1 向量代数

* 1. 向量

左手坐标系和右手坐标系

Direct3D采用的是左手坐标系（left-handed coordinate system）。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[4]。

Unity是左手坐标系。OpenGL是右手坐标系。

* 1. 长度和单位向量
  2. 点积

点积的定义为：

u·v = uxvx + uyvy + uzvz

点积和向量的集合关系

u·v = |u||v|cosθ

p为向量v落在向量n上的正交投影，表示为：

p = projn(v)

如果n是单位向量

p = (v · n) n

如果n不是单位向量

p = (v · n) n / |n|2

正交化

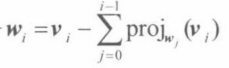
如果向量集{v0, …, vn-1}每个向量都是互相正交且皆具单位长度，那么我们就称此集合是规范正交的。

将向量集规范正交化：

格拉姆-施密特正交化：

基本步骤：设w0 = v0

对于1≤ I ≤ n-1，令



规范化步骤：令wi = wi / |wi|

1.4 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。

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。

叉积的计算方法为：

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

若实际采用的是左手坐标系，则遵守左手拇指法则(左手定则)：如果伸出左手并拢手指，令它们指向第一个向量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。

通过叉积来进行正交化处理

步骤：

1. 令w0 = v0 / |v0|
2. 令w2 = w0 x v1 / |w0 x v1|
3. 令w1 = w2 x w0
   1. 点
   2. 利用DirectXMath库进行向量计算

DirectXMath是为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。这对于向量运算带来的好处是不言而喻的。

* + 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。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。在开启SSE2后，此类型的定义为：

typedef \_\_m128 XMVERCTOR;

类中的数据成员，分别使用XMFLOAT2，XMFLOAT3和XMFLOAT4类型来加以代替。但是直接把上述这些类型用于计算，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。转换的过程可以通过DirectXMath库的加载函数实现。

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实例来进行运算；
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

为了使代码更具通用性，不受具体平台，编译器的影响，我们利用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。

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.
5. 在编写构造函数时，前3个XMVECTOR参数用FXMVECTOR**类型，其余**XMVECTOR参数用CXMVECTOR类型。
6. XMVECTOR类型的常量实例应当用XMVECTOR32类型来表示。
   * 1. 常向量
     2. 重载运算符
     3. 杂项
     4. Setter函数
     5. 向量函数
     6. 浮点数误差