Representations of 3D rotations

- 3x3 matrices
- Euler angles
- Axis + Angle
- Quaternions *This lecture!*

40

A flashback: Complex Numbers in a nutshell 1/3

- It all starts with a «fantasy» assumption, which is: there is an imaginary number i such that $i^2 = -1$
  - And for any other purpose, i behaves just like a (non-zero) Real number
- Consequences:
  - We now have number of the form $\underbrace{a}_{\text{real part}} + \underbrace{b}_{\text{imaginary part}}\, i$, with $a, b \in \mathbb{R}$, called complex numbers (the set is $\mathbb{C}$)
  - The algebra of complex numbers (how to sum, multiply, invert them…) is simply determined by the «fantasy» assumption above

41

A flashback: Complex Numbers in a nutshell 2/3

- For example, sum: *real part* *imaginary part*

$$(a + b\,i) + (c + d\,i) = (a + c) + (b + d)i$$

- For example, product (remembering $i^2 = -1$):

$$(a + b\,i) * (c + d\,i) = (ac - bd) + (ad + bc)i$$

- For example, inverse (check):

$$(a + b\,i)^{-1} = \frac{(a \; - b\,i)}{a^2 + b^2}$$

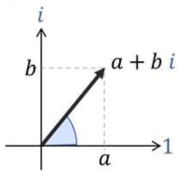
the «coniugate» of $(a + b\,i)$

the squared «magnitude» of $(a + b\,i)$

- What is interesting to us is the geometric interpretation of these objects & operations

42

A flashback: Complex Numbers in a nutshell 3/3



- Geometric interpretation:
  - $a + b\,i$ represents the vector/point (a, b)
  - Complex sum is vector sum
  - Complex conjugate is mirroring with the Real axis (horizontal)
  - Product is… add angles (with Real axis), multiply magnitudes
- Therefore,
  - product with a unitary (magnitude = 1) complex number is a pure 2D rotation
  - A complex number $c \in \mathbb{C}$ with $||c|| = 1$ represents a 2D rot; multiply vector $(x + y\,i)$ with c means to rotate it

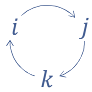
Wouldn't it be cool to have the same for 3D rotations?

43

Quaternions

- New «fantasy» assumption: there are three different "imaginary" numbers i , j , k such that:
  - for any other purpose, i, j, k behave like real numbers

$$\begin{cases} i^2 = k^2 = j^2 = -1 \\ ij = k \ , \quad ji = -k \\ jk = i \ , \quad jk = -i \\ ki = j \ , \quad kj = -j \end{cases}$$



as a table:

×	i	j	k
i	-1	$+k$	$-j$
j	$-k$	-1	$+i$
k	$+j$	$-i$	-1

- Consequences:
  - We now have number of the form $\underbrace{a\,i + b\,j + c\,k}_{\text{imaginary parts}} + \underbrace{d}_{\text{real part}}$, with $a, b, c, d \in \mathbb{R}$, called Quaternions (their set is $\mathbb{H}$)
  - The algebra of quaternions (how to sum, multiply, invert them…) is simply determined by the «fantasy» assumption
  - Again, what is interesting to us is the geometric interpretation…

45

Quaternions: how to write them (equivalently)

- Algebraic form: $a\,i + b\,j + c\,k + d$
  - often, omitting the zeros, e.g. $i + 2\,k$ is a quaternion
- As vectors of $\mathbb{R}^4$: $(a \ , \ b \ , \ c \ , \ d \ )$
- As vector & scalar pair: $(\vec{v} \ , \ d)$

imaginary part, a vector $\begin{pmatrix} a \\ b \\ c \end{pmatrix}$ real part, a scalar

- Conjugate of a quaternion: invert the sign of the imaginary part

46

Quaternions: operations how-to

$$q \in \mathbb{H} \qquad\qquad q = ai + bj + ck + d$$

- Sum, Scale, Interpolate , etc.: trivial
  - same as 4D vectors
- Magnitude

$$\|q\| = \sqrt{a^2 + b^2 + c^2 + d^2}$$
$$\|q\|^2 = a^2 + b^2 + c^2 + d^2$$

  - «unitary» if it's 1
  - same as 4D vectors

47

Quaternions: operations how-to

$$q \in \mathbb{H} \qquad\qquad q = ai + bj + ck + d$$

- Product: just apply «fantasy» assumptions
  - Observe: product is not commutative (nor anticommut.)
  - (see next 3 slides for the math)
- «Coniugate»:

Flip imaginary parts

  - like for complex numbers: $\bar{q} = -ai - bj - ck + d$
- Inverse: (like for complex numbers) $q^{-1} = \bar{q} \,/\, \|q\|^2$
  - For unitary quat, it's just the coniugate

48

Quaternion Product

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

50

Quaternion Product

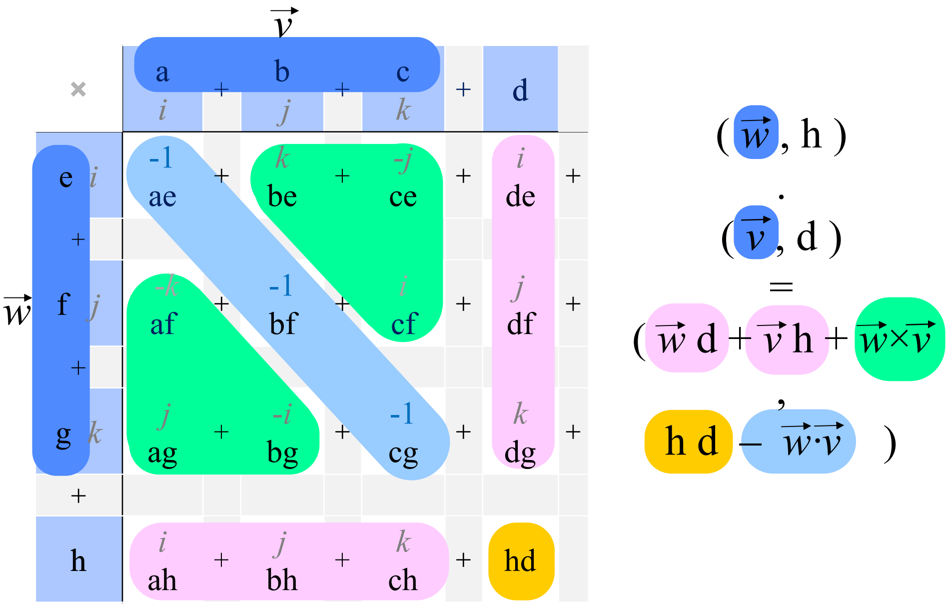
$\vec{v}$ = (a, b, c); $\vec{w}$ = (e, f, g)

×	a i	+	b j	+	c k	+	d	
e i	-1 ae	+	k be	+	$-j$ ce	+	i de	+
+								
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	

$(\vec{w}, h) \cdot (\vec{v}, d) =$ (some vector , some scalar)

51

Quaternion Product



52

Quaternions: Geometric Interpretation!

- A quaternion $\mathrm{q} = (\vec{\mathrm{v}}\ ,\ d\ )$ represents :
  - the 3D point or vector $\vec{\mathrm{v}}$, when $d = 0$
  - a 3D rotation, when q is unit, i.e. $\|\mathrm{q}\|^2 = \|\vec{\mathrm{v}}\|^2 + d^2 = 1$
  - (neither, otherwise)
- If q is a rotation and p is a point ($\mathrm{q}, \mathrm{p} \in \mathbb{H}$) then…
  - $\mathrm{q} \cdot \mathrm{p} \cdot \bar{\mathrm{q}}$ is the rotated point / vector
  - $\bar{\mathrm{q}}$ is the inverse rotation
  - $\mathrm{q}_0 \cdot \mathrm{q}_1$ is the composited rotation (first q_1 then q_0)
  - (so, $\bar{\mathrm{q}} \cdot \mathrm{p} \cdot \mathrm{q}$ is the pt rotated… in the *other* direction)

53

Compositing Quaternions: why it works

$q_0, q_1, p \in \mathbb{H}$
q_0, q_1 represent rotations
p represents a point

$$\overbrace{q_0 \cdot \overbrace{(q_1 \cdot p \cdot \bar{q}_1)}^{\text{p rotated by q1}} \cdot \bar{q}_0}^{\text{p rotated by q1, rotated by q0}}$$

product is associative
(like for complex numbers)

$$=$$

$$(q_0 \cdot q_1) \cdot p \cdot (\bar{q}_1 \cdot \bar{q}_0)$$

$\bar{r} \cdot \bar{s} = \overline{s \cdot r}$
(rules of quaternions)
(remember: product is not commutative)

$$=$$

$$(q_0 \cdot q_1) \cdot p \cdot \overline{(q_0 \cdot q_1)}$$

54

3D Rotations as Quaternions

- quaternion q representing the 3D rotation of angle α around axis $\hat{a}$:
  - $q = \left(\sin\left(\frac{\alpha}{2}\right) \hat{a} \,,\; \cos\left(\frac{\alpha}{2}\right) \right)$

  that is
  - $q = \sin\left(\frac{\alpha}{2}\right) \hat{a}_x i + \sin\left(\frac{\alpha}{2}\right) \hat{a}_y j + \sin\left(\frac{\alpha}{2}\right) \hat{a}_z k + \cos\left(\frac{\alpha}{2}\right)$
- Observe that $||q||^2 = 1$

verify

55

3D Rotations as Quaternions: a problem

- Around axis $\hat{a}$ by angle α :

$$q = \left(\sin\left(\frac{\alpha}{2}\right) \hat{a} \,, \cos\left(\frac{\alpha}{2}\right) \right)$$

- Around axis $-\hat{a}$ by angle $(-\alpha)$: (it's the same rotation!)

$$q' = \left(-\sin\left(\frac{-\alpha}{2}\right) \hat{a} \,, \cos\left(\frac{-\alpha}{2}\right) \right) = q$$

same quaternion :-)

Good! But:

- Around axis $\hat{a}$ by angle $(\alpha + 360°)$: (it's the same rotation!)

$$q'' = \left(\sin\left(\frac{\alpha}{2} + 180°\right) \hat{a} \,, \cos\left(\frac{\alpha}{2} + 180°\right) \right) =$$

$$= \left(-\sin\left(\frac{\alpha}{2}\right) \hat{a} \,, -\cos\left(\frac{\alpha}{2}\right) \right) = -q$$

different quaternion :-(

- Conclusion:
  quaternion q and quaternion $-q$ encode the same rotation

56

3D Rotations as Quaternions: a problem

Given a quaternion which is a rotation:

- Flip its real part: invert rotation
- Flip its imaginary part (conjugate): same
- Flip everything: same rotation

Every rotation is encoded
by two different quaternions.

57

Interpolating two quaternions representing rotations

Good results, but two *caveats*:

⚠ Take the "shortest path" (as usual):
flip 2nd quaternion first, if this makes them closer

- Distance defined as dot product in 4D
  (they are 4D unit vectors!)

⚠ Loss of normality

- Needs re-normalization (NLERP),
- Or SLERP
  (again, consider them 4D unit vectors)

58

Quaternions: exercises

- Which quaternion encodes a turnabout?
  - (ita: «*un dietrofront*»: turning 180° around the up vector?)
- Apply that quaternion to rotate a point in (x,y,z)
  - Use plain quaternion algebra, and algebraic notation
- Which quaternion encodes the identity rotation?
  - Is it the only one? If not, which other does?
  - Verify by applying it (or them)
- Which quaternion encodes a turn of 90° to the left?
- Uses your previous *two* answers to find the quat. encoding turn 45° to the left, *by using interpolation*
  - Do you need SLERP in this case? Is NLERP enough? Why?
  - Verify the solution is correct using the axis-angle formula

59

Quaternions as rotations

- Almost as compact as possible to store (4 scalars)
- Trivial to invert
- Fast to composite
- Fast to apply
- Easy to ensure they are still rotations (just normalize)
  - Even after long sequences of cumulations, unlike matrices
- Behaves well under interpolation
  - Even with just NLERP – better with SLERP
- The favourite representation in 3D games
  - but, other solutions still useful in one context or another

60

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
Efficient / easy to: Invert (produce inverse)	super easy transpose, 3 swaps	
Efficient / easy to: Composite (with another rotation)	matrix multiplication (9 dots)	
Efficient / easy to: Interpolate (with another rotation)		easy to do… bad result
Intuitive? (e.g. to manually set)	?!?	roll & yaw & pitch
Notes…	Free extra skew + scale!	GIMBAL LOCK

62

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
	Invert (produce inverse)	super easy flip axis or angle	super easy flip imaginary or real part
	Cumulate (with another rotation)		super easy: 1 quat product
	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)	

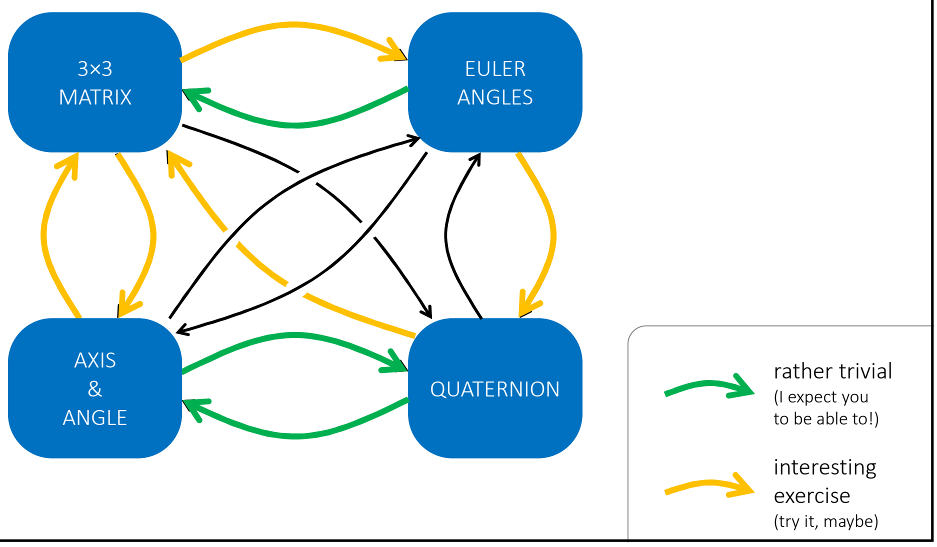
63

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-and-axis
  - because they have the best MIX (easy + most natural results)
- Obviously, the 3x3 matrices
  - because they can also express (non uinf) scaling, and skewing
  - because its three columns are the X, Y, Z axes of the local space (useful)

66

Switching between representations



67

What defines a rotation, for you?

« Roll, pitch, and yaw! »

then you are… a pilot, or an astronaut

« X-angle, Y-angle, and Z-angle! »

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

« An angle! »

then you are… a flatland citizen

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

then you are… a physicist

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

then you are… a computer graphicist, or a Graphics API

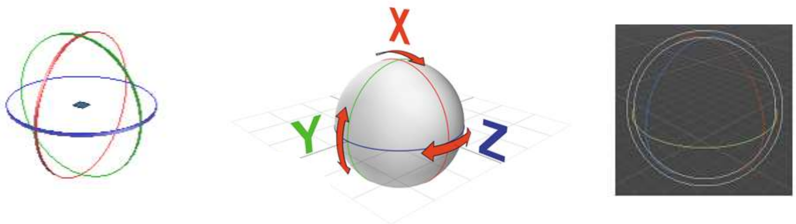
« A quaternion! »

then you are… a game developer

68

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)

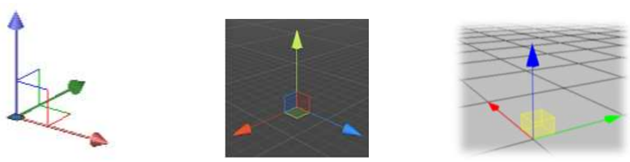


convention: Red = X Green = Y Blue = Z

69

GUI: how to author translations in 3D?

- translation gizmo
  - handles to traslate along axes or planes

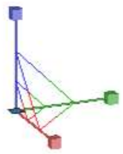
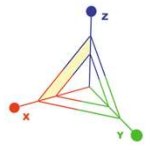
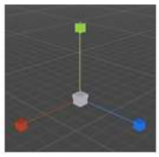


convention: Red = X Green = Y Blue = Z

70

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

71

Rotations in unity (class Quaternion)

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

72

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)
  - From-to vector pairs (`FindBetween` method)
- convert to:
  - `ToAxisAndAngle`, `Euler`, `Rotator`,
  - matrix columns `GetAxis`(`X`|`Y`|`Z`)
  - also, with names: `Get`(`Forward`|`Right`|`Up`)`Vector`,
- methods: invert with `Inverse`,
  blend with `FastSlerp`
  or `FastSlerpFullPath` (no shortest path)
  apply with `RotateVector` / `UnrotateVector`
  composite with `operator *`

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

73

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

74