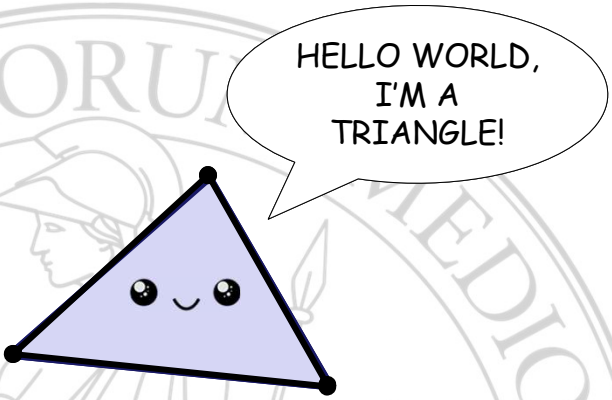
Marco Tarini - Computer Graphics 2025/2026
Università degli Studi di Milano

Hello Triangle (in three.js)



1

I nostri piccoli progetti

Libreria

API

Driver

5

Primo microprogetto – plan of attack
Vedi file: cgLab00.html sul sito

1. Costruiamo una paginetta per il nostro programma ← in HTML
2. Includiamo la lib three.js
3. Prepariamo una mesh in memoria con un solo tri
4. Instanziamo un motore di rendering
5. Mandiamo a schermo la mesh

← in JavaScript con la libreria three.js

6

La pagina HTML (esempio)

```
<html>
<head>
  <title>Hello Triangle</title>
  <style>
        /* qui il css (opzionale) */
  </style>
</head>
<body>
  <h1>Hello triangle!</h1>
  <canvas  id     = “unCanvas"
                width  = 500
                height = 500
   ></canvas>
   <script>
          /* qui il JavaScript */
   </script>
</body>
</html>
```

header

body

7

Procurarsi three.js

✓Scaricare la versione min da

```
https://unpkg.com/three@0.159.0/build/three.min.js
```

✓(oppure, hotlinking)

9

JavaScript: caricamento della Liberia Three.js e poi uso (“inline”)

Prima (per es, dentro l’header)…

```
<script src="three.min.js"></script>
```

source

Scaricare ed posizionare nella stessa cartella del file html, oppure specificare il path, oppure anche hot-linking usando lo URL precedente

Poi (per es, dentro al tag body)…

```
<script>
/* codice JS che usa la lib */
</script>
```

10

JavaScript: caricamento della Liberia Three.js e poi uso (in un file separato)

Prima (per es, dentro l'header)…

```
<script src="three.min.js"></script>
```

source

Scaricare ed posizionare nella stessa cartella del file html, oppure specificare il path, oppure anche hot-linking

Poi (per es, dentro al tag body)…

```
<script src="codice.js"></script>
```

Mio codice JavaScript che usa la lib

11

JavaScript + three.js: inizio

- ✓ Recuperare l'elemento del DOM chiamato "mioCanvas":

```
var elementoCanvas = document.getElementById("unCanvas");
```

- ✓ Costuire una classe che avrà tutto lo stato di WebGL e gestirà la pipeline di rendering:

```
var motore = new THREE.WebGLRenderer(
                    {canvas: elementoCanvas} );
```

Tutte le funzioni di rendering sono disponibili come metodi di questa var

Come parametro, specifichiamo che il viewport del rendering è l'elemento del DOM

- ✓ Primo test: cancelliamo lo schermo di un colore prefissato

```
motore.clear();
```

Invoca (tramite three.js) la funzione di WebGL che setta tutti i pixel del viewport al colore di cancellazione

- ✓ Opzionalmente, si può prima specificare quale colore vada usato:

Blu scuro (vedi lezione sul colore)

```
motore.setClearColor( 0x0000AA );
```

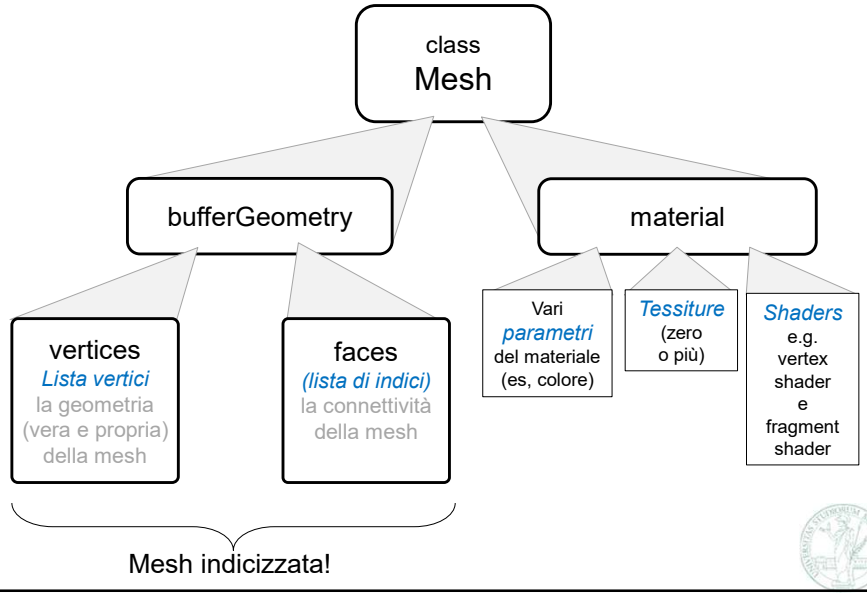
12

JavaScript + three.js: definizione della mesh

- ✓ Definiamo una mesh da renderizzare
- ✓ Come sappiamo, una mesh sono due buffer (due tabelle)
  - ⇒ Uno di vertici, cioè la “geometria”
  - ⇒ Uno di facce cioè la “connettività”
- ✓ In three.js, i due buffer sono in una classe detta bufferGeometry
- ✓ Una Mesh è costituita da un buffer geometry (la mesh stessa) e un materiale, che è una descrizione dell’aspetto di ogni punto della sua superficie.
  Ad esempio, specifica «di che colore è»
- ✓ Costruiamo una mesh semplice che contiene solo tre vertici e il triangolo che li connette

13

Struttura per le Mesh su Three.js



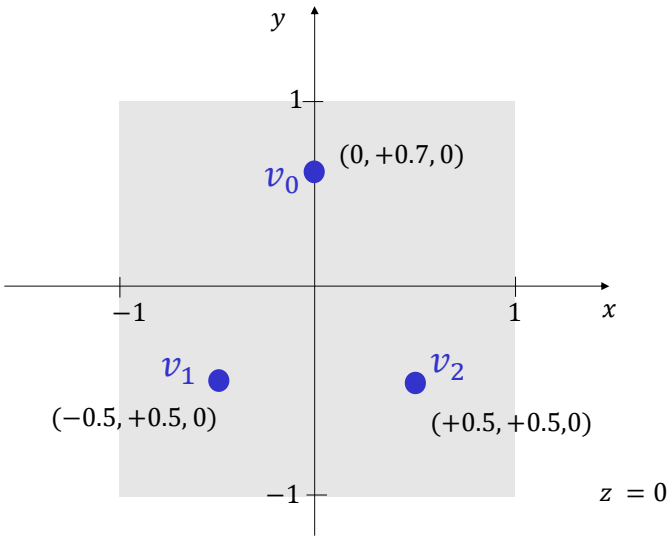
14

Costruiamo una mesh di un solo triangolo: Geometria + connettività

```
var bufferDellaMesh = new THREE.BufferGeometry();
bufferDellaMesh.setFromPoints( vertici );
bufferDellaMesh.setIndex( facce );
```

15

In spazio oggetto (ma anche mondo e clip)



17

Costruiamo una mesh di un solo triangolo: Geometria + connettività

- ✓ Nota: una classe per punti (e/o vettori) 3D in three.js è solo un *oggetto* JavaScript (quindi, espresso in JSON) con tre campi: x, y, z.
- ✓ Per es:

```
var geometria = [
   { x: -0.8 , y: +0.2  , z: 0 }, // v0
   { x: +0.4 , y: +0.4  , z: 0 }, // v1
   { x: -0.7 , y: -0.65 , z: 0 }, // v2
   { x: +0.4 , y: -0.65 , z: 0 }, // v4 (se lo volessimo)
];

var connettività = [
    0 , 2 , 1 ,   // prima faccia
    3 , 1 , 2 ,   // 2da faccia (se la volessimo)
]; // connettività
```

- ✓ Per definizione, la geometria è specificata in spazio oggetto!
  - ⇒ ma visto che M, V e P saranno identità, sono anche già in spazio CLIP+
- ✓ Nota l'edge confiviso (2-1) percorso in due versi opposti

18

Costruiamo un "materiale" in three.js

- ✓ In CG, un «materiale» è la una descrizione formale del modo in cui una superficie reagisce alla luce
  - ⇒ Per es, specifica che la superficie sia opaca, o lucida, o rossa, o cangiante…
- ✓ In contesto di programmazione CG, un materiale è la descrizione dello stato del **pipeline** al momento di disegnare una mesh. Quindi:
  - ⇒ i settaggi del rendering (per es, opzioni per il rasterizer),
  - ⇒ le eventuali tessiture da caricare,
  - ⇒ il vertex-program (o -shader) da eseguire per vertice, e uno per fragment-shader
- ✓ Usiamo il materiale più semplice possibile:

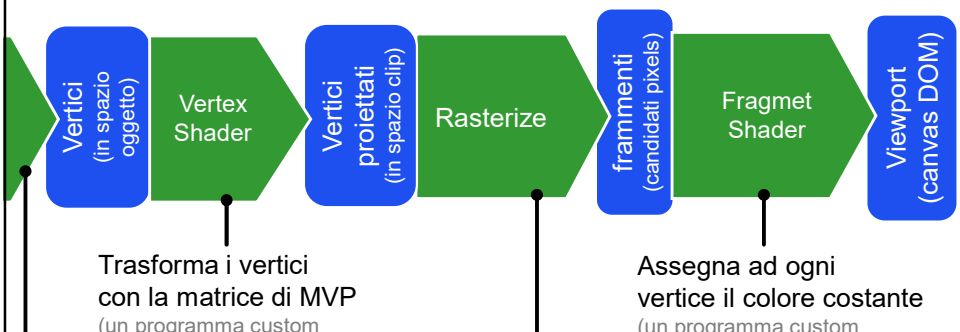
```
var materialeGrigio = new THREE.MeshBasicMaterial();
```

- ✓ Il materiale corrisponde a queste scelte:
  - ⇒ settaggi: tutti default
  - ⇒ programma per vertice: il «minimo sindacale»: trasforma gli oggetti da spazio oggetto a spazio clip tramite la matrice di Model-View-Projection
  - ⇒ programma per frammento: assegna a tutti i frammenti un colore RGB costante

```
materialeGrigio.color = { r: 0.5, g: 0.5, b: 0.5 };
```

19

Cosa fa il materiale «Mesh Basic Material» (cioè: il minimo possibile)



Trasforma i vertici
con la matrice di MVP
(un programma custom
fornito da Three.js
per il materiale dato)

Assegna ad ogni
vertice il colore costante
(un programma custom
fornito da Three.js
per il materiale dato)

Prendi le pos dal buffer dei vertici
come indicizzati dal buffer delle facce
(una funzionalità FISSA dell'HW,
con alcuni settaggi determinati
da Three.js per il materiale dato)

Porta in spazio viewport,
produce i frammenti, etc
(una funzionalità FISSA dell'HW,
con alcuni settaggi determinati
da Three.js per il materiale dato)

20

Rendering

- ✓ L'oggetto di tipo mesh ora può essere costruito:

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

- ✓ A questa mesh corrisponderà la sua matrice di modellazione **matrice di modellazione** (vedi)
- ✓ La possiamo osservare nel suo campo `unaMesh.matrix`
  - ⇒ Di default, questa matrice è l'identità (quindi, per ora, I due spazi **oggetto** e **mondo** coincidono);

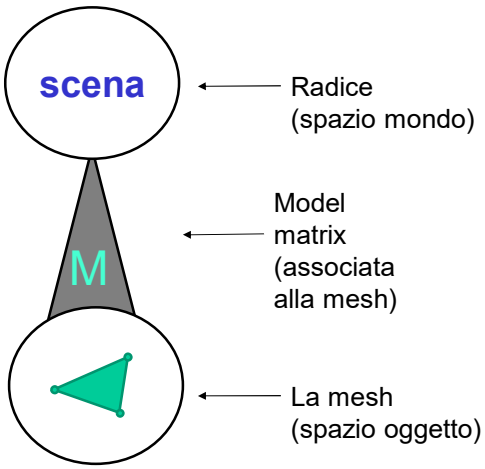
21

JavaScript + three.js: definizione della mesh

- ✓ In three.js, il contenuto è rappresentato in una apposita classe detta “scene”
  - ⇒ Contiene tutti gli oggetti (**mesh**, etc) che compongolo la scena
  - ⇒ Ciascuno di loro è provvisto della propria **matrice di modellazione** che determina il suo posizionamento in spazio mondo
- ✓ Costruiamo una scena semplice che contiene solo la nostra mesh “mono-triangolo”

22

Three.js: scena
Ogni oggetto, la sua matrice



23

Three.js: scena e camera

✓ Costruiamo la scena e appendiamo la mesh costruita

```
var scena = new THREE.Scene();
scena.add(unaMesh);
```

✓ Per disegnare la scena, abbiamo bisogno anche di una camera

```
var telecamera = new THREE.OrthographicCamera(-1,+1,+1,-1,+1,-1);
```

left right top bottom zNear zFa

Modella la (foto) camera.

(Perché questo nome?
è l'inversa della matrice che porta allo spazio mondo dallo spazio vista)

Contiene: la **matrice di VISTA** (`miaCamera.matrixWorldInverse`)
la **matrice di PROIEZIONE** (`miaCamera.projectionMatrix`)

Qui: è una camera **ortografica** (non **prospettica**: il view frustum è un cubo)
Entrambe le matrici sono l'identità:
la P: perché abbiamo settato i limiti del view frustum come lo spazio clip.
la V: perché non abbiamo settato i parametri estrinseci.

24

Rendering

✓ Ora possiamo instruire il rendering

✓ I pixel prodotti apparianno nel canvas associato a `rastrizzatore`

```
rasterizzatore.render( scena, telecamera );
```

Ogni elemento della scena include la sua matrice di Modellazione

Include le matrici di Vista e Proiezione

25

Esercizio

- ✓ Produrre una mesh che rappresenta un poligono QUADRANGOLARE aggiungendo un vertice e un triangolo (ben orientato!)
- ✓ Stessa cosa, per un poligono PENTAGONALE

27

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 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`)

```
	// matrice di traslazione di ( 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;
```

32

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`)

```
        // matrice di rotazione attorno a Z di 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;
}
```

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).

35

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();
```

NO

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)

36

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).

Per es:

```
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 appena 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

37

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

38

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

39