Quaternion Multiplication

x	a i	+	b j	+	c k	+	d	
e i		+		+		+		+
+								
f j		+		+		+		+
+								
g k		+		+		+		+
+								
h		+		+		+		

Quaternion Multiplication

$\vec{v}$

x	a i	+	b j	+	c k	+	d	
e i	-1 ae	+	k be	+	$-j$ ce	+	i de	+
+								
$\vec{w}$ f j	$-k$ af	+	-1 bf	+	i cf	+	j df	+
+								
g k	j ag	+	$-i$ bg	+	-1 cg	+	k dg	+
+								
h	i ah	+	j bh	+	k ch	+	hd	

$$(\vec{w}, h) \cdot (\vec{v}, d) = (\vec{w}\, d + \vec{v}\, h + \vec{w} \times \vec{v}, \; h\, d - \vec{w} \cdot \vec{v})$$

Rotations: defined by “from / to” versor pair

- Given two versors d0 and d1, find the minimal rotation moving s.t. R(d0) = d1
- exercise!
  - hint: which representation is easiest to find?

Recap: representing rotations 1/2

		3x3 Matrix	Euler Angles
Space efficient? (in RAM, GPU, storage…)		9 scalars	3 scalars (even small int!)
Efficient / easy to	Apply (to points/vectors)	9 products (3 dot products)	trigonometry sin/cos
	Invert (produce inverse)	just transpose	
	Cumulate (with another rotation)	matrix multiplication (9 dots)	
	Interpolate (with another rotation)		easy to do… unintuitive result
Intuitive? (e.g. to manually set)		?!?	roll & pitch & yaw
Notes…		Free skew + scale!	GIMBAL LOCK

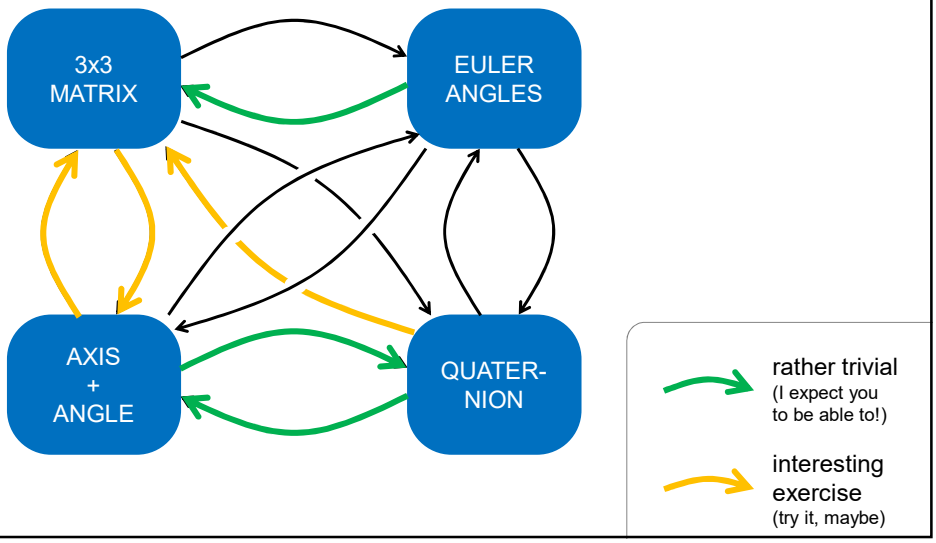
Recap: representing rotations 2/2

	axis + angle	(unitary) quaternion
Space efficient? (in RAM, GPU, storage…)	4 scalars (or 3) (but precision needed)	4 scalars (but precision needed)
Efficient / easy to: Apply (to points/vectors)	to matrix? + trigonometry	easy 2 quat product
Efficient / easy to: Invert (produce inverse)	super easy flip axis or angle	super easy flip imaginary or real part
Efficient / easy to: Cumulate (with another rotation)		super easy: 1 quat product
Efficient / easy to: Interpolate (with another rotation)	easy + best result	easy + good result (except angular speed)
Intuitive? (e.g. to manually set)	sometimes	not really
Notes…	*two* representations for each rotation (flip all → no effect) (for different reasons)	

And the winner is…

- Obviously, the quaternions
  - because they are more efficient with each operation
- Obviously, the Euler angles
  - because they are the most intuitive (and compact)
- Obviously, angle,axis
  - because they have the best LERP (easy and most natural result)
- Obviously, the 3x3 matrices
  - because they can also express (non uinf) scaling, and skewin
  - because its three columns are the X, Y, Z axes (useful)

Switching between representations



What defines a rotation, for you?

« Roll, pitch, and yaw! »

you are a… pilot, or an astronaut

« X-angle, Y-angle, and Z-angle! »

you are a… digital artist (an animator or a scener)

« An angle! »

you are a… flatland citizen

« A vector! the dir is the axis the magnitude the angle »

you are a… physicist

« A 3x3 matrix! the submatrix of a 4x4 transform »

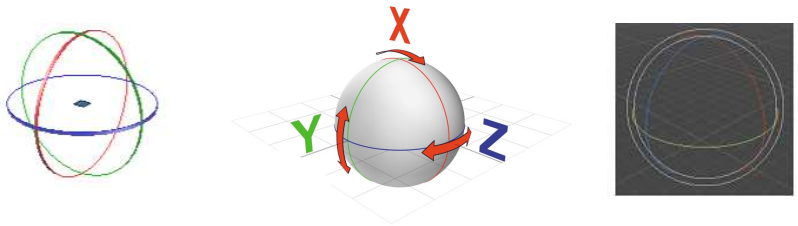
you are a… computer graphicist, or a Graphics API

« A quaternion! »

you are a game developer

GUI: how to author rotations in 3D?

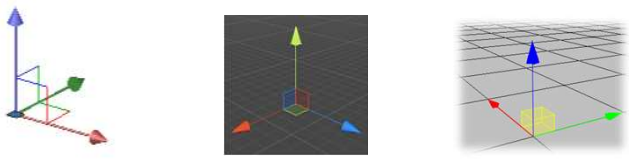
- Typical way: rotation gizmo
  - (also: «arcball» or «trackball»)
  - 3 handles to control the three Euler angles
  - or "free", drag-n-drop mode (trackball metaphor)



convention: Red = X Green = Y Blue = Z

GUI: how to author translations in 3D?

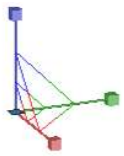
- translation gizmo
  - handles to traslate along axes or planes

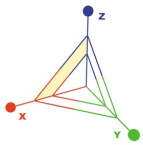


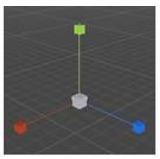
convention: Red = X Green = Y Blue = Z

GUI: how to author scalings in 3D?

- scale gizmo
  - 3 handles for anisotropic scalings
    1 handle (middle) for uniform scalings







convention: Red = X Green = Y Blue = Z

Rotations in unity (class Quaternion)

- In the GUI :
  - See / set as Euler Angles (intuitive)
- Internally:
  - Quaternions
- In the C# scripts:
  - programmer choice: quat, euler, axis+angle, matrices
    thanks to C# «properties»
    (setter/getter methods in disguise)
  - gives the illusion to be whichever kind you think they are

Rotations in UNREAL ENGINE

fields: `W X Y Z`

Class **`FQuat`** :

- convert from:
  - axis+angle, matrix4x4, FRotator, euler (vec3) (by constructors)
  - Euler angles (`makeFromEuler` method)
  - to-from vectors `FindBetween` method)
- convert to:
  - `ToAxisAndAngle`, `Euler`, `Rotator`,
  - matrix columns `GetAxisX,Y,Z`
  - also, with names: `Get`(`Up`|`Right`|`Forward`)`Vector`,
- methods: invert with `Inverse`,
  blend with `FastSlerp`
  or `FastSlerpFullPath` (no shortest path)
  apply with `RotateVector` / `UnrotateVector`
  cumulate with `operator *`

Class `FRotator`
for "nautical" Euler angles:
fields: `Pitch Roll Yaw`

Rotations in OpenGL

- In the «old school» API:
  (and now in many similar libraries)
  - API: glRotate3f
    - takes: angle+axis
  - Internally:
    - matrices
    - jointly as any other spatial transform
    - separated in MODEL+VIEW+PROJECT transform