Microprogetto 1 – piano
File sul sito: `cgLab01.html`

Propositi:

1. Settiamo la matrice di modellazione del nostro quad "a mano"
2. Eseguiamo una mini animazione per far ruotare il nostro quad

30

Esercizio: costruiamo manualmente una metrice di traslazione

La classe `Matrix4` di `three.js` è preposta a rappresentare **matrici 4x4 (le nostre atrici di trasformazione)**

Come esercizio, costruiamo manualmente una matrice di traslazione:
(avremmo potuto usare funzioni esistenti di `three.js`)

```
function matriceDiTraslazione( dx, dy, dz ) {
	var M = new THREE.Matrix4();
	M.set(
		1,0,0,dx, // 1ma RIGA della matrice
		0,1,0,dy, // 2da RIGA della matrice
		0,0,1,dz, // 3za RIGA della matrice
		0,0,0,1   // 4ta RIGA della matrice
	);
	return M;
}
```

31

Esercizio: costruiamo manualmente una metrice di rotazione

Come esercizio, costruiamo manualmente anche una matrice di rotazione attorno all'asse delle Z in senso anitorario
(di nuovo, avremmo potuto usare funzioni esistenti di `three.js`)

```
function matriceDiRotazioneZ( angoloInGradi ) {
        var M = new THREE.Matrix4();
        var angoloInRadianti = angoloInGradi/180*Math.PI;
        var c = Math.cos( angoloInRadianti );
        var s = Math.sin( angoloInRadianti );
        M.set(
                c,-s,0,0, // 1ma RIGA della matrice
               +s, c,0,0, // 2da RIGA della matrice
                0, 0,1,0, // 3za RIGA della matrice
                0, 0,0,1  // 4ta RIGA della matrice
        );
        return M;
}
```

Similmente, possiamo fare le matrici di rotazioni attorno agli altri due assi

32

Campo matrix della mesh: assegnamento automatico da parte di three.js

- ✓ Il campo `matrix` della mesh contiene la model-matrix
  - ⇒ altri campi utili presenti: matrice `modelViewMatrix` che viene automaticamente aggiornato
- ✓ Per manipolare questa matrice in modo intuitivo, Three.js consente di specificare alcuni campi di mesh che determinano la posizione, orientamento e la scala dell'oggetto:
  - ⇒ Scalatura (`scale`)
  - ⇒ Rotazione (`rotation`)
  - ⇒ Traslazione (`position`)
- ✓ Di default, `matrix` viene automaticamente settata in modo da posizionare e orientare l'oggetto nella posizione voluta
- ✓ Cioè, viene settata al prodotto di tre metrici: di traslazione, rotazione, e scaling $M = T \cdot R \cdot S$
- ✓ Come esercizio, a noi ora interessa settare questa la matrice di modellazione "a mano", decidendo noi il suo valore

33

Assegnamo la matrice di modellazione "a mano"

Assegnamo la nostra matrice come ModelView dell'oggetto **`unaMesh`**.

Per far questo, dobbiamo prima disabilitare il meccanismo
che costruisce automaticamente la matrice in funzione dei campi "posizione, rotazione, e scala" dell'oggetto.

```
unaMesh.matrixAutoUpdate = false;
      // altrimenti Three.js la sovrascrive (come descritto sopra)

unaMesh.matrix = matriceDiRotazioneZ( 30 );
```

Nota: ruotare attorno all'asse Z in spazio vista (o clip)
significa ruotare il disegno in 2D.
Per noi, lo spazio clip è anche lo spazio oggetto e mondo e vista (per ora).

34

Animazione in un'applicazione interattiva

Animazione = sequenza di real-time rendering
(con parametri o contenuti diversi)

Nel nostro caso, da un rendering all'altro cambierà
la **matrice di modellazione** associata all'unico oggetto (per ora)

Non possiamo certo implementare la sequenza come un loop,
perchè si perderebbe l'**interattività** della pagina,

```
while (true) do render_next_frame();
```

Invece, chiediamo al sistema di eseguire la nostra funzione
`render_next_frame()` ad intervalli regolari
(senza trascurare al resto del funzionamento della pagina
fra una chiamata e l'altra, compreso interazione con utente)

35

Animazione nella pagina web in JavaScript

E' molto semplice, in JavaScript, produrre un'animazione in una pagina web.

1. Scriviamo la funzione che intendiamo eseguire in ogni frame.
   (qui: incrementa una variabile "`angolo`", assegna la matrice di modellazione della mesh come una rotazione attorno all'asse delle Z dell'`angolo` corrente, e ripete il rendering)
2. In fondo alla funzione, chiediamo alla macchina virtuale JavaScript di eseguire la funzione stessa nuovamente, una volta appena sia necessario un nuovo frame
3. Invocare la funzione una prima volta (che ne richiederà una 2da invocazione, che ne richiederà una 3za, etc)

```
var angolo = 0; // una variabile globale

function eseguimiOgniFrame() {
        angolo += 1; // un grado a frame

        miaMesh.matrix = matriceDiRotazioneZ( angolo );
        renderizzatore.render(miaScena, miaCamera );

        // "per cortesia, esegui questa stessa funzione appema possibile"
        requestAnimationFrame( eseguimiOgniFrame );
}

eseguimiOgniFrame();
```

Nota JavaScript: occhio alla differenza fra *invocare* una funzione (con "`()`"), come nell'ultima riga, e *riferirsi* ad una funzione (senza "`()`"), come l'argomento che passiamo a `requestAnimationFrame` per indicare cosa essere invocato nel prossimo frame

36

Rotazione attorno ad un punto arbitario

- ✓ Fin qui, il quad ruota attorno al suo centro
- ✓ Come modificare il codice in modo che ruoti attorno ad un punto arbitario, per es, attorno al vertice v1 della mesh?
- ✓ Basta comporre le trasformazioni
  - ⇒ vedi schema alla pagina successiva
  - ⇒ cioè moltiplicare le matrici
  - ⇒ ricordarsi che se M = AB allora moltiplicare per M è equivalente ad effettuare B *seguito da* A

37

Comporre la matrice di modellazione tramite moltiplicazioni matriciali

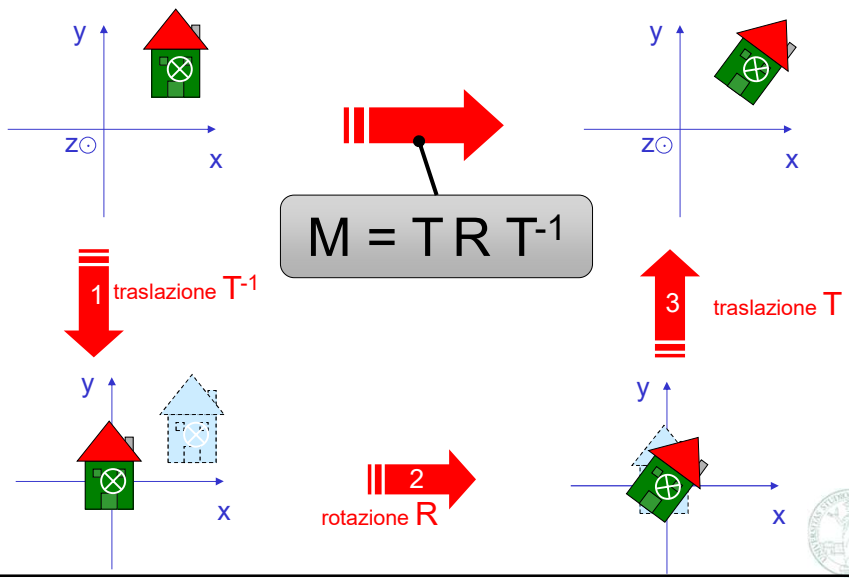
matrice di modellazione dopo il comando

```
unaMesh.matrix.identity(  ); // M = I     ( M = I )
unaMesh.matrix.multiply( C ); // M *= C    ( M = C )
unaMesh.matrix.multiply( B ); // M *= B    ( M = C*B )
unaMesh.matrix.multiply( A ); // M *= A    ( M = C*B*A )
```

- ✓ Questo codice effettua sull'oggetto una mesh le trasformazioni **A**, poi **B**, poi **C** in sequenza
- ✓ La matrice **M** finale vale **C*B*A**
- ✓ L'*ultimo* comando rappresenta la *prima* operazione svolta
- ✓ L'ordine concettuale delle trasformazioni va dall'ultima riga del codice verso l'alto

38

Per ruotare attrono ad un punto arbitrario… (vedere nel codice la traduzione in codice)



39

Microprogetto 2 – piano
File sul sito: `cgLab02.html`

Propositi:

1. Mostriamo un oggetto 3D qualsiasi (non più 2D, sul piano Z = 0)
2. Usiamo una matrice di proiezione prospettica (invece che ortogonale)
3. Ruotiamo l'oggetto in 3D (combinando rotazioni attorno ad assi diversi)
4. Vediamo cosa cambia con un viewport (il canvas html) non quadrato ma rettangolare

40

Geometria procedurale

✓ Rimpiazziamo la nostra geometria "mono quad" con una mesh di un cubo (un "box") generato proceduralmente da un'apposite funzione di three.js

```
var materiale = new THREE.MeshBasicMaterial( ... );

var listaVertici = [
   { x: -0.8 , y: +0.2  , z: 0 }, // v0
   { x: +0.4 , y: +0.4  , z: 0 }, // v1
   ... etc
];
var listaFacce = [
    0 , 2 , 1 ,   // prima faccia
    ... etc
];
var bufferDellaMesh = new THREE.BufferGeometry();
bufferDellaMesh.setFromPoints( listaVertici );
bufferDellaMesh.setIndex( listaFacce );

var unaMesh = new THREE.Mesh( bufferDellaMesh , materiale );
```

sostituisce

```
var geometria = new THREE.BoxBufferGeometry();
```

41

Pin-hole camera (perspective camera) in three.js: parametri intrinseci

- ✓ Per disegnare la scena, avremo bisogno di una `camera` che rappresenta la macchina fotografica virtuale
- ✓ I suoi parametri intrinseci vengono settati nel costruttore

```
var camera = new THREE.PerspectiveCamera( 60, 1.0, 0.1, 10 );
```

FOV — Aspect ratio (il nostro canvas è, per ora, quadrato) — Z_near — Z_far

- ✓ Possiamo osservare la **matrice di proiezione** (vedi) che questo produce guardando il suo campo `camera.projectionMatrix`
  - ⇒ Nota: i suoi sedici numeri `camera.projectionMatrix.elements` descrivono la matrice colonna-per-colonna (cioè in «column-major order»): I primi 4 sono la prima colonna, etc.

42

Spostiamo il cubo lontano dalla camera

- ✓ Ora che la camera non è più ortografica, la sua posizione rispetto al cubo è importante
- ✓ Attualmente (matrice vista = identità) la camera è nell'origine, quindi la camera è dentro al cubo (lo inquadra dall'interno)
- ✓ Dato che vogliamo inquadrare il cubo nella sua interezza, è necessario spostarlo di fronte alla macchina, cioè… traslarlo sulla z di una fattore negativo (per es, di 2 unità)

```
unaMesh.matrix.identity();
/*2do*/ unaMesh.matrix.multiply( matriceDiTraslazione(0,0,-2) );
/*1mo*/ unaMesh.matrix.multiply( matriceDiRotazioneY( angle ) );
```

- ✓ La rotazione effettuata attorno all'asse verticale per prima cosa, («quando il cubo è ancora nell'origine») fa ruotare il cubo attorno a se stesso (la variable **`angle`** cambia ad ogni frame)

43

Spostiamo il cubo lontano dalla camera

- ✓ Esercizio: osservare cosa succede se inverto l'ordine delle due trasformazioni:
  Risposta: invece di ruotare attorno a se stesso, il cubo orbita attorno all'origine (dove è posizionata la camera)
  (Capire il perché!)
- ✓ Ricordare: l'ordine delle trasformazioni cambia il risultato. Infatti il prodotto matriciale NON è commutativo.
- ✓ Come ultima cosa, aggiungiamo una rotazione attorno ad un altro asse, ottenendo una rotazione un po' a caso…

```
unaMesh.matrix.identity();
/*3zo*/unaMesh.matrix.multiply( matriceDiTraslazione(0,0,-2.5) );
/*2do*/unaMesh.matrix.multiply( matriceDiRotazioneY(angle) );
/*1mo*/unaMesh.matrix.multiply( matriceDiRotazioneX(angle/2) );
```

- ✓ (la teoria assicura che ruotando tre volte attorno ai tre assi possiamo ottenere qualsiasi orientamento del cubo)

44

Canvas size e matrice di proiezione

- ✓ Proviamo ora a cambiare la dimensione del canvas:

```
<canvas width="500" height="500" id="unCanvas"></canvas>
```

```
<canvas width="800" height="400" id="unCanvas"></canvas>
```

- ✓ Risultato: la scena 3D viene deformata con una scalatura anisotropica ☹
  - ⇒ Motivo: la trasf di viewport porta un clip space quadrato in uno screen space rettangolare, introducendo una scalatura anisotropica
- ✓ Rimedio: fornire l'aspect ratio (cioè 2 = 800:400) alla definizione della camera
  - ⇒ Vedere come questo si riflette nella matrice di Proiezione

46

Microprogetto 3 – piano
File sul sito: `cgLab03.html`

Propositi:
(dopo aver per ora annullato l'animazione
e ripristinato la matrice di modellazione del cubo all'identità)

1. Cambiamo la matrice di vista
   fornendo all'oggetto camera
   i suoi parametri estrinseci
   (cioè determinando la sua posizione e orientamento nello spazio mondo)

47

Camera in three.js: parametri intrinseci

✓ I suoi parametri estrinseci vengono settati ad esempio da…

```
camera.position.set( 0, 0, 4 ); // setta la posizione (il POV)
camera.up.set( 0, 1, 0 ); // setta l'up vector
camera.lookAt( 0, 0, 0 ); // setta il target position
```

⇒ Vedi lezione corrispondente

✓ Possiamo osservare la
matrice di vista (vedi)
che questo produce
guardando il suo campo `camera.matrixWroldInverse`

⇒ chiamata così in three.js perché la matrice di vista
(che va dal spazio mondo allo spazio camera)
è l'inversa della matrice che va da this (spazio camera) a spazio mondo

⇒ Nota: i suoi sedici `elements`
sono ancora (come sempre) in «column-major order»

48