第1章 向量代数

1，向量

向量由大小和方向决定，而与位置无关。

如何用代数方法来表示向量？通过平移操作使向量的尾部都位于原点，我们凭借向量头部的坐标来确定该向量。

如何判定左手坐标系？我们伸出左手，并拢手指，假设它们指向的是x轴的正方向，再弯曲四指指向y轴的正方向，则最后伸直拇指的方向大约就是z轴的正方向。

向量的基本运算：向量的相等，加法运算，标量乘法运算和减法运算。

向量加法运算和向量减法运算的几何意义。

2，长度和单位向量

把一个向量的长度变为单位长度称为向量的规范化处理。

3，点积

点积的代数表示：u\*v=uxvx + uyvy + uzvz

点积的几何关系：u\*v=|u||v|cosθ

当n为单位向量时，向量v落在向量n上的正交投影：projn(v) = (v\*n)n

向量perpn(v) = v - projn(v)时作用力v在n正交方向上的分力。

如果n不具有单位长度，projn(v) = (v\*n)n/|n|2

点积的两个应用：求夹角和计算投影。

正交化

如果向量集{v0,v1,……,vn-1}中的每个向量都是互相正交且皆具单位长度，那么我们就称此集合是规范正交的。

格拉姆-施密特正交化。

4，叉积

叉积的计算结果亦为向量。只有3D向量的叉积有定义，不存在2D向量叉积。

叉积的计算方法：w = u x v = (uyvz – uzvy, uzvx – uxvz, uxvy - uyvx)

两个3D向量u与v的叉积得到的是：即正交于u也正交于v的向量w。如果伸出左手，使并拢的左手手指指向向量u的方向，再以0≤θ≤π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。这就是所谓的左手拇指法则。

通过叉积来进行正交化处理：

1. 令w0 = v0 / |v0|；
2. 令w2 = w0 x v1 / | w0 x v1|；
3. 令w1 = w2 x w0

5，点

6，利用DirectXMath库进行向量运算

DirectXMath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。

例如，u + v = (ux + vx , uy + vx , uz + vz)，我们按普通的计算方式只能对分量逐个相加，而通过SIMD技术，我们可以仅仅通过一条SIMD加法指令来取代4条普通的标量指令。

为了使用DirectXMath库，我们需要向代码中添加头文件#include <DirectXMath.h>，而为了一些相关的数据类型还要加入头文件#include <DirectXPackedVector.h>。

DirectXMath.h文件中的代码都存在于DirectX命名空间中，而DirectXPackedVector.h文件中的代码则都位于DirectX::PackedVector命名空间中。

6.1 向量类型

在DirectXMath库中，核心的向量类型是XMVECTOR， 被映射到SIMD硬件寄存器。在开启SSE2后，此类型的定义是:typedef \_\_m128 XMVECTOR。

对于类中的数据成员，建议分别使用XMFLOAT2,XMFLOAT3和XMFLOAT4类型来加以代替。但是，如果直接把上述类型用于计算，却依然不能充分发挥出SIMD技术的高效特性。为此，我们还需要将这些类型的实例转换为XMVECTOR类型。

总结一下：

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或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。

6.2 加载方法和存储方法

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

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

float XM\_CALLCONV XMVectorGetX(FXMVECTOR V);

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

6.3 参数的传递

为了提高效率，可以将XMVECTOR类型的值作为函数的参数，直接传送至SSE/SSE2寄存器里，而不存于栈内。以此方式传递的参数数量取决于用户使用的平台和编译器。因此，为了使代码更具通用性，不受具体平台，编译器的影响，我们将利用FXMVECTOR，GXMVECTOR,HXMVECTOR和CXMVECTOR类型来传递XMVECTOR类型的参数。基于特定的平台和编译器，它们会被自动地定义为适当的类型。此外，一定要把调用约定注解XM\_CALLCONV加在函数名之前，它会根据编译器的版本确定出对应的调用约定属性。

传递XMVECTOR参数的规则如下：

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。

这些类型在32位Windows平台和编译器(编译器需要支持\_\_fastcall和新增的\_\_vectorcall调用约定)上的定义：

在32位的Windows系统上，编译器将根据\_\_fastcall调用约定将前3个XMVECTOR参数传递到寄存器中，而且其余参数都存在栈上

typedef const XMVECTOR FXMVECTOR

typedef const XMVECTOR& GXMVECTOR

typedef const XMVECTOR& HXMVECTOR

typedef const XMVECTOR& CXMVECTOR

在32位的Windows系统上，编译器将通过\_\_vectorcall调用约定将前6个XMVECTOR参数传递到寄存器中，而且其余参数都存在栈上

typedef const XMVECTOR FXMVECTOR

typedef const XMVECTOR GXMVECTOR

typedef const XMVECTOR HXMVECTOR

typedef const XMVECTOR& CXMVECTOR

在编写构造函数时，前3个XMVECTOR参数用FXMVECTOR类型，其余XMVECTOR参数则用CXMVECTOR类型。另外，对于构造函数不要使用XM\_CALLCONV注解。

6.4 常向量

XMVECTOR类型的常量实例应当用XMVECTORF32类型来表示。

数学库中还提供了将它转换至XMVECTOR类型的运算符。