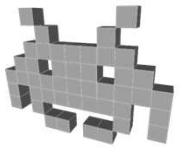
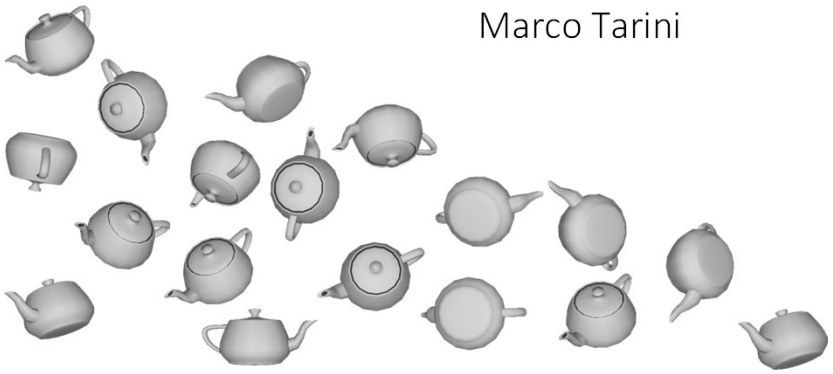
3D videogames

Representing 3D Rotations



Marco Tarini



1

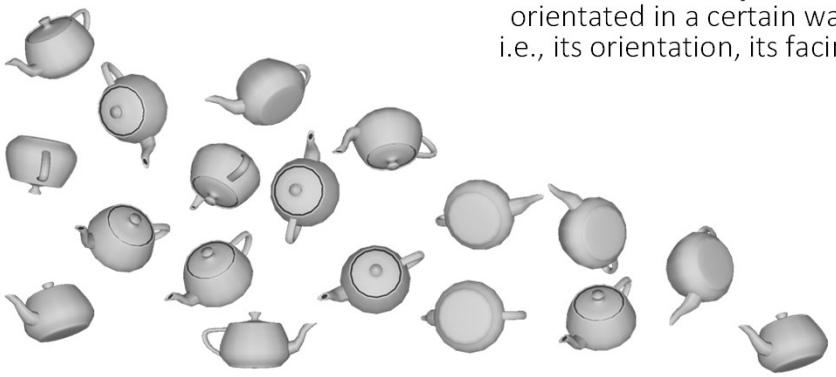
Course Plan

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: **3D Audio** for 3D Games
lec. 11: **Networking** for 3D Games
lec. 12: **Artificial Intelligence** for 3D Games
lec. 13: **Rendering Techniques** for 3D Games

2

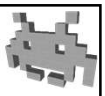
How to encode a 3D rotation ?

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



7

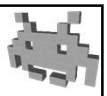
Plan for this lecture



- We will explore the internal representations for a 3D rotation (an *action*, a *motion*)
  - Which also serve as representations for an orientation (a *state*)
- We need representations that allow easy / efficient
  - storage
  - application (to points, vectors & versors)
  - composition (with another rotation)
  - inversion (of a given rotation)
  - interpolation (with another rotation)
  - assignment (creation by humans, e.g., intuitive manual specification by artists)
- We will see several solutions, each with pros and cons
  - And then, exercises

8

Preamble: representing *translations* (in 3D)



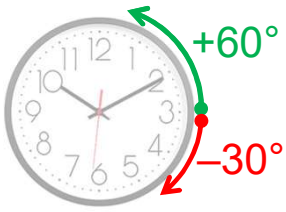
- How to represent a translation
  - That is, an *action*, a *motion*
  - Which also represents the object position - a *state*
- Trivial:
  displacement vector (3 scalars)!
  - equivalently, a point (when considered a *state*)
  - perfect under all criteria above
    (quick exercise: verify each)

9

Preamble: representing rotations *in 2D*

(if this course was titled "2D videogames")

- Trivial!



10

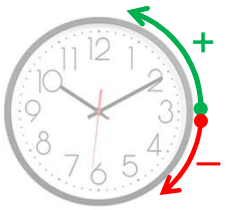
Preamble: representing rotations *in 2D*

(if this course was titled "2D videogames")

- Trivial representation: one angle (a dimensionless scalar)

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

- Semantic (traditionally):
  - 0° ⇔ three 'o clock
  - If positive: counter-clockwise
  - If negative: clockwise



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

11

Preamble: representing rotations *in 2D*

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

two scalar constants (for a given α)

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

and N-E = 45°

12

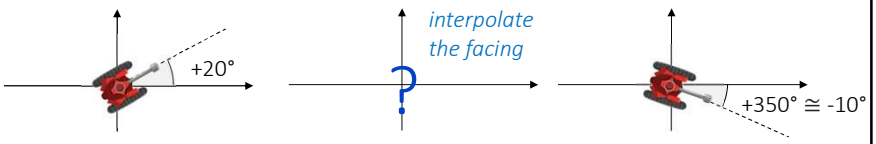
Preamble: representing rotations *in 2D*

- How to *interpolate* it?

$\alpha\,(1-t) + \beta\,t$

Can we just… $\mathrm{mix}(\,\alpha, \beta, t\,)$

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



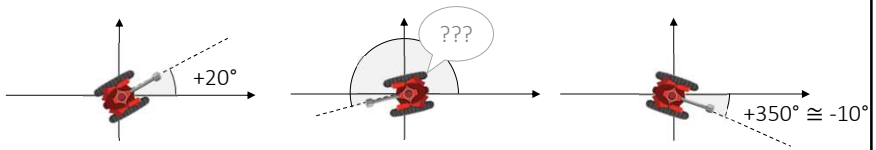
Time = 1 Time = 2 Time = 3

13

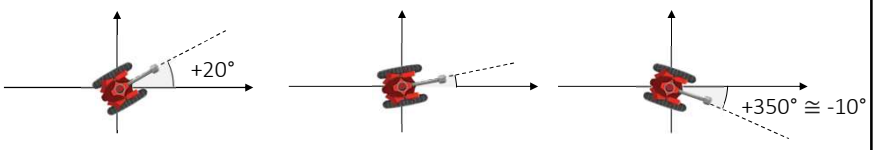
Preamble: representing rotations *in 2D*

E.g. in an animation

- Which is the correct interpolation?



mix(20° , 350°, 0.5) = 185°



mix(20° , -10°, 0.5) = 5°

Time = 1 Time = 2 Time = 3

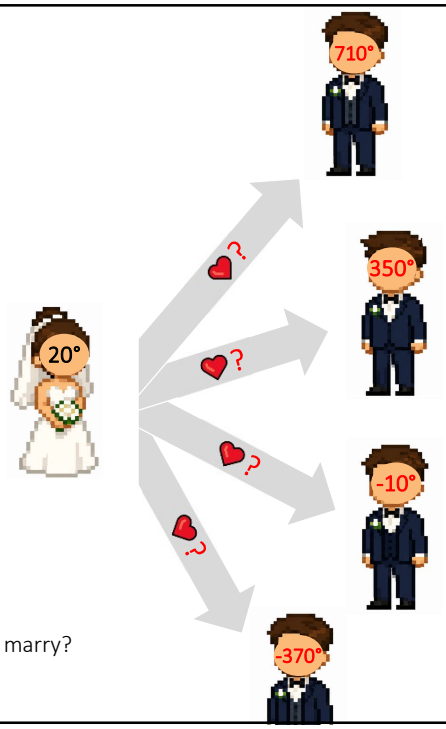
14

Preamble: representing rotations *in 2D*

- Is it convenient to… interpolate? ✓ *Yes, but…*
- Problem: equivalent representation of the same rotation (as a state! Interpolation is about states):
- Here: angles α and α+360° are equivalent
  $\alpha \approx \alpha + 360 \approx \alpha + k\ 360°$ (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 } aka "take the shortest path"
     (here: choose between β and $\beta - 360°$)
  2. Interpolation (mix) between α and β' (not β)

15

Shortest path for dummies

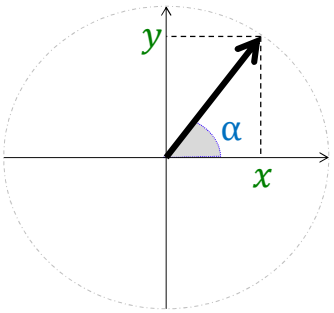


Which of the equivalent brothers to marry?
Answer: **«like attracts like»**
(ita: «*chi si assomiglia si piglia*»)

16

Preamble: representing rotations *in 2D*

- How to go *angle* ⟶ *2D-versor*

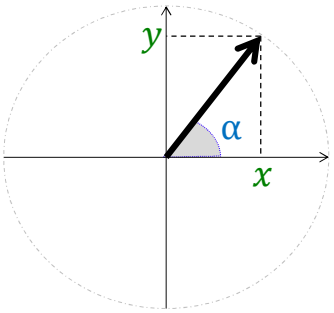


$$= \begin{pmatrix} \cos(\alpha) \\ \sin(\alpha) \end{pmatrix}$$

17

Preamble: representing rotations *in 2D*

- How to go *2D-versor* ⟶ *angle*

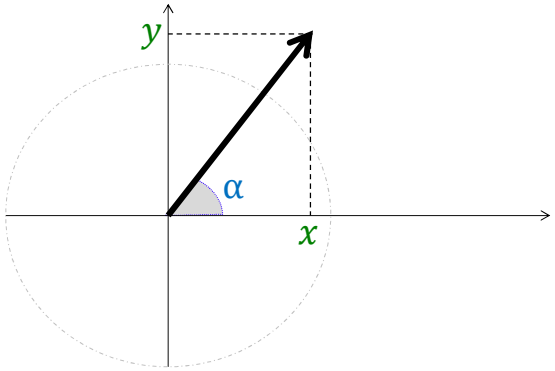


pro tip: use $\mathrm{atan2}$ in any language: $\alpha = \mathrm{atan2}(y, x)$

18

Preamble: representing rotations *in 2D*

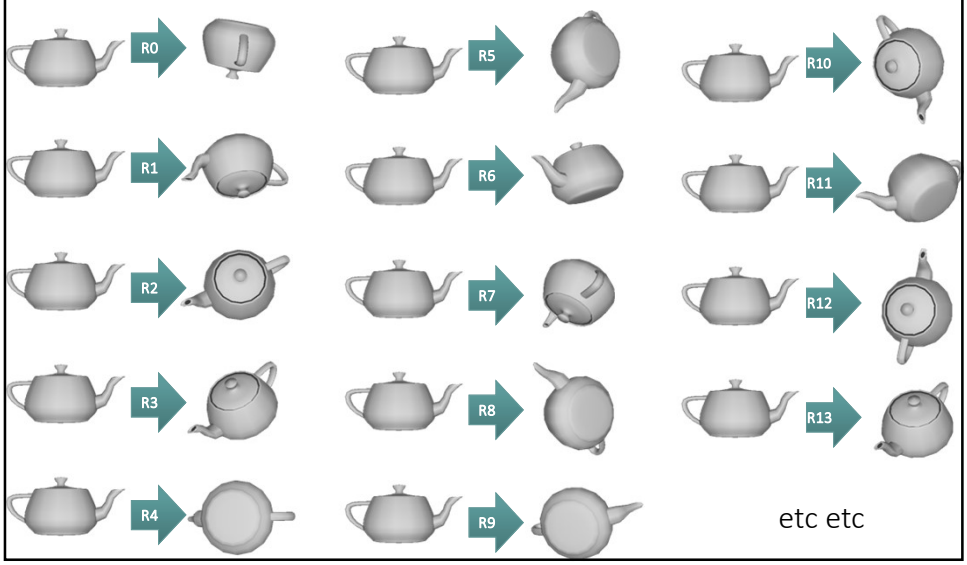
- How to go *2D-vector* ⟶ *angle*



still use atan2 ! The scale is irrelevant: $\alpha = \text{atan2}(y, x)$

19

3D rotations: how many dimensions?



20

3D rotations: how many dimensions?

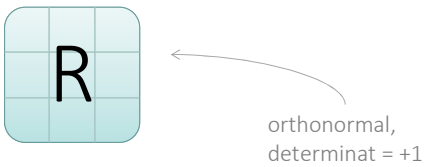
(clearly, they include the identity too)
(we just «rotate by nothing»)

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

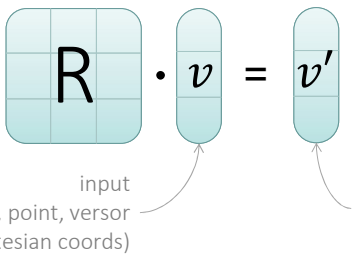
21

Solution 1: rotations are 3x3 matrices

- A rotation
  =
  a 3x3 matrix

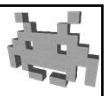


- Application
  =
  matrix / vector multiplciation



22

Rotations as 3x3 matrices



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

- They can be extended (with the identity) to get a 4x4 «pure» rotation affine matrix

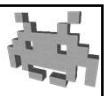
$$\begin{bmatrix} R & \begin{matrix}0\\0\\0\end{matrix} \\ \begin{matrix}0 & 0 & 0\end{matrix} & 1 \end{bmatrix}$$

Note: combined with translations, they represent rotations around any axis (aka rigid motions, aka isometries)

23

24

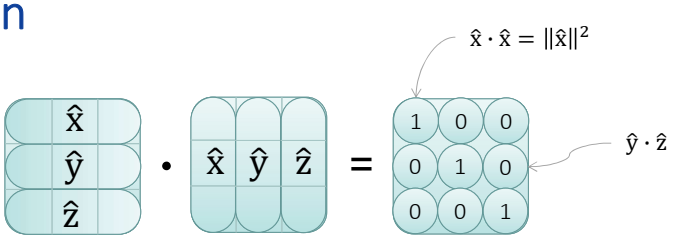
Rotations as 3x3 matrices (9 scalars)
A useful property



- 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)

25

Rotations as 3x3 matries: Inversion

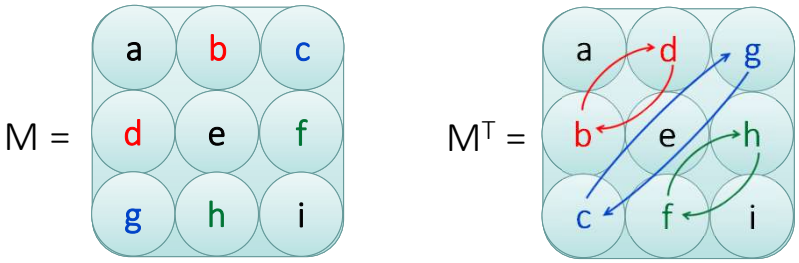


$$R^T \cdot R = I$$

- Conclusion: for a rotation matrix to transpose = to invert

26

Note: transposing a 3x3 matrix



- Just three swaps of elements!
  Super easy and quick.
  (and no numerical errors)

27

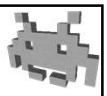
Rotations as 3x3 matrices (9 scalars): compositions

- 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.

28

Rotations as 3x3 matries



- Wasteful in RAM (9 scalars, versus a minimum of 3)
- Fast to apply (matrix-vector prod: just 9 multiplications)
- Possible to composite (matrix-matrix prod: 27 mults)
  BUT: cumulates numerical errors
  (resulting matrix no longer a rotation matrix)
  why?
- Interpolation is problematic:

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

NOT a rotation
(NOT orthonormal)

29

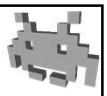
Rotations as 3x3 matrices (9 scalars)
A useful property

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)
  - (understand why!)

30

Summary, as 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 the 9 values
Notes…		Useful to extract local axes in global space (or vice-versa).

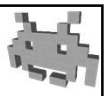
31

Representations of 3D rotations

- 3x3 matrices
- Euler angles
  - the most compact way
  - probably the most *intuitive* way to express a rotation
  - well understood by digital artists!

32

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: its "coordinates",
    three scalars identifying that rotation

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

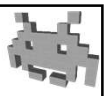
33

Euler angles: choosing the order of axis

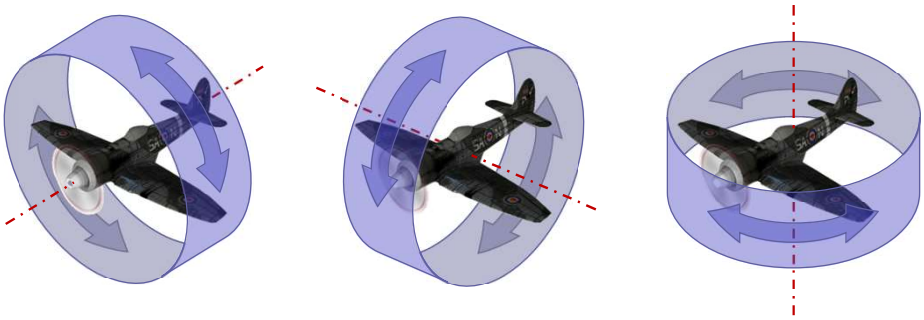
- The ordering of axis is arbitrary
  - Any can be chosen: X => Y => Z,
    Y => X => Z etc…
  - Even repeated axes: X => Y => X,
    Z => X => Z,
  - (as long as no axis is consecutively repeated twice, e.g. no X => X => Z)
- All the properties of Euler angles apply regardless to the choice made
  - But a choice must be made, and fixed once and for all in a system, to make it possible to interpret the angles
- Let's see a popular choice that makes Euler angles particularly intuitive to set (using Unity axis terminology: Z = forward, etc)
  - This confers to each angle a "nautical" semantics
  - Useful e.g. for 3D artists / non mathematicians
  - (note: speaking of intuition, "degrees" are friendlier than "radiants")

34

Rotations as Euler angles (3 scalars)



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



roll
(*rollio*)

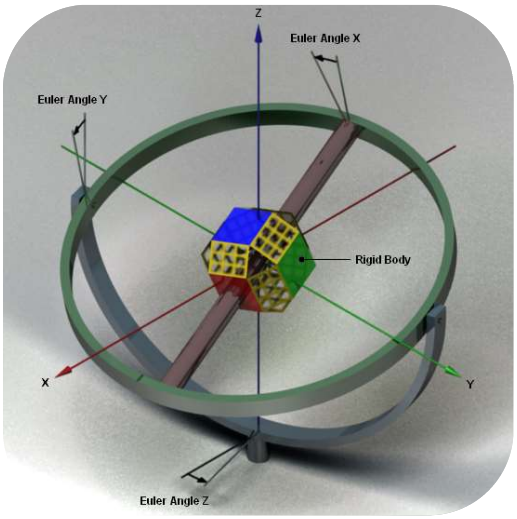
pitch
(beccheggio)

yaw
(imbardata)

35

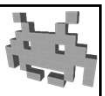
Rotations as Euler angles (3 scalars)

- A physical implementation: “three axes globe”

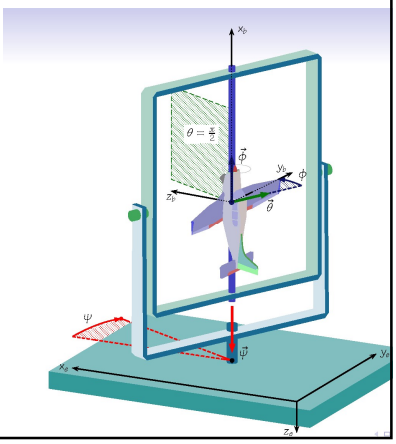


36

Rotations as Euler angles (3 scalars)

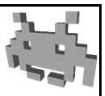


- Is it 1:1 ?
  - 1 rotation ⇔ 1 triple of angles?
- Almost
  - assuming angles are properly bounded (exercise: how?)
- Ugly exception: "GIMBAL LOCK"
  - when 1st rotation makes the axes of the next two rotations *coincide*
  - this cannot be avoided, no matter which axes are chosen



37

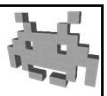
Euler angles: boundaries



- 1st and 3rd angles: **any value** (e.g., bound in 0° to 360°, or -180° to +180°)
- Middle angle: **bounded in -90° to +90°**
  - Under these boundaries, each rotation R is expressible in exactly one way
  - …barring the gimbal lock ☹ that one R can be expressed in infinite ways ☹
- again, all this is true regardless from the choice of axes
  - Try to understand it in the Z => X => Y case by looking at the globe example

38

Rotations as Euler angles (3 scalars)



- Conciseness: perfect! 3 scalars for 3 DOF
- Application : a bit work-intensive
  - apply three rotations in succession
- Interpolation : you can do that…
  - just interpolate the three angles

⚠ (remember to always "pick the *shortest path*" whenever you interpolates angles: that is, you 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?

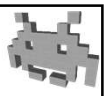
39

The "score sheet" (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 = 27 scalar mult.) Numerical errors	★☆☆☆☆
Efficient / easy to: Interpolate (with another rotation)	★☆☆☆☆ Introduces unwanted 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

40

Representations of 3D rotations



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

42

Rotations as axis & angle

Euler's rotation theorem (1776)

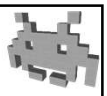
- Any 3D rotation can be expressed as *one* rotation by some angle around some axis

appropriately chosen

- Angle: a scalar
- Axis: a versor (3 scalars)
  - note: as always, we are considering "pure" rotations: the axis passes through the origin; for the more general case, combine with translations.

43

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
  - "Rodrigue's rotation formulas" (look it up if you want)
  - Note: all of the above require evaluating trigonometric function (sin, cos) – uncovenient!
- 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! 🤔

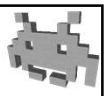
44

Rotations as axis & angle: equivalent representations

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

45

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

46

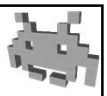
Rotations as axis and angle, variant: as axis *times* angle

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… but 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 for the identity – i.e the rotation by 0° (why?)

47

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
Efficient/easy to: Invert (produce inverse)	★★★★★	Just flip axis OR angle
Efficient/easy to: Composite (with another rotation)	★☆☆☆☆	
Efficient/easy to: 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*	

48

Representations of 3D rotations

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

next lecture!

49