



Lecture 07: 3D旋转与复数表示

SSE315: 计算机图形学
Computer Graphics

陈壮彬

软件工程学院

chenzhb36@mail.sysu.edu.cn

Today's topics

□ 3D 旋转介绍

□ 复数表示 (complex representation)

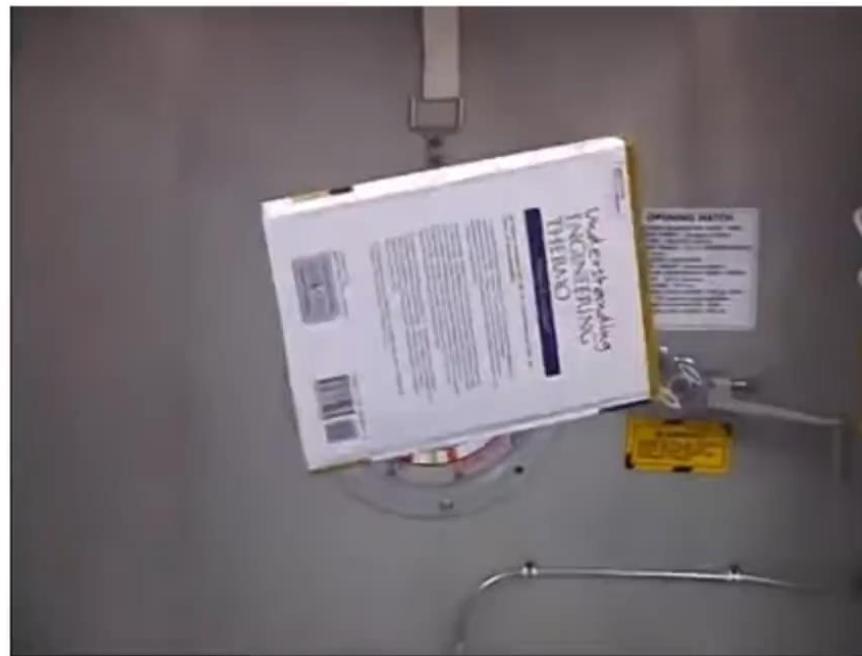
□ 四元数 Quaternions

3D 中的旋转

□直观上，什么是旋转？

□当你看到旋转时，你怎么知道它就是旋转？

- 保持长度 length/距离 distance (无拉伸/切变)
- 保持方向 orientation (例如，文本保持可读)
- 保持原点 origin (否则为旋转 + 平移)



3D 旋转 – 自由度 Degrees of freedom

- 我们需要多少个维度来指定一个 3D 旋转？
- 直觉上是 3 个， x, y, z ，但我们需要所有 3 个维度吗？
- 将匹兹堡旋转到另一个城市（比如圣保罗），我们必须指定两个数字：纬度和经度
- 这是从匹兹堡到圣保罗的唯一旋转吗？是否需要其他数字？

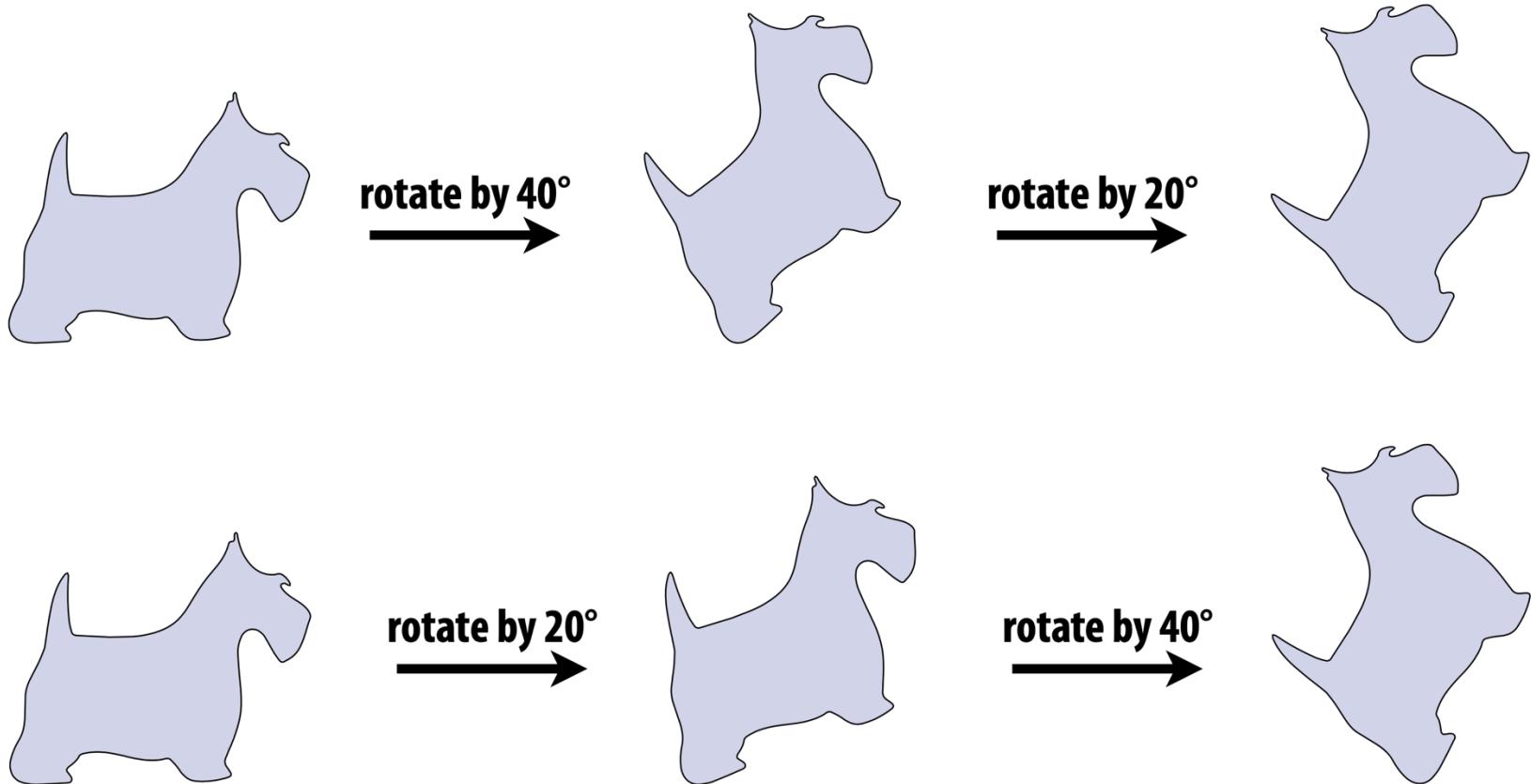
No, 到达圣保罗后，仍可以绕着圣保罗继续旋转地球

因此，我们必须拥有三个自由度
three degrees of freedom



旋转交换律 Commutativity – 2D

□ In 2D, order of rotations does not matter



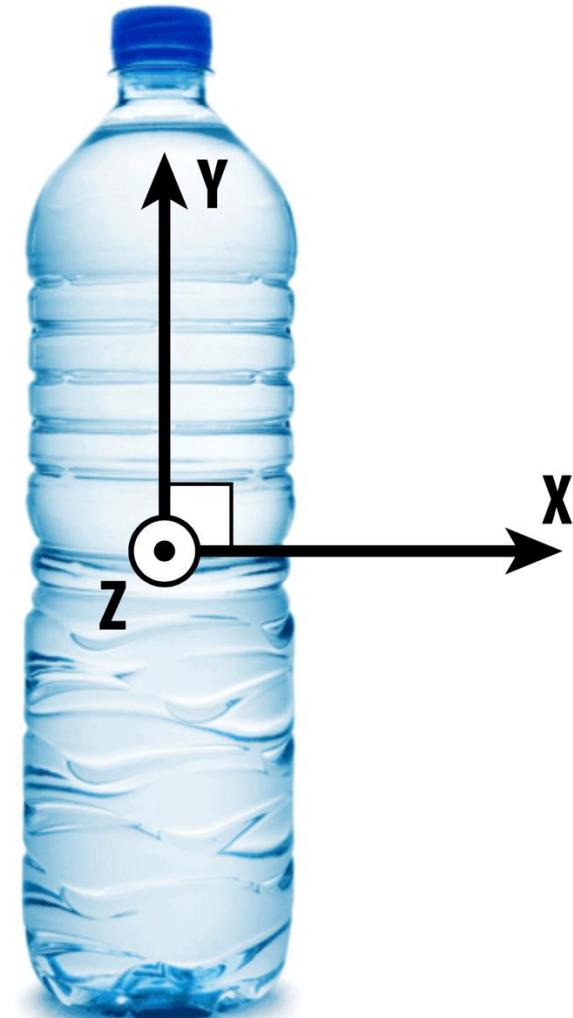
Same result! (“2D rotations commute”)

旋转交换律 Commutativity – 3D

□ What about in 3D?

□ 我们用一瓶水做实验

- Rotate 90° around Y, then 90° around Z, then 90° around X
- Rotate 90° around Z, then 90° around Y, then 90° around X
- 有什么不一样吗？



**如果我们不小心处理旋转的顺序，
可能会发生意料之外的事情！**

旋转的表示 - 2D

□ 我们如何得到 2D 中的旋转矩阵？

□ 假设我有一个函数 $S(\theta)$ ，对于给定的角度 θ ，它给出逆时针绕圆的点 (x, y)

- 暂且不关心 $S(\theta)$ 的表达式

□ What is \mathbf{e}_1 rotated by θ ? $\tilde{\mathbf{e}}_1 = S(\theta)$

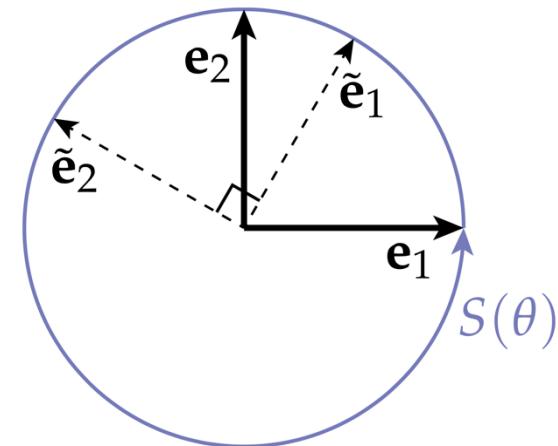
□ What is \mathbf{e}_2 rotated by θ ? $\tilde{\mathbf{e}}_2 = S(\theta + \pi/2)$

□ 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?

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



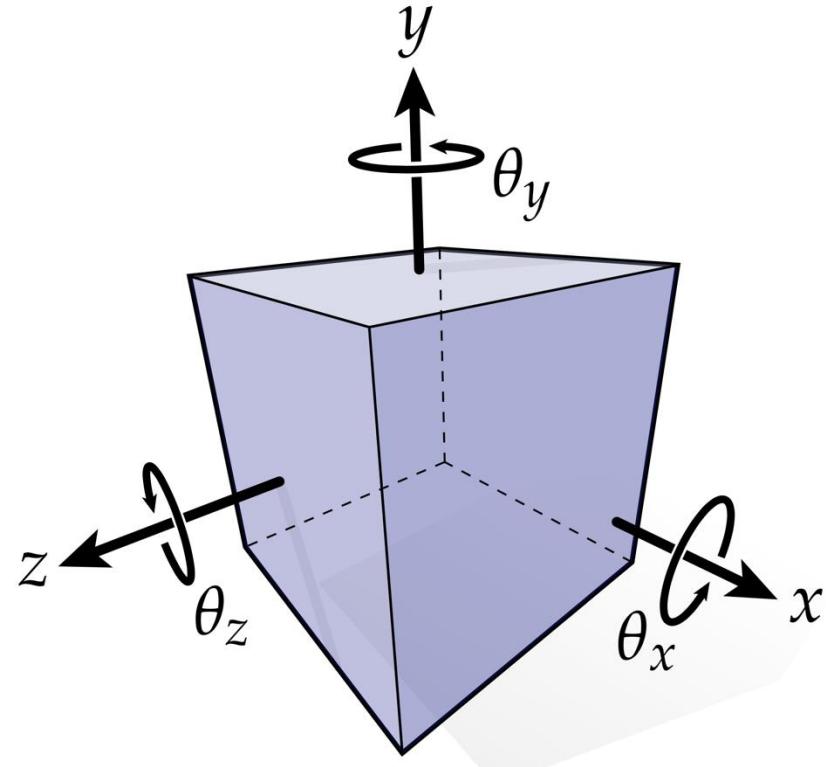
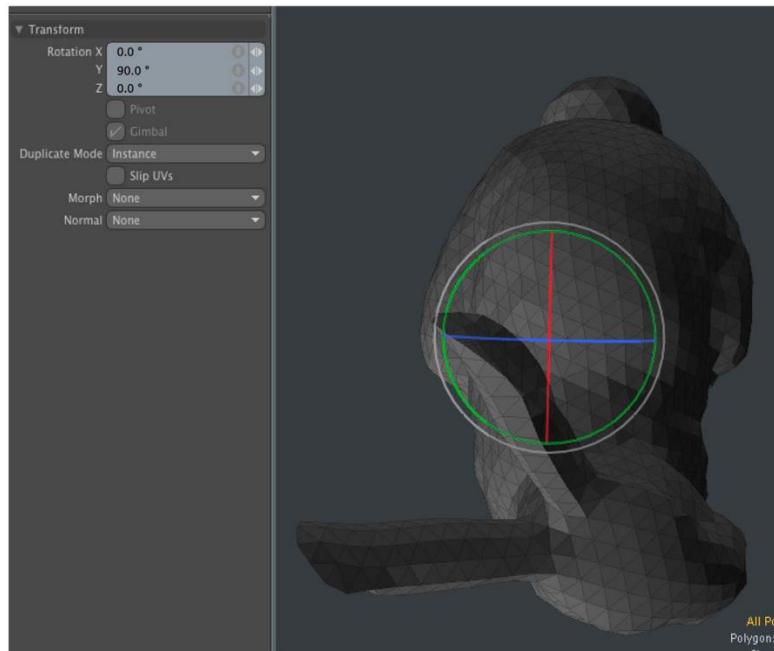
3D 中的旋转表示 – 欧拉角 Euler Angles

□ 如何在 3D 中表示旋转？

□ 一个简单的想法，能否直接将 2D 中的方法用到 3D 中的 x, y, z 轴？

□ 我们称这种方案 (scheme) 为欧拉角，有什么优点？

□ “Gimbal lock” 万向节锁



Gimbal Lock

□ 当使用欧拉角 $\theta_x, \theta_y, \theta_z$ 时，可能会出现无法绕其中一个轴旋转的情况

□ 回顾一下绕三个轴的旋转矩阵

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

□ 这些矩阵的乘积表示基于欧拉角的 3D 旋转

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

□ 考虑一个特殊例子， $\theta_y = \pi/2$ (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 万向节锁

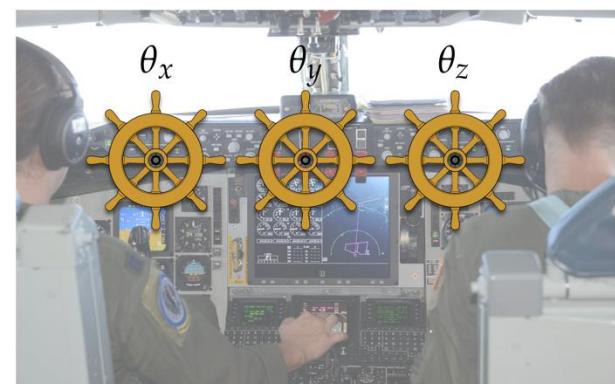
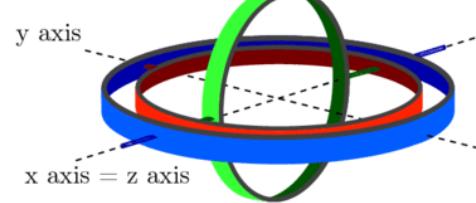
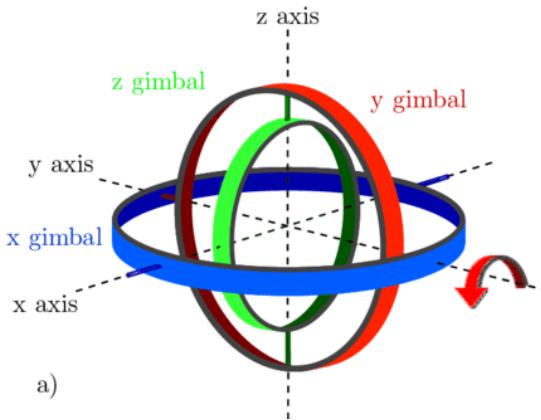
口简化上一张幻灯片中的矩阵，我们得到

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

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

Q: What does this matrix do?

口我们被“锁定”在一个单一的旋转轴上



口对飞机控制是个很糟糕的设计！

从轴/角度旋转

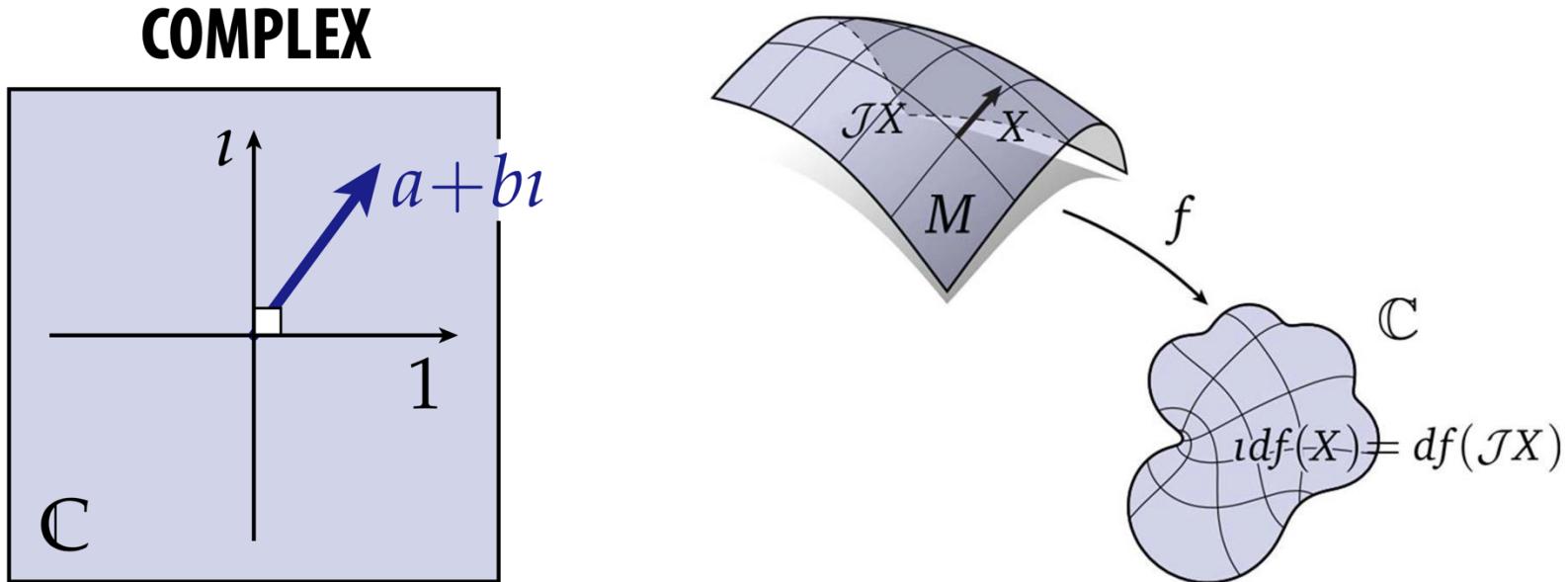
□ 绕开 Gimbal lock, 对于绕给定轴 \mathbf{u} 旋转给定角度 θ 的矩阵, 有一个通用表达式

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

□ 之后我们会学习一个更简单的方式...

复数分析 Complex analysis

- 编码二维几何变换的一种自然方式，比 2D 向量更好
- 简化代码 code/符号 notation/调试 debugging/思考 thinking
- 降低计算成本/带宽/存储等
- 熟练地使用复数分析可以为问题找到更深入/新颖的解决方案，比如曲面参数化、纹理映射等



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

DO: Imagine we're simply defining
additional operations (like dot and
cross) on these “vectors.”

虚数单位 Imaginary Unit

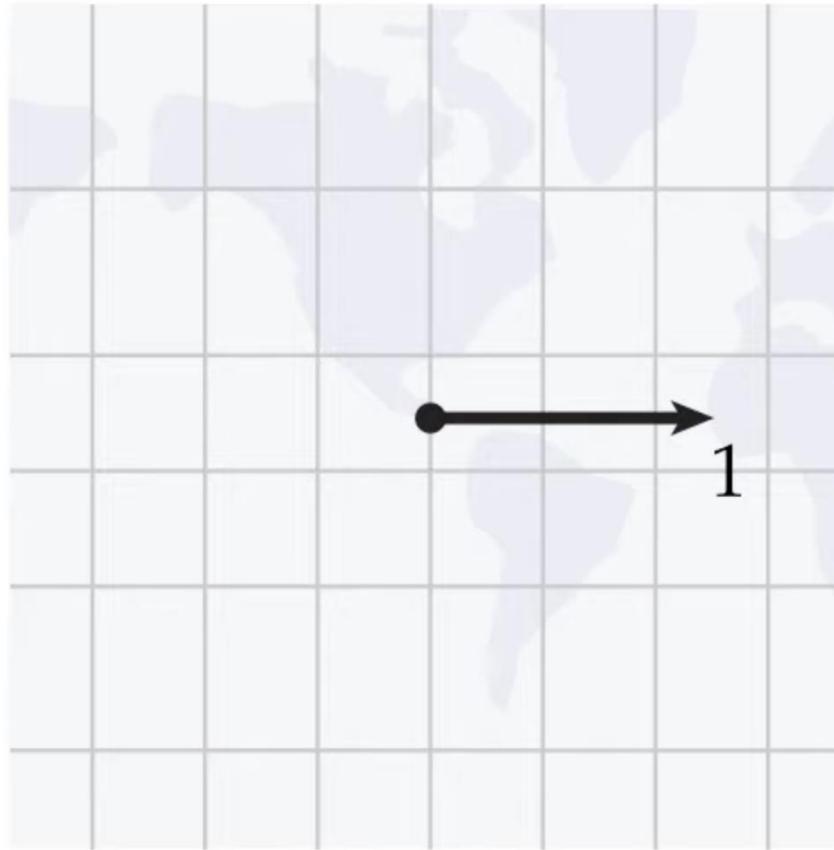
$$i = \sqrt{-1}$$
A large red 'X' is drawn over the equation $i = \sqrt{-1}$, indicating that it is incorrect or misleading.

□更重要的是：模糊了几何意义

虚数单位 – 几何描述

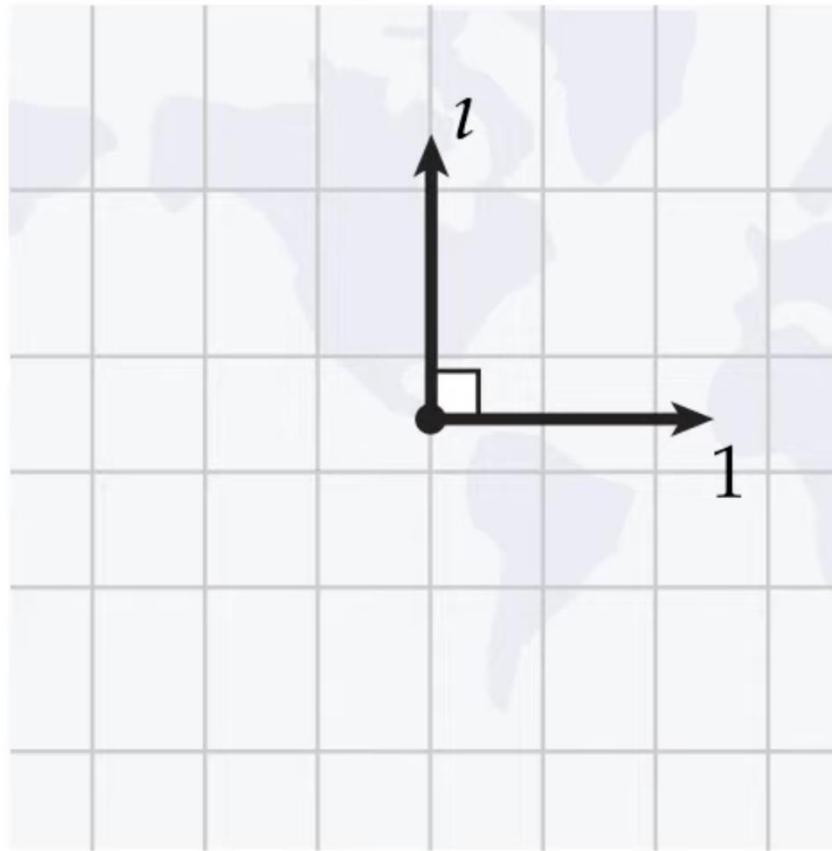
虚数单位只是沿逆时针方向旋转四分之一圈

虚数单位 – 几何描述



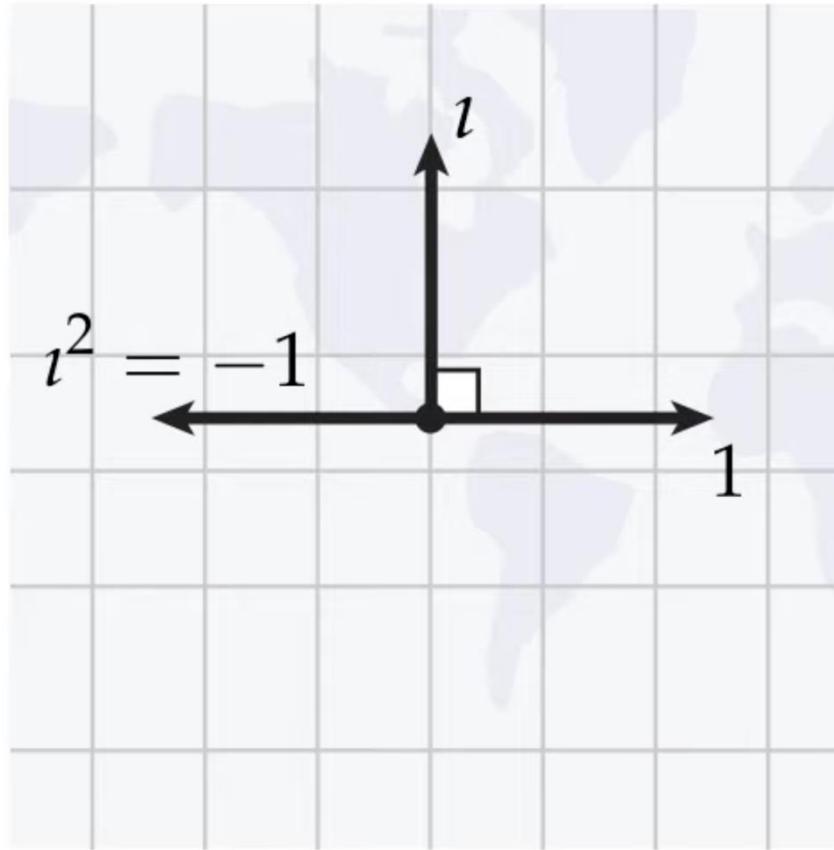
虚数单位只是沿逆时针方向旋转四分之一圈

虚数单位 – 几何描述



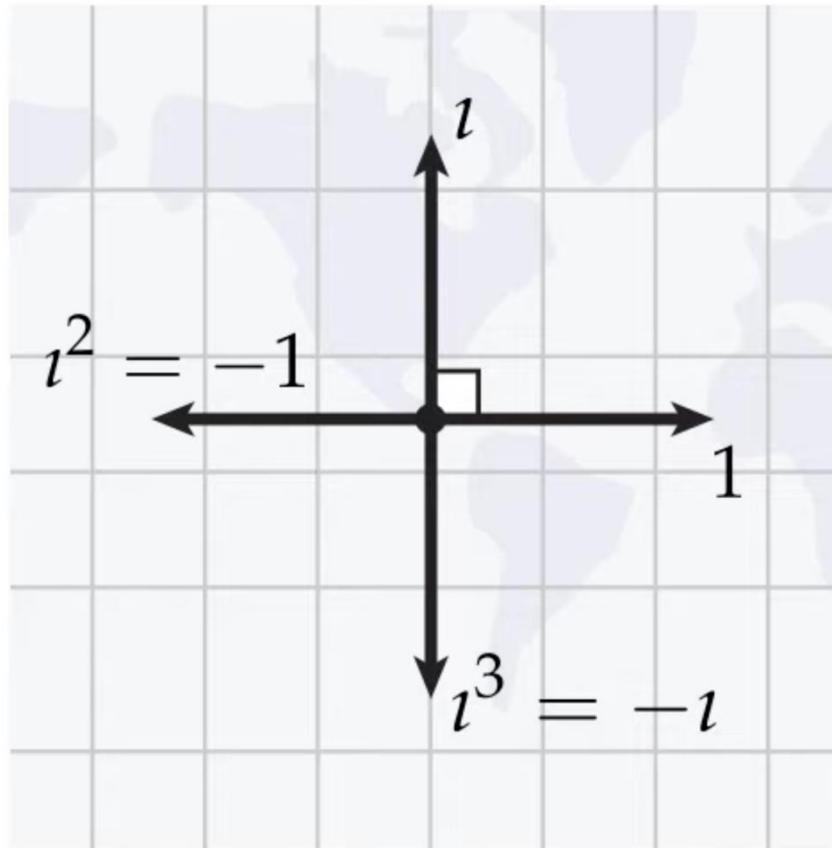
虚数单位只是沿逆时针方向旋转四分之一圈

虚数单位 – 几何描述



虚数单位只是沿逆时针方向旋转四分之一圈

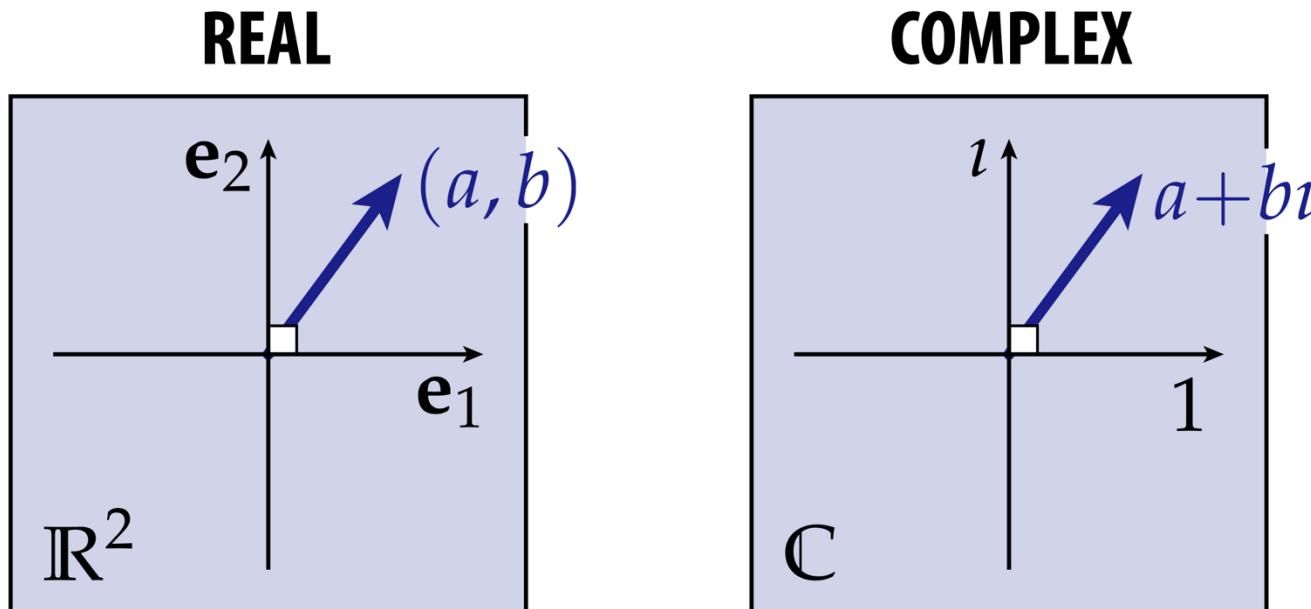
虚数单位 – 几何描述



虚数单位只是沿逆时针方向旋转四分之一圈

复数 Complex numbers

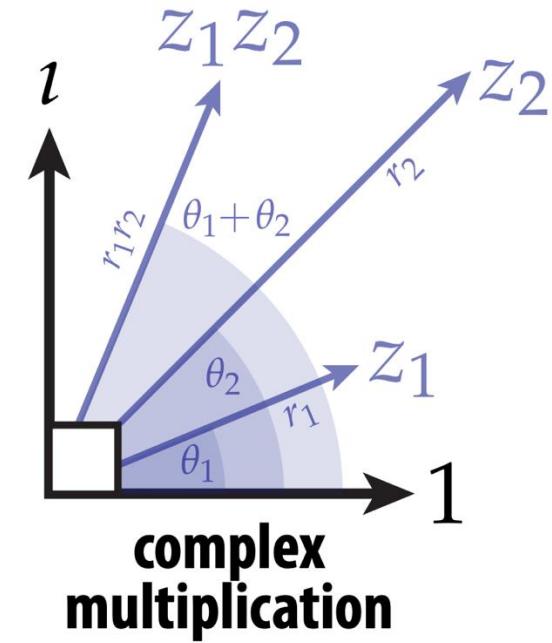
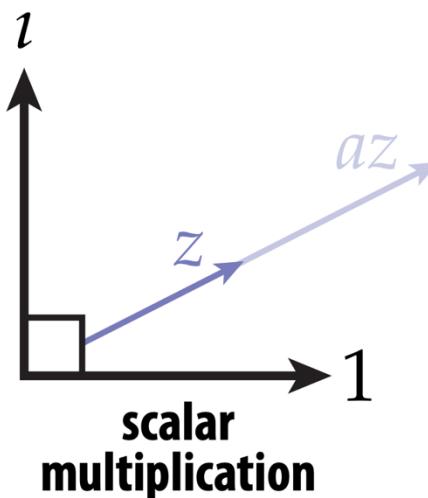
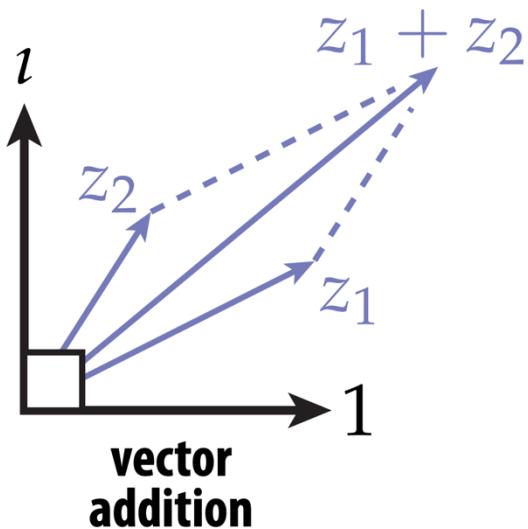
- 复数可以理解成一个 2 维向量
- 使用 “1” 和 “ i ” 而不是 e_1, e_2 来表示两个坐标基
- 除此之外，其他表现与 2 维空间无异



- 除了一个非常有用的关注两个向量乘积的新操作

复数算术 Complex arithmetic

□ 与以前相同的操作，再加上一个



□ 复数乘法

- 角度 (angles) 相加
- 长度 (magnitudes) 相乘

"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.

复数乘积 – 矩形形式 Rectangular Form

口在“矩形”坐标 $(1, i)$ 中的复数乘积

$$z_1 = (a + bi)$$

$$z_2 = (c + di)$$

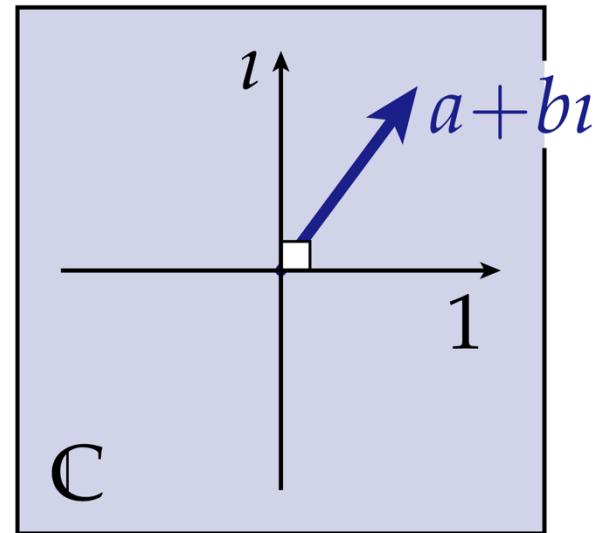
$$z_1 z_2 = ac + adi + bci + \textcolor{red}{bdi^2} =$$

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

↑
“real part”
 $\text{Re}(z_1 z_2)$

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

两个逆时针 $1/4$
旋转，等于 -1



口我们在这里使用了很多“规则，”你能用几何方法证明它们的合理性吗？

口这个乘积与我们(上一张幻灯片)中的几何描述一致吗？

复数乘积 – 极坐标形式 Polar Form

□可能是数学里最美的等式：

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

□是 Euler's formula 的一种特殊情况：

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

□可用于“实现”复数乘积

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

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



与实数的指数运算一样，指数相加



Leonhard Euler
(1707–1783)

2D 旋转: Matrices vs. Complex

假设我们想将向量 \mathbf{u} 旋转角度 θ , 然后再旋转角度 φ

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

图形中的普遍主题：

某个操作通常有许多“等价”
表示 representations

要选择让问题更简单（或更
高效、准确）的那一个

四元数 Quaternions

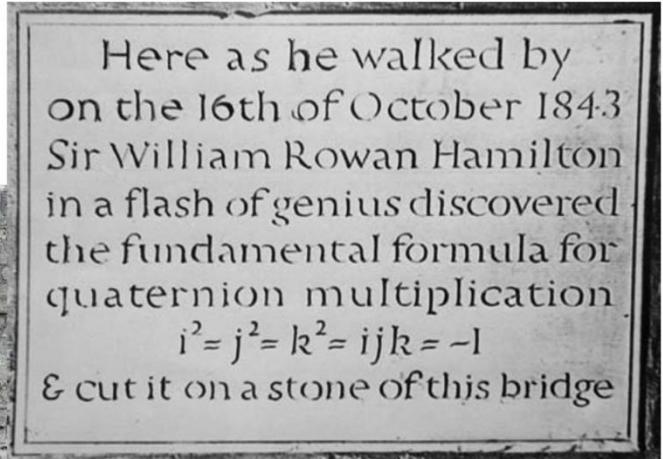
- 口有点类似复数，不过是针对 3D 旋转的
- 口有点奇怪：不同于 2D 旋转仅需 2 个部分（长度+角度），只有 3 个部分无法进行 3D 旋转



William Rowan Hamilton
(1805-1865)



(Not Hamilton)



坐标系中的四元数

□ 汉密尔顿：为了模仿 2D 复数的方式进行 3D 旋转，实际上需要 **4 个坐标**

□ 一个实数，三个虚数

"H" is for Hamilton!

$$\mathbb{H} := \text{span}(\{1, i, j, k\})$$

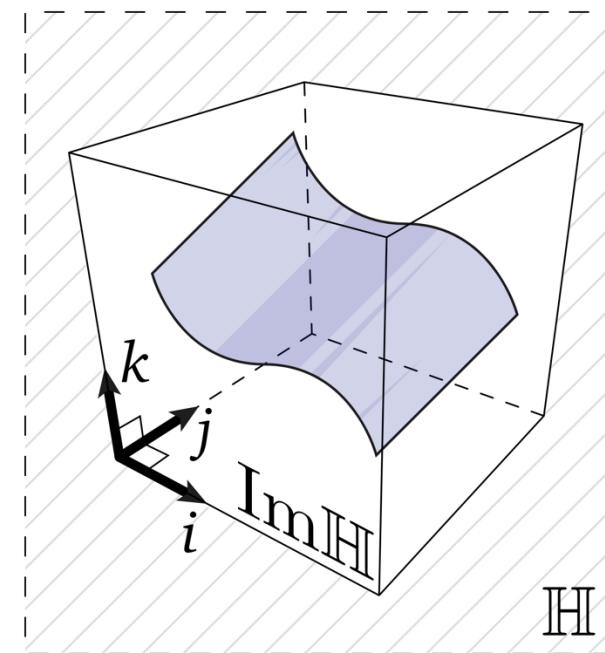
□ 四元数乘积由以下计算决定

$$\left\{ \begin{array}{l} i^2 = j^2 = k^2 = ijk = -1 \\ ij = k, ji = -k \\ jk = i, kj = -i \\ ki = j, ik = -j \end{array} \right.$$

以及一些规则 (分配律 *distributivity*、结合律 *associativity* 等)

□ Warning：乘积不满足交换律 (为什么？)

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



四元数乘积

口给定两个四元数

$$q = a_1 + b_1\imath + c_1J + d_1k$$
$$p = a_2 + b_2\imath + c_2J + d_2k$$

口它们的乘积为

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

口幸运的是，我们有一个更好的表达式...

四元数 – 标量+向量形式

□ 基于四个成分，我们要如何描述 3D 中的点？

□ 一个自然的想法，既然我们有 3 个虚数，为和不把它们编码成 3D 向量？

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

□ 可以把一个四元数想成一个对 (pair)

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

□ 四元数乘积因此有了更简单的形式

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

□ \mathbb{R}^3 中的向量乘法

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

基于四元数的 3D 变换

口四元数在图形学中的主要应用：3D 旋转 Rotations

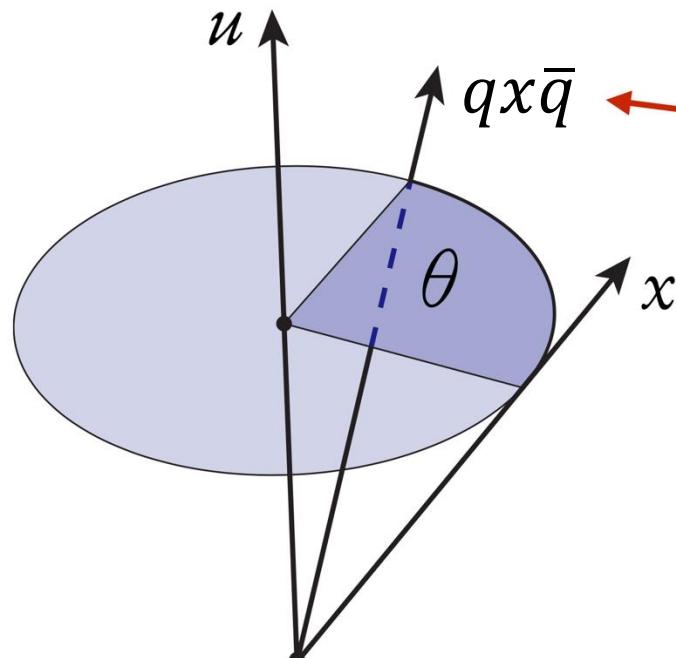
口考虑向量 x (pure imaginary) 和单位四元数 q

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

$$q = w + xi + yj + zk$$

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

$$w^2 + x^2 + y^2 + z^2 = 1$$



always expresses
some rotation

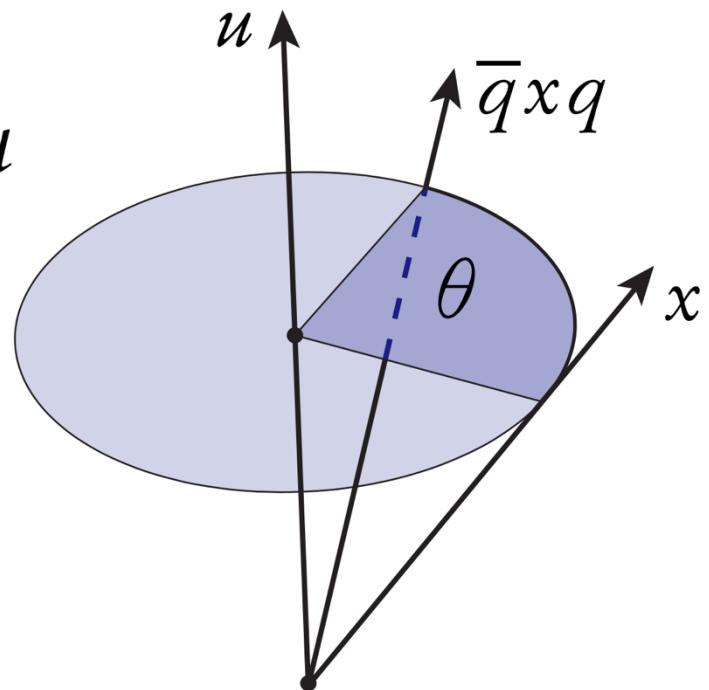
\bar{q} 是 q 的共轭复数 (conjugate),
通过改变虚部的符号得到, i.e.,
 $\bar{q} = w - xi - yj - zk$

回顾从轴/角度旋转

口给定轴 u , 角度 θ , 表示旋转的四元数 q 为

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

Angle **Unit vector**



基于四元数的 3D 变换

□ 给定 3D 向量 $x = (0, \mathbf{v})$ 和单位四元数 $q = (a, \mathbf{u})$

□ 旋转结果 $qx\bar{q}$

$$\begin{aligned} qx\bar{q} &= (-\mathbf{u} \cdot \mathbf{v}, a\mathbf{v} + \mathbf{u} \times \mathbf{v})(a, -\mathbf{u}) \\ &= (0, (a^2 - \mathbf{u} \cdot \mathbf{u})\mathbf{v} + 2(\mathbf{u} \cdot \mathbf{v})\mathbf{u} + 2a(\mathbf{u} \times \mathbf{v})) \end{aligned}$$

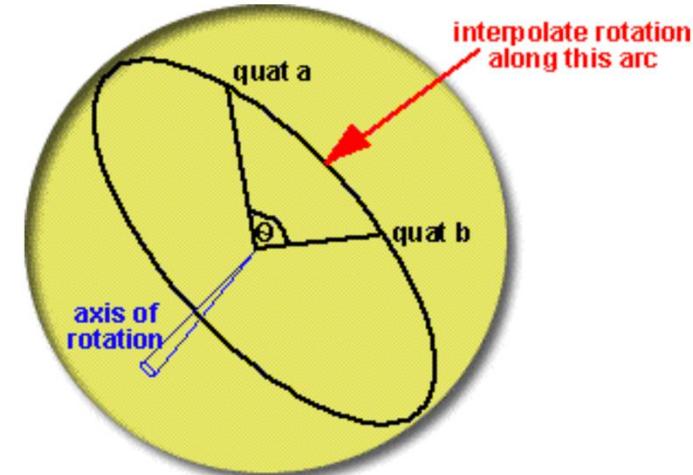
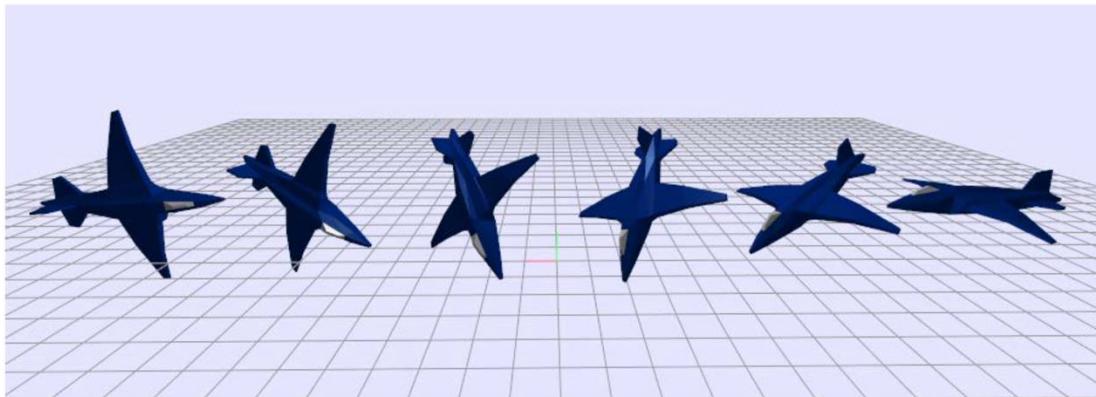
□ 比矩阵更容易记忆 (和操作)

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

对旋转进行插值

- 假设我们想在两个旋转之间平滑插值（例如，飞机转向）
- 插值欧拉角会产生**奇怪的路径**和**不均匀的旋转速度**
- 一个基于四元数的简单方法*： SLERP (spherical linear interpolation, 球面线性插值)

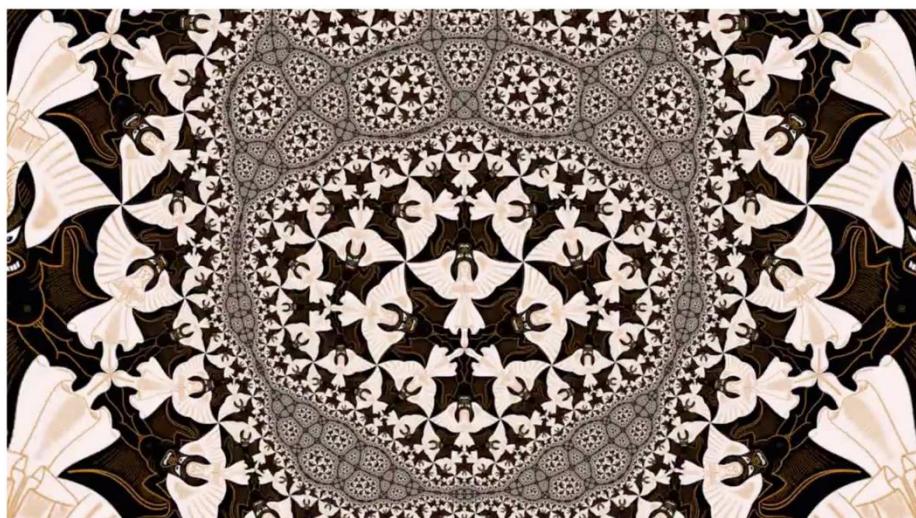
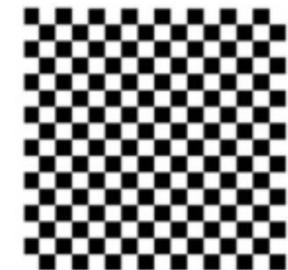
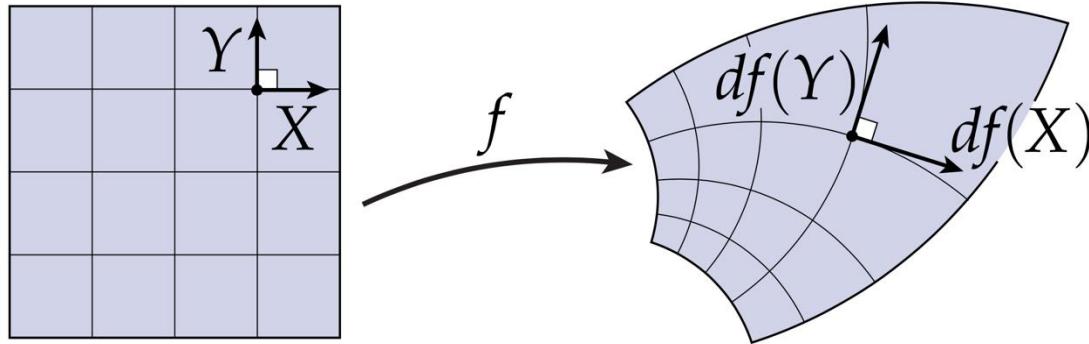
$$\text{Slerp}(q_0, q_1, t) = q_0(q_0^{-1}q_1)^t, \quad t \in [0, 1]$$



**在计算机图形学中，(超)复数
(hyper-)complex numbers
还有哪些用途？**

为纹理贴图生成坐标

口复数是保持角度 (共形 conformal) 映射的自然语言

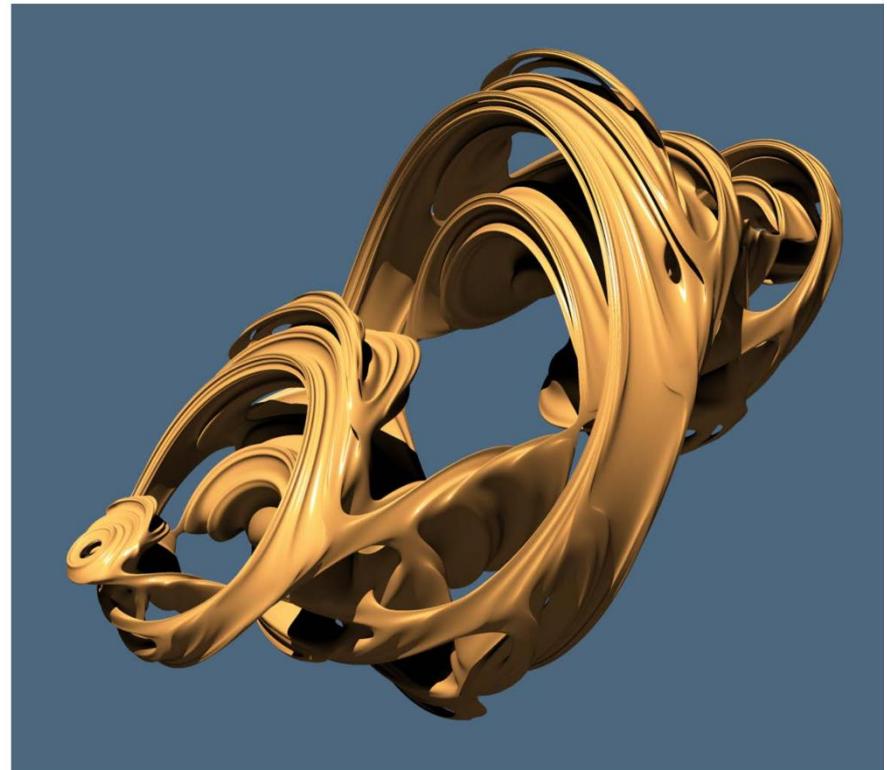
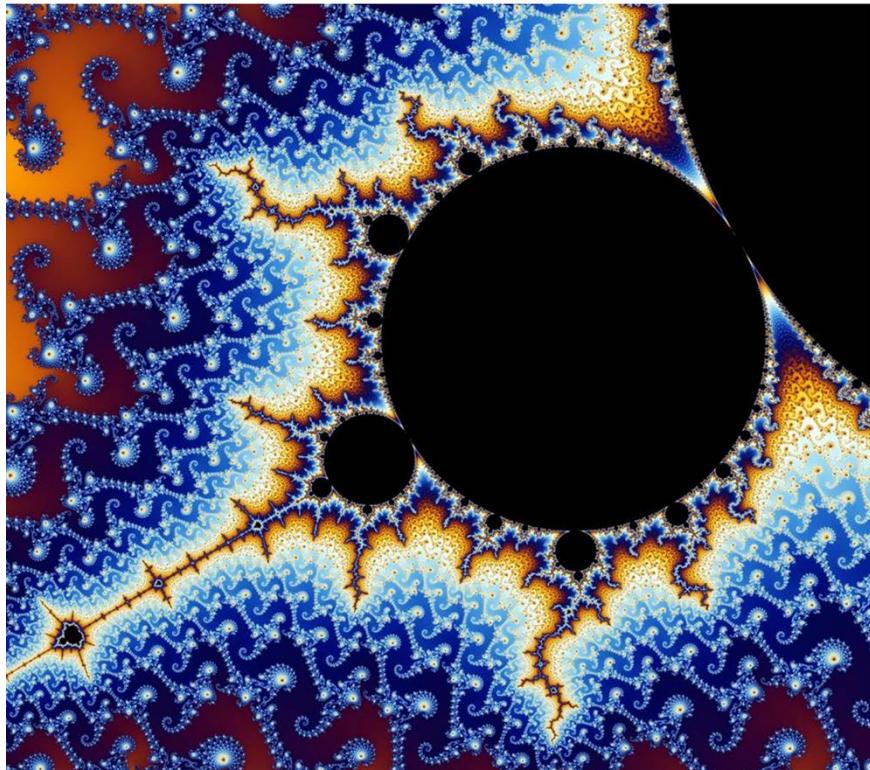


在纹理中保持角度，使其符合人类感官

<https://lixuan.xyz/blog/2015-08-28/1481.html>

无用但漂亮的例子：分形学 Fractals

□根据（超）复数的迭代所定义的



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

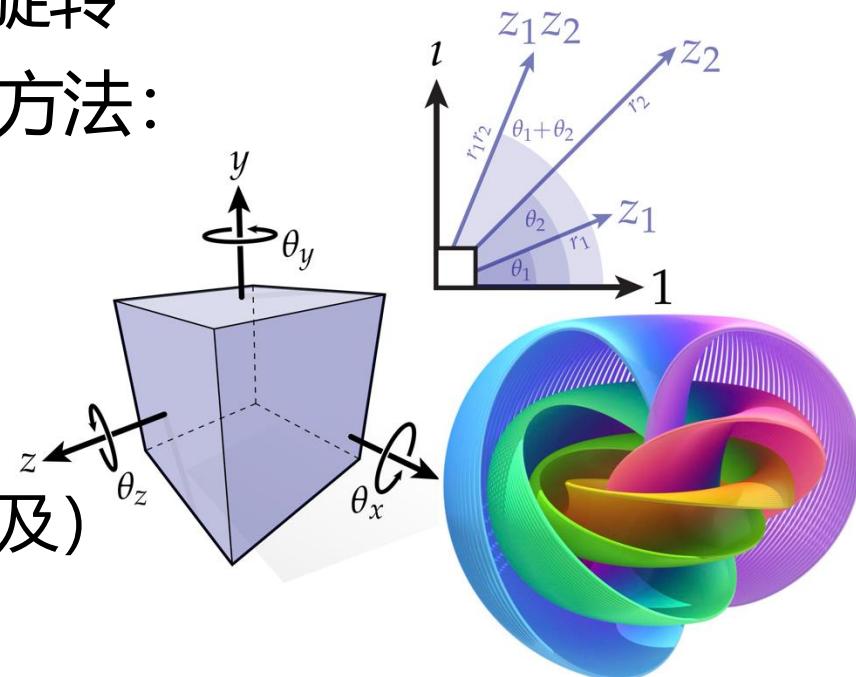
旋转与负数表示 – 总结

□ 在三维空间中，旋转非常复杂！

□ 今天，我们探讨了复数表示如何帮助理解
和处理三维（以及二维）中的旋转

□ 一般来说，有多种不同的表示方法：

- 欧拉角
- 轴角
- 四元数
- 李群 Lie group/李代数（未涉及）
- 几何代数（未涉及）



□ 没有“正确”或“最佳”的方法

□ 知道得越多，你能做的就越多！



中山大學 软件工程学院
SUN YAT-SEN UNIVERSITY SCHOOL OF SOFTWARE ENGINEERING

谢谢

陈壮彬
软件工程学院
chenzhb36@mail.sysu.edu.cn