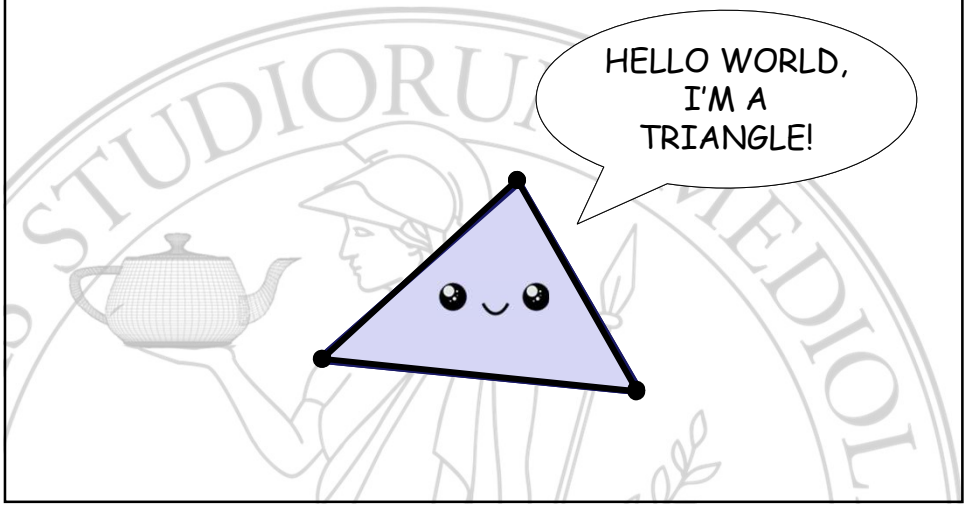
Marco Tarini - Computer Graphics 2020/2021
Università degli Studi di Milano

Hello Triangle (in three.js)



37

Primo microprogetto – plan of attack
File nella pagina del corso: `lab00.html`

1. Costruiamo una paginetta per il nostro programma ← HTML
2. Perpariamo three.js
3. Prepariamo una mesh in memoria con un solo tri
4. Instanziamo un rendering
5. Disegnamo

← JavaScript + three.js

38

La pagina HTML (esempio)

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

head

body

39

JavaScript: preliminare: caricamento della Liberia Three.js

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

source

Scaricare ed posizionare nella stessa cartella del file html

```
<script>
/* comandi che usano la lib */
</script>
```

40

JavaScript + three.js: inizio

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

```
var elem = document.getElementById("mioCanvas");
```

- ✓ Costuire una classe che avrà tutto lo stato di WebGL e gestirà la pipeline di rendering:
  - ⇒ Come parametro, specifichiamo che il viewport di questo rendering sarà l'elemento del DOM

```
var renderizzatore = new THREE.WebGLRenderer({canvas:elem});
```

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

```
renderizzatore.setClearColor( 0x0000BB , 1.0 );
renderizzatore.clear();
```

Blu scuro (vedi lezione sul colore)

Invoca (tramite three.js) la funzione di WebGL che mette tutto lo screen buffer al colore specificato

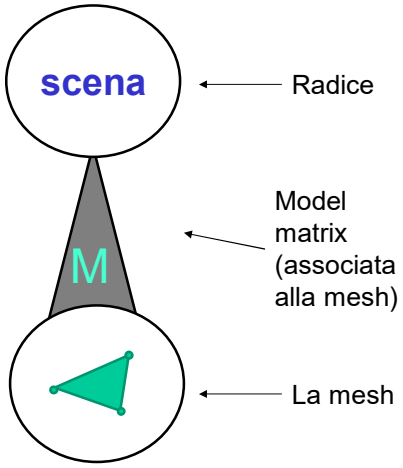
41

JavaScript + three.js: definizione della scena

- ✓ Per prima cosa, dovremo definire del contenuto 3D da disegnare
- ✓ In three.js, il contenuto è contenuto in una apposite classe detta "scene"
  - ⇒ Contiene tutti gli oggetti da disegnare
  - ⇒ Ciascuno, provvisto della sua matrice di modellazione
- ✓ Costruiamo una scena semplice che contiene una sola mesh che contiene un solo triangolo

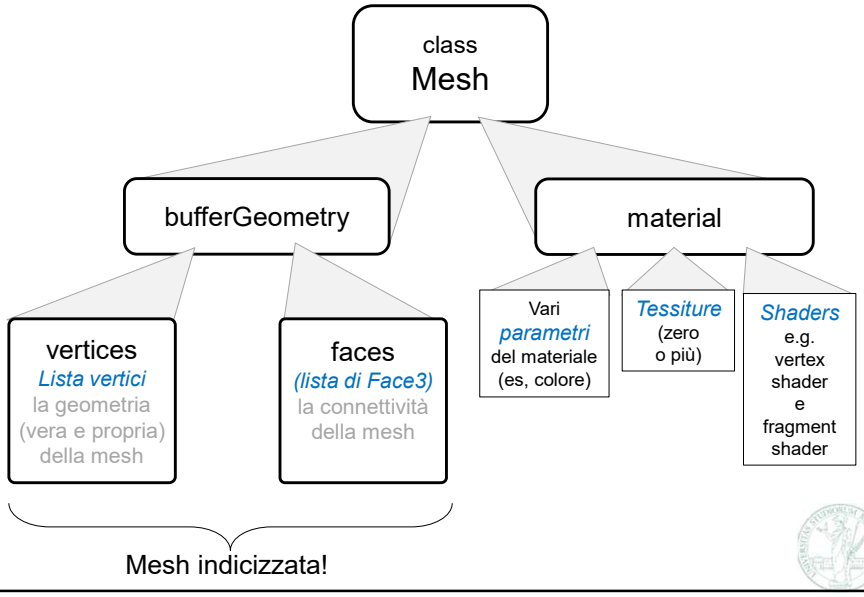
42

Three.js: scena
struttura ad albero (per ora: di due soli nodi)



43

Struttura per le Mesh su Three.js



45

Costruiamo una mesh di un solo triangolo: Geometria + connettività

- ✓ Nota: una class punto in three.js è solo un oggetto JavaScript (cioè JSON) con tre campi: x, y, z

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

var p0 = { x:+0.0 , y:+0.7 , z:2 };
var p1 = { x:-0.5 , y:-0.5 , z:2 };
var p2 = { x:+0.5 , y:-0.5 , z:2 };

geometria.setFromPoints( [p0,p1,p2] ); // geometria (lista verts)
geometria.setIndex( [ 0,1,2 ] ); // connettività (lista facce)
```

- ✓ Ovviamente, la geometria è specificata in spazio oggetto!

46

Costruiamo un “materiale” in three.js

- ✓ Un materiale in CG è la descrizione del modo in cui un oggetto reagisce alla luce
  - ⇒ Per es, un materiale può essere opaco, lucido, rosso, cangiante…
- ✓ In contesto di programmazione CG, un materiale è la descrizione dello stato del pipeline al momento di disegnare (ad esempio) una mesh. Quindi:
  - ⇒ i settaggi del rendering (per es, opzioni per il rasterizer),
  - ⇒ le eventuali tessiture da caricare,
  - ⇒ il programma (shader) da eseguire per vertice e per frammento
- ✓ Usiamo il materiale più semplice possibile:

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

- ✓ Il materiale corrisponde a queste scelte:
  - ⇒ settaggi: tutti default
  - ⇒ tessiture da caricare: nessuna
  - ⇒ processo per vertice: il «minimo sindacale»: trasformare gli oggetti da spazio oggetto a spazio clip tramite la matrice di Model-View-Projection
  - ⇒ processo per frammento: a tutti i frammenti creati assegnamo un colore costante: di default, il bianco

47

Rendering

- ✓ La mesh ora può essere costruita:

```
var miaMesh = new THREE.Mesh( geometria , materiale );
```

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

48

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 nosro canvas è quadrto)
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.

49

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

50

Rendering!

- ✓ Possiamo ora eseguire il rendering (su GPU) invocando

```
renderizzatore.render( miaScena, camera );
```

Quale scena (una sola mesh di un solo triangolo)

Vista con quale camera

- ✓ Questo causerà la trasfornazione del triangolo
  - ⇒ Usando la matrice di modellazione, vista e proiezione viste sopra
- ✓ Seguita dalla sua rasterizzazione
- ✓ Seguita dal processamento di tutti i frammenti generati
  - ⇒ Per ora, producendo altrettanto pixel bianchi

51