Creating a scene in three.js with a 3D object and other simple examples

This is a brief introduction to three.js. It starts by setting up a scene, with a spinning cube. The first example was adapted from:
https://threejs.org/docs/index.html#manual/en/introduction/Creating-a-scene

Before starting

Before using three.js, it is needed somewhere to display it. The following HTML should be saved to a file with a copy of three.js in the js/ directory, and opened it in your browser.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>My first three.js app</title>
        <style>
            body { margin: 0; }
            canvas { width: 100%; height: 100% }
        </style>
    </head>
    <body>
        <script src="js/three.js"></script>
        <script>
            // Our Javascript will go here.
        </script>
    </body>
</html>
```

All the code below goes into the empty <script> tag.

Note: To run the example without installing three.js, instead of:

<script src="js/three.js"></script> use this code:

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

<script src="https://rawgithub.com/mrdoob/three.js/master/build/three.js"></script>

Creating the scene

To display anything with three.js, three things are necessary: scene, camera and renderer.

```
var scene = new THREE.Scene();
var camera = new THREE.PerspectiveCamera( 75, window.innerWidth /
 window.innerHeight, 0.1, 1000 );

var renderer = new THREE.WebGLRenderer();
renderer.setSize( window.innerWidth, window.innerHeight );
document.body.appendChild( renderer.domElement );
```

The scene, the camera and the renderer are set up.

There are different cameras in three.js. Let's use a **PerspectiveCamera**.

The first attribute is the **field of view**. FOV is the extent of the scene that is seen on the display at any given moment. The value is in degrees.

The second one is the **aspect ratio**. It is usual to use the width of the element divided by the height, or the image will look distorted.

The next attributes are the **near** and **far** clipping planes. Objects further away from the camera than the value of **far** or closer than **near** won't be rendered. It may be used to get better performance.

In addition to the WebGLRenderer, three.js has other renderers, often used for older browsers when users don't have WebGL support.

In addition to creating the renderer instance, it is needed to set the size at which to render the app. It's a good idea to use the width and height of the browser window. For performance intensive apps, it is possible to use **setSize** smaller values (as **window.innerWidth/2** and **window. innerHeight/2)** making the app render at half size.

setSize(window.innerWidth/2, window.innerHeight/2, false) will render the app at half resolution, given that the <canvas> has 100% width and height.

Last the **renderer** should be added to the HTML document. This is a <canvas> element the renderer uses to display the scene.

```
var geometry = new THREE.BoxGeometry( 1, 1, 1 );
var material = new THREE.MeshBasicMaterial( { color: 0x00ff00 } );
var cube = new THREE.Mesh( geometry, material );
scene.add( cube );

camera.position.z = 5;
```

To create a cube a **BoxGeometry** is used. This is an object that contains all the **vertices** and the **faces** of the cube.

In addition to the geometry, it is necessary a material to color. Three.js comes with several materials, the **MeshBasicMaterial** is simpler. All materials have properties which will be applied to them. For the moment to keep it very simple, use a color attribute of **0x00ff00** (green in rgb).

A **Mesh** is also necessary; it is an object that takes a geometry, and applies a material, which can be inserted in the scene.

By default, when **scene.add()** is called, the thing will be added to the coordinates **(0,0,0)**. This would cause both the camera and the cube to be inside each other. To avoid this, it is necessary to move the camera.

Rendering the scene

Including the code into the HTML file, nothing is seen; it is necessary to render the scene; this is done by a **render or animate loop**.

```
function animate() {
    requestAnimationFrame( animate );
    renderer.render( scene, camera );
}
animate();
```

This creates a loop that causes the renderer to draw the scene every time the screen is refreshed (e.g. 60 fps). Using **requestAnimationFrame** has a number of advantages as it pauses when the user navigates to another browser tab, not wasting processing power and battery life.

Figure 1 shows a printscreen of the result.

Animating the cube

To rotate the cube it is necessary to add the following code above the **renderer.render** call in the animate function:

```
cube.rotation.x += 0.01;
cube.rotation.y += 0.01;
```

This will be run every frame (e.g 60 fps), and give the cube a rotation animation. To move or change anything while the app is running it has to go through the animate loop. It is possible to call other functions from there, to keep the animate function with a reasonable number of lines.

The code is available in https://jsfiddle.net/mkba0ecu/ and it is possible to edit and run again. The file Example_three_1.html available in Moodle contains the working code, and it is possible to run it locally dropping the file at the browser or opening it.

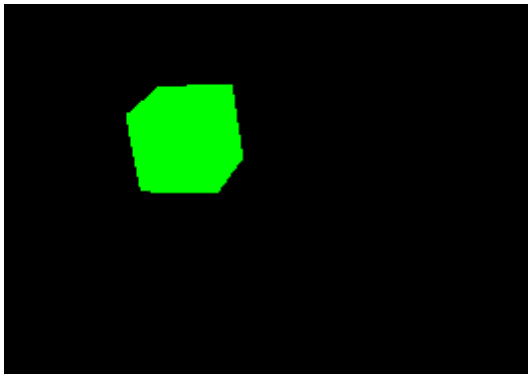

Fig.1 - Screenshot of the animated cube

Play with the code, changing:
size and proportion of the cube
color of the cube
rotation speed and axis of the cube,
camera position
background color
…

The code of this first example

(Example_three_1.html contains this code adapted to run without three.js locally)

```
<html>
    <head>
        <title>My first three.js app</title>
        <style>
            body { margin: 0; }
            canvas { width: 100%; height: 100% }
        </style>
    </head>
    <body>
        <script src="js/three.js"></script>
        <script>
            var scene = new THREE.Scene();
            var camera = new THREE.PerspectiveCamera( 75,
 window.innerWidth/window.innerHeight, 0.1, 1000 );

            var renderer = new THREE.WebGLRenderer();
            renderer.setSize( window.innerWidth, window.innerHeight );
            document.body.appendChild( renderer.domElement );

            var geometry = new THREE.BoxGeometry( 1, 1, 1 );
            var material = new THREE.MeshBasicMaterial( { color: 0x00ff00 } );
            var cube = new THREE.Mesh( geometry, material );
            scene.add( cube );

            camera.position.z = 5;

            var animate = function () {
                requestAnimationFrame( animate );

                cube.rotation.x += 0.01;
                cube.rotation.y += 0.01;

                renderer.render( scene, camera );
            };

            animate();
        </script>
    </body>
</html>
```

2D Primitives

Change the previous example to visualize a pink triangle on a grey background with vertices coordinates:
(-1,-1,0)
(1,-1,0)
(1,1,0)

To add vertices to a mesh use the following code:

```
var geometry = new THREE.Geometry();
geometry.vertices.push(new THREE.Vector3(x,y,z));
…
geometry.faces.push(new THREE.Face3(index0, index1, index2));
```

Use the function setClearColor of Renderer to set the background color to grey.

The file Example_three_2Dtriangle.html available in Moodle contains the code and Fig. 2 shows the result.

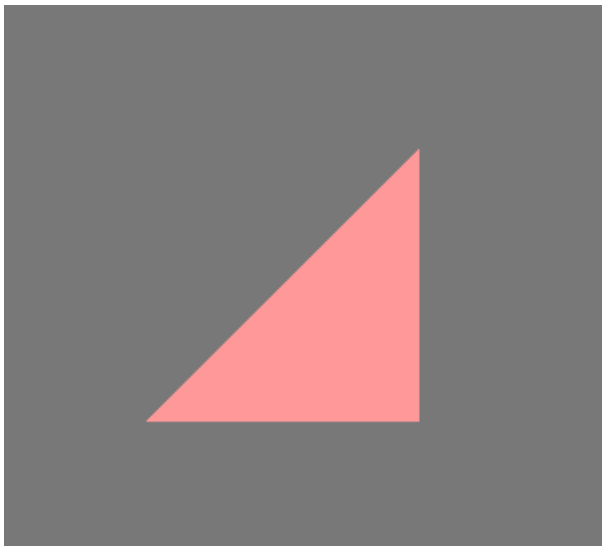

Fig. 2 - Screenshot of the result

Viewport update

Try to resize the browser window of the first example (Example_three_1.html). If you decrease the size the cube will no longer be rendered; this is due to a lack of viewport updating.
To solve this it is necessary to call a function:

```
window.addEventListener('resize', function () {
                    var w = window.innerWidth;
                    var h = window.innerHeight;
                    renderer.setSize(w, h);
                    camera.aspect = w / h;
                    camera.updateProjectionMatrix();
                    console.log("ole");
                });
                //window.addEventListener('resize', resize );
```

The file Example_cube_materials.html available in Moodle contains the code.

Materials

Change the material of the cube to make it wireframe using the wireframe property.

Look for it at https://threejs.org/docs/#api/en/materials/MeshBasicMaterial and experiment with the example shown in Fig. 3.

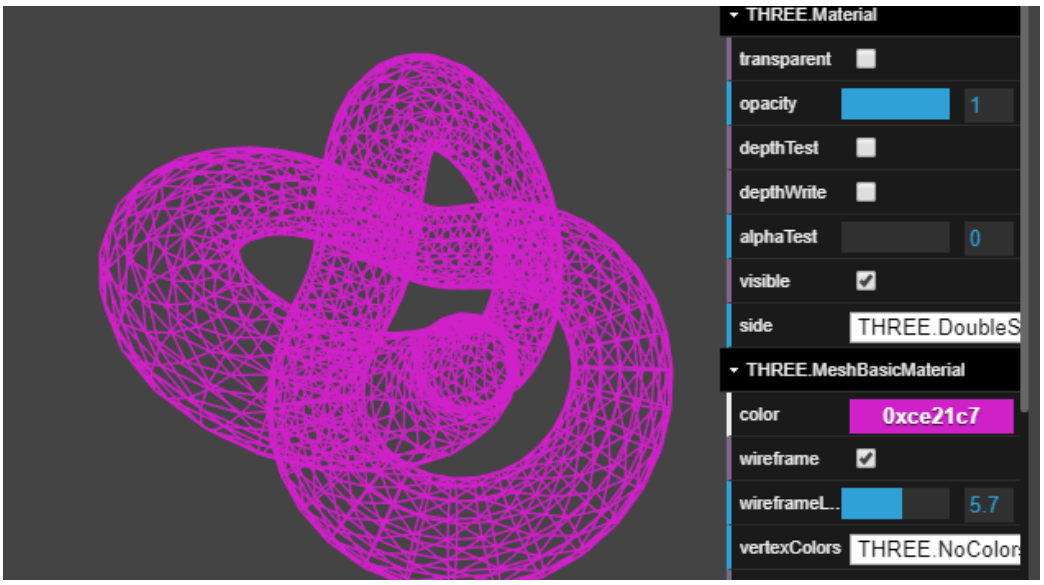

Fig.3 - Test example at https://threejs.org/docs/#api/en/materials/MeshBasicMaterial