Computer Graphics
-Transforms

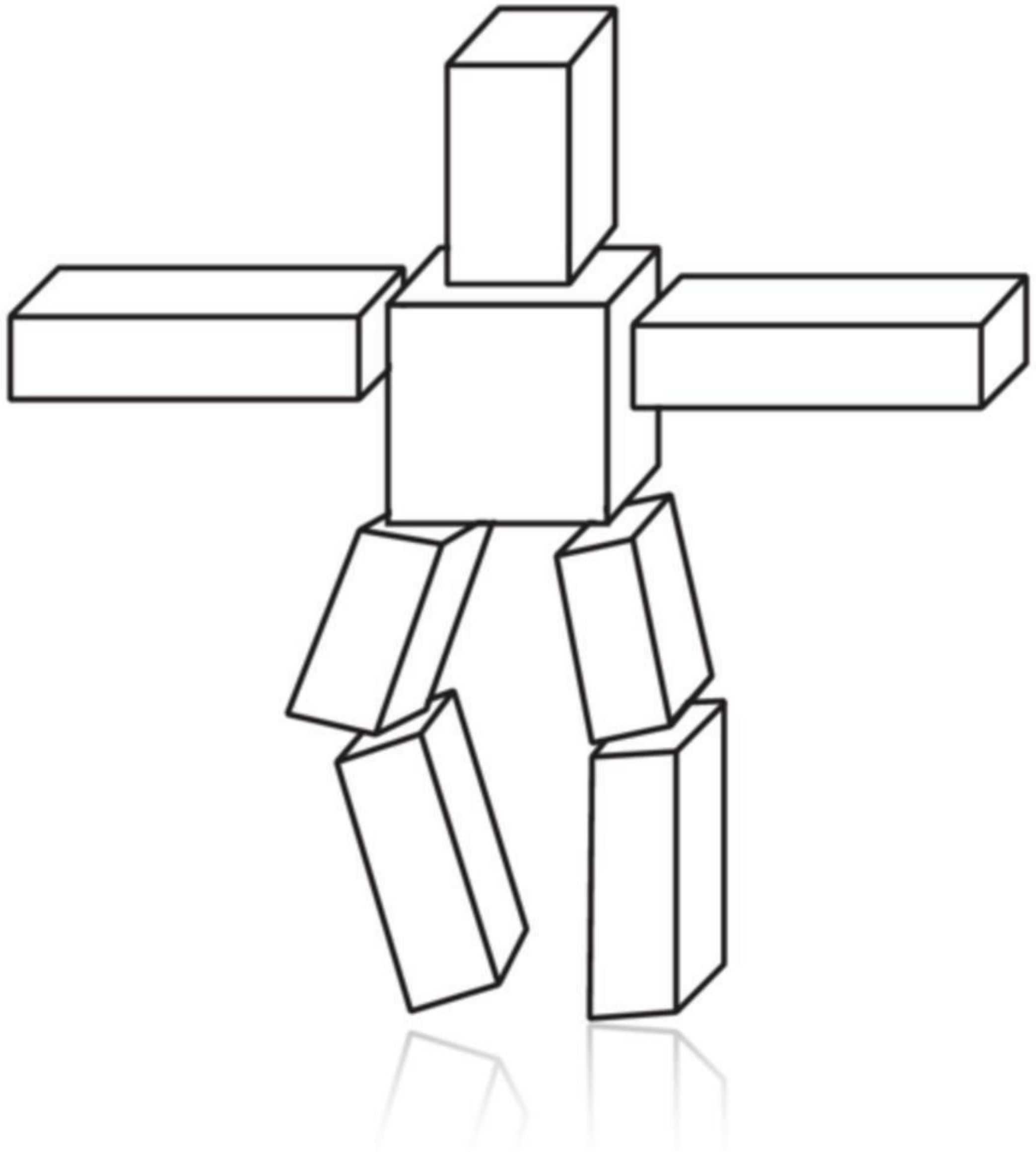
Junjie Cao @ DLUT

Spring 2017

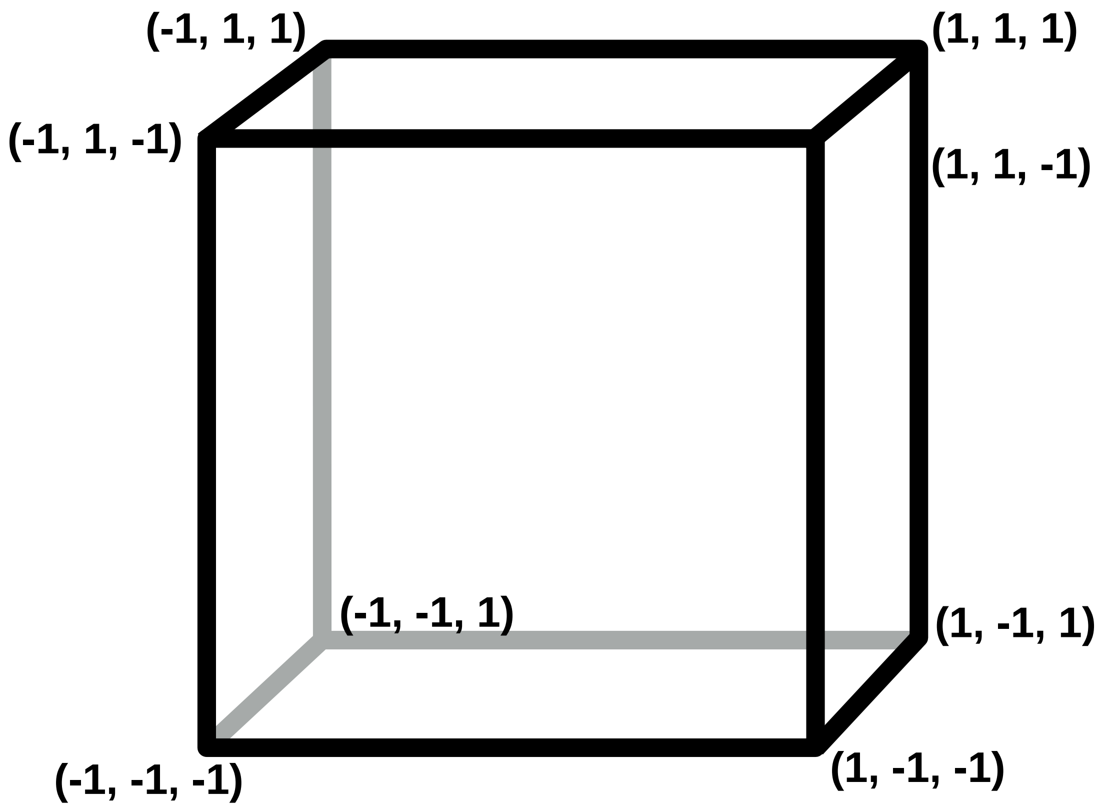
http://jjcao.github.io/ComputerGraphics/

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

What in the world is this?



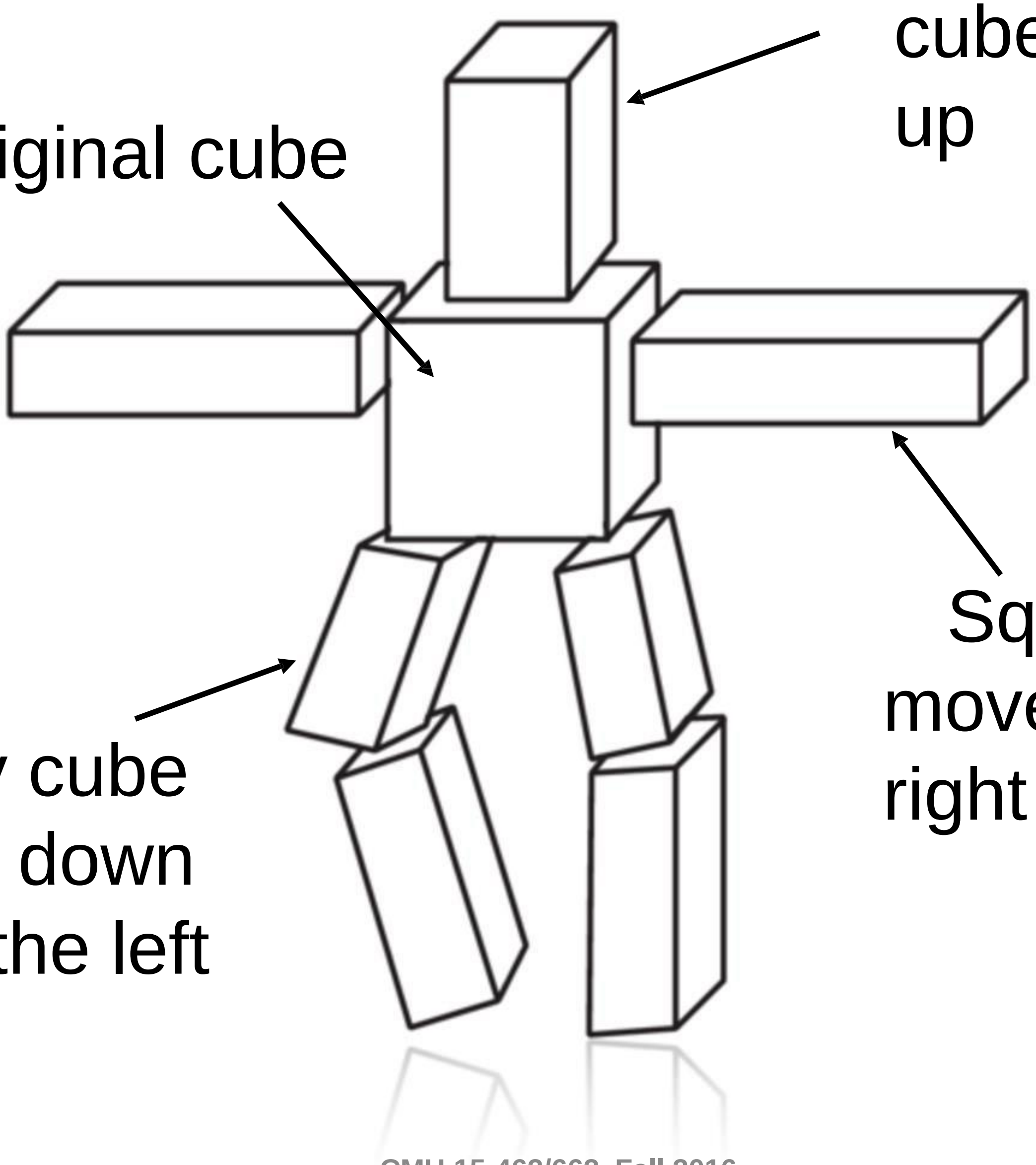
Cube



Cube man

Stretched out cube moved up

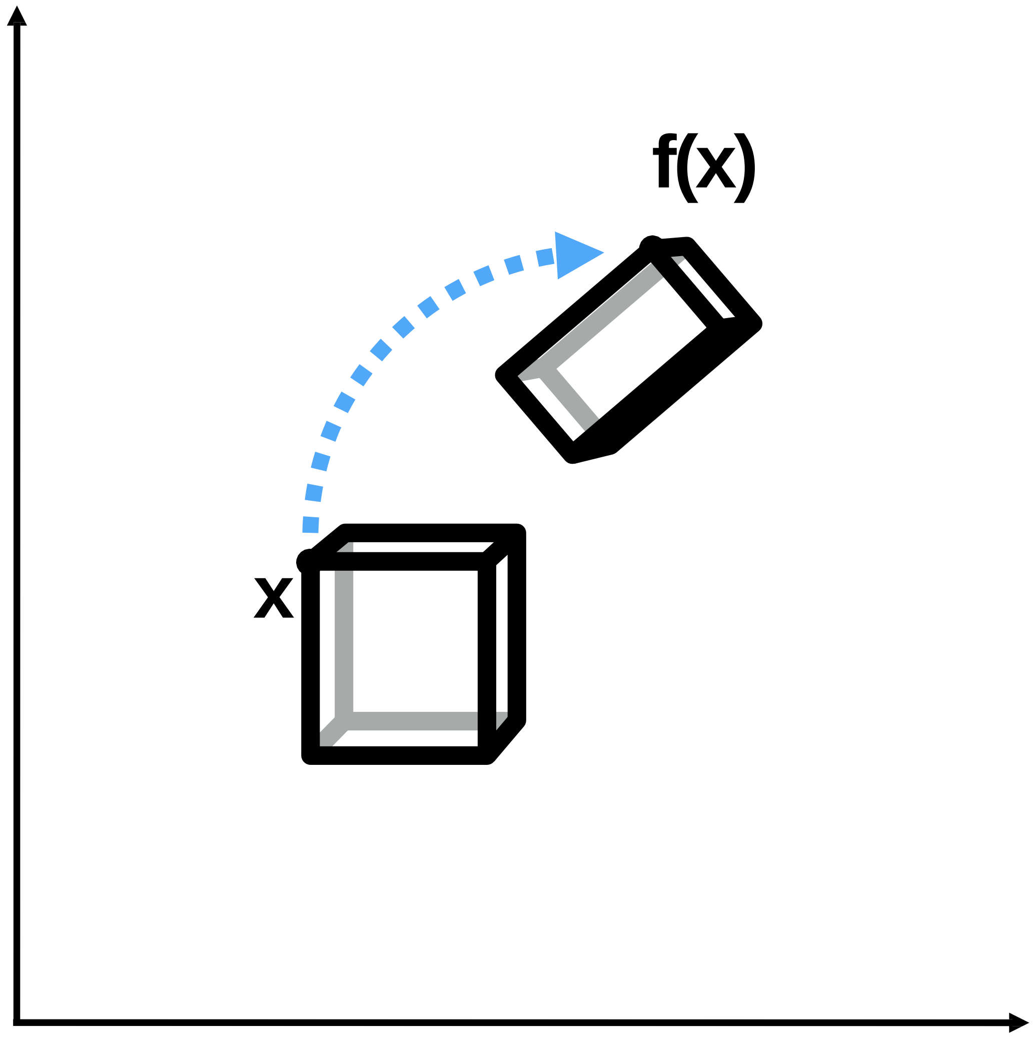
The original cube



Squishy cube moved to the right

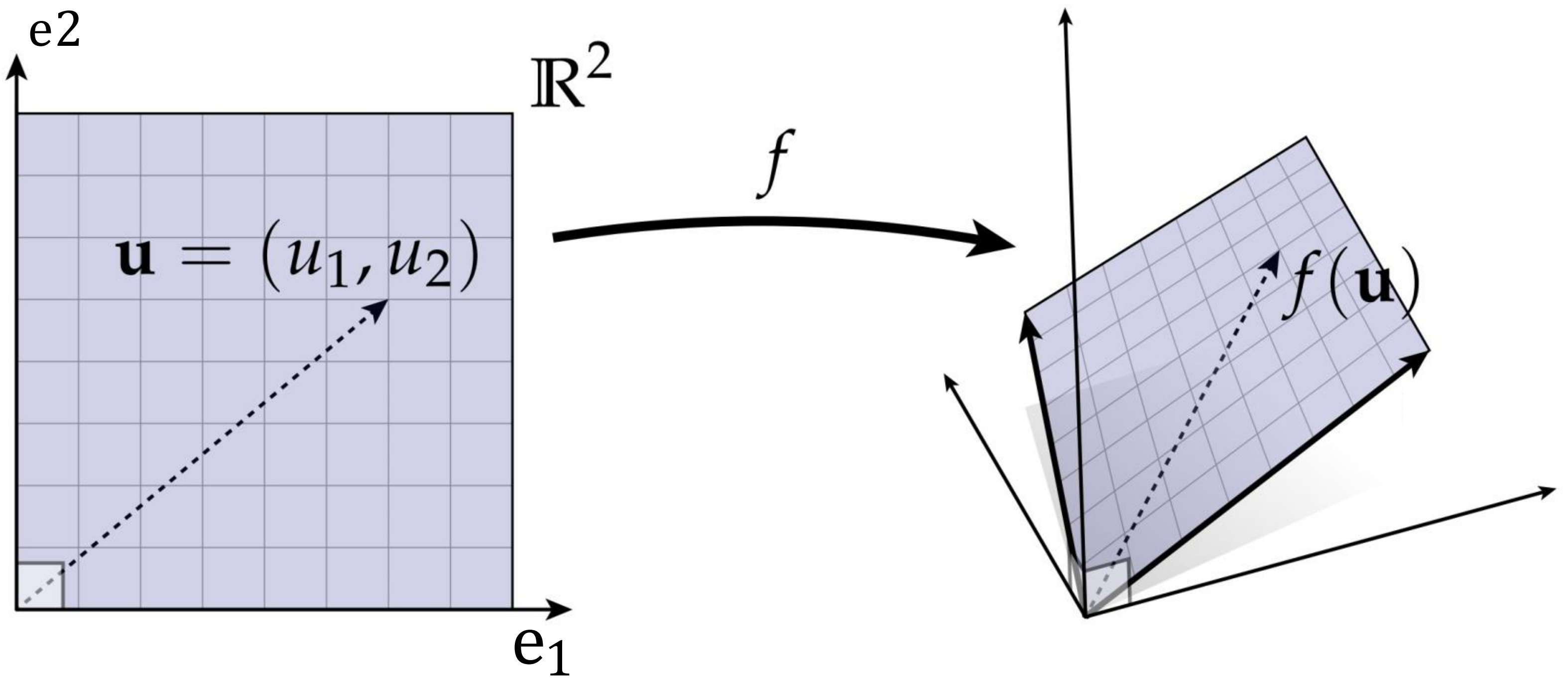
Slanty cube moved down and to the left

f transforms x to f(x)



And what is our favorite type of transformation?

Linear transforms



- **But what does it mean?**

Linear transforms

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

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

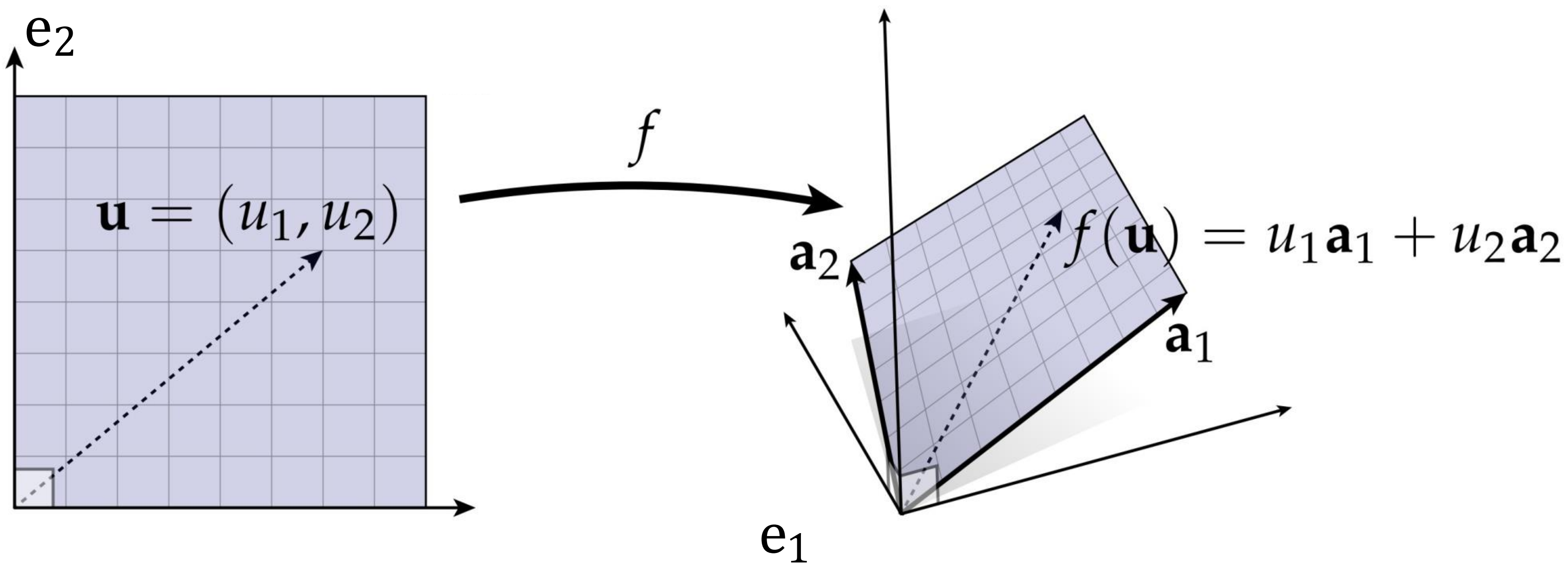
Linear transforms

If a map can be expressed as

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

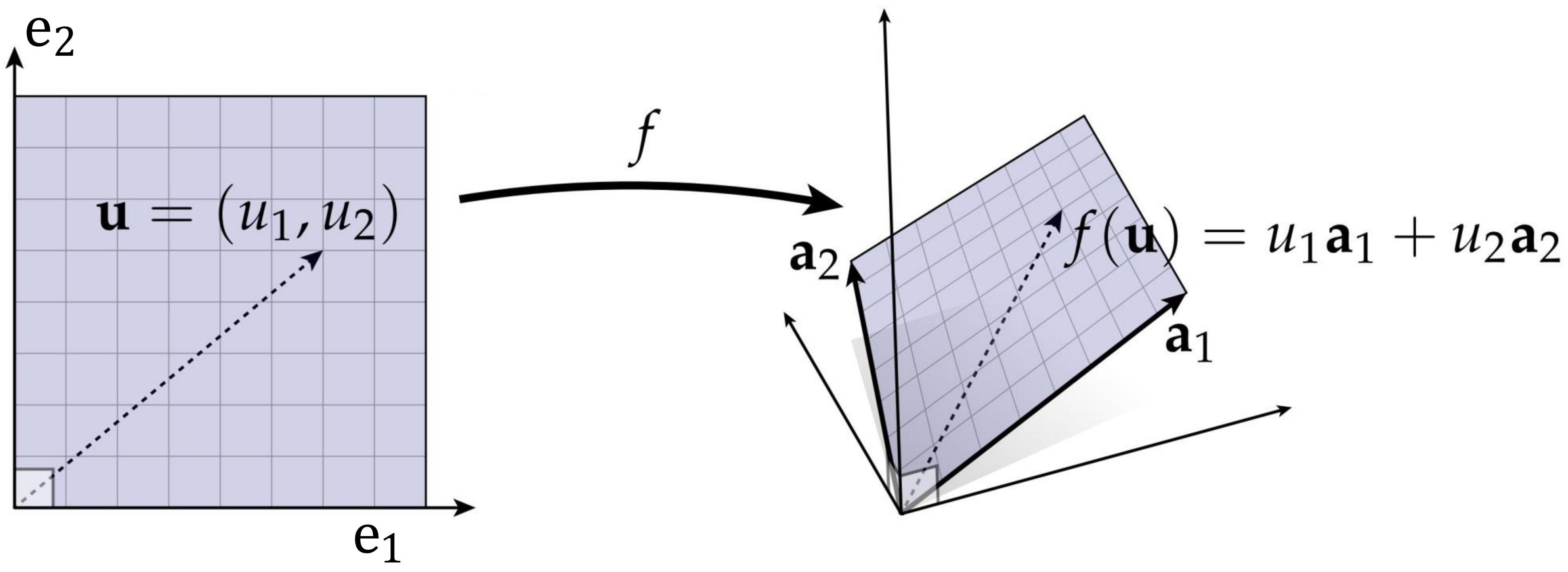
with fixed vectors a_i, then it is linear

Linear transforms


e_2
f
$\mathbf{u} = (u_1, u_2)$
$\mathbf{a}_2$
$f(\mathbf{u}) = u_1\mathbf{a}_1 + u_2\mathbf{a}_2$
$\mathbf{a}_1$
e_1

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

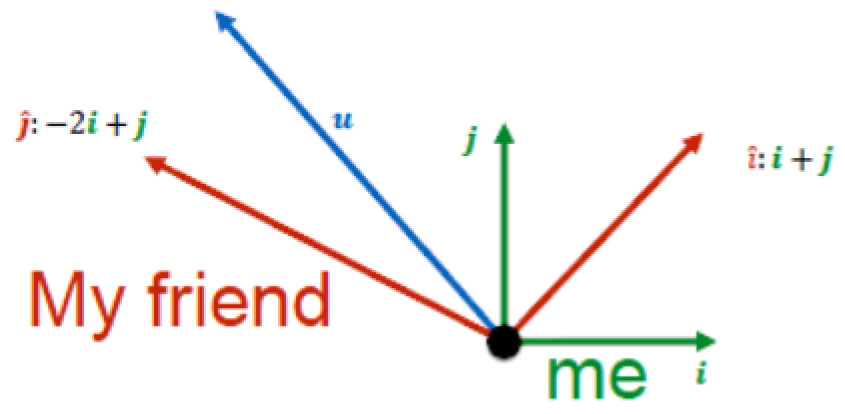
Linear transforms



e_2
$\mathbf{u} = (u_1, u_2)$
e_1
f
$\mathbf{a}_2$
$f(\mathbf{u}) = u_1\mathbf{a}_1 + u_2\mathbf{a}_2$
$\mathbf{a}_1$

- $\boldsymbol{u}$ 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!

An example: Coordinate transformations



My friend says, look at 3 o'clock (in their coordinate frame that means one "forward" and one to the "right")!

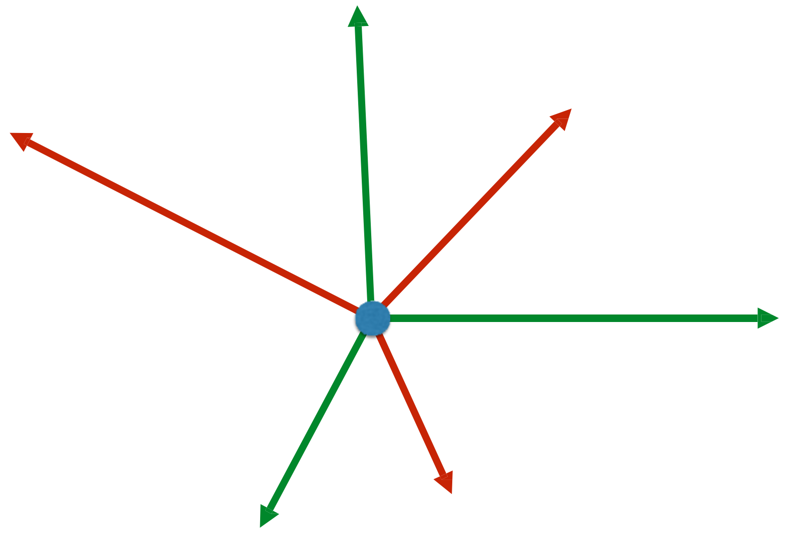
Where should I look?

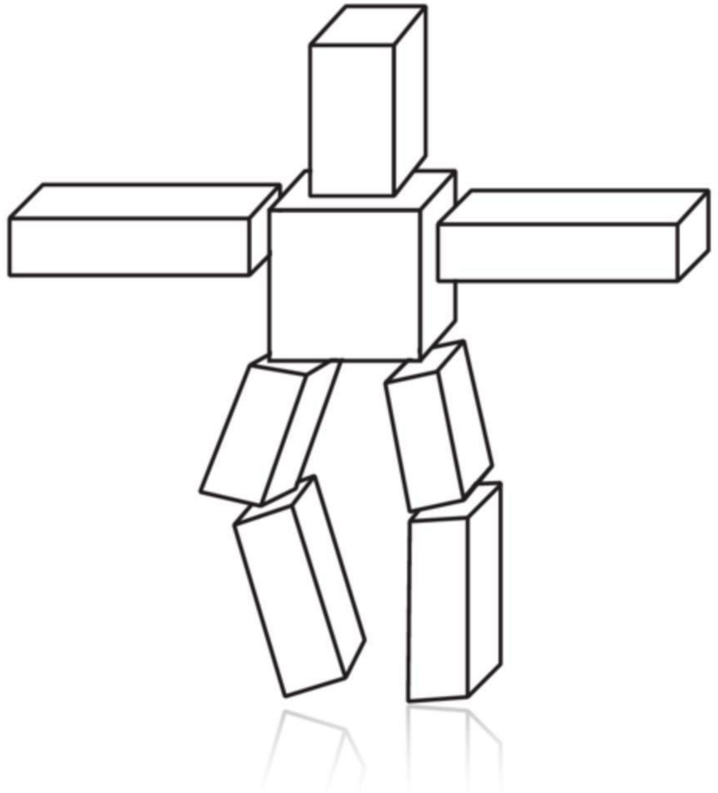
Direction in my friend's coordinate frame

$$\underbrace{f(\boldsymbol{u})}_{\text{Same direction in my coordinate frame}} = f(u_1\hat{\imath} + u_2\hat{\jmath}) = u_1 f(\hat{\imath}) + u_2 f(\hat{\jmath}) = u_1 \begin{bmatrix} 1 \\ 1 \end{bmatrix} + u_2 \begin{bmatrix} -2 \\ 1 \end{bmatrix}$$

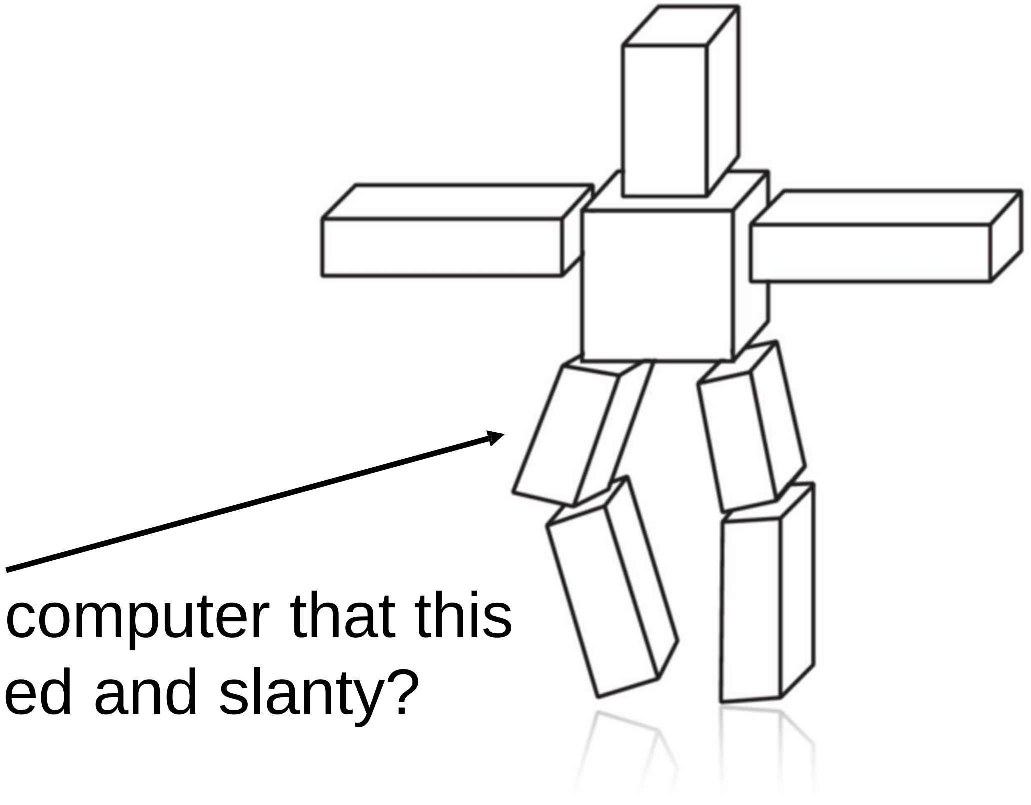
Linear maps

- In graphics we often talk about changing coordinate frames (go from local to world to camera to screen coordinates)
- Equally useful to think about **maps transforming a space (and everything in it!)**



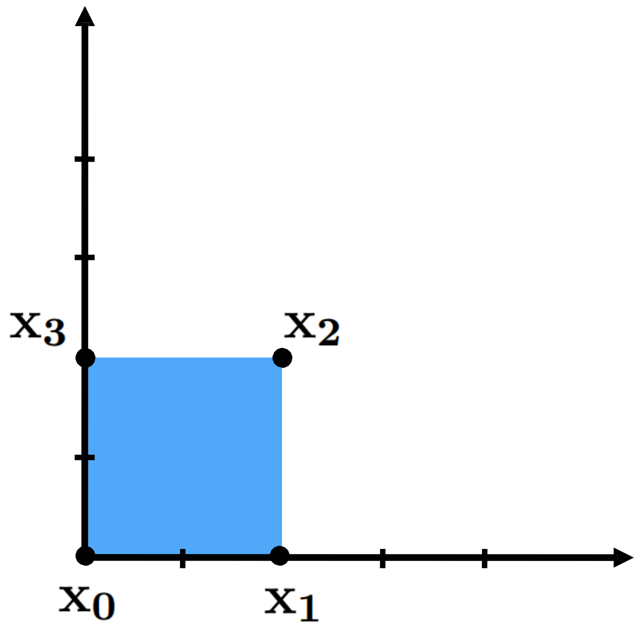


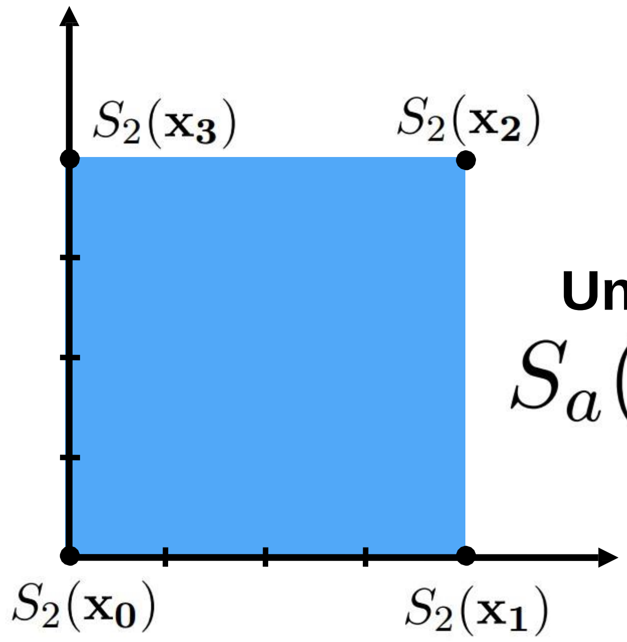
Let’s look at some transforms that are important in graphics…



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

Scale



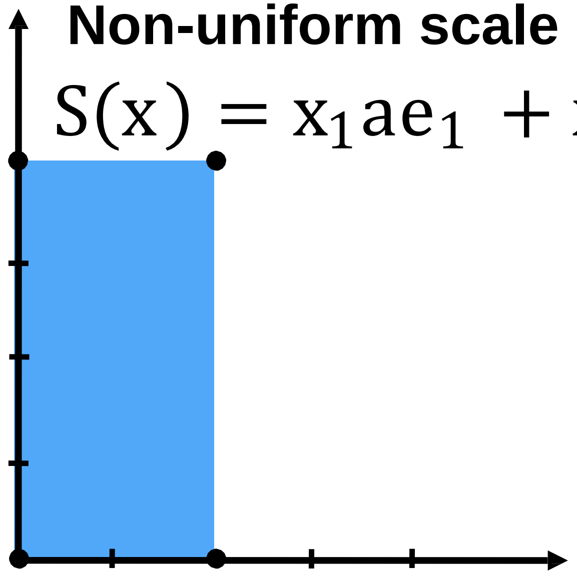


Uniform scale:

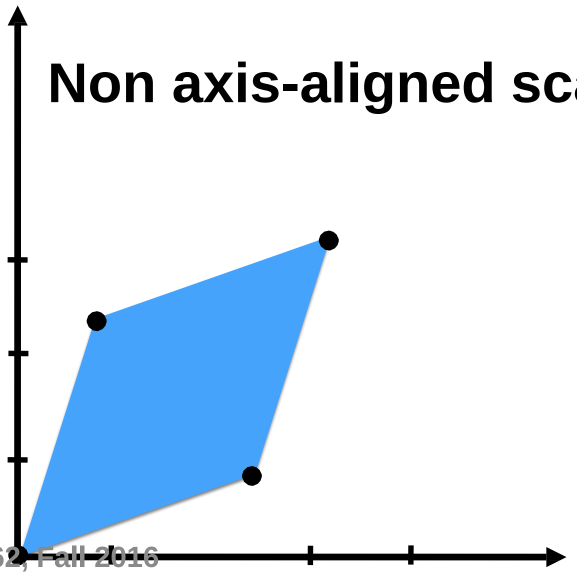
$$S_a(\mathbf{x}) = a\mathbf{x}$$

Non-uniform scale

$$S(x) = x_1 a e_1 + x_2 b e_2$$

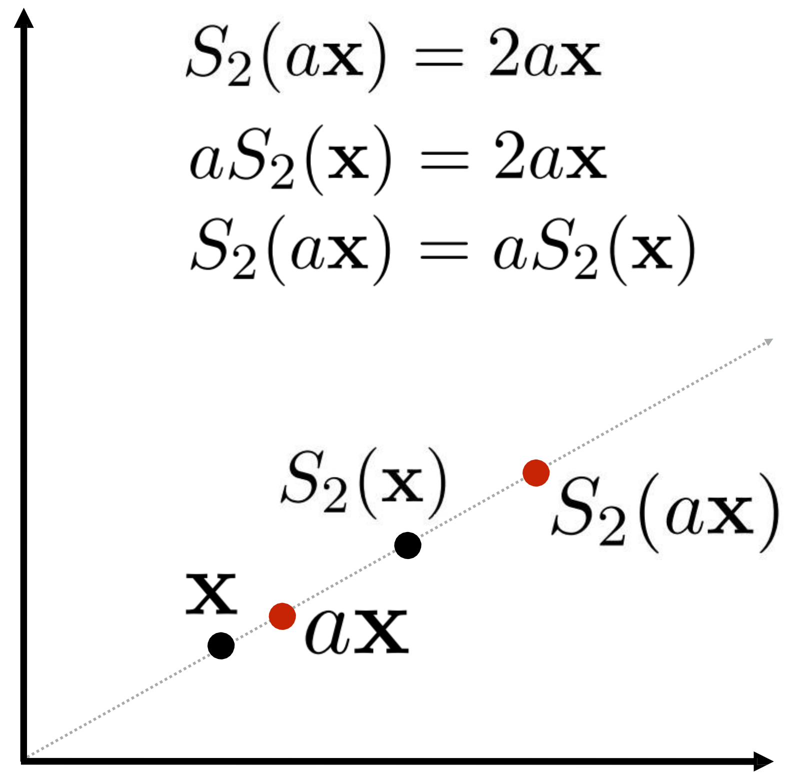


Non axis-aligned scaling??

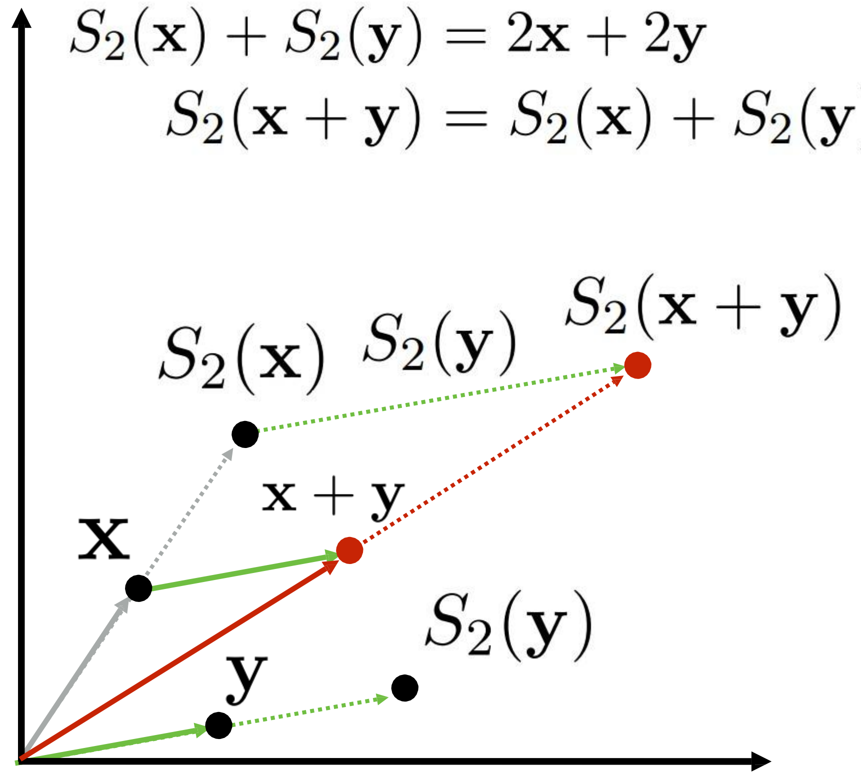


Is uniform scale a linear transform?

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

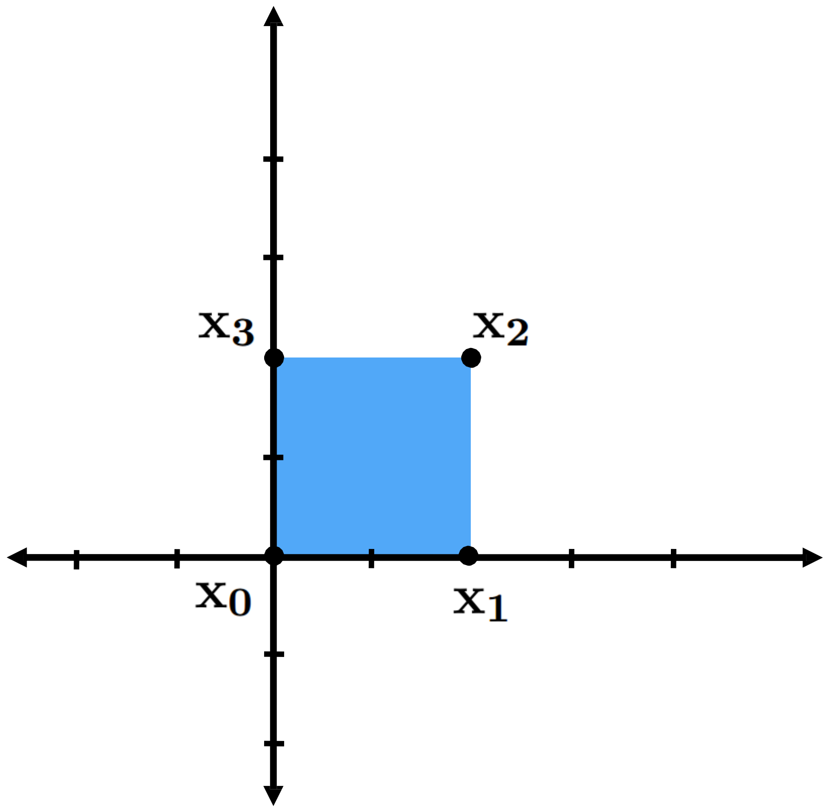


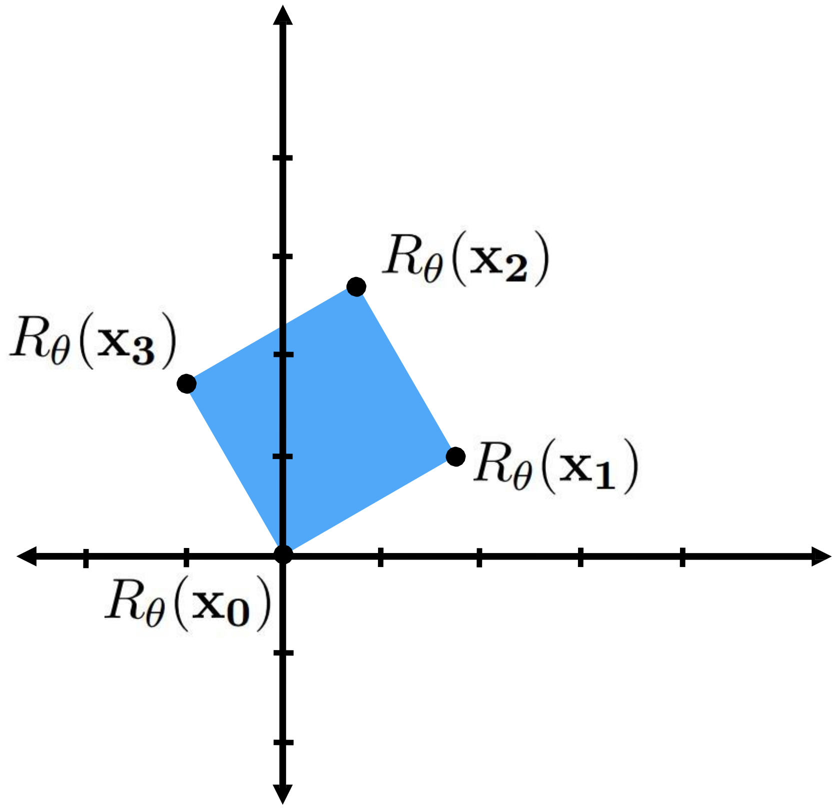
$$S_2(\mathbf{x}+\mathbf{y}) = 2(\mathbf{x}+\mathbf{y})$$
$$S_2(\mathbf{x}) + S_2(\mathbf{y}) = 2\mathbf{x} + 2\mathbf{y}$$
$$S_2(\mathbf{x}+\mathbf{y}) = S_2(\mathbf{x}) + S_2(\mathbf{y})$$



Yes!

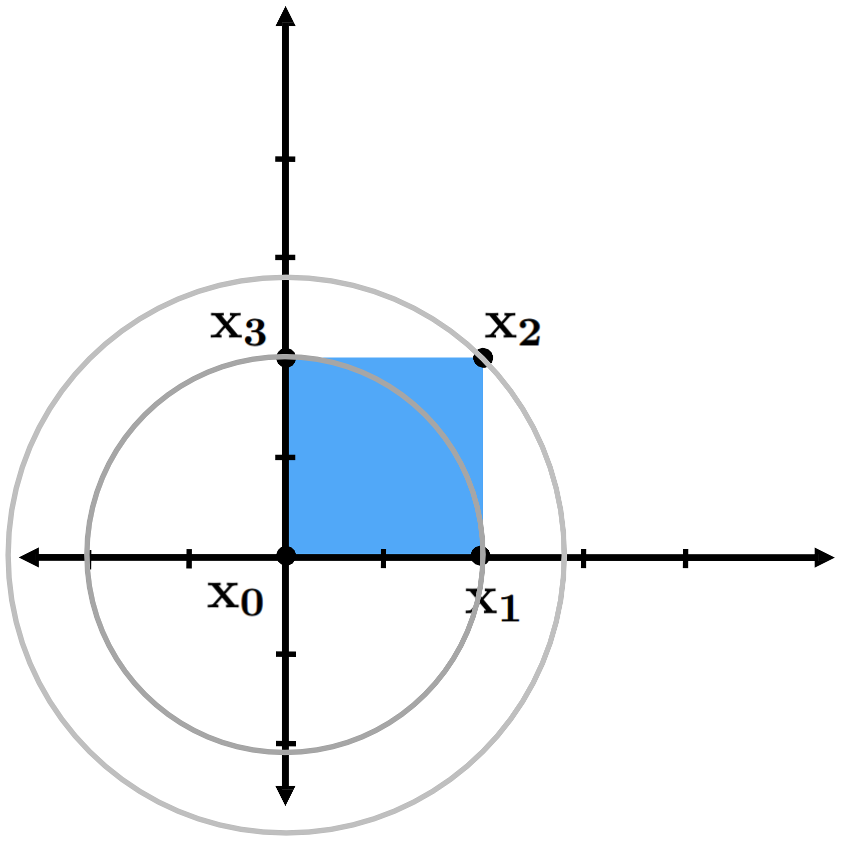
Rotation

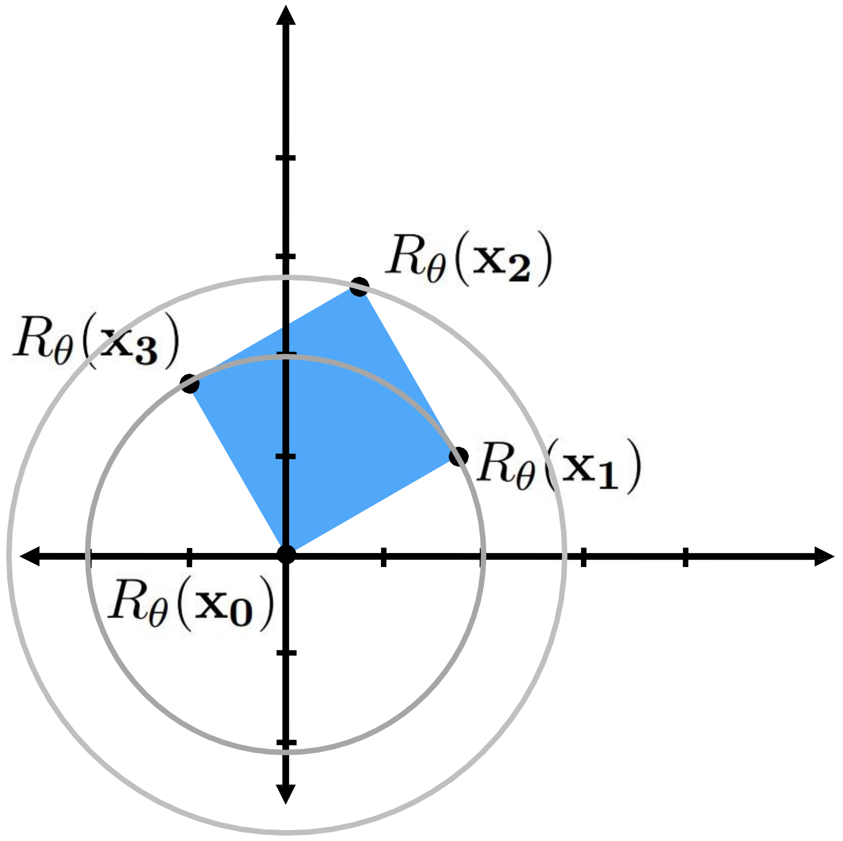




R_θ **= rotate counter-clockwise by** θ

Rotation

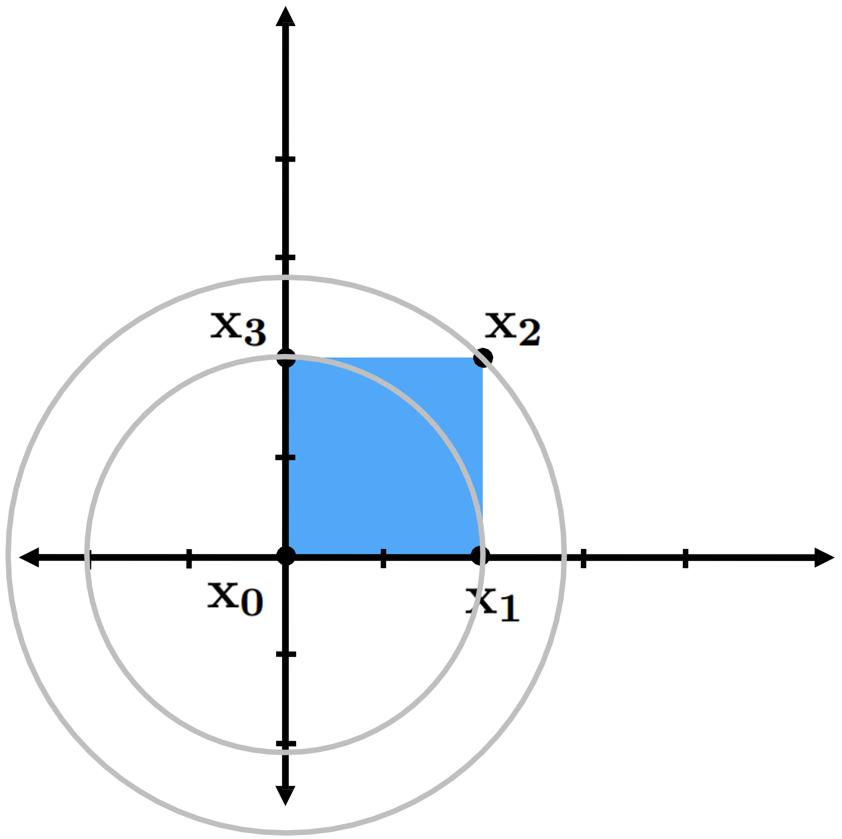


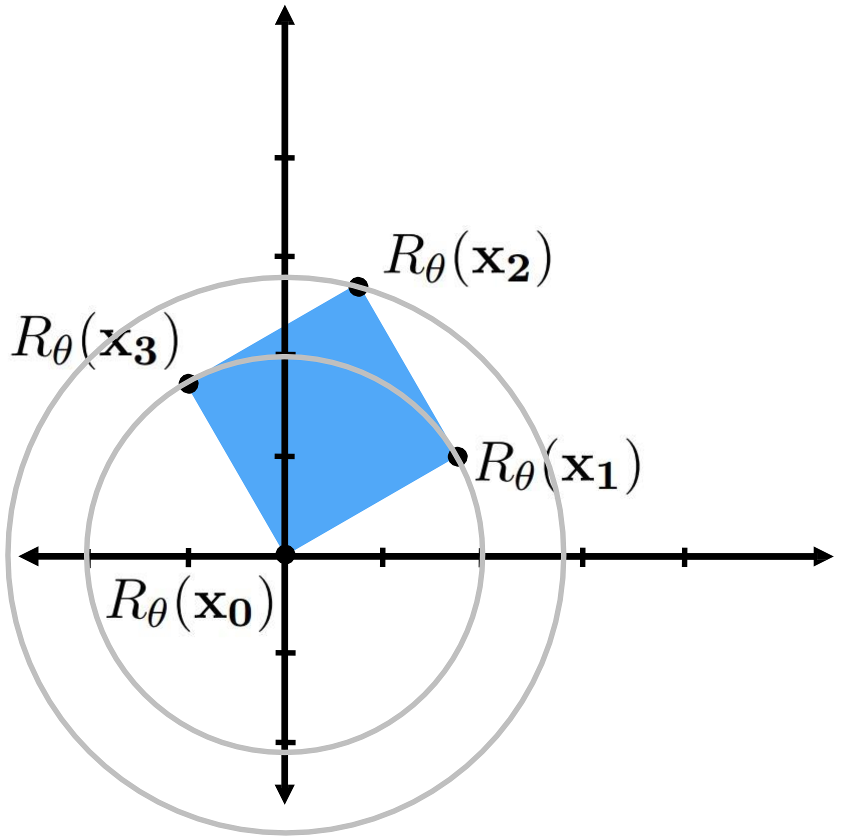


R_θ = **rotate counter-clockwise by** θ

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

Rotation





R_θ = rotate counter-clockwise by θ

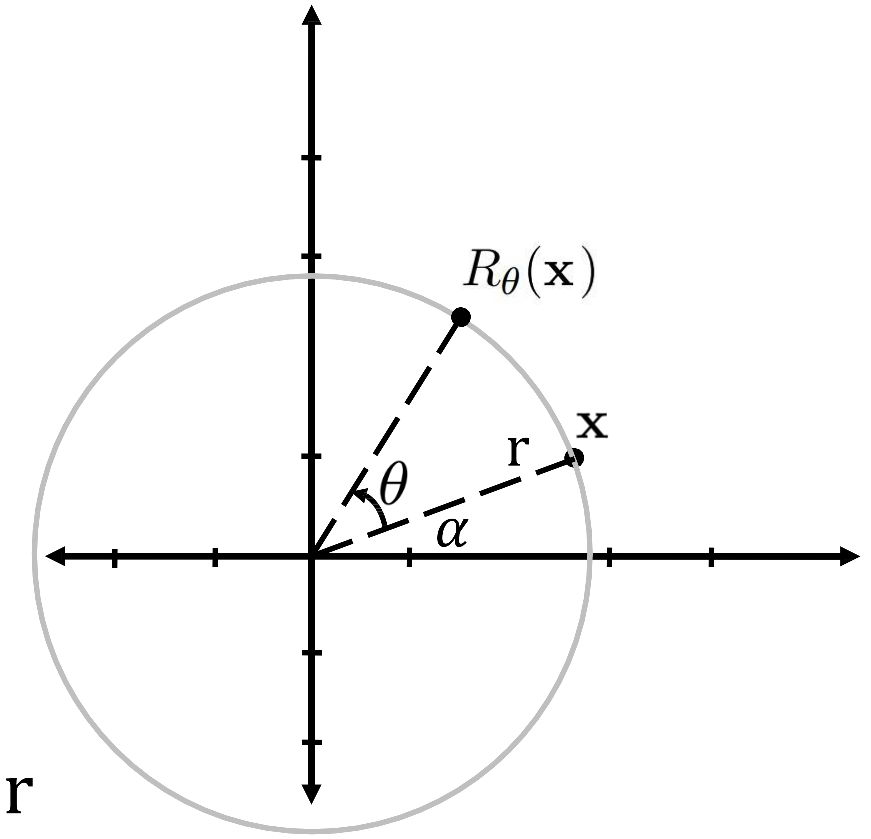
As angle changes, points move along *circular* trajectories.

Shape (distance between any two points) does not change!

(Rigid or isometric transformation)

Rotation

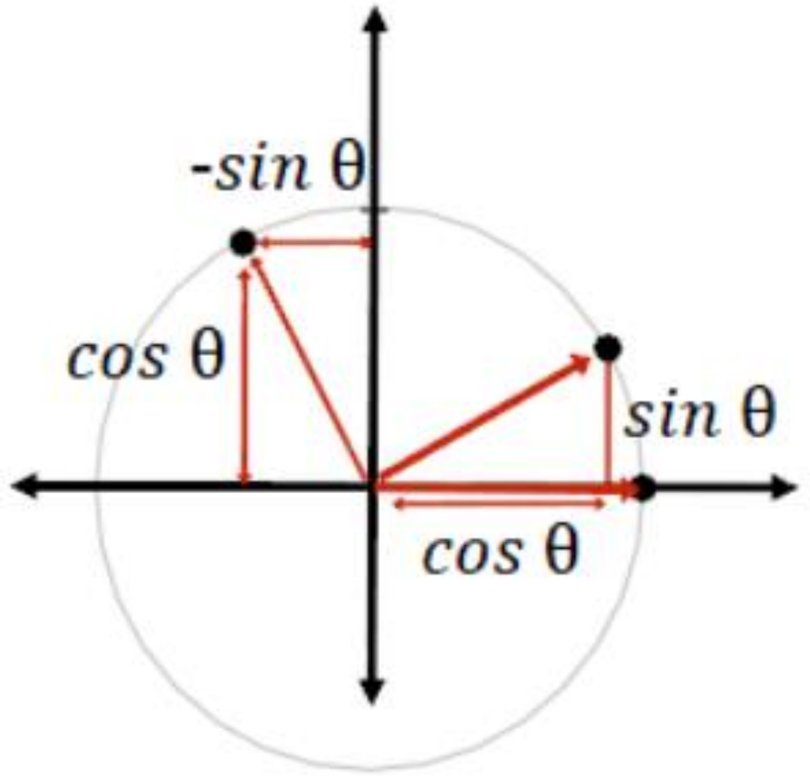
What does R_θ look like?



- **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!**

Rotation

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



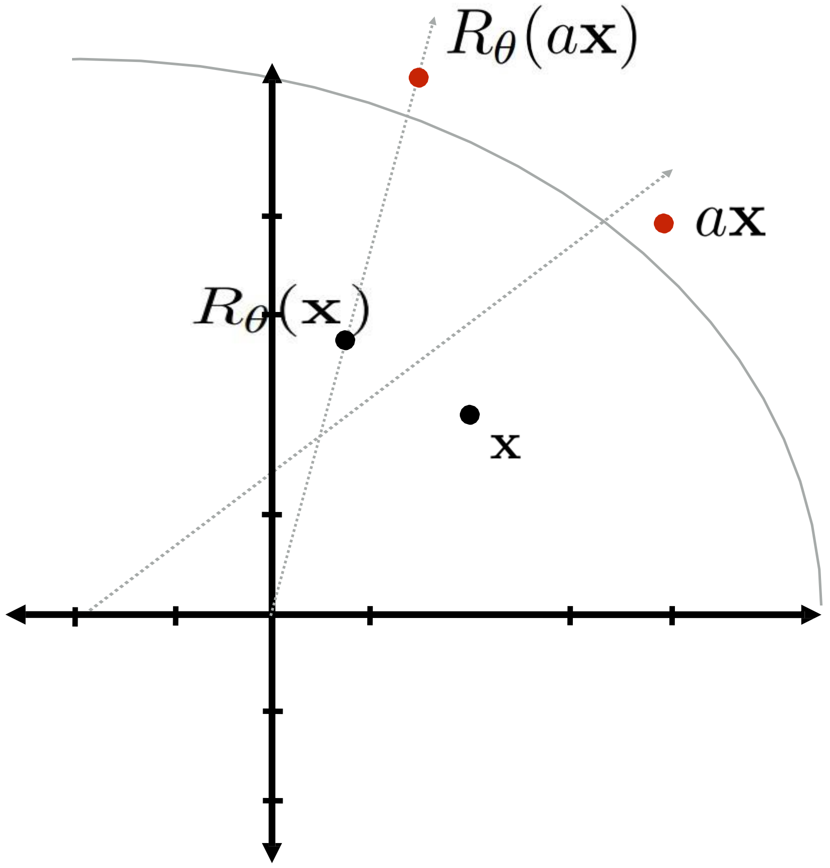
Answer:

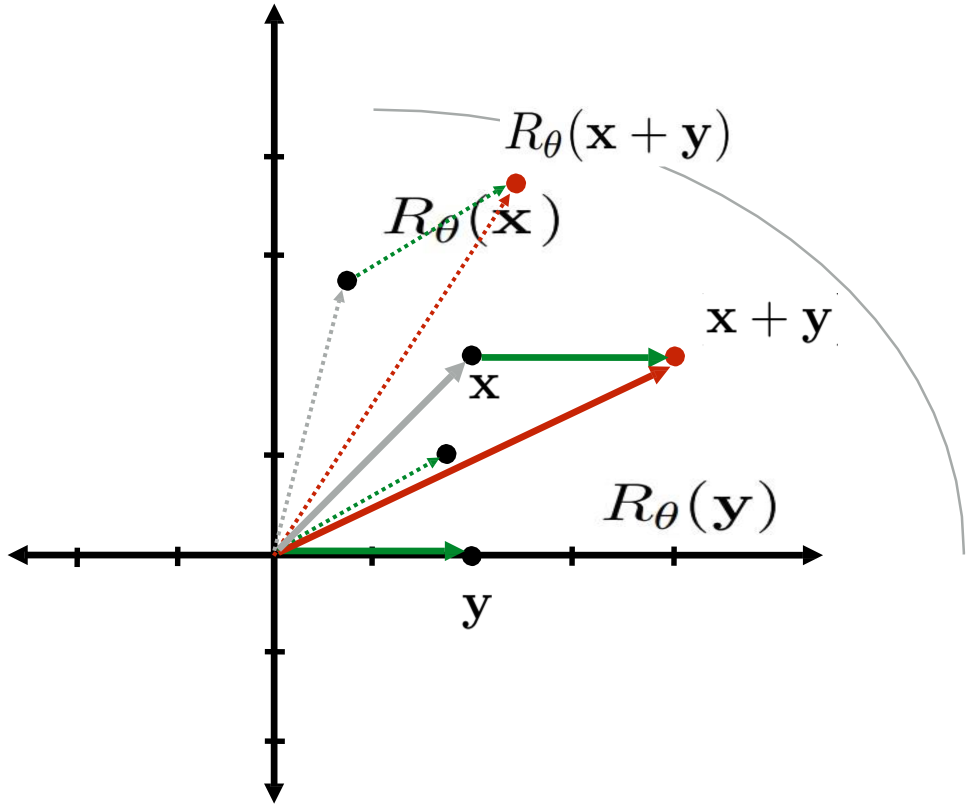
$$R_\theta(\boldsymbol{e}_1) = (\cos\theta, \sin\theta) = \boldsymbol{a}_1$$
$$R_\theta(\boldsymbol{e}_2) = (-\sin\theta, \cos\theta) = \boldsymbol{a}_2$$

So:

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

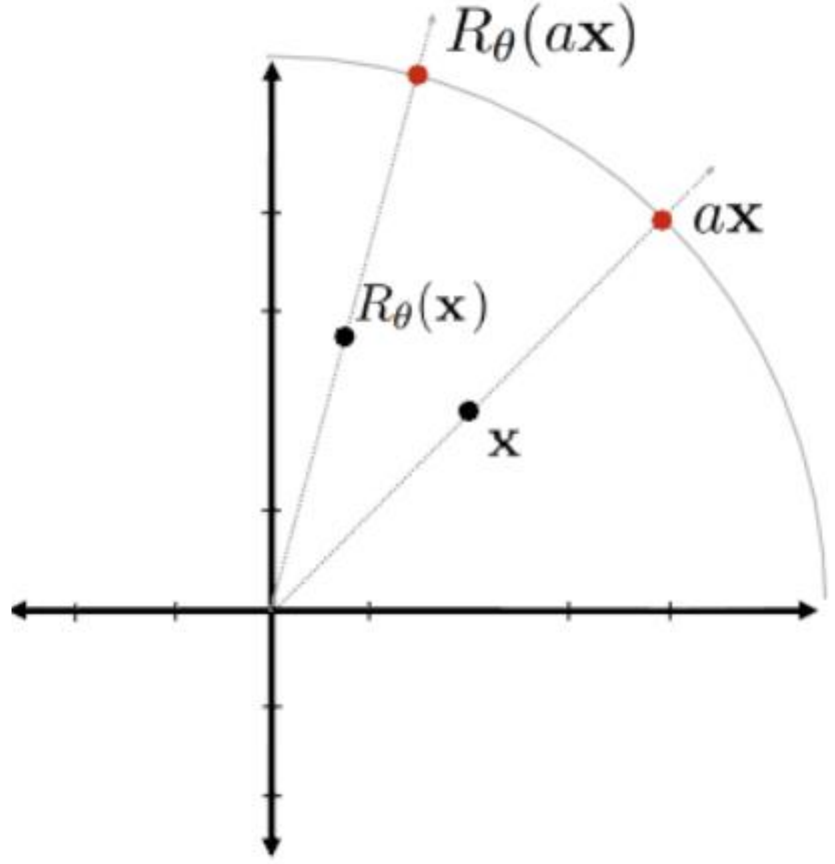
Is rotation linear?

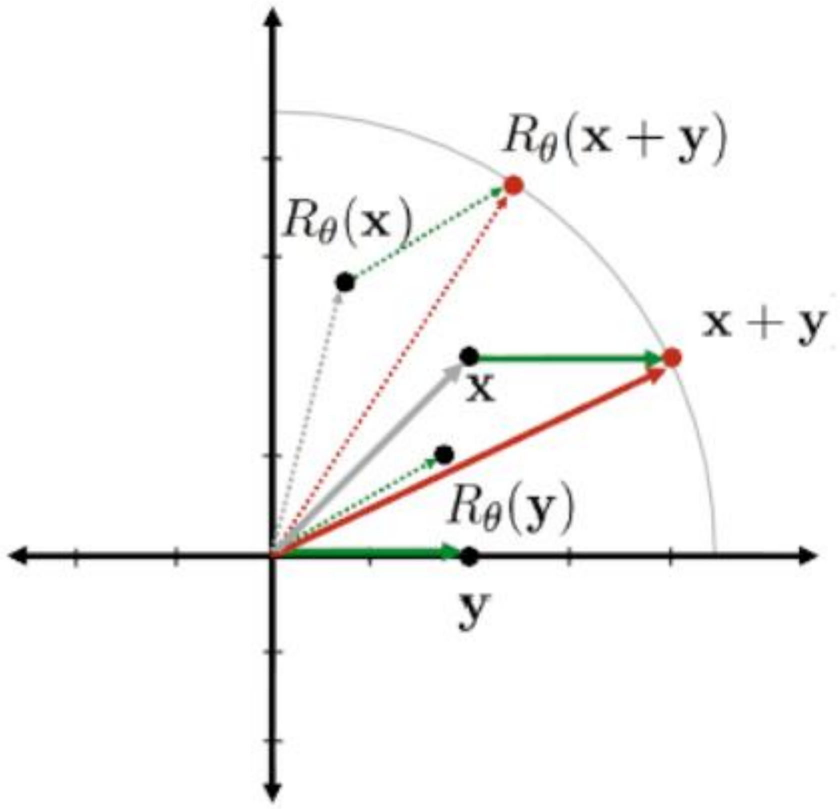




Yes!

Is rotation linear?

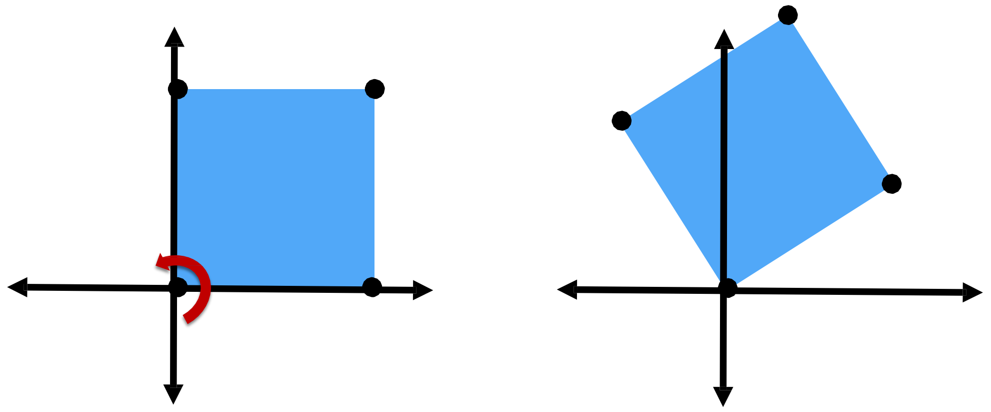


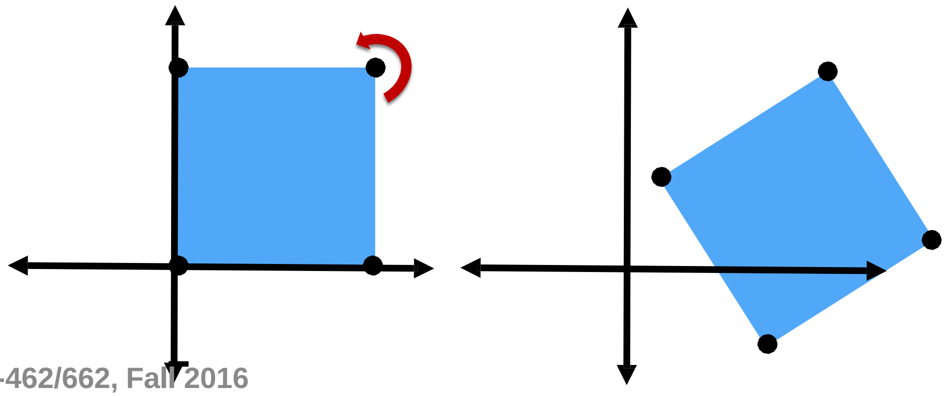


Yes!

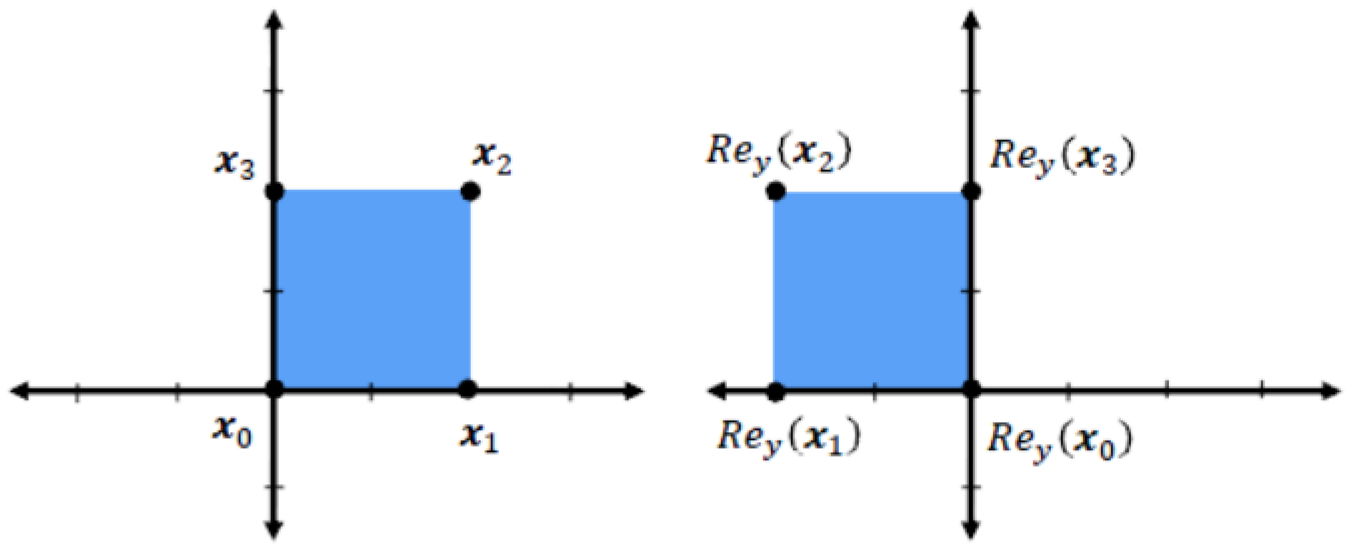
Rotation

- **Note: all points are rotated about the origin**
  - **By the way, what are we actually transforming here?**
- **What if we want to rotate about another point?**





Reflection



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

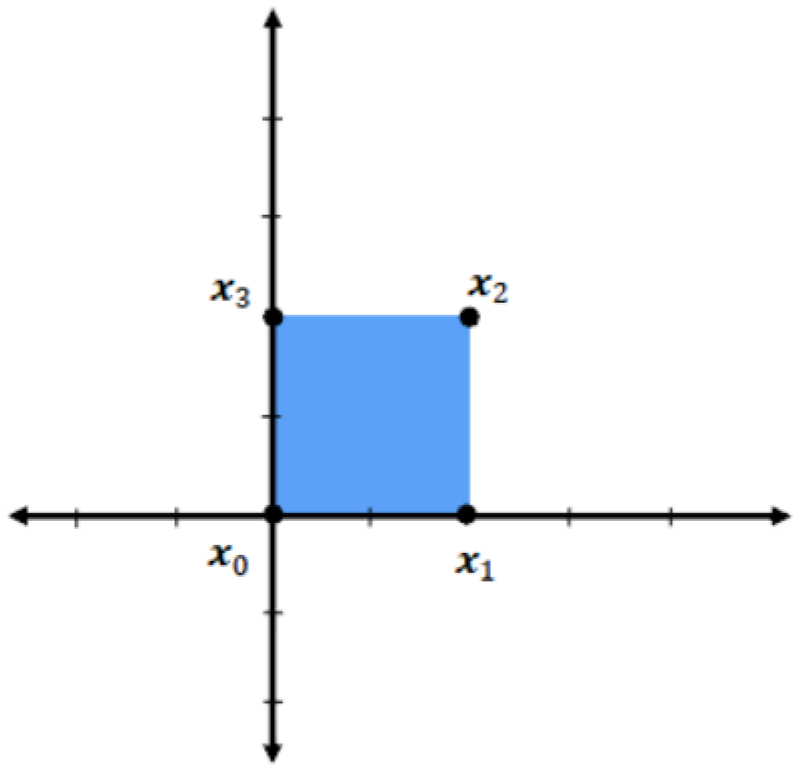
Reflections change "handedness"...

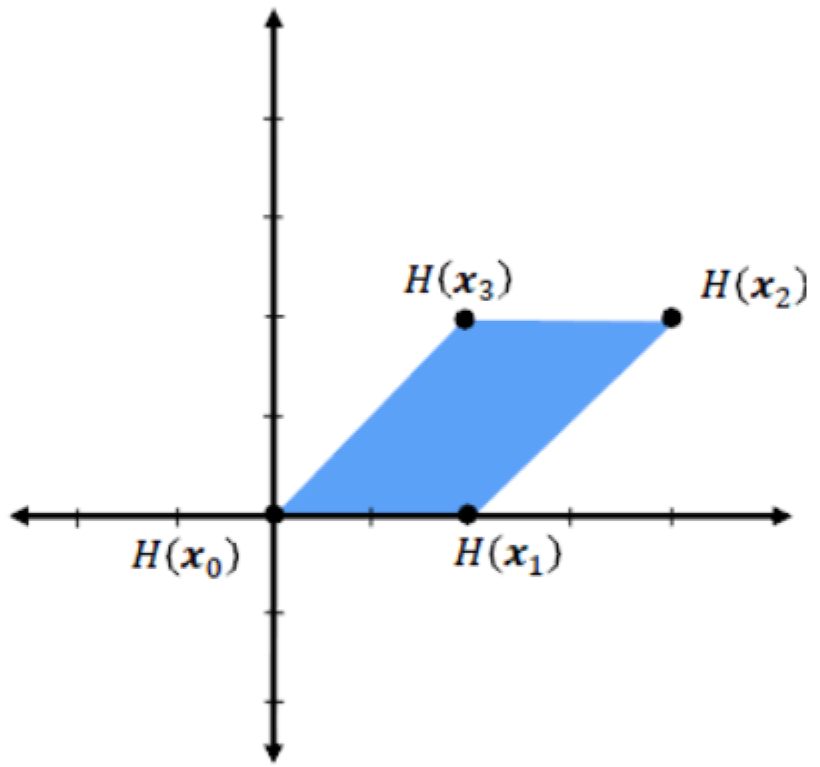
Do you know what $Re_y(\boldsymbol{x})$ looks like?

Is reflection a linear transform?

Do you know how to reflect about an arbitrary axis?

Shear (in x direction)



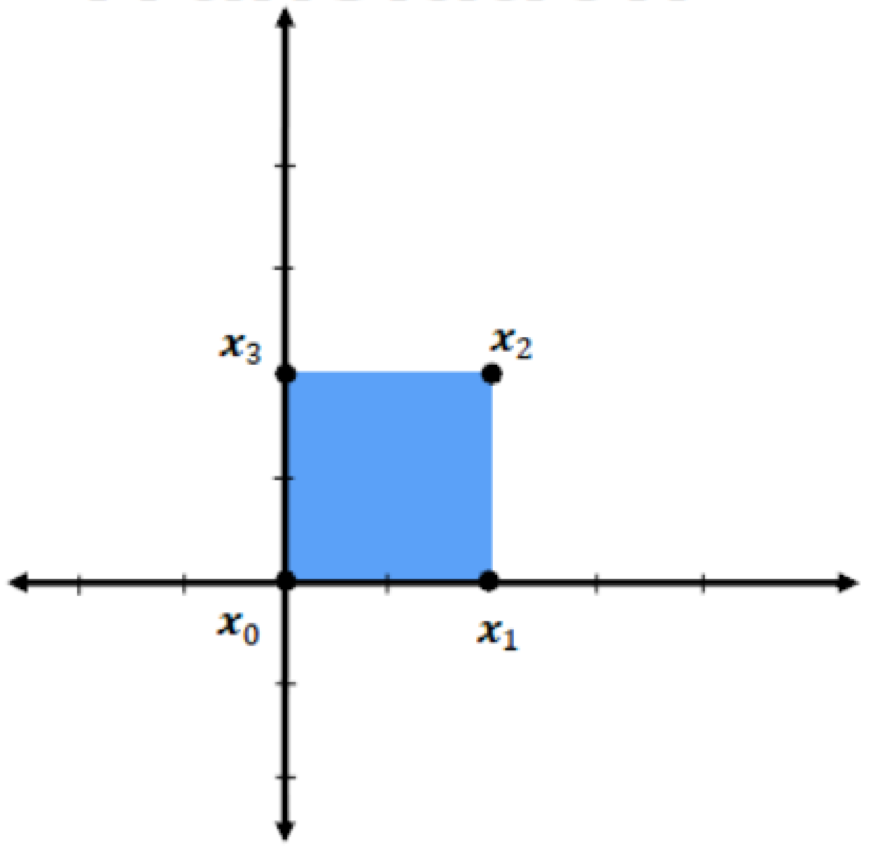


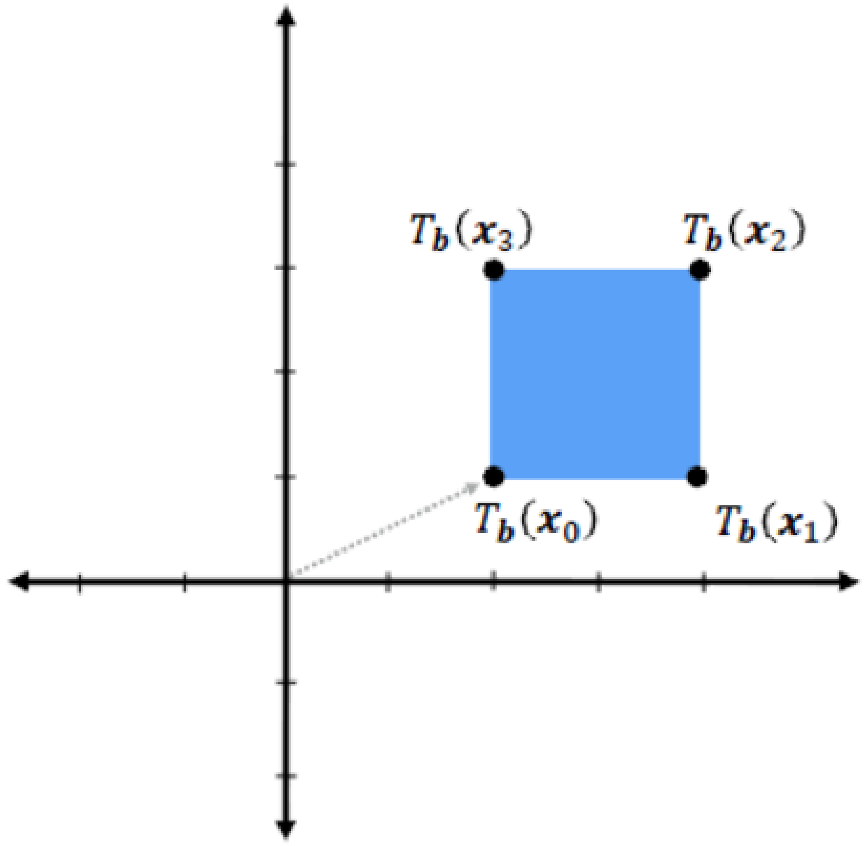
What does $H(\boldsymbol{x})$ look like?

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

Is shearing a linear transformation?

Translation



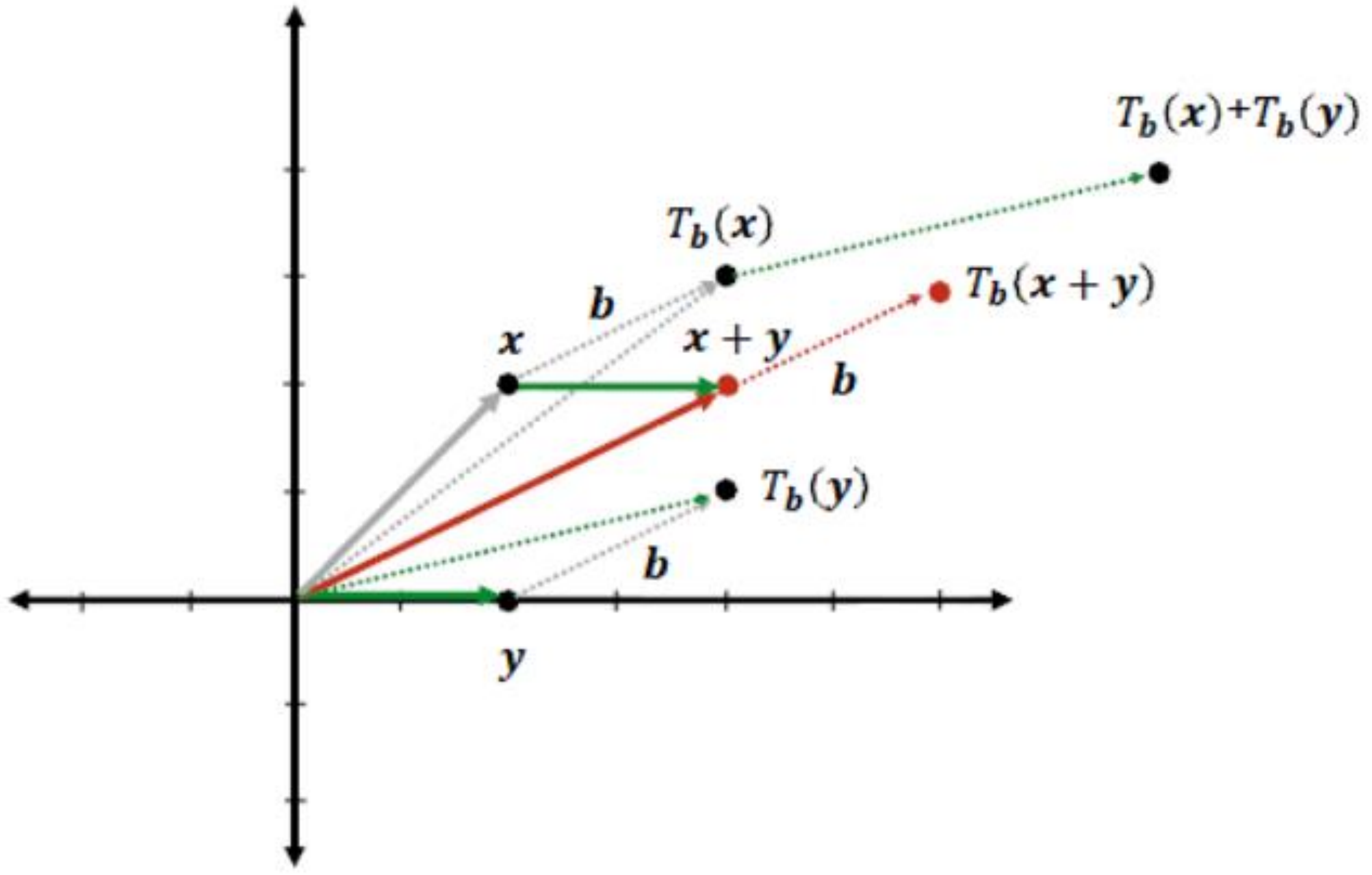


Let's write $T_b(\boldsymbol{x})$ in the form

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

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

Is translation linear?



No. Translation is affine.

Summary of basic transforms

Linear:

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

Scale
Rotation
Reflection
Shear

Not linear:

Translation

Affine:

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

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

Not affine: perspective projection (will discuss later)

Euclidean: (Isometries)

Preserve distance between points (preserves length)

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

Translation
Rotation
Reflection

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

When at first you don’t succeed…

- We’ll turn affine transformations into linear ones via

  Homogeneous coordinates
  (aka projective coordinates)

- But first, let’s use matrix notation to represent linear transforms

Linear transforms as matrix-vector products

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

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

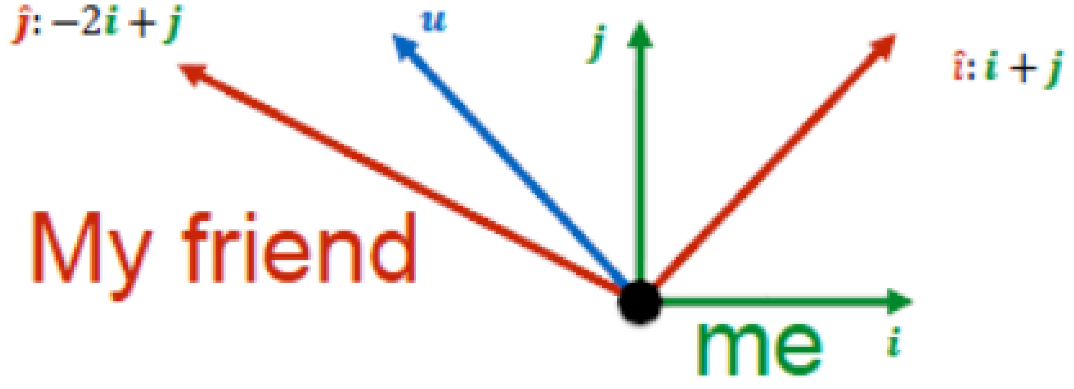
$$f(\boldsymbol{x}) = \sum_{i=1}^{m} x_i \boldsymbol{a}_i = \boldsymbol{Ax}$$

Linear transforms as matrix-vector products

Change of coordinate systems

$$f(x) = x_1 \begin{bmatrix} 1 \\ 1 \end{bmatrix} + x_2 \begin{bmatrix} -2 \\ 1 \end{bmatrix}$$

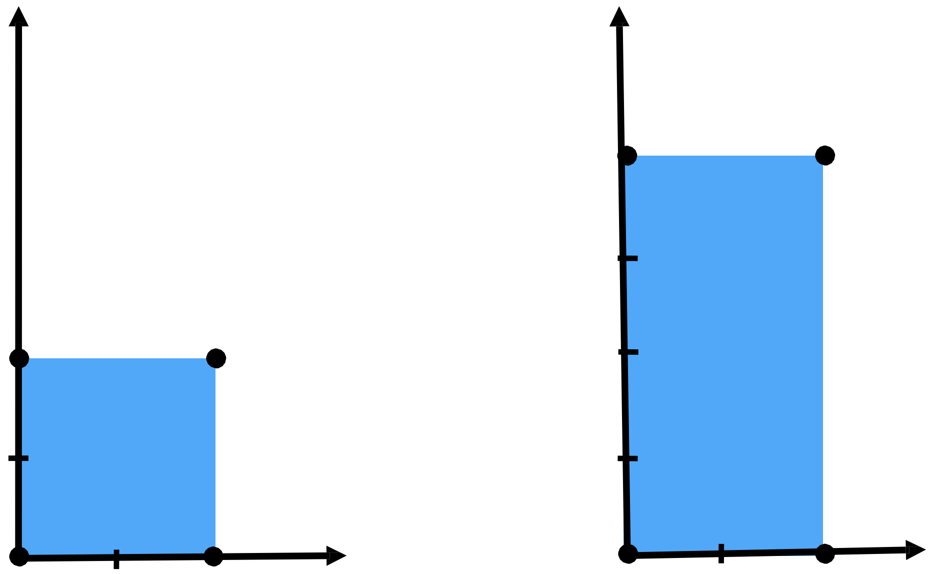
$$= \begin{bmatrix} 1 & -2 \\ 1 & 1 \end{bmatrix} x$$



Linear transforms as matrix-vector products

Non-uniform scale

$$S(\boldsymbol{x}) = x_1 a\boldsymbol{e}_1 + x_2 b\boldsymbol{e}_2$$
$$= \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}\boldsymbol{x}$$



Linear transforms as matrix-vector products

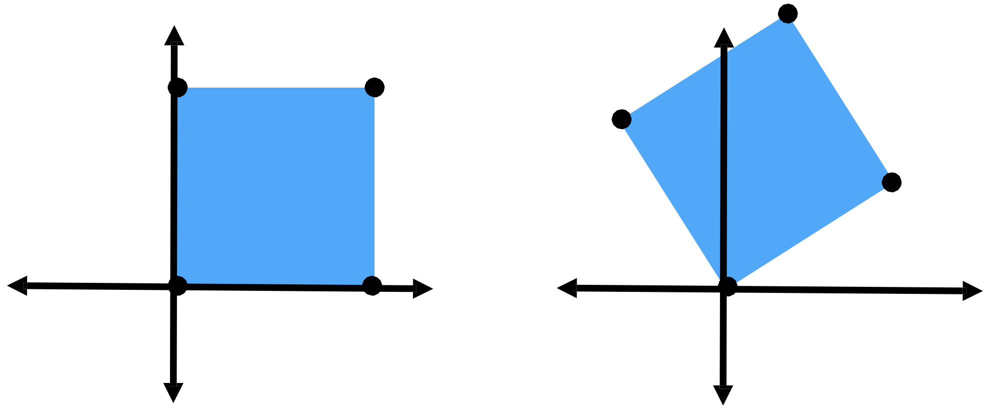
Rotation

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

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

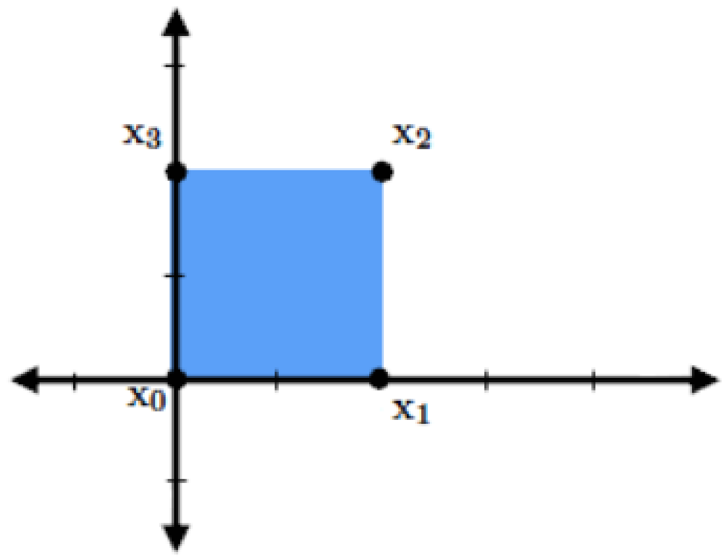
$$R_\theta(\boldsymbol{x}) = x_1\boldsymbol{a}_1 + x_2\boldsymbol{a}_2$$

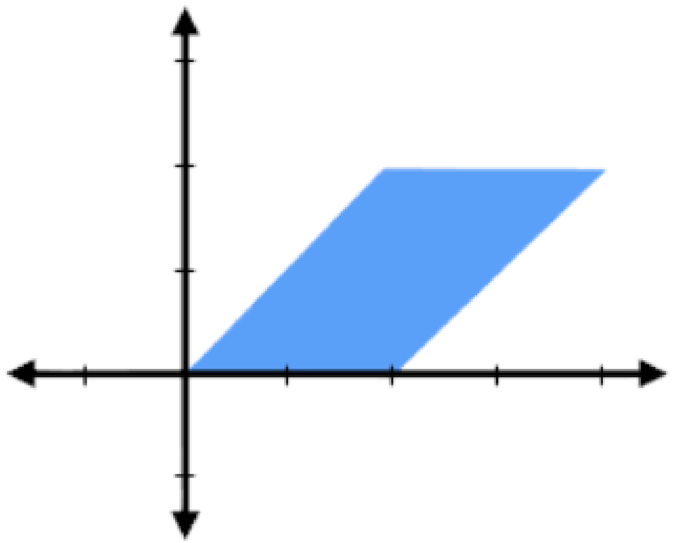
$$= \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \boldsymbol{x}$$



Linear transforms as matrix-vector products

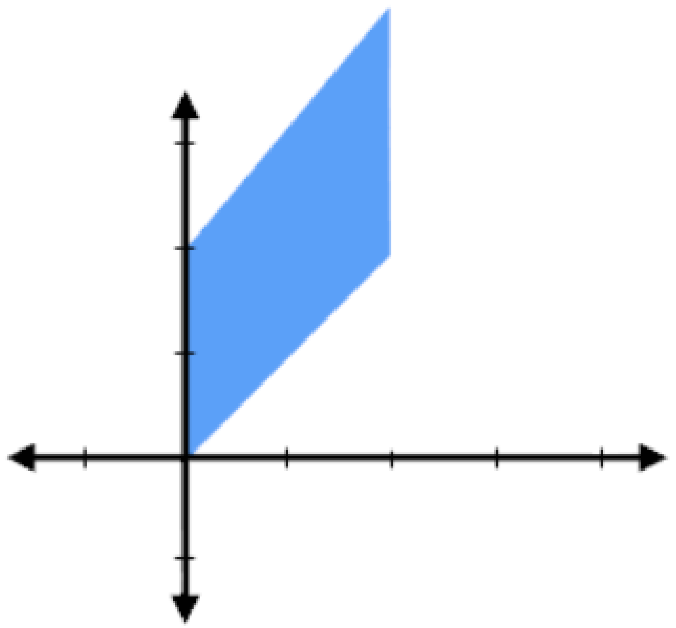
- **Shear**





Shear in x:

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



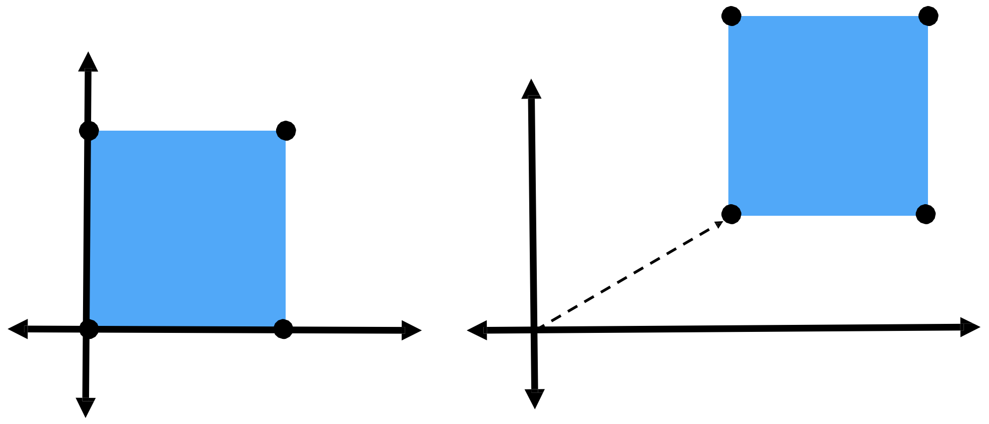
Shear in y:

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

Linear transforms as matrix-vector products

Translation

Not a linear map*…



***when we're using Cartesian coordinates**

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:

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

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

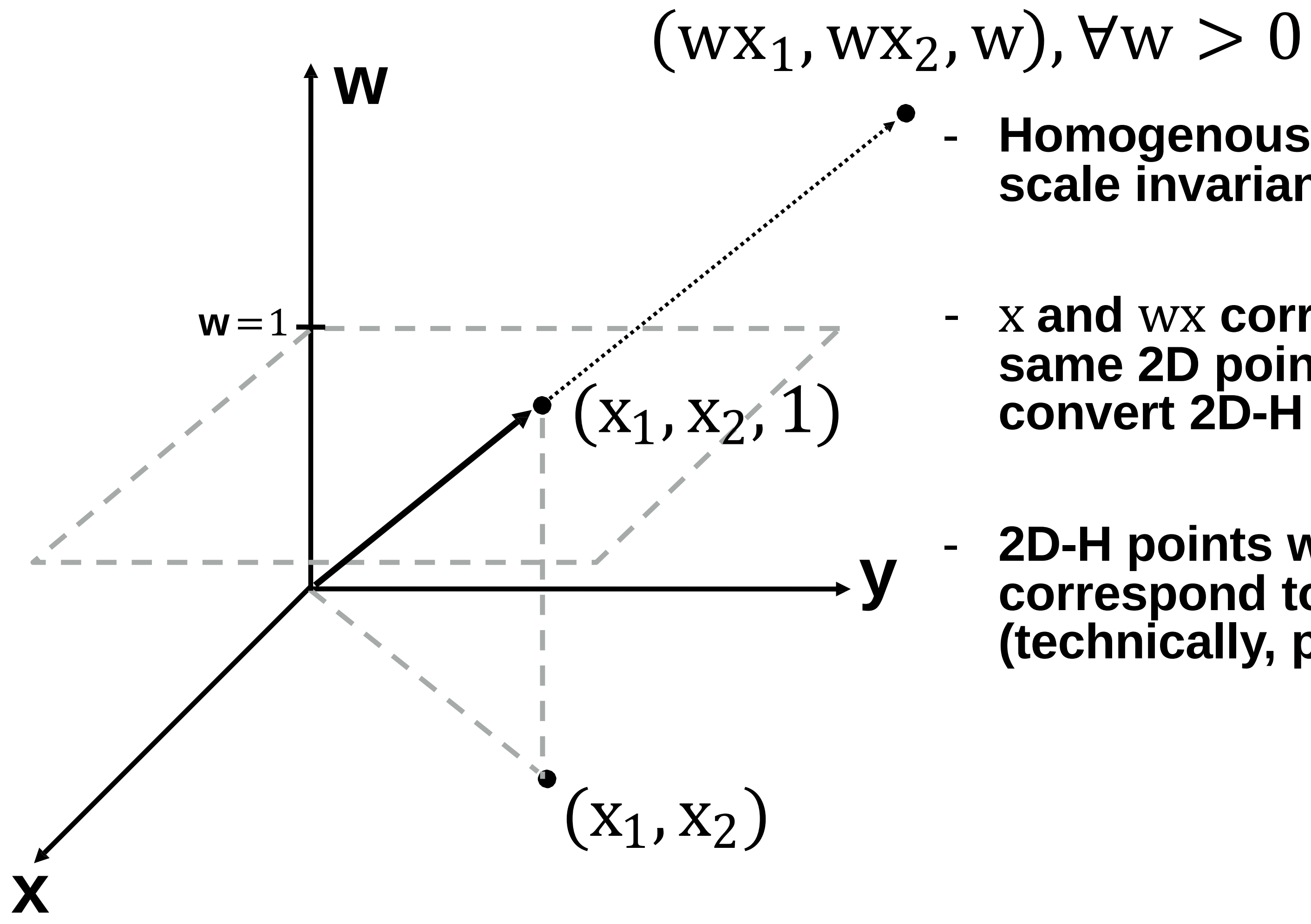
Translation in 2D-H coords

Translation expressed as 3x3 matrix multiplication:

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

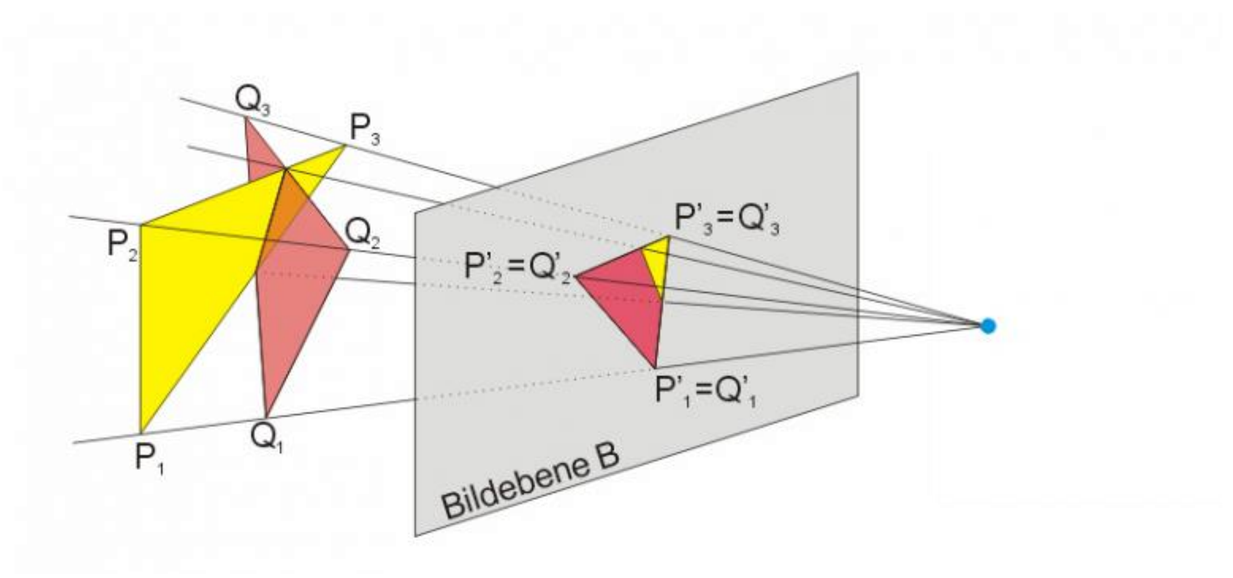
In homogeneous coordinates, translation is a linear transformation!

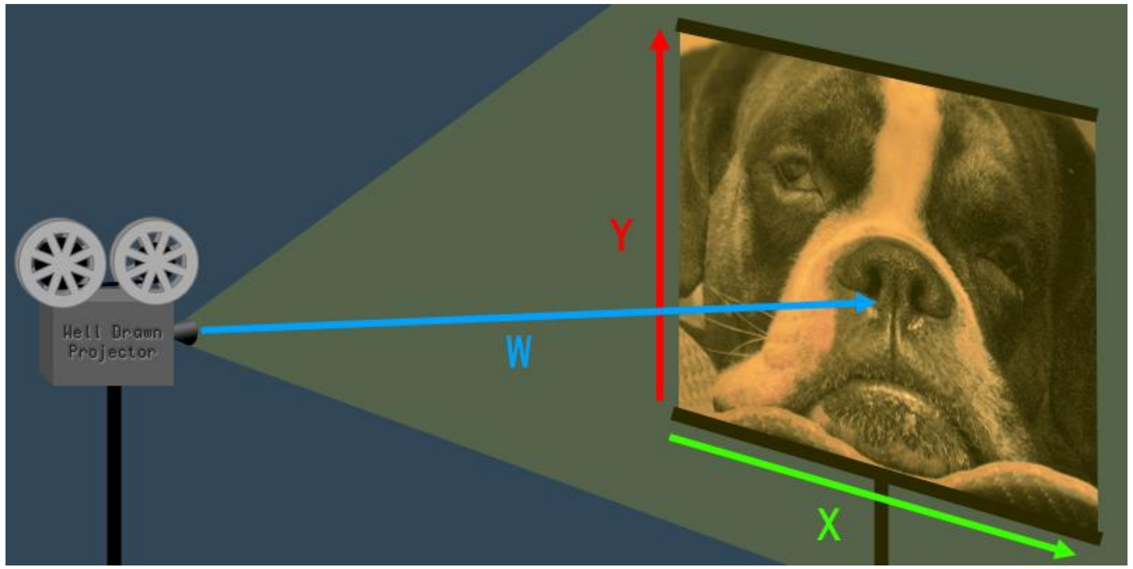
Homogeneous/projective coordinates



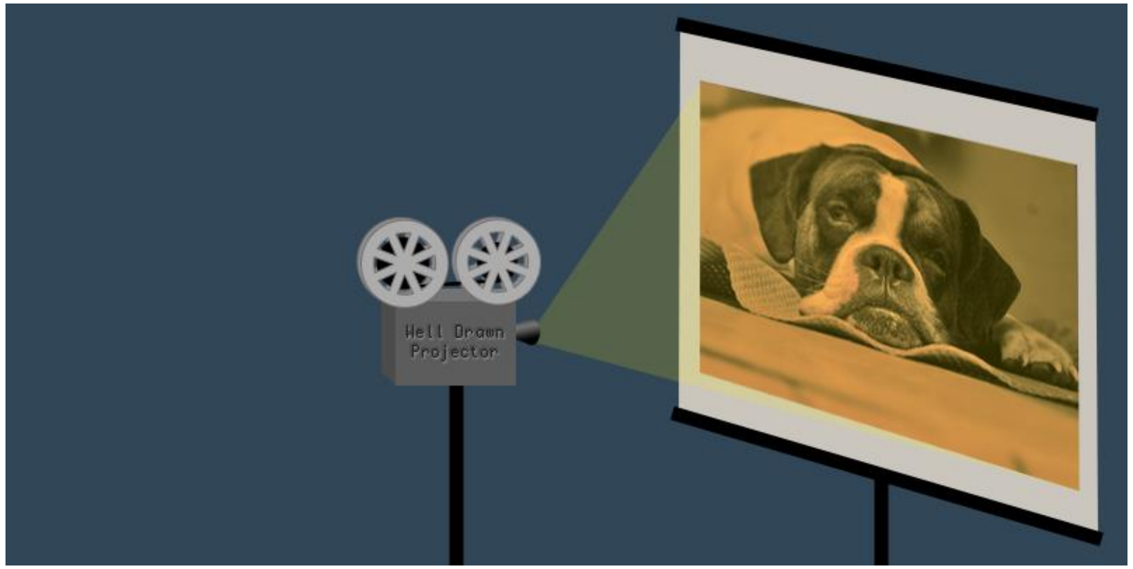
- Homogenous coordinates are scale invariant
- x and wx correspond to the same 2D point (divide by w to convert 2D-H back to 2D)
- 2D-H points with $w = 0$ correspond to 2D vectors (technically, points at infinity)

Homogeneous Coordinates & Projective geometry





The value of WW *affects the size (a.k.a. scale) of the image.*



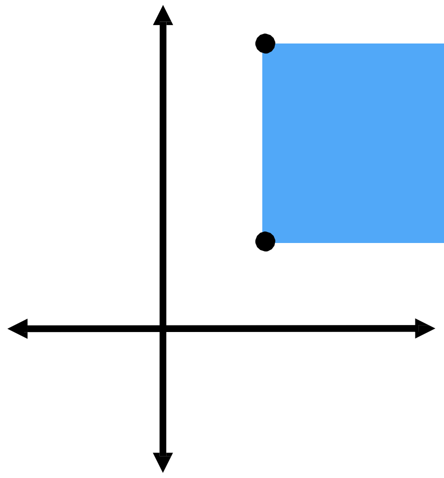
Summary so far…

- We know how to transform (scale, rotate, reflect, shear, translate) 2D points and vectors
  - All these transforms are linear maps
    - expressed as matrix-vector products when
    - using (slightly) higher-dimensional homogenous coordinates
  - How about other types of transforms (e.g. rotate about an arbitrary point)?
  - How about 3D transforms?

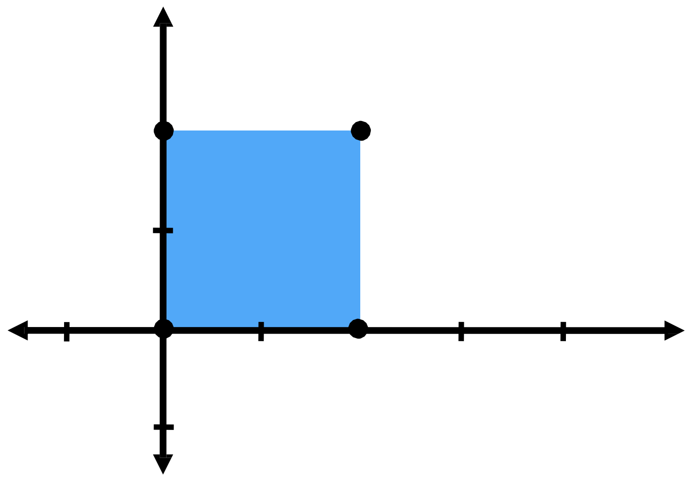
Onto more complex transforms

- How would you transform this object such that it gets twice as large?

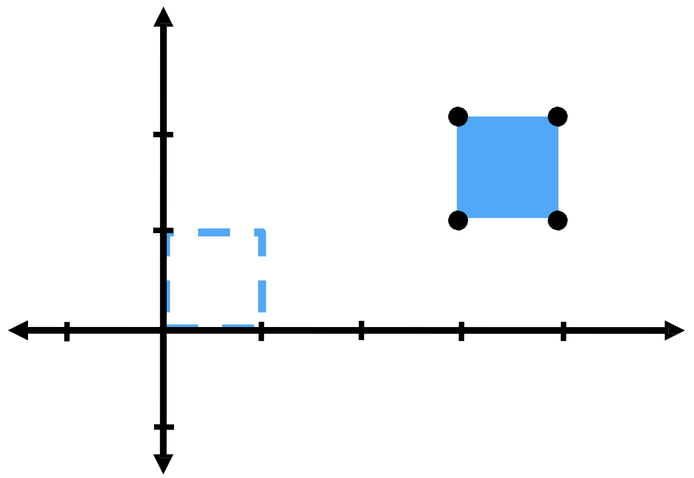
- - but remains where it is…



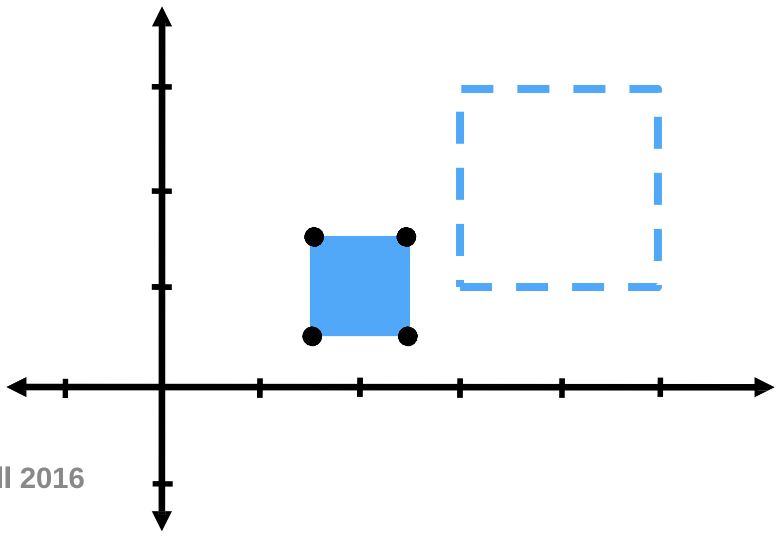
Composition of basic transforms



Scale by 0.5, then translate by (3,1)

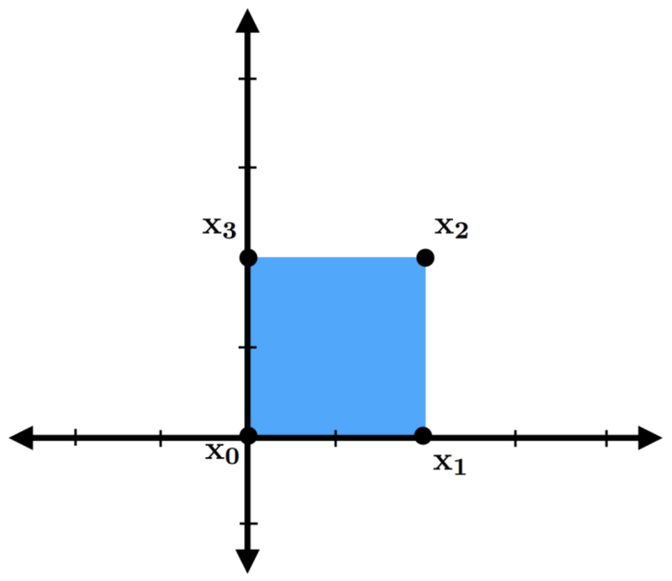


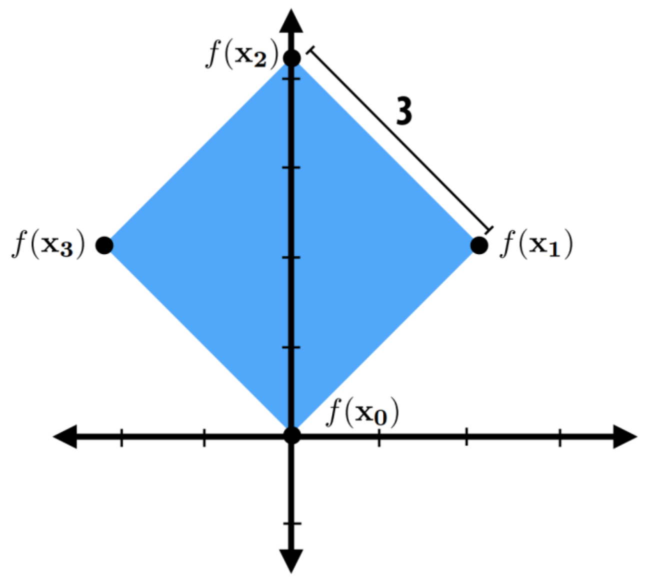
Translate by (3,1), then scale by 0.5



Note 1: order of composition matters!
Note 2: common source of bugs!

How do we compose linear transforms?



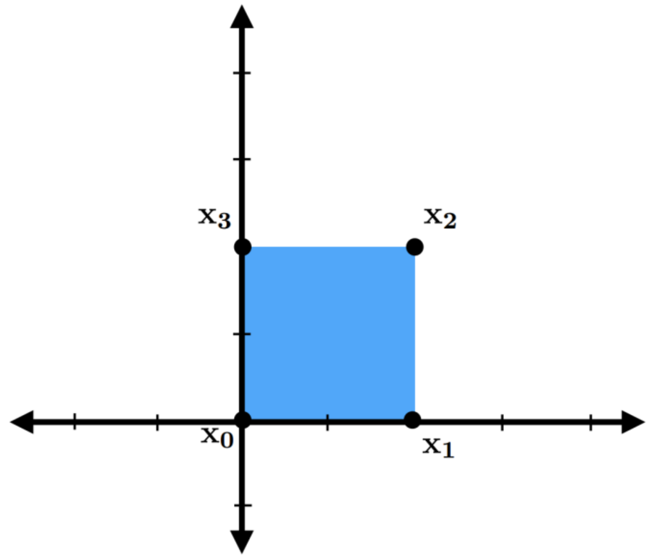


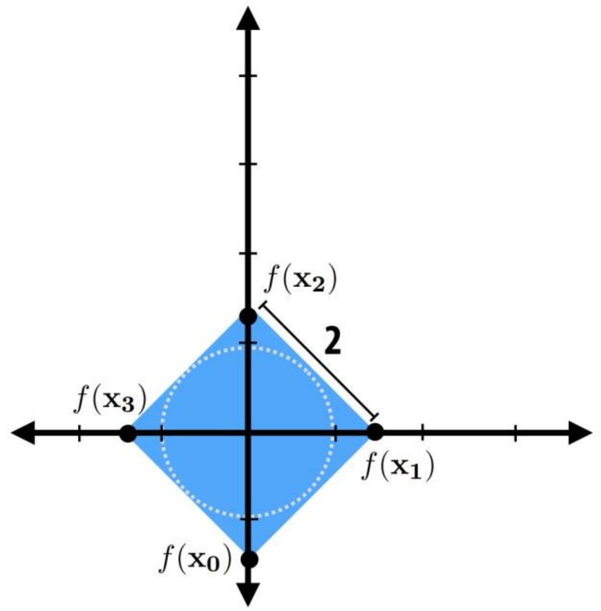
$$f(\mathbf{x}) = R_{\pi/4}\mathbf{S}_{[1.5,1.5]}\mathbf{x}$$

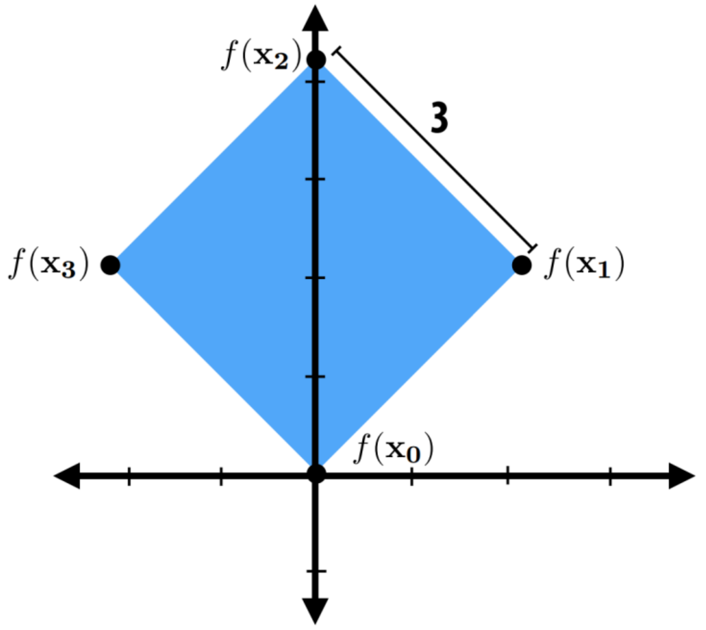
Compose linear transforms via matrix multiplication.

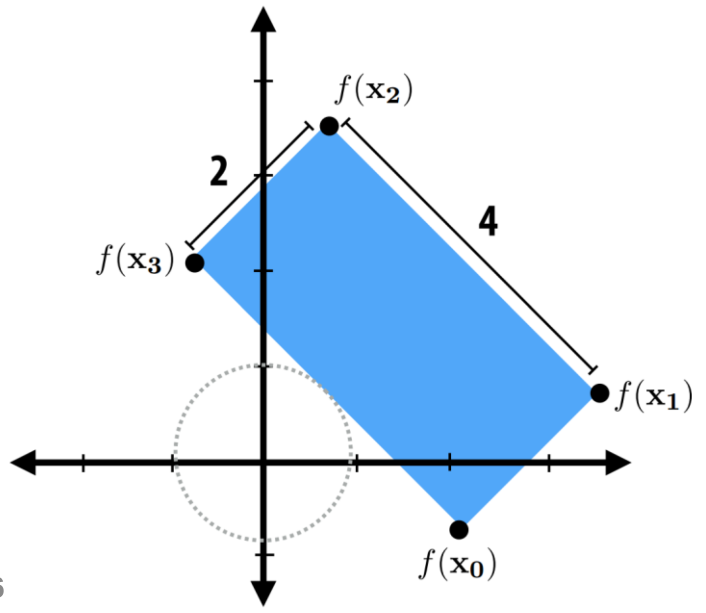
Enables simple & efficient implementation: reduce complex chain of transforms to a single matrix.

How would you perform these transformations?

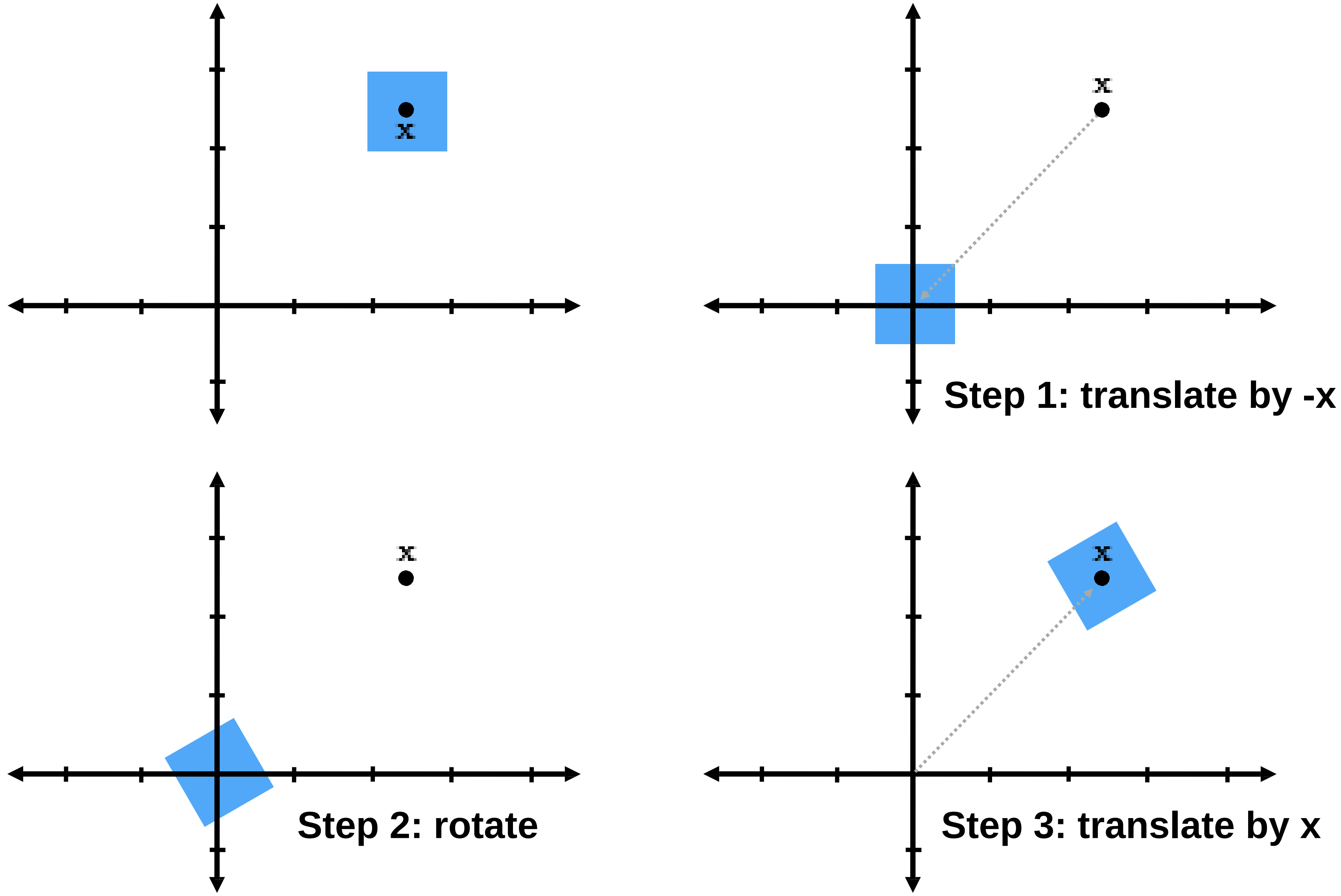








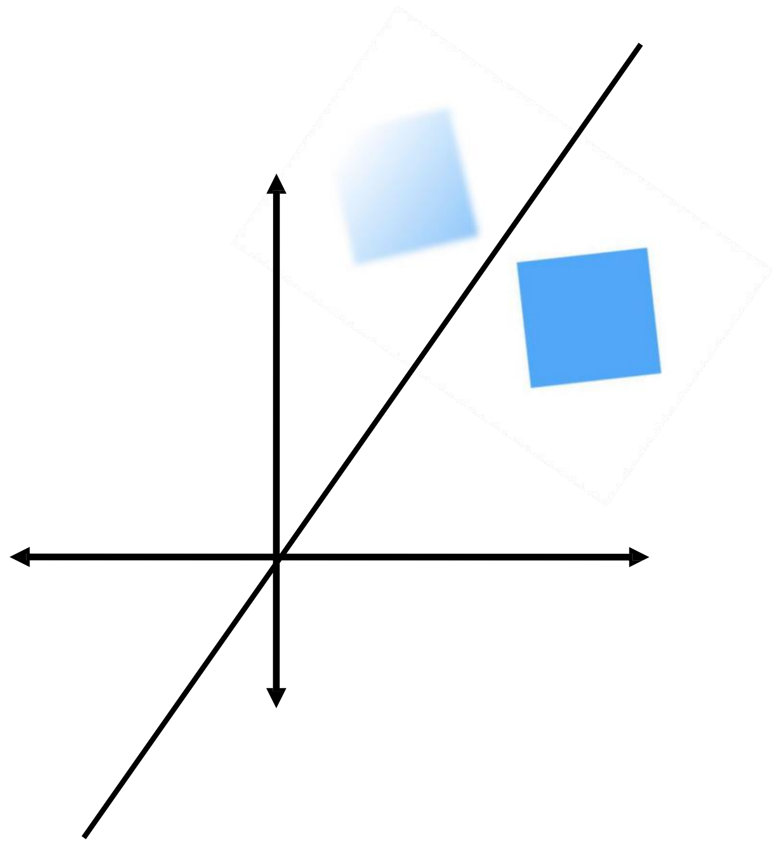
Common pattern: rotation about point x



Q: In homogenous coordinates, what does the corresponding transformation matrix look like?

Exercise

- **Reflection about an arbitrary line**



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):

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

Translate:

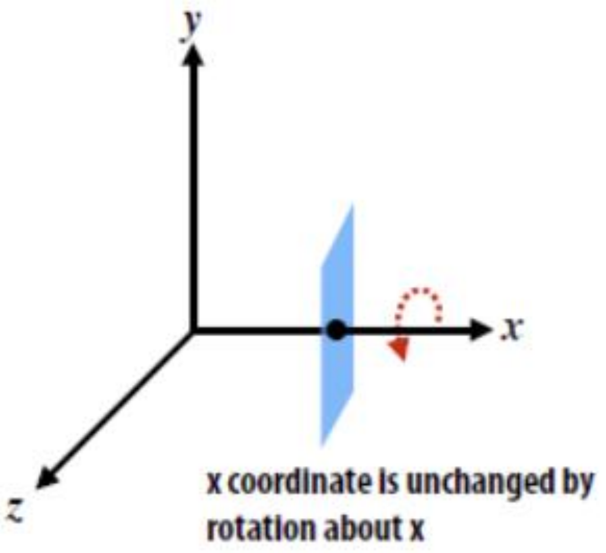
3D-H

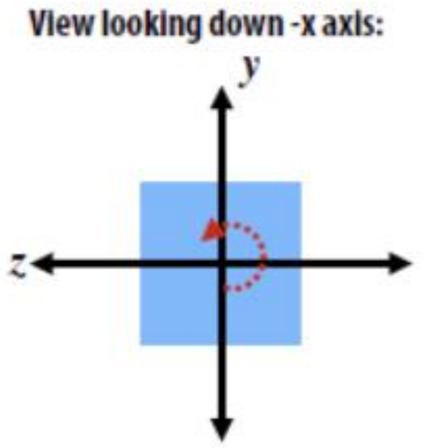
$$\mathbf{T_b} = \begin{bmatrix} 1 & 0 & 0 & \mathbf{b}_x \\ 0 & 1 & 0 & \mathbf{b}_y \\ 0 & 0 & 1 & \mathbf{b}_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotations in 3D

Rotation about x axis:

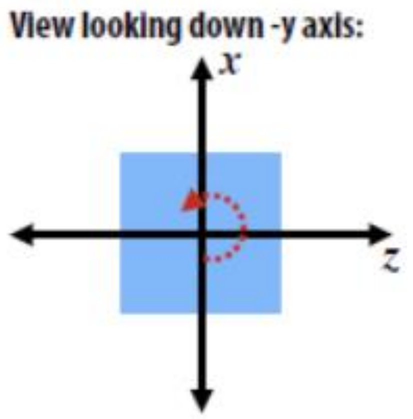
$$\mathbf{R}_{x,\theta} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$





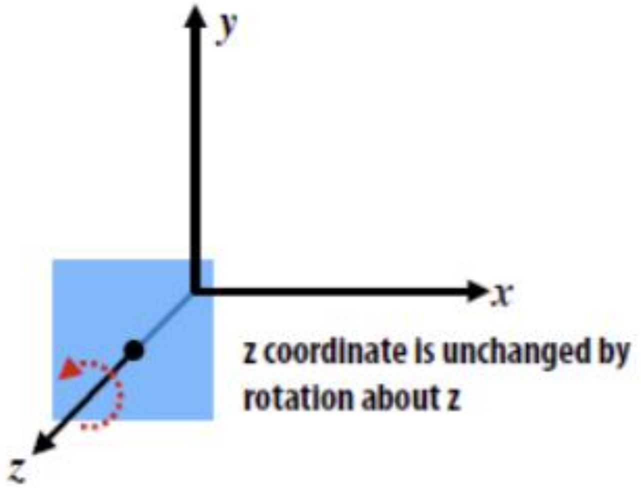
Rotation about y axis:

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



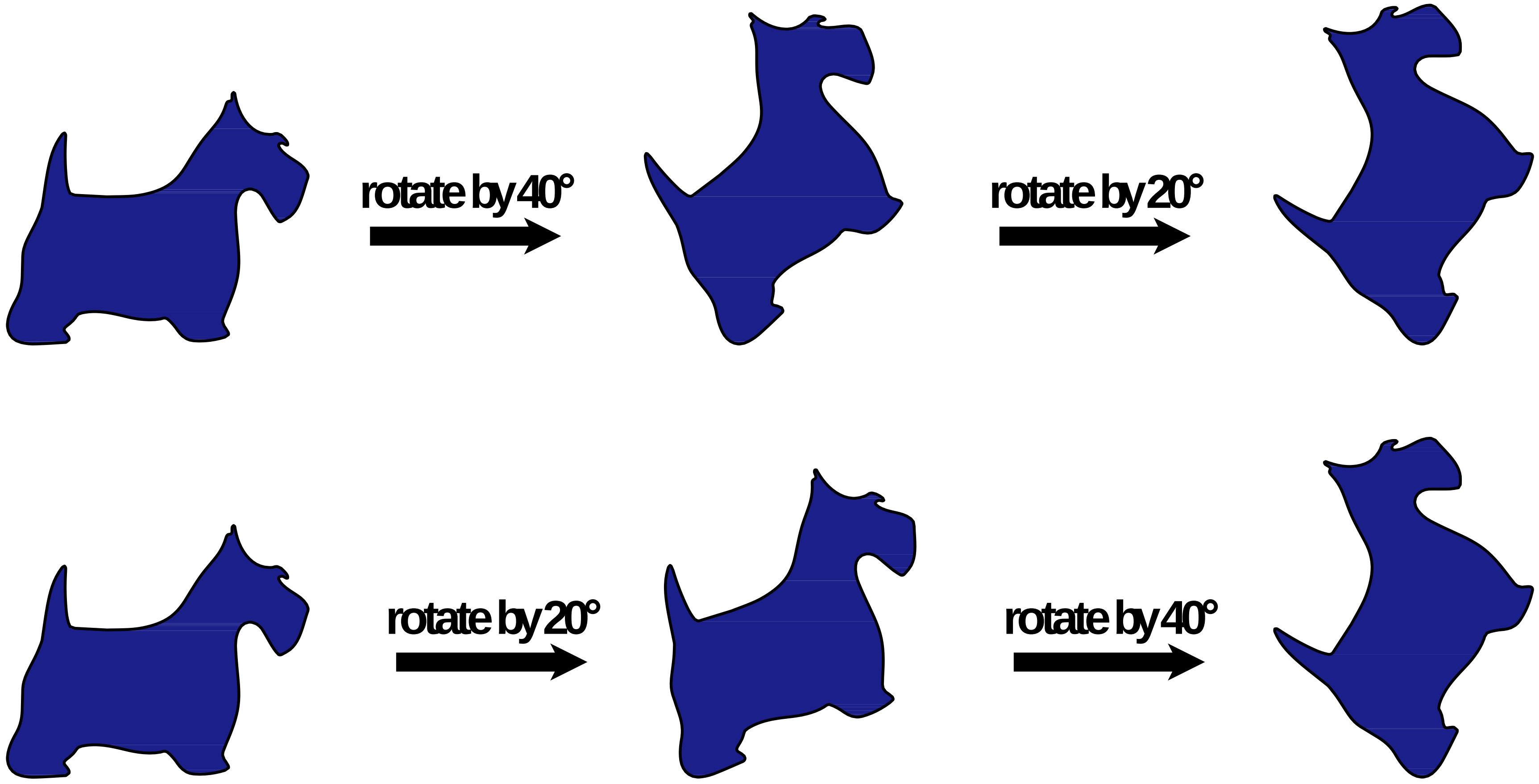
Rotation about z axis:

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



Commutativity of Rotations—2D

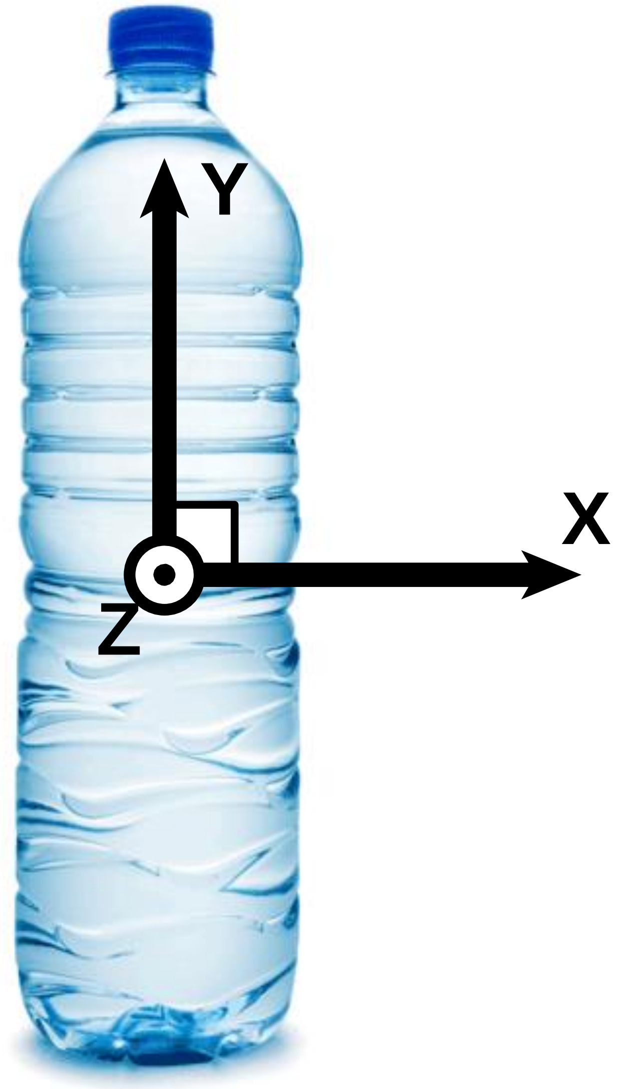
- In 2D, order of rotations doesn't matter:



Why not?

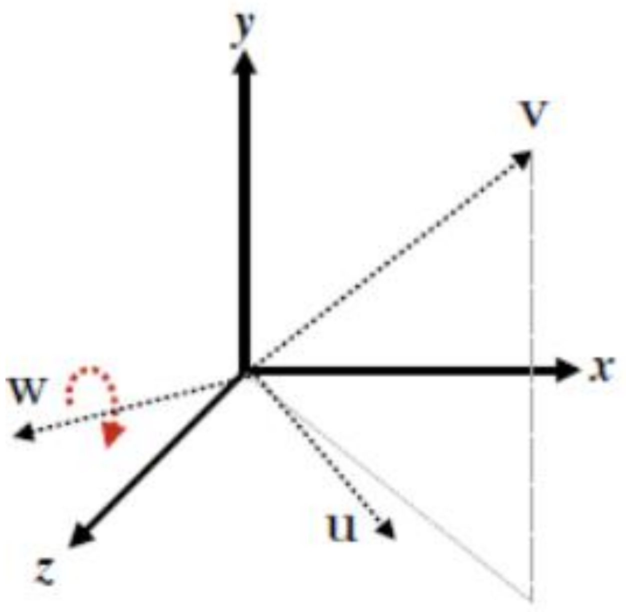
Commutativity of Rotations—3D

- 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!

Rotation about an arbitrary axis



To rotate by θ about $\mathbf{w}$:

1. Form orthonormal basis around $\mathbf{w}$ (see $\mathbf{u}$ and $\mathbf{v}$ in figure)
2. Rotate to map $\mathbf{w}$ 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: $\mathbf{R}_{z,\theta}$
4. Rotate back to original coordinate space: $\mathbf{R}_{\mathbf{uvw}}^{\mathbf{T}}$

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

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

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:**

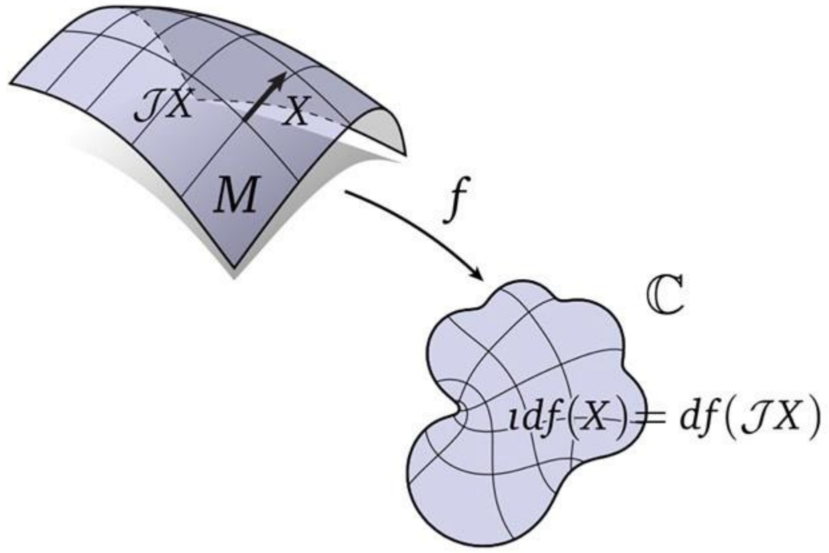
$$\begin{bmatrix} \cos\theta + u_x^2(1-\cos\theta) & u_x u_y(1-\cos\theta) - u_z\sin\theta & u_x u_z(1-\cos\theta) + u_y\sin\theta \\ u_y u_x(1-\cos\theta) + u_z\sin\theta & \cos\theta + u_y^2(1-\cos\theta) & u_y u_z(1-\cos\theta) - u_x\sin\theta \\ u_z u_x(1-\cos\theta) - u_y\sin\theta & u_z u_y(1-\cos\theta)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

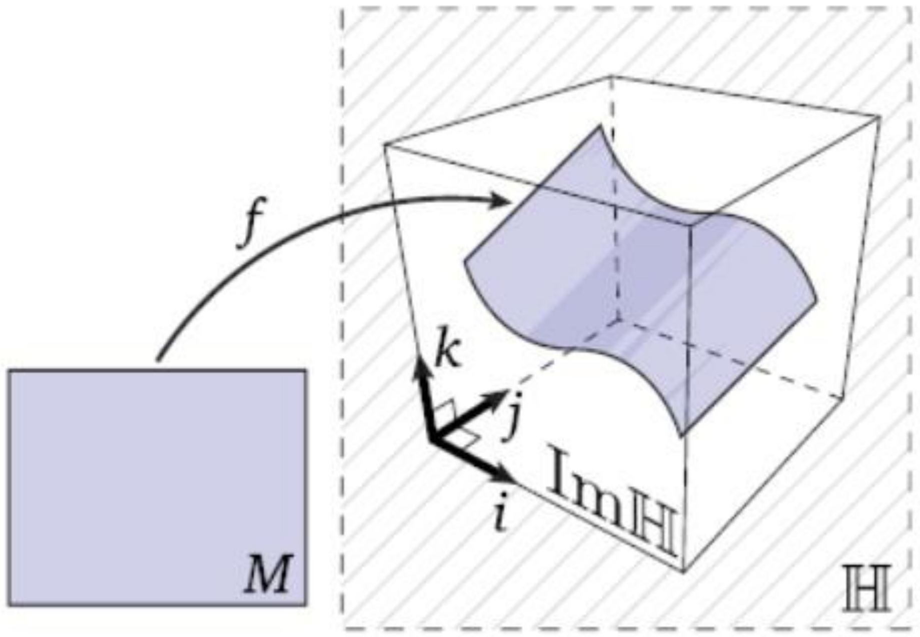
Just memorize this matrix! :-)

…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!

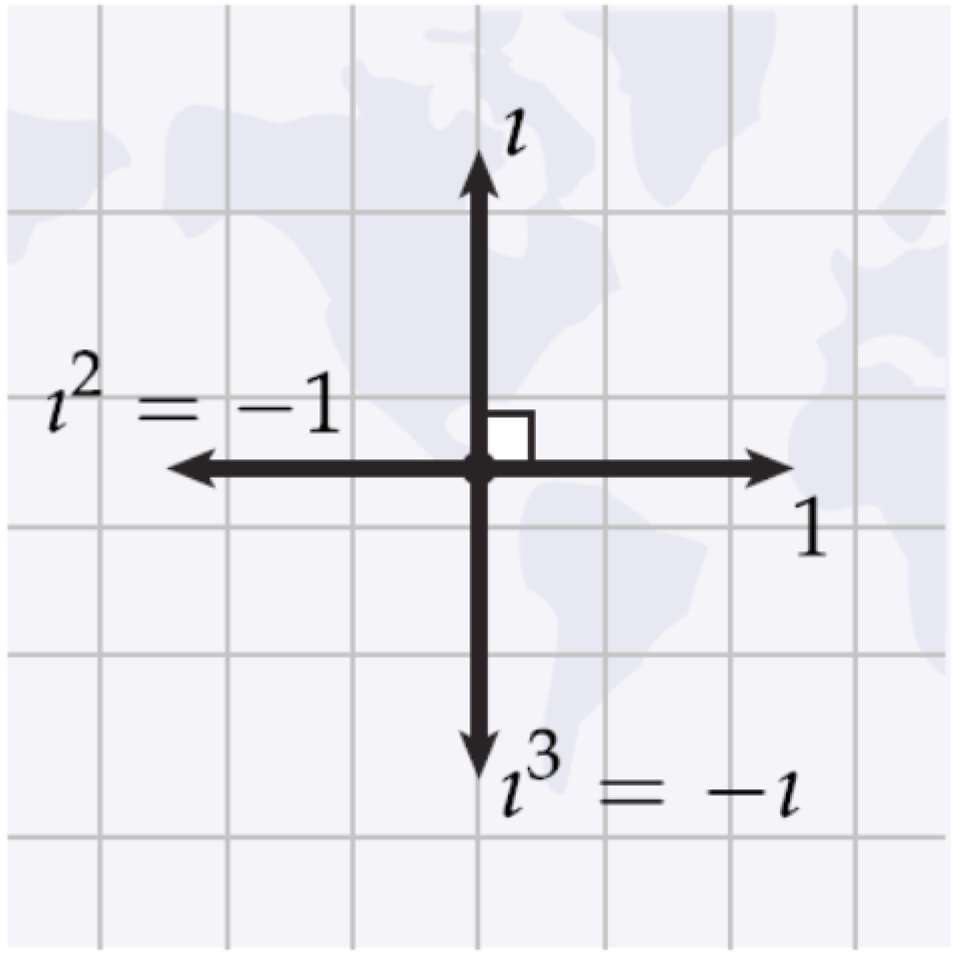
Imaginary Unit

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

nonsense!

`More importantly: obscures geometric meaning.`

Imaginary Unit—Geometric Description

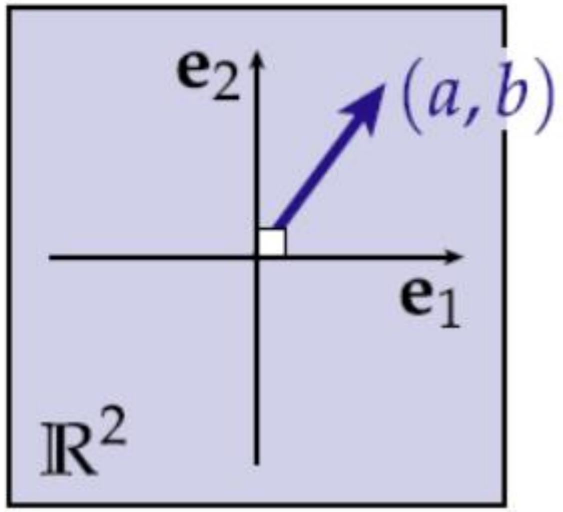


Symbol $\imath$ denotes quarter-turn in the counter-clockwise direction.

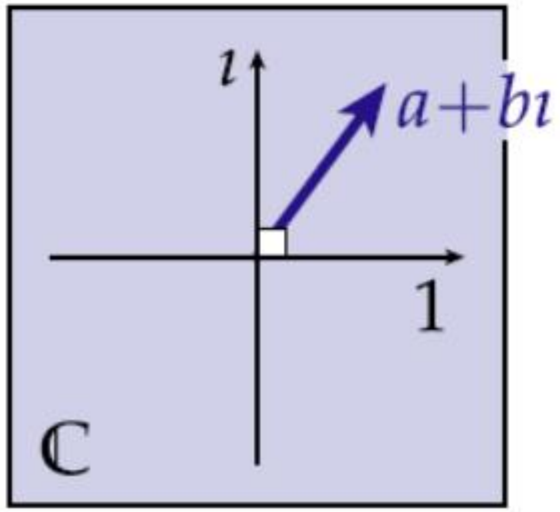
Complex Numbers

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

REAL



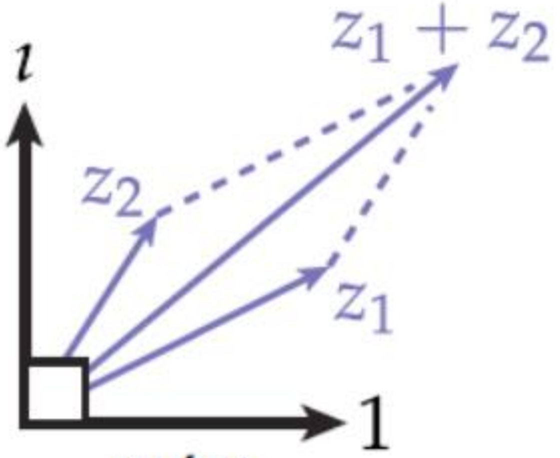
COMPLEX



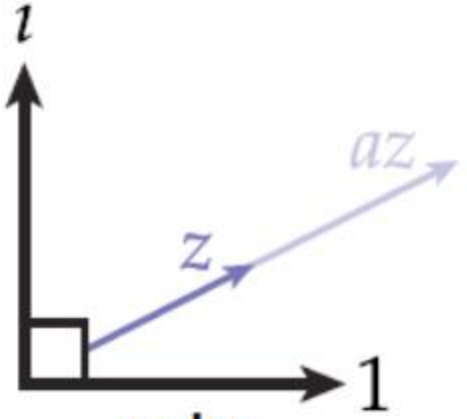
- …except that we’re going to define a useful new notion of the product between two vectors.

Complex Arithmetic

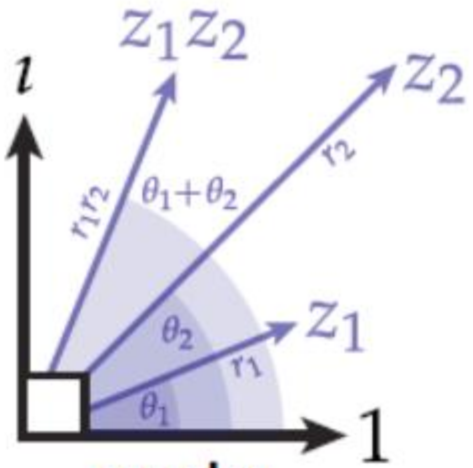
- Same operations as before, plus one more:



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—Rectangular Form

- Complex product in "rectangular" coordinates (1, ı):

$$z_1 = (a + b\imath)$$

$$z_2 = (c + d\imath)$$

two quarter turns—same as -1

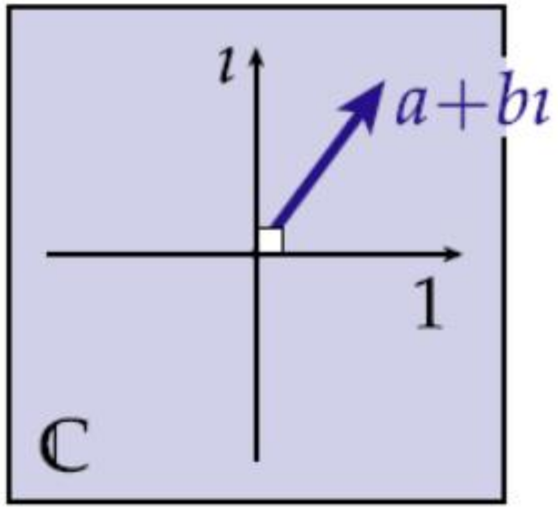
$$z_1 z_2 = ac + ad\imath + bc\imath + bd\imath^2 =$$

$$\boxed{(ac - bd) + (ad + bc)\imath.}$$

"real part" $\mathrm{Re}(z_1 z_2)$

"imaginary part" $\mathrm{Im}(z_1 z_2)$

- We used a lot of "rules" here. Can you justify them geometrically?
- Does this product agree with our geometric description (last slide)?



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*)

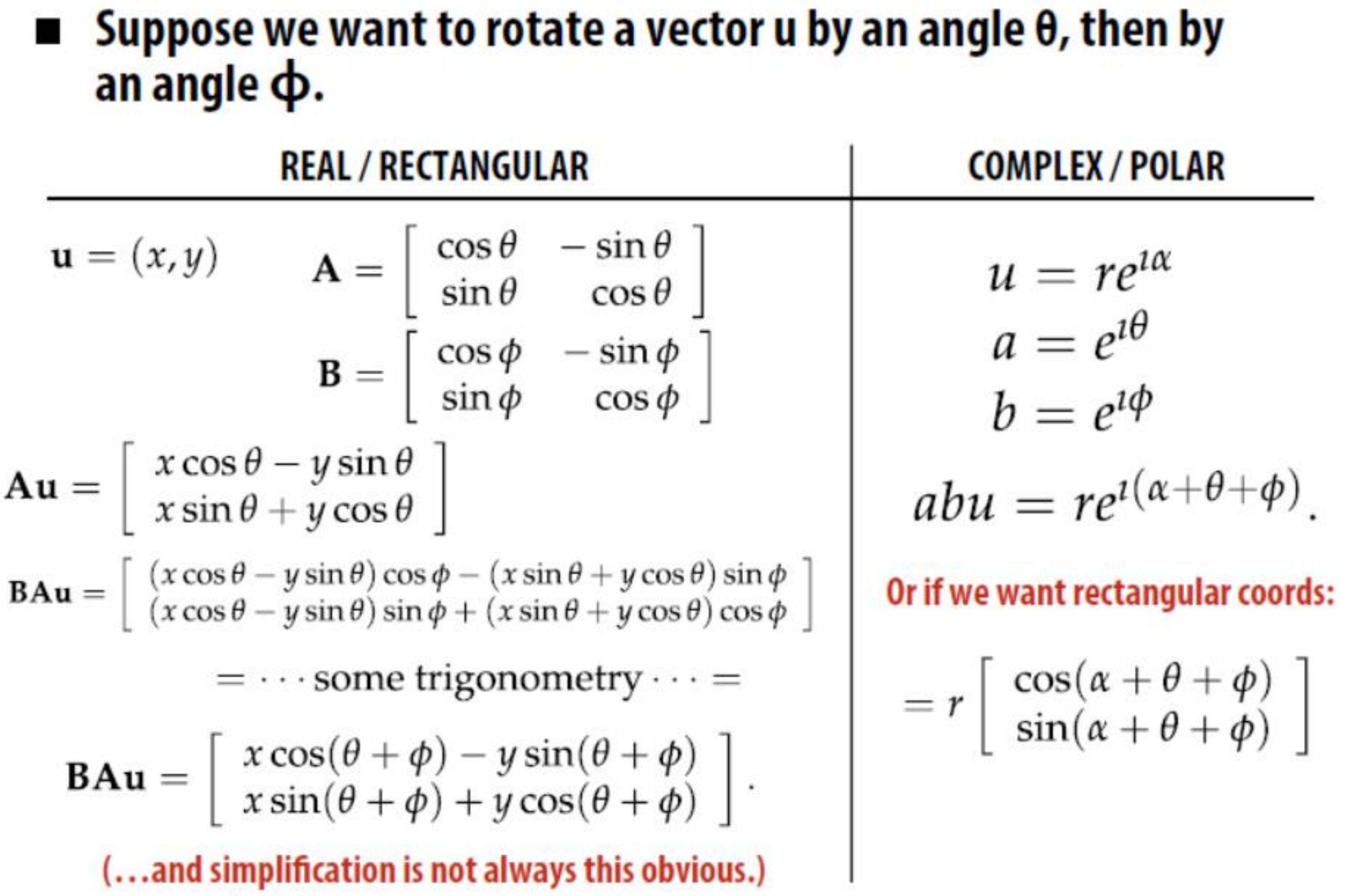
Leonhard Euler (1707–1783)

- 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

$$\mathbf{u} = (x, y) \qquad \mathbf{A} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix}$$

$$\mathbf{Au} = \begin{bmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \end{bmatrix}$$

$$\mathbf{BAu} = \begin{bmatrix} (x\cos\theta - y\sin\theta)\cos\phi - (x\sin\theta + y\cos\theta)\sin\phi \\ (x\cos\theta - y\sin\theta)\sin\phi + (x\sin\theta + y\cos\theta)\cos\phi \end{bmatrix}$$

$$= \cdots \text{some trigonometry} \cdots =$$

$$\mathbf{BAu} = \begin{bmatrix} x\cos(\theta+\phi) - y\sin(\theta+\phi) \\ x\sin(\theta+\phi) + y\cos(\theta+\phi) \end{bmatrix}.$$

(…and simplification is not always this obvious.)

COMPLEX / POLAR

$$u = re^{\imath\alpha}$$

$$a = e^{\imath\theta}$$

$$b = e^{\imath\phi}$$

$$abu = re^{\imath(\alpha+\theta+\phi)}.$$

Or if we want rectangular coords:

$$= r\begin{bmatrix} \cos(\alpha+\theta+\phi) \\ \sin(\alpha+\theta+\phi) \end{bmatrix}$$

Pervasive theme in graphics:

Sure, there are often many "equivalent" representations.

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

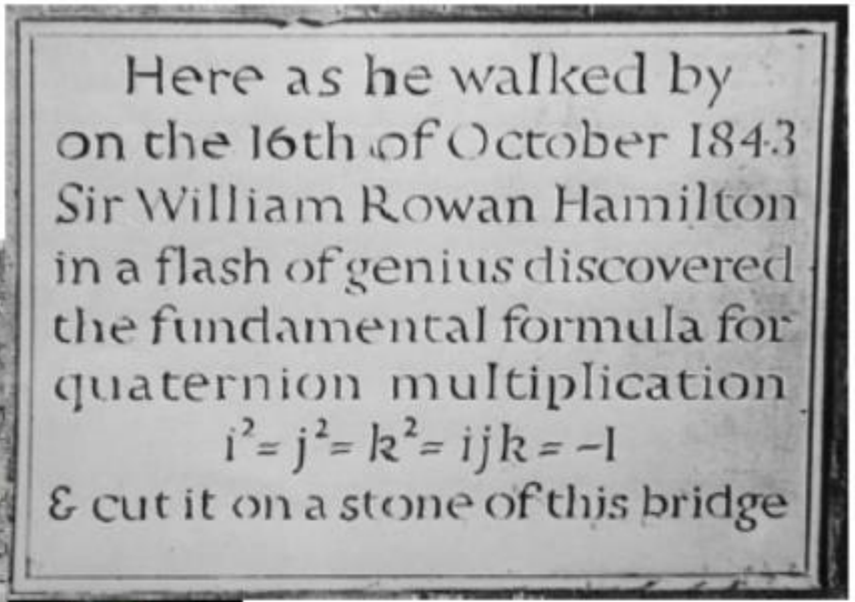
*Or most efficient, or most accurate…

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)



Here as he walked by
on the 16th of October 1843
Sir William Rowan Hamilton
in a flash of genius discovered
the fundamental formula for
quaternion multiplication
$i^2 = j^2 = k^2 = ijk = -1$
& cut it on a stone of this bridge

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

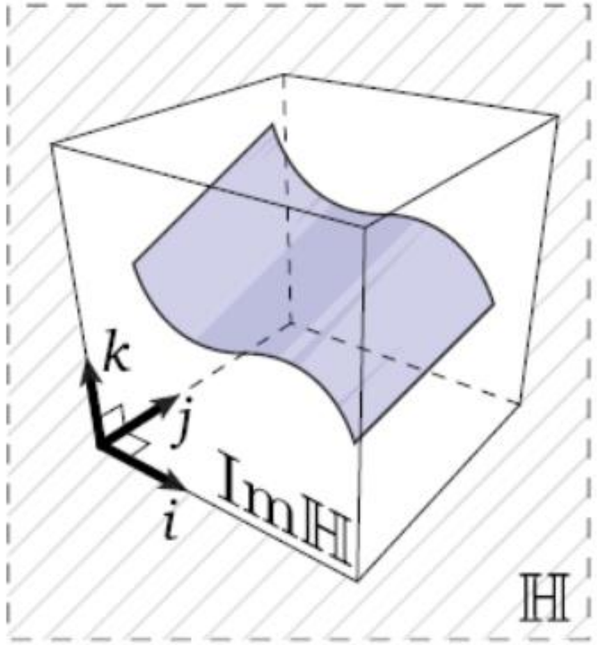
$$\imath^2 = \jmath^2 = k^2 = \imath\jmath k = -1$$

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**

$$q = a_1 + b_1\imath + c_1\jmath + d_1k$$
$$p = a_2 + b_2\imath + c_2\jmath + d_2k$$

- **Can express their product as**

$$\begin{aligned} qp = {} & a_1a_2 - b_1b_2 - c_1c_2 - d_1d_2 \\ & +(a_1b_2 + b_1a_2 + c_1d_2 - d_1c_2)\imath \\ & +(a_1c_2 - b_1d_2 + c_1a_2 + d_1b_2)\jmath \\ & +(a_1d_2 + b_1c_2 - c_1b_2 + d_1a_2)k \end{aligned}$$

...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

$$(\,\underset{\mathbb{R}}{\underset{\rotatebox{90}{$\in$}}{\text{scalar}}},\ \underset{\mathbb{R}^3}{\underset{\rotatebox{90}{$\in$}}{\text{vector}}}\,) \in \mathbb{H}$$

- 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:

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

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 $\tilde{q}$.

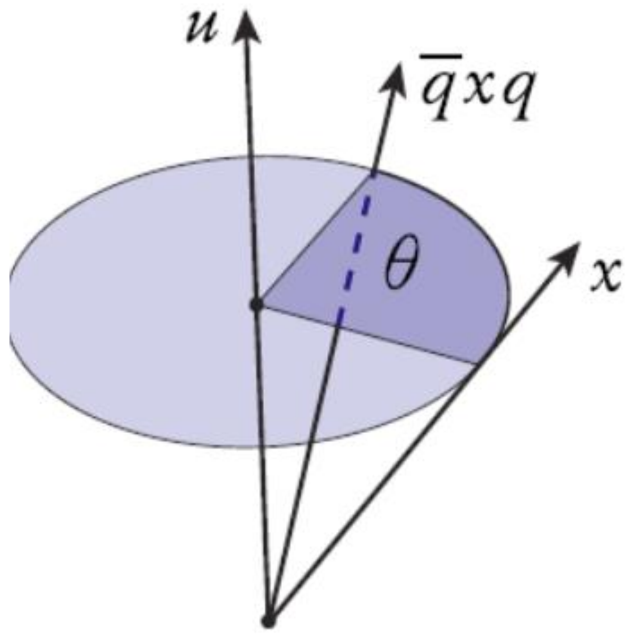
Conjugation is an involution, meaning **that it is its own inverse**, so conjugating an element twice returns the original element.

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

3D Transformations via Quaternions

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

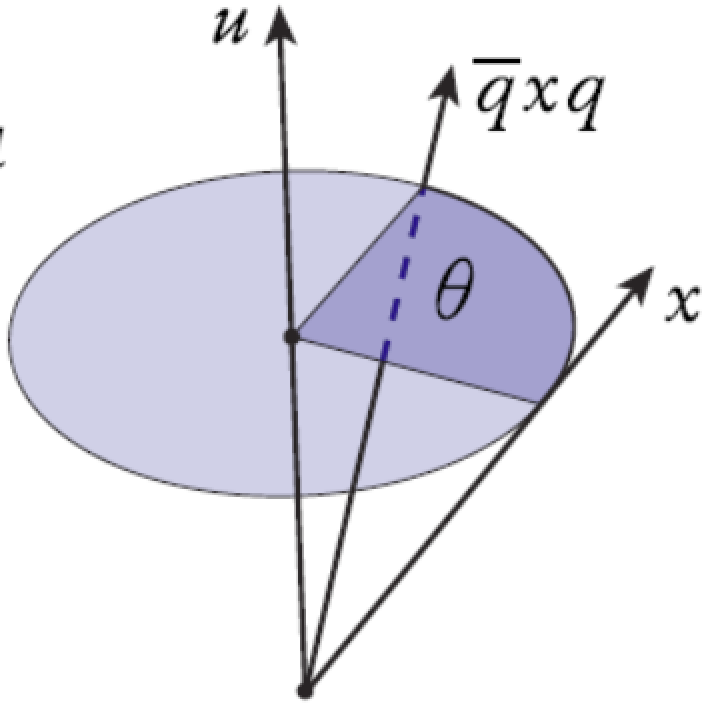
$$x \in \mathrm{Im}(\mathbb{H})$$
$$q \in \mathbb{H}, \quad |q|^2 = 1$$



Rotation from Axis/Angle, Revisited

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

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

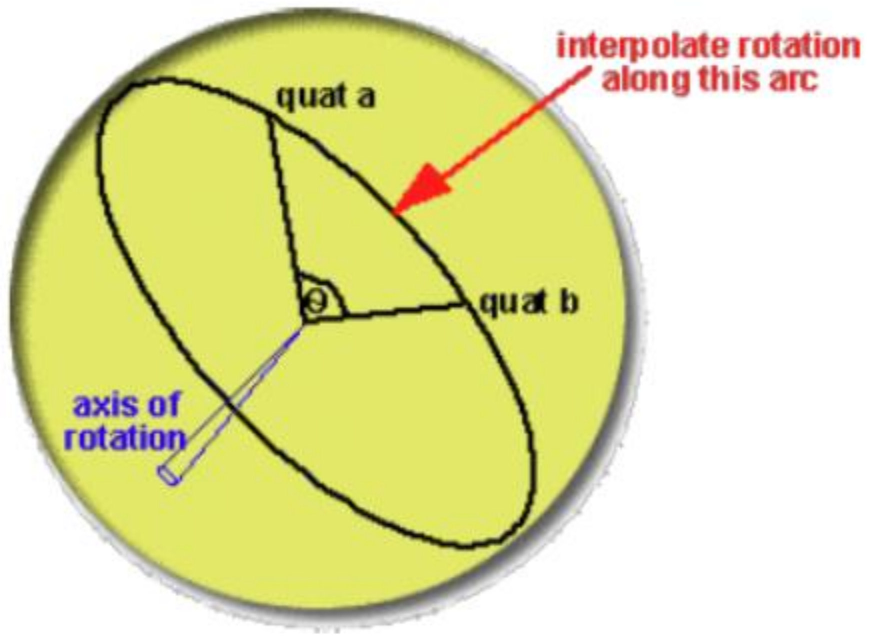


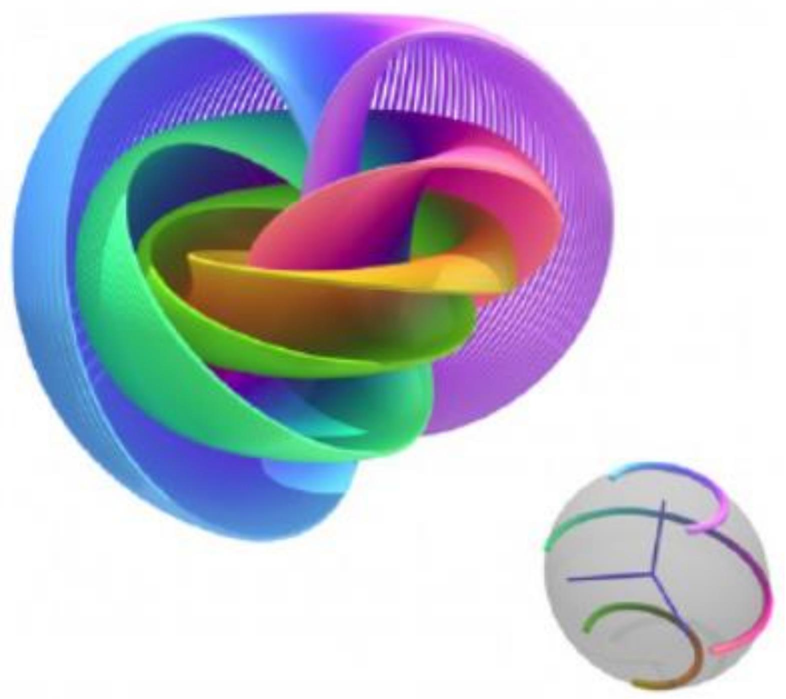
- Slightly easier to remember (and manipulate) than matrix:

$$\begin{bmatrix} \cos\theta + u_x^2(1-\cos\theta) & u_xu_y(1-\cos\theta) - u_z\sin\theta & u_xu_z(1-\cos\theta) + u_y\sin\theta \\ u_yu_x(1-\cos\theta) + u_z\sin\theta & \cos\theta + u_y^2(1-\cos\theta) & u_yu_z(1-\cos\theta) - u_x\sin\theta \\ u_zu_x(1-\cos\theta) - u_y\sin\theta & u_zu_y(1-\cos\theta) + u_x\sin\theta & \cos\theta + u_z^2(1-\cos\theta) \end{bmatrix}$$

More Quaternions and Rotation

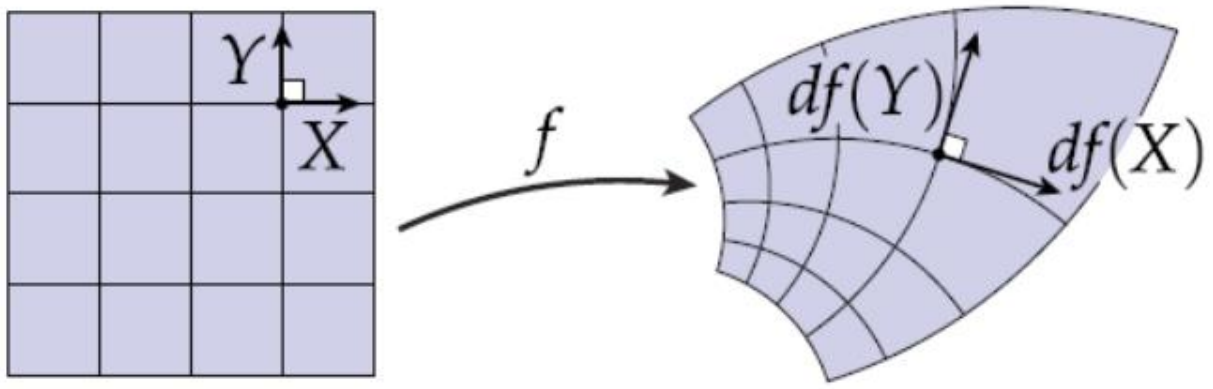
- Don't have time to cover everything, but…
- Quaternions provide some very nice utility/perspective when it comes to rotations:
  - Spherical linear interpolation ("slerp")
  - *Hopf fibration / "belt trick"*
  - …

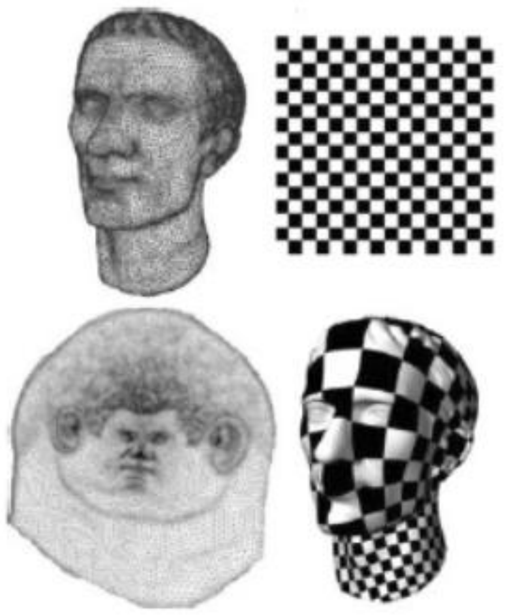


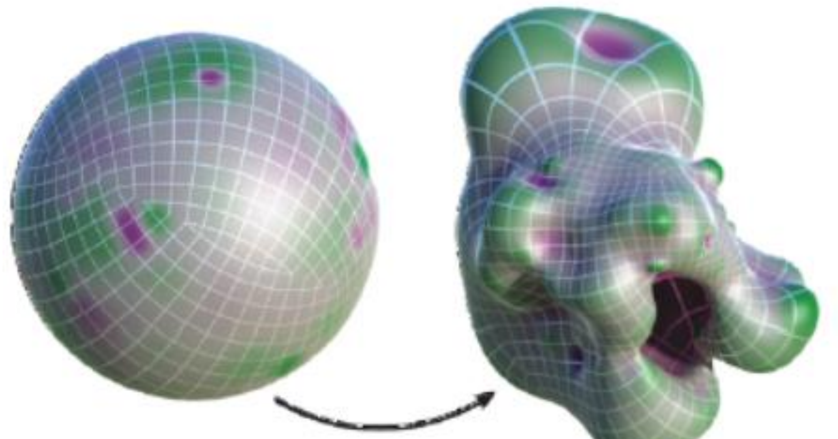


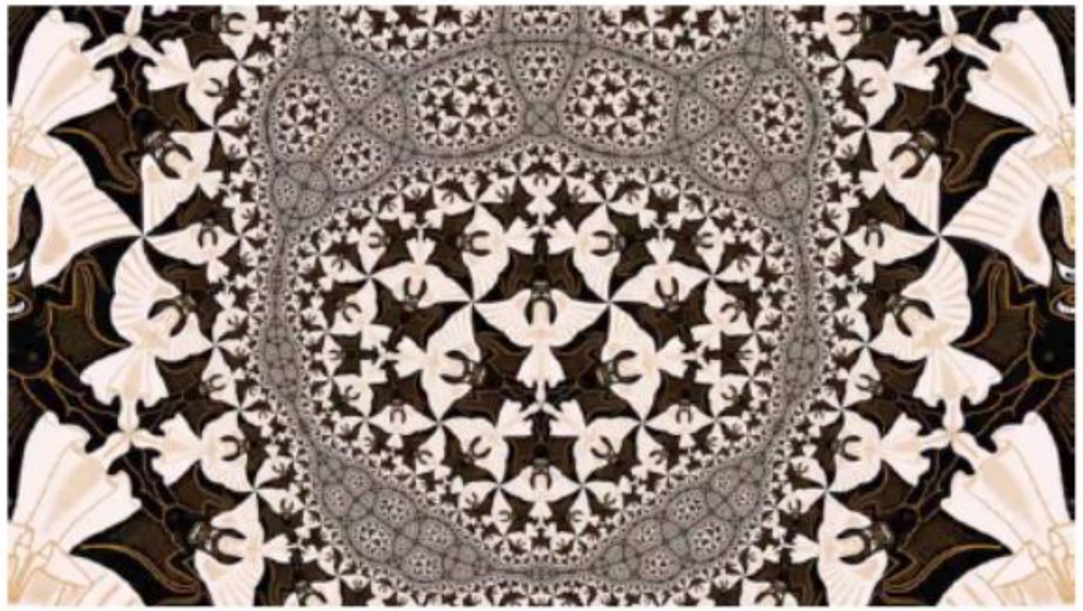
Where else are (hyper-)complex numbers useful in computer graphics?

Complex #s: Language of *Conformal Maps*



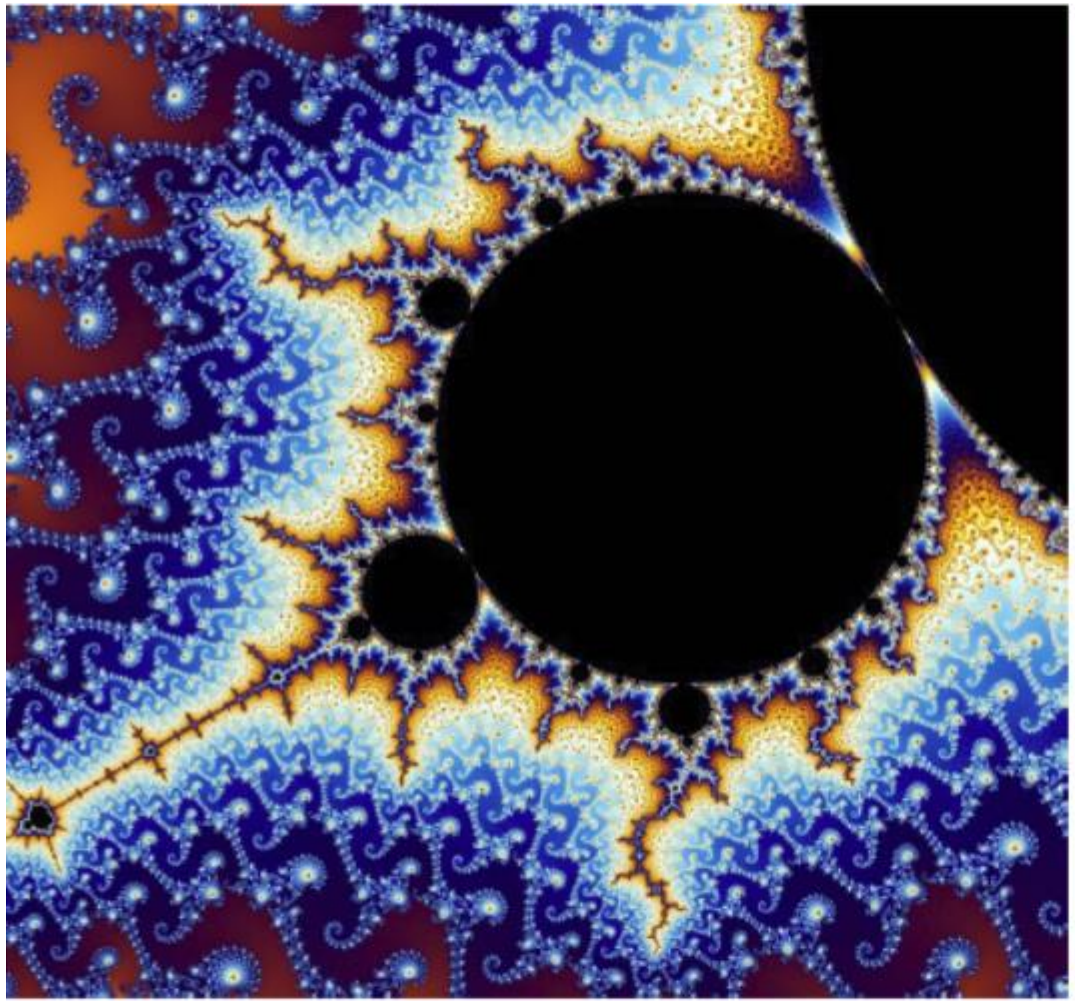


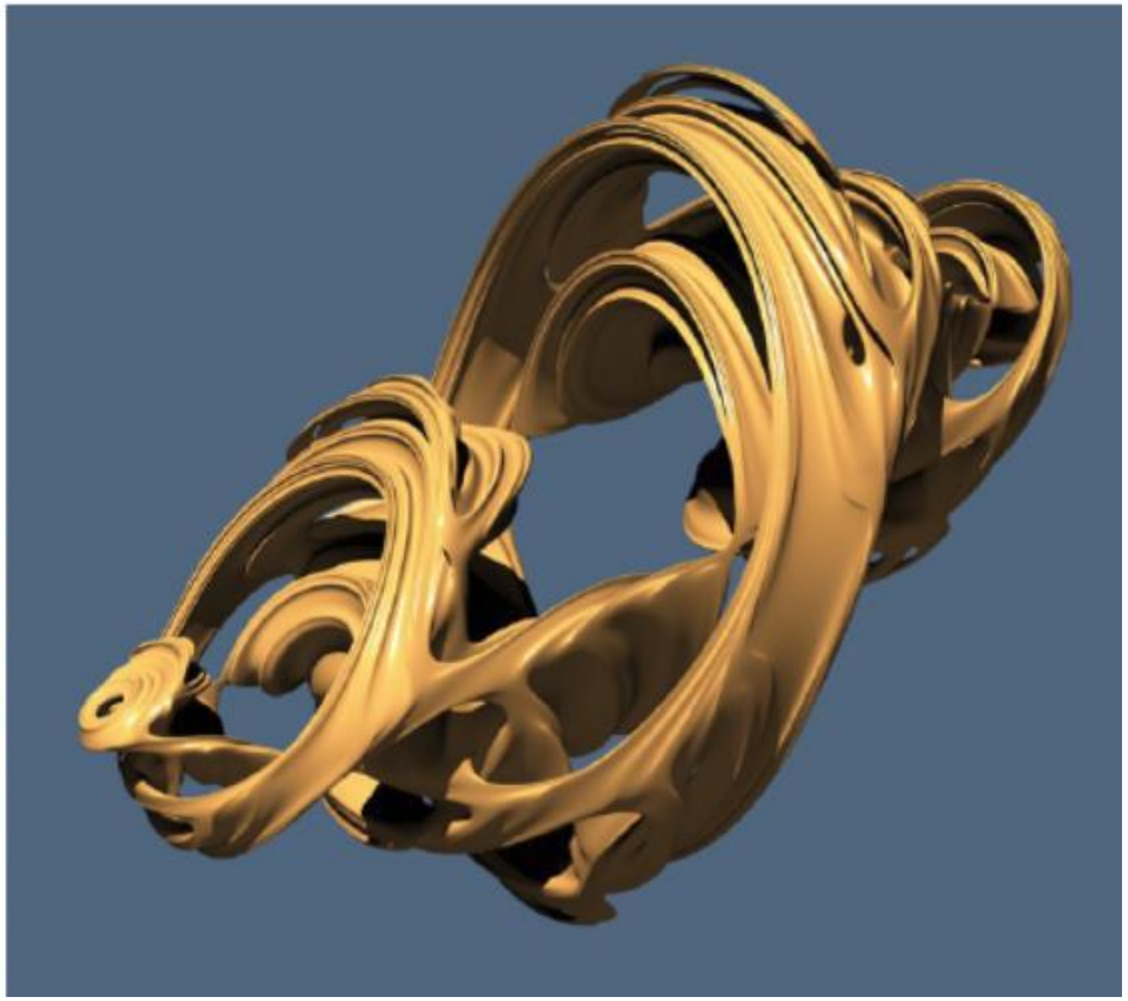




Useless-But-Beautiful Example: Fractals

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





Thanks