# 数学公式

## 点乘

向量A和向量B的点乘

A点乘B = |A||B|Cos(θ)

向量B在A上的阴影为（A点乘B）/|A|=|B|Cos(θ)

A( x1, y1)点乘B(x2, y2) = x1\*x2 + y1\*y2

在谁上的阴影，就除以谁的模

## 叉乘

2维空间中的叉乘是：

A(x1, y1) X B(x2, y2) = x1y2 – y1x2

A x B = |A||B|Sin(θ)

可以用来计算点到线的距离，A，B，C三个点，画出向量AB(b)和AC(c)，夹角为θ，距离=|c|Sin(θ)=(b x c)/|b|

## 三维向量的点乘

点乘比较简单，是相应元素的乘积的和：

V1( x1, y1, z1)·V2(x2, y2, z2) = x1\*x2 + y1\*y2 + z1\*z2;

注意结果不是一个向量，而是一个标量（Scalar）。点乘有什么用呢，我们有：

A·B = |A||B|Cos(θ)

θ是向量A和向量B见夹角。这里|A|我们称为向量A的模(norm)。这样我们就和容易计算两条线的夹角：

Cos(θ) = A·B /(|A|\*|B|)

点到面的距离:

设某三维平面表达式为

a\*x+b\*y+c\*z+d = 0;

则其法向量即为: (a,b,c).

任意一点 p = (x1, y1, z1)到该平面的距离为：

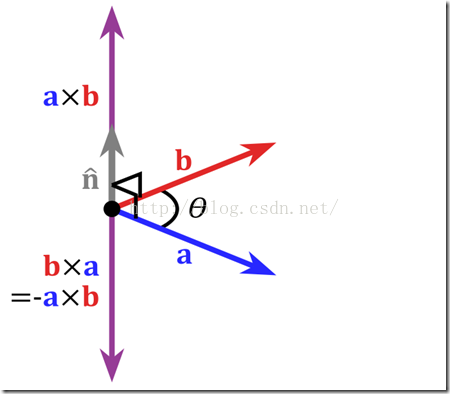
(a\*x1 + b\*y1 + c\*z1+d) / (a\*a + b\*b +c\*c)

## **三维向量的差乘**

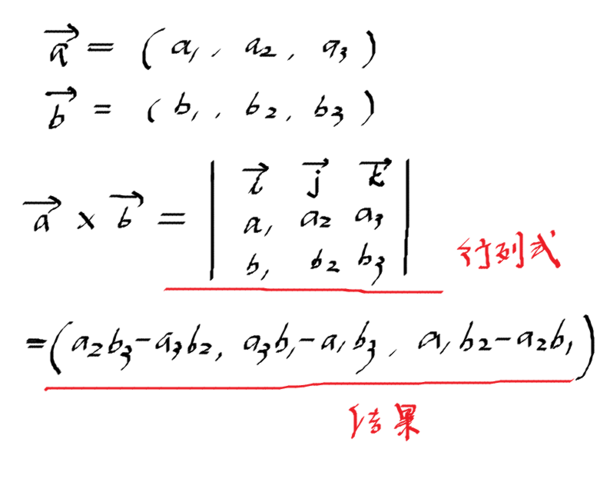
首先我们知道 ，对于向量u和v, u x v的结果，是得到一个既垂直于u又垂直于v的向量,假设记作n.

则有下面公式：n = u x v;

而n的方向，是由右手法则决定的。 即伸出右手，四个手指方向从u绕到v. 此时，大姆指的方向，就是n的方向。 我们通常叫做右向量。

引用一下维基百科的图来说明问题，有兴趣的兄弟可以照图比划一下。 （注：图中向量是用的a x b来表示）  


三维叉乘公式



**行列式知识点：**

**{**

**IMG_256**

**结果为:**

**a1·b2·c3+b1·c2·a3+c1·a2·b3-a3·b2·c1-b3·c2·a1-c3·a2·b1(注意对角线就容易记住了）**

**}**i,j,k.

i,j,k满足以下特点

i = j x k; j = k x i; k = i x j;

k x j = –i; i x k = –j; j x i = –k;

i x i = j x j = k x k = 0; (0是指0向量）

由此可知，i,j,k是三个相互垂直的向量。它们刚好可以构成一个坐标系。

这三个向量的特例就是 i = (1,0,0) j = (0,1,0) k = (0,0,1)。

好，那对于处于i,j,k构成的坐标系中的向量a,b我们可以如下表示

a = X1\*i + Y1\*j + Z1\*k;

b = X2\*i + Y2\*j + Z2\*k;

那么 a x b = (X1\*i + Y1\*j + Z1\*k) x (X2\*i + Y2\*j + Z2\*k)

= X1\*X2\*(ixi) + X1\*Y2\*(i x j) + X1\*Z2\*(i x k) + Y1\*X2\*(j x i) + Y1\*Y2\*(j x j) + Y1\*Z2\*(j x k) + Z1\*X2\*( k x i ) + Z1\*Y2(k x j) + Z1\*Z2(k x k)

由于上面的i,j,k三个向量的特点，所以，最后的结果可以简化为

a x b = (Y1\*Z2 – Z1\*Y2)\*i + (Z1\*X2 – X1\*Z2)j + (X1\*Y2 – Y1\*X2)k;

于是，在i,j,k构成的坐标系中。集就是上面的结果。

当i = (1,0,0) j = (0,1,0) k = (0,0,1)时，我们通常省略i,j,k的写法。最终也就得到了我们的右向量。

叉乘的意义

叉乘表示垂直于axb的右向量。

使用的地方

可以通过叉乘，修正向量关系，从而构建坐标系。 常见的有 摄相机矩阵和TBN空间转换矩阵的构建。

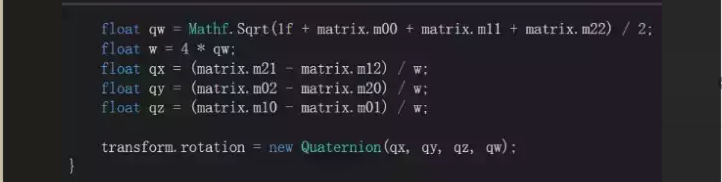
## 四元数和欧拉角的相互转换

四元数q转欧拉角v  
Vector3 v = q.eulerAngles;

欧拉角v转四元数q  
Quaternion q = Quaternion.Euler(v);

四元数的计算转化

matrix为Matrix4x4对象是一个4x4的矩阵

Matrix4x4.identity,获得单位矩阵  


# [Unity3d中UnityEngine.Object的几个注意点](http://www.cnblogs.com/flamepenguin/p/3469864.html)

UnityEngine.Object继承自system.Object，是Unity所涉及所有物体的基类。

先看一下Object重载的运算符：

Operators

|  |  |
| --- | --- |
| bool | Does the object exist? |
| operator != | Compares if two objects refer to a different object. |
| operator == | Compares if two objects refer to the same. |

下面几个注意点是我在最近使用发现的。

（1）Object类重载了类型隐式转换运算符“bool”。这个如果不注意，在某些情况下可能会造成意料外的调用。

例

class Test:MonoBehaviour  
{

     void Start()  
     {  
        Fun(transform);//此函数目的是为了调用 void Fun(system.Object objFlag)

     }

     void Fun(bool value)  
     {  
         Debug.Log("call the bool param fun,the value is:"+value.ToString();)  
     }

     void Fun(system.Object objFlag)  
     {  
         Debug.Log("call the param system.Object fun,the value is:"+value.ToString();)  
     }

}

打印的日志是：call the bool param fun,the value is: true

可以看出，与我们的目的不一致。通过追溯Transform父类，其顶层基类是UnityEngine.Object。这就找到原因了，在这样的情况下这两个Fun函数存在调用歧义。transform先被判断是否存在的引用，然后调用了void Fun(bool value)

建议：同一个类尽量不要使用同名函数，如有要使用同名函数时要特别注意这两种情况。

（2）Object类重载了类型隐式转换运算符“==”。这个在与null比较时要特别注意，既使是有效的引用结果有可能是true的。

例：

GameObject go = new GameObject();

Debug.Log (go == null); // false

Object obj = new Object();

Debug.Log (obj == null); // true

可以知道删除GameObjetc/Component 等就是替换了所有引用判断空就只能通过这两种方法：

1. obj==null||obj.Equals(null) // 通过它重载的Equals
2. If(obj) // 隐式转换的bool值就是非空判断