Costruzione transformazione di vista da parametri estrinseci

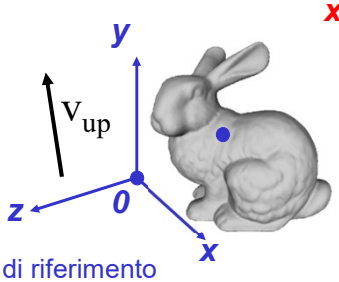
Input:

1) camera position: p_{eye}
2) target position: p_{target}
3) vettore di alto: v_{up}

Output:

Matrice di Trasformazione

world space ⟶ *view space*



sistema di riferimento mondo *(world frame)*

sistema di riferimento della camera *(view space)*

15

Costruzione transformazione di vista da parametri estrinseci

Input:

1) camera position: p_{eye}
2) target position: p_{target}
3) vettore di alto: v_{up}

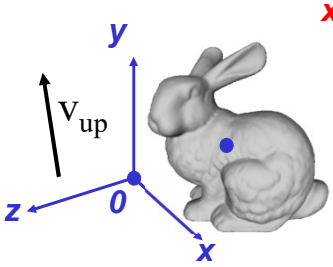
PSEUDOCODICE

```
vec3 oe = p_eye;
vec3 xe, ye, ze;

ze = p_target - p_eye;
ze = -ze;
ze = normlize( ze );

xe = cross( vup , ze );
xe = normalize( xe );

ye = cross(ze, xa);
```



x_e	y_e	z_e	o_e
0	0	0	1

matrice che va da spazio vista a spazio mondo.
E' l'inversa di quella che volevamo.
Ergo, va invertita.

16

Esempio tipico di costruzione transformazione di vista

Origine e assi del sistema vista espressi nelle coord del sistema mondo

Per def, l'asse zeta della vista va verso l'osservatore

Deve essere reso unitario

Operazioni dette "completamento di base"

Normalizzaz non necessaria (perché?)

Osservaz:
è possibile fallire? le due normalizzazioni possono essere divisioni per by 0? Quando succede?

```
vec3 oe = p_eye;
vec3 xe, ye, ze;

ze = p_eye - p_target;
ze = normlize( ze );

xe = cross( vup , ze );
xe = normalize( xe );

ye = cross(ze, xa);
```

x_e	y_e	z_e	o_e
0	0	0	1

matrice che va da **spazio vista** a **spazio mondo**.
E' l'inversa di quella che cerchiamo.
Ergo, va invertita.

17

Inversione di una rotazione (ripetiamoci)

$$\begin{bmatrix} R & 0 \\ 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} R^T & 0 \\ 0 & 1 \end{bmatrix}$$

Matr. di rotazione qualsiasi
(con asse passante per origine)

dove:
R rotazione 3x3,
cioè ortonormale a det 1,
cioè $v_0\ v_1\ v_2$:
- unitari
- ortogonali a due a due

Nota:

Se $R = [v_0\ v_1\ v_2]$

allora $R^T = \begin{bmatrix} v_0 \\ v_1 \\ v_2 \end{bmatrix}$

19

Inversione di una traslazione (ripetiamoci)

$$\begin{bmatrix} I & t \\ 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} I & -t \\ 0 & 1 \end{bmatrix}$$

matrice 4x4 di traslazione

matrice 4x4 di traslazione del vettore opposto

21

Roto-traslazione (cioè isometria) (cioè trasformazione rigida)

$$\begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} I & t \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} R & 0 \\ 0 & 1 \end{bmatrix}$$

roto-traslazione (4x4)
(o *isometria*)

traslazione ← rotazione
(asse passante per origine)

Verificare eseguendo
il prodotto «riga per colonna» fra i blocchi

22

Inversione di una roto-traslazione

nb: $(A\,B)^{-1} = B^{-1}\,A^{-1}$

$$\begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}^{-1} = \left(\begin{bmatrix} I & t \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} R & 0 \\ 0 & 1 \end{bmatrix} \right)^{-1} =$$

$$= \begin{bmatrix} R^T & 0 \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} I & -t \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} R^T & \\ 0 & 1 \end{bmatrix}$$

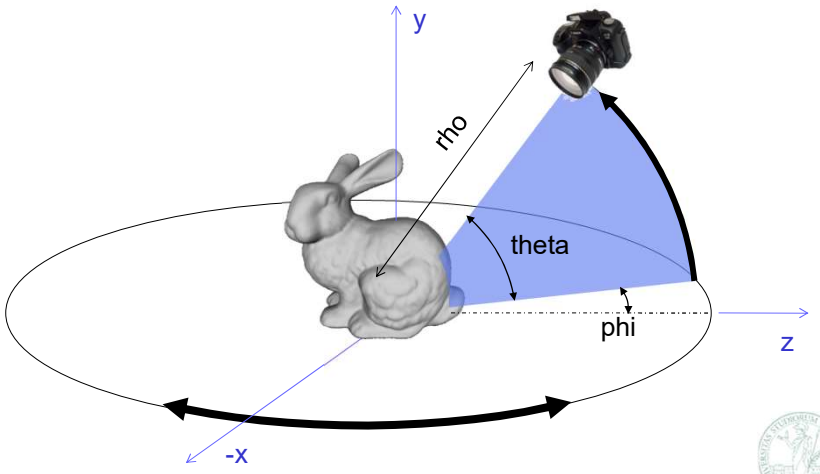
$$R^T * -t$$

Inverto ogni passaggio
ma anche il loro ordine

23

Oppure: possiamo definire la *matrice di vista* combinando rotazioni e traslazioni

Una semplice "Trackball":



24

Trackball (come dispositivo fisico di input)

25

Trackball (come elemento di un'interfaccia grafica)

- ✓ Interfaccia usata per consentire all'utente di selezionare posizione e orientamento di un oggetto
  - ⇒ attraverso un pointer device (come mouse o touch screen)
  - ⇒ quando l'oggetto è la camera: la trackball determina la matrice di vista
- ✓ Una semplice trackball: posizione & orientamento della camera controllata da soli tre parametri:
  - ⇒ due angoli (`phi` e `theta`) e una distanza (`rho`)
  - ⇒ es mappati su assi X Y mouse + mousewheel
  - ⇒ detta «orbit control»
- ✓ Utile per visualizzare un piccolo oggetto 3D
  - ⇒ permettere alla camera di orbitare intorno all'origine
  - ⇒ controllo semplice e intuitivo

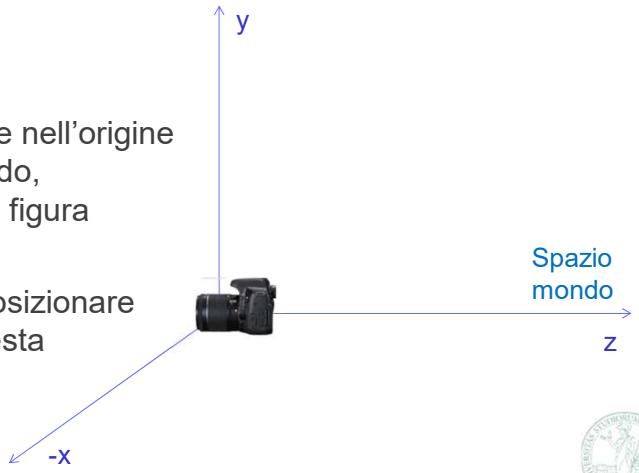
26

Passo 0

Se la matrice di vista è l'identità, allora lo spazio vista coincide con lo spazio mondo

La camera è dunque nell'origine dello spazio mondo, orientate come in figura

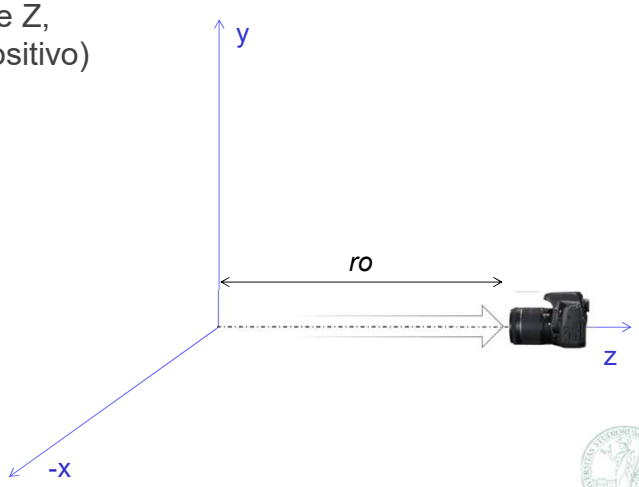
Immaginiamo di riposizionare la camera da questa situazione.



27

Passo 1

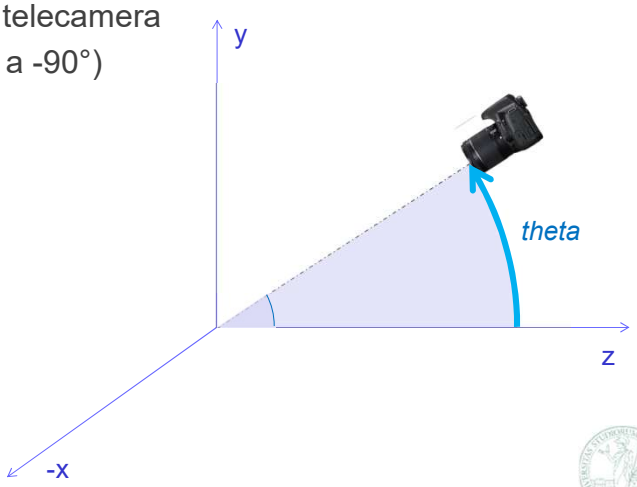
Spostare la macchina all'indietro di *ro* unità all'indietro (dunque, sull'asse Z, con *ro* numero positivo)



28

Passo 2

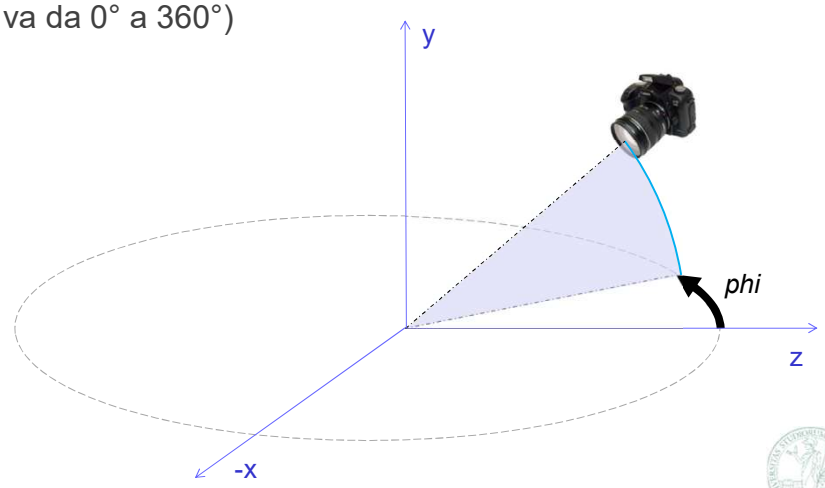
Ruotare di theta gradi
attorno all'asse delle X,
quindi alzando la telecamera
(theta va da +90° a -90°)



29

Passo 3

Ruotare la camera di phi
gradi attorno all'asse delle Y
(phi va da 0° a 360°)



30

La matrice di Vista controllata dalla Trackball

- ✓ Matrice che sposta la telecamera nel posto voluto:

$$\mathbf{R}_{\mathrm{Y}}(phi) \cdot \mathbf{R}_{\mathrm{X}}(theta) \cdot \mathbf{T}(0,0,rho)$$

(matrice di) **rotazione** attorno all'asse delle Y di angolo phi

seguita da ← ← ←

(matrice di) **rotazione** attorno all'asse delle X di angolo theta

seguita da ← ← ←

(matrice di) **traslazione** di ro sull'asse delle z

- ✓ Questa è la matrice di modellazione dell'oggetto camera
  - ⇒ va da spazio camera a spazio mondo!
- ✓ La **matrice di vista** è dunque la sua *inversa*, e cioè:

$$\mathbf{T}(0,0,-rho) \cdot \mathbf{R}_{\mathrm{X}}(-theta) \cdot \mathbf{R}_{\mathrm{Y}}(-phi)$$

Le inverse, in ordine inverso

35

Nel nostro progetto (vedi `cgLab04.html` e `.js`)

- ✓ La trackball come oggetto JavaScript (o JSON):

```
var unaTrackball = {
      rho: 3,
      phi: 15,    // in gradi!
      theta: -35, // in gradi!
}
```

- ✓ Per costruire la matrice di vista (in un apposito metodo):

```
var V = new THREE.Matrix4();
V.identity();
V.multiply( matriceDiTraslazione( 0, 0, -this.rho ) );
V.multiply( matriceDiRotazioneX( -this.theta ) );
V.multiply( matriceDiRotazioneY( -this.phi ) );
```

(vedi progetto nella pagina del corso per altri dettagli)

37