Matrices

Oregon State University

Mike Bailey

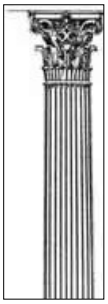
mjb@cs.oregonstate.edu


```
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 ];
                }
        }
}
```


Matrices

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

A column:



4 columns

3 rows

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}$$

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

Oregon State University
Computer Graphics

Square Matrices

A square matrix has the same number of rows and columns

3 columns

3 rows

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

This is a 3x3 matrix

Oregon State University
Computer Graphics

Row and Column Matrices

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

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$$

This is a 1x3 matrix

$$\begin{Bmatrix} 4 \\ 5 \\ 6 \end{Bmatrix}$$

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

$$\begin{matrix} 4 & 5 & 6 \\ * & * & * \\ \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \end{matrix} * \begin{Bmatrix} 4 \\ 5 \\ 6 \end{Bmatrix} \longrightarrow 4*1 + 5*2 + 6*3 \longrightarrow \mathbf{32}$$

A **B** **C**

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.

$$\underset{\mathbf{A}}{\begin{bmatrix} [1 \quad 2 \quad 3] \end{bmatrix}} * \underset{\mathbf{B}}{\begin{bmatrix} \begin{Bmatrix} 4 \\ 5 \\ 6 \end{Bmatrix} \end{bmatrix}} \Rightarrow \underset{\mathbf{C}}{\begin{bmatrix} 32 \end{bmatrix}}$$

I x K K x J I x J

Oregon State University
Computer Graphics

Matrix Multiplication in Software

Here's how to remember how to do it:

$$\underset{\mathbf{A}}{\begin{bmatrix} [1 \quad 2 \quad 3] \end{bmatrix}} * \underset{\mathbf{B}}{\begin{bmatrix} \begin{Bmatrix} 4 \\ 5 \\ 6 \end{Bmatrix} \end{bmatrix}} \Rightarrow \underset{\mathbf{C}}{\begin{bmatrix} 32 \end{bmatrix}}$$

1. **C = A * B**
2. **[I x J] = [I x K] * [K x J]**

I x J = I x K K x J

```
C[ i ][ j ] =     A[ i ][ k ] * B[ k ][ j ] ;
```

Oregon State University
Computer Graphics

Matrix Multiplication in Software

$$\underset{A}{\begin{bmatrix} [1 \;\; 2 \;\; 3] \end{bmatrix}} * \underset{B}{\begin{bmatrix} \begin{Bmatrix} 4 \\ 5 \\ 6 \end{Bmatrix} \end{bmatrix}} \Rightarrow \underset{C}{\begin{bmatrix} 32 \end{bmatrix}}$$

```
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 ];
                }
        }
```

Oregon State University
Computer Graphics

Note: numAcols *must* == numBrows !

Matrix Multiplication in Software

$$\underset{A}{\begin{bmatrix} [1 \;\; 2 \;\; 3] \end{bmatrix}} * \underset{B}{\begin{bmatrix} \begin{Bmatrix} 4 \\ 5 \\ 6 \end{Bmatrix} \end{bmatrix}} \Rightarrow \underset{C}{\begin{bmatrix} 32 \end{bmatrix}}$$

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:

```
C[ i ][ j ] = A[i][0] * B[0][j]  +  A[i][1] * B[1][j]  +  A[i][2] * B[2][j]  +  A[i][3] * B[3][j] ;
```

Oregon State University
Computer Graphics

Matrix Multiplication where B and C are Column Matrices

```
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:

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

Oregon State University
Computer Graphics

A Special Matrix

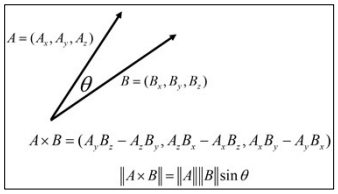
Consider the matrix * column situation below:

$$\begin{Bmatrix} C_x \\ C_y \\ C_z \end{Bmatrix} = \begin{bmatrix} 0 & -A_z & A_y \\ A_z & 0 & -A_x \\ -A_y & A_x & 0 \end{bmatrix} \begin{Bmatrix} B_x \\ B_y \\ B_z \end{Bmatrix}$$

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 is actually the **Cross Product** *AxB:*



Oregon State University
Computer Graphics

Determinants

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:

$$\det\begin{bmatrix} A & B \\ C & D \end{bmatrix} = A * D - B * C$$

Oregon State University
Computer Graphics

Determinants

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

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

$$A * \det\begin{bmatrix} E & F \\ H & I \end{bmatrix} - B * \det\begin{bmatrix} D & F \\ G & I \end{bmatrix} + C * \det\begin{bmatrix} D & E \\ G & H \end{bmatrix}$$

$$= A * (EI - FH) - B * (DI - FG) + C * (DH - EG)$$

Oregon State University
Computer Graphics

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:

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

Inverses

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

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

The determinant of 4x4 and larger matrices can be done in a similar way, but usually isn't. Gauss Elimination is more efficient.

17

Sidebar: The i-j-k order doesn't matter as long as the "C[i][j] +=" line is right – different ordering affects performance

```
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 ];
                }
        }
}
```

Oregon State University
Computer Graphics

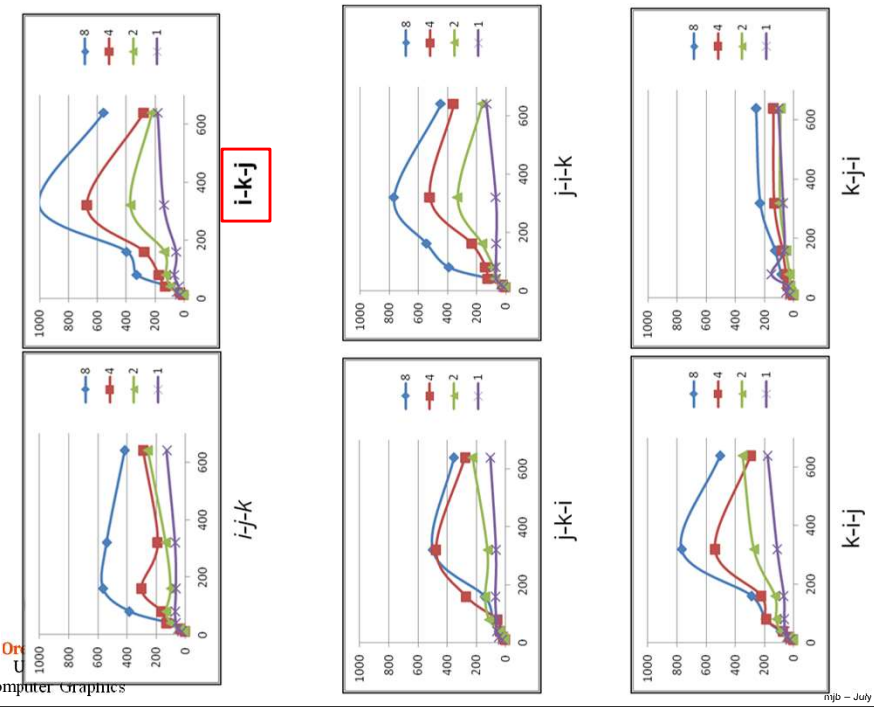
We'll talk about this in CS 475/575 – Parallel Programming

mjb – July 30, 2021

17

18

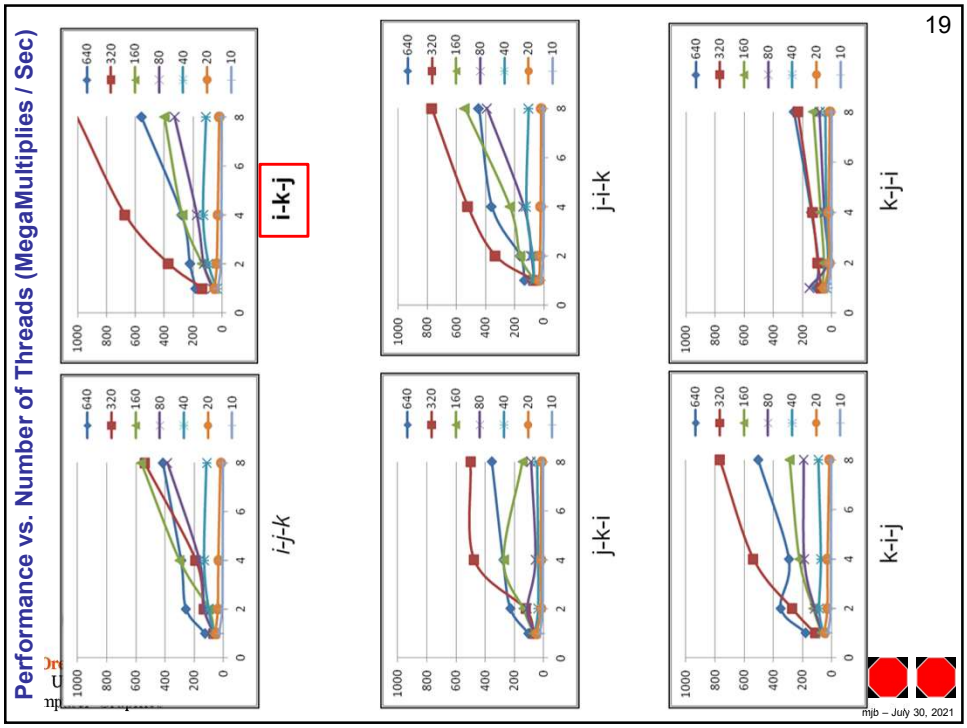
Performance vs. Matrix Size (MegaMultiplies / Sec)



Oregon State University
Computer Graphics

mjb – July 30, 2021

18


Performance vs. Number of Threads (MegaMultiplies / Sec)
19
i-j-k
i-k-j
j-k-i
j-i-k
k-i-j
k-j-i
640
320
160
80
40
20
10
1000
800
600
400
200
0
0
2
4
6
8
mjb – July 30, 2021

19