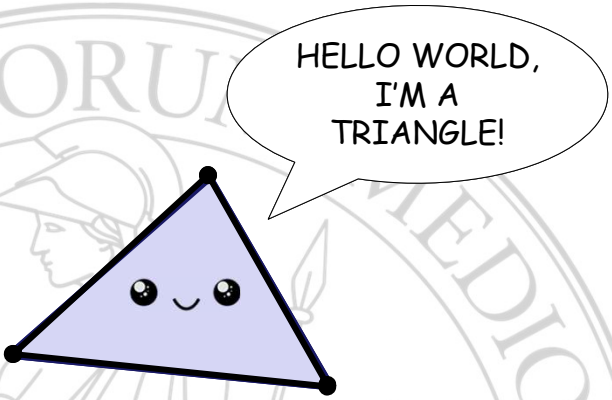
Marco Tarini - Computer Graphics 2024/2025
Università degli Studi di Milano

Hello Triangle (in three.js)



1

3D sul web: tentativi di soluzione **in passato**

✓Soluzioni:

⇒**Remote rendering** interattivo
- Il client richiede, il server renderizza e manda immagini
- Esempio: **Stadia** by Google (per games)
- (fra l'altro: ottima scelta per DRM !)

⇒Rendering come **preprocessing**
- Esempio: **QTVR** by Apple, Inc.

⇒Web-oriented formats for 3D interactive scenes
- Esempi: **VRML**, **X3D, Unity bundles**

⇒**Plugins**:
- **Flash** by Adobe
- **Silverlight** by Microsoft

3

3D sul the Web: alcune soluzioni pratiche oggi

- ✓ **WebGL** by Khronos (API)
  - ⇒ Graphics Hardware acceleration from browsers
  - ⇒ Javascript based
  - ⇒ Native! – no plugin
  - ⇒ ⊂ **GLES 2.0** ⊂ **OpenGL**
- ✓ **three.js**
  - ⇒ Higher level
  - ⇒ OpenSource, *free*, MIT licensed
- ✓ **Emscripten**
  - ⇒ transpiler: [C++] **+** [OpenGL] → [WASM] **+** [WebGL]
- ✓ **Unity** (game engine)
  - ⇒ exports projects as web applications
- ✓ Ad hoc higher level dev-tools:
  - ⇒ **3DHOP** - 3D Heritage Online Presenter
  - ⇒ **Google Earth Engine**
  - ⇒ ...

4

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

- ✓ Si può scaricare scaricare la versione (“minifiaca”) da…

https://tarini.di.unimi.it/**tmp/three.min.js**

- ✓ Hotlinking: tollerato

8

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 il link precedente

Poi (per es, dentro al tag body)…

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

10

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