## 1.在D3DX库中使用D3DXVECTOR3结构图表示向量，它继承自D3DVECTOR3。（在d3d9math.h中定义）

结构图的定义：

typedef struct D3DXVECTOR3 : public D3DVECTOR

{

public:

D3DXVECTOR3() {};

D3DXVECTOR3( CONST FLOAT \* );

D3DXVECTOR3( CONST D3DVECTOR& );

D3DXVECTOR3( CONST D3DXFLOAT16 \* );

D3DXVECTOR3( FLOAT x, FLOAT y, FLOAT z );

// casting

operator FLOAT\* ();

operator CONST FLOAT\* () const;

// assignment operators

D3DXVECTOR3& operator += ( CONST D3DXVECTOR3& );

D3DXVECTOR3& operator -= ( CONST D3DXVECTOR3& );

D3DXVECTOR3& operator \*= ( FLOAT );

D3DXVECTOR3& operator /= ( FLOAT );

// unary operators

D3DXVECTOR3 operator + () const;

D3DXVECTOR3 operator - () const;

// binary operators

D3DXVECTOR3 operator + ( CONST D3DXVECTOR3& ) const;

D3DXVECTOR3 operator - ( CONST D3DXVECTOR3& ) const;

D3DXVECTOR3 operator \* ( FLOAT ) const;

D3DXVECTOR3 operator / ( FLOAT ) const;

friend D3DXVECTOR3 operator \* ( FLOAT, CONST struct D3DXVECTOR3& );

BOOL operator == ( CONST D3DXVECTOR3& ) const;

BOOL operator != ( CONST D3DXVECTOR3& ) const;

} D3DXVECTOR3, \*LPD3DXVECTOR3;

D3DVECTOR的定义如下：

typedef struct \_D3DVECTOR {

float x;

float y;

float z;

} D3DVECTOR;

## 2.关于向量的近似相等问题

由于向量的成员是float类型，而计算机不能精确表示浮点数，所以只能使用近似相等原则来比较两个向量：

方法是定义一个非常小的浮点数EPSILON(0.001f)

如果向量v1-v2<EPSILON,则认为他们近似相等。

D3DXVECTOR3的函数已经做好了这个工作。

3.计算向量的长度，也就是向量的模

公式：



在D3DX库中，使用以下函数计算

D3DXINLINE FLOAT D3DXVec3Length( CONST D3DXVECTOR3 \*pV );

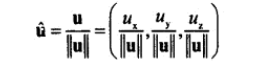
如：

|  |
| --- |
| D3DXVECTOR3 v(0.1f,0.2f,0.3f);  float magnitude=D3DXVec3Length(&v);//计算向量的模 |

## 4.关于向量的规范化

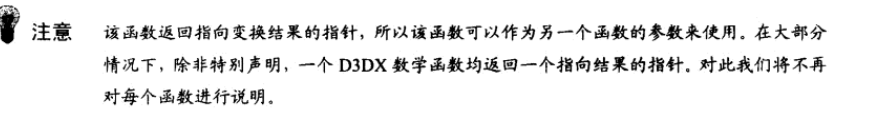
向量的规范化就是将向量的模变为1，即变为单位向量

公式：





**D3DX库中提供函数：D3DXVECTOR3\* WINAPI D3DXVec3Normalize( D3DXVECTOR3 \*pOut, CONST D3DXVECTOR3 \*pV );** 来实现向量的规范化，返回一个指向结果的指针，可以作为另外一个函数的参数。

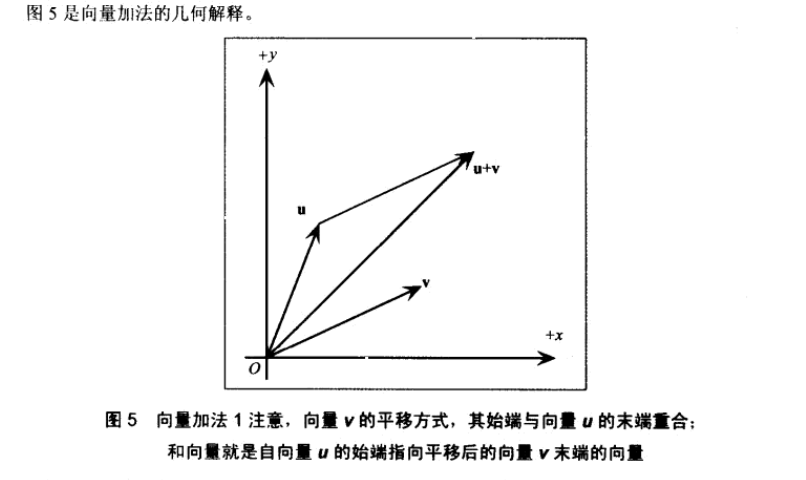


## 5.向量的加法

注意：只有维数相同的向量才能进行加操作！

两个向量相加等于两个向量的x，y，z分别相加

如：v1+v2=v(v1.x+v2.x,v1.y+v2.y,v1.z+v2.z)

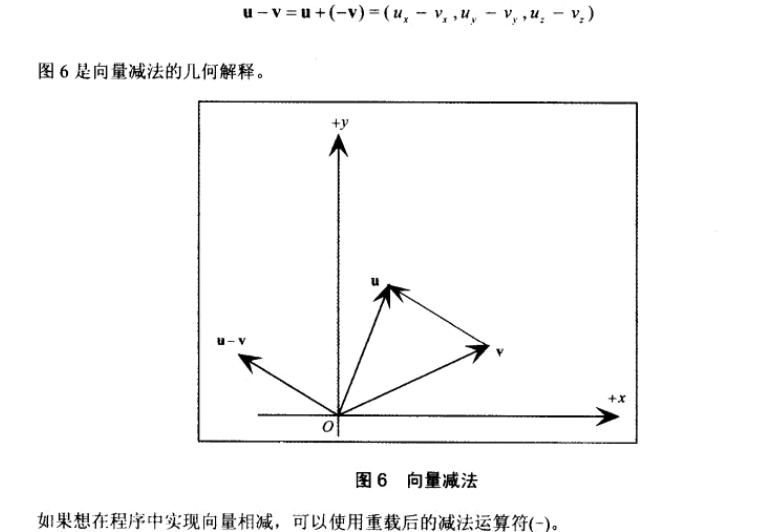


例如：

|  |
| --- |
| **D3DXVECTOR3 u(2.0f,0.0f,1.0f);**  **D3DXVECTOR3 v(0.0f,-1.0f,5.0f);**  **D3DXVECTOR3 res=u+v;//res((2.0f+0.0f),(0.0f-1.0f),(1.0f-5.0f))** |

## 6.向量的减法，等于向量的每一个成员相减的差构成的新向量

注意：两个向量只有维数相等才能相减！



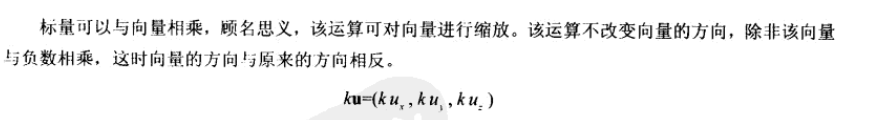
例如：

|  |
| --- |
| **D3DXVECTOR3 u(2.0f,0.0f,1.0f);**  **D3DXVECTOR3 v(0.0f,-1.0f,5.0f);**  **D3DXVECTOR3 deff=u-v; //deff((2.0f-0.0f),(0.0f-1.0f),(1.0f=5.5f))** |

## 7.标量与向量相乘

标量与向量相乘，等于标量与向量的每一个成员相乘得到一个新向量

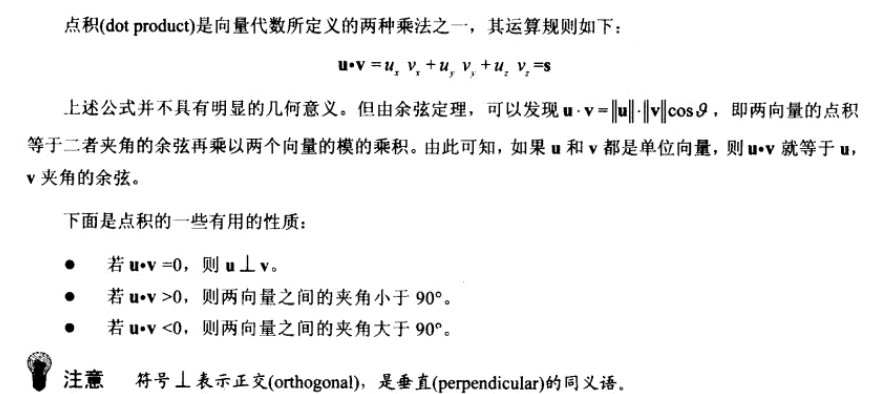
K\*u=（k\*ux,k\*uy,k\*uz）



## 8.向量的点乘

**规则：两个向量的点乘=两个向量对应的成员分别相乘后相加，结果是标量**

**U.v=ux.vx+uy.vy+uz.vz**



D3DX库中做向量点乘的函数：

**D3DXINLINE FLOAT D3DXVec3Dot( CONST D3DXVECTOR3 \*pV1, CONST D3DXVECTOR3 \*pV2 )**

**如果两个向量的点乘结果为0，相互那么他们是垂直的，夹角为90度**

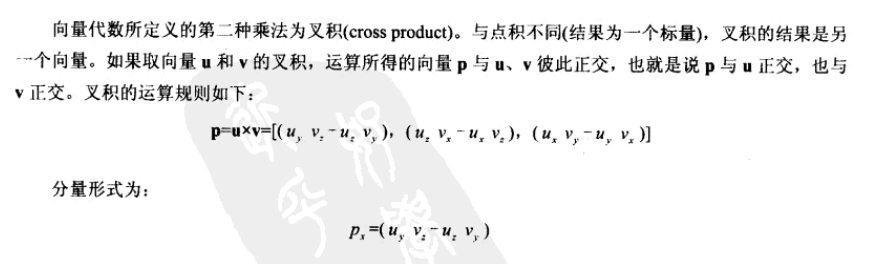
**如果两个向量的点乘结果大于0，相互那么他们的夹角小于90度**

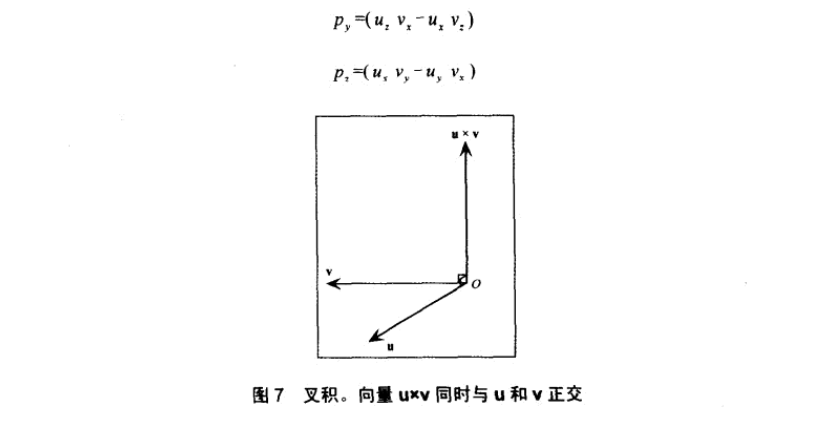
**如果两个向量的点乘结果小于0，相互那么他们的夹角大于90度**

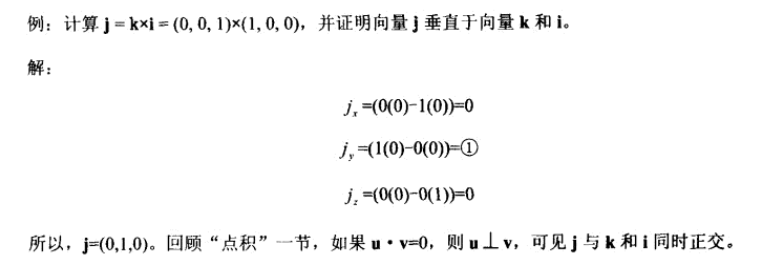
|  |
| --- |
| **D3DXVECTOR3 u(2.0f,0.0f,1.0f);**  **D3DXVECTOR3 v(0.0f,-1.0f,5.0f);**  **float dotRes= D3DXVec3Dot(u,v);** |

## 9.向量的叉乘（CrossProduct）

规则：p=UxV=[( UyVz-UzVy),(UzVx-UxVz),(UxVy-UyVx)]







D3DX库中有提供向量叉乘的函数：

**D3DXINLINE D3DXVECTOR3\* D3DXVec3Cross(/\*保存结果的标量\*/ D3DXVECTOR3 \*pOut, /\*左边向量\*/CONST D3DXVECTOR3 \*pV1, /\*右边向量\*/CONST D3DXVECTOR3 \*pV2 );**

