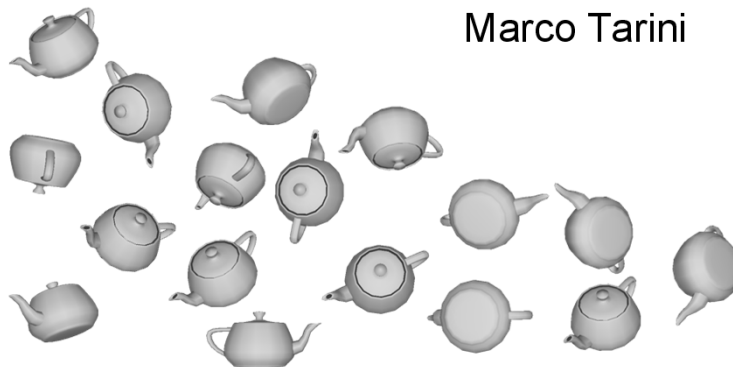


Master Game Dev 2014/2015

Game Engines

## Trasformazioni Spaziali

Marco Tarini

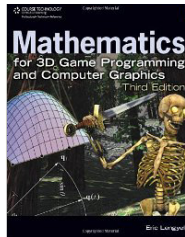


## Trasformazioni spaziali: intro

Concetto molto generale...

- Parte di molte strutture dati:
  - nello scene graph (trasf. di modellazione)
  - nelle animazioni cinematiche
  - nelle animazioni rigged (skeleton based)
- Vengono usate in molti processi
  - nel rendering, dalla GPU (vertex shader!)
    - trasformazione di posizione, di vista, di proiezione
    - trasformano geometria e normali delle mesh renderizzate
  - nella modellazione, interattivamente, dall'artista
    - x deformare un'intero oggetto o una sua sottoparte

## Punti, Vettori, Trasformazioni Spaziali



*Serve un manuale?*

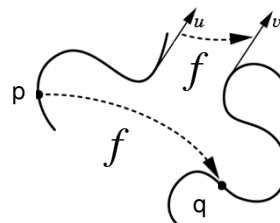
**Mathematics** for 3D Game Progr. and C.G. (3za ed)  
Eric Lengyel

Capitoli 2, 3, 4

## Trasformazioni spaziali

- Funzioni

- input: un punto  
(oppure un vettore)
- output: un punto  
(oppure un vettore)



- punti e vettori:

- rappresentati da coordinate

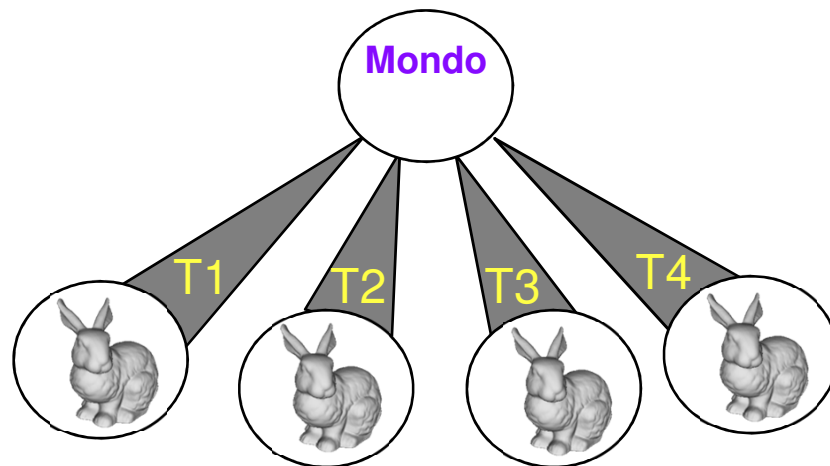
- trasformazioni:

- “cambiano le coordinate”
- nuove coords = funz( vecchie coords )

$$q = f(p)$$

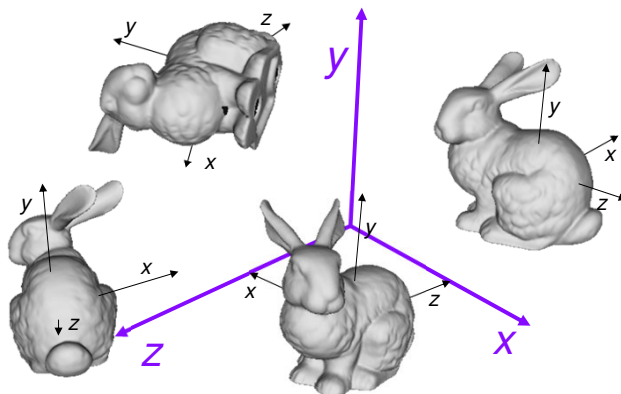
$$v = f(u)$$

## Es: scene graph

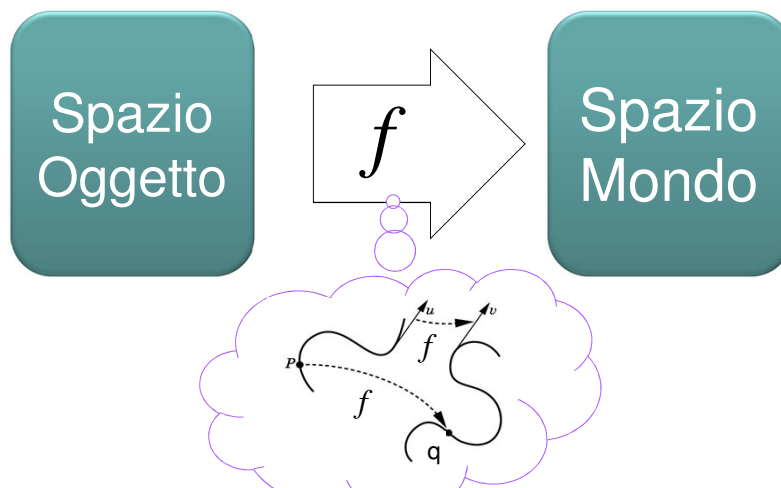


## Object Coordinates

- Dare ad ogni oggetto il suo sistema di coordinate privato: il suo **Object Space**;



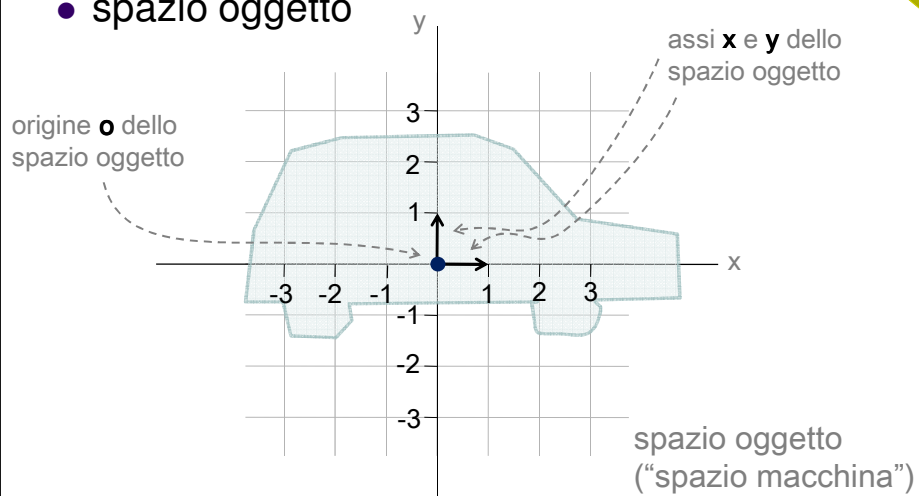
## Es: trasformazione spaziale di modellazione



## Object Space (spazio oggetto)

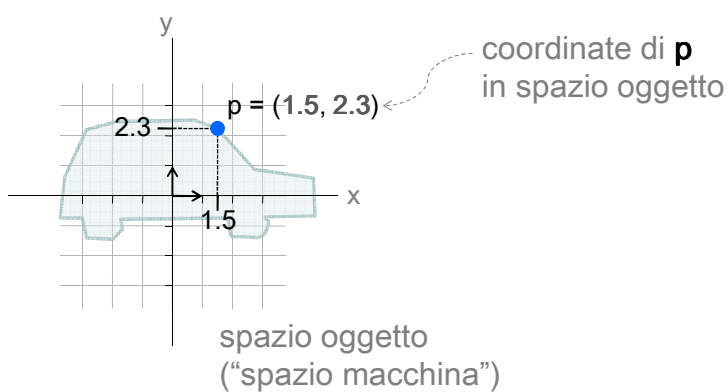
(analogo in 2D)

- spazio oggetto



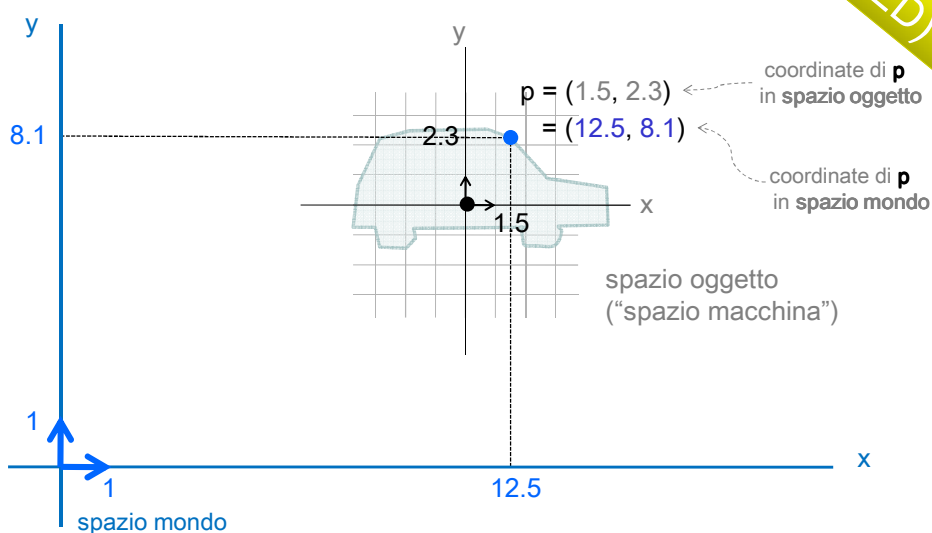
## Object Coordinates

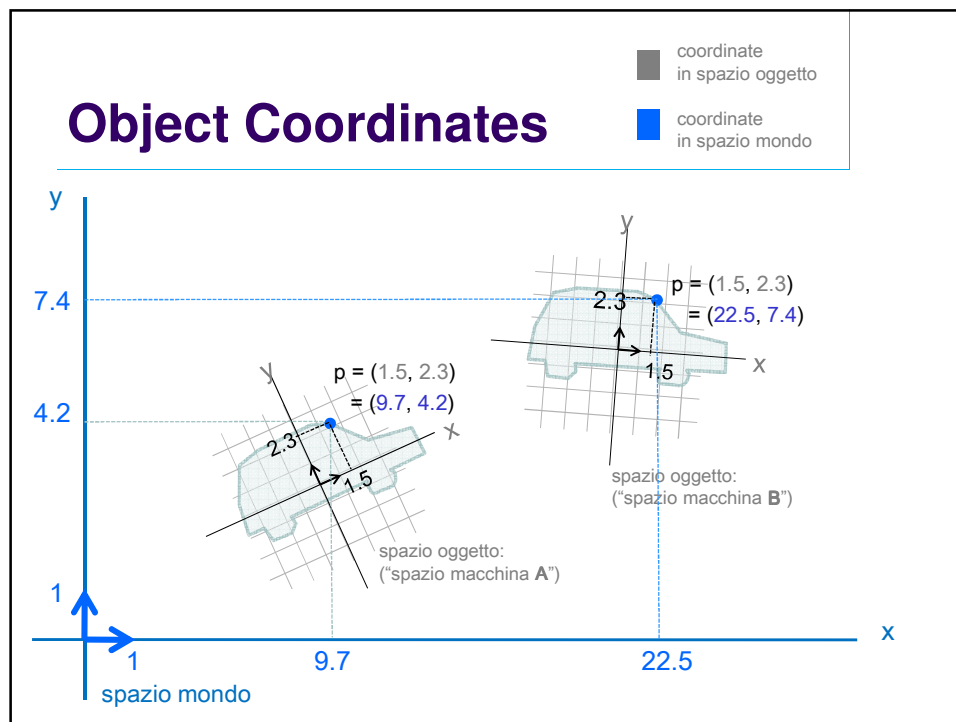
(analogo in 2D)



## Object Coordinates

(analogo in 2D)





## Classi utili di trasformazioni spaziali

- **Isometrie** (rototraslazioni)

- “Mantengono le distanze”
- Rotaz + Traslaz

Nota:  
sono chiuse rispetto  
a combinazione

- **Similitudini** (trasformaz. conformali)

- “Mantengono gli angoli”
- Rotaz + Traslaz + Scaling uniforme

- **Lineari** (trasformaz. affini)

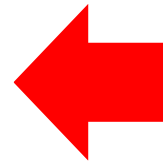
$$f(\alpha v_0 + \beta v_1) = \alpha f(v_0) + \beta f(v_1)$$



## Classi utili di trasformazioni spaziali

- **Isometrie** (rototraslazioni)
  - “Mantengono le distanze”
  - Rotaz + Traslaz
- **Similitudini** (trasformaz. *conformali*)
  - “Mantengono gli angoli”
  - Rotaz + Traslaz + Scaling uniforme

- **Lineari** (trasformaz. *affini*)
 
$$f(\alpha v_0 + \beta v_1) = \alpha f(v_0) + \beta f(v_1)$$



## Trasformazioni affini

- Definizioni equivalenti:
  - trasf esprimibili con moltiplicaz con matrice 4x4  
con ultima riga: 0,0,0,1
    - si moltiplica la matr il vett di coordinate **affini**
  - cambio di frame  
(di origine + sistema di assi catesiani)
    - da: spazio di origine a: spazio di destinazione
  - trasf lineari: cioè t.c.
 
$$f(\alpha v_0 + \beta v_1) = \alpha f(v_0) + \beta f(v_1)$$
- trasformazioni di un tetraedro in un tetraedro (in 3D)
  - (in 2D: di un triangolo in un triangolo)

## Inciso: Coordinate affini

### POSIZIONI

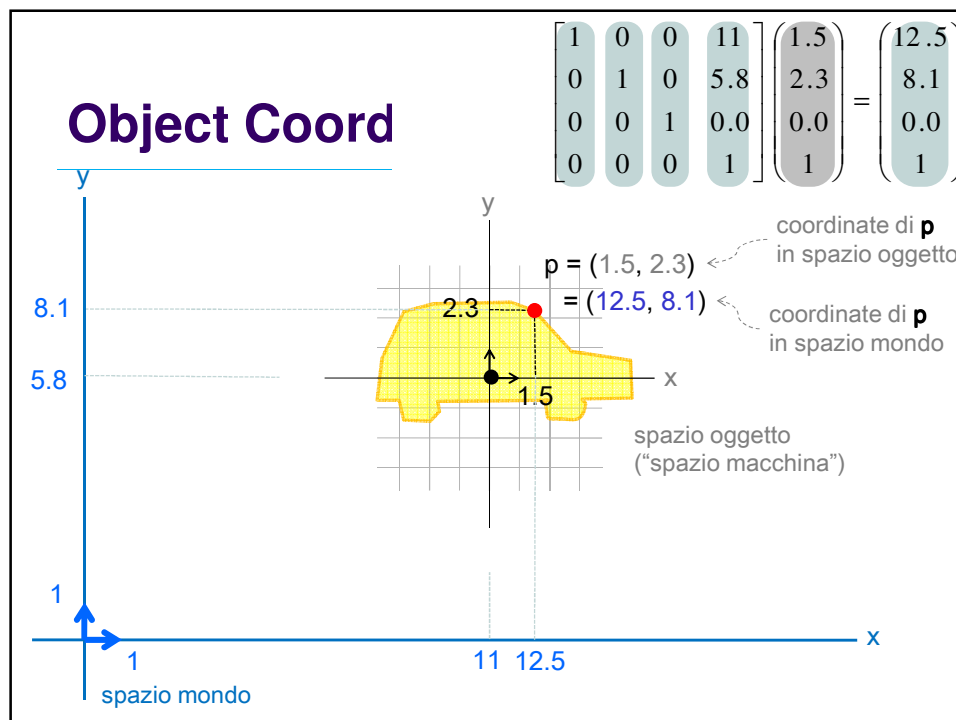
- punto  $p$  con coordinate (catesiane):  $x, y, z$

→ coordiante affini di  $p$  :  $(x, y, z, 1)$

### DIREZIONI / SPOSTAMENTI

- vettore  $v$  con coordinate (catesiane):  $x, y, z$

→ coordiante affini di  $v$  :  $(x, y, z, 0)$





## Trasformazioni affini: cosa fanno

- mantengono sempre:
  - rapporti fra volumi
  - parallelismo fra rette
- non mantengono (in generale):
  - volumi, o aree, o angoli, o lunghezze
  - o rapporti fra aree, o fra lunghezze
- non includono:
  - deformazioni prospettiche

## Trasformazioni affini: cosa fanno

(lista esaustiva!)

- Rotazioni
- Traslazioni
  - (di punti – x le direzioni non ha senso)
- Scalature
  - uniformi o non
- Skewing

(infatti includono tutte le isometrie)

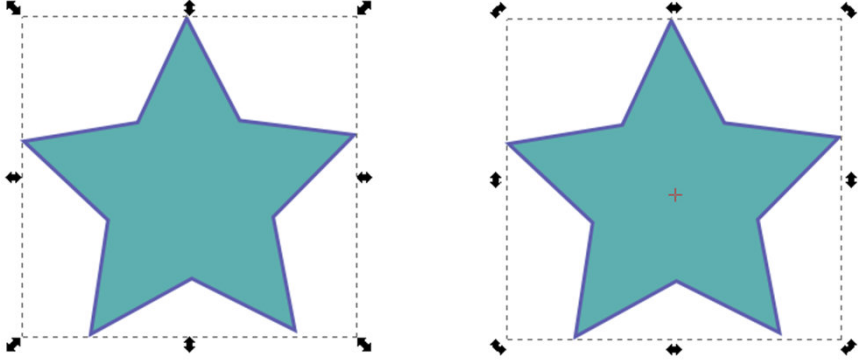


- ... e combinazioni

infatti la classe è chiusa rispetto a combinazioni...  
(basta moltiplicare le matrici!)

[DEMO]

## Trasformazioni affini (nelle GUI) : come si possono specificare



nota: questi controlli familiari (piu' la traslazione, effettuata con drag-and-drop) sono sufficienti a specificare ogni possibile trasf affine in 2D

## Trasformazioni affini (nelle GUI) : come si possono specificare

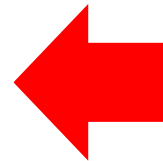
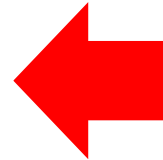
**Controllers** (concetto generale)

- aka: **handles** (maniglie)
- aka (specie in 3D): **glifi** (glyphs) o **gizmos**
- Elementi delle GUI per specificare... cose
- **Esempio:** handles per specificare le trasf affini in 2D:



## Classi utili di trasformazioni spaziali

- **Isometrie** (rototraslazioni)
  - “Mantengono le distanze”
  - **Rotaz** + **Traslaz**
- **Similitudini** (trasformaz. *conformali*)
  - “Mantengono gli angoli”
  - **Rotaz** + **Traslaz** + **Scaling uniforme**
- **Lineari** (trasformaz. *affini*)
 
$$f(\alpha v_0 + \beta v_1) = \alpha f(v_0) + \beta f(v_1)$$

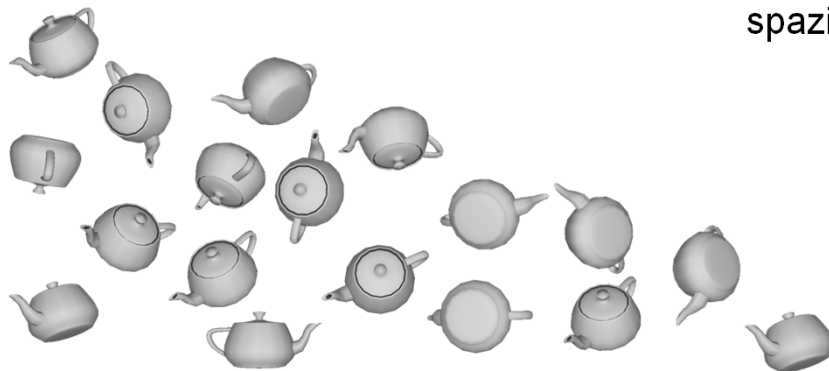


## Una classe di trasf (senza nome) spesso usata nei game engines

- Trasformazioni ottenibili combinando:
  - **Rotazioni**
  - **Traslazioni**
  - **Scalature**... *ma anche NON uniformi*
- (un altro sottoinsieme delle trasformazioni affini)
- Utile in pratica
  - facile da specificare, abb. flessibile e intuitiva
- Brutta in teoria
  - **non e' chiusa rispetto a combinazione :-O :-()**
  - (non mantiene... angoli, nulla)

## Come rappresento (internamente) una rotazione in 3D?

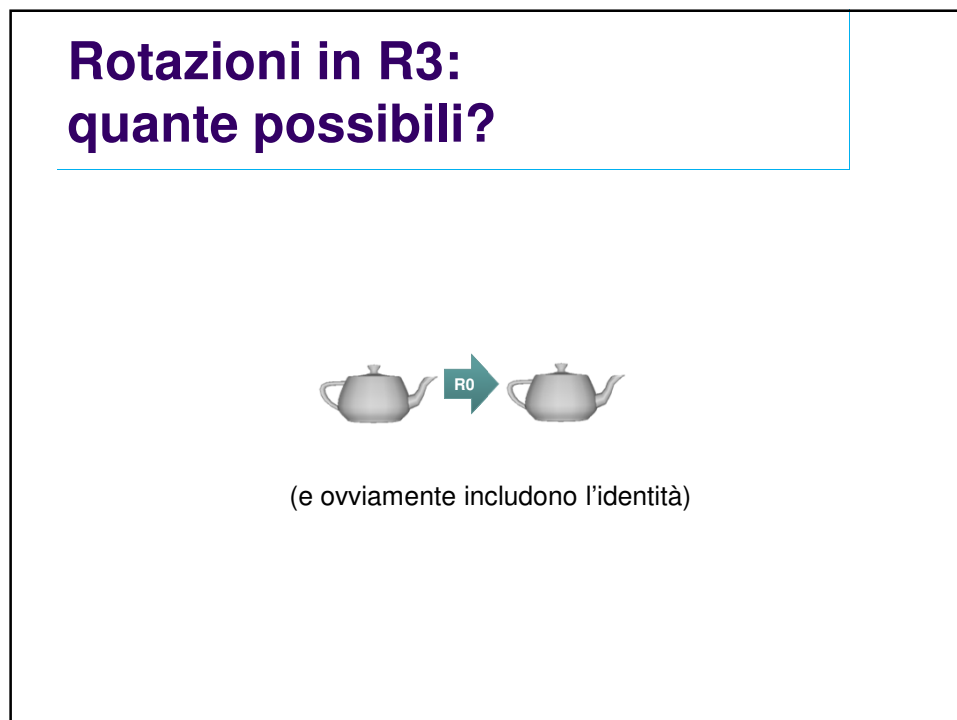
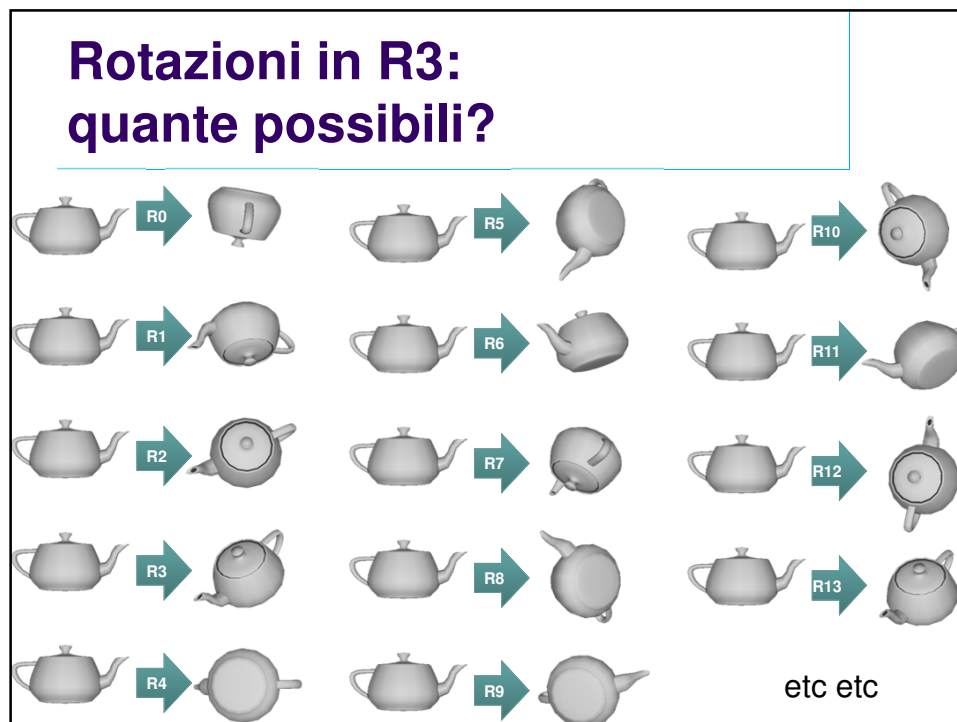
cioè anche gli  
orientamenti  
di un oggetto nello  
spazio



## Reminder

- Tutte e sole le isometrie (trasf. rigide)
  - = roto-traslazioni
  - = rotazioni (\*) + traslazioni
- Rotazioni (\*) :
  - quante possibili?
  - come rappresentarle (internamente)?

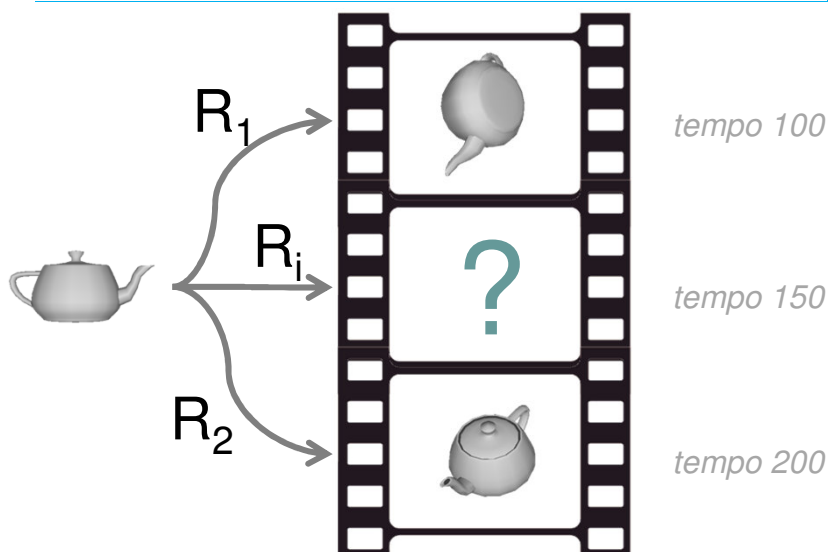
(\*) generiche = attorno ad assi passanti per l'origine



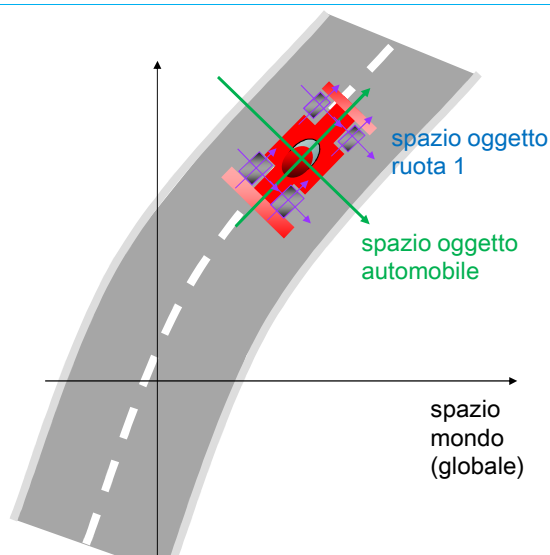
## Representazioni possibili per rotazioni: criteri

- Buone (o meno) per:
  - **compattezza**
    - quanto sono prolisse in memoria?
  - **facilità di applicazione**
    - quanto è oneroso applicare ad uno (o ventimila) punti / vettori?
  - **interpolabilità**
    - è possibile/facile trovare un'interpolazione fra N rotazioni date?
    - quanto è "buono" il risultato
  - **combinabilità**
    - è facile trovare la risultante di N rotazioni date, eseguite in successione?
  - **invertibilità**
    - è facile trovare l'inversa?
  - **intuitività**
    - quanto è difficile spiegarla ai modellatori / editori di scene / etc

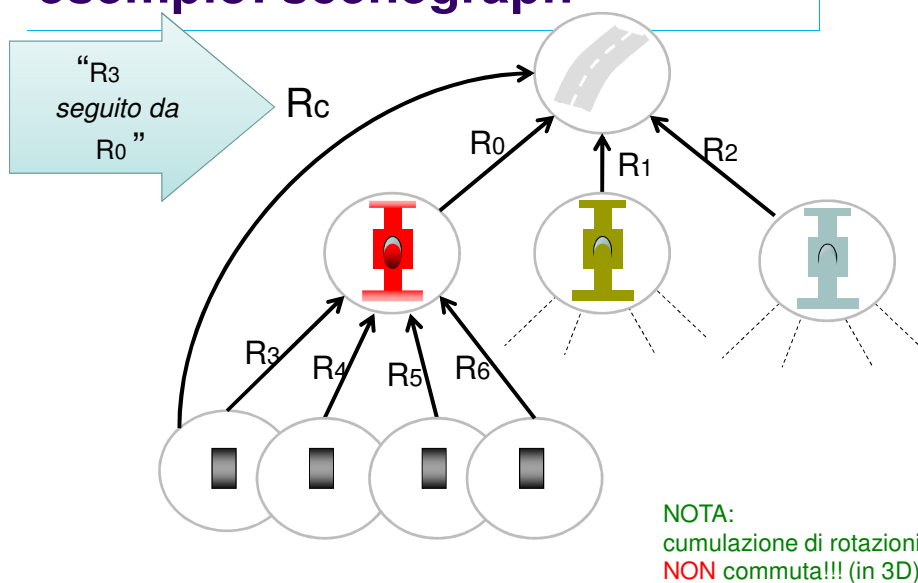
## Perché è utile *interpolare* rotazioni: esempio: animazioni



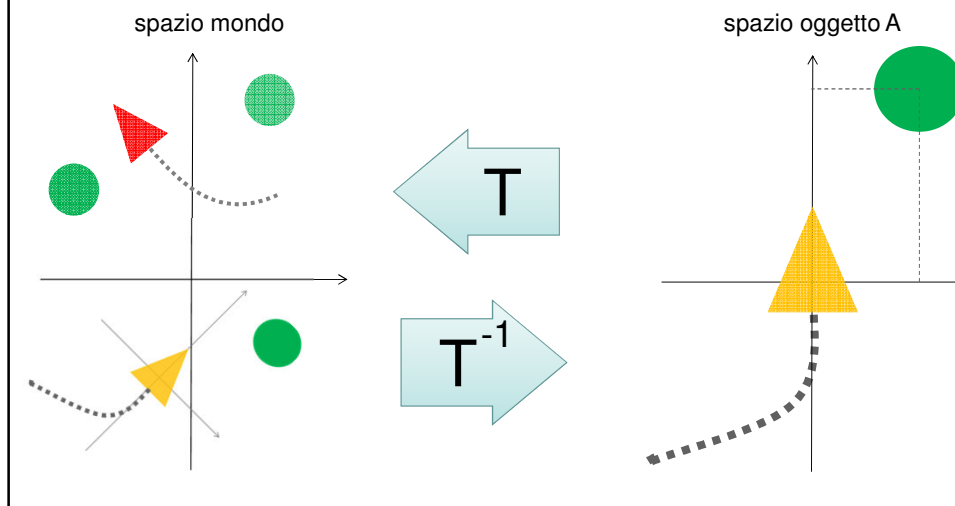
## Perché è utile *cumulare* rotazioni: esempio: scenegraph



## Perché è utile *cumulare* rotazioni: esempio: scenegraph



## Perché è utile *invertire* rotazioni: switch between spaces



## Per paragone: rappresentazione delle *traslazioni*

- Banale:  
vettore di displacement (tre float)!
  - perfetta secondo tutti i criteri (verificare!)

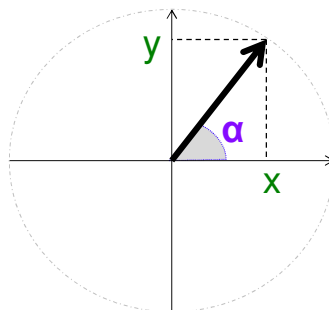


**Per paragone:**  
reppr. delle *rotazioni in 2D*

- Banale:  
un angolo (un float)
  - perfetta secondo tutti i criteri (verificare per es.!)
    - (unica scelta: degrees or radians?)  
 $[0,360)$        $[0,2\cdot\text{Pi})$

**Per paragone:**  
reppr. delle *rotazioni in 2D*

- Passaggio **angolo** → **vettore**

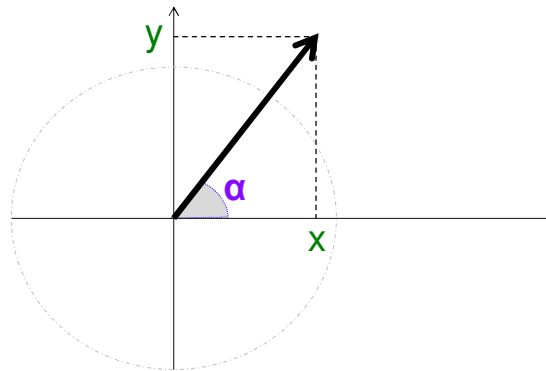


(digressione)

## Per paragone: reppr. delle *rotazioni in 2D*

(digressione)

- Passaggio **angolo** ← **vettore**



pro tip: use `atan2`

## Representazioni per rotazioni (in 3D)

- Molte possibili,  
vanno più o meno bene coi vari criteri
- Tutte molto diffuse ed usate
- Modi per passare da una rappr. all'altra?

## Reppresentazioni principali delle rotazioni

- Matrici 3x3
  - quello tipic. usato durante il rendering (nella GPU)

## Modo 1: matrice 3x3 (9 floats)

- dopotutto, una rot è un es di trasf affine
- (sottomatrice 3x3 della matrice di trasf 4x4)

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

- come sappiamo, R ortonormale con  $\det = 1$

## Modo 1: matrice 3x3 (9 floats)

- Prolissa (9 numeri invece di 3)
- Facile da applicare (molt matrice-vettore)
  - come sappiamo, cumulabile con qualunque altra trasf. affine
- Abb. facile da cumulare (molt matrice-matrice)
- Facilissima da invertire (trasposiz matrice)
- Problematica da interpolare:

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

perché?

in genere NON di rotazione (non ortonormale)

## Modo 1: matrice 3x3 (9 floats)

- Molto efficiente da applicare
  - prodotti e somme, no trigonometria
- cumulabile con tutte le altre trasf affini
  - come una sola matrice 4x4
- ➔ metodo tipic. adottato per memorizzare ed eseguire trasformazioni spaziali nel GPU-based rendering! (nel vertex shader)

## Reppresentazioni principali delle rotazioni

- Matrici 3x3
- Angoli di Eulero
  - il più intuitivo dei metodi per specificare a mano una rot
  - molto usato nei software di modellazione

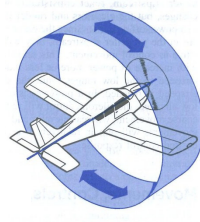
## Modo 2: angoli di eulero (3 floats)

- Qualunque rotazione (\*) può essere espressa come:
  - rotazione lungo asse X (di  $\alpha$  gradi), seguita da:
  - rotazione lungo asse Y (di  $\beta$  gradi), seguita da:
  - rotazione lungo asse Z (di  $\gamma$  gradi):
- Angoli  $\alpha \beta \gamma$  :  
“angoli di Eulero” di quella rotazione
  - (quindi: le “coordinate” di quella rotaz)

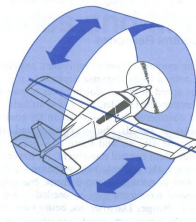
ordine (X-Y-Z)  
arbitrariamente  
scelto,  
(ma 1 volta x  
tutte)

## Modo 2: angoli di eulero (3 floats)

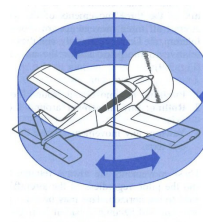
- In linguaggio nautico / aeronautico:  
angoli di “rollio, beccheggio, imbardata”



rollio  
(*roll*)



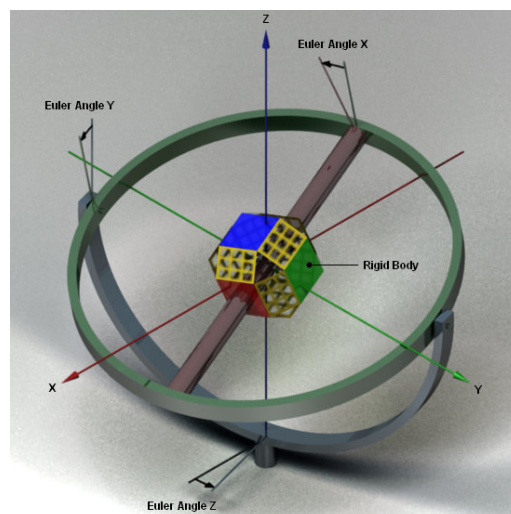
beccheggio  
(*pitch*)



imbardata  
(*yaw*)

## Modo 2: angoli di eulero (3 floats)

- Implementaz.  
fisica:  
“mappamondo  
a tre assi”



## Modo 2: angoli di eulero

### (3 floats)

- Compattezza: perfect!
- Da applicare: un po' faticoso
  - (tre rotazioni in fila)
- Da interpolare: possibile...
  - interpolaz dei tre angoli
  - (occhio ad interpolare angoli: ricordarsi equivalenza angoli:  $\alpha = \alpha + 360 (k)$  )  
...ma risultati non sempre intuitivi)
- Da cumulare e invertire: problematico...

perché sommare e invertire gli angoli non funziona?

## da: angoli di eulero

### a: matrice 3x3

- Facile!

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

- Il viceversa?
  - (solo a suon di conti e funz trigon. inverse)


## Reppresentazioni principali delle rotazioni

- Matrici 3x3
- Angoli di Eulero
- Asse + angolo

## Modo 3: asse e angolo

- Qualunque rotazione (\*) data può essere espressa come:
  - una rotazione (di un **angolo** ) attorno ad un **asse**
- **Angolo**: uno scalare (1 float)
- **Asse**: un vettore unitario (3 float)
  - (asse passante per l'origine)

opportunamente  
scelti





## Modo 3: asse e angolo

- Compattezza: buono
- Interpolazione: ottimo!
  - interpolare asse, interpolare angolo (nb: rinormalizzare asse!)
  - funziona alla grande
- Applicare: maluccio ☹️
  - modo migliore: passare a matrice 3x3 (come?) (o a quaternione)
- Cumulare: non del tutto immediato
- Invertire: facilissimo
  - (invertire angolo *oppure* asse)

## Modo 3: asse e angolo: variante

- asse:  $\mathbf{v}$  (vett normale,  $|\mathbf{v}| = 1$ )
- angolo:  $\alpha$  (scalare)
- rappresentarli internamente come 1 solo vett:  $\mathbf{v}'$  (3 float in tutto)
  - $\mathbf{v}' = \alpha \mathbf{v}$ 
    - angolo  $\alpha = |\mathbf{v}'|$
    - asse  $\mathbf{v} = \mathbf{v}' / |\mathbf{v}'|$
    - (nota: se angolo = 0, asse si perde... infatti non conta)
- Più coinciso, ma per il resto equivalente
- (molto comune es. in fisica)

**da: asse e angolo**  
**a: matrice 3x3**

- esercizio!

**Reppresentazioni principali  
delle rotazioni**

- Matrici 3x3
- Angoli di Eulero
- Asse + angolo
- Quaternioni

## Ripasso: numeri complessi

Assunzione  
"fantasiosa":

c'è un  $i$  t.c.

$$i^2 = -1$$

- **Conseguenze:**

- "Num complesso":  $(a + b i)$ 
  - *interpretaz geom*: punti 2D  $(a, b)$
- Moltiplicaz fra complessi: ...
  - *interpretaz geom*: ...
- Dunque:
  - moltiplicare per numero complesso  $\rightarrow$  ruotare in 2D (a norma 1) (attorno all'origine)
  - numeri complessi  $\rightarrow$  rappresentaz rotazioni in 2D (a norma 1)

## Passare ai quaternioni

Assunzione  
"fantasiosa":

ci sono  $i, j, k$  t.c.

x	i	j	k
i	-1	-k	-j
j	k	-1	-i
k	j	i	-1

cioè:

$$\begin{aligned}
 i^2 &= j^2 = k^2 = -1 \\
 ij &= k & ji &= -k \\
 jk &= i & kj &= -i \\
 ki &= j & ik &= -j
 \end{aligned}$$

- **Conseguenze:**

- "Quaternione":  $(a i + b j + c k + d)$ 
  - *interpr. geom*: punti 3D  $(a, b, c)$ , quando  $d=0$
- Moltiplicare due quat: ...
- Invertire un quat: ...
- Coniungere due quat  $q$  e  $p$  (fare  $\bar{q} p q$ ): ...
  - *interpretaz geom*: ruotare  $p$  con la rotaz def da  $q$
- Dunque:
  - coniugare con un quat (con norma 1)  $\rightarrow$  ruotare in 3D (asse pass. x ori)
  - quaternioni (a norma 1)  $\rightarrow$  rappresentaz rotazioni in 3D

## Modo 4: “quaternioni” (4 float)

- Compattezza: buono
- Cumulare: facilissimo ;)
- Invertire: facilissimo ;)
- Interpolare: facilissimo ;) ...e best results!
- Applicazione diretta: facilissimo ;)
- (ma, per cumulare con altre trasformazioni, meglio passare a forma matriciale cmq)

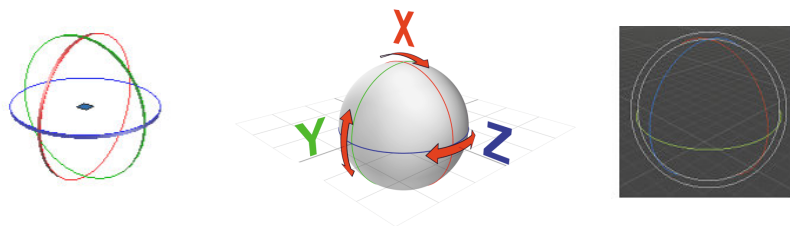
## Modo 4: “quaternioni” (cenni)

- analogo (in 4D) dei numeri complessi (in 2D)
- struttura simile ad asse + angolo:
 
$$q = (asse_x, asse_y, asse_z, \cos(angolo / 2))$$

$$|q| = 1$$
- teoria molto elegante e solida
- ecco un altro “vec4” molto utile!
- storia:
  - roba di mezz’800!
  - cross e dot products emergono dalla loro teoria (!)
  - nati proprio per lo scopo (rappresentare rotazioni)

## GUI: come *specifico* le rotazioni in 3D?

- Metodo ricorrente: **rotation gizmo**
  - (a volte: «**arcball**» o «**trackball**»)
  - tre handles per controllare i tre angoli di Eulero
  - o “free”, drag-n-drop “intuitivo” (metafora trackball)



convenzione: Rosso = X Verde = Y Blu = Z

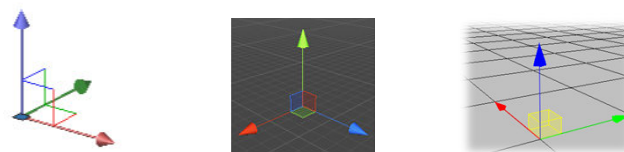
## GUI: come *specifico* le traslazioni in 3D?

- “free”: drag-and-drop 2D in **spazio immagine**,
  - mantenendo distanza da osservatore
 oppure
- **translation gizmo**
  - orientato nello **spazio mondo**, oppure
  - orientato nello **spazio oggetto**
  - **handles** per **trasl** lungo **assi** e/o **piani**

es: x spostare un oggetto “qui, in questo punto dello schermo”

es: x spostare un oggetto “in basso”

es: x spostare un oggetto lungo il “suo” basso

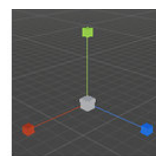
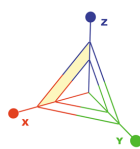
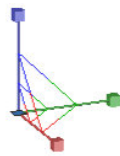


convenzione: Rosso = X Verde = Y Blu = Z

## GUI: come *specifico* le scalature in 3D?

- **scaling gizmo**

- (tipic. orientato in **spazio oggetto**)
- tre **handles** per le **scalature anisotropiche**  
+ un **handle** centrale per **scalature uniformi**



convenzione: Rosso = X Verde = Y Blu = Z

## Rotazioni in unity



- Nella GUI del game tool:
  - Euler Angles
- Internamente:
  - Quaternions
  - Dunque, negli scripts: class quaternion


## Rotazioni in OpenGL

- Nelle API «old school»: `glRotate3f`
  - Asse e angolo
- Internamente:
  - Matrici
  - (come tutte le altre trasformazioni spaziali)

## Reppresentare rotazioni

- Matrici 3x3
- Angoli di Eulero
- Asse + angolo
- Quaternioni

## Reppresentare ~~rotazioni~~ roto-**traslazioni**

- Matrici 3x3
  - Angoli di Eulero
  - Asse + angolo
  - Quaternioni
- 
- + Traslazione  
(displ. vec)
- Dual Quaternioni