XMVECTORとXMFLOATどっち？問題

XMVECTOR

`XMVECTOR`は、DirectXMathライブラリの中核をなすデータ型です。その特徴は以下の通りです。

- **SIMD (Single Instruction, Multiple Data) 最適化**: 複数の浮動小数点数を同時に処理できるCPU命令（SSE/AVXなど）を利用するために設計されています。これにより、ベクトル演算や行列演算が非常に高速に行われます。
- **アラインメント**: SIMD命令が効率的に動作するために、通常16バイト境界にアラインメントされています。
- **レジスタでの操作**: 多くの場合、CPUのレジスタ上で直接操作されるため、メモリアクセスによるオーバーヘッドが少ないです。
- **直接的なアクセスが難しい**: `XMVECTOR`の個々の要素に直接アクセス（例: `vec.x`）することは通常できません。要素にアクセスするには、`XMVectorGetX()`, `XMVectorSetX()`などの関数を使用する必要があります。

`XMVECTOR`を使うべき場面:

- **ベクトルや行列の計算**: 加算、減算、乗算、内積、外積、変換など、あらゆる数学的な演算を行う場合。
- **シェーダーへの定数バッファのアップロード**: シェーダーで利用するベクトルや行列データは、通常`XMVECTOR`で計算し、`XMFLOAT4`などで構成された定数バッファに格納してからGPUへ送ります。
- **一時的な計算結果**: 関数内で中間的な計算を行う場合。

例:

C++

```
#include <DirectXMath.h>

// XMVECTOR同士の加算
XMVECTOR v1 = XMVectorSet(1.0f, 2.0f, 3.0f, 0.0f);
XMVECTOR v2 = XMVectorSet(4.0f, 5.0f, 6.0f, 0.0f);
XMVECTOR vResult = XMVectorAdd(v1, v2);

// 行列とベクトルの乗算
XMMATRIX m = XMMatrixRotationY(XM_PIDIV4);
XMVECTOR transformedVec = XMVector3TransformCoord(v1, m);
```

XMFLOAT (例: XMFLOAT3, XMFLOAT4)

`XMFLOAT`シリーズのデータ型（`XMFLOAT2`, `XMFLOAT3`, `XMFLOAT4`など）は、C++の通常の構造体として定義されています。

- **メモリ上でのデータ格納**: メモリ上にデータをシンプルに格納するために使用されます。
- **アクセスしやすい**: `float x, y, z, w;`のようにメンバが直接公開されているため、個々の要素にアクセスしやすいです。
- **アラインメントの制約が少ない**: 特別なアラインメントを考慮する必要がない場合が多いです（ただし、定数バッファなどGPUに送るデータの場合はアラインメントが重要になります）。
- **SIMD最適化はされない**: `XMFLOAT`自体はSIMD最適化の恩恵を受けません。

`XMFLOAT`を使うべき場面:

- **構造体やクラスのメンバ変数**: オブジェクトの位置、回転、スケールなど、永続的に保持したいデータ。
- **頂点バッファのレイアウト**: 頂点データ（位置、法線、UVなど）を定義する場合。
- **ファイルへの保存や読み込み**: データをファイルに書き込んだり、ファイルから読み込んだりする場合。
- **GUIでの表示など、個々の要素へのアクセスが必要な場合**。
- **`XMVECTOR`での計算結果の格納**: `XMVECTOR`で計算した結果をメモリに保存する際、`XMStoreFloat3()`や`XMStoreFloat4()`関数を使用して`XMFLOAT`型に変換します。

例:

C++

```
#include <DirectXMath.h>

// 構造体のメンバとして位置を保持
struct GameObject {
    XMFLOAT3 position;
    XMFLOAT4 color;
    // ...
};

GameObject player;
player.position = XMFLOAT3(0.0f, 1.0f, 0.0f);
player.color = XMFLOAT4(1.0f, 0.0f, 0.0f, 1.0f);

// XMVECTORの計算結果をXMFLOAT3に格納
XMVECTOR vec = XMVectorSet(10.0f, 20.0f, 30.0f, 0.0f);
XMFLOAT3 resultFloat3;
XMStoreFloat3(&resultFloat3, vec); // ここで変換
```

変換について

`XMVECTOR`と`XMFLOAT`の間は、以下の関数を使って相互に変換できます。

- **`XMFLOAT`→`XMVECTOR`:**
  - `XMLoadFloat2()`
  - `XMLoadFloat3()`
  - `XMLoadFloat4()`
  - `XMLoadFloat3x3()` (行列用)
  - `XMLoadFloat4x4()` (行列用) これらの関数は、`XMFLOAT`型のデータを`XMVECTOR`型（または`XMMATRIX`型）にロードし、SIMD演算の準備をします。
- **`XMVECTOR`→`XMFLOAT`:**
  - `XMStoreFloat2()`
  - `XMStoreFloat3()`
  - `XMStoreFloat4()`
  - `XMStoreFloat3x3()`
  - `XMStoreFloat4x4()` これらの関数は、`XMVECTOR`型の計算結果を`XMFLOAT`型のメモリ領域にストアします。

まとめと推奨されるフロー

DirectXプログラミングにおける理想的なデータフローは以下のようになります。

1. **データ格納:** オブジェクトの状態（位置、回転、色など）は、クラスや構造体のメンバとして**`XMFLOAT`**型で保持します。
2. **計算開始:** 計算が必要になったら、**`XMLoadFloat`**関数を使用して**`XMFLOAT`**型のデータを**`XMVECTOR`**型にロードします。
3. **計算実行:** **`XMVECTOR`**型を使って、DirectXMathの関数（`XMVectorAdd`, `XMMatrixMultiply`など）で高速なSIMD演算を行います。
4. **計算結果の格納:** 計算が終わったら、**`XMStoreFloat`**関数を使用して**`XMVECTOR`**型の結果を再び**`XMFLOAT`**型にストアし、永続化します。

Revision #1
★Created 1 July 2025 05:19:53 by youe2
✎Updated 13 June 2026 03:19:12 by youe2