**矩阵类型**

使用 XMMATRIX 类表示 4x4 矩阵. 该类使用 4 个 XMVECTOR 类型存储数据以利用

SIMD. 定义如下

#if (defined(\_XM\_X86\_) || defined(\_XM\_X64\_)) && defined(\_XM\_NO\_IN TRINSICS\_)

typedef struct \_XMMATRIX

#else

typedef \_DECLSPEC\_ALIGN\_16\_ struct \_XMMATRIX

#endif

{

union

{

struct

{

FLOAT \_11, \_12, \_13, \_14;

FLOAT \_21, \_22, \_23, \_24;

FLOAT \_31, \_32, \_33, \_34;

FLOAT \_41, \_42, \_43, \_44;

};

FLOAT m[4][4];

}

}

#ifdef \_\_cplusplus

\_XMMATRIX() {};

\_XMMATRIX( FXMVECTOR R0, FXMVECTOR R1, FXMVECTOR R2, CXMVECTOR R3);

\_XMMATRIX( FLOAT m00, FLOAT m01, FLOAT m02, FLOAT m03,

FLOAT m10, FLOAT m11, FLOAT m12, FLOAT m13,

FLOAT m20, FLOAT m21, FLOAT m22, FLOAT m23,

FLOAT m30, FLOAT m31, FLOAT m32, FLOAT m33);

\_XMMATRIX(CONST FLOAT \*pArray);

FLOAT operator() (UINT Row, UINT Column) CONST { return m[Row][Column]; }

FLOAT& operator() (UINT Row, UINT Column) { return m[Row][Column]; }

\_XMMATRIX& operator= (CONST \_XMMATRIX& M);

#ifndef XM\_NO\_OPERATOR\_OVERLOADS

\_XMMATRIX& operator\*= (CONST \_XMMATRIX& M);

\_XMMATRIX operator\* (CONST \_XMMATRIX& M) CONST;

#endif

#endif

} XMMATRIX;

除了使用构造函数, 还可以使用 XMMatrixSet 函数创建 XMMATRIX 实例

XMMATRIX XMMatrixSet( FLOAT m00, FLOAT m01, FLOAT m02, FLOAT m03,

FLOAT m10, FLOAT m11, FLOAT m12, FLOAT m13,

FLOAT m20, FLOAT m21, FLOAT m22, FLOAT m23,

FLOAT m30, FLOAT m31, FLOAT m32, FLOAT m33);

作为类成员时, 使用 XMFLOAT4X4 类型存储数据

typedef struct \_XMFLOAT4X4

{

union

{

struct

{

FLOAT \_11, \_12, \_13, \_14;

FLOAT \_21, \_22, \_23, \_24;

FLOAT \_31, \_32, \_33, \_34;

FLOAT \_41, \_42, \_43, \_44;

};

FLOAT m[4][4];

};

#ifdef \_\_cplusplus

\_XMFLOAT4X4() {};

\_XMFLOAT4X4(FLOAT m00, FLOAT m01, FLOAT m02, FLOAT m03,

FLOAT m10, FLOAT m11, FLOAT m12, FLOAT m13,

FLOAT m20, FLOAT m21, FLOAT m22, FLOAT m23,

FLOAT m30, FLOAT m31, FLOAT m32, FLOAT m33);

\_XMFLOAT4X4(CONST FLOAT \*pArray);

FLOAT operator() (UINT Row, UINT Column) CONST { return m[Row][Column]; }

FLOAT& operator() (UINT Row, UINT Column) { return m[Row][Column]; }

\_XMFLOAT4X4& operator= (CONST \_XMFLOAT4X4& Float4x4);

#endif

} XMFLOAT4X4

**参数传递**

XMMATRIX 类型在作为参数传递时应使用类型 CXMMATRIX

#if defined(\_XM\_VMX128\_INTRINSICS\_)

typedef const XMMATRIX CXMMATRIX;

#elif defined(\_\_cplusplus)

typedef const XMMATRIX& CXMMATRIX;

**矩阵操作函数**

XMMATRIX XMMatrixIdentity(); //返回单位向量

BOOL XMMatrixIsIdentity( CXMMATRIX M ); //返回 TRUE 如果 M 是单位向量

XMMATRIX XMMatrixMultiply( CXMMATRIX A, CXMMATRIX B ); //返回 AB

XMMATRIX XMMatrixTranspose( CXMMATRIX M); //返回 M 的转置矩阵

XMVECTOR XMMatrixDeterminant( CXMMATRIX M ); //返回 (det M, det M, det M, det M)

XMMATRIX XMMatrixInverse( XMVECTOR\* pDeterminant, CXMMATRIX M); //返回 M 的逆矩阵

**示例**

#include <Windows.h>

#include <DirectXMath.h>

#include <iostream>

using namespace DirectX;

using namespace std;

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

{

XMFLOAT4 dest;

XMStoreFloat4(&dest, v);

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

return os;

}

std::ostream& operator<<(std::ostream& os, CXMMATRIX m)

{

XMFLOAT4X4 dest;

XMStoreFloat4x4(&dest, m);

for (int i = 0; i < 4; i++)

{

for (int j = 0; j < 4; j++)

{

os << dest(i, j) << "\t";

}

os << endl;

}

return os;

}

int main()

{

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

//检查 CPU 是否支持 SSE2

//Pentinum 4 , AMD K8 或以上

if (!XMVerifyCPUSupport())

{

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

}

XMMATRIX A ( 1.0f, 0.0f, 0.0f, 0.0f,

0.0f, 2.0f, 0.0f, 0.0f,

0.0f, 0.0f, 4.0f, 0.0f,

1.0f, 2.0f, 3.0f, 1.0f);

XMMATRIX B = XMMatrixIdentity();

XMMATRIX C = A \* B;

XMMATRIX D = XMMatrixTranspose(A);

XMVECTOR det = XMMatrixDeterminant(A);

XMMATRIX E = XMMatrixInverse(&det, A);

XMMATRIX F = A \* E;

cout << "A = " << endl << A << endl;

cout << "B = " << endl << B << endl;

cout << "C = A\*B = " << endl << C << endl;

cout << "D = transpose(A) = " << endl << D << endl;

cout << "det = determinant(A) = " << det << endl << endl;

cout << "E = inverse(A) = " << endl << E << endl;

cout << "F = A\*E = " << endl << F << endl;

return 0;

}