I. Mathematical Prerequisites
2. Matrix Algebra

Game Graphic Programming
Kyung Hee University
Software Convergence
Prof. Daeho Lee

Matrices

- An $m \times n$ matrix $\mathbf{M}$ is a rectangular array of real numbers with m rows and n columns.
- Matrix multiplication
  - If $\mathbf{A}$ is a $m \times n$ matrix and $\mathbf{B}$ is a $n \times p$ matrix, then the product $\mathbf{AB}$ is defined and is a $m \times p$ matrix $\mathbf{C}$, where the ijth entry of the product $\mathbf{C}$ is given by taking the dot product of the ith row vector in $\mathbf{A}$ with the jth column vector in $\mathbf{B}$.

$$\mathbf{C}_{ij} = \mathbf{A}_{i,*} \cdot \mathbf{B}_{*,j}$$

- Associativity
  - Matrix multiplication has some algebraic properties.
    - Matrix multiplication distributes over addition: $\mathbf{A}(\mathbf{B} + \mathbf{C}) = \mathbf{AB} + \mathbf{AC}$ and $(\mathbf{A} + \mathbf{B})\mathbf{C} = \mathbf{AC} + \mathbf{BC}$.
    - The associative law of matrix multiplication from time to time, which allows us to choose the order we multiply matrices: $(\mathbf{AB})\mathbf{C} = \mathbf{A}(\mathbf{BC})$

DirectXMath uses **row-major matrices**, **row vectors**, and **pre-multiplication**.
Handedness is determined by which function version is used (RH vs. LH).

Transpose and Identity

- The transpose of a matrix is found by interchanging the rows and columns of the matrix.
- The transpose of an $m \times n$ matrix is an $n \times m$ matrix: the transpose of a matrix $\mathbf{M}$ is denoted by $\mathbf{M}^T$.

- The identity matrix is a square matrix that has zeros for all elements except along the main diagonal; the elements along the main diagonal are all ones.

Determinant

- The determinant of a square matrix $\mathbf{A}$ is commonly denoted by $\det(\mathbf{A})$, $\det \mathbf{A}$, or $|\mathbf{A}|$.
- It can be shown that the determinant has a geometric interpretation related to volumes of boxes and that the determinant provides information on how volumes change under linear transformations.
  - Given an $n \times n$ matrix $\mathbf{A}$, the minor matrix $\bar{\mathbf{A}}_{ij}$ is the $(n-1)\times(n-1)$ matrix found by deleting the ith row and jth column of $\mathbf{A}$.

$$\det \mathbf{A} = \sum_{j=1}^{n} A_{1j} (-1)^{1+j} \det \bar{A}_{1j}$$

$$\det \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} = a_{11} \det\left(a_{22}\right) - a_{12} \det\left(a_{21}\right) = a_{11}a_{22} - a_{12}a_{21}$$

Kyung Hee University
nize@khu.ac.kr

Adjoint and Inverse

- $\mathbf{A}$ is an $n \times n$ matrix.
  - Cofactor of $\mathbf{A}_{ij}$:

$$C_{ij} = (-1)^{i+j} \det \bar{A}_{ij}$$

  - Cofactor matrix of $\mathbf{A}$:

$$\mathbf{C}_{\mathbf{A}} = \begin{pmatrix} C_{11} & C_{12} & \cdots & C_{1n} \\ C_{21} & C_{22} & \cdots & C_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ C_{n1} & C_{n2} & \cdots & C_{nn} \end{pmatrix}$$

  - Adjoint of $\mathbf{A}$:

$$\operatorname{adj}(\mathbf{A}) = \mathbf{C}_{\mathbf{A}}^{T}$$

- Inverse matrix of $\mathbf{A}$:

$$\mathbf{A}^{-1} = \frac{\operatorname{adj}(\mathbf{A})}{\det(\mathbf{A})}$$

struct XMMATRIX

```
#ifdef _XM_NO_INTRINSICS_
    struct XMMATRIX
#else
    XM_ALIGNED_STRUCT(16) XMMATRIX
#endif
    {
#ifdef _XM_NO_INTRINSICS_
        union {
            XMVECTOR r[4];
            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];
        };
#else
        XMVECTOR r[4];
#endif
        XMMATRIX() = default;
   /* … */
};
```

struct XMFLOAT4X4 (1)

- A 4*4 floating-point matrix.
  - **XMMATRIX** uses four **XMVECTOR** instances to use SIMD.
  - It is recommended, by the DirectXMath documentation to use the **XMFLOAT4X4** type to store matrices as class data members.

struct XMFLOAT4X4 (2)

```
struct XMFLOAT4X4 {
  union {
    struct {float _11;      float _12;      float _13;      float _14;
      float _21;      float _22;      float _23;      float _24;
      float _31;      float _32;      float _33;      float _34;
      float _41;      float _42;      float _43;      float _44;
    };
    float m[4][4];
  };
  void         XMFLOAT4X4();
  void         XMFLOAT4X4(const XMFLOAT4X4 & unnamedParam1);
  XMFLOAT4X4 & operator=(const XMFLOAT4X4 & unnamedParam1);
  void         XMFLOAT4X4(XMFLOAT4X4 && unnamedParam1);
  XMFLOAT4X4 & operator=(XMFLOAT4X4 && unnamedParam1);
  void         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) noexcept;
  void         XMFLOAT4X4(const float *pArray) noexcept;
  float        operator()(size_t Row,    size_t Column) noexcept;
  float &      operator()(size_t Row,    size_t Column) noexcept;
  bool         operator==(const XMFLOAT4X4 & unnamedParam1);
  auto         operator<=>(const XMFLOAT4X4 & unnamedParam1);
};
```

struct XMFLOAT4X4 (3)

```
// Loads an XMFLOAT4X4 into an XMMATRIX.
XMMATRIX XM_CALLCONV XMLoadFloat4x4(
  [in] const XMFLOAT4X4 *pSource
) noexcept;

// Stores an XMMATRIX in an XMFLOAT4X4.
void XM_CALLCONV XMStoreFloat4x4(
  [out] XMFLOAT4X4 *pDestination,
  [in]  FXMMATRIX  M
) noexcept;
```


Kyung Hee University
nize@khu.ac.kr

DirectX Math Matrix Code (1)

```
#include <windows.h> // for XMVerifyCPUSupport
#include <DirectXMath.h>
#include <DirectXPackedVector.h>
#include <iostream>
using namespace std;
using namespace DirectX;
using namespace DirectX::PackedVector;

// Overload the  "<<" operators so that we can use cout to
// output XMVECTOR and XMMATRIX objects.
ostream& XM_CALLCONV operator << (ostream& os, FXMVECTOR v) {
    XMFLOAT4 dest;                XMStoreFloat4(&dest, v);
    os << "(" << dest.x << ", " << dest.y << ", " << dest.z << ", "
       << dest.w << ")";
    return os;
}
```

DirectX Math Matrix Code (2)

```
ostream& XM_CALLCONV operator << (ostream& os, FXMMATRIX m) {
    for (int i = 0; i < 4; ++i)     {
        os << XMVectorGetX(m.r[i]) << "\t";
        os << XMVectorGetY(m.r[i]) << "\t";
        os << XMVectorGetZ(m.r[i]) << "\t";
        os << XMVectorGetW(m.r[i]);
        os << endl;
    }
    return os;
}
```

DirectX Math Matrix Code (3)

```
int main() {
    // Check support for SSE2 (Pentium4, AMD K8, and above).
    if (!XMVerifyCPUSupport()) {
        cout << "directx math not supported" << endl;
        return 0;
    }

    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;
```

DirectX Math Matrix Code (4)

```
    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;
}
```

DirectX Math Matrix Code (5)

```
A =
1       0       0       0
0       2       0       0
0       0       4       0
1       2       3       1

B =
1       0       0       0
0       1       0       0
0       0       1       0
0       0       0       1

C = A*B =
1       0       0       0
0       2       0       0
0       0       4       0
1       2       3       1
```


Kyung Hee University
nize@khu.ac.kr

DirectX Math Matrix Code (6)

```
D = transpose(A) =
1       0       0       1
0       2       0       2
0       0       4       3
0       0       0       1

det = determinant(A) = (8, 8, 8, 8)

E = inverse(A) =
1       0       0       0
0       0.5     0       0
0       0       0.25    0
-1      -1      -0.75   1

F = A*E =
1       0       0       0
0       1       0       0
0       0       1       0
0       0       0       1
```