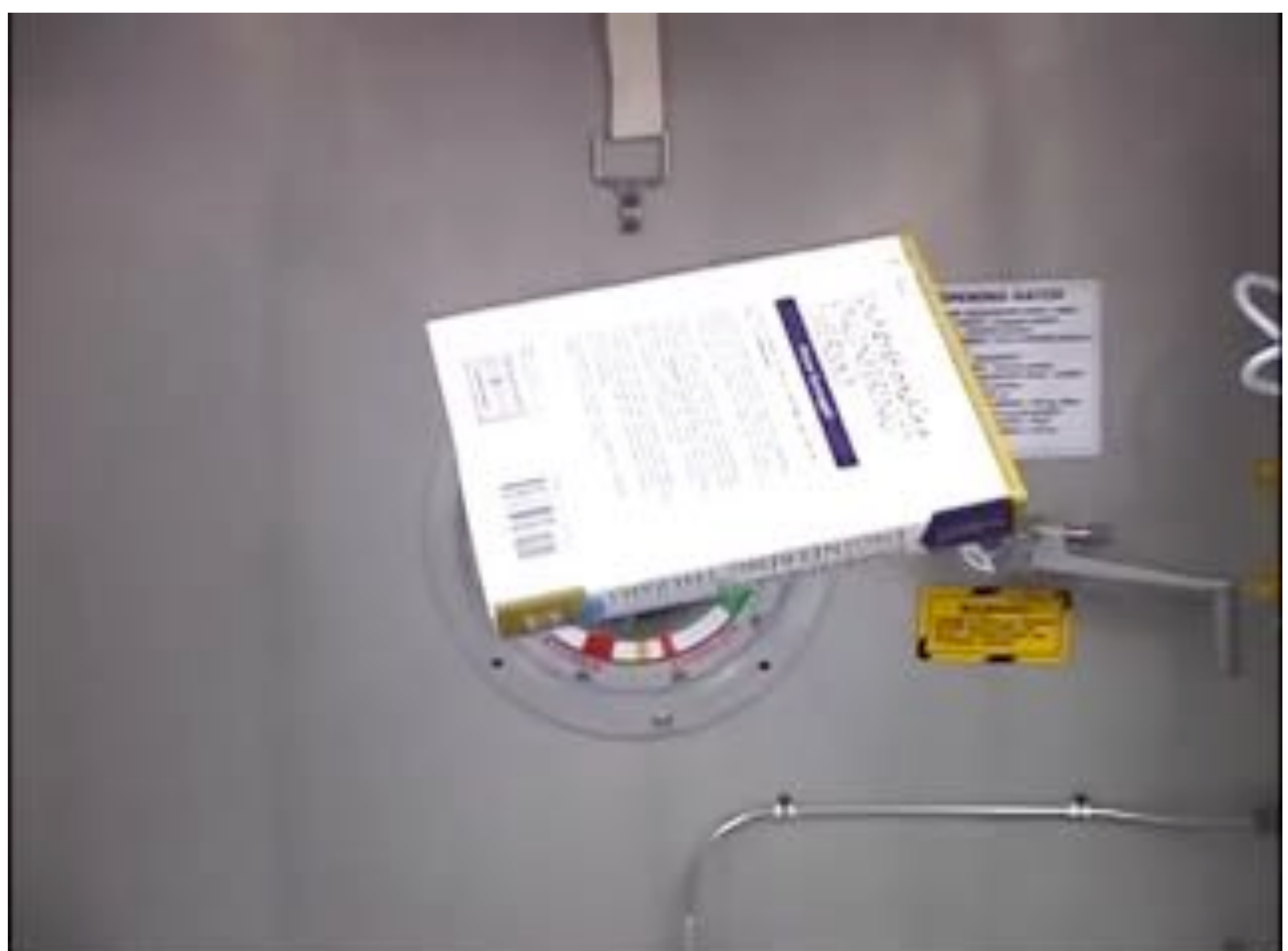
3D Rotations and Complex Representations

Computer Graphics

CMU 15-462/15-662, Spring 2018

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)



3D Rotations—Degrees of Freedom

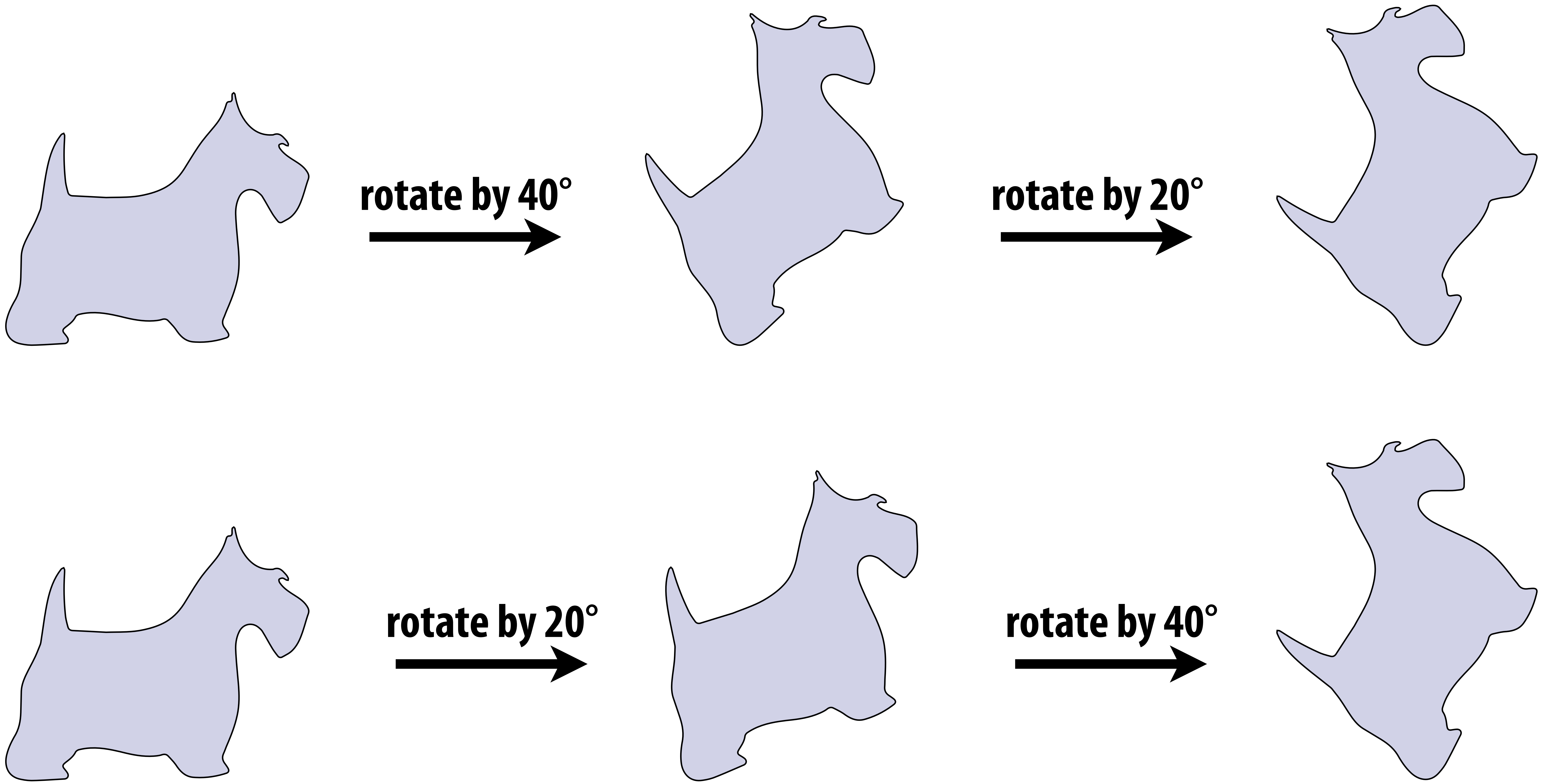
- How many numbers do we need to specify a rotation in 3D?
- For instance, we could use rotations around X, Y, Z. But do we need all three?
- Well, to rotate Pittsburgh to another city (say, São Paulo), we have to specify two numbers: latitude & longitude:
- Do we really need both latitude and longitude?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

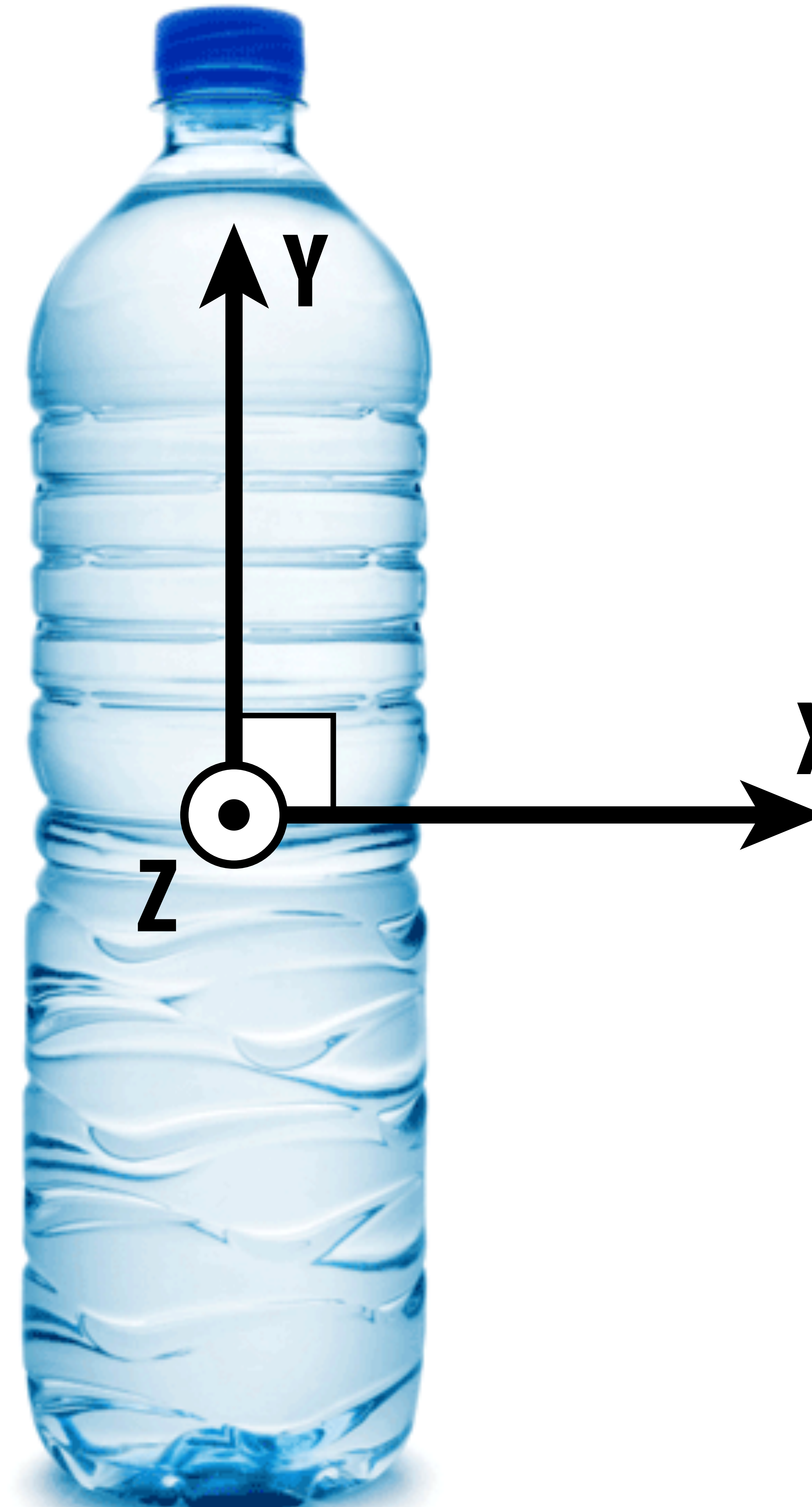
- In 2D, order of rotations doesn't matter:



Same result! ("2D rotations commute")

Commutativity of Rotations—3D

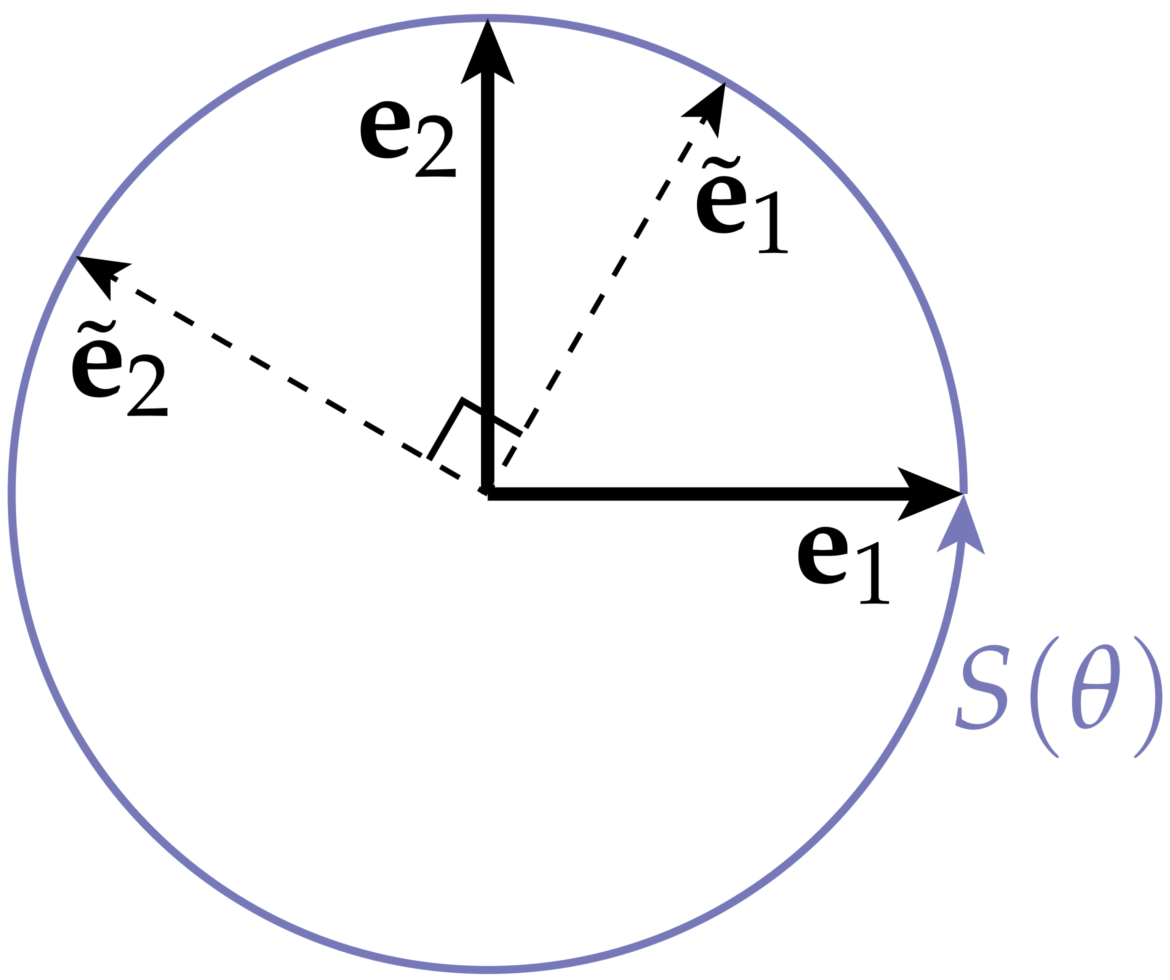
- What about in 3D?
- IN-CLASS ACTIVITY:
  - Rotate 90° around Y, then 90° around Z, then 90° around X
  - Rotate 90° around Z, then 90° around Y, then 90° around X
  - (Was there any difference?)



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

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(θ) 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 e1 rotated by θ?** $\tilde{\mathbf{e}}_1 = S(\theta)$
- **What's e2 rotated by θ?** $\tilde{\mathbf{e}}_2 = S(\theta + \pi/2)$
- **How about** $\mathbf{u} := a\mathbf{e}_1 + b\mathbf{e}_2$ **?**



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

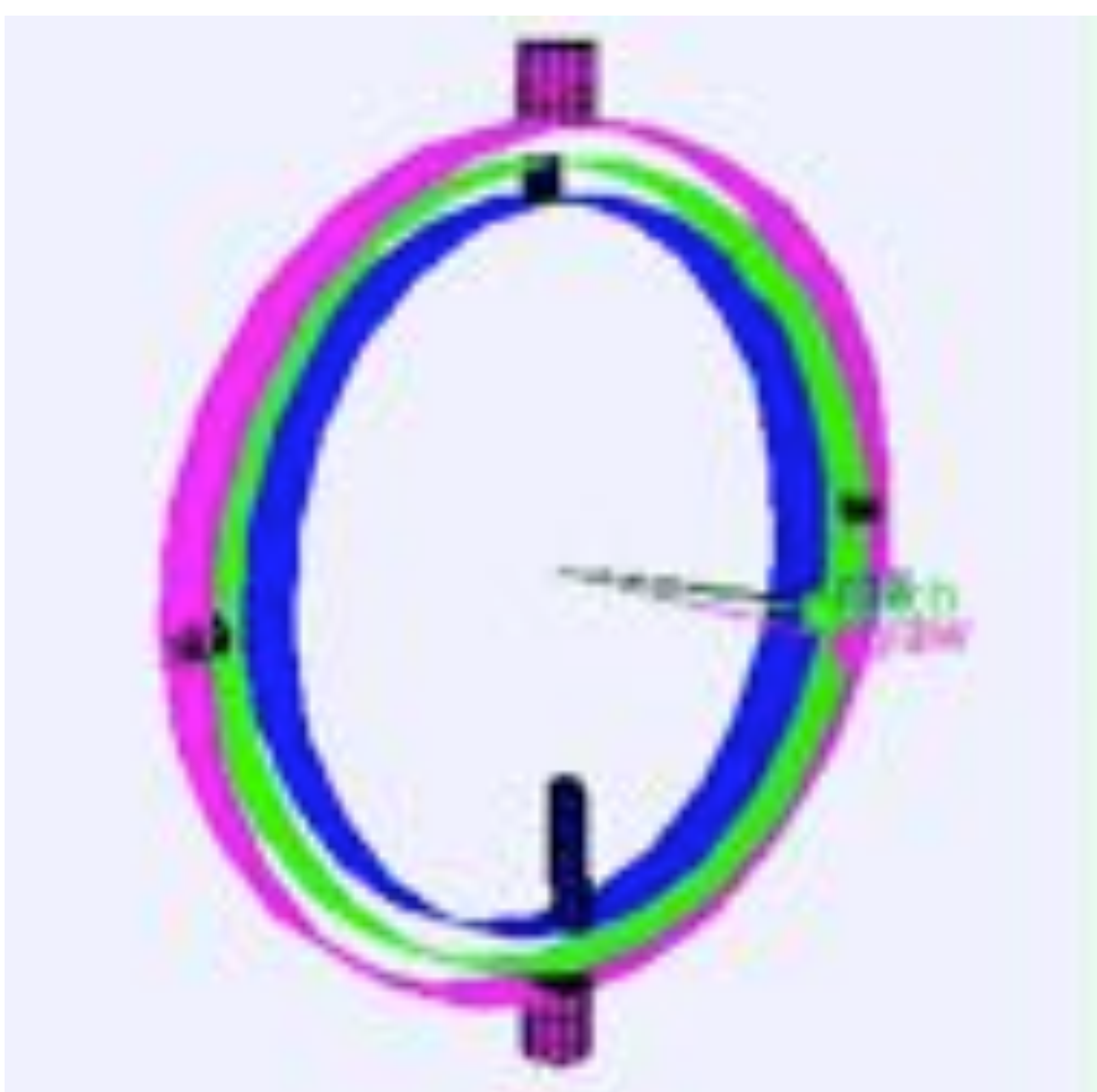
What then must the matrix look like?

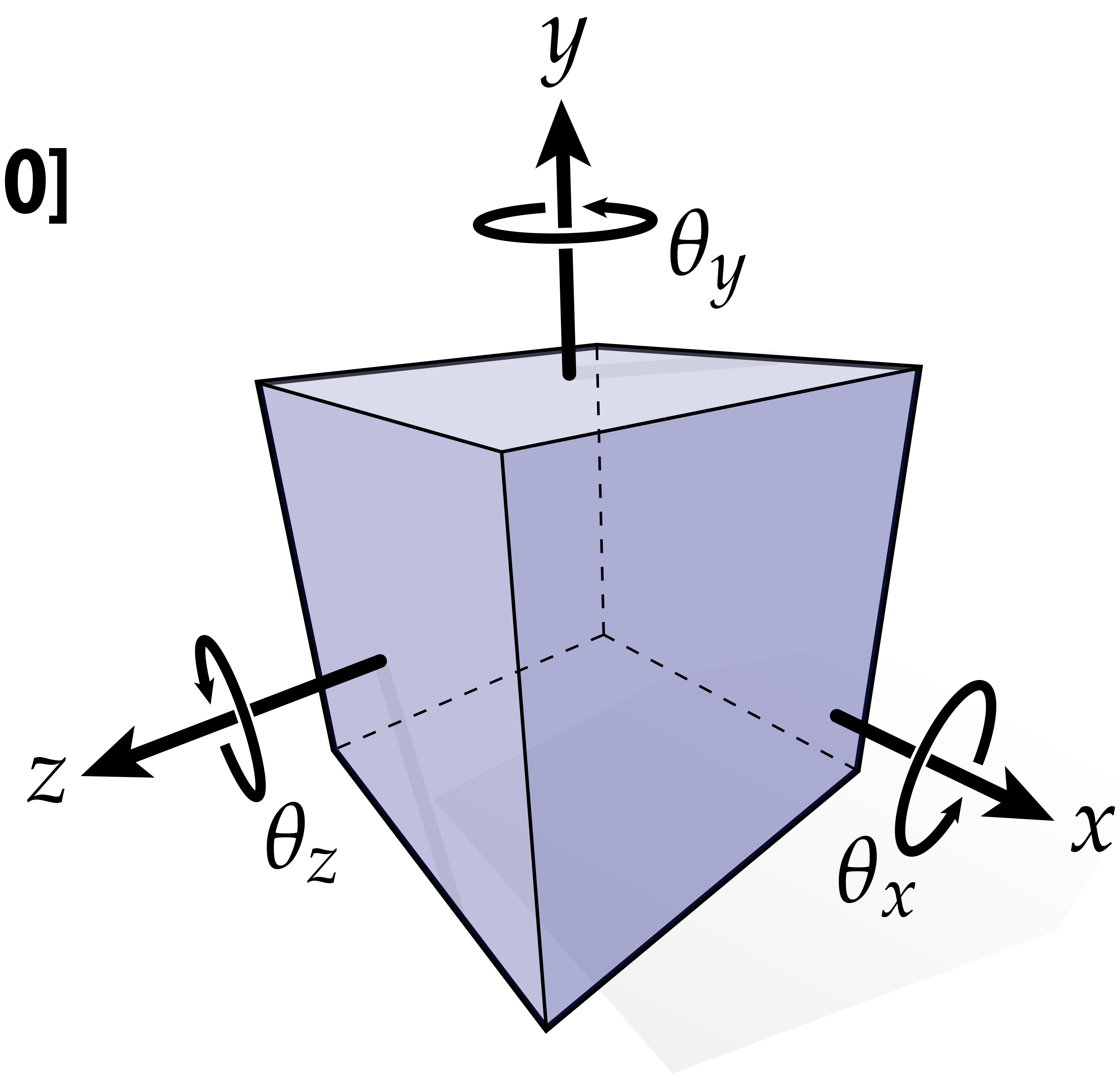
$$\left[\begin{array}{cc} S(\theta) & S(\theta+\pi/2) \end{array}\right] = \left[\begin{array}{cc} \cos(\theta) & \cos(\theta+\pi/2) \\ \sin(\theta) & \sin(\theta+\pi/2) \end{array}\right] = \left[\begin{array}{cc} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{array}\right]$$

***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
- PROBLEM: “Gimbal Lock” [DEMO]





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:**

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

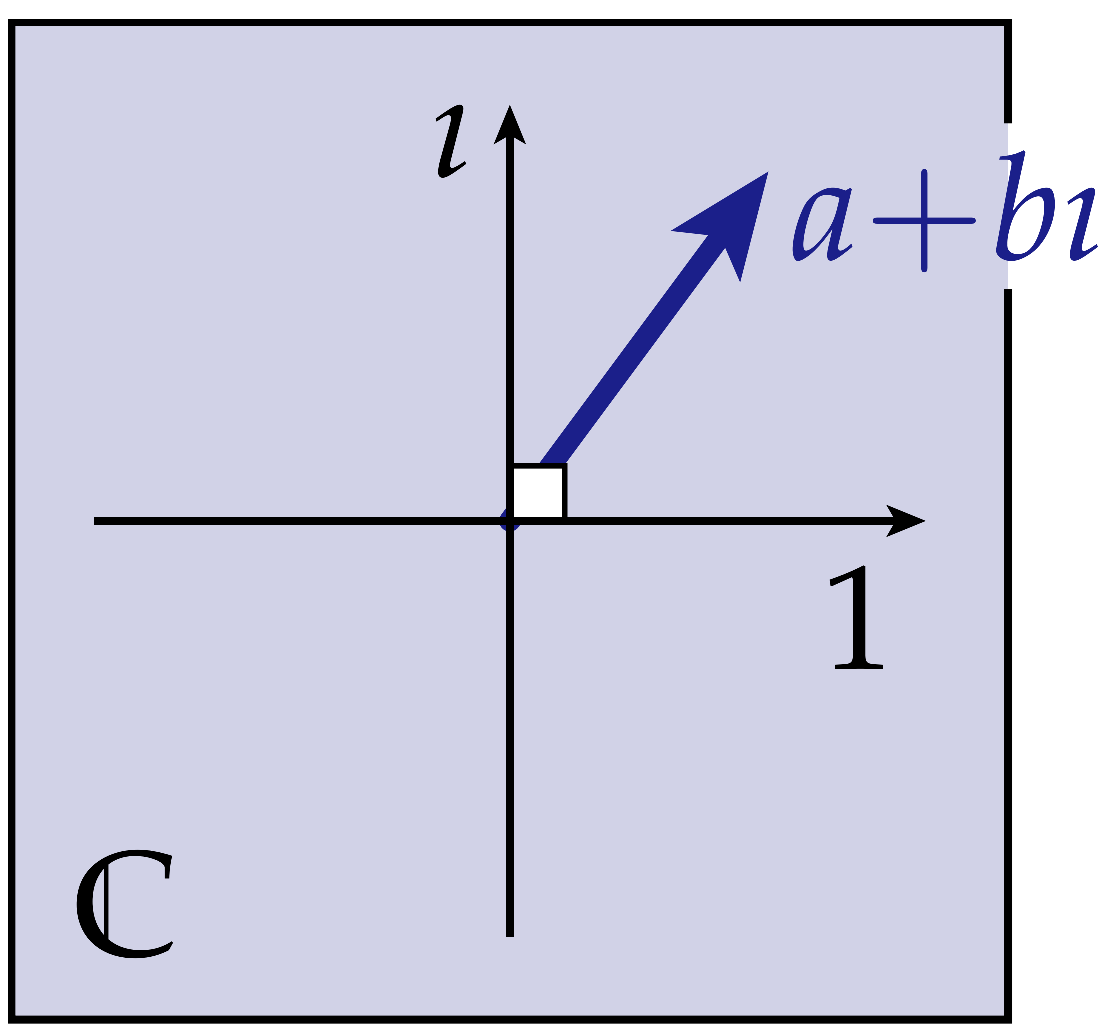
Just memorize this matrix! :-)

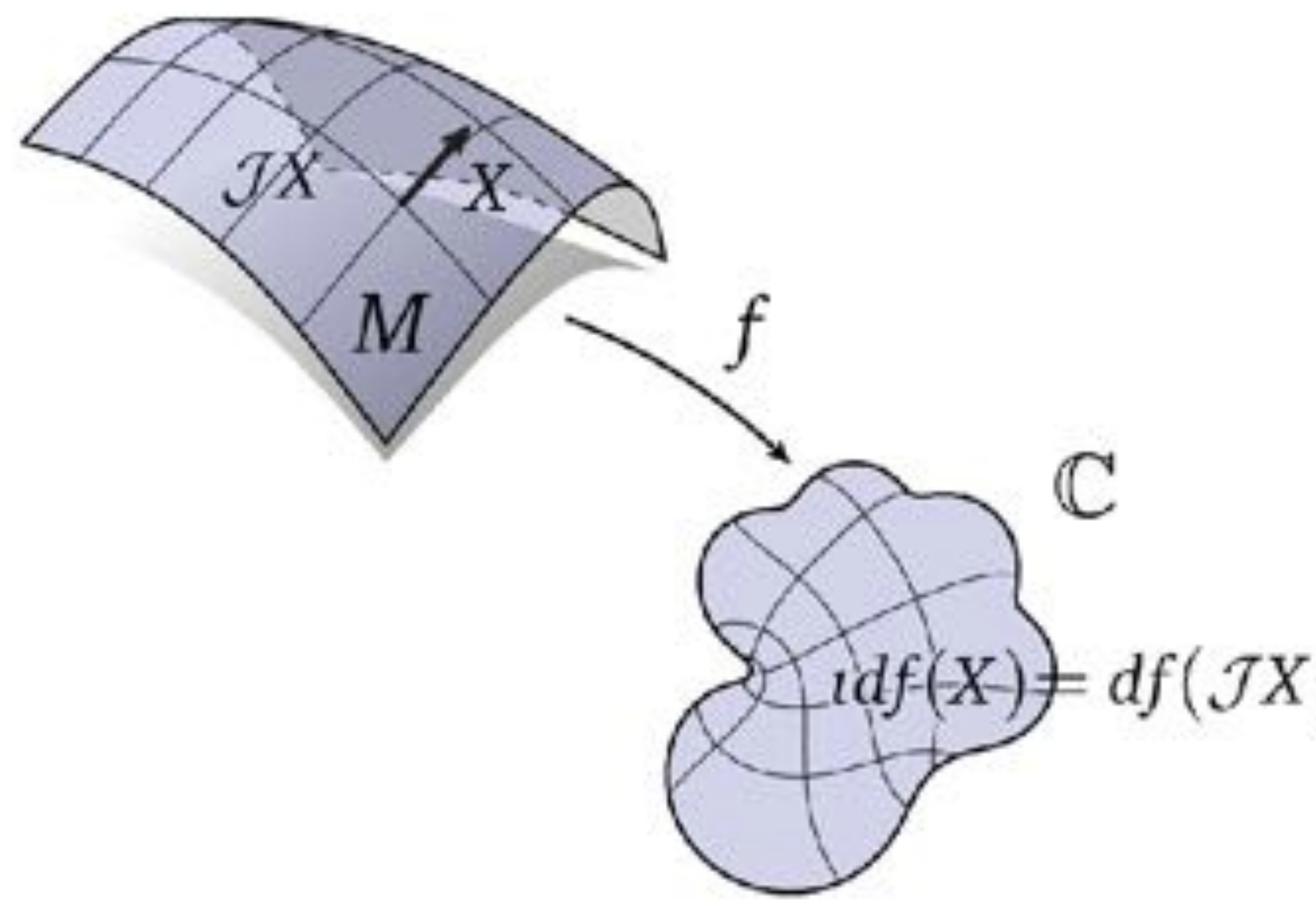
…we'll see a different way, later on.

Complex Analysis—Motivation

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

COMPLEX





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).

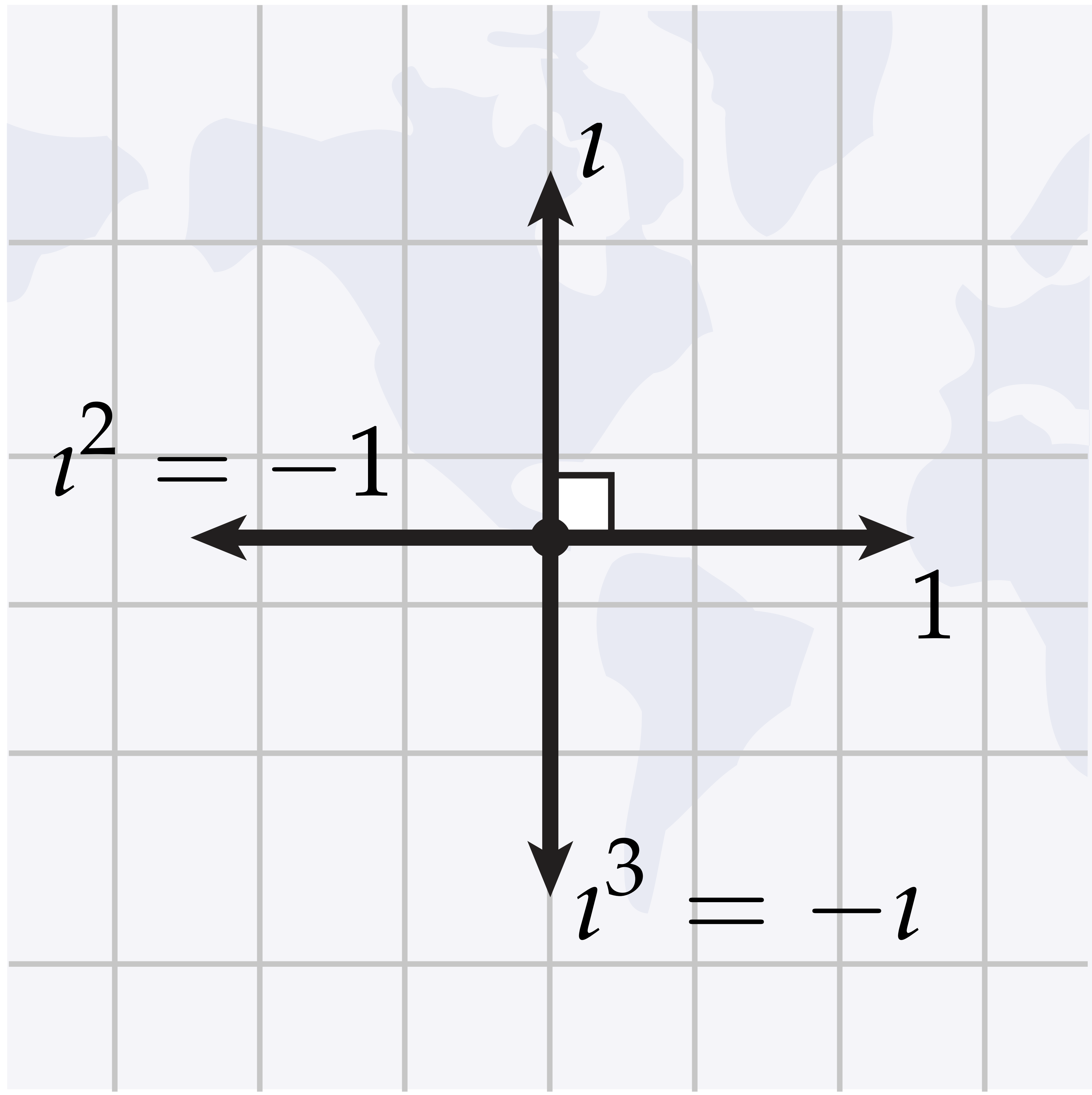
Imaginary Unit

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

nonsense!

More importantly: obscures geometric meaning.

Imaginary Unit—Geometric Description

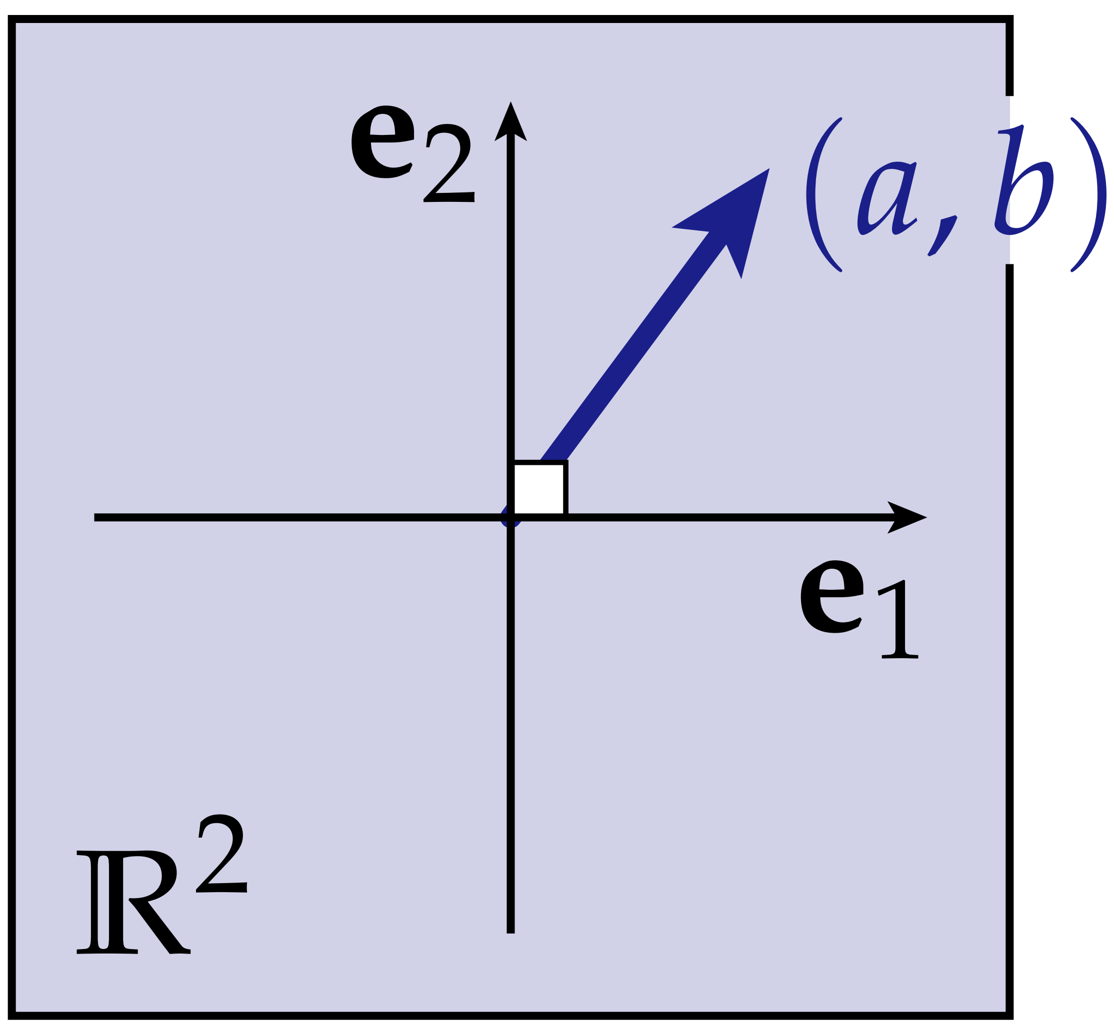


Imaginary unit is just a quarter-turn in the counter-clockwise direction.

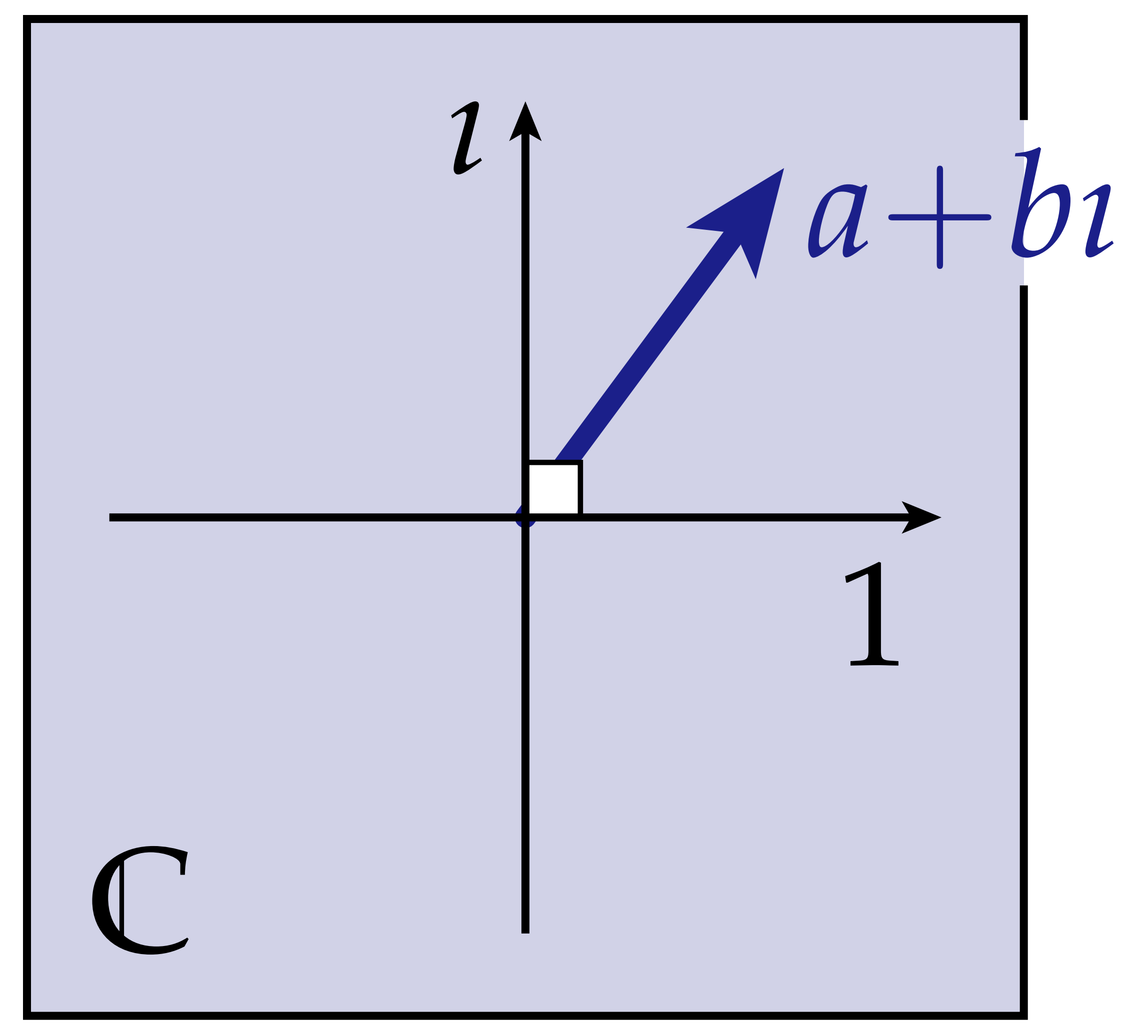
Complex Numbers

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

REAL



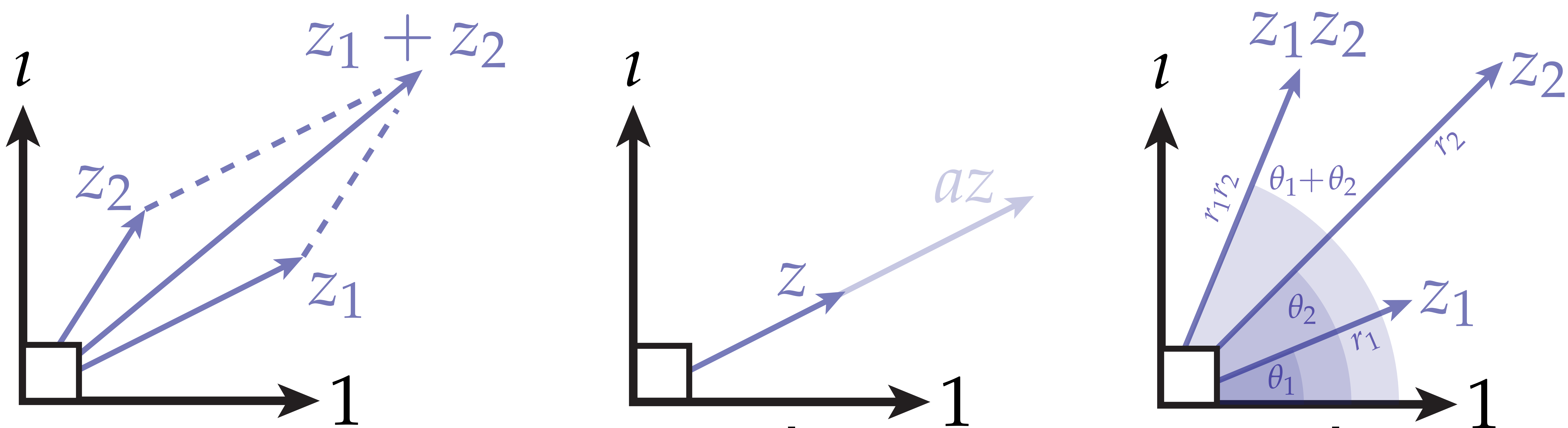
COMPLEX



- ...except that we're 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, ı):

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

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

two quarter turns—
same as -1

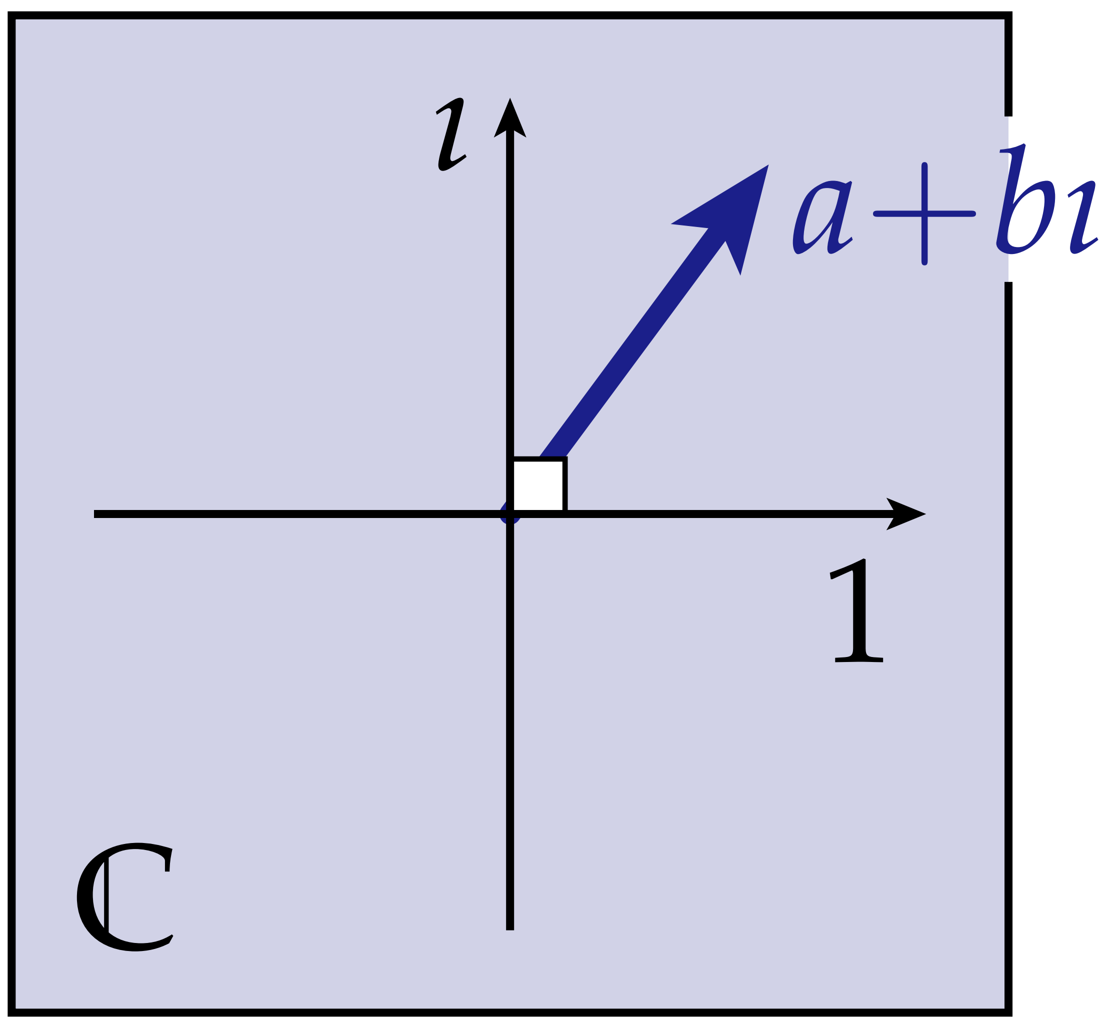
$$z_1 z_2 = ac + ad\imath + bc\imath + bd\imath^2 =$$

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

"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)

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

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

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

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

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

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

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

(…and simplification is not always this obvious.)

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

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

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

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

Pervasive theme in graphics:

Sure, there are often many "equivalent" representations.

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

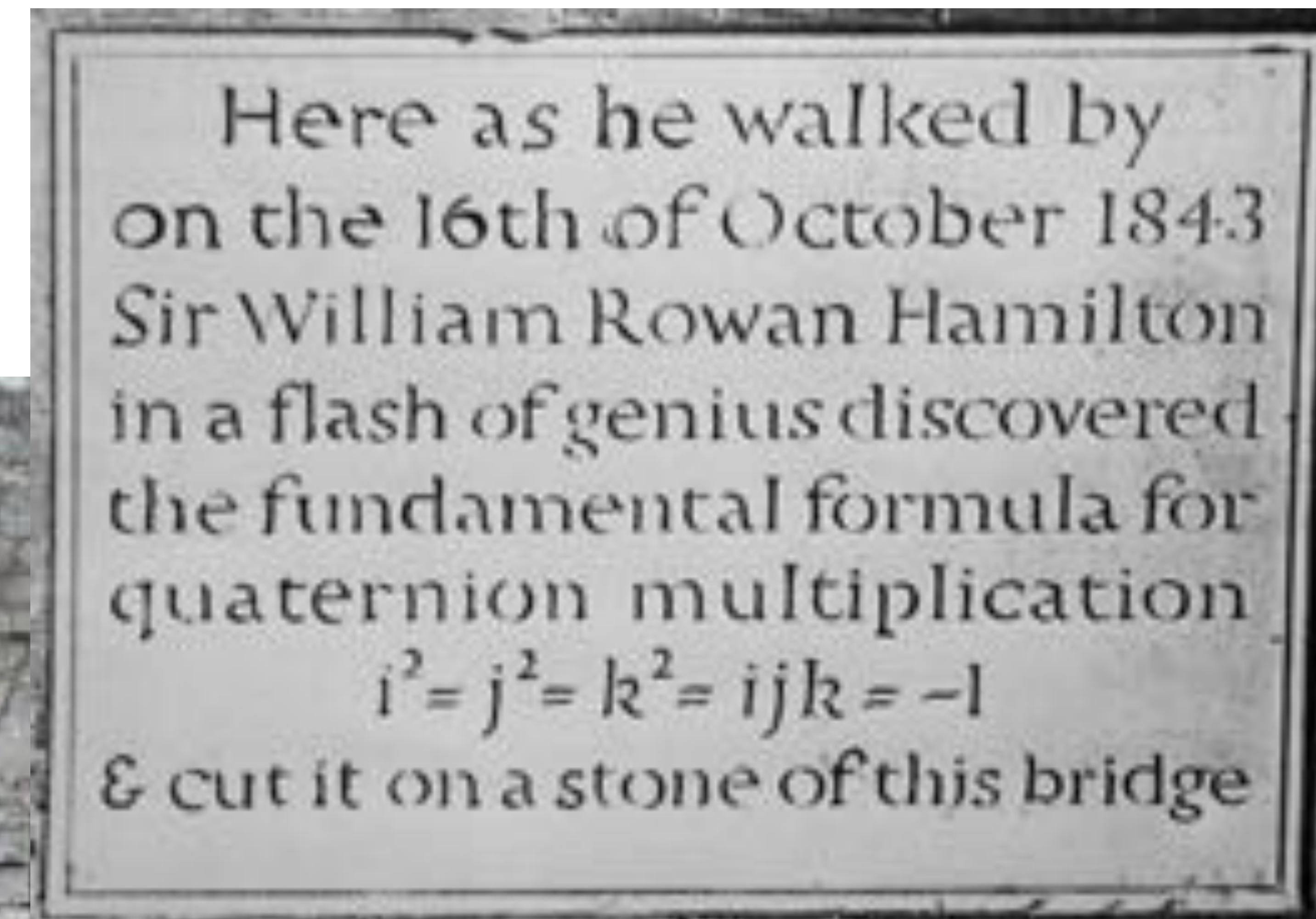
*Or most efficient, or most accurate...

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)



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:**

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

"H" is for Hamilton!

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

- **Quaternion product determined by**

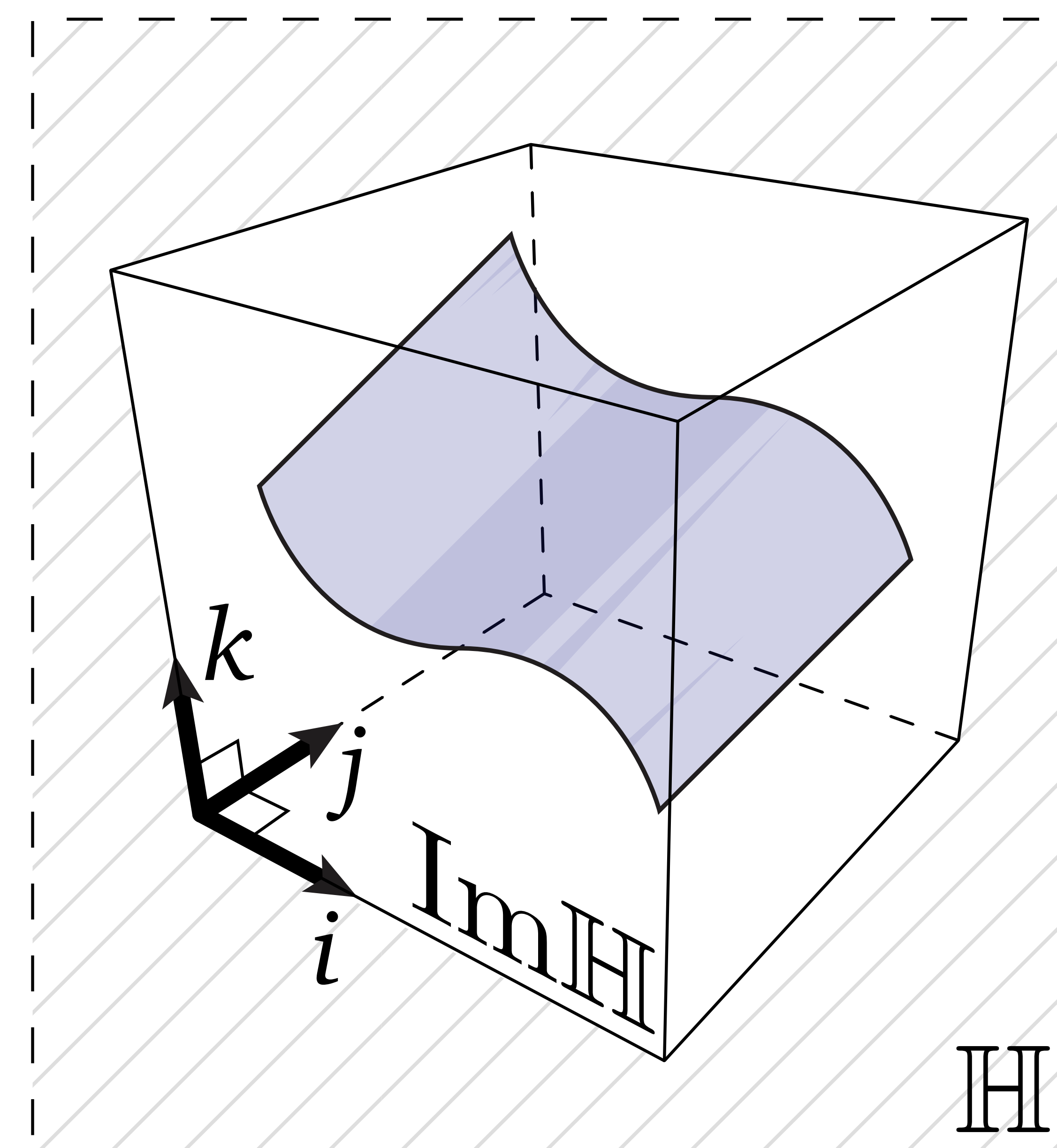
$$\imath^2 = \jmath^2 = k^2 = \imath\jmath k = -1$$

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

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

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

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



Quaternion Product in Components

- **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?**

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

- **Alternatively, can think of a quaternion as a pair**

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

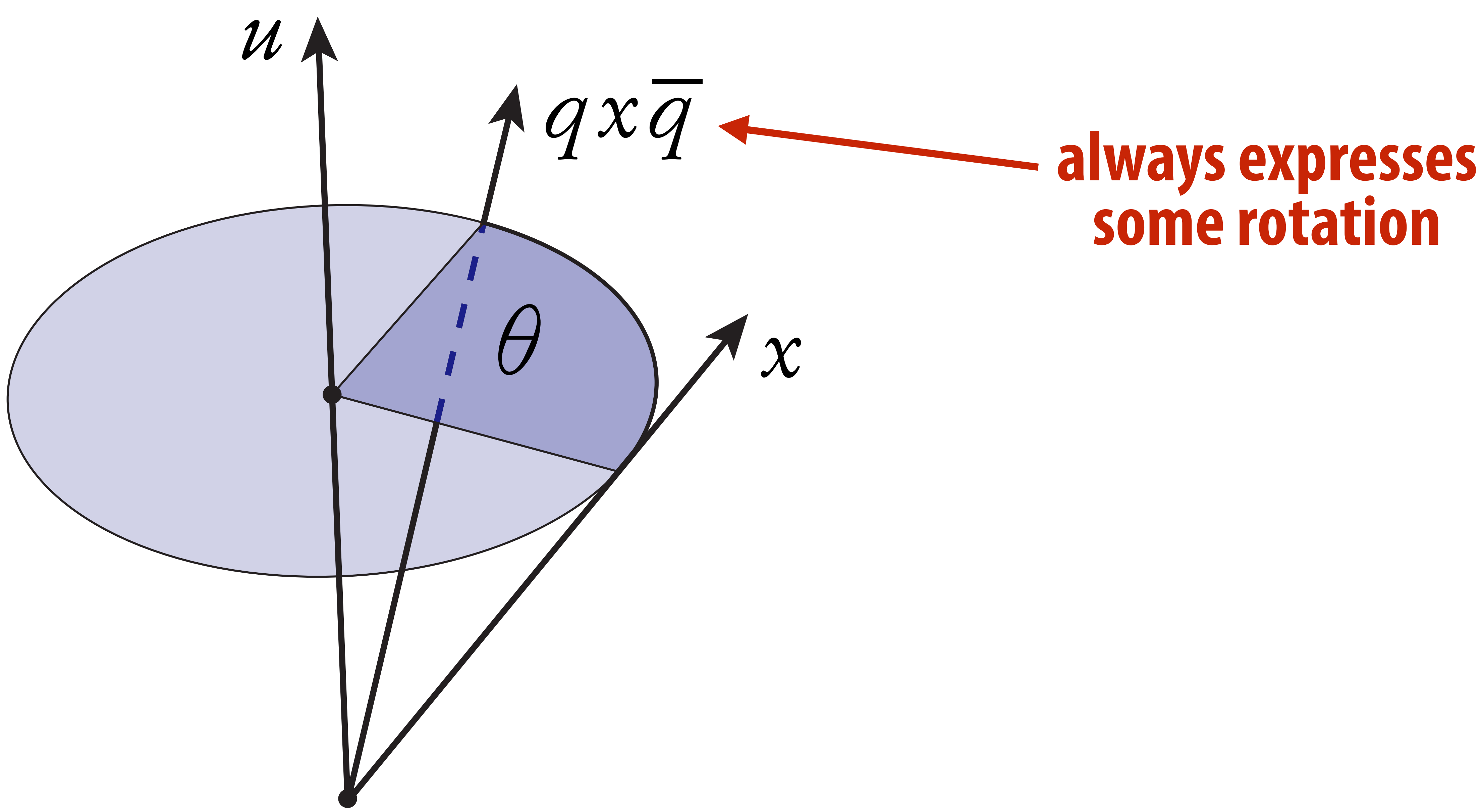
- **Quaternion product then has simple(r) form:**

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

3D Transformations via Quaternions

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

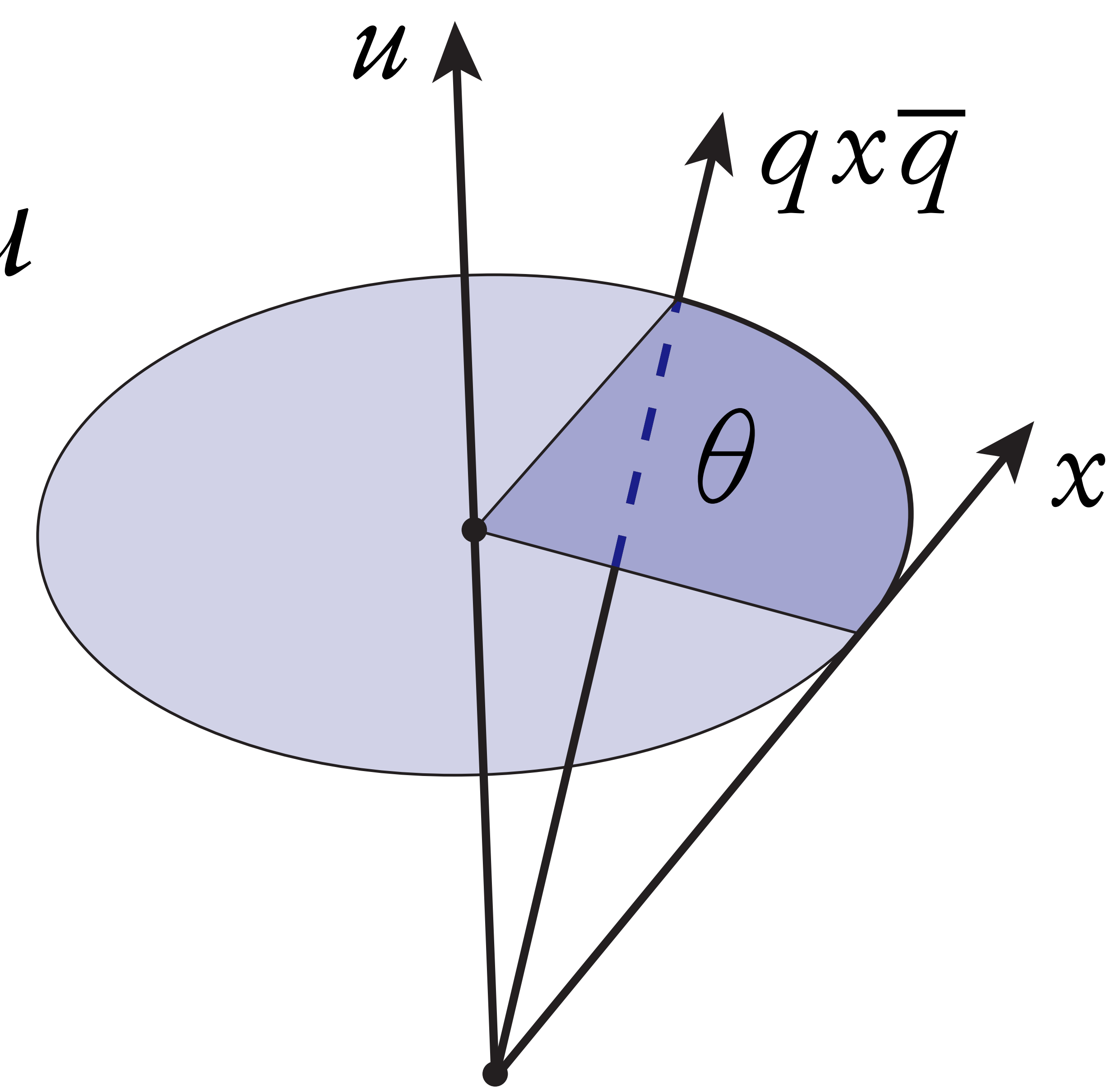
$$x \in \mathrm{Im}(\mathbb{H})$$
$$q \in \mathbb{H}, \quad |q|^2 = 1$$



Rotation from Axis/Angle, Revisited

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

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

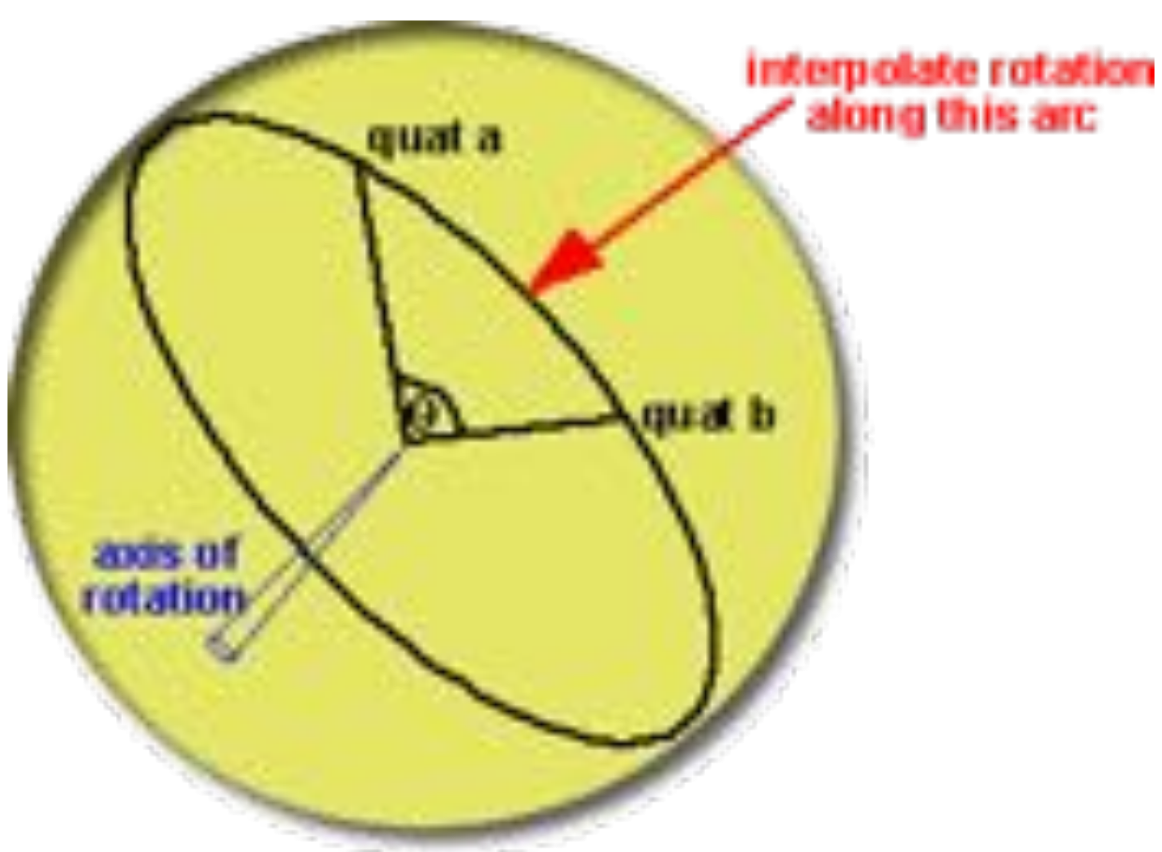


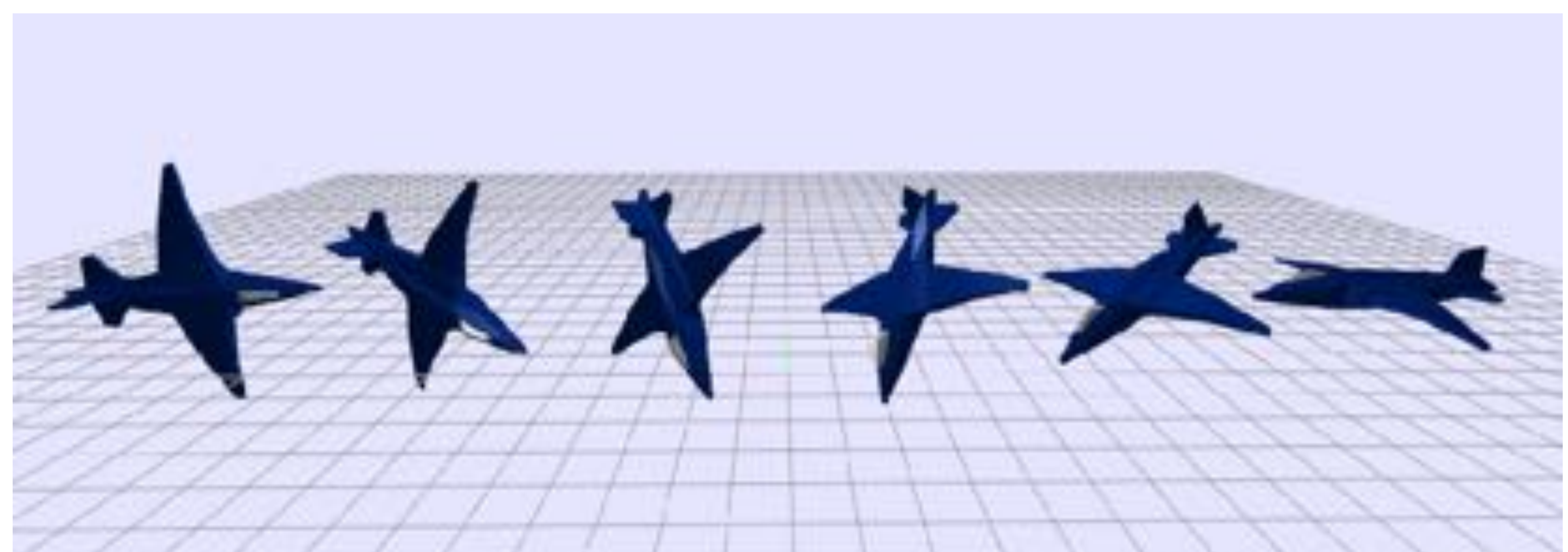
- **Slightly easier to remember (and manipulate) than matrix!**

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

More Quaternions and Rotation

- Don't have time to cover everything, but…
- Quaternions provide some very nice utility/perspective when it comes to rotations:
  - E.g., spherical linear interpolation ("slerp")
  - Easy way to smoothly transition between orientations
  - No gimbal lock!

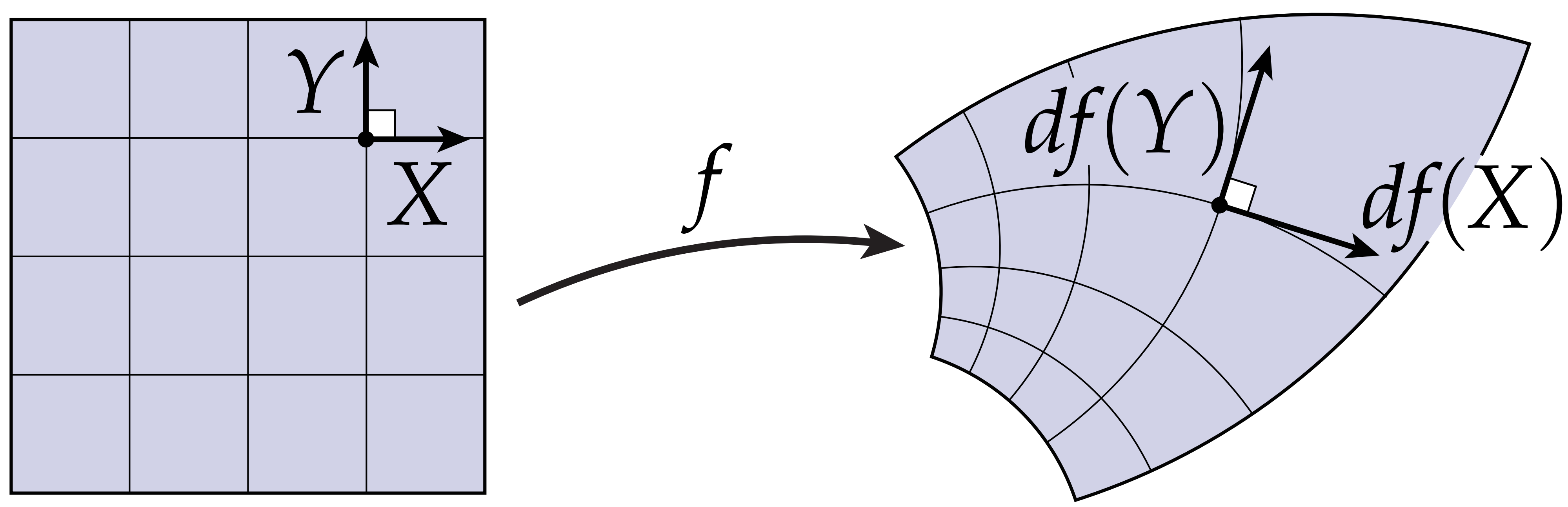


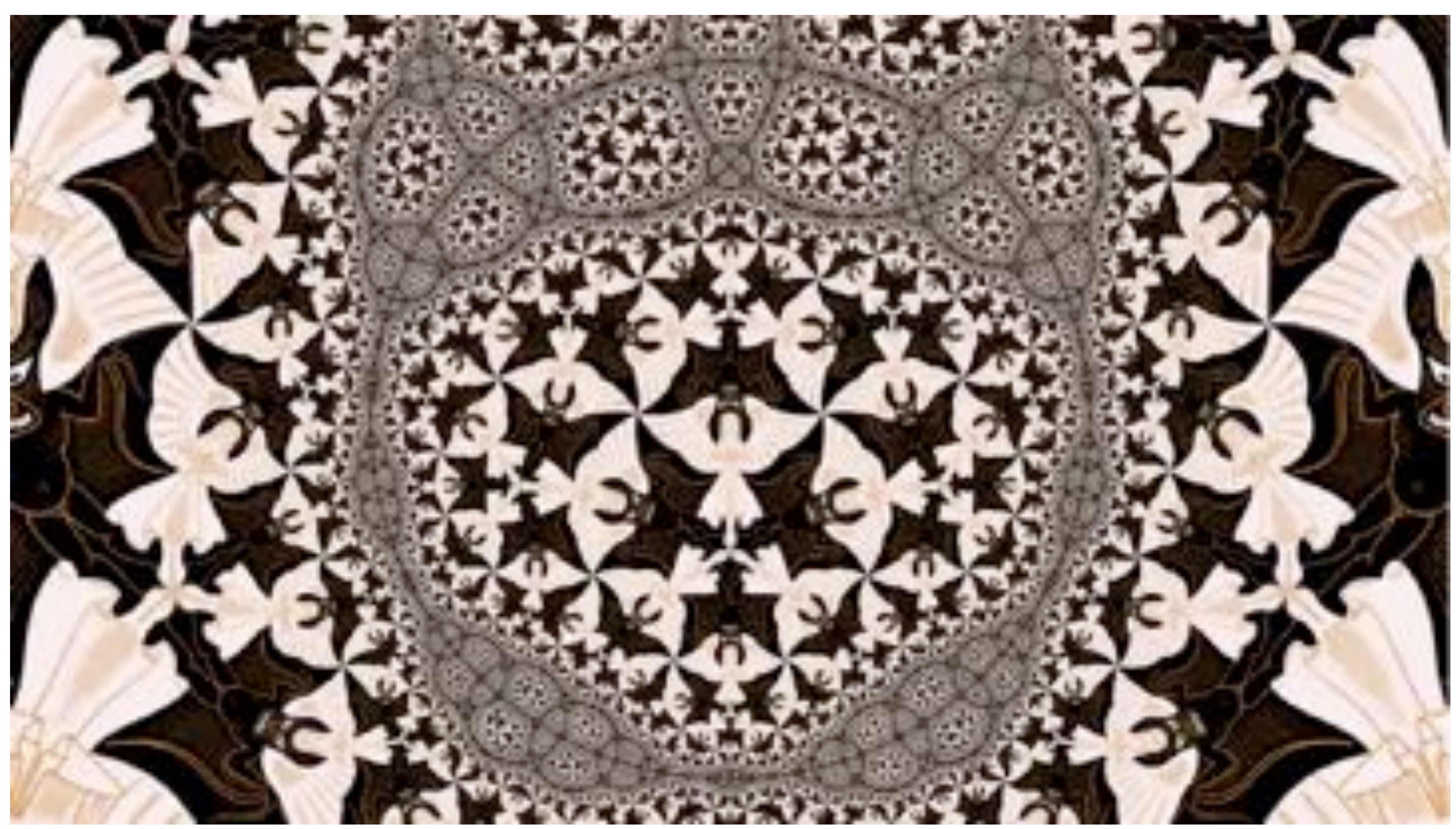


Where else are (hyper-)complex numbers useful in computer graphics?

Generating Coordinates for Texture Maps

Complex #s natural language for angle-preserving ("conformal") maps

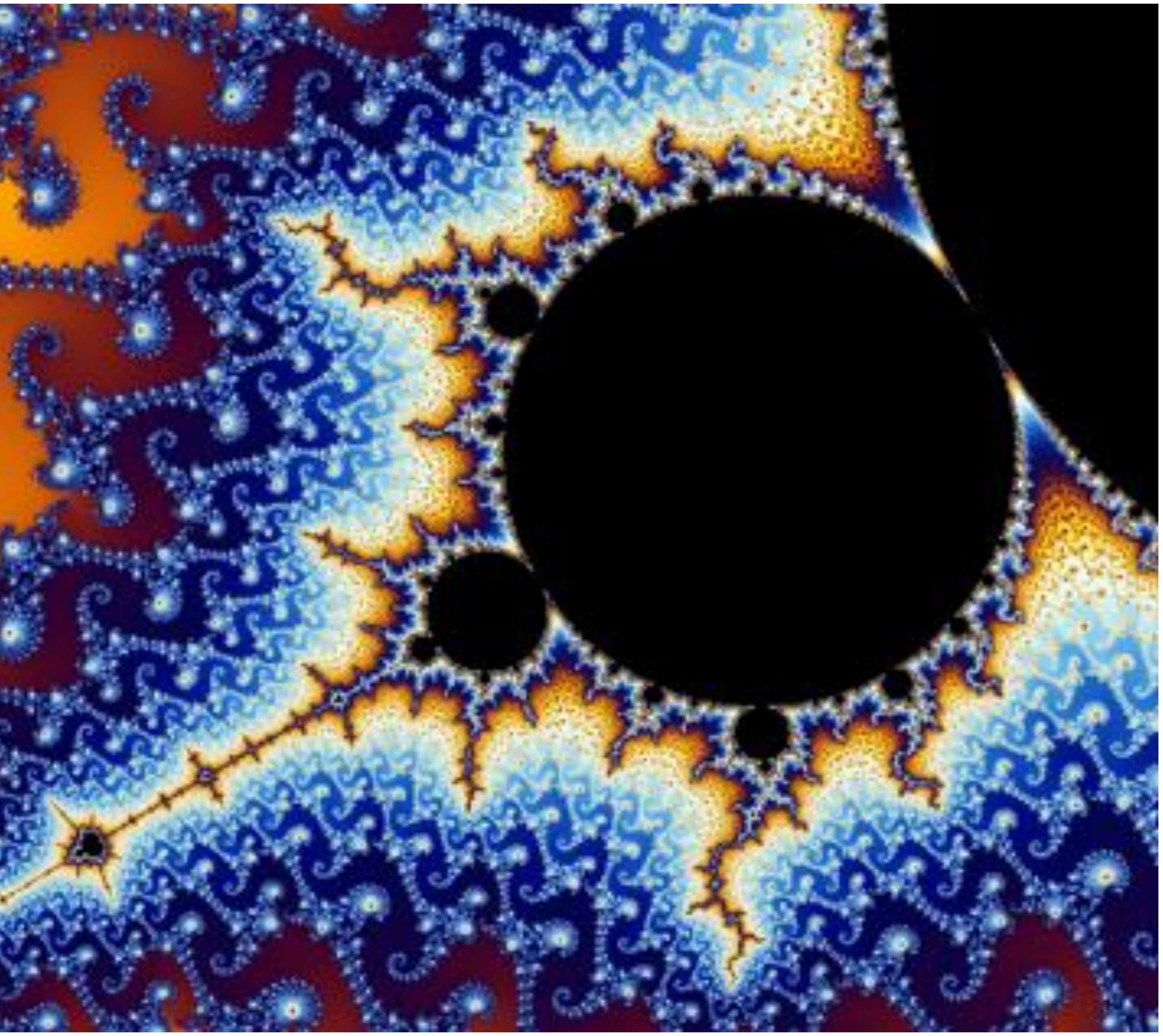


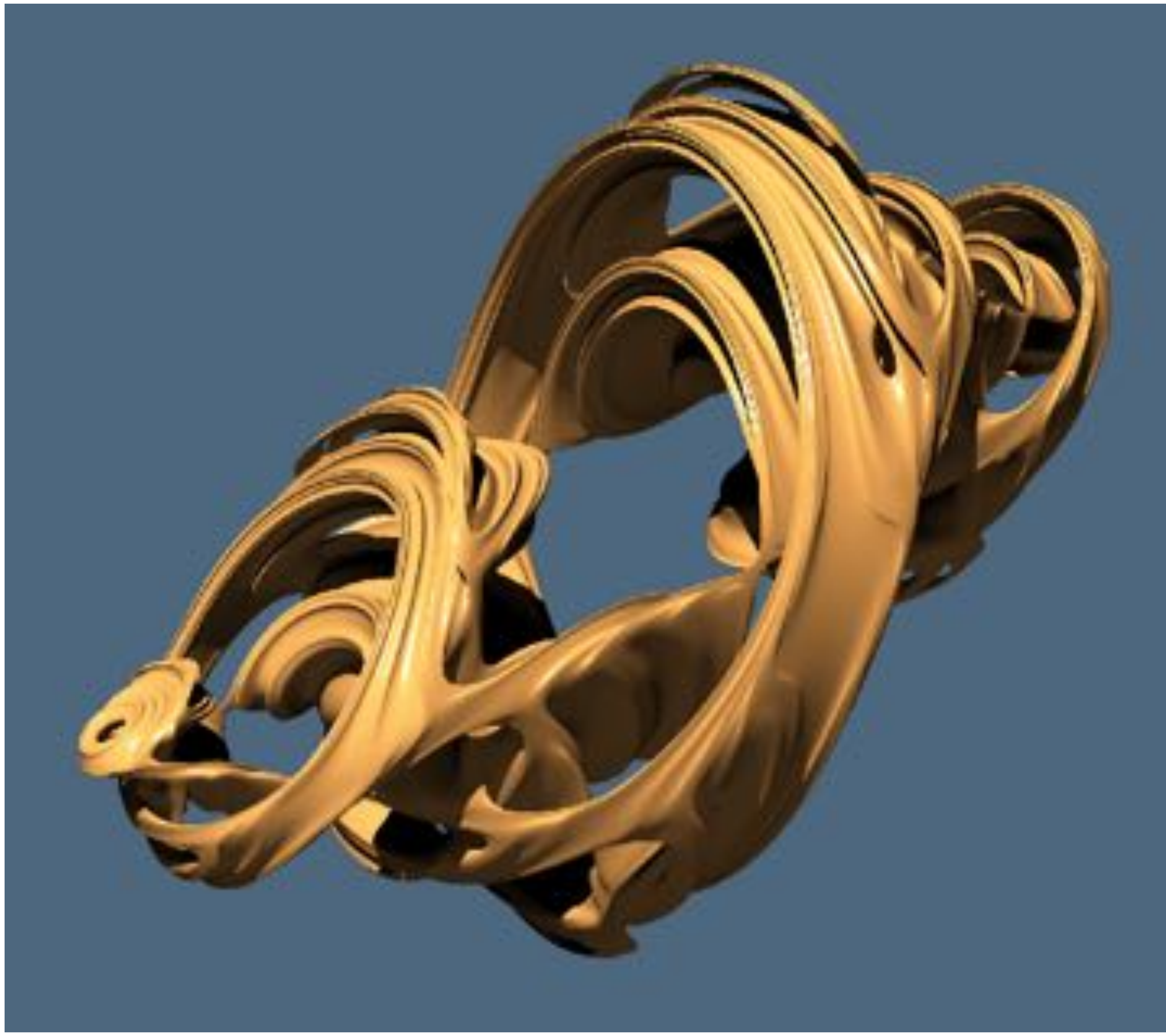


Preserving angles in texture well-tuned to human perception…

Useless-But-Beautiful Example: Fractals

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





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