P.PORTO

POLITÉCNICO
DO PORTO
ESMAD

COMPUTAÇÃO GRÁFICA
TSIW

Three.js

- Even in simple scenes, graphic programming (WebGL) can be quite complex
  - Maths
  - GLSL (shaders)…

Three.js

- **Three.js**: 3D library in JavaScript, free, abstract layer on top of WebGL
- Allows renderings in Canvas2D, WebGL or SVG
- Created in 2010; author Ricardo Cabello (Mr. Doob)
- https://threejs.org/
- Latest release: 159

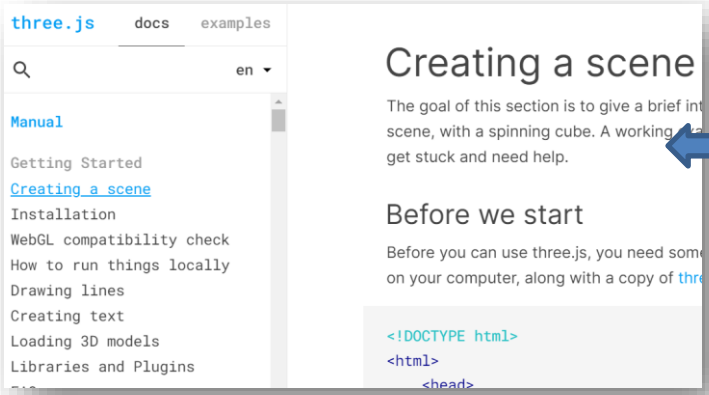
Mr.doob

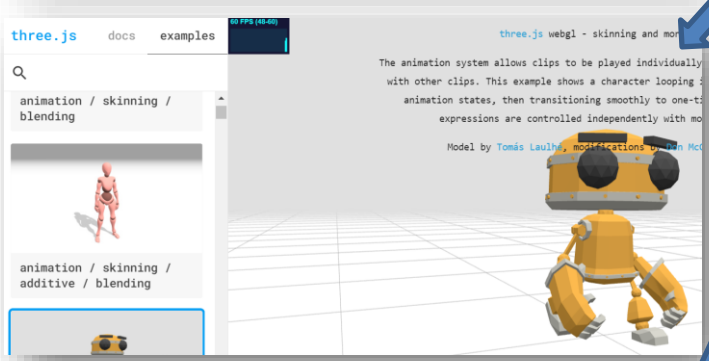
@mrdoob

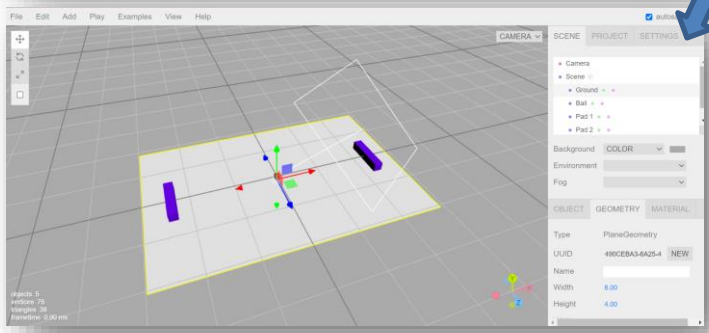
Non-creative award-losing junior developer.

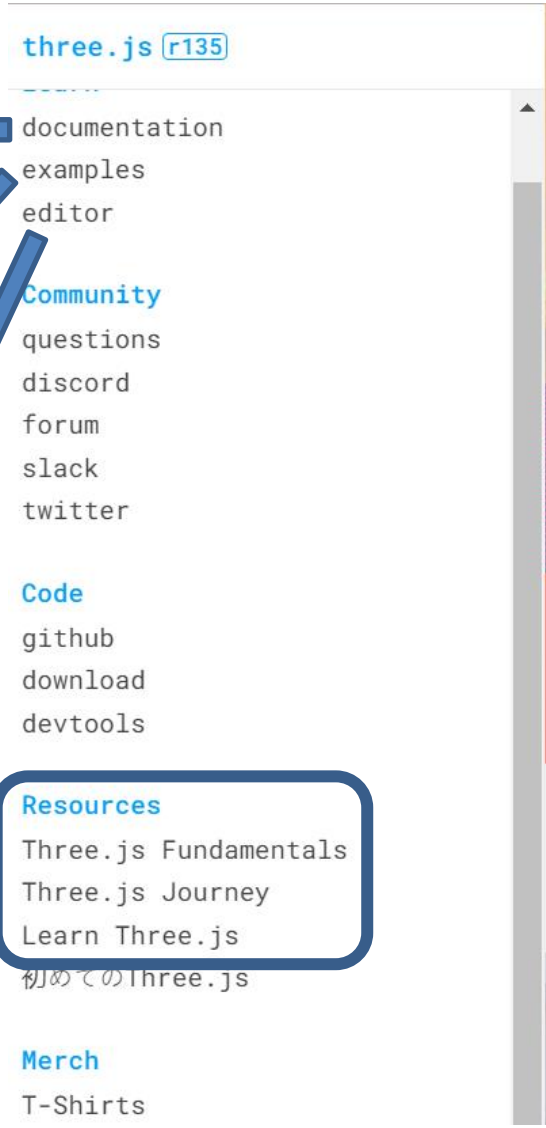
Barcelona, Spain · http://mrdoob.com

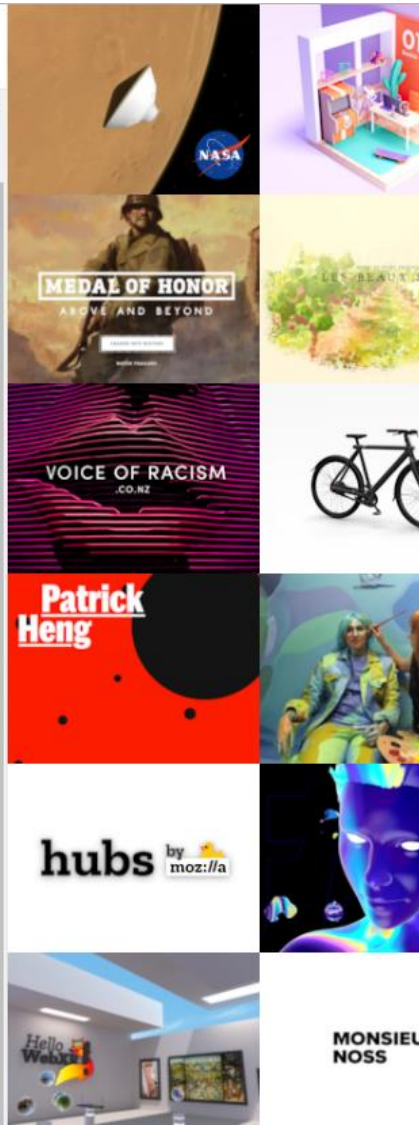
Three.js


three.js
docs
examples
en
Manual
Getting Started
Creating a scene
Installation
WebGL compatibility check
How to run things locally
Drawing lines
Creating text
Loading 3D models
Libraries and Plugins
Creating a scene
The goal of this section is to give a brief int
scene, with a spinning cube. A working
get stuck and need help.
Before we start
Before you can use three.js, you need som
on your computer, along with a copy of thr
<!DOCTYPE html>
<html>
<head>


three.js
docs
examples
animation / skinning / blending
animation / skinning / additive / blending
three.js webgl - skinning and mor
The animation system allows clips to be played individually
with other clips. This example shows a character looping
animation states, then transitioning smoothly to one-ti
expressions are controlled independently with mo
Model by Tomás Laulhé, modifications by Eric McC


File Edit Add Play Examples View Help
CAMERA
SCENE
PROJECT
SETTINGS
Camera
Scene
Ground
Ball
Pad 1
Pad 2
Background
COLOR
Environment
Fog
OBJECT
GEOMETRY
MATERIAL
Type
PlaneGeometry
UUID
490CEBA3-6A25-4
NEW
Name
Width
6.00
Height
4.00


three.js r135
documentation
examples
editor
Community
questions
discord
forum
slack
twitter
Code
github
download
devtools
Resources
Three.js Fundamentals
Three.js Journey
Learn Three.js
初めてのThree.js
Merch
T-Shirts


NASA
MEDAL OF HONOR
ABOVE AND BEYOND
VOICE OF RACISM
.CO.NZ
Patrick
Heng
hubs
by
moz://a
Hello
WebXR
MONSIEU
NOSS

P.PORTO

ESCOLA SUPERIOR DE MEDIA ARTES E DESIGN

Three.js – how to use?

- Install the Three.js npm module, or get started quickly with a CDN
- Three.js project basic structure:

index.html

```
<!DOCTYPE html>
<html>
<head>
    <style>
        body { /* use the full screen */ margin: 0; overflow: hidden; }
    </style>
    <script type="importmap"> /* define where to get the THREE.JS package */
     {
       "imports": {
         "three": "https://unpkg.com/three/build/three.module.js" /* main */
        }
      }
    </script>
</head>
…
```

P.PORTO

ESCOLA SUPERIOR DE MEDIA ARTES E DESIGN

Three.js – how to use?

- You can install the Three.js npm module, or get started quickly with just static hosting or a CDN
- Three.js project basic structure:

index.html

```
…

<body>
    <!-- HTML body will hold the Output -->
    <script type="module" src="main.js"></script>
</body>

</html>
```

Main script file

Notice that your DOM does not have a Canvas element!

P.PORTO

ESCOLA SUPERIOR DE MEDIA ARTES E DESIGN

Three.js – how to use?

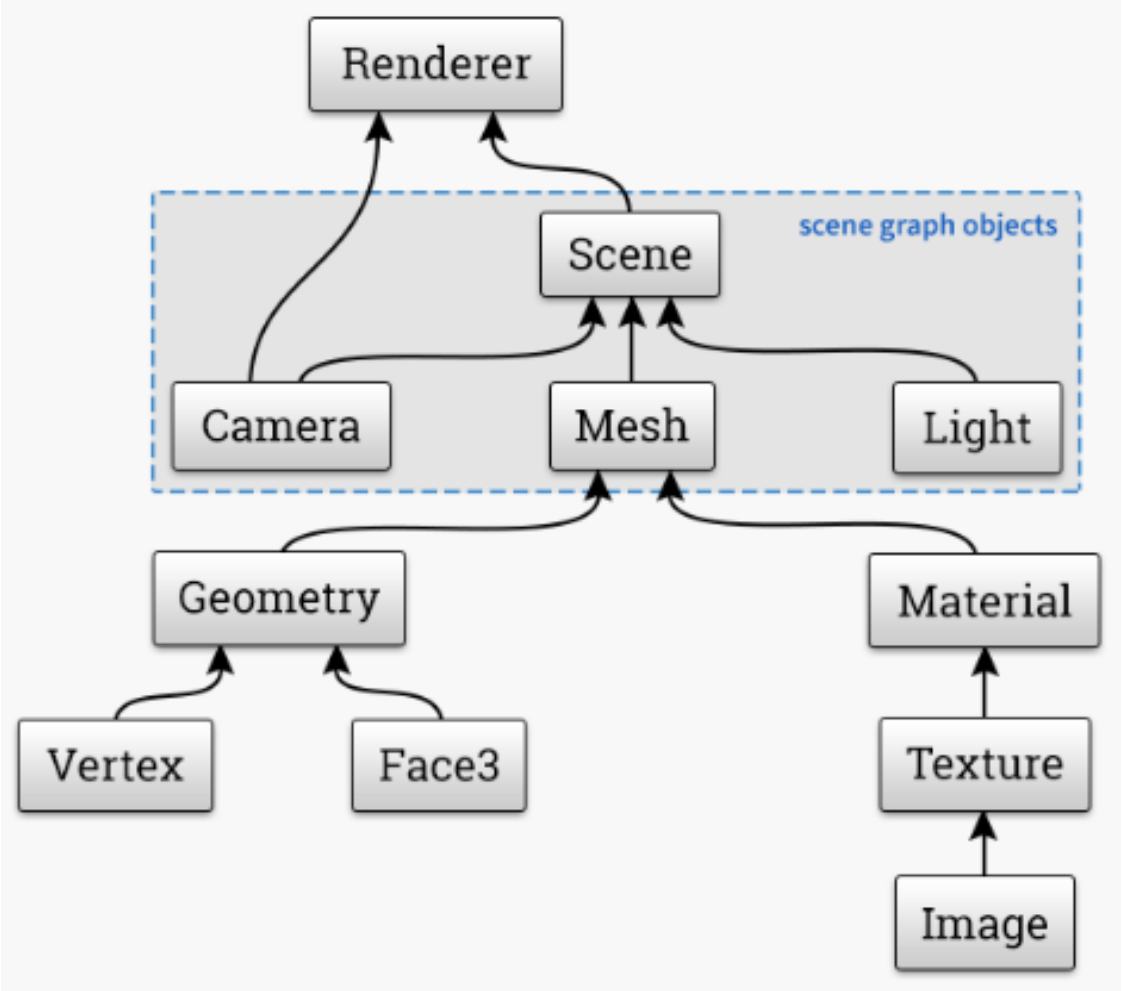
- You can install the Three.js npm module, or get started quickly with just static hosting or a CDN
- Three.js project basic structure:

main.js

```
import * as THREE from 'three';

/**********************************************************************
* - Create a scene, a camera and a renderer
* - add the renderer output to an HTML element (in the example, the body)
* - create an object 3D and add it to the scene
* - define the animation function
* - render the scene ("draw" the scene into the Canvas, using the camera's
point of view)
* *********************************************************************/
```

Three.js – basic application

- Three.js components



P.PORTO

ESCOLA SUPERIOR DE MEDIA ARTES E DESIGN

Three.js – basic application

main.js

```
import * as THREE from 'three';

/* SCENE: create an empty scene, that will hold all the elements */
const scene = new THREE.Scene();

/* CAMERA: create a camera, which defines where we're looking at */
const camera = new THREE.PerspectiveCamera(75, window.innerWidth /
window.innerHeight, 0.1, 1000);
camera.position.z = 5; // move the camera to the world position (0,0,5)

/* RENDERER: create a renderer */
const renderer = new THREE.WebGLRenderer();
// set its size (the full page)
renderer.setSize(window.innerWidth, window.innerHeight);
// configure the clear color
renderer.setClearColor("#000000");
// add the output of the renderer to an HTML element (this case, the body)
document.body.appendChild(renderer.domElement);

...
```

Scene

Camera

Renderer

Three.js – basic application

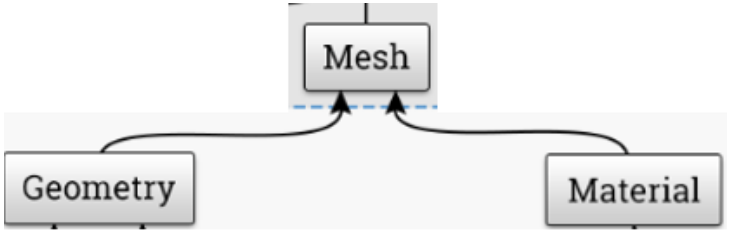
- **If** your page already as a Canvas element, use this:

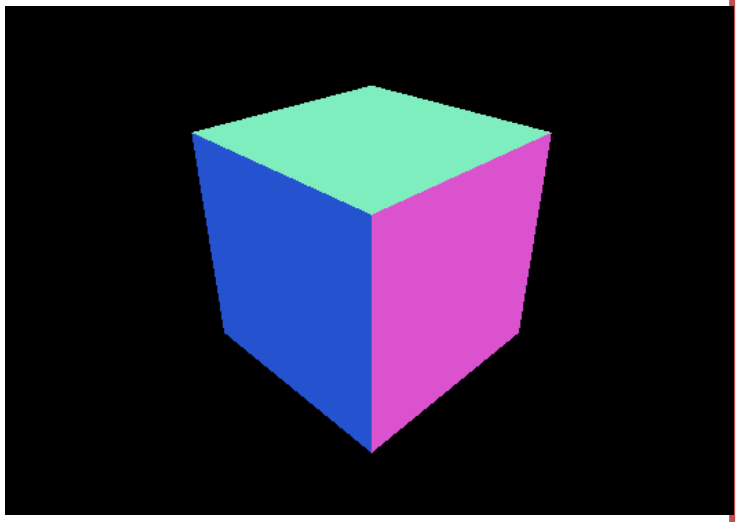
```
/* RENDERER: create a renderer */
const renderer = new THREE.WebGLRenderer(canvas:canvasID );
...
```

If your DOM already has a Canvas element, just indicate its ID in the WebGLRenderer *canvas* property

Three.js – basic application

```
...
/* MESH - create a 3D object - a cube
* ******************/
// create an object 3D - a cube
let geometry = new THREE.BoxGeometry(2, 2, 2);
let material = new THREE.MeshNormalMaterial();
const cube = new THREE.Mesh(geometry, material);
// add the cube to the scene
scene.add(cube);

// set the animation funcion
renderer.setAnimationLoop(render);

function render() {
    // rotate the cube around its axes
    cube.rotation.y += 0.01;
    cube.rotation.z += 0.01;

    // "draw" the scene into the Canvas, using the camera's point of view
    renderer.render(scene, camera);
};
```





P.PORTO

ESCOLA SUPERIOR DE MEDIA ARTES E DESIGN

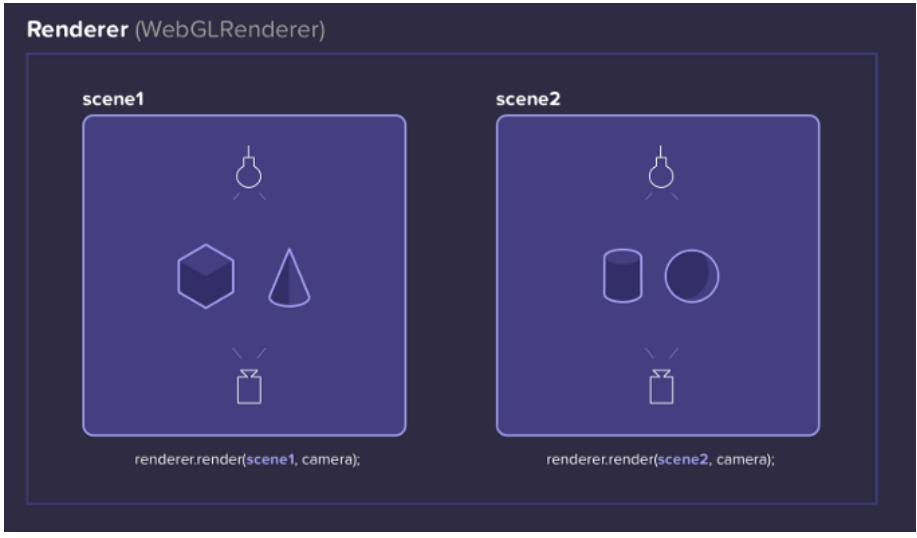
Three.js – basic application

MAIN STEPS

1. **Create a *renderer***
   DOM element where a scene is visualized; simply draws everything from the scene to the WebGL canvas

   https://threejs.org/docs/#api/renderers/WebGLRenderer

2. **Create a scene**
   Composed by light(s) + mesh(es) + camera(s); a program can have as many scenes as wanted, but one renderer can draw only one scene at once
   https://threejs.org/docs/#api/scenes/Scene



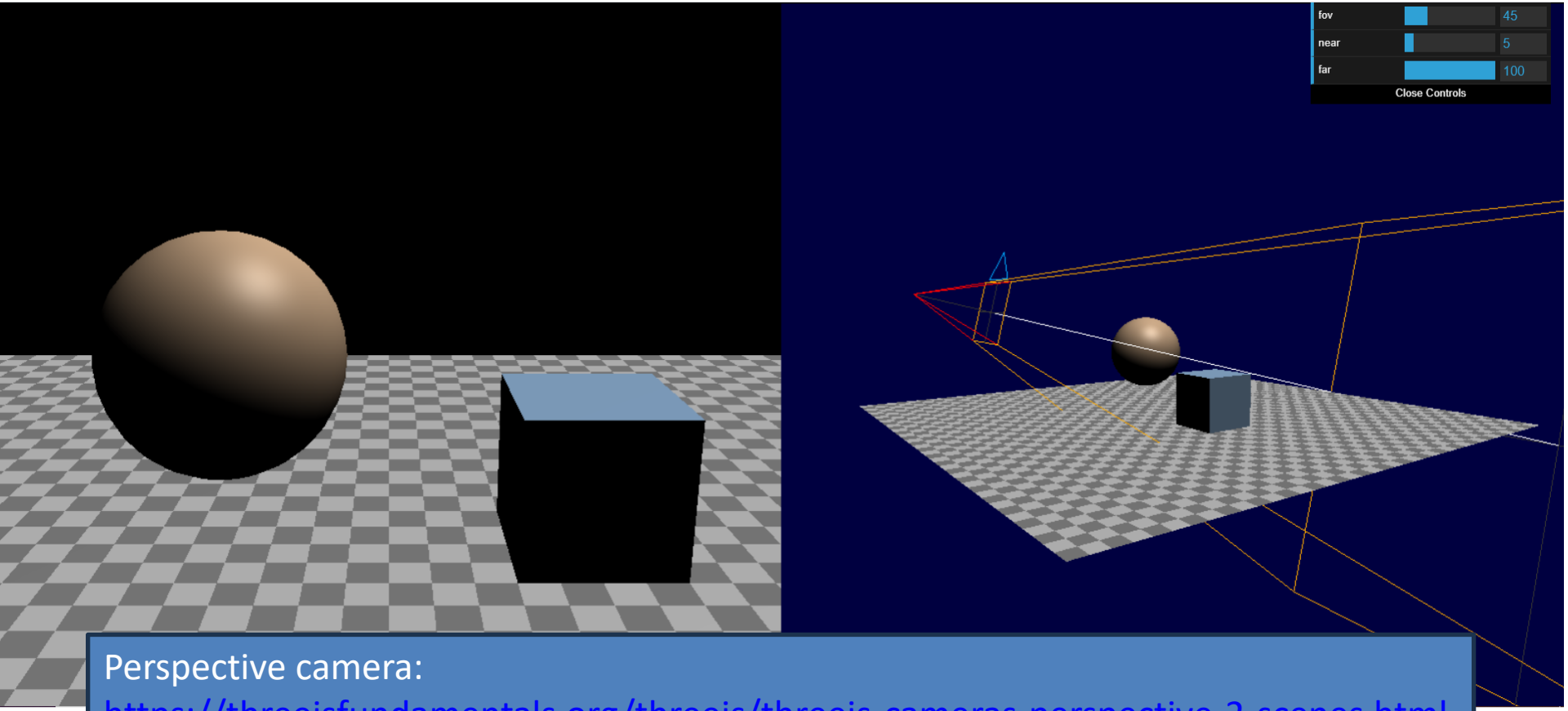
P.PORTO

ESCOLA SUPERIOR DE MEDIA ARTES E DESIGN

Three.js – basic application

3. **Create a camera** (orthographic, perspective, …)

https://threejs.org/docs/#api/cameras/Camera

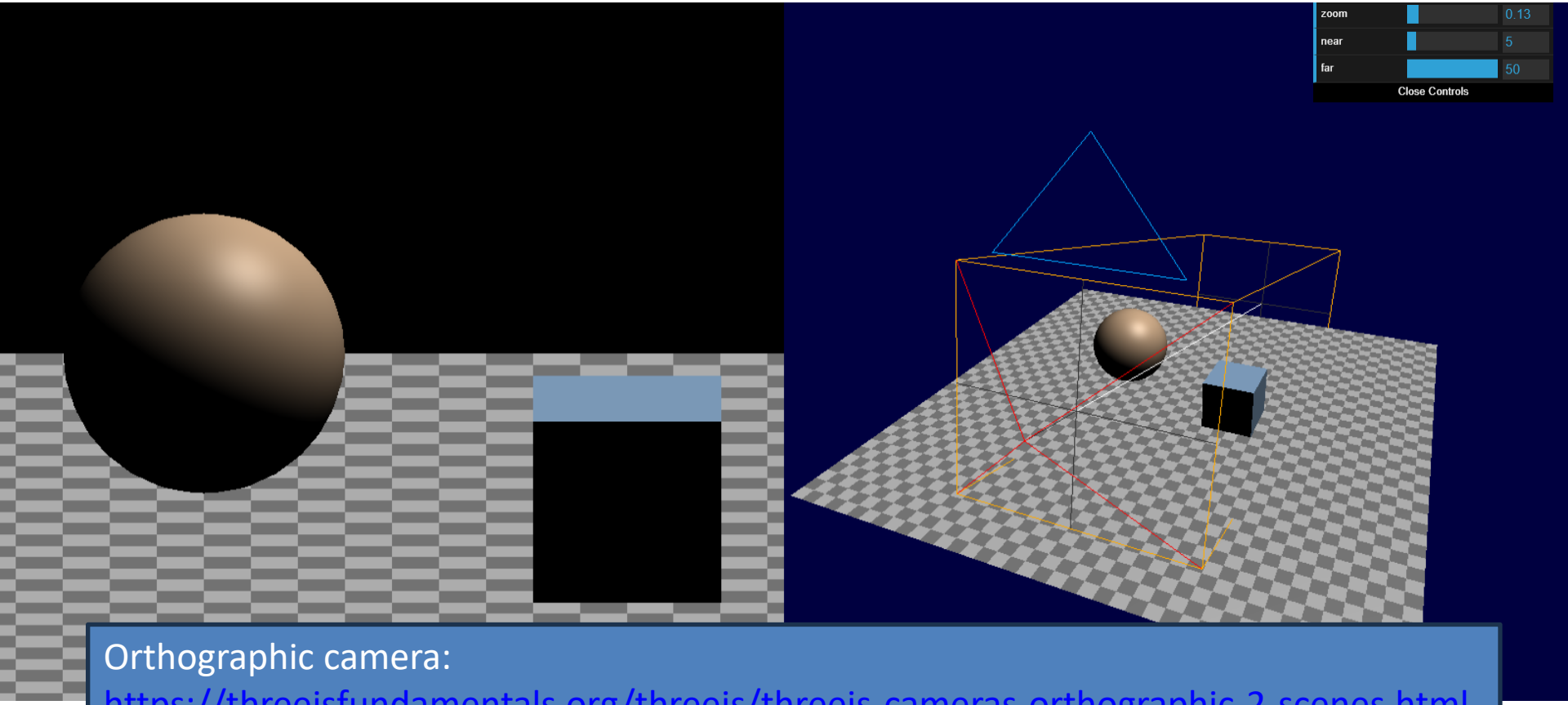


Perspective camera:
https://threejsfundamentals.org/threejs/threejs-cameras-perspective-2-scenes.html

Three.js – basic application

3. **Create a camera** (orthographic, perspective, …)

https://threejs.org/docs/#api/cameras/Camera



Orthographic camera:
https://threejsfundamentals.org/threejs/threejs-cameras-orthographic-2-scenes.html

P.PORTO

ESCOLA SUPERIOR DE MEDIA ARTES E DESIGN

Three.js – basic application

3. **Create a geometry**

   https://threejs.org/docs/#api/core/Geometry

4. **Create a material**

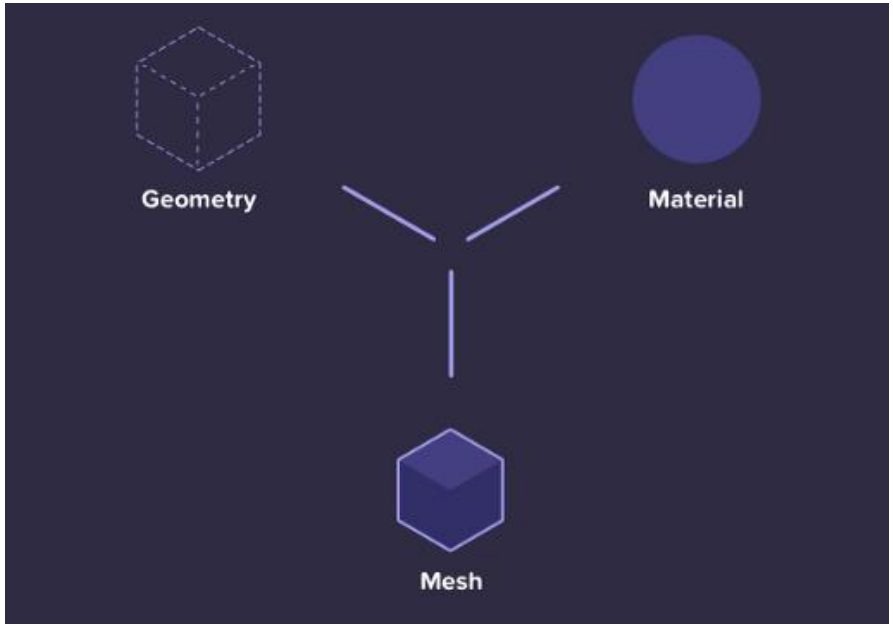
   Sets the appearance of an object - https://threejs.org/docs/#api/materials/Material

5. **Create a mesh**

   Triangular polygon mesh, composed by geometry + material

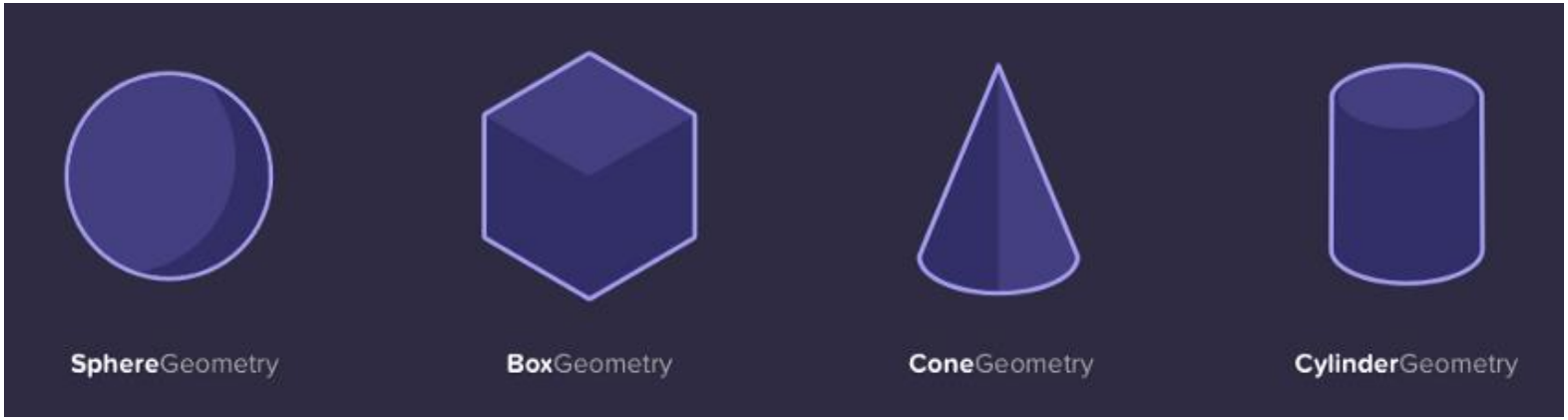
   https://threejs.org/docs/#api/objects/Mesh

6. **Add** the mesh to the scene

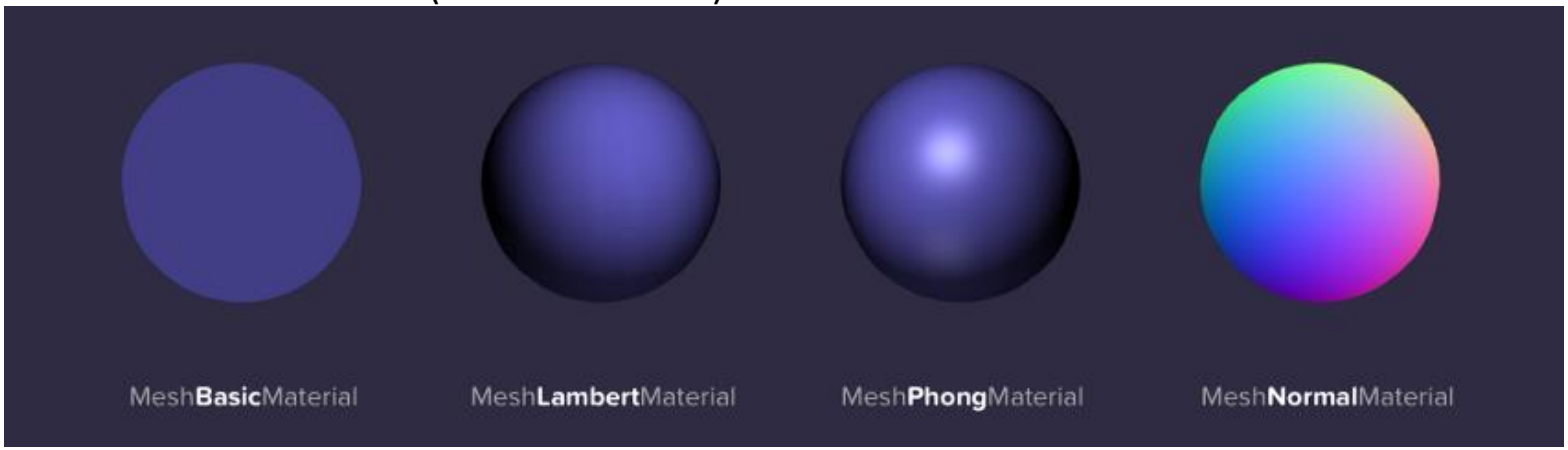


Three.js – basic application

- Built-in geometries (some of them…)



- Built-in materials (some of them…)



Three.js – basic application

- From the Three.js documentation:

Geometries

BoxGeometry
CapsuleGeometry
CircleGeometry
ConeGeometry
CylinderGeometry
DodecahedronGeometry
EdgesGeometry
ExtrudeGeometry
IcosahedronGeometry
LatheGeometry
OctahedronGeometry
PlaneGeometry
PolyhedronGeometry
RingGeometry
ShapeGeometry
SphereGeometry
TetrahedronGeometry
TorusGeometry
TorusKnotGeometry
TubeGeometry
WireframeGeometry

Materials

LineBasicMaterial
LineDashedMaterial
Material
MeshBasicMaterial
MeshDepthMaterial
MeshDistanceMaterial
MeshLambertMaterial
MeshMatcapMaterial
MeshNormalMaterial
MeshPhongMaterial
MeshPhysicalMaterial
MeshStandardMaterial
MeshToonMaterial
PointsMaterial
RawShaderMaterial
ShaderMaterial
ShadowMaterial
SpriteMaterial

Three.js - basic application

- There are several **pre-defined** (built-in) geometries in Three.js
  - E.g: documentation for the BOX geometry

three.js docs examples

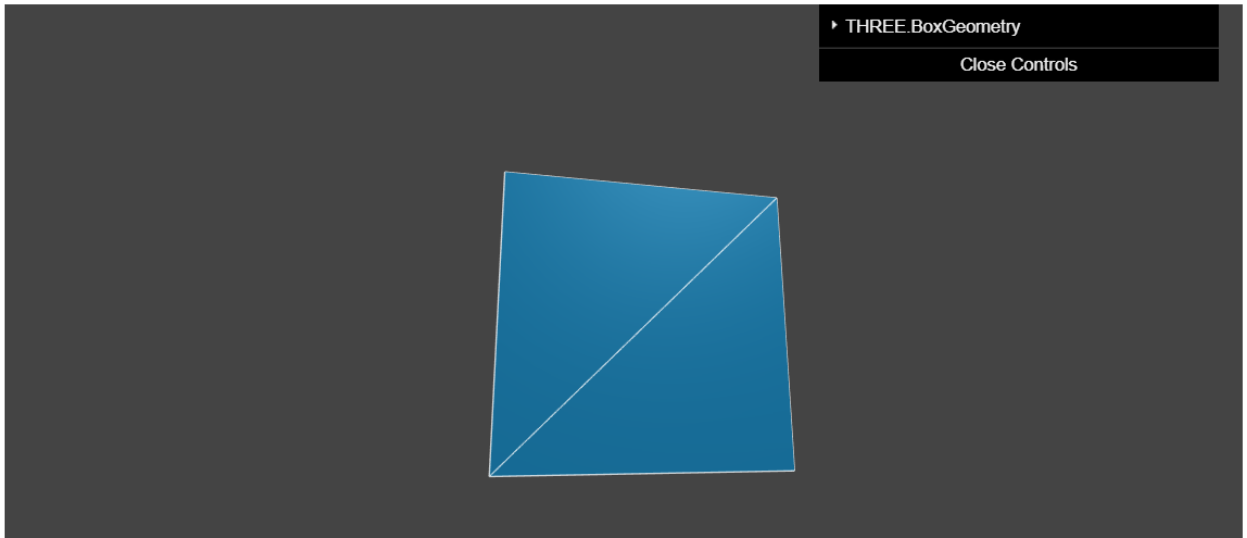
geometry

Geometries
BoxBufferGeometry
BoxGeometry
CircleBufferGeometry
CircleGeometry
ConeBufferGeometry
ConeGeometry
CylinderBufferGeometry
CylinderGeometry
DodecahedronBufferGeometry
DodecahedronGeometry
EdgesGeometry
ExtrudeBufferGeometry
ExtrudeGeometry
IcosahedronBufferGeometry
IcosahedronGeometry
LatheBufferGeometry
LatheGeometry

Geometry →

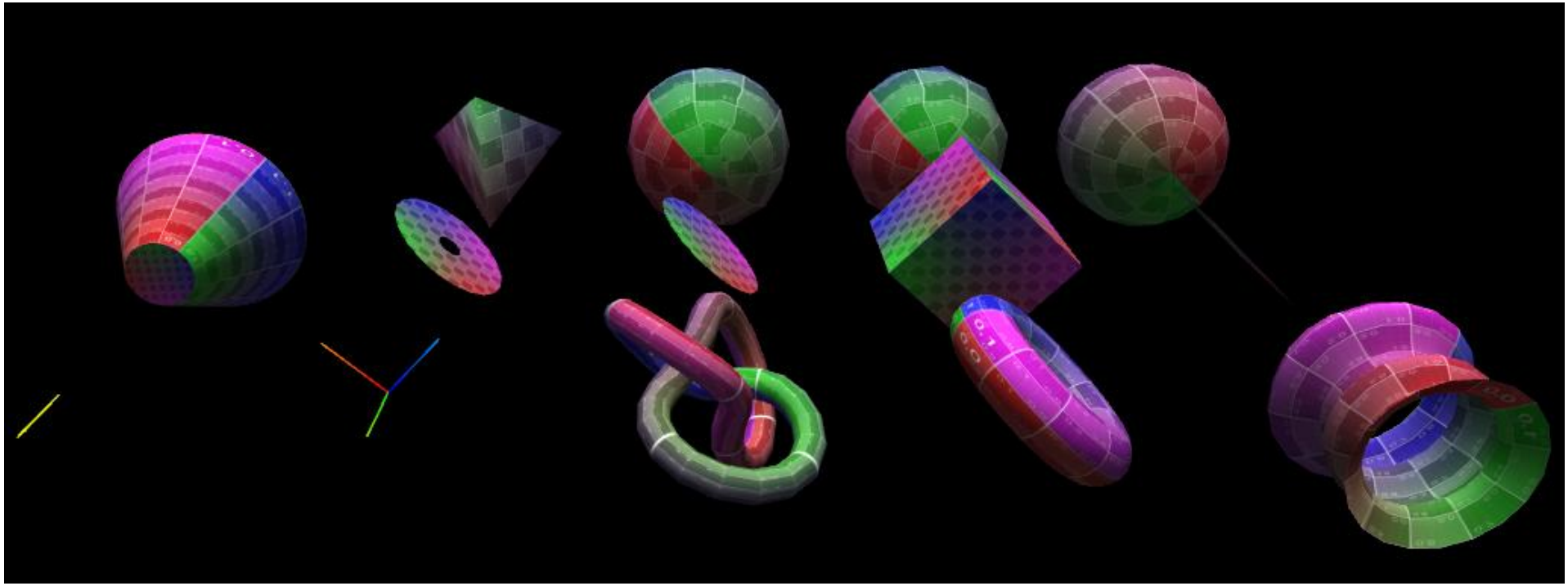
BoxGeometry

BoxGeometry is a geometry class for a rectangular cuboid with a given 'width', 'height', and 'depth'. On creation, the cuboid is centred on the origin, with each edge parallel to one of the axes.



Three.js - basic application

- There are several **pre-defined** (built-in) geometries in Three.js
  - Example: https://threejs.org/examples/#webgl_geometries



Three.js – basic application

8. Animate the scene

Change parameters of the objects, lights and/or cameras

Object 3D transformations:

https://threejs.org/docs/#api/core/Object3D

// object's local rotation - in **radians**
object.**rotation.x** / *object*.**rotation.y** / *object*.**rotation.z** = THREE.Math.degToRad(θdegrees)
object.**rotation.set**(θ_X, θ_Y, θ_Z)

object.**rotateX**(θ) / *object*.**rotateY**(θ) / *object*.**rotateZ**(θ)

// object's local position – can be altered
object.**position.x** / *object*.**position.y** / *object*.**position.z**
object.**position.set(X,Y,Z)**

// translate object in object's space coordinates
object.**translateX(***distance***)** / *object*.**translateY(***distance***)** / *object*.**translateZ(***distance***)**

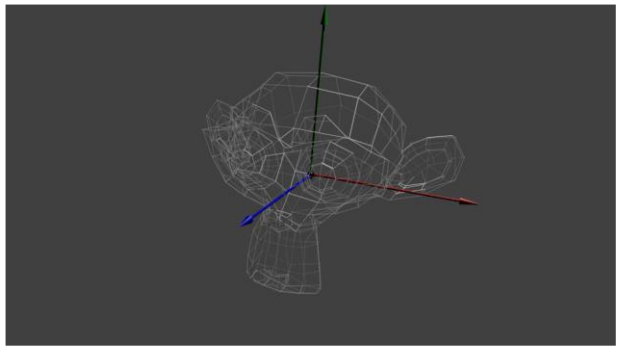
// object's local scale
object.**scale.x** / *object*.**scale.y** / *object*.**scale.z**
object.**scale.set(X,Y,Z)**

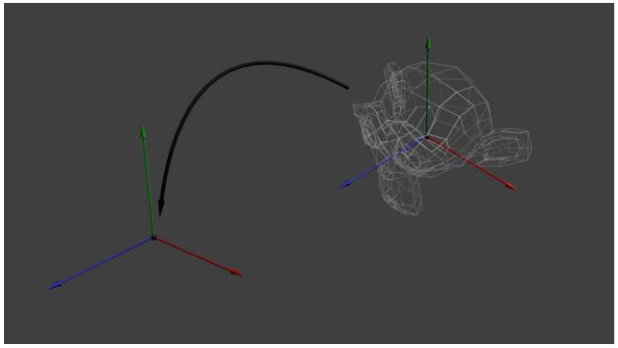
Three.js – basic application

Object's transform matrices (4x4):

https://threejs.org/docs/#api/core/Object3D

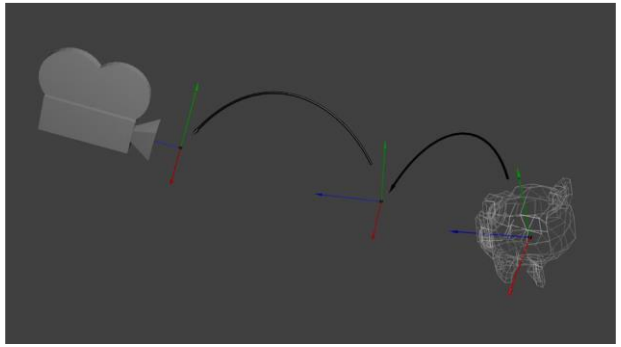
// object's local transform matrix
object.**matrix**





// object's global transform matrix (if the objet has no parent, it is identical to its local transform matrix)
object.**matrixWorld**

// model & view matrix passed to the vertex shader and used to calculate the position of the object
object.**modelViewMatrix**



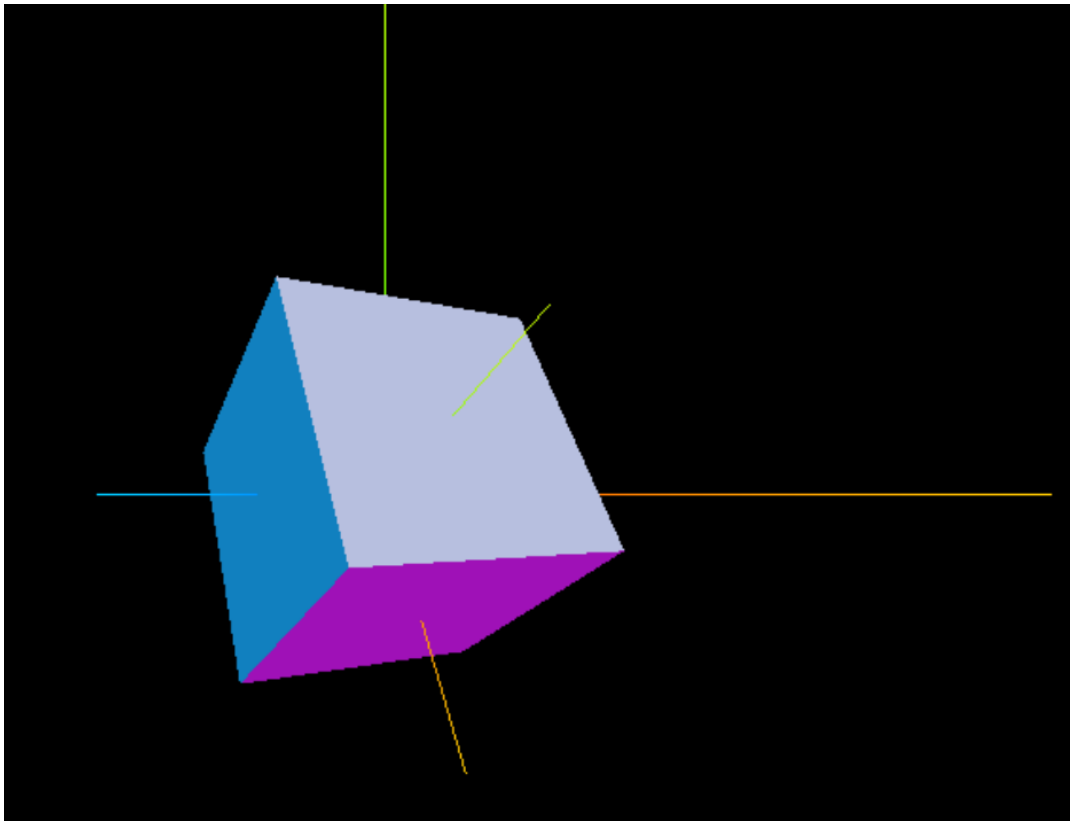
P.PORTO

ESCOLA SUPERIOR DE MEDIA ARTES E DESIGN

Three.js – AxesHelper

- Use the ***AxesHelper*** to visualize the 3 axes of a coordinate system (CS)
  - https://threejs.org/docs/#api/helpers/AxesHelper
  - before the `render` function, add one to the scene (World CS) and other to the cube (Model CS)

```
/************************
 * AXES HELPER
 ************************/
// show axes for the WORLD CS
let axes = new THREE.AxesHelper(3);
scene.add(axes);
// show smaller axes for the CUBE CS
let axesCube = new THREE.AxesHelper(2);
cube.add(axesCube);


/*****************************
 * ANIMATE
 * ***************************/
// set the animation function
renderer.setAnimationLoop(render);
```



Three.js – OrbitControls

- Other three.js components — such as controls, loaders, and post-processing effects — are part of the addons/ directory
  - Addons do not need to be installed separately, but do need to be imported separately

index.html

```
…
    <script type="importmap"> /* define where to get the THREE.JS package */
     {
       "imports": {
         "three": "https://unpkg.com/three/build/three.module.js", /* main */
         "three/addons/": "https://unpkg.com/three/examples/jsm/", /* add-ons */
        }
      }
    </script>
…
```

Three.js – OrbitControls

- Orbit controls allow the camera to orbit around the center of the scene or around a target
  - Create a new instance of the orbit controls and pass the camera to be controlled and the HTML element used for event listeners

main.js

```
import * as THREE from 'three';
import { OrbitControls } from 'three/addons/controls/OrbitControls.js';

…
const controls = new OrbitControls(camera, renderer.domElement);
…

function render() {
    …
    // update the controls (it is not always required: read docs)
    controls.update();
    …
};
```

Three.js - geometries

- It is possible to define **your own geometry**, knowing that a mesh is built with **vertices**, that form **faces**
  - **Example**: add a **triangle** to our previous cube

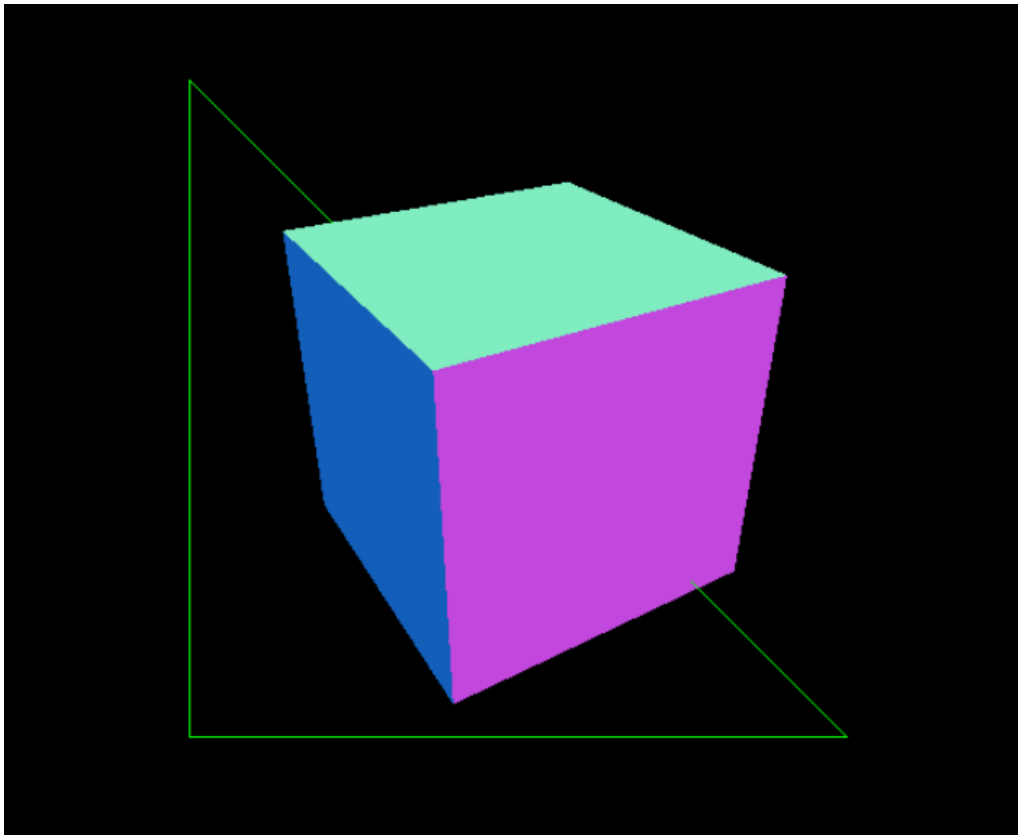
```
...
  // TRIANGLE - build a custom geometry, composed of 3 vertices
  geometry = new THREE.BufferGeometry();

  const points = [] // define array of vertices
  points.push(new THREE.Vector3(-2, 2, 0))
  points.push(new THREE.Vector3(-2, -2, 0))
  points.push(new THREE.Vector3(2, -2, 0))

  geometry.setFromPoints(points);

  material = new THREE.MeshBasicMaterial({ color: 0x00FF00, wireframe: true });
  let triangle = new THREE.Mesh(geometry, material);

  scene.add(triangle); // add the triangle to the scene
...
```

?

What if you add the triangle, NOT to the scene, but **to the cube**?

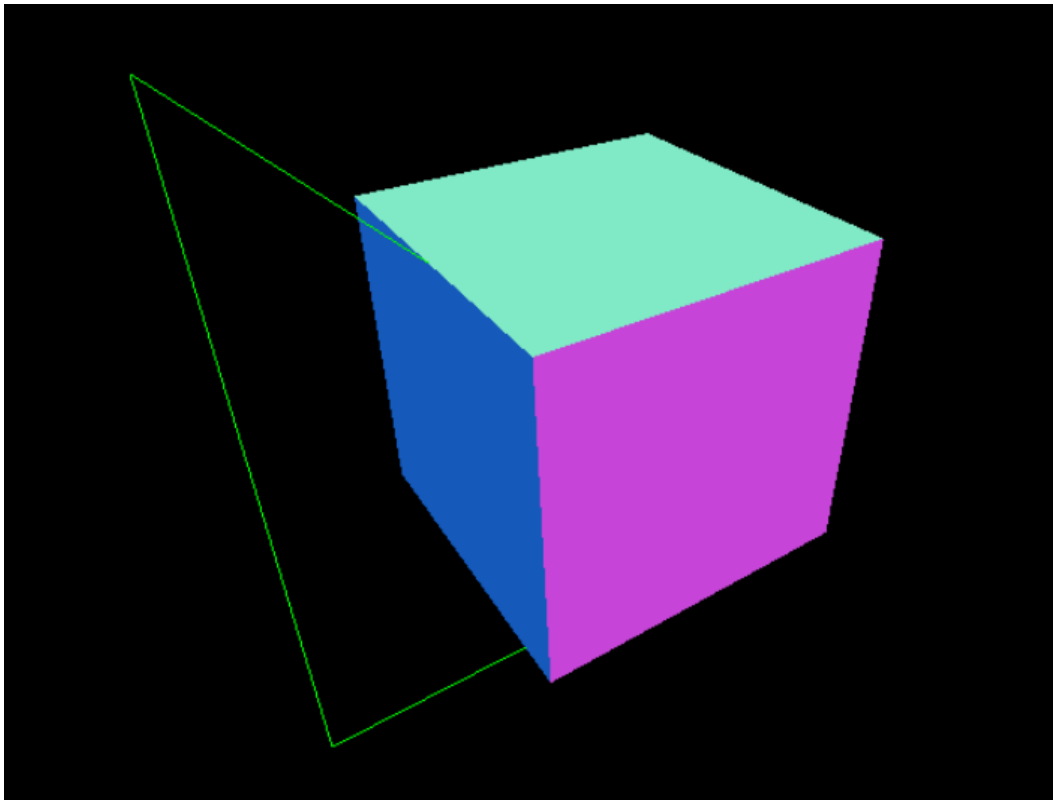
Three.js - geometries

- **Example 02**: add a **triangle** to the scene



Three.js - geometries

- Alter the example 02: instead of adding the triangle to the scene, **add it to the cube**
  (this is a **composed/hierarchical object** – will be better explained latter)



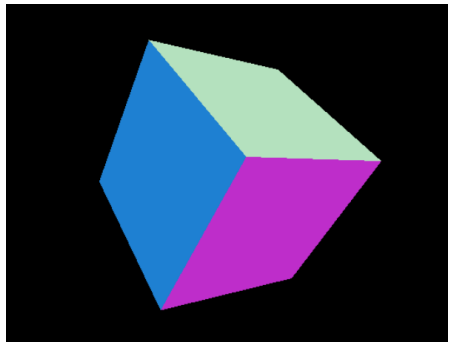
HINT: to inspect THREE.JS objects, add them to the `window` JS object (to be able to log them on the browser console)

```
window.cube = cube
```

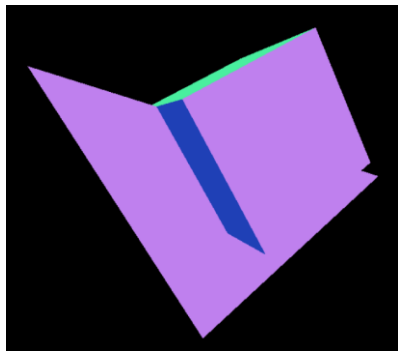
```
> cube                    ← logging the object
◂ ▸ Mesh {isObject3D: true, uuid: 'eb16b97
    e: '', type: 'Mesh', parent: Scene, …}
> cube.children           ← Object with children (hierarchical object)
◂ ▸ (2) [Mesh, AxesHelper]
```

Three.js - geometries

- Example 03:
  - Use the same **normal material** of the cube in the triangle too; since normal material uses the normal vectors to assign colors, add instruction `geometry.computeVertexNormals();` before assigning the geometry to the mesh
  - observe that sometimes you can not see the triangle: this is because its normal vectors are facing in the same direction as the camera
  - To override this default behaviour, alter the material property `side` to `THREE.DoubleSide`



```
geometry.computeVertexNormals();

material = new THREE.MeshNormalMaterial(
                { side: THREE.DoubleSide } );

let triangle = new THREE.Mesh(geometry, material);
```

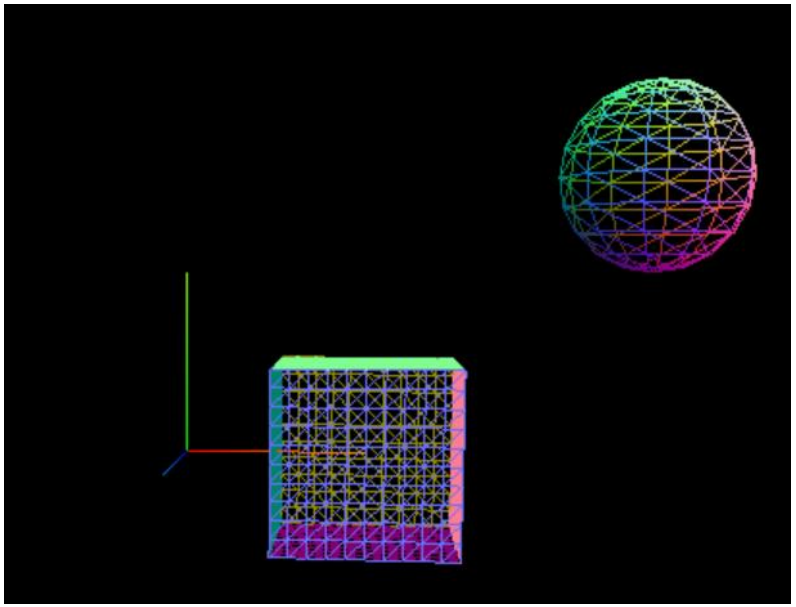


Three.js – hierarchical objects

- It is possible to build **hierarchical objects** (objects composed or built from various child objects) using class `THREE.Object3D()` and its method `.add()`
  - An object of type `THREE.Object3D` has its own coordinate system
  - It can be transformed independently from other objects
- Uses a father-son hierarchy
  - Transformations applied to the father object affect all child objects
  - It is possible to individualize transformations of the child objects; but note that those transformations are relative to the parent object
  - The technique of constructing objects by combining a hierarchy of discrete parts and animating them is known as **articulated animation**

Three.js – hierarchical objects

- **Example 04**: composite object with 2 meshes
  - A cube, positioned at (2,2,2) and can rotate along any of its axes
  - A sphere, child of the cube, and can also rotate along any of its axes; it is placed at (3,3,0) – **local coordinates**, meaning the the initial world position is (5,5,2)



cube (Mesh)

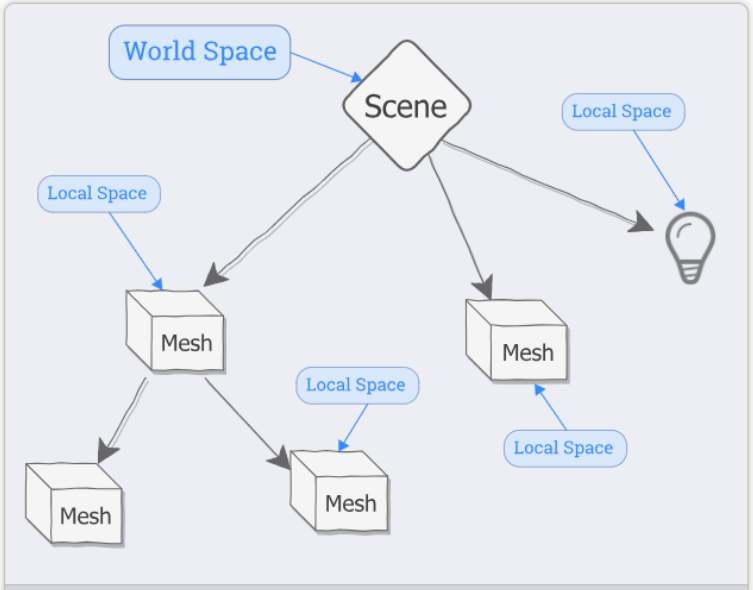
sphere (Mesh)

```
> sphere.matrixWorld
Matrix4 {elements: Array(16)}
  elements: Array(16)
    0: 1
    1: 0
    2: 0
    3: 0
    4: 0
    5: 1
    6: 0
    7: 0
    8: 0
    9: 0
    10: 1
    11: 0
    12: 5
    13: 5
    14: 2
    15: 1
```

```
> sphere.matrix
Matrix4 {elemen
  elements: Arr
    0: 1
    1: 0
    2: 0
    3: 0
    4: 0
    5: 1
    6: 0
    7: 0
    8: 0
    9: 0
    10: 1
    11: 0
    12: 3
    13: 3
    14: 0
    15: 1
```

Three.js – hierarchical objects

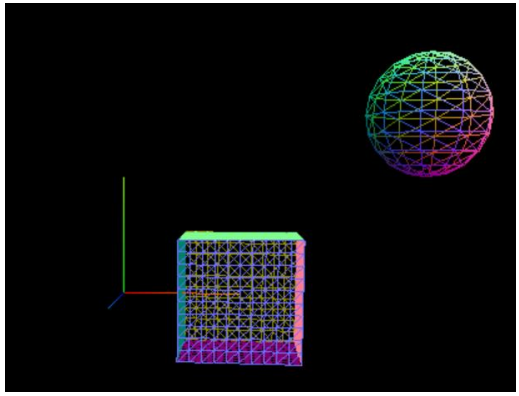
- Remember: the scene defines **world space**, and every other object defines its own **local space**
  - When we **add an object directly to the scene**, the object will move relative to world space - that is, relative to the center of the scene
  - When a mesh is created, a new **local coordinate system** is also created, with the mesh at its center
  - When an **object is added to another object**, the child object is embedded within the parent's local space
  - So, the **child objects move relatively to the parent** object's coordinate system



The scene graph is a series of embedded coordinate systems, with world space at the top

Source: https://discoverthreejs.com/book/first-steps/transformations/

Three.js – hierarchical objects



- **Example 04**: composite object with 2 meshes

Cube: local matrix = world matrix

$$\begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Sphere: local matrix

$$\begin{bmatrix} 1 & 0 & 0 & 3 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

world matrix

$$\begin{bmatrix} 1 & 0 & 0 & 5 \\ 0 & 1 & 0 & 5 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
> sphere.matrixWorld
◂ ▾ Matrix4 {elements: Array(16)}
    ▾ elements: Array(16)
        0: 1
        1: 0
        2: 0
        3: 0
        4: 0
        5: 1
        6: 0
        7: 0
        8: 0
        9: 0
        10: 1
        11: 0
        12: 5
        13: 5
        14: 2
        15: 1
```

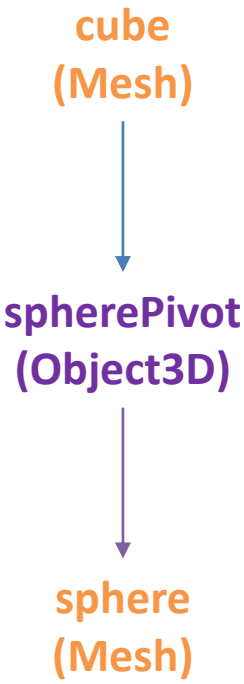
```
> sphere.matrix
◂ ▾ Matrix4 {elements:
    ▾ elements: Array(
        0: 1
        1: 0
        2: 0
        3: 0
        4: 0
        5: 1
        6: 0
        7: 0
        8: 0
        9: 0
        10: 1
        11: 0
        12: 3
        13: 3
        14: 0
        15: 1
```

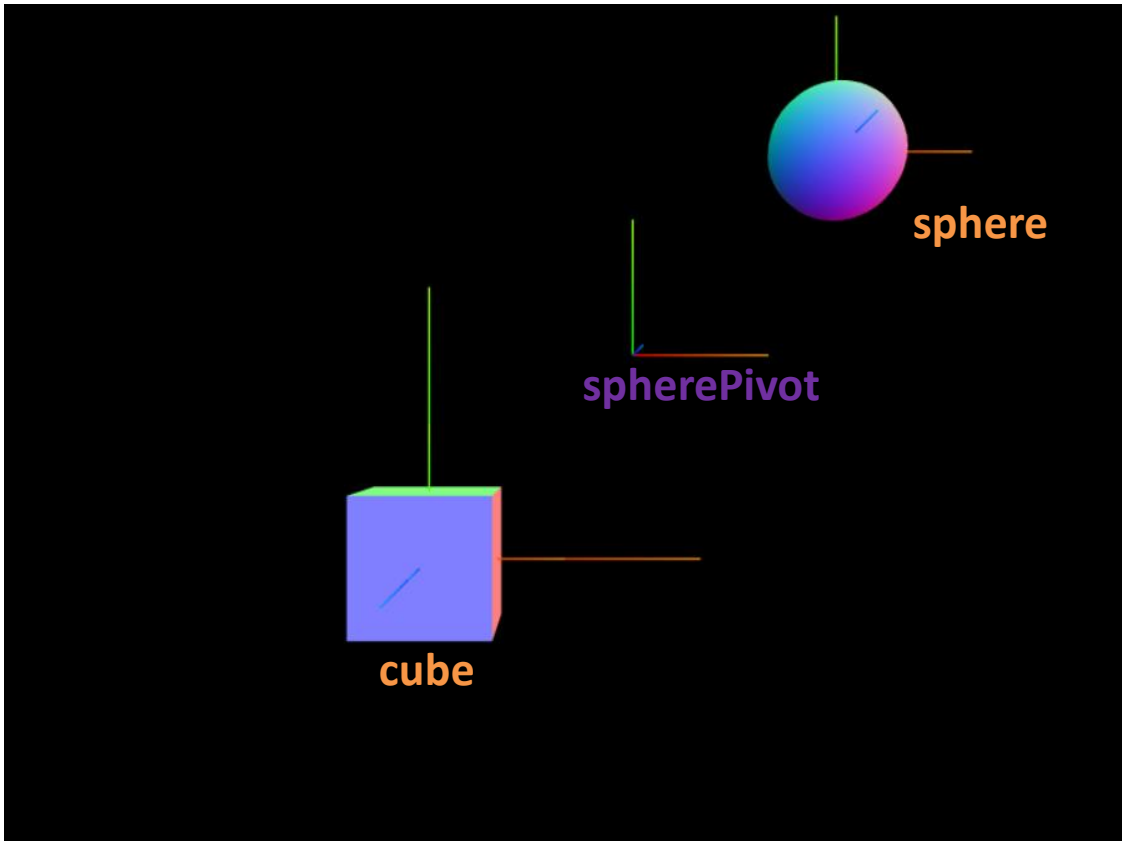
Three.js - hierarchical objects

- **Example 04**: composite object with 2 meshes
  - With this hierarchy, would it be possible to rotate cube WITHOUT affecting the sphere?
  - With this hierarchy, would it be possible to rotate the SPHERE on an axis not located on its center?

Three.js - hierarchical objects

- **Example 05**: composite object with 2 meshes and a one virtual object





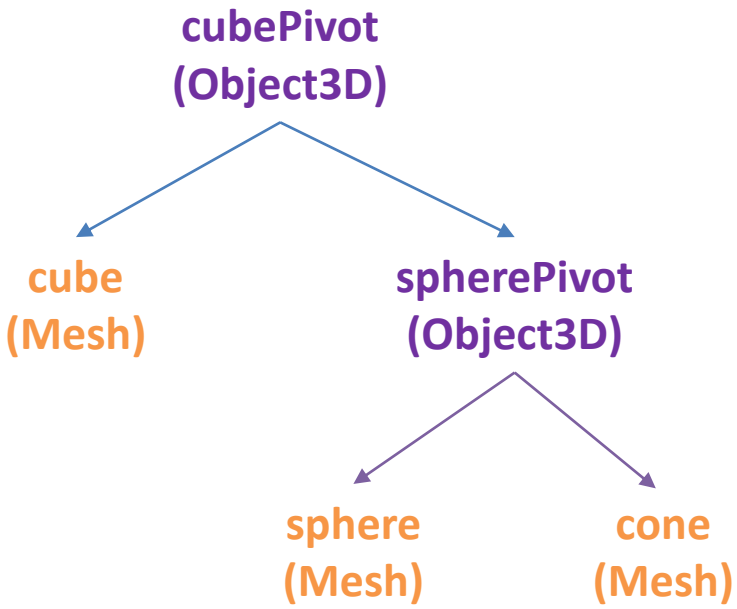
Three.js - hierarchical objects

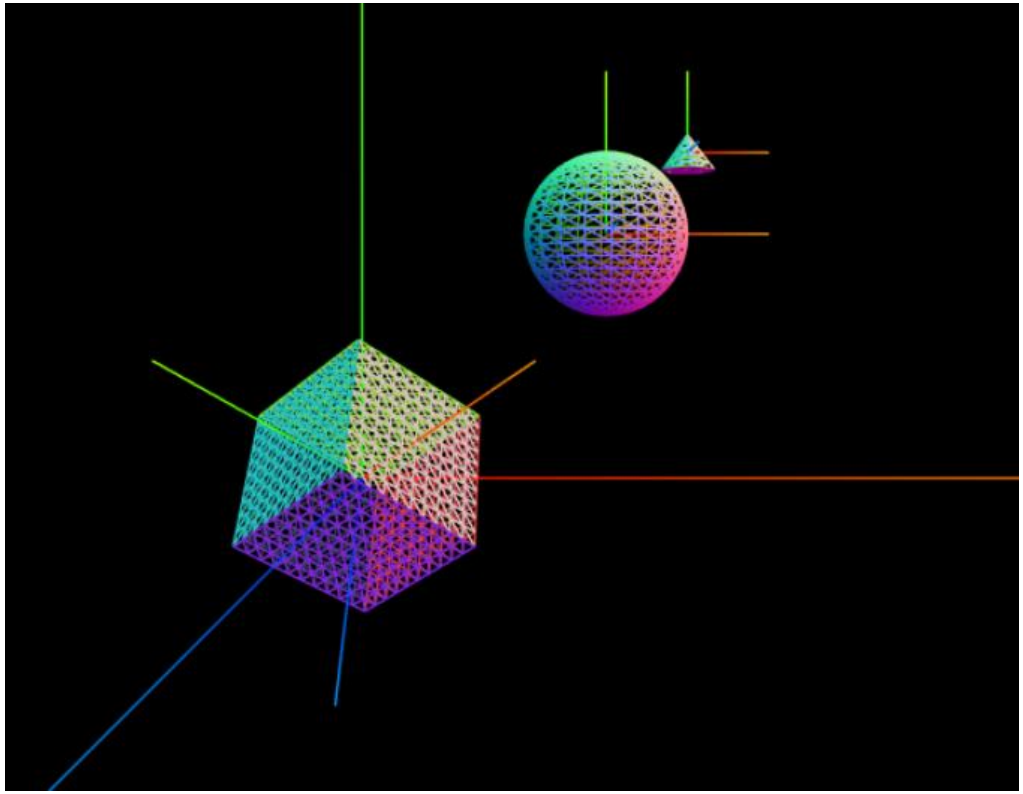
- **Example 05**: composite object with 2 meshes and a one virtual object
  - Where is the rotation axis for both meshes?
  - With this hierarchy, would it be possible to rotate cube WITHOUT affecting the sphere?
  - With this hierarchy, would it be possible to rotate the SPHERE on an axis not located on its center?

Three.js - exercises

1. Use **Example 05** as base for the following exercise:

   a) Implement the hierarchy of the graph below: all child meshes are centered at corresponding father's (pivot) object except for the cone

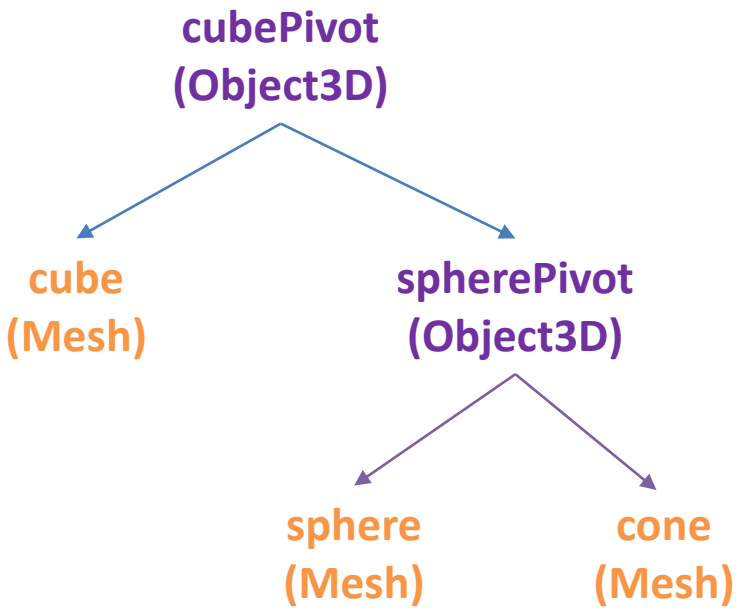


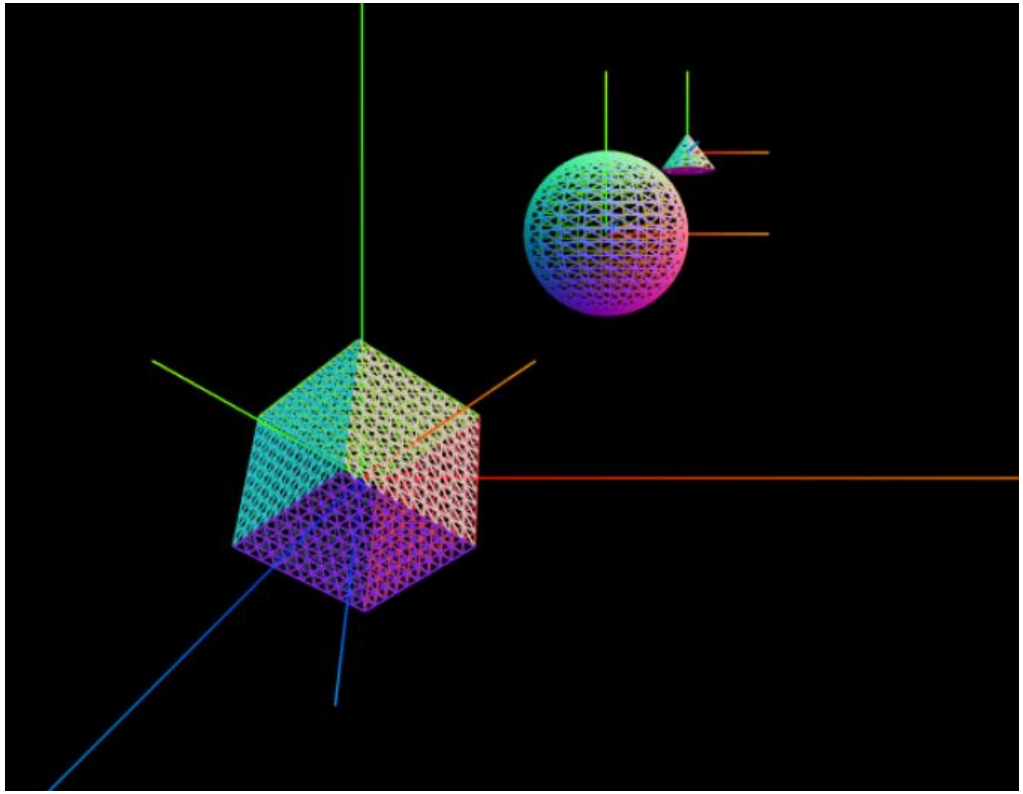


Three.js - exercises

1. Use **Example 05** as base for the following exercise:

   b) Implement the detection of keys “1” to “5” that should rotate all objects present in the scene; observe the dependencies





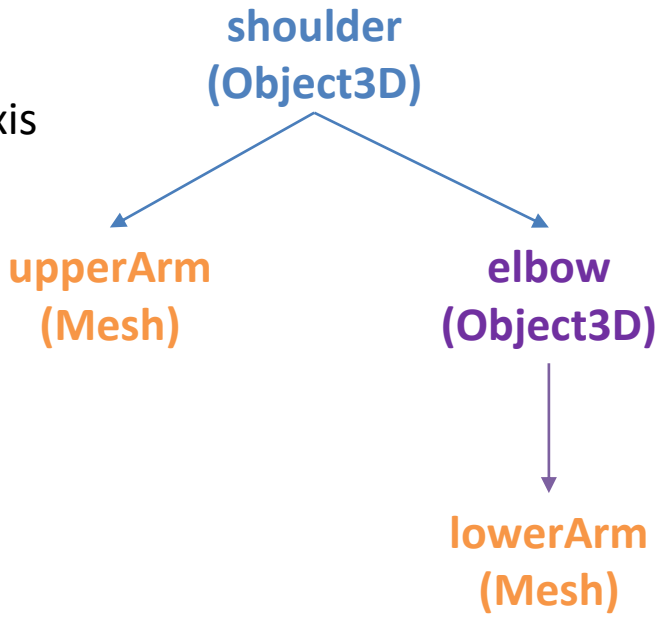
Three.js - exercises

2. **Robot Arm** – implement a RobotArm

Do not forget to animate all its parts, using keys R, S and E.

a) *Shoulder* and *Elbow* are of type `Object3D`; each one has one `BoxGeometry` mesh as a child (width = 2, height = 0.5, depth = 1)

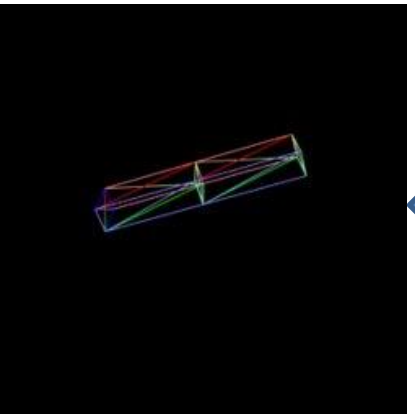
b) Key R (Shift + R): rotate arm around the X axis
Key S (Shift + S): rotate arm around the Z axis between [-90º, 90º]
Key E (Shift + E): rotate elbow around the Z axis between [0º, 145º]
Key T: alter the mesh material wireframe between values true and false



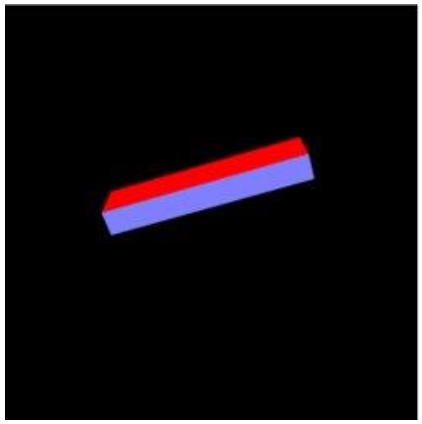
P.PORTO

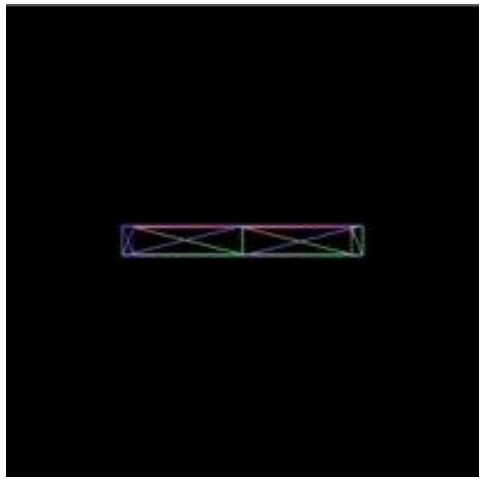
ESCOLA SUPERIOR DE MEDIA ARTES E DESIGN

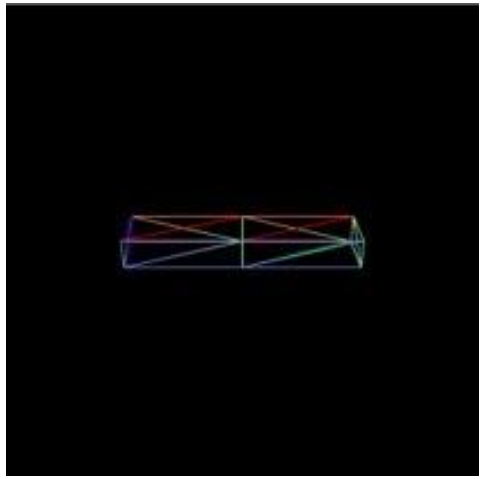
Three.js - exercises

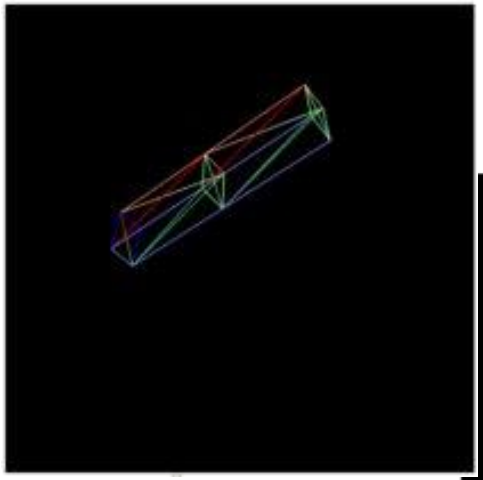


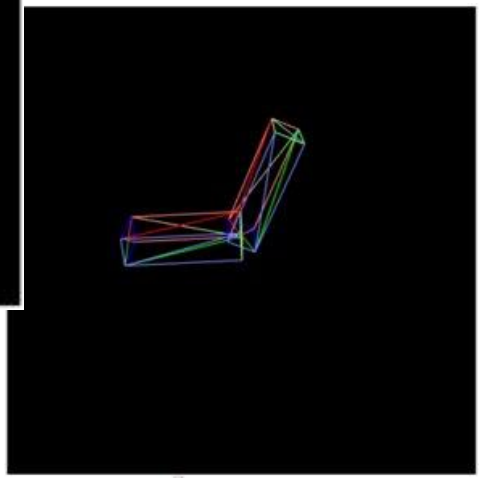
key “T”











shoulder (Object3D)

upperArm (Mesh)

elbow (Object3D)

lowerArm (Mesh)