**密级**：

数学

|  |  |  |  |
| --- | --- | --- | --- |
| **版本** | **作者** | **时间** | **操作** |
| 0.1 | 洪智标 | 2011-3-29 | 初稿 |
|  |  |  |  |

目录

[1 引言 4](#_Toc289180533)

[1.1 编写目的 4](#_Toc289180534)

[1.2 项目背景 4](#_Toc289180535)

[1.3 定义 4](#_Toc289180536)

[1.4 参考资料 4](#_Toc289180537)

[2 总体要求 4](#_Toc289180538)

[2.1 总体功能要求 4](#_Toc289180539)

[2.2 软件平台要求 4](#_Toc289180540)

[3 软件开发 5](#_Toc289180541)

[3.1 需求概述 5](#_Toc289180542)

[3.2 概要设计 5](#_Toc289180543)

[3.3 详细设计 5](#_Toc289180544)

[3.4 编码实现 5](#_Toc289180545)

[3.4.1 编码规范 5](#_Toc289180546)

[3.4.2 XX模块 5](#_Toc289180547)

[3.5 软件测试 5](#_Toc289180548)

[3.6 软件的交付准备 6](#_Toc289180549)

[3.7 软件的鉴定验收 6](#_Toc289180550)

# 引言

## 编写目的

阐明编写详细设计说明书的目的，指明读者对象。

本文档用于指导软件开发者为XXXX应用进行设计和开发的过程，通过规范软件项目承担单位的开发过程达到提高软件质量，降低维护成本的目的。

## 项目背景

应包括项目的来源和主管部门等。

## 定义

列出本文档中所用到的专门术语的定义和缩写词的愿意。

## 参考资料

● 列出有关资料的作者、标题、编号、发表日期、出版单位或资料来源

●项目经核准的计划任务书、合同或上级机关的批文；项目开发计划；需求规格说明书；概要设计说明书；测试计划（初稿）；用户操作手册

● 文档所引用的资料、软件开发的标准或规范。

建筑的永恒之道

<http://hi.baidu.com/timeless/blog/index/0>

# 概率

## 排列



## 组合



# 空间几何

## 向量

**投影**



两向量的数量积等于其中一个向量的模和另一个向量在这向量的方向上的投影的乘积。

**数量积**

向量数量积的运算规律

1.交换律：α·β=β·α

2.分配律：(α+β)·γ=α·γ+β·γ

3.若λ为数：(λα)·β=λ(α·β)=α·(λβ) 　　若λ、μ为数：：(λα)·(μβ)=λμ(α·β)

4.α·α=|α|^2 ，此外：α·α=0〈=〉α=0。

向量的数量积不满足消去律，即一般情况下：α·β=α·γ，α≠0 ≠〉β=γ。

向量的数量积不满足结合律，即一般（α·β)·γ ≠〉α·（β·γ）

相互垂直的两向量数量积为0

## 直线

如果一个非零向量平行于一条已知直线，这个向量就叫做这条直线的方向向量。

由于过空间一点可作而且只能作一条直线平行于一已知直线，所以当直线L上一点和它的一方向向量=(m,n,p)为已知时，直线L的位置就完全确定了。

设点M(x,y,z)是直线L上的任一点，那么向量与L的方向向量平行。所以两向量的对应坐标成比例。故有：



反过来，如果点M不在直线L上，那么由于与不平行，这两向量的对应坐标就不成比例。因此此方程叫做**直线的对称式方程**或**点向式方程**。

由直线的对称式方程容易导出直线的参数方程。如设



那么



就是**直线的参数方程**。

**点在直线上的投影**

ＰＳ：如果使用初中高中的方法，公式复杂易错。使用向量求解问题，便捷易懂。

1、首先假设已知直线上两点P1、P2、以及直线外一点P3。

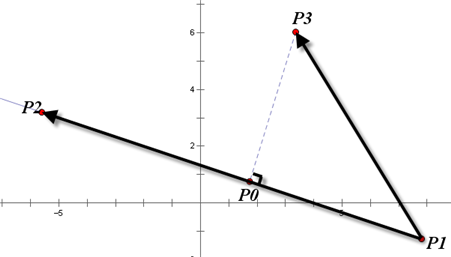
2、令投影点为P0。

3、因为P0、P1、P2都在同一条直线上，所以可得k \*（P2 - P1） = P0 - P1

k = |P0-P1|/|P2-P1|。 只要求出比例因子k，便可求出P0的值。

4、令v1 = P3 - P1 ， v2 = P2 - P1，v1与v2进行点乘得：v1\*v2=cos(seta)|P3-P1||P2-P1|=|P0-P1|\*|P2-P1|，于是

k = |P0-P1|/|P2-P1| = ( (v1\*v2)/|P2-P1| ) / |P2-P1| = (P3 - P1) \* (P2 - P1) / (|P2 - P1| \* |P2 - P1|)



## 平面

**平面的点法式方程**

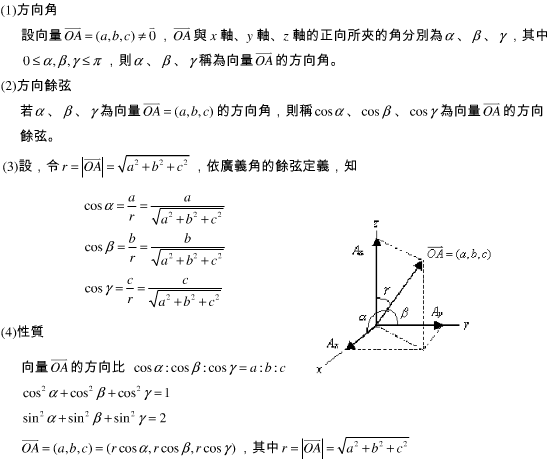
如果一非零向量垂直于一平面，这向量就叫做该平面的法线向量。

已知平面上一点和它的法向量=(A,B,C)，设M(x,y,z)为平面上的任一点，有



此方程称为**平面的点法式方程**

## 方向余弦



<http://wenku.baidu.com/view/4840e9d076eeaeaad1f330c5.html>

# 三角函数



# 矩阵

## 定义

定义 m行n列矩阵为m╳n矩阵；

行数与列数都等于n的矩阵称为n阶矩阵；

两个矩阵的行数相等、列数相等时，就称它们为同型矩阵；

元素都是零的矩阵称为零矩阵，记作O；

从左上角到右下角的主对角线上的元素都是1，其它元素都是0的矩阵称为单位矩阵；

不在对角线上的元素都是0的矩阵称为对角矩阵，记为

A=diag(λ1, λ2，……，λn)

## n阶行列式

**定义6** 由n阶方阵A的元素所构成的行列式（各元素的位置不变），称为方阵A的行列式，记作|A|或detA。

n阶方阵具有如下性质：

1. |AT|=|A|
2. |λA|=λn|A|
3. |AB|=|A||B|（证明见《线性代数》p40）



其中数aij为行列式D的（i,j）元

**性质2** 互换行列式的两行（列），行列式变号（证明见《线性代数》p10）

**推论** 如果行列式有两行（列）完全相同，则此行列式等于零

证 把这两行互换，有D=-D，故D=0

**性质5** 若行列式的某一列（行）的元素都是两数之和，例如第i列的元素都是两数之和：



则D等于下列两个行列式之和：



在n阶行列式中，把（i，j）元aij所在的第i行和第j列划去后，留下来的n-1阶行列式叫做（i，j）元aij的**余子式**，记作Mij；记

Aij=(-1)i+jMij

Aij叫做(i,j)元aij的**代数余子式**

**引理** 一个n阶行列式，如果其中第i行所有元素除（i,j）元aij外都为零，那么这行列式等于aij与它的代数余子式的乘积，即

D=aijAij（证明见《线性代数》p16）

**定理** 行列式等于它的任一行（列）的各元素与其对应的代数余子式乘积之和，即

D=ai1Ai1+ai2Ai2+……+ainAin(i=1,2,……,n)

或D=a1jA1j+a2jA2j+……+anjAnj(j=1,2,……,n) （证明见《线性代数》p17）

这个定理叫做行列式按行（列）展开法则。

## 矩阵的加法

## 数与矩阵相乘

## 矩阵与矩阵相乘

**定义**：设A=（aij）是一个m╳s矩阵，B=（bij）是一个s╳n矩阵，那么规定矩阵A与矩阵B的乘积是一个m╳n矩阵C=（cij），其中



记作**C=AB**

必须注意：只有当第一个矩阵的列数与第二矩阵的行数相等时，两个矩阵才能相乘。

在矩阵的乘法必须注意矩阵相乘的顺序。AB是A左乘B的乘积，BA是A右乘B的乘积

矩阵的乘法虽不满足交换律，但仍满足下列结合律和分配律（假设运算都是可行的）：

1. （AB）C=A（BC）
2. λ（AB）=（λA）B=A（λB）
3. A（B+C）=AB+AC

对于单位矩阵E，容易验证



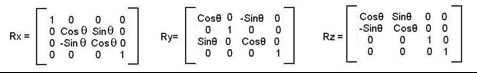
或简写成

EA=AE=A

左乘行变换，右乘列变换，可视，牢记。

### 旋转

旋转在三维里应该是最麻烦的东西，因为人类对旋转的描述是很有限。光靠矩阵控制旋转很多情况是非常复杂和不形象的，所以我们还发明了欧拉角和四元数这些对旋转的控制更优良的办法，这些我们以后会讲到。

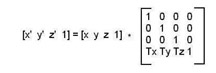


角度θ表示的是延某轴旋转的角度。

下面三个矩阵分别表示了点绕x轴，y轴，z轴的旋转矩阵。

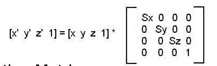
### 移动

从公式我们很容易能看出，第4行1，2，3列的数，分别控制着点在x,y,z方向上的移动。



### 缩放

从公式我们很容易能看出，对角线上的n11,n22,n33，分别控制着点在x,y,z方向上的缩放。



这些就是最简单的变换矩阵，熟悉了这些，我们就又向前迈进了一步。

图形学应该是一个很有趣的东西，并非像很多学校和学生教授的的那种痛苦不堪的东西。其实道理非常简单，如果自己再具备一点编程知识，很容易就能做出自己的成像程序。

## 转置矩阵

**定义**：把矩阵A的行换成同序数的列得到的一个新矩阵，叫做A的转置矩阵，记作AT

如



的转置矩阵为



矩阵的转置也是一种运算，满足下列运算规律（假设运算都是可行的）：

1. （AT）T=A
2. （A+B）T=AT+BT
3. （λA）T=λAT
4. （AB）T=BTAT（公式推导见《线性代数》p39）

## 伴随矩阵

行列式|A|的各个元素的代数余子式Aij所构成的如下的矩阵



称为矩阵A的伴随矩阵，试证

**AA\*=A\*A=|A|E**

（证明见《线性代数》p41，不明）

## 满秩矩阵

矩阵的秩: 用初等行变换将矩阵A化为阶梯形矩阵, 则矩阵中非零行的个数就定义为这个矩阵的秩, 记为r（A）。

满秩矩阵（non-singular matrix）: 设A是n阶矩阵, 若r（A） = n, 则称A为满秩矩阵。

满秩矩阵是一个很重要的概念, 它是判断一个矩阵是否可逆的充分必要条件

## 奇异矩阵（Singular matrix）

也叫不可逆矩阵。

当|A|=0时，A称为奇异矩阵，否则称为非奇异矩阵。

A是可逆矩阵的充分必要条件是|A|≠0，即可逆矩阵就是非奇异矩阵。（见《线性代数》p43）

## 逆矩阵

**定义** 对于n阶矩阵A，如果有一个n阶矩阵B，使

**AB=BA=E**

则说矩阵A是可逆的，并把矩阵B称为A的逆矩阵。

如果矩阵A是可逆的，那么A的逆矩阵是唯一的。

A的逆矩阵记作A-1

**定理1** 若矩阵A可逆，则|A|≠0

**定理2** 若|A|≠0，则矩阵A可逆，且



其中A\*为矩阵A的伴随矩阵

综上，A是可逆矩阵的充分必要条件是|A|≠0

方阵的逆矩阵满足下述运算规律：

1. 若A可逆，则A-1亦可逆，且（A-1）-1=A
2. 若A可逆，数λ≠0，则λA可逆，且（λA）-1=1/λA-1
3. 若A、B为同阶矩阵且均可逆，则AB亦可逆，且（AB）-1=B-1A-1

证：（AB）（B-1A-1）=A（BB-1）A-1=AEA-1=AA-1=E

### 矩阵求逆的快速算法（高斯-约当法）

算法介绍

矩阵求逆在3D程序中很常见，主要应用于求Billboard矩阵。按照定义的计算方法乘法运算，严重影响了性能。在需要大量Billboard矩阵运算时，矩阵求逆的优化能极大提高性能。这里要介绍的矩阵求逆算法称为全选主元高斯-约旦法。

高斯-约旦法（全选主元）求逆的步骤如下：

首先，对于 k 从 0 到 n - 1 作如下几步：

1.从第 k 行、第 k 列开始的右下角子阵中选取绝对值最大的元素，并记住次元素所在的行号和列号，在通过行交换和列交换将它交换到主元素位置上。这一步称为全选主元。

2.m(k, k) = 1 / m(k, k)

3.m(k, j) = m(k, j) \* m(k, k)，j = 0, 1, ..., n-1；j != k

4.m(i, j) = m(i, j) - m(i, k) \* m(k, j)，i, j = 0, 1, ..., n-1；i, j != k

5.m(i, k) = -m(i, k) \* m(k, k)，i = 0, 1, ..., n-1；i != k

最后，根据在全选主元过程中所记录的行、列交换的信息进行恢复，恢复的原则如下：在全选主元过程中，先交换的行（列）后进行恢复；原来的行（列）交换用列（行）交换来恢复。

摘自：<http://dev.gameres.com/Program/Visual/3D/Mnquick.htm>

## 正交矩阵

如果A为N阶实矩阵，A满足A·AT=E,则称A 为正交矩阵.

则下列诸条件是等价的:

1) A 是正交矩阵

2) A·AT=E为单位矩阵

3) AT是正交矩阵

4) A的各行是单位向量且两两正交

5) A的各列是单位向量且两两正交

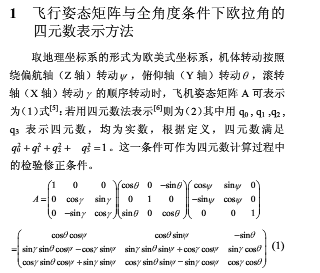
6) (Ax,Ay)=(x,y) x,y∈R

7)A-1=AT

性质：

1. 正交矩阵之积为正交矩阵
2. 正交矩阵的转置为正交矩阵
3. 正效矩阵的伴随矩阵为正交矩阵

## 姿态矩阵



<http://www.docin.com/p-192199688.html>

## 维数、基

定义：在线性空间V中，如果存在n个元素a1,a2,……,an满足：

1. a1,a2,……,an线性无关；
2. V中任一元素a总可由a1,a2,……,an线性表示；

那么，a1,a2,……,an就称为线性空间V的一个基，n称为线性空间V的维数

只含一个零元素的线性空间没有基，规定它的维数为0。

维数为n的线性空间称为n维线性空间，记作Vn。

## 根据变换矩阵求三分量（平移 旋转(四元数) 缩放）方法

我说一下原理吧，以下矩阵都是opengl形式的。

平移矩阵

T =

1 0 0 tx

0 1 0 ty

0 0 1 tz

旋转矩阵

R =

ux uy uz 0

vx vy vz 0

nx ny nz 0

0 0 0 1

缩放矩阵

S =

sx 0 0 0

0 sy 0 0

0 0 sz 0

0 0 0 1

组合在一起就是

C = T \* R \* S =

ux \* sx uy \* sy uz \* sz tx

vx \* sx vy \* sy vz \* sz ty

nx \* sx ny \* sy nz \* sz tz

0 0 0 1

给矩阵元素标号，下面叙述方便

m00 m10 m20 m30

m01 m11 m21 m31

m02 m12 m22 m32

m03 m13 m23 m33

得到平移向量很简单，就是m30，m31，m32

下面算缩放向量sx，sy，sz，

m00^2 + m01^2 + m02^2 =

(ux \* sx)^2 + (vx \* sx)^2 + (nx \* sx)^2

= (ux^2 + vx^2 + nx^2) \* sx^2

这里有个旋转矩阵的性质，

旋转矩阵是单位正交矩阵，

其转置矩阵就是其逆矩阵，

也是单位正交矩阵，即也构成一个旋转矩阵，

那么，(ux^2 + vx^2 + nx^2) = 1，

可求出sx，同理可求出sy和sz，

然后

m00/sx m10/sy m20/sz 0

m01/sx m11/sy m21/sz 0

m02/sx m12/sy m22/sz 0

0 0 0 1

就是一个旋转矩阵了，再转换为四元数即可。

<http://hack.gameres.com/showthread.asp?threadid=133185>

# 四元数

1843年，William Rowan Hamilton发明了四元数，但直到1985年才有一个叫Ken Shoemake的人将四元数引入计算机图形学处理领域。四元数在3D图形学中主要用于旋转，骨骼动画等。

四元数是基于复数的，理解起来有一点抽象。当然，四元数并不代表现实世界中的任何东西，只在数学意义上存在。

简单地来说，四元数描述了一次旋转：绕任意一个轴旋转一个角度。四元数的定义形式：(w, x, y, z)。假如，绕轴向量v(\_x,\_y,\_z)正向（右手旋转法则）旋转角度p，则对应得四元数q为：

q = (cos(p/2), sin(p/2) \* \_x, sin(p/2) \* \_y, sin(p/2) \* \_z)

用四元数来表示旋转，不如用欧拉角（偏航/yaw，俯仰/pitch，横滚/ roll）来表示直接，但是用欧拉角来处理旋转有个不可回避的问题：万向节死锁。我们可以通过将欧拉角转换到四元数来避免这个问题。在DirectX中提供了从欧拉角到四元数，从四元数到矩阵（Direct3D用旋转来实现旋转）的变换函数。下面参看：\Microsoft DirectX SDK\Include\d3dx9math.h的函数声明

从四元数变换到轴与旋转角

// Compute a quaternin's axis and angle of rotation. Expects unit quaternions.

void WINAPI D3DXQuaternionToAxisAngle

( CONST D3DXQUATERNION \*pQ, D3DXVECTOR3 \*pAxis, FLOAT \*pAngle );

从旋转矩阵构造一个四元数

// Build a quaternion from a rotation matrix.

D3DXQUATERNION\* WINAPI D3DXQuaternionRotationMatrix

( D3DXQUATERNION \*pOut, CONST D3DXMATRIX \*pM);

从一个轴与旋转角构造一个四元数

// Rotation about arbitrary axis.

D3DXQUATERNION\* WINAPI D3DXQuaternionRotationAxis

( D3DXQUATERNION \*pOut, CONST D3DXVECTOR3 \*pV, FLOAT Angle );

从欧拉角构造一个四元数

// Yaw around the Y axis, a pitch around the X axis,

// and a roll around the Z axis.

D3DXQUATERNION\* WINAPI D3DXQuaternionRotationYawPitchRoll

( D3DXQUATERNION \*pOut, FLOAT Yaw, FLOAT Pitch, FLOAT Roll );

下面我们来看看如何使用：若某物体要绕Y轴旋转fYaw，绕X轴旋转fPitch，绕Z轴旋转fRoll，那么

对应的旋转矩阵matRot可以这么计算：

D3DXQUATERNION qR;

D3DXMATRIX matRot;

D3DXQuaternionRotationYawPitchRoll(&qR, fYaw, fPitch, fRoll);

D3DXMatrixRotationQuaternion(&matRot, &qR);

在讨论「四元数」之前，我们来想想对三维直角座标而言，在物体旋转会有何影响，可以扩充三维直角座标系统的旋转为三角度系统（Three-angle system），在Game Programming Gems中有提供这么一段：

Quaternions do not suffer from gimbal lock. With a three-angle(roll, pitch, yaw) system, there are always certain orientations in which there is no simple change to the trhee values to represent a simple local roation. You often see this rotation having "pitched up" 90 degree when you are trying to specify a local yaw for right.

简单的说，三角度系统无法表现任意轴的旋转，只要一开始旋转，物体本身即失去对任意轴的自主性。

四元数（Quaternions）为数学家Hamilton于1843年所创造的，您可能学过的是复数，例如：a + b i 这样的数，其中i \* i = -1，Hamilton创造了三维的复数，其形式为 w + x i + y j + z k，其中i、j、k的关係如下：



假设有两个四元数：



四元数的加法定义如下：



四元数的乘法定义如下：



注意：



**共轭四元数**：

实数部分相等而虚数部分相反的两个数，复数q的共轭复数用表示

对于四元数q，计算其共轭q\*的方法与复数相同：只需要将虚数分量qv的符号反过来即可。

给定：



可以通过将虚部的符号反转，来计算四元数的共轭：



来看一下q与的积：

或者



它是各个分量系数的平方和。计算四元数的范数和倒数时，这个性质很有用



**四元数的范数**：



**四元数的倒数**：

四元数的倒数对我们来说特别重要，因为它可以简化四元数旋转。实际上，之前介绍的所有内容都是为了使用四元数来旋转向量，而倒数又是这种操作所必不可少的。

给定四元数q，我们要找另一个四元数q-1，使得：



如果q是一个单位四元数，则|q|2=1，可以进一步将倒数简化为：



上面的公式是使用四元数执行旋转成为可能的全部原因。因此，大多数时候，我们都假设所有四元数是单位四元数，这样便可以使用上面的公式，而不会出现任何问题。

**四元数转换为欧拉角**

在刚体定点转动理论中有一个著名的欧拉定理：刚体绕固定点的任一位移，可由绕通过此点的某一轴转过一个角度而得到。在单位时间间隔△t内假设刚体角速度为ω，则该转动轴的方向e及绕该轴转过的角度Φ分别为：



相应四元数表示为：

满足约束条件：

以超复数形式表示有：



满足约束条件：



利用三角公式：



可将四元数转化成姿态矩阵：（how?）



**欧拉角转换为四元数**

根据四元数的定义，可以将欧拉角转换为四元数，以欧拉转动为例：

（为什么虚数部分是负号？）

第一次先绕Z轴转动，φ=θ=0，四元数表示为：

第二次先绕Y轴转动，φ=ψ=0，四元数表示为：

第三次先绕X轴转动，θ=ψ=0，四元数表示为：

由绕三轴转动的合成为即：



（结论不对啊？=）由irrlicht源码可知结论是对的，正确的是下面的公式）

第一次先绕Z轴转动，φ=θ=0，四元数表示为：

第二次先绕Y轴转动，φ=ψ=0，四元数表示为：

第三次先绕X轴转动，θ=ψ=0，四元数表示为：

由绕三轴转动的合成为即：





该方法在±360度内均有效，是一对一的关系

同理，如果是按YXZ顺序旋转，则有：





有关四元数更详细的数学计算，转换过程可以参考下面的链接：

<http://www.euclideanspace.com/maths/algebra/realNormedAlgebra/quaternions/index.htm>

<http://www.cppblog.com/kesalin/archive/2008/03/16/44606.html>

<http://blog.csdn.net/zdl1016/article/details/1589372>

《3D游戏编程大师技巧》上册

<http://wenku.baidu.com/view/c789b628915f804d2b16c149.html>

# 欧拉角与万向锁(Gimbal Lock)

见/Opengl/math/ 3D 图形学(6) - 欧拉旋转之万向锁(Gimbal Lock)问题解释 .avi

见/Opengl/math/一种新的全角度四元数与欧拉角的转换算法.pdf

首先来看一下什么是欧拉角（Euler angles）？

**构件在三维空间中的有限转动，可依次用三个相对转角表示，即进动角、章动角和自旋角，这三个转角统称为欧拉角。**——引自百度百科

莱昂哈德·欧拉用欧拉角来描述刚体在三维欧几里得空间的取向。对于任何一个参考系，一个刚体的取向，是依照顺序，从这参考系，做三个欧拉角的旋转而设定的。所以，**刚体的取向可以用三个基本旋转矩阵来决定。换句话说，任何关于刚体旋转的旋转矩阵是由三个基本旋转矩阵复合而成的。**——引自wikipedia

好了，引完了，我来说一下我的理解吧，欧拉角是对旋转的一种刻画方式，就像其他刻画方式一样如旋转矩阵，四元数。欧拉角对应的旋转矩阵可以看作是三个绕轴旋转的旋转矩阵的复合。

问题来了，三个绕轴旋转的旋转矩阵绕的是什么坐标系下的轴？

对于坐标系E下的欧拉角(α，β，r)和以下哪个旋转矩阵是等价的

1.绕坐标系E下的x轴旋转α，绕坐标系E下的y轴旋转β，绕坐标系E下的z轴旋转r，三个矩阵的复合

2.绕坐标系E下的x轴旋转α，绕 坐标系E在绕x轴旋转α后的新系E'下的y轴旋转β，绕 坐标系E'在绕y轴旋转β后的新系E''下的z轴旋转r，三个矩阵的复合

通俗的讲，我们在旋转时，要不要把坐标系一起转动？

事实上两种理解都可以，当然，两种转法并不等价，下面我来解释这个问题，

当我们讲到坐标系E下的欧拉角(α，β，r)时，这句话是有歧义的，我们必须定义旋转顺序，因为旋转顺序会影响旋转结果。

如果假设旋转顺序是先绕x轴再y轴再z轴，x-y-z，那么这个欧拉角对应的旋转矩阵是指上述的2所表示的旋转矩阵。

如果假设旋转顺序是先绕z轴再y轴再x轴，z-y-x，那么这个欧拉角对应的旋转矩阵是指上述的1所表示的旋转矩阵，等等，你肯定会问，这难道不是把2中的先后顺序换一下就行了吗，"绕坐标系E下的z轴旋转r，绕 坐标系E在绕z轴旋转r后的新系E'下的y轴旋转β，绕 坐标系E'在绕y轴旋转β后的新系E''下的x轴旋转α，三个矩阵的复合"难道不是这样吗？是的，当然也是这样。

下面我来证明两种复合方式是相等的，

为了方便证明我先定义一些记号，

记：

绕坐标系E下的x轴旋转α的旋转矩阵为Rx,

绕坐标系E下的y轴旋转β的旋转矩阵为Ry,

绕坐标系E下的z轴旋转r的旋转矩阵为Rz,

绕坐标系E下的z轴旋转r的旋转矩阵为Rr（Rr=Rz），

绕 坐标系E在绕z轴旋转r后的新系E'下的y轴旋转β的旋转矩阵为Rb，

绕 坐标系E'在绕y轴旋转β后的新系E''下的x轴旋转α的旋转矩阵为Ra，

另外，将矩阵R的逆记作R~

求证：Rx\*Ry\*Rz = Rr\*Rb\*Ra

证明：

Rr = Rz 定义就是一样的，显然相等

Rb = Rr~\*Ry\*Rr 要得到绕 坐标系E在绕z轴旋转r后的新系E'下的y轴旋转β的旋转矩阵为Rb，可以先应用Rr~这时可以视作在E下，然后使用E下的旋转Ry绕旧的y轴旋转，在应用Rr转回到E'

Ra = (Rr\*Rb)~\*Rx\*(Rr\*Rb) 理由同上

所以 右边=Rr\*Rb \* Ra

=Rr\*Rb \* (Rr\*Rb)~\*Rx\*(Rr\*Rb)

=(Rr\*Rb)\* (Rr\*Rb)~\*Rx\*(Rr\*Rb)

=Rx\*(Rr\*Rb)

=Rx\*(Rr\*Rr~\*Ry\*Rr)

=Rx\*Ry\*Rz =左边

#证毕

摘自：<http://www.cnitblog.com/luckydmz/archive/2010/09/07/68674.html>

Irricht的setRotationRadians( const vector3d<T>& rotation ),(αβθ)



经程序验证为：

zyx

cc\*cb,-sc\*ca+cc\*sb\*sa,-sc\*-sa+cc\*sb\*ca

sc\*cb,cc\*ca+sc\*sb\*sa,cc\*-sa+sc\*sb\*ca

-sb,cb\*sa,cb\*ca

对应的右手系为：

zyx

cc\*cb,sc\*ca+cc\*-sb\*-sa,sc\*sa+cc\*-sb\*ca

-sc\*cb,cc\*ca+-sc\*-sb\*-sa,cc\*sa+-sc\*-sb\*ca

sb,cb\*-sa,cb\*ca

|  |
| --- |
| 注：以下模型及代码以右手系，列优先为准 |



# 四元数与欧拉角之间的转换

在3D图形学中，最常用的旋转表示方法便是四元数和欧拉角，比起矩阵来具有节省存储空间和方便插值的优点。本文主要归纳了两种表达方式的转换，计算公式采用3D笛卡尔坐标系：

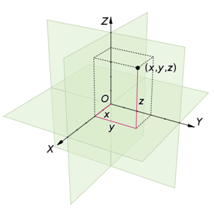


图1 3D Cartesian coordinate System (from wikipedia)

    定义121309_1044_2分别为绕Z轴、Y轴、X轴的旋转角度，如果用Tait-Bryan angle表示，分别为Yaw、Pitch、Roll。

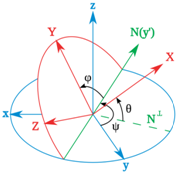
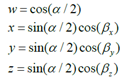


图2 Tait-Bryan angles (from wikipedia)

一、四元数的定义

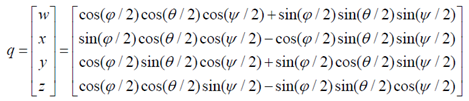
121309_1044_4

   通过旋转轴和绕该轴旋转的角度可以构造一个四元数：

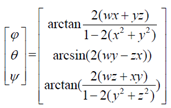


       其中121309_1044_6是绕旋转轴旋转的角度，121309_1044_7为旋转轴在x,y,z方向的分量（由此确定了旋转轴）。

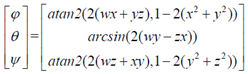
二、欧拉角到四元数的转换



三、四元数到欧拉角的转换

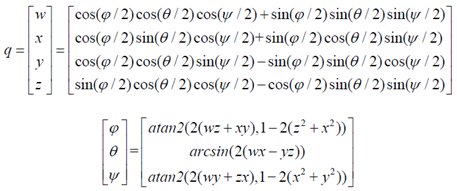


       arctan和arcsin的结果是121309_1044_10，这并不能覆盖所有朝向(对于121309_1044_11角121309_1044_12的取值范围已经满足)，因此需要用atan2来代替arctan。



四、在其他坐标系下使用

在其他坐标系下，需根据坐标轴的定义，调整一下以上公式。如在Direct3D中，笛卡尔坐标系的X轴变为Z轴，Y轴变为X轴，Z轴变为Y轴（无需考虑方向）。



五、示例代码

<http://www.cppblog.com/Files/heath/Euler2Quaternion.rar>  
Demo渲染两个模型，左边使用欧拉角，右边使用四元数，方向键Up、Left、Right旋转模型。

参考文献：

[1] http://en.wikipedia.org/wiki/Conversion\_between\_quaternions\_and\_Euler\_angles

[2] Ken Shoemake, Animating Rotation with Quaternion Curves, 1985

摘自：<http://www.cppblog.com/heath/archive/2009/12/13/103127.html>

# 3D class &四元数 Flex/AS

//转自闪客帝国 作者:luke 日期:2007-01-18

3D建模里有一些常用的概念，我先贴在下面：

l）对象和世界坐标

对象坐标是描述3D对象基本形状的xyz维。世界坐标是描述对象的xyz坐标，当对象被放置在一个指定的3D环境中某个位置和旋度上。所以称为世界坐标是因为3D环境被称为3D世界。

（2）摄影机和彩像面坐标

摄影机（Camera）坐标是xyz坐标，它描述该对象对于一个观察者如何出现在3D环境中的指定位置。摄影机位置被称为视点（观察点）（Viewpoint）。

影像面坐标是xy坐标，它将出现在一个两维的设置在摄影机和3D对象之间的影像面（或窗口）上。

（3）屏幕坐标

屏幕坐标是x y坐标，它能够在计算机显示屏幕上报绘出来。屏幕坐标也称为显示器（display）坐标，它是将影像面定标以适合显示屏幕的结果。

在这个3D操作过程中计算每组坐标都包括从矩阵数学导出的正弦（sine）和余弦（cosine）公式。

我们真的需要四元数吗？

作者是一个经验丰富的游戏开发者，数学学士。

第一部分：等价

最近我的一些顾客被要求在图形绘制里加入支持3D旋转的四元数。我进行了它的研究并为我的发现而惊讶。好象有一类四元数的狂热爱好者。四元数在书籍、文章、网上被广泛的证明。开发者用它们来开发游戏。但是它确实是对3D图形绘制程序有用的工具吗？在我看来，未必。

四元数是一个数学概念。他们包括虚数和第四维度，是一独特的代数分支，使用一种枯燥的乘法运算。规则如下：ij=k, ji=-k and i\*i= -1,它可以创建一函数来设定在3D空间里旋转的点。四元数可以做到。它做被要求做的工作。但它是否可以做任何事情，而这些是传统的数学无法轻易作到的吗？我不认为是这样。

下面是一个常见的四元数旋转的矩阵表示.

其中,(x,y,z)是旋转轴上的单位向量,0是旋转的角度.

这是一标准的任意轴旋转的矩阵表示.

同样,(x,y,z)是旋转轴上的单位向量,0是旋转的角度.

第一个矩阵是基于四元数,第二个矩阵建立于传统的欧拉坐标系.作者应用了大量的数学公式来证明,他们是可以互相转化的.作者第二部分是基于R3（欧拉坐标系）的绕任意轴旋转的矩阵证明。它的目的就是两者的对比，来比较优劣。我突然懒的翻译以后的部分，我想简单的做一四元数介绍更为恰当，如果有感兴趣的爱好者，可以看原文理解。

序：一个标准的四元数的表达应该是q\*q的平方根。对于我，四元数这个术语好象是世界之外的东西，象是关于量子论里的术语，拥有神秘的黑暗力量。如果你，同样被它黑暗的色彩迷惑，这篇文章，希望能带给你启迪。文章将教你如何应用四元数进行3D旋转，并使得你理解它的含义。

为什么使用四元数？回答它之前，先讨论一些向量表达。

欧拉法。这是最简单实用的方向表达。每一轴，都有一个特定的旋转角度。这样，我们有3个变量：

x,y,z ←旋转角度基于默认系统坐标系。它的变化在0-360度之间。她们是旋转，倾斜，偏移的表示。方向是由3个角度的旋转复数的乘积得到（通过你定义的顺序）。旋转是由全局坐标系决定的。这意味着第一个旋转不能改变第二、第三个旋转的轴。从而造成"万向锁"的情况，我将在一会讨论。

任意轴表示法：

这种表示法的好处是避免了欧拉表示法造成的"万向锁"问题。它由任意轴的单位向量，以及旋转角度的变量构成。

x, y, z <-- 任意轴的单位向量表示

angle <-- 该轴的旋转角度

这些表示法有什么缺点吗？

万向锁：

欧拉系中的旋转涉及固定的全局轴。一个轴的旋转毫不影响另一个轴向的旋转。导致你失去自由的角度。这就是万向锁。比方，一个向量（平行于X轴）绕Y轴的旋转。。。。。。。。。。。。

插值问题：

尽管任意轴坐标系没有万向锁的问题，但当你需要在两个旋转中插入旋转的时候，会出现问题。计算出的内插值方向将不平滑，变成一种紊乱的运动。欧拉也有这样的问题。

正式开始

我们先需要建立一些假设。否则会造成很多数学上的混淆。

坐标系：假定为右手法则，象是OpenGL。如果你在使用Direct3D等左手法则系统，你将需要做一些转换。(bluepoint提示：在图形学和工程学的规则中，右手法则是表达和操作三维坐标的标准数学约定。不过只要您清楚所选坐标系的含义，并且做到前后一致，那么具体选择什么坐标系其实并不重要。）注意Direct3D 范例里有四元数库，你可以在使用前看一看。

旋转顺序：

(bluepoint:由于矩阵乘法在通常情况下具有不可传递性，我们规定了以下顺序：)欧拉坐标系的旋转的次序是x,y,z；矩阵形式：

RotX \* RotY \* RotZ <-- 很重要

[ 0 4 8 12

1 5 9 13

2 6 10 14

3 7 11 15 ]

矢量和点：

旋转矩阵

[ vx

vy

vz

1 ] 4\*1 矢量

什么是四元数？

复数的虚部有一个i单位来构成，其中i\*i=-1；四元数是复数的扩展。不同的是，它有3个单位属于-1的平方根，定义为i,j,k；这样，一个四元数可以表示为：q = w + xi + yj + zk。W是一个实数，其余的是虚数。另一种常用的表示：q=[ w,v ]。v = (x, y, z)（V代表向量，W是一个标量），尽管V称为向量，不要视其为典型的3D向量。它是4D空间的向量，无法直觉的想象到。

定义四元数

我们有两种方式：

q= [1,(0, 0, 0)]

q= [0,(0, 0, 0)]（这种不是我们要使用的）

使用四元数

首先要指出四元数不是向量，所以不要把你预想的向量数学带进来。解释需要一些数学理论，请多多包涵。

我们先定义四元数的量级。

|| q ||= Norm(q) =sqrt(w2 + x2 + y2 + z2)

单位四元数有下面的属性：w2 + x2 + y2 + z2=1

规范化四元数的表达，q = q / || q || = q / sqrt(w2 + x2 + y2 + z2)

单位四元数的特殊就在于它描绘了一个3D空间的方向。所以你可以用它来代替前面讨论的两种方法来表示方向。要应用它，你需要把四元数的表示方法转换一下。

观察四元数

你可以把四元数想象成4D空间里的旋转。（x,y,z）形成任意轴，w形成旋转的角度。所有的单位四元数形成了一个球状的4D空间。它是很难以直觉理解。注意：只有单位四元数才可以用来描述方向。我们的讨论就是基于这点。

四元数的转化形式

为了有效的使用四元数，需要把它们转换成另一种表示法。

四元数变为矩阵。

等价的旋转矩阵表示四元数的形式：

Matrix = [ w2 + x2 - y2 - z2 2xy - 2wz 2xz + 2wy

2xy + 2wz w2 - x2 + y2 - z2 2yz - 2wx

2xz - 2wy 2yz + 2wx w2 - x2 - y2 + z2 ]

使用恒等式简化一下：w2 + x2 + y2 + z2 = 1

Matrix = [ 1 - 2y2 - 2z2 2xy - 2wz 2xz + 2wy

2xy + 2wz 1 - 2x2 - 2z2 2yz - 2wx

2xz - 2wy 2yz + 2wx 1 - 2x2 - 2y2 ]

四元数转化为任意轴形式

如果旋转轴是ax, ay, az)，角度是theta ，那么角度angle= 2 \* acos(w)；

ax= x / scale

ay= y / scale

az= z / scale

其中 scale = x2 + y2 + z2，另一种我发现的表达是scale = sin(acos(w)). 它们应该是等价的。如果scale=0，那么旋转角度就为0。

单位四元数表示3D空间的旋转度，那么多个四元数的乘积将是另一个旋转度的表示。

给出两个四元数：

Q1=(w1, x1, y1, z1);

Q2=(w2, x2, y2, z2);

Q1 \* Q2 =( w1.w2 - v1.v2, w1.v2 + w2.v1 + v1\*v2)；

其中v1= (x1, y1, z1)

v2 = (x2, y2, z2)

需要注意的是q1 \* q2 不等于 q2 \* q1。

转变为四元数。

任意轴系到四元数

w = cos(theta/2)

x = ax \* sin(theta/2)

y = ay \* sin(theta/2)

z = az \* sin(theta/2)

欧拉系到四元数

你可以把角度分为3个独立的四元数，然后使用乘法来组合得到最终的四元数表达。

Qx = [ cos(a/2), (sin(a/2), 0, 0)]

Qy = [ cos(b/2), (0, sin(b/2), 0)]

Qz = [ cos(c/2), (0, 0, sin(c/2))]

Q=Qx \* Qy \* Qz.

总结如下：

四元数是[x, y, z]里面的第四个元素，用来描述三个分量的向量。四元数是使用矩阵方法进行3-D旋转的一种选择，四元数提供你一个3-D物体沿着一个轴进行旋转的能力，但是真正发挥四元数能力的操作是：合成与插值运算 （Composition and Interpolation）。

**合成两个四元数的意思是：“先沿着一个轴旋转一个给定角度，然后再沿着另一个轴旋转一个给定的角度”。合成的两个四元数可以用以下标记来表示：Q=q1ｏq2**

进行两个四元数的插值运算可以使程序运算出从同一个轴的一点到另一点的平滑且合理的路径。所以，q1与q2进行插值运算是实现3D动画的一种简单方法。

记住下面的话：四元数是一实数和一矢量的和的表达式，有四个项，一个为实数项，另外三个为虚数项。它是矩阵表达法的一种替换形式，尤其是和三维坐标旋转。 （在Diana Gruber答读者问的时候，有一些观点好象和用AS2创建3D类的作者Chad Corbin相左。理论上说来，你可以用其它的矩阵表示法来推导出旋转公式）

<http://fortomxq.blog.163.com/blog/static/27978759200842324536511/>

# 三维坐标系

通常三维图形应用程序使用两种笛卡尔坐标系：左手系和右手系。识别这两个坐标系有两种方法：

1. 通过沿正x轴方向到正y轴方向握拳，大姆指的指向就是相应坐标系统的正z轴的指向。
2. 张开手指，保持拇指与食指垂直，然后弯曲中指，使中指与拇指、食指所在平面垂直，此时，拇指指向X轴方向，食指指向Y轴方向，中指指向Z轴方向。

在开始设计3D虚拟世界之前，我们需要预先做一些约定：比如采用左手坐标系还是右手坐标系，+y指向哪个方向等等。

Microsoft® Direct3D®使用左手坐标系。如果正在移植基于右手坐标系的应用程序，必须将传给Direct3D的数据做两点改变。

1. 颠倒三角形顶点的顺序，这样系统会从正面以顺时针的方向遍历它们。换句话说，如果顶点是v0，v1，v2，那么以v0，v2，v1的顺序传给Direct3D。
2. 用观察矩阵对世界空间中的z值取反。要做到这一点，将表示观察矩阵的D3DMATRIX结构的\_31、\_32、\_33和\_34成员的符号取反。

# 几何变换

应用于对象几何描述并改变它的位置、方向或大小的操作称为几何变换。

## 二维变换

### 平移

通过将位移量加到一个点的坐标上来生成一个新的坐标位置，实现平移。

x’=x+ tx

y’=y+ty

平移距离（tx，ty）称为平移向量

使用列向量来表示坐标位置和平移向量：

，，

可得二维平移方程：

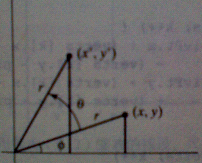
P’=P+T

### 旋转

通过指定一个旋转轴和一个旋转角度，可以进行一次旋转。

一个对象的二维旋转通过在xy平面上沿圆路径将对象重定位来实现。

首先确定当基准点为坐标原点时点位置P进行旋转的变换方程。如下图：



R是点到原点的固定距离，角Φ是点的原始角度位置与水平线的夹角，θ是旋转角。应用标准的三角等式，我们得到：

X’=rcos(Φ+θ)=rcosΦcosθ-rsinΦsinθ

Y’=rsin(Φ+θ)=rcosΦsinθ+rsinΦcosθ

在极坐标中，点的原始坐标为

X=rcosΦ, y=rsinΦ 代入以上方程，得：

X’=xcosθ-ysinθ

Y’=xsinθ+ycosθ

使用列向量表达式表示坐标位置，那么旋转方程的矩阵形式为：

P’=R·P

其中旋转矩阵为



## 三维变换

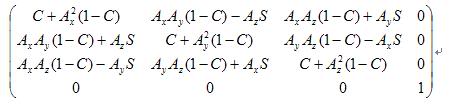
### 平移



### 旋转

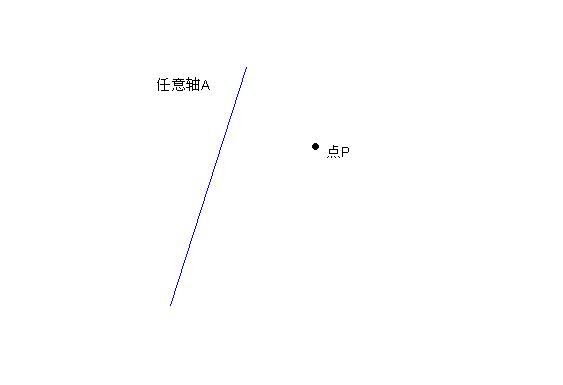
### 任意轴旋转矩阵的推导

左手坐标系下，一点绕任意轴旋转θ角的右乘矩阵：



其中C为cosθ，S为sinθ，A为单位化的旋转轴

以下推导均为左手坐标



首先我们将P看成从原点出发的自由向量，将其分解为平行于轴A与垂直于轴A的分量A1，A2的形式：

27601fdfb55835aa76c63883    （公式1）

如图2：

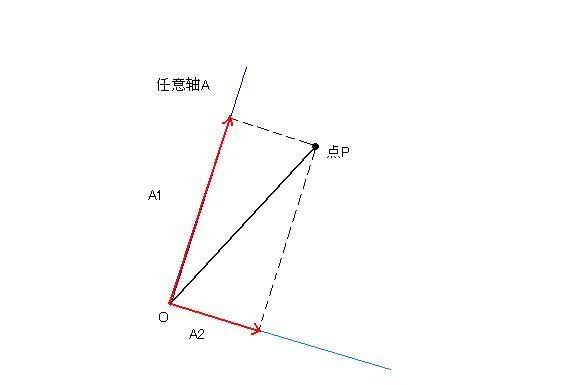


图2

d4935ded98523853fdfa3c83                        （公式2）（不明，可能使用向量投影）

6c4e6714a5b30a14f3de3283              （公式3）

由于平行分量A1在旋转过程中保持不变，问题就在于垂直分量A2。先将A2旋转θ度后（A3）再加上A1就可以得到最终的旋转结果。如图3：

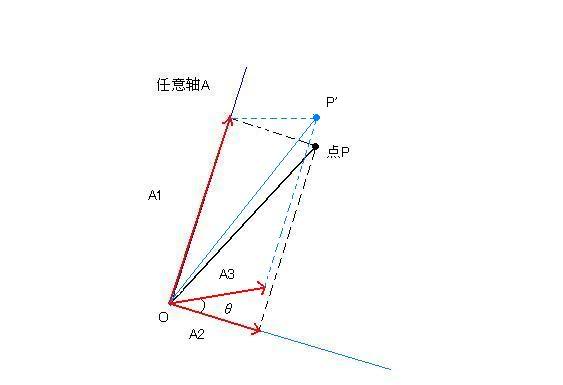


图3蓝色的点为P点旋转θ度后的位值

旋转后的位置点P＇为：

2e016eecae569f6c2df53483   （公式4）

因为A2到A3的旋转是在垂直于A轴平面内进行的，所以可以将A3分解为A2与A2逆时针方向旋转90度的向量A4上的两个分量的形式。如图4：

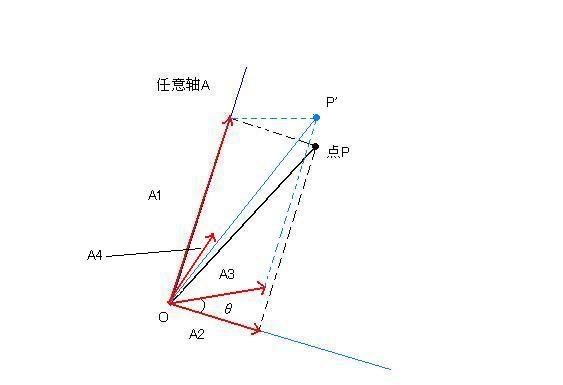


图4

下面我们来求A4  
A2可以看成φ的对边，如图5：

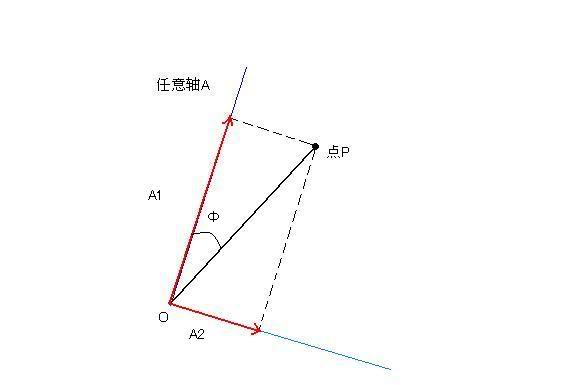


图5

A2的长度为：

3b704633382991ed5edf0e83   （公式5）

我们想要的向量A4正好为：

fe35ceaa389628f4cb130c83   （公式6）（根据？）

fc8a77b3a35aec1109230283    （公式7）

因为A为单位向量所以（公式7）与（公式5）相等，这样长度相等并且同时垂直于A1，A2，所以就是我们要求的A4。

接下为求A3，我们用一个垂直于A的平面图来看一下，这样更直接，如图6：

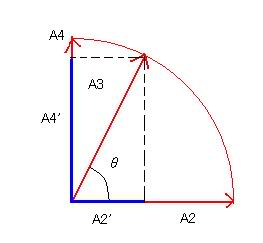


图6

A2＇与A4＇分别为A3在A2与A4上的分量。

aea62dc87082af03f21fe783   （公式8)

ab7627563a07371dd1090683   （公式9）

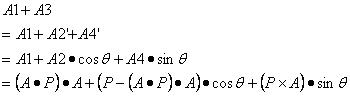
f41b685abf40169b9c820483（公式10）

因为：A2，A3，A4的长度都相等所以上面的两式化简为：

d99ef044ff561b4c6b63e583   （公式11）

735c49dceccb741dccbf1a83   （公式12）

将以上公式代入到（公式4）中计算最终结果：



整理后得：

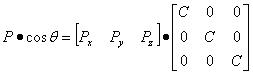
c9360c267a29a9768744f983   （公式13）

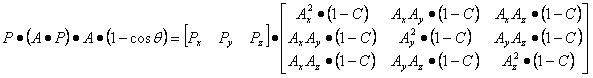
设：

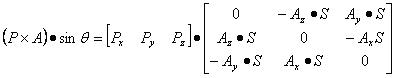
6eea9683f443bc9dbc3e1e83

225fe3daca77147033fa1c83

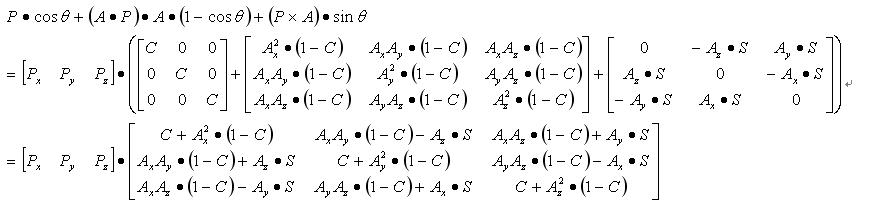
写成矩阵的形式为：

   （公式14）

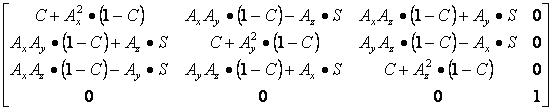
   （公式15）

   （公式16）

代入（公式13）得：



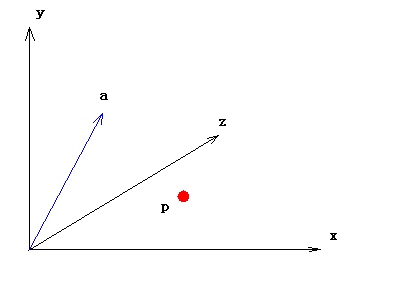
最终齐次的旋转矩阵为：

   （公式17）

摘自：

<http://hi.baidu.com/jlgwxq/blog/item/a11a82f2dda194cd7931aa9a.html>

### 快速做3D空间任意轴旋转的方法



如上所示，空间坐标系中的一个局部坐标系xyz中有一个向量a(2,5,3)和一个点p(8,4,2)现在我要让p点围绕a向量旋转60度，得到p'点，该如何做呢？从目前掌握的旋转知识来看，我们有两个理论基础：

1）在一个坐标系中的一个点，如果要它围绕该坐标系中一个坐标轴旋转，就给它的坐标值乘相应的旋转矩阵，如

[cosA -sinA 0]

[sinA cosA 0]

[0 0 1]

等等。

2）我们已经学习了局部坐标系的理论了，知道空间中一个点在不同的坐标系中的坐标不同。利用这一点，我们可以很方便的让一个点或者向量在不同的坐标系之间转换。

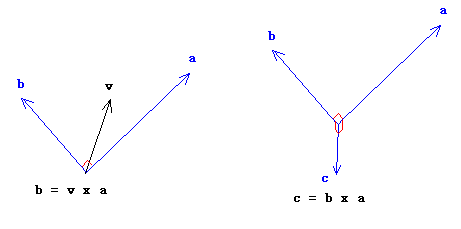
我们联系这两个理论根据，得出我们的思路：

1、构造另一个局部坐标系abc，使得a成为该坐标系的一个坐标轴。

2、把p的坐标变换到abc中，得到p'，用旋转公式让p'围绕已经成为坐标轴的a旋转，得到p''。

3、把p''再变换回坐标系xyz，得到p'''，则p'''就是p围绕a旋转后的点。

下面我们逐步说明。



……

向量几何在游戏编程中的使用6

<http://blog.csdn.net/popy007/article/details/376952>

# 仿射空间

## 仿射空间与向量空间的区别

“In mathematics, a vector space (or linear space) is a collection of objects (called vectors) that, informally speaking, may be scaled and added. More formally, a vector space is a set on which two operations, called (vector) addition and (scalar) multiplication, are defined and satisfy certain natural axioms which are listed below. Vector spaces are the basic objects of study in linear algebra, and are used throughout mathematics, science, and engineering.”

就像介绍里说的，向量空间的对象是向量（a collection of objects called vectors), 向量空间里的一切都是向量，这里的关键在于，向量空间有一个原点，所以向量空间中连点也可以看成一个向量（从原点出发指向该点的矢量）。

再看一下仿射空间的介绍（张量分析及其应用，李开泰，黄艾香，科学出版社）：

“在仿射空间里，点和向量是基本的概念，无需用逻辑方法再定义。当然，这不是说点和向量没有实在的内容。例如向量就可理解为速度和力等。

考察一个点和向量的集合，它满足以下公理（1）至少存在一个点。（2）任意给定一对有顺序的点A和B，对应一个且仅对应一个向量。通常记此向量为AB。”

可见，点在仿射空间中有独立的地位，即便是存在点和矢量的对应也得是两个有序点。之所以是这样，是因为仿射空间里没有原点。

举个例子，某空间中有两个点，如果是在向量空间，则我们可以对两个点加减，即两个点对应与原点相连的矢量按照平行四边形法则加减，从而得到第三个点。然而在仿射空间中，两个点的加减是没有意义的，但两点之间的距离可以计算，距离是个不变量，独立于坐标系。

引入仿射空间的原因是要对独立于坐标系的不变量进行描述，它实际上放宽了向量空间的要求，从而促使人们在更一般的空间上研究某些不变的性质。这就像欧氏空间的假设被放宽后使得我们开始研究更一般的非欧几何一样。仿射空间是张量代数和张量分析的基础。

摘自：<http://www.baisi.net/thread-776927-1-1.html>

# 刚体的有限转动 欧拉定理

将刚体上的不动点记为*O*，如图4-2所示过该点建立刚体的连体基和考察刚体运动的参考基，分别记为Image1021和Image1022。

|  |
| --- |
| **Image1733** |
| 图4-2 定点运动刚体的连体基 |

刚体在参考基Image1023上的姿态与该刚体连体基Image1024相对于参考基Image1023的姿态是一致。它可以用基Image1024相对于基Image1023的方向余弦阵(即***A****rb*)来描述，由式[(1.3-3)](http://zyk.thss.tsinghua.edu.cn/27/Tm/dynamics/ch01/ch010301.html)，有

|  |  |
| --- | --- |
| Image1734 | (4.1-1) |

从刚体运动的角度，刚体当前的姿态Image1024是以前某一姿态的改变，这种改变称为刚体绕定点*O*的**有限转动**。如果认为参考基Image1023是刚体的前一个姿态，那么刚体当前的姿态相对于前一个姿态的方向余弦阵为***A****rb*。考虑该方阵本征根方程，由于式[(1.3-8)](http://zyk.thss.tsinghua.edu.cn/27/Tm/dynamics/ch01/ch010303.html)与[(1.3-15)](http://zyk.thss.tsinghua.edu.cn/27/Tm/dynamics/ch01/ch010304.html)，它可表为

|  |  |
| --- | --- |
| Image1735 |  |
| Image1736 | |
| 图4-3 两个基的一次转动矢量 | |

可知该本征根方程至少存在一个 **=1 的根。将该本征根**=1的本征矢量记为Image1737。在基Image208的坐标阵为***p****b* ，代入本征方程，有Image1738。展开得Image1739，考虑到式[(1.3-13)](http://zyk.thss.tsinghua.edu.cn/27/Tm/dynamics/ch01/ch010304.html)，有Image1740。由此可得到如下结论：对于任意两个基Image1741与基Image208存在一个矢量Image1737，它在两基的坐标阵相等(见图4-3)。此结论也可理解为将矢量Image1737作为一个旋转轴，基Image208是基Image210绕Image1737转过一个有限角度后到达的新的方位。考虑到此矢量的存在性，可得到如下的定理：刚体绕定点的任意有限转动可由绕过该点的某根轴一次有限转过某个有限角度实现。

此定理称为欧拉有限转动定理。将单位矢量Image1737称为由基Image1023到基Image208**一次转动矢量**。转过的有限角称为**一次转动角**，记为**。从上面分析不难看出，刚体相对于基Image1023的不同姿态均可绕相应的一次转动矢量和作相应的一次转动角来实现，也就是说刚体的不同姿态对应不同的一次转动矢量和一次转动角。

http://zyk.thss.tsinghua.edu.cn/27/Tm/dynamics/ch04/ch040102.html

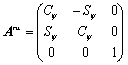
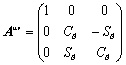
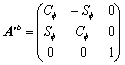
# 姿态坐标 欧拉角

在1.3节已经指出，式[(4.1-1)](http://zyk.thss.tsinghua.edu.cn/27/Tm/dynamics/ch04/ch040102.html)的方向余弦阵***A****rb* 中第1、2与3列分别为基Image1024、基矢量Image1036、Image1742与Image1743在基Image1023上的坐标阵。因此在某瞬时如果已知方向余弦阵的9个元素，根据方向余弦阵上述的几何意义，就可确定连体基Image1024在基Image1023上的姿态。也就是说，对于给定的参考基Image1023，可以定义方向余弦阵的9个元素为定点运动刚体的姿态坐标。然而，在1.3节还指出，方向余弦阵的9个元素必须满足几何约束方程式[(1.3-6)](http://zyk.thss.tsinghua.edu.cn/27/Tm/dynamics/ch01/ch010303.html)至[(1.3-8)](http://zyk.thss.tsinghua.edu.cn/27/Tm/dynamics/ch01/ch010303.html)，故刚体的这9个姿态坐标中只有3个是独立的。也就是说只要3个姿态坐标就能描述定点运动刚体的姿态。可以适当选取上述9个元素中的3个作为姿态坐标，但这样做的随意性很大，3个姿态坐标的物理意义也不太清楚。为此将根据欧拉定理采取下面的方法来定义刚体的姿态坐标。其基本思想是初始连体基Image1744与参考基Image1745重合，连体基的当前姿态是由其初始状态绕空间三个不同的基矢量分别连续作三次有限转动后实现。

|  |
| --- |
| [Image1752](javascript:;) |
| 图4-4 欧拉角 |

现定义连体基第一次转动是绕参考基Image1745的基矢量Image1746转过有限角**，到达的方位用过渡基Image1747表示(见图4-4a)。第二次转动是绕基Image1747的基矢量Image1748转过有限角**，到达的新的方位用过渡基Image1749表示(见图4-4b)。第三次转动是绕基Image1749的基矢量Image1750转过有限角**，到达连体基Image1744当前的姿态(见图4-4c)。称这三次有限转动的三个角Image1751为刚体的**欧拉角**，其中称 ** 为**进动角**，** 为**章动角**，** 为**自转角**。对照图4-4，上述的依次的转动关系也可由图4-5表达。

根据方向余弦阵的定义式[(4.1-1)](http://zyk.thss.tsinghua.edu.cn/27/Tm/dynamics/ch04/ch040102.html)，基Image1747相对于基Image1745、基Image1749相对于基Image1753和Image1744相对于基Image1749的方向余弦阵分别为

，， (4.1-2)

|  |
| --- |
| Image1757 |
| 图4-5 欧拉角的三次有限转动 |

以上三式中的缩写Image1758与Image1759分别为Image1760与Image1761，其余类推。由方向余弦阵的性质[(3)](http://zyk.thss.tsinghua.edu.cn/27/Tm/dynamics/ch01/ch010304.html)(见1.3节)，可得到连体基Image1762关于基Image1745的方向余弦阵为

|  |  |
| --- | --- |
| Image1763 | (4.1-3) |

将式(4.1-2)代入上式，经整理得

|  |  |
| --- | --- |
| Image1764 | (4.1-4) |

上式给出了描述连体基姿态的方向余弦阵与刚体欧拉角间的关系，可见刚体欧拉角可以用来描述连体基的姿态，定义它们为刚体的姿态坐标，记为

|  |  |
| --- | --- |
| Image1765 | (4.1-5) |

需要特别注意，当章动角** 等于零时，基Image1766、Image1749与Image1767将不可分，进动角** 与自转角**将混淆，上述定义的三次转动不能得到唯一的欧拉角坐标。章动角** 等于零称为**欧拉角的奇异点**。根据上述情况，在定义连体基与参考基时需考虑连体基的姿态不能出现章动角** 等于零的情况。

<http://zyk.thss.tsinghua.edu.cn/27/Tm/dynamics/ch04/ch040103.html>

# 欧拉角至方向余弦转换

欧拉角指定欧拉角，以弧度为单位。

phi指定第一个轴第一个旋转的角度，以弧度为单位。

theta指定第二个轴旋转的角度，以弧度为单位。

psi指定第三个轴第二个旋转的角度，以弧度为单位。

**旋转顺序**指定了旋转坐标的坐标轴的次序。 例如，**X-Y-Z**表示第一个、第二个和第三个旋转分别为x轴、y轴和z轴。**Z-X-Z**是默认次序。 

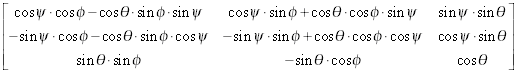
|  |  |
| --- | --- |
| 0 | X-Y-Z |
| 1 | X-Z-Y |
| 2 | Y-X-Z |
| 3 | Y-Z-X |
| 4 | Z-X-Y |
| 5 | Z-Y-X |
| 6 | X-Y-X |
| 7 | X-Z-X |
| 8 | Y-X-Y |
| 9 | Y-Z-Y |
| 10 | Z-X-Z |
| 11 | Z-Y-Z |

方向余弦返回3 x 3的方向余弦矩阵。方向余弦中的每个元素必须在[-1, 1]的范围内。

错误返回VI的任何错误或警告。将错误连接至错误代码至错误簇转换VI，可将错误代码或警告转换为错误簇。

欧拉角至方向余弦转换详细信息

可通过[方向余弦](http://zone.ni.com/reference/zhs-XX/help/371361H-0118/gmath/3d_cartesian_coord_rotation_direction/)或[欧拉角](http://zone.ni.com/reference/zhs-XX/help/371361H-0118/gmath/3d_cartesian_coordinate_rotation_euler/)表述旋转。下列等式为**欧拉角**和**方向余弦**之间的关系（假定旋转次序为默认的**Z-X-Z**）：

R = 

R是3×3**方向余弦**输出矩阵。phi(pi < phi leq pi)、theta (0 leq theta leq pi)和psi(－pi < psi leq pi)为输入**欧拉角**，以弧度为单位。

<http://zone.ni.com/reference/zhs-XX/help/371361H-0118/gmath/euler_angles_to_direction_cosines/>