Course on Virtual Reality & Serious Games

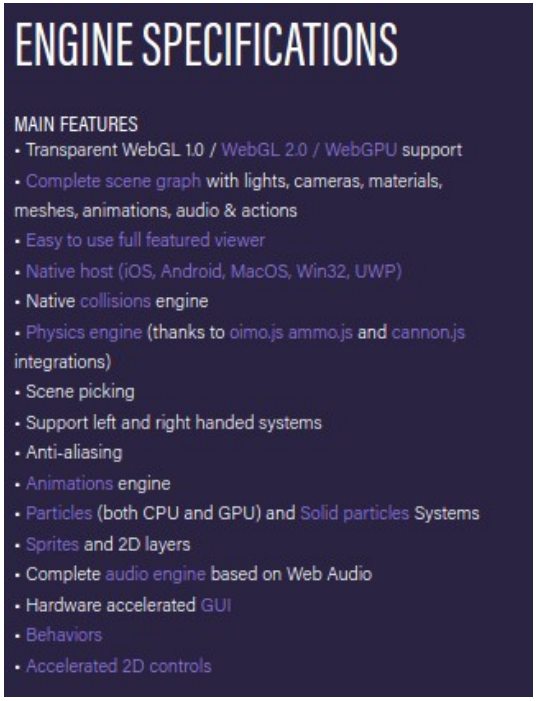
Session 10

Babylon

Dani Tost

Babylon

https://
www.babylonjs.com/



A Web rendering and game engine. It provides a layer onto webgl making it easier to program 2D and 3D graphics, animations, special effects and realistic rendering. It includes several tools such as the **playground** to test the applications and the **sandbox** to tests models.

Very good documentation at: https://doc.babylonjs.com/

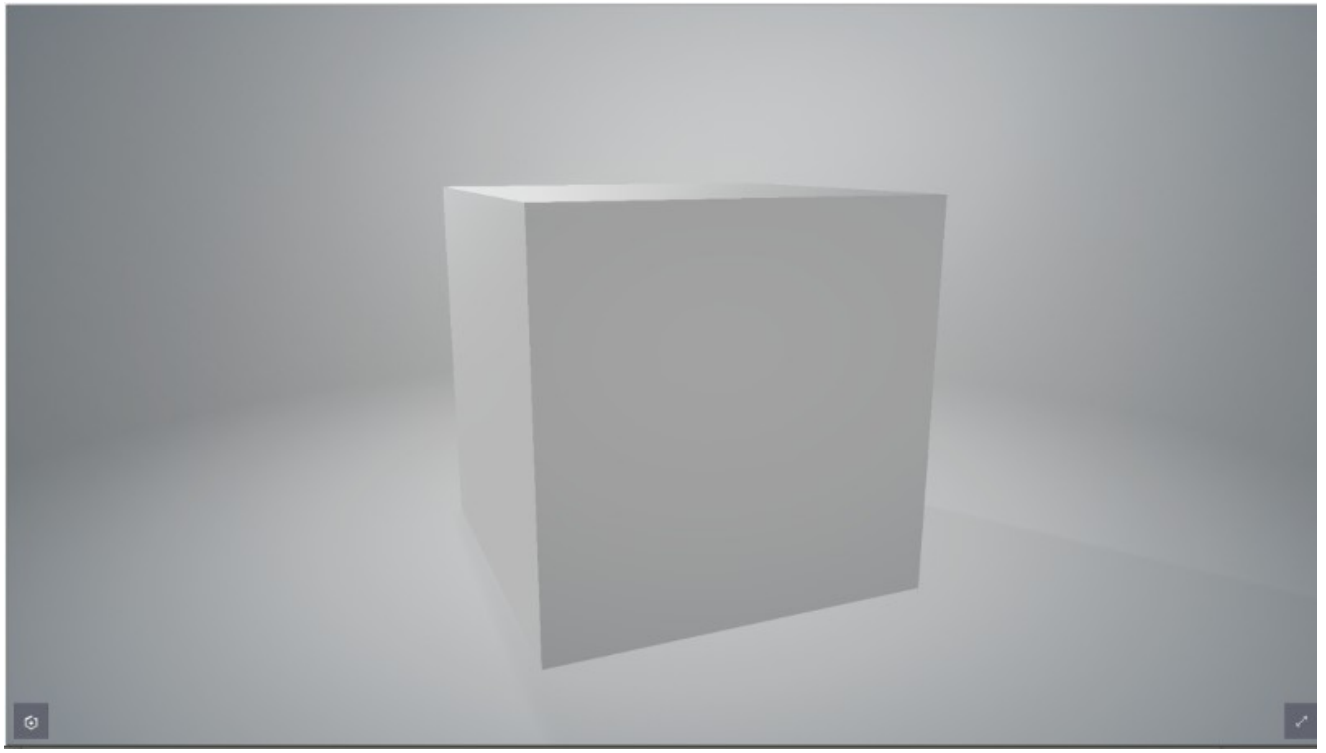
Babylon

Let test the first example. Run `ex_babylon_1.html` by clicking on it or opening a browser from the command line with the html file (e.g `x-www-browser example_bab_2.html` in linux or, in windows, `start chrome ex_babylon_1.html`).

It creates a canvas and renders in it a scene model (`box.glb`) stored in babylon's server and a script (`babylon.viewer.js`) also stored in babylon's server.

```
<html>
    <head>
      <h1> This is a first example without css</h1>
      <script src="https://cdn.babylonjs.com/viewer/babylon.viewer.js"></script>
        <meta name="viewport" content="width=device-width, initial-scale=1"></meta>
    </head>
    <body>
      <canvas id="renderCanvas"></canvas>
      <div >
        <babylon model="https://assets.babylonjs.com/meshes/box.glb"></babylon>
      </div>
    </body>
</html>
```

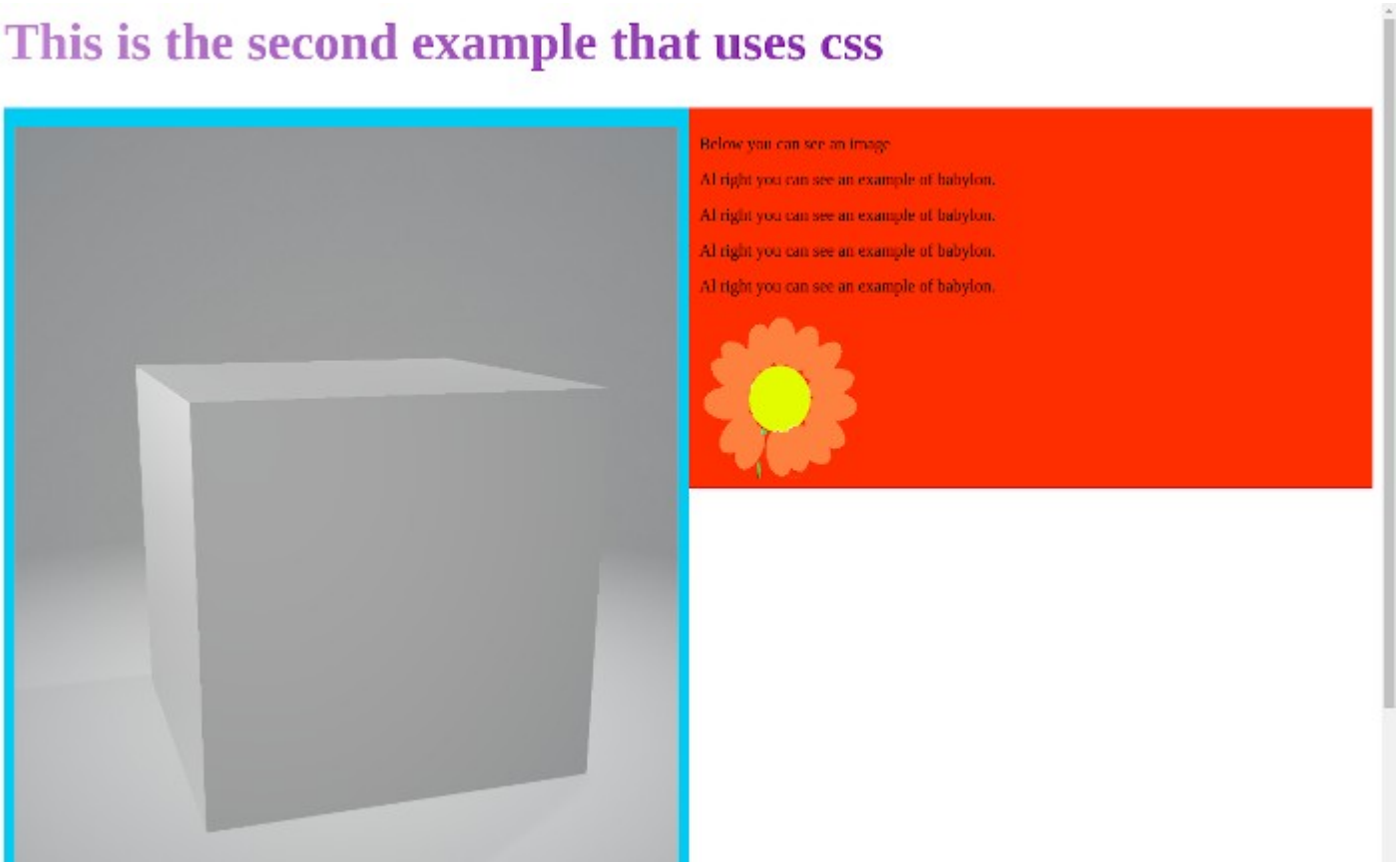
Babylon example 1



You can see a cube, lighted and rendered from a specific viewpoint. The cube rotates around axis y. These elements: camera, object and light have been created and stored as the scene `box.glb` that is now rendered.

Babylon example 2

Now, run (open in a browser) `example_babylon_2.html`. You should get something like this. It is the same as example 1, but using a css file.

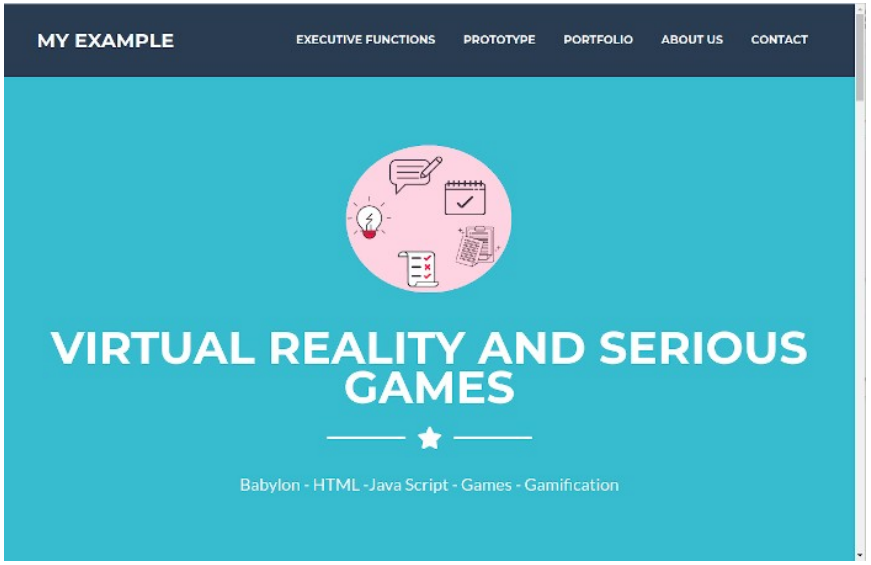


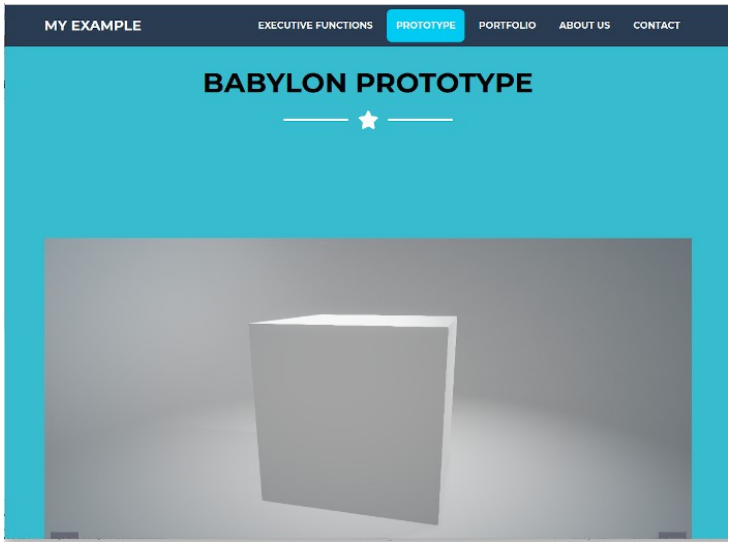
Bootstrap and babylon integration

Download a boostrap template. Customize it.

Add a section with the babylon template.

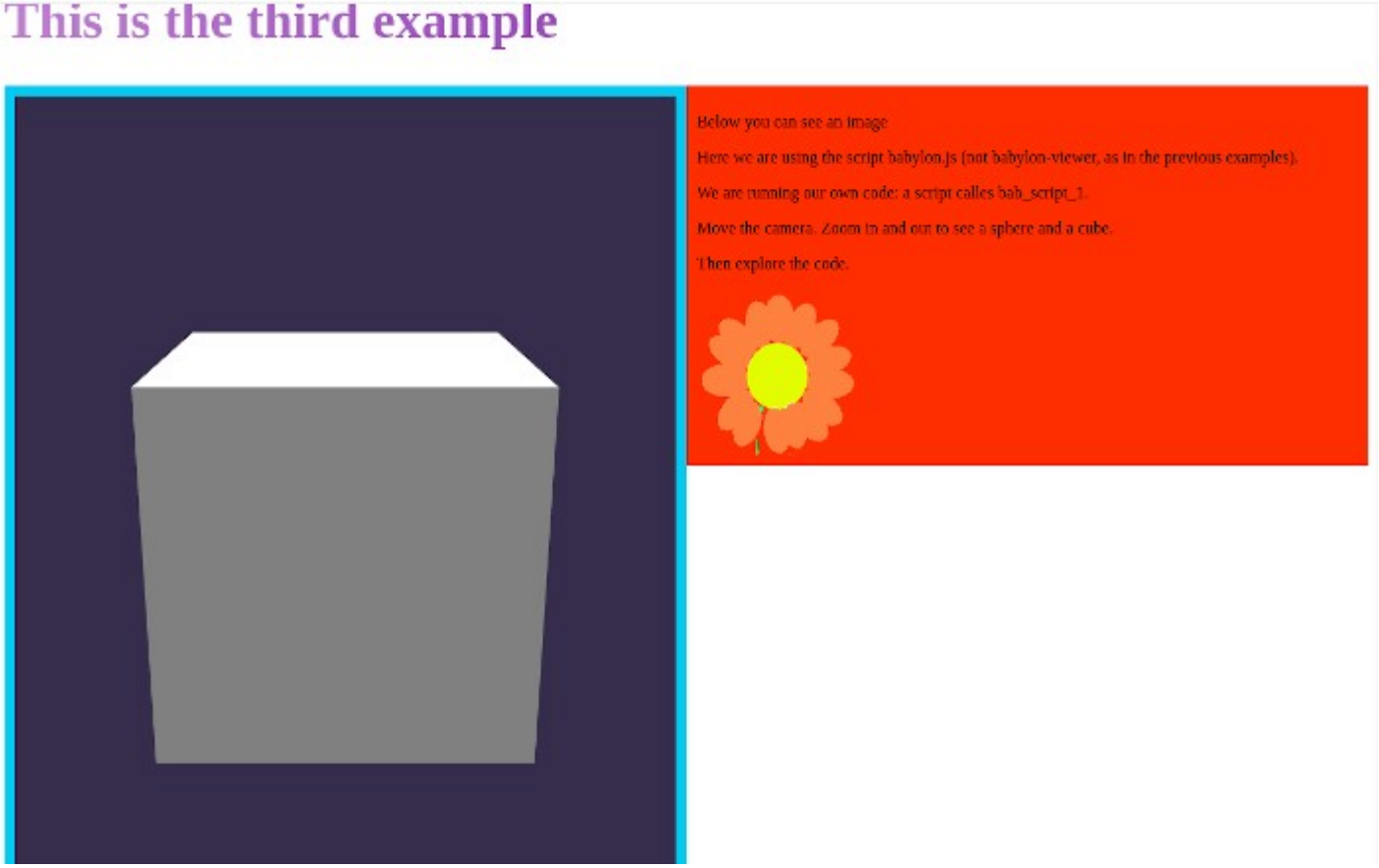
See for instance:





Babylon example 3

We'll learn how to create our own scene and render it. First, run (open in a browser) `example_babylon_3.html`. You should get something very similar to example_babylon_2. The difference is that now, you are running your own babylon script `bab_script_1.js` in the canvas.



Babylon example 3

As you'll probably remember for Medical Images, a scene is composed of three elements:
- the objects (meshes, volumes)
- the camera
- the lights

Look how we define the scene in Babylon in script `bab_script_1.js`:

```
var createScene = function createScene(){
    const scene = new BABYLON.Scene(engine);
    const camera = new BABYLON.ArcRotateCamera("camera", -Math.PI / 2, Math.PI / 2.5, 3, new BABYLON.Vector3(0, 0, 0
    camera.attachControl(canvas, true);
    const light = new BABYLON.HemisphericLight("light", new BABYLON.Vector3(0, 1, 0));
    const box = BABYLON.MeshBuilder.CreateBox("box", {});
    const sphere = BABYLON.MeshBuilder.CreateSphere("sphere",  {diameter:1.5});
    sphere.translate(new BABYLON.Vector3(0, 1, 0), 3, BABYLON.Space.LOCAL);
return scene;
}

const canvas = document.getElementById("renderCanvas"); // Get the canvas element
const engine = new BABYLON.Engine(canvas, true); // Generate the BABYLON 3D engine
const scene = createScene(); //Call the createScene function
// Register a render loop to repeatedly render the scene
engine.runRenderLoop(function () {
    scene.render();
});
// Watch for browser/canvas resize events
window.addEventListener("resize", function () {
    engine.resize();
});
```

Remember what we talked about javascript html Document Object Model?

More on JavaScript

Analyze the function `createScene` in script `bab_script_1.js`. Observe the header of the function. It is defined **as an expression**
In JS, there are 3 main ways of defining functions:

Function as a expression:

```
var name = function funcname(params) {
    // Code
    return expression;
}
```

Function as a Statement:

```
Function name(params) {
    // Code
    return expression;
}
```

Arrow functions:

```
var funcname = (params) => { //code
                             return expression;
                           }
```

Experiment in
https://playcode.io/empty_javascript

More on java script

script.js

```
1   var add1 = function(a, b) {
2       return a+b;
3   }
4   console.log(add1(3, 6));
5
6   function add2(a, b){
7     return a+b;
8   }
9   console.log(add2(8, 4));
10
11  var add3=(a, b)=>(a+b);
12  console.log(add3(7, 6));
```

Console

```
9
12
13
```

Babylon example 3

Back to the script `bab_script_1.js`. In babylon, there are various camera models. This one is based on 2 angles of rotation around a fixed target position

> This represents an orbital type of camera.
>
> This camera always points towards a given target position and can be rotated around that target with the target as the centre of rotation. It can be controlled with cursors and mouse, or with touch events.

We have also define 1 light:

> The HemisphericLight simulates the ambient environment light,
> so the passed direction is the light reflection direction, not the incoming direction.

And 2 objects: a sphere and a box that are procedural objects represented as polygonal meshes

- CreateBox
- CreateCapsule
- CreateCylinder
- CreateDashedLines
- CreateDecal
- CreateDisc
- CreateGround
- CreateGroundFrom HeightMap
- CreateHemisphere
- CreateIcoSphere
- CreateLathe
- CreateLines
- CreatePlane
- CreatePolygon
- CreatePolyhedron
- CreateRibbon
- CreateSphere
- CreateTiledGround
- CreateTorus
- CreateTorusKnot
- CreateTube
- ExtendToGoldberg
- ExtrudePolygon
- ExtrudeShape
- ExtrudeShapeCustom

Babylon example 2

Let's analyze the code.
In the header, we refer to our CSS style in file example_2.css. Try to change <h1> tag to have the text 'OHHH'... in a different color and the background also in a different color (alert: the background of the page not of the canvas!).
In the body we define a header and the canvas.
Then, we refer to a java script contained in the file myscript.js

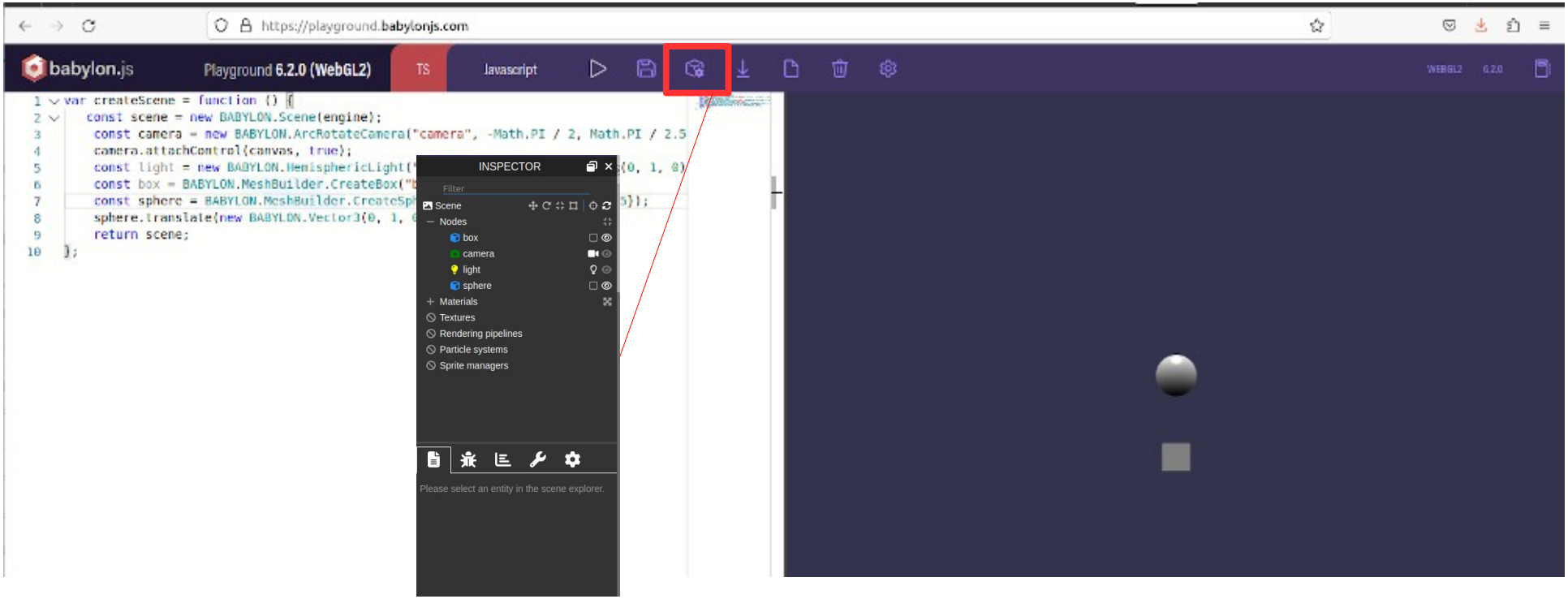
```
<html>
    <head>
      <h1> This is the second example that uses css</h1>
      <script src="https://cdn.babylonjs.com/babylon.js"></script>
      <meta name="viewport" content="width=device-width, initial-scale=1"></meta>
      <link rel="stylesheet" type="text/css" href="example2.css"/>
    </head>
    <body>
      <div class="row">
        <div class="column" style="background-color:#0dcaf0;">
          <canvas id="renderCanvas"></canvas>
        </div>

        [...]

      </div>

      [...]
    </body>
    <script src="bab_script_1.js"> </script>
</html>
```

Not the viewer anymore but babylon.

The script we are running

Babylon playground

Babylon provides a web onto which we can test the scene that we create: the Babylon.js Playground. You just have to upload the function createScene (in the expression function form) and that's it. Go to: https://playground.babylonjs.com and try. In the editor at right, paste your code. In the canvas at right, observe. Open the inspector window.



Babylon example 3

Try to modify the code:
- First, try to comment or temporally remove the line:

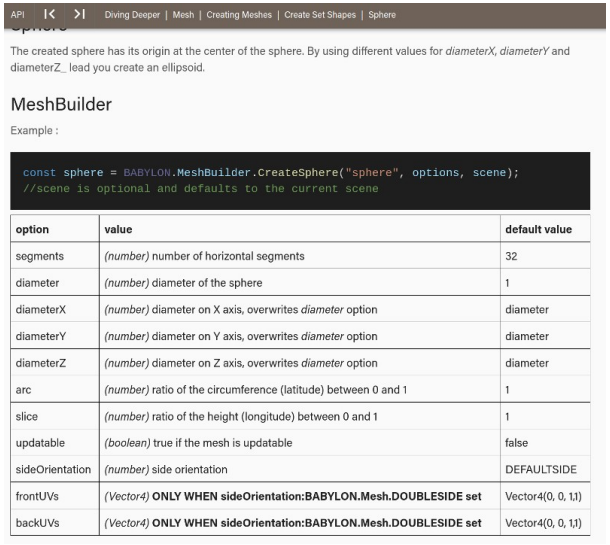
```
camera.attachControl(canvas, true);
```

What happens?

- Next, try to add others procedural objects. Check in Babylon's API documentation which primitives are available. For instance:

```
const sphere = BABYLON.MeshBuilder.CreateSphere("sphere",  {diameter:5.5});
```

https://doc.babylonjs.com/divingDeeper/mesh/creation/set



API | Diving Deeper | Mesh | Creating Meshes | Create Set Shapes | Sphere

The created sphere has its origin at the center of the sphere. By using different values for *diameterX*, *diameterY* and diameterZ_ lead you create an ellipsoid.

MeshBuilder

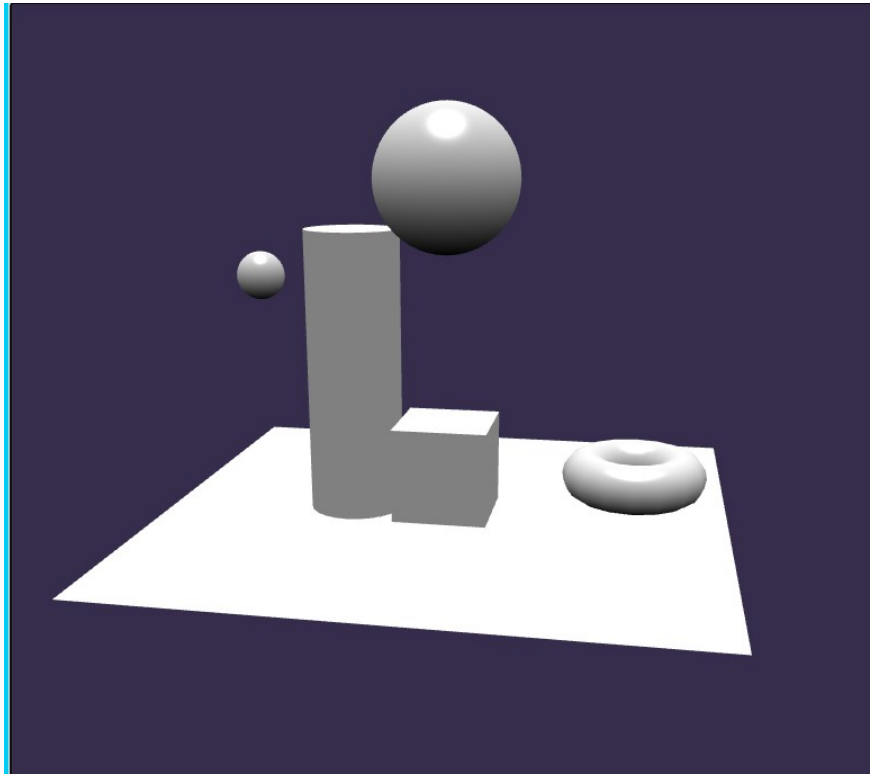
Example :

```
const sphere = BABYLON.MeshBuilder.CreateSphere("sphere", options, scene);
//scene is optional and defaults to the current scene
```

option	value	default value
segments	*(number)* number of horizontal segments	32
diameter	*(number)* diameter of the sphere	1
diameterX	*(number)* diameter on X axis, overwrites *diameter* option	diameter
diameterY	*(number)* diameter on Y axis, overwrites *diameter* option	diameter
diameterZ	*(number)* diameter on Z axis, overwrites *diameter* option	diameter
arc	*(number)* ratio of the circumference (latitude) between 0 and 1	1
slice	*(number)* ratio of the height (longitude) between 0 and 1	1
updatable	*(boolean)* true if the mesh is updatable	false
sideOrientation	*(number)* side orientation	DEFAULTSIDE
frontUVs	*(Vector4)* **ONLY WHEN sideOrientation:BABYLON.Mesh.DOUBLESIDE set**	Vector4(0, 0, 1,1)
backUVs	*(Vector4)* **ONLY WHEN sideOrientation:BABYLON.Mesh.DOUBLESIDE set**	Vector4(0, 0, 1,1)

Babylon example 3

Try to get something like that:

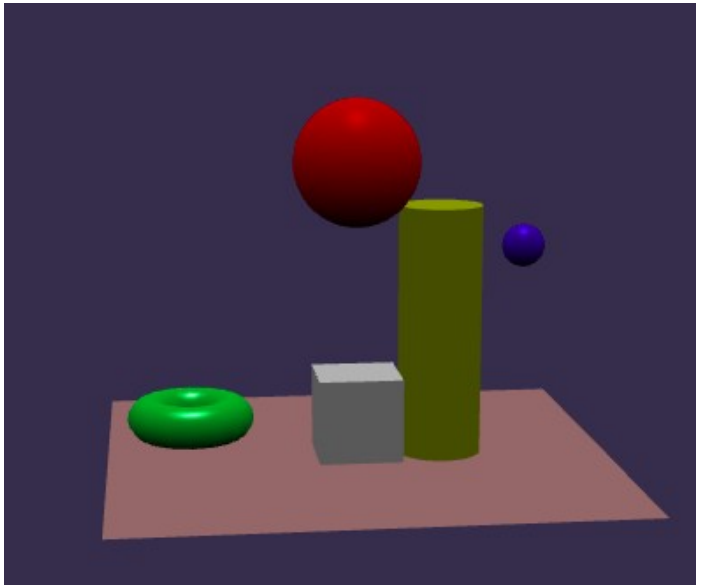


Materials

Remember what we learned about light transport and optical properties in Medical Images? Obviously the model is not different in the Babylon's implementation. See how we can create a material. Try to define different materials for your objects.

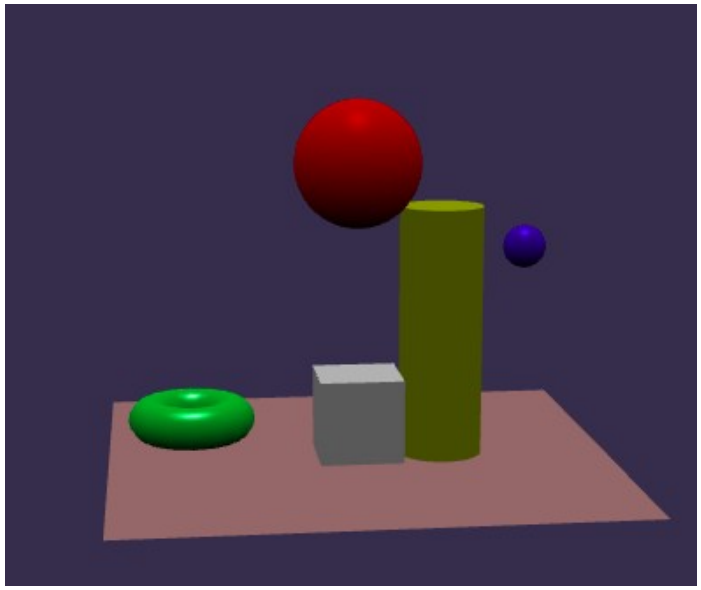
```
const myMaterial = new BABYLON.StandardMaterial("myMaterial",
scene);

myMaterial.diffuseColor = new BABYLON.Color3(1, 0, 1);
myMaterial.specularColor = new BABYLON.Color3(0.5, 0.6, 0.87);
myMaterial.emissiveColor = new BABYLON.Color3(1, 1, 1);
myMaterial.ambientColor = new BABYLON.Color3(0.23, 0.98, 0.53);

mesh.material = myMaterial;
```

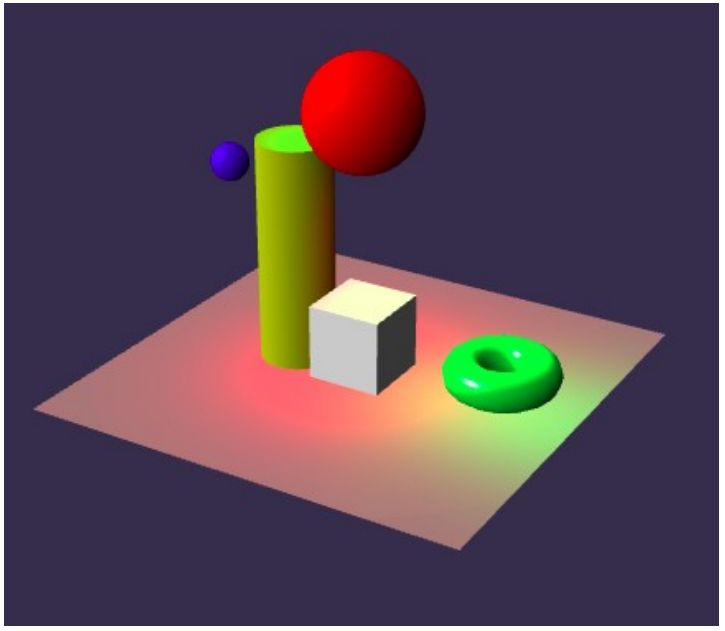
https://doc.babylonjs.com/features/featuresDeepDive/materials/using/materials_introduction



Lights

Explore the different types of lights. Add some to get a better lighting.





https://doc.babylonjs.com/features/featuresDeepDive/materials/using/materials_introduction

Textures

Textures are used to provide local variations of the optical properties.

To create a texture you have to reference an image that will be warped or projected on top of the object's surface. The texture can be static or a movie.

```
const myMaterial = new BABYLON.StandardMaterial("myMaterial", scene);
myMaterial.diffuseTexture = new BABYLON.Texture("PATH TO IMAGE", scene);
myMaterial.specularTexture = new BABYLON.Texture("PATH TO IMAGE",
scene);
myMaterial.emissiveTexture = new BABYLON.Texture("PATH TO IMAGE",
scene);
myMaterial.ambientTexture = new BABYLON.Texture("PATH TO IMAGE", scene);

mesh.material = myMaterial;
```

https://doc.babylonjs.com/features/featuresDeepDive/materials/using/materials_introduction#texture

Textures and web resources

This slide is very important!!!

We'll explain it step by step in the next slides.

1. Try to create a texture refrencing an image that you have in assets/images. See that it does not work.
2. Open the console and check the error. Do you understand? **Babylon is web-based**. It is looking for resources on the web. **It will not look for pictures in your local repository.**
3. How can we fix that? In run time, we'll upload the images in the server. By now, we'll open a local server and make our computer act as a server (although only locally):

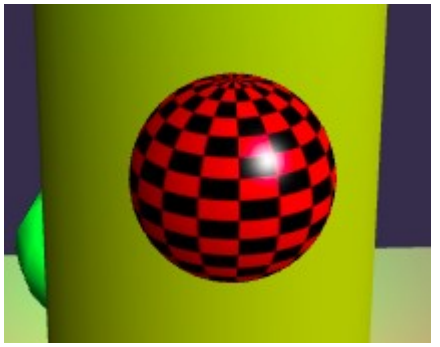
```
$ python3 -m http.server 8000
```

Open your html page in the address: localhost:8000. Click on your file.

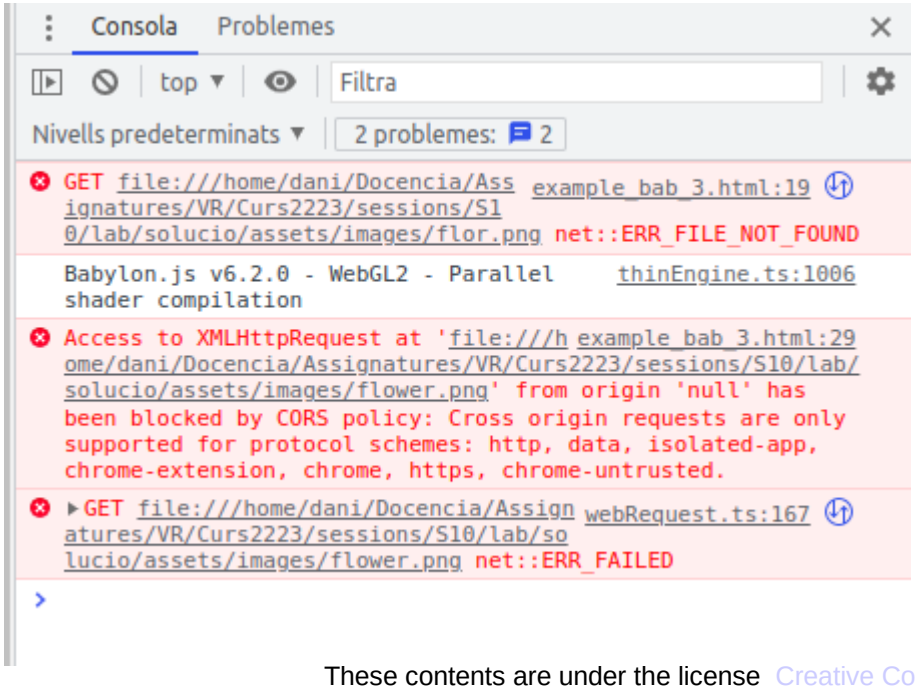
Textures

Try to add a texture to the sphere.

```
const sphere2 = BABYLON.MeshBuilder.CreateSphere("sphere2",  {diameter:0.5}, scene);
sphere2.translate(new BABYLON.Vector3(1, 1, 0), 2, BABYLON.Space.LOCAL);
var mat2 = new BABYLON.StandardMaterial("flor", scene);
mat2.diffuseTexture = new BABYLON.Texture("assets/images/flower.png", scene);
sphere2.material = mat2;
```



The chessboard pattern indicates that the texture is not found. Check in the consola:



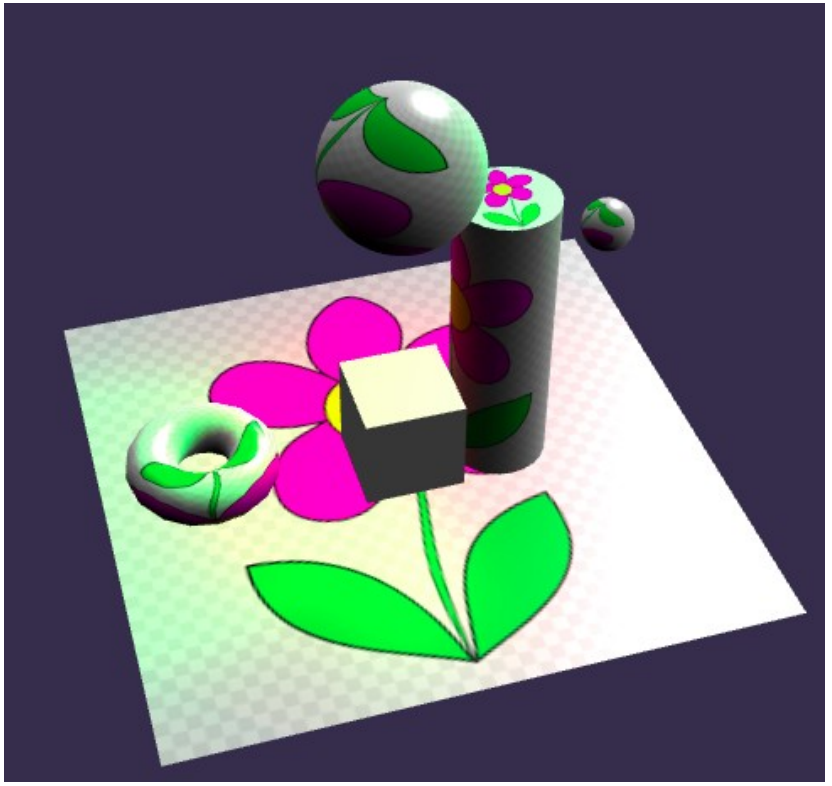
The local resource can not be retrieved!!

Textures

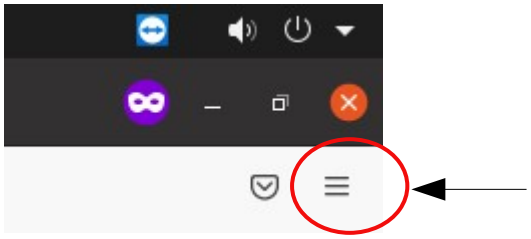
In a terminal, in your working directory open a python server:

```
$ python3 -m http.server 8000
```

In your browser, look at **localhost:8000**



Remember: If you make changes and nothing happens, it may be because the browser does not refresh and uses a static copy of the webpage. To avoid that use a **private** or **incognito** window. In the browser, deploy the menu



Thank you