**DirectXMath 库**

官方文档

http://msdn.microsoft.com/en-us/library/windows/desktop/hh437833(v=vs.85).aspx

头文件

#include <DirectXMath.h>

命名空间

using namespace DirectX

**向量类型**

核心类型为 XMVECTOR, 128 位, 映射到 SIMD 硬件寄存器, 可用一条 SIMD 指令 操作 4 个 32 位 float 类型

当 SIMD 可用时, XMVECTOR 如下定义

typedef \_m128 XMVECTOR;

XMVECTOR 需要 16 位对齐, 这对于局部的全局对象会自动完成, 对于类成员, 使用

XMFLOAT2, XMFLOAT3, XMFLOAT4 结构

typedef struct \_XMFLOAT2{

FLOAT x;

FALOT y;

}XMFLOAT2;

typedef struct \_XMFLOAT3{

FLOAT x;

FALOT y;

FALOT z;

} XMFLOAT3;

typedef struct \_XMFLOAT4{

FLOAT x;

FALOT y;

FALOT z;

FALOT w;

} XMFLOAT4;

为了在计算时利用 SIMD, 需要使用 XMVECTOR 类型

使用载入函数把 XMFLOAT? 转换为 XMVECTOR

使用存储函数吧XMVECTOR 转换为 XMFLOAT?

**载入和存储函数**

XMFLOAT? 载入到 XMVECTOR

//将一个 XMFLOAT2 类型载入到 XMVECTOR

XMVECTOR XMLoadFloat2 ( CONST XMFLOAT2 \*pSource );

//将一个 XMFLOAT3 类型载入到 XMVECTOR

XMVECTOR XMLoadFloat3 ( CONST XMFLOAT3 \*pSource );

//将一个 XMFLOAT4 类型载入到 XMVECTOR

XMVECTOR XMLoadFloat4 ( CONST XMFLOAT4 \*pSource );

XMVECTOR 存储到 XMFLOAT?

//存储一个 XMVECTOR 到 XMFLOAT2

VOID XMStoreFloat2( XMFLOAT2 \*pDestination, FXMVECTOR V );

//存储一个 XMVECTOR 到 XMFLOAT3

VOID XMStoreFloat3( XMFLOAT3 \*pDestination, FXMVECTOR V );

//存储一个 XMVECTOR 到 XMFLOAT4

VOID XMStoreFloat4( XMFLOAT4 \*pDestination, FXMVECTOR V );

其他一些类型也可以载入到 XMVECTOR

XMVECTOR XMLoadInt3 (CONST UINT\* pSource);

XMVECTOR XMLoadColor(CONST XMCOLOR \*pSource);

XMVECTOR XMLoadByte4(CONST XMBYTE4 \*pSource);

XMVECTOR 还能存储为其他类型

VOID XMStoreInt3(UINT\* pDestination, FXMVECTOR V);

VOID XMStoreColor(XMCOLOR\* pDestination, FXMVECTOR V);

VOID XMStoreByte4(XMBYTE4 \*pDestination, FXMVECTOR V);

如果只 get 或 set XMVECTOR 中的某一个元素

FLOAT XMVectorGetX ( FXMVECTOR V);

FLOAT XMVectorGetY ( FXMVECTOR V);

FLOAT XMVectorGetZ ( FXMVECTOR V);

FLOAT XMVectorGetW ( FXMVECTOR V);

XMVECTOR XMVectorSetX ( FXMVECTOR V, FLOAT x );

XMVECTOR XMVectorSetY ( FXMVECTOR V, FLOAT y );

XMVECTOR XMVectorSetZ ( FXMVECTOR V, FLOAT z );

XMVECTOR XMVectorSetW ( FXMVECTOR V, FLOAT w );

**参数传递**

XMVECTOR 作为参数传递的规则会因平台而变, 为了达到平台无关性, 在传递参数时 使用其别名 CXMVECTOR 和 FXMVECTOR. 这些类型会根据平台定义为正确的类型

//32 位 Windows

typedef const XMVECTOR FXMVECTOR;

typedef const XMVECTOR& CXMVECTOR;

//64 位 Windows

typedef const XMVECTOR& FXMVECTOR;

typedef const XMVECTOR& CXMVECTOR;

使用规则:　前三个 XMVECTOR 参数使用类型 FXMVECTOR, 任何额外的 XMVECTOR 使用类型 CXMVECTOR

XMINLINE XMMATRIX XMMatrixTransformation (

FXMVECTOR ScalingOrigin,

FXMVECTOR ScalingOrientationQuaternion,

FXMVECTOR Scaling,

CXMVECTOR RotationOrigin,

CXMVECTOR RotationQuaternion,

CXMVECTOR Translation);

**常量**

对于常量或者需预先赋值的 XMVECTOR类型, 使用 XMVECTORF32 或 XMVECTORU32 类型.

static const XMVECTORF32 g\_vHalfVector={0.5f, 0.5f, 0.5f, 0.5f};

static const XMVECTORU32 vGrabY = {

0x00000000,0xFFFFFFFF,0x00000000,0x00000000

};

XMVECTORF32 vRightTop = {

vViewFrust.RightSlope,

vViewFrust.TopSlope,

1.0f, 1.0f };

**运算符重载**

XMVECTOR 具有多个运算符重载函数. 可通过定义 XM\_NO\_OPERATOR\_OVERLOADS 关 闭运算符重载

#if defined(\_\_cplusplus) && !defined(XM\_NO\_OPERATOR\_OVERLOADS)

XMVECTOR operator+ (FXMVECTOR V);

XMVECTOR operator- (FXMVECTOR V);

XMVECTOR& operator+= (XMVECTOR& V1, FXMVECTOR V2);

XMVECTOR& operator-= (XMVECTOR& V1, FXMVECTOR V2);

XMVECTOR& operator\*= (XMVECTOR& V1, FXMVECTOR V2);

XMVECTOR& operator/= (XMVECTOR& V1, FXMVECTOR V2);

XMVECTOR& operator\*= (XMVECTOR& V, FLOAT S);

XMVECTOR& operator/= (XMVECTOR& V, FLOAT S);

XMVECTOR operator+ (FXMVECTOR V1, FXMVECTOR V2);

XMVECTOR operator- (FXMVECTOR V1, FXMVECTOR V2);

XMVECTOR operator\* (FXMVECTOR V1, FXMVECTOR V2);

XMVECTOR operator/ (FXMVECTOR V1, FXMVECTOR V2);

XMVECTOR operator\* (FXMVECTOR V, FLOAT S);

XMVECTOR operator\* (FLOAT S, FXMVECTOR V);

XMVECTOR operator/ (FXMVECTOR V, FLOAT S);

#endif

**杂项**

有关 π 的常量

#defineXM\_PI 3.141592654f

#defineXM\_2PI 6.283185307f

#defineXM\_1DIVPI 0.318309886f

#defineXM\_1DIV2PI 0.159154943f

#defineXM\_PIDIV2 1.570796327f

#defineXM\_PIDIV4 0.785398163f

角度和弧度的转换函数

XMFINLINE FLOAT XMConvertToRadians(FLOAT fDegrees)

{

return fDegrees \* (XM\_PI / 180.0f);

}

XMFINLINE FLOAT XMConvertToDegrees(FLOAT fRadians)

{

return fRadians \* (180.0f / XM\_PI);

}

判断大小的宏

#define XMMin(a, b) (((a) < (b)) ? (a) : (b))

#define XMMax(a, b) (((a) > (b)) ? (a) : (b))

**设置函数**

用于设置 XMVECTOR 的内容的函数

XMVECTOR XMVectorZero(); //返回 (0, 0, 0, 0)

XMVECTOR XMVectorSplatOne(); //返回 (1, 1, 1, 1)

XMVECTOR XMVectorSet(FLOAT x, FLOAT y, FLOAT z,FLOAT w); //返回 (x, y, z, w)

XMVECTOR XMVectorReplicate(FLOAT s); //返回 (s, s, s, s)

XMVECTOR XMVectorSplatX(FXMVECTOR V); //返回 (V.x,V.x,V.x,V.x)

示例

#include <Windows.h>

#include <DirectXMath.h>

#include <iostream>

using namespace DirectX;

std::ostream& operator<<(std::ostream& os, FXMVECTOR v)

{

XMFLOAT3 dest;

XMStoreFloat3(&dest, v);

os << "(" << dest.x << ", " << dest.y << ", " << dest.z << ")" << std::endl;

return os;

}

int main()

{

std::cout.setf(std::ios\_base::boolalpha);

//检查 CPU 是否支持 SSE2

//Pentinum 4 , AMD K8 或以上

if (!XMVerifyCPUSupport())

{

std::cout << "SSE2 not supported" << std::endl;

}

XMVECTOR v1 = XMVectorZero();

XMVECTOR v2 = XMVectorSplatOne();

XMVECTOR v3 = XMVectorSet(1.0f, 2.0f, 3.0f, 4.0f);

XMVECTOR v4 = XMVectorReplicate(-2.0f);

XMVECTOR v5 = XMVectorSplatZ(v3);

std::cout << v1 << std::endl;

std::cout << v2 << std::endl;

std::cout << v3 << std::endl;

std::cout << v4 << std::endl;

std::cout << v5 << std::endl;

}

**向量操作函数**

下面只列出 3D 版本的, 同样存在 2D 和 4D 版本, 只需把函数名的维度改成对应的

XMVECTOR XMVector3Length(FXMVECTOR V); //返回向量长度

XMVECTOR XMVector3LengthSq(FXMVECTOR V); //返回向量长度的平方

XMVECTOR XMVector3Dot(FXMVECTOR V1, FXMVECTOR V2); //返回V1 和 V2 的点积

XMVECTOR XMVector3Cross(FXMVECTOR V1, FXMVECTOR V2); //返回V1 和 V2 的叉积

XMVECTOR XMVector3Normalize(FXMVECTOR V); //规范化

XMVECTOR XMVector3Orthogonal(FXMVECTOR V); //返回一个与 V 垂直的向量

XMVector3AngleBetweenVectors(FXMVECTOR V1, FXMVECTOR V2); //返回 V1 与 V2 的角度

BOOL XMVector3Equal(FXMVECTOR V1, FXMVECTOR V2); //是否相等

BOOL XMVector3NotEqual(FXMVECTOR V1, FXMVECTOR V2); //是否不相等

XMFINLINE BOOL XMVector3NearEqual( FXMVECTOR U, FXMVECTOR V, FXMVECTOR Epsilon); //使用 Epsilon 判断是否相等

注意, 一些计算结果是标量的函数依然返回 XMVECTOR 类型, 在返回的 XMVECTOR 中每个元素都是同样的计算结果

如 XMVector3Dot ( v1 , v2 ) 结果为 ( v1\*v2, v1\*v2, v1\*v2, v1\*v2)

DirectXMath 还包含一些估计函数. 这些函数计算精度较低, 但速度更快

MFINLINE XMVECTOR XMVector3LengthEst(XMVECTOR V);

MFINLINE XMVECTOR XMVector3NormalizeEst(XMVECTOR V);