Computer Graphics
-Transformation

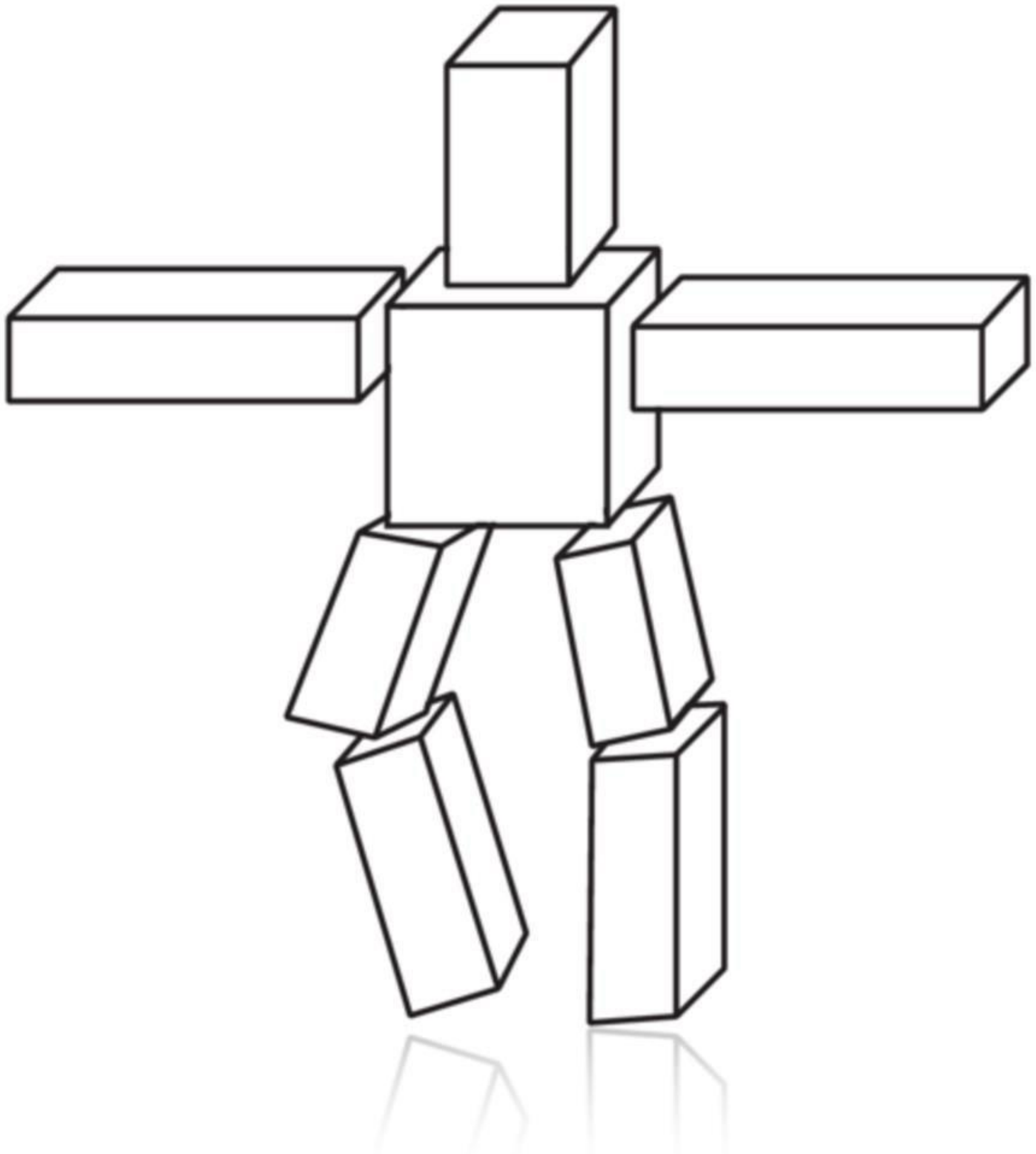
Junjie Cao @ DLUT

Spring 2018

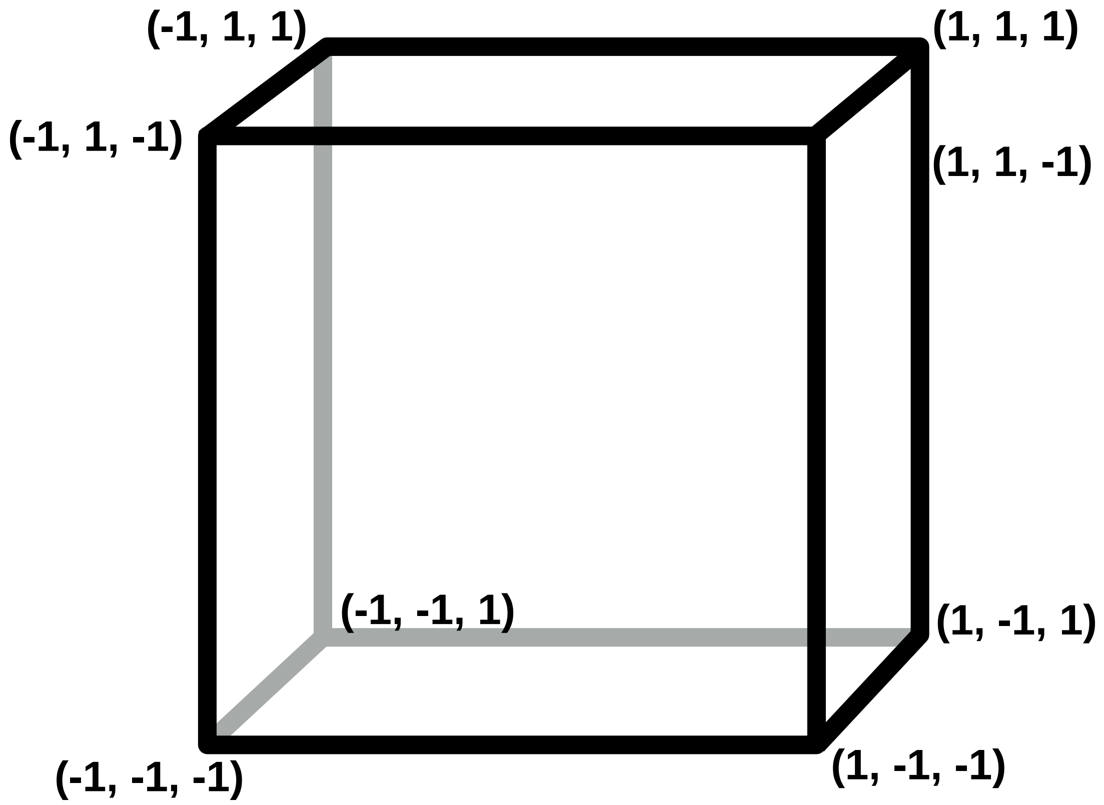
http://jjcao.github.io/ComputerGraphics/

Pleasure may come from illusion, but happiness can come only of reality.

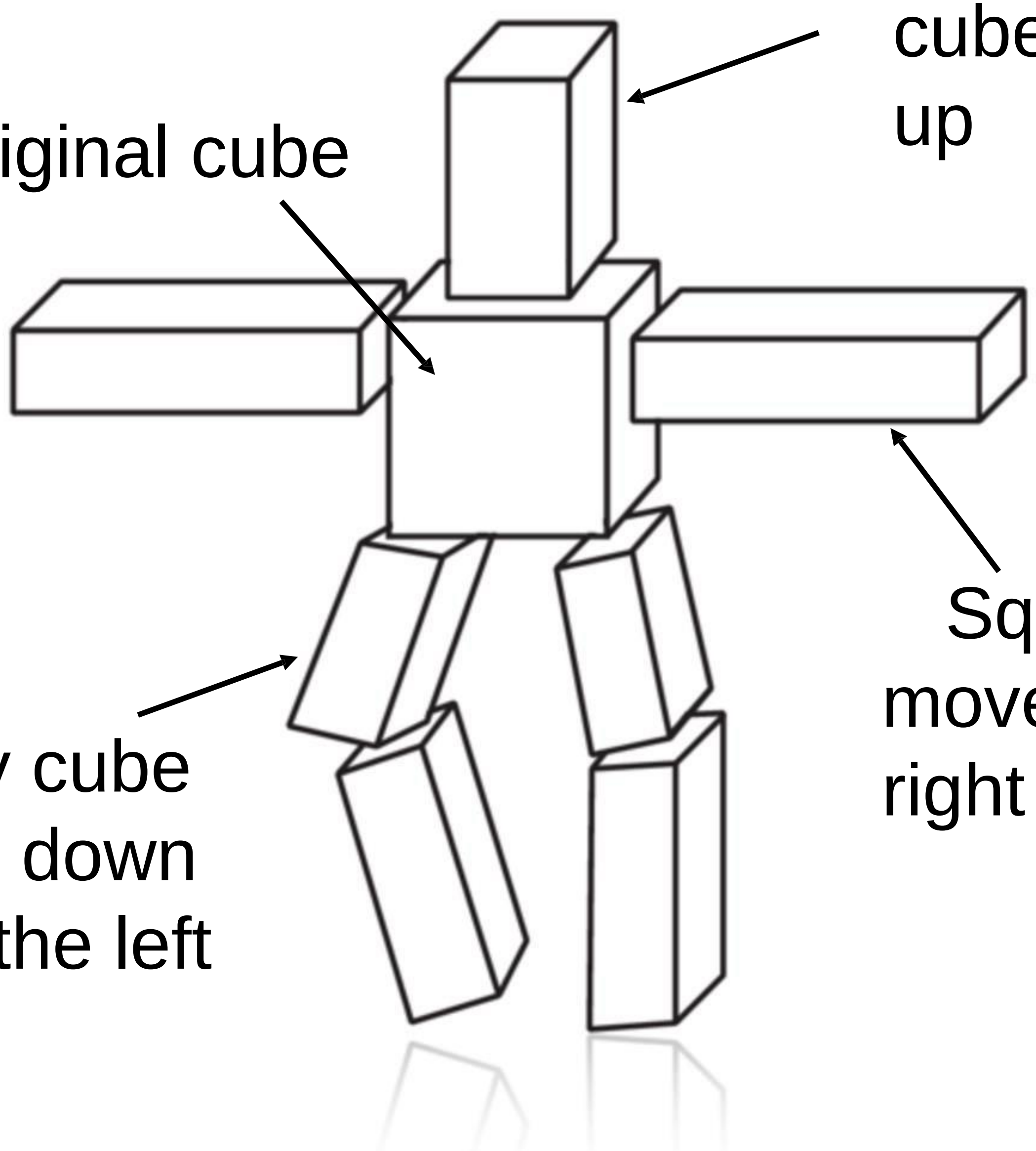
What in the world is this?



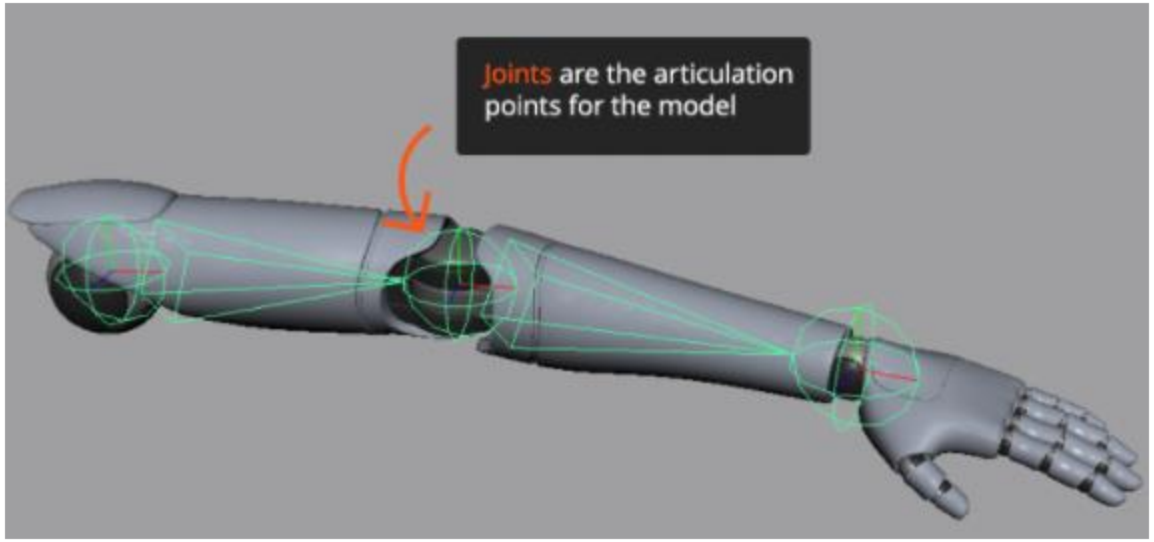
Cube

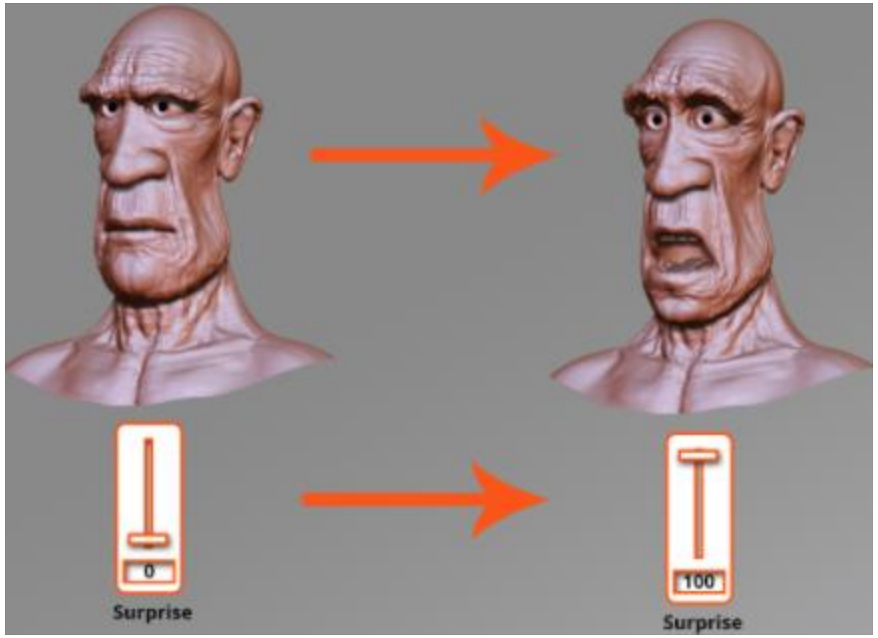


Cube man

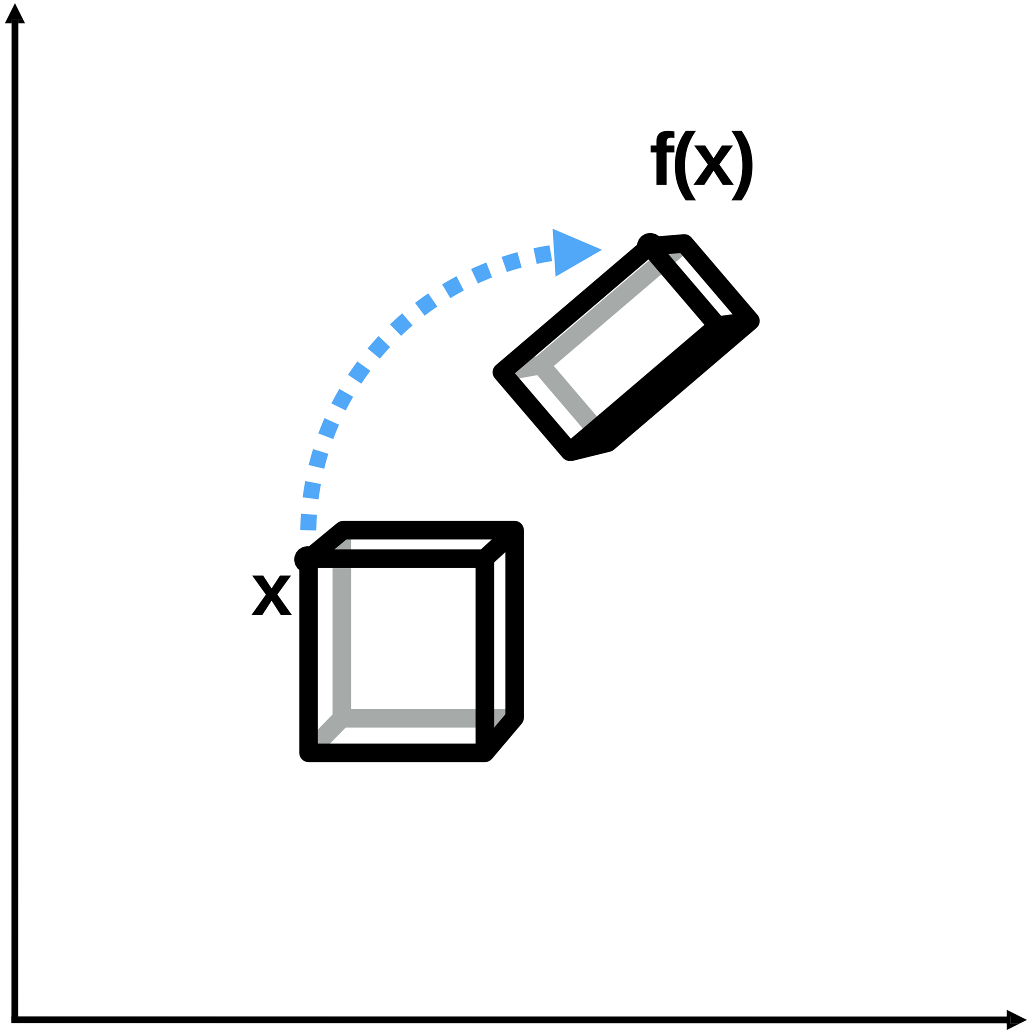

Stretched out cube moved up
The original cube
Squishy cube moved to the right
Slanty cube moved down and to the left

Transformations in Rigging





Basic idea: f transforms x to f(x)



And what is our favorite type of transformation?

What can we do with linear transformations?

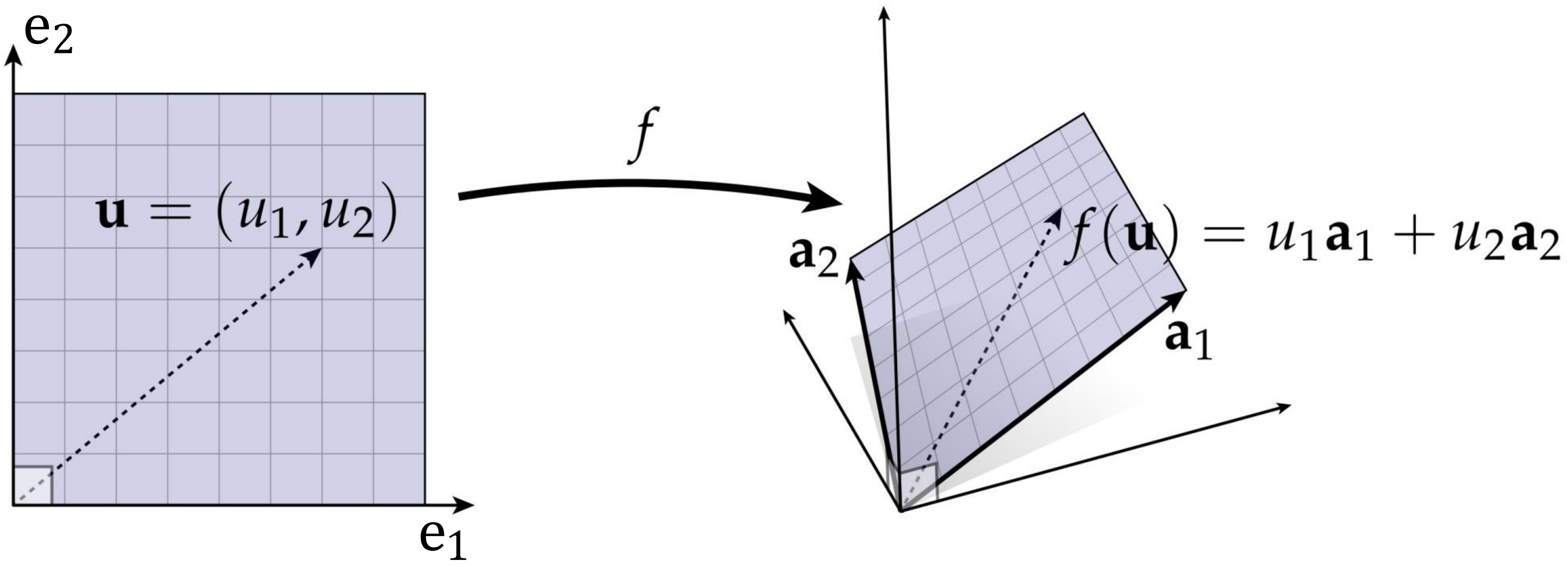
- **What did *linear* mean?**

$$f(u + v) = f(u) + f(v)$$

$$f(au) = af(u)$$

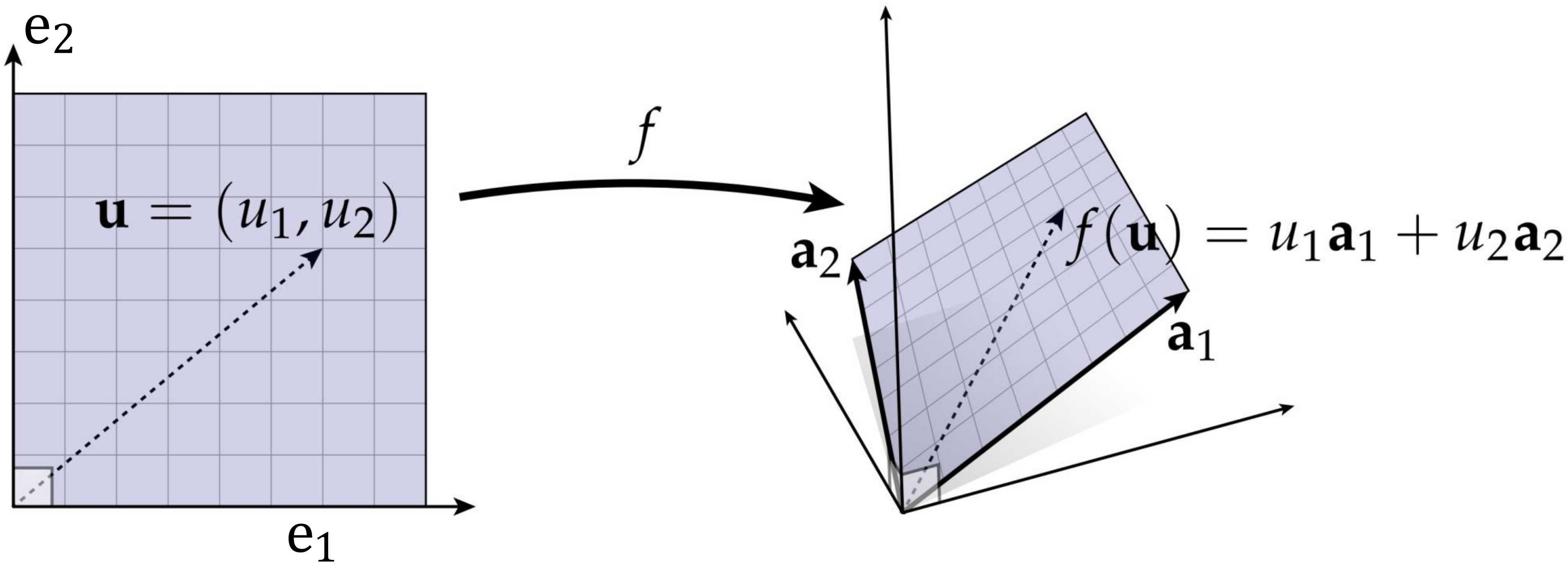
- **Cheap to compute**
- **Composition of linear transformations is linear**
  - **Leads to uniform representation of transformations**
  - **E.g., in graphics card (GPU) or graphics APIs**

Linear transforms



- **Do you know…**
  - **what u_1 and u_2 are?**
  - **what a_1 and a_2 are?**

Linear transforms



- $\boldsymbol{u}$ **is a linear combination of** e_1 **and** e_2
- $f(\boldsymbol{u})$ **is that same linear combination of** a_1 **and** a_2
- a_1 **and** a_2 **are** $f(e_1)$ **and** $\mathbf{f}(e_2)$
- **by knowing what** e_1 **and** e_2 **map to, you know how to map the entire space!**

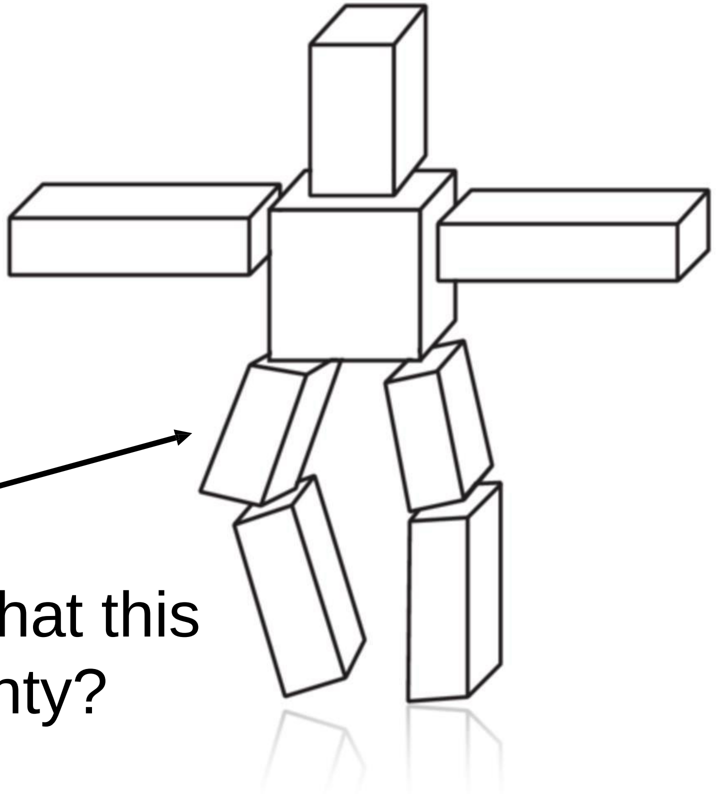
Linear transforms

If a map can be expressed as

$$\mathbf{f}(\mathbf{u}) = \sum_{i=1}^{m} u_i \mathrm{a}_i$$

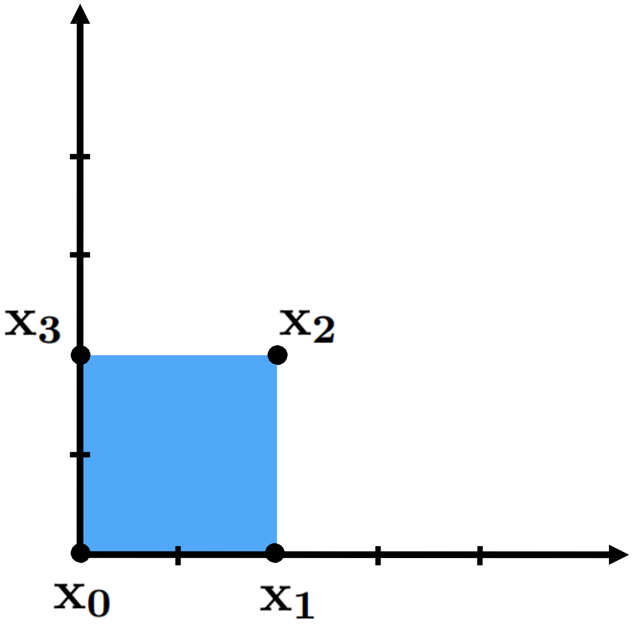
with fixed vectors a_i, then it is linear

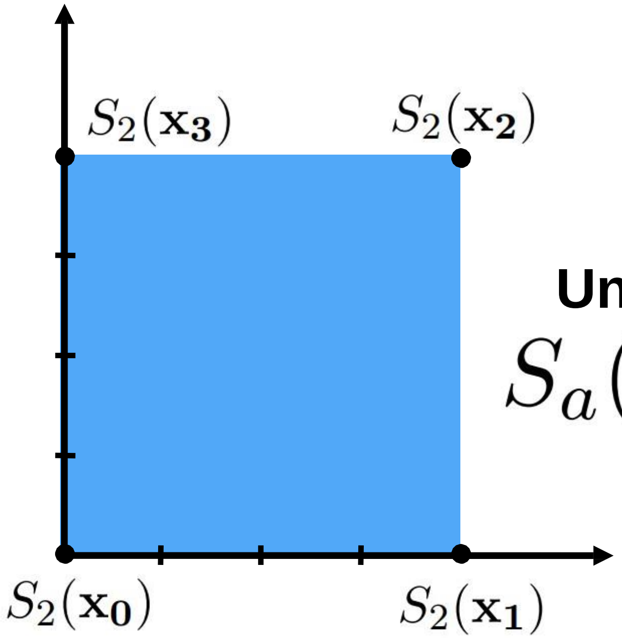
Let's look at some transforms that are important in graphics…



How do you formally tell a computer that this cube should be squished and slanty?

Scale



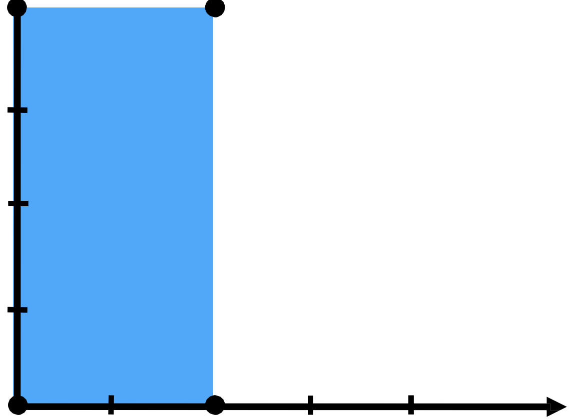


Uniform scale:

$$S_a(\mathbf{x}) = a\mathbf{x}$$

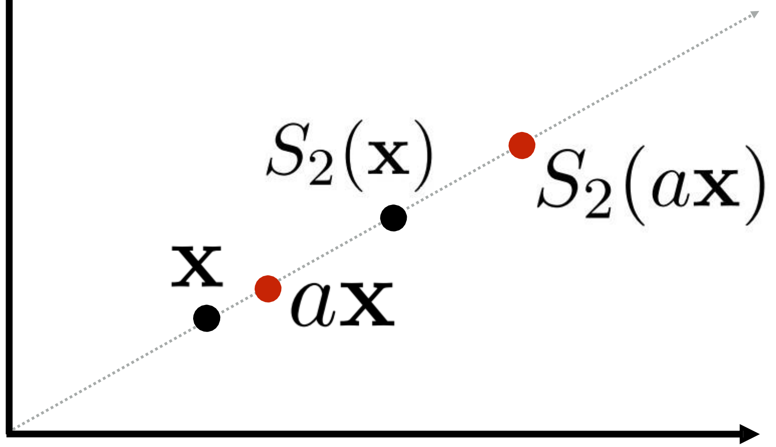
Non-uniform scale

$$S(\mathrm{x}) = \mathrm{x}_1 a \mathrm{e}_1 + \mathrm{x}_2 b \mathrm{e}_2$$

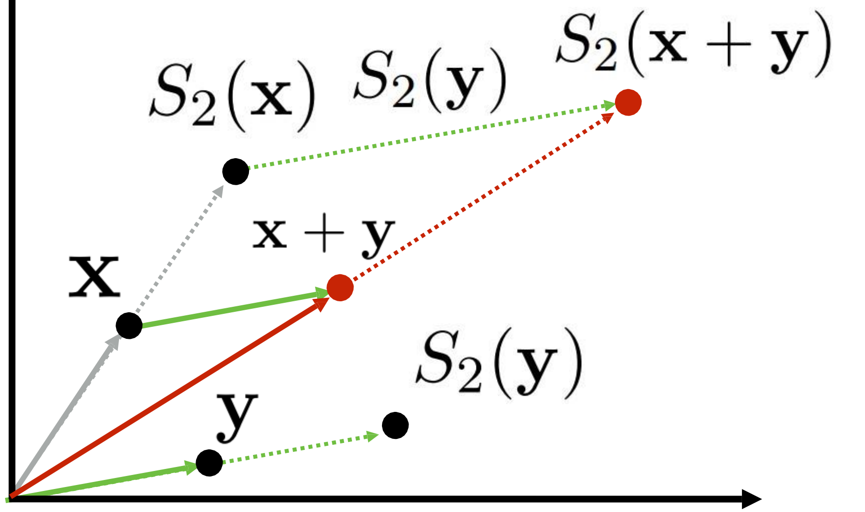


Is uniform scale a linear transform?

$$S_2(\mathbf{x}) = 2\mathbf{x}$$
$$S_2(a\mathbf{x}) = 2a\mathbf{x}$$
$$aS_2(\mathbf{x}) = 2a\mathbf{x}$$
$$S_2(a\mathbf{x}) = aS_2(\mathbf{x})$$

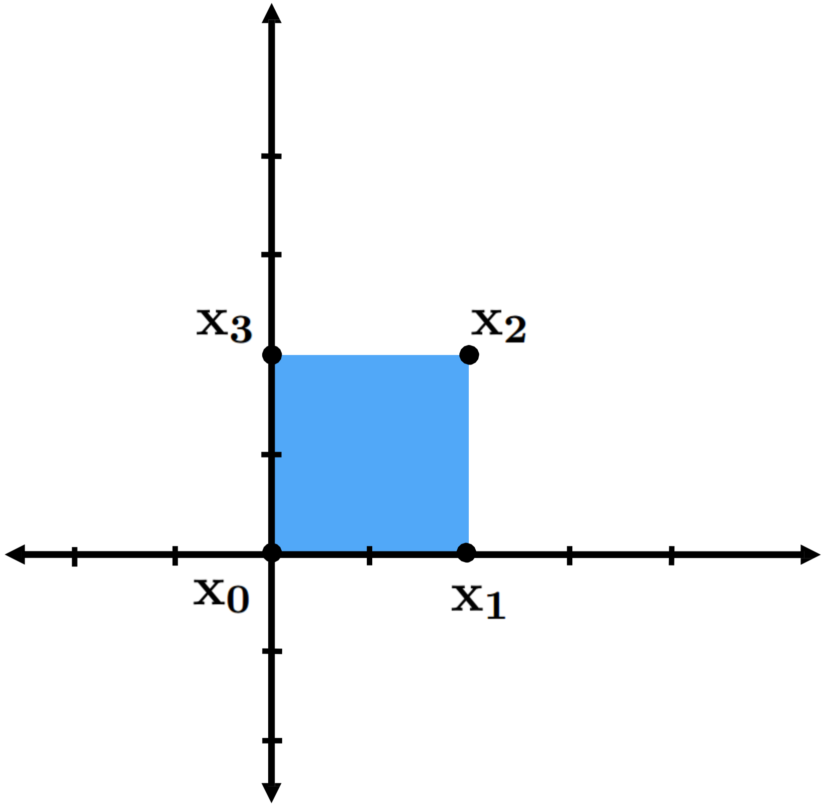


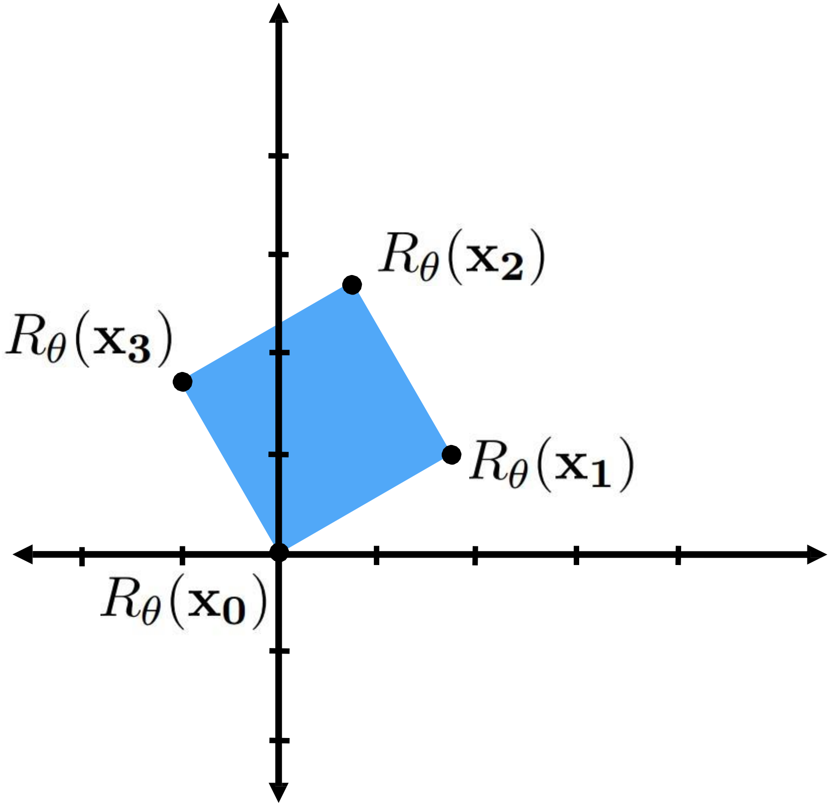
$$S_2(\mathbf{x}+\mathbf{y}) = 2(\mathbf{x}+\mathbf{y})$$
$$S_2(\mathbf{x}) + S_2(\mathbf{y}) = 2\mathbf{x} + 2\mathbf{y}$$
$$S_2(\mathbf{x}+\mathbf{y}) = S_2(\mathbf{x}) + S_2(\mathbf{y})$$



Yes!

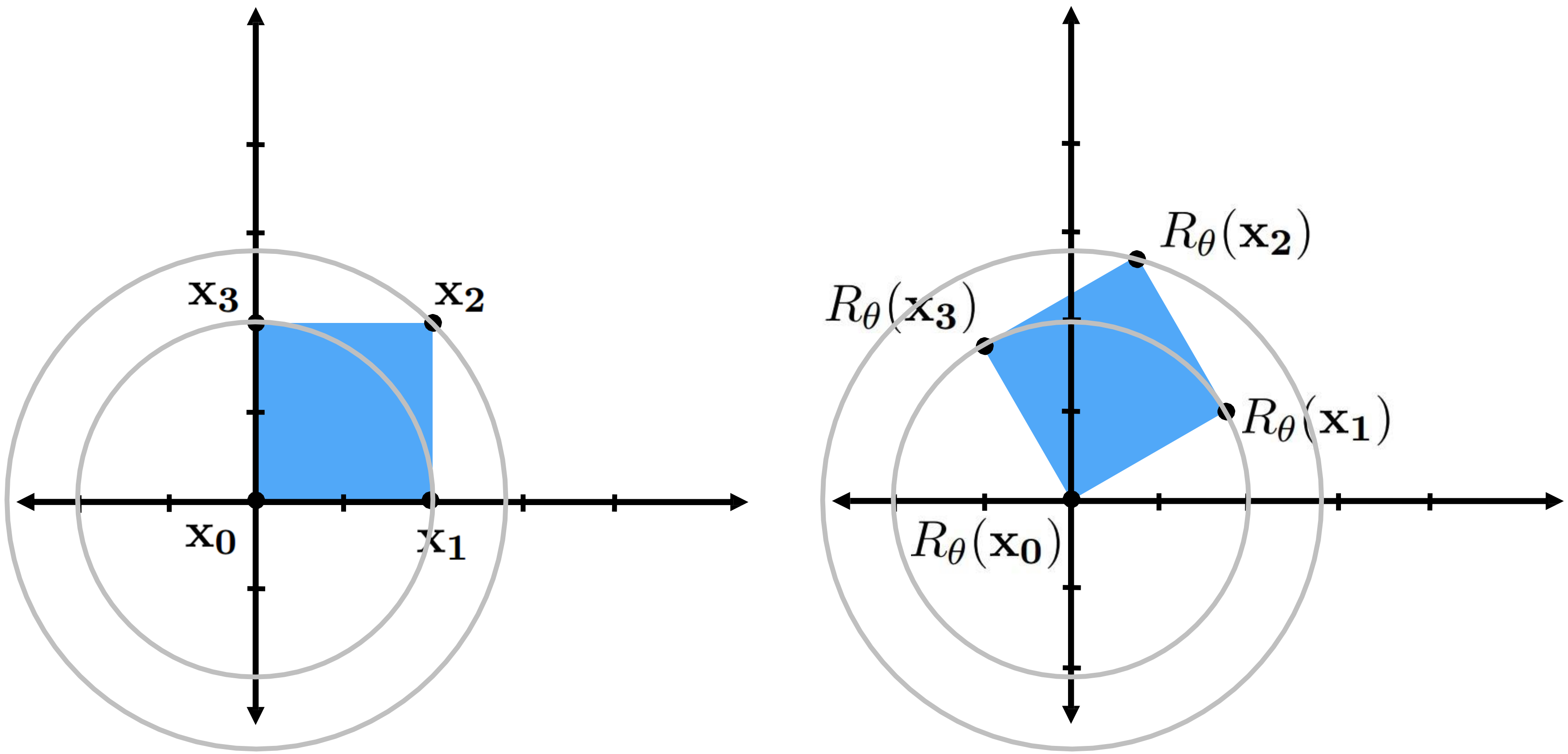
Rotation





R_θ = **rotate counter-clockwise by** θ

Rotation



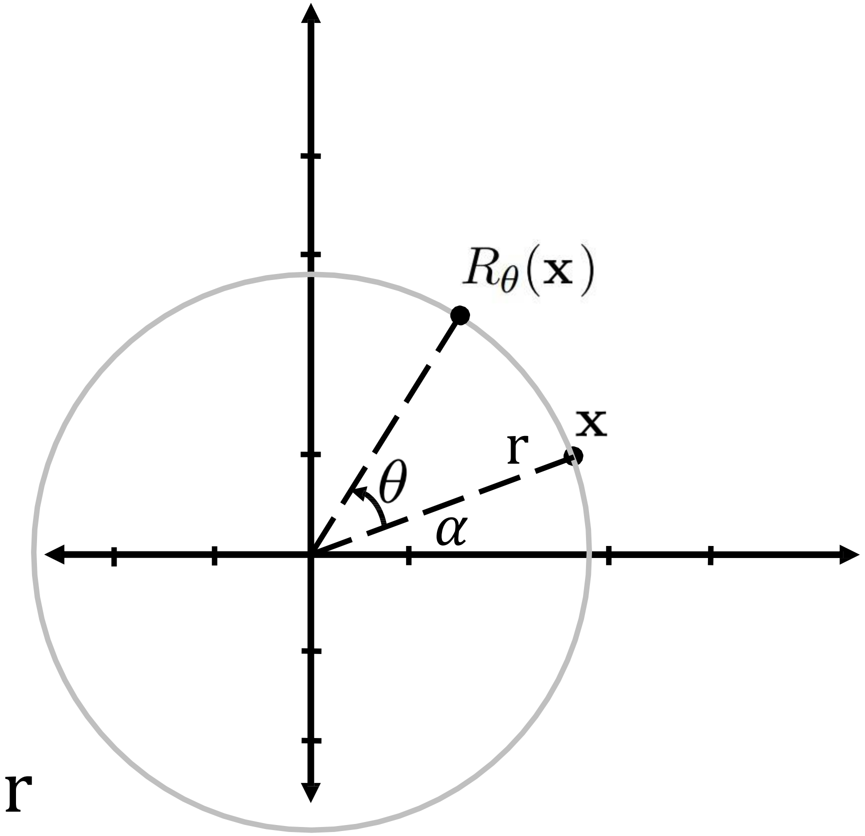
$$\mathrm{R}_{\boldsymbol{\theta}} = \boldsymbol{rotate\ counter - clockwise\ by}\ \theta$$

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

Hence, rotations preserve length of vectors: $|\mathrm{R}_{\boldsymbol{\theta}}(\boldsymbol{x})| = |\boldsymbol{x}|$

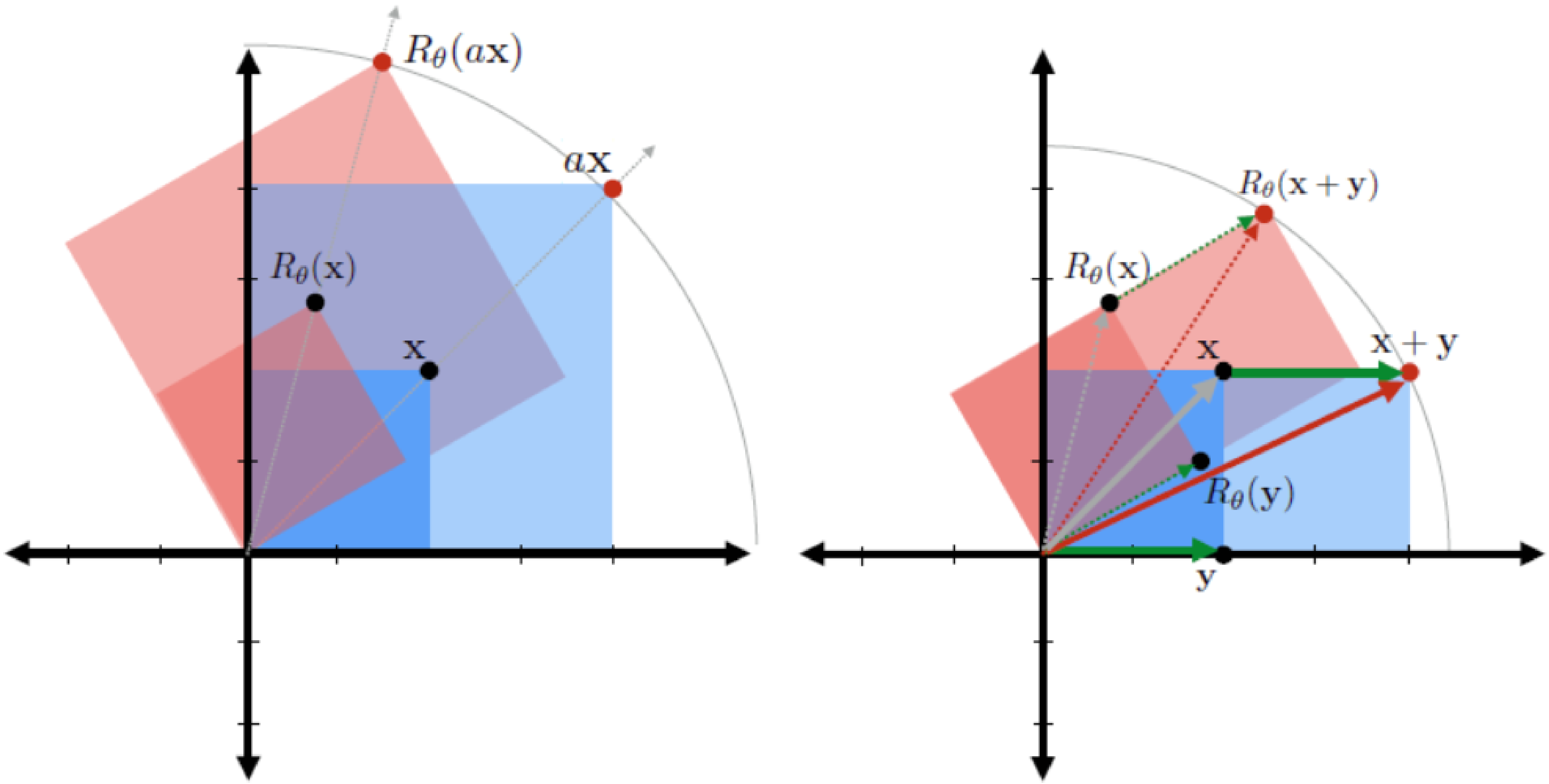
Rotation

What does R_θ look like?



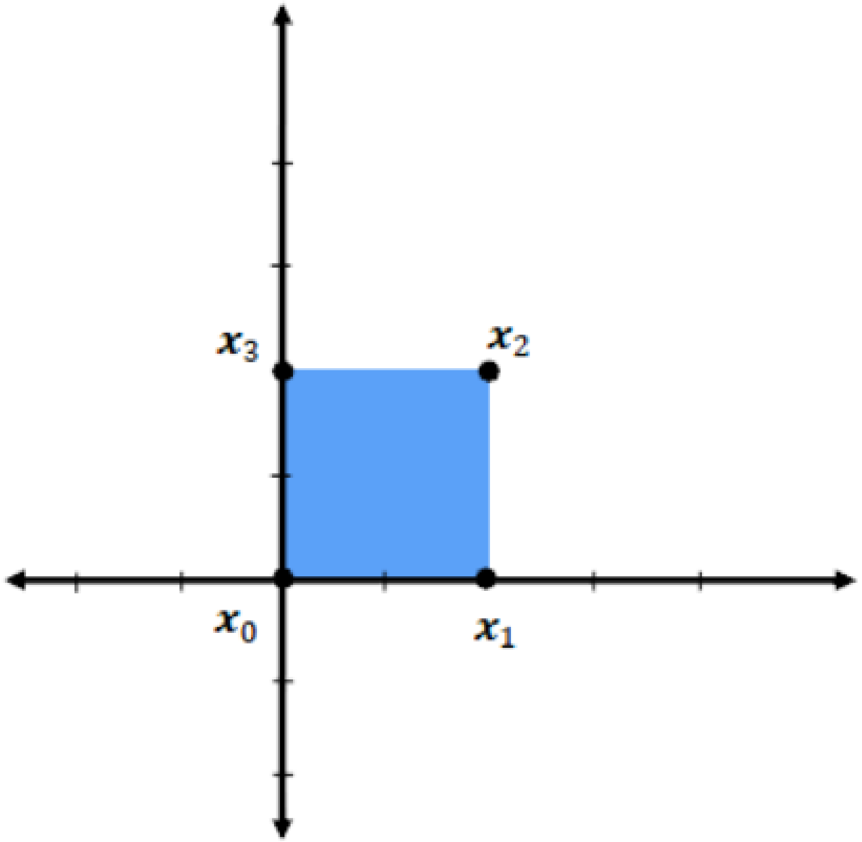
- **From x, compute α and r**
- **Write down $R_\theta(\mathrm{x})$ as a function of α, θ and r**
  (i.e. vector $(\mathrm{r},0)$ rotated by $\alpha + \theta$)
- **Apply sum of angle formulae…**
- **Fine, but remember, we only need to know how e_1 and e_2 are transformed!**

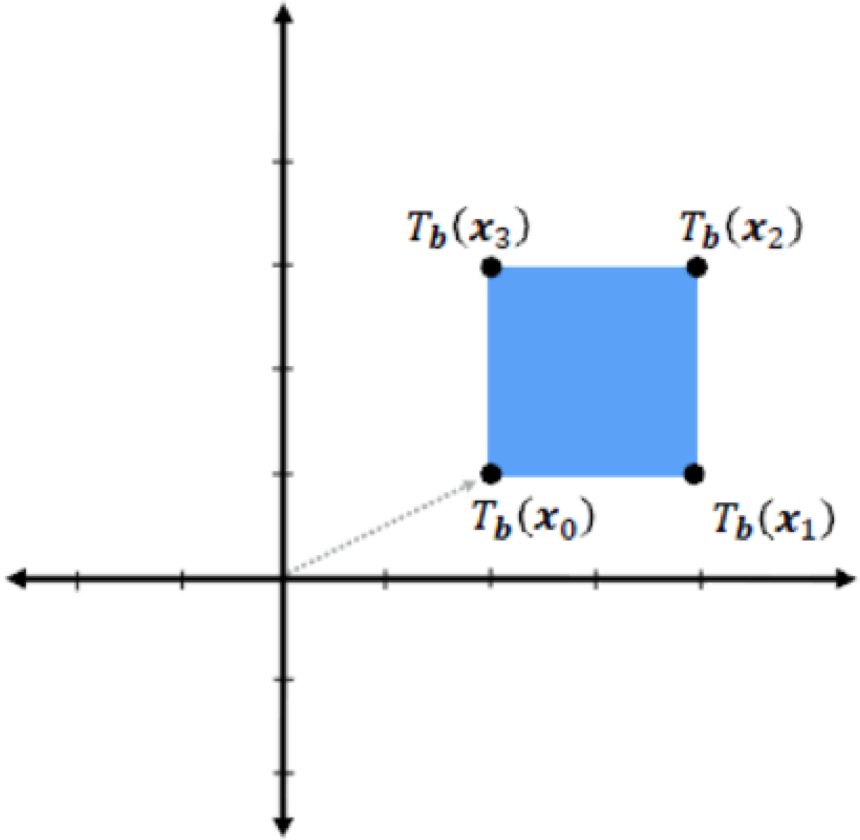
Is rotation linear?



- Yes

Translation



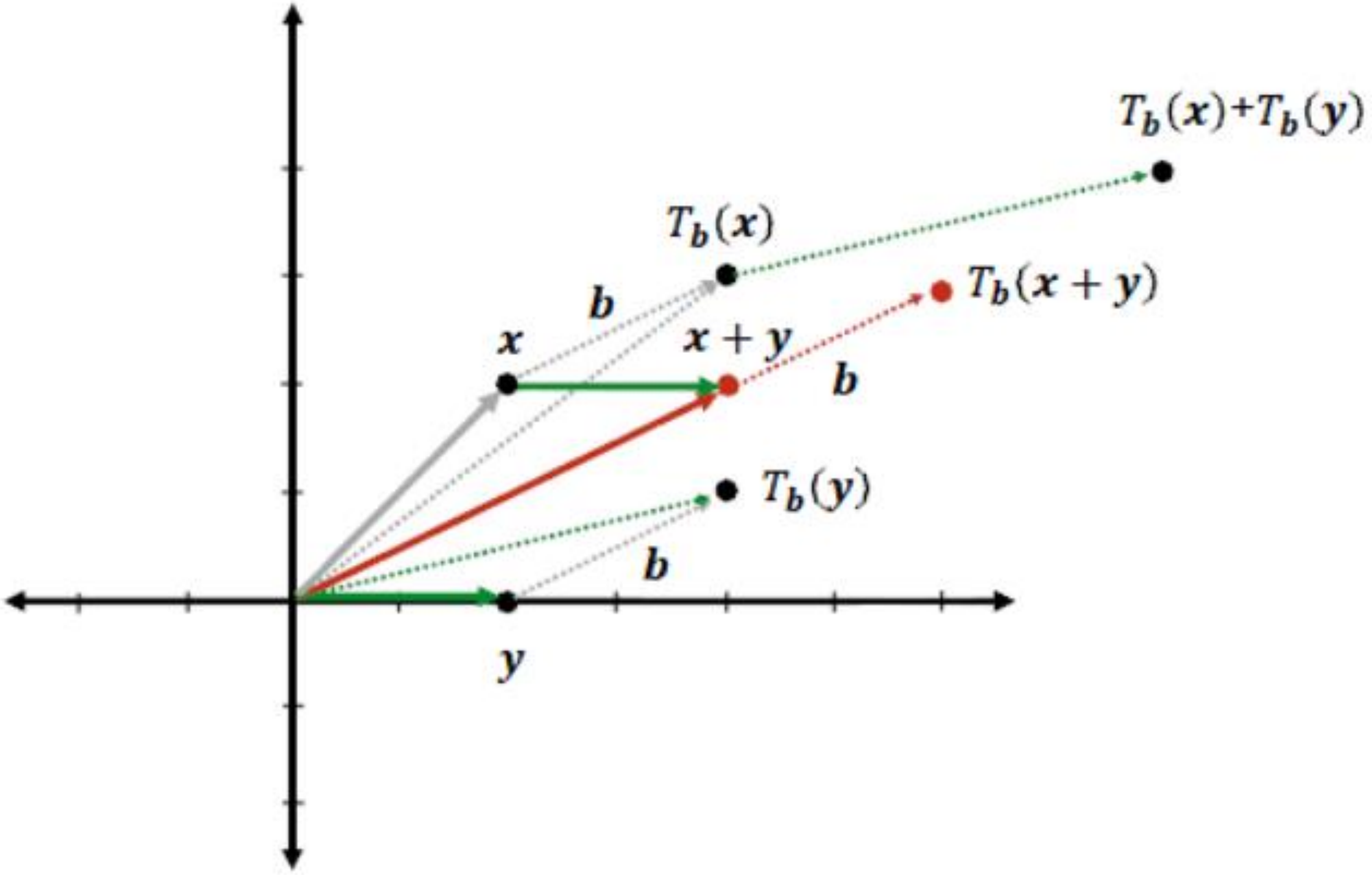


Let's write $T_b(\boldsymbol{x})$ in the form

$$T_{\boldsymbol{b}}(\boldsymbol{x}) = x_1 \begin{bmatrix} ? \\ ? \end{bmatrix} + x_2 \begin{bmatrix} ? \\ ? \end{bmatrix}$$

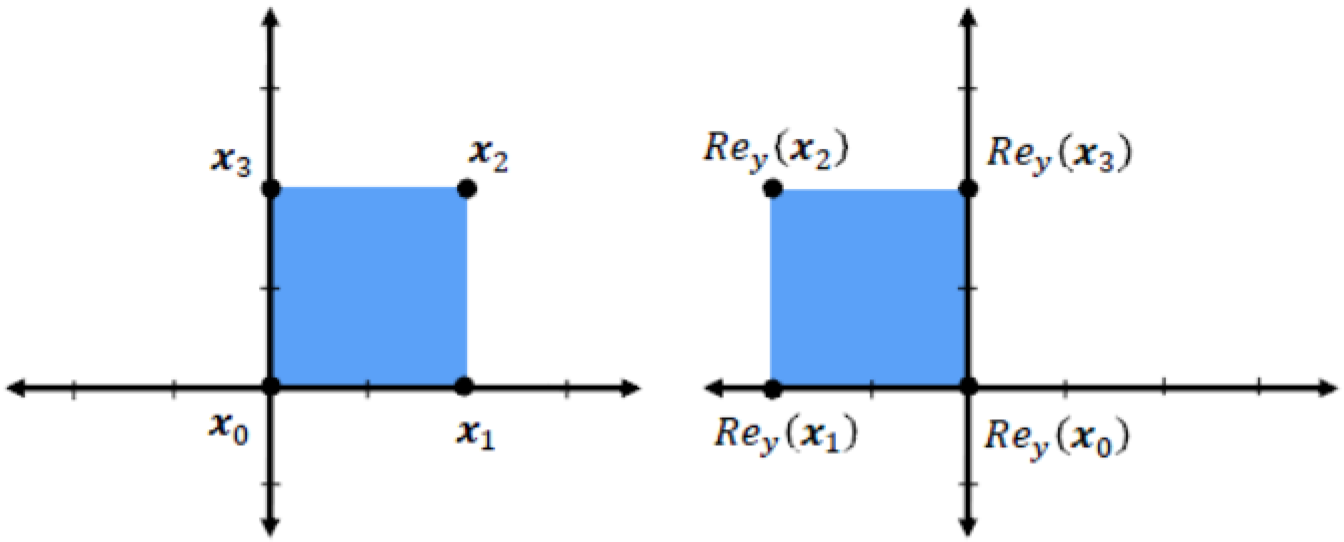
such that $T_{\boldsymbol{b}}(\boldsymbol{x}) = \boldsymbol{x} + \boldsymbol{b}$

Is translation linear?



No. Translation is affine.

Reflection



$Re_y(\boldsymbol{x})$: reflection about y-axis

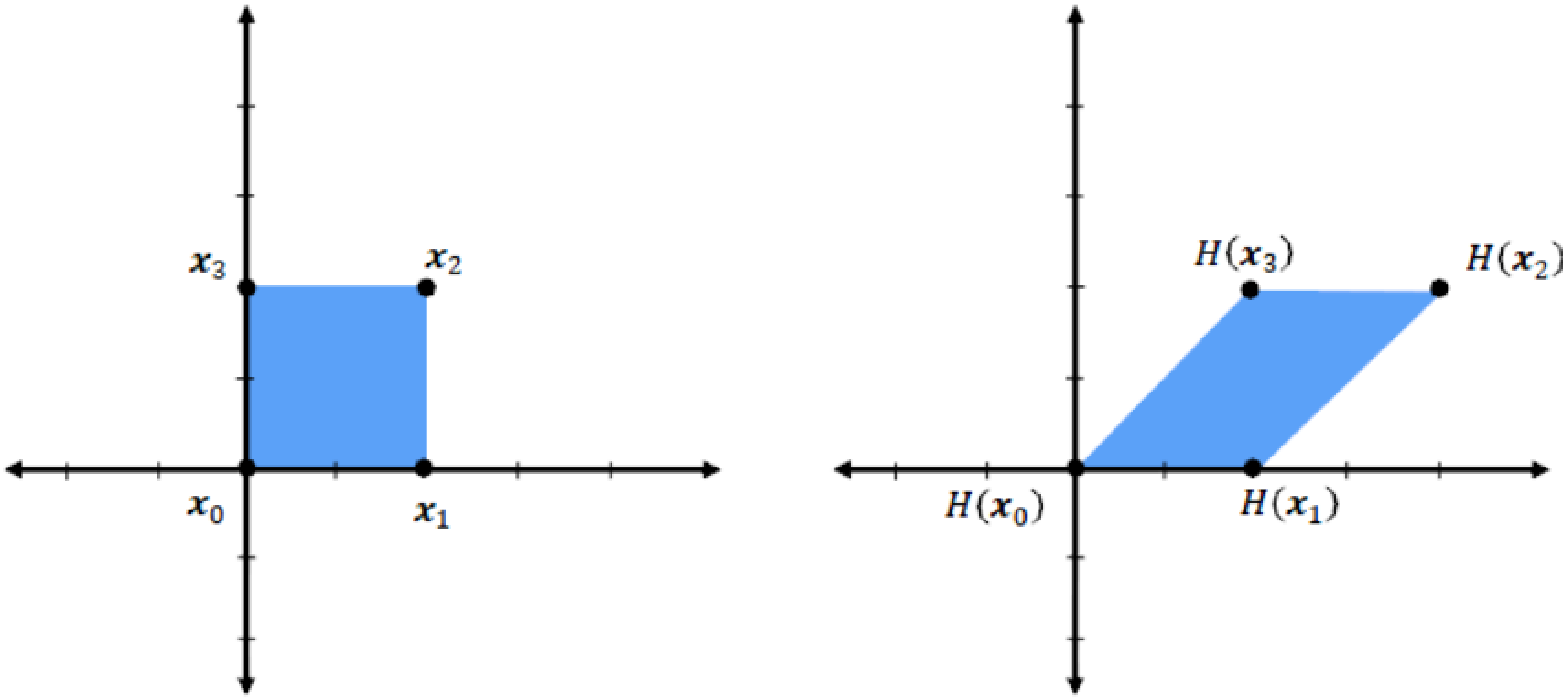
Reflections change "handedness"...

Do you know what $Re_y(\boldsymbol{x})$ looks like?

Is reflection a linear transform?

Do you know how to reflect about an arbitrary axis?

Shear (in x direction)

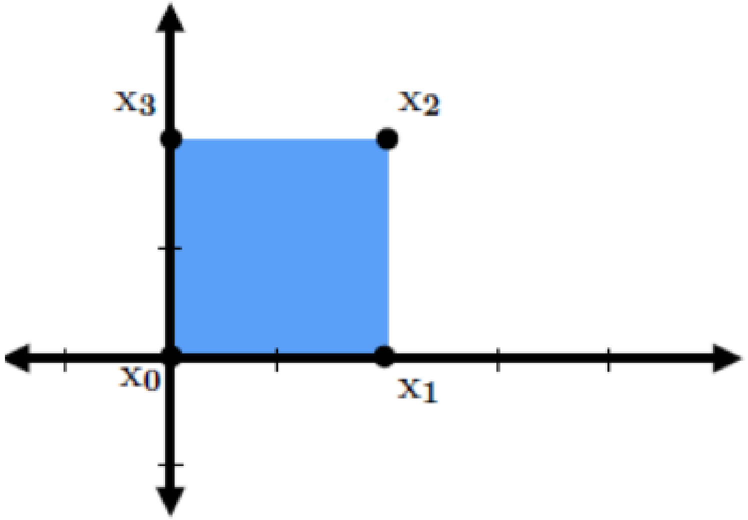


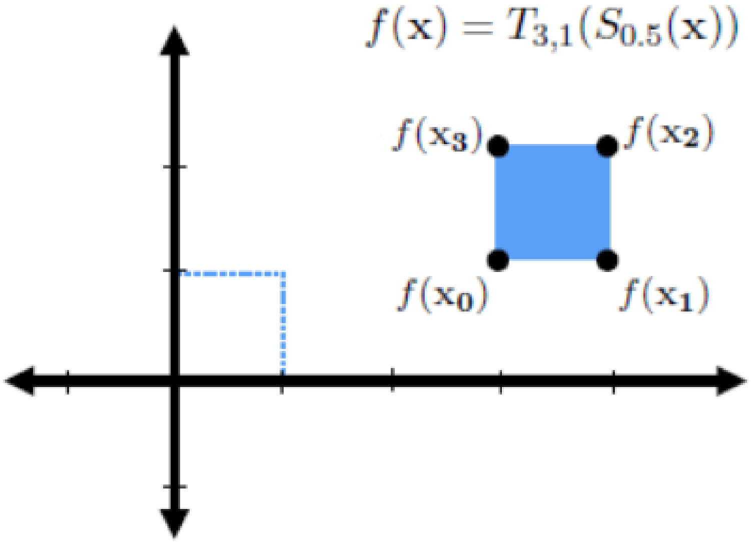
What does $H(x)$ look like?

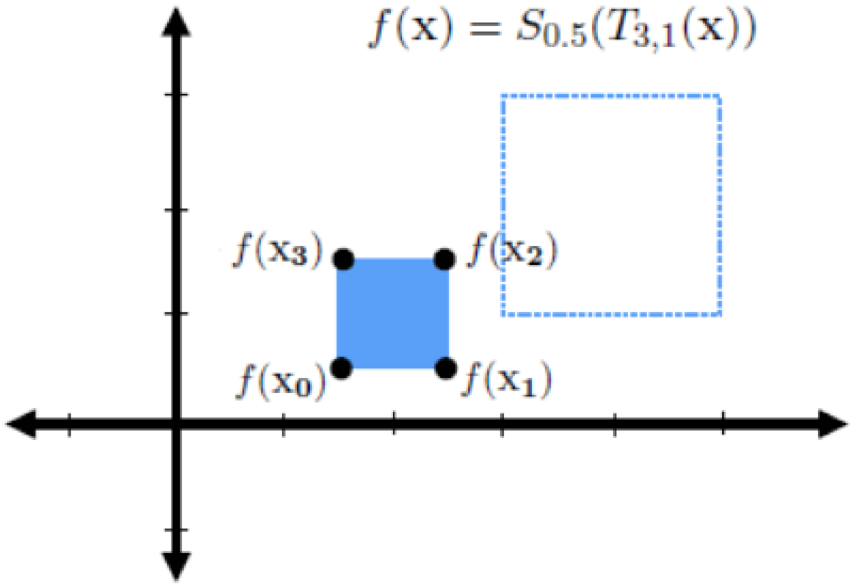
$$H_a(x) = x_1 \begin{bmatrix} 1 \\ 0 \end{bmatrix} + x_2 \begin{bmatrix} a \\ 1 \end{bmatrix}$$

Is shearing a linear transformation?

Compose basic transformations to construct more complicated ones



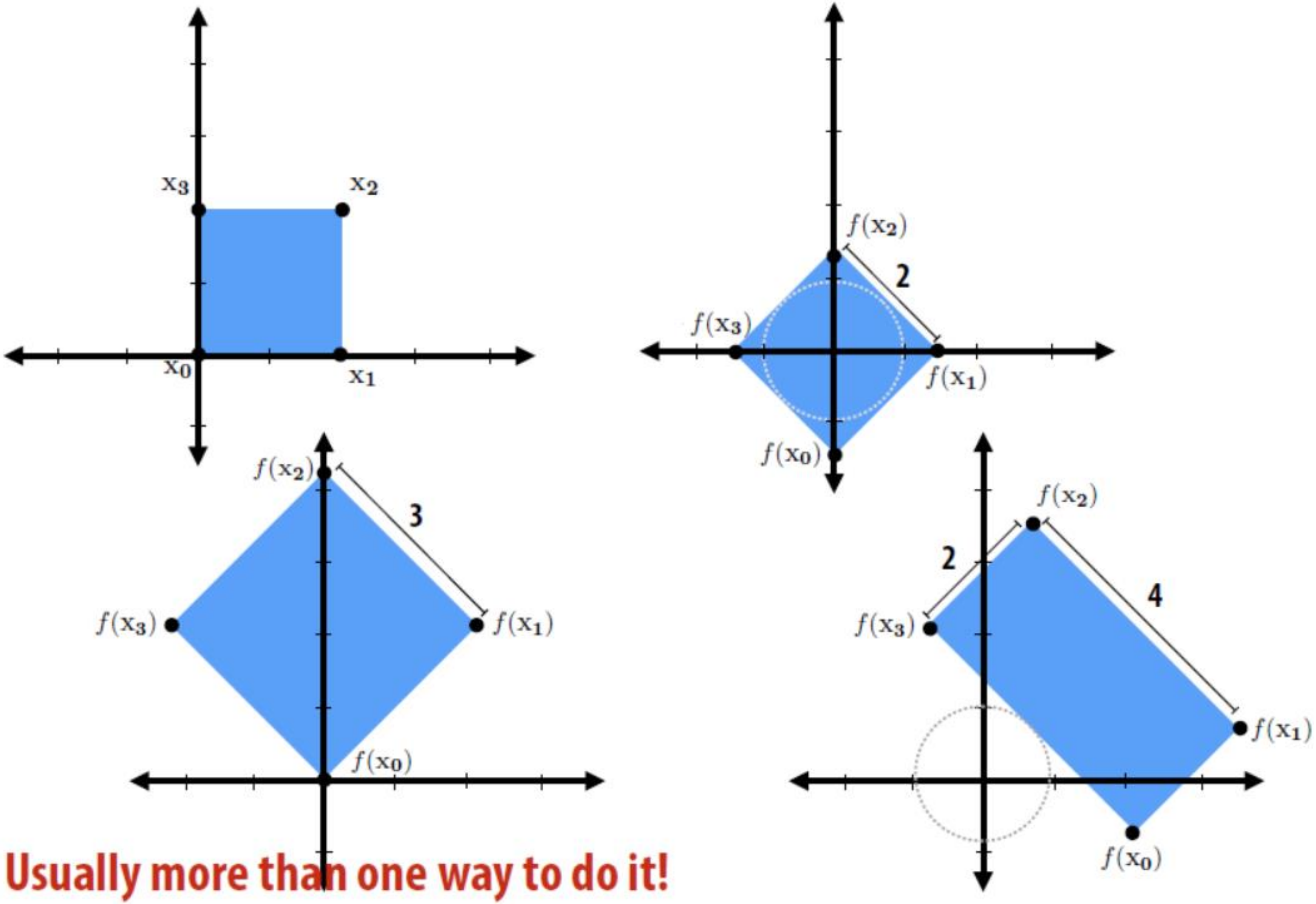




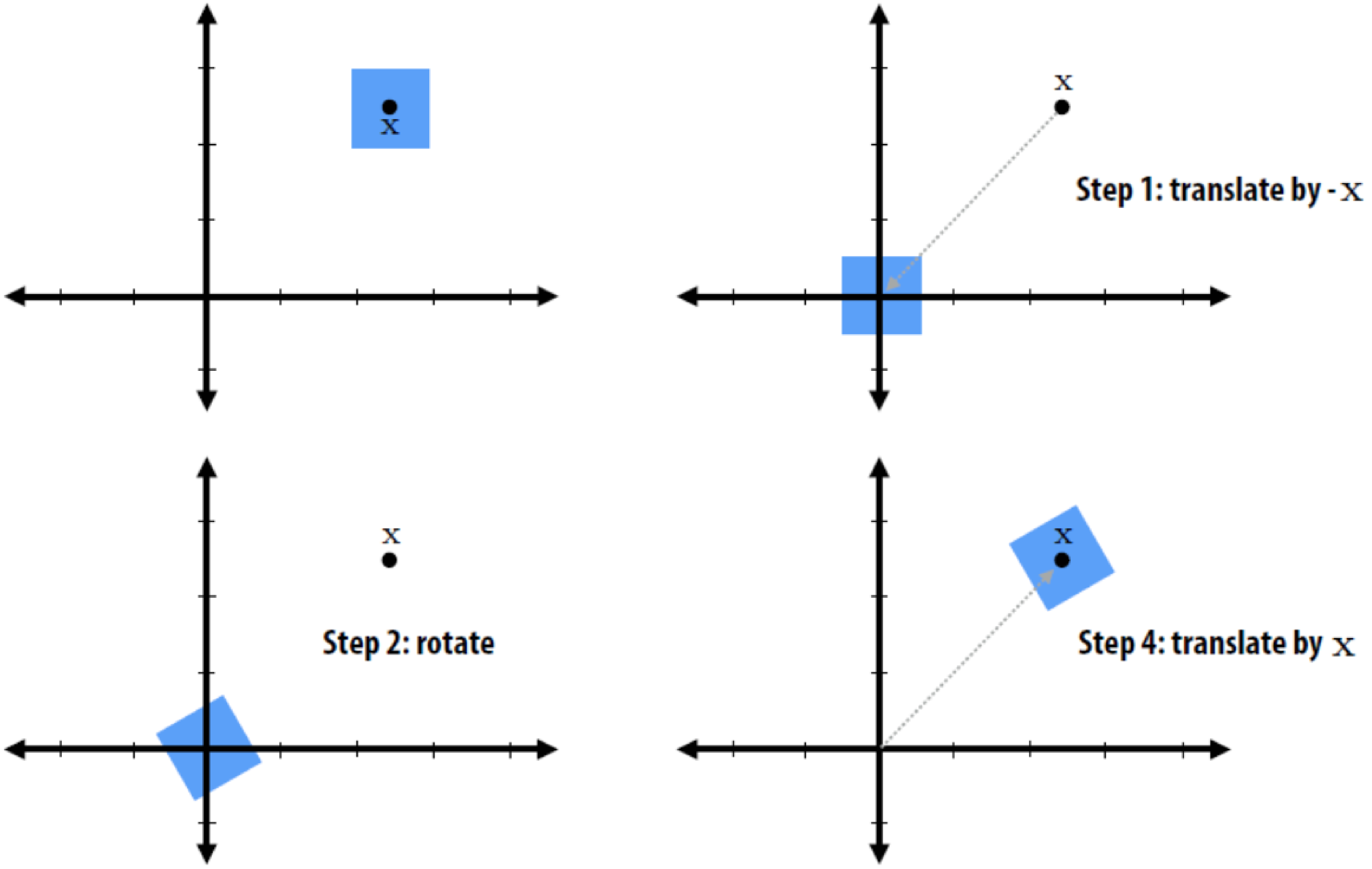
Note: order of composition matters

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

How would you perform these transformations?



Common task: rotate about a point x



Summary of basic transforms

Linear:

$$f(\mathbf{x} + \mathbf{y}) = f(\mathbf{x}) + f(\mathbf{y})$$
$$f(a\mathbf{x}) = af(\mathbf{x})$$

Scale
Rotation
Reflection
Shear

Not linear:

Translation

Affine:

Composition of linear transform + translation
(all examples on previous two slides)

$$f(\mathbf{x}) = g(\mathbf{x}) + \mathbf{b}$$

Not affine: perspective projection (will discuss later)

Euclidean: (Isometries)

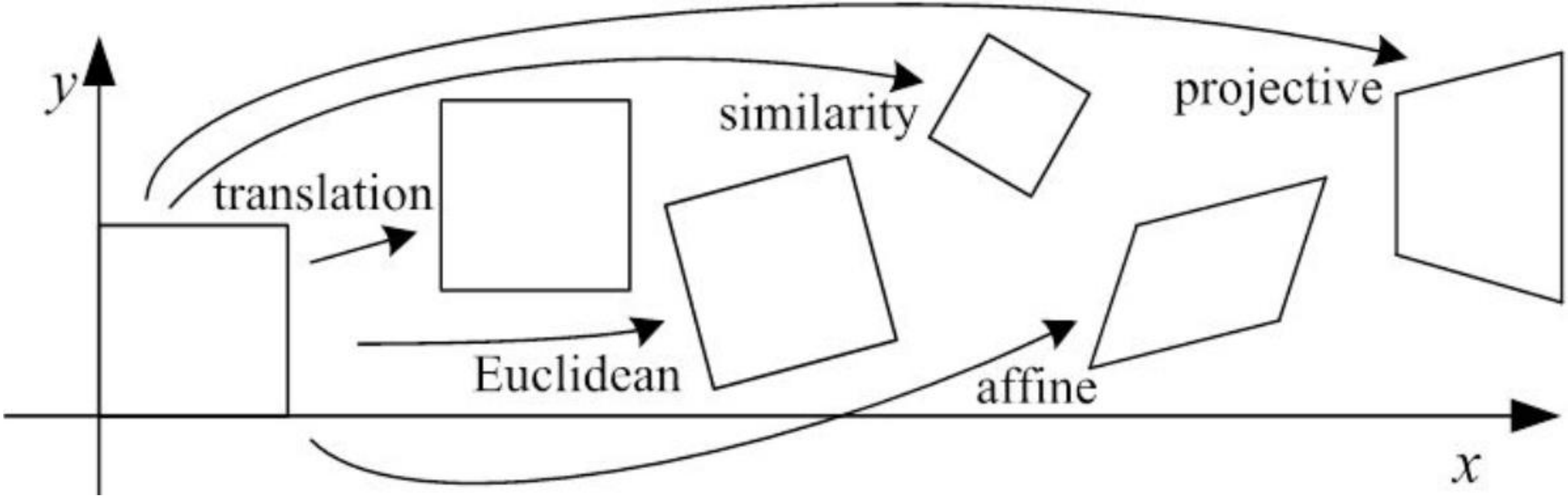
Preserve distance between points (preserves length)

$$|f(\mathbf{x}) - f(\mathbf{y})| = |\mathbf{x} - \mathbf{y}|$$

Translation
Rotation
Reflection

"Rigid body" transforms are Euclidean transforms that also preserve "winding" (does not include reflection)

2D Geometric Transformations



Representing Transformations in Coordinates

Review: representing points in a coordinate space

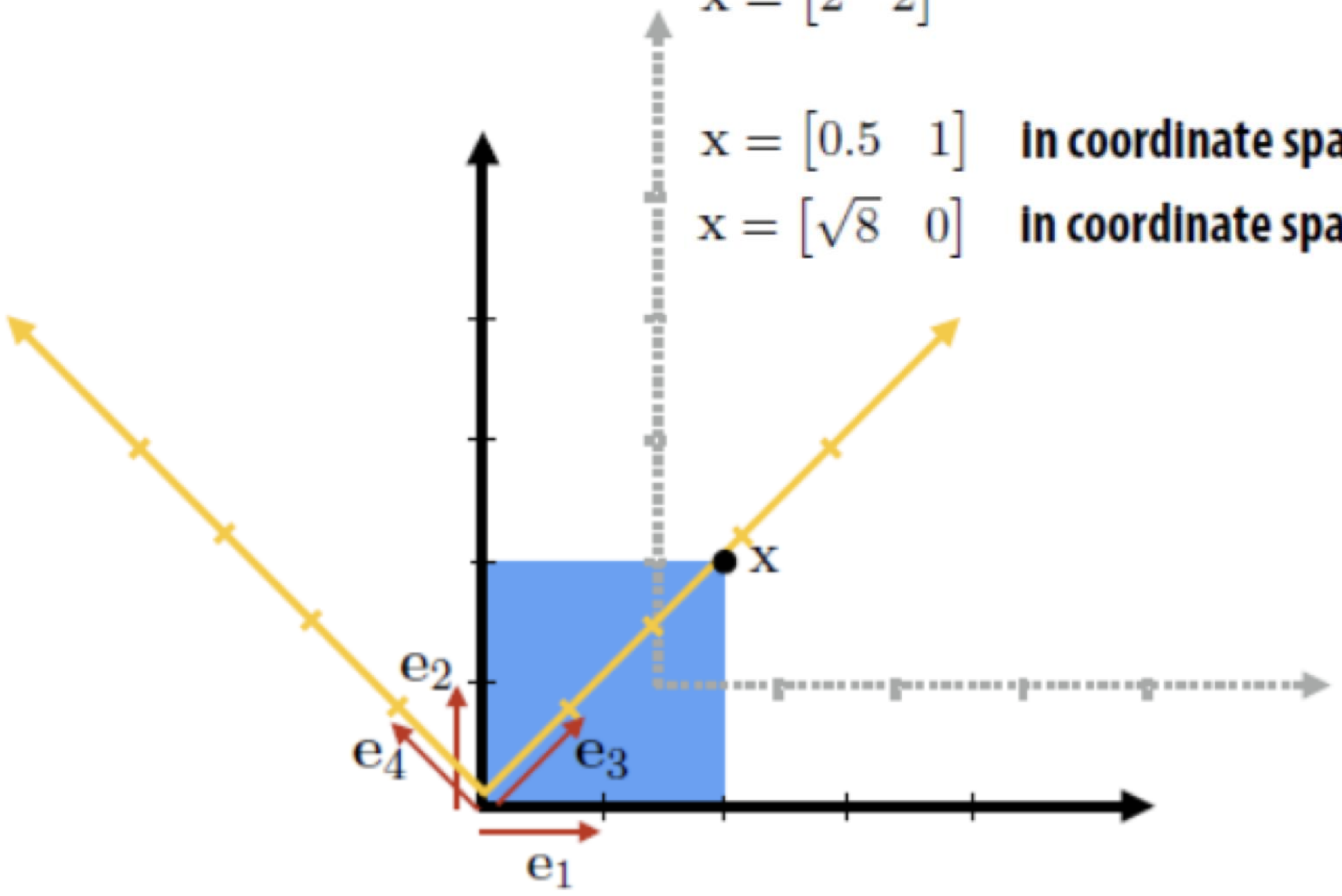
Consider coordinate space defined by orthogonal vectors e_1 and e_2

$$\mathrm{x} = 2\mathrm{e}_1 + 2\mathrm{e}_2$$

$$\mathrm{x} = \begin{bmatrix} 2 & 2 \end{bmatrix}$$

$\mathrm{x} = \begin{bmatrix} 0.5 & 1 \end{bmatrix}$ in coordinate space defined by e_1 and e_2, with origin at (1.5, 1)

$\mathrm{x} = \begin{bmatrix} \sqrt{8} & 0 \end{bmatrix}$ in coordinate space defined by e_3 and e_4, with origin at (0, 0)



Review: matrix multiplication

$$\underbrace{\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}}_{\boldsymbol{A}} \overset{*}{} \underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{\boldsymbol{x}} = \begin{bmatrix} a_{11}x_1 + a_{12}x_2 \\ a_{21}x_1 + a_{22}x_2 \end{bmatrix}$$

$$= x_1 \begin{bmatrix} a_{11} \\ a_{21} \end{bmatrix} + x_2 \begin{bmatrix} a_{12} \\ a_{22} \end{bmatrix} = \underbrace{x_1 \boldsymbol{a_1} + x_2 \boldsymbol{a_2}}$$

$$f(\boldsymbol{x}) = \sum_{i=1}^{m} x_i \boldsymbol{a_i} = \boldsymbol{A}\boldsymbol{x}$$

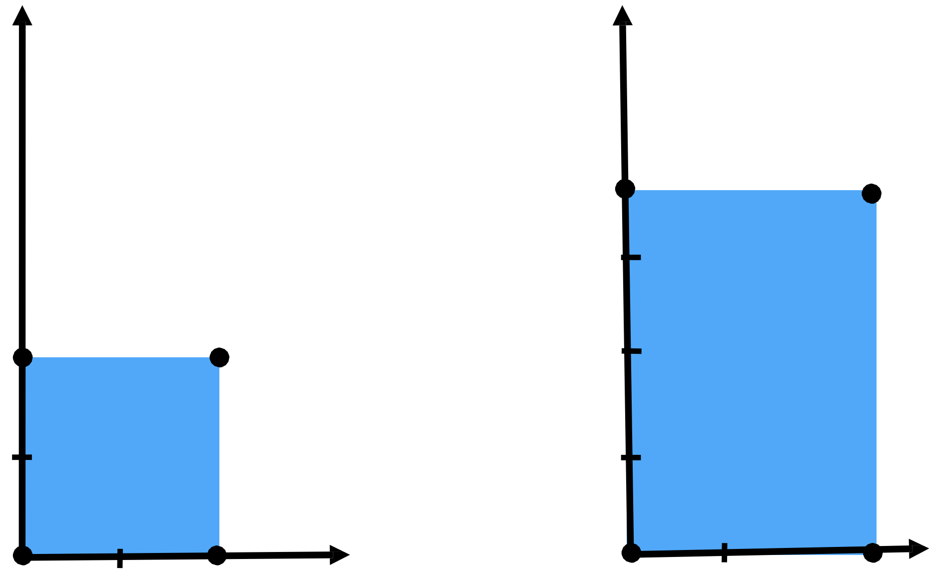
- **Matrix multiplication is linear combination of columns**
- **Encodes a linear map!**

Linear transforms in 2D is 2*2 matrices

Non-uniform scale

$$S(\boldsymbol{x}) = x_1 a\boldsymbol{e}_1 + x_2 b\boldsymbol{e}_2$$

$$= \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} \boldsymbol{x}$$



Rotation

So, what happens to vectors (1, 0) and (0, 1) after rotation by θ?

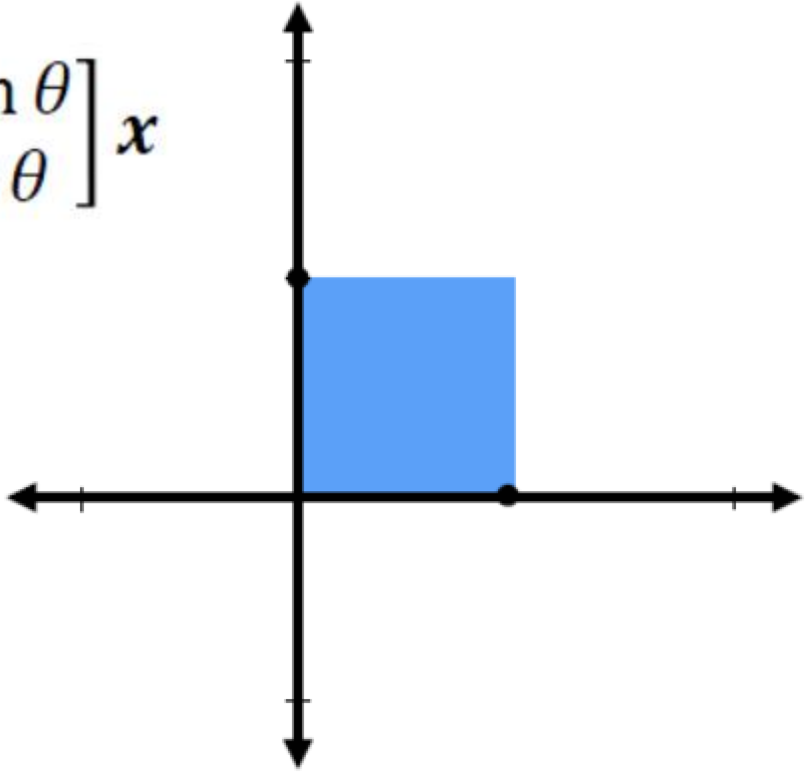
Rotation

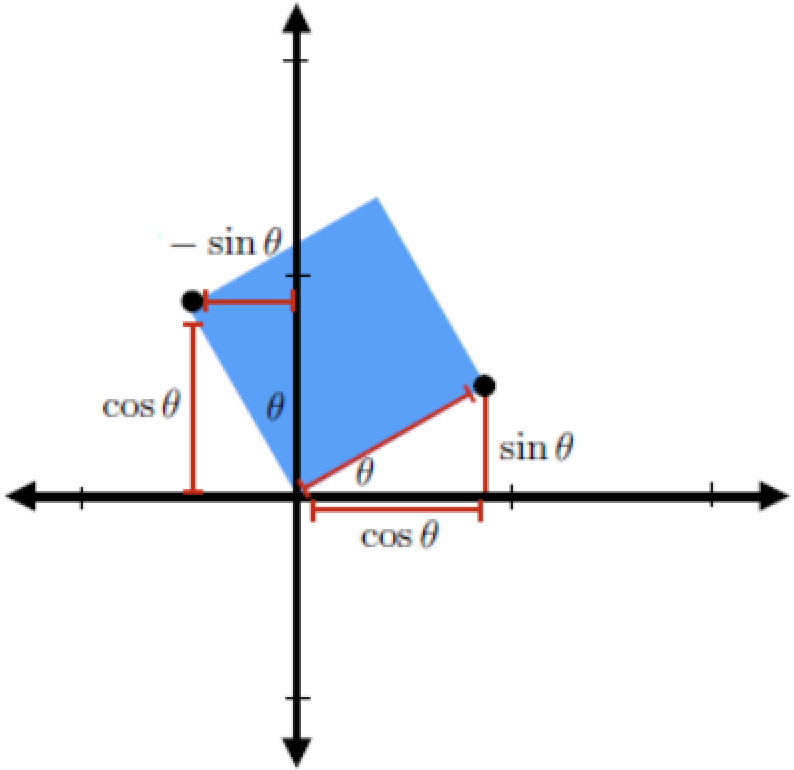
$$R_\theta(\boldsymbol{e}_1) = (cos\ \theta, sin\ \theta) = \boldsymbol{a}_1$$

$$R_\theta(\boldsymbol{e}_2) = (-sin\ \theta, cos\ \theta) = \boldsymbol{a}_2$$

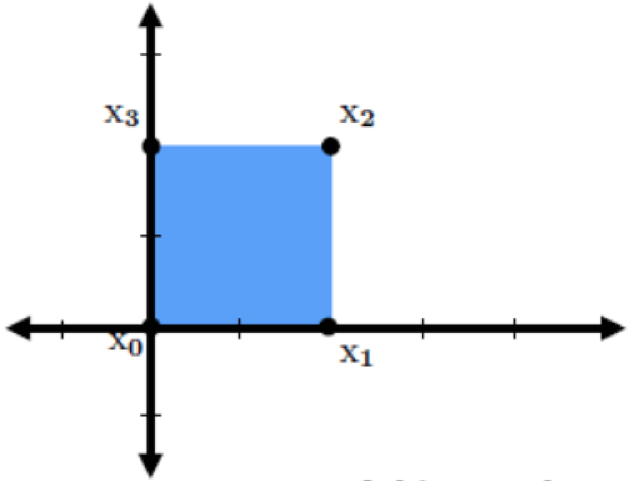
$$R_\theta(\boldsymbol{x}) = x_1\boldsymbol{a}_1 + x_2\boldsymbol{a}_2$$

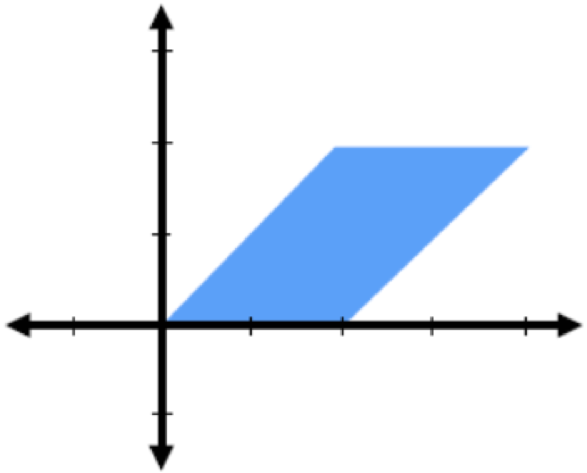
$$= \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \boldsymbol{x}$$





Shear



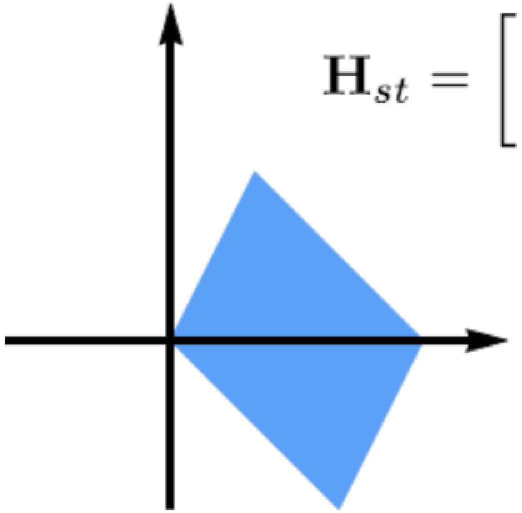


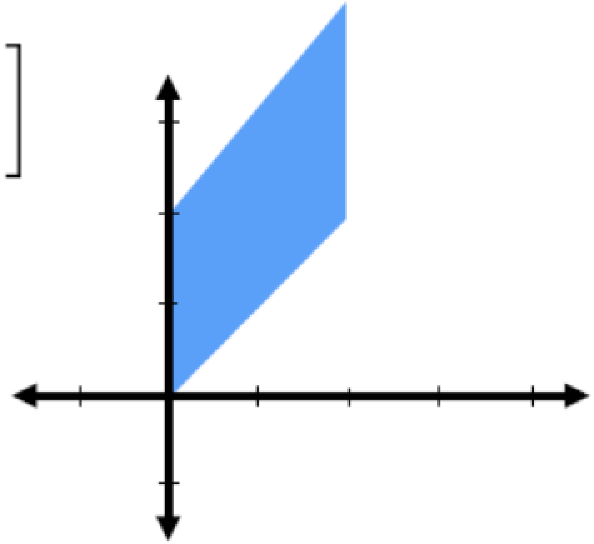
Shear in x:

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

Arbitrary shear:

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





Shear in y:

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

General shear mappings

if V is the direct sum of W :and W', and we write vectors as: v=w+w'

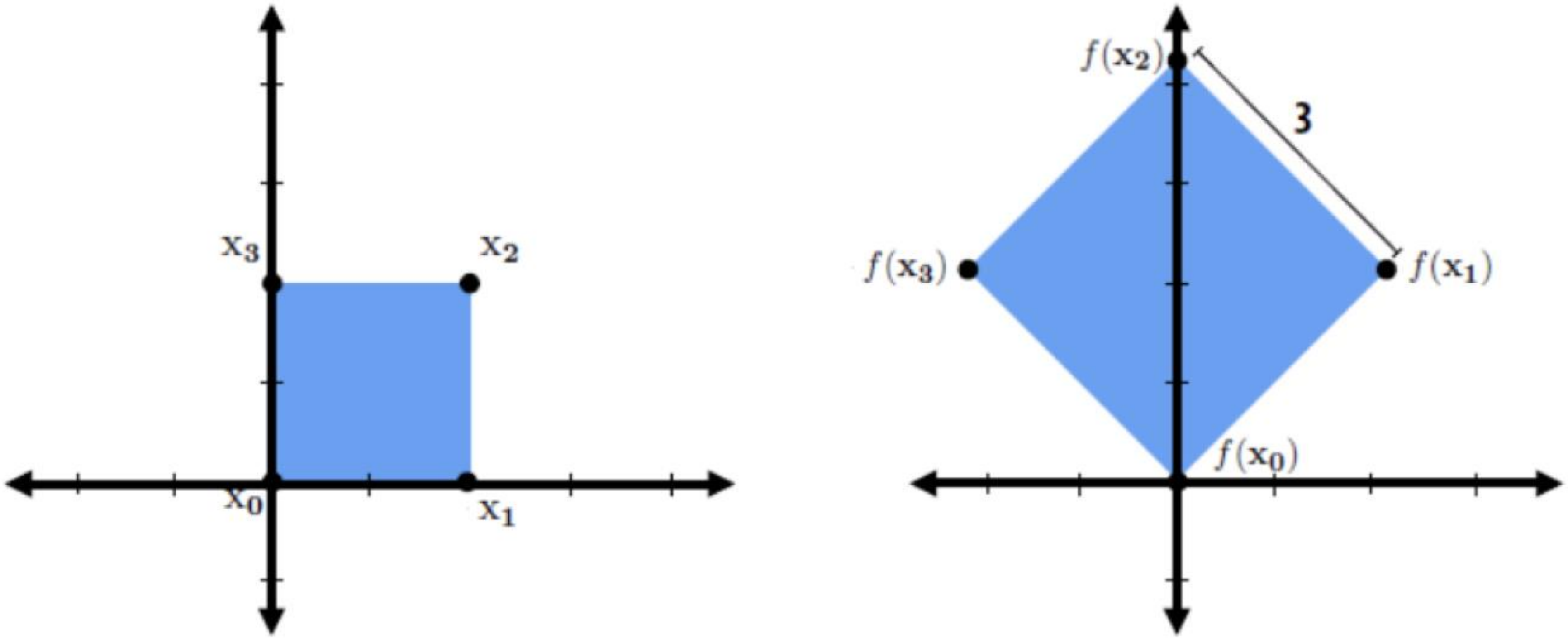
***L*(*v*) = (*w* + *Mw'*) + *w* '**

where M is a linear mapping from W' into W.

In block matrix terms L can be represented as:

$$\begin{bmatrix} \boldsymbol{I} & \boldsymbol{M} \\ \mathbf{0} & \boldsymbol{I} \end{bmatrix}$$

How do we compose linear transformations?



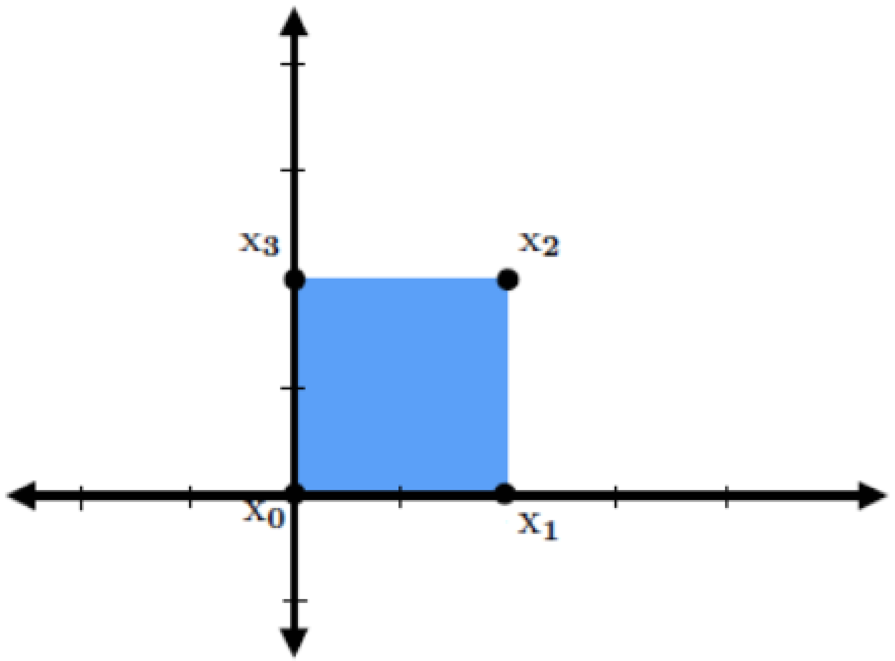
Compose linear transformations via matrix multiplication.
This example: uniform scale, followed by rotation

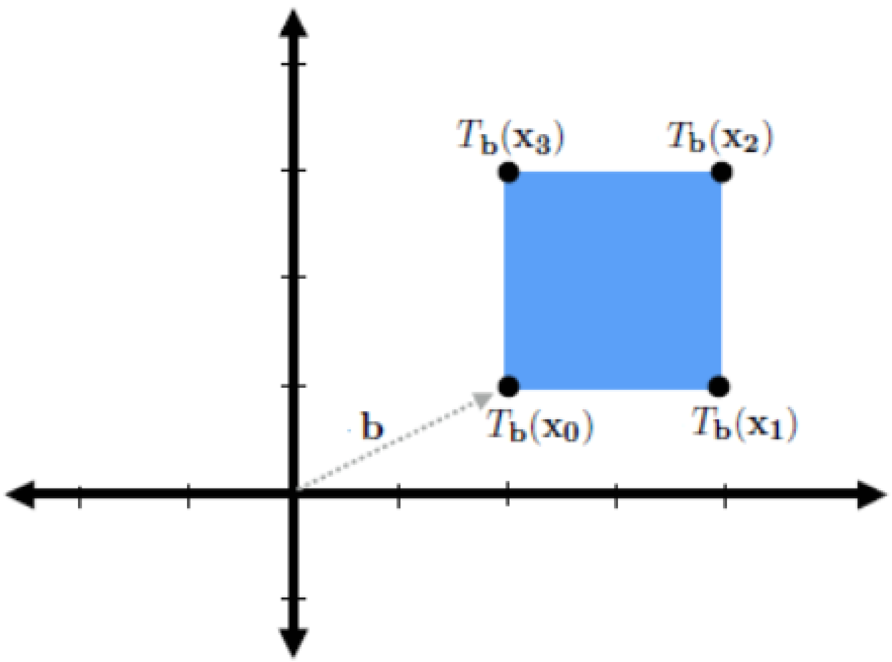
$$f(\mathbf{x}) = R_{\pi/4}\mathbf{S}_{[1.5,1.5]}\mathbf{x}$$

Enables simple, efficient implementation: reduce complex chain of transformations to a single matrix multiplication

How do we deal with translation? (Not linear)

$$T_{\mathbf{b}}(\mathbf{x}) = \mathbf{x} + \mathbf{b}$$





Recall: translation is not a linear transform

→ Output coefficients are not a linear combination of input coefficients

→ Translation operation cannot be represented by a 2x2 matrix

$$\mathbf{x}_{\text{out}_x} = \mathbf{x}_x + \mathbf{b}_x$$

$$\mathbf{x}_{\text{out}_y} = \mathbf{x}_y + \mathbf{b}_y$$

Translation math

2D homogeneous coordinates (2D-H)

Key idea: lift 2D points to a 3D space

So the point $(\mathrm{x}_1, \mathrm{x}_2)$ **is represented as the 3-vector:** $\begin{bmatrix} \mathrm{x}_1 \\ \mathrm{x}_2 \\ 1 \end{bmatrix}$

And 2D transforms are represented by 3x3 matrices

For example: 2D rotation in homogeneous coordinates:

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathrm{x}_1 \\ \mathrm{x}_2 \\ 1 \end{bmatrix}$$

Q: how do the transforms we've seen so far affect the last coordinate?

Translation in 2D-H coords

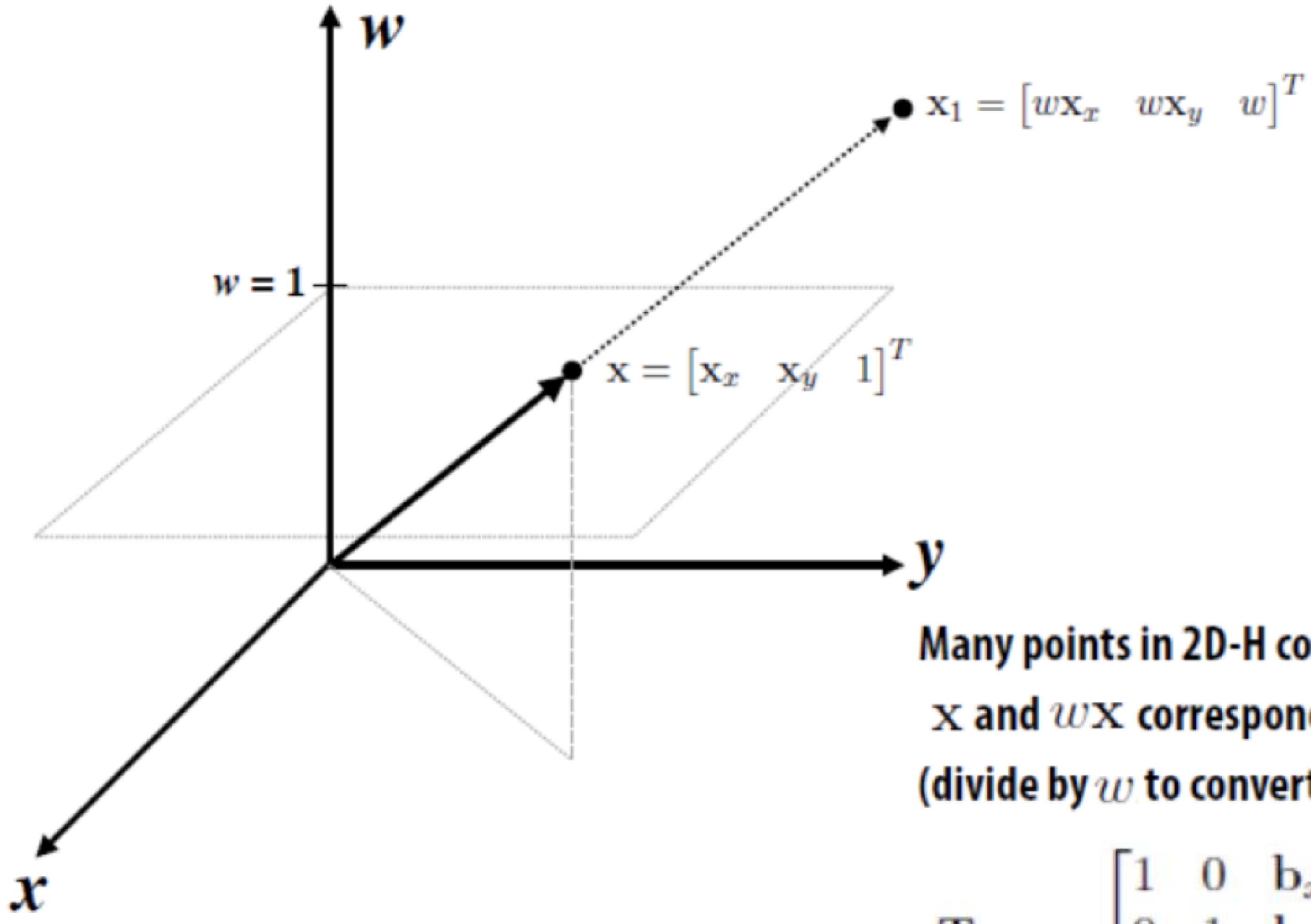
Translation expressed as 3x3 matrix multiplication:

$$\boldsymbol{T}(\boldsymbol{x}) = \boldsymbol{x} + \boldsymbol{b} = \begin{bmatrix} 1 & 0 & b_1 \\ 0 & 1 & b_2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix} = \begin{bmatrix} x_1 + b_1 \\ x_2 + b_2 \\ 1 \end{bmatrix}$$

(remember: linear combination of columns!)

Cool: In homogeneous coordinates, translation is a linear transformation!

Homogeneous coordinates: some intuition



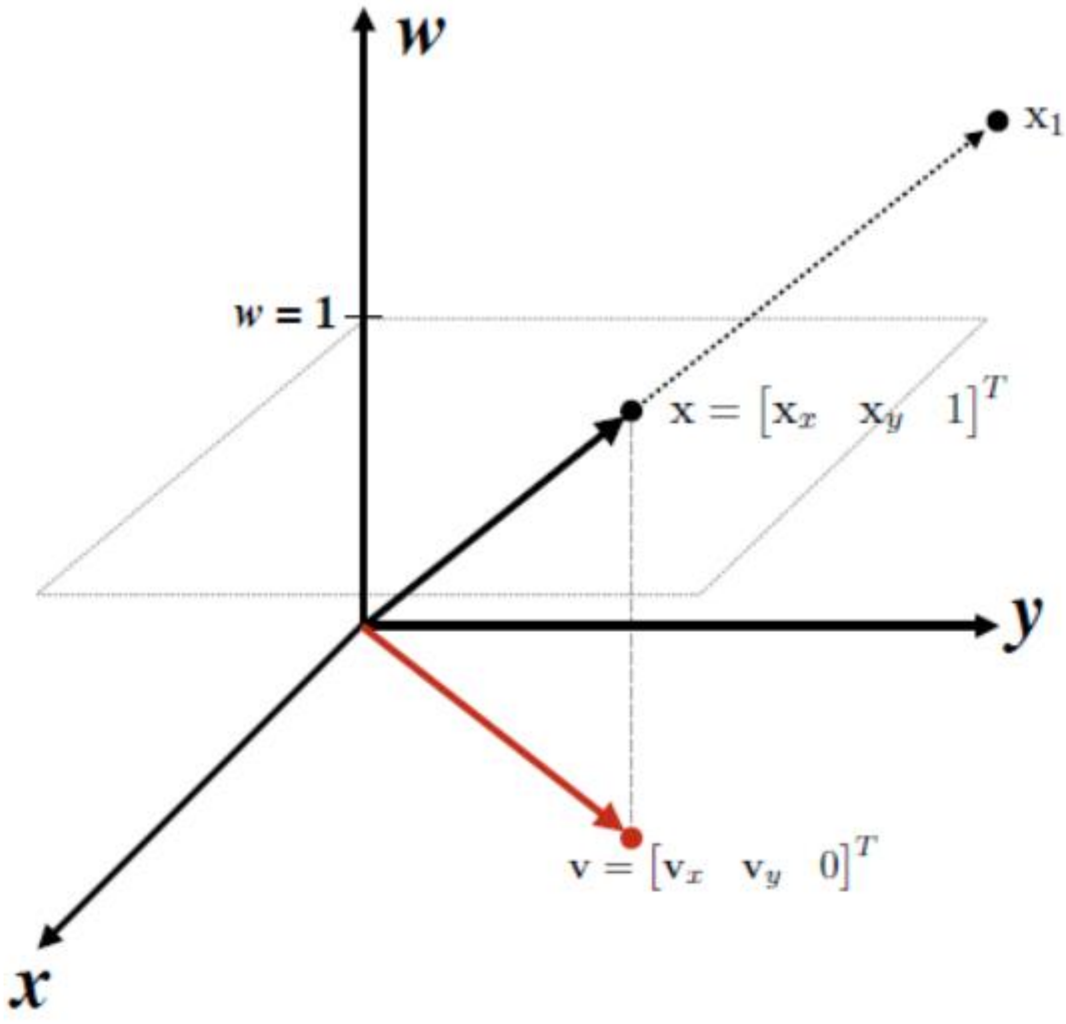
Many points in 2D-H correspond to same point in 2D

x **and** $w\mathrm{x}$ **correspond to the same 2D point**

(divide by w **to convert 2D-H back to 2D)**

$$\mathrm{T_b x} = \begin{bmatrix} 1 & 0 & \mathrm{b}_x \\ 0 & 1 & \mathrm{b}_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} w\mathrm{x}_x \\ w\mathrm{x}_y \\ w \end{bmatrix} = \begin{bmatrix} w\mathrm{x}_x + w\mathrm{b}_x \\ w\mathrm{x}_y + w\mathrm{b}_y \\ w \end{bmatrix}$$

Homogeneous coordinates: points vs. vectors



2D-H points with w=0 represent 2D vectors (think: directions are points at infinity)

Unlike 2D, points and directions are distinguishable by their representation in 2D-H

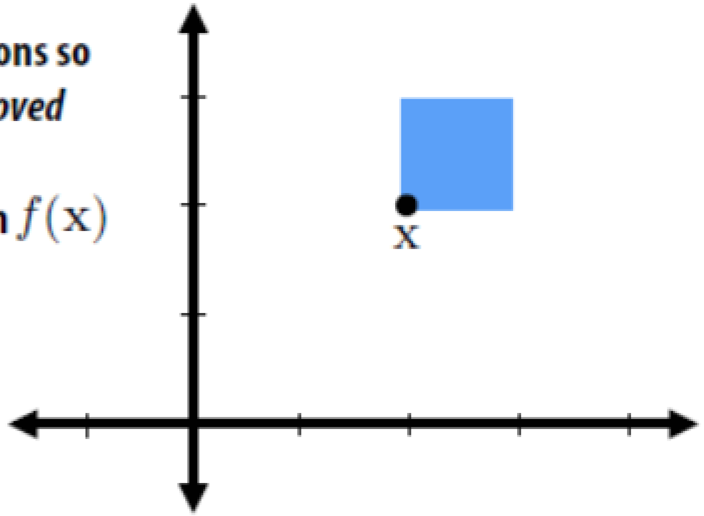
Note: translation does not modify directions:

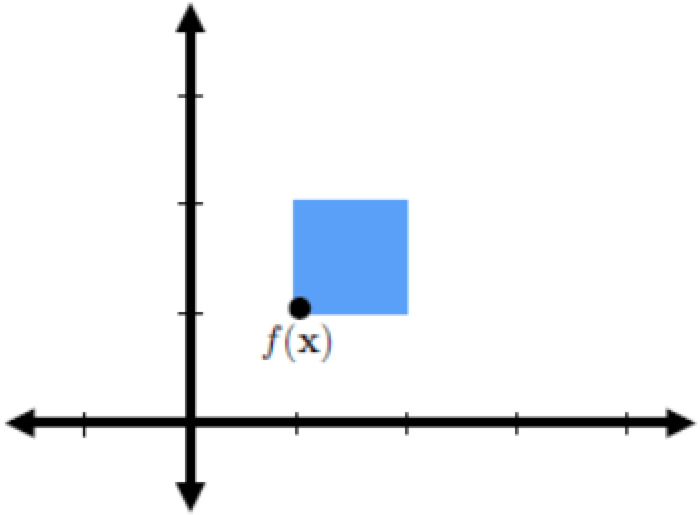
$$\mathbf{T_b v} = \begin{bmatrix} 1 & 0 & \mathbf{b}_x \\ 0 & 1 & \mathbf{b}_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{v}_x \\ \mathbf{v}_y \\ 0 \end{bmatrix} = \begin{bmatrix} \mathbf{v}_x \\ \mathbf{v}_y \\ 0 \end{bmatrix}$$

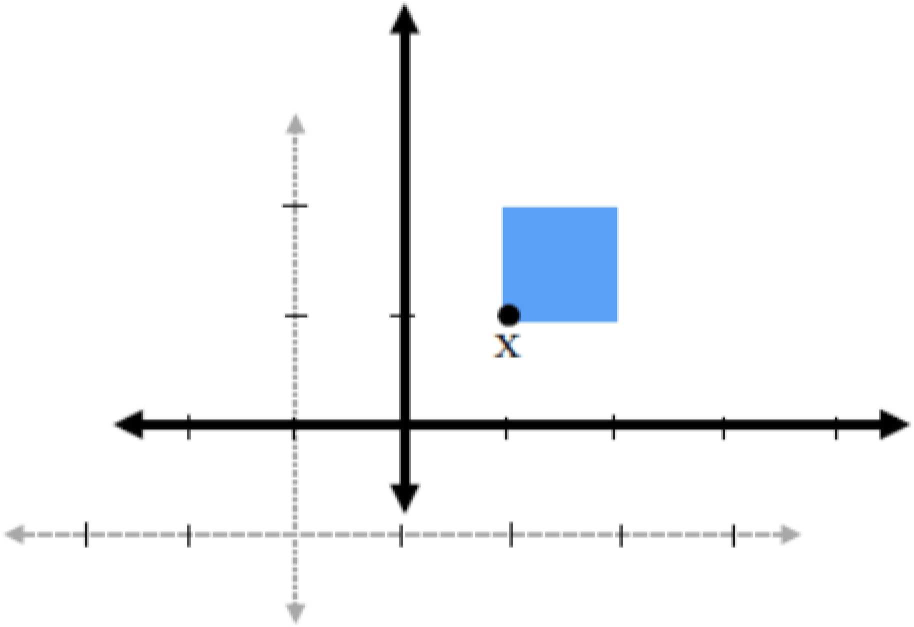
Another way to think about transformations: change of coordinates

Interpretation of transformations so far in this lecture: *points get moved*

Point $\mathbf{x}$ moved to new position $f(\mathbf{x})$







Alternative interpretation:

Transformations induce of change of coordinates: Representation of $\mathbf{x}$ changes since point is now expressed in new coordinates

Moving to 3D (and 3D-H)

Represent 3D transforms as 3x3 matrices and 3D-H transforms as 4x4 matrices

Scale:

3D

$$\mathbf{S_s} = \begin{bmatrix} \mathbf{S}_x & 0 & 0 \\ 0 & \mathbf{S}_y & 0 \\ 0 & 0 & \mathbf{S}_z \end{bmatrix}$$

3D-H

$$\mathbf{S_s} = \begin{bmatrix} \mathbf{S}_x & 0 & 0 & 0 \\ 0 & \mathbf{S}_y & 0 & 0 \\ 0 & 0 & \mathbf{S}_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Shear (in x, based on y,z position):

$$\mathbf{H}_{x,\mathbf{d}} = \begin{bmatrix} 1 & \mathbf{d}_y & \mathbf{d}_z \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{H}_{x,\mathbf{d}} = \begin{bmatrix} 1 & \mathbf{d}_y & \mathbf{d}_z & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

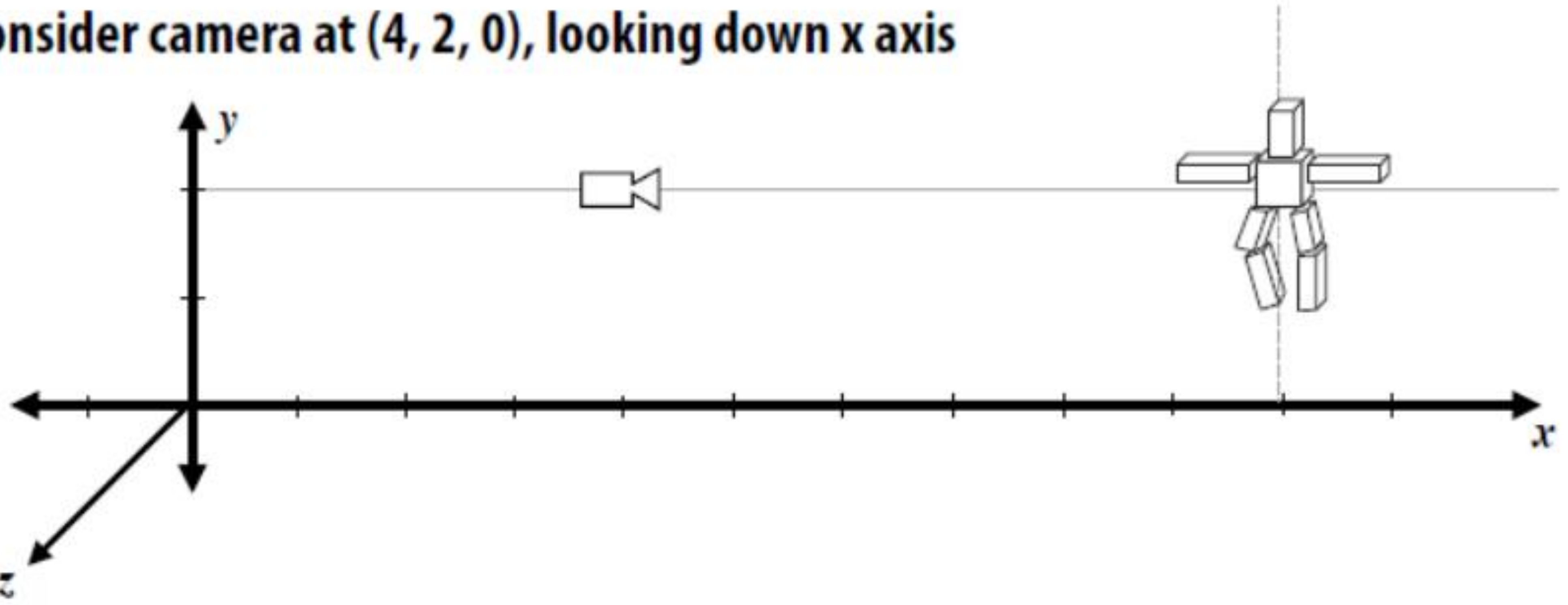
Translate:

3D-H

$$\mathbf{T_b} = \begin{bmatrix} 1 & 0 & 0 & \mathbf{b}_x \\ 0 & 1 & 0 & \mathbf{b}_y \\ 0 & 0 & 1 & \mathbf{b}_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

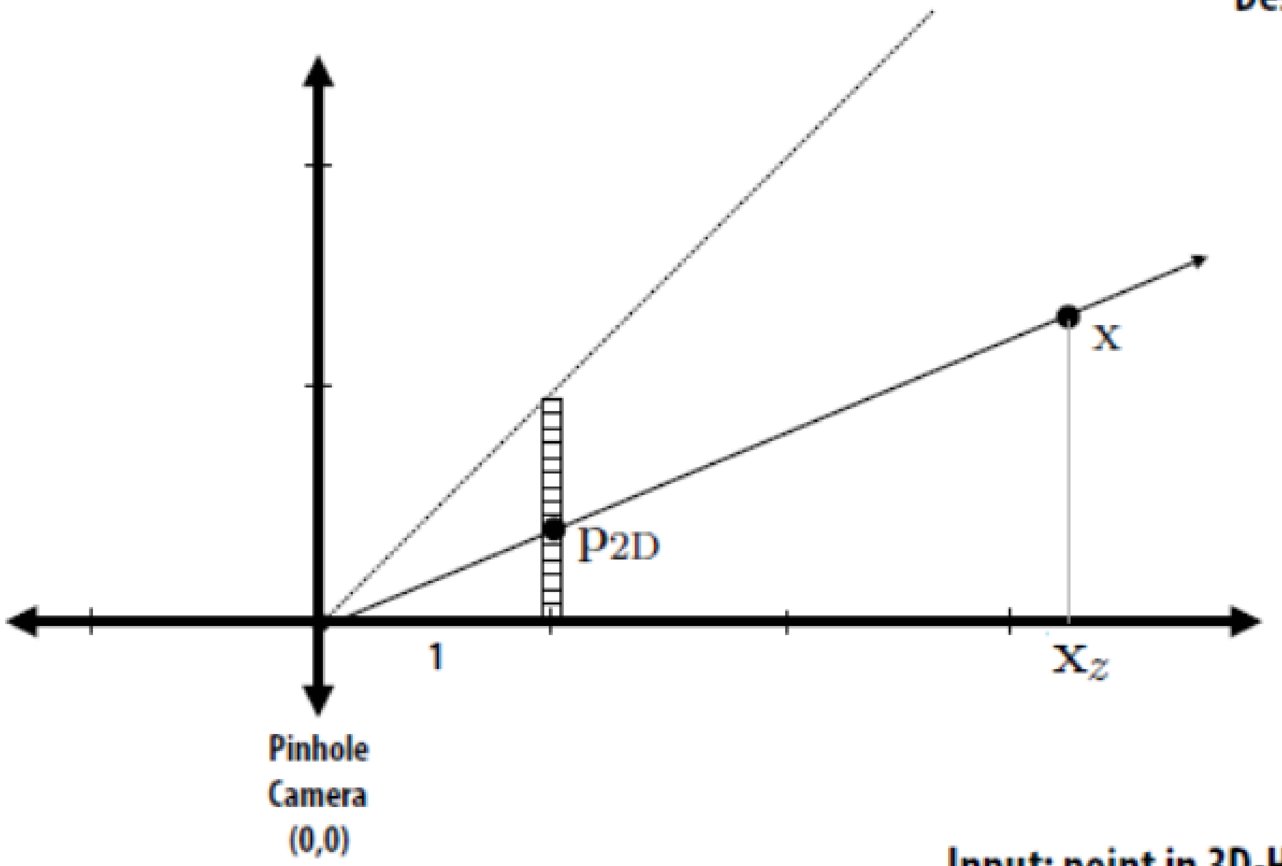
Example: simple camera transform

- Consider object in world at (10, 2, 0)
- Consider camera at (4, 2, 0), looking down x axis



- Translating object vertex positions by (-4, -2, 0) yields position relative to camera.
- Rotation about y by $-\pi/2$ gives position of object in coordinate system where camera's view direction is aligned with the z axis *

Basic perspective projection



Desired perspective projected result (2D point):

$$\mathbf{p}_{2D} = \begin{bmatrix} \mathbf{x}_x/\mathbf{x}_z & \mathbf{x}_y/\mathbf{x}_z \end{bmatrix}^T$$

$$\mathbf{P} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Input: point in 3D-H $\quad \mathbf{x} = \begin{bmatrix} \mathbf{x}_x & \mathbf{x}_y & \mathbf{x}_z & 1 \end{bmatrix}$

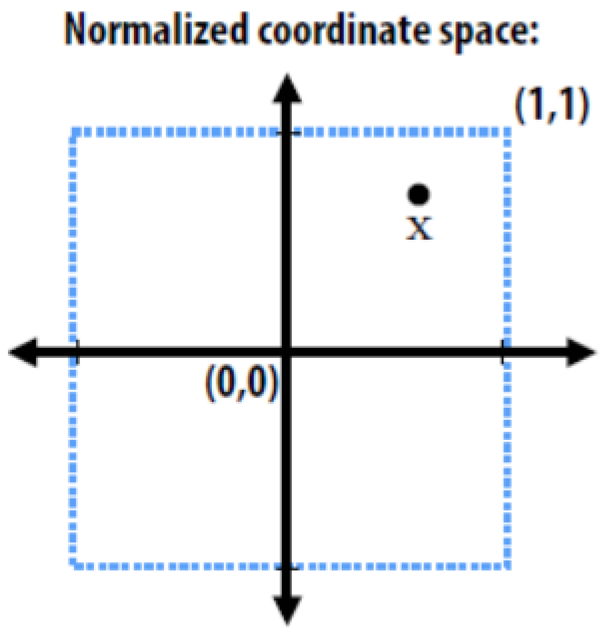
After applying $\mathbf{P}$: point in 3D-H $\quad \mathbf{P}\mathbf{x} = \begin{bmatrix} \mathbf{x}_x & \mathbf{x}_y & \mathbf{x}_z & \mathbf{x}_z \end{bmatrix}^T$

After homogeneous divide: $\quad \begin{bmatrix} \mathbf{x}_x/\mathbf{x}_z & \mathbf{x}_y/\mathbf{x}_z & 1 \end{bmatrix}^T$

(throw out third component)

Screen transformation

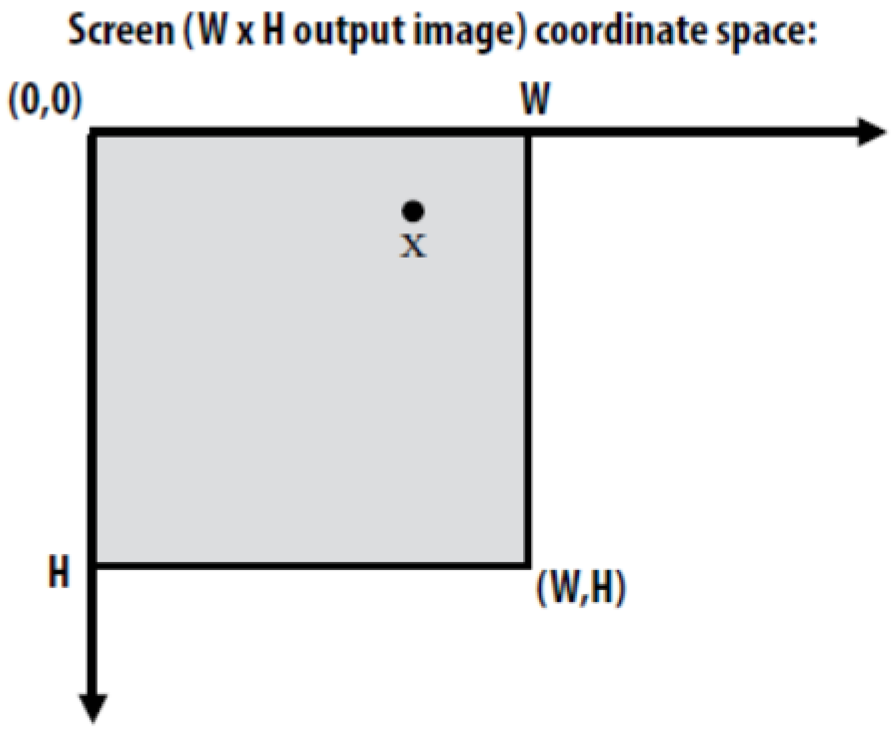
- Convert points in normalized coordinate space to screen pixel coordinates
- Example:
  - All points within (-1,1) to (1,1) region are on screen
  - (1,1) in normalized space maps to (W,0) in screen



Step 1: reflect about x

Step 2: translate by (1,1)

Step 3: scale by (W/2,H/2)



$$f_1\left(f_2\left(f_3(x)\right)\right) = M_1 M_2 M_3 x$$

Summary so far…

- Transformations can be interpreted as operations that move points in space
  - e.g., for modeling, animation
- Or as a change of coordinate system
  - e.g., screen and view transforms

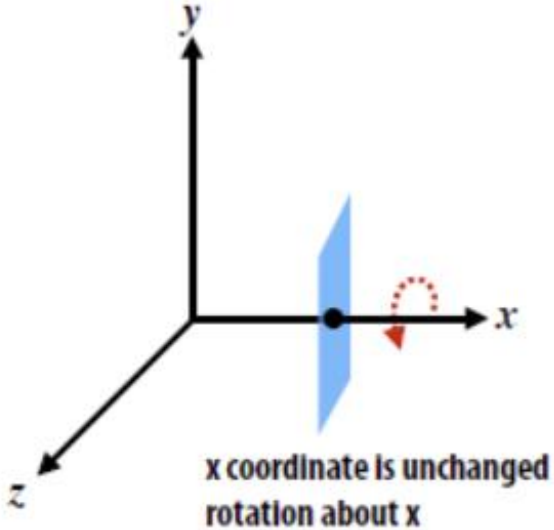
- Construct complex transformations as compositions of basic transforms

- Homogeneous coordinate representation allows for expression of non-linear transforms (e.g., affine, perspective projection) as matrix operations (linear transforms) in higher-dimensional space
  - Matrix representation affords simple implementation & efficient composition

Represenging Rotations in 3D: Euler Angles

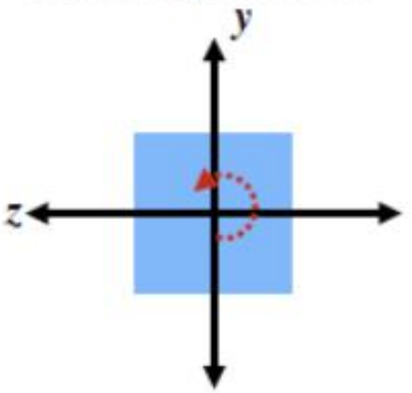
Rotation about x axis:

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



x coordinate is unchanged by rotation about x

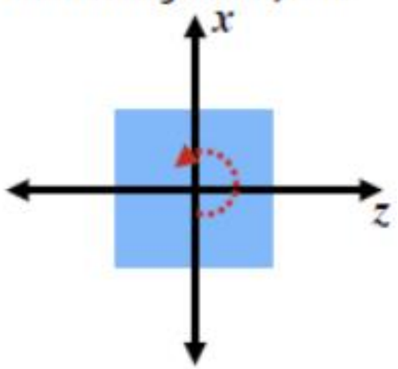
View looking down -x axis:



Rotation about y axis:

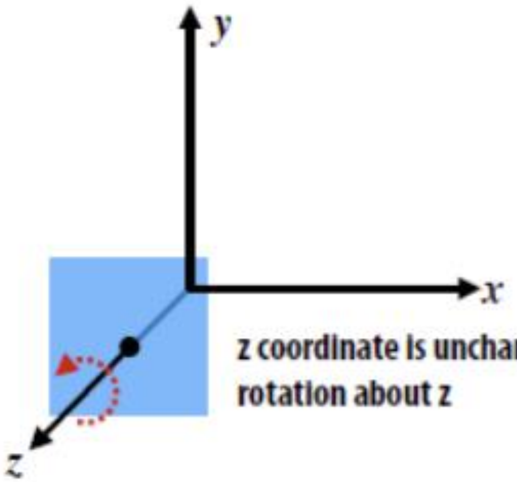
$$\mathbf{R}_{y,\theta} = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

View looking down -y axis:



Rotation about z axis:

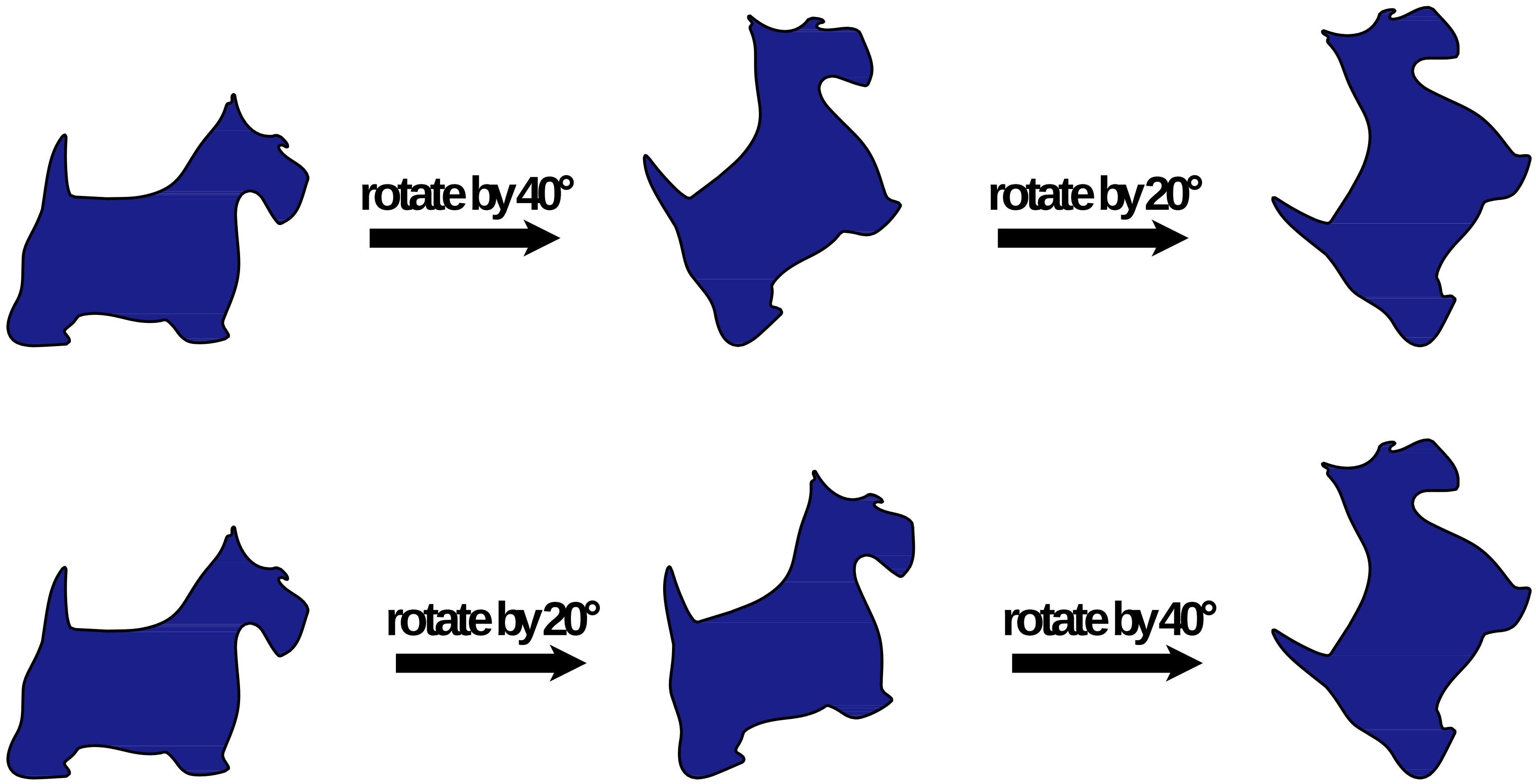
$$\mathbf{R}_{z,\theta} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



z coordinate is unchanged by rotation about z

Commutativity of Rotations—2D

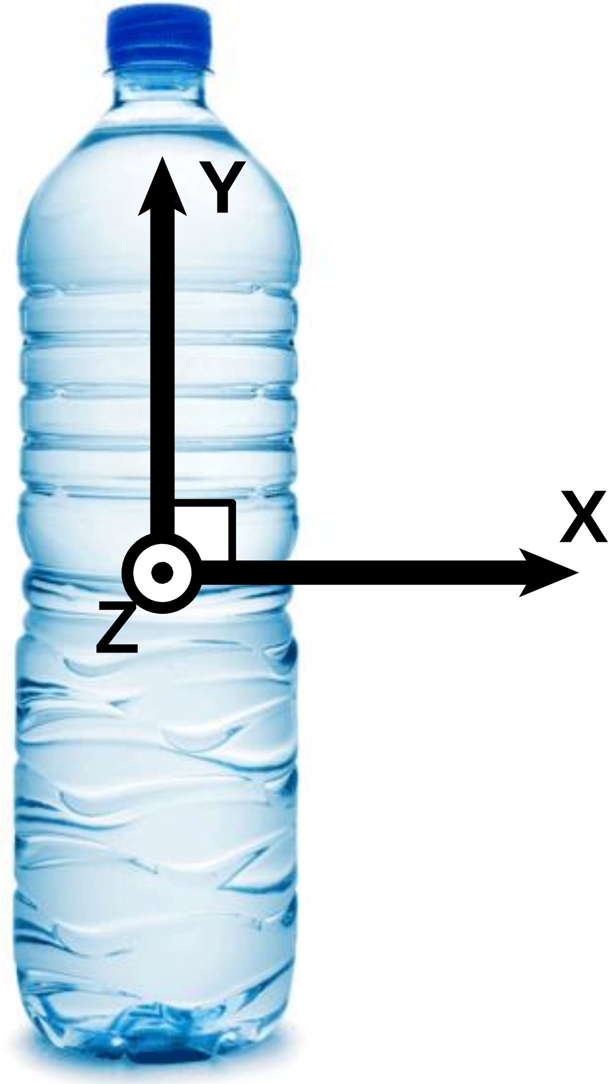
- **In 2D, order of rotations doesn't matter:**



Same result! 2D rotations commute

Commutativity of Rotations—3D

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



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

Recall: Rotation

So, what happens to vectors (1, 0) and (0, 1) after rotation by θ?

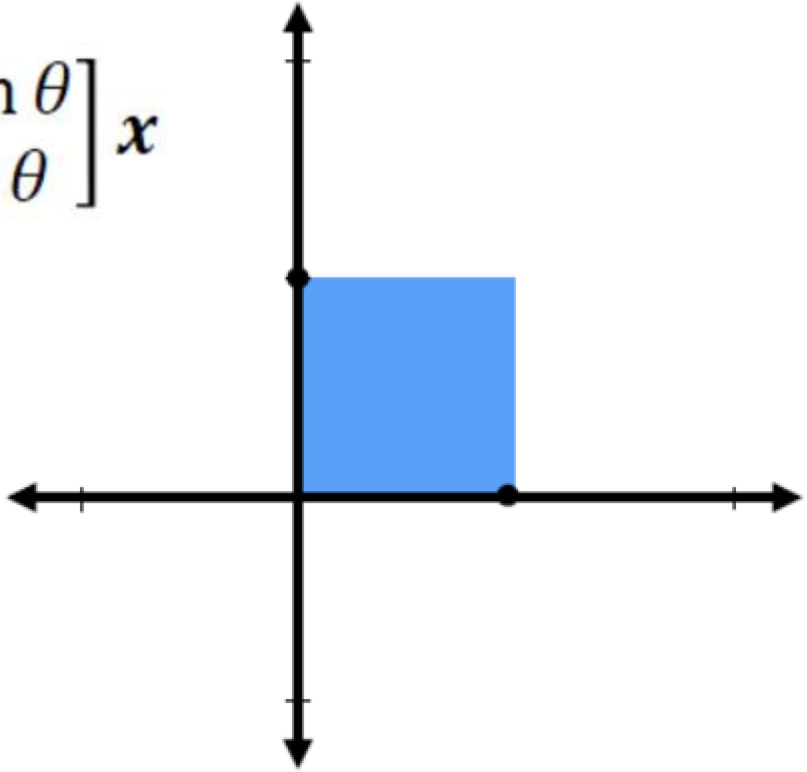
Rotation

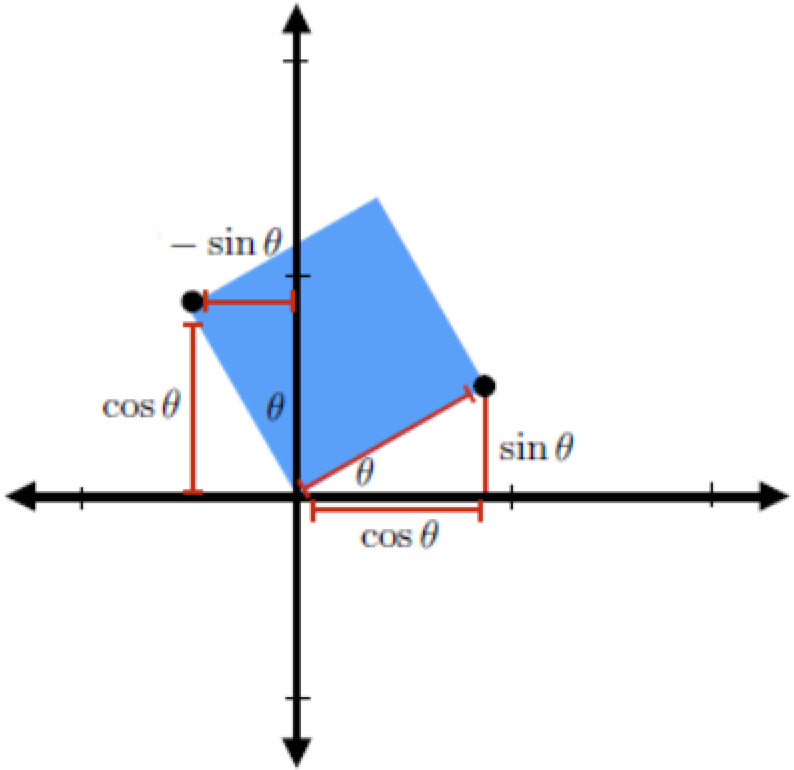
$$R_\theta(\boldsymbol{e}_1) = (cos\ \theta, sin\ \theta) = \boldsymbol{a}_1$$

$$R_\theta(\boldsymbol{e}_2) = (-sin\ \theta, cos\ \theta) = \boldsymbol{a}_2$$

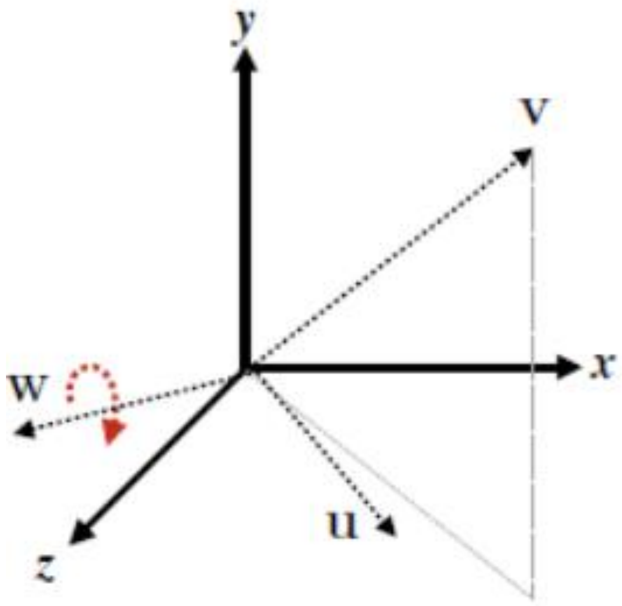
$$R_\theta(\boldsymbol{x}) = x_1\boldsymbol{a}_1 + x_2\boldsymbol{a}_2$$

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





Rotation about an arbitrary axis



To rotate by θ about w:

1. Form orthonormal basis around w (see u and v in figure)
2. Rotate to map w to [0 0 1] (change in coordinate space)

$$\mathbf{R}_{uvw} = \begin{bmatrix} \mathbf{u}_x & \mathbf{u}_y & \mathbf{u}_z \\ \mathbf{v}_x & \mathbf{v}_y & \mathbf{v}_z \\ \mathbf{w}_x & \mathbf{w}_y & \mathbf{w}_z \end{bmatrix}$$

$$\mathbf{R}_{uvw}\mathbf{u} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$$

$$\mathbf{R}_{uvw}\mathbf{v} = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}$$

$$\mathbf{R}_{uvw}\mathbf{w} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$$

3. Perform rotation about z: $\mathrm{R}_{z,\theta}$
4. Rotate back to original coordinate space: $\mathrm{R}_{\mathbf{uvw}}^{\mathbf{T}}$

$$\mathbf{R}_{uvw}^{-1} = \mathbf{R}_{uvw}^{T} = \begin{bmatrix} \mathbf{u}_x & \mathbf{v}_x & \mathbf{w}_x \\ \mathbf{u}_y & \mathbf{v}_y & \mathbf{w}_y \\ \mathbf{u}_z & \mathbf{v}_x & \mathbf{w}_z \end{bmatrix}$$

$$\mathrm{R}_{\mathbf{w},\theta} = \mathrm{R}_{\mathbf{uvw}}^{\mathbf{T}} \mathrm{R}_{z,\theta} \mathrm{R}_{\mathbf{uvw}}$$

Rotation from Axis/Angle

- **Alternatively, there is a general expression for a matrix that performs a rotation around a given axis u by a given angle θ:**

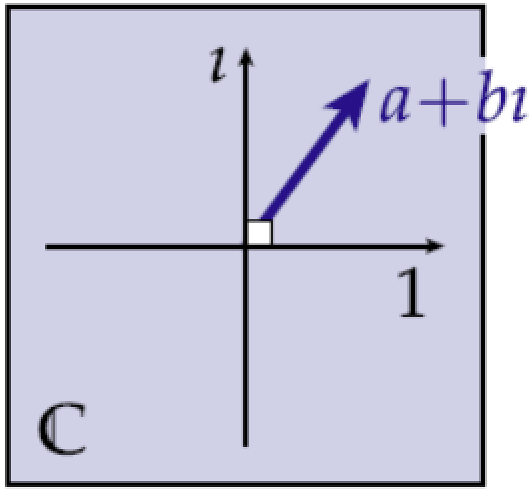
$$\begin{bmatrix} \cos\theta + u_x^2(1-\cos\theta) & u_x u_y(1-\cos\theta) - u_z\sin\theta & u_x u_z(1-\cos\theta) + u_y\sin\theta \\ u_y u_x(1-\cos\theta) + u_z\sin\theta & \cos\theta + u_y^2(1-\cos\theta) & u_y u_z(1-\cos\theta) - u_x\sin\theta \\ u_z u_x(1-\cos\theta) - u_y\sin\theta & u_z u_y(1-\cos\theta) + u_x\sin\theta & \cos\theta + u_z^2(1-\cos\theta) \end{bmatrix}$$

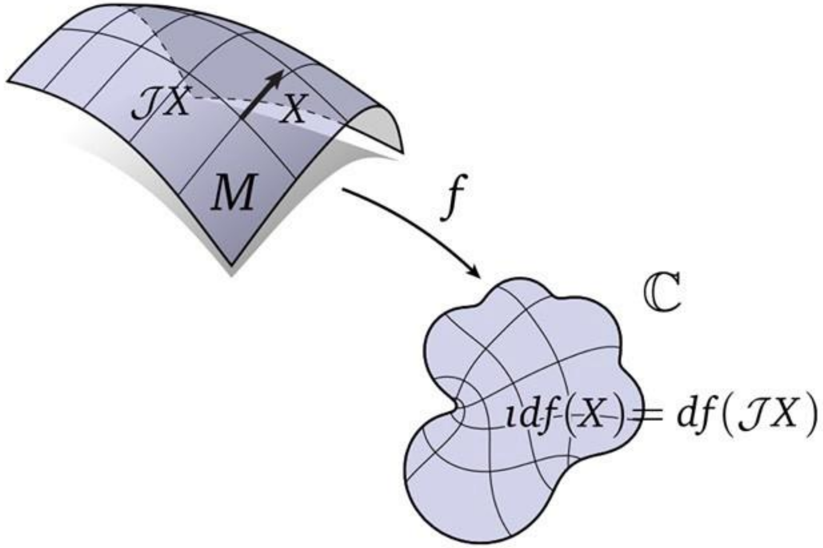
Just memorize this matrix! :-)

…we' ll see a different way, later on.

Complex Analysis—Motivation

- **Natural** way to encode geometric transformations in 2D, 3D
- **Simplifies** notation / thinking / debugging
- ***Moderate*** **reduction** in computational cost/bandwidth/storage
- Fluency with complex analysis can lead into **deeper/novel solutions** to problems…





DON'T: Think of these numbers as "complex."

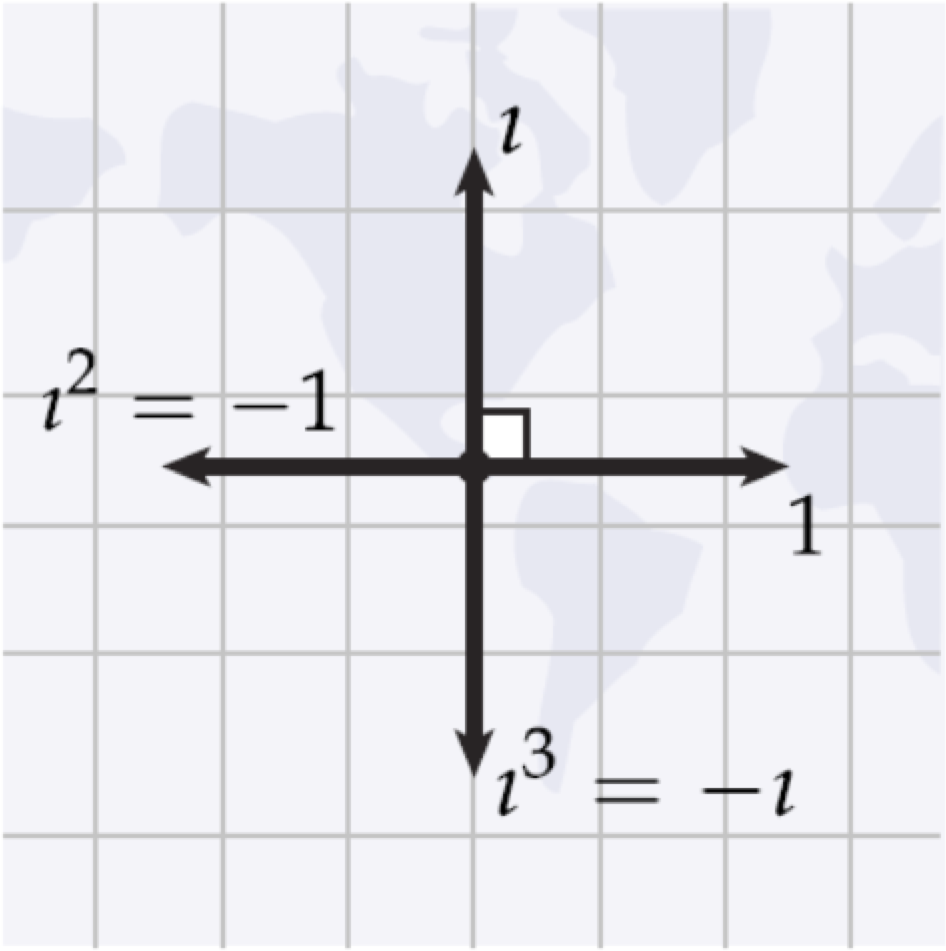
DO: Imagine we're simply defining additional operations (like dot and cross).

*A bit of an oversimplification, but go with it for now!

Imaginary Unit—Geometric Description

$$\cancel{\imath := \sqrt{-1}}$$

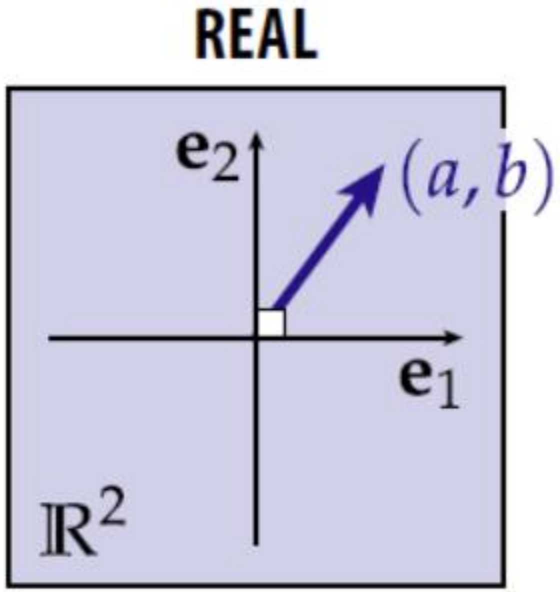
nonsense!



Symbol $\imath$ denotes quarter-turn in the counter-clockwise direction.

Complex Numbers

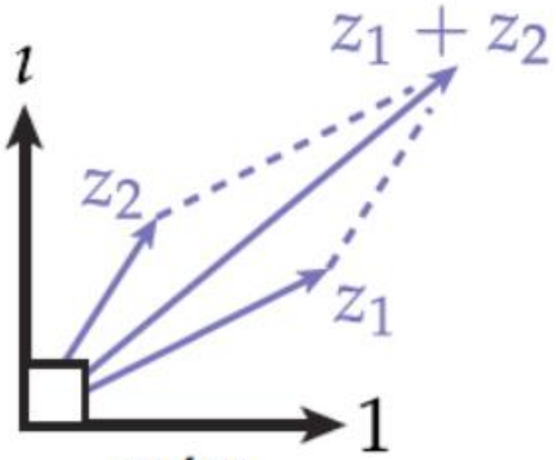
- Complex numbers are then just 2-vectors
- Instead of e_1,e_1, use "1" and "ı" to denote the two bases
- Otherwise, behaves exactly like a real 2-dimensional space



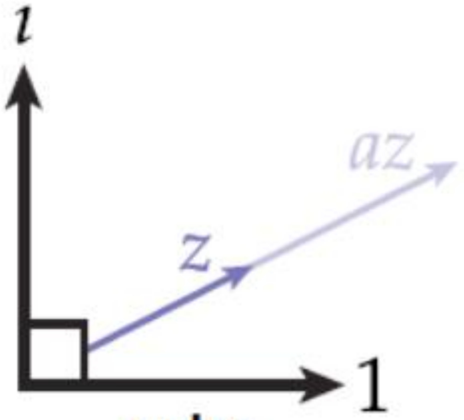
- …except that we're going to define a useful new notion of the product between two vectors.

Complex Arithmetic

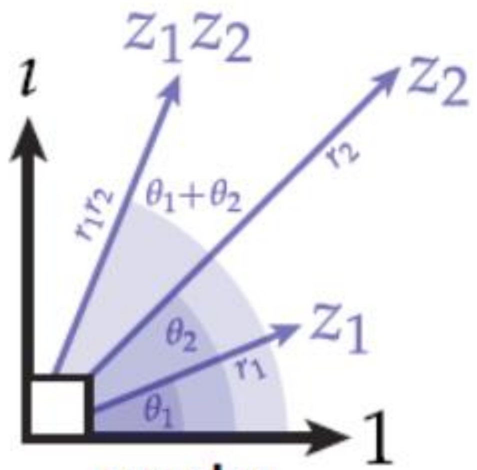
- Same operations as before, plus one more:



vector addition



scalar multiplication



complex multiplication

- Complex multiplication:
  - angles *add*
  - magnitudes *multiply*

"POLAR FORM"*:

$$z_1 := (r_1, \theta_1)$$
$$z_2 := (r_2, \theta_2)$$
$$z_1 z_2 = (r_1 r_2, \theta_1 + \theta_2)$$

have to be more careful here!

*Not *really* now it works, but useful geometric intuition.

Complex Product—Polar Form

- **Perhaps most beautiful identity in math:**

$$e^{\imath\pi} + 1 = 0$$

- **Specialization of *Euler's formula*:**

$$e^{\imath\theta} = \cos(\theta) + \imath\sin(\theta)$$

- **Can use to "implement" complex product:**

$$z_1 = ae^{\imath\theta}, \quad z_2 = be^{\imath\phi}$$

$$z_1 z_2 = abe^{\imath(\theta+\phi)}$$

(as with real exponentiation, exponents *add*)

Leonhard Euler (1707–1783)

- Most prolific mathematician of all time
- Opera Omnia—1 vol./yr. starting 1911
- Still going! Now ~75 vols., 25k pages
- 228 papers posthumously
- Many later works while blind
- (Work was also *good*...)

[source: William Dunham]

Q: How does this operation differ from our earlier, "fake" polar multiplication?

2D Rotations: Matrices vs. Complex

- Suppose we want to rotate a vector u by an angle θ, then by an angle ϕ.

REAL / RECTANGULAR	COMPLEX / POLAR
$\mathbf{u} = (x, y)$ $\quad \mathbf{A} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$	$u = re^{\imath\alpha}$
	$a = e^{\imath\theta}$
$\mathbf{B} = \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix}$	$b = e^{\imath\phi}$
$\mathbf{Au} = \begin{bmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \end{bmatrix}$	$abu = re^{\imath(\alpha+\theta+\phi)}.$
$\mathbf{BAu} = \begin{bmatrix} (x\cos\theta - y\sin\theta)\cos\phi - (x\sin\theta + y\cos\theta)\sin\phi \\ (x\cos\theta - y\sin\theta)\sin\phi + (x\sin\theta + y\cos\theta)\cos\phi \end{bmatrix}$	Or if we want rectangular coords:
$= \cdots \text{some trigonometry} \cdots =$	$= r\begin{bmatrix} \cos(\alpha+\theta+\phi) \\ \sin(\alpha+\theta+\phi) \end{bmatrix}$
$\mathbf{BAu} = \begin{bmatrix} x\cos(\theta+\phi) - y\sin(\theta+\phi) \\ x\sin(\theta+\phi) + y\cos(\theta+\phi) \end{bmatrix}.$	
(…and simplification is not always this obvious.)	

Pervasive theme in graphics:

Sure, there are often many “equivalent” representations.

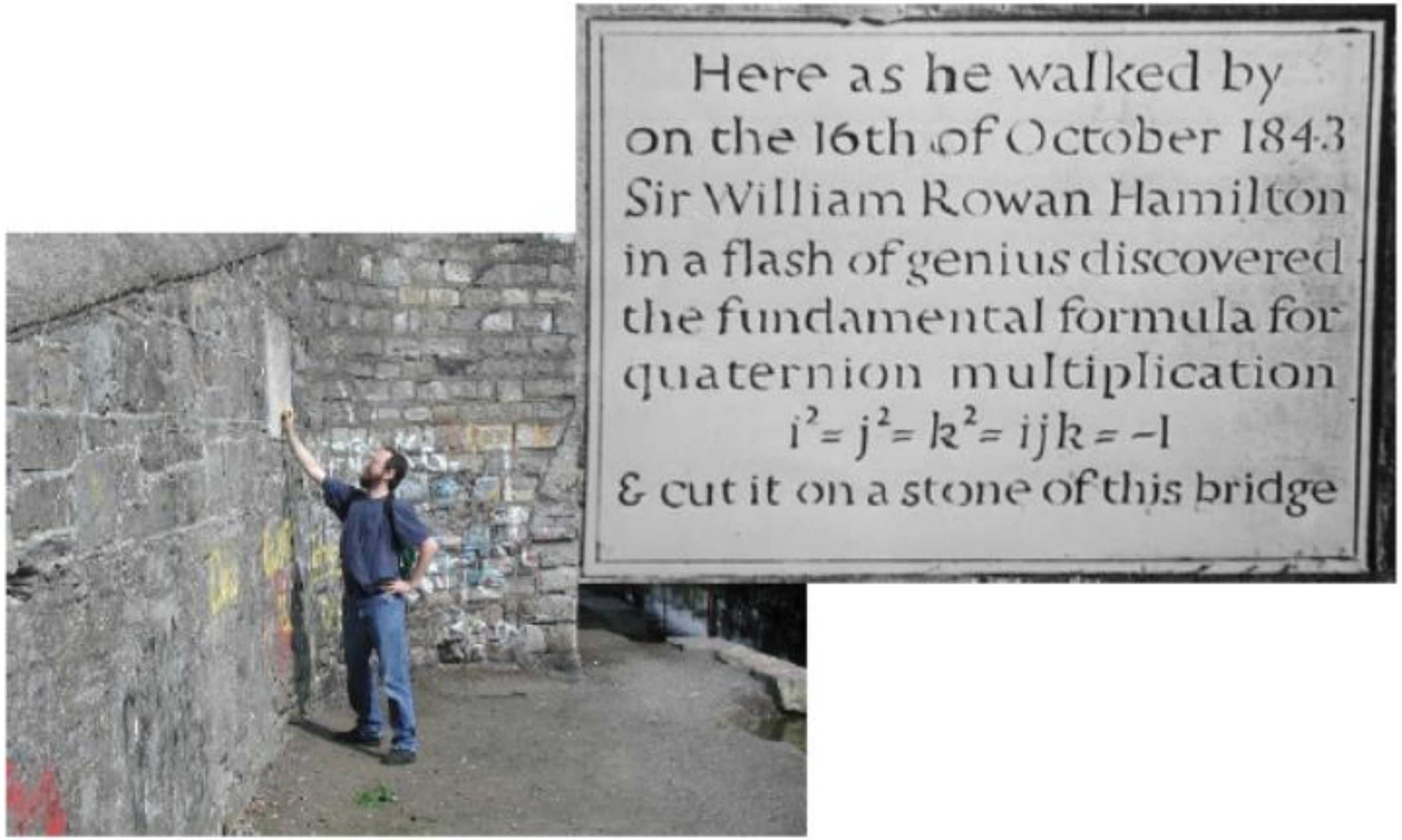
…But why not choose the one that makes life easiest*?

*Or most efficient, or most accurate…

Quaternions

- TLDR: Kind of like complex numbers but for 3D rotations
- Weird situation: can't do 3D rotations w/ only 3 components!

William Rowan Hamilton
(1805-1865)



(Not Hamilton)

Quaternions in Coordinates

- **Hamilton's insight: in order to do 3D rotations in a way that mimics complex numbers for 2D, actually need FOUR coords.**
- **One real, *three* imaginary:**

$$\mathbb{H} := \text{span}(\{1, \imath, \jmath, k\})$$

"H" is for *Hamilton!*

$$q = a + b\imath + c\jmath + dk \in \mathbb{H}$$

- **Quaternion product determined by**

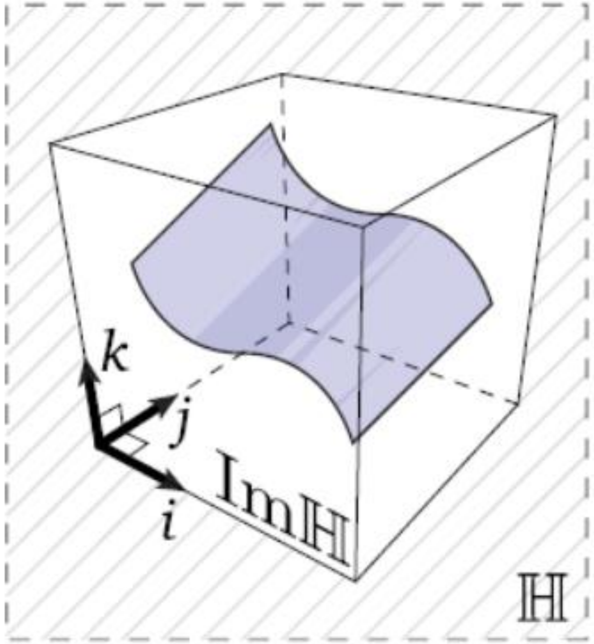
$$\imath^2 = \jmath^2 = k^2 = \imath\jmath k = -1$$

together w/ "natural" rules (distributivity, associativity, etc.)

- **WARNING: product no longer commutes!**

$$\text{For } q, p \in \mathbb{H}, \quad qp \neq pq$$

(Will understand this *a lot* better when we study transformations.)



Noncommutativity of quaternion multiplication

×	1	*i*	*j*	*k*
1	1	*i*	*j*	*k*
i	*i*	−1	*k*	−*j*
j	*j*	−*k*	−1	*i*
k	*k*	*j*	−*i*	−1

Quaternion Product / Hamilton product

- **Given two quaternions**

$$q = a_1 + b_1 \imath + c_1 \jmath + d_1 k$$
$$p = a_2 + b_2 \imath + c_2 \jmath + d_2 k$$

- **Can express their product as**

$$\begin{aligned} qp = &\ a_1 a_2 - b_1 b_2 - c_1 c_2 - d_1 d_2 \\ &+ (a_1 b_2 + b_1 a_2 + c_1 d_2 - d_1 c_2)\imath \\ &+ (a_1 c_2 - b_1 d_2 + c_1 a_2 + d_1 b_2)\jmath \\ &+ (a_1 d_2 + b_1 c_2 - c_1 b_2 + d_1 a_2)k \end{aligned}$$

...fortunately there is a (much) nicer expression.

Quaternions—Scalar + Vector Form

- If we have *four* components, how do we talk about pts in 3D?
- Natural idea: we have three imaginary parts—why not use these to encode 3D vectors?

$$(x, y, z) \mapsto 0 + x\imath + y\jmath + zj$$

- Alternatively, can think of a quaternion as a pair

$$\begin{array}{c} (\ \text{scalar},\ \text{vector}\ ) \in \mathbb{H} \\ \text{\rotatebox{90}{$\in$}} \qquad\qquad\ \text{\rotatebox{90}{$\in$}} \\ \ \ \ \ \ \ \ \mathbb{R} \qquad\quad \mathbb{R}^3 \end{array}$$

- Quaternion product then has simple(r) form:

$$(a, \mathbf{u})(b, \mathbf{v}) = (ab - \mathbf{u} \cdot \mathbf{v}, a\mathbf{v} + b\mathbf{u} + \mathbf{u} \times \mathbf{v})$$

- For vectors in R3, gets even simpler:

$$\mathbf{u}\mathbf{v} = \mathbf{u} \times \mathbf{v} - \mathbf{u} \cdot \mathbf{v}$$

Conjugation & Norm

To define it, let $q = a + bi + cj + dk$ be a quaternion. The **conjugate** of q is the quaternion $q^* = a - bi - cj - dk$. It is denoted by q^*, $\overline{q}$,[6] q^t, or $\tilde{q}$.

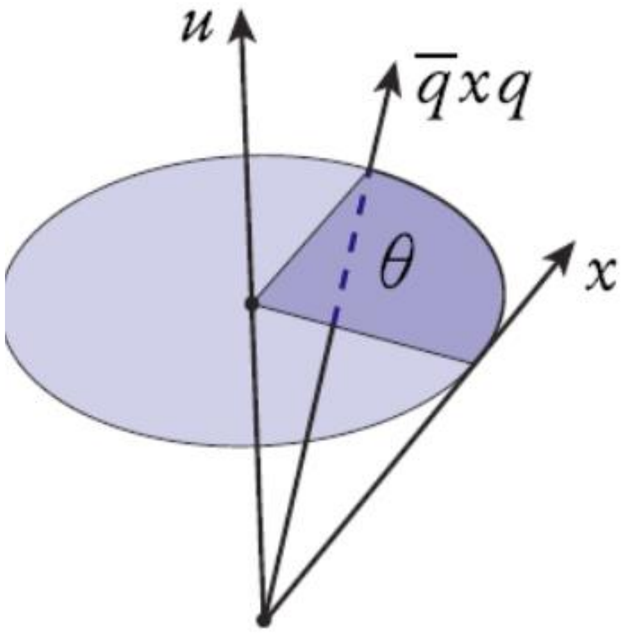
Conjugation is an involution, meaning **that it is its own inverse**, so conjugating an element twice returns the original element.

$$\|q\| = \sqrt{qq^*} = \sqrt{q^*q} = \sqrt{a^2 + b^2 + c^2 + d^2}$$

3D Transformations via Quaternions

- **Main use for quaternions in graphics? *Rotations.***
- **Consider vector x ("pure imaginary") and *unit* quaternion q:**

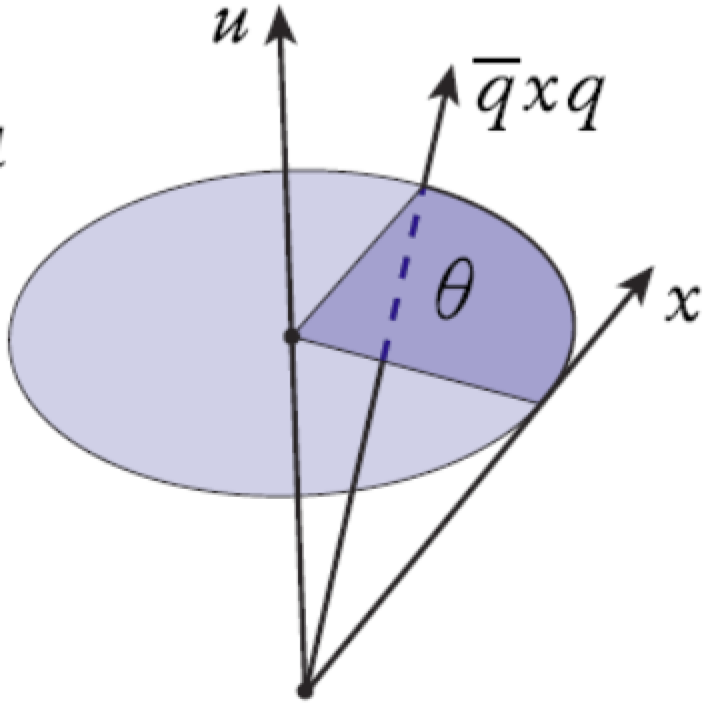
$$x \in \mathrm{Im}(\mathbb{H})$$
$$q \in \mathbb{H}, \quad |q|^2 = 1$$



Rotation from Axis/Angle, Revisited

- **Given axis u, angle θ, quaternion q representing rotation is**

$$q = \cos(\theta/2) + \sin(\theta/2)u$$

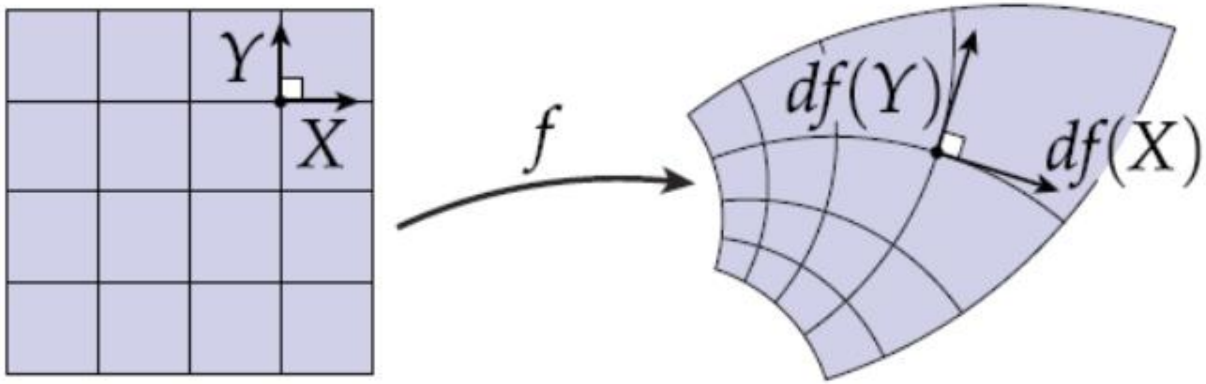


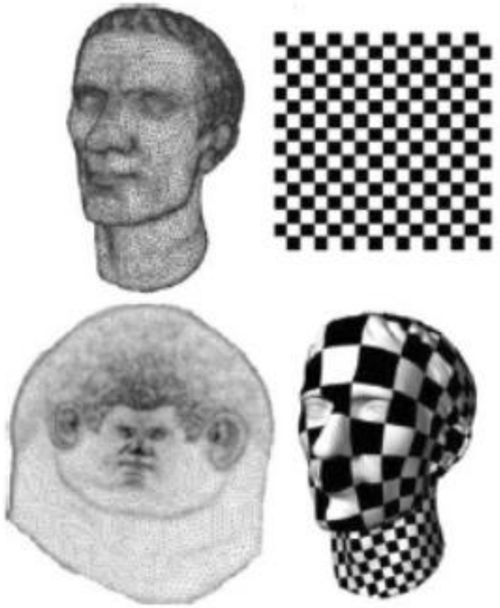
- **Slightly easier to remember (and manipulate) than matrix:**

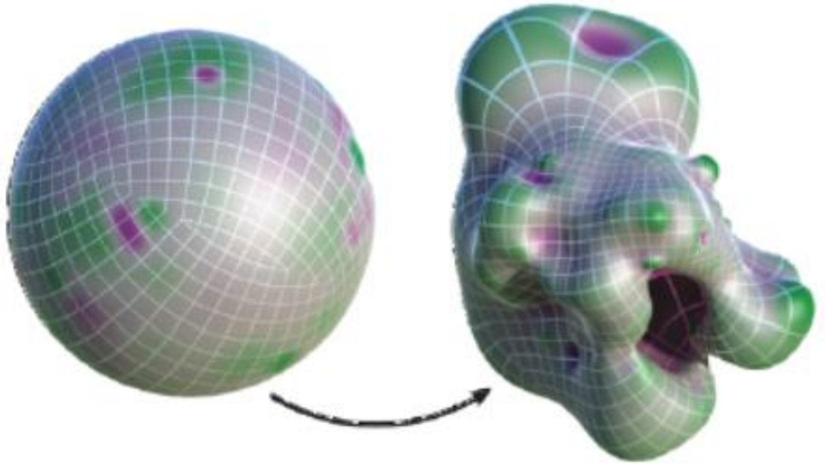
$$\begin{bmatrix} \cos\theta + u_x^2(1-\cos\theta) & u_x u_y(1-\cos\theta) - u_z\sin\theta & u_x u_z(1-\cos\theta) + u_y\sin\theta \\ u_y u_x(1-\cos\theta) + u_z\sin\theta & \cos\theta + u_y^2(1-\cos\theta) & u_y u_z(1-\cos\theta) - u_x\sin\theta \\ u_z u_x(1-\cos\theta) - u_y\sin\theta & u_z u_y(1-\cos\theta) + u_x\sin\theta & \cos\theta + u_z^2(1-\cos\theta) \end{bmatrix}$$

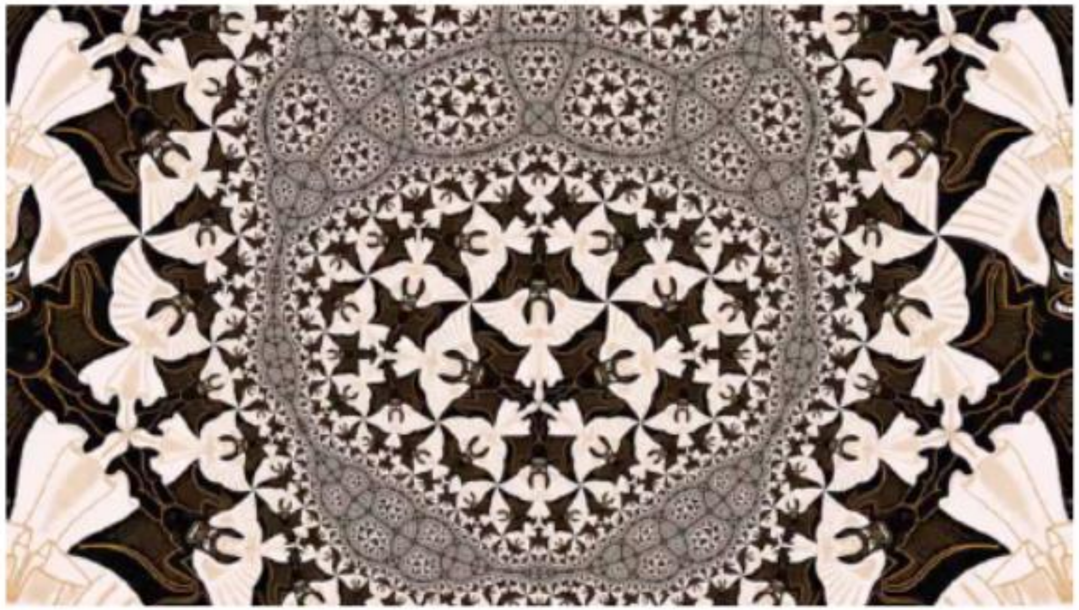
Where else are (hyper-)complex numbers useful in computer graphics?

Complex #s: Language of *Conformal Maps*



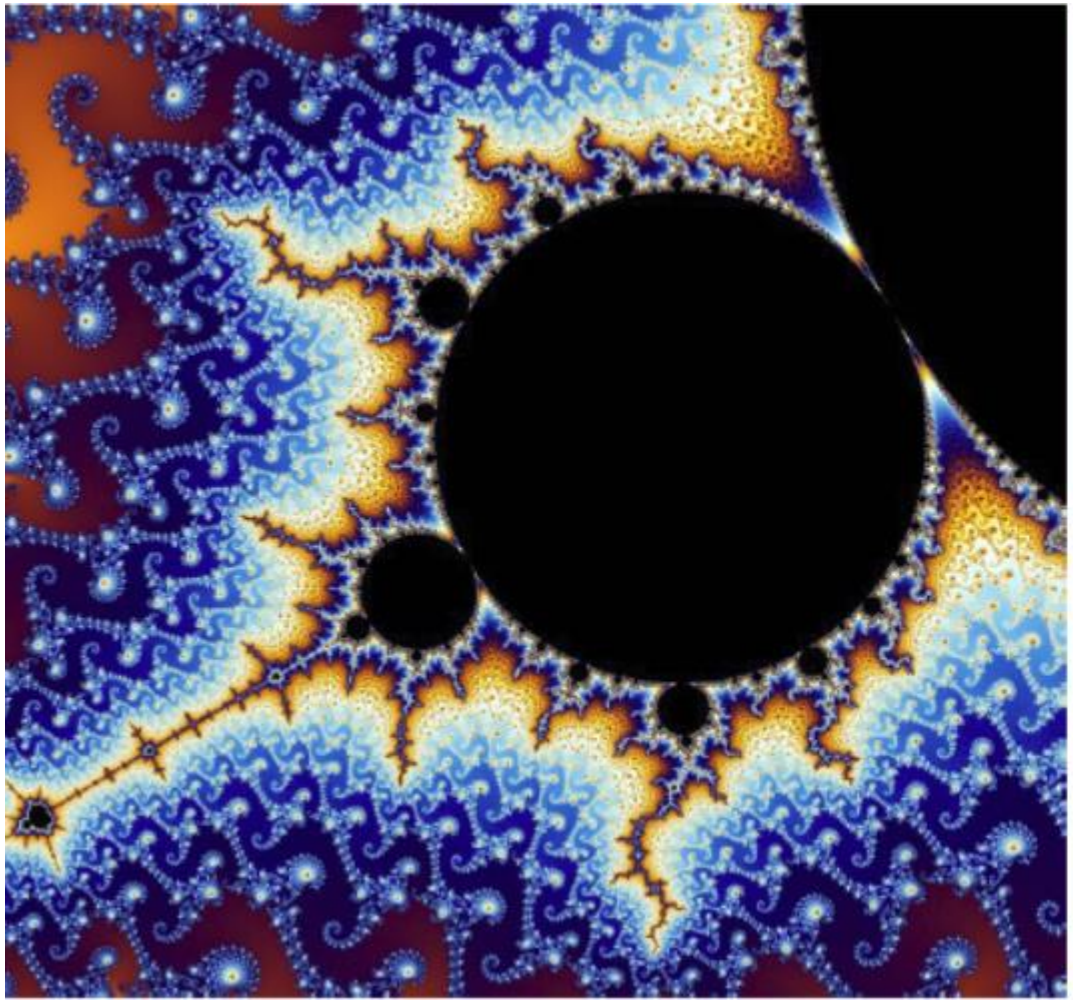


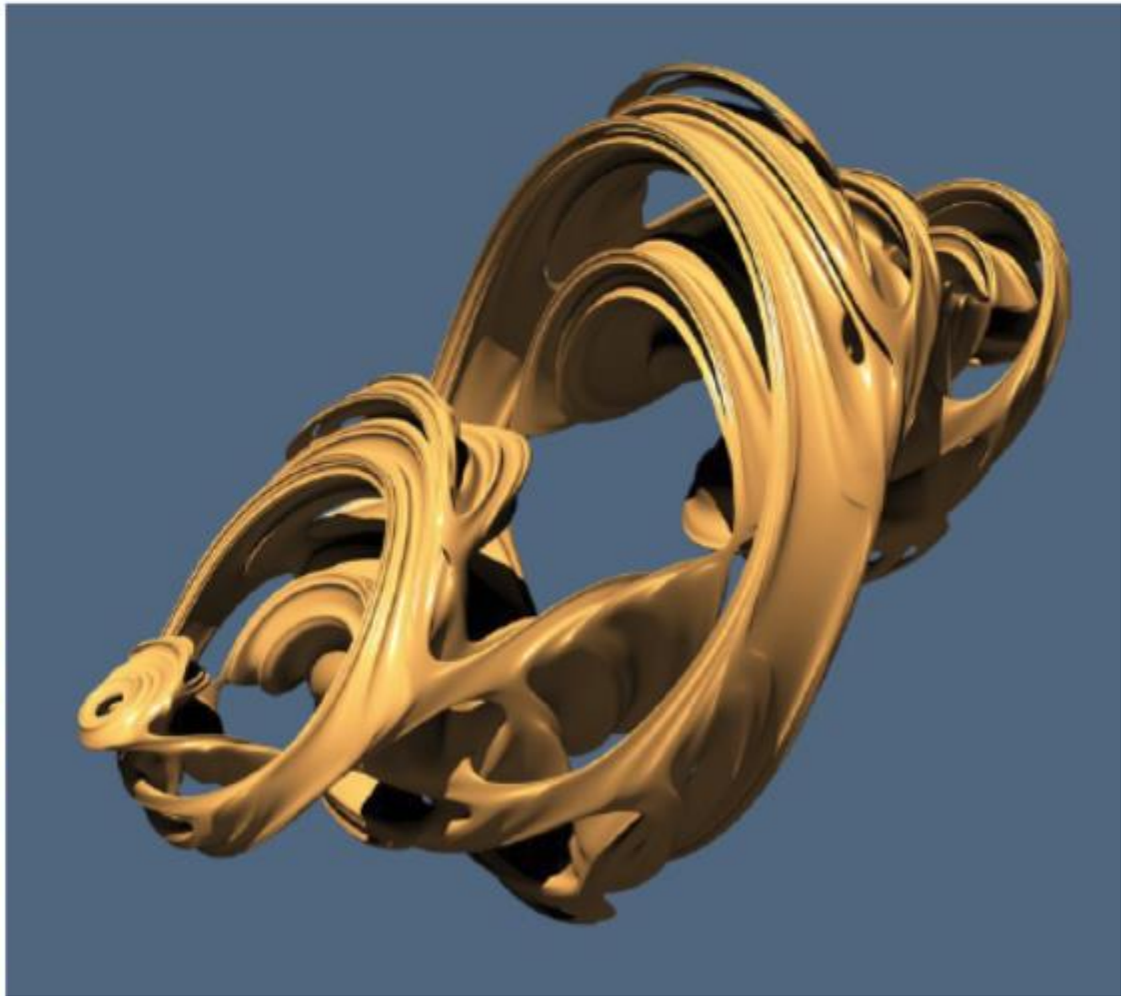




Useless-But-Beautiful Example: Fractals

- **Defined in terms of iteration on (hyper)complex numbers:**





Thanks