



Lecture 06: 空间变换

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

□ 线性变换的基本介绍

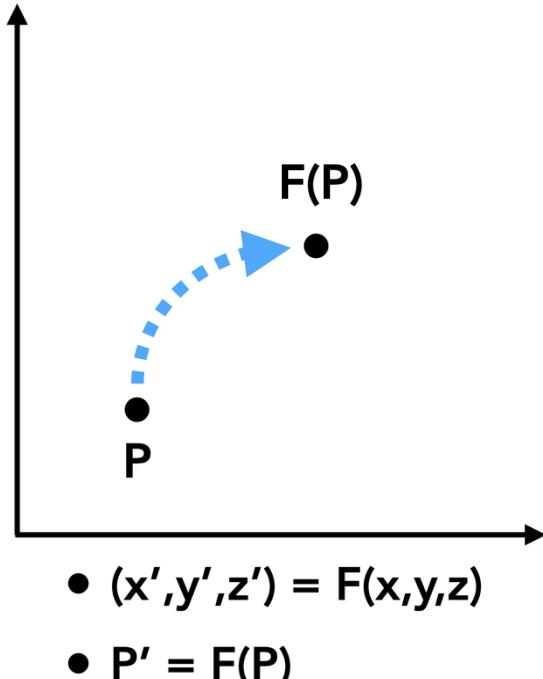
□ 常见的线性变换

□ 齐次坐标系

空间变换 Spatial transformation

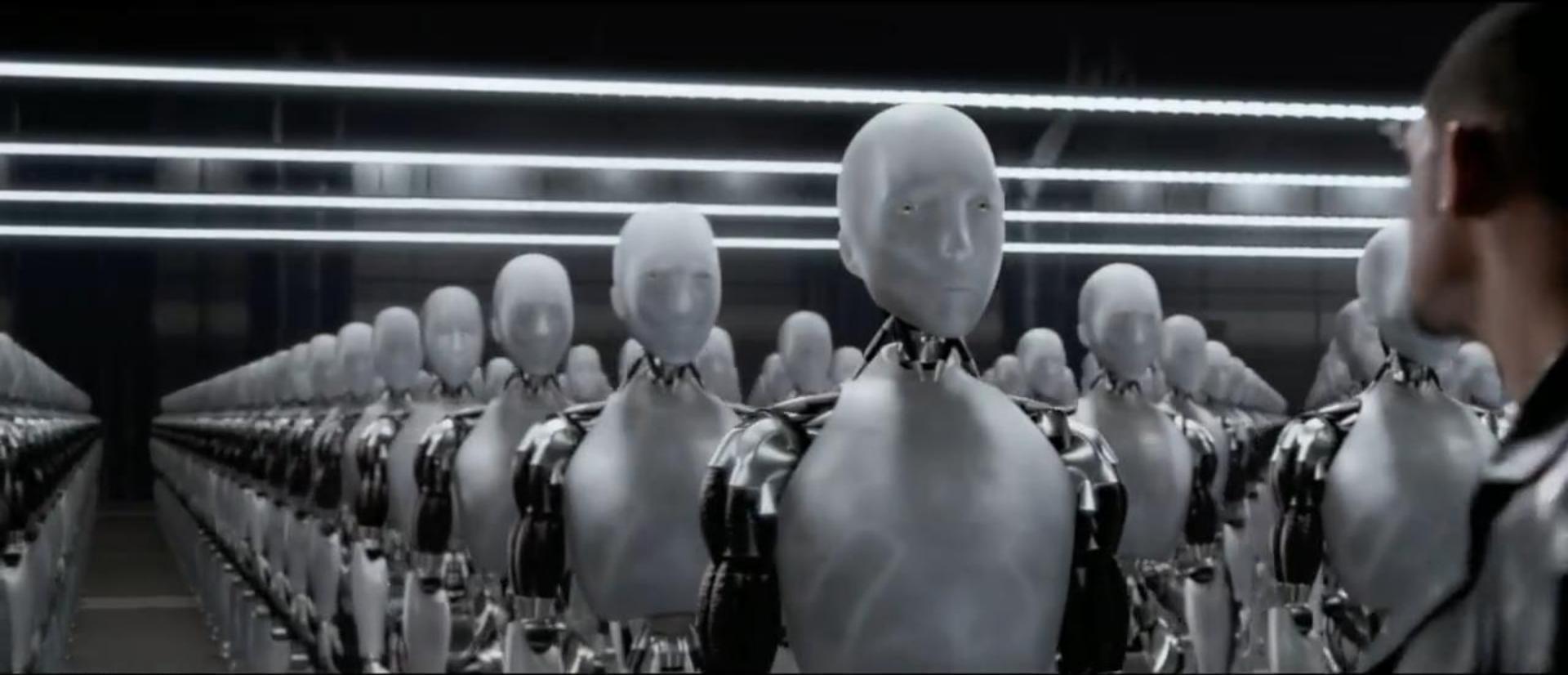
口任何为每个点分配新位置的函数

口关注常见的**基于线性映射 (linear maps) 的空间变换**,
如旋转 (rotation)、缩放 (scaling) 等



为什么要学习线性变换？

在空间中移动物体



FANDANGO
MOVIECLIPS

3D 物体建模



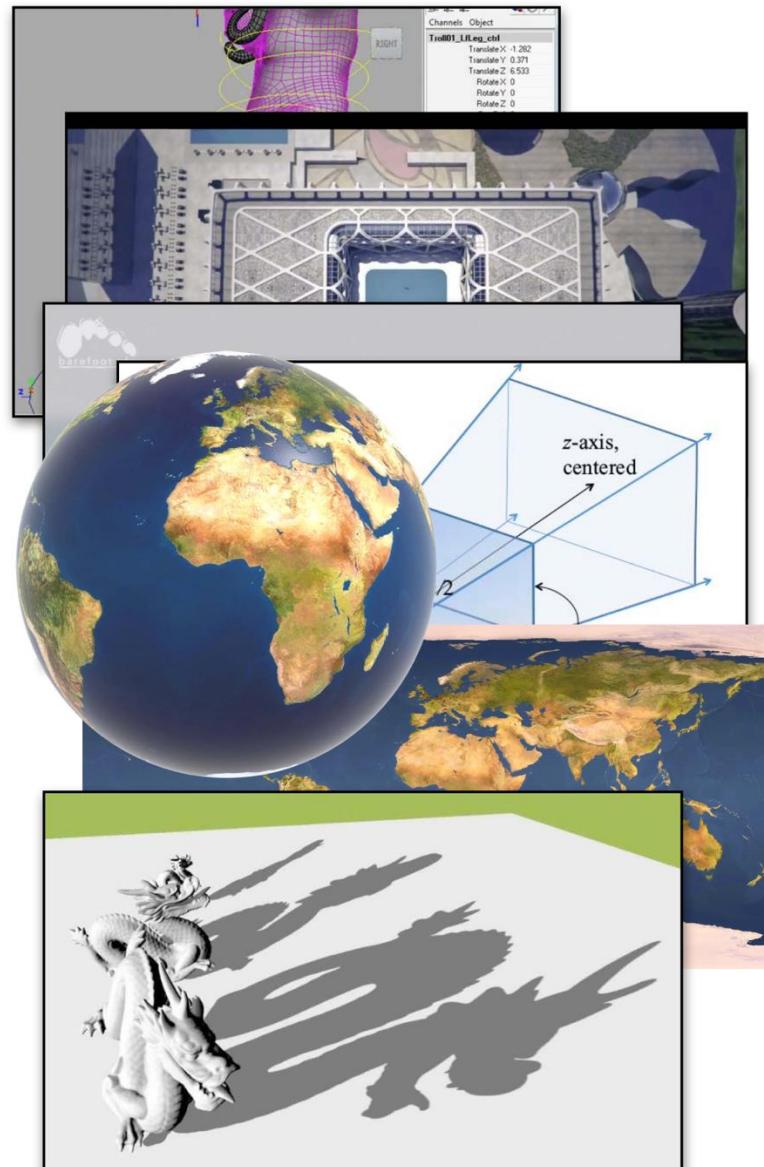
游戏设计



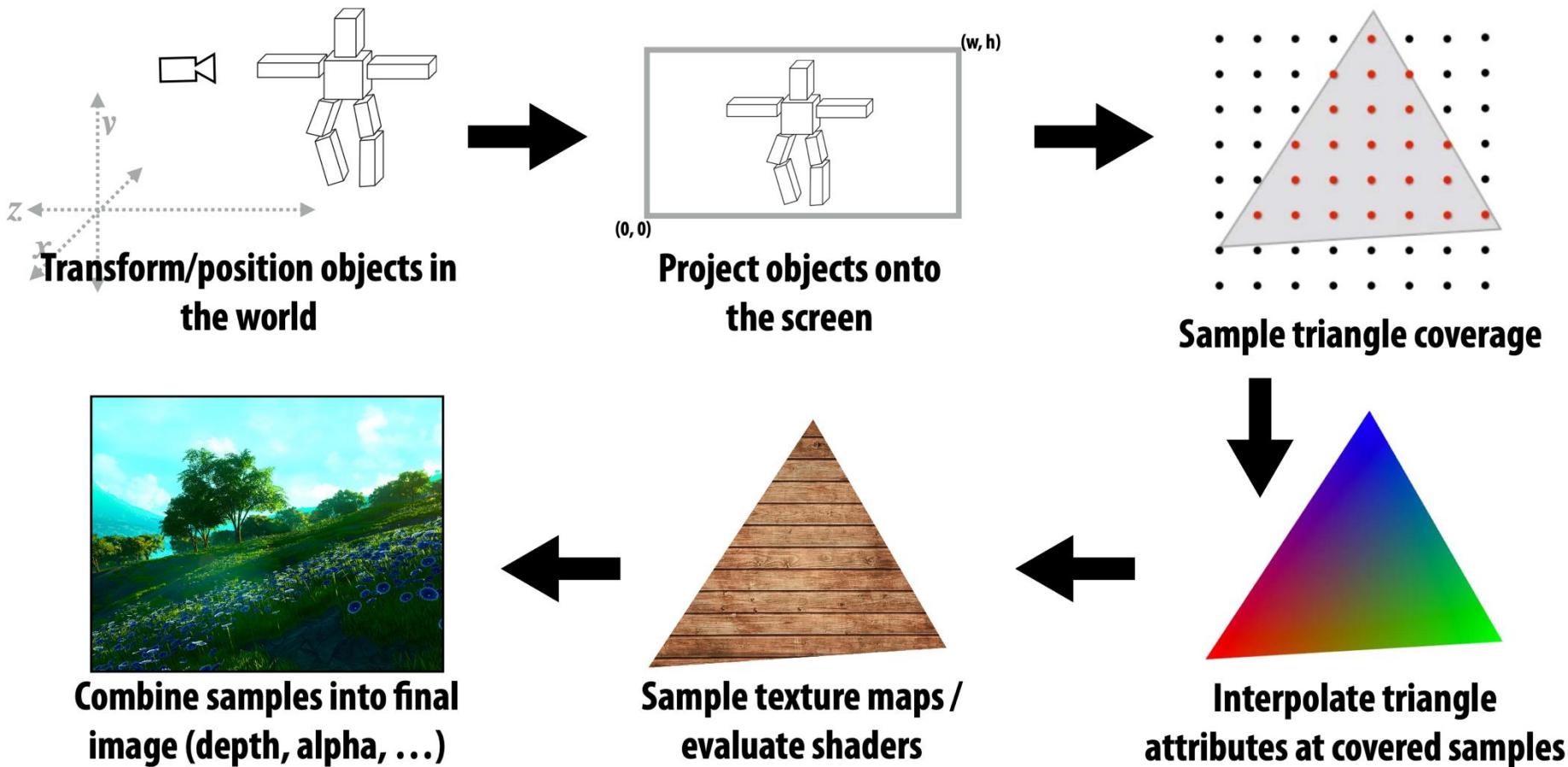
计算机图形学中的空间变换

□ All over the place!

- 在空间中定位/变形对象
- 移动相机
- 随时间设置对象动画
- 将 3D 对象投影到 2D 图像上
- 将 2D 纹理映射到 3D 对象上
- 将 3D 对象的阴影投影
到其他 3D 对象上
- ...



光栅化流程



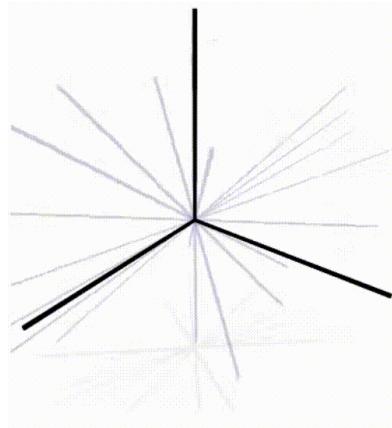
光栅化流程



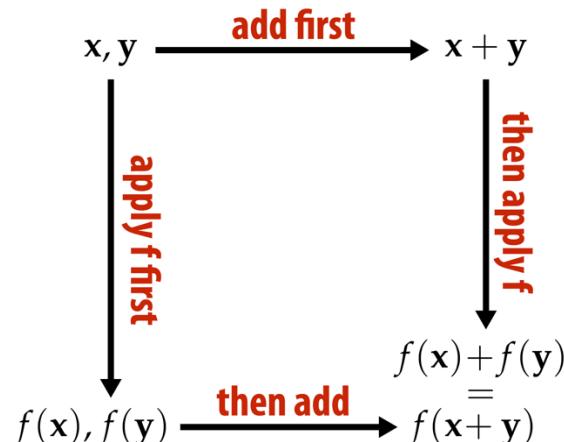
线性映射回顾

□ Q: 一个映射 $f: \mathbb{R}^n \rightarrow \mathbb{R}^n$ 是线性的意味着什么?

□ 几何上: 它将直线映射为直线, 并保持原点



□ 代数上: 保持向量空间运算 (加法和缩放)



为什么关注线性变换？

- 廉价/高效的操作
- 通常很容易求解（线性系统）
- 线性变换的组合还是线性的

- 多个矩阵的乘积是一个矩阵
- 给出变换的统一表示 (uniform representation)
- 简化图形算法和相关系统 (例如 GPU 和 API)

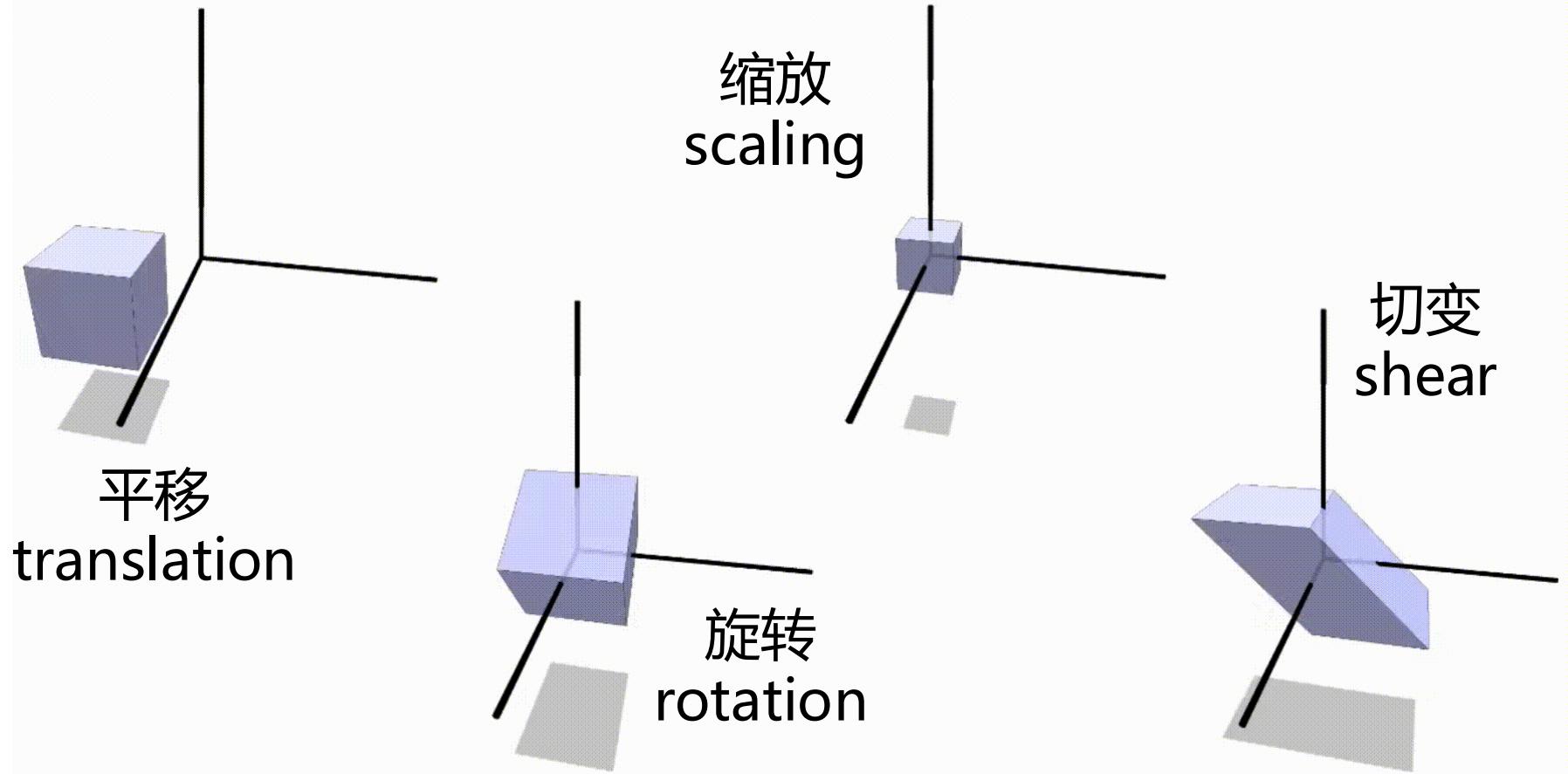
$$\begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{bmatrix} \begin{bmatrix} \cos \theta & 0 & \sin \theta \\ 0 & 1 & 0 \\ -\sin \theta & 0 & \cos \theta \end{bmatrix} \cdots = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix}$$

rotation *scale* *rotation* *composite transformation*

我们有哪些线性变换？

变换的种类

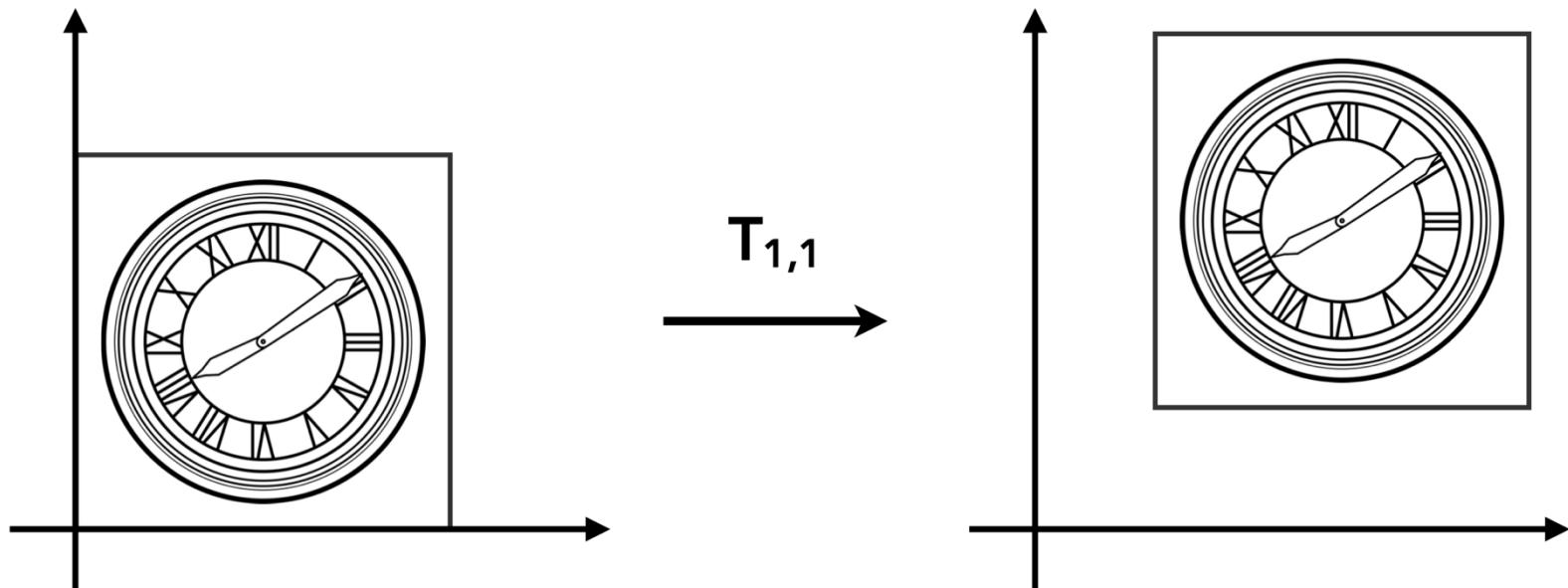
□ 你能说出下列每种变换的名字吗？



□ Q: 你是怎么知道的？(在没有看到具体公式的情况下)

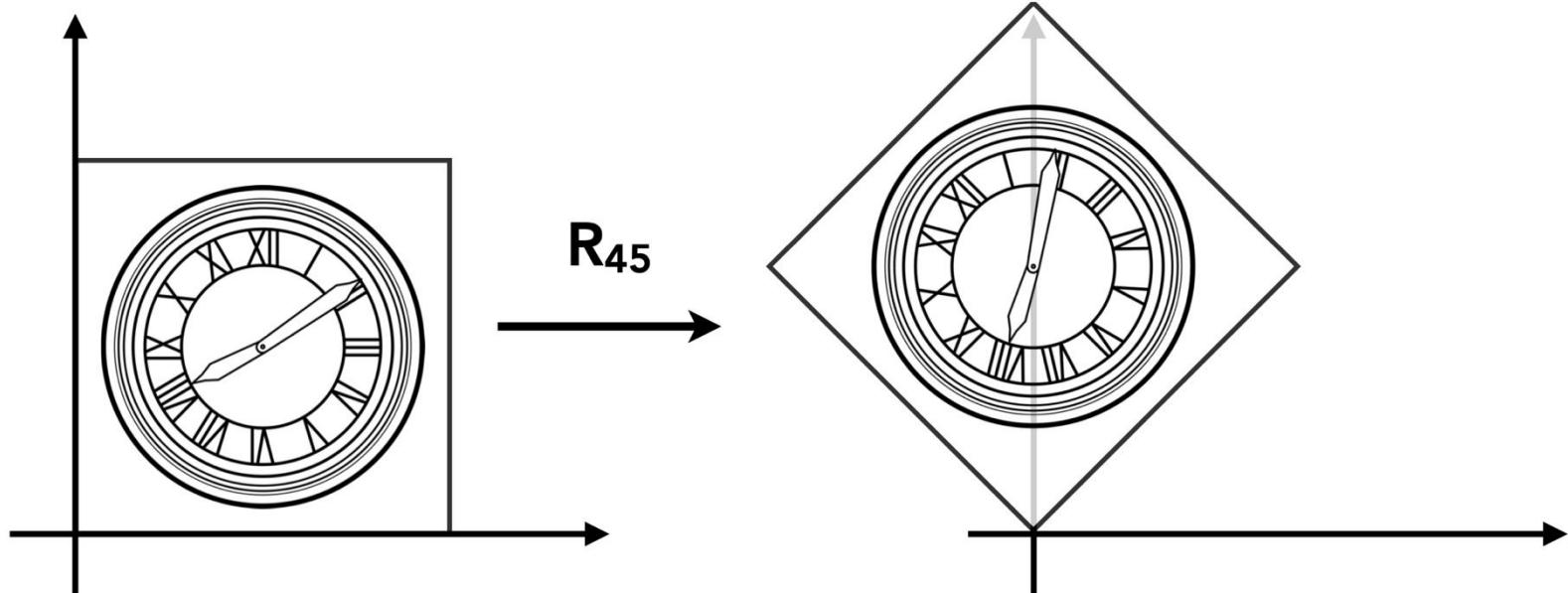
2D 视角的变换

□ 平移 Translation



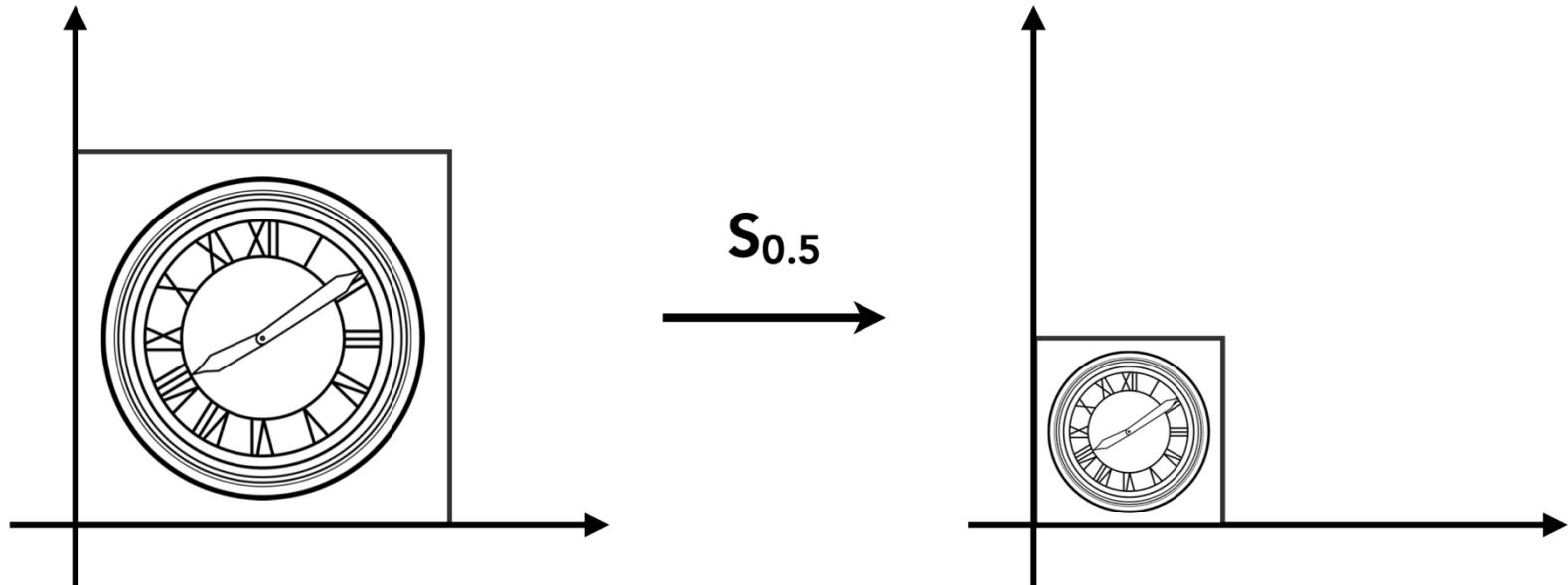
2D 视角的变换

□ 旋转 Rotation



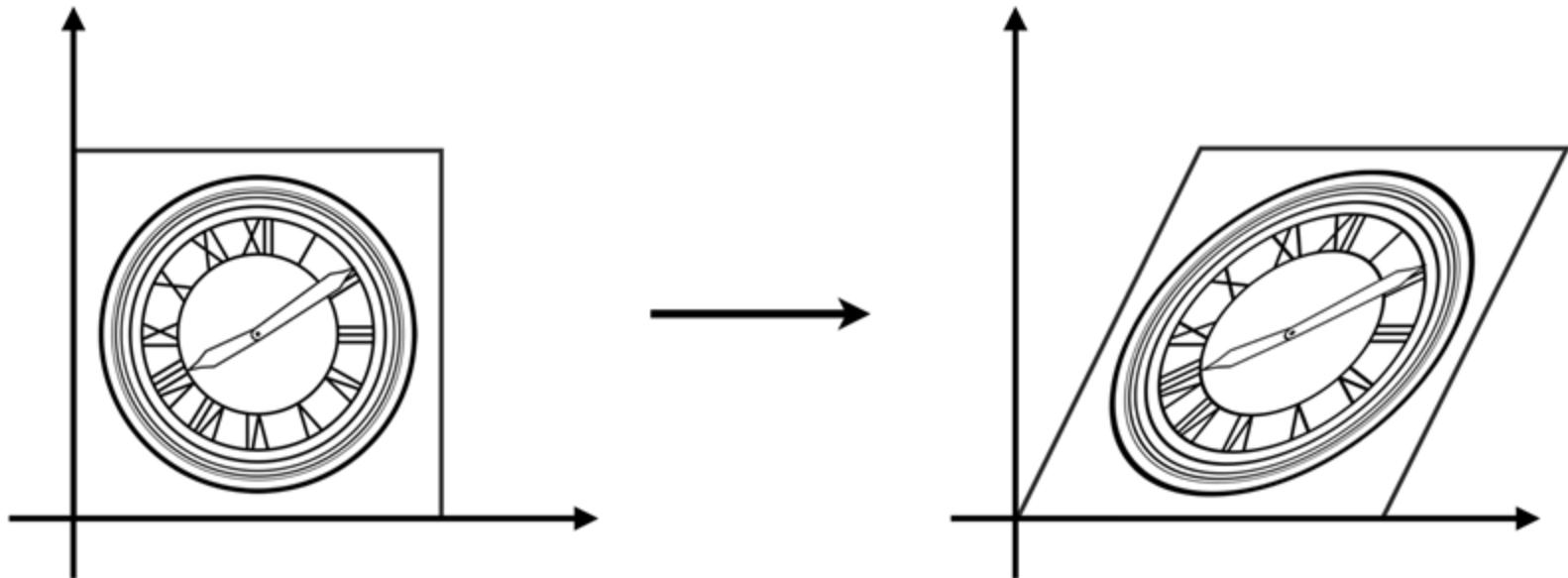
2D 视角的变换

□缩放 Scaling



2D 视角的变换

□切变 Shear



变换的不变性 invariants

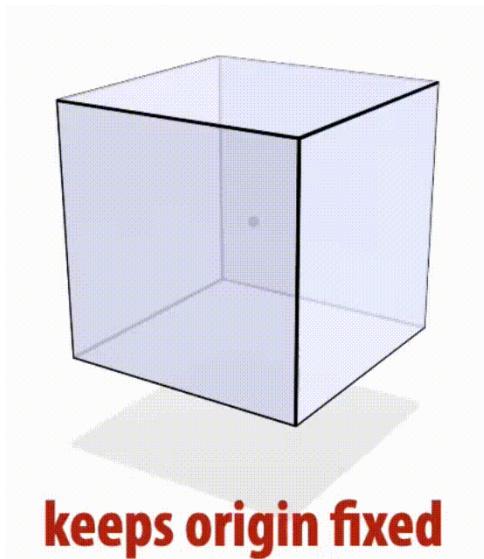
□一个变换是由它保留的不变性决定的

transformation	invariants	algebraic description
linear	<i>straight lines / origin</i>	$f(a\mathbf{x}+\mathbf{y}) = af(\mathbf{x}) + f(\mathbf{y}),$ $f(0) = 0$
translation	<i>differences between pairs of points</i>	$f(\mathbf{x}-\mathbf{y}) = \mathbf{x}-\mathbf{y}$
scaling	<i>lines through the origin / direction of vectors</i>	$f(\mathbf{x})/ f(\mathbf{x}) = \mathbf{x}/ \mathbf{x} $
rotation	<i>origin / distances between points / orientation</i>	$ f(\mathbf{x})-f(\mathbf{y}) = \mathbf{x}-\mathbf{y} ,$ $\det(f) > 0$
...

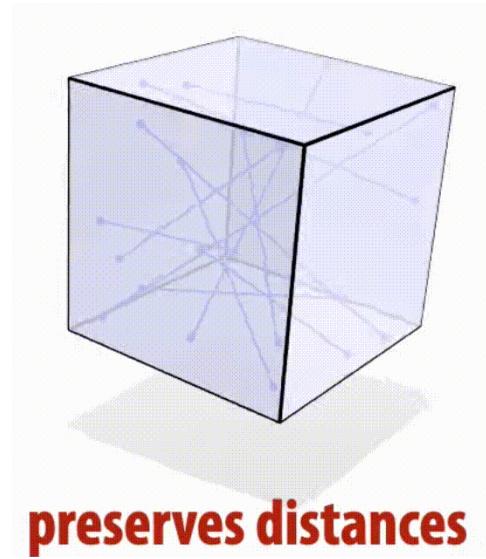
□这是本质上你的大脑能认出上述变换的原因

旋转 Rotation

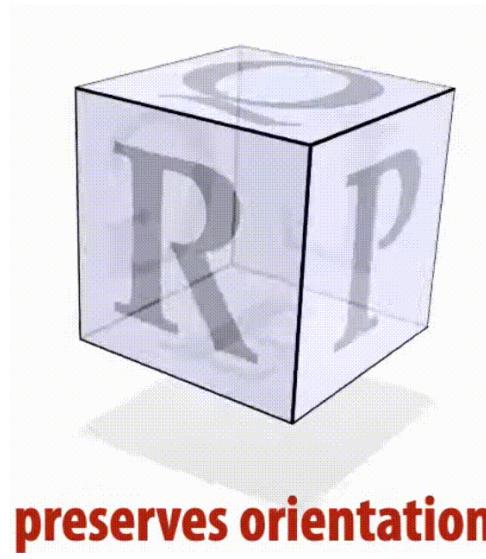
□ 旋转由三个基本特性定义：



keeps origin fixed



preserves distances



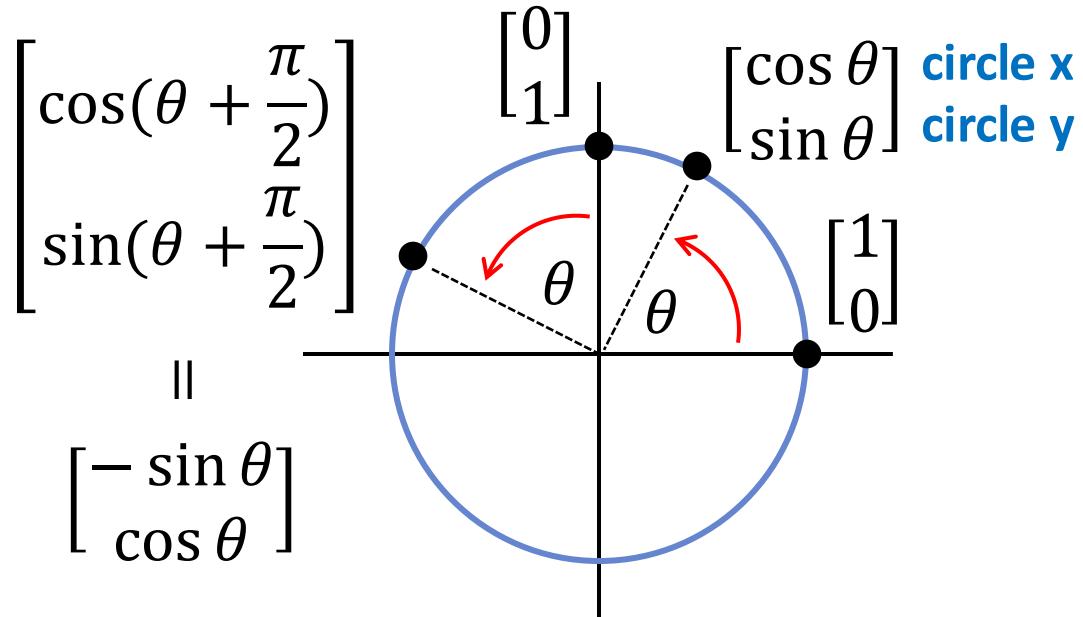
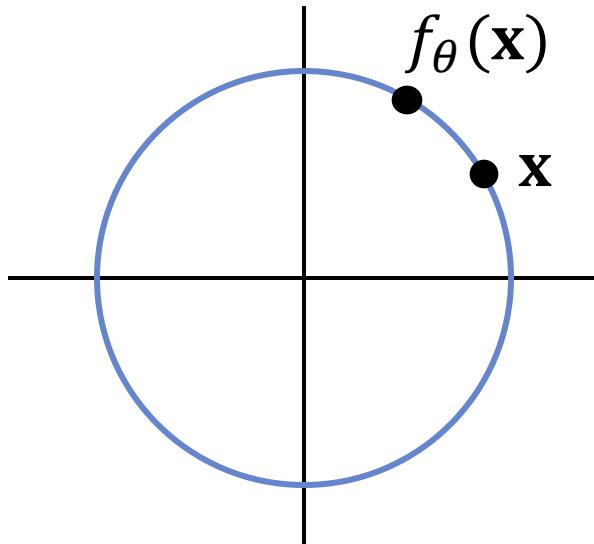
preserves orientation

□ 前两个性质已表明旋转是线性 (linear) 的

下节课将讨论更多关于旋转的内容

2D 旋转 – 矩阵表示

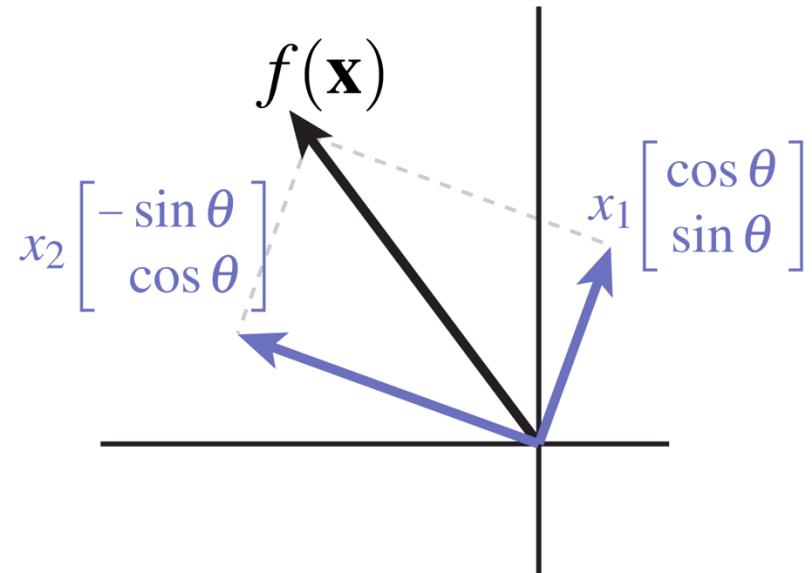
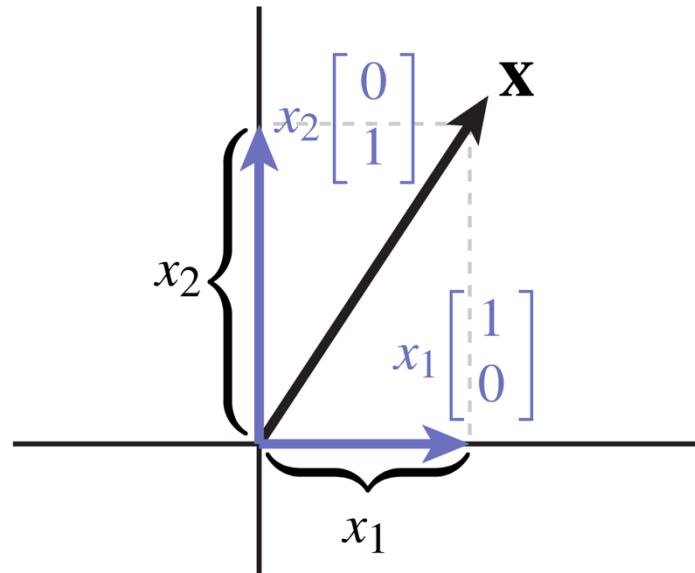
□ 旋转保留了长度和原点，因此，角度为 θ 的 2D 旋转将点 x 映射到点 $f_\theta(x)$ ，且 $f_\theta(x)$ 处在半径为 $|x|$ 的圆上



- 如果我们逆时针旋转 θ , $x = (1, 0)$ 将映射到哪个点?
- $x = (0, 1)$ 呢?

更一般的向量 $x = (x_1, x_2)$ 呢?

2D 旋转 – 矩阵表示



$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = x_1 \begin{bmatrix} 1 \\ 0 \end{bmatrix} + x_2 \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$f(\mathbf{x}) = x_1 \begin{bmatrix} \cos \theta \\ \sin \theta \end{bmatrix} + x_2 \begin{bmatrix} -\sin \theta \\ \cos \theta \end{bmatrix}$$

□ 我们要如何用矩阵表示一个 2D 旋转函数 $f_\theta(x)$?

即用矩阵表示
一个线性映射

$$f_\theta(\mathbf{x}) = \begin{bmatrix} \cos \theta & -\sin(\theta) \\ \sin \theta & \cos(\theta) \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

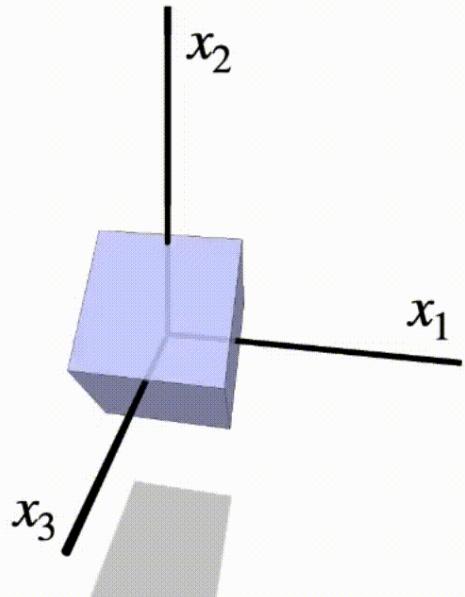
3D 旋转

□Q: 在 3D 中, 我们如何沿着 x_3 坐标轴旋转?

□A: 对 x_1 和 x_2 应用相同的转换, 保持 x_3 不变

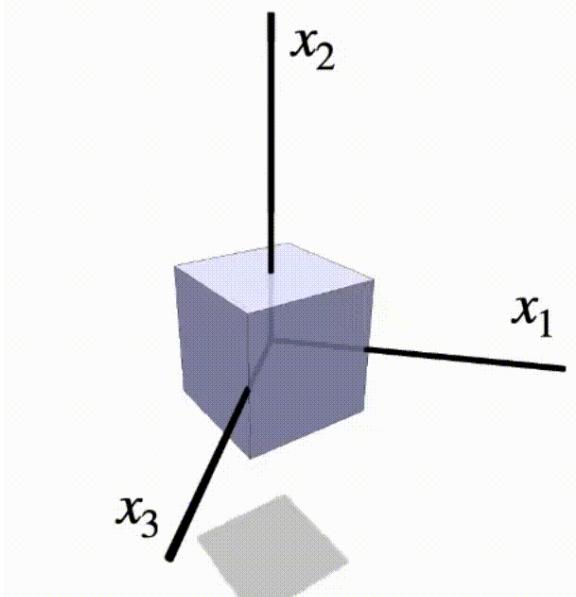
rotate around x_1

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & -\sin(\theta) \\ 0 & \sin \theta & \cos(\theta) \end{bmatrix}$$



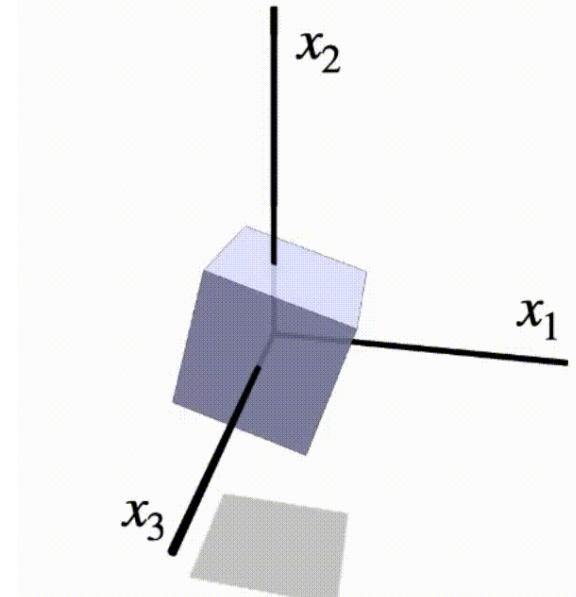
rotate around x_2

$$\begin{bmatrix} \cos \theta & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin \theta & 0 & \cos(\theta) \end{bmatrix}$$



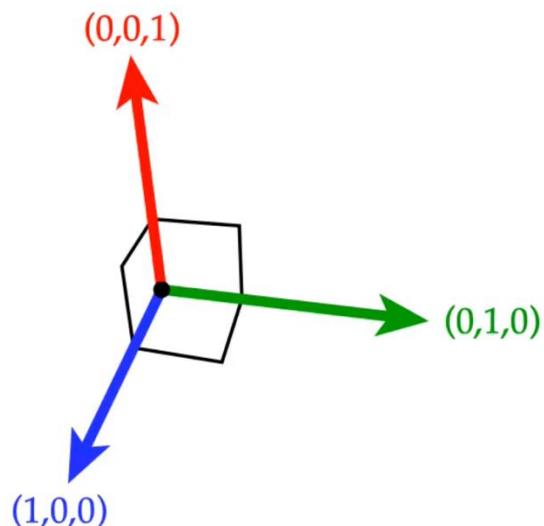
rotate around x_3

$$\begin{bmatrix} \cos \theta & -\sin(\theta) & 0 \\ \sin \theta & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



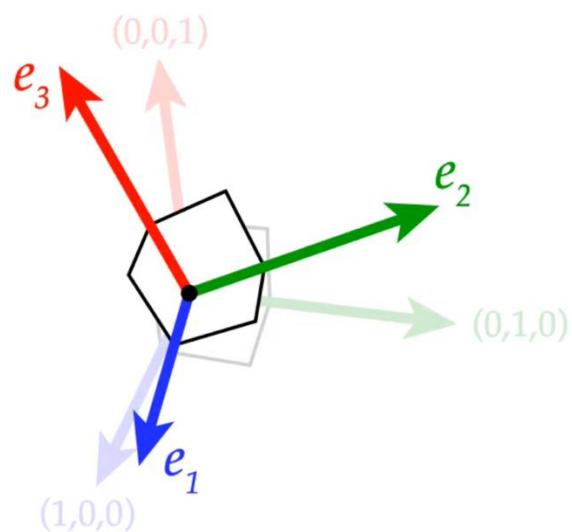
旋转变矩阵的转置等于其逆矩阵

口旋转将标准基 (standard basis) 映射到另一个正交基 (orthonormal basis) e_1, e_2, e_3



旋转变矩阵的转置等于其逆矩阵

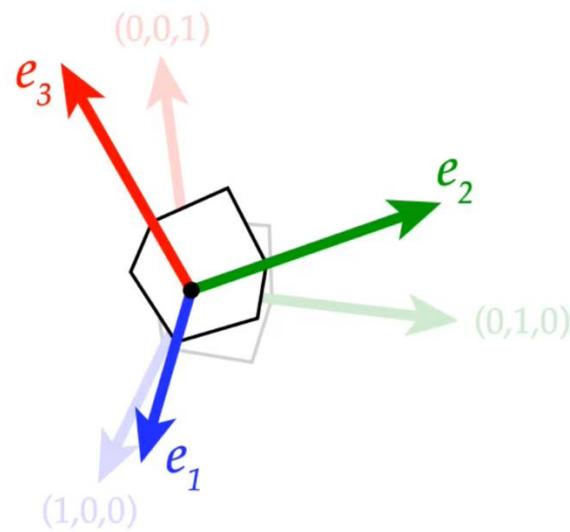
口旋转将标准基 (standard basis) 映射到另一个正交基 (orthonormal basis) e_1, e_2, e_3



$$\begin{aligned} R^T & \quad R \\ \left[\begin{array}{c|c|c} \text{---} e_1^T \text{---} & | & \text{---} e_1 \text{---} \\ \text{---} e_2^T \text{---} & | & \text{---} e_2 \text{---} \\ \text{---} e_3^T \text{---} & | & \text{---} e_3 \text{---} \end{array} \right] & = \left[\begin{array}{ccc} e_1^T e_1 & e_1^T e_2 & e_1^T e_3 \\ e_2^T e_1 & e_2^T e_2 & e_2^T e_3 \\ e_3^T e_1 & e_3^T e_2 & e_3^T e_3 \end{array} \right] \end{aligned}$$

旋转变矩阵的转置等于其逆矩阵

口旋转将标准基 (standard basis) 映射到另一个正交基 (orthonormal basis) e_1, e_2, e_3

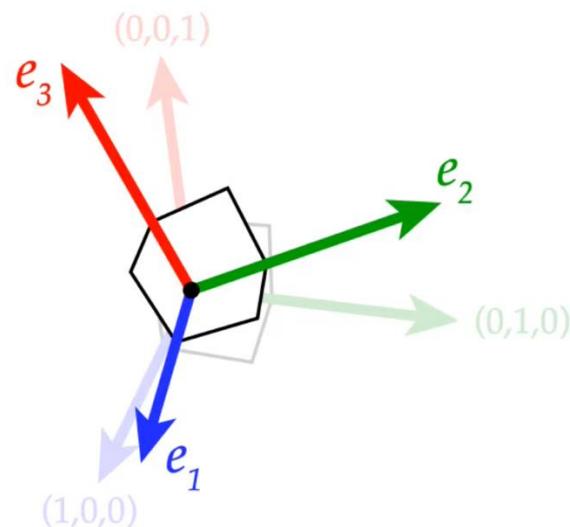


$$R^T = \begin{bmatrix} e_1^T & e_2^T & e_3^T \end{bmatrix} \quad R = \begin{bmatrix} e_1 & e_2 & e_3 \end{bmatrix}$$

$$= \begin{bmatrix} \text{Diagram showing the transformation of the standard basis vectors e1, e2, and e3 into the orthonormal basis e1, e2, e3 through a series of rotations.} \end{bmatrix}$$

旋转变矩阵的转置等于其逆矩阵

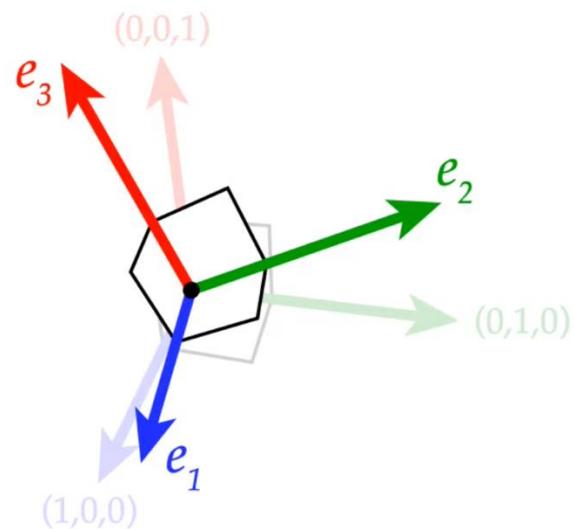
口旋转将标准基 (standard basis) 映射到另一个正交基 (orthonormal basis) e_1, e_2, e_3



$$\begin{aligned} & R^T \quad R \\ & \left[\begin{array}{c|c|c} \textcolor{blue}{e}_1^T & \textcolor{green}{e}_2^T & \textcolor{red}{e}_3^T \end{array} \right] \left[\begin{array}{c|c|c} e_1 & e_2 & e_3 \end{array} \right] \\ & = \left[\begin{array}{ccc} \textcolor{blue}{\swarrow} & 0 & 0 \\ 0 & \textcolor{green}{\searrow} & 0 \\ 0 & 0 & \textcolor{red}{\nwarrow} \end{array} \right] \end{aligned}$$

旋转变矩阵的转置等于其逆矩阵

□ 旋转将标准基 (standard basis) 映射到另一个正交基 (orthonormal basis) e_1, e_2, e_3



$$\begin{aligned} R^T &= \begin{bmatrix} e_1^T \\ e_2^T \\ e_3^T \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} I \\ R &= \begin{bmatrix} e_1 & e_2 & e_3 \end{bmatrix} \end{aligned}$$

□ 因此, $R^T R = I$, 或者 $R^T = R^{-1}$

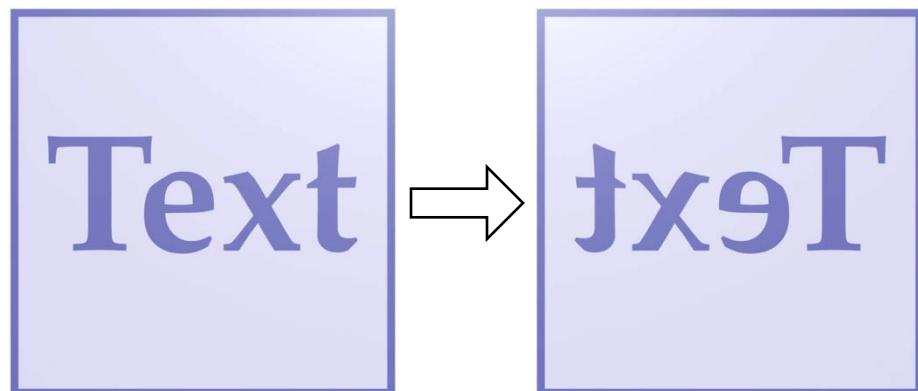
反射 Reflection

- Q: 是否所有满足 $Q^T Q = I$ 的正交矩阵均描述了一个旋转?
- 旋转必须保持**原点 origin**、**距离 distance** 和**方位 orientation**
- 考虑如下的矩阵

$$Q = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \quad Q^T Q = \begin{bmatrix} (-1)^2 & 0 \\ 0 & 1 \end{bmatrix} = I$$

- Q: 这个矩阵是否描述了一个旋转? 如果不是, 哪个量未能保持?

- A: No! 它代表了一个沿着 y 轴的反射, 因此未能保持方向

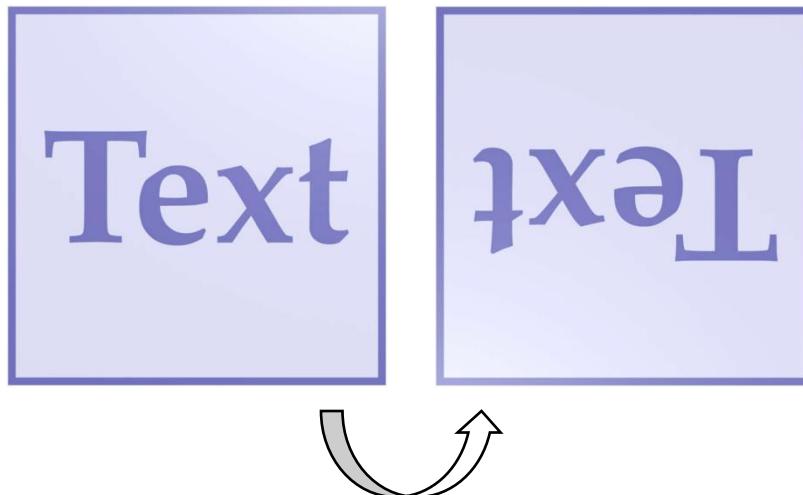


正交变换 Orthogonal transformations

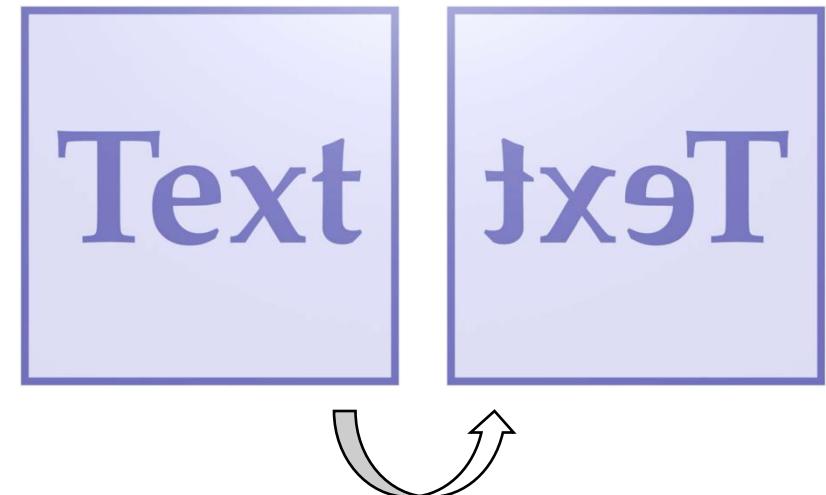
保持 **距离 distance** 和 **原点 origin** 的变换称为正交变换

用矩阵 $Q^T Q = I$ 表示

- **Rotations** additionally preserve orientation: $\det(Q) > 0$
- **Reflections** reverse orientation: $\det(Q) < 0$



rotation



reflection

缩放 Scaling

□ 每个向量 \mathbf{u} 映射到标量倍数

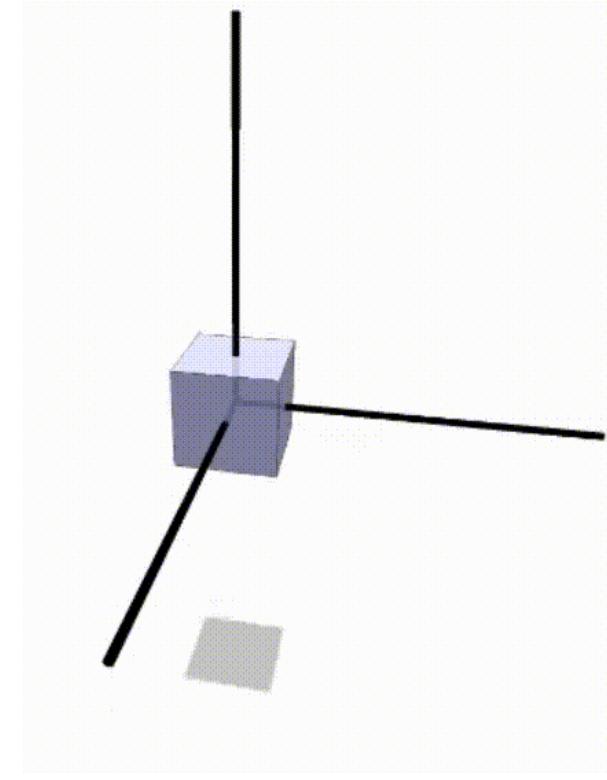
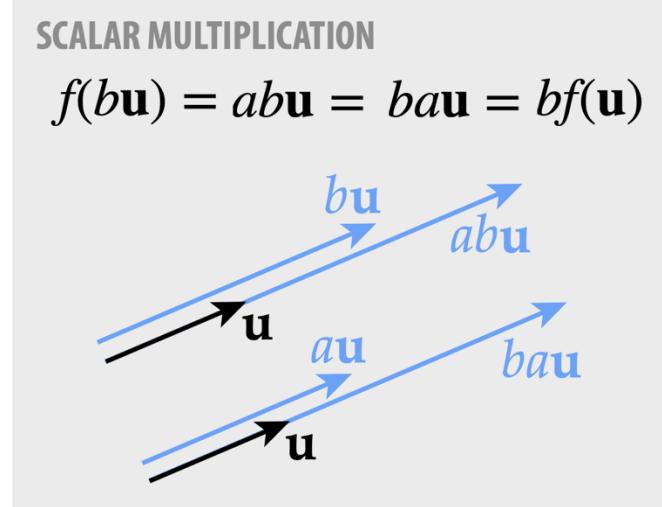
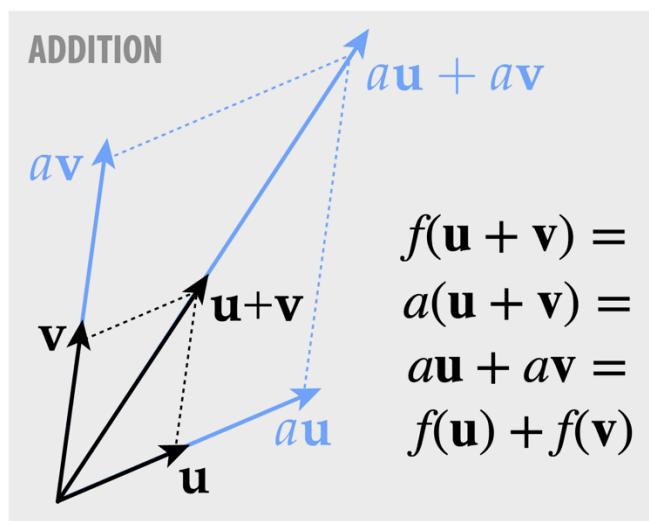
$$f(\mathbf{u}) = a\mathbf{u}, \quad a \in \mathbb{R}$$

□ 保持了所有向量的方向 direction*

$$\frac{\mathbf{u}}{|\mathbf{u}|} = \frac{a\mathbf{u}}{|a\mathbf{u}|} \quad * \text{假定 } a \neq 0, \mathbf{u} \neq \mathbf{0}$$

□ Q：缩放是线性变换吗？

□ A：是的



缩放 – 矩阵表示

□假设我们想以倍数 a 缩放向量 $\mathbf{u} = (u_1, u_2, u_3)$, 应该如何用矩阵表示此操作?

□只需构建一个对角矩阵 D , 其对角线上的元素为 a

$$\underbrace{\begin{bmatrix} a & 0 & 0 \\ 0 & a & 0 \\ 0 & 0 & a \end{bmatrix}}_D \underbrace{\begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}}_{\mathbf{u}} = \underbrace{\begin{bmatrix} au_1 \\ au_2 \\ au_3 \end{bmatrix}}_{a\mathbf{u}}$$

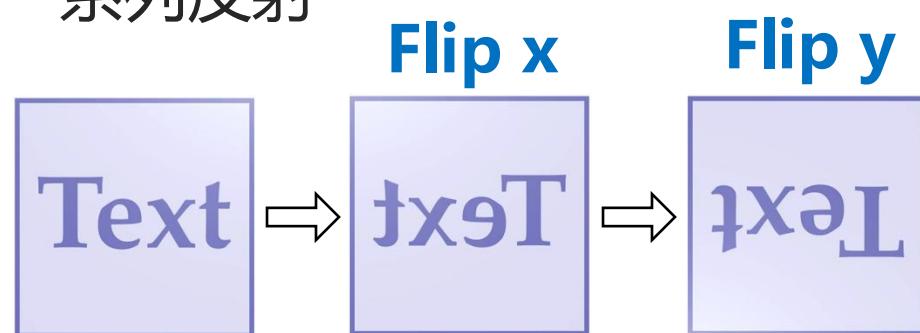
Q: What happens if a is negative?

负缩放 Negative scaling

□ 若 $a = -1$, 可以把缩放当成是一系列反射

□ 例如, 在 2D 中

$$\begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

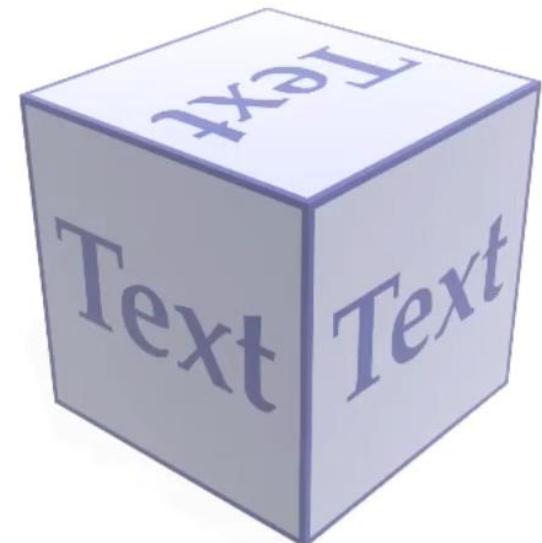


□ 每次反射均颠倒方位 orientation, 因此最终方位保持

□ 3D 呢?

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix} =$$

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$



□ 我们反射了三次, 因此方向颠倒了!

非均匀缩放 (轴对齐的)

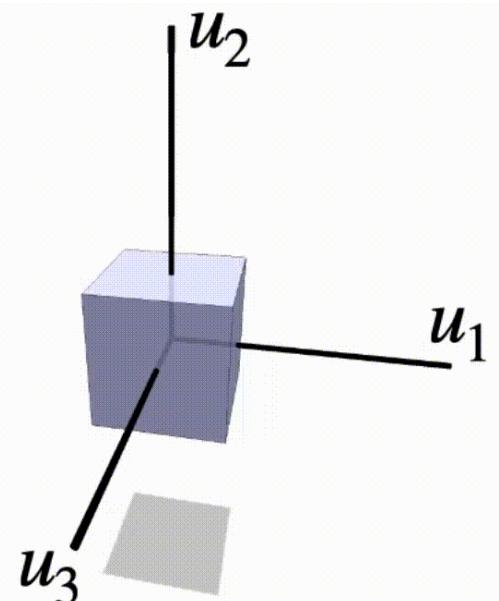
□ 我们还可以按不同的倍数缩放每个轴

$$f(u_1, u_2, u_3) = (au_1, bu_2, cu_3), \quad a, b, c \in \mathbb{R}$$

□ Q: 对应的矩阵表示是什么?

□ A: 只需把 a, b, c 放到矩阵的对角线上

$$\begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} au_1 \\ bu_2 \\ cu_3 \end{bmatrix}$$



如果我们想沿着其他非标准轴缩放呢?

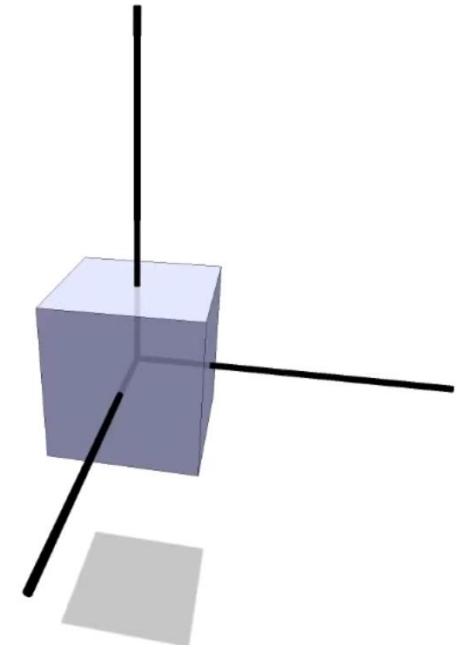
非均匀缩放

□ 我们可以

- 旋转到新的坐标轴 (R)
- 应用对角缩放 diagonal scaling (D)
- 旋转回到原来的坐标轴* (R^T)

□ 注意到整体的变换可以用一个对称矩阵 (symmetric matrix) 表示

$$A := R^T D R$$



$$f(\mathbf{x}) = R^T D R \mathbf{x}$$

Q: 是否所有对称矩阵都表示基于某些坐标轴的非均匀缩放?

*回忆一下对于旋转矩阵 R , 我们有 $R^{-1} = R^T$

谱定理 Spectral Theorem

□ A: 是的! 谱定理 (Spectral theorem) 表明, 对于一个对称矩阵 $A = A^T$, 其有

- 正交特征向量 (orthonormal eigenvectors) $e_1, \dots, e_n \in \mathbb{R}^n$
- 实特征值 (real eigenvalues) $\lambda_1, \dots, \lambda_n \in \mathbb{R}$

□ 可将此关系写为 $AR = RD$, 其中

$$Ae_i = \lambda_i e_i$$

$$R = \begin{bmatrix} e_1 & \cdots & e_n \end{bmatrix} \quad D = \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{bmatrix}$$

□ 亦即, $A = RDR^T$

□ 因此, 每个对称矩阵都表示沿着某组正交轴的非均匀缩放

□ 如果是正定的 (positive definite), 即 $\lambda_i > 0$, 那么缩放是正的 (i.e., non-negative scaling)

切变 Shear

□ 切变将沿着方向 \mathbf{u} 移动 \mathbf{x} , 移动的大小由 \mathbf{x} 沿着固定向量 \mathbf{v} 的距离决定:

$$f_{\mathbf{u}, \mathbf{v}}(\mathbf{x}) = \mathbf{x} + \langle \mathbf{v}, \mathbf{x} \rangle \mathbf{u}$$

点积计算 \mathbf{x} 沿着 \mathbf{v} 的距离

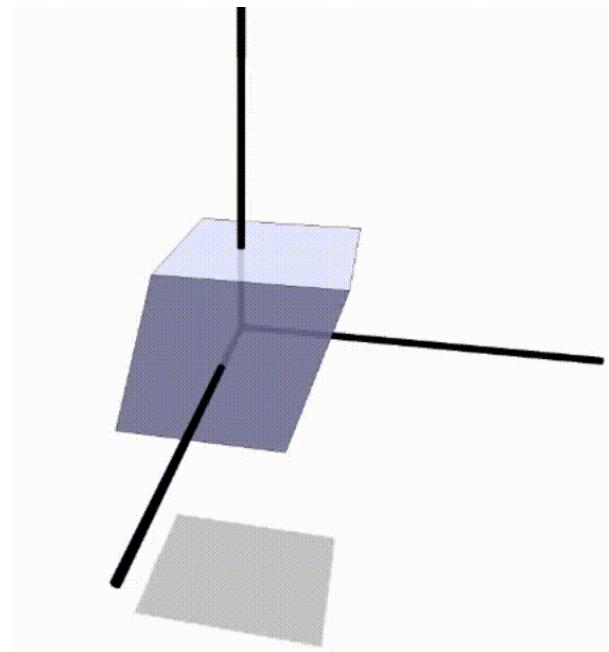
□ Q: 此变换是否线性?

□ A: Yes, 可用如下矩阵表示

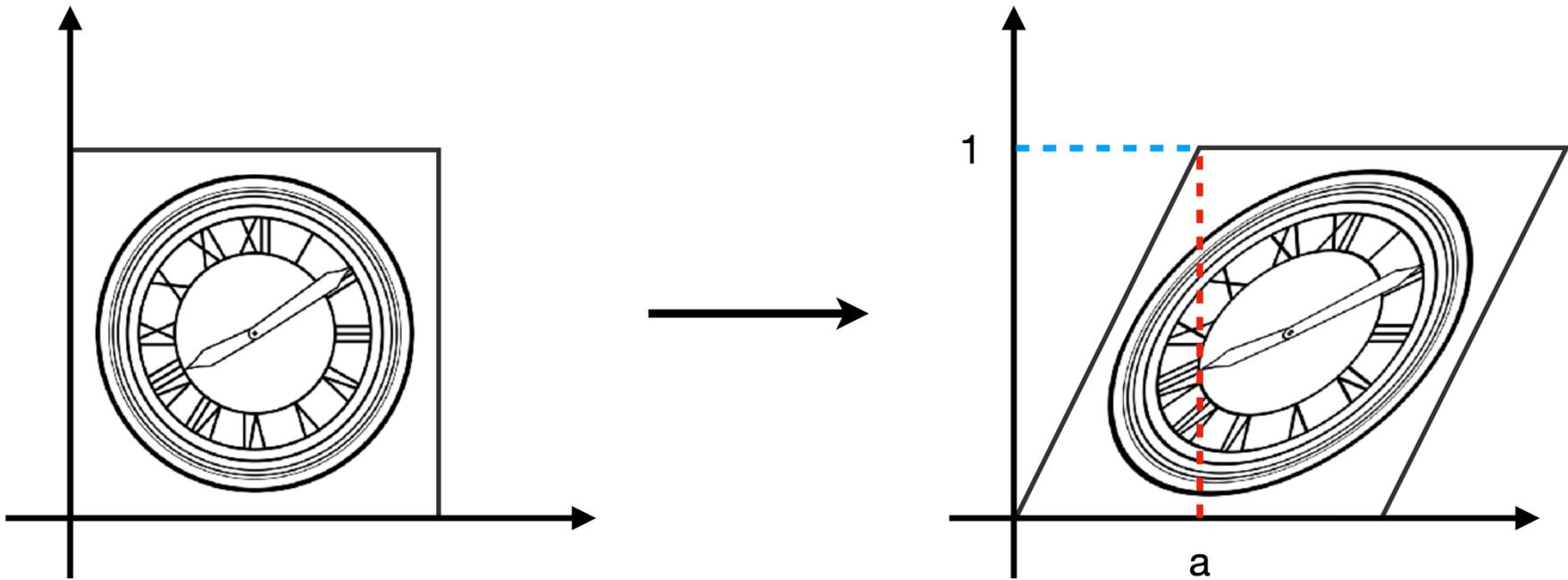
$$A_{\mathbf{u}, \mathbf{v}} = I + \mathbf{u}\mathbf{v}^T$$

□ 例子

$$\begin{aligned} \mathbf{u} &= (\cos(t), 0, 0) & A_{\mathbf{u}, \mathbf{v}} &= \begin{bmatrix} 1 & \cos(t) & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ \mathbf{v} &= (0, 1, 0) \end{aligned}$$



切变 Shear – 2D



Hints:

Horizontal shift is 0 at $y=0$

Horizontal shift is a at $y=1$

Vertical shift is always 0

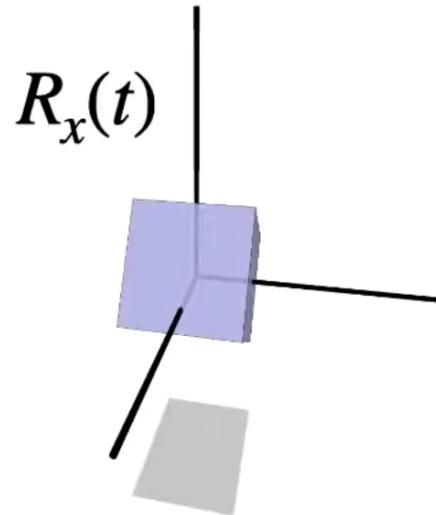
$$f_{\mathbf{u}, \mathbf{v}}(\mathbf{x}) = \mathbf{x} + \langle \mathbf{v}, \mathbf{x} \rangle \mathbf{u}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\begin{cases} \mathbf{u} = [1, 0] \\ \mathbf{v} = [0, a] \end{cases}$$

复合变换 Composite Transformations

根据这些基本变换 (旋转、反射、缩放、切变...), 我们现在可以通过**矩阵乘法**进行复合变换



The last
matrix gets
applied first!

**How do we decompose a linear transformation into pieces?
(rotations, reflections, scaling, ...)**

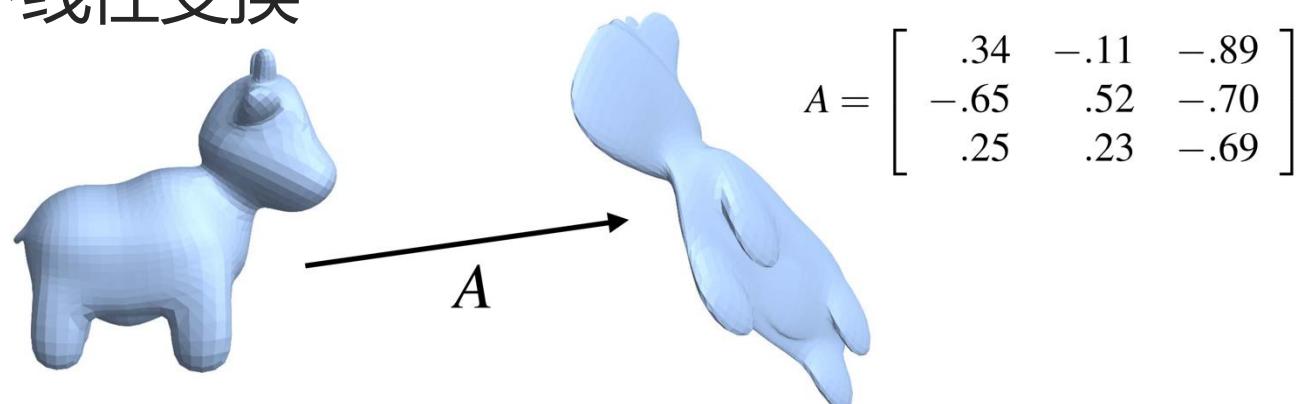
线性变换的分解

□ 一般来说，**没有唯一的方法**可以将给定的线性变换分解成基本变换的组合！

□ 然而，存在很多有用的分解

- 奇异值分解 (singular value decomposition), **适合信号处理**
- LU 分解 (LU factorization), **适合求解线性系统**
- 极性分解 (polar decomposition), **适合空间变换**
- ...

□ 考虑如下线性变换



极性和奇异值分解

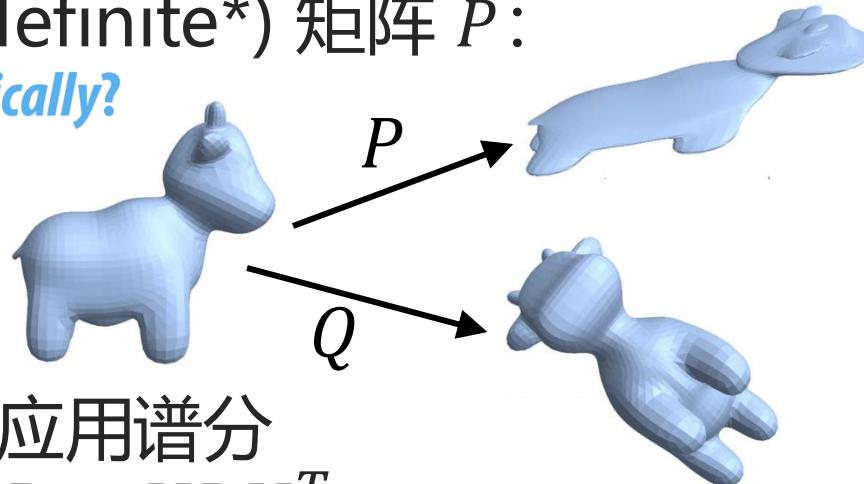
*Both Q and V are orthogonal, so is U

□ 极性分解将任何矩阵 A 分解为正交 (orthogonal) 矩阵 Q 和对称半正定 (positive-semidefinite*) 矩阵 P :

Q: What do each of the parts mean geometrically?

$$A = QP$$

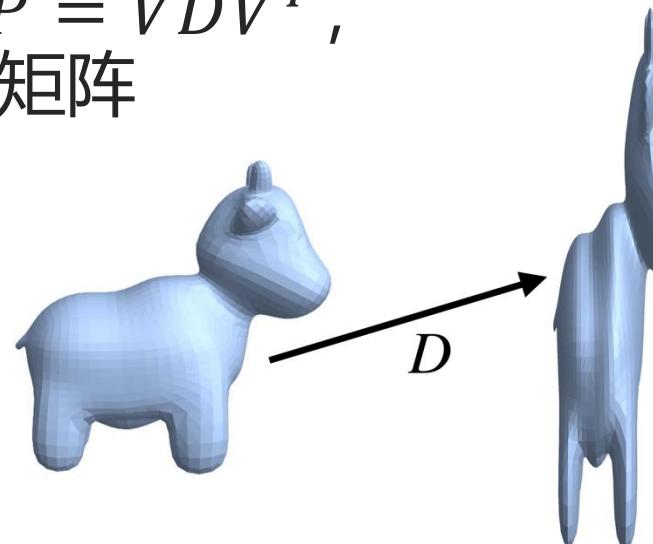
rotation/reflection nonnegative
nonuniform scaling



□ 由于 P 是对称的，可以进一步应用谱分解 (spectral decomposition) $P = VDV^T$ ，其中 V 为正交矩阵*， D 为对角矩阵

$$A = \underbrace{QV}_{U} DV^T = UDV^T$$

③ rotation ② axis-aligned scaling ① rotation



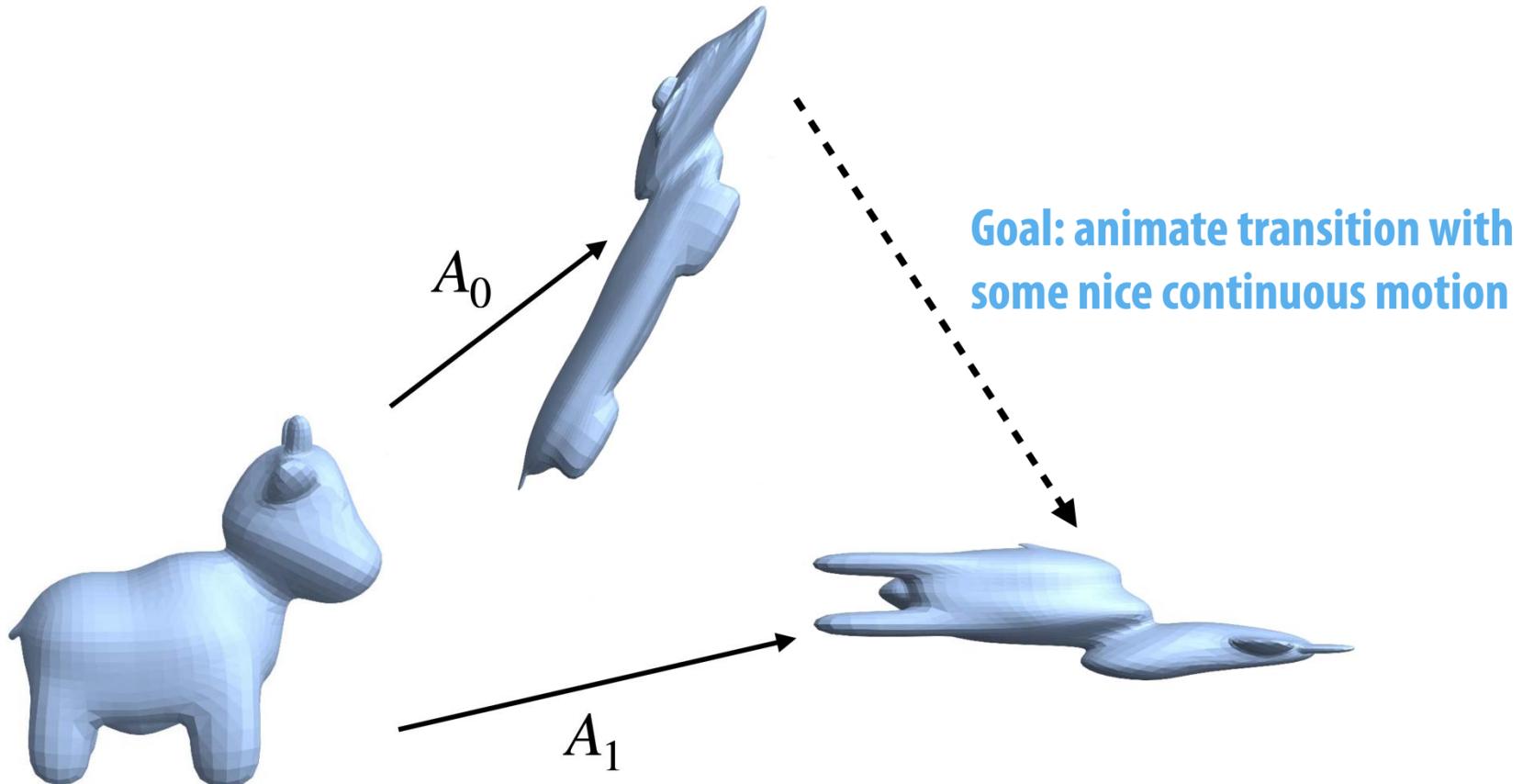
□ $A = UDV^T$ 也称奇异值分解

* https://en.wikipedia.org/wiki/Positive_semidefinite

插值变换 Interpolating Transformations

□ 这些分解对图形变换有什么用？

□ 比如在某个模型的两个线性变换 A_0, A_1 之间进行插值



插值变换 – 线性的

一个想法是：将两个矩阵进行线性组合，按当前时间加权
 $t \in [0, 1]$

$$A(t) = (1 - t)A_0 + tA_1$$



正确命中了起点和终点，但中间的变换看起来很糟糕...

插值变换 – 极性的

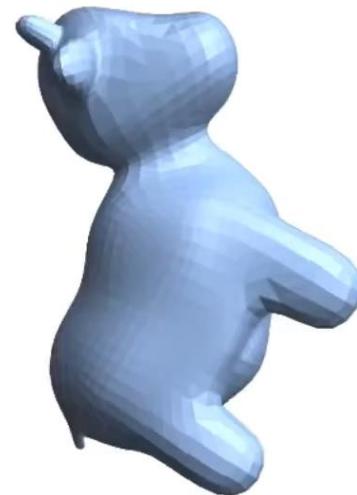
更好的想法是：单独插值极性分解的不同分量

$$A_0 = Q_0 P_0, \quad A_1 = Q_1 P_1$$

scaling



rotation



final interpolation



$$P(t) = (1 - t)P_0 + tP_1$$

$$\begin{aligned}\widetilde{Q}(t) &= (1 - t)Q_0 + tQ_1 \\ \widetilde{Q}(t) &= Q(t)X(t)\end{aligned}$$

$$A(t) = Q(t)P(t)$$

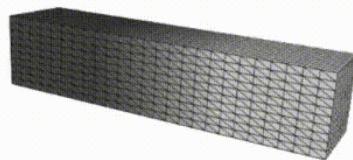
looks better...

示例：线性混合蒙皮 (Skinning)

口朴素的线性插值在角色变换之间进行混合时也会导致伪影，即“糖果包装效应”

口关于变换混合的替代方法有很多研究

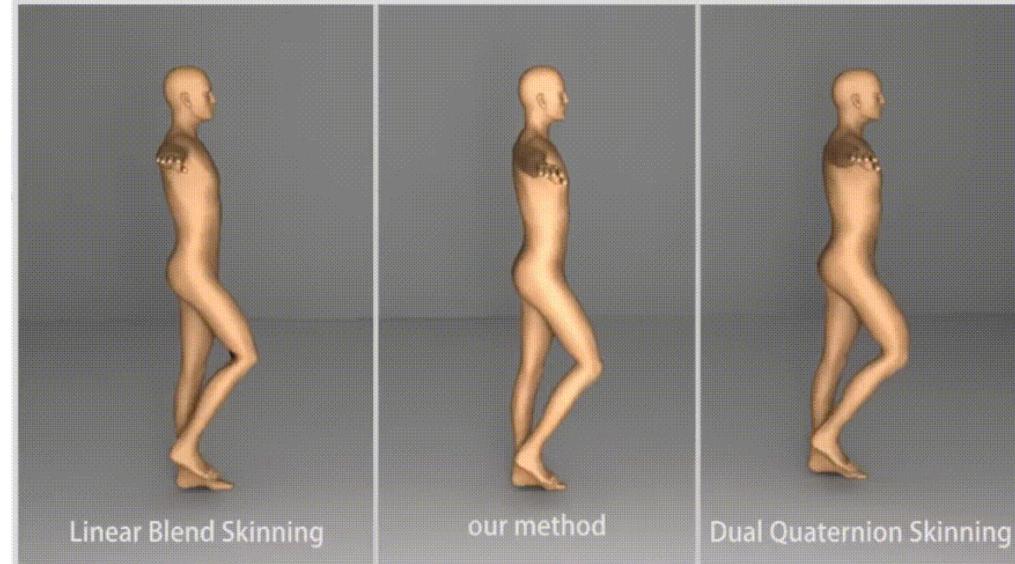
LBS: candy-wrapper artifact



14

Jacobson, Deng, Kavan, & Lewis (2014)
"Skinning: Real-time Shape Deformation"

Rumman & Fratarcangeli (2015)
"Position-based Skinning for Soft Articulated Characters"



平移 Translations

□ 到目前为止，我们忽略了一个基本的变换 – 平移

□ 平移只是在给定点 x 上加一个偏移 u

$$f_u(x) = x + u$$

□ Q: 此变换是否线性?

□ 让我们检查其是否符合定义

additivity

$$f_u(x + y) = x + y + u$$

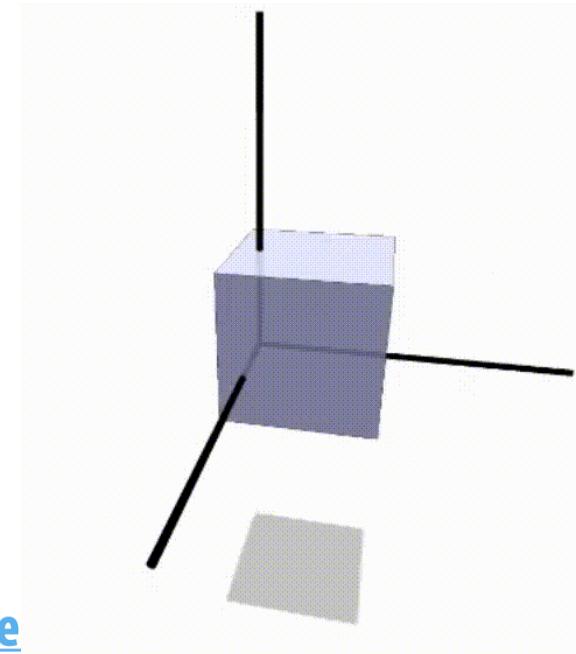
$$f_u(x) + f_u(y) = x + y + 2u$$

homogeneous

$$f_u(ax) = ax + u$$

$$af_u(x) = ax + au$$

A: No, 平移是仿射, 而不是线性的



变换的组成

□ 我们可以通过矩阵乘法来组成线性变换

$$A_3(A_2(A_1 \mathbf{x})) = (A_3 A_2 A_1) \mathbf{x}$$

□ 组成平移很简单 – 仅需把向量加起来

$$f_{\mathbf{u}_3}(f_{\mathbf{u}_2}(f_{\mathbf{u}_1}(\mathbf{x}))) = f_{\mathbf{u}_1 + \mathbf{u}_2 + \mathbf{u}_3}(\mathbf{x})$$

□ 如何混合平移以及其他线性变换（旋转、缩放、切变等）？

$$A_2(A_1 \mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2 = (A_2 A_1) \mathbf{x} + (A_2 \mathbf{b}_1 + \mathbf{b}_2)$$

□ 现在我们必须跟踪一个矩阵和一个向量

□ 此外，我们还将看到，这种编码不适用于其他重要情况，例如透视转换 (perspective transformations)

There must be a better way...

Strange idea:
**如果我们进入四维，那么平移可
能就是线性变换了...**



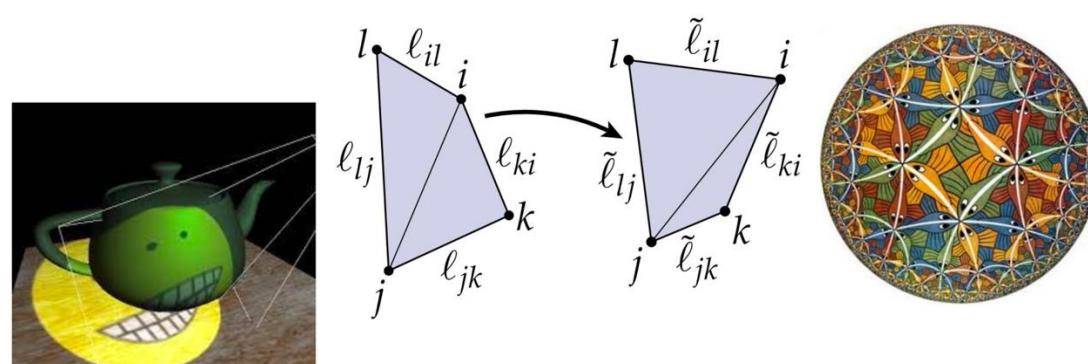
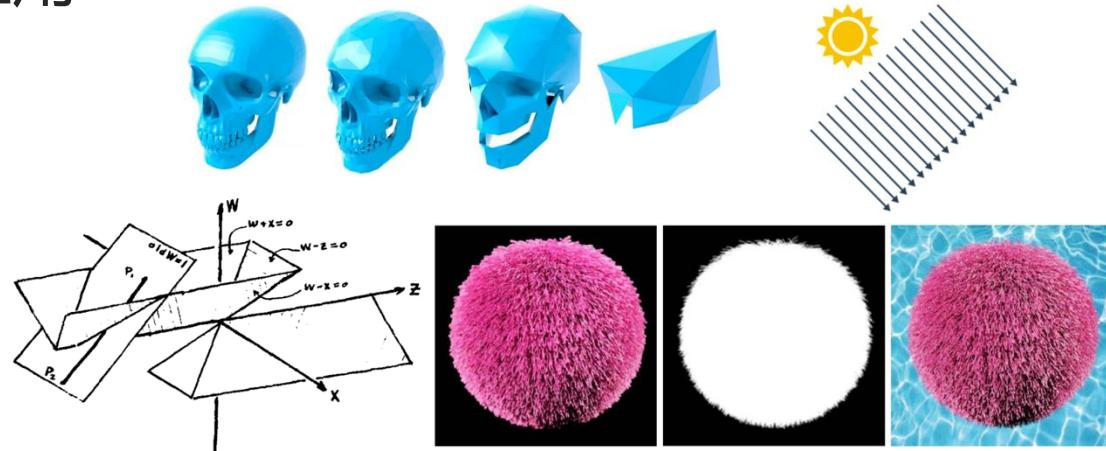
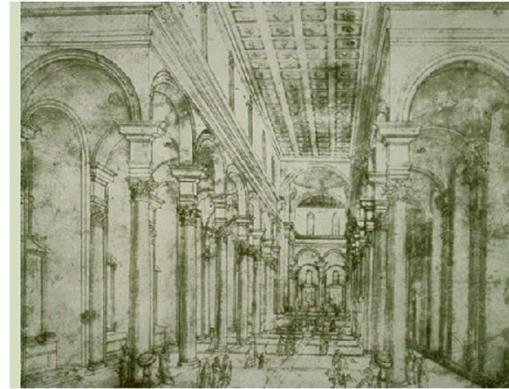
齐次坐标 Homogeneous Coordinates

□ 源于透视的相关研究

□ 一种为直线指定坐标的自然方式 (由 Möbius 引入)

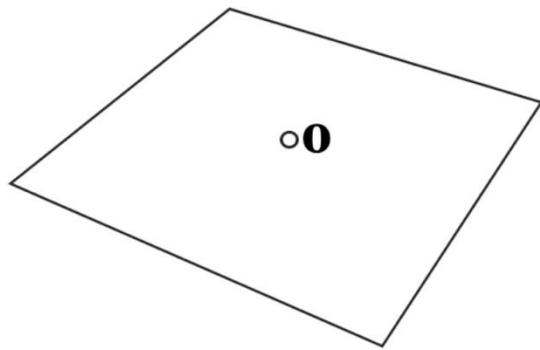
□ 在图形学中有大量应用

- 三维变换
- 透视投影
- 二次误差简化
- 预乘 alpha
- 阴影映射
- 投影纹理映射
- 离散共形几何
- 双曲几何
- 剪裁
- 方向灯



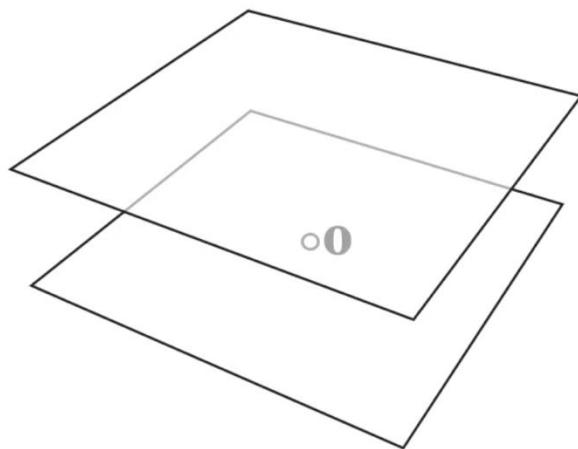
齐次坐标 – 基本理念

□ 考虑任何不通过 3D 原点 o 的 2D 平面



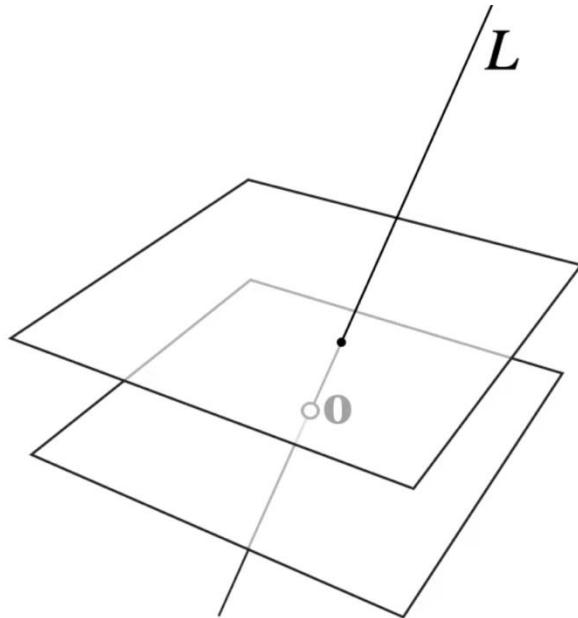
齐次坐标 – 基本理念

□ 考虑任何不通过 3D 原点 o 的 2D 平面



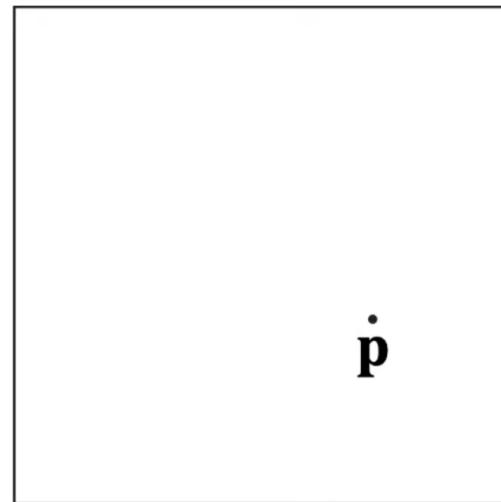
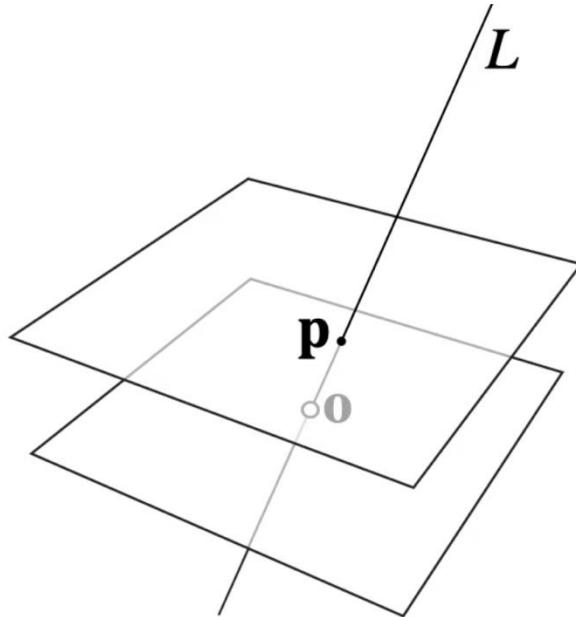
齐次坐标 – 基本理念

- 考虑任何不通过 3D 原点 o 的 2D 平面
- 通过 3D 原点的线将穿过 2D 平面中的一个点



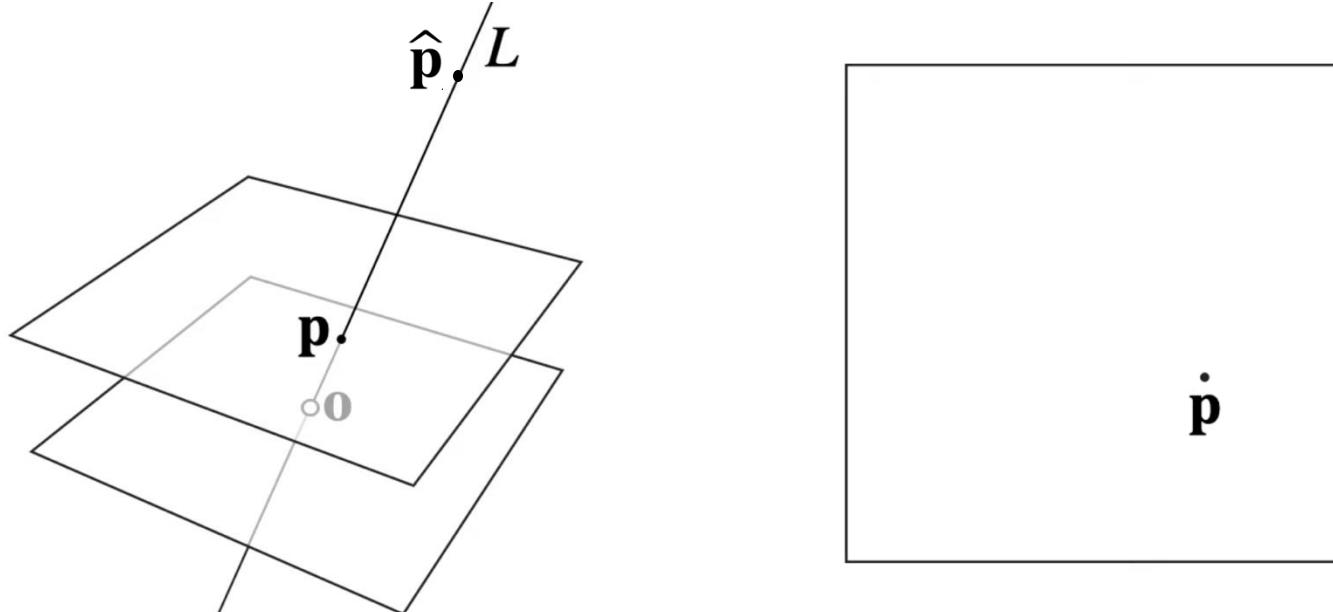
齐次坐标 – 基本理念

- 考虑任何不通过 3D 原点 o 的 2D 平面
- 通过 3D 原点的线将穿过 2D 平面中的一个点
 - 找到直线 L 穿过平面的点 p



齐次坐标 – 基本理念

- 考虑任何不通过 3D 原点 o 的 2D 平面
- 通过 3D 原点的线将穿过 2D 平面中的一个点
 - 找到直线 L 穿过平面的点 p



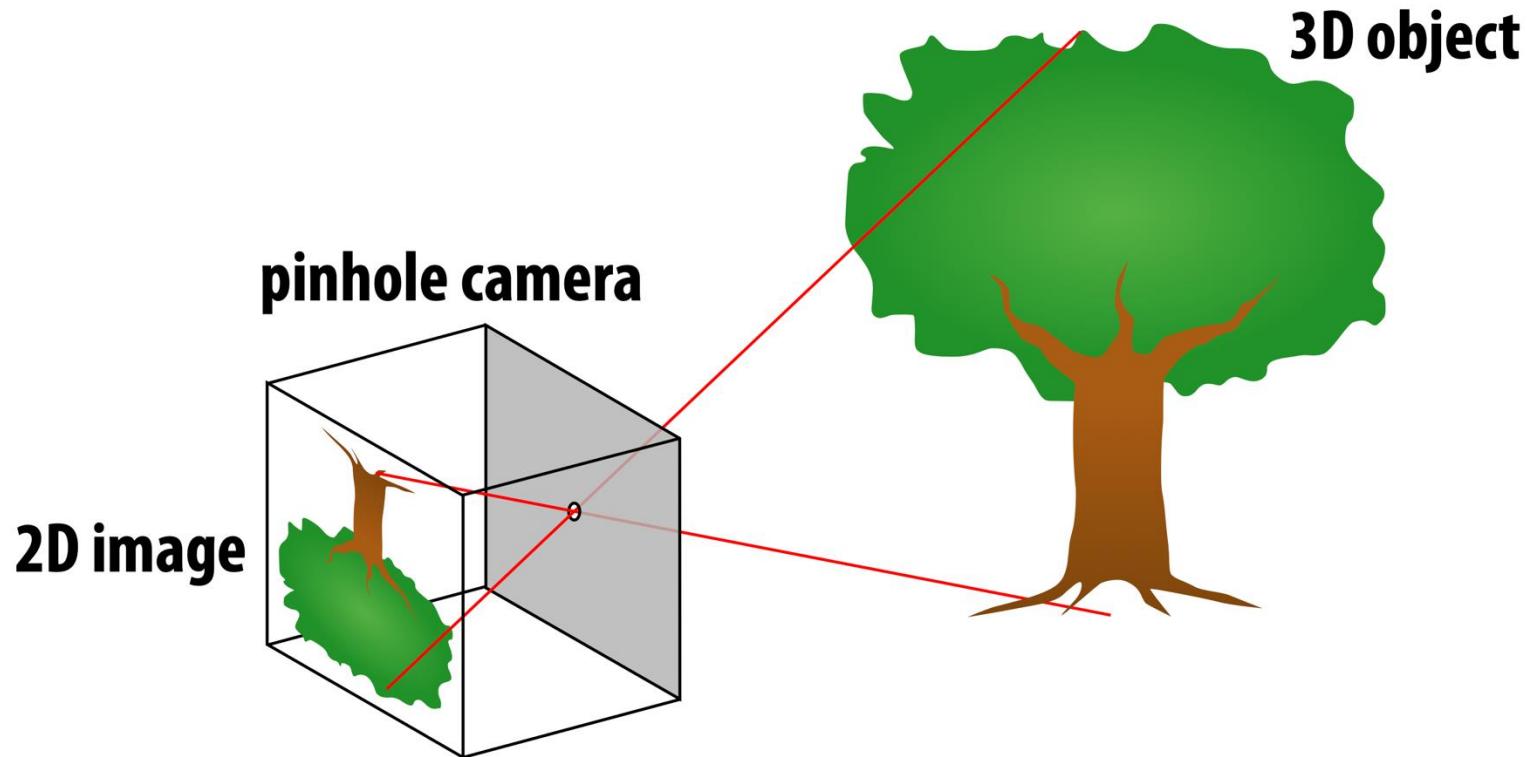
- L 上的任何点 \hat{p} 都可以用来表示点 p

Q: 这个做法有没有让你回想起什么?

回顾：透视投影

□希望有让你回忆起我们的“针孔相机”

□在同一条线的物体都将投影到同一点



图像上的点是不能知道它的具体位置的，只能知道它属于哪一条线

齐次坐标 (2D)

□ 更直接地，考虑 3D 中的点 $p = (x, y)$

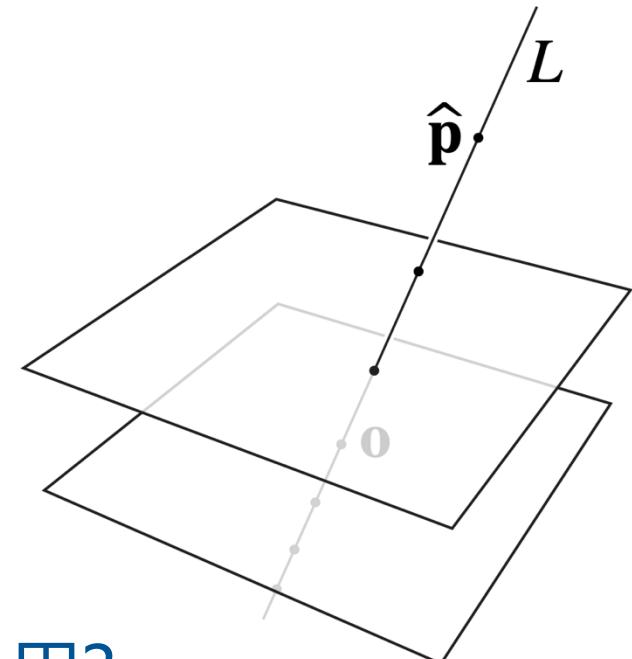
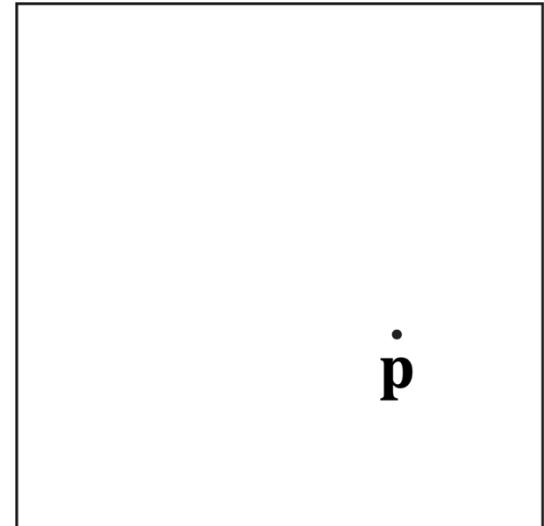
和平面 $z = 1$

□ 任意三个数 $\hat{p} = (a, b, c)$ 使得
 $(a/c, b/c) = (x, y)$ 即为 p 点
的齐次坐标

- 比如 $(x, y, 1)$
- 更一般的, (cx, cy, c) for $c \neq 0$

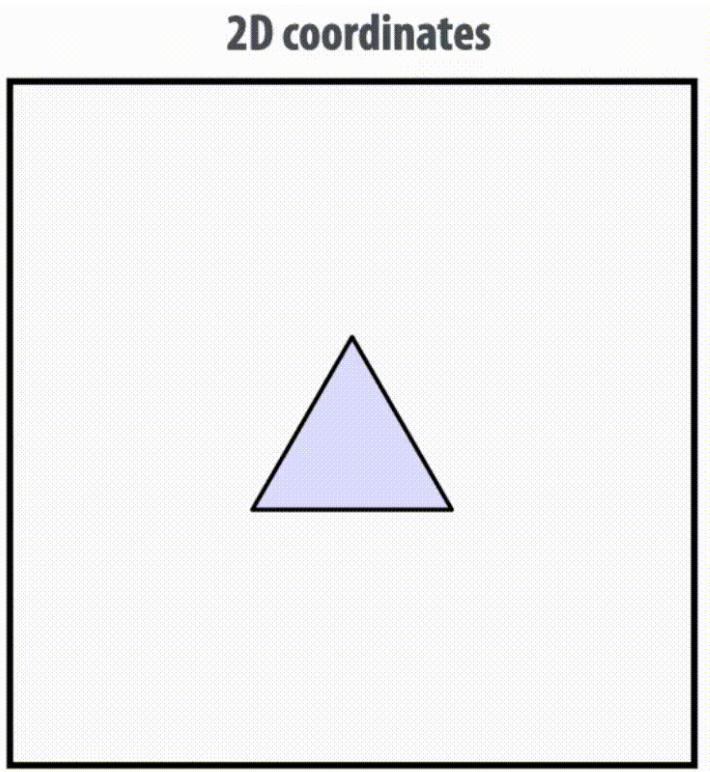
□ Hence, two points $\hat{p}, \hat{q} \in \mathbb{R}^3 \setminus \{O\}$
describe the same point in 2D
(and line in 3D) if $\hat{p} = \lambda \hat{q}$ for
some $\lambda \neq 0$

Great, 但是这个对我们的变换有什么用?



齐次坐标中的平移

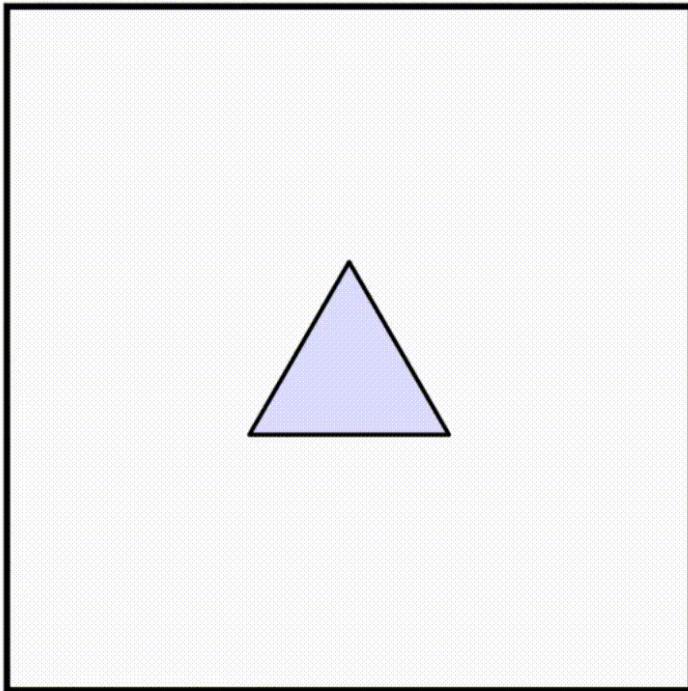
口让我们思考一下，如果我们对二维坐标 p 进行平移，齐次坐标 \hat{p} 会发生什么



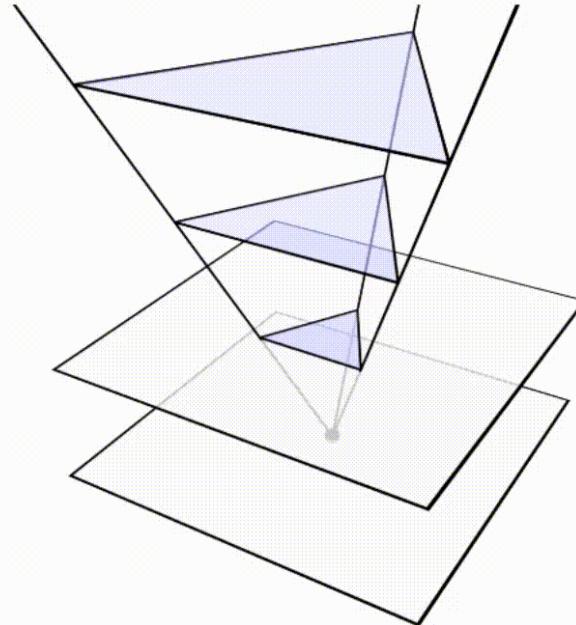
齐次坐标中的平移

口让我们思考一下，如果我们对二维坐标 p 进行平移，齐次坐标 \hat{p} 会发生什么

2D coordinates



homogeneous coordinates

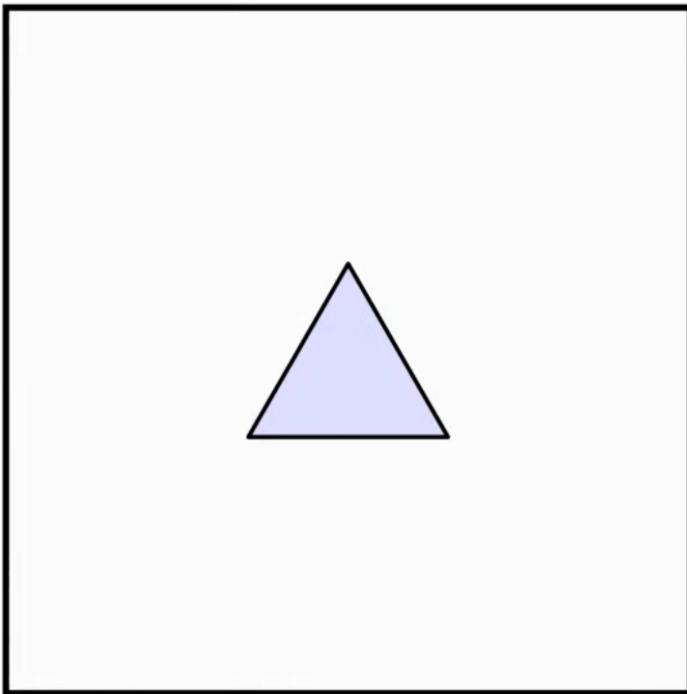


这个看起来像哪种变换？

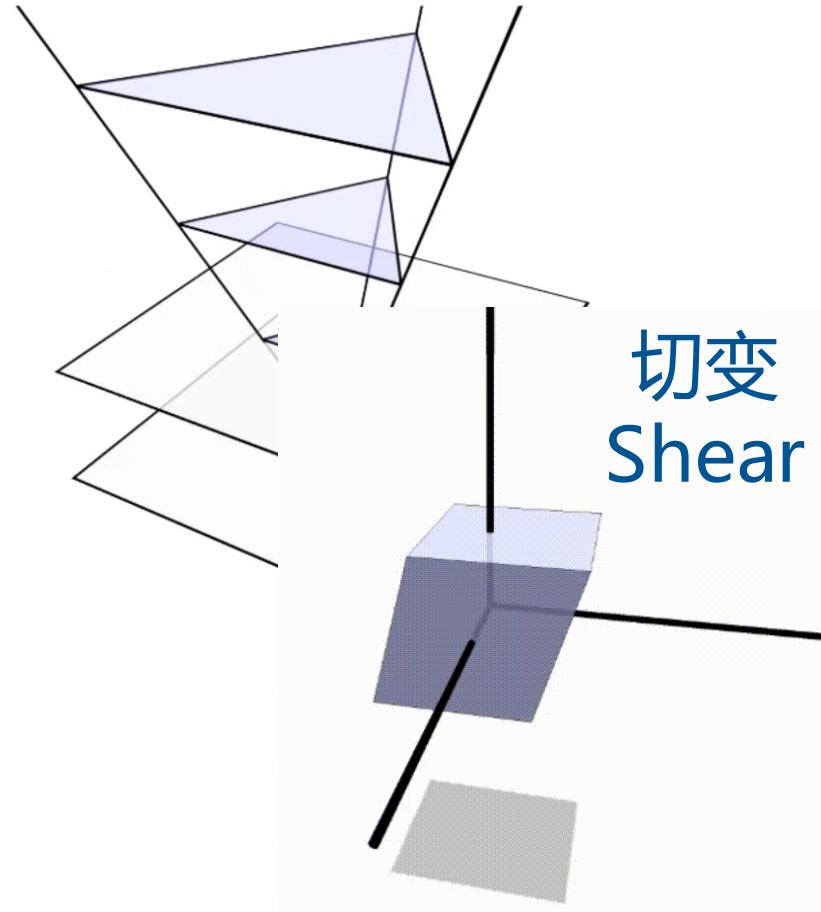
齐次坐标中的平移

口让我们思考一下，如果我们对二维坐标 p 进行平移，齐次坐标 \hat{p} 会发生什么

2D coordinates



homogeneous coordinates



这个看起来像哪种变换？

齐次坐标中的平移

- 但是，切变 shear 是一个线性变换！
- 这正确吗？让我们检查一下坐标
- 假设我们用向量 $\mathbf{u} = (u_1, u_2)$ 平移点 $\mathbf{p} = (p_1, p_2)$ ，得到 $\mathbf{p}' = (p_1 + u_1, p_2 + u_2)$
- 齐次坐标 $\hat{\mathbf{p}} = (cp_1, cp_2, c)$ 变成 $\hat{\mathbf{p}}' = (cp_1 + cu_1, cp_2 + cu_2, c)$
- 注意我们将 $\hat{\mathbf{p}}$ 移动了一个量 $c\mathbf{u}$ ，这个量与沿着第三轴的距离 c 成正比，此操作即为切变

Using homogeneous coordinates, we can represent an affine transformation in 2D as a linear transformation in 3D

齐次坐标 – 矩阵表示

□写成矩阵的形式，切变沿着方向 $\mathbf{u} = (u_1, u_2)$ 移动 \mathbf{x} ，移动的大小由 \mathbf{x} 沿着固定向量 \mathbf{v} 的距离决定：

$$f_{\mathbf{u}, \mathbf{v}}(\mathbf{x}) = \mathbf{x} + \langle \mathbf{v}, \mathbf{x} \rangle \mathbf{u}$$

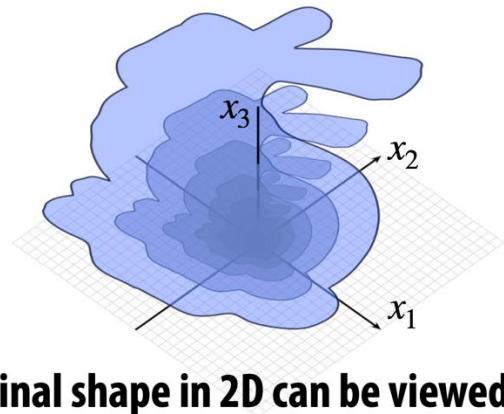
□矩阵表示为

$$f_{\mathbf{u}, \mathbf{v}}(\mathbf{x}) = (I + \mathbf{u}\mathbf{v}^T) \mathbf{x}$$

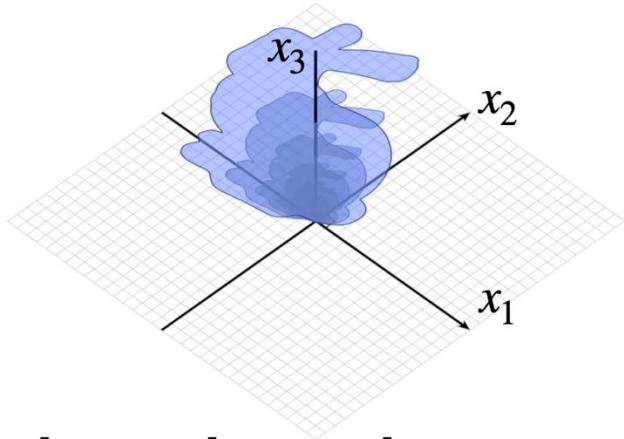
□在齐次坐标中， \mathbf{v} 总是 $(0, 0, 1)$ ，因此我们得到矩阵

$$\begin{bmatrix} 1 & 0 & u_1 \\ 0 & 1 & u_2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cp_1 \\ cp_2 \\ c \end{bmatrix} = \begin{bmatrix} c(p_1 + u_1) \\ c(p_2 + u_2) \\ c \end{bmatrix} \xrightarrow{1/c} \begin{bmatrix} p_1 + u_1 \\ p_2 + u_2 \\ 1 \end{bmatrix}$$

齐次坐标中的其他 2D 变换

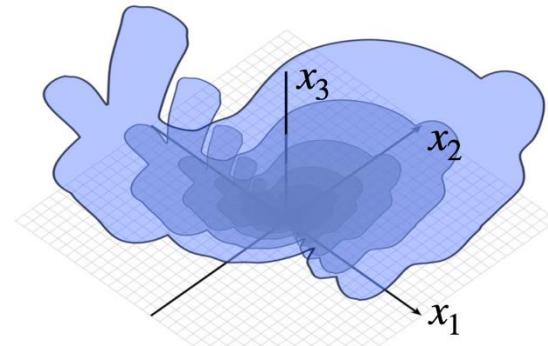


Original shape in 2D can be viewed as
many copies, uniformly scaled by x_3

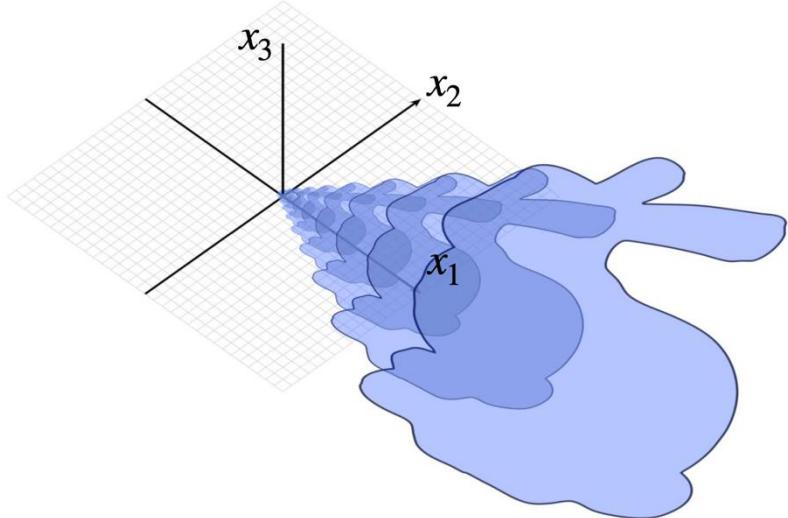


2D scale \leftrightarrow scale x_1 and x_2 ; preserve x_3

(Q: what happens to 2D shape if you
scale x_1 , x_2 , and x_3 uniformly?)



2D rotation \leftrightarrow rotate around x_3



2D translate \leftrightarrow shear

现在可以很容易地
组合这些变换了！

齐次坐标中的 3D 变换

- 3D (或更高维度) 的情况类似，只需附加一个齐次坐标轴
- 3D 线性变换的矩阵表示只增加一个额外的单位行/列 (identity row/column); 平移同样是切变

rotate (x, y, z) around y by θ

$$\begin{bmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

point in 3D

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

scale x, y, z
by a, b, c

$$\begin{bmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

shear (x, y) by z
in (s, t) direction

$$\begin{bmatrix} 1 & 0 & s & 0 \\ 0 & 1 & t & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

translate (x, y, z)
by (u, v, w)

$$\begin{bmatrix} 1 & 0 & 0 & u \\ 0 & 1 & 0 & v \\ 0 & 0 & 1 & w \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

点 Points vs. 向量 Vectors

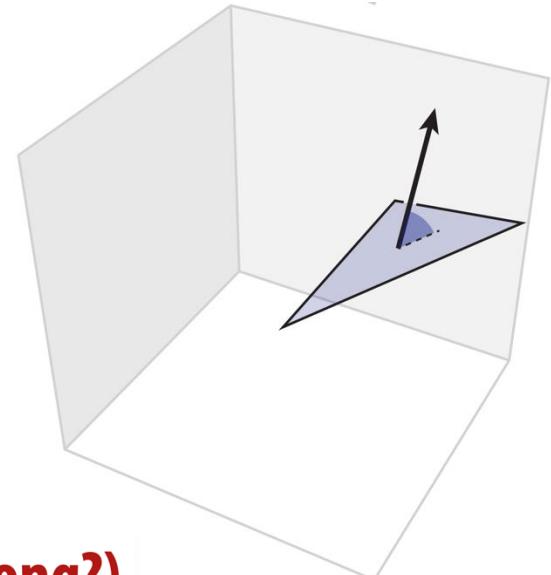
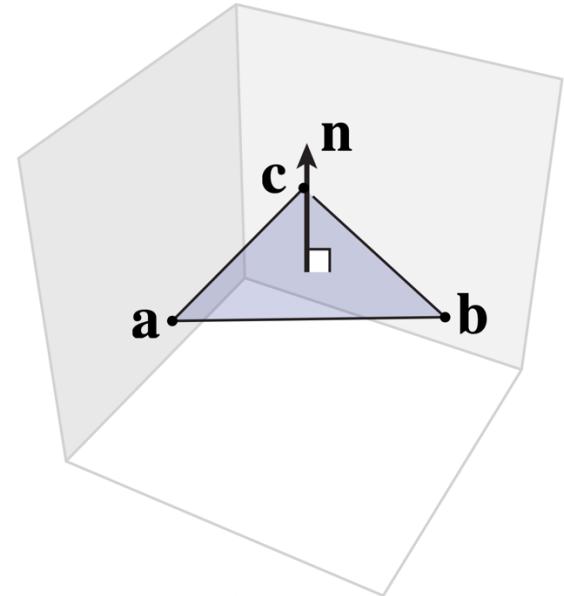
□ 齐次坐标有另一个有用的特征：帮助区分点和向量

□ 考虑一个三角形

- 点 Vertices $a, b, c \in \mathbb{R}^3$
- 法向量 Normal vector $n \in \mathbb{R}^3$

□ 我们应用如下变换：在点 a, b, c, n 后附加一个 1，然后分别乘以下面的矩阵

$$\left[\begin{array}{ccc|c} \cos \theta & 0 & \sin \theta & u \\ 0 & 1 & 0 & v \\ -\sin \theta & 0 & \cos \theta & w \\ 0 & 0 & 0 & 1 \end{array} \right] \xrightarrow{\hspace{1cm}} \text{rotation} \quad \text{translation}$$



Normal is not orthogonal to triangle! (What went wrong?)

点 Points vs. 向量 Vectors

口让我们想想当我们把法向量 \mathbf{n} 乘以矩阵时会发生什么

rotate normal around y by θ

$$\begin{bmatrix} \cos \theta & 0 & \sin \theta \\ 0 & 1 & 0 \\ -\sin \theta & 0 & \cos \theta \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \\ 1 \end{bmatrix} \begin{bmatrix} n_1 \\ n_2 \\ n_3 \\ 1 \end{bmatrix}$$

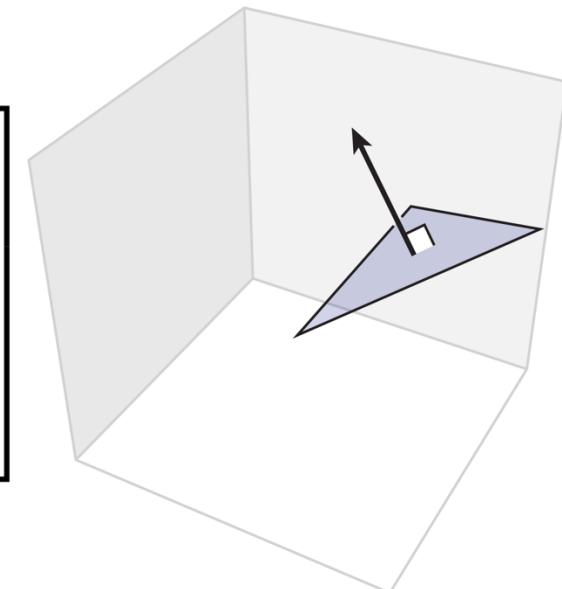
translate normal by
(u, v, w)

口但是当我们旋转/平移三角形时，
它的法线应该只是旋转*！

口解决方案：将齐次坐标设置为零！

口平移现在被忽略了；
法线与三角形正交

$$\begin{bmatrix} n_1 \\ n_2 \\ n_3 \\ 0 \end{bmatrix}$$



*回想一下，向量只有方向和大小，而没有“基点”

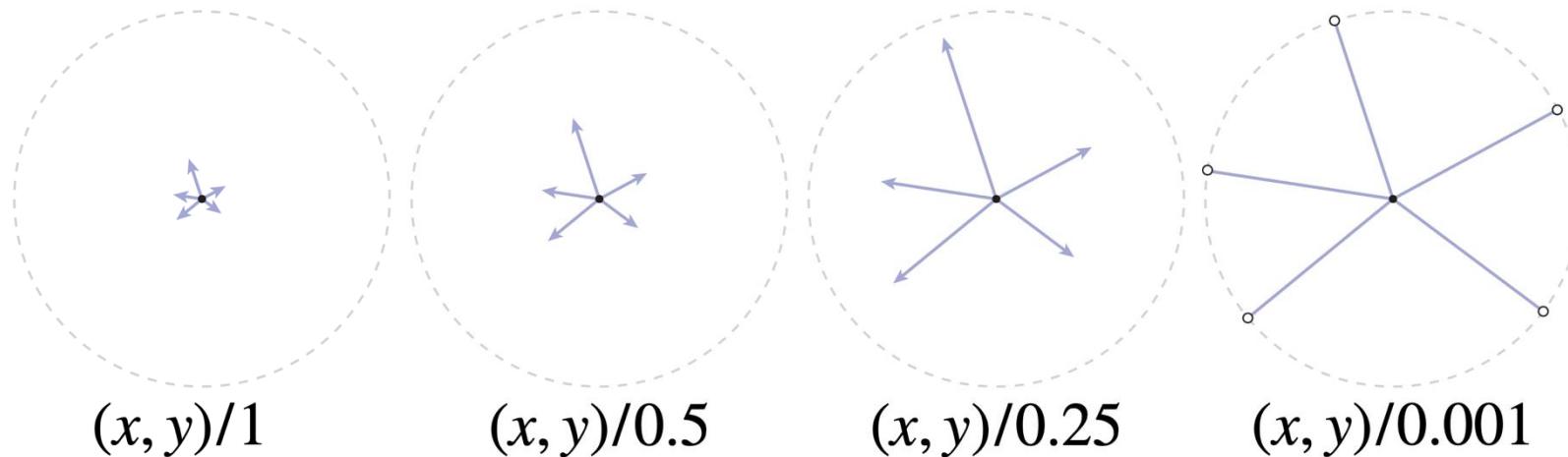
齐次坐标中的点 vs. 向量

□一般而言

- 点具有非零齐次坐标 ($c = 1$)
- 向量的齐次坐标为零 ($c = 0$)

□当时 $c = 0$ 时，除以它意味着什么？

□我们可以逐步考虑 $c \rightarrow 0$



Can think of vectors as “points at infinity” (sometimes called “ideal points”)

(In practice: still need to check for divide by zero!)

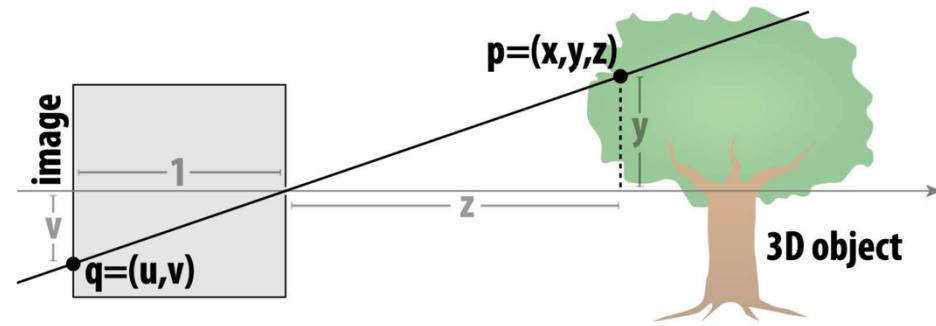
齐次坐标中的透视投影

口如何使用齐次坐标进行
透视投影*?

口我们的针孔相机模型的
基本思想是“除以 z ”

口因此，我们可以构造一个
矩阵，将 z 坐标“复制”到
齐次坐标中

口通过齐次坐标的除法，我们
现在可以得到在 $z = 1$ 平面
上的透视投影



$$(x, y, z) \mapsto (x/z, y/z)$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ z \end{bmatrix}$$

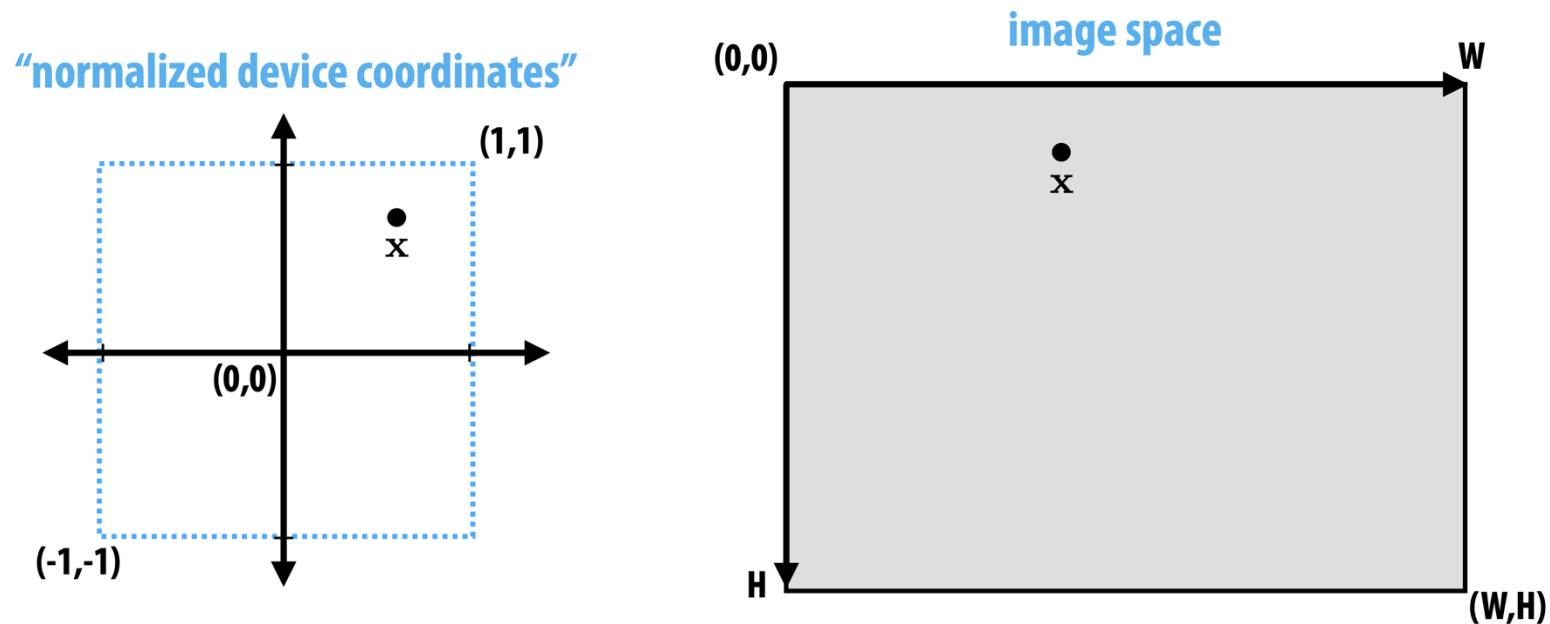
$$\Rightarrow \begin{bmatrix} x/z \\ y/z \\ 1 \end{bmatrix}$$

*假设针孔相机在 $(0, 0, 0)$ 处向负方向观察 z 轴

屏幕变换 Screen transformation

口光栅化流程中需要的最后一个变换：从观察平面到像素坐标的变换 (viewing plane -> pixel coordinates)

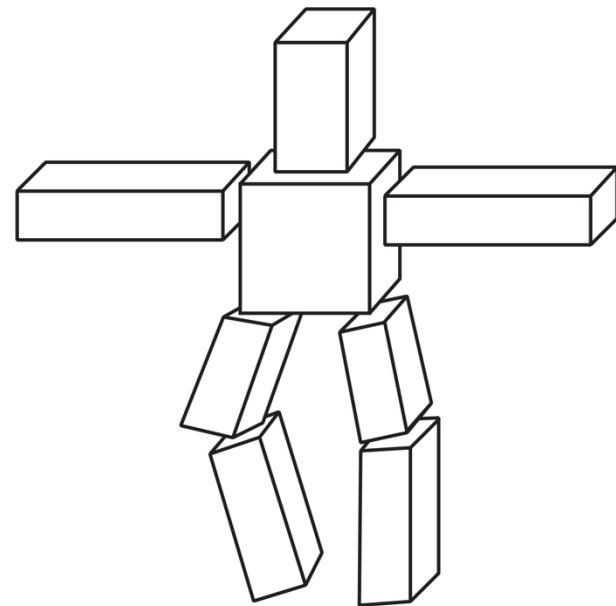
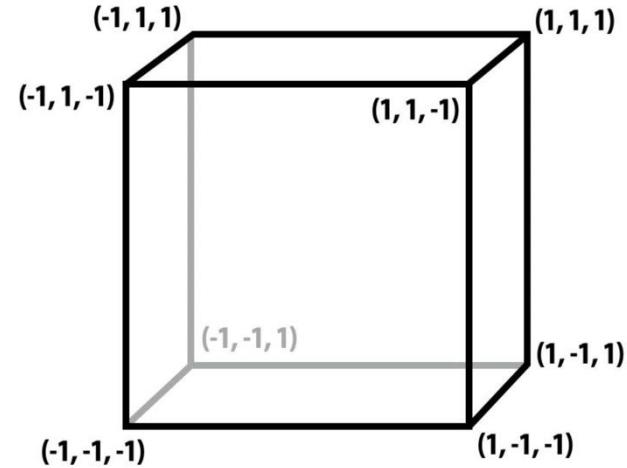
口假设我们想将 $z = 1$ 平面上的正方形 $[0, 1] \times [0, 1]$ 内的所有点绘制成 $W \times H$ 像素的图像



Q: What transformation(s) would you apply? (Careful: y is now down!)

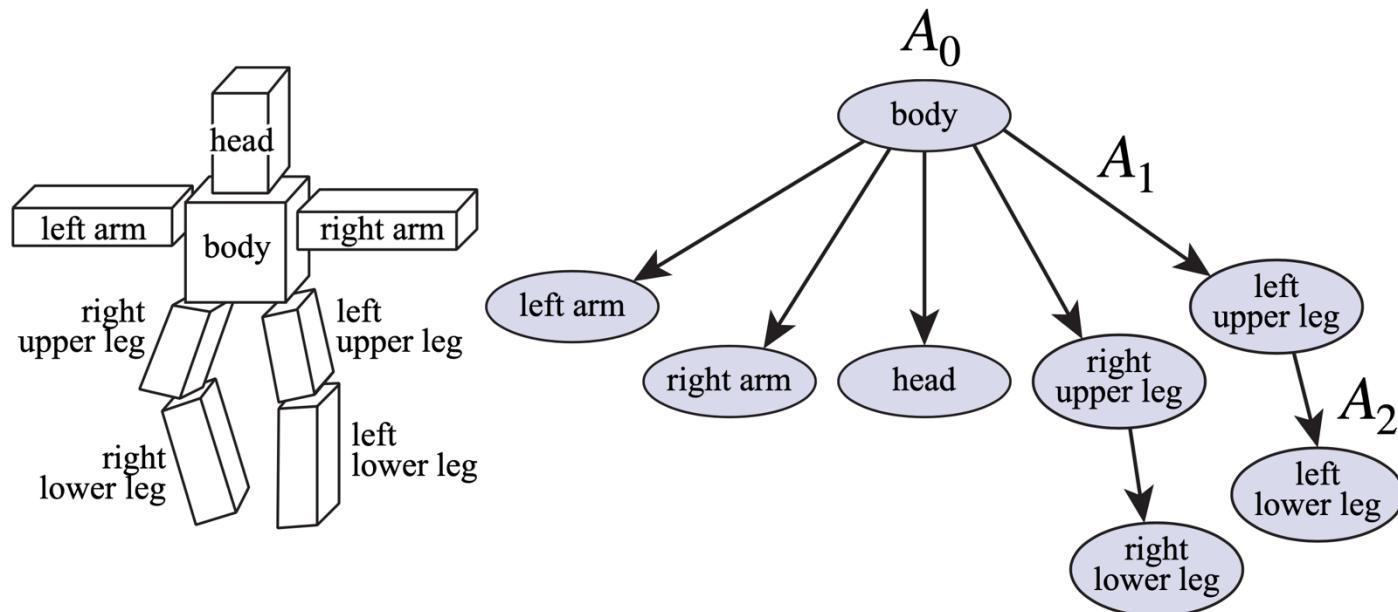
场景图 Scene graph

- 对于一些复杂的场景 (比如多个立方体), 场景图能帮助管理不同的变换
- 假设我们像通过变换一个单位立方体来创造一个立方体生物
- 很难直接指定具体的变换
- 相反地, 我们可以利用底层的变换来进一步构建上层的变换
 - 比如说, 先指定身体
 - 然后变换身体以构建上肢
 - 接着变换上肢以构建下肢
 - ...



场景图

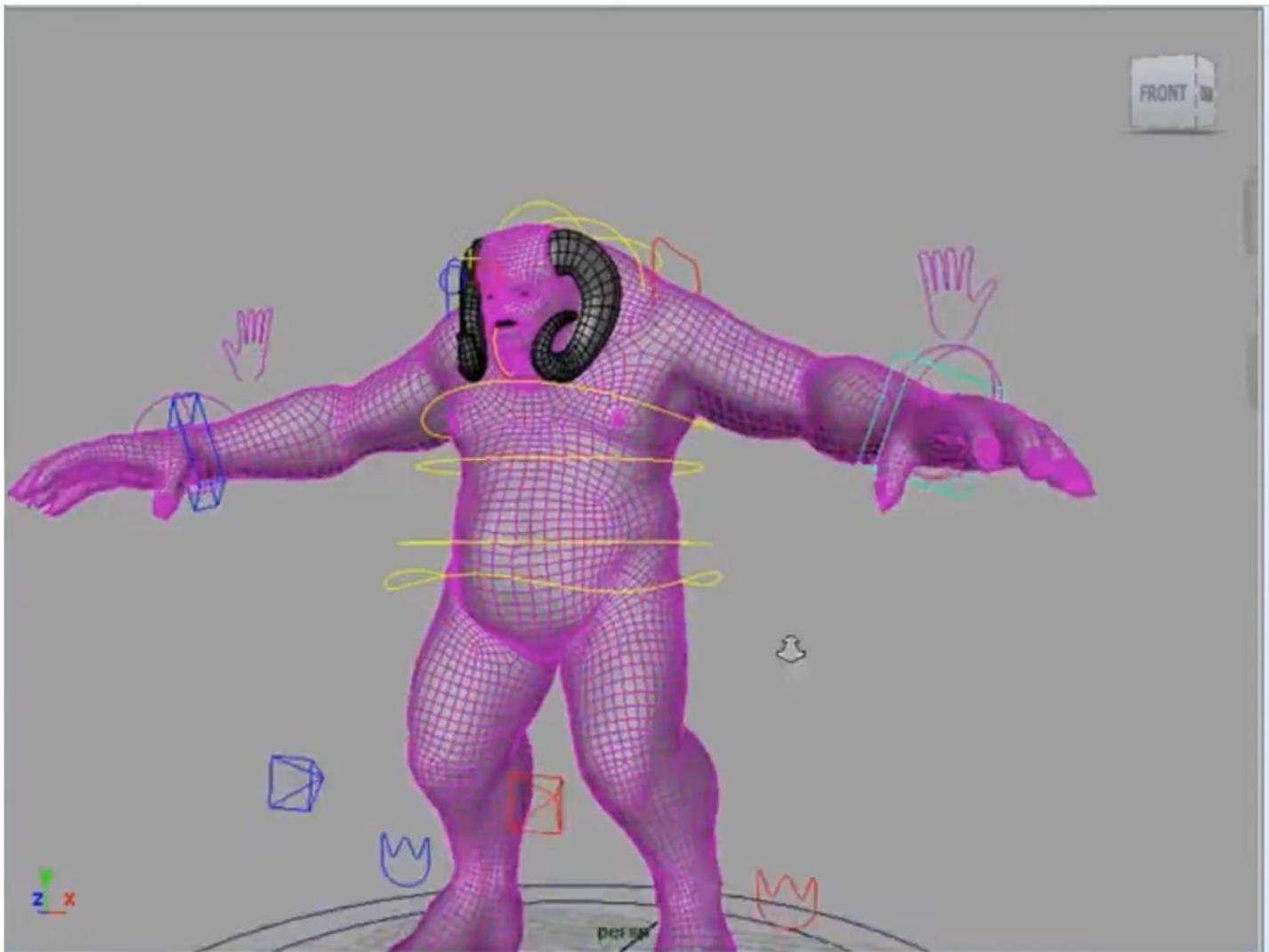
- 场景图在有向图中存储相对变换 (relative transformation)
- 每条边 (加上根) 存储一个线性变换 (例如, 4×4 矩阵)
- 变换的组合应用于节点



- 比如, A_1A_0 产生左大腿, $A_2A_1A_0$ 产生左小腿
- 用堆栈存储变换, 以减少冗余乘法

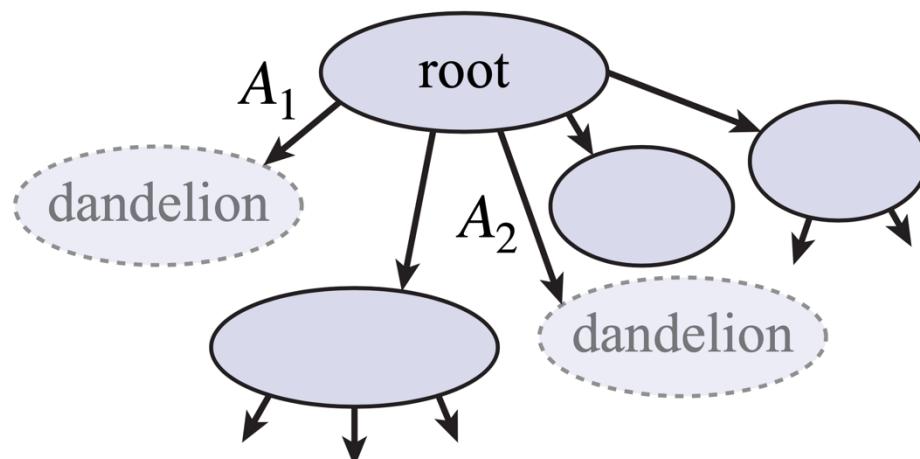
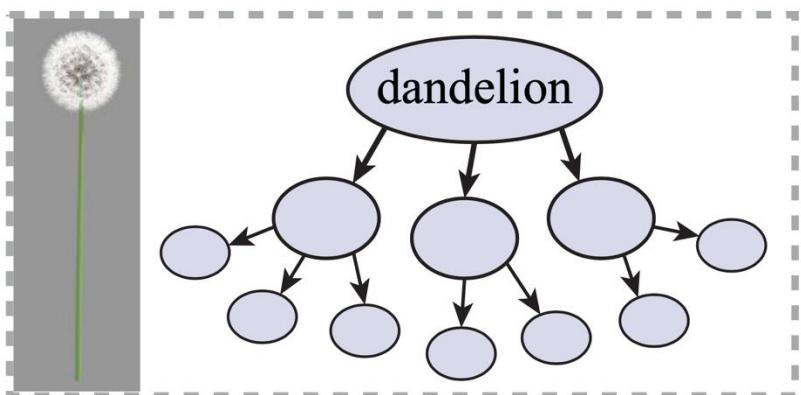
场景图 – 例子

口通常用于构建复杂的“操控”，不同部位相互影响



实例化渲染

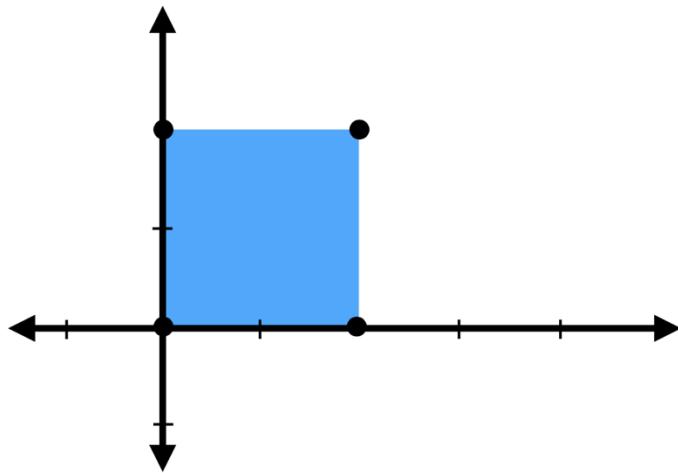
- 如果我们想要场景中同一对象的多个副本，该怎么办？
- 与其存储多个几何体的副本，不如在场景图中放置一个“指针”节点
- 与任何其他节点一样，可以在每个传入边上指定不同的变换



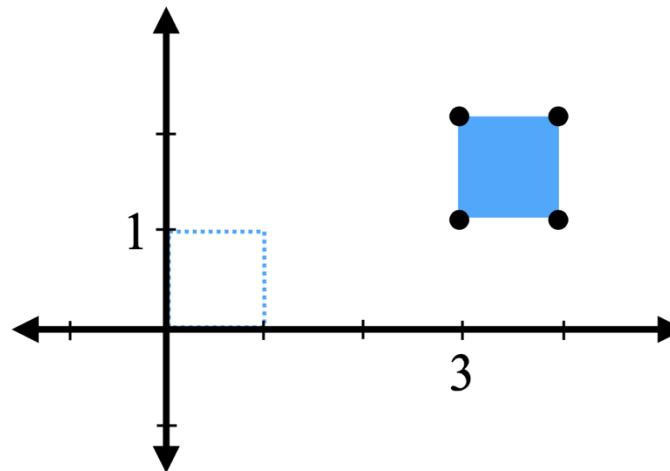
实例化渲染



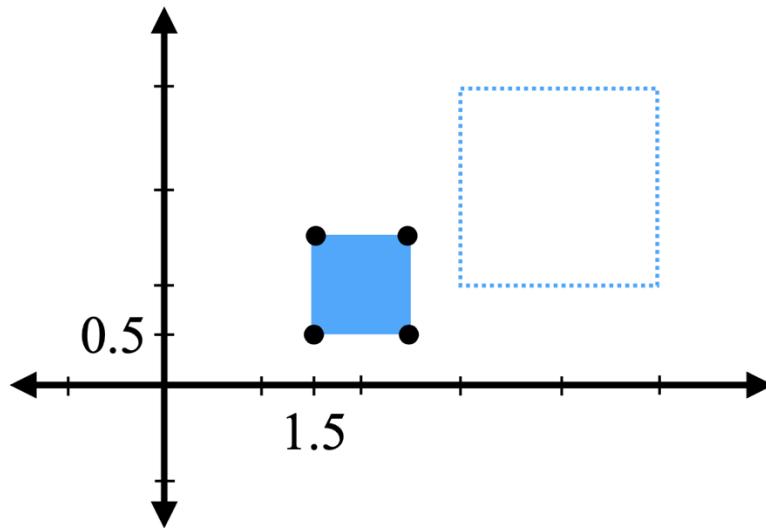
组成变换时顺序很重要



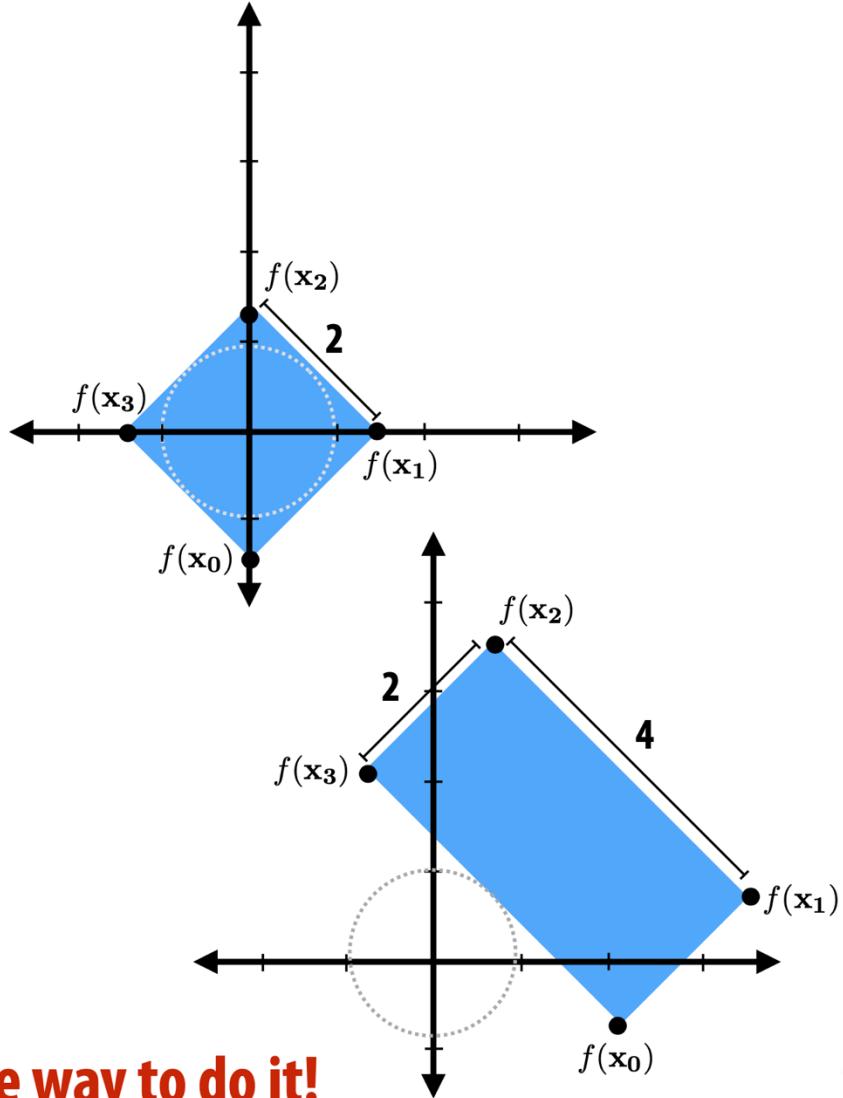
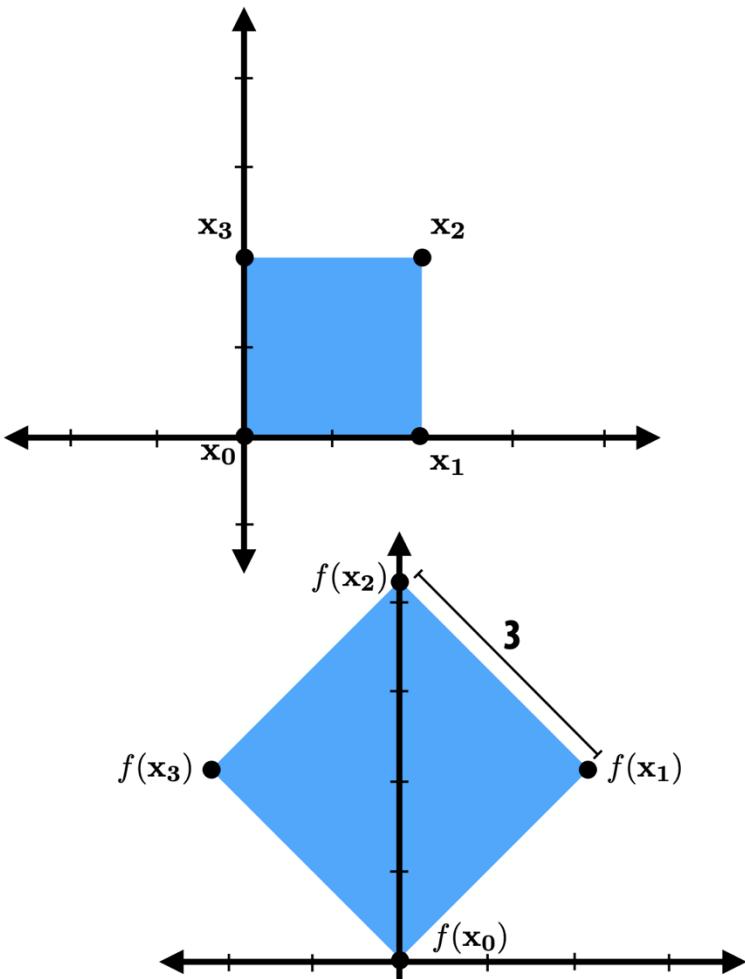
scale by 1/2, then translate by (3,1)



translate by (3,1), then scale by 1/2

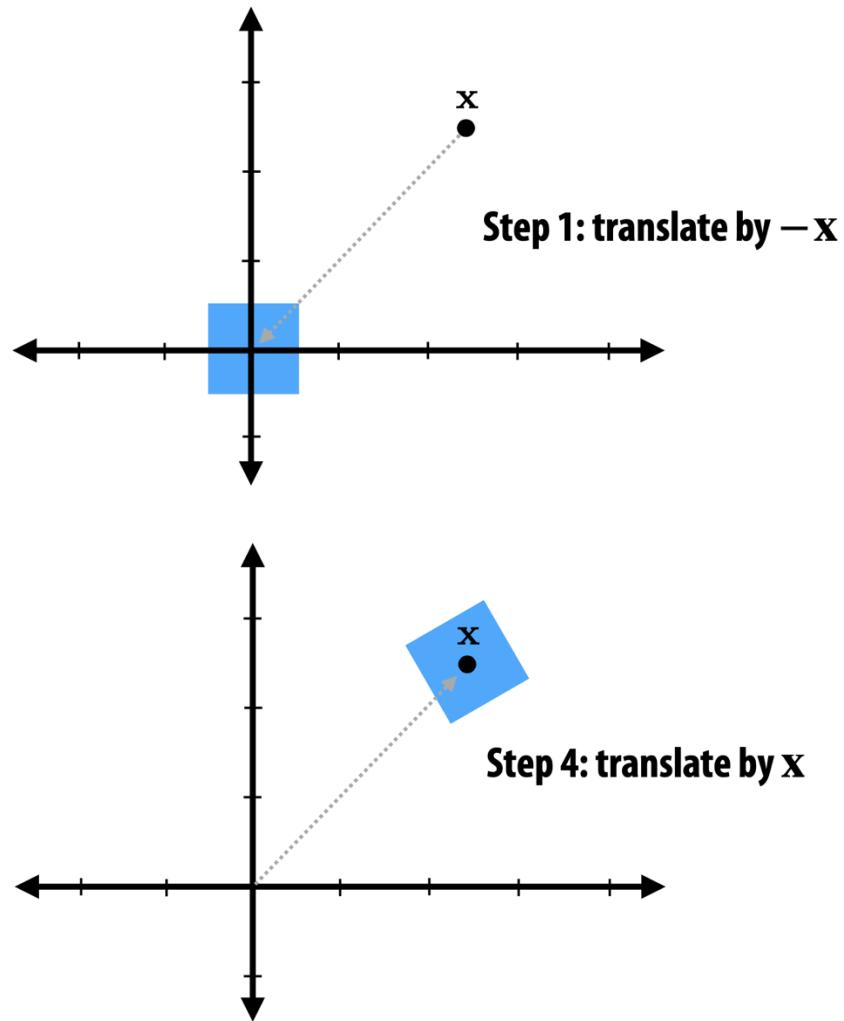
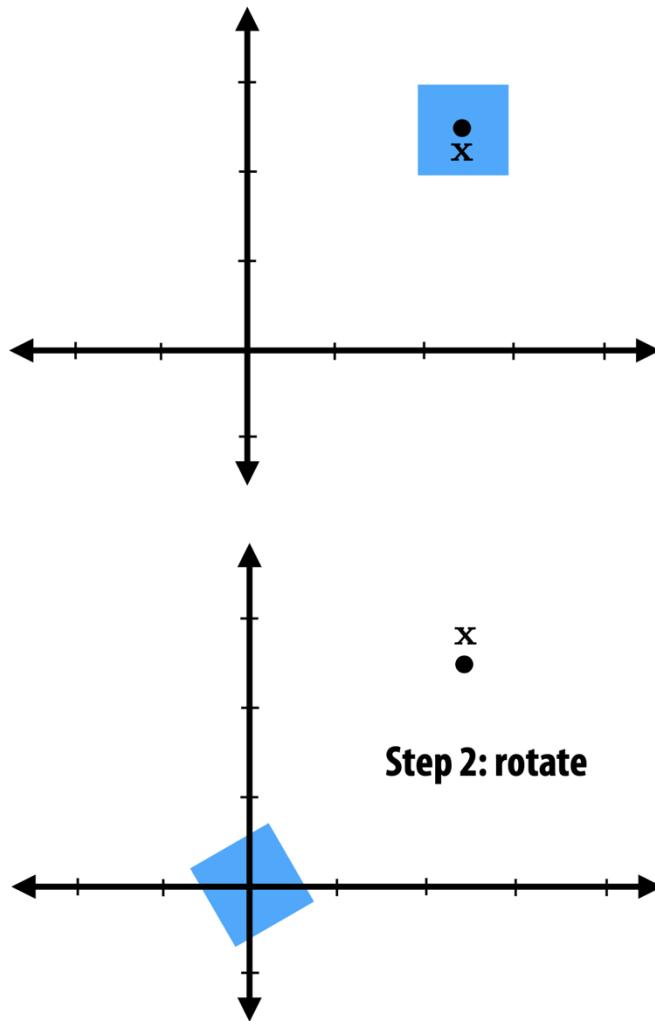


如何组成下面的变换?



Remember: always more than one way to do it!

基于某一点做旋转



Q: What happens if we just rotate without translating first?

画一个立方体生物

□ 从我们的 3D 立方体开始，我们想制作一个“立方体生物”的 2D、透视正确的图像

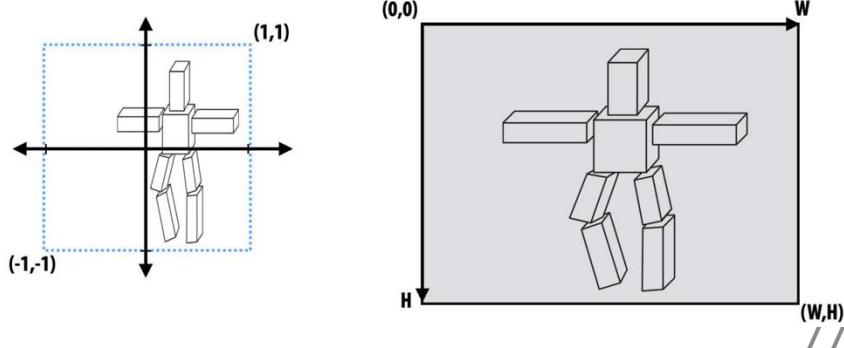
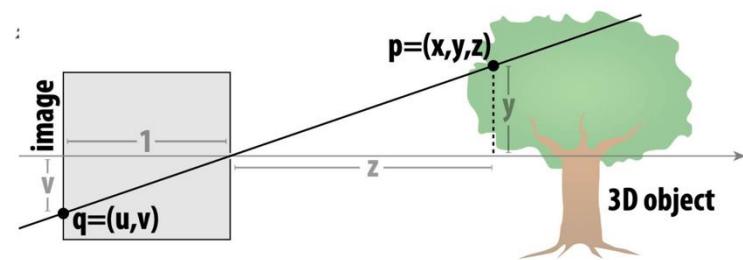
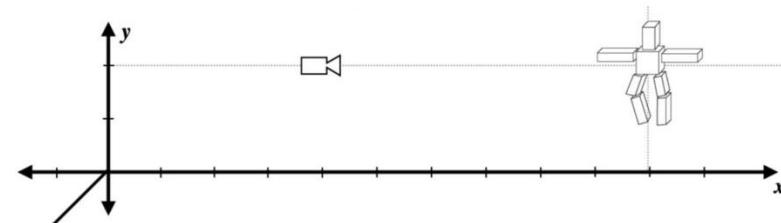
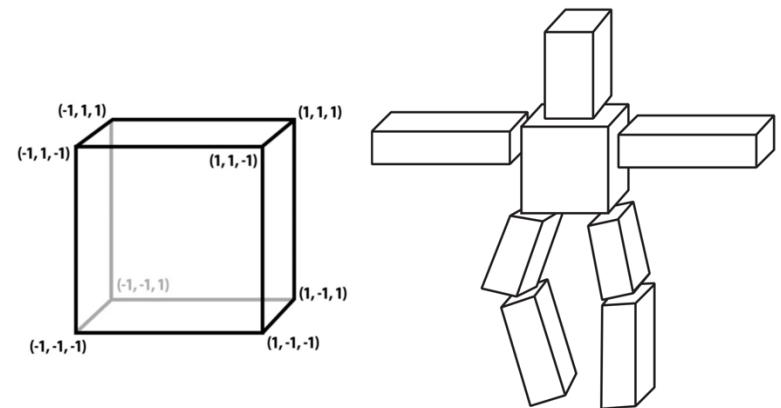
□ 首先，我们使用场景图将 3D 变换应用于立方体的几个副本

□ 然后我们应用 3D 变换来定位我们的相机

□ 然后是透视投影

□ 最后，我们转换为图像坐标并光栅化

□ Easy, right?





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