**向量代数**

**向量**

**What:**

向量是一种兼具大小和方向的量。

**Why:**

向量在计算机图形学、碰撞检测和物理模拟中扮演着关键的角色。

**How:**

**左手坐标系和右手坐标系**

**What:**

**Why:**

判断叉积方向

**How:**

**Direct3D采用的是左手坐标系还是右手坐标系？**

左手坐标系

**如何判断左手坐标系？**

如果我们伸出左手，并拢手指，假设它们指向的是轴的正方向，再弯曲四指指向轴的正方向，则最后伸直拇指的方向大约就是轴的正方向。

**点积**

**What:**

设向量u = (ux,uy,uz)，v = (vx,vy,vz)，则点积的定义为u·v = uxvx + uyvy + uzvz。

**Why:**

1，计算夹角；2，投影；3，规范正交化

**How:**

**点积的几何意义**

u·v = |u||v|cosθ

如果u和v都是单位向量：u·v = |u||v|cosθ

**正交投影**

为向量落在向量上的正交投影，通常将它表示为：

p = projn(v) = n(v·n)/|n|2

perpn(v) = v-p

**规范正交**

**What:**

如果向量集中的每个向量都是互相正交且皆具单位长度，就称此集合是规范正交的。

**Why:**

构建摄像机观察矩阵

**How:**

**如何创建规范正交集？**

1. 使用格拉姆-施密特正交化方法进行处理。

基本步骤：设w0=v0

对于i≤i≤n-I，令



规范化步骤：令wi = wi/|wi|

1. 通过叉积来进行正交化处理

基本步骤：1，令w0 = v0 / |v0|

2，令w2 = w0 x v1 / |w0 x v1|

3，令w1 = w2 x w0

**叉积**

**What:**

设向量u = (ux,uy,uz)，v = (vx,vy,vz)，则叉积的定义为

u x v = ( uyvz – uzvy, uzvx – uxvz, uxvy - uyvx)

**Why:**

1，求法向量；2，判断点在三角形中，3，规范正交化

**How:**

**叉积的方向**

两个3D向量u与v的叉积得到的是：既正交于u也正交于v的向量。如果伸出左手，使并拢的左手手指指向向量u的方向，再以的角度弯曲四指，使之指向向量v的方向，那么最后伸直的大拇指约略指向的即为w= u x v的方向。这就是所谓的左手定则。

**DirectXMath库**

**What:**

DirectXMath是一款为Direct3D量身打造的3D数学库，也是WindowsSDK的一部分。

**Why:**

该数学库采用了SIMD流指令扩展2指令集。借助128位宽的单指令多数据寄存器，利用一条SIMD指令即可同时对4个32位浮点数或整数进行运算。

**How:**

**为了使用DirectXMath库，需要哪些头文件和命名空间？**

DirectXMath.h和DirectXPackedVector.h

DirectX和DirectXPackedVector命名空间

**XMVECTOR**

**What:**

DirectXMath库中，核心的向量类型是XMVECTOR，它将被映射到SIMD硬件寄存器。

**Why:**

通过SIMD指令的配合，利用这种具有128位的类型能一次性处理4个32位的浮点数。

**How:**

**XMVECTOR和XMFLOATn使用方法**

1．局部变量或全局变量用XMVECTOR类型。

2．对于类中的数据成员，使用XMFLOAT2、XMFLOAT3和XMFLOAT4类型。

3．在运算之前，通过加载函数将XMFLOATn类型转换为XMVECTOR类型。

4．用XMVECTOR实例来进行运算。

5．通过存储函数将XMVECTOR类型转换为XMFLOATn类型。

**传递XMVECTOR参数的规则**

1．前3个XMVECTOR参数应当用类型FXMVECTOR；

2．第4个XMVECTOR参数应当用类型GXMVECTOR；

3．第5、6个XMVECTOR参数应当用类型HXMVECTOR；

4．其余的XMVECTOR参数应当用类型CXMVECTOR。

5．构造函数的前3个XMVECTOR参数用FXMVECTOR类型，其余XMVECTOR参数则用CXMVECTOR类型。

**常用的DirectXMath库函数**

**加载和存储方法**

将数据从XMFLOAT3类型中加载到XMVECTOR类型

XMVECTOR XM\_CALLCONV XMLoadFloat3(const XMFLOAT3 \*pSource);

将数据从XMVECTOR类型中存储到XMFLOAT3类型

void XM\_CALLCONV XMStoreFloat3(XMFLOAT3 \*pDestination, FXMVECTOR V);

XMVECTOR类型时，相关的存取方法如下：

float XM\_CALLCONV XMVectorGetX(FXMVECTOR V);

XMVECTOR XM\_CALLCONV XMVectorSetX(FXMVECTOR V, float x);

**Setter函数**

//返回零向量0

XMVECTOR XM\_CALLCONV XMVectorZero();

// 返回向量(1, 1, 1, 1)

XMVECTOR XM\_CALLCONV XMVectorSplatOne();

// 返回向量(x, y, z, w)

XMVECTOR XM\_CALLCONV XMVectorSet(float x, float y, float z, float w);

// 返回向量(Value, Value, Value, Value)

XMVECTOR XM\_CALLCONV XMVectorReplicate(float Value);

// 返回向量(vx, vx, vx, vx)

XMVECTOR XM\_CALLCONV XMVectorSplatX(FXMVECTOR V);

**向量函数**

// 返回|v|

XMVECTOR XM\_CALLCONV XMVector3Length(FXMVECTOR V);

// 返回|v|2

XMVECTOR XM\_CALLCONV XMVector3LengthSq(FXMVECTOR V);

// 返回v1·v2

XMVECTOR XM\_CALLCONV XMVector3Dot(FXMVECTOR V1,FXMVECTOR V2);

// 返回v1×v2

XMVECTOR XM\_CALLCONV XMVector3Cross(FXMVECTOR V1,FXMVECTOR V2);

// 返回v/||v||

XMVECTOR XM\_CALLCONV XMVector3Normalize(FXMVECTOR V);

// 返回一个正交于v的向量

XMVECTOR XM\_CALLCONV XMVector3Orthogonal(FXMVECTOR V);

// 返回v1和v2之间的夹角

XMVECTOR XM\_CALLCONV

XMVector3AngleBetweenVectors(FXMVECTOR V1,FXMVECTOR V2);

// 返回projn(v)和perpn(v)

void XM\_CALLCONV XMVector3ComponentsFromNormal(XMVECTOR\* pParallel,

XMVECTOR\* pPerpendicular,FXMVECTOR V, FXMVECTOR Normal);

// 返回v1 == v2？

bool XM\_CALLCONV XMVector3Equal(FXMVECTOR V1,FXMVECTOR V2);