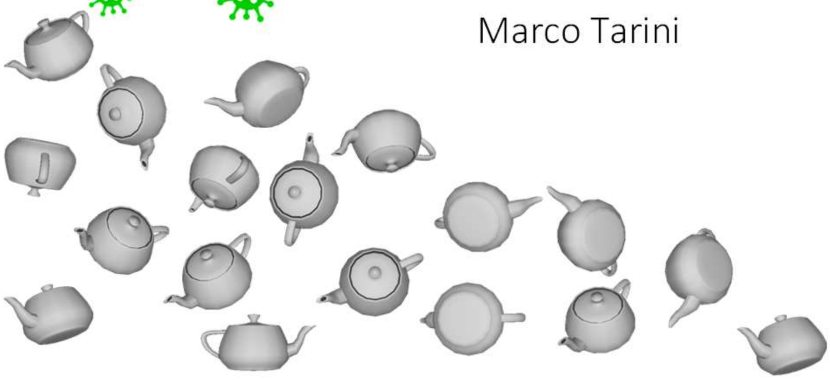
3D VideoGames

Representing Rotations

Marco Tarini



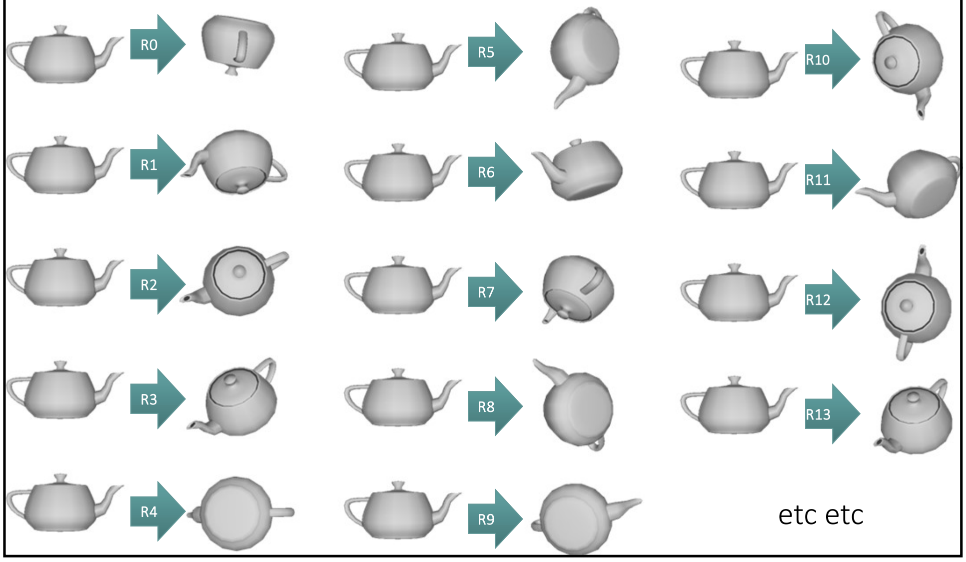
1

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. 8: Game 3D Animations ●●●
- lec. 9: Game 3D Audio ●
- lec. 10: Networking for 3D Games ●
- lec. 11: Artificial Intelligence for 3D Games ●
- lec. 12: Game 3D Rendering Techniques ●●

2

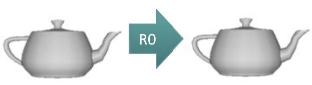
3D rotations:
how many dimension?



etc etc

4

3D rotations:
how many dimension?



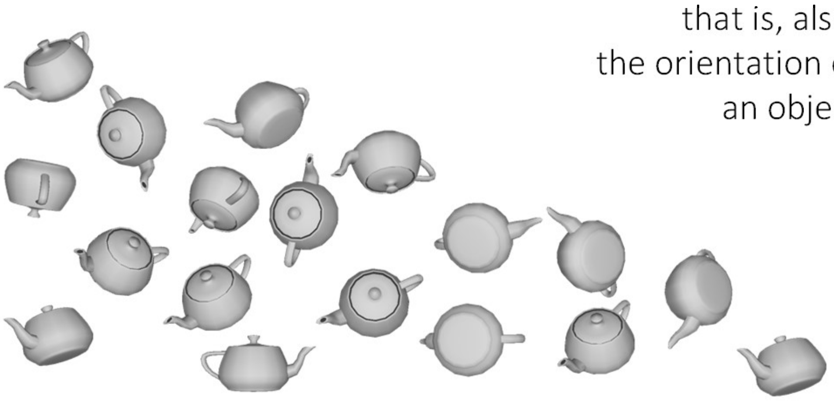
(clearly, they include the identity too)

Answer: 3 DOF (degrees of freedom).
i.e., rotations in 3D are a 3 dimensional space.

5

How to (internally) represent a 3D rotation ?

that is, also,
the orientation of
an object



6

Compare with: representing *translations* in 3D

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

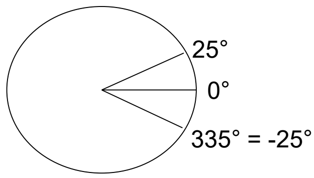
8

Compare with: representing rotations *in 2D*

- Trivial: one angle (a dimensionless scalar)
  - perfect under all criteria (exercise: verify)
  - (only choice: degrees or radiants?)

[0,360°) [0,2·Pi)

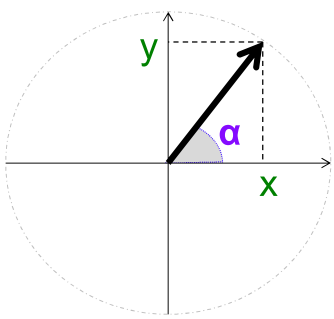
caveat: interpolation!
«pick the shortest path»
mix(25°, 335° , 0.5) = 0
(but, still easy to do)

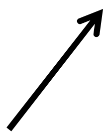


9

Compare with: representing rotations *in 2D*

- Compute *angle* ⟶ *vector*



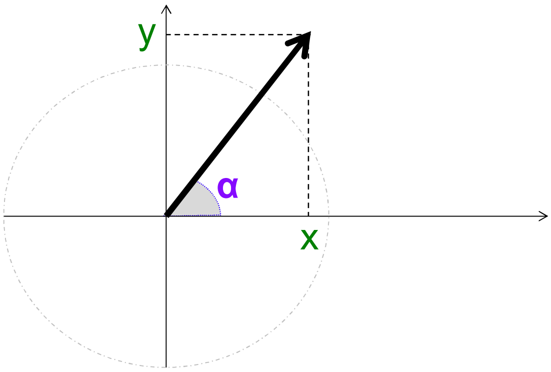


$$= \begin{pmatrix} \cos(\alpha) \\ \sin(\alpha) \end{pmatrix}$$

10

Compare with: representing rotations *in 2D*

- Compute *angle* ← *vector*



pro tip: use *atan2* in any language: α = *atan2*(y , x)

11

Rotations as 3x3 matrices (9 scalars)

- after all, rotations are linear operators
- Rot = 3x3 submatrix of a 4x4 rotation affine matrix

$$\begin{array}{ccc|c} & R & & 0 \\ & & & 0 \\ & & & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}$$

- Reminder: R is orthonormal, with det = +1

12

Rotations as 3x3 matries (9 scalars)

- 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)
- Immediate to invert (just transpose)
- Interpolate: troubles

why?

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

NOT a rotation

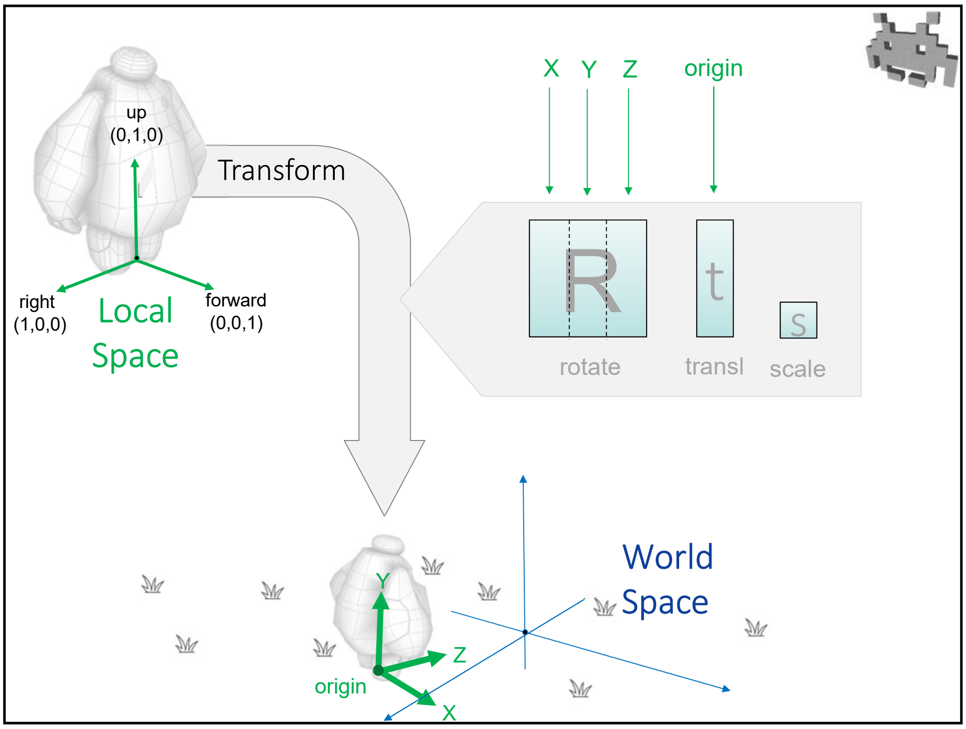
13

Rotations as 3x3 matrices (9 scalars): compositions

- Multiplying matrices cumulates 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.

14



16

Rotations as 3x3 matrices (9 scalars)

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

17

Rotations as 3x3 matrices exercise: "look-at" rotation

- Given observer position A and observed point B
  - or, directly, a look direction $v = (B - A) / || B - A ||$

  find the rotation (i.e. the orientation)
  for a character who must be looking in that direction
- Incomplete specification!
  We also need in input: a «target up-vector» u
  - the character wants to keep its up-direction as similar as possible to u, while looking toward B
  - Usually, the (world) up-vector, e.g. (in Unity) (0,1,0)
- Very useful for characters looking at something / facing toward something

19

Rotations as 3x3 matrices exercise: "look-at" rotation

- Solution:
  - find the x, y, z directions of this local character
  - note: they must be 3 reciprocally orthogonal versors
  - make them the columns of the 3x3 rotation matrix
- for example (using Unity conventional axis names):
  - $z = v$ (easy! the forward direction is exactly v!)
  - $y = u$? NO! it wouldn't be necessarily orthogonal with z
  - but, $x = u \times z \ / \ || u \times z ||$ (note the re-normalization)
    i.e. the right vector is orthogonal to both z and u
  - finally, $y = z \times x$

20

Representations of 3D rotations

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

21

Rotations as Euler angles (3 scalars)

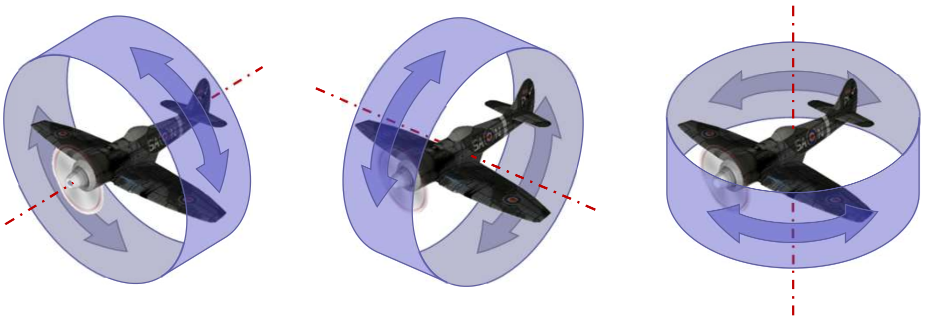
- Any 3D rotation can be expressed as:
  - a rotation around X axis (by α degrees), followed by:
  - a rotation around Y axis (by β degrees), followed by :
  - a rotation around Z axis (by γ degrees):
- Angles α β γ :
  "Euler angles" of a specific rotation
  - (therefore: its "coordinates")

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

22

Rotations as Euler angles (3 scalars)

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



roll
(*rollio*)

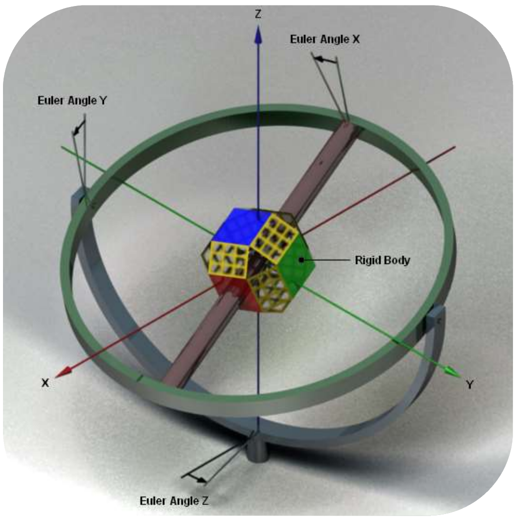
pitch
(*beccheggio*)

yaw
(*imbardata*)

24

Rotations as Euler angles (3 scalars)

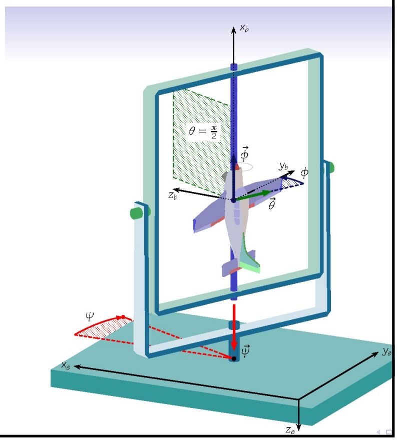
- A physical implementation: “three axes globe”



26

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



27

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 !
- Cumulate / invert: not easy nor immediate…

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

28

from: euler angles
to: 3x3 matrix

- Easy to write down!

$$M = R_z(\gamma) \cdot R_y(\beta) \cdot R_x(\alpha)$$

  - but requires several sin / cos evaluations

- What about the vice-versa?
  - a medium difficulty exercise
  - not very convenient:
    many inverse trigonometric functions

29

Comparing representations (so far)

		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)	just transpose	
Efficient / easy to	Cumulate (with another rotation)	matrix multiplication (9 dots)	
Efficient / easy to	Interpolate (with another rotation)		easy to do... unintuitive result
	Intuitive? (e.g. to manually set)	?!?	roll & yaw & pitch
	Notes...	Free extra skew + scale!	GIMBAL LOCK

30

Representations of 3D rotations

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

31

Rotations as axis & angle

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

must be 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.

32

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) or to quaternion: see later
- 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! 🤪

33

Rotations as axis & angle: equivalent representations

- Therefore: $(\ a_x\ ,\ a_y\ \ ,\ \ a_z\ ,\ \ \alpha)$
  and $(-a_x, -a_y, -a_z\ , -\alpha)$
  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 a_x, a_y, a_z
- This is always a bit inconvenient
  - Complicates interpolation ("shortest path" problems)
  - Complicates testing for equality/similarity, etc.

34

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 "right" rotation

35

Rotations as axis and angle, variant: as axis angle

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

36

Axis and angle - exercise the «from-to» rotation

- Problem: given a pair of versors *v* and *w*, (*v* = *from* and *w* = *to*) find the minimal rotation that brings *v* into *w*
  - useful problem in several contexts
- Solution:
  - the axis *a* is found as $v \times w$ (renormalizing it)
  - of the angle α , we know that the cosine is ($v \cdot w$) and the sine is $\| v \times w \|$. so α = atan2($\| v \times w \|$, $v \cdot w$)

minimal angle

e.g. AI aiming a bazooka, a ball rolling…

38

Representations of 3D rotations

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

39