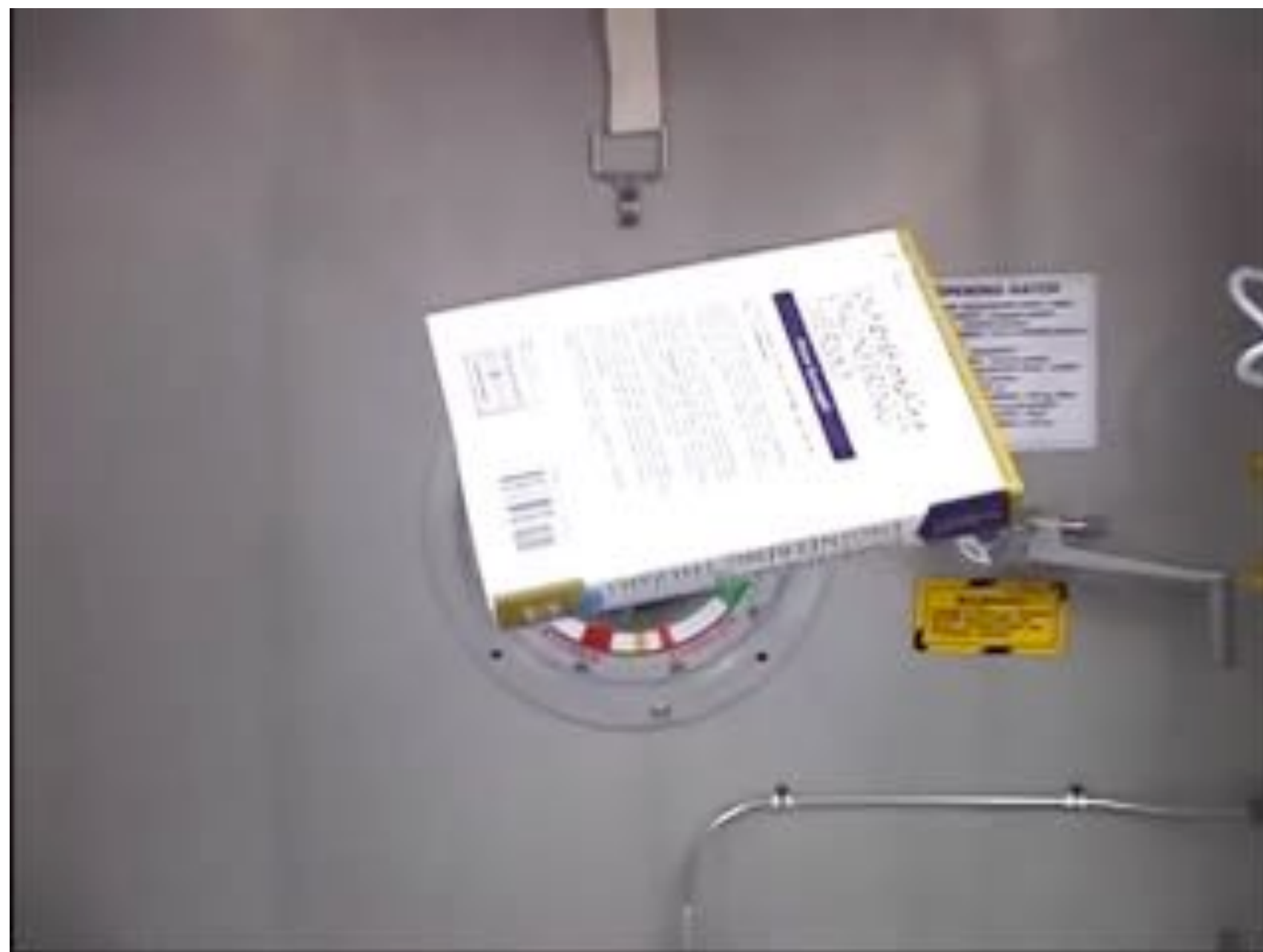


3D Rotations and Complex Representations

Computer Graphics
CMU 15-462/15-662

Rotations in 3D

- **What is a rotation, intuitively?**
- **How do you know a rotation when you see it?**
 - **length/distance is preserved (no stretching/shearing)**
 - **orientation is preserved (e.g., text remains readable)**
 - **origin is preserved (otherwise it's a rotation + translation)**



3D Rotations—Degrees of Freedom

- How many numbers do we need to specify a rotation in 3D?
- For instance, we could use rotations around X, Y, Z. But do we need all three?
- Well, to rotate Pittsburgh to another city (say, São Paulo), we have to specify two numbers: latitude & longitude:
- Do we really need both latitude and longitude? Or will one suffice?
- Is that the only rotation from Pittsburgh to São Paulo? (How many more numbers do we need?)

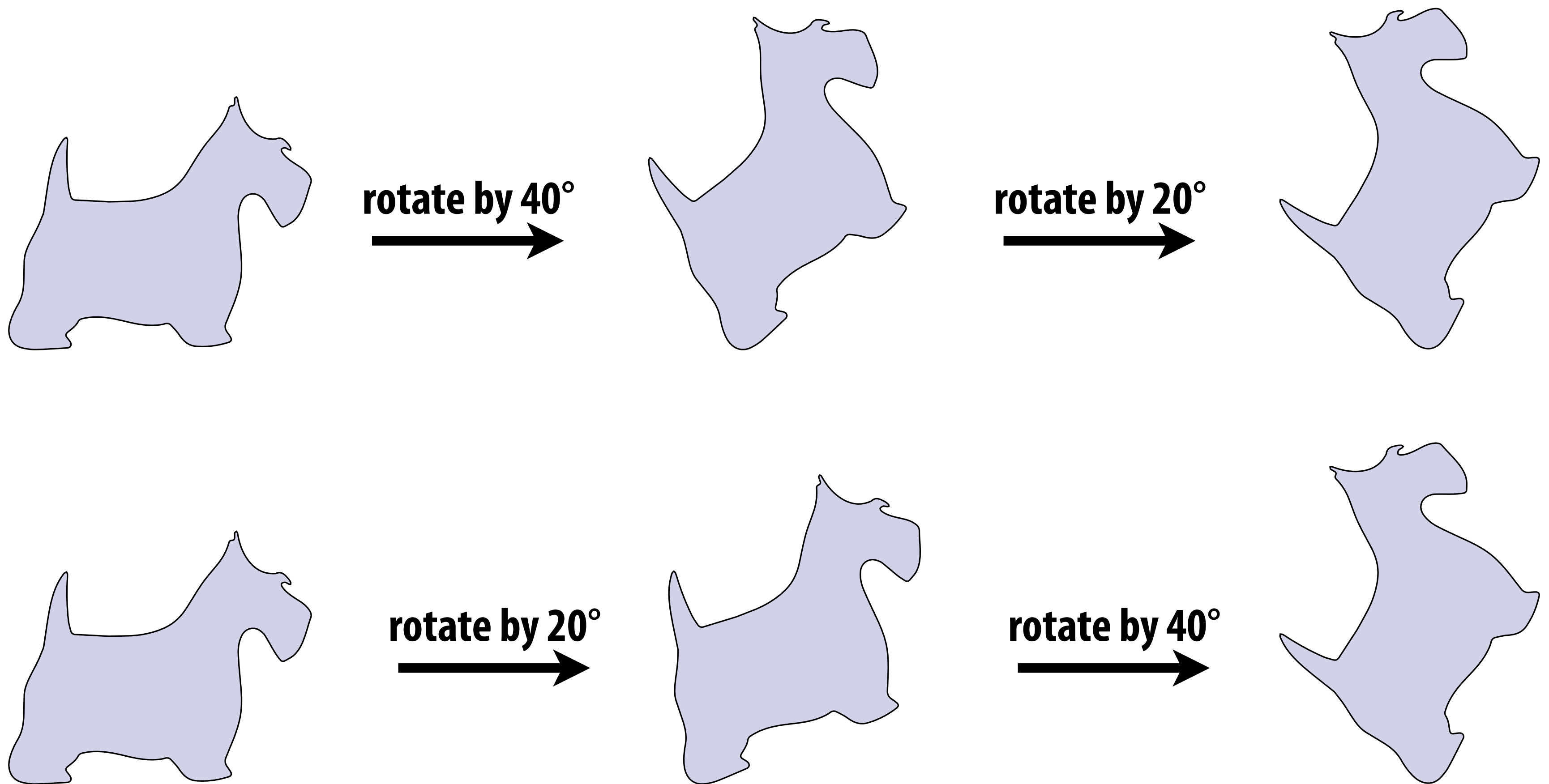
NO: We can keep São Paulo fixed as we rotate the globe.

Hence, we MUST have three degrees of freedom.



Commutativity of Rotations—2D

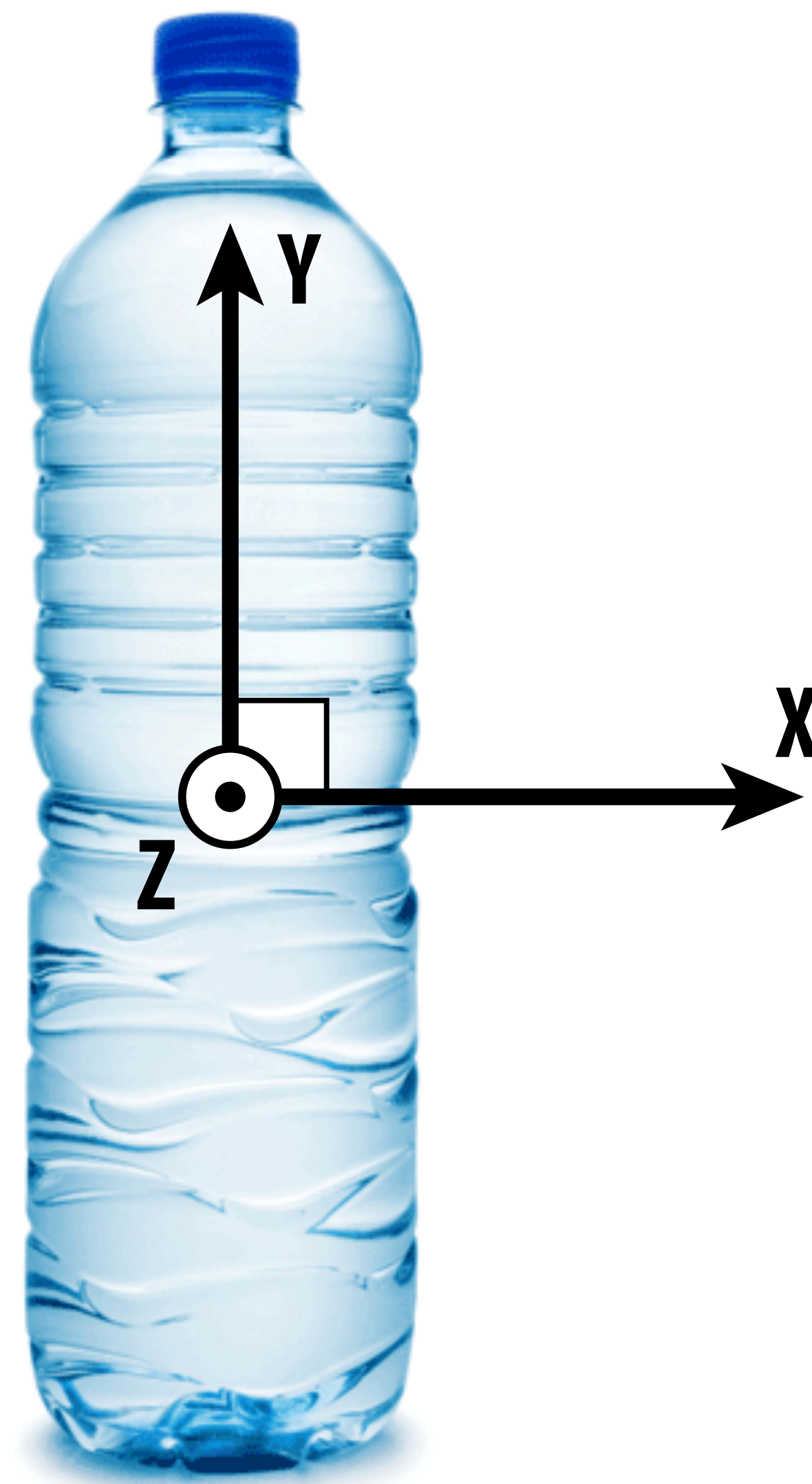
- In 2D, order of rotations doesn't matter:



Same result! ("2D rotations commute")

Commutativity of Rotations—3D

- What about in 3D?
- Try it at home—grab a water bottle!
 - 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!

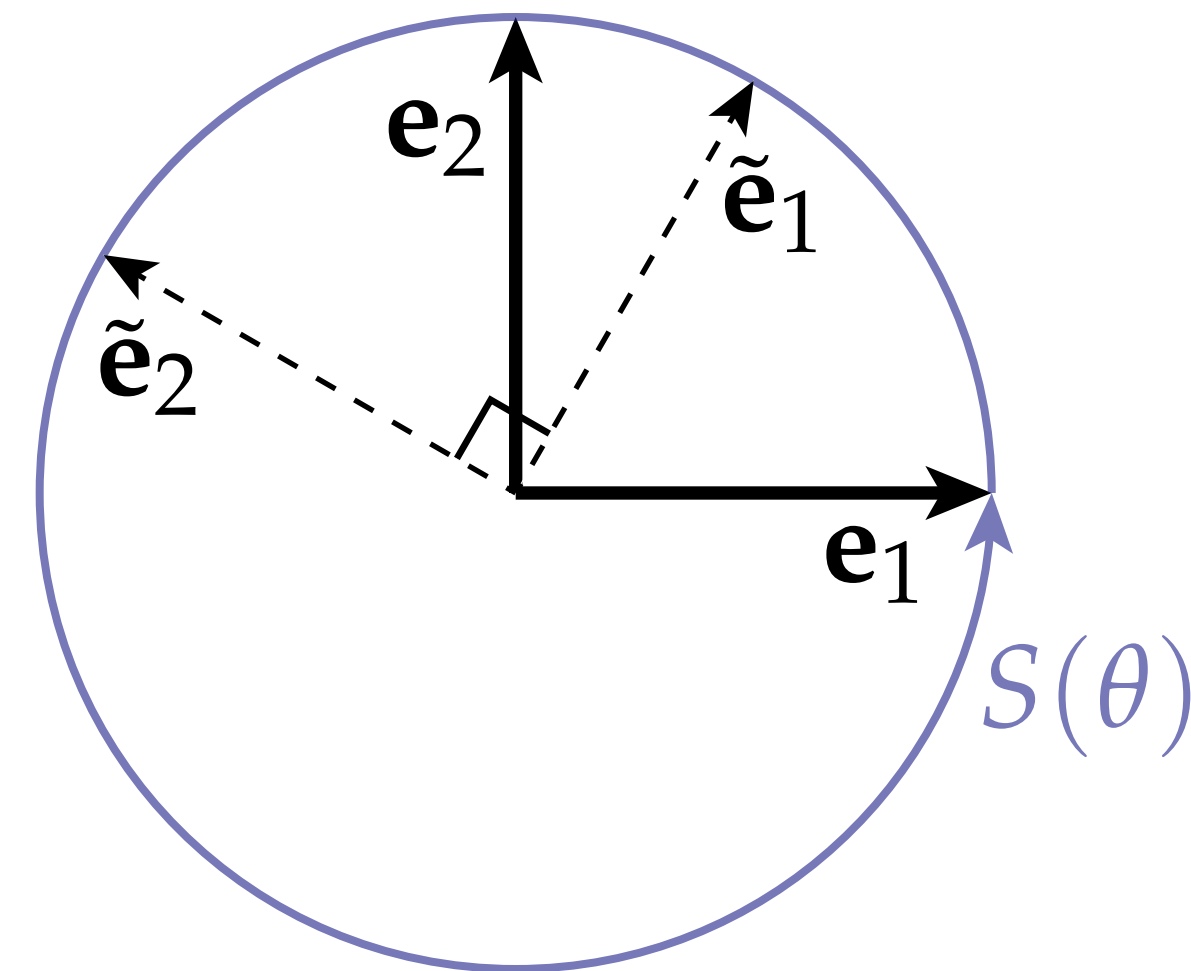
Representing Rotations—2D

- First things first: how do we get a rotation matrix in 2D?
(Don't just regurgitate the formula!)
- Suppose I have a function $S(\theta)$ that for a given angle θ gives me the point (x,y) around a circle (CCW).

- Right now, I do not care how this function is expressed!*

- What's e_1 rotated by θ ? $\tilde{e}_1 = S(\theta)$
- What's e_2 rotated by θ ? $\tilde{e}_2 = S(\theta + \pi/2)$
- How about $u := ae_1 + be_2$?

$$u := aS(\theta) + bS(\theta + \pi/2)$$



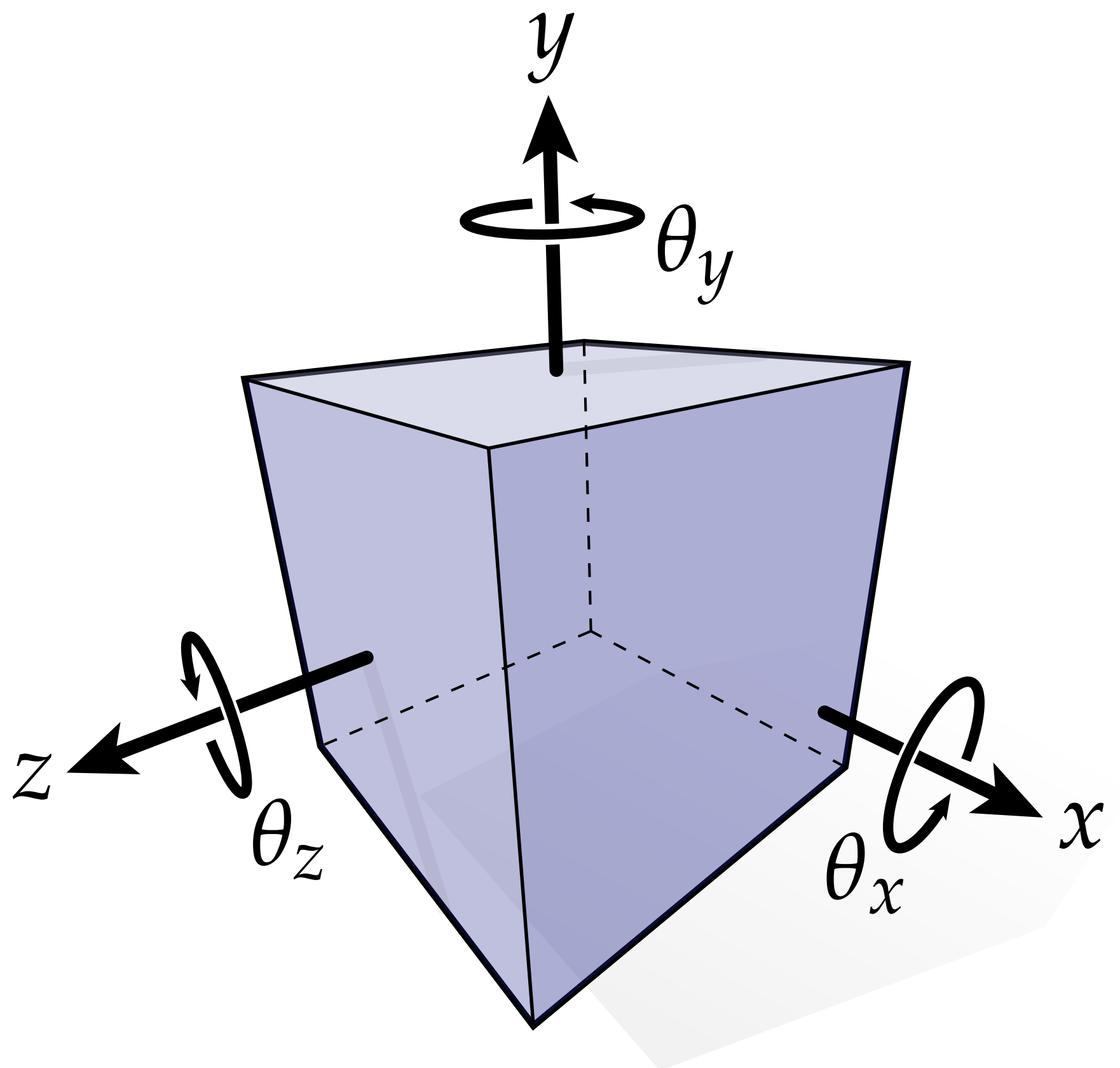
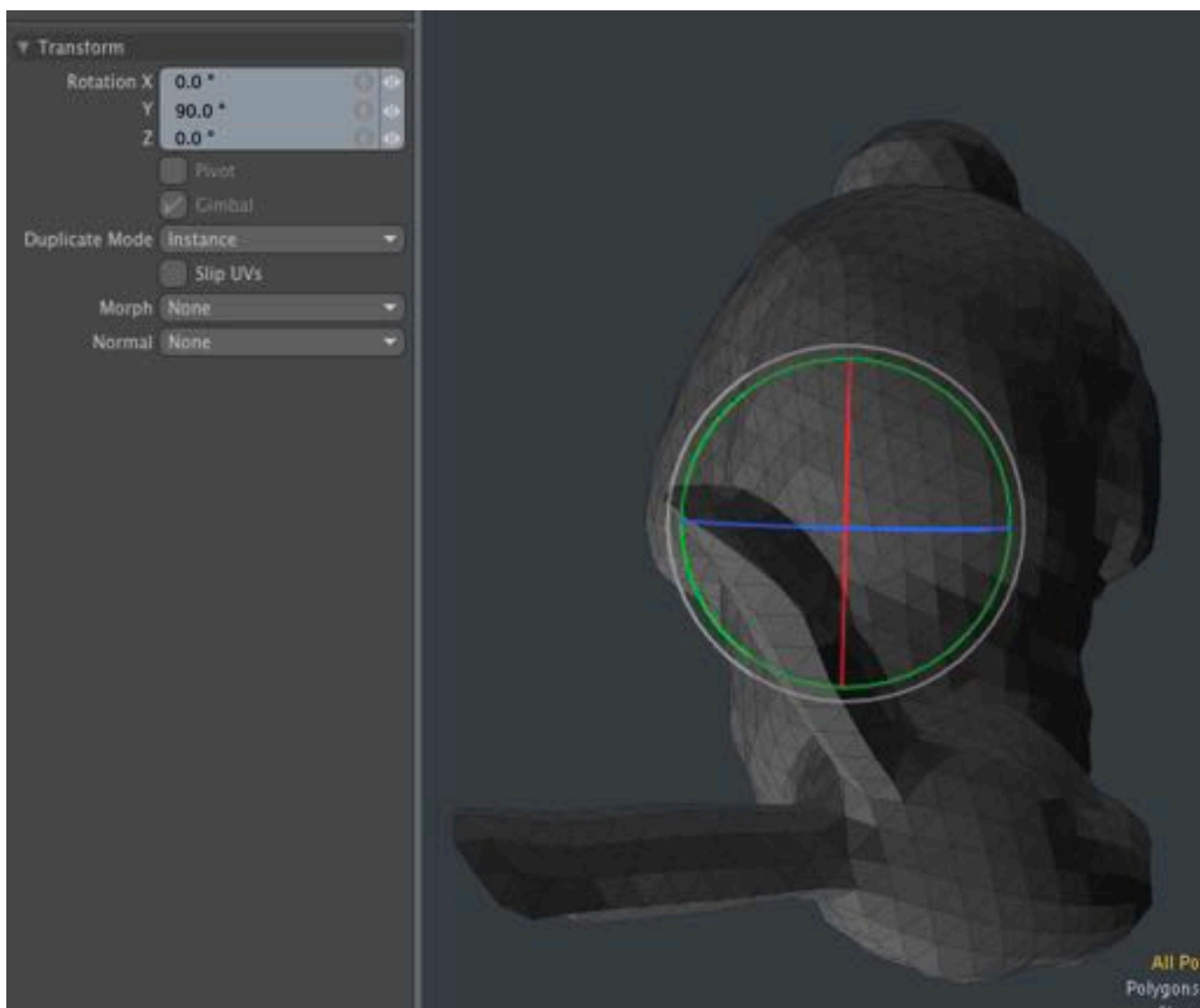
What then must the matrix look like?

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

*I.e., I don't yet care about sines and cosines and so forth.

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
- “Gimbal Lock”



Gimbal Lock

- When using Euler angles $\theta_x, \theta_y, \theta_z$, may reach a configuration where there is no way to rotate around one of the three axes!

- Recall rotation matrices around three axes:

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta_x & -\sin \theta_x \\ 0 & \sin \theta_x & \cos \theta_x \end{bmatrix} \quad R_y = \begin{bmatrix} \cos \theta_y & 0 & \sin \theta_y \\ 0 & 1 & 0 \\ -\sin \theta_y & 0 & \cos \theta_y \end{bmatrix} \quad R_z = \begin{bmatrix} \cos \theta_z & -\sin \theta_z & 0 \\ \sin \theta_z & \cos \theta_z & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Product of these matrices represents rotation by Euler angles:

$$R_x R_y R_z = \begin{bmatrix} \cos \theta_y \cos \theta_z & -\cos \theta_y \sin \theta_z & \sin \theta_y \\ \cos \theta_z \sin \theta_x \sin \theta_y + \cos \theta_x \sin \theta_z & \cos \theta_x \cos \theta_z - \sin \theta_x \sin \theta_y \sin \theta_z & -\cos \theta_y \sin \theta_x \\ -\cos \theta_x \cos \theta_z \sin \theta_y + \sin \theta_x \sin \theta_z & \cos \theta_z \sin \theta_x + \cos \theta_x \sin \theta_y \sin \theta_z & \cos \theta_x \cos \theta_y \end{bmatrix}$$

- Consider special case $\theta_y = \pi/2$ (so, $\cos \theta_y = 0$, $\sin \theta_y = 1$):

$$\Rightarrow \begin{bmatrix} 0 & 0 & 1 \\ \cos \theta_z \sin \theta_x + \cos \theta_x \sin \theta_z & \cos \theta_x \cos \theta_z - \sin \theta_x \sin \theta_z & 0 \\ -\cos \theta_x \cos \theta_z + \sin \theta_x \sin \theta_z & \cos \theta_z \sin \theta_x + \cos \theta_x \sin \theta_z & 0 \end{bmatrix}$$

Gimbal Lock, continued

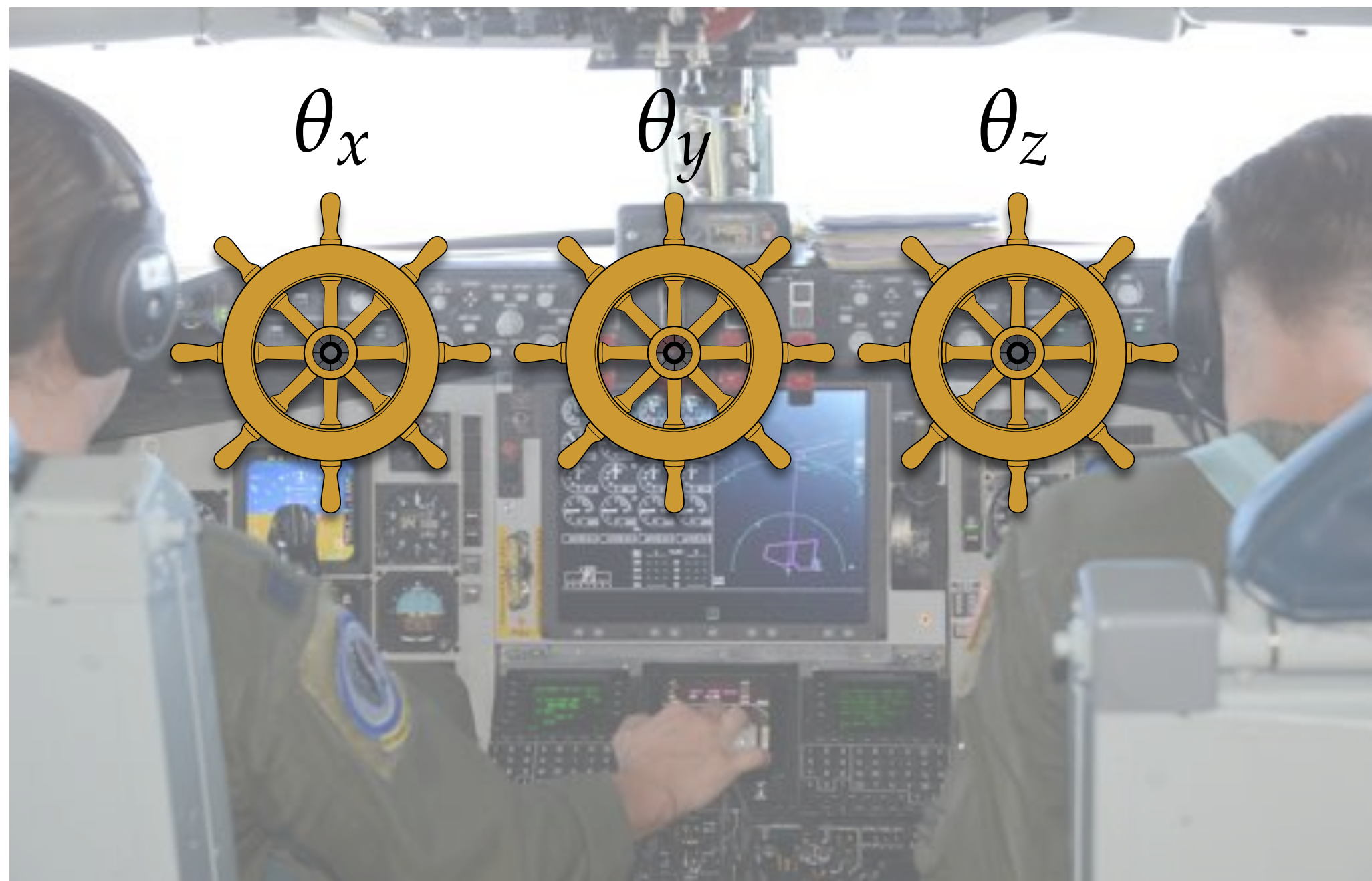
- Simplifying matrix from previous slide, we get

no matter how we adjust θ_x , θ_z ,
can only rotate in one plane!

$$\begin{bmatrix} 0 & 0 & 1 \\ \sin(\theta_x + \theta_z) & \cos(\theta_x + \theta_z) & 0 \\ -\cos(\theta_x + \theta_z) & \sin(\theta_x + \theta_z) & 0 \end{bmatrix}$$

Q: What does this matrix do?

- We are now “locked” into a single axis of rotation
- Not a great design for airplane controls!



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 :

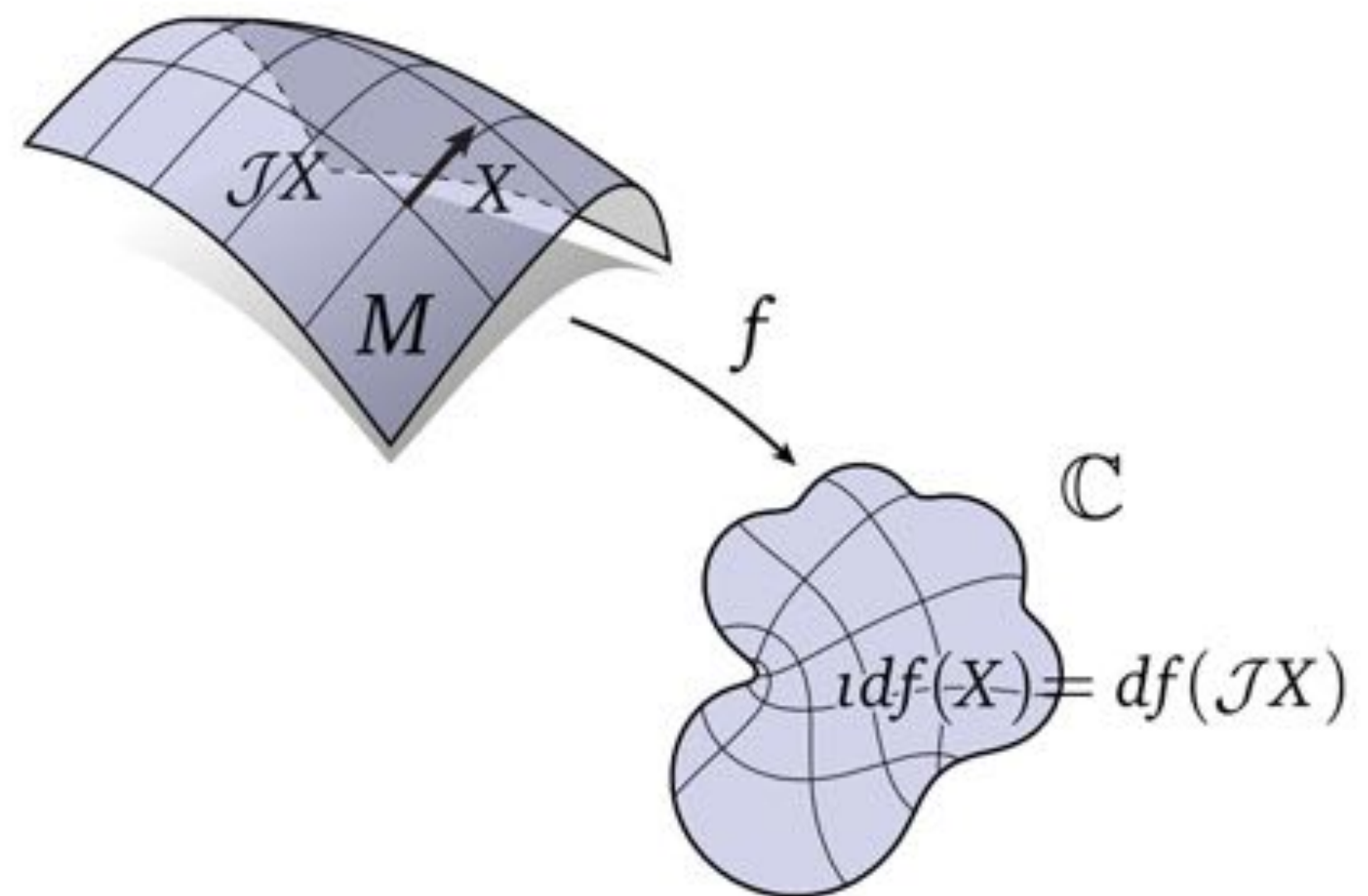
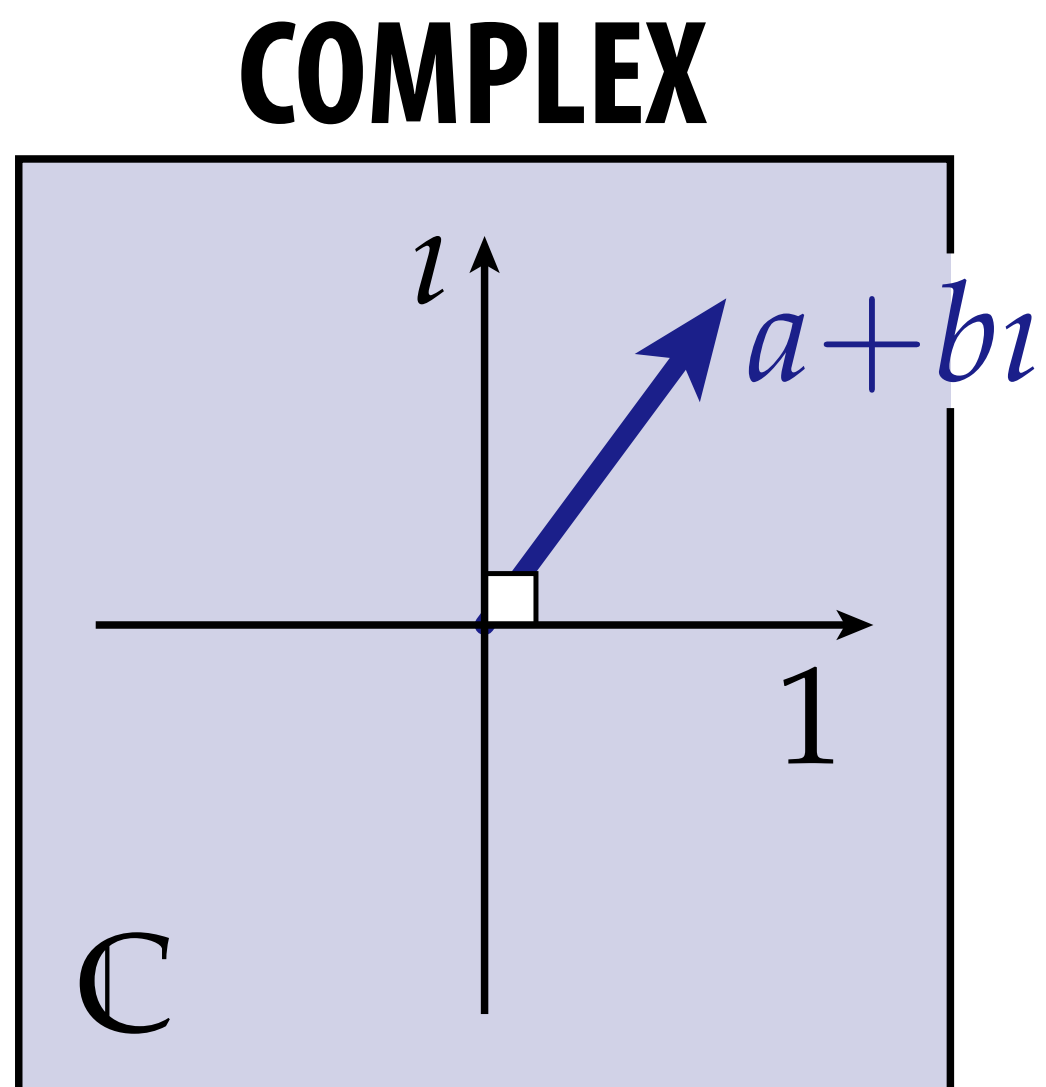
$$\begin{bmatrix} \cos \theta + u_x^2 (1 - \cos \theta) & u_x u_y (1 - \cos \theta) - u_z \sin \theta & u_x u_z (1 - \cos \theta) + u_y \sin \theta \\ u_y u_x (1 - \cos \theta) + u_z \sin \theta & \cos \theta + u_y^2 (1 - \cos \theta) & u_y u_z (1 - \cos \theta) - u_x \sin \theta \\ u_z u_x (1 - \cos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

Just memorize this matrix! :-)

...we'll see a much easier way, later on.

Complex Analysis—Motivation

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

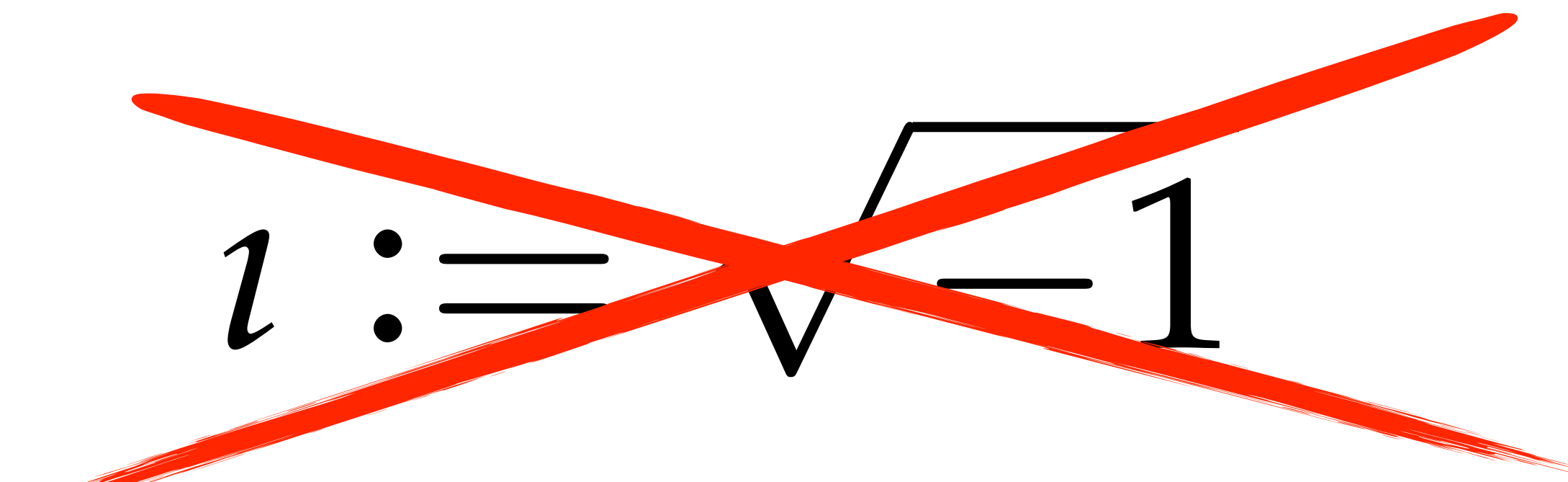


Truly: no good reason to use 2D vectors instead of complex numbers...

DON'T: Think of these numbers as “complex.”

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

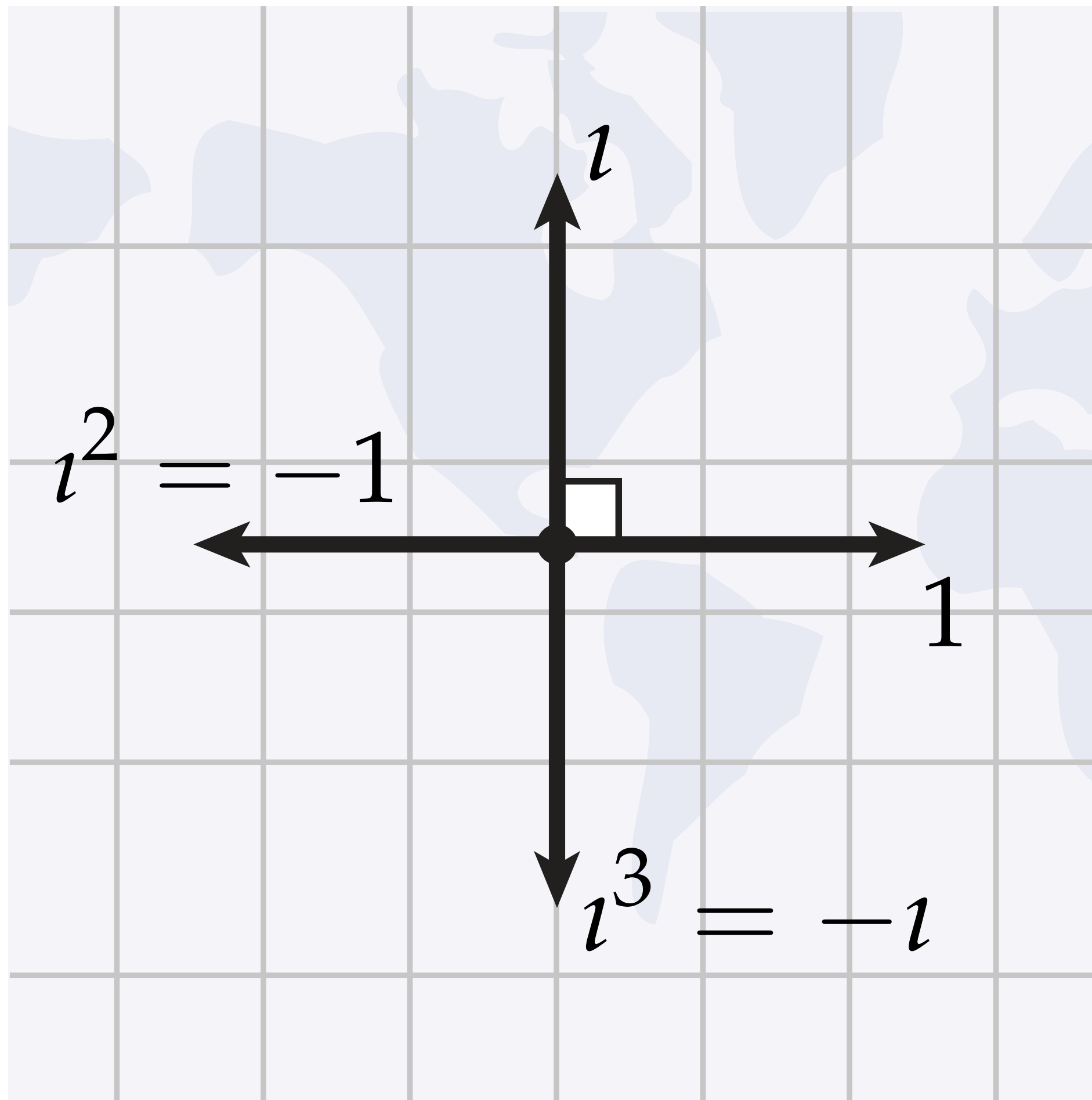
Imaginary Unit


$$i := \sqrt{-1}$$

nonsense!

More importantly: obscures geometric meaning.

Imaginary Unit—Geometric Description

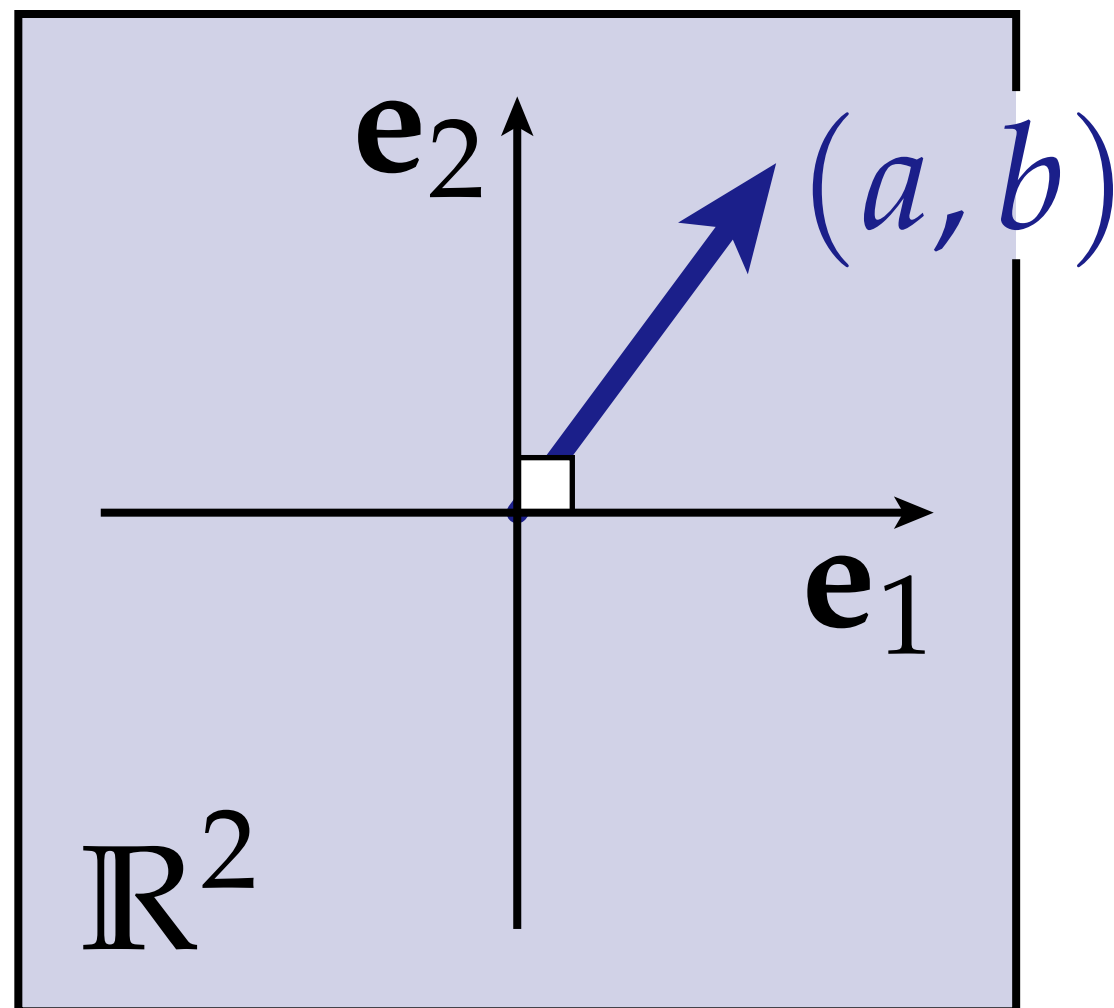


**Imaginary unit is just a quarter-turn
in the counter-clockwise direction.**

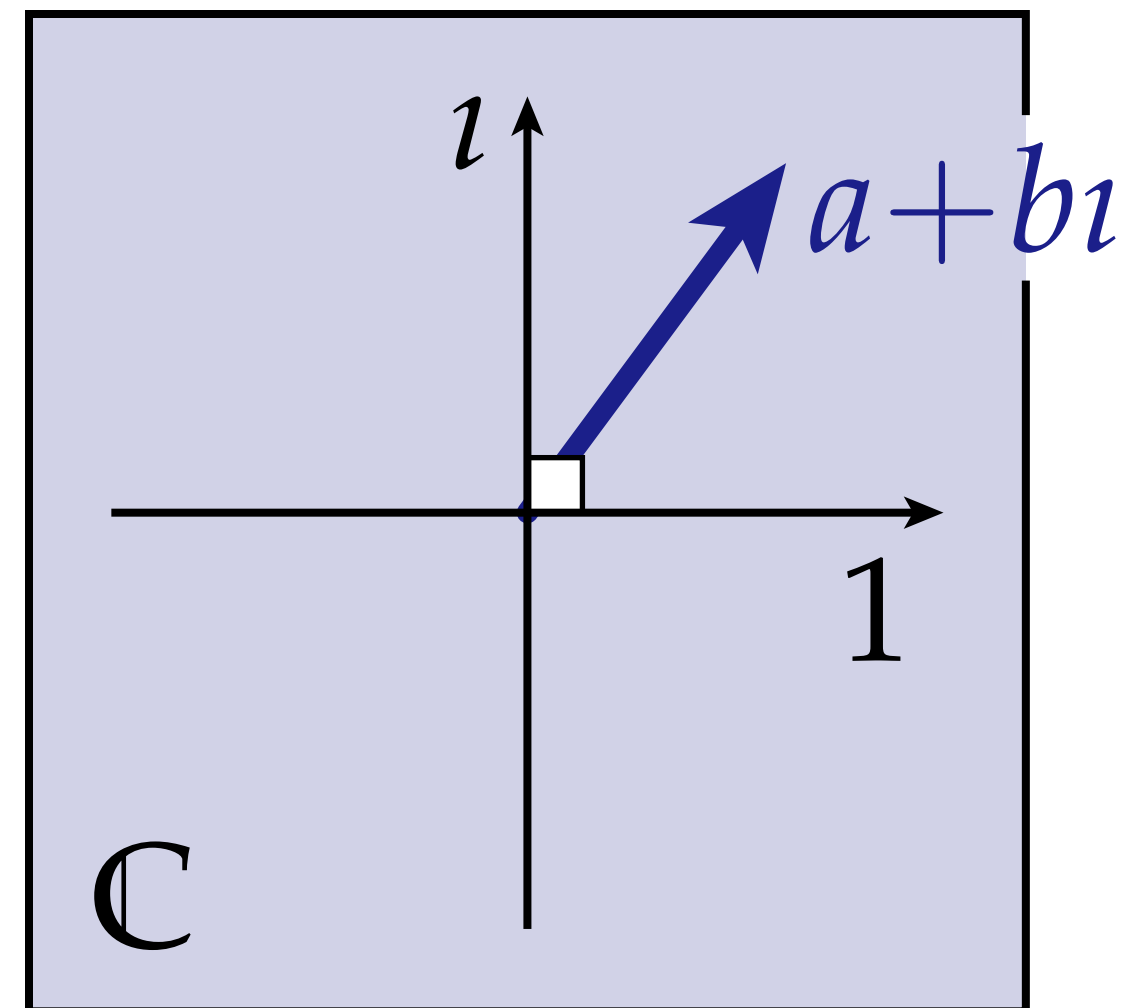
Complex Numbers

- Complex numbers are then just 2-vectors
- Instead of e_1, e_2 , use “1” and “i” to denote the two bases
- Otherwise, behaves exactly like a real 2-dimensional space

REAL



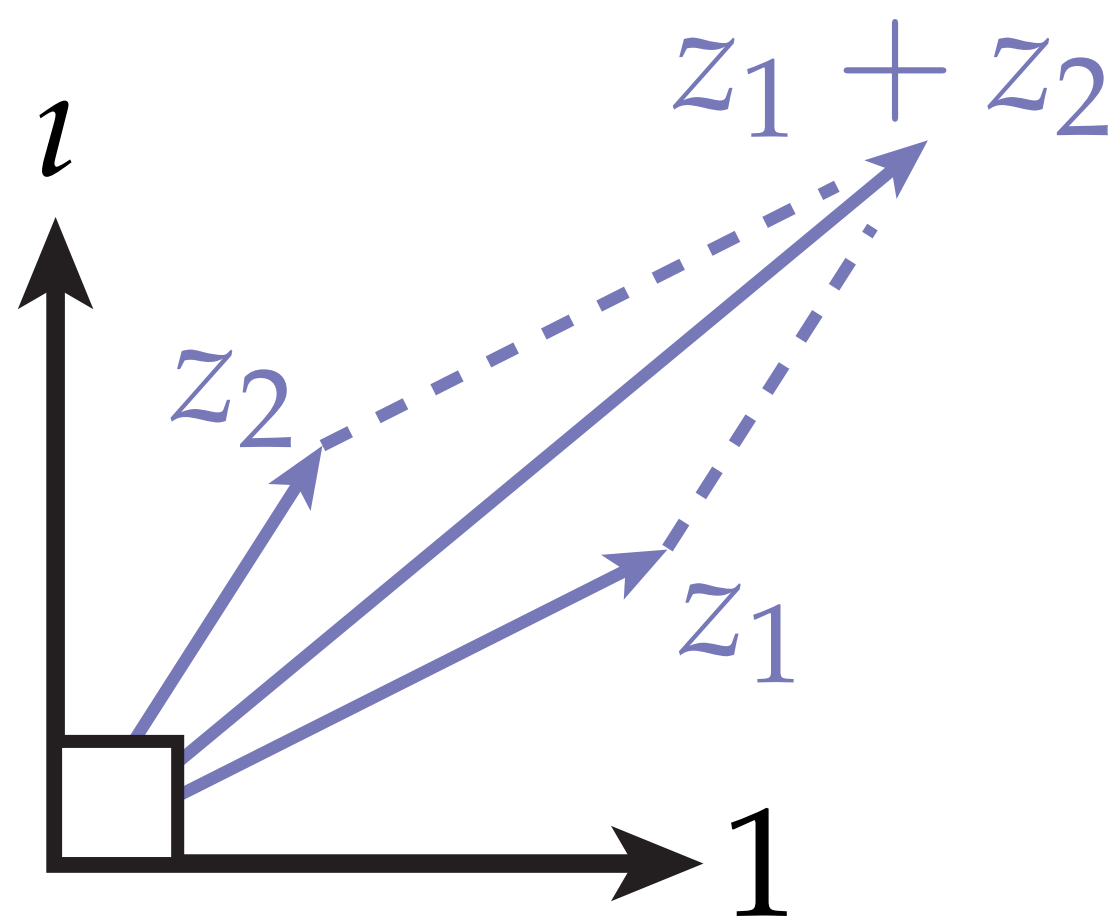
COMPLEX



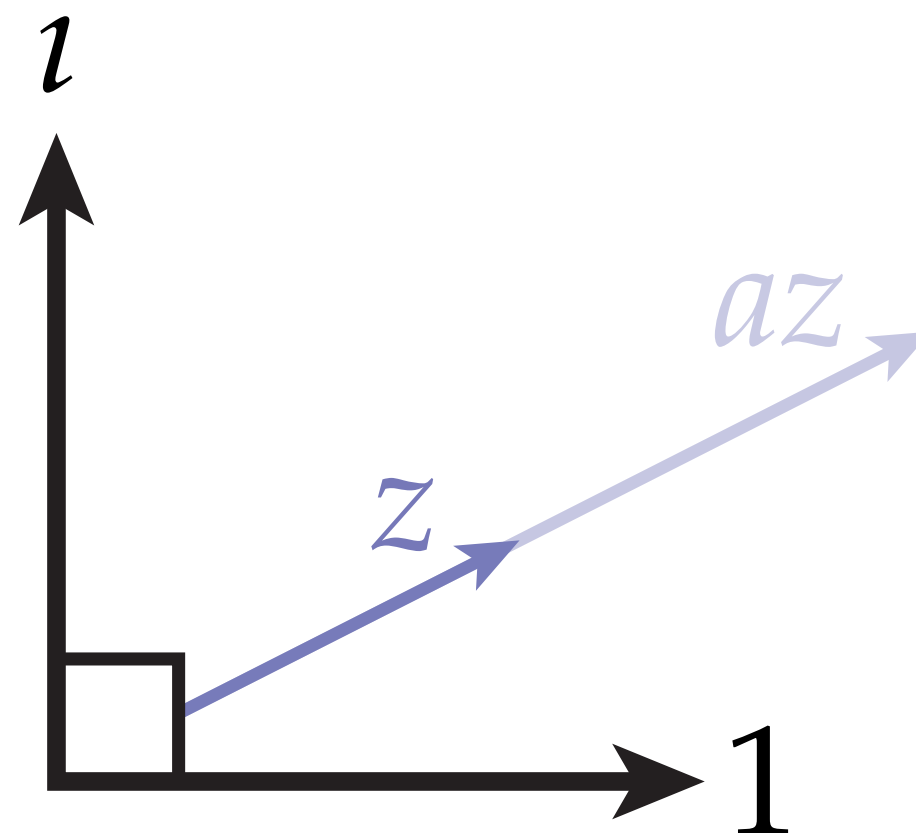
- ...except that we're also going to get a very useful new notion of the product between two vectors.

Complex Arithmetic

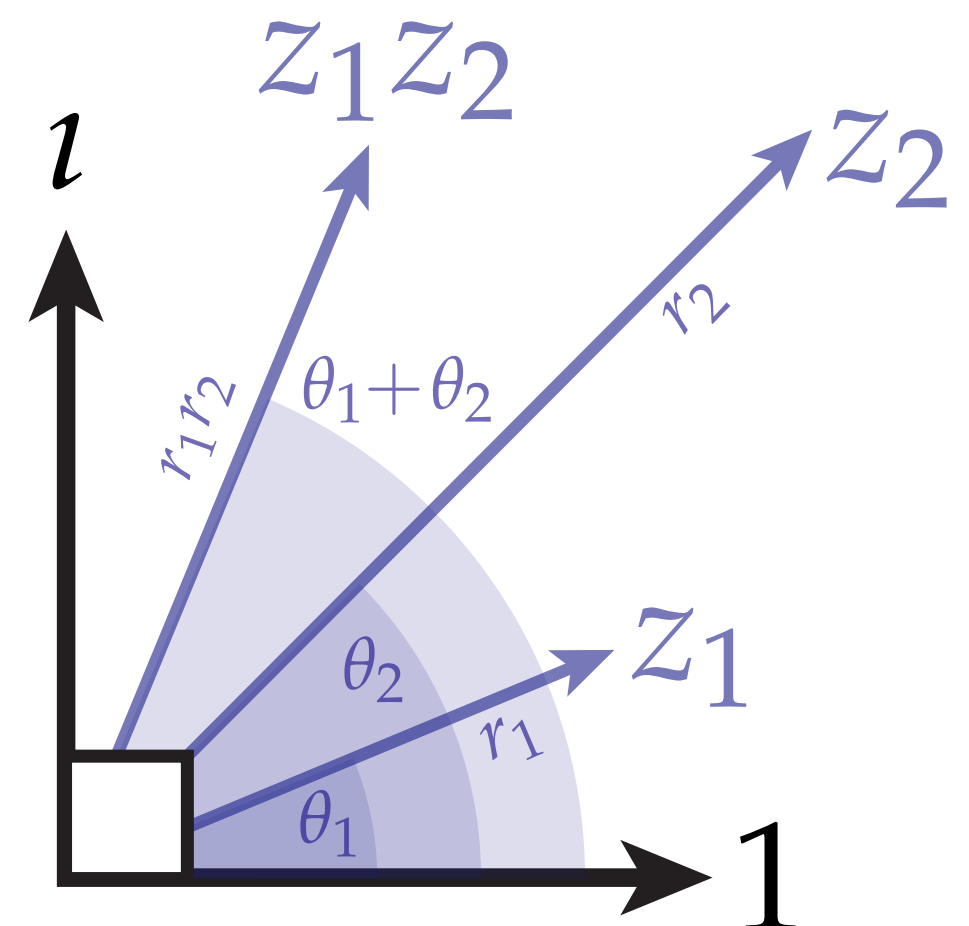
- Same operations as before, plus one more:



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 quite how it really works, but basic idea is right.

Complex Product—Rectangular Form

- Complex product in “rectangular” coordinates (1, i):

$$z_1 = (a + bi)$$

$$z_2 = (c + di)$$

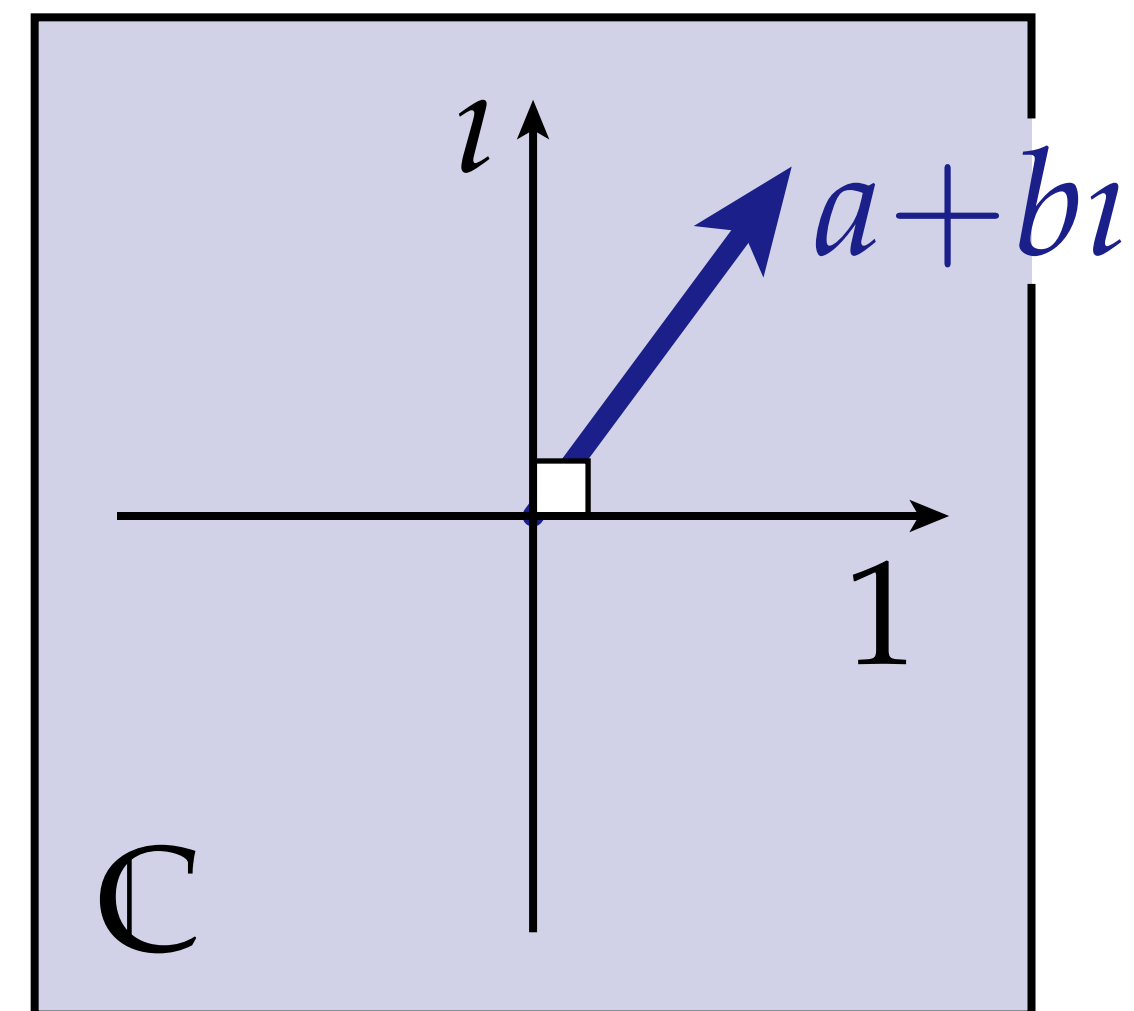
$$z_1 z_2 = ac + adi + bci + bd \overset{\text{two quarter turns—}}{\underset{\text{same as } -1}{i^2}} =$$

$$(ac - bd) + (ad + bc)i.$$

↑
“real part”
 $\text{Re}(z_1 z_2)$

↑
“imaginary part”
 $\text{Im}(z_1 z_2)$

- 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:

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

- Specialization of Euler's formula:

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

- Can use to “implement” complex product:

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

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

(as with real exponentiation, exponents add)



Leonhard Euler
(1707–1783)

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

2D Rotations: Matrices vs. Complex

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

REAL / RECTANGULAR	COMPLEX / POLAR
$\mathbf{u} = (x, y)$ $\mathbf{A} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}$ $\mathbf{B} = \begin{bmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{bmatrix}$ $\mathbf{A}\mathbf{u} = \begin{bmatrix} x \cos \theta - y \sin \theta \\ x \sin \theta + y \cos \theta \end{bmatrix}$ $\mathbf{B}\mathbf{A}\mathbf{u} = \begin{bmatrix} (x \cos \theta - y \sin \theta) \cos \phi - (x \sin \theta + y \cos \theta) \sin \phi \\ (x \cos \theta - y \sin \theta) \sin \phi + (x \sin \theta + y \cos \theta) \cos \phi \end{bmatrix}$ $= \dots \text{some trigonometry} \dots =$ $\mathbf{B}\mathbf{A}\mathbf{u} = \begin{bmatrix} x \cos(\theta + \phi) - y \sin(\theta + \phi) \\ x \sin(\theta + \phi) + y \cos(\theta + \phi) \end{bmatrix}.$	$u = re^{i\alpha}$ $a = e^{i\theta}$ $b = e^{i\phi}$ $abu = re^{i(\alpha + \theta + \phi)}.$

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:

"H" is for Hamilton! $\mathbb{H} := \text{span}(\{1, i, j, k\})$

$$q = a + bi + cj + dk \in \mathbb{H}$$

- Quaternion product determined by

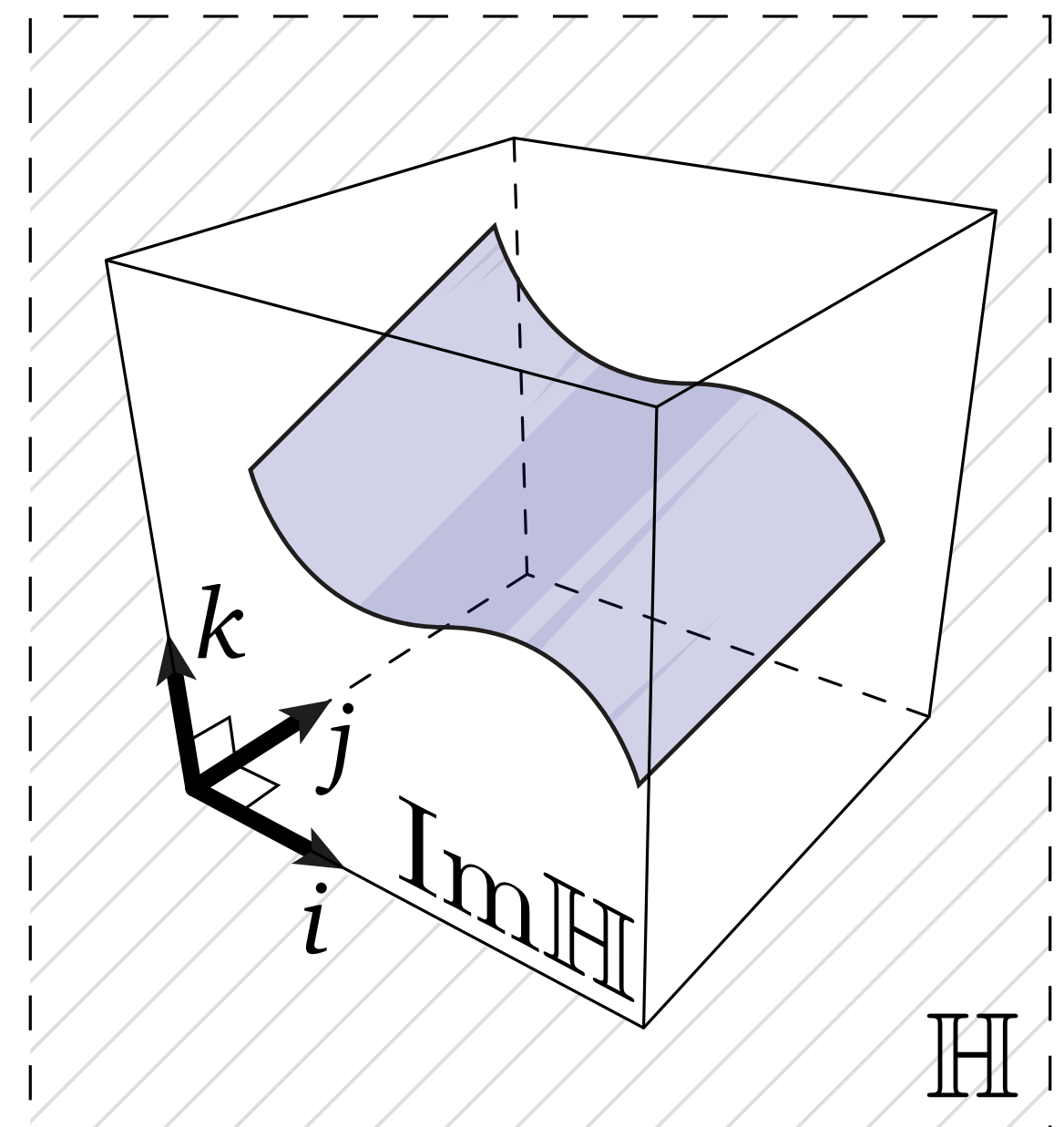
$$i^2 = j^2 = k^2 = ijk = -1$$

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

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

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

(Why might it make sense that it doesn't commute?)



Quaternion Product in Components

- Given two quaternions

$$q = a_1 + b_1i + c_1j + d_1k$$

$$p = a_2 + b_2i + c_2j + d_2k$$

- Can express their product as

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

...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?

$$(x, y, z) \mapsto 0 + xi + yj + zk$$

- Alternatively, can think of a quaternion as a pair

$$\left(\begin{array}{c} \text{scalar} \\ \cap \\ \mathbb{R} \end{array}, \begin{array}{c} \text{vector} \\ \cap \\ \mathbb{R}^3 \end{array} \right) \in \mathbb{H}$$

- 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 \mathbb{R}^3 , gets even simpler:

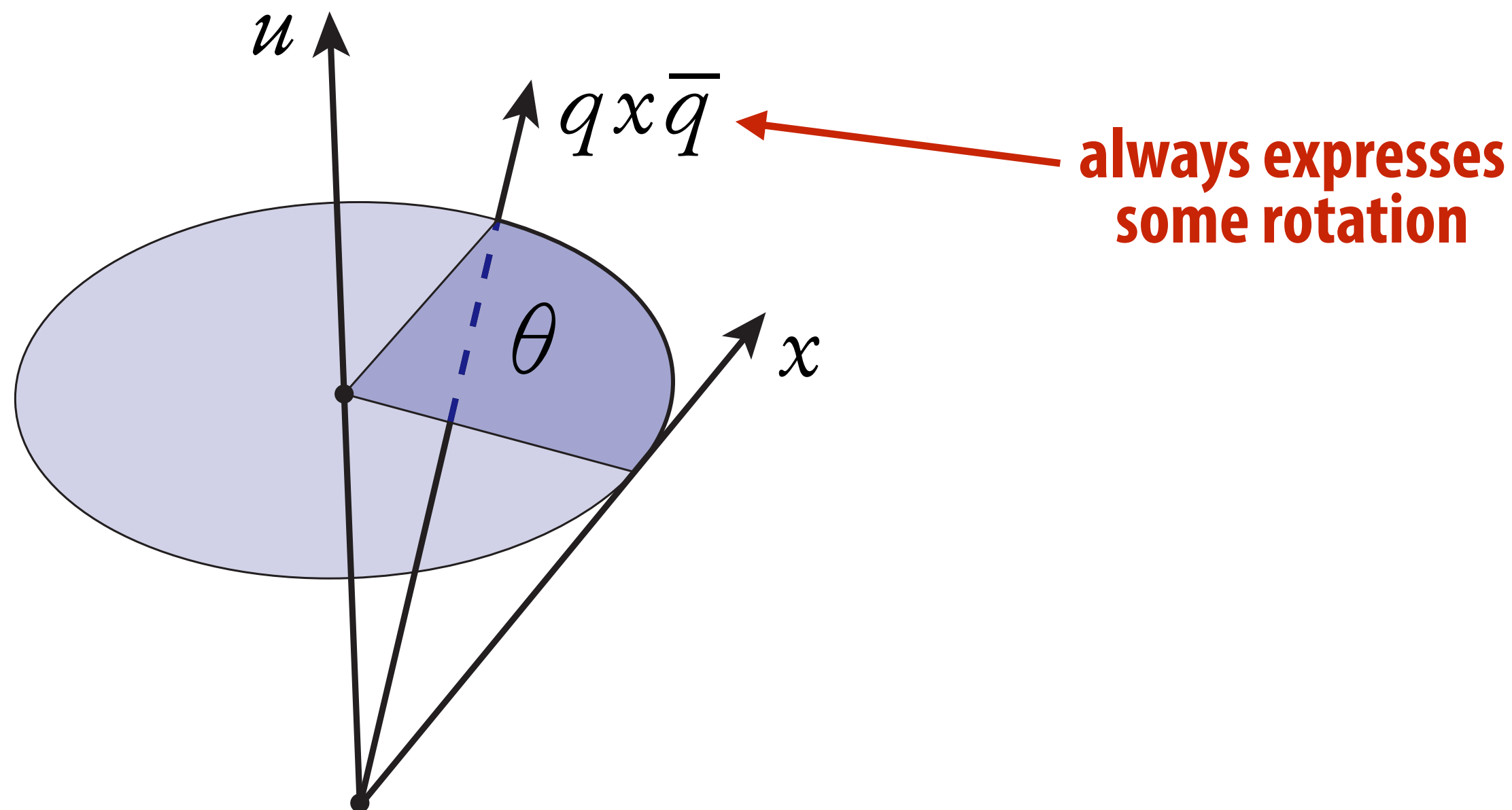
$$\mathbf{u}\mathbf{v} = \mathbf{u} \times \mathbf{v} - \mathbf{u} \cdot \mathbf{v}$$

3D Transformations via Quaternions

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

$$x \in \text{Im}(\mathbb{H})$$

$$q \in \mathbb{H}, \quad |q|^2 = 1$$



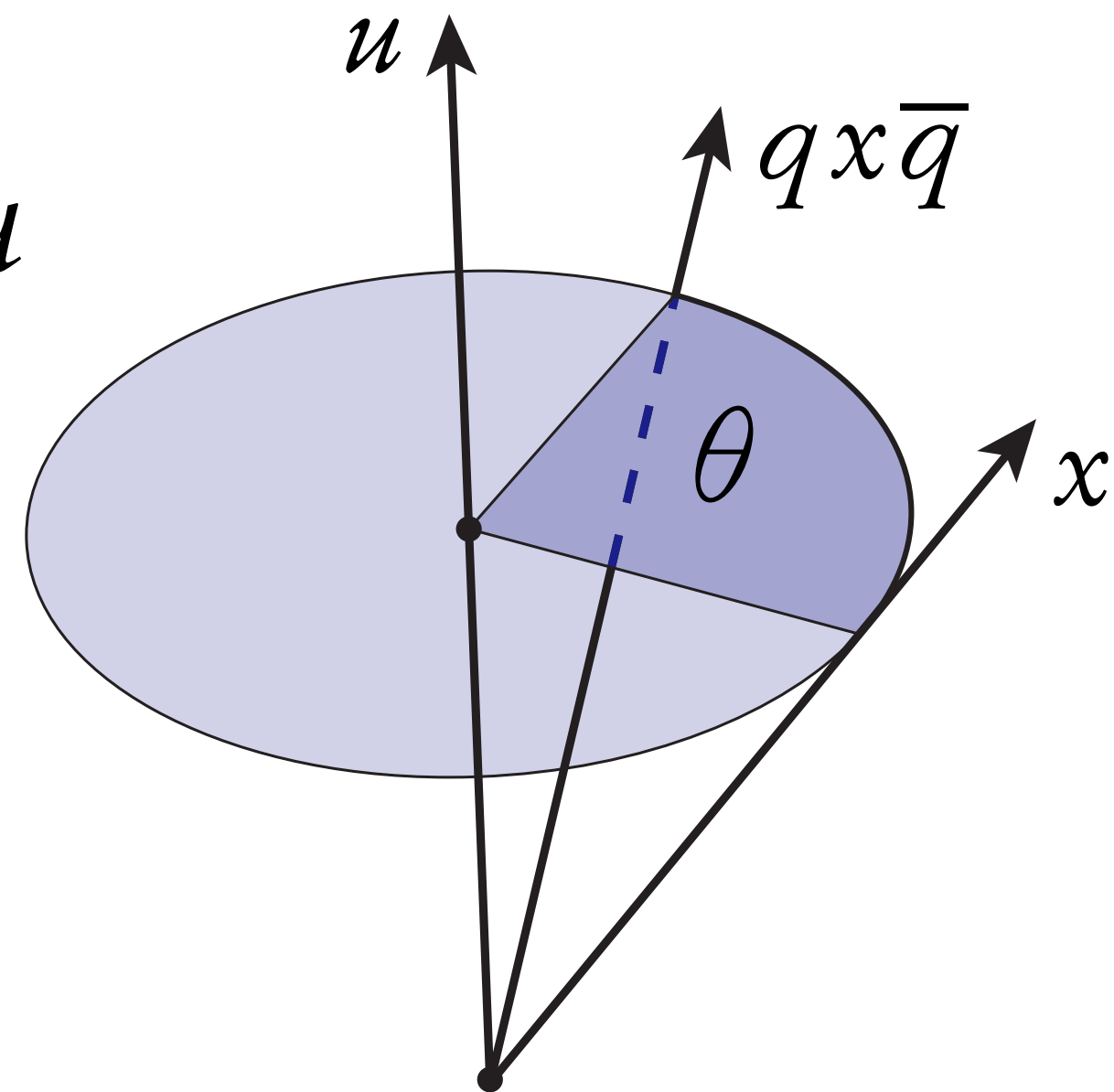
Rotation from Axis/Angle, Revisited

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

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

angle

axis



- Much easier to remember (and manipulate) than matrix!

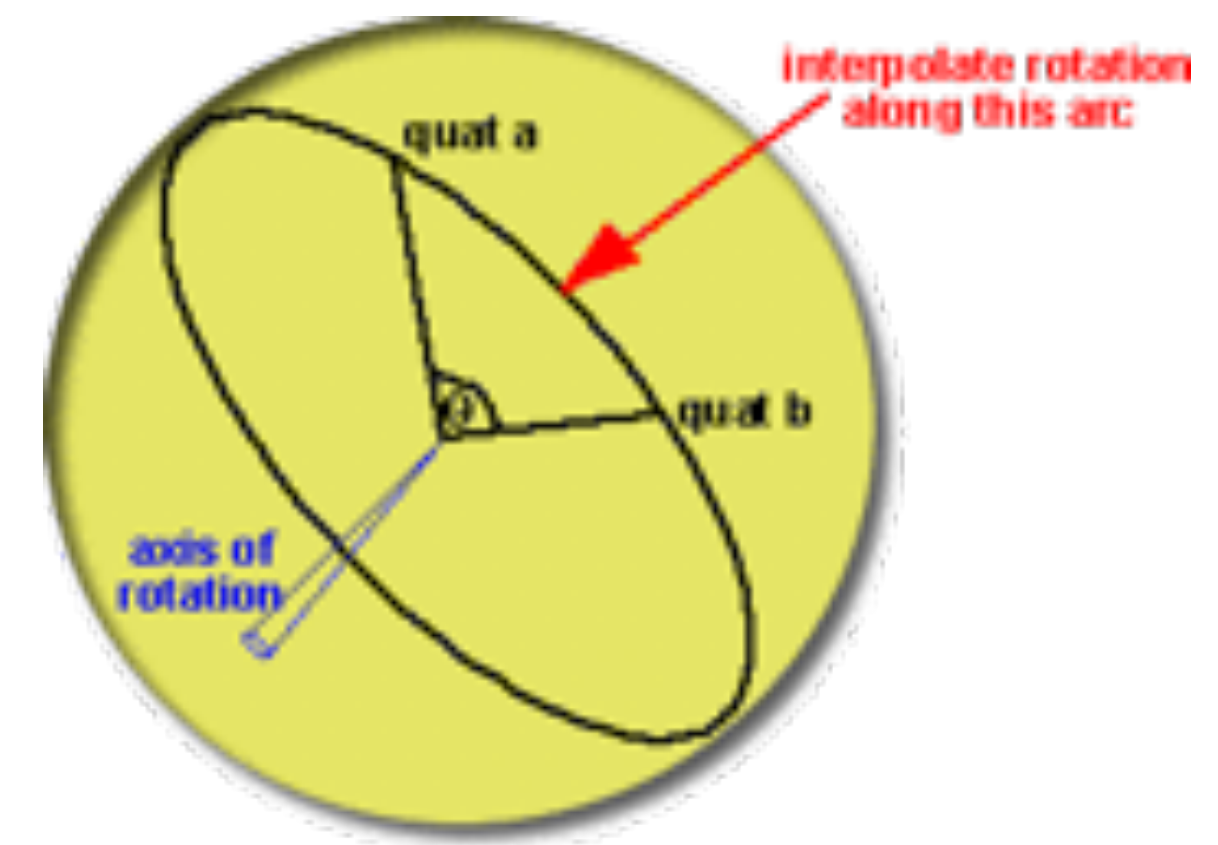
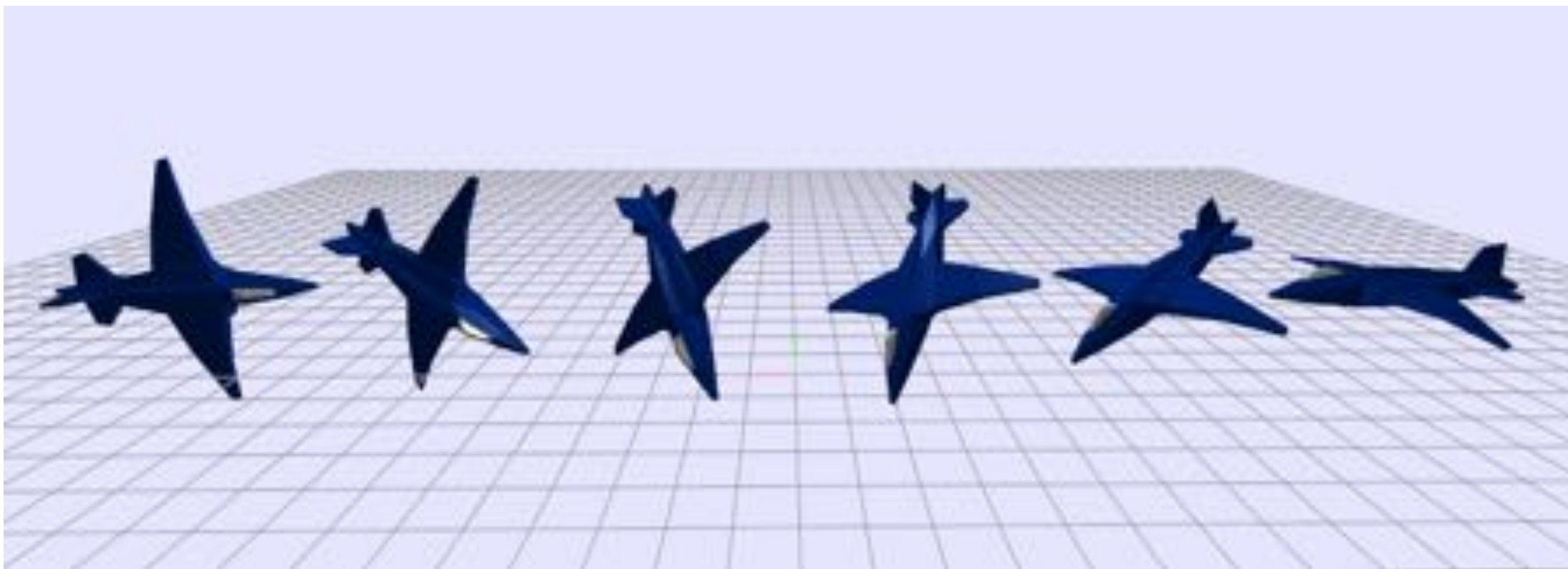
$$\begin{bmatrix} \cos \theta + u_x^2 (1 - \cos \theta) & u_x u_y (1 - \cos \theta) - u_z \sin \theta & u_x u_z (1 - \cos \theta) + u_y \sin \theta \\ u_y u_x (1 - \cos \theta) + u_z \sin \theta & \cos \theta + u_y^2 (1 - \cos \theta) & u_y u_z (1 - \cos \theta) - u_x \sin \theta \\ u_z u_x (1 - \cos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

Note: the quaternion conjugate is the same as the inverse for a unit quaternion. Can you create an inverse?

Interpolating Rotations

- Suppose we want to smoothly interpolate between two rotations (e.g., orientations of an airplane)
- Interpolating Euler angles can yield strange-looking paths, non-uniform rotation speed, ...
- Simple solution* w/ quaternions: “SLERP” (spherical linear interpolation):

$$\text{Slerp}(q_0, q_1, t) = q_0(q_0^{-1}q_1)^t, \quad t \in [0, 1]$$

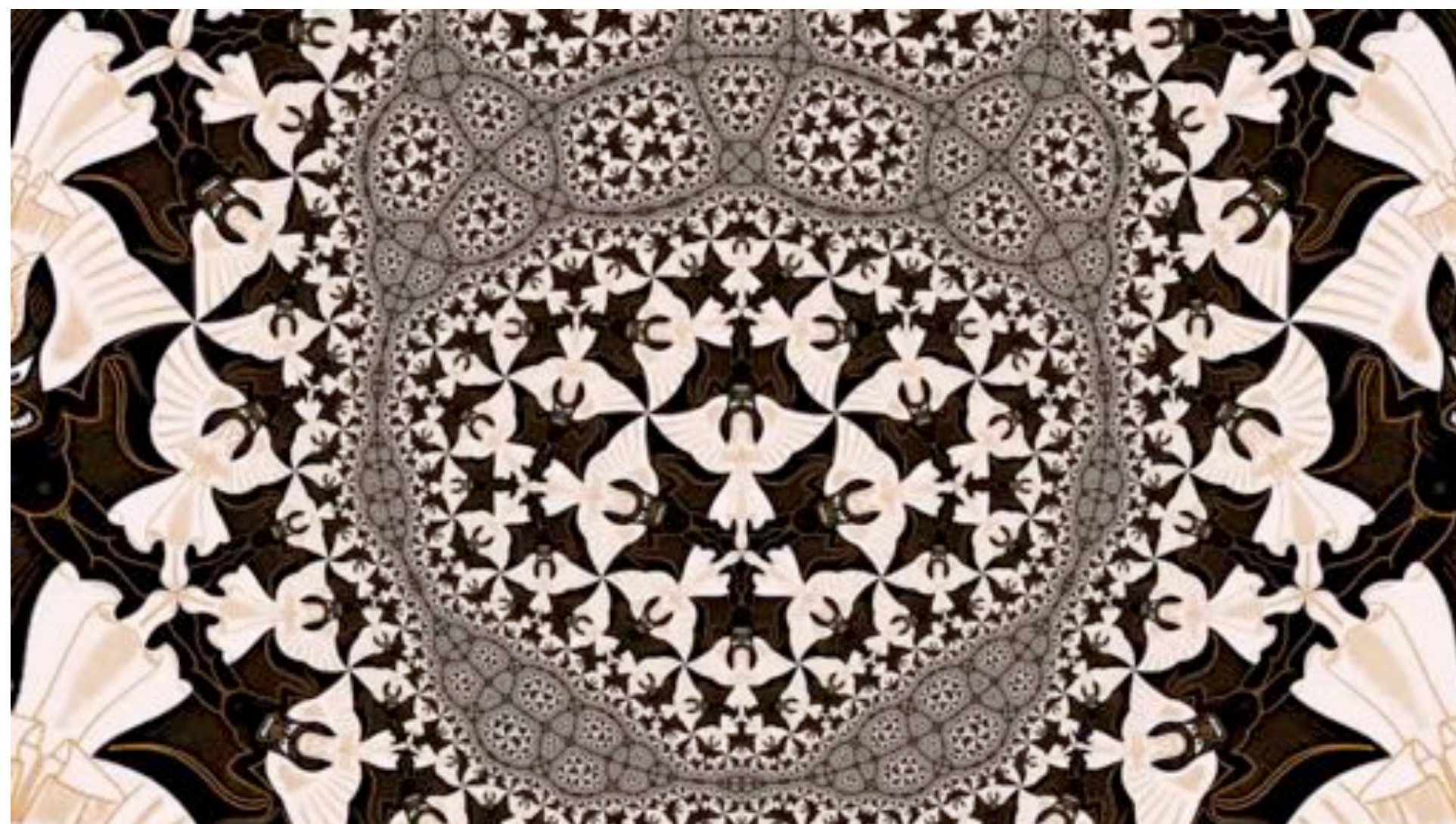
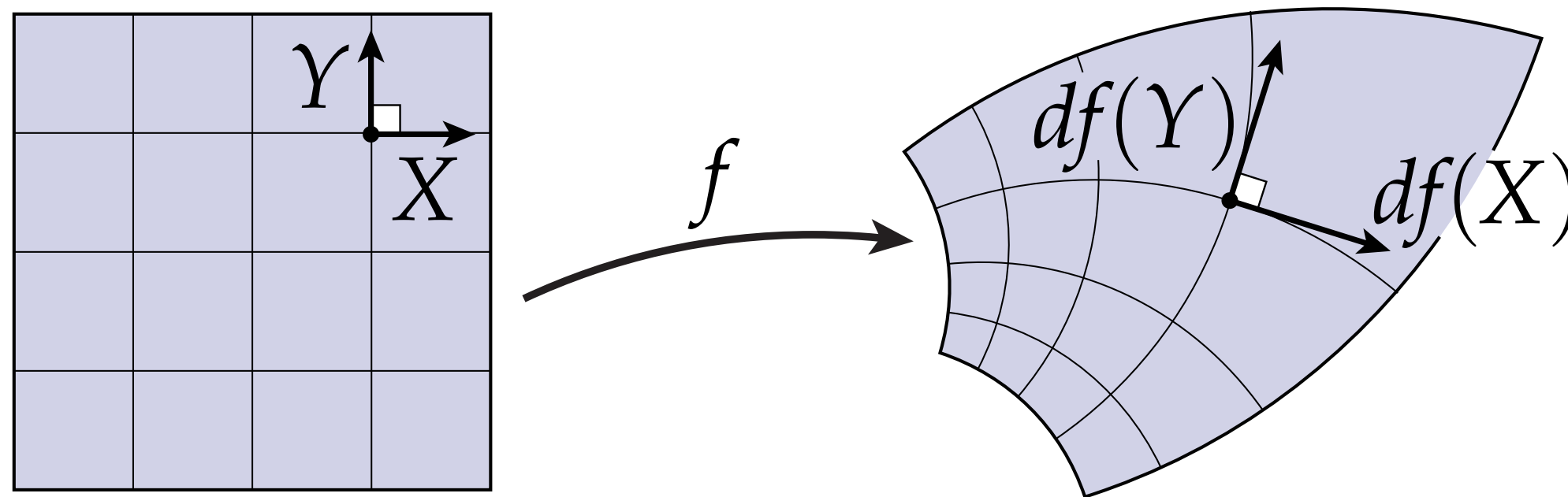


*Shoemake 1985, “Animating Rotation with Quaternion Curves”

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

Generating Coordinates for Texture Maps

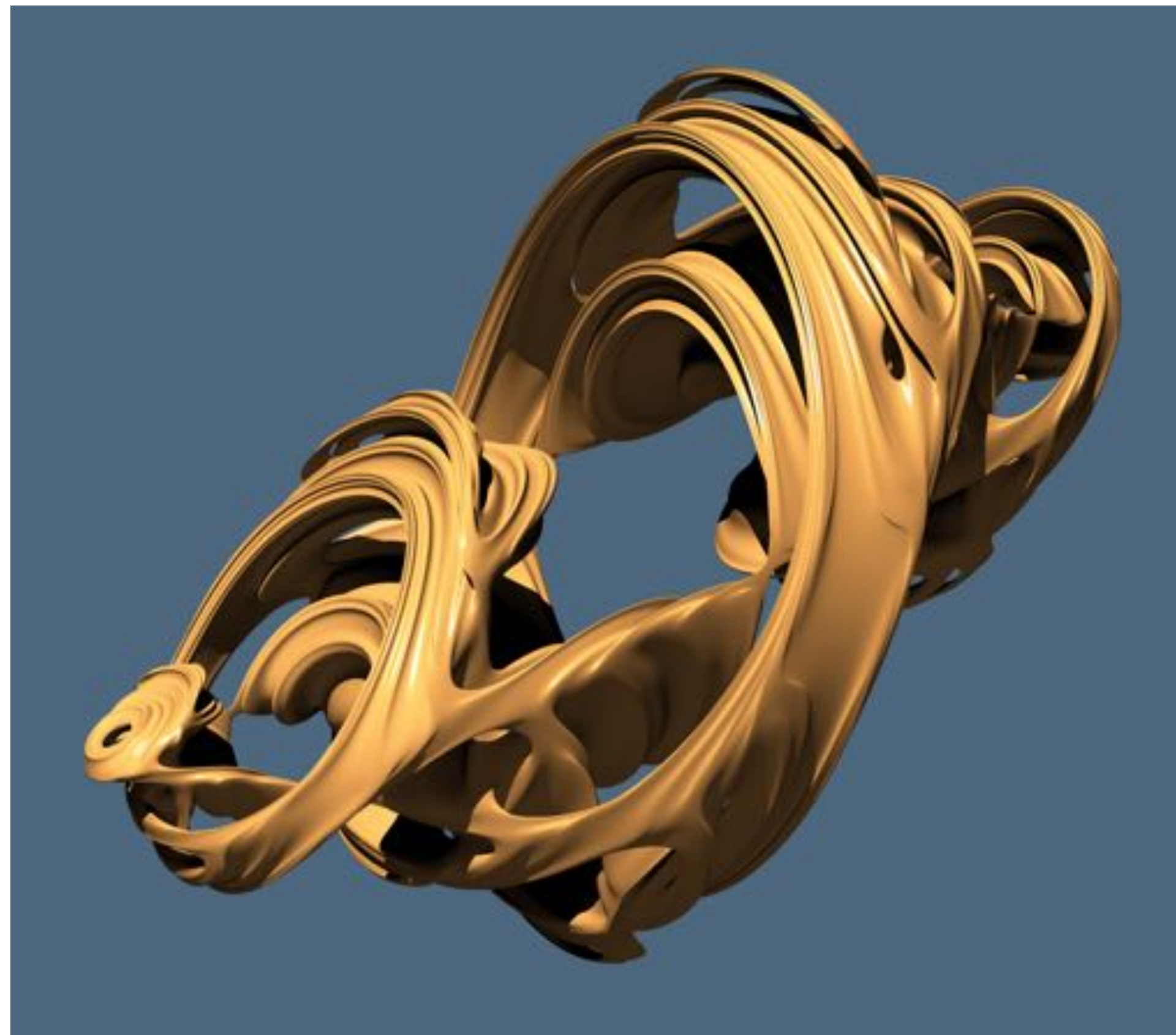
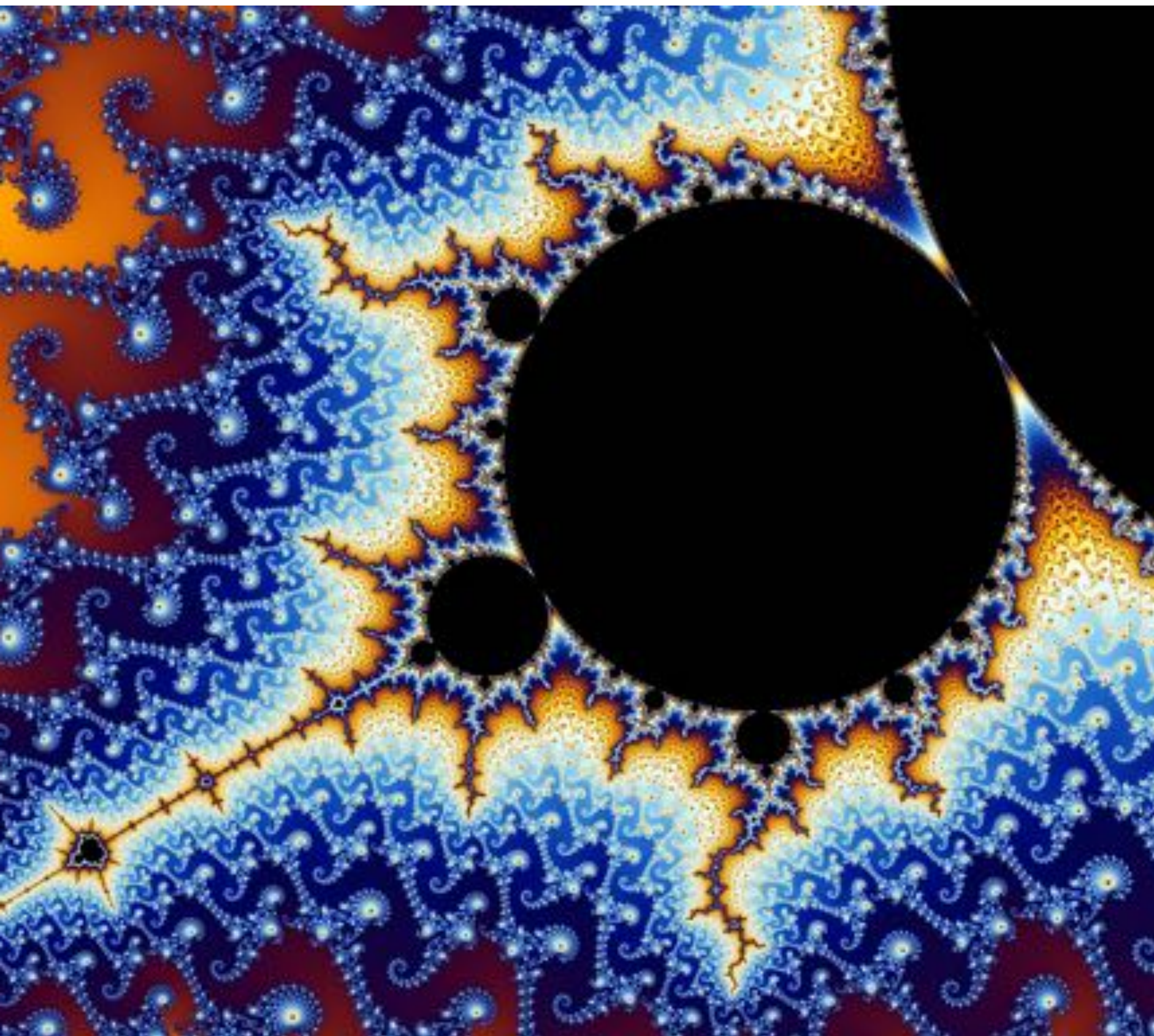
Complex numbers are natural language for angle-preserving (“conformal”) maps



Preserving angles in texture well-tuned to human perception...

Useless-But-Beautiful Example: Fractals

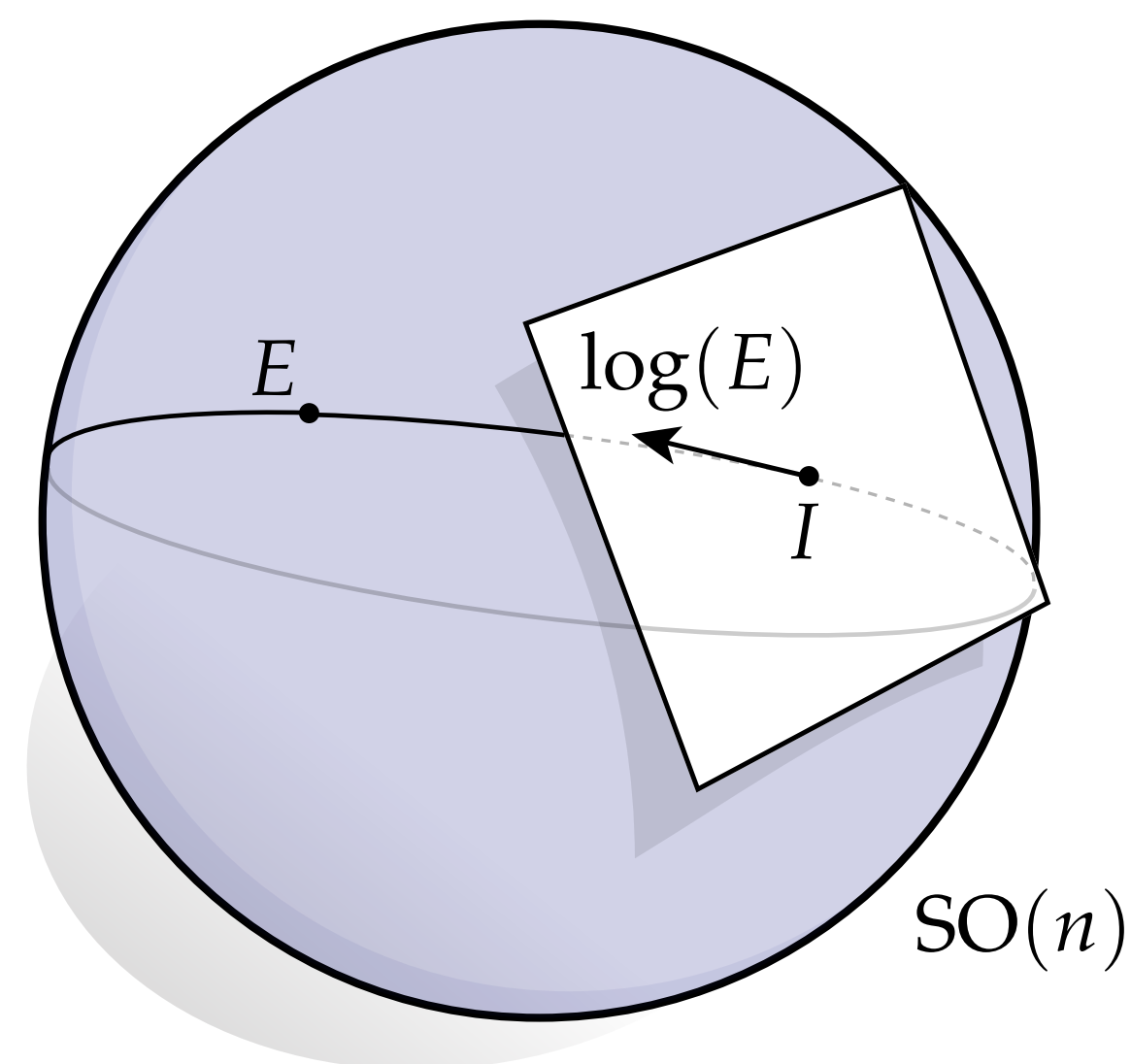
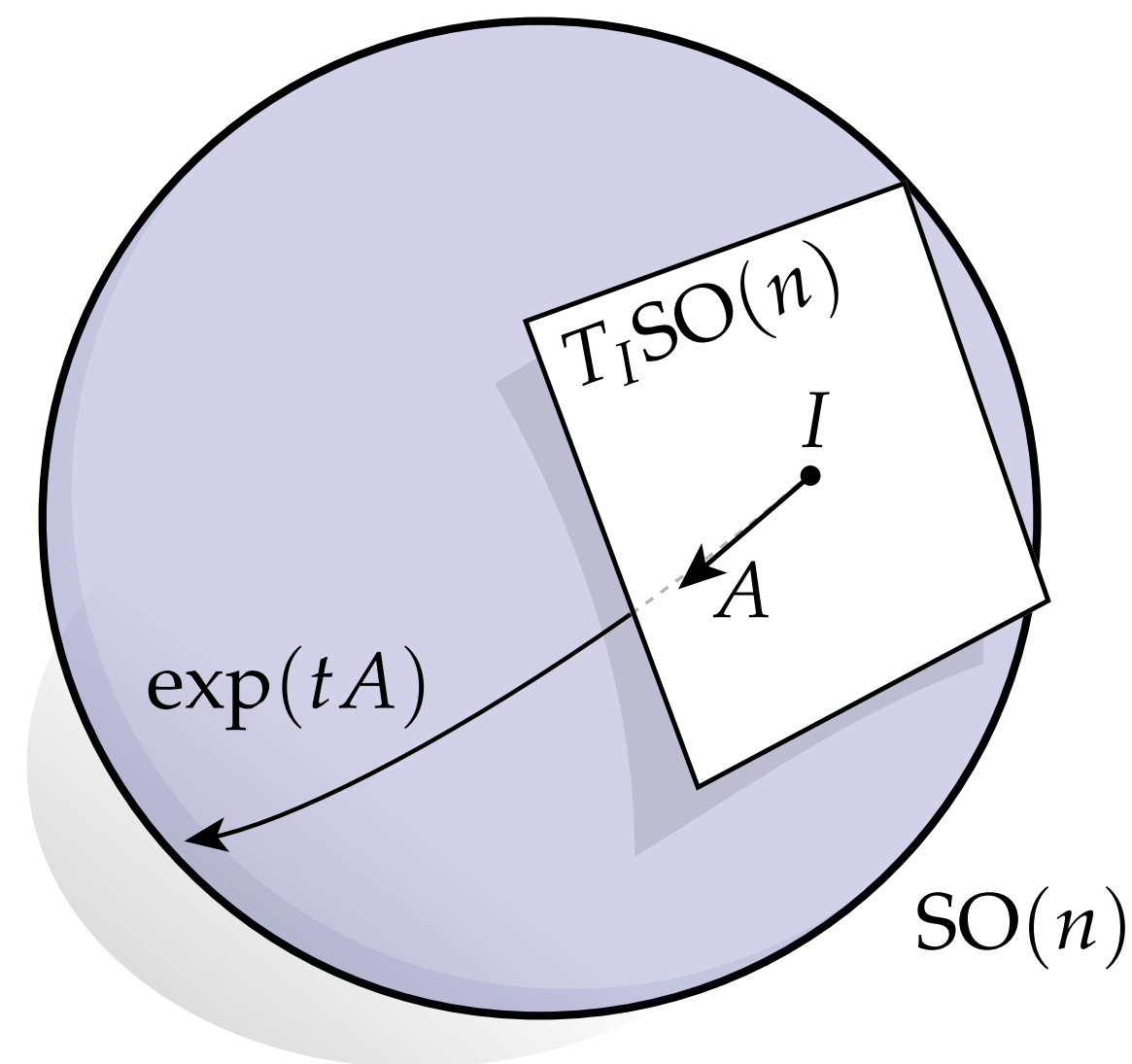
- Defined in terms of iteration on (hyper)complex numbers:



(Will see exactly how this works later in class.)

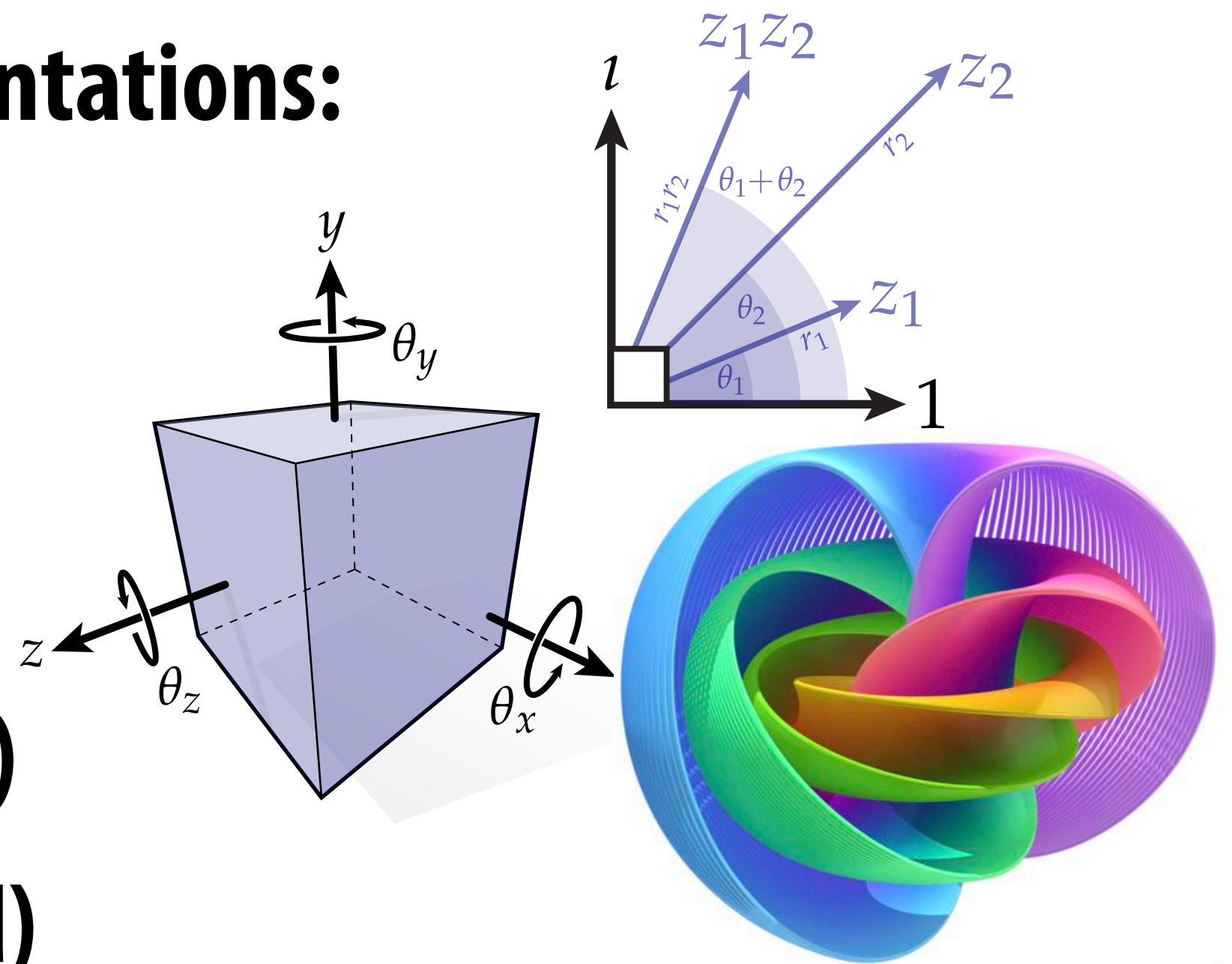
Not Covered: Lie algebras/Lie Groups

- Another super nice/useful perspective on rotations is via “Lie groups” and “Lie algebras”
- More than we have time to cover!
- Many benefits similar to quaternions (easy axis/angle representation, no gimbal lock, ...)
- Nice for encoding angles bigger than 2π
- Also very useful for taking averages of rotations
- (Very) short story:
 - exponential map takes you from axis/angle to rotation matrix
 - logarithmic map takes you from rotation matrix to axis/angle



Rotations and Complex Representations—Summary

- Rotations are surprisingly complicated in 3D!
- Today, looked at how complex representations help understand/work with rotations in 3D (& 2D)
- In general, many possible representations:
 - Euler angles
 - axis-angle
 - quaternions
 - Lie group/algebra (not covered)
 - geometric algebra (not covered)
- There's no “right” or “best” way—the more you know, the more you'll be able to do!



What else do we need to know to generate images like these?

GEOMETRY

How do we describe complex shapes (so far just triangles...)

RENDERING

How does light interact w/ materials to produce color?

ANIMATION

How do we describe the way things move?



("Moana", Disney 2016)

Course roadmap

