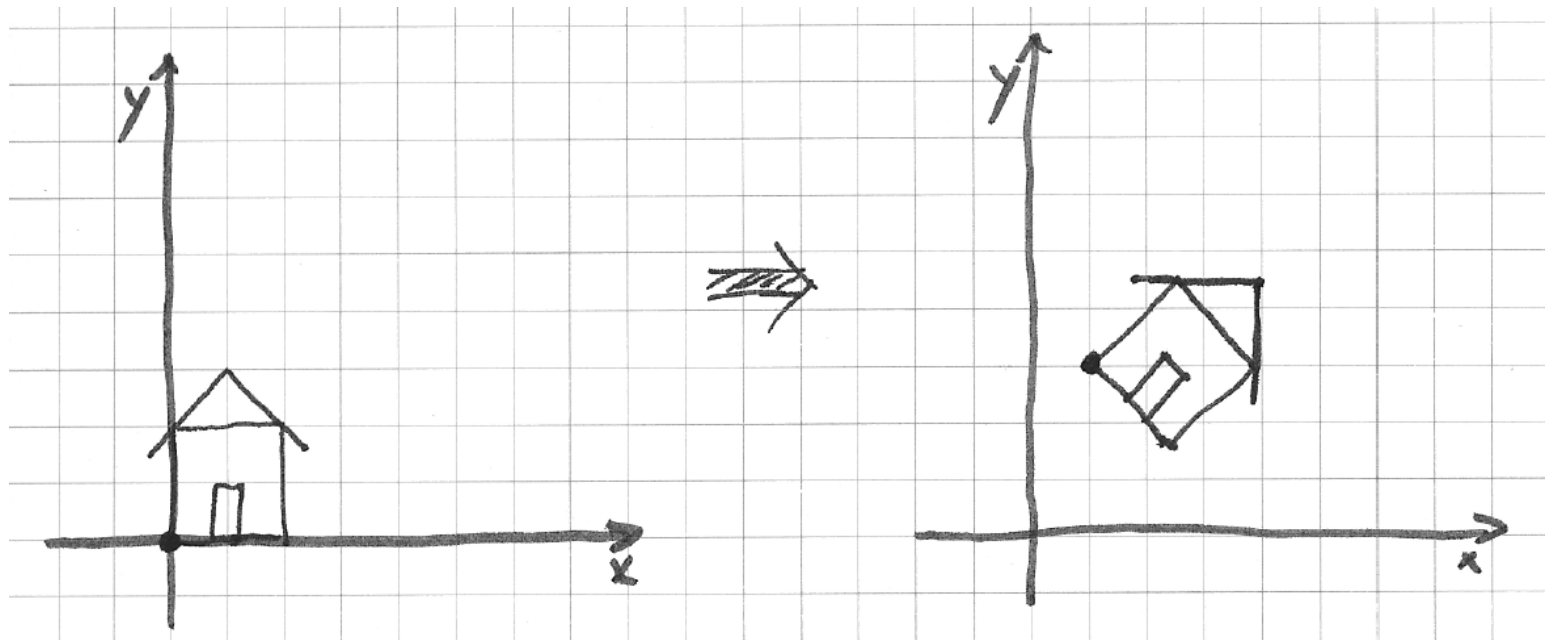
```
Matrix M;  
M.rotate();  
M.translate();  
draw();
```



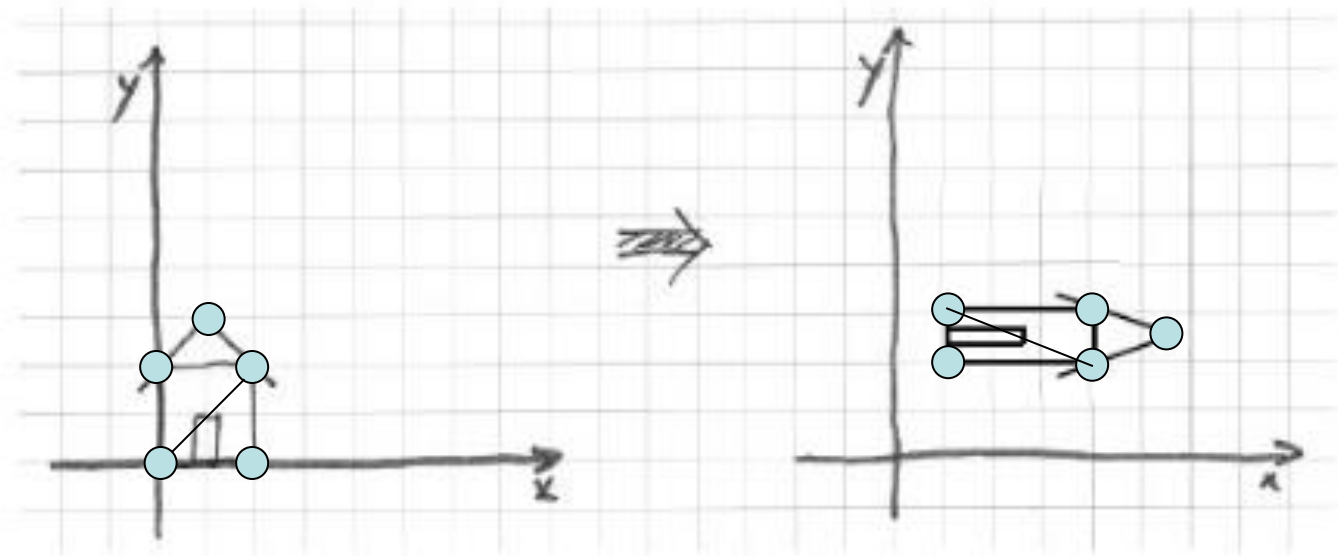
[http://www.cs.brown.edu/exploratories/freeSoftware/repository/edu/brown/cs/exploratories/applets/transformationGame/transformation\\_game\\_java\\_browser.html](http://www.cs.brown.edu/exploratories/freeSoftware/repository/edu/brown/cs/exploratories/applets/transformationGame/transformation_game_java_browser.html)

You need to download the .jar file and run locally

2. What is the matrix necessary to perform the following 2D transformation?  
(You don't need to actually solve the math, just set up far enough that I would get an actual matrix result if the math was completed.)



# Exercise: Find the function()



Vertex buffer array

0,0  
2,2  
2,0  
0,2  
1,3  
2,2  
....

→  $f()$  →

Vertex buffer array

1,3  
4,2  
1,2  
4,3  
5.5, 2.5  
4,2  
....

# 3D Transformations

# Scaling & Translation in 3D

---

## Scaling

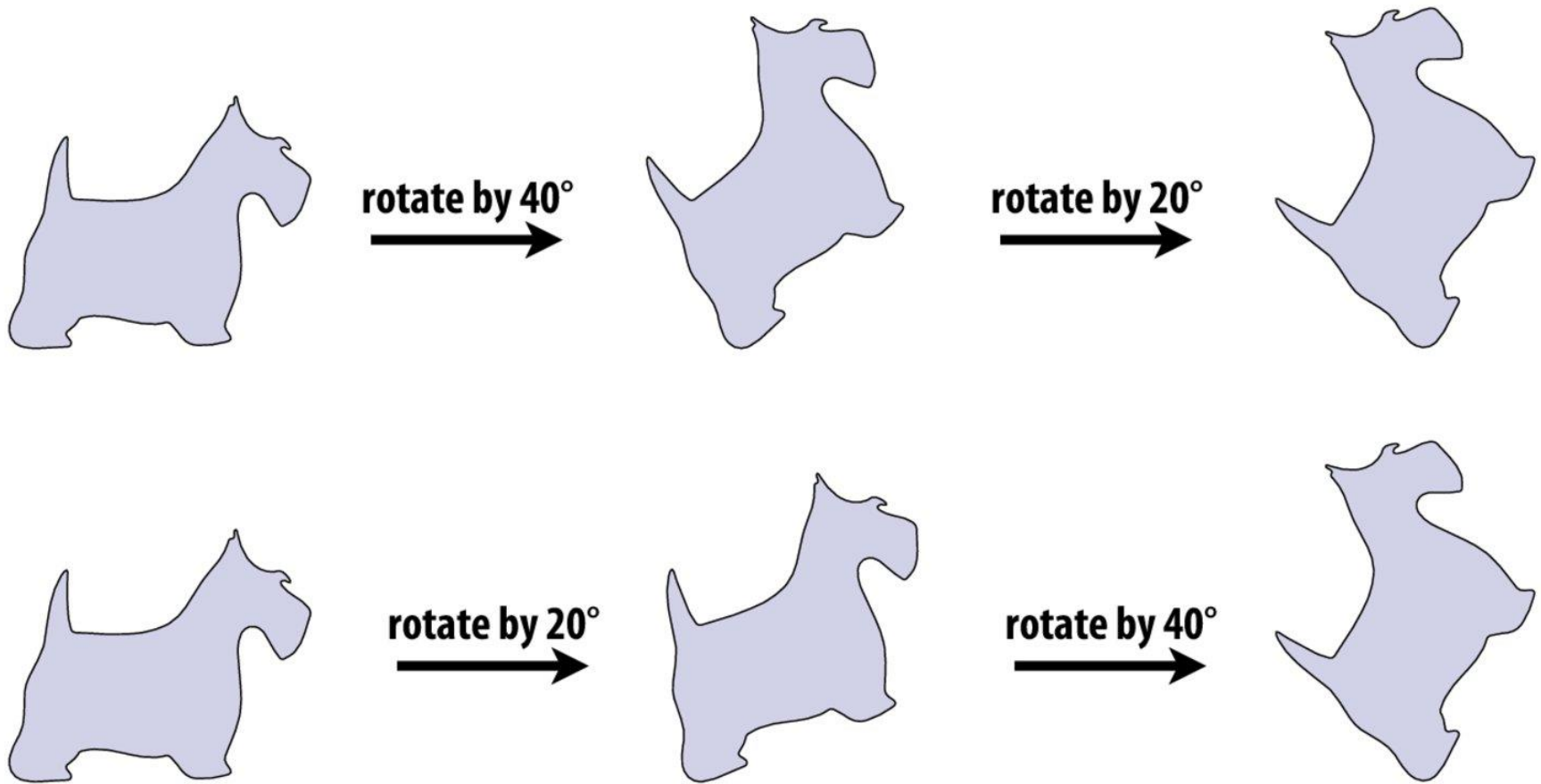
$$\mathbf{S}(r, s, t) = \begin{bmatrix} r & 0 & 0 & 0 \\ 0 & s & 0 & 0 \\ 0 & 0 & t & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

## Translation

$$\mathbf{T}(x, y, z) = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# Commutativity of rotations—2D

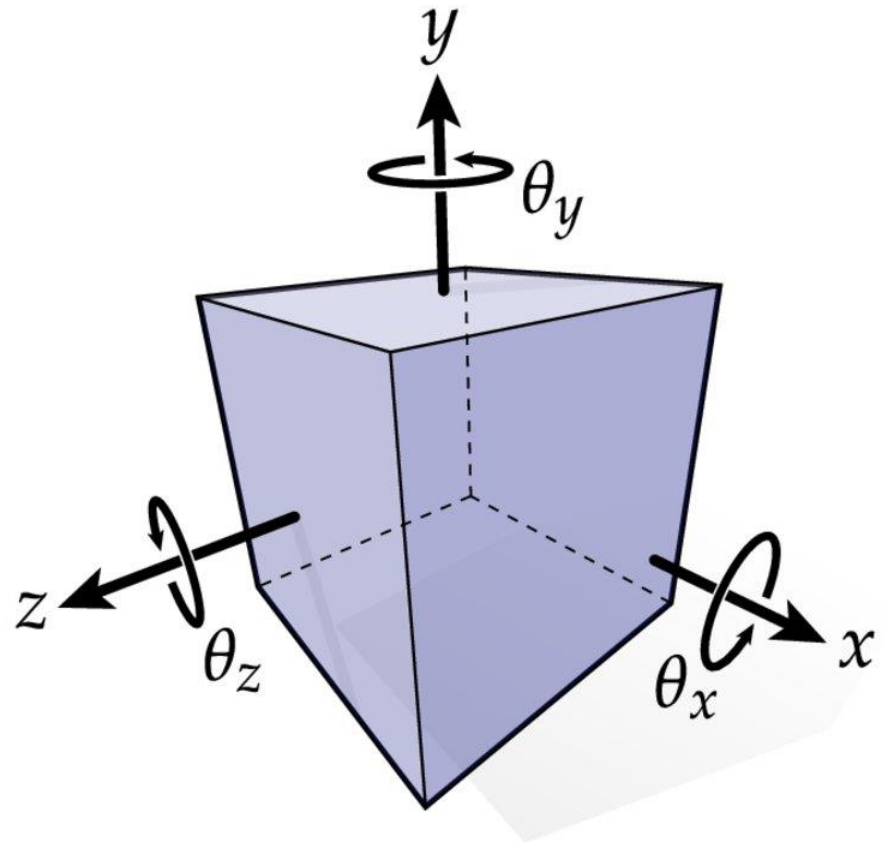
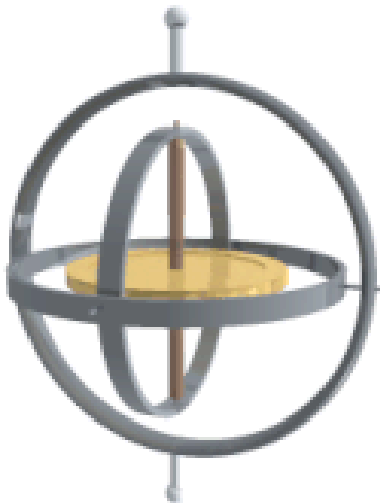
- In 2D, order of rotations doesn't matter:



**Same result! ("2D rotations commute")**

# Representing rotations in 3D—euler angles

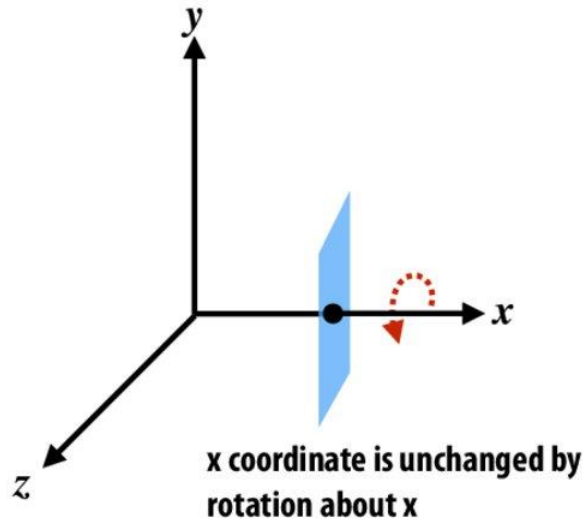
- How do we express rotations in 3D?
- One idea: we know how to do 2D rotations
- Why not simply apply rotations around the three axes? (X,Y,Z)
- Scheme is called *Euler angles*



# Rotations in 3D

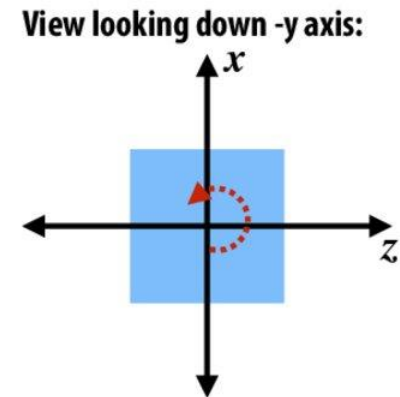
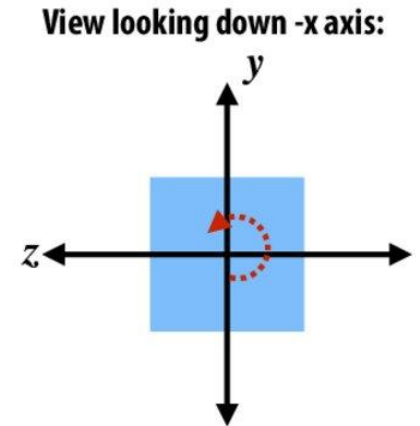
## Rotation about x axis:

$$\mathbf{R}_{x,\theta} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta \\ 0 & \sin \theta & \cos \theta \end{bmatrix}$$



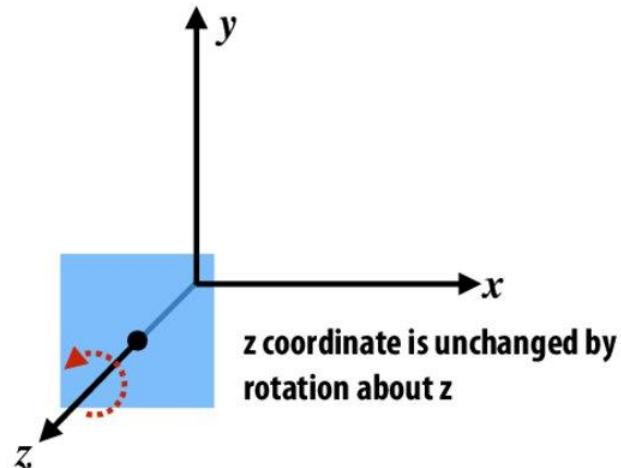
## Rotation about y axis:

$$\mathbf{R}_{y,\theta} = \begin{bmatrix} \cos \theta & 0 & \sin \theta \\ 0 & 1 & 0 \\ -\sin \theta & 0 & \cos \theta \end{bmatrix}$$



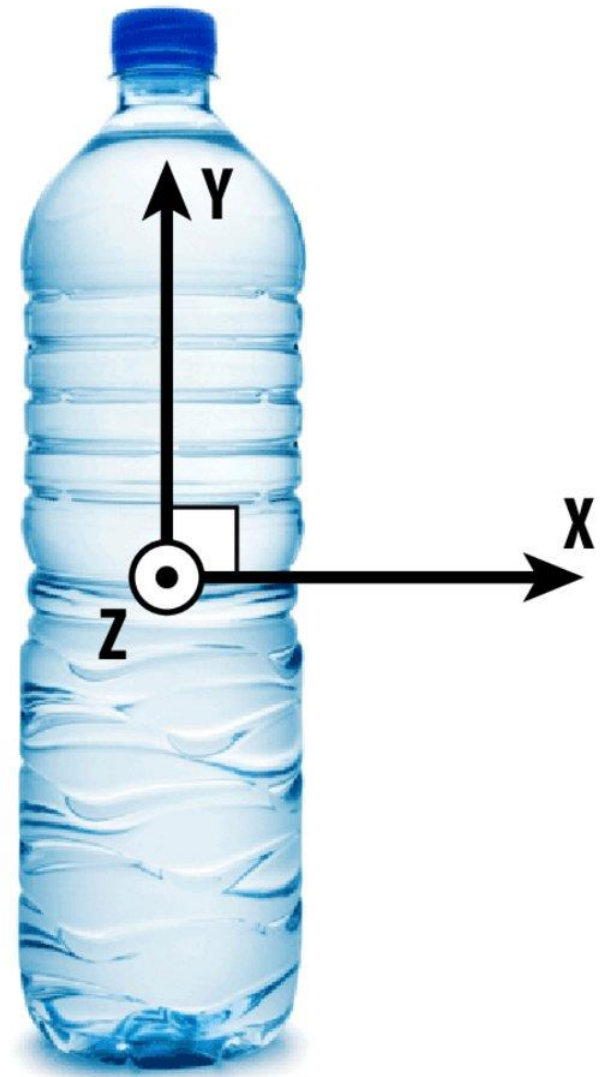
## Rotation about z axis:

$$\mathbf{R}_{z,\theta} = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



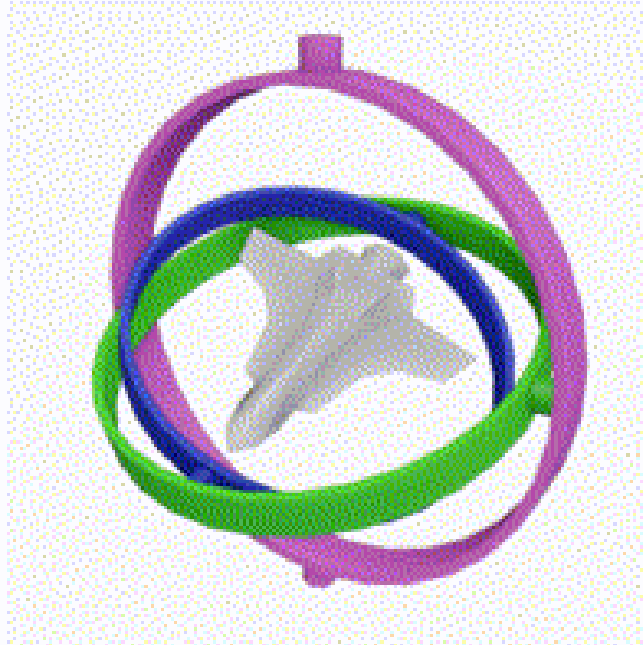
# Commutativity of rotations—3D

- What about in 3D?
- IN-CLASS ACTIVITY:
  - Rotate  $90^\circ$  around Y, then  $90^\circ$  around Z, then  $90^\circ$  around X
  - Rotate  $90^\circ$  around Z, then  $90^\circ$  around Y, then  $90^\circ$  around X
  - (Was there any difference?)



**CONCLUSION: bad things can happen if we're not careful about the order in which we apply rotations!**

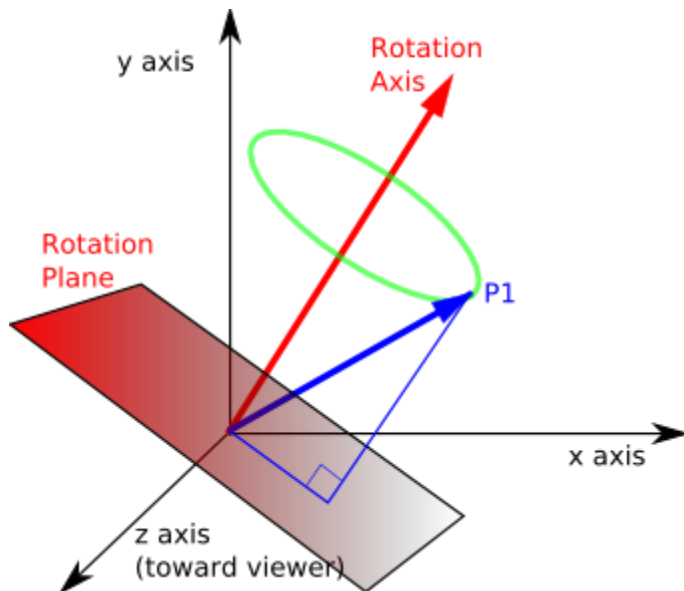
# Gimble Lock – lose a degree of freedom



Gimbal locked airplane. When the pitch (green) and yaw (magenta) gimbals become aligned, changes to roll (blue) and yaw apply the same rotation to the airplane. (from Wikipedia)

# Alternate rotation : axis-angle

- `glRotate(angle, x,y,z);`



`glRotate{F}d(angle, x,y,z);`

↓ produces this  
4x4 matrix

↑ vector around which rotation will occur,  
+angle ⇔ ccw rotation when  
looking along vector towards origin.

$$\begin{bmatrix} x^2(1-\cos\theta)+\cos\theta & xy(1-\cos\theta)-z\sin\theta & xz(1-\cos\theta)+y\sin\theta & 0 \\ yx(1-\cos\theta)+z\sin\theta & y^2(1-\cos\theta)+\cos\theta & yz(1-\cos\theta)-x\sin\theta & 0 \\ xz(1-\cos\theta)-y\sin\theta & yz(1-\cos\theta)+x\sin\theta & z^2(1-\cos\theta)+\cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# Alternate rotation: Quaternions

Not in this class

## Quaternions [\[ edit \]](#)

*Main article: [Quaternions](#)*

The [complex numbers](#) can be defined by introducing an abstract symbol  $\mathbf{i}$  which satisfies the usual rules of algebra and additionally the rule  $\mathbf{i}^2 = -1$  for arithmetic: for example:

$$(a + b\mathbf{i})(c + d\mathbf{i}) = ac + a d\mathbf{i} + b c\mathbf{i} + b d\mathbf{i}^2 = ac + a d\mathbf{i} + b c\mathbf{i} + b d(-1) = (ac - bd) + (bc + ad)\mathbf{i}.$$

In the same way the quaternions can be defined by introducing abstract symbols  $\mathbf{i}$ ,  $\mathbf{j}$ ,  $\mathbf{k}$  which satisfy the rules  $\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1$  (an example of such a noncommutative multiplication is [matrix multiplication](#)). From this all of the rules of quaternion arithmetic follow, such as  $\mathbf{ij} = -\mathbf{ji}$ . One can show that:

$$(a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k})(e + f\mathbf{i} + g\mathbf{j} + h\mathbf{k}) = (ae - bf - cg - dh) + (af + be + ch - dg)\mathbf{i} + (ag - bh + ce + df)\mathbf{j} + (ah + bg - cf + de)\mathbf{k}.$$

The imaginary part  $b\mathbf{i} + c\mathbf{j} + d\mathbf{k}$  of a quaternion behaves like a vector  $\vec{v} = (b, c, d)$  in three dimension vector space, and the real part  $a$  can be thought of as a scalar. It is convenient to define them as [a scalar plus a vector](#):

$$a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k} = a + \vec{v}.$$

Those who have studied vectors at school might find it strange to add a *number* to a *vector*, as they are objects of very different natures, but if one remembers that it is a mere notation for the real and imaginary parts of a quaternion, it becomes more legitimate. In other words, the quaternion is the sum of a scalar/real part, and another one with zero scalar/real part:

$$a + \vec{v} = (a, \vec{0}) + (0, \vec{v}).$$

# Heirarchical Transforms

# Skeleton - hierarchical representation

torso

head

right arm

upper arm

lower arm

hand

left arm

upper arm

lower arm

hand

right leg

upper leg

lower leg

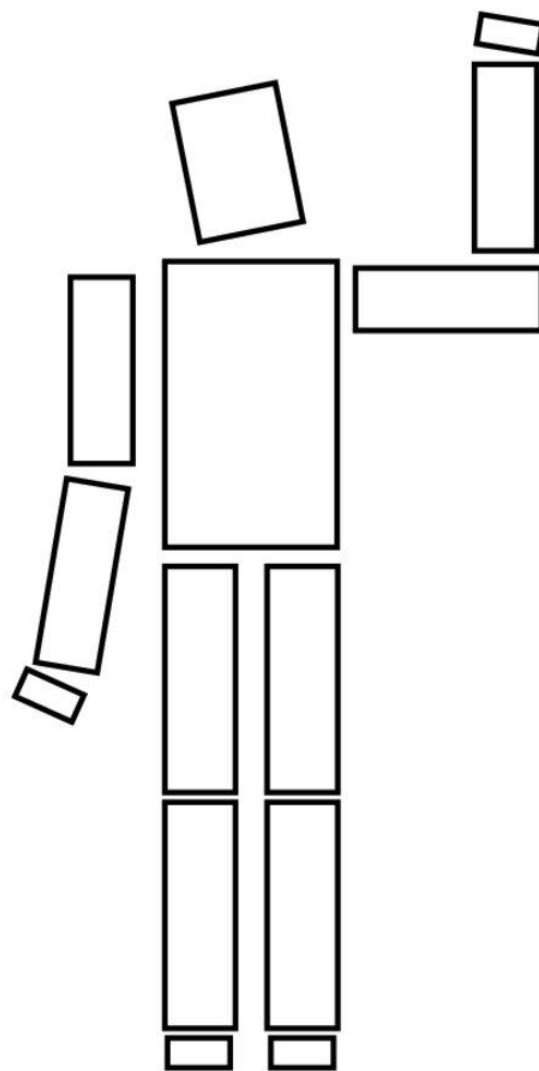
foot

left leg

upper leg

lower leg

foot



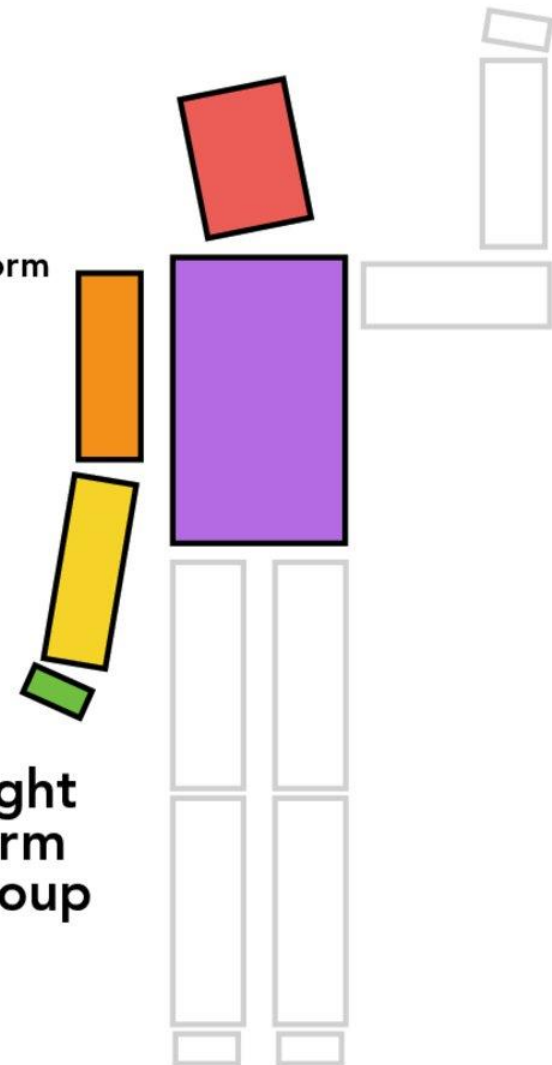
# Skeleton - hierarchical representation

```
translate(0, 10);
drawTorso();
pushmatrix(); // push a copy of transform onto stack
  translate(0, 5); // right-multiply onto current transform
  rotate(headRotation); // right-multiply onto current transform
  drawHead();
popmatrix(); // pop current transform off stack
pushmatrix(); -----
  translate(-2, 3);
  rotate(rightShoulderRotation);
  drawUpperArm();
  pushmatrix(); -----
    translate(0, -3);
    rotate(elbowRotation);
    drawLowerArm();
    pushmatrix(); -----
      translate(0, -3);
      rotate(wristRotation);
      drawHand();
    popmatrix(); -----
  popmatrix(); -----
popmatrix(); -----
....
```

right  
hand

right  
lower  
arm  
group

right  
arm  
group



# Skeleton - hierarchical representation

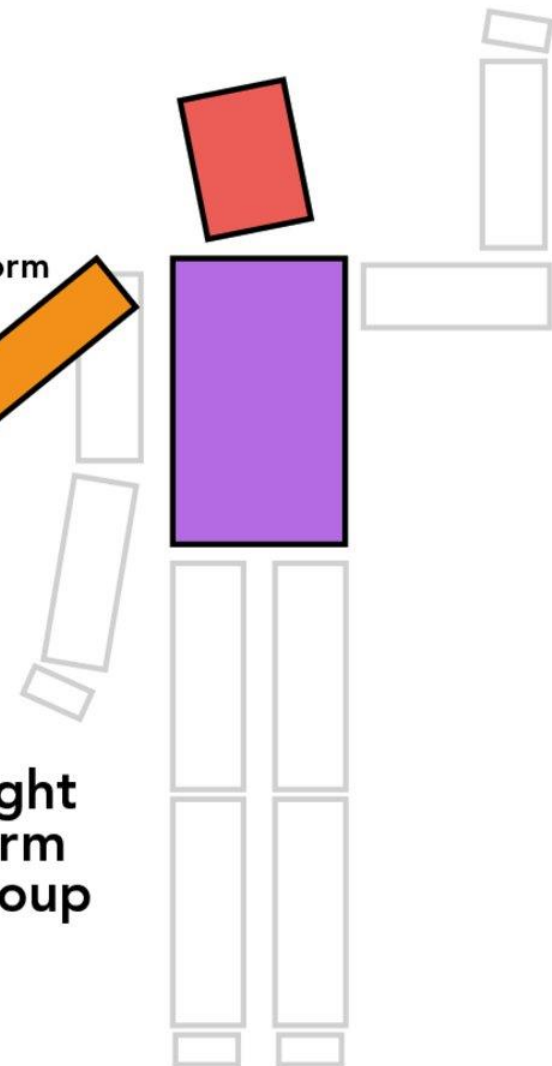
```
translate(0, 10);
drawTorso();
pushmatrix(); // push a copy of transform onto stack
  translate(0, 5); // right-multiply onto current transform
  rotate(headRotation); // right-multiply onto current transform
  drawHead();
popmatrix(); // pop current transform off stack
pushmatrix(); -----
  translate(-2, 3);
  rotate(rightShoulderRotation);
  drawUpperArm();
  pushmatrix(); -----
    translate(0, -3);
    rotate(elbowRotation);
    drawLowerArm();
    pushmatrix(); -----
      translate(0, -3);
      rotate(wristRotation);
      drawHand();
    popmatrix(); -----
  popmatrix(); -----
popmatrix(); -----
....
```



right  
hand

right  
lower  
arm  
group

right  
arm  
group



# WebGL transforms

# Vertex shader: add a matrix

Listing 3.6 RotatedTriangle\_Matrix.js

```
1 // RotatedTriangle_Matrix.js
2 // Vertex shader program
3 var VSHADER_SOURCE =
4     'attribute vec4 a_Position;\n' +
5     'uniform mat4 u_xformMatrix;\n' +
6     'void main() {\n' +
7     '    gl_Position = u_xformMatrix * a_Position;\n' +
8     '}\n';
9
```

$$P' = [M]p$$

# Javascript: Pass the matrix in a Uniform

```
19 function main() {  
    ...  
36    // Set the positions of vertices  
37    var n = initVertexBuffers(gl);  
    ...  
43    // Create a rotation matrix  
44    var radian = Math.PI * ANGLE / 180.0; // Convert to radians  
45    var cosB = Math.cos(radian), sinB = Math.sin(radian);  
46  
47    // Note: WebGL is column major order  
48    var xformMatrix = new Float32Array([  
49        cosB, sinB, 0.0, 0.0,  
50        -sinB, cosB, 0.0, 0.0,  
51        0.0, 0.0, 1.0, 0.0,  
52        0.0, 0.0, 0.0, 1.0  
53    ]);  
54  
55    // Pass the rotation matrix to the vertex shader  
56    var u_xformMatrix = gl.getUniformLocation(gl.program, 'u_xformMatrix');  
    ...  
61    gl.uniformMatrix4fv(u_xformMatrix, false, xformMatrix);  
62  
63    // Set the color for clearing <canvas>  
    ...  
69    // Draw a triangle  
70    gl.drawArrays(gl.TRIANGLES, 0, n);  
71 }
```

Rotation matrix

Send to GPU

Draw

#### Listing 4.2 RotatedTranslatedTriangle.js

```
1 // RotatedTranslatedTriangle.js
2 // Vertex shader program
3 var VSHADER_SOURCE =
4     'attribute vec4 a_Position;\n' +
5     'uniform mat4 u_ModelMatrix;\n' +
6     'void main() {\n' +
7     '    gl_Position = u_ModelMatrix * a_Position;\n' +
8     '}\n';
9 // Fragment shader program
10 ...
16 function main() {
17     ...
33     // Set the positions of vertices
34     var n = initVertexBuffers(gl);
35     ...
40     // Create Matrix4 object for model transformation
41     var modelMatrix = new Matrix4();
42     ...
43     // Calculate a model matrix
44     var ANGLE = 60.0; // Rotation angle
45     var Tx = 0.5; // Translation distance
46     modelMatrix.setRotate(ANGLE, 0, 0, 1); // Set rotation matrix
47     modelMatrix.translate(Tx, 0, 0); // Multiply modelMatrix by the calculated
48                                     // translation matrix
49     // Pass the model matrix to the vertex shader
50     var u_ModelMatrix = gl.getUniformLocation(gl.program, 'u_ModelMatrix');
51     ...
56     gl.uniformMatrix4fv(u_ModelMatrix, false, modelMatrix.elements);
57     ...
63     // Draw a triangle
64     gl.drawArrays(gl.TRIANGLES, 0, n);
65 }
```

Composing multiple matrices,  
but still just one in vertex shader

$[M] = \text{identityMatrix}$

$[M] = [R]$   
 $[M] = [R][T]$

Send to GPU

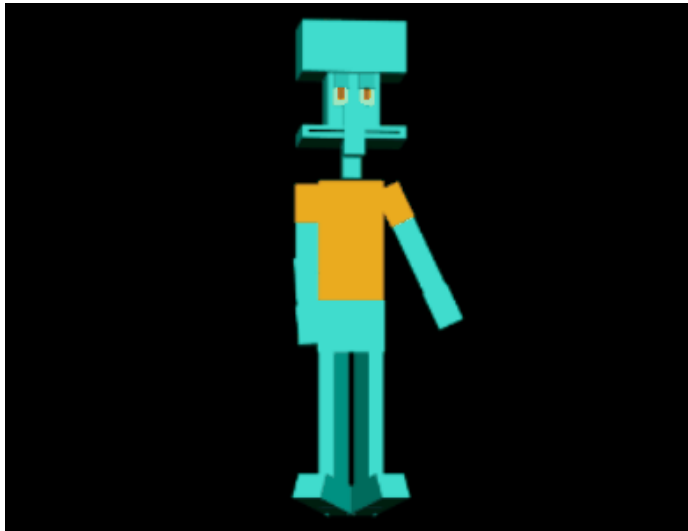
Draw

**Administrative**

# Due Dates

- Due *Already* (now late, but no point penalty yet)
  - Lab 0, HW1, Quiz1
- Due next Monday
  - Lab Assignment 1(Paint)
- Due 2 weeks Monday
  - HW2

# Assignment 2



**Your name here.**

UCSC CSE-160-01 Pages Hall of Fame

2020 Spring Quarter View All Pages

Hall of Fame

Spring 2020

Winter 2020

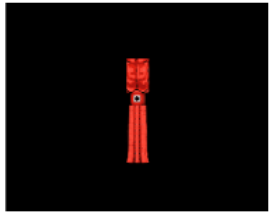
Assignment 2 - Bloody Animal



1st Place - Dragon - Herman Wu



2nd Place - Squidward by Sehej Singh Sohal- <https://people.ucsc.edu/~sssohal/A2/>



2nd Place - Squid by Kevin Macnair Pinney - <https://people.ucsc.edu/~kpinney/cse160/ps2>

Q&A

End