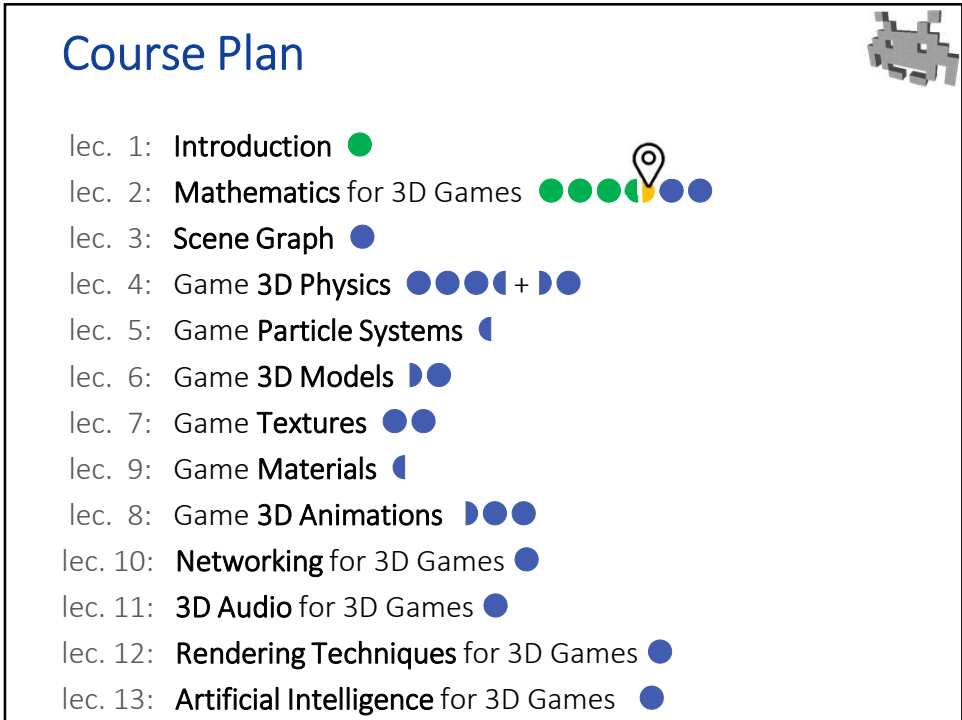


1

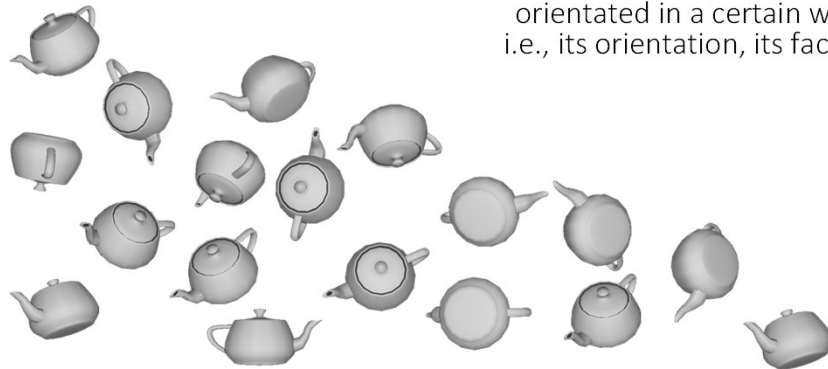


2

## 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 need 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



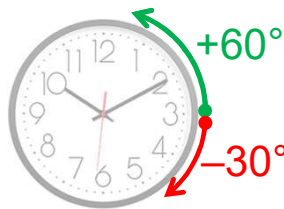
- Trivial:  
displacement vector (3 scalars)!
- perfect under all criteria above  
(exercise: verify)

6

## Preamble: representing rotations *in 2D*



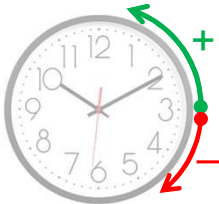
- Trivial!



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:*  $r_\alpha \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \cos(\alpha)x - \sin(\alpha)y \\ \sin(\alpha)x + \cos(\alpha)y \end{pmatrix}$ 
      - two constants (for a given  $\alpha$ )
  - Invert?
    - ✓ *Just flip the sign*
  - Cumulate?
    - ✓ *Just sum the two angles (modulo 360°)*
  - Design / manually assign?
    - ✓ *easy.* and N-E = 45°
    - E.g. 0° = east. 90° = north. 180° = west. 270° = South.

9

### Preamble: representing rotations *in 2D*

- Is it convenient to...
  - Interpolate?  $\alpha (1 - t) + \beta t$
  - Can we just...  $\text{mix}(\alpha, \beta, t)$
  - Example: mid-way between North =  $90^\circ$  and West =  $180^\circ$   
 $\text{mix}(90^\circ, 180^\circ, 0.5) = 135^\circ = \text{NW}$  ... checks out!
  - But consider this case:

Time = 1                      Time = 2                      Time = 3

10

### Preamble: representing rotations *in 2D*

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

Time = 1                      Time = 2                      Time = 3

Time = 1                      Time = 2                      Time = 3

11

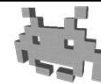
## Preamble: representing rotations *in 2D*



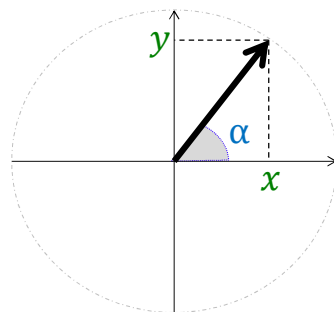
- Is it convenient to... interpolate?  $\checkmark$  Yes, but,
- Problem: angles  $\alpha$  and  $\alpha+360^\circ$  are **equivalent**  
 $\alpha \approx \alpha + 360 \approx \alpha + k 360^\circ$  (any  $k \in \mathbb{Z}$ )
- General solution:  
to interpolate between two rotations  $\alpha$  and 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

12

## Preamble: representing rotations *in 2D*



- How to go *angle*  $\rightarrow$  *2D-versor*



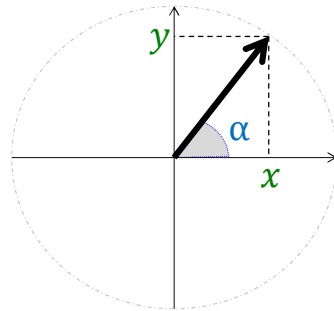
$$\vec{v} = \begin{pmatrix} \cos(\alpha) \\ \sin(\alpha) \end{pmatrix}$$

13

Preamble:  
representing rotations *in 2D*



- How to go 2D-vector  $\rightarrow$  angle



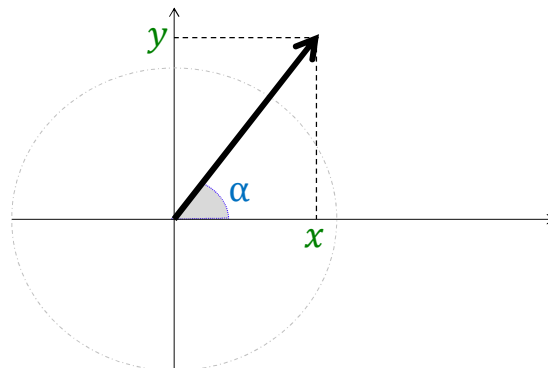
pro tip: use `atan2` in any language:  $\alpha = \text{atan2}(y, x)$

14

Preamble:  
representing rotations *in 2D*



- How to go 2D-vector  $\rightarrow$  angle



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

15

### 3D rotations: how many dimensions?

etc etc

16

### 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)”)

17



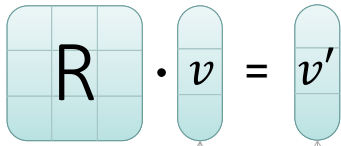
### Solution 1: rotations are 3x3 matrices

- A rotation  
= a 3x3 matrix



orthonormal,  
determinant = +1

- Application  
= matrix / vector multiplication



input  
vector, point, versor  
(cartesian coords)

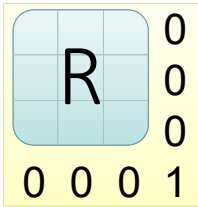
rotated  
vector, point, versor  
(cartesian coords)

18

### Rotations as 3x3 matrices

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

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



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

19

## 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.

20

## Rotations as 3x3 matrices

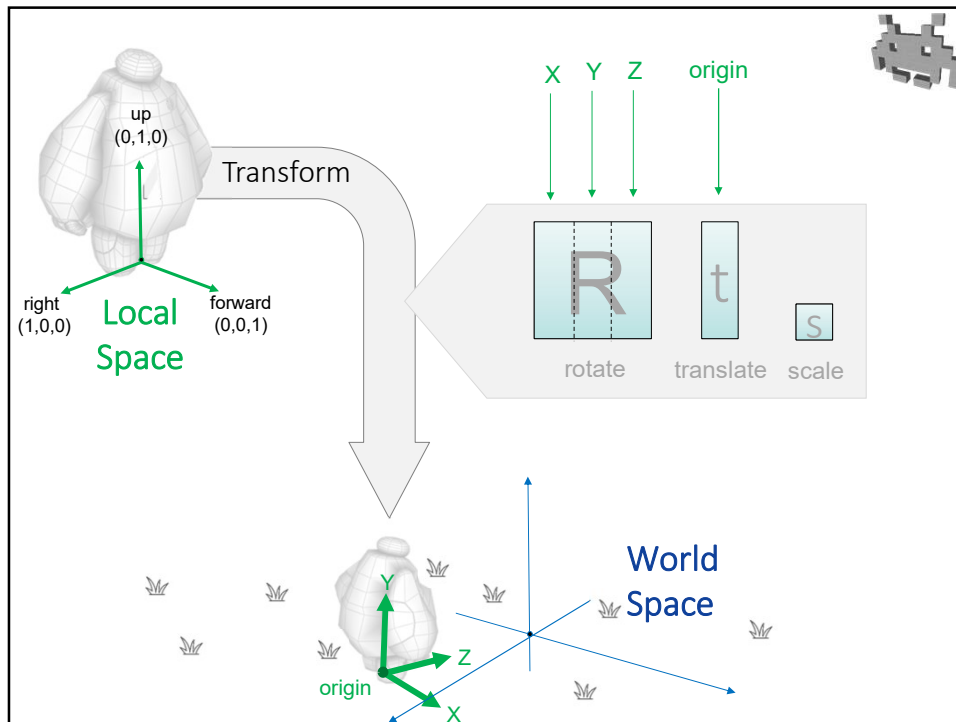


- Wasteful in RAM (9 scalars, versus a minimum of 3)
- Easy to apply (matrix-vector prod: 9 multiplications)
- Relat. quick to composite (matrix-matrix prod: 27 mults)  
BUT: cumulates numerical errors  
(resulting matrix no longer a rotation matrix) why?
- Interpolation is problematic:

$$k \begin{matrix} \boxed{R_0} \end{matrix} + (1 - k) \begin{matrix} \boxed{R_1} \end{matrix} = \begin{matrix} \boxed{M} \end{matrix}$$

NOT a rotation  
(NOT orthonormal)

21



22

## 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)

23

### Rotations as 3x3 matrices: Inversion

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

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

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

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

- For rotation matrices:  
to transpose = to invert

Just swap three pairs of elements!

24

### Note: transposing a 3x3 matrix

$$M = \begin{matrix} a & b & c \\ d & e & f \\ g & h & i \end{matrix} \quad M^T = \begin{matrix} a & d & g \\ b & e & h \\ c & f & i \end{matrix}$$

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


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

## 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 viceversa).	



27