Computer Graphics

Basic Mathematical Background

Fernando Geraci

Subtopics

Conventions

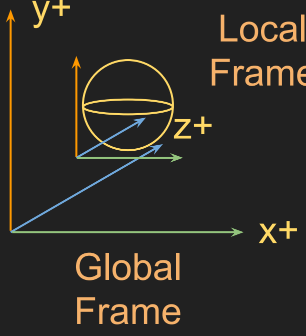
From the get go, we use:

- Left handed coordinates system
- Row vectors
- Visual Studio 2015 Community and Direct3D
- Windows Universal Applications framework

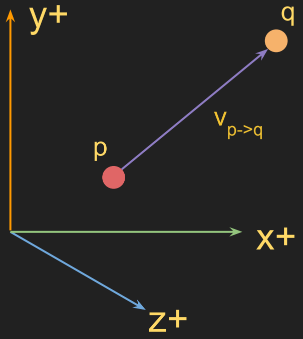
Coordinate System

Left vs. Right handed coordinates systems, A.K.A reference frames

Left Handed



Right Handed



Points vs Vectors

Points, express position, and these, can also we called “Position Vectors”. The tip of a position vector, given by p(x,y,z), express a physical location relative to its frame of reference

When dealing with points, we must be careful to understand these are NOT vectors, but some functionalities might be extended:

Given two points, p and q, we could:

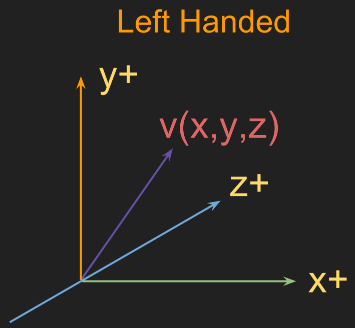
1) Find the vector between from point p to q → $v_{p\text{->}q} = q - p$
2) Find a point p by adding a vector v to the point q → $p = v_{p\text{->}q} + q$

Vectors

Geometrically speaking, is a line segment with two primary qualities:

Magnitude	(length)	-	given a vector v, is expressed ||v||
Direction	(aim)		

A vector is NOT affected when translated within a coordinates space.



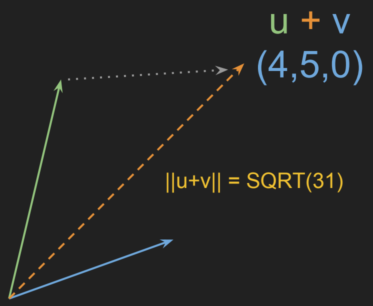
Length is given by the Pythagorean Theorem - SQRT(v1^2 + … + vn^2)

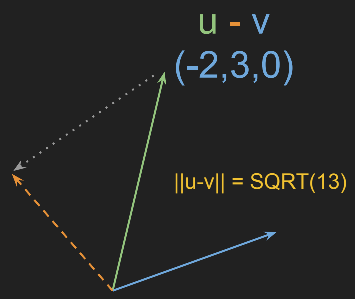
Basic Operations / Properties

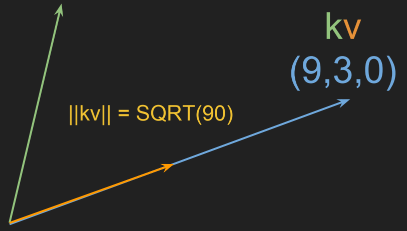
Addition	v + u	= u + v = (v1 + u1, … , vn + un)
Scalar Multiplication	kv	= (kv1, ... , kvn)
Subtraction	v - u	= v + (-1)u

Fernando Geraci

Vectors - Addition, Subtraction, Scalar Multiplication







u(1,4,0)
v(3,1,0)

then u + v

(1+3,4+1,0+0)
= (4,5,0)

u(1,4,0)
v(3,1,0)

then u - v

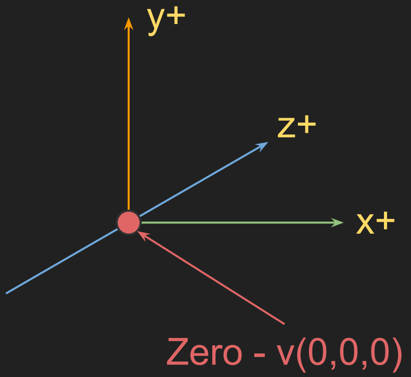
(1-3,4-1,0-0)
= (-2,3,0)

v(3,1,0)
k = 3

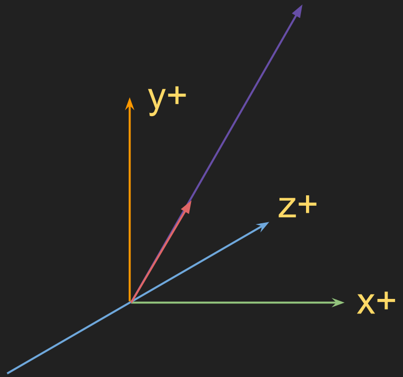
then kv

(3*3,1*3,0*3)
= (9,3,0)

Vectors - Zero Vector and Unit Vectors



A vector is said to be the zero vector when all its components equal 0.



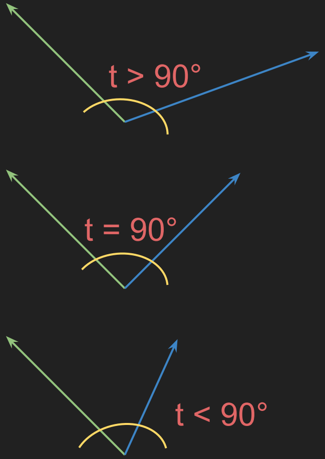
A vector is said to be a “unit” vector when its length is exactly 1. Most useful for direction vectors on which magnitude plays no role.

Normalizing a Vector
This is easily achieved by dividing each of its components by its magnitude: $Norm(v) = (v_1 / ||v||, \dots , v_n / ||n||)$

Vectors - Dot Product and Orthogonalization

We add the product of each component of two vectors. This operation results in a scalar value. This is obtained by:

$$u \cdot v = (u1 * v1) + \ldots + (un * vn) = \mathrm{DOTP}(v,u)$$



$$\mathrm{DOTP}(u,v) \begin{cases} < 0 \text{ (obtuse)} \\ = 0 \text{ orthogonal (perpendicular)} \\ > 0 \text{ (acute)} \end{cases}$$

Properties

if u and v are normalized:

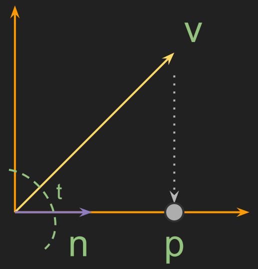
$$\mathrm{DOTP}(u,v) = \cos t$$

Hence

$$\cos^{-1}(\mathrm{DOTP}(u,v)) = t$$

Vectors - Dot Product and Orthogonalization

Projection - p is the orthogonal projection of v onto n.



Given two vectors, v and n. Vector p is the projection of v onto n.

Observation 1

Remember that DOTP(v,u) = ||v|| ||n|| cos(t) - Given by the law of cosines
Let ||n|| = 1, then p = kn where k is a positive scalar, so ||p|| = ||kn|| = |k| ||n||

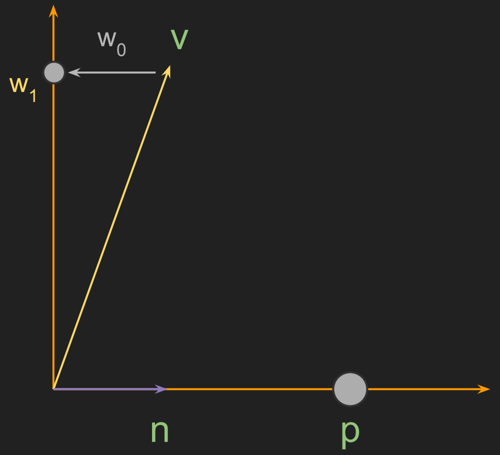
Observation 2

Note how
p = (||v|| cos(t)) n → if p = kn, then
kn = (||v|| cos(t)) n (remember that n is a unit vector)
it follows that kn = (||v|| 1 cos(t)) n = (||v|| ||n|| cos(t)) n
so → (DOTP(v,n)) n → (DOTP(v, n / ||n||)) / ||n||

Generally speaking, PROJ(v on n) = (DOTP(v,n) / ||n||^2) n → DOTP(v,n)*n if n is normalized

We can say that if v and n are both normalized, then cos(t) = DOTP(v,n)

Vectors - Dot Product and Orthogonalization



An orthonormal set is a collection of vectors on which each vector is orthogonal (perpendicular) to each other.

Among other reasons, given to floating point precision errors, sometimes, these sets needs to be corrected.

Assume we want to correct v's position such that it becomes orthogonal to n. Note n is a unit vector (of length 1). We would simply need to move v parallel to the y-axis.

Note that

if w_0 = PROJ(v on n) , then w_1 = v - PROJ(v on n)

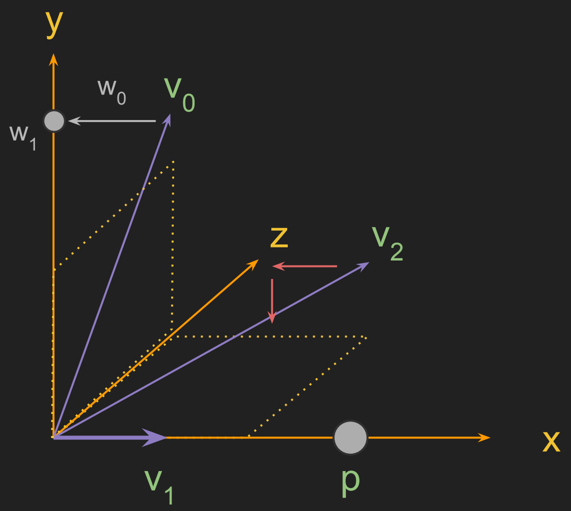
Now if v = w_1 , then v is orthogonal to n

The vector orthogonal to n, can also be expressed as:

w_1 = PERP(v to n) = v - PROJ(v on n)

Fernando Geraci

Vectors - Dot Product and Orthogonalization



For 3 vectors, for instance, we follow the same procedure with an addition:

1. V_0 has to be orthogonal to V_1
2. Now V_2 need to be:
   a. made orthogonal to V_0
   b. made orthogonal to V_1

This procedure is generalized by the Gram-Schmidt Orthogonalization algorithm.

1. if $w_1 = v_0 - \mathrm{PROJ}(v_0 \text{ on } v_1)$ - Make v_0 orthogonal to v_1
2. then $v_2 - \mathrm{PROJ}(v_2 \text{ on } v_1) - \mathrm{PROJ}(v_2 \text{ on } w_1)$ - Make v_2 orthogonal to v_0 and v_1

Vectors - Dot Product and Orthogonalization

Gram-Schmidt Orthogonalization Process

1. Let $v_0 = w_0$ (unit vector)
2. For every vector $1 <= i <= n-1$ ($n-1$ since we used v_0 already)

$$W_i = v_i - SUM_{j = 0 \rightarrow i - 1} (proj_{wj}(v_i))$$

So to orthogonalize a set of three vectors:

$$v_0 = w_0 \rightarrow \text{Then } w_1 = v_1 - proj_{v1}(w_0) \rightarrow \text{Then } w_2 = v_2 - proj_{v2}(w_1) - proj_{v2}(w_0)$$

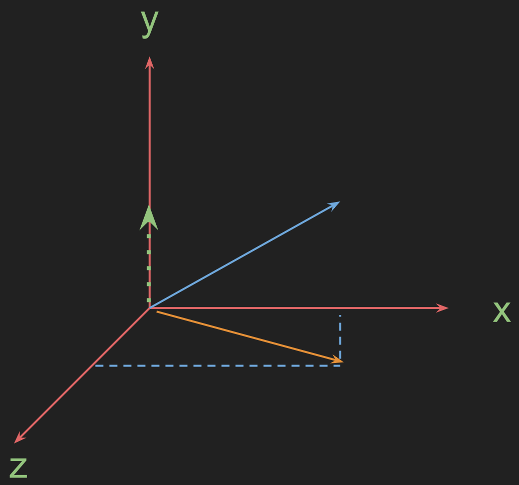
READ $proj_u(v) \rightarrow$ Vector v projected on vector u

Vectors - Dot Product and Orthogonalization

Plane Projection

Given a 3D plane, we can exclude a component of a vector’s orientation by:

$$\text{proj}_v(\text{plane}_{xz}) = v - \text{projec}_v(n_y)$$



This is really useful when we need to calculate an angle for orientation or any other use which doesn’t need all components of a vector.

Vectors - Cross Product and Orthogonalization

Cross product is only defined for 3D vectors, and it results in yet another vector, w, which is mutually orthogonal to both the original operands.

The operations are the following:

$$w = u \times v = (u_y v_z - u_z v_y \,,\, u_z v_x - u_x v_z \,,\, u_x v_y - u_y v_x)$$

Assume u = (1,3,-1) and v = (2,1,0) then u x v = (3*0 - (-1)*1 , (-1)*2 - 1*0 , 1*1 - 3*2) = (1 , -2 , -5) = w

now DOTP(u , w) = DOTP(v , w) = 0 , hence w is indeed orthogonal to both u and v.

Furthermore, we could use a modified version of dot product and cross product in 2D vectors to find an orthogonal vector w from a given 2D vector u. Let u’ = $(- u_x , u_y)$

$$\text{DOTP}(u,u') = \text{DOTP}\left((u_x u_y) , (- u'_y u'_x) \right) = u_x(-u'_y) + u_y u'_x = 0$$

e.g. given u (1 , -3) = 1*- (-3) + (-3)*1 = 0 - observe that DOTP(u , -u) also holds true.

Vectors - Cross Product and Orthogonalization

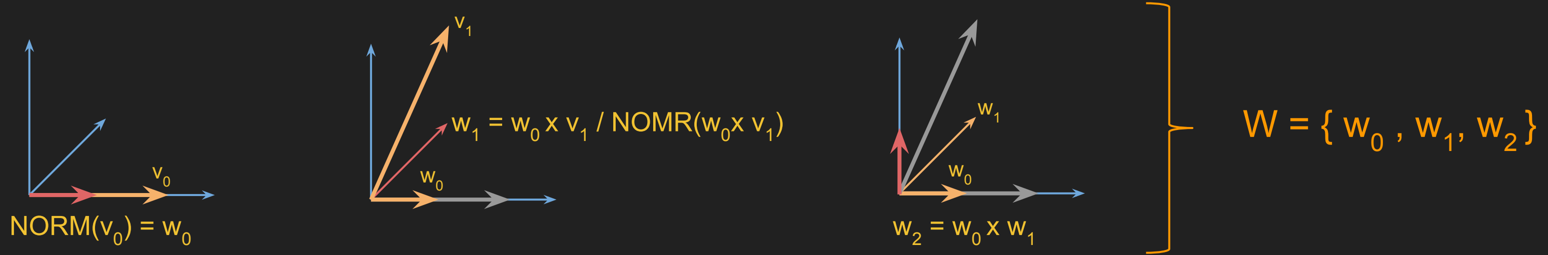
Given a set of vectors V $\{v_0 , v_1 , v_2\}$, we can correct their position to obtain an orthonormal set W $\{w_0 , w_1 , w_2\}$. We do this by using dot product twice.

Recall that the product of cross product is a third vector orthogonal to both operands.

For this, we first need to select a vector in V, say v_0 . Let NORM(v_0) = w_0

Then we let $w_1 = w_0 \times v_1$ / NORM($w_0 \times v_1$) . This yields to an orthogonal vector to w_0 and v_1

And finally it follows that $w_2 = w_0 \times w_1$


v_0
NORM(v_0) = w_0
v_1
$w_1 = w_0 \times v_1$ / NOMR($w_0 \times v_1$)
w_0
w_1
w_0
$w_2 = w_0 \times w_1$
W = { w_0 , w_1, w_2 }

Floating Point Operations

When doing calculation, we are always risking to suffer from floating point precision loses.

Two ways to handle this:

1) Define Delta-Tolerance value of, for instance, 0.0001

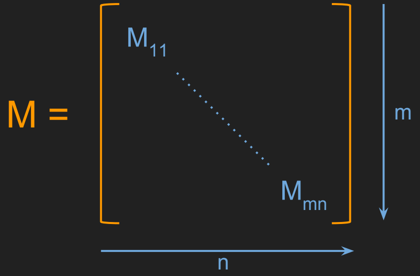
   |floatA - floatB| <= Delta-Tolerance → OK!

2) Use API built in functions for comparing vectors.

Matrix - Basics

A matrix is a two dimensional array of components, with **m** rows and **n** columns.

We use matrices to represent and express transform operations as linear combinations on vectors.



Given a matrix M, each component of the matrix is expressed as $M_{\text{row col}}$

Single column / row matrices are known as COLUMN / ROW vectors

Notation-wise, we can refer to an entire row / column of a matrix by:

$M_{k,*}$ - row k
$M_{*,k}$ - col k

Matrix - Basics

Some basic operations include addition, subtraction, multiplication and scaling.

> Of course, only equal size matrices (same **m** and **n**) can be added and subtracted, since those operations are carried components-wise.

Two matrices, A and B, can only be multiplied if and only if A’s columns equals to B’s rows. Multiplication is, not generally, commutative but yes associative - A(BC) = (AB)C

Assume A is a m x n matrix while B is a n x p one,then A x B = C of size m x p.

$$A \times B = C \text{ where } C_{ij} = DOTP(A_{i,*}, B_{*,j})$$

e.g.

$$A = \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \quad x \quad B = \begin{pmatrix} B_{1,1} \\ A_{2,1} \end{pmatrix} \quad = \quad C \rightarrow \begin{pmatrix} A_{1,1} * B_{1,1} + A_{1,2} * B_{2,1} \\ A_{2,1} * B_{1,1} + A_{2,2} * B_{2,1} \end{pmatrix}$$

Matrix - Basics

Linear Combinations

Used for vector-matrix multiplications. Given a vector v and a matrix A,

The product of

$$vA = (v_1 * A_{1,1} + \dots + v_n * A_{1,m} , \dots , v_1 * A_{n,1} + \dots + v_n * A_{n,m})$$

Transposing

Simply inverting the row,col assignment of each component.

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \longrightarrow A^T = \begin{pmatrix} a & c \\ b & d \end{pmatrix}$$

Properties:

$$\begin{aligned} (A+B)^T &= A^T + B^T \\ (cA)^T &= cA^T \\ (AB)^T &= B^T A^T \\ (A^T)^T &= A \\ (A^{-1})^T &= (A^T)^{-1} \end{aligned}$$

Matrix - Basics

Identity Matrix

Square matrix, with all 0s as components, except of its diagonal, which are all 1s.

i.e

$$I_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Given a matrix $A_{m,n}$ then:

$AI = IA = A$ (commutative)

$$v = \begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix}$$

$$\begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix}$$

Fernando Geraci

Matrix - Determinant

Definition

The determinant of a matrix is a single scalar value which is used, among other things, to find and compute the inverse of a given square n x n matrix A.

This value is given by $\det A = \mathrm{SUM}^{n}_{j=1}(A_{1j} * (-1)^{1+j} * \det(\min(A_{1j})))$

In computer graphics, we care specifically about computing determinant of 4x4 matrices - NOTE: this is recursive

Matrix Minor

$\min(A_{ij})$ is the submatrix A_{sub} of size (n-1) x (n-1) obtained from removing the i^th row and the j^th column of A.

$$A = \begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix} \longrightarrow \min(A_{11}) = \begin{pmatrix} A_{22} \\ A_{23} \\ A_{32} \\ A_{33} \end{pmatrix}$$

Once we reached this point, we can compute the scalar value by:

$$(A_{22}*A_{33}) - (A_{23}*A_{32})$$

Matrix - Determinant

Using the equation, we can find the determinant of the following matrix:

$$A = \begin{pmatrix} 1 & 2 & 1 \\ 3 & 0 & 4 \\ 8 & 4 & 10 \end{pmatrix} \longrightarrow \det A = 1 * \det \begin{pmatrix} 0 & 4 \\ 4 & 10 \end{pmatrix} - 2 * \det \begin{pmatrix} 3 & 4 \\ 8 & 10 \end{pmatrix} + 1 * \det \begin{pmatrix} 3 & 0 \\ 8 & 4 \end{pmatrix}$$

$$= 1 * (0*10 - 4*4) - 2 * (3*10 - 4*8) +1 * (3*4 - 0*8) = 0$$

For a 4x4 matrix, we will repeat the steps from above for each 3x3 matrix produced by the algorithm.

Matrix - Cofactor and Adjoint Matrix

Cofactor

It is the square matrix, with components

$$COF(A) = \begin{pmatrix} c_{ij} & c_{ij} & c_{ij} \\ c_{ij} & c_{ij} & c_{ij} \\ c_{ij} & c_{ij} & c_{ij} \end{pmatrix}$$

$$C_{ij} = (-1)^{i+j} * \det(\min(A_{ij}))$$

Adjoint

$$COF(A)^T = \begin{pmatrix} c_{ji} & c_{ji} & c_{ji} \\ c_{ji} & c_{ji} & c_{ji} \\ c_{ji} & c_{ji} & c_{ji} \end{pmatrix}$$

It is the transposed cofactor matrix of A. Denoted $A^* = COF(A)^T$

The adjoint matrix is very useful for computing the inverse of matrix A.

Matrix - Inverse - Properties

Definition

An inverse matrix is the ‘reciprocal’ matrix denoted as A^{-1} such that:

- Given a matrix A, $A*A^{-1} = I$ - identity matrix - (it’s reciprocal)
- Since only square matrices have inverses, A^{-1} is square
- If a matrix A is invertible, then det(A) != 0
- A^{-1} can only result from inverting A, and not all square matrices are invertible (known as ‘singular’ matrices)
- A^{-1}, if exists, is unique
- If A and B are both invertible square matrices, then: $(AB)^{-1} = B^{-1}A^{-1}$
- For a small matrix A (say 4X4), their inverse can be found:

$$A^{-1} = A^* / \det(A)$$

Exercises - Set Up

Setting DirectXMath up in Visual Studio 2015 Community (Includes Direct3D SDK)

- Clone the repository: <url_tbd_per_semester>
- Each instruction is 128-bit long, to represent a 4D vector
  - SIMD: Single Instruction Multiple Data set. Operates on 4 32-bit floats.
- Reference: DirectXMath Programming Reference
- Main headers:
  - DirectXMath.h namespace: DirectX
  - DirectXPackedVector.h namespace: DirectX::PackedVector
- DirectX Project Properties:
  - C++ → Code Generation
    - Enhanced Instruction Set : SSE2 - allow x86 platforms to understand SIMD sets
    - Floating Point Model : Fast

API

Vectors - XMVECTOR

Operators: + - * /
XMVectorSet
XMVector3Length / Sq
XMVector3Dot
XMVector3Cross
XMVector3Normalize
XMVector3Orthogonal
XMVector3AngleBetweenVectors

Matrices - XMMATRIX

Operators: + - * /
XMMatrixSet
XMMatrixIdentity
XMMatrixIsIdentity
XMMatrixMultiply
XMMatrixTranspose
XMMatrixDeterminant
XMMatrixInverse

XMLoadFloat<type> returns SIMD, XMStoreFloat<type> returns No-SIMD

Fernando Geraci

Exercises - Vectors and Matrices

1. Set up a Win32 Console application:
   - Add the required DirectX libraries
   - Verify if the CPU has the required components to use SIMD
   - Overload the FXMVECTOR class to handle standard output stream
   - Overload the FXMMATRIX class to handle os output
2. Provide examples for the following functions:
   - XMVector3Length
   - XMVector3AngleBetweenVectors
   - XMVector3Equal
   - XMVector3NotEqual
3. Using the above referenced functions, implement (and all those needed for them):
   - GetDotProduct
   - GetCrossProduct
   - NormalizeVector
   - GetOrthogonal2DVector / GetOrthogonal3DVector
   - ProjectVector
   - Orthogonalize_GramSchmidt
   - GetMatrixMinor
   - GetMatrixTranspose
   - GetDeterminant
   - GetMatrixAdjoint
   - GetMatrixInverse

Compute the total complexity of these functions