# **[【Unity编程】四元数(Quaternion)与欧拉角](https://www.cnblogs.com/driftingclouds/p/6592319.html)**

## **欧拉旋转、四元数、矩阵旋转之间的差异**

除了欧拉旋转以外，还有两种表示旋转的方式：矩阵旋转和四元数旋转。接下来我们比较它们的优缺点。

### **欧拉角**

* 优点：三个角度组成，直观，容易理解。
* 优点：可以进行从一个方向到另一个方向旋转大于180度的角度。
* 弱点：死锁问题。 前面[《【Unity编程】欧拉角与万向节死锁（图文版）》](http://blog.csdn.net/andrewfan/article/details/60981437)已经介绍过万向节死锁问题。

### **四元数**

内部由四个数字（在Unity中称为x，y，z和w）组成，然而这些数字不表示角度或轴，并且通常不需要直接访问它们。除非你特别有兴趣深入了解四元数学，你只需要知道四元数表示三维空间中的旋转，你通常不需要知道或修改x，y和z属性。

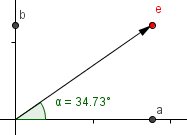
* 优点：四元旋转不存在万向节锁问题。
* 优点：存储空间小，计算效率高。
* 弱点：单个四元数不能表示在任何方向上超过180度的旋转。
* 弱点：四元数的数字表示不直观。

由于Unity中的旋转使用四元数，因此本文重点就放在四元数数学上。矩阵还是某些场合需要用到，比如坐标变换。不过你几乎不需要手动去执行矩阵计算，除非你在做Shader编程，或者是某些需要极端提高效率的场合。后续我计划写一篇文章专门介绍矩阵变换。

## **四元数的数学**

由于前面两篇文章均尽可能采用无数字计算的方式，为的是方便理解。而这里由于需要理解四元数的计算，我们还是需要理解一些复数和欧拉旋转计算方面的基本数学。

### **复数**



首先了解一下复数，上图中，左右方向的轴（X轴）称为实数轴，上下的轴(Y轴)称为虚数轴。  
任意一个二维矢量，都可以使用一个复数形式进行表示。也就是：**e = a + bi**其中a是实数，i是虚数，**i \* i=-1。**

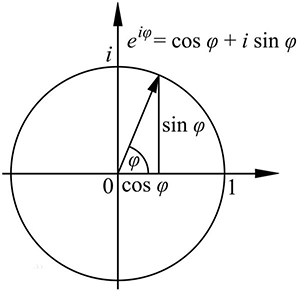
至于什么是虚数，我的理解它就是个不同维度的后缀标记和符号而已。i标记了实数轴和虚数轴之间的差异，也就是旋转90度旋转的差别，乘以i意味着这种旋转关系。  
为什么i \* i=-1?也就是两个虚数i乘积又变成了实数-1？因为从实数轴(1表示)围绕上图中逆时针90度(1i)到达虚数轴(=i)，再逆时针旋转90(1 i \* i)就到达了负实数轴方向(=-1)。所以这里规定i \* i=-1。

### **复数运算**

* 加法： (a+bi)+(c+di)=(a+c)+(b+d)i
* 减法： (a+bi)-(c+di)=(a-c)+(b-d)i
* 乘法： (a+bi)(c+di)=ac+bci+adi+bdi^{2}=(ac-bd)+(bc+ad)i

更多复数相关知识，请参考[“维基百科-复数\_(数学)”](https://zh.wikipedia.org/wiki/%E5%A4%8D%E6%95%B0_(%E6%95%B0%E5%AD%A6))

### **欧拉旋转定理**



为了方便讨论旋转，我们避开矢量长度的影响，也就是假设问题是基于长度为1的矢量去讨论。由上图可以看出，当长度为1时，矢量落在长度为1的圆形上，此时实数轴上的a = cos（φ），虚数轴上的b = sin（φ），其中φ为旋转角度。

此时的表示形式为 e = cos（φ） + sin（φ）i

那么，使用这样一种形式到底有什么意义呢？数学从来都是工具，如果没有用处，它也就不会存在了。我们接下来看它的作用。

当圆上的一个矢量进行了连续的旋转时时，假设先旋转φ，再旋转θ，则结果应该是两个旋转的角的和的复数形式，即 e = cos（φ+θ）+sin（φ+θ）i

假设：  
**e1= cos（φ） + sin（φ）i** (表示旋转了φ角度)  
**e2= cos（θ） + sin（θ）i** (表示旋转了θ角度)

那么e和e1、e2之间的关系是怎样的？  
- - -  
根据[和差公式](https://zh.wikipedia.org/wiki/%E9%A4%98%E5%BC%A6)可以得知

cos（φ+θ）+sin（φ+θ）i  
= （cos（φ）cos（θ）-sin（φ）sin（θ））+(sin（θ）cos（φ）+cos（θ）sin（φ）)i  
= cos（φ）cos（θ）+cos（φ）sin（θ）i+cos（θ）sin（φ）i- sin（φ）sin（θ）  
= (cos（φ） + sin（φ）i)(cos（θ） + sin（θ）i)  
= e1 e2

=>

e = e1\*e2  
- - -  
也就是说，连续的旋转（例如这里旋转φ再旋转θ），可以使用两个复数的乘积进行表示。在计算机运算中：

* 如果知道φ=30度，那么计算此单位矢量再旋转θ=40度是很容易的，只需要直接运算cos（φ+θ）和sin（φ+θ）即可。
* 如果知道一个矢量(a+bi)，不知其φ值，要对其进行旋转θ=40度，那么也就有了快速的计算方法，即：  
  (a+bi) \* (cos(40)+sin(40)i) = (a \* cos(40)-b \* sin(40)) + (b \* cos(40) + a \* sin(40))i  
  结果：二维向量 x = a \* cos(40)-b \* sin(40)，y= b \* cos(40) + a \* sin(40)  
  可以快速计算出二维矢量结果，而不必先求φ。

**因此这就是复数和欧拉旋转定理的作用，它可以使用复数很方便地表示出二维矢量的旋转变化。**

## **四元数**

相对于复数为二维空间，为了解决三维空间的旋转变化问题，爱尔兰数学家威廉·卢云·哈密顿把复数进行了推广，也就是四元数。

以下均为定义，所谓定义，就是我们人为设置的概念和计算方法，它们本身或许没有什么意义，但是如果按照这些概念和方法计算出某些有意义的结果，那么这些定义也就有了相应的意义。

### **四元数定义**

四元数定义i、j、k三个虚数单位参与运算，并有以下运算规则：

j∗k=i,k∗j=−i;j∗k=i,k∗j=−i;

j∗k=i,k∗j=−i;j∗k=i,k∗j=−i;

k∗i=j,i∗k=−j;k∗i=j,i∗k=−j;

i∗i=j∗j=k∗k=i∗j∗k=−1i∗i=j∗j=k∗k=i∗j∗k=−1

i、j、k仍然理解为旋转，其中：

* i旋转代表X轴与Y轴相交平面中X轴正向向Y轴正向的旋转
* j旋转代表Z轴与X轴相交平面中Z轴正向向X轴正向的旋转
* k旋转代表Y轴与Z轴相交平面中Y轴正向向Z轴正向的旋转
* -i、-j、-k分别代表i、j、k旋转的反向旋转

一个普通四元数可以写成如下形式：

q¯¯=a+bi+cj+dkq¯=a+bi+cj+dk

四元数的i、j、k之间乘法的性质与向量之间的叉积结果形式很类似，于是四元数有了另外一种表示形式：

q¯¯=(（x,y,z）→,w)=(u⃗ ,w)q¯=(（x,y,z）→,w)=(u→,w)

### **加法定义**

四元数加法，跟复数、矢量和矩阵一样，两个四元数之和需要将不同的元素加起来，加法遵循实数和复数的所有交换律和结合律：

q¯¯1+q¯¯2=w1+w2+u1→+u2→=(w1+w2)+(x1+x2)i+(y1+y2)j+(z1+z2)kq¯1+q¯2=w1+w2+u1→+u2→=(w1+w2)+(x1+x2)i+(y1+y2)j+(z1+z2)k

### **格拉斯曼积定义**

四元数的乘法有很多种，最常见的一种定义，与数学多项式乘法相同，称为格拉斯曼积。(注意，下面乘积的式子是由多项式形如a+bi+cj+dk的多项式进行多对多乘法(比如4项x4项=16项)计算后，使用矢量的点积和叉积替代部分计算项后形成)：

q¯¯1q¯¯2=w1w2−u1→⋅u2→+w1u2→+w2u1→+u1→×u2→q¯1q¯2=w1w2−u1→⋅u2→+w1u2→+w2u1→+u1→×u2→

把向量部分和实数部分分开，可以写成：

q¯¯1q¯¯2=((w1u2→+w2u1→+u1→×u2→),(w1w2−u1→⋅u2→))q¯1q¯2=((w1u2→+w2u1→+u1→×u2→),(w1w2−u1→⋅u2→))

注意，格拉斯曼积符合结合率，也就是q¯¯1q¯¯2q¯¯3=(q¯¯1q¯¯2)q¯¯3=q¯¯1(q¯¯2q¯¯3)q¯1q¯2q¯3=(q¯1q¯2)q¯3=q¯1(q¯2q¯3)，但不符合交换律，一般来说，q¯¯1q¯¯2≠q¯¯2q¯¯1q¯1q¯2≠q¯2q¯1。

### ****点积定义****

点积也叫做欧几里得内积，四元数的点积等同于一个四维矢量的点积。点积的值是q¯¯1q¯1中每个元素的数值与 q¯¯2q¯2 中相应元素的数值的一对一乘积(比如4项x4项=4项)的和。这是四元数之间的可换积，并返回一个标量。

q¯¯1⋅q¯¯2=w1w2+u1→⋅u2→=w1w2+x1x2+y1y2+z1z2q¯1⋅q¯2=w1w2+u1→⋅u2→=w1w2+x1x2+y1y2+z1z2

### **叉积定义**

四元数叉积：p × q  
四元数叉积也称为奇积。它和矢量叉积等价，并且只返回一个矢量值：

p¯¯¯×q¯¯=p¯¯¯q¯¯−q¯¯p¯¯¯2p¯×q¯=p¯q¯−q¯p¯2

p¯¯¯×q¯¯=u⃗ ×v⃗ p¯×q¯=u→×v→

p¯¯¯×q¯¯=(cz−dy)i+(dx−bz)j+(by−xc)kp¯×q¯=(cz−dy)i+(dx−bz)j+(by−xc)k

### **共轭定义**

四元数的共轭的定义。即实部相同，虚部相反。（与复数概念类似）

q¯¯∗=(（−x,−y,−z）→,w)=(−v⃗ ,w)q¯∗=(（−x,−y,−z）→,w)=(−v→,w)

### **定义的推论**

从以上三个定义，可以得出一个推论，一个四元数与其共轭的格拉斯曼积等于此四元数与其自身的点积：

q¯¯q¯¯∗=q¯¯∗q¯¯=q¯¯⋅q¯¯=x2+y2+z2+w2q¯q¯∗=q¯∗q¯=q¯⋅q¯=x2+y2+z2+w2

证明：  
由格拉丝曼积定义可知

q¯¯1q¯¯2=w1w2−u1→⋅u2→+w1u2→+w2u1→+u1→×u2→q¯1q¯2=w1w2−u1→⋅u2→+w1u2→+w2u1→+u1→×u2→

而此时，

q¯¯1=q¯¯,q¯¯2=q¯¯∗q¯1=q¯,q¯2=q¯∗

也就是，

u1→=u⃗ ,u2→=−u⃗ ,w2=w1=wu1→=u→,u2→=−u→,w2=w1=w

代入

q¯¯1q¯¯2=w1w2−u1→⋅u2→+w1u2→+w2u1→+u1→×u2→q¯1q¯2=w1w2−u1→⋅u2→+w1u2→+w2u1→+u1→×u2→

得到

q¯¯q¯¯∗=ww+u⃗ ⋅u⃗ −wu⃗ +wu⃗ +u⃗ ×u⃗ q¯q¯∗=ww+u→⋅u→−wu→+wu→+u→×u→

由于第三四项可以消去，第五项本身为0向量，所以得出

q¯¯q¯¯∗=ww+u⃗ ⋅u⃗ =q¯¯⋅q¯¯q¯q¯∗=ww+u→⋅u→=q¯⋅q¯

于是得证，q¯¯∗q¯¯q¯∗q¯也可以同理得证，结果是与之相同的。

### **模的定义**

四元数的模的定义。与矢量的模类似，都表示长度或者绝对值的意思。四元数的绝对值是四元数到原点的距离。

|q¯¯|=x2+y2+z2+w2−−−−−−−−−−−−−−√=w2+u⃗ ⋅u⃗ −−−−−−−−√=q¯¯⋅q¯¯−−−−√=q¯¯q¯¯∗−−−√|q¯|=x2+y2+z2+w2=w2+u→⋅u→=q¯⋅q¯=q¯q¯∗

### **逆的定义**

四元数的逆通过q¯¯−1q¯¯=1q¯−1q¯=1来定义。在此式左右两端同时乘以 q¯¯∗q¯∗，可以得到：

q¯¯−1q¯¯q¯¯∗=q¯¯∗q¯−1q¯q¯∗=q¯∗

由于格拉丝曼积符合结合率，左式后两项先乘，得到：

q¯¯−1(q¯¯q¯¯∗)=q¯¯∗q¯−1(q¯q¯∗)=q¯∗

再由上述的推论，可以得出：

q¯¯−1(q¯¯⋅q¯¯)=q¯¯∗q¯−1(q¯⋅q¯)=q¯∗

由于点积是个实数，可以使用除法，所以最后简化成：

q¯¯−1=q¯¯∗q¯¯⋅q¯¯q¯−1=q¯∗q¯⋅q¯

### **单位四元数的定义**

所谓的单位量，都是指，任意原始量乘以单位量之后保持原始量不变。如对于实数而言，任意原始实数乘以实数1等于原始实数，因此1被定义为单位。  
对于四元数同样，任意原始四元数q¯¯q¯乘以某个四元数q¯¯iq¯i,原始的四元数q¯¯q¯保持不变，则称此四元数q¯¯iq¯i为单位四元数。注意这里的乘以是指格拉丝曼积。根据格拉丝曼积的定义：

q¯¯q¯¯i=((wui→+wiu⃗ +u⃗ ×ui→),(wwi−u⃗ ⋅ui→))q¯q¯i=((wui→+wiu→+u→×ui→),(wwi−u→⋅ui→))

若希望结果保持不变，也就是结果仍然为：

q¯¯q¯¯i=q¯¯=(w,u⃗ )q¯q¯i=q¯=(w,u→)

可以推测

wi=1,u⃗ i=0wi=1,u→i=0

也就是，单位四元数为:

q¯¯i=(0⃗ ,1)q¯i=(0→,1)

### **四元数的归一化**

归一化也叫单位化，与矢量的概念一样，就是为了把长度调整到单位长度。现在单位四元数已经得知，其单位长度显然是1。我们也知道了模的计算方法，因此把四元数的归一化方法如下：

q¯¯,=q¯¯|q¯¯|=q¯¯w2+u⃗ ⋅u⃗ −−−−−−−−√q¯,=q¯|q¯|=q¯w2+u→⋅u→

前述的单位复数表示形式为：e = cos(φ）+ sin(φ)i，因为此时，复数的长度为cos(φ)2+sin(φ)2=1cos(φ)2+sin(φ)2=1，我们使用类似形式表示归一化的四元数：

q¯¯=(s⃗ in(φ)n⃗ ,cos(φ))q¯=(s→in(φ)n→,cos(φ))

也就是实部现在使用cos(φ)cos(φ)表示，虚部使用sin(φ)n⃗ sin(φ)n→表示，其中n⃗ n→是归一化的三维矢量，由于a⃗ ⋅b⃗ =|a⃗ ||b⃗ |cos(θab)a→⋅b→=|a→||b→|cos(θab)，因此有n⃗ ⋅n⃗ =1n→⋅n→=1。让我们尝试计算此时的模：

|q¯¯|=w2+u⃗ ⋅u⃗ −−−−−−−−√=cos(φ)2+sin(φ)2(n⃗ ⋅n⃗ )−−−−−−−−−−−−−−−−−−−√=1|q¯|=w2+u→⋅u→=cos(φ)2+sin(φ)2(n→⋅n→)=1

### **四元数映射\_实数**

我们可以将实数映射到四元数空间，所谓的映射，也就是说1对1的找到相应的存在，并且在不同空间施加施加相对应运算的法则时获得相同的结果。  
例如，对应实数w，尝试进行如下映射：

q¯¯=(0⃗ ,w)q¯=(0→,w)

那么是否能够遵守相应的运算法则？也就是说，如果实数wawa与实数wbwb均被映射到了四元数空间，那么它们的加法和乘法的结果是否还能映射回去，与原始空间的运算获得相同的结果？加法显然可以，这里进行乘法计算测试：

q¯¯a=(0⃗ ,wa)q¯a=(0→,wa)

q¯¯b=(0⃗ ,wb)q¯b=(0→,wb)

q¯¯aq¯¯b=(0⃗ ,wa)(0⃗ ,wb)=wawb−0⃗ ⋅0⃗ +wa0⃗ +wb0⃗ +0⃗ ×0⃗ q¯aq¯b=(0→,wa)(0→,wb)=wawb−0→⋅0→+wa0→+wb0→+0→×0→

=>

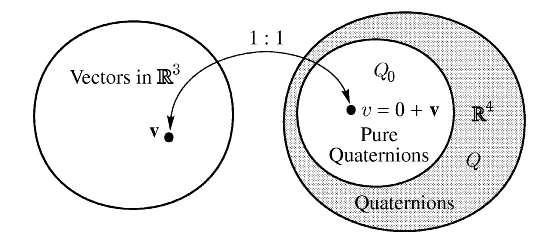
q¯¯aq¯¯b=wawb+0⃗ =(0⃗ ,wawb)q¯aq¯b=wawb+0→=(0→,wawb)

显然，依照映射原则，可以将(0⃗ ,wawb)(0→,wawb)映射回去变成wawawbwb，与实数空间的计算结果相同。因此，此映射法则是可行的：

q¯¯=(0⃗ ,w)q¯=(0→,w)

### **四元数映射\_矢量直接映射**

与实数映射类似，我们也可以尝试将三维矢量映射到四元数。



对于矢量v⃗ v→映射法则如下：

q¯¯=(v⃗ ,0)q¯=(v→,0)

同样，加法运算显然可以映射回去，我们来计算乘法运算。

q¯¯a=(v⃗ a,0)q¯a=(v→a,0)

q¯¯b=(v⃗ b,0)q¯b=(v→b,0)

q¯¯aq¯¯b=(v⃗ a,0)(v⃗ b,0)=00−va→⋅vb→+0vb→+0va→+va→×vb→q¯aq¯b=(v→a,0)(v→b,0)=00−va→⋅vb→+0vb→+0va→+va→×vb→

=>

q¯¯aq¯¯b=(va→×vb→,−v⃗ a⋅v⃗ b)q¯aq¯b=(va→×vb→,−v→a⋅v→b)

可以看到，此时出现了之前两个四元数中均不存在的实部。因此这个映射法则是错误的。（本身两个向量之间如果直接使用多对多的格拉斯曼积式乘法也是没有意义的，所以结果可想而知。那么，是不是就不能映射矢量了？不是，只是映射法则错误。回想之前在在复数中也曾使用过格拉斯曼积式的乘法  
如果矢量用e=cos(φ)+sin(φ)ie=cos(φ)+sin(φ)i表示，那么此时的ee表示了一个二维旋转，向量的旋转可以使用格拉斯曼积形式的复数乘法e1∗e2e1∗e2表示，因此，映射法则应该与角度相关。）

### **四元数映射\_矢量间接映射**

### **同态**

先考虑一个同态概念：假设M，M′是两个乘集，也就是说M和M′是两个各具有一个封闭的具有结合律的运算与的代数系统。φ是M射到M′的映射，并且任意两个元的乘积的像是这两个元的像的乘积，即对于M中任意两个元a,b,满足  
φ(ab)=φ(a)φ(b)；  
也就是说，当a→φ(a)，b→φ(b)时，ab→φ(a)φ(b)，  
那么这映射φ就叫做M到M′上的同态。我前面所解释的可以互相映射大致意思也就是同态。

### **四元数中的同态函数**

为什么我们研究同态？因为这里涉及到我们希望出现的向量的旋转，一个三维空间的向量被旋转之后，它应该还是一个三维矢量。这个旋转函数应该满足以下要求：

①、旋转完的矢量长度应该保持不变。也就是

|φ(A)|=|A||φ(A)|=|A|

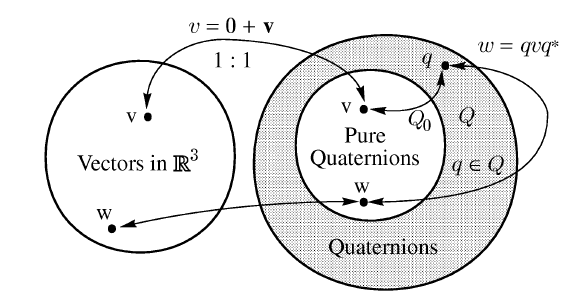
②、假如对向量A和B都执行φ旋转，那么向量A、B的夹角θ应该与φ(A)、φ(B)的夹角应该相同。由矢量点积我们知道A⋅B=|A||B|cos(θ)A⋅B=|A||B|cos(θ)，在满足①的情况下，也就是模的乘积已经不变，那么如果点积结果相同，也就能保证角度不变，也就是：

φ(A)⋅φ(B)=A⋅Bφ(A)⋅φ(B)=A⋅B

③、假如对向量A和B都执行φ旋转，那么旋转完毕的φ(A)、φ(B)的相对方向关系与A、B应该保持一致，也就是所谓的手向性应该保持一致（比如原来A的正左侧恰好B，旋转完毕应该还是在正左侧，仅有夹角是不够的，正右侧也是90度关系）。假如满足下面的式子，则手向性不变

φ(A)×φ(B)=A×Bφ(A)×φ(B)=A×B

接着，我们详细了解间接映射的步骤。如下图所示：



我们通过间接映射将矢量V映射到四元数，先如直接映射那样，映射成一个纯虚四元数，即(V⃗ ,0)(V→,0)，再执行φ()变换。简写为

φ(V)φ(V)

由于

AB=wawb−ua⋅ub+wa⋅ub+wb⋅ua+ua×ubAB=wawb−ua⋅ub+wa⋅ub+wb⋅ua+ua×ub

此时的A、BA、B均是纯虚四元数，因此wa=wb=0wa=wb=0，可以简化为

AB=−ua⋅ub+ua×ubAB=−ua⋅ub+ua×ub

由于A、BA、B均是纯虚四元数，因此A⋅B=ua⋅ubA⋅B=ua⋅ub，再由于四元数叉积和矢量叉积等价，因此上式可以转化为

AB=−A⋅B+A×BAB=−A⋅B+A×B

如果存在这样的同态函数函数：

φ(A)φ(B)=φ(AB)φ(A)φ(B)=φ(AB)

也就能满足②、③的要求。  
[此处还可以解释得更明确，不过太过复杂了]

### **旋转四元数**

最后，给出旋转四元数，也就是一直在追求的φ()函数。

q¯¯=(s⃗ in(θ2)n⃗ ,cos(θ2))q¯=(s→in(θ2)n→,cos(θ2))

它的含义是围绕n⃗ n→轴旋转θθ角度。  
具体旋转的方法使用以下函数：

p¯¯¯,=q¯¯p¯¯¯q¯¯∗p¯,=q¯p¯q¯∗

其中p¯¯¯p¯是被旋转四元数，p¯¯¯,p¯,是结果四元数。这里的q¯¯∗q¯∗写成q¯¯−1q¯−1也是可以的，因为根据前述对逆的计算，对于此时模为1的q¯¯q¯来说，它们结果相同的。  
具体推究方法以及证明方法过于复杂，就不在这里给出了。

## **总结**

四元数是高阶复数的数学，它用在游戏中的作用主要是计算三维矢量的旋转，它使用先将矢量映射到纯虚四元数，再应用旋转四元数的方式进行映射。最后可以达成旋转目的。  
前面的四元数性质不甚了解也没有太大关系，大概了解到是如何映射的也就可以了。甚至在Unity编程过程中你对四元数的映射关系不了解都无所谓。后续文章中我再介绍具体的用法。