

专业限选系列课程

智能机器人技术

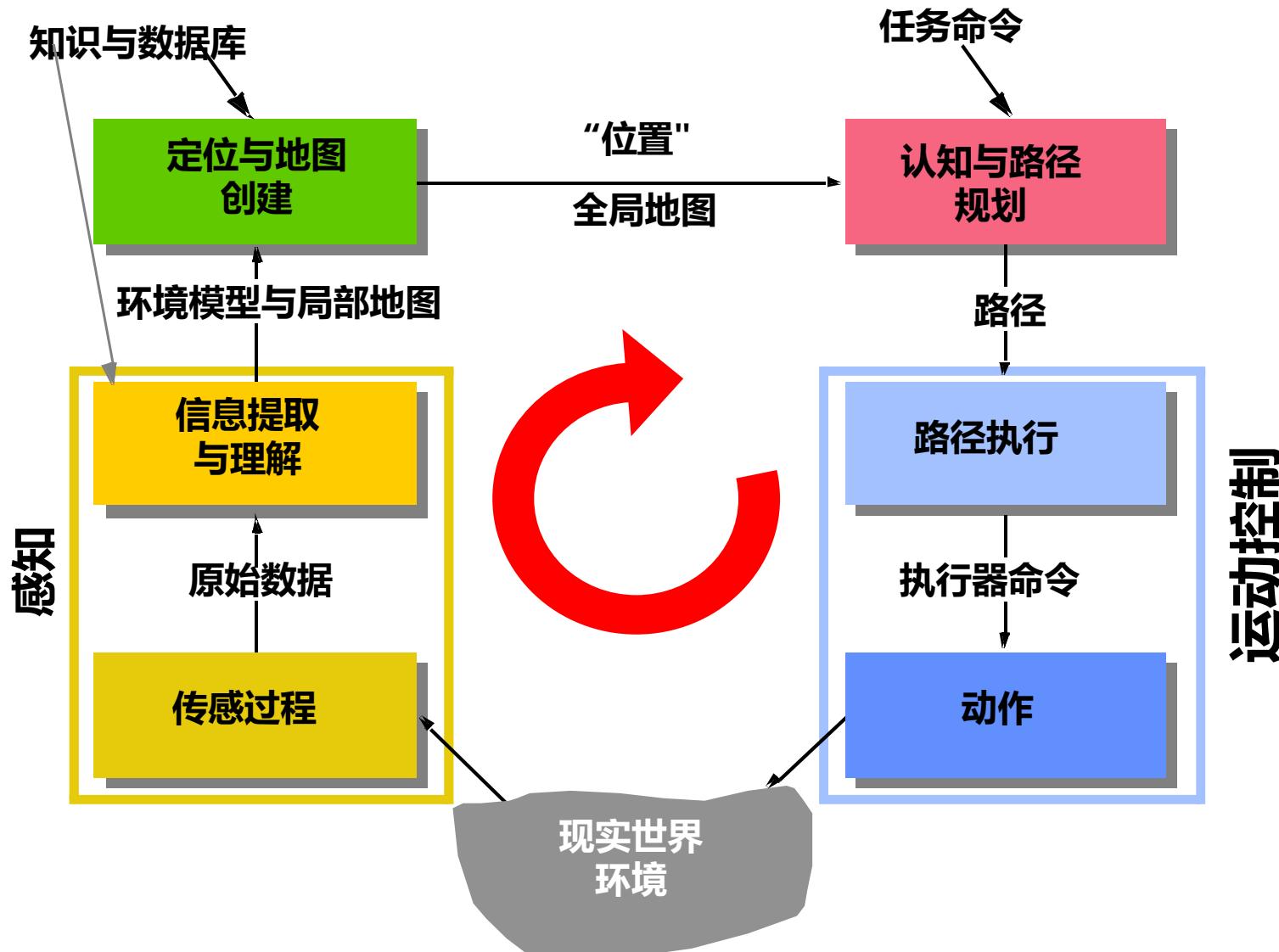
---Introduction to Intelligent Robotics

赵振刚 gavin@ustc.edu.cn

Review

1. What is an Intelligent Robot ?
2. Intelligent Robot in USTC
3. Typical technology in Intelligent Robot
4. Development trend under AI
5. Syllabus

课程概要



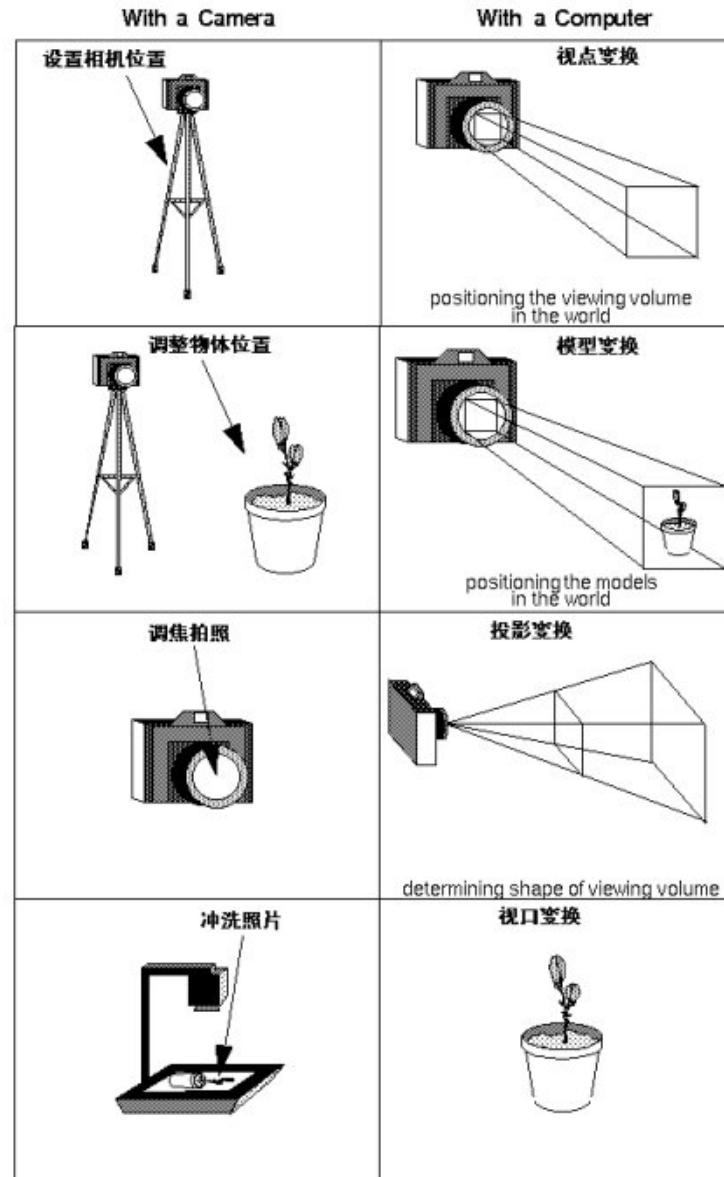
本章概要

1. 简述机器人的起源与发展，智能机器人研究现状，人工智能背景下的智能机器人典型技术和应用场景，给出课程大纲；
2. 讨论机器人“移动”的数学描述，包括空间任意点的位置和姿态变换、齐次坐标变换、四元数、物体的变换和逆变换等；

Outline

1. 空间坐标定义
2. 齐次坐标系
3. 空间变换矩阵
4. 旋转向量
5. 欧拉角
6. 四元数
7. 矩阵运算开源库 Eigen

移动的基础是相对坐标



1. 空间坐标定义

- ❖ 笛卡尔坐标系

在数学里，笛卡儿坐标系（Cartesian坐标系），也称直角坐标系，是一种正交坐标系。

- ❖ 左手坐标系与右手坐标系

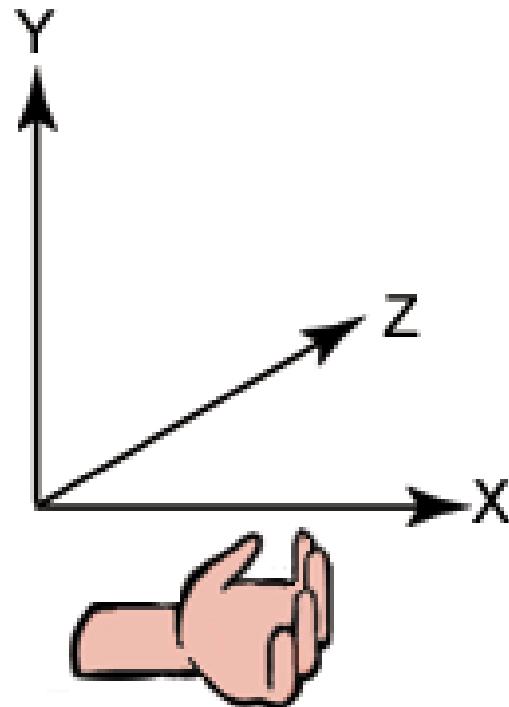
任何两个2D笛卡尔坐标都可以通过平移和旋转使两个坐标系重合

但是3D笛卡尔坐标就不一定了，实际上存在两种完全不同的3D笛卡尔坐标系，左手系和右手系。

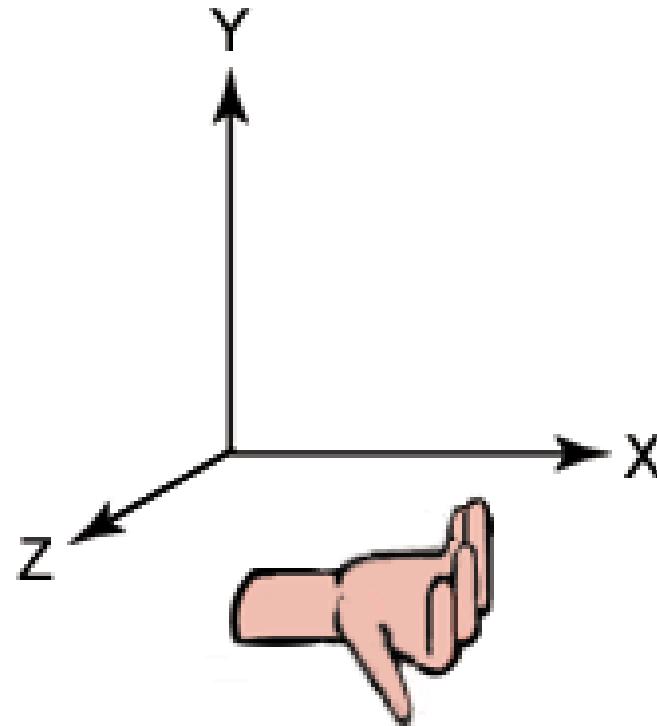
1. 空间坐标定义

- ❖ 左手坐标系与右手坐标系

Left-handed
Cartesian Coordinates



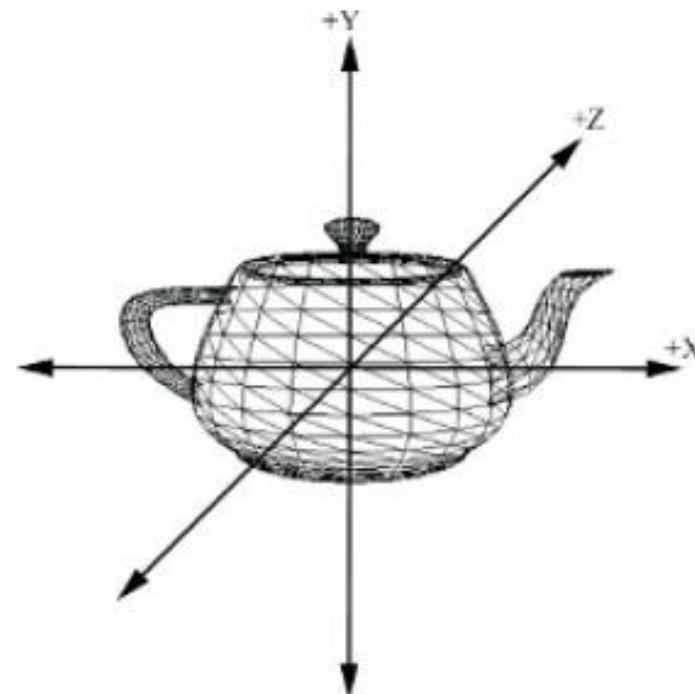
Right-handed
Cartesian Coordinates



1. 空间坐标定义

❖ 模型坐标系（物体坐标系）

- 和特定物体相关联的坐标系
- 每个物体都有它们独立的坐标系。
当物体移动或者改变方向时，和该物体相关联的坐标系就随之移动或者改变方向。
- “前”“后”“左”“右”只有在物体坐标系中才有意义
- 物体坐标系也能像指定方向一样指定位置



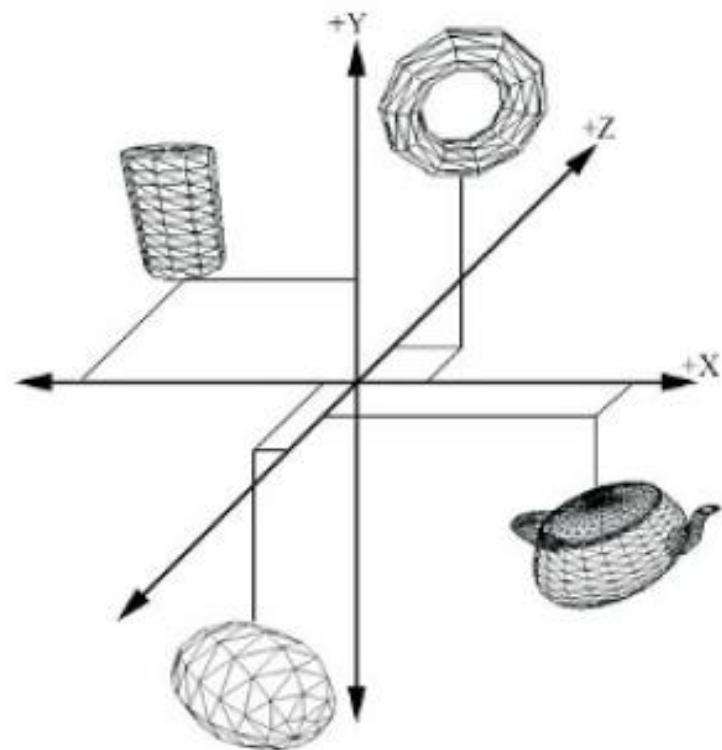
1. 空间坐标定义

❖ 世界坐标系（全局坐标系 / 宇宙坐标系）

建立了描述其他坐标系所需要的参考框架。从非技术意义上讲，它是我们关心的最大坐标系，所以世界坐标系不必是整个世界

世界坐标的典型问题：

- 每个物体的位置和方向
- 摄像机的位置和方向
- 世界中每一点的地形是什么
- 物体从那里来到那里去。



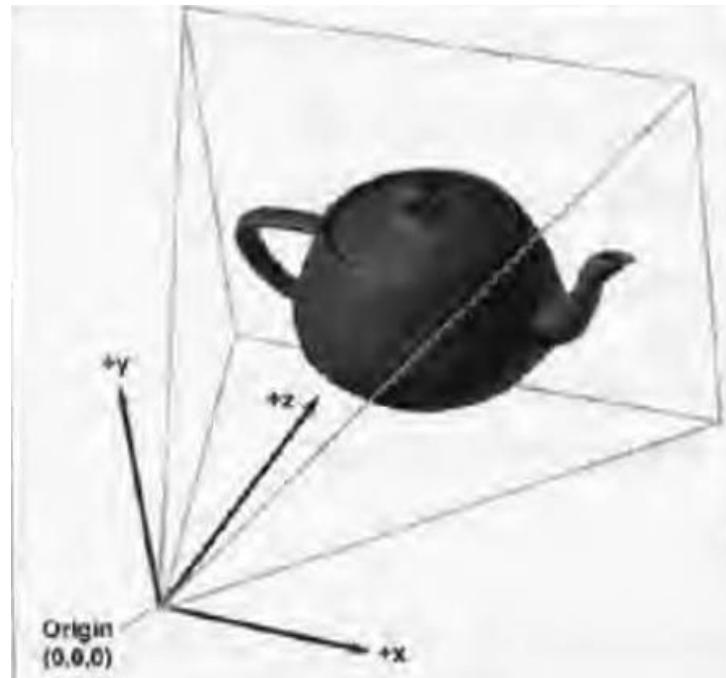
1. 空间坐标定义

◆ 摄像机坐标系

是和观察者密切相关的坐标系。该坐标系定义在摄像机的屏幕可视区域。

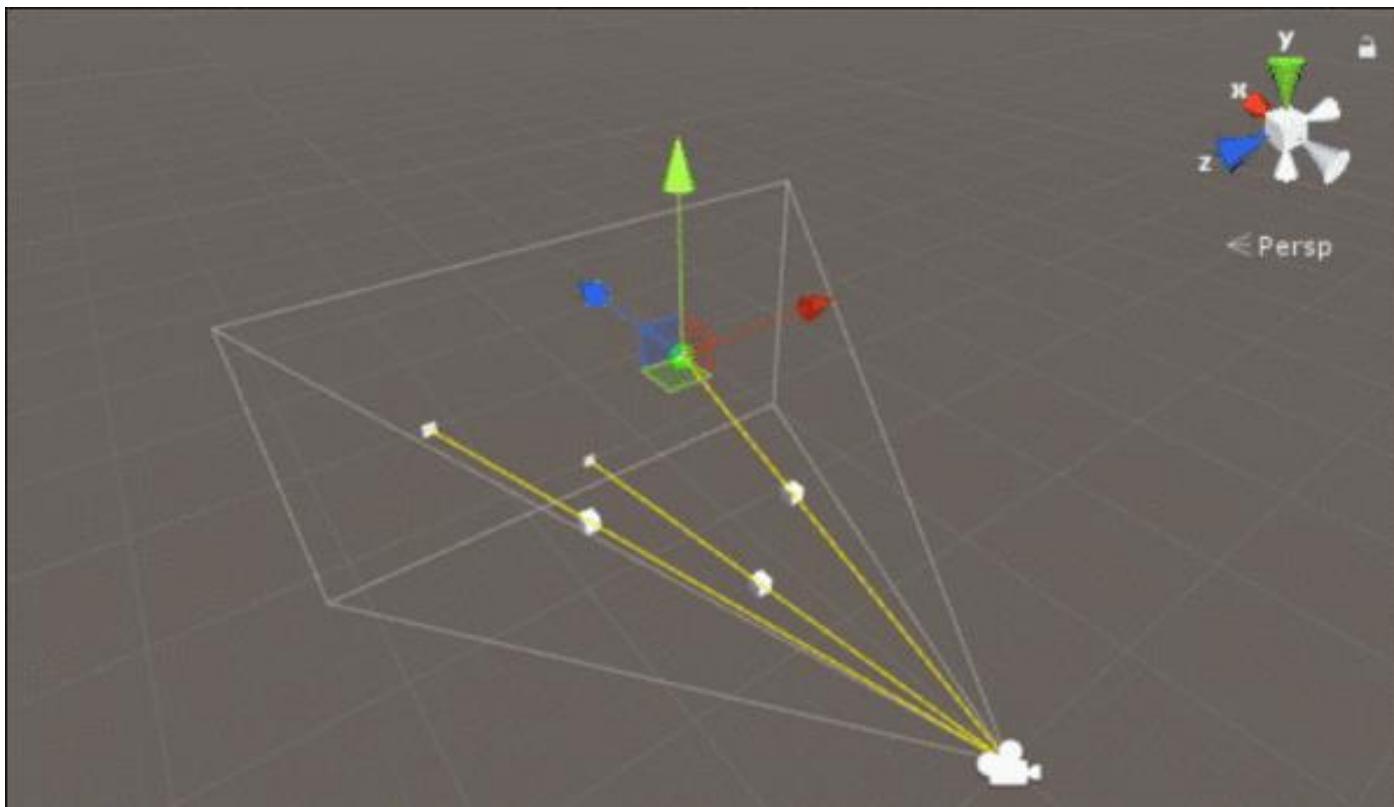
摄像坐标系的典型问题：

- 3D空间中给定点在摄像机前方吗？
- 3D空间中的给定点在视锥区域内还是在视锥区域外？
- 物体是完全在视锥内还是部分在视锥内？
- 两个物体，谁在前面？

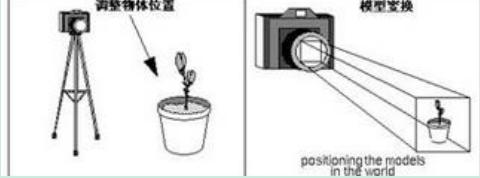
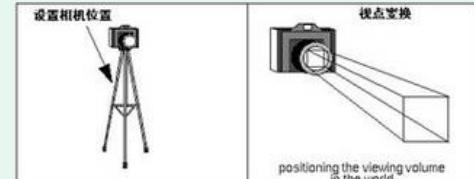
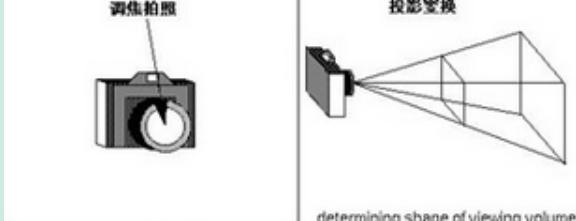
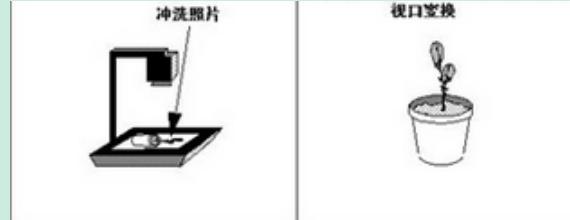


1. 空间坐标定义

- ❖ 设备坐标 (DC) 或屏幕坐标：显示设备的坐标系称为设备坐标，该坐标系依赖于具体的显示设备。

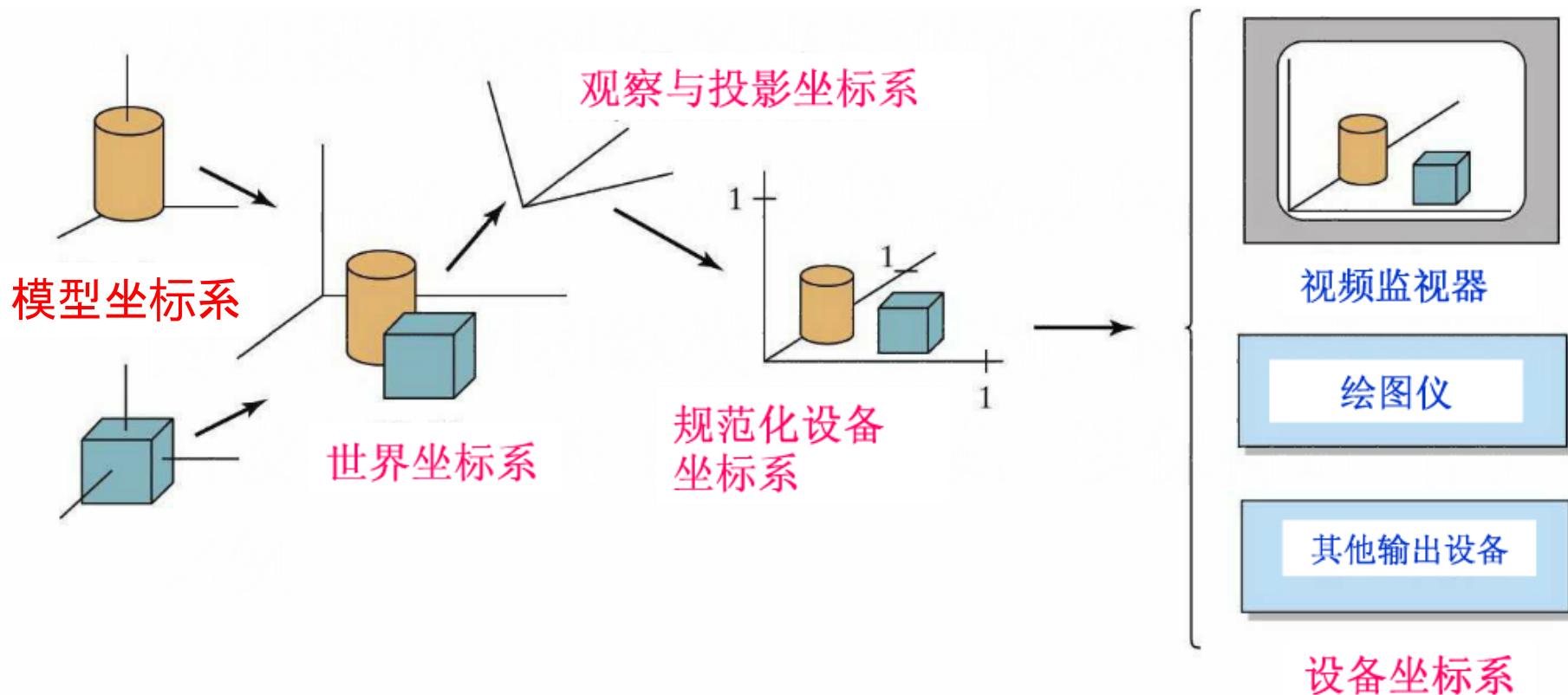


机器三维观察流水线

拍照流程	
在摄影棚中布置场景	
确定照相机位置	
调焦	
拍照	
冲洗照片/屏幕显示	

1. 空间坐标定义

❖ 机器三维观察流水线



1. 空间坐标定义

位置描述

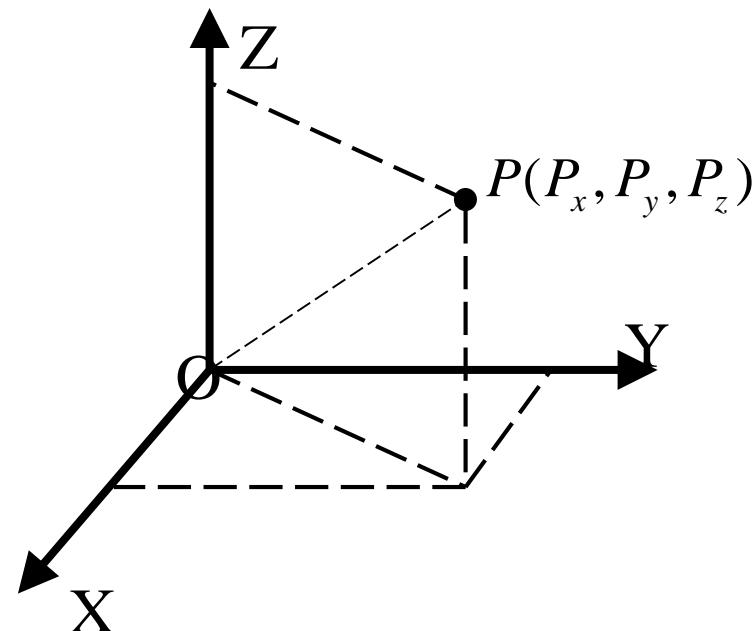
笛卡尔坐标系：

- 在选定的直角坐标系 $\{A\}$ 中，空间任一点P的位置可用位置矢量 ${}^A P$ 表示：

$${}^A p = p_x i + p_y j + p_z k$$

- 利用 3×1 矩阵表示：

$${}^A P = \begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix}$$



笛卡尔坐标系

Outline

1. 空间坐标定义
2. 齐次坐标系
3. 空间变换矩阵
4. 旋转向量
5. 欧拉角
6. 四元数
7. 矩阵运算开源库 Eigen

2. 齐次坐标系

齐次坐标 Homogeneous

在原有的坐标上增加一个维度：

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

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

新增的维度并不会增加自由度：

$$(x, y, z, w) \quad w \neq 0 \rightarrow (x / w, y / w, z / w)$$

2. 齐次坐标系

位置描述

三维空间点P的齐次坐标:加入一个比例因子w, 位置向量可以写为:

$$P = \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} = \begin{bmatrix} wP_x \\ wP_y \\ wP_z \\ w \end{bmatrix} = \begin{bmatrix} a \\ b \\ c \\ w \end{bmatrix}$$

- ⑩ 假设i\j\k是直角坐标系中X\Y\Z坐标轴的单位向量，则X\Y\Z轴可表示为

$$X = [1 \ 0 \ 0 \ 0]^T$$

$$Y = [0 \ 1 \ 0 \ 0]^T$$

$$Z = [0 \ 0 \ 1 \ 0]^T$$

为什么比例因子为0?

2. 齐次坐标系

位置描述

- 给出点的齐次表达式 $[X \ Y \ w]$, 就可求得其二维笛卡尔坐标, 即

$$[X \ Y \ w] \rightarrow \begin{bmatrix} X & Y & w \\ \hline w & w & w \end{bmatrix} = [x \ y \ 1]$$

这个过程称为正常化处理

- $n+1$ 维的齐次坐标中如果 $w=0$, 实际上就表示了 n 维空间的一个无穷远点

为什么比例因子为0?

$$X = [1 \ 0 \ 0 \ 0]^T$$

$$Y = [0 \ 1 \ 0 \ 0]^T$$

$$Z = [0 \ 0 \ 1 \ 0]^T$$

2. 齐次坐标系

变换描述

使用2D坐标完成平移：加法

$$x' \rightarrow \begin{bmatrix} u' \\ v' \end{bmatrix} = \begin{bmatrix} u + t_u \\ v + t_v \end{bmatrix} = x + t$$

使用齐次坐标完成平移：乘法

$$x' \rightarrow \begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_u \\ 0 & 1 & t_v \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = Tx$$

2. 齐次坐标系

位置描述

直线和平面的一般式方程

使用齐次坐标判断点是否在线上

$$l = \begin{bmatrix} l_1 \\ l_2 \\ l_3 \end{bmatrix}$$

$$x^T l = [u \ v \ 1] \begin{bmatrix} l_1 \\ l_2 \\ l_3 \end{bmatrix} = 0$$

使用齐次坐标判断点是否在平面上

$$\pi = \begin{bmatrix} n_1 \\ n_2 \\ n_3 \\ d \end{bmatrix}$$

$$x^T \pi = [x \ y \ z \ 1] \begin{bmatrix} n_1 \\ n_2 \\ n_3 \\ d \end{bmatrix} = 0$$

2. 齐次坐标系

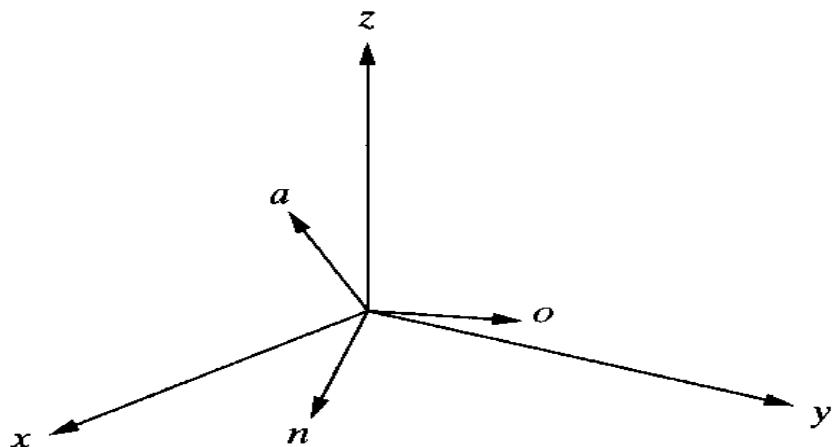
位置描述

坐标系的表示：

I. 坐标系原点与固定参考坐标系原点重合时的表示：

用三个相互垂直的单位向量来表示一个中心位于参考坐标系原点的坐标系，分别为 n, o, a ，这样，坐标系就可以由三个向量以矩阵的形式表示为

$$F = \begin{bmatrix} n_x & o_x & \alpha_x \\ n_y & o_y & \alpha_y \\ n_z & o_z & \alpha_z \end{bmatrix}$$



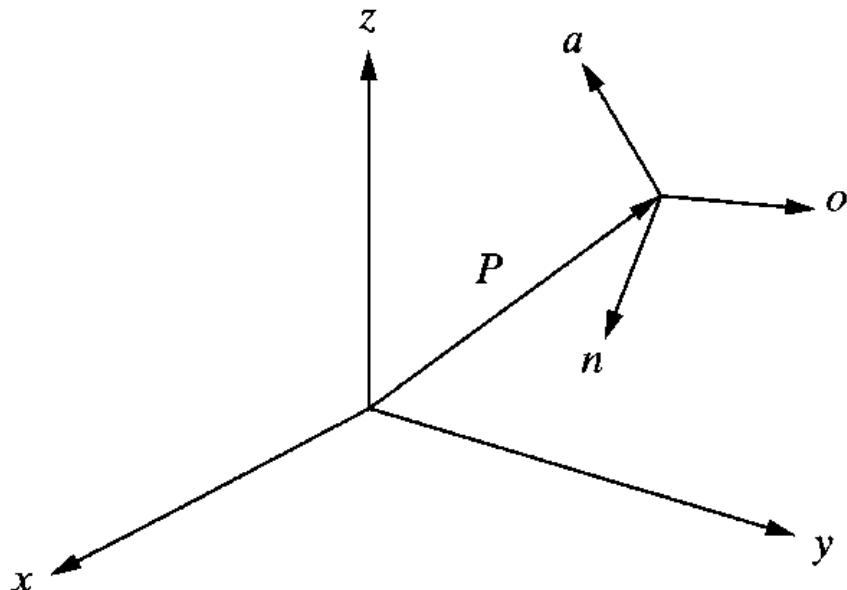
2. 齐次坐标系

位置描述

II. 坐标系原点与固定参考坐标系的原点不重合时的表示：

可以在该坐标系的原点与参考坐标系原点之间做一个向量，而这个向量由上节中提到的参考坐标系的三个坐标向量表示。这样，这个坐标系就可以由三个表示方向的单位向量以及第四个位置向量来表示。

$$F = \begin{bmatrix} n_x & o_x & \alpha_x p_x \\ n_y & o_y & \alpha_y p_y \\ n_z & o_z & \alpha_z p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

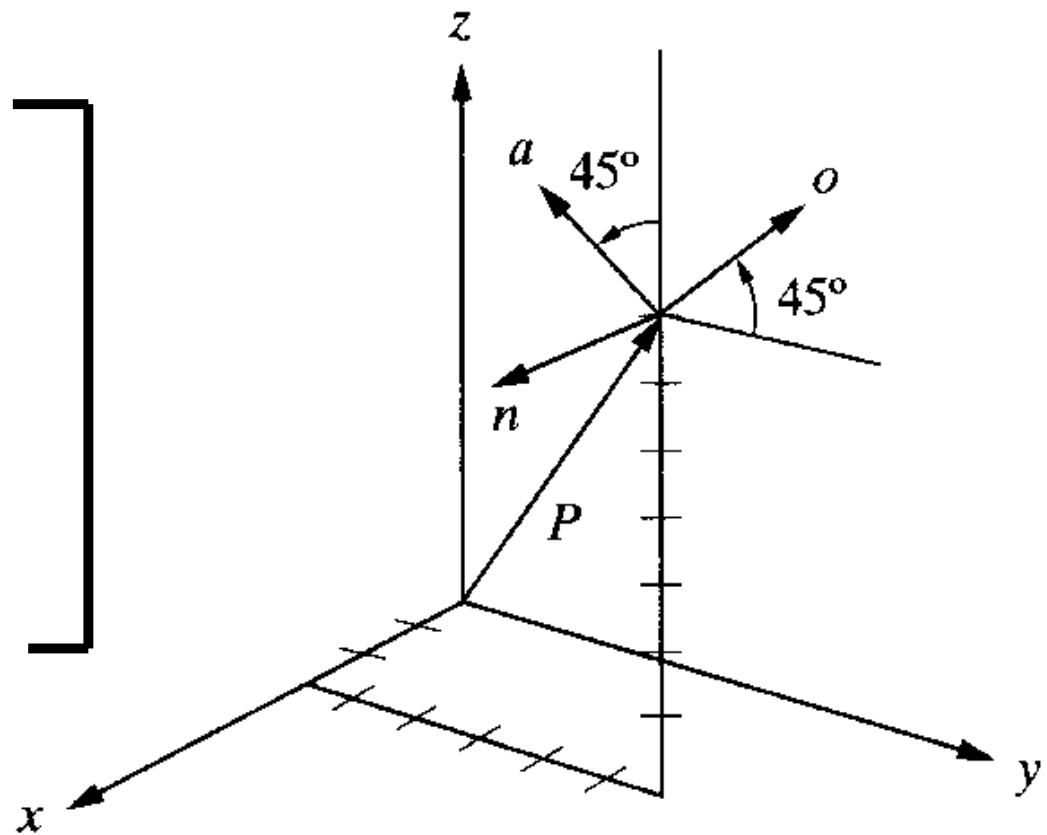


2. 齐次坐标系

位置描述

- 示例：坐标系位于参考坐标系的3, 5, 7的位置。n轴与x轴平行，o轴相对于y轴角度45°，a轴相对于z轴角度45°）

$$F = \begin{bmatrix} n & o & \text{alpha} & p \\ 1 & 0 & 0 & 3 \\ 0 & 0.707 & -0.707 & 5 \\ 0 & 0.707 & 0.707 & 7 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



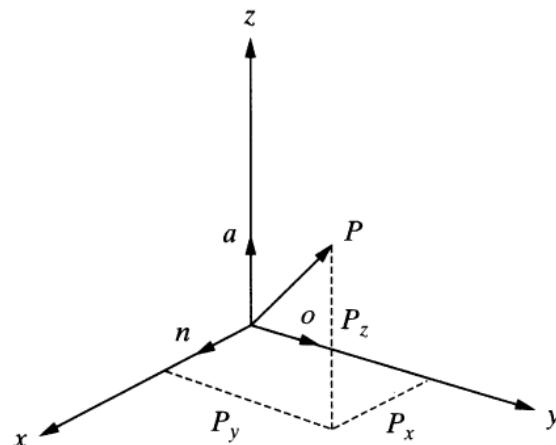
Outline

1. 空间坐标定义
2. 齐次坐标系
3. 空间变换矩阵
4. 旋转向量
5. 欧拉角
6. 四元数
7. 矩阵运算开源库 Eigen

3. 空间变换矩阵

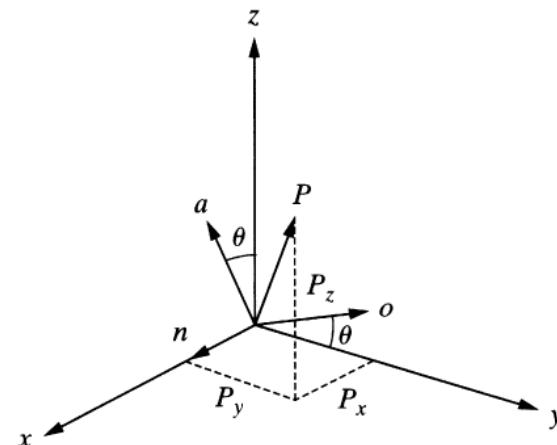
旋转变换 (Rotation transformation)

- 假设坐标系B (n , o , a) 位于参考坐标系A (x , y , z) 的原点，坐标系B (n , o , a) 绕参考坐标系A的x轴旋转一个角度 θ ，再假设旋转坐标系B (n , o , a) 上有一点P相对于参考坐标系A的坐标为 P_x , P_y 和 P_z ，相对于运动坐标系的坐标为 P_n , P_o 和 P_a 。
- 当坐标系统绕x轴旋转时，坐标系上的点P也随坐标系一起旋转，就称为旋转坐标变换。



旋转前

(a)



旋转后

(b)

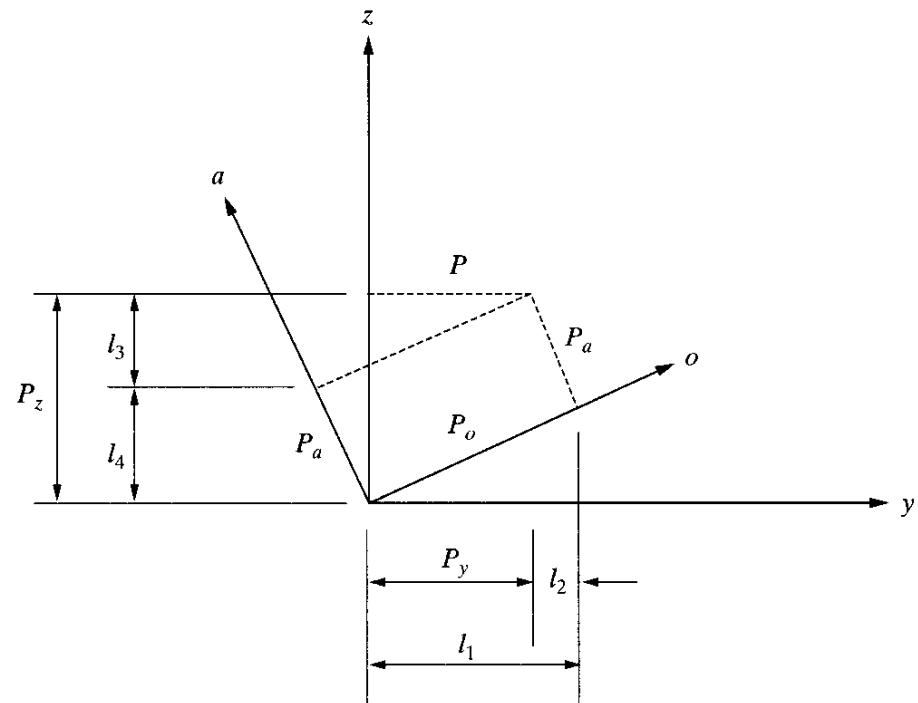
3. 空间变换矩阵

■ 旋转后，该点坐标 P_n , P_o 和 P_a 在旋转坐标系 B 中保持不变，但在参考坐标系 A 中：

$$P_x = P_n$$

$$P_y = l_1 - l_2$$

$$P_z = l_3 + l_4$$

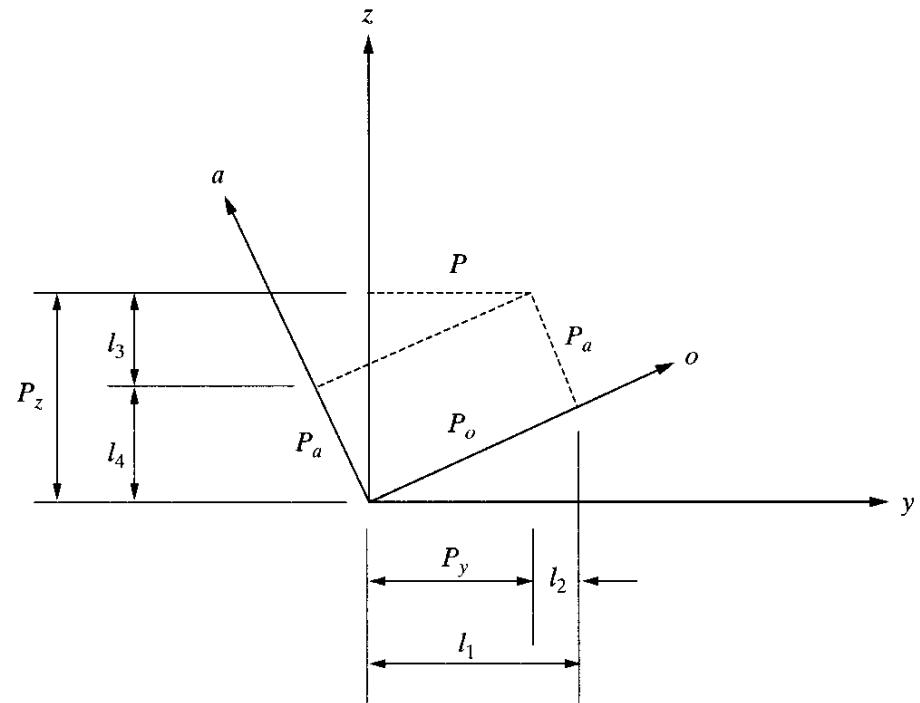


$$\rightarrow \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta \\ 0 & \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} p_n \\ p_o \\ p_a \end{bmatrix}$$

3. 空间变换矩阵

■ 旋转后，该点坐标 P_n , P_o 和 P_a 在旋转坐标系B中保持不变，但在参考坐标系A中：

$$\begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} p_n \\ p_o \\ p_a \end{bmatrix}$$



$$\rightarrow \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} = Rot(x, \theta) \begin{bmatrix} P_n \\ P_o \\ P_a \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P_n \\ P_o \\ P_a \\ 1 \end{bmatrix} \Rightarrow {}^A p = {}_B^A R \cdot {}^B p$$

旋转变换矩阵

3. 空间变换矩阵

左乘表示变换

依此类推

■ 绕 x, y, z 轴分别旋转 θ 角的相应齐次变换是：

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

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

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

■ 假设坐标系 (n, o, a) 和参考坐标系 (x, y, z) 的原点不重合。

■ 用位置矢量表示 $\{B\}$ 的原点相对 $\{A\}$ 的位置，用旋转矩阵表示 $\{B\}$ 相对与 $\{A\}$ 的方位。

$${}^A p_B = {}^A R \cdot {}^B p + {}^A p_B$$

3. 空间变换矩阵

- 更一般地，考虑一次旋转
 - 坐标系 (e_1, e_2, e_3) 发生了旋转，变成 (e'_1, e'_2, e'_3)
 - 向量 a 不动，那么它的坐标如何变化？

$$[e_1, e_2, e_3] \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = [e'_1, e'_2, e'_3] \begin{bmatrix} a'_1 \\ a'_2 \\ a'_3 \end{bmatrix} \xrightarrow{\text{ }} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} e_1^T e'_1 & e_1^T e'_2 & e_1^T e'_3 \\ e_2^T e'_1 & e_2^T e'_2 & e_2^T e'_3 \\ e_3^T e'_1 & e_3^T e'_2 & e_3^T e'_3 \end{bmatrix} \begin{bmatrix} a'_1 \\ a'_2 \\ a'_3 \end{bmatrix} \stackrel{\Delta}{=} Ra'.$$

3. 空间变换矩阵

- R 称为旋转矩阵
- 可以验证：
 - R 是一个正交矩阵；
 - R 的行列式为+1。体积不变，刚体运动
- 满足这两个性质的矩阵称为旋转矩阵

$$SO(n) = \{R \in \mathbb{R}^{n \times n} | RR^T = I, \det(R) = 1\}.$$

- Special Orthogonal Group 特殊正交群

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} e_1^T e_1' & e_1^T e_2' & e_1^T e_3' \\ e_2^T e_1' & e_2^T e_2' & e_2^T e_3' \\ e_3^T e_1' & e_3^T e_2' & e_3^T e_3' \end{bmatrix} \begin{bmatrix} a_1' \\ a_2' \\ a_3' \end{bmatrix} \triangleq Ra'.$$

于是，1到2的旋转可表达为：

$$a_1 = R_{12}a_2$$

$$\text{反之: } a_2 = R_{21}a_1$$

$$\text{矩阵关系: } R_{21} = R_{12}^{-1} = R_{12}^T$$

3. 空间变换矩阵

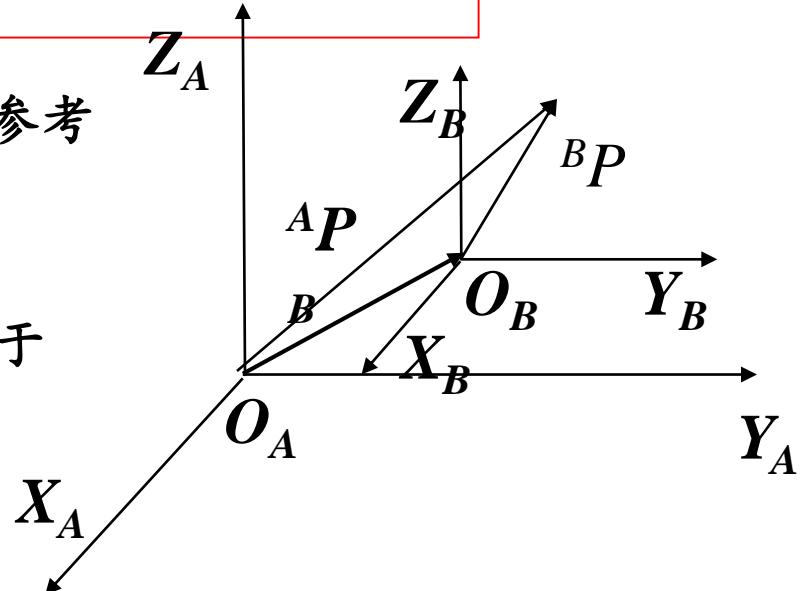
坐标变换

平移变换 (Translation transformation) : 坐标系 {B} 与 {A} 的方向向量平行, 但原点不同。

$$T = \begin{bmatrix} 1 & 0 & 0 & p_x \\ 0 & 1 & 0 & p_y \\ 0 & 0 & 1 & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad {}^A p = T \cdot {}^B p = T \begin{bmatrix} {}^B p_x \\ {}^B p_y \\ {}^B p_z \\ 1 \end{bmatrix} = \begin{bmatrix} p_x + {}^B p_x \\ p_y + {}^B p_y \\ p_z + {}^B p_z \\ 1 \end{bmatrix} \Rightarrow {}^A p = {}^B p + {}^A p_B$$

其中 p_x , p_y 和 p_z 是纯平移向量 ${}^A P_B$ 相对于参考坐标系 x, y 和 z 轴的三个分量。

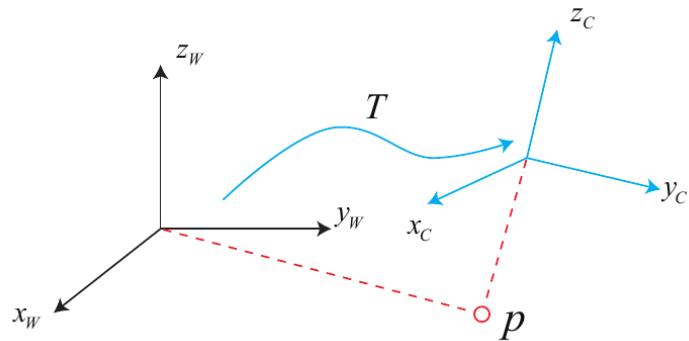
■ 矩阵的前三列表示没有旋转运动 (等同于单位阵), 而最后一列表示平移运动。



3. 空间变换矩阵

- 旋转加平移

$$\mathbf{a}' = \mathbf{R}\mathbf{a} + \mathbf{t}.$$



- 两个坐标系间的运动可用 \mathbf{R}, \mathbf{t} 完全描述
- 欧拉定理(Euler's rotation theorem): 刚体在三维空间里的一般运动, 可分解为刚体上方某一点的平移, 以及绕经过此点的旋转轴的转动。

3. 空间变换矩阵

- 齐次坐标与变换矩阵

$$\mathbf{a}' = \mathbf{R}\mathbf{a} + \mathbf{t}.$$

- 旋转加平移在表达复合情况下有不便之处：

$$\mathbf{b} = \mathbf{R}_1\mathbf{a} + \mathbf{t}_1, \quad \mathbf{c} = \mathbf{R}_2\mathbf{b} + \mathbf{t}_2. \quad \longrightarrow \quad \mathbf{c} = \mathbf{R}_2(\mathbf{R}_1\mathbf{a} + \mathbf{t}_1) + \mathbf{t}_2.$$

- 齐次形式(Homogeneous)：

变换矩阵

$$\begin{bmatrix} \mathbf{a}' \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{a} \\ 1 \end{bmatrix} \triangleq \mathbf{T} \begin{bmatrix} \mathbf{a} \\ 1 \end{bmatrix}. \quad \tilde{\mathbf{b}} = \mathbf{T}_1\tilde{\mathbf{a}}, \quad \tilde{\mathbf{c}} = \mathbf{T}_2\tilde{\mathbf{b}} \Rightarrow \tilde{\mathbf{c}} = \mathbf{T}_2\mathbf{T}_1\tilde{\mathbf{a}}.$$

- 变换矩阵的集合称为特殊欧氏群 SE(3) (Special Euclidean Group)

$$SE(3) = \left\{ \mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \in \mathbb{R}^{4 \times 4} \mid \mathbf{R} \in SO(3), \mathbf{t} \in \mathbb{R}^3 \right\}.$$

逆形式：

$$\mathbf{T}^{-1} = \begin{bmatrix} \mathbf{R}^T & -\mathbf{R}^T \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix}.$$

3. 空间变换矩阵

位置描述

----->

姿态描述

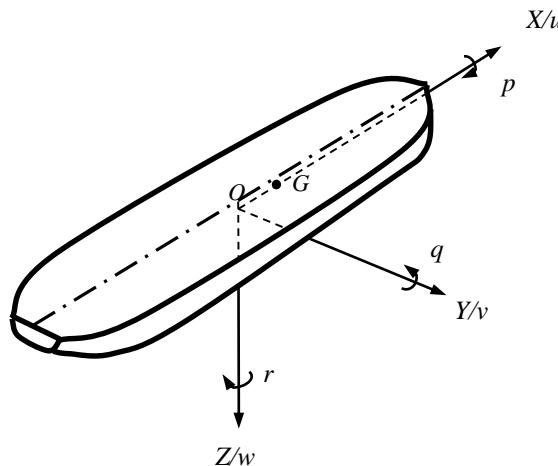
❖ 姿态描述: 刚体的空间表示。

- 通常的做法是: 定义两个坐标系

空间固定坐标系和刚体固定坐标系。

- 通常用自由度来描述刚体运动路径多少

- 思考: 一个刚体在空间有几个自由度?



1. 空间坐标变换

位姿描述----旋转+平移变换

- ❖ 位置与姿态简称位姿。刚体B在参考坐标系 {A} 中的位姿利用坐标系 {B} 描述。
 - 齐次变换矩阵形式

$$\{B\} = \{{}^A R_B, {}^A p_B\} ==> {}^A T_B = \begin{bmatrix} n_x & o_x & \alpha_x & p_x \\ n_y & o_y & \alpha_y & p_y \\ n_z & o_z & \alpha_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3. 空间变换矩阵

姿态描述

❖ 旋转矩阵的性质：

$${}^A R_B = [n \quad o \quad \alpha] = \begin{bmatrix} n_x & o_x & \alpha_x \\ n_y & o_y & \alpha_y \\ n_z & o_z & \alpha_z \end{bmatrix}$$

- 单位向量之间相互垂直，正交。

$$\begin{aligned} {}^A x_B \cdot {}^A x_B &= {}^A y_B \cdot {}^A y_B = {}^A z_B \cdot {}^A z_B = 1 \\ {}^A x_B \cdot {}^A y_B &= {}^A y_B \cdot {}^A z_B = {}^A z_B \cdot {}^A x_B = 0 \end{aligned}$$

- 正交矩阵：

$${}^A R_B^{-1} = {}^A R_B^T, \quad |{}^A R_B^T| = 1$$

任意刚体运动都对应一个正交矩阵

3. 空间变换矩阵

位置描述

----->

姿态描述

❖ 姿态描述: 刚体的空间表示。

▪ 常用的姿态描述:

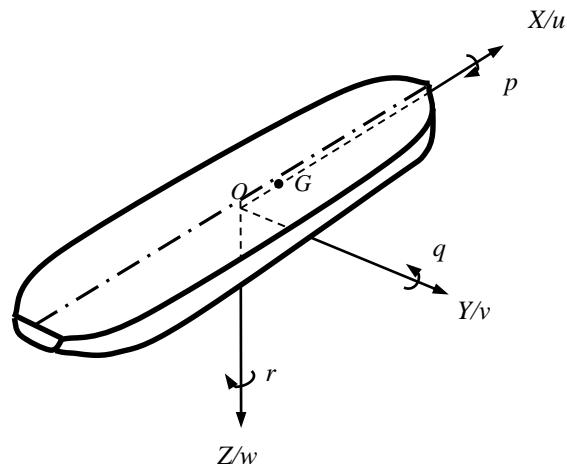
✓ 旋转矩阵的姿态描述 (笛卡尔坐标系下)

✓ 旋转向量

✓ 四元数

✓ 欧拉 (Euler) 角的姿态描述,

利用横滚 (R: Roll) 、俯仰 (P: pitch) 、偏转 (Y: yaw) 角描述。



固定坐标系下六个自由度上的运动分量

3. 空间变换矩阵

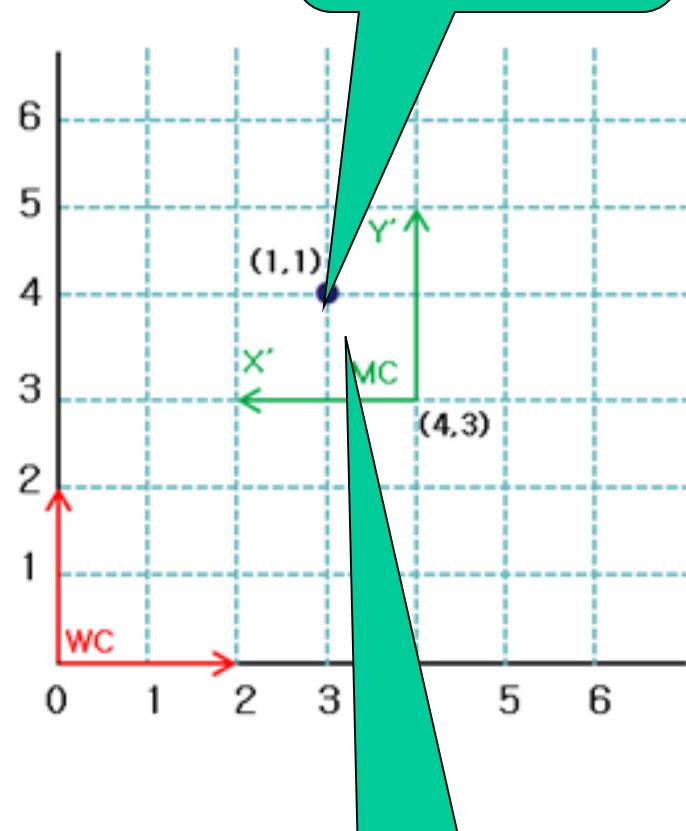
❖ 简单实例，坐标变换

第一列为MC中x轴在WC中向量表示；

第二列为MC中y轴在WC中向量表示；

第三列为MC中原点在WC中坐标；

$$MC = \begin{bmatrix} -1 & 0 & 4 \\ 0 & 1 & 3 \\ 0 & 0 & 1 \end{bmatrix}$$



3. 空间变换矩阵

- 将齐次矩阵作为变换矩阵，乘以MC中坐标，得
WC中坐标
或者说，MC中点坐标左乘MC的齐次坐标

$$\begin{bmatrix} -1 & 0 & 4 \\ 0 & 1 & 3 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \\ 1 \end{bmatrix}$$

3. 空间变换矩阵

加入观察坐标

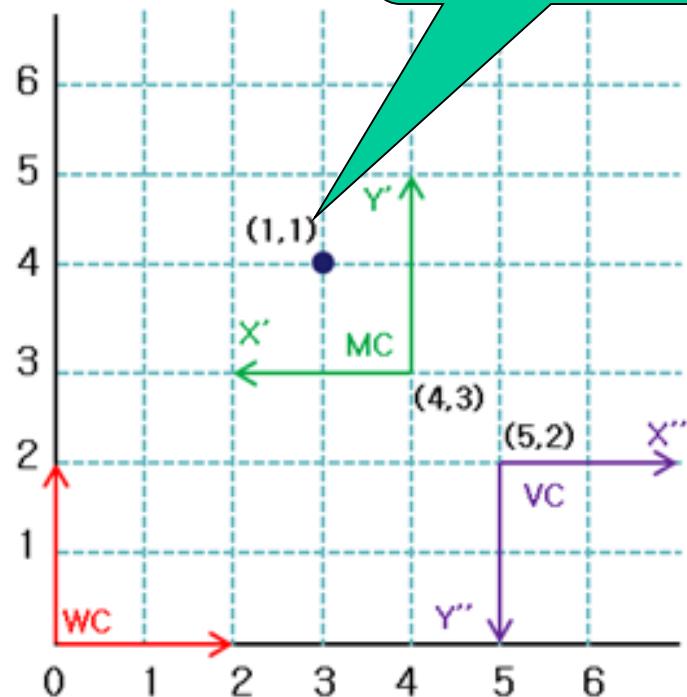
- VC对世界标系WC的齐次矩阵及其逆为

$$VC = \begin{bmatrix} 1 & 0 & 5 \\ 0 & -1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \quad VC^{-1} = \begin{bmatrix} 1 & 0 & -5 \\ 0 & -1 & 2 \\ 0 & 0 & 1 \end{bmatrix}$$

- 世界坐标到观察坐标

$$\begin{bmatrix} 1 & 0 & -5 \\ 0 & -1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 4 \\ 1 \end{bmatrix} = \begin{bmatrix} -2 \\ -2 \\ 1 \end{bmatrix}$$

观察坐标系中的坐标为(-2,-2)



3. 空间变换矩阵

坐标变换

- 任何变换都可以分解为按一定顺序的一组平移和旋转变换。
- 示例：假设坐标系 (n, o, a) 位于参考坐标系 (x, y, z) 的原点，坐标系 (n, o, a) 上的点 $P(7, 3, 2)$ 经历如下变换，求出变换后该点相对于参考坐标系的坐标。
 - (1) 绕 z 轴旋转 90 度；
 - (2) 接着绕 y 轴旋转 90 度；
 - (3) 接着再平移 $[4, -3, 7]$ 。

$$P_{xyz} = Trans(4, -3, 7) Rot(y, 90) Rot(z, 90) P_{noa}$$

3. 空间变换矩阵

例题：

- ❖ $\{B\}$ 和 $\{A\}$ 位姿重合。现在将 $\{B\}$ 绕 $\{A\}$ z_A 轴转 30 度，再沿 $\{A\}$ 的 x_A 轴移动 12 单位，再沿 $\{A\}$ 的 y_A 轴移动 6 单位。假设点 p 在 $\{B\}$ 中位置为 $[5, 9, 0]^T$ ，求点 p 在 $\{A\}$ 中位置。

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

- ${}^A p_B = [12, 6, 0, 1]^T$

- ${}^A p = [11.1, 13.6, 0, 1]^T$

3. 空间变换矩阵

位置描述

----->

投影描述

使用齐次坐标完成旋转和平移：

$$x' \rightarrow \begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & t_u \\ \sin\theta & \cos\theta & t_v \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_u \\ 0 & 1 & t_v \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = TRx$$

使用齐次坐标完成平移和放缩：

$$x' \rightarrow \begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} = \begin{bmatrix} s_u & 0 & s_u t_u \\ 0 & s_v & s_v t_v \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} s_u & 0 & 0 \\ 0 & s_v & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & t_u \\ 0 & 1 & t_v \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = STx$$

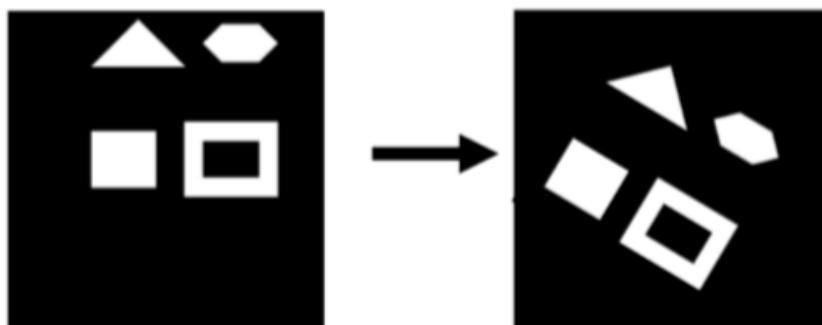
3. 空间变换矩阵

位置描述
等距变换



投影描述

2D的等距变换具有三自由度，这种变换是保距离的



$$\begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

R为旋转矩阵，t为平移向量

3. 空间变换矩阵

位置描述
相似变换

----->

投影描述

2D的相似变换具有四自由度，这种变换是保角度的



$$\begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} = \begin{bmatrix} sR & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

s为相似变换因子

3. 空间变换矩阵

位置描述
仿射变换

----->

投影描述

2D的仿射变换具有六自由度，这种变化是保平行的



$$\begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} = \begin{bmatrix} A & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = R(\theta)R(-\phi)SR(\phi)$$

$$S = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$$

3. 空间变换矩阵

位置描述 -----> 投影描述
射影变换

2D的射影变换具有八自由度，这种变化是保同线性的



$$\begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} = \begin{bmatrix} A & t \\ v & b \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

$$H = \begin{bmatrix} A & t \\ v & b \end{bmatrix}$$

3. 空间变换矩阵

位置描述 -----> 投影描述

2D齐次坐标的线性变换



$$\begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$



$$\begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} = \begin{bmatrix} sR & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$



$$\begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} = \begin{bmatrix} A & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$



$$\begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} = \begin{bmatrix} A & t \\ v & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

Outline

1. 空间坐标定义
2. 齐次坐标系
3. 空间变换矩阵
4. 旋转向量
5. 欧拉角
6. 四元数
7. 矩阵运算开源库 Eigen

4. 旋转向量

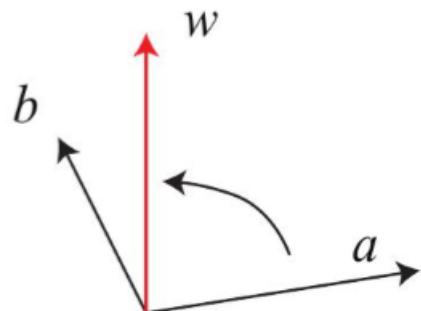
- 给定参考系, 向量的运算可由坐标运算表达
- 加法和减法
- 内积

$$\mathbf{a} \cdot \mathbf{b} = \mathbf{a}^T \mathbf{b} = \sum_{i=1}^3 a_i b_i = |\mathbf{a}| |\mathbf{b}| \cos \langle \mathbf{a}, \mathbf{b} \rangle.$$

反对称矩阵

- 外积

$$\mathbf{a} \times \mathbf{b} = \begin{bmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{bmatrix} = \begin{bmatrix} a_2 b_3 - a_3 b_2 \\ a_3 b_1 - a_1 b_3 \\ a_1 b_2 - a_2 b_1 \end{bmatrix} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \mathbf{b} \triangleq \mathbf{a} \wedge \mathbf{b}.$$



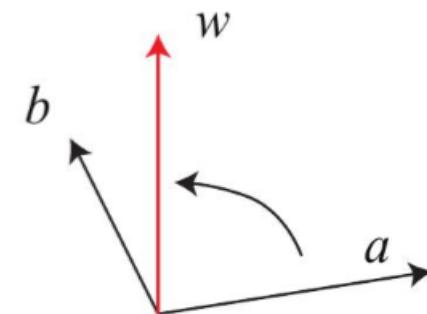
4. 旋转向量

- ❖ 除了旋转矩阵表示欧式变换之外，还存在其他的表示方式
- ❖ 旋转矩阵 R 有九个元素，但仅有三个自由度
- ❖ 能否以更少的元素表达旋转？

旋转向量

方向为旋转轴，长度为转过的角度

称为角轴/Angle Axis 或旋转向量
(Rotation Vector)



4. 旋转向量

- ❖ 旋转向量与变换矩阵的不同：
 - 仅有三个量
 - 无约束
 - 更直观
- ❖ 它们可以是同一个东西的不同表达方式
- ❖ 罗德里格斯公式 (Rodrigues's Formula) :

$$\mathbf{R} = \cos \theta \mathbf{I} + (1 - \cos \theta) \mathbf{n} \mathbf{n}^T + \sin \theta \mathbf{n}^\wedge$$

旋转矩阵转向量：

角度： $\theta = \arccos\left(\frac{\text{tr}(\mathbf{R}) - 1}{2}\right)$

轴： $\mathbf{Rn} = \mathbf{n}$

旋转轴上的向量在旋转后不变

Outline

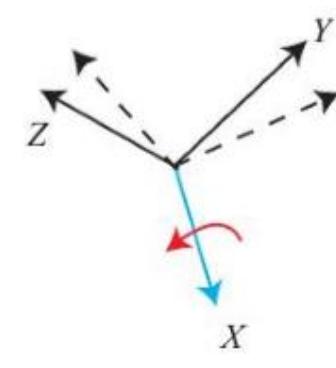
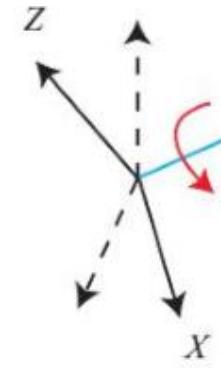
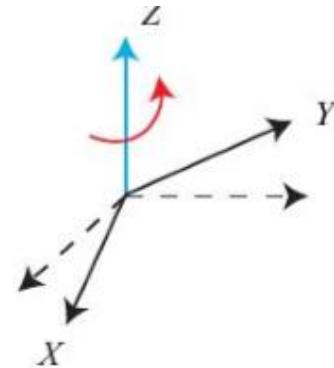
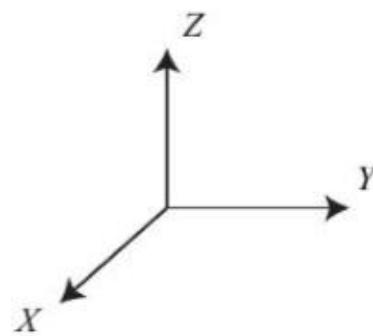
1. 空间坐标定义
2. 齐次坐标系
3. 空间变换矩阵
4. 旋转向量
5. 欧拉角
6. 四元数
7. 矩阵运算开源库 Eigen

5. 欧拉角

将旋转分解为三个方向上的转动

- 例, 按Z-Y-X顺序转动
- 轴可以是定轴或动轴, 顺序亦可不同
- 常见的有: yaw-pitch-roll, 东北天
- 不同领域的习惯有所不同

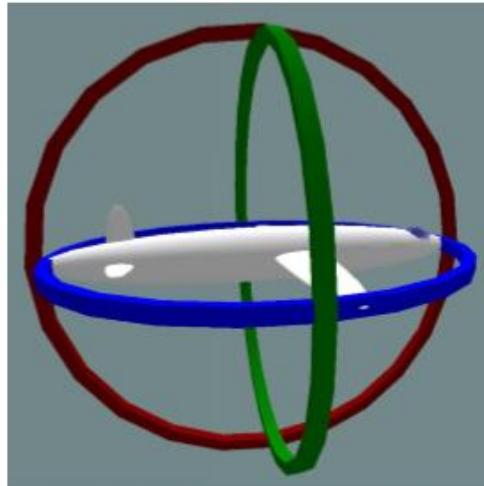
1. 绕物体的Z轴旋转, 得到偏航角yaw;
2. 绕旋转之后的Y轴旋转, 得到俯仰角pitch;
3. 绕旋转之后的X轴旋转, 得到滚转角roll



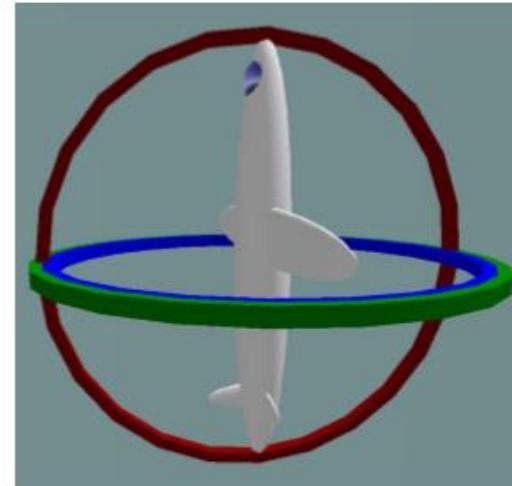
5. 欧拉角

万向锁(Gimbal Lock)

- 欧拉角的奇异性问题
- 在特定值时，旋转自由度减1；
- Yaw-pitch-roll顺序下，当pitch为90度时，存在奇异性



正常情况



奇异性情况

Outline

1. 空间坐标定义
2. 齐次坐标系
3. 空间变换矩阵
4. 旋转向量
5. 欧拉角
6. 四元数
7. 矩阵运算开源库 Eigen

四元数

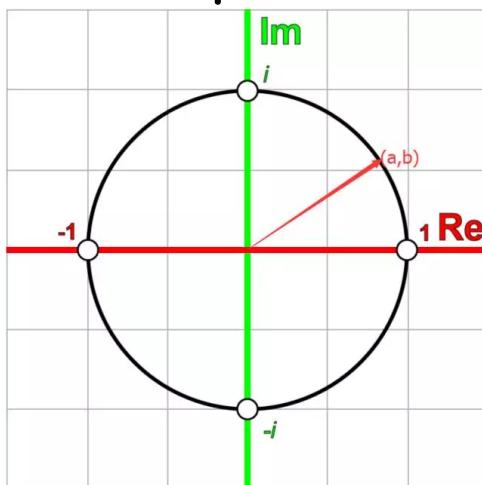
- 从虚数开始，虚数形式如下：

$$i^2 = -1$$

虚数和实数结合定义成复数，它的形式如下：

$$p = a + bi \quad (a \text{ 和 } b \text{ 都是实数, } i \text{ 是虚数})$$

复数放到二维平面上可以理解成向量， a 是二维平面X轴上的分量， bi 是Y轴上的分量，即 $p=a+bi$ 可以看作成向量 (a,b)



复数乘法具有
向量旋转功能

图中虚数的方向是处于90度和270度方向上的，如果一个实数 b 乘以虚数 i ，那它在二维平面上的表现就相当于将向量 $(b,0)$ 绕垂直于平面的Z轴逆时针旋转90度，得到新的向量 $(0,b)$

四元数

- 假设需要旋转的角度是 θ 度，可以先画出二维平面上 θ 角对应的单位向量 $(\cos\theta, \sin\theta)$ ，再根据这个单位向量定义一个复数旋转数：

$$q = \cos\theta + \sin\theta i$$

假设要转动的向量复数为：

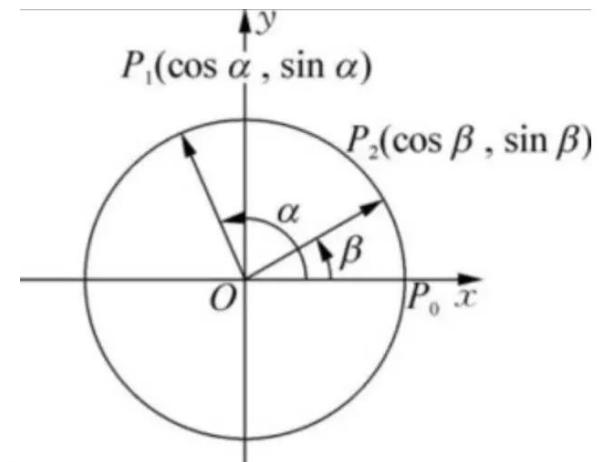
$$p = a + bi$$

将要转动的向量复数乘以复数旋转数：

$$pq = (a + bi)(\cos\theta + \sin\theta i)$$

得到旋转后的复数：

$$p' = a' + b'i = a\cos\theta - b\sin\theta + (a\sin\theta + b\cos\theta)i.$$



四元数

❖ 将2维旋转推广到3维

由于三维空间比二维平面多了一个轴，因而三维的复数应该比二维的复数增加一个虚数，假设增加的虚数为j，j也满足虚数性质：

$$j^2 = -1$$

则三维空间的复数可以定义为：

$$p = a_1 + b_1 i + c_1 j$$

$$q = a_2 + b_2 i + c_2 j$$

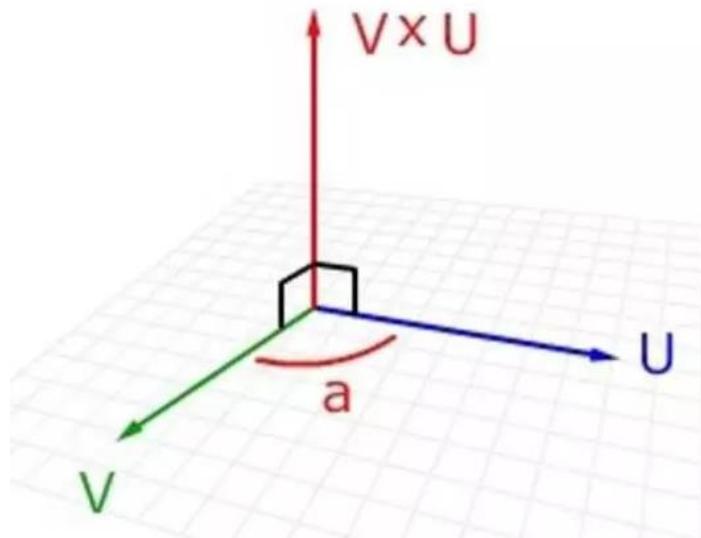
多出了两个项
ij和ji

将p和q相乘有 $pq = (a_1 + b_1 i + c_1 j)(a_2 + b_2 i + c_2 j)$

$$\begin{aligned} &= a_1 a_2 + a_1 b_2 i + a_1 c_2 j + a_2 b_1 i - b_1 b_2 + b_1 c_2 ij + a_2 c_1 j + c_1 b_2 ji - c_1 c_2 \\ &= (a_1 a_2 - b_1 b_2 - c_1 c_2) + (a_1 b_2 + b_1 a_2)i + (a_1 c_2 + c_1 a_2)j + b_1 c_2 ij + c_1 b_2 ji \end{aligned}$$

四元数

- 两个向量的叉积或者向量积，等于垂直于这两个向量的向量



虽然 a 垂直于 i 和 j 构成的平面，但是 ij 不等于实数，怎么办？

四元数

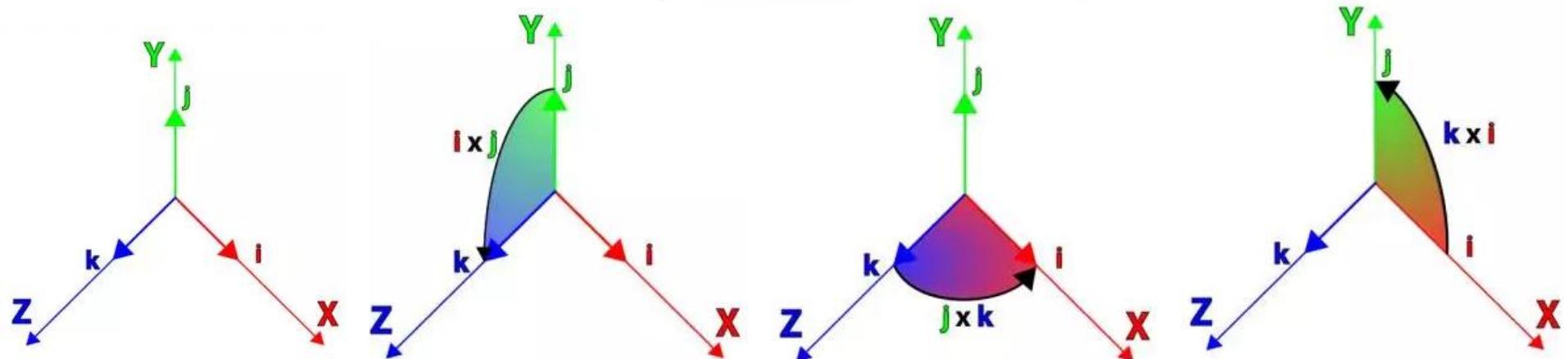
◆ 3维到4维

四元数发明者哈密尔顿引入四维空间的概念，四维空间里存在四维向量或者说四维的复数，这个四维的复数被哈密尔顿定义为四元数，它的形式如下：

$$q = s + xi + yj + zk$$

将s当成第四维，而ijk对应xyz三维空间，则有：

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



四元数

❖ 运算满足

$$ij = k, jk = i, ki = j$$

$$ji = -k, kj = -i, ik = -j$$

则: $i^2 = j^2 = k^2 = ijk = -1$.

s 是实数, 对应特殊的第四维, \mathbf{v} 对应正常空间的三个维度,
通常表达如下:

$$\mathbf{q} = [s, \mathbf{v}]$$

四元数

◆ 表达形式

$$q = [s, xi + yj + zk]$$

设：

$$q_a = [s_a, \mathbf{a}] \text{ 或者 } q_a = (s_a + x_a i + y_a j + z_a k)$$

$$q_b = [s_b, \mathbf{b}] \text{ 或者 } q_b = (s_b + x_b i + y_b j + z_b k)$$

$$\begin{aligned} q_a q_b &= (s_a + x_a i + y_a j + z_a k)(s_b + x_b i + y_b j + z_b k) \\ &= (s_a s_b - x_a x_b - y_a y_b - z_a z_b) + \\ &\quad + (s_a x_b + s_b x_a + y_a z_b - y_b z_a) i + \\ &\quad + (s_a y_b + s_b y_a + z_a x_b - z_b x_a) j + \\ &\quad + (s_a z_b + s_b z_a + x_a y_b - x_b y_a) k \end{aligned}$$

$$q_a q_b = [s_a s_b - \mathbf{a} \cdot \mathbf{b}, s_a \mathbf{b} + s_b \mathbf{a} + \mathbf{a} \times \mathbf{b}]$$

式中 $\mathbf{a} \cdot \mathbf{b}$ 是向量点乘，运算结果是个实数，故而放在实数区，而 $\mathbf{a} \times \mathbf{b}$ 是叉积，运算结果是个垂直于 \mathbf{a} 和 \mathbf{b} 向量，故而放在虚数区

四元数

◆ 属性分析

由于要旋转的向量是个三维空间的向量，所以它是一个实数轴为0的纯四元数，则可以将 q_a 修改成：

$$q_a = [0, \mathbf{a}] \text{ 或者 } q_a = (0 + x_a i + y_a j + z_a k)$$

将 q_a 代入 $q_a q_b$ 相乘结果中，有：

$$q_a q_b = [-\mathbf{a} \cdot \mathbf{b}, s_b \mathbf{a} + \mathbf{a} \times \mathbf{b}]$$

要研究的是三维空间中的旋转，所以需要三维向量乘以四元数旋转后还是三维向量，需要使得 $q_a q_b$ 相乘结果中的实数项为0，即 $\mathbf{a} \cdot \mathbf{b}$ 值为0，也就是两个向量互相垂直正交

于是定义 q_b ，为了使 q_b 能让纯四元数 q_a 旋转 θ 度， q_b 的旋转四元数也参照二维空间的旋转数复数来表示：

$$q_b = [\cos \theta, \sin \theta \mathbf{v}]$$

四元数

$$q_a q_b = [-\mathbf{a} \cdot \mathbf{b}, s_b \mathbf{a} + \mathbf{a} \times \mathbf{b}]$$

◆ 属性分析

$$q_a q_b = [0, \cos\theta \mathbf{a} + \mathbf{a} \times \sin\theta \mathbf{v}]$$

假设要旋转的三维向量为：

$$q_a = [0, 2i]$$

假设要旋转的角度是45度，则： $q_b = \left[\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2} \mathbf{v} \right]$

\mathbf{v} 需要垂直于 q_a 的三维向量部分，即 \mathbf{v} 垂直于 $2i$ ，不妨先将 \mathbf{v} 设定成 k ，即z轴，有：

$$q_b = \left[\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2} k \right]$$

$$q_a q_b = [0, \cos\theta \mathbf{a} + \mathbf{a} \times \sin\theta \mathbf{v}]$$

qa绕z轴旋转了45度角，是顺时针旋转。

四元数

◆ 改变旋转方向

通常数学上的
旋转都是逆时
针

$$q_b q_a = [0, \cos\theta \mathbf{a} + \sin\theta \mathbf{v} \times \mathbf{a}]$$

以上的四元数运算都只能使得 qa 可以往任意一个方向旋转，它的旋转轴始终垂直于自身，而现实情况可能还需要能绕任意的旋转轴旋转，即旋转轴的三维向量 b 不垂直于 a ，进而使得公式：

$$q_a q_b = [-\mathbf{a} \cdot \mathbf{b}, s_b \mathbf{a} + \mathbf{a} \times \mathbf{b}]$$

中的实数项不为0，即三维向量旋转到四维空间去了，如何解决？

四元数

◆ 共轭四元数的提出

共轭复数

$$z = a + bi$$

$$z' = a - bi$$

实数不变，虚数相反，四元数是高阶复数，也遵循这个原理

$q = [s, \mathbf{v}]$ 的共轭四元数是 $q' = [s, -\mathbf{v}]$

四元数

◆ 共轭四元数

$\mathbf{q} = [s, \mathbf{v}]$ 的共轭四元数是 $\mathbf{q}' = [s, -\mathbf{v}]$

假设要旋转的向量是： $q_a = [0, \mathbf{a}]$

哈密尔顿用了如下方式来解决 qa 绕任意轴旋转的问题：

$$q_b q_a q_b' = [s_b, \mathbf{b}] [0, \mathbf{a}] [s_b, -\mathbf{b}]$$

“再乘以共轭四元数后，从4维回到3维”

四元数

❖ 实例

$$q_a = [0, 2i]$$

旋转角度也还是选择45度，但旋转四元数 q_b 的三维向量 b 不再垂直于 a ，设：

$$\begin{aligned} q_b &= [\cos 45, \sin 45 (\frac{\sqrt{2}}{2}i + \frac{\sqrt{2}}{2}k)] \\ &= [\frac{\sqrt{2}}{2}, \frac{1}{2}(i+k)] \end{aligned}$$

即 q_b 的三维向量 b 是 x 轴和 z 轴的中间斜线，不再垂直于 a 向量，则：

$$q_b q_a q_b' = [\frac{\sqrt{2}}{2}, \frac{1}{2}(i+k)] [0, 2i] [\frac{\sqrt{2}}{2}, -\frac{1}{2}(i+k)]$$

四元数

❖ 实例

利用以下公式：

$$q_a q_b = [s_a s_b - \mathbf{a} \cdot \mathbf{b}, s_a \mathbf{b} + s_b \mathbf{a} + \mathbf{a} \times \mathbf{b}]$$

代入得：

$$\begin{aligned} q_b q_a q_b' &= \left[-\frac{1}{2}(i+k) \cdot 2i, \frac{\sqrt{2}}{2}2i + \frac{1}{2}(i+k) \times 2i \right] \left[\frac{\sqrt{2}}{2}, -\frac{1}{2}(i+k) \right] \\ &= [-1, \sqrt{2}i + j] \left[\frac{\sqrt{2}}{2}, -\frac{1}{2}(i+k) \right] \\ &= \left[-\frac{\sqrt{2}}{2} - \frac{\sqrt{2}}{2}i^2, \frac{1}{2}(i+k) + i + \frac{\sqrt{2}}{2}j + \frac{\sqrt{2}}{2}j + \frac{1}{2}k - \frac{1}{2}i \right] \\ &= [0, i + \sqrt{2}j + k] \end{aligned}$$

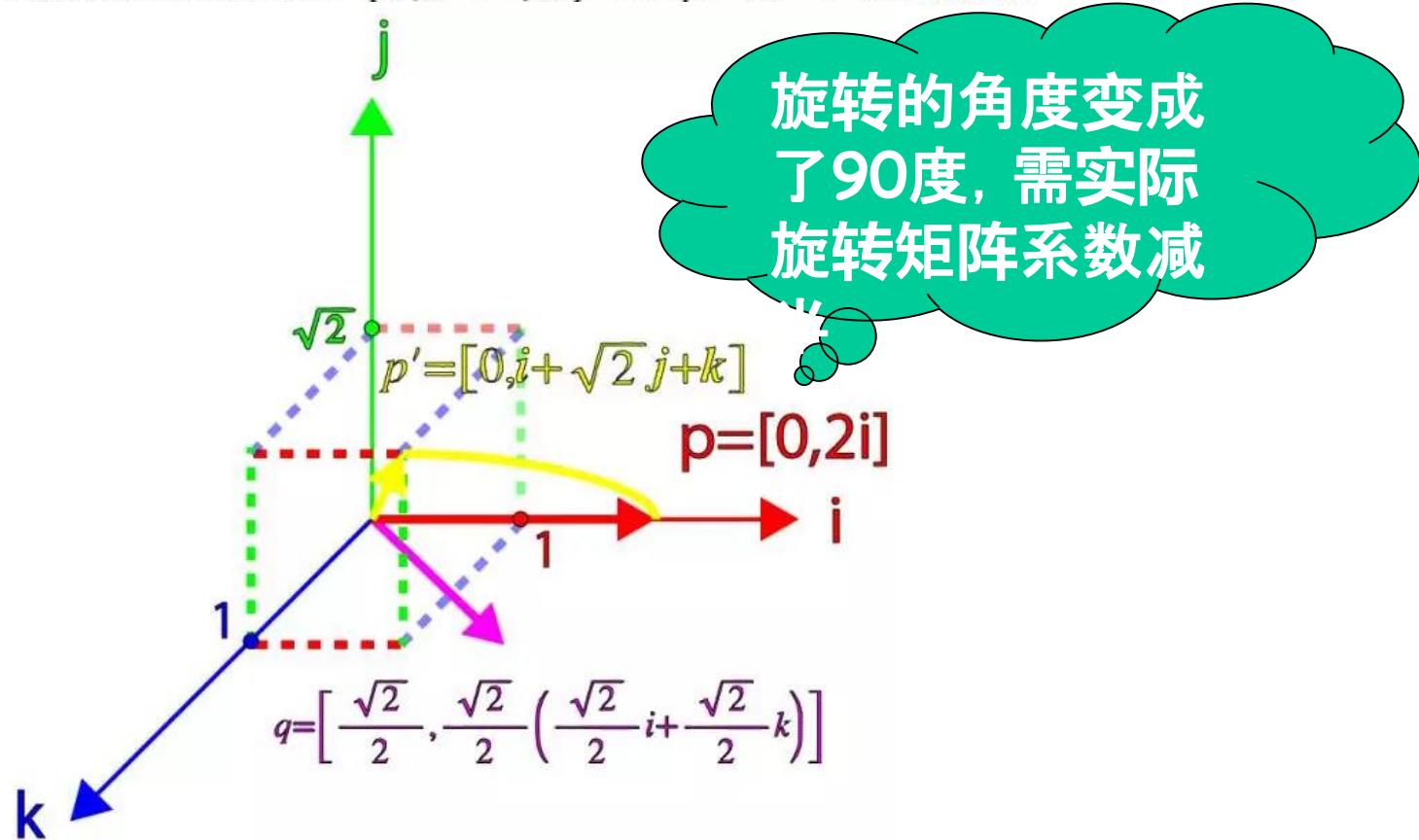
四元数

❖ 实例

旋转向量 $q_a = [0, 2i]$

旋转轴 $q_b = [\cos 45, \sin 45 \left(\frac{\sqrt{2}}{2} i + \frac{\sqrt{2}}{2} k \right)]$

实际旋转结果是: $[0, i + \sqrt{2}j + k]$, 如下图所示:



四元数

- 四元数的运算

$$\mathbf{q}_a \pm \mathbf{q}_b = [s_a \pm s_b, \mathbf{v}_a \pm \mathbf{v}_b].$$

$$\mathbf{q}_a^* = s_a - x_a i - y_a j - z_a k = [s_a, -\mathbf{v}_a].$$

$$\begin{aligned}\mathbf{q}_a \mathbf{q}_b &= s_a s_b - x_a x_b - y_a y_b - z_a z_b \\&\quad + (s_a x_b + x_a s_b + y_a z_b - z_a y_b) i \\&\quad + (s_a y_b - x_a z_b + y_a s_b + z_a x_b) j \\&\quad + (s_a z_b + x_a y_b - y_b x_a + z_a s_b) k.\end{aligned}$$

$$\|\mathbf{q}_a\| = \sqrt{s_a^2 + x_a^2 + y_a^2 + z_a^2}.$$

$$k\mathbf{q} = [ks, k\mathbf{v}].$$

$$\mathbf{q}_a \cdot \mathbf{q}_b = s_a s_b + x_a x_b i + y_a y_b j + z_a z_b k.$$

$$\mathbf{q}_a \mathbf{q}_b = [s_a s_b - \mathbf{v}_a^T \mathbf{v}_b, s_a \mathbf{v}_b + s_b \mathbf{v}_a + \mathbf{v}_a \times \mathbf{v}_b].$$

四元数

❖ 四元数的运算

四元数到旋转向量角轴:

$$\begin{cases} \theta = 2 \arccos q_0 \\ [n_x, n_y, n_z]^T = [q_1, q_2, q_3]^T / \sin \frac{\theta}{2} \end{cases} .$$

旋转向量角轴到四元数:

$$q = \left[\cos \frac{\theta}{2}, n_x \sin \frac{\theta}{2}, n_y \sin \frac{\theta}{2}, n_z \sin \frac{\theta}{2} \right]^T .$$

四元数

❖ 小结

- ✓ 四元数是一种高阶复数，相当于四维空间向量，元等价于维度。
- ✓ 四元数这种高阶复数的实数对应四维空间的第四维度，而剩下的3个虚数对应三维空间的三个维度。
- ✓ 复数相乘等价于空间旋转，四元数是四维复数，四元数相乘就是四维空间向量旋转。
- ✓ 四元数的3个虚数就是三维空间中的旋转轴。
- ✓ 四元数能绕任意轴旋转，而欧拉角只能绕xyz轴旋转。

Outline

1. 空间坐标定义
2. 齐次坐标系
3. 空间变换矩阵
4. 旋转向量
5. 欧拉角
6. 四元数
7. 矩阵运算开源库 Eigen

开源库 Eigen

一个高层次的C++库，有效支持线性代数，矩阵和矢量运算，

- ✓ 使用CMake建立配置文件和单元测试，自动安装；
- ✓ 如果使用Eigen库，只需包特定模块的头文件即可
- ✓ In Eigen, all matrices and vectors are objects of the Matrix template class.

`Matrix<typename Scalar, int RowsAtCompileTime, int ColsAtCompileTime>`

Example:

```
typedef Matrix<float, 4, 4> Matrix4f;
```



开源库 Eigen

- ✓ in Eigen, vectors are just a special case of matrices, with either 1 row or 1 column
- ✓ The case where they have 1 column is the most common; such vectors are called column-vectors

a (column) vector of 3 floats

```
typedef Matrix<float, 3, 1> Vector3f;
```

row-vectors

```
typedef Matrix<int, 1, 2> RowVector2i;
```

```
Matrix3f m;
```

```
m << 1, 2, 3,
```

1	2	3
---	---	---

```
4, 5, 6,
```

4	5	6
---	---	---

```
7, 8, 9;
```

7	8	9
---	---	---

```
std::cout << m;
```

开源库 Eigen

- ✓ For matrices, the row index is always passed first. For vectors, just pass one index. The numbering starts at 0.

Example:

```
#include <iostream>
#include <Eigen/Dense>

using namespace Eigen;

int main()
{
    MatrixXd m(2,2);
    m(0,0) = 3;
    m(1,0) = 2.5;
    m(0,1) = -1;
    m(1,1) = m(1,0) + m(0,1);
    std::cout << "Here is the matrix m:\n" << m << std::endl;
    VectorXd v(2);
    v(0) = 4;
    v(1) = v(0) - 1;
    std::cout << "Here is the vector v:\n" << v << std::endl;
}
```

Output:

```
Here is the matrix m:
 3  -1
 2.5 1.5
Here is the vector v:
 4
 3
```

开源库 Eigen

Eigen提供了常见矩阵形式，包括用于描述空间变换的各类型数据结构

旋转矩阵 (3×3): Eigen::Matrix3d。

旋转向量 (3×1): Eigen::AngleAxisd。

欧拉角 (3×1): Eigen::Vector3d。

四元数 (4×1): Eigen::Quaterniond。

欧氏变换矩阵 (4×4): Eigen::Isometry3d。

仿射变换 (4×4): Eigen::Affine3d。

射影变换 (4×4): Eigen::Projective3d。

```
#include <iostream>
#include <cmath>
using namespace std;

#include <Eigen/Core>
// Eigen 几何模块
#include <Eigen/Geometry>

/******************
* 本程序演示了 Eigen 几何模块的使用方法
******************/

int main( int argc, char** argv )
{
    // Eigen/Geometry 模块提供了各种旋转和平移的表示
    // 3D 旋转矩阵直接使用 Matrix3d 或 Matrix3f
    Eigen::Matrix3d rotation_matrix = Eigen::Matrix3d::Identity();
    // 旋转向量使用 AngleAxis，它底层不直接是 Matrix，但运算可以当作矩阵（因为重载了运算符）
    Eigen::AngleAxisd rotation_vector ( M_PI/4, Eigen::Vector3d ( 0,0,1 ) ); // 沿 Z 轴旋转 45 度
    cout .precision(3);
    cout<<"rotation matrix =\n"<<rotation_vector.matrix() <<endl; //用      matrix() 转换成矩阵
    // 也可以直接赋值
    rotation_matrix = rotation_vector.toRotationMatrix();
    // 用 AngleAxis 可以进行坐标变换
    Eigen::Vector3d v ( 1,0,0 );
    Eigen::Vector3d v_rotated = rotation_vector * v;
    cout<<"(1,0,0) after rotation = "<<v_rotated.transpose()<<endl;
```

```
// 或者用旋转矩阵
v_rotated = rotation_matrix * v;
cout<<"(1,0,0) after rotation = "<<v_rotated.transpose()<<endl;

// 欧拉角：可以将旋转矩阵直接转换成欧拉角
Eigen::Vector3d euler_angles = rotation_matrix.eulerAngles ( 2,1,0 ); // ZYX 顺序，即 yaw pitch roll
顺序
cout<<"yaw pitch roll = "<<euler_angles.transpose()<<endl;

// 欧氏变换矩阵使用 Eigen::Isometry
Eigen::Isometry3d T=Eigen::Isometry3d::Identity(); //
T.rotate ( rotation_vector ); //
T.pretranslate ( Eigen::Vector3d ( 1,3,4 ) ); //
cout << "Transform matrix = \n" << T.matrix() <<endl;

// 用变换矩阵进行坐标变换
Eigen::Vector3d v_transformed = T*v; //
cout<<"v tranformed = "<<v_transformed.transpose()<<endl;
```

虽然称为 *3d*，实质上是 4×4 的矩阵
按照 *rotation_vector* 进行旋转
把平移向量设成 $(1,3,4)$

相当于 $R \cdot v + t$

```
// 四元数
// 可以直接把 AngleAxis 赋值给四元数，反之亦然
Eigen::Quaterniond q = Eigen::Quaterniond ( rotation_vector );
cout<<"quaternion = \n"<<q.coeffs() <<endl; // 请注意 coeffs 的顺序是 (x,y,z,w), w 为实部，前三者为虚部
// 也可以把旋转矩阵赋给它
q = Eigen::Quaterniond ( rotation_matrix );
cout<<"quaternion = \n"<<q.coeffs() <<endl;
// 使用四元数旋转一个向量，使用重载的乘法即可
v_rotated = q*v; // 注意数学上是  $qvq^{-1}$ 
cout<<"(1,0,0) after rotation = "<<v_rotated.transpose()<<endl;

return 0;
}
```



如何对程序进行
编译、运行？

开源库 Eigen 实践

- ❖ linux 简介
- ❖ Linux组成
- ❖ 系统目录结构
- ❖ 程序编辑与编译
- ❖ Cmake

Linux简介

Linux is a family of free and open-source software operating systems built around the Linux kernel.



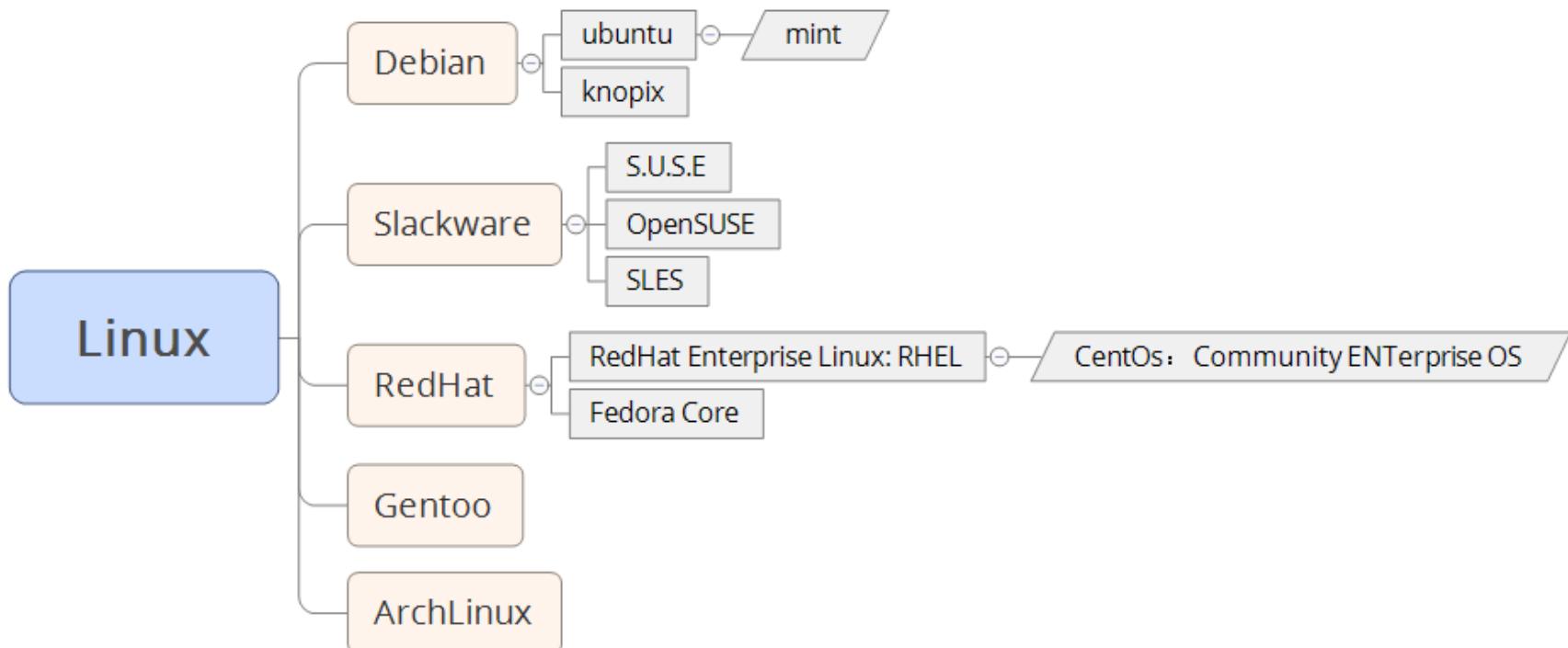
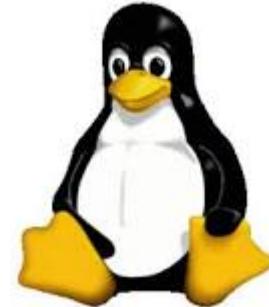
通常我们使用的 linux 系统是一个集 linux 内核、工具集、各种库、桌面管理器、应用程序等一体的一个发布包 (发行版)

从技术上说 linux 是一个内核



Linux简介

Linux is a family of free and open-source software operating systems built around the Linux kernel.



Linux简介

Linux is a family of free and open-source software operating systems built around the Linux kernel.



The image shows a screenshot of the Ubuntu 16.10 desktop environment. The desktop has a purple gradient background. On the left side, there is a vertical dock containing 15 icons: Dash, Examples, Softpedia, Terminal, Dash, Examples, Dash, Dash, Dash, Dash, Dash, Dash, Dash, Dash, Dash.

A terminal window titled "ubuntu@ubuntu: ~" is open in the center. The window contains the following text:

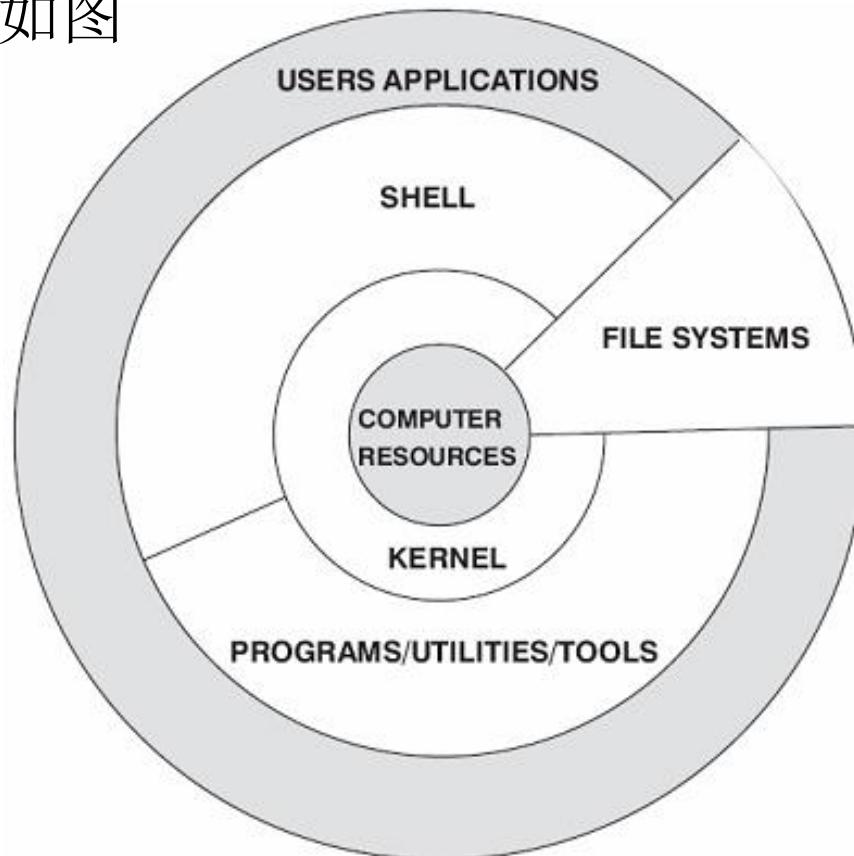
```
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ubuntu:~$ uname -a
Linux ubuntu 4.4.0-21-generic #37-Ubuntu SMP Mon Apr 18 18:33:37 UTC 2016 x86_64
x86_64 x86_64 GNU/Linux
ubuntu@ubuntu:~$ cat /etc/issue
Ubuntu Yakkety Yak (development branch) \n \l

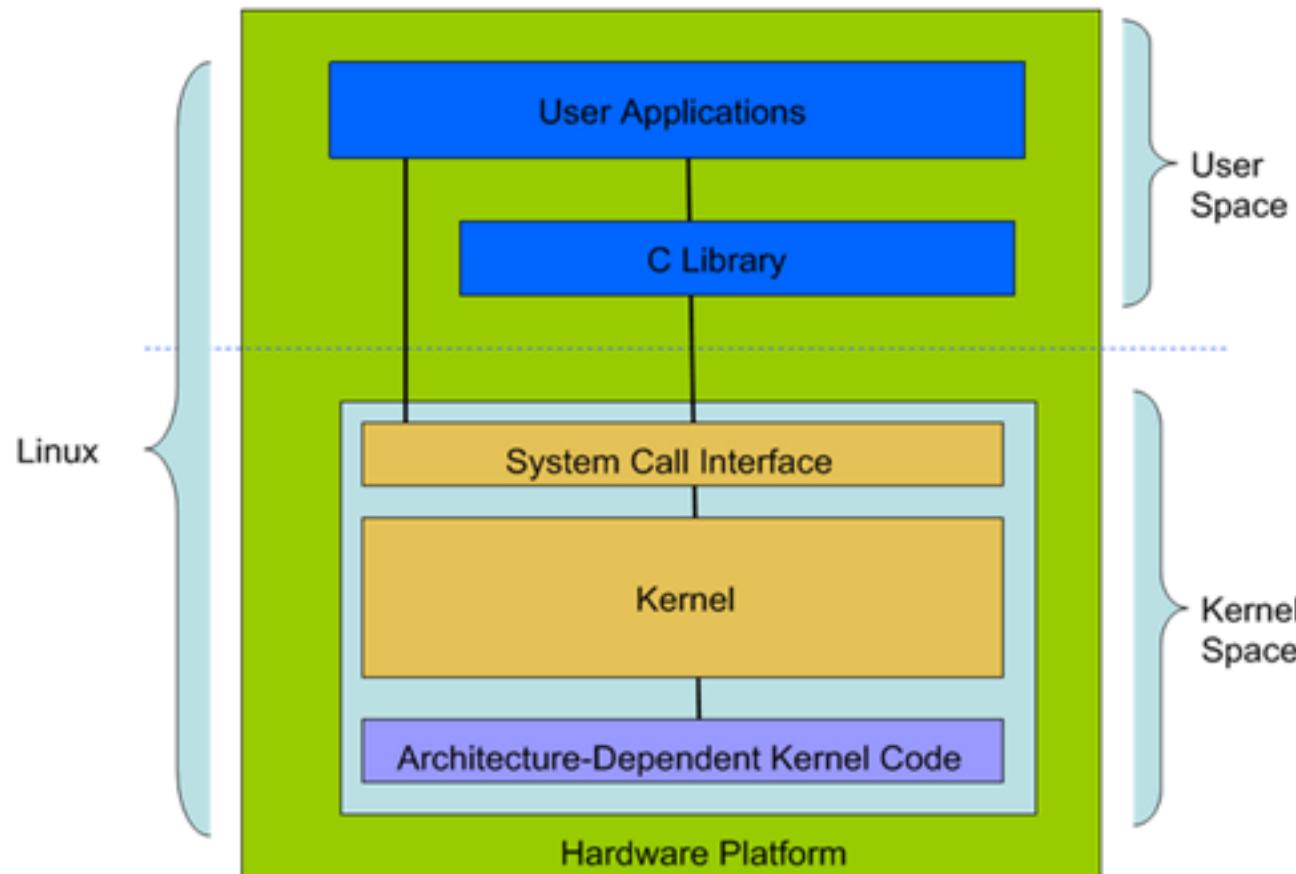
ubuntu@ubuntu:~$ cat /etc/lsb-release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=16.10
DISTRIB_CODENAME=yakkety
DISTRIB_DESCRIPTION="Ubuntu Yakkety Yak (development branch)"
ubuntu@ubuntu:~$ █
```

完整linux体系结构

完整的Linux系统有4个主要部分：内核、shell、文件系统和应用程序。前三者一起形成了基本的操作系统结构，使得用户可以运行程序、管理文件并使用系统。层次结构如图

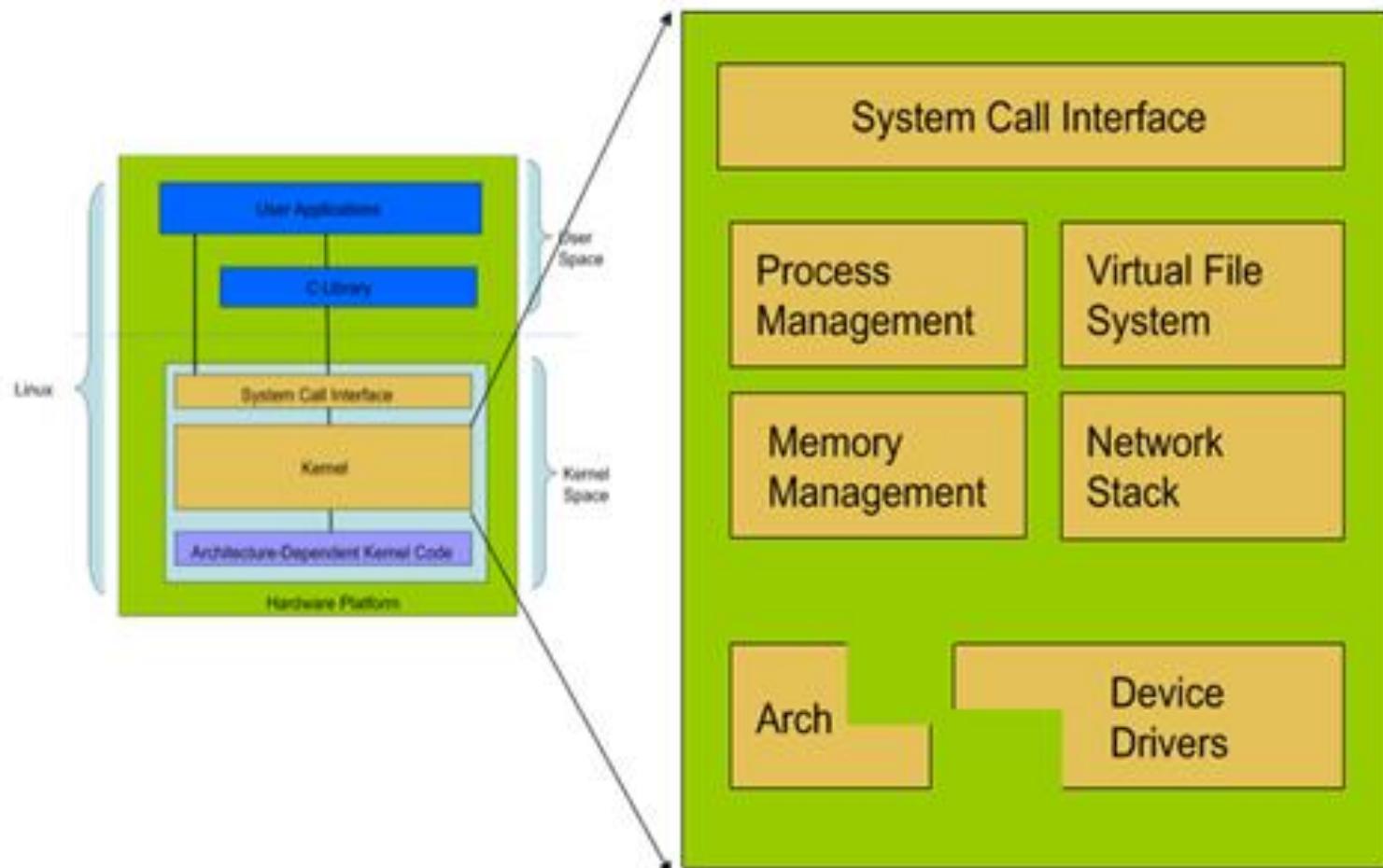


从运行空间看linux体系结构



linux内核组成

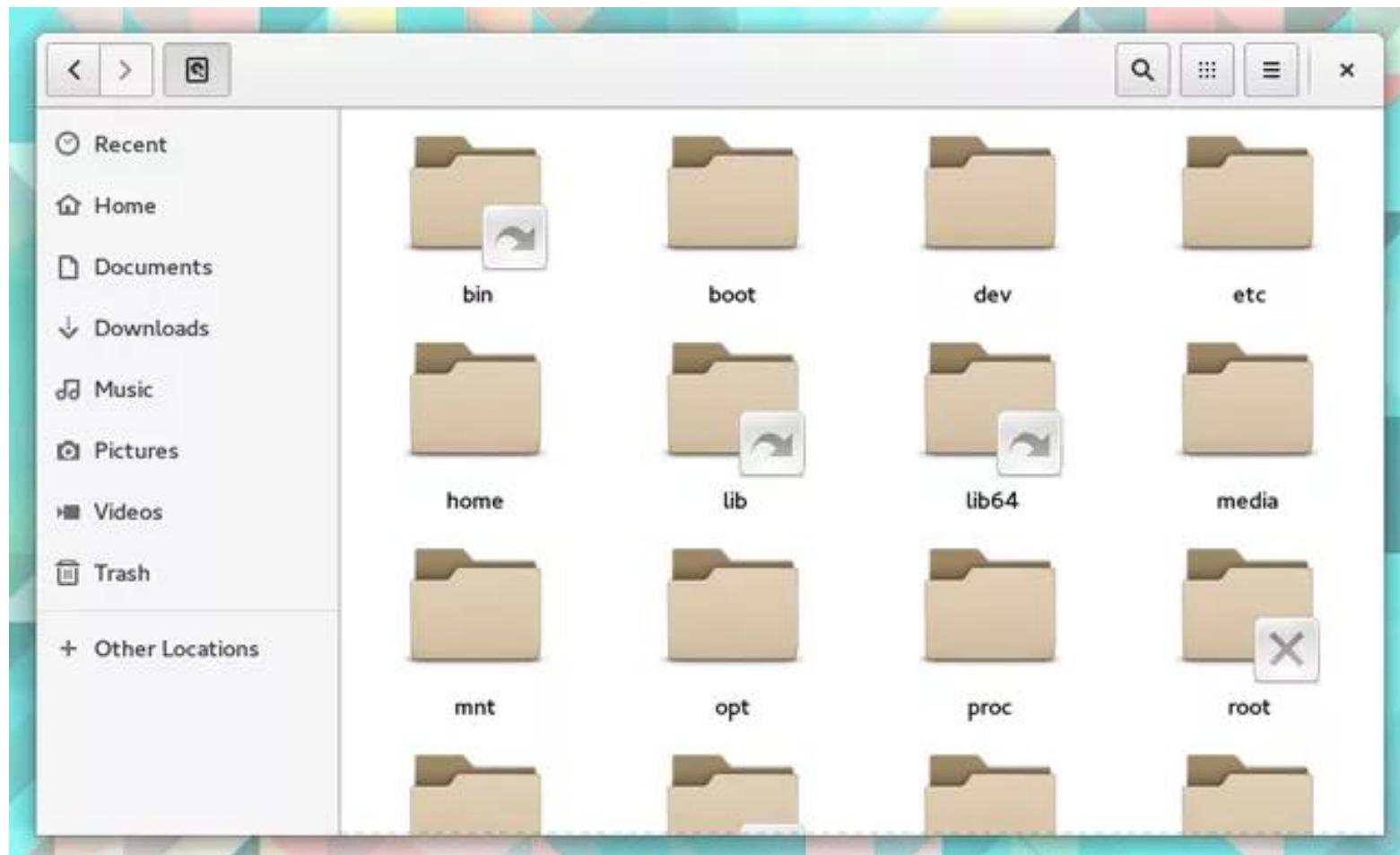
具体来说，linux内核包括如图所示 7个部分



Linux的版本

- ❖ 通常我们使用的 Linux 系统是一个集 Linux 内核、工具集、各种库、桌面管理器、应用程序等一体的一个发布包（发行版）
- ❖ 内核版本-----主版本号.次版本号.修订次数
 - 奇数版本---开发版本
⑩2.5.1
 - 偶数版本---稳定版本
⑩2.6.1
 - www.kernel.org

linux系统目录结构



linux系统目录结构

- 在linux中，目录是一个层次（或树状结构），根是所有目录的起始点，根目录主要有以下子目录
- **/bin**: 包含二进制文件，即可执行程序，这些程序是系统必需的文件
- **/sbin**: 也用于存储二进制文件，只有超级用户**root**才可以使用
- **/etc**: 存放配置文件，如**passwd**,**inittab**等
- **/boot**: 系统引导时使用的文件，系统中非常重要的内核**vmlinu**x就放在该目录下
- **/dev**: 存放设备文件，用户可以通过这些文件访问外部设备
- **/lib**: 存放程序运行时所需要的库文件
- **/temp**: 存放各种临时文件
- **/mnt**: 安装软盘，光盘，u盘的挂载点
- **/root**: 超级用户的个人主目录
- **/usr**: 该目录的空间比较大，用于安装各种应用程序
- **/proc**: 是一个虚拟目录，存放当前内存的映像，由内核自动产生
- **/var**: 存放一些会随时改变的文件

Linux/Unix操作系统目录结构的来历

举例来说，根目录下面有一个子目录/bin，用于存放二进制程序。但是，/usr子目录下面还有/usr/bin，以及/usr/local/bin，也用于存放二进制程序；某些系统甚至还有/opt/bin。它们有何区别？

原来Unix目录结构是历史造成的。

话说1969年，Ken Thompson和Dennis Ritchie在小型机PDP-7上发明了Unix。1971年，他们将主机升级到了PDP-11。

当时，他们使用一种叫做RK05的储存盘，一盘的容量大约是1.5MB。

没过多久，操作系统(根目录)变得越来越大，一块盘已经装不下了。于是，他们加上了第二块RK05，并且规定第一块盘专门放系统程序，第二块盘专门放用户自己的程序，因此挂载的目录点取名为/usr。



Linux/Unix操作系统目录结构的来历

也就是说，根目录"/"挂载在第一块盘，"**/usr**"目录挂载在第二块盘。除此之外，两块盘的目录结构完全相同，第一块盘的目录(/bin, /sbin, /lib, /tmp...)都在**/usr**目录下重新出现一次。

后来，第二块盘也满了，他们只好又加了第三盘RK05，挂载的目录点取名为**/home**，并且规定**/usr**用于存放用户的程序，**/home**用于存放用户的数据。

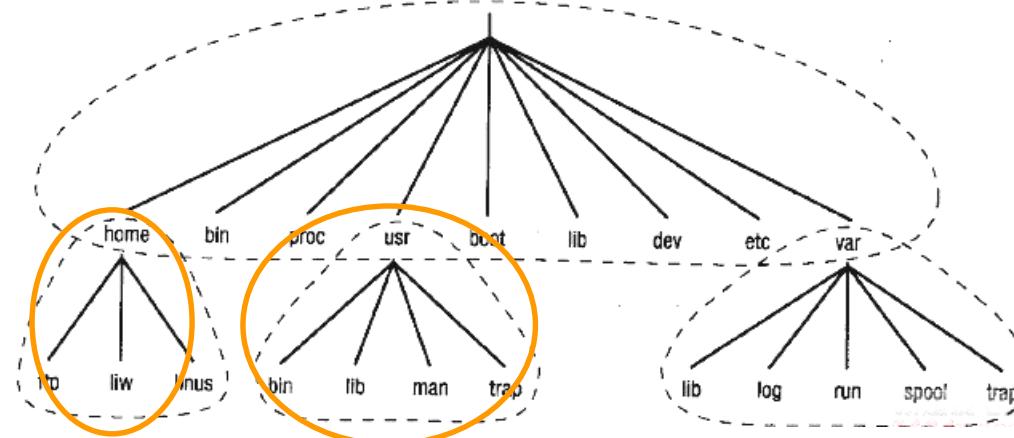
从此，这种目录结构就延续了下来。随着硬盘容量越来越大，各个目录的含义进一步得到明确。

/: 存放系统程序，也就是At&t开发的Unix程序。

/usr: 存放Unix系统商(unix software resource 比如IBM和HP)开发的程序。

/usr/local: 存放用户自己安装的程序。

/opt: 在某些系统，用于存放第三方厂商开发的程序，所以取名为option，意为"选装"。



常用linux命令

文件命令

ls – 列出目录
ls -al – 使用格式化列出隐藏文件
cd dir - 更改目录到 *dir*
cd – 更改到 home 目录
pwd – 显示当前目录
mkdir dir – 创建目录 *dir*
rm file – 删除 *file*
rm -r dir – 删除目录 *dir*
rm -f file – 强制删除 *file*
rm -rf dir – 强制删除目录 *dir**
cp file1 file2 – 将 *file1* 复制到 *file2*
cp -r dir1 dir2 – 将 *dir1* 复制到 *dir2*; 如果 *dir2* 不存在则创建它
mv file1 file2 – 将 *file1* 重命名或移动到 *file2*; 如果 *file2* 是一个存在的目录则将 *file1* 移动到目录 *file2* 中
ln -s file link – 创建 *file* 的符号连接 *link*
touch file – 创建 *file*
cat > file – 将标准输入添加到 *file*
more file – 查看 *file* 的内容
head file – 查看 *file* 的前 10 行
tail file – 查看 *file* 的后 10 行
tail -f file – 从后 10 行开始查看 *file* 的内容

文件权限

chmod octal file – 更改 *file* 的权限

- 4 – 读 (r)
- 2 – 写 (w)
- 1 – 执行 (x)

示例:

chmod 777 – 为所有用户添加读、写、执行权限
chmod 755 – 为所有者添加 rwx 权限, 为组和其他用户添加 rx 权限
更多选项参阅 **man chmod**.

进程管理

ps – 显示当前的活动进程
top – 显示所有正在运行的进程
kill pid – 杀掉进程 id *pid*
killall proc – 杀掉所有名为 *proc* 的进程 *
bg – 列出已停止或后台的作业
fg – 将最近的作业带到前台
fg n – 将作业 *n* 带到前台

常用linux命令

系统信息

date – 显示当前日期和时间

cal – 显示当月的日历

uptime – 显示系统从开机到现在所运行的时间

w – 显示登录的用户

whoami – 查看你的当前用户名

finger user – 显示 *user* 的相关信息

uname -a – 显示内核信息

cat /proc/cpuinfo – 查看 cpu 信息

cat /proc/meminfo – 查看内存信息

man command – 显示 *command* 的说明手册

df – 显示磁盘占用情况

du – 显示目录空间占用情况

free – 显示内存及交换区占用情况

压缩

tar cf file.tar files – 创建包含 *files* 的 tar 文件
file.tar

tar xf file.tar – 从 *file.tar* 提取文件

tar czf file.tar.gz files – 使用 Gzip 压缩创建 tar 文件

tar xzf file.tar.gz – 使用 Gzip 提取 tar 文件

tar cjf file.tar.bz2 – 使用 Bzip2 压缩创建 tar 文件

tar xjf file.tar.bz2 – 使用 Bzip2 提取 tar 文件

gzip file – 压缩 *file* 并重命名为 *file.gz*

gzip -d file.gz – 将 *file.gz* 解压缩为 *file*

常用linux命令

SSH

ssh user@host – 以 *user* 用户身份连接到 *host*

ssh -p port user@host – 在端口 *port* 以 *user* 用户身份连接到 *host*

ssh-copy-id user@host – 将密钥添加到 *host* 以实现无密码登录

搜索

grep pattern files – 搜索 *files* 中匹配 *pattern* 的内容

grep -r pattern dir – 递归搜索 *dir* 中匹配 *pattern* 的内容

command | grep pattern – 搜索 *command* 输出中匹配 *pattern* 的内容

安装

从源代码安装:

./configure

make

make install

dpkg -i pkg.deb – 安装包 (Debian)

rpm -Uvh pkg.rpm – 安装包 (RPM)

网络

ping host – ping *host* 并输出结果

whois domain – 获取 *domain* 的 whois 信息

dig domain – 获取 *domain* 的 DNS 信息

dig -x host – 逆向查询 *host*

wget file – 下载 *file*

wget -c file – 断点续传

快捷键

Ctrl+C – 停止当前命令

Ctrl+Z – 停止当前命令，并使用 **fg** 恢复

Ctrl+D – 注销当前会话，与 **exit** 相似

Ctrl+W – 删除当前行中的字

Ctrl+U – 删除整行

!! – 重复上次的命令

exit – 注销当前会话

* 小心使用。

常用linux命令

✓ Linux基本操作

提示符 "\$" 为普通用户, "#" 为超级用户

linux常用的编辑工具有 nano , vi/vim (vim是vi的增强版)等

建议首先删除默认vi编辑器

`sudo apt-get remove vim-common`

然后重装vim

`sudo apt-get install vim`

开源库 Eigen 实践

- ❖ linux 简介
- ❖ Linux组成
- ❖ 系统目录结构
- ❖ 程序编辑与编译
- ❖ Cmake

linux程序编辑

编辑器

➤ Emacs和Vim

➤ Gedit

括号匹配、自动缩进以及为包括C, C++, Java, HTML, XML, Python, Perl, 以及许多其它编程语言进行语法高亮

➤ Kate

➤ Sublime Text

支持多种编程语言如C#, D, Dylan, Erlang, Groovy, Haskell, Lisp, Lua, MATLAB, OCaml, R, 甚至 SQL 语法高亮
自动补全, 全部源代码的滚动预览图

➤ UltraEdit

语法高亮、代码折叠以及拥有项目管理特性

linux程序编辑

✓ Vi编辑器 shell下输入 **vimtutor**

工作模式

- ✓ 正常模式：可以使用快捷键命令，或按:输入命令行。
- ✓ 插入模式：可以输入文本，在正常模式下，**按i、a、o等**都可以进入插入模式。
- ✓ 可视模式：正常模式下**按v**可以进入可视模式，在可视模式下，移动光标可以选择文本。**按V**进入**可视行**模式，总是整行整行的选中。**ctrl+v**进入可视块模式。
- ✓ 替换模式：正常模式下，按R进入

光标移动：**h i j k**, **ctrl+f**: 下翻屏 ,**ctrl+b**: 上翻屏

查找与替换 **/something:** **?something:** **n:** **N:**

切换窗口：**ctrl+w** 切换到下一个窗口

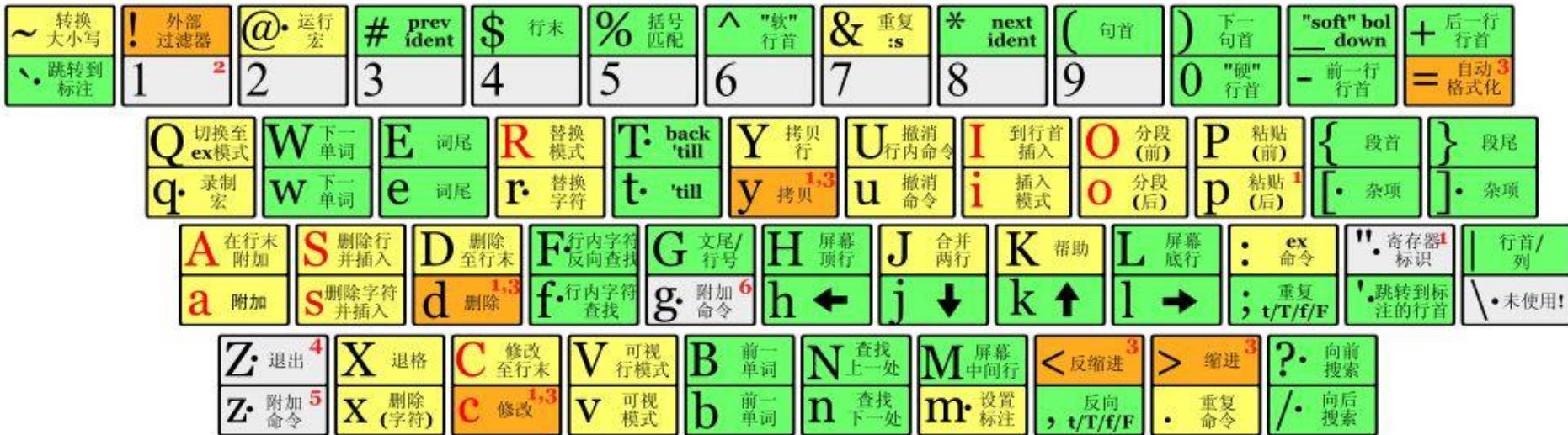
撤销与重做：**u**: 取消一个改动 **ctrl + r**: 重做最后的改动

文档操作：**:wq / ZZ / :x** -- 保存并退出

vi / vim 键盘图

ESC

命令
模式



动作 移动光标, 或者定义操作的范围

命令 直接执行的命令,
红色命令进入编辑模式

操作 后面跟随表示操作范围的指令

extra 特殊功能,
需要额外的输入

q· 后跟字符参数

w,e,b命令

小写(b): `quux([foo], bar, baz);`
大写(B): `quux(foo, bar, baz);`

主要ex命令:

:w (保存), :q (退出), :q! (不保存退出)
:e f (打开文件 f),
:%s/x/y/g ('y' 全局替换 'x'),
:h (帮助 in vim), :new (新建文件 in vim)

其它重要命令:

CTRL-R: 重复 (vim),
CTRL-F/-B: 上翻/下翻,
CTRL-E/-Y: 上滚/下滚,
CTRL-V: 块可视模式 (vim only)

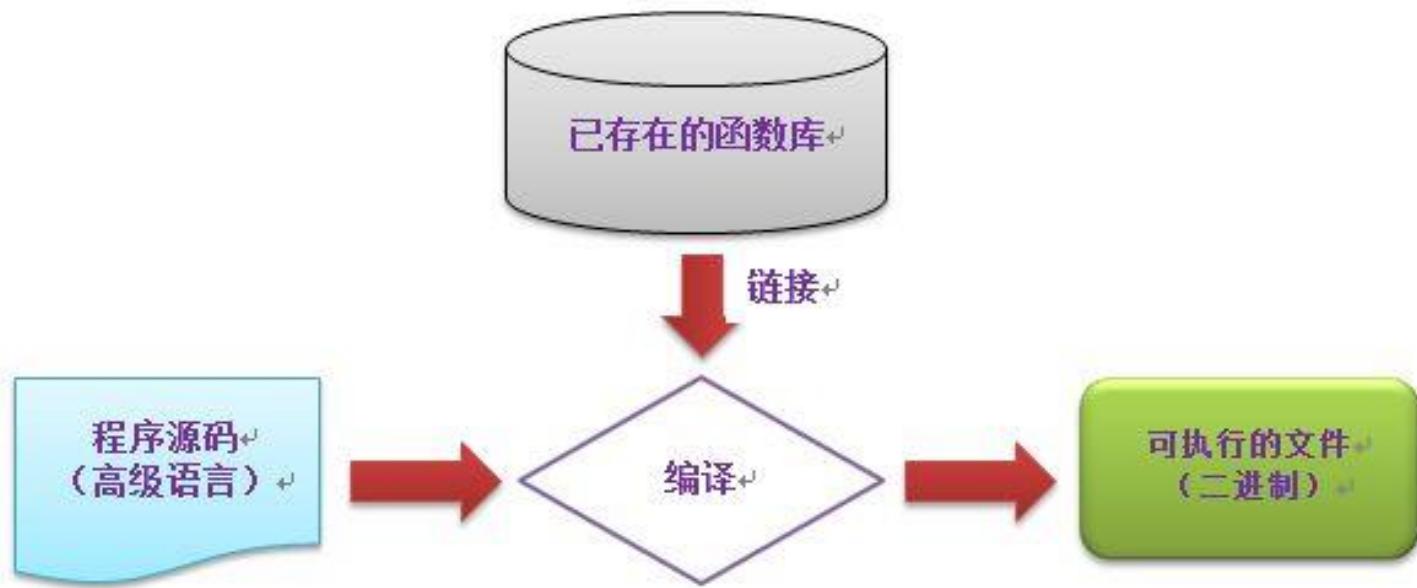
可视模式:

漫游后对选中的区域执行操作 (vim only)

备注:

- (1) 在 拷贝/粘贴/删除 命令前使用 "x (x=a..z,*)
使用命令的寄存器('剪贴板')
(如: "ay\$ 拷贝剩余的行内容至寄存器 'a')
- (2) 命令前添加数字
多遍重复操作
(e.g.: 2p, d2w, 5i, d4j)
- (3) 重复本字符在光标所在行执行操作
(dd = 删除本行, >> = 行首缩进)
- (4) ZZ 保存退出, ZQ 不保存退出
- (5) zt: 移动光标所在行至屏幕顶端,
zb: 底端, zz: 中间
- (6) gg: 文首 (vim only),
gf: 打开光标处的文件名

Linux 程序编译



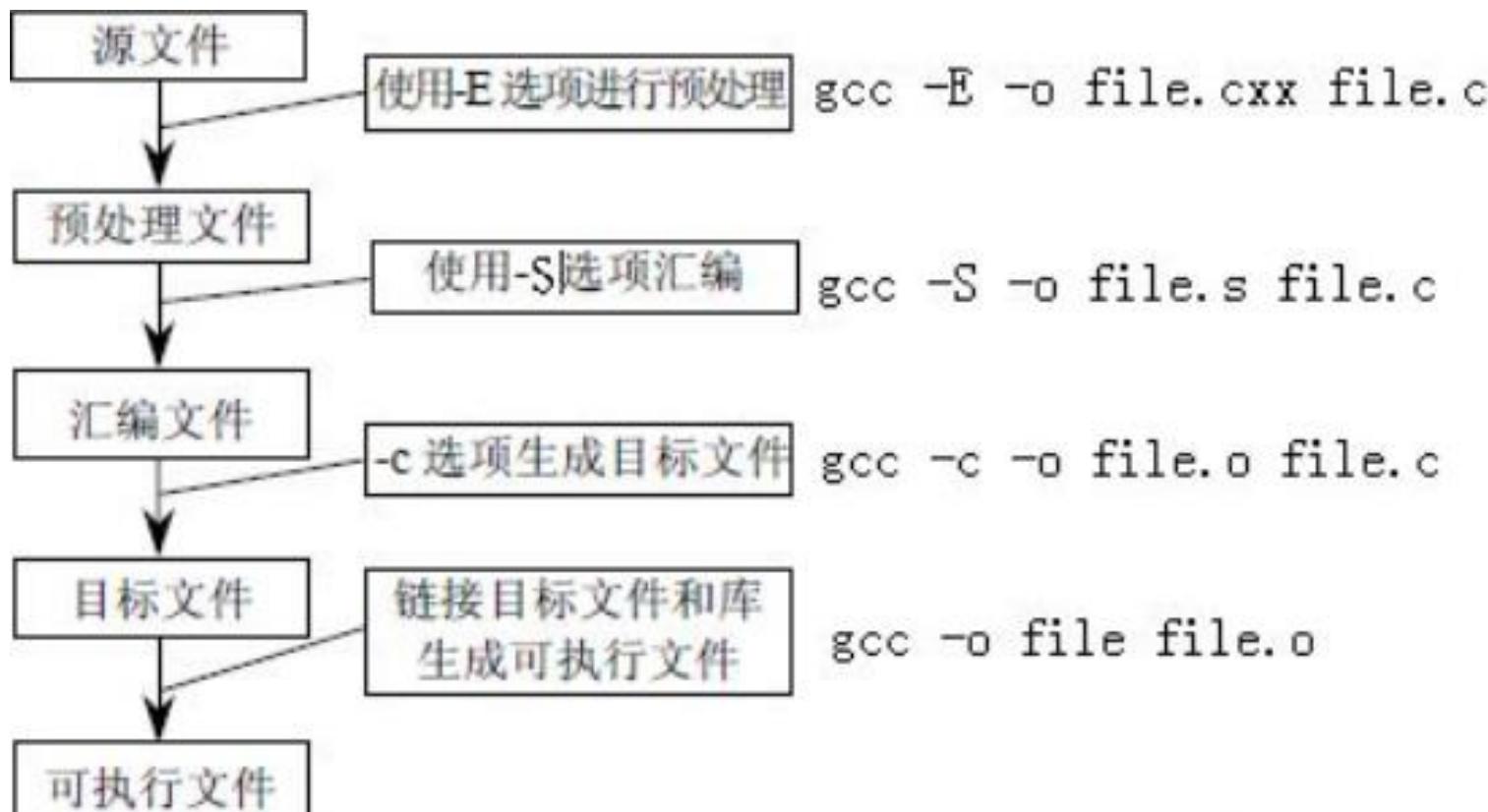
GCC:GNU Compiler Collection(GUN 编译器集合), 它可以编译C、C++、JAVA、Fortran、Pascal、Object-C、Ada等语言。

gcc是GCC中的GUN C Compiler(C 编译器)

g++是GCC中的GUN C++ Compiler(C++编译器)

Linux 程序编译

ubuntu下自带gcc编译器。可以通过“`gcc -v`”命令来查看是否安装



g++

❖ gcc和g++的主要区别

1. 对于 *.c 和 *.cpp 文件, gcc 分别当做 c 和 cpp 文件编译 (c 和 cpp 的语法强度不一样)
2. 对于 *.c 和 *.cpp 文件, g++ 则统一当做 cpp 文件编译
3. 使用 g++ 编译文件时, g++ 会自动链接标准库 STL, 而 gcc 不会自动链接 STL
4. gcc 在编译 C 文件时, 可使用的预定义宏是比较少的

g++

- ❖ 安装g++编译器，可以通过命令“`sudo apt-get install build-essential`”实现。
- ❖ 该命令可完成`gcc`,`g++`,`make`的安装，`build-essential`是一整套工具，包含`gcc`, `libc`等等。
- ❖ 通过“`g++ -v`”可以查看g++是否安装成功

```
//安装g++编译器
```

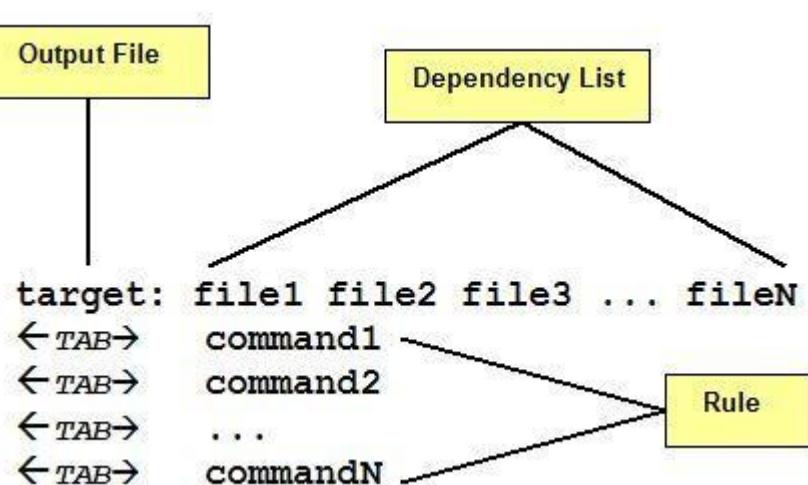
```
sudo apt-get install build-essential
```

```
//修复g++编译器
```

```
sudo apt-get install build-essential --fix-missing
```

Linux下程序编辑与编译

gcc与g++编译, 特别是多个文件时, make都依赖Makefile



A screenshot of a terminal window showing a Makefile. The menu bar includes **文件(F)**, **编辑(E)**, **查看(V)**, **搜索(S)**, **终端(T)**, and **帮助(H)**. The terminal content shows the following Makefile rules:

```
hello:hello.o
    gcc hello.o -o hello
hello.o:hello.c
    gcc -c hello.c
clean:
    rm -f *.o *~ hello
~
```

Linux下程序编辑与编译

cmake全称为“cross platform make”,是一个开源的跨平台自动化构建系统,是一种常用、方便的,用于组织Linux下C++程序的工具。有许多库,例如OpenCV、g2o、Ceres等,都用cmake组织它们的工程

Ubuntu一般自带cmake,安装方法

```
//cmake-*.**tar.gz为下载下来的源码包  
tar xvf cmake-*.**.tar.gz  
cd cmake-*.**  
./bootstrap  
make  
make install
```

Linux下程序编辑与编译

官方例程 tutorial.cxx

```
1 // A simple program that computes the square root of a number
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <math.h>
5 int main (int argc, char *argv[])
6 {
7     if (argc < 2)
8     {
9         fprintf(stdout,"Usage: %s number\n",argv[0]);
10    return 1;
11 }
12 double inputValue = atof(argv[1]);
13 double outputValue = sqrt(inputValue);
14 fprintf(stdout,"The square root of %g is %g\n",
15         inputValue, outputValue);
16 return 0;
17 }
```

Linux下程序编辑与编译

Cmake 与 CMakeLists.txt

```
PROJECT(projectname [CXX] [C] [Java])
ADD_EXECUTABLE(hello ${SRC_LIST})
```

```
cmake_minimum_required(VERSION 2.6)
project(Tutorial)
add_executable(Tutorial tutorial.cxx)
```

```
cmake directory
make
```

Linux下程序编辑与编译

CMakeLists.txt 添加版本号和配置头文件

```
1 cmake_minimum_required(VERSION 2.6)
2 project(Tutorial)
3 # The version number.
4 set(Tutorial_VERSION_MAJOR 1)
5 set(Tutorial_VERSION_MINOR 0)
6
7 # configure a header file to pass some of the CMake settings
8 # to the source code
9 configure_file(
10     "${PROJECT_SOURCE_DIR}/TutorialConfig.h.in"
11     "${PROJECT_BINARY_DIR}/TutorialConfig.h"
12 )
13
14 # add the binary tree to the search path for include files
15 # so that we will find TutorialConfig.h
16 include_directories("${PROJECT_BINARY_DIR}")
17
18 # add the executable
19 add_executable(Tutorial tutorial.cxx)
```

Linux下程序编辑与编译

CMakeLists.txt 中声明 编译并添加一个库

```
SET(LIBTUTORIAL_SRC tutorial.hxx)
```

```
ADD_LIBRARY(tutorial SHARED ${LIBTUTORIAL_SRC})
```

```
cmake directory  
make
```

Linux下程序编辑与编译

Eigen C++ 开源线性代数库。它提供了快速的有关矩阵的线性代数运算，用于描述空间变换的各类型数据结构，还 包括解方程等功能。

许多上层的软件库也使用 Eigen 进行矩阵运算，包括 g2o、Sophus，还有ORBSLAM等经典例程包括

安装 Eigen并查找路径

```
sudo apt-get install libeigen3-dev
```

```
sudo updatedb  
locate eigen3
```

```
#include <iostream>
#include <cmath>
using namespace std;

#include <Eigen/Core>
// Eigen 几何模块
#include <Eigen/Geometry>

/******************
* 本程序演示了 Eigen 几何模块的使用方法
******************/

int main( int argc, char** argv )
{
    // Eigen/Geometry 模块提供了各种旋转和平移的表示
    // 3D 旋转矩阵直接使用 Matrix3d 或 Matrix3f
    Eigen::Matrix3d rotation_matrix = Eigen::Matrix3d::Identity();
    // 旋转向量使用 AngleAxis，它底层不直接是 Matrix，但运算可以当作矩阵（因为重载了运算符）
    Eigen::AngleAxisd rotation_vector ( M_PI/4, Eigen::Vector3d ( 0,0,1 ) ); // 沿 Z 轴旋转 45 度
    cout .precision(3);
    cout<<"rotation matrix =\n"<<rotation_vector.matrix() <<endl; //用      matrix() 转换成矩阵
    // 也可以直接赋值
    rotation_matrix = rotation_vector.toRotationMatrix();
    // 用 AngleAxis 可以进行坐标变换
    Eigen::Vector3d v ( 1,0,0 );
    Eigen::Vector3d v_rotated = rotation_vector * v;
    cout<<"(1,0,0) after rotation = "<<v_rotated.transpose()<<endl;
```

```
// 或者用旋转矩阵
v_rotated = rotation_matrix * v;
cout<<"(1,0,0) after rotation = "<<v_rotated.transpose()<<endl;

// 欧拉角：可以将旋转矩阵直接转换成欧拉角
Eigen::Vector3d euler_angles = rotation_matrix.eulerAngles ( 2,1,0 ); // ZYX 顺序，即 yaw pitch roll
顺序
cout<<"yaw pitch roll = "<<euler_angles.transpose()<<endl;

// 欧氏变换矩阵使用 Eigen::Isometry
Eigen::Isometry3d T=Eigen::Isometry3d::Identity(); //
T.rotate ( rotation_vector ); //
T.pretranslate ( Eigen::Vector3d ( 1,3,4 ) ); //
cout << "Transform matrix = \n" << T.matrix() <<endl;

// 用变换矩阵进行坐标变换
Eigen::Vector3d v_transformed = T*v; //
cout<<"v tranformed = "<<v_transformed.transpose()<<endl;
```

虽然称为 $3d$ ，实质上是 4×4 的矩阵
按照 $rotation_vector$ 进行旋转
把平移向量设成 $(1,3,4)$

相当于 $R \cdot v + t$

```
// 四元数
// 可以直接把 AngleAxis 赋值给四元数，反之亦然
Eigen::Quaterniond q = Eigen::Quaterniond ( rotation_vector );
cout<<"quaternion = \n"<<q.coeffs() <<endl; // 请注意 coeffs 的顺序是 (x,y,z,w), w 为实部，前三者为虚部
// 也可以把旋转矩阵赋给它
q = Eigen::Quaterniond ( rotation_matrix );
cout<<"quaternion = \n"<<q.coeffs() <<endl;
// 使用四元数旋转一个向量，使用重载的乘法即可
v_rotated = q*v; // 注意数学上是  $qvq^{-1}$ 
cout<<"(1,0,0) after rotation = "<<v_rotated.transpose()<<endl;

return 0;
}
```



如何对程序进行
编译、运行？

homework1

- ❖ 安装Ubuntu14.04或以上版本的linux系统并熟悉基本操作；
- ❖ 书写一个由 `cmake` 组织的 `C++` 工程，并书写 `CMakeLists.txt`, 要求如下：
 1. 参考示例代码, 编译 `include/hello.h` 和 `src/hello.c` 构成 `libhello.so` 库。其中 `hello.c` 中提供一个函数 `sayHello()`, 调用此函数时往屏幕输出一行“Hello”
 2. 文件 `useHello.c` 中含有一个 `main` 函数, 它可以编译成一个可执行文件, 名为“`sayhello`”。
 3. 默认用 `Release` 模式编译这个工程。
 4. 支持使用命令 `sudo make install`, 该命令将 `hello.h` 放至 `/usr/local/include/` 下, 将 `libhello.so` 放至 `/usr/local/lib/` 下
- ❖ 运行参考书《视觉SLAM十四讲》P45-47的 `eigenMatrix.cpp` 程序并熟悉代码；

Thanks