AMD EPYC

AMD RYZEN

AMD RADEON

AMD GPUOpen

THE MATRIX COMPENDIUM, FOUR FACES OF TRANSFORMATION

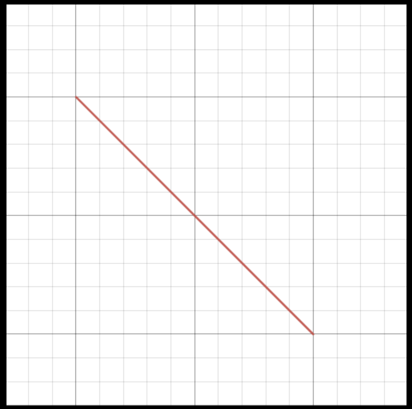
DR, ŁUKASZ IZDEBSKI
DEVELOPER TECHNOLOGY ENGINEER
AMD

together we advance_

INTRODUCTION

- I will try to explain WHY this presentation is needed.
- WHY do 3D transformations have four faces?

Expectation

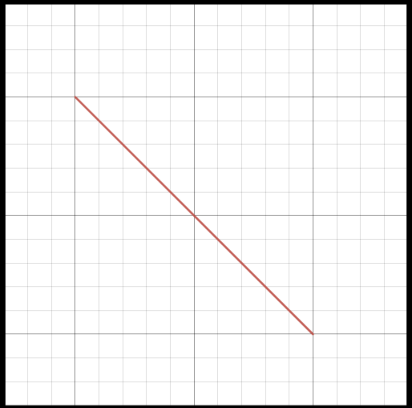


$$\int \frac{1}{x^4} \mathrm{d}x = -\frac{1}{3x^3} + C$$

INTRODUCTION

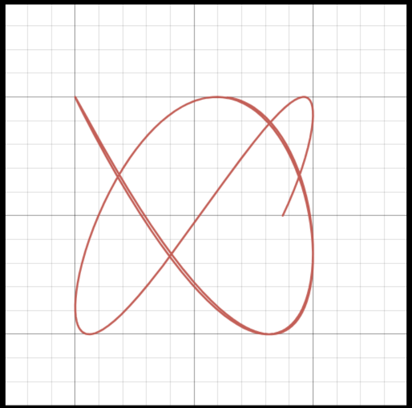
- I will try to explain WHY this presentation is needed.
- WHY do 3D transformations have four faces?

Expectation



$$\int \frac{1}{x^4} \mathrm{d}x = -\frac{1}{3x^3} + C$$

Reality



$$\int \frac{1}{x^4+1} \mathrm{d}x = \frac{\sqrt{2}}{8}\left(\ln\left(x\cdot\left(x+\sqrt{2}\right)+1\right) - \ln\left(x\cdot\left(x-\sqrt{2}\right)+1\right)\right) + \frac{\sqrt{2}}{4}\left(\arctan\left(\sqrt{2}\,x+1\right)+\arctan\left(\sqrt{2}\,x-1\right)\right)$$

THE ROOT OF ALL PROBLEMS

1. Matrix multiplication, which in general is not commutative.

$$A*B \neq B*A$$

2. Storage of 2D data in 1D memory.

3. In a three-dimensional Cartesian coordinate system, the result of the cross-product of two non-collinear vectors is anticommutative.

$$\vec{a} \times \vec{b} = -\vec{b} \times \vec{a}$$

4. The order of the basis vectors.

$$\{e_0, e_1, e_2, \dots, e_n\} \neq (e_0, e_1, e_2, \dots, e_n)$$

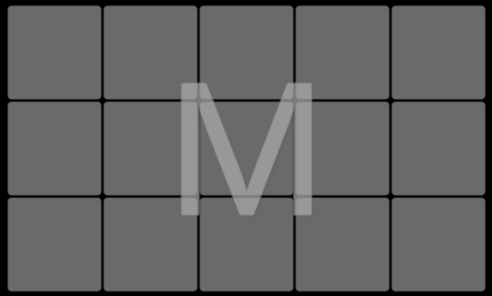
WHAT IS MATRIX?

- A brief reminder. What is a matrix?

In mathematics, a matrix is defined as a rectangular array of numbers arranged in rows and columns.

- For example, the matrix M below has 3 rows and 5 columns and can be referred to as a 3×5 matrix.

$$M = \begin{bmatrix} a & b & c & d & e \\ f & g & h & i & j \\ k & l & m & n & o \end{bmatrix}$$

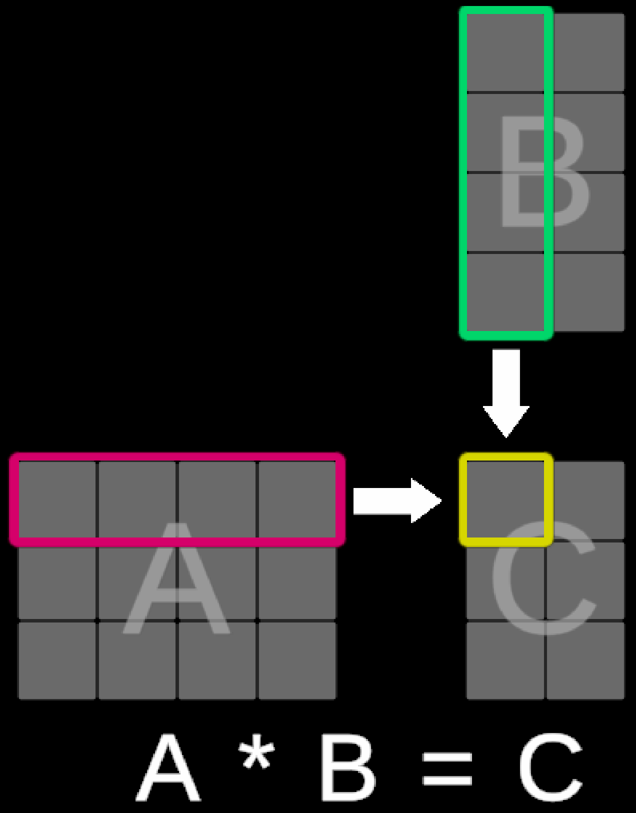


MATRICES IN COMPUTER GRAPHICS

- In computer graphics, matrices are used for the following purposes:
  - As a tool for an algebraic approach to Euclidean, affine, or projective geometry.
  - To store 3D transformations like affine or projective (matrices up to 4x4).
  - Transformation concatenation is represented by matrix multiplication.
  - To store mesh vertices (matrices 4x1 column vector or 1x4 row vector).
  - Transforming a vertex from one space to another involves multiplying it by an appropriate matrix.
  - They can be easily used (implemented) in computer graphics applications.

MATRIX MULTIPLICATION 101

- In general, matrix multiplication is not commutative. There are examples of pairs of matrices whose multiplication is commutative, they are only an exception to the rule.



MATRIX MULTIPLICATION ORDER

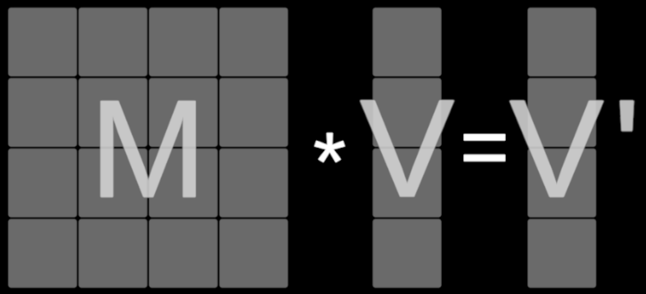
- In linear algebra or physics topics, Post-multiplication order is mostly used.
- In game engines or modeling software, both Pre- and Post-multiplication orders can be found.

Pre-multiplication “DirectX® style”
Vertex V is treated as a 1x4 matrix
(row vector)

$$V * M = V'$$

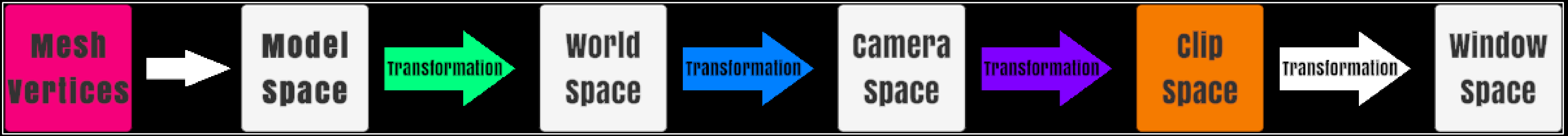
Post-multiplication “OpenGL® style”
Vertex V is treated as a 4x1 matrix
(column vector)

$$M * V = V'$$



TRANSFORMATION CONCATENATION 1

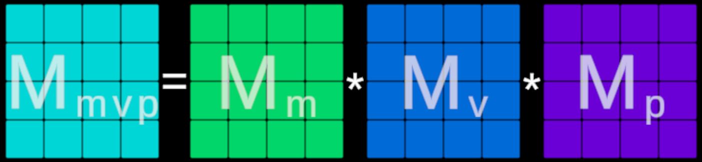
- Graphics Pipeline Transformation Queue



Pre-multiplication “DirectX® style”

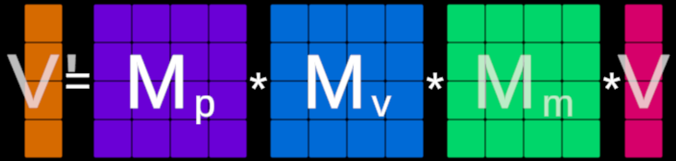
$$V' = V * M_m * M_v * M_p$$

$$M_{mvp} = M_m * M_v * M_p$$

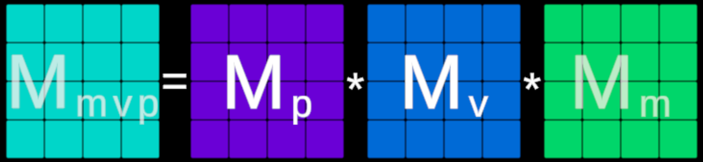


Post-multiplication “OpenGL® style”

$$V' = M_p * M_v * M_m * V$$

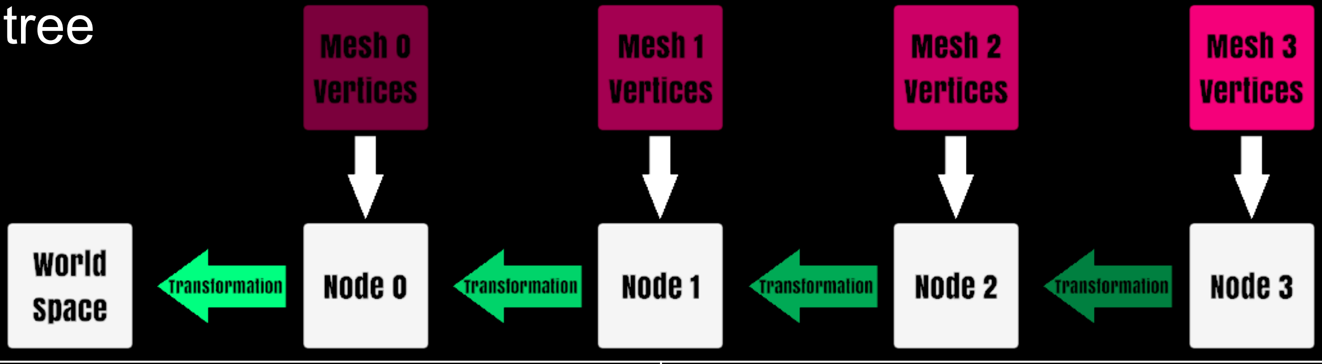


$$M_{mvp} = M_p * M_v * M_m$$



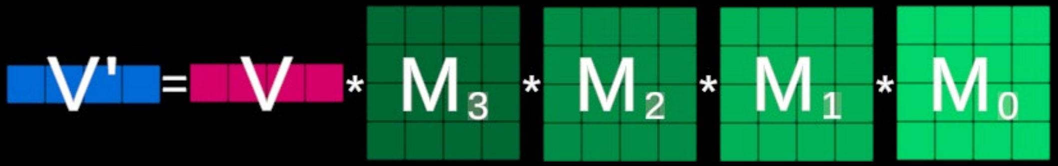
TRANSFORMATION CONCATENATION 2

- Hierarchy of scene tree transformations



Pre-multiplication “DirectX® style”

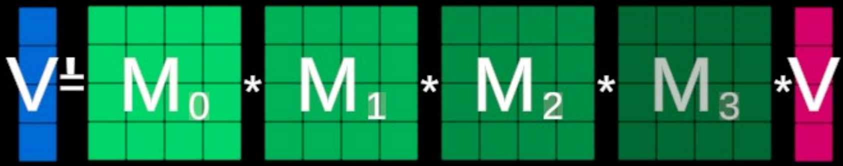
$$V' = V * M_3 * M_2 * M_1 * M_0$$



$$M_m = M_3 * M_2 * M_1 * M_0$$

Post-multiplication “OpenGL® style”

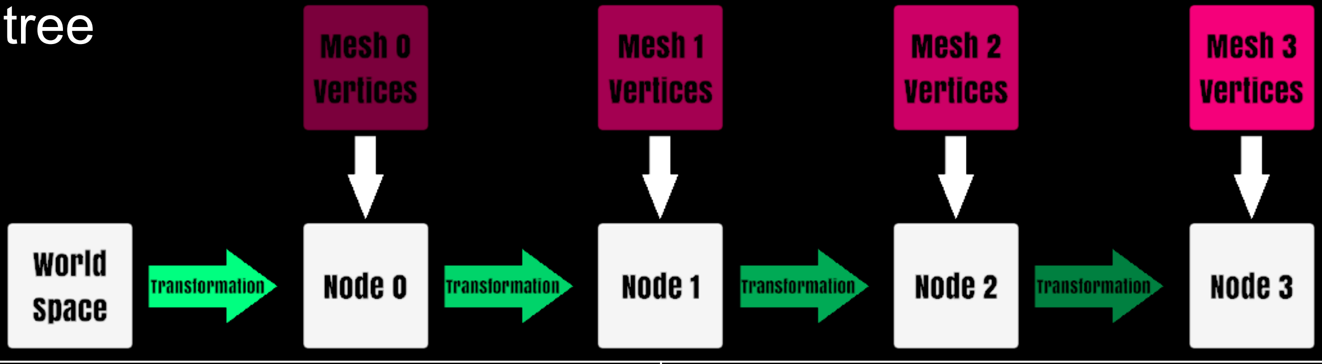
$$V' = M_0 * M_1 * M_2 * M_3 * V$$



$$M_m = M_0 * M_1 * M_2 * M_3$$

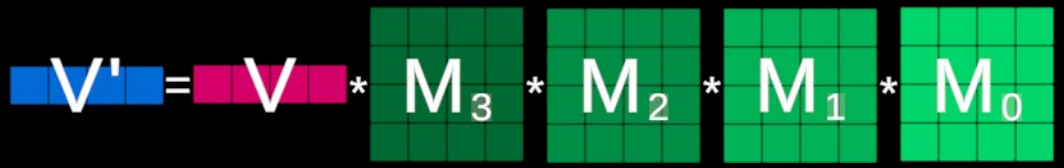
INVERSE TRANSFORMATION CONCATENATION

- Hierarchy of scene tree transformations

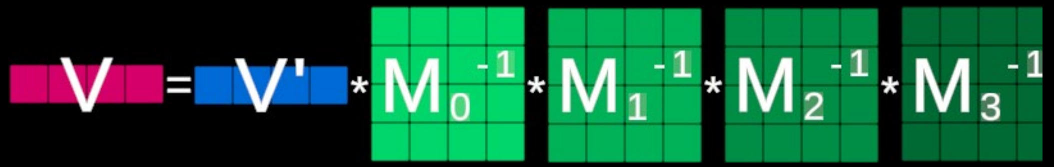


Pre-multiplication “DirectX® style”

$$V' = V * M_3 * M_2 * M_1 * M_0$$

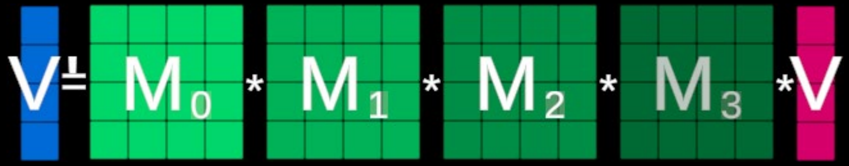


$$V = V' * M_0^{-1} * M_1^{-1} * M_2^{-1} * M_3^{-1}$$

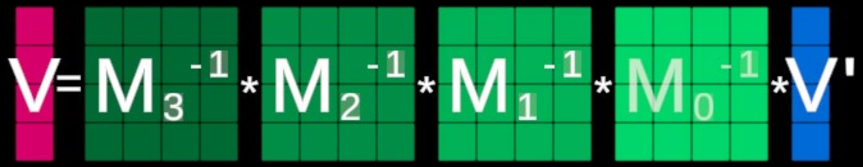


Post-multiplication “OpenGL® style”

$$V' = M_0 * M_1 * M_2 * M_3 * V$$



$$V = M_3^{-1} * M_2^{-1} * M_1^{-1} * M_0^{-1} * V'$$



AFFINE TRANSFORMATIONS

- In game engines or modeling software, we can find three (sometimes four) affine transformations:

Translation

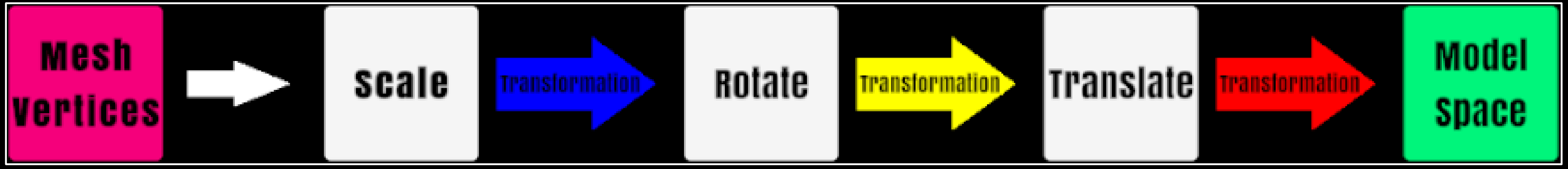
Rotate

Scale

Shear

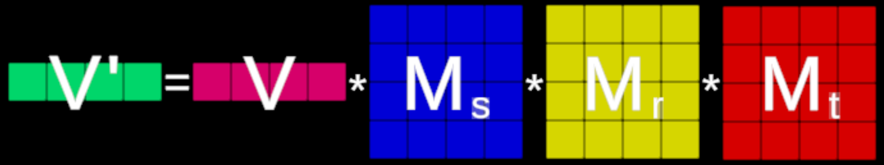
TRANSFORMATION CONCATENATION 3

- Affine transformations are the building blocks of node transformations in the scene hierarchy. Rotate M_r and Scale M_s transform every point in space except the origin (0,0,0).

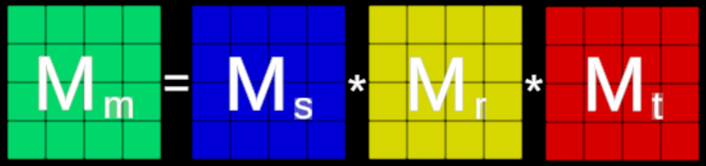


Pre-multiplication "DirectX® style"

$$V' = V * M_s * M_r * M_t$$

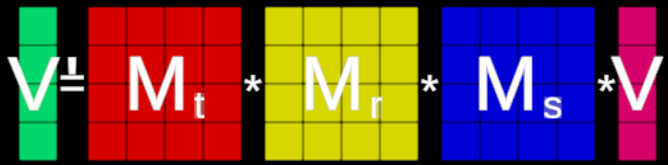


$$M_m = M_s * M_r * M_t$$

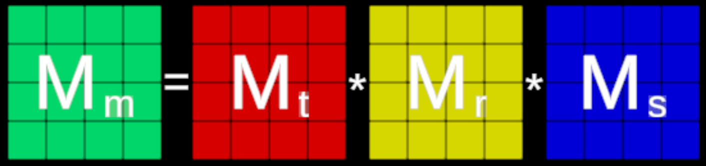


Post-multiplication "OpenGL® style"

$$V' = M_t * M_r * M_s * V$$

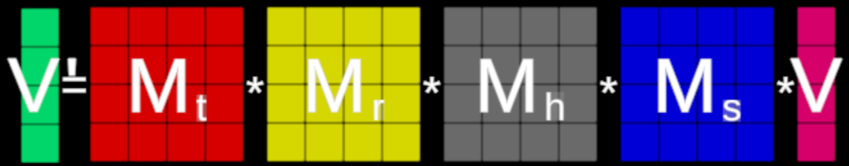
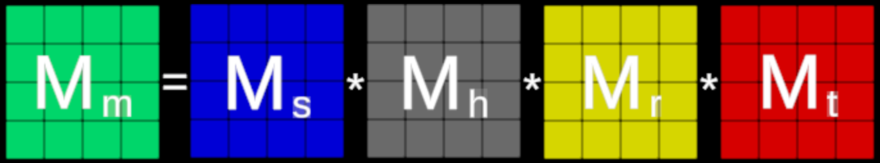


$$M_m = M_t * M_r * M_s$$



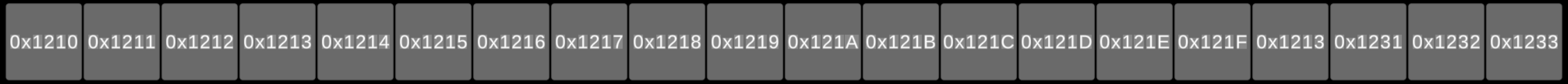
TO SHEAR OR NOT TO SHEAR? THAT IS THE QUESTION.

- Yes, we can do Shear M_h transformation, M_h transform every point in space except the origin. But where can we put Shear in the transformation queue?

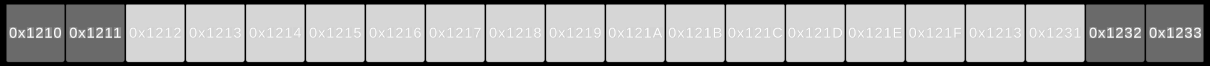
Pre-multiplication "DirectX® style"	Post-multiplication "OpenGL® style"
$V' = V * M_s * M_h * M_r * M_t$	$V' = M_t * M_r * M_h * M_s * V$
	
$M_m = M_s * M_h * M_r * M_t$	$M_m = M_t * M_r * M_h * M_s$
	

MATRIX STORAGE IN COMPUTER MEMORY

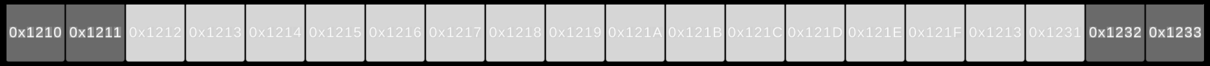
- To be able to use the matrices in applications, they must be stored in the computer's memory. There is a small problem because computer memory is linear (1D).



- A matrix is defined as a rectangular array of numbers arranged in rows and columns (2D).



  - Row Major



  - Column Major

MATRIX STORAGE IN COMPUTER MEMORY

- To be able to use the matrices in applications, they must be stored in the computer's memory. There is a small problem because computer memory is linear (1D).

0x1210	0x1211	0x1212	0x1213	0x1214	0x1215	0x1216	0x1217	0x1218	0x1219	0x121A	0x121B	0x121C	0x121D	0x121E	0x121F	0x1213	0x1231	0x1232	0x1233

- A matrix is defined as a rectangular array of numbers arranged in rows and columns (2D).

0x1210	0x1211	0x1212	0x1213	0x1214	0x1215	0x1216	0x1217	0x1218	0x1219	0x121A	0x121B	0x121C	0x121D	0x121E	0x121F	0x1213	0x1231	0x1232	0x1233

  - Row Major — C, C++, D, Java™, Rust, Python™

0x1210	0x1211	0x1212	0x1213	0x1214	0x1215	0x1216	0x1217	0x1218	0x1219	0x121A	0x121B	0x121C	0x121D	0x121E	0x121F	0x1213	0x1231	0x1232	0x1233

  - Column Major — Fortran, Matlab®

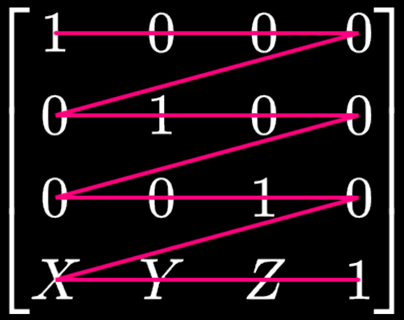
MATRIX MULTIPLICATION AND STORAGE IN ACTION

- How can a translation matrix be stored in computer memory? And what are the consequences of this?

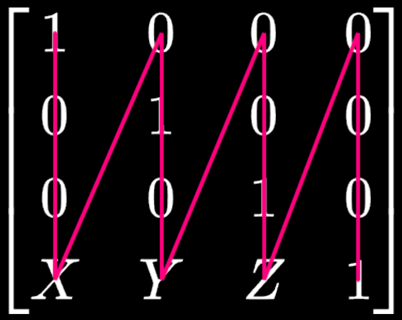
Pre-multiplication "DirectX® style"

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ X & Y & Z & 1 \end{bmatrix}$$

Row-Major



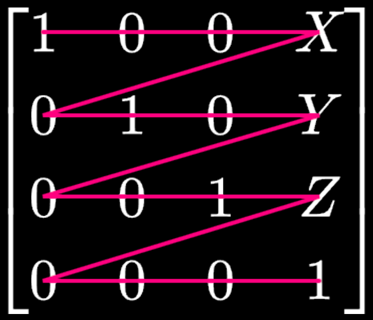
Column-Major



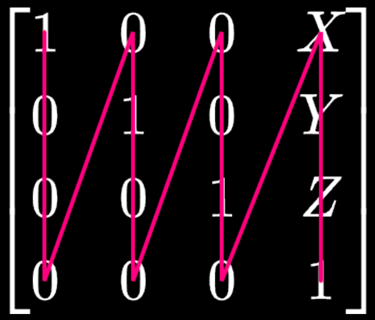
Post-multiplication "OpenGL® style"

$$\begin{bmatrix} 1 & 0 & 0 & X \\ 0 & 1 & 0 & Y \\ 0 & 0 & 1 & Z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Row-Major

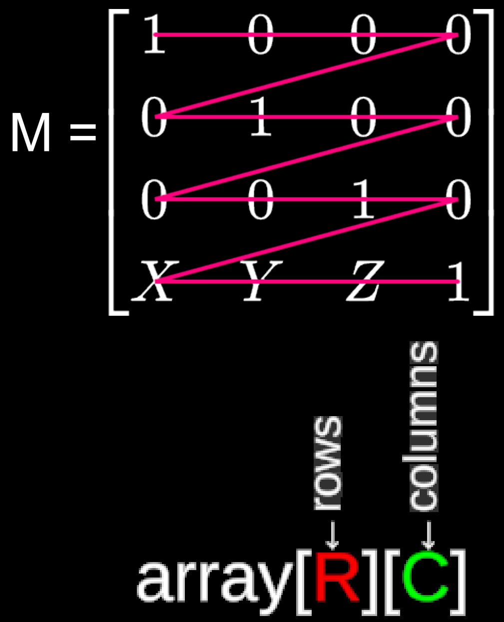
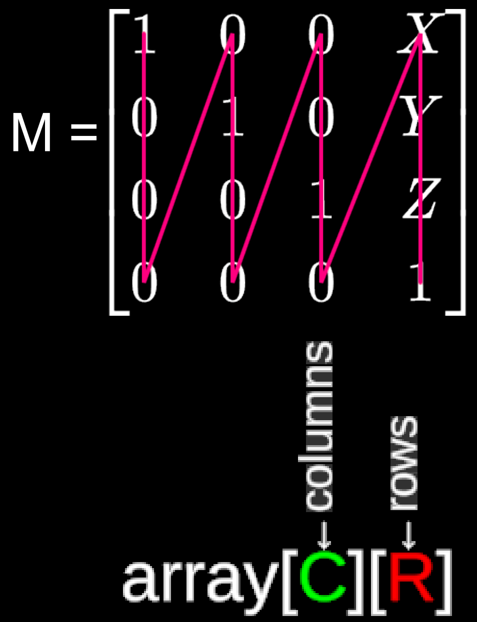


Column-Major



MATRIX STORAGE IN C++

- **R×C** matrix (R number of rows, C number of columns) is stored in C/C++ as a multidimensional array in Row Major order.

Pre-multiplication "DirectX® style"	Post-multiplication "OpenGL® style"
	
The order is the same as for the matrix definition.	The order is **NOT** the same as for matrix definition.

MATRIX MULTIPLICATION IN HLSL AND GLSL

- HLSL or GLSL shader languages are **independent** from the point of view of matrix multiplication or matrix storage.
- Hardware and APIs, such as DirectX® and Vulkan®, are also **independent**. With two small exceptions that I will talk about in detail later.

HLSL		GLSL	
Matrix multiplication:	`mul() function`	Matrix multiplication:	`*operator`
Dot product:	`dot() function`	Dot product:	`dot() function`
Component-wise multiplication:	`*operator`	*Component-wise multiplication:*	`*operator`
Matrix default storage order:	`row_major`	Matrix default storage order:	`column_major`
Storage modifiers:	`row_major`	Storage qualifiers:	`layout(row_major)`
	`column_major`		`layout(column_major)`

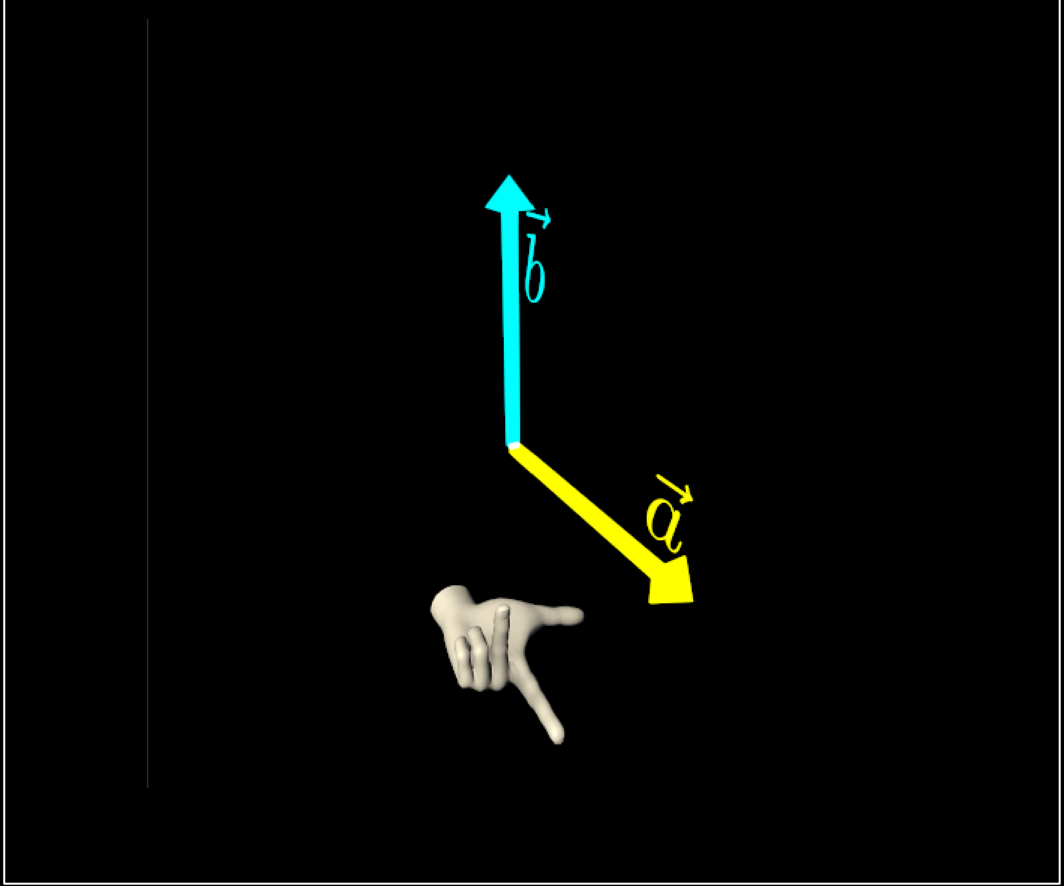
COORDINATE SYSTEM “HANDEDNESS”

- To determine a three-dimensional Cartesian coordinate system, we need to choose three vectors (axes) that are perpendicular to each other. The cross-product is a tool that will help determine the direction of the axes of the coordinate system.

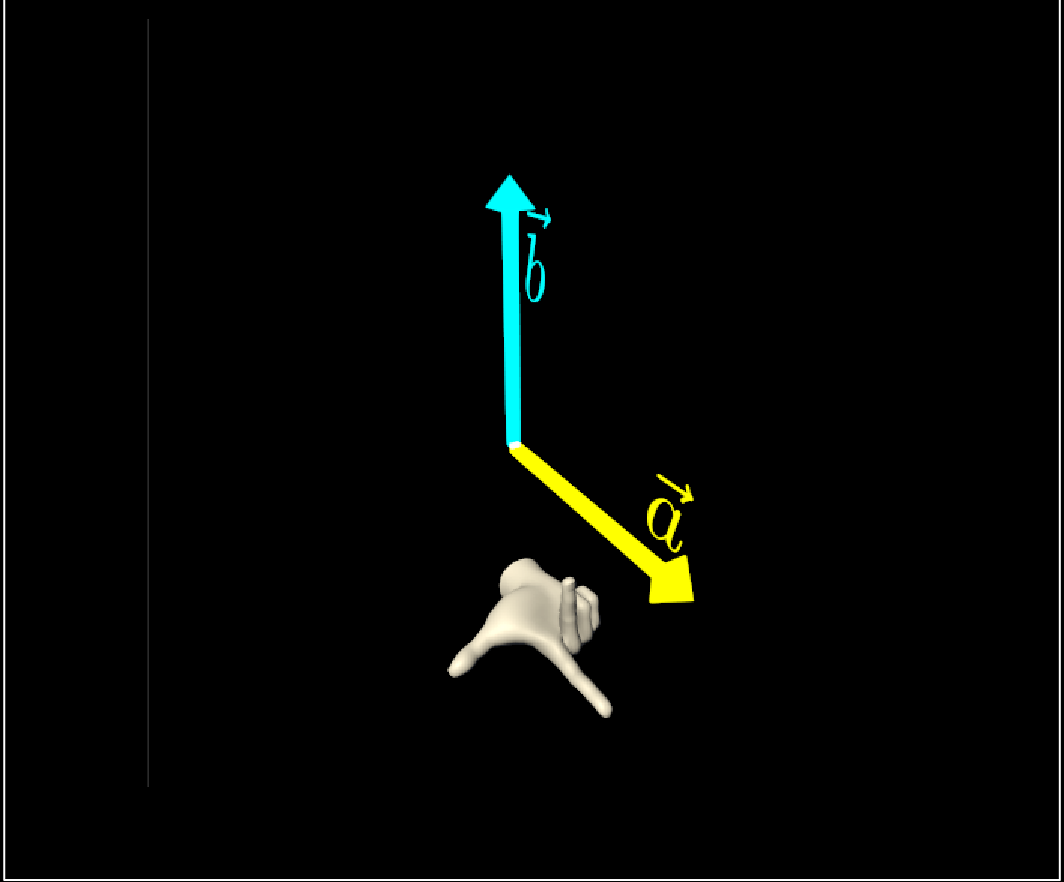
RIGHT OR LEFT-HAND RULE

- The rule makes it easy to quickly determine the unique consistent result of the cross-product vector direction.

Left-hand rule



Right-hand rule

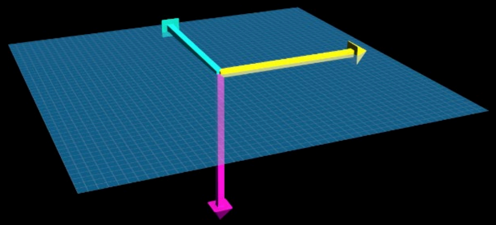


RIGHT OR LEFT-HAND RULE

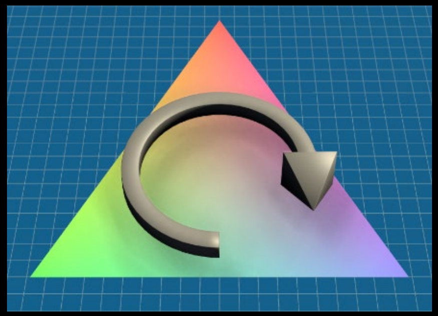
- Choosing the direction of the result of the cross product also determines the vector normal to the plane and the winding order of the triangle vertices (front face).

Left-hand rule

- normal vector

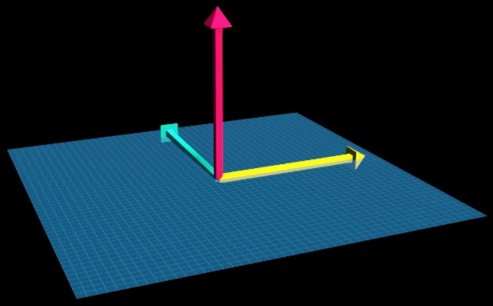


- winding order

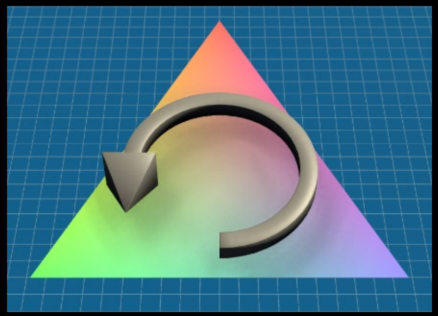


Right-hand rule

- normal vector



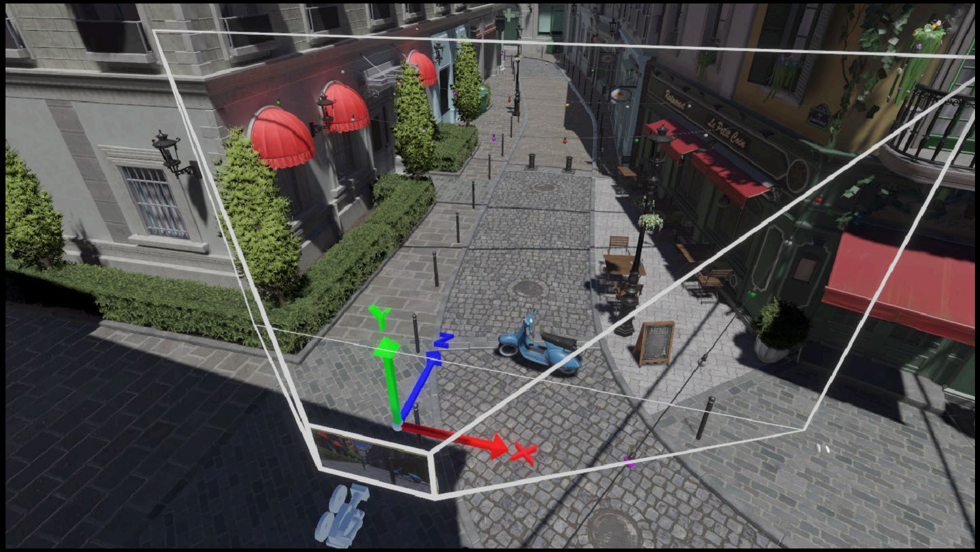
- winding order



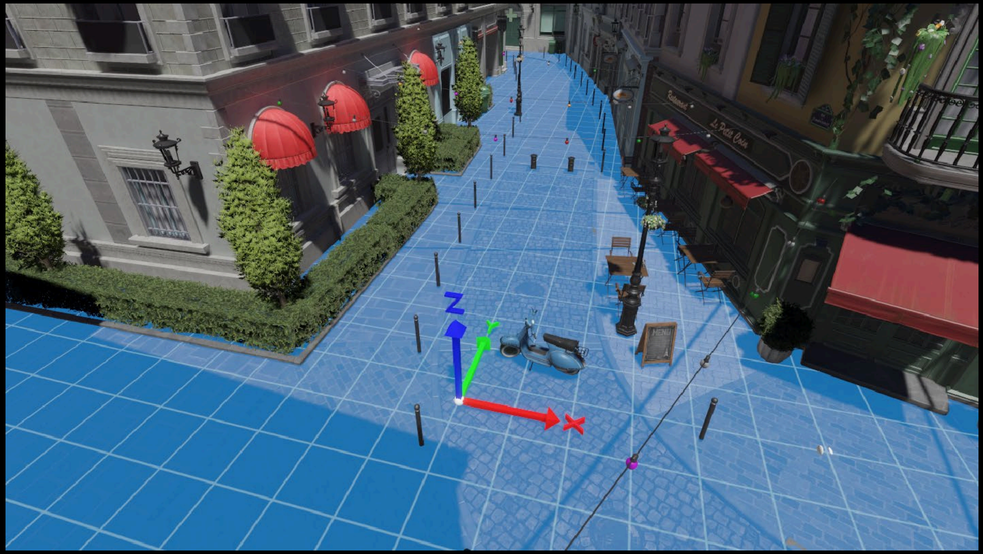
“UP DIRECTION”

- After defining the coordinate system, we need to interpret the directions of these axes and translate them into real-world directions.
  - The X-axis is the easiest, since “everyone” agreed that it should point to the right.
  - But which axis will point upward? There are two schools of thought.

Y-Axis is Up Direction

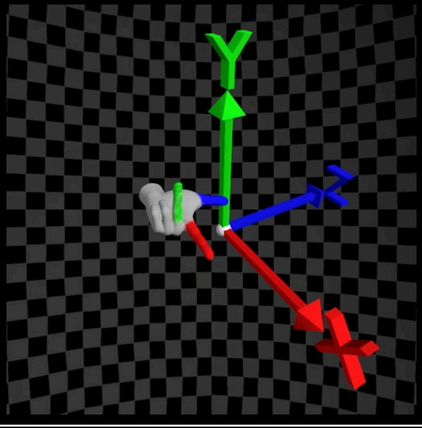
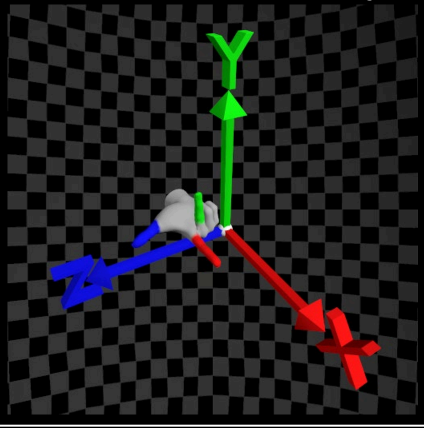
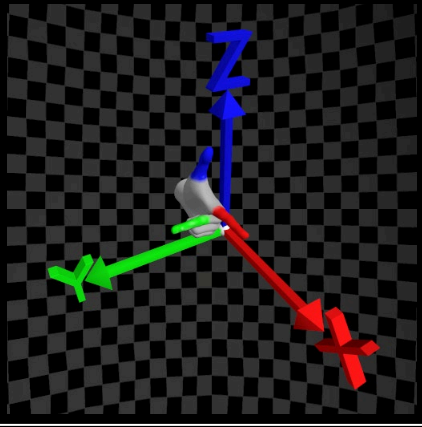
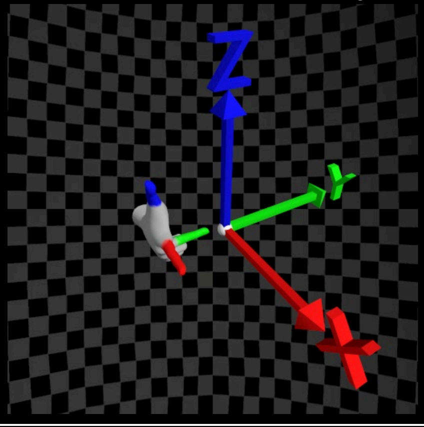


Z-Axis is Up Direction



COORDINATE SYSTEMS

- There are four coordinate systems that can be found in game engines or modeling software.

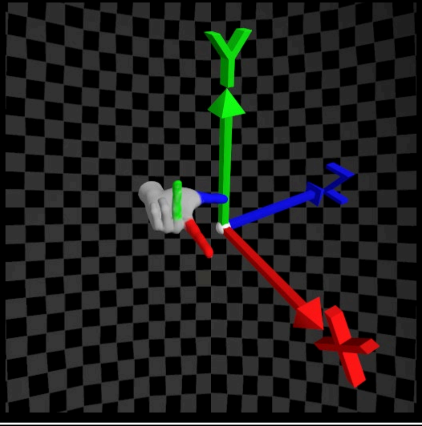
Left-handed Y-Up coordinate system	Right-handed Y-Up coordinate system
	
Left-handed Z-Up coordinate system	**Right-handed Z-Up coordinate system**
	

COORDINATE SYSTEMS

- There are four coordinate systems that can be found in game engines or modeling software.

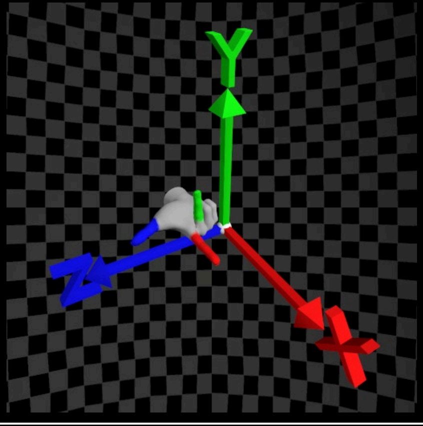
Left-handed Y-Up coordinate system

LightWave 3D®
Zbrush®
Cinema 4D
Unity®



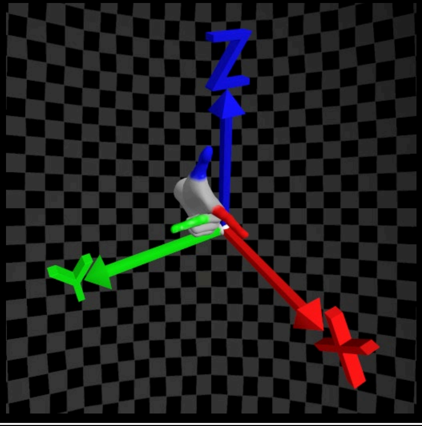
Right-handed Y-Up coordinate system

Autodesk Maya ®
Modo ®
Houdini 3D™
Substance Painter 3D®
Marmoset Toolbag
Godot



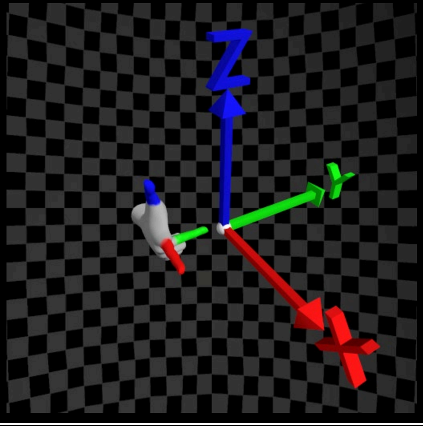
Left-handed Z-Up coordinate system

Unreal Engine*
Open 3D Engine



Right-handed Z-Up coordinate system

Autodesk 3ds Max
Blende ®
SketchUp
Autodesk AutoCAD®
CRYENGINE®
UNIGINE

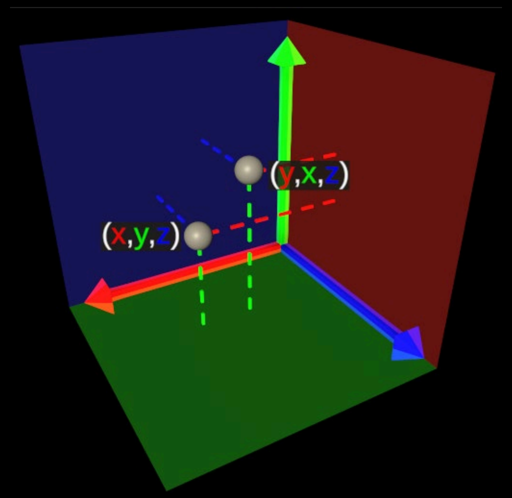


BASIS VECTORS ORDER

- The order of the basis vector of the coordinate system uniquely determines what we can call "Positive Direction." "Positive Direction" is the direction of rotation with increasing angle of rotation.

(X, Y, Z) basis vector order

Cartesian coordinate system

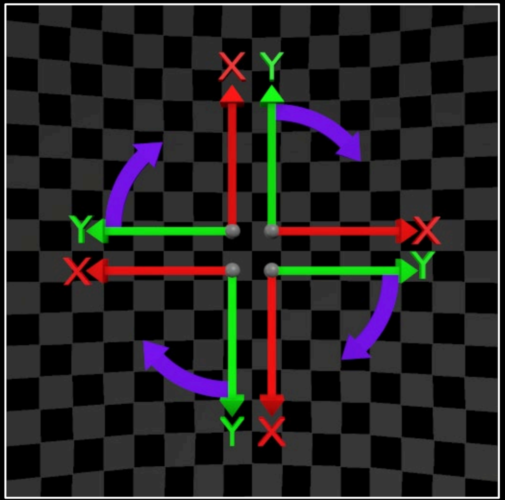


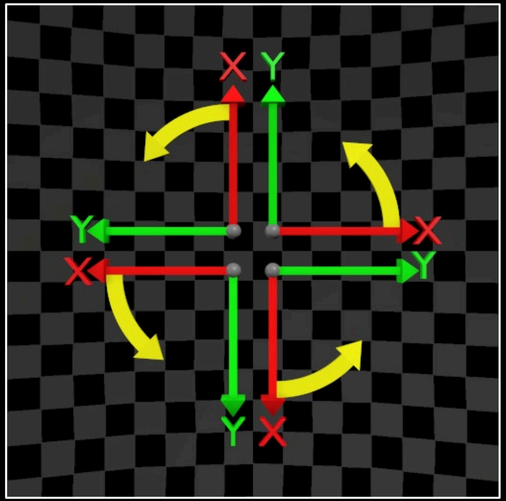
(X, Z, Y) basis vector order

ROTATION MATRIX VS POSITIVE DIRECTION

- Let's see what happens to “Positive Direction” when we use it in different multiplication orders.

$$R = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix}$$



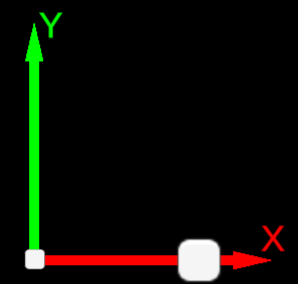


MATRIX MULTIPLICATION ORDER VS POSITIVE DIRECTION

- Let's see what happens to "Positive Direction" when we use it in different multiplication orders.

$$R = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix}$$

Post-multiplication "OpenGL® style"
Counterclockwise Direction
$V' = R * V$



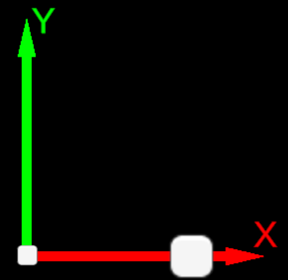
MATRIX MULTIPLICATION ORDER VS POSITIVE DIRECTION

- Let's see what happens to “Positive Direction” when we use it in different multiplication orders.

$$R = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix}$$

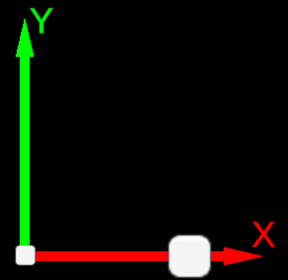
Pre-multiplication “DirectX® style”
Clockwise Direction

$$V' = V * R$$



Post-multiplication “OpenGL® style”
Counterclockwise Direction

$$V' = R * V$$

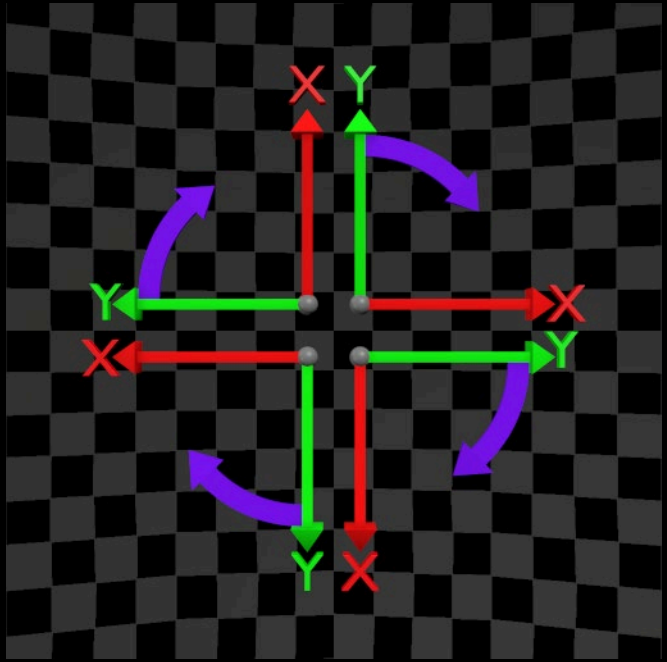


2D ROTATION VS BASIS VECTOR ORDER

- Depending on the order of matrix multiplication, we get two different “Positive Directions.”

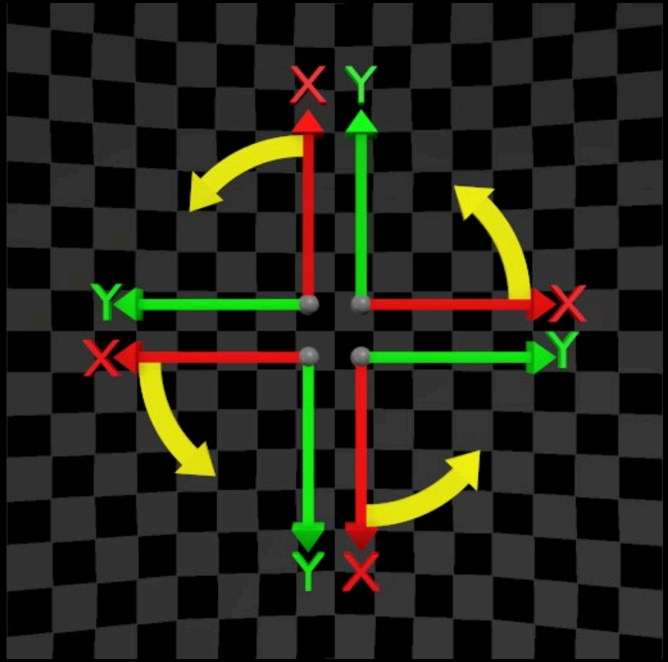
$$R = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix}$$

Clockwise Direction



(Y, X) basis vector order

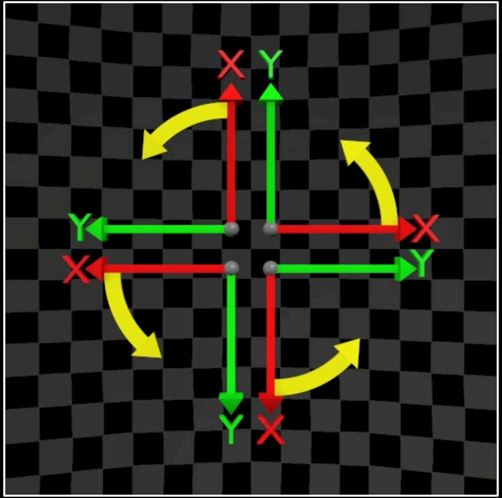
Counterclockwise Direction



(X, Y) basis vector order

MULTIPLICATION MATRIX ORDER VS POSITIVE DIRECTION

- To describe the same "Positive Direction," we must use different rotation matrices depending on the order of the matrix.



Pre-multiplication "DirectX® style"

(X, Y) basis vector order

$$R = \begin{bmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{bmatrix}$$

Post-multiplication "OpenGL® style"

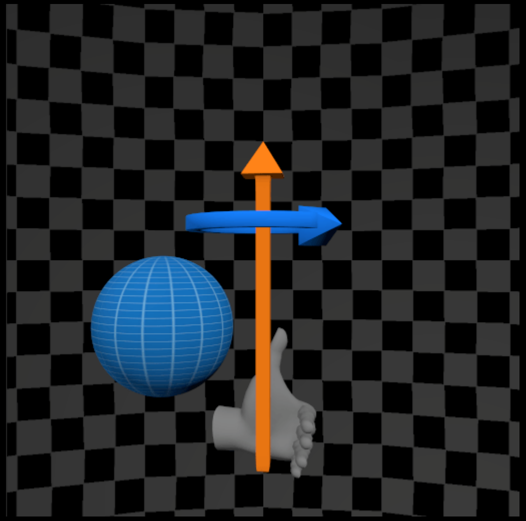
(X, Y) basis vector order

$$R = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix}$$

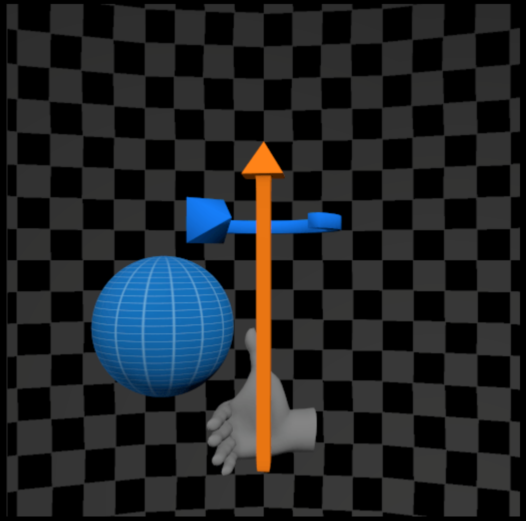
RIGHT OR LEFT-HAND RULE FOR POSITIVE DIRECTION

- As with the cross-product, the right- or left-rule can help us determine the “Positive Direction.”

Left-hand rule



Right-hand rule

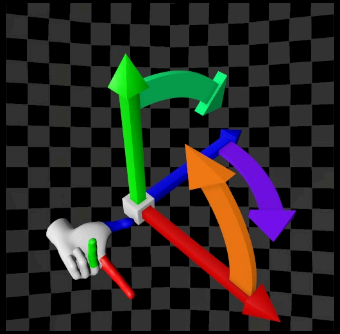


3D ROTATION VS BASIS VECTOR ORDER

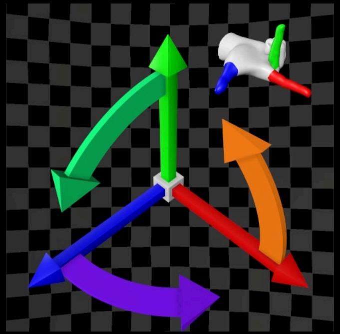
- There are more permutations in 3D space. Eventually, they can be grouped into two groups.

SAME hand for cross-product and "Positive Direction"

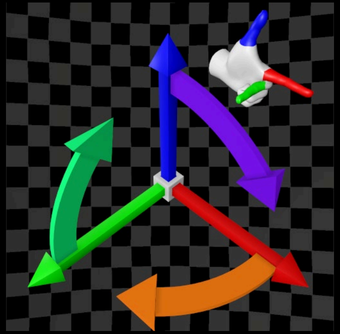
Left-hand | Right-hand



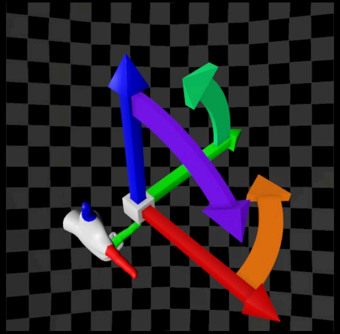
Y-Up



(X, Y, Z) | (X, Y, Z)



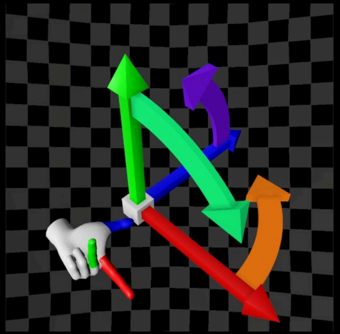
Z-Up



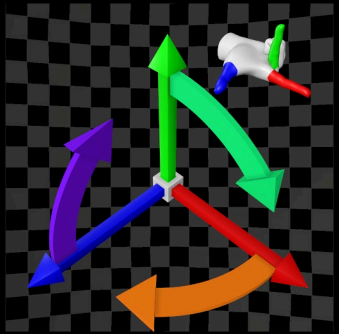
(X, Y, Z) | (X, Y, Z)

OPPOSITE hands for cross-product and "Positive Direction"

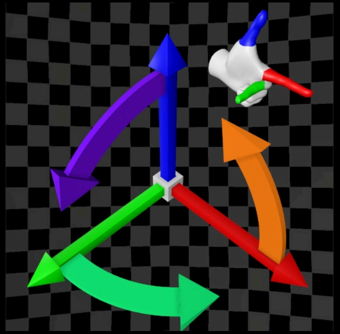
Left-hand | Right-hand



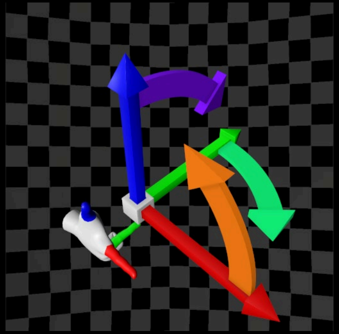
Y-Up



(X, Z, Y) | (X, Z, Y)



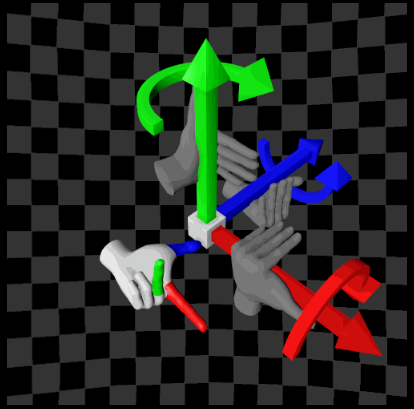
Z-Up



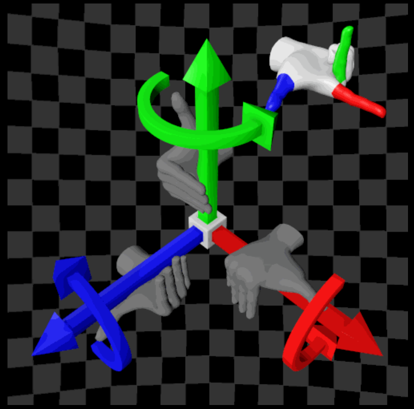
(X, Z, Y) | (X, Z, Y)

3D ROTATIONS AND MATRIX MULTIPLICATION ORDER

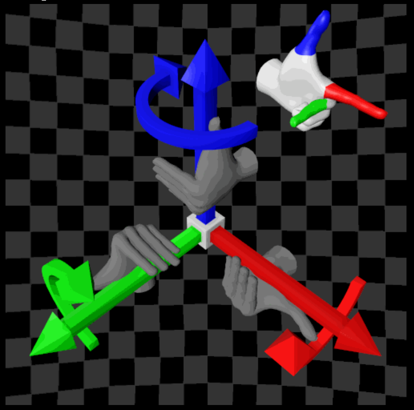
Left-handed Y-Up coordinate system
Pre-multiplication "DirectX® style"



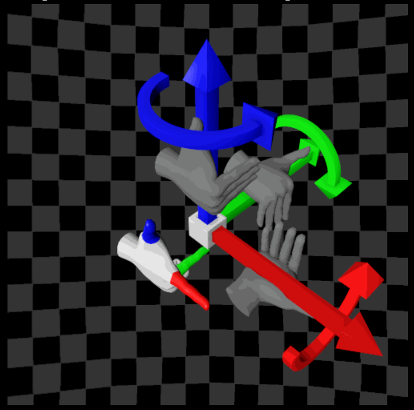
Right-handed Y-Up coordinate system
Post-multiplication "OpenGL® style"



Left-handed Z-Up coordinate system
Pre-multiplication "DirectX® style"



Right-handed Z-Up coordinate system
Post-multiplication "OpenGL® style"



UNREAL ENGINE 4/5 ISSUE???

- Fun fact. Don't be surprised if “something weird is going on” when using Unreal Engine `FRotator`.

QUATERINIONS

- In mathematics, Quaternion is defined q=$a+b\mathbf{i}+c\mathbf{j}+d\mathbf{k}$. In computer graphics unit quaternions $|q|$=1 can represent spatial orientation and rotations in 3D space.

Hamilton's convention “OpenGL® style”

$$\mathbf{ij} = \mathbf{k}$$
$$\mathbf{jk} = \mathbf{i}$$
$$\mathbf{ki} = \mathbf{j}$$
$$\mathbf{ijk} = -\mathbf{1}$$

- “Positive Direction” = right-hand rule.

$$R = \begin{bmatrix} 1-c^2-d^2 & 2bc-2ad & 2bd+2ac \\ 2bc+2ad & 1-b^2-d^2 & 2cd-2ab \\ 2bd-2ac & 2cd+2ab & 1-b^2-c^2 \end{bmatrix}$$

- “Sandwich” Product

$$V' = q * V * q^{-1}$$

QUATERINIONS

- In mathematics, Quaternion is as defined q=$a+b\boldsymbol{i}+c\boldsymbol{j}+d\boldsymbol{k}$. In computer graphics unit quaternions $|q|$=1 can represent spatial orientation and rotations in 3D space.

Shuster's convention “DirectX® style”

$$\boldsymbol{ij} = -\boldsymbol{k}$$
$$\boldsymbol{jk} = -\boldsymbol{i}$$
$$\boldsymbol{ki} = -\boldsymbol{j}$$
$$\boldsymbol{ijk} = \boldsymbol{1}$$

- “Positive Direction” = left-hand rule.

$$R = \begin{bmatrix} 1-c^2-d^2 & 2bc+2ad & 2bd-2ac \\ 2bc-2ad & 1-b^2-d^2 & 2cd+2ab \\ 2bd+2ac & 2cd-2ab & 1-b^2-c^2 \end{bmatrix}$$

- “Sandwich” Product

$$V' = q^{-1} * V * q$$

Hamilton's convention “OpenGL® style”

$$\boldsymbol{ij} = \boldsymbol{k}$$
$$\boldsymbol{jk} = \boldsymbol{i}$$
$$\boldsymbol{ki} = \boldsymbol{j}$$
$$\boldsymbol{ijk} = -\boldsymbol{1}$$

- “Positive Direction” = right-hand rule.

$$R = \begin{bmatrix} 1-c^2-d^2 & 2bc-2ad & 2bd+2ac \\ 2bc+2ad & 1-b^2-d^2 & 2cd-2ab \\ 2bd-2ac & 2cd+2ab & 1-b^2-c^2 \end{bmatrix}$$

- “Sandwich” Product

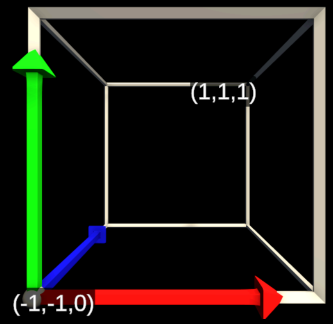
$$V' = q * V * q^{-1}$$

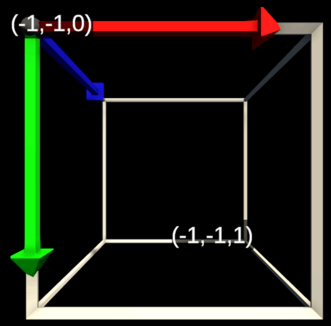
API FIXED COORDINATE SYSTEMS

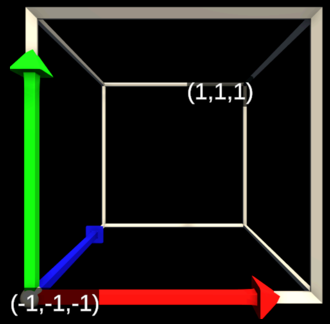
- The APIs are independent of the specific convolution of the coordinate space. With two exceptions:

1. Clip Space (or Normalized Device Coordinates Space).

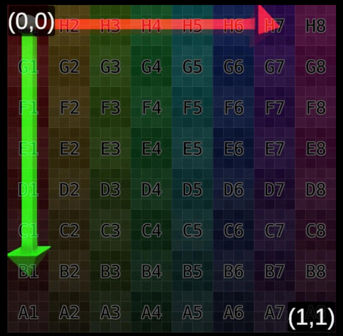
DirectX® | Vulkan®* | OpenGL®**

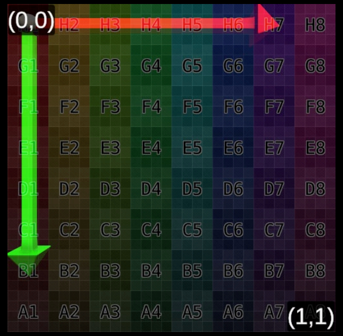


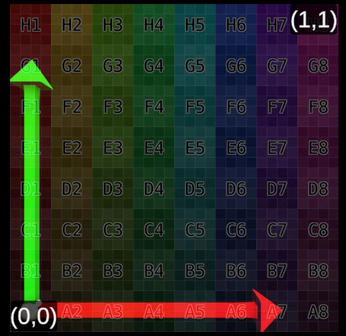




2. Texture Space (and Window Space).







AMD FidelityFX

AMD FidelityFX Super Resolution 2

AMD FidelityFX Denoiser

AMD FidelityFX Ambient Occlusion

AMD FidelityFX Variable Shading

AMD FidelityFX Screen Space Reflections

AMD TressFX

AMD FidelityFX Super Resolution

AMD FidelityFX Super Resolution 3

AMD FidelityFX Downsampler

AMD FidelityFX HDR Mapper

AMD FidelityFX Contrast Adaptive Sharpening

AMD FidelityFX Parallel Sort

AMD Cauldron Framework

AMD Render Pipeline Shaders

AMD Brotli-G SDK

AMD RADEON ProRender

AMD GPUOpen

AMD Compressonator

AMD RADEON ML

AMD RADEON Rays

AMD GPU Services Library

AMD Orochi

AMD HIP Ray Tracing

AMD Advanced Media Framework SDK

AMD GPU Performance API

AMD Display Library

AMD Device Library eXtra

AMD RADEON Developer Tool Suite

AMD RADEON Developer Panel

AMD RADEON GPU Analyzer

AMD RADEON GPU Profiler

AMD RADEON Memory Visualizer

AMD RADEON Raytracing Analyzer

THANK YOU FOR YOUR ATTENTION!

ANY QUESTIONS?

AMD

together we advance_