Matrices

A matrix is a 2D array of numbers, arranged in rows that go across and columns that go down:

A column:

3 rows

4 columns

Matrix sizes are termed “#rows x #columns”, so this is a 3x4 matrix
Matrix Transpose

A matrix transpose is formed by interchanging the rows and columns:

\[
\begin{bmatrix}
1 & 2 & 3 & 4 \\
5 & 6 & 7 & 8 \\
9 & 10 & 11 & 12
\end{bmatrix}^T = \begin{bmatrix}
1 & 5 & 9 \\
2 & 6 & 10 \\
3 & 7 & 11 \\
4 & 8 & 12
\end{bmatrix}
\]

This is a 3x4 matrix

This is a 4x3 matrix

Square Matrices

A square matrix has the same number of rows and columns

\[
\begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9
\end{bmatrix}
\]

This is a 3x3 matrix
**Row and Column Matrices**

A matrix can have a single row (a “row matrix”) or just a single column (a “column matrix”)

\[
\begin{bmatrix}
1 & 2 & 3
\end{bmatrix}
\quad \begin{bmatrix}
4 \\
5 \\
6
\end{bmatrix}
\]

This is a 1x3 matrix

This is a 3x1 matrix

Sometimes these are called row and column vectors, but that overloads the word “vector” and we won’t do it…

---

**Matrix Multiplication**

The basic operation of matrix multiplication is to pair-wise multiply a single row by a single column

\[
\begin{bmatrix}
1 & 2 & 3
\end{bmatrix} \begin{bmatrix}
4 \\
5 \\
6
\end{bmatrix} \rightarrow 4 \times 1 + 5 \times 2 + 6 \times 3 \rightarrow 32
\]

\[
\begin{bmatrix}
4 \\
5 \\
6
\end{bmatrix}
\]

1x3 3x1 1x1
Matrix Multiplication

Two matrices, A and B, can be multiplied if the number of columns in A equals the number of rows in B. The result is a matrix that has the same number of rows as A and the same number of columns as B.

\[
\begin{bmatrix}
1 & 2 & 3
\end{bmatrix}
\begin{bmatrix}
4 \\
5 \\
6
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
32
\end{bmatrix}
\]

Matrix Multiplication in Software

Here's how to remember how to do it:

1. \( C = A \times B \)

\[
\begin{bmatrix}
1 & 2 & 3
\end{bmatrix}
\begin{bmatrix}
4 \\
5 \\
6
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
32
\end{bmatrix}
\]

2. \( [I \times J] = [I \times K] \times [K \times J] \)

\[
\begin{bmatrix}
C[i][j]
\end{bmatrix} = \begin{bmatrix}
A[i][k]
\end{bmatrix} \times \begin{bmatrix}
B[k][j]
\end{bmatrix} ;
\]
Matrix Multiplication in Software

\[
\begin{bmatrix}
1 & 2 & 3 \\
\end{bmatrix}
\begin{bmatrix}
4 & 5 & 6 \\
\end{bmatrix}
= \begin{bmatrix}
32 \\
\end{bmatrix}
\]

for( int i = 0; i < numArows; i++ )
{
    for( int j = 0; j < numBcols; j++ )
    {
        C[i][j] = 0.;
        for( int k = 0; k < numAcols; k++ )
        {
            C[i][j] += A[i][k] * B[k][j];
        }
    }
}

Note: numAcols must == numBrows!

Note that:
C[i][j] = 0.;
for( int k = 0; k < numAcols; k++ )
{
    C[i][j] += A[i][k] * B[k][j];
}

Is like saying:
Matrix Multiplication where B and C are Column Matrices

```cpp
for( int i = 0; i < numArows; i++ )
{
    C[ i ] = 0.;
    for( int k = 0; k < numAcols; k++ )
    {
        C[ i ] += A[ i ][ k ] * B[ k ];
    }
}
```

To help you remember this, think of the “C[i]” lines as:

```cpp
C[ i ][ 0 ] = 0.;
...
C[ i ][ 0 ] += A[ i ][ k ] * B[ k ][ 0 ];
```

A Special Matrix

Consider the matrix * column situation below:

\[
\begin{pmatrix}
C_x \\
C_y \\
C_z
\end{pmatrix} =
\begin{bmatrix}
0 & -A_z & A_y \\
A_z & 0 & -A_x \\
-A_y & A_x & 0
\end{bmatrix}
\begin{bmatrix}
B_x \\
B_y \\
B_z
\end{bmatrix}
\]

This gives:

\[
C = (A_y B_z - A_z B_y, A_z B_x - A_x B_z, A_x B_y - A_y B_x)
\]

Which you hopefully recognize as the Cross Product \( A \times B \)
The determinant is important in graphics applications. It represents sort of a “scale factor”, when the matrix is used to represent a transformation.

The determinant of a 2x2 matrix is easy:

$$\begin{vmatrix} A & B \\ C & D \end{vmatrix} = A \cdot D - B \cdot C$$

The determinant of a 3x3 matrix is done in terms of its component 2x2 sub-matrices:

$$\begin{vmatrix} A & B & C \\ D & E & F \\ G & H & I \end{vmatrix} =$$

$$A \cdot \begin{vmatrix} E & F \\ H & I \end{vmatrix} - B \cdot \begin{vmatrix} D & F \\ G & I \end{vmatrix} + C \cdot \begin{vmatrix} D & E \\ G & H \end{vmatrix}$$

$$= A \cdot (EI - FH) - B \cdot (DI - FG) + C \cdot (DH - EG)$$
Inverses

The matrix inverse is also important in graphics applications because it represents the undoing of the original transformation matrix. It is also useful in solving systems of simultaneous equations.

The inverse of a 2x2 matrix is the transpose of the cofactor matrix divided by the determinant:

\[
\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} = \frac{1}{A*D - B*C} \begin{bmatrix} D & -B \\ -C & A \end{bmatrix}
\]

The determinant of a 3x3 matrix is done in terms of its component 2x2 sub-matrices:

\[
\begin{bmatrix} A & B & C \\ D & E & F \\ G & H & I \end{bmatrix}^{-1} = \frac{1}{\text{det}} \begin{bmatrix} \text{det} \begin{bmatrix} E & F \\ H & I \end{bmatrix} & \text{det} \begin{bmatrix} D & F \\ G & I \end{bmatrix} & \text{det} \begin{bmatrix} D & E \\ G & H \end{bmatrix} \\ \text{det} \begin{bmatrix} B & C \\ H & I \end{bmatrix} & \text{det} \begin{bmatrix} A & C \\ G & I \end{bmatrix} & \text{det} \begin{bmatrix} A & B \\ G & H \end{bmatrix} \\ \text{det} \begin{bmatrix} B & C \\ E & F \end{bmatrix} & \text{det} \begin{bmatrix} A & C \\ D & F \end{bmatrix} & \text{det} \begin{bmatrix} A & B \\ D & E \end{bmatrix} \end{bmatrix}
\]

The determinant of 4x4 and larger matrices can be done in a similar way, but usually isn’t. Gauss Elimination is more efficient.
Sidebar: The i-j-k order doesn’t matter as long as the “C[i][j] +=” line is right – different ordering affects performance

```cpp
for( int i = 0; i < numArows; i++ )
{
    for( int j = 0; j < numBcols; j++ )
    {
        for( int k = 0; k < numAcols; k++ )
        {
            C[i][j] += A[i][k] * B[k][j];
        }
    }
}
```

We’ll talk about this in CS 475/575 – Parallel Programming
Performance vs. Number of Threads (MegaMultiples / Sec)

- $k_j$
- $i_jk$
- $j-k$
- $j-k_i$
- $k_{j-i}$
- $k_{i-j}$

Graphs showing the relationship between performance and the number of threads for different operations.