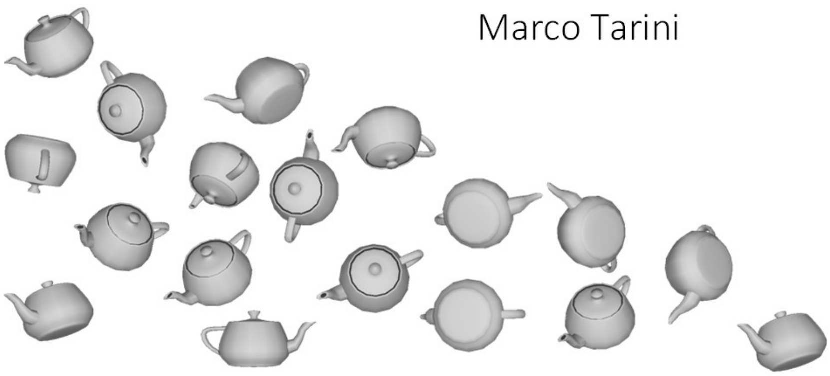


3D videogames 2023-2024

Representing Rotations

Marco Tarini



1

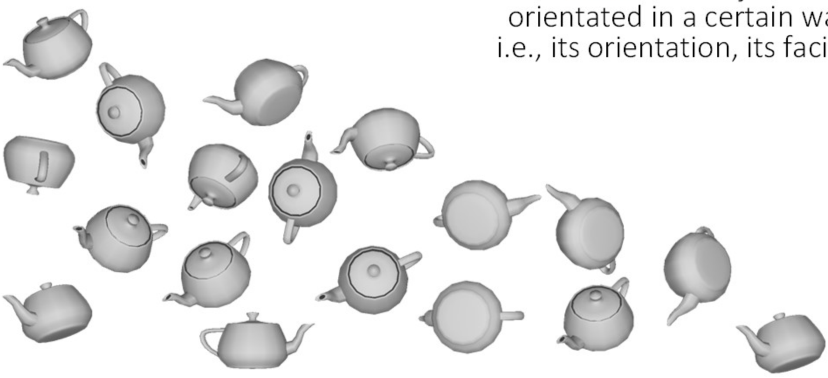
Course Plan

lec. 1: Introduction
lec. 2: Mathematics for 3D Games
lec. 3: Scene Graph
lec. 4: Game 3D Physics +
lec. 5: Game Particle Systems
lec. 6: Game 3D Models
lec. 7: Game Textures
lec. 9: Game Materials
lec. 8: Game 3D Animations
lec. 10: Networking for 3D Games
lec. 11: 3D Audio for 3D Games
lec. 12: Rendering Techniques for 3D Games
lec. 13: Artificial Intelligence for 3D Games

2

How to (internally) represent a 3D rotation ?

Used to encode:
the *act* of spinning an object around,
but also
the *state* of an object to be orientated in a certain way, i.e., its orientation, its facing



4

Plan for this lecture (and the next two)

- We will discuss the internal representations for 3D rotations
- We want representations that allow easy / efficient
  - storage
  - application (to points, vectors & versors)
  - composition (with another rotation)
  - inversion
  - interpolation (with another rotation)
  - assignment (creation by humans, e.g., by manual specification)
- We will see several solutions, with pros and cons

5

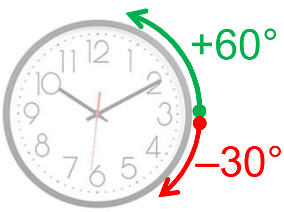
Preamble: representing *translations* in 3D

- Trivial:
  displacement vector (3 scalars)!
  - perfect under all criteria
    (exercise: verify)

6

Preamble: representing rotations *in 2D*

- Trivial!



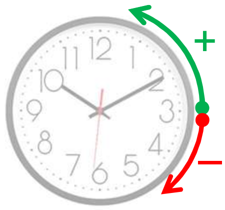
7

Preamble: representing rotations *in 2D*

- Trivial representation:
  one angle (a dimensionless scalar)

a «pseudo-scalar»
(as it changes sign if we mirror the scene)

- Semantic (traditionally):
  - If positive: counter-clockwise
  - If negative: clockwise



- Choices:
  - unity of measure: degree or radians?
  - which interval? E.g. [0..360) or (-180..+180]

8

Preamble: representing rotations *in 2D*

- Is it convenient to…
  - Store?
    ✓ *it's one scalar*
  - Apply?
    ✓ *easy & fast:*

two constants
(for a given α)

$$r_\alpha \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \cos(\alpha)\, x - \sin(\alpha)\, y \\ \sin(\alpha)\, x + \cos(\alpha)\, y \end{pmatrix}$$

  - Invert?
    ✓ *Just flip the sign*
  - Cumulate?
    ✓ *Just sum the two angles (modulo 360°)*
  - Design / manually assign?
    ✓ *easy.*
    E.g. 0° = east. 90° = north. 180° = west. 270° = South.

and N-E = 45°

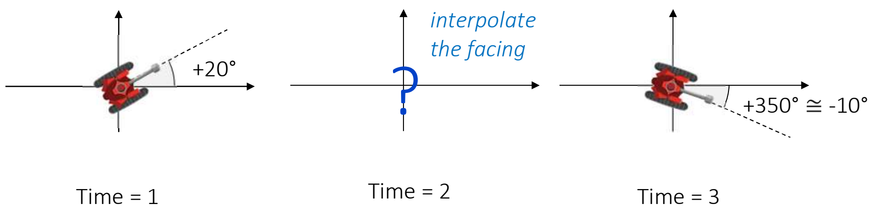
9

Preamble: representing rotations *in 2D*

- Is it convenient to…
  - Interpolate?

Can we just… $\text{mix}(\ \alpha, \beta, t\ )$ $\alpha\,(1-t) + \ \beta\ t$

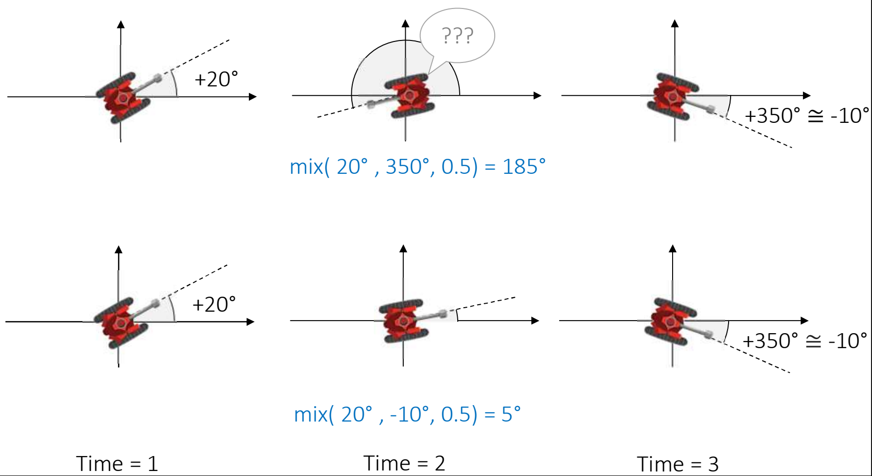
  - Example: mid-way between North = 90° and West = 180°
    $\text{mix}(\ 90°, 180°, 0.5\ )$ = 135° = NW … checks out!
  - But consider this case:



10

Preamble: representing rotations *in 2D*

- Which is the correct interpolation? *E.g. in an animation*



11

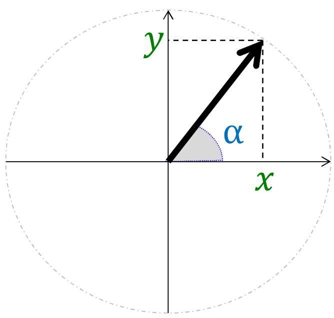
Preamble:
representing rotations *in 2D*

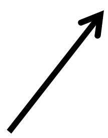
- Is it convenient to... interpolate? ✓ *Yes, but,*
- Problem: angles α and α+360° are equivalent
  $\alpha \cong \alpha + 360 \cong \alpha + k\,360°$ (any $k \in \mathbb{Z}$)
- General solution:
  to interpolate between two rotations α and β ...
  1. Find β' equivalent to β that is most similar to α (here: choose between β and $\beta - 360°$) — aka "take the shortest path"
  2. Linear interpolation (mix) between α and β' (not β)
- We will encounter the same problem/solution again...

12

Preamble:
representing rotations *in 2D*

- How to go *angle* ⟶ *2D-versor*

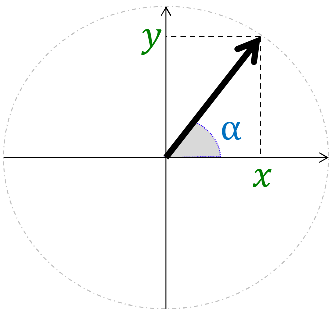



$= \begin{pmatrix} \cos(\alpha) \\ \sin(\alpha) \end{pmatrix}$

13

Preamble:
representing rotations *in 2D*

- How to go *2D-versor* ⟶ *angle*

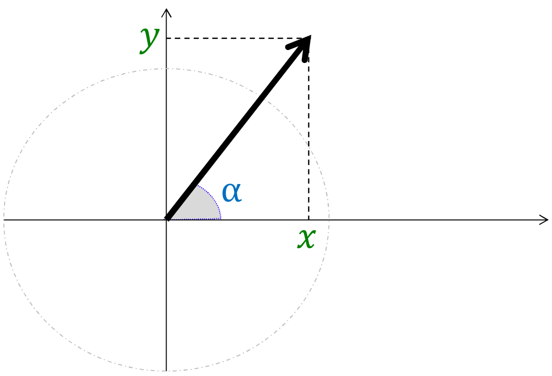


pro tip: use atan**2** in any language: $\alpha = \text{atan}\mathbf{2}(y, x)$

14

Preamble:
representing rotations *in 2D*

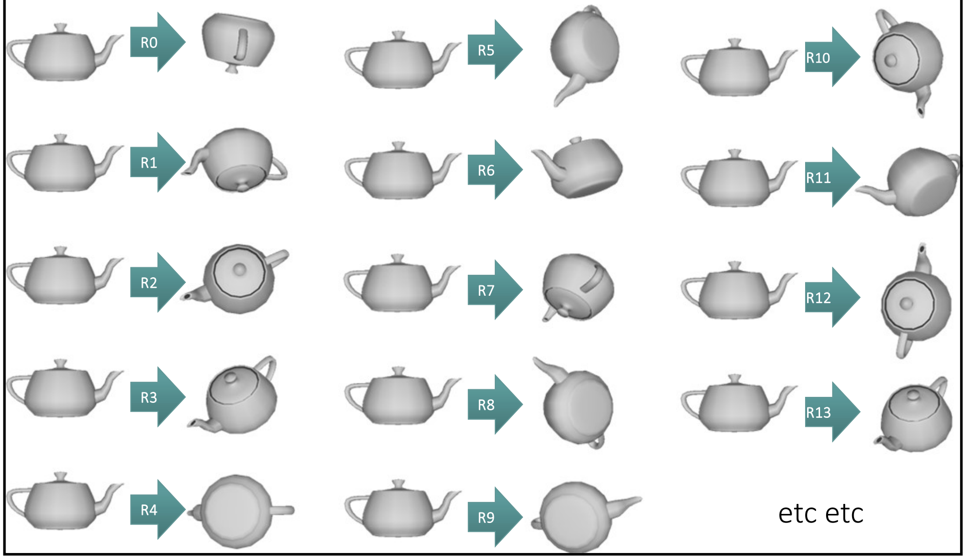
- How to go *2D-vector* ⟶ *angle*



still use atan**2** ! The scale is irrelevant: $\alpha = \text{atan}\mathbf{2}(y, x)$

15

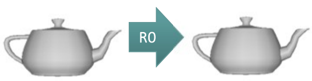
3D rotations: how many dimension?



etc etc

16

3D rotations: how many dimension?



(clearly, they include the identity too)

Answer: 3 DOF (degrees of freedom).
i.e., rotations in 3D are a “3-dimensional group”.
(which is called “SO(3)”)

17

Solution 1: rotations are 3x3 matrices

- A rotation
  =
  a 3x3 matrix

$$R$$

orthonormal,
determinat = +1

- Application
  =
  matrix / vector multiplciation

$$R \cdot v = v'$$

input
vector, point, versor
(cartesian coords)

rotated
vector, point, versor
(cartesian coords)

18

Rotations as 3x3 matrices

No translation.
i.e: the origin stays fixed.
i.e.: the rotation axis
passes through the origin

- Rotation =
  the 3x3 submatrix of a 4x4 «pure» rotation affine matrix

$$\begin{array}{cccc} & R & & \begin{matrix}0\\0\\0\end{matrix} \\ 0 & 0 & 0 & 1 \end{array}$$

Note: by combining with translations, we can obtain rotations around any point

19

Rotations as 3x3 matries

- Wasteful in RAM (9 scalars, versus a minimum of 3)
- Easy to apply (matrix-vector prod: 9 mults)
- Relat. easy to compose (matrix-matrix prod: 27 x mult)

why?

- Interpolate: problematic:

$$k\,\boxed{R_0} + (1-k)\,\boxed{R_1} = \boxed{M}$$

NOT a rotation
(NOT orthonormal)

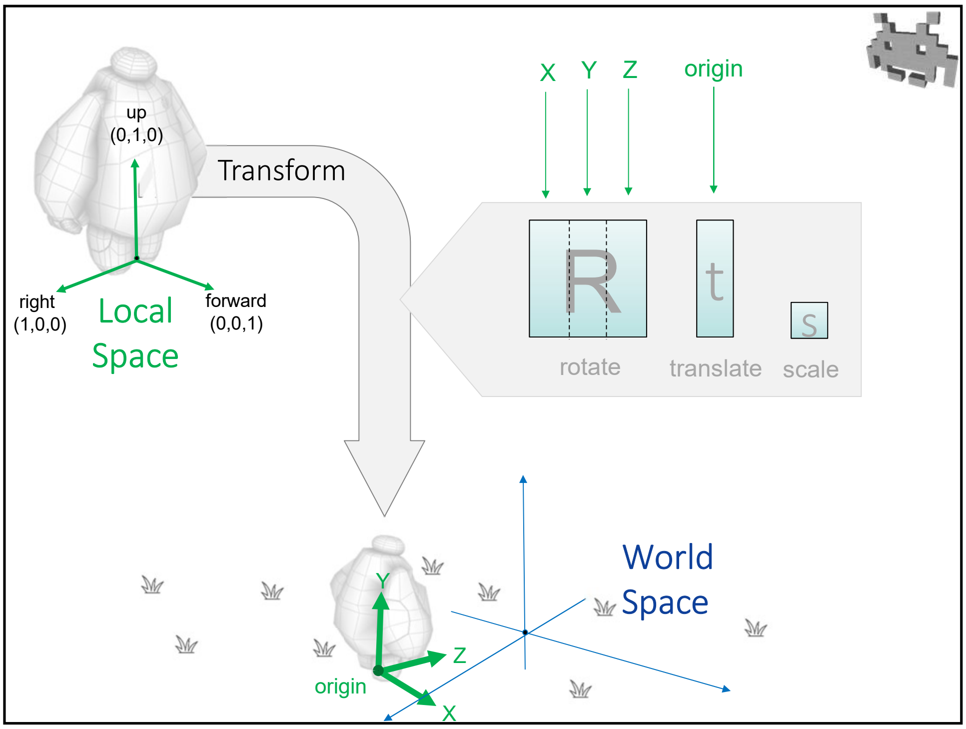
20

Rotations as 3x3 matrices (9 scalars): compositions

- Multiplying matrices composites the rotation
  - remember: neither matrix-matrix product, nor composition of 3D rotations, is commutative!
- e.g.: $R_{TOT} = R_0 \cdot R_1$
  - rotate as R_1 followed by R_0
  - with $R_0 \cdot R_1$ rotation matrices
  - i.e. orthonormal matrices with det = 1
- R_{TOT} is a rotation matrix too, *in theory*

⚠ in practice, approximation errors can break that
  - especially after long sequences of compositions.

21



23

Rotations as 3x3 matrices (9 scalars)
A useful property

- its three columns encode
  the three versors representing
  the X , Y , Z axis of the *local* space
  expressed in global space
  - i.e. the world-space versors
    representing local right, upward, forward (in Unity)
    or local forward, right, upward (in Unreal engine)

24

Rotations as 3x3 matries: Inversion

$\hat{x} \cdot \hat{x} = ||\hat{x}||^2$

$$\begin{bmatrix} \hat{x} \\ \hat{y} \\ \hat{z} \end{bmatrix} \cdot \begin{bmatrix} \hat{x} & \hat{y} & \hat{z} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$\hat{y} \cdot \hat{z}$

$$R^T \cdot R = I$$

- For rotation matrices:
  to transpose = to invert

Just swap three
pairs of elements!

25

Rotations as 3x3 matrices (9 scalars) A useful property

The columns of
its transposed

- its three rows encode
  the three versors representing
  the X , Y , Z axis of the *global* space
  expressed in local space
  - i.e. the three local-space versors
    representing the
    global eastward, upward, northward
    directions (for example)

26

Recap: a score sheet :-)

		3x3 Matrix	
Space efficient? (in RAM, GPU, storage…)		★☆☆☆☆	9 scalars
Efficient / easy to	Apply (to points/vectors)	★★★★☆	9 products (3 dot products)
	Invert (produce inverse)	★★★★★	just transpose (three swaps)
	Composite (with another rotation)	★★☆☆☆	Matrix multipl (9 dot products) Numerical errors
	Interpolate (with another rotation)	★☆☆☆☆	Introduces shear/scale
Intuitive? (e.g., to manually set)		★☆☆☆☆	Impossible to manually set
Notes…		Useful to extract local axes.	

27

Representations of 3D rotations

- 3x3 matrices
- Euler angles
  - the most intuitive way to express a rotation
  - e.g., well understood by digital artists!

28

Rotations as Euler angles (3 scalars)

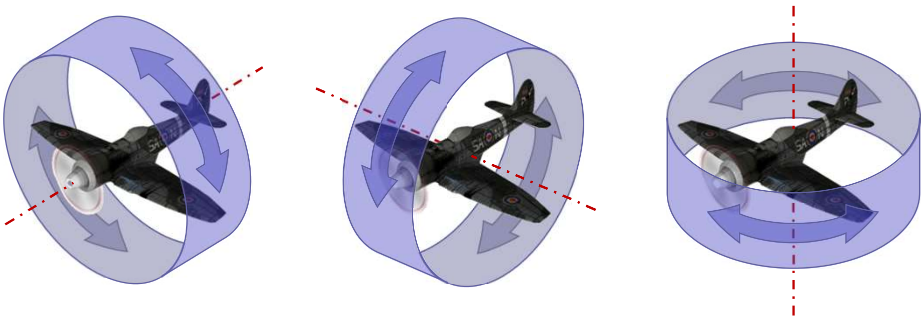
- Any 3D rotation can be expressed as:
  - a rotation around X axis (by α degrees), followed by:
  - a rotation around Y axis (by β degrees), followed by :
  - a rotation around Z axis (by γ degrees):
- Angles α β γ :
  "Euler angles" of a specific rotation
  - therefore: the "coordinates" of that rotation

this order (X-Y-Z) is chosen arbitrary but once and for all!
(in a given game engine / lib / etc)

29

Rotations as Euler angles (3 scalars)

- In nautical / aeronautical language, the three angles have names:



roll
(*rollio*)

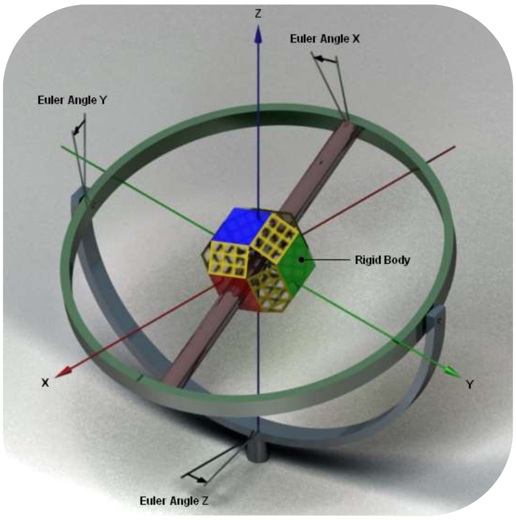
pitch
(*beccheggio*)

yaw
(*imbardata*)

30

Rotations as Euler angles (3 scalars)

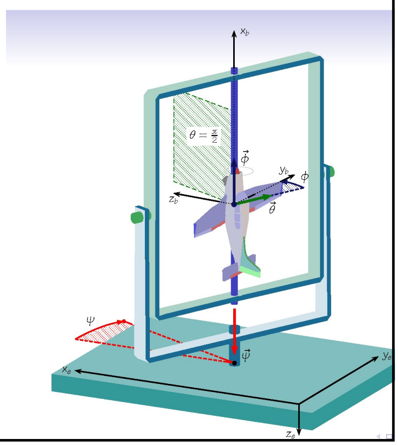
- A physical implementation: “three axes globe”



31

Rotations as Euler angles (3 scalars)

- Is it 1:1 ?
  - 1 rotation ⇔ 1 euler angle triplet ?
- Almost
  - assuming angles are properly bounded (exercise: how?)
- Ugly exception: “GIMBAL LOCK”
  - when 1st rotation makes the axes of the next two axes *coincide*
  - this cannot be avoided, no matter how axes are chosen



32

Rotations as Euler angles (3 scalars)

- Conciseness: perfect! 3 scalars for 3 DOF
- Application : a bit work-intensive
  - three rotations in succession
- Interpolation : you can do that...
  - just interpolate the three angles
  - ⚠ (remember to always "pick the *shortest path*" whenever interpolating angles: that is, must take in account the $\alpha \approx \alpha + 360\,k$ equivalence)

  ...but results won't always be nice !
- Composite / invert: not easy nor immediate...

exercise: why just summing / flipping the three angles won't work?

33

Comparing representations (so far)

		3x3 Matrix	Euler Angles
	Space efficient? (in RAM, GPU, storage...)	★☆☆☆☆ 9 scalars	★★★★★ 3 scalars (even as small int!)
Efficient /easy to	Apply (to points/vectors)	★★★★☆ 9 products (3 dot products)	★☆☆☆☆ requires trigonometry sin/cos
Efficient /easy to	Invert (produce inverse)	★★★★★ just transpose	★☆☆☆☆
Efficient /easy to	Composite (with another rotation)	★★☆☆☆ Matrix multipl (9 dots) Numerical errors	★☆☆☆☆
Efficient /easy to	Interpolate (with another rotation)	★☆☆☆☆ Introduces shear/scale	★☆☆☆☆ easy to do, unintuitive result (⚠ shortest-path required!)
	Intuitive? (e.g. to manually set)	★☆☆☆☆	★★★★★ roll yaw pitch
	Notes...	Free extra shear + scale. Useful to extract local axes.	⚠ GIMBAL LOCK

34

Representations of 3D rotations

- 3x3 matrices
- Euler Angles
- Axis + angle
  - Most common way in physics (and *game* physics)

36

Rotations as axis & angle

- Any rotation can be expressed as:
  - *one* rotation by some angle around some axis

appropriately chosen

- Angle: a scalar
- Axis: a versor (3 scalars)
  - note: the axis is considered to pass around the origin. For the more general case, combine with translations.

37

Rotations as axis & angle

- Compactness: good, 4 scalars
  - Just one more than bare minimum
- Ease of application: not too good ☹
  - Ways include: switch to 3x3 matrix (exercise: how to)
  - Switch to a quaternion: see later
  - "Rodrigues' rotation formula" (look it up)
  - Note: they all require trigonometric function (sin, cos)
- Invert: super easy / quick
  - just flip the angle sign *or* the axis vector
  - question: what if both?
    answer: Rotation is inverted twice:
    it's back to the same rotation again! 🤔

38

Rotations as axis & angle: equivalent representations

- Therefore: $\big(\underbrace{a_x\ ,\ a_y\ ,\ a_z}_{\text{axis}}\ ,\ \underbrace{\alpha}_{\text{angle}} \big)$
  and $\big(-a_x, -a_y, -a_z\ , -\alpha \big)$
  represent the same rotation
- Any rotation has two equivalent representations in this format
  - except the identity, which has infinitely many:
    angle α = 0, with any axis $\hat{a}$ = (a_x , a_y , a_z)
- This is always a bit inconvenient!
  - Complicates interpolation ("shortest path" necessary)
  - Complicates testing for equality/similarity, etc.

39

Rotations as axis & angle

- Compositing rotations:
  not at all immediate or easy to do ☹
- Interpolating rotations: very good!
  - Just interpolate axis and angle separately
  - Some *caveat*:
    - 1) *shortest path* for axes: first, flip either rotation (both its axis & angle) when this makes the two axes closer (how to test?)
    - 2) *shortest path* for angles: as usual, angles must then be interpolated… «modulo 360°»,
    - 3) interpolate between axes requires SLERP or NLERP (when interpolating versors)
    - 4) beware degenerate cases (opposite axes); point 1 avoids this
  - best results!
    Usually produces the "expected" intermediate rotation

40

Rotations as axis and angle, variant: as axis *times* angle

Sometimes called «pseudo-vector» because it flips sign if the world is mirrored

- axis: $\hat{a}$ (versor, $\|\hat{a}\| = 1$)
- angle: α (scalar)
- can be represented as one vector $\vec{a}$ (3 scalars)
  $\vec{a} = \alpha\, \hat{a}$
  - angle $\alpha = \|\vec{a}\|$
  - axis $\hat{a} = \vec{a} \,/\, \alpha$
  - note: when $\alpha = 0$, the axis is lost… it's ok, we don't need it!
- more compact, but otherwise equivalent
  - actually, better:
    we now have only 1 representation per rotation (why?)
    … including the identity (why?)

41

Rotation. representations: score sheet (continued)

		axis , angle	
Space efficient? (in RAM, GPU, storage…)		★★★★☆	4 scalars (or 3) (precision needed)
Efficient / easy to	Apply (to points/vectors)	★★☆☆☆	Requires trigonometry
	Invert (produce inverse)	★★★★★	Just flip axis OR angle
	Composite (with another rotation)	★☆☆☆☆	
	Interpolate (with another rotation)	★★★★★	
Intuitive? (e.g. to manually set)		★☆☆☆☆	Not easy to see which axis to use
Notes…		*Best if axis scaled by angle*	

42

Representations of 3D rotations

- 3x3 matrices
- Euler angles
- Axis , Angle
- Quaternions

Next lecture!

43

