1 向量代数

* 1. 向量

向量是一种兼具大小和方向的量。

用几何方法来描述向量的数学特征：通过图像中的一条有向线段即可表示一个向量，其中线段长度代表向量的模，箭头的指向代表向量的方向。向量的绘制位置之于其自身是无足轻重的。

* + 1. 向量与坐标系

计算机无法直接处理以几何方式表示的向量，需要寻求一种用数学表示向量的方法加以代替。通过平移操作使向量的尾部都位于原点，接着，我们就能凭借向量头部的坐标来确定该向量。

同一向量v在不同的坐标系中有着不同的坐标表示。

* + 1. 左手坐标系和右手坐标系

Direct3D采用的是左手坐标系。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。右手坐标系同理。

* + 1. 向量的基本运算
  1. 长度和单位向量

向量大小的几何意义是对应有向线段的长度。3D向量的模可通过运用两次毕达哥拉斯定理得出。

把一个向量的长度变为单位长度称为向量的规范化处理。具体实现方法是，将向量的每个分量分别除以该向量的模。

* 1. 点积

点积是一种计算结果为标量值的向量乘法运算。点积的运算公式(待续)。点积就是向量间对应分量的乘积之和。

点积的定义并没有明显地体现出其几何意义，但是我们能根据余弦定理找到两向量点积的几何关系：u·v=|u||v|cosθ。

当n是单位向量时，向量v落在向量n上的正交投影为：

projn(v)=(v·n)n

perpn(v)=v – projn(v)

如果n不具有单位长度，就先对它进行规范化。通过把向量n替换为单位向量n/|n|，即可得到根据一般性的投影公式：

proj(v) = (v·n)n/|n|2

正交化

如果向量集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。

我们可以使用格拉姆-施密特正交化方法进行规范正交化处理。在将给定集合内的向量vi添加到规范正交集中时，我们需要令vi减去它在现有规范正交集中其他向量{w0,w1,…,wi-1}方向上的分量，这样就可确保新加入规范正交集的向量与该集合中的其他向量互相正交。

* 1. 叉积

叉积的计算结果亦为向量。此外，只有3D向量的叉积有定义，不存在2D向量叉积。假设3D向量u和v的叉积得到的是另一个向量w，则w与向量u,v彼此正交。叉积的计算方法：

W=uxv=(uyvz – uzvy, uzvx – uxvz, uxvy – uyvx)

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=uxv的方向，这就是所谓的左手拇指法则。

一般来说，向量的叉积不满足交换律，但是满足反交换律。

* + 1. 2D向量的伪叉积
    2. 通过叉积来进行正交化处理

1，令w0 = v0/|v0|;

2，w2 = w0 x v1 / |w0 x v1|

3，w1 = w2 x w0

* 1. 点
  2. 利用DirectXMath库进行向量计算

DirectXMath是一款为Direct3D应用程序量身打造的3D数学库，它采用了SIMD流指令扩展2(SSE2)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。

* + 1. 向量类型

在DirectXMath库中，核心的向量类型是XMVECTOR，它将被映射到SIMD硬件寄存器。在开启SSE2后，此类型在x86和x64平台的定义是：

typedef \_\_m128 XMVECTOR;

这里的\_\_m128是一种特殊的SIMD类型。在计算向量的过程中，必须通过此类型才可充分地利用SIMD技术。

置于类中的数据成员，建议分别使用XMFLOAT2，XMFLOAT3和XMFLOAT4类型来加以代替。

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实例进行运算；
5. 通过存储函数将XMVECTOR类型转换为XMFLOATn类型。
   * 1. 加载方法和存储方法
     2. 参数的传递

为了提高效率，可以将XMVECTOR类型的值作为参数，直接传送至SSE/SSE2寄存器里，而不存在于栈内。以此方法传递的参数数量取决于用户使用的平台和编译器。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。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，它会更具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。

在32为的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，而把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为的Windows系统上，编译器将根据\_\_vectorcall调用约定将前6个XMVECTOR参数传递到寄存器中，而把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。

* + 1. 常向量

XMVECTOR类型的常量实例应当用XMVECTORF32类型来表示。

* + 1. 重载运算符
    2. 杂项
    3. Setter函数
    4. 向量函数
    5. 浮点数误差

DirectXMath库提供了XMVector3NearEqual函数，用于以Epsilon作为容差，测试比较的向量是否相等。