utad

Tutorial 6 – Manual de Referência *ThreeJS*

Nos tutoriais anteriores, aprendeste a linguagem de baixo nível para programação em WebGL. No entanto, existem bibliotecas que permitem simplificar a programação em WebGL, permitindo assim uma maior facilidade no desenvolvimento de aplicações. Este tutorial foca-se na utilização de uma das mais conhecidas bibliotecas de *WebGL*, a biblioteca *ThreeJS*.

1.Objetivos de aprendizagem

O objetivo deste tutorial é o de servir como manual de referência para o desenvolvimento do trabalho prático. Lembra-te que o *ThreeJS* também dispõe de um website com muitos recursos de referência assim como vários exemplos de funcionalidades com a respetiva implementação. Neste tutorial em particular, são abordados os seguintes tópicos:

- Secção 2.1: Preparação de um projeto novo;
- Secção 2.2: Adicionar luzes à cena;
- Secção 2.3: Importação de objetos 3D e animações;
- Secção 2.4: Exploração na cena
- Secção 2.5: Interação com a cena (exemplo de navegação na mesma)
- Secção 2.6: Criação de Skybox (ambiente de fundo da cena virtual).

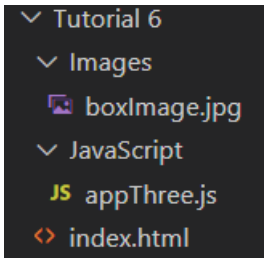
2. Tutorial

2.1. Preparação de um projeto novo

1. Copia a pasta do tutorial 5 e altera o nome para “Tutorial 6”

2. Neste tutorial, apenas queremos trabalhar com o ThreeJS pelo que vamos apagar todos os ficheiros relativos ao WebGL. Assim, elimina os seguintes ficheiros:
   - app.js
   - camara.js
   - matrizes.js
   - shaders.js
   - index.html

3. Como apagamos o ficheiro index.html e queremos que a nossa aplicação de threejs seja a *default* ao abrir o website, atualizamos o nome do ficheiro "threejs.html" para "index.html". No final da eliminação dos ficheiros e da alteração do nome do ficheiro "threejs.html" para "index.html", o teu projeto deverá ter, pelo menos, os seguintes ficheiros:



4. Abre o teu novo "index.html" e atualiza-o de forma a ficar igual ao da imagem seguinte. Este será o teu HTML base para o desenvolvimento de qualquer aplicação em ThreeJS.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>
            WebGL - Three.js
        </title>
    </head>
    <body>
        <!-- Como alguns browsers não suportam nativamente a importação dinâmica ( importmaps ),
             temos que importar este script de forma a garantir que a importação de bibliotecas
             externas de forma dinâmica -->
        <script async="" src="https://unpkg.com/es-module-shims@2.0.10/dist/es-module-shims.js"></script>

            <!-- Aqui especificamos as bibliotecas externas a serem utilizadas pela nossa aplicação-->
            <script type="importmap">
              {
                  "imports":  {
                                  "three": "https://cdn.jsdelivr.net/npm/three@0.173.0/build/three.module.js"
                              }
              }
            </script>
        <script type="module" src="./JavaScript/appThree.js"></script>
    </body>
</html>
```

5. Agora, se executares o projeto, deverás ver o resultado da versão em ThreeJS do Tutorial 5 (cubo texturizado a rodar).

6. De forma a tornar o nosso projeto mais simples, vamos limpar o ficheiro appThree.js, para depois usá-lo como base para os diferentes exemplos que irão ser apresentados neste tutorial. Para o efeito, atualiza o teu *appThree.js* para ficar idêntico ao exemplo apresentado abaixo. ***Nota:*** podes criar um ficheiro novo de raiz ou apagar código até que o ficheiro fique idêntico ao da imagem abaixo.

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

document.addEventListener('DOMContentLoaded', Start);

var cena = new THREE.Scene();
var camara = new THREE.OrthographicCamera(- 1, 1, 1, - 1, -10, 10);
var renderer = new THREE.WebGLRenderer();

var camaraPerspetiva = new THREE.PerspectiveCamera(45, 4/3, 0.1, 100);

renderer.setSize(window.innerWidth -15, window.innerHeight-80);
renderer.setClearColor(0xaaaaaa);

document.body.appendChild(renderer.domElement);

var geometriaCubo = new THREE.BoxGeometry(1, 1, 1);

var textura = new THREE.TextureLoader().load('./Images/boxImage.jpg');
var materialTextura = new THREE.MeshBasicMaterial({map:textura});

var meshCubo = new THREE.Mesh( geometriaCubo, materialTextura );
meshCubo.translateZ(-6.0);

function Start(){

    cena.add( meshCubo );

    renderer.render(cena, camaraPerspetiva);

    requestAnimationFrame(loop);
}

function loop() {

    meshCubo.rotateY(Math.PI/180 * 1);

    renderer.render(cena, camaraPerspetiva);

    requestAnimationFrame(loop);
}
```

7. Neste momento, o resultado final deverá ser o cubo com a textura do tutorial 5 a rodar no centro do ecrã

2.2. Adicionar luzes à cena

Até agora, apesar de conseguirmos ver os objetos na nossa cena, eles não estão a ser afetados pela luz. Para que seja possível tirar partido da iluminação na cena, as meshes têm que ter tipos de materiais associados que sejam afetados pela mesma (podes consultar mais sobre os diferentes materiais aqui). De seguida é ilustrado como podemos fazer com que os objetos na cena sejam afetados pela luz e como adicionar luzes à nossa cena.

1. Primeiro, atualiza o material da mesh tal como indicado na imagem abaixo.

```
18  var textura = new THREE.TextureLoader().load('./Images/boxImage.jpg');
19  //Atualizamos o tipo de material de BasicStandardMaterial para MeshStandardMaterial
20  var materialTextura = new THREE.MeshStandardMaterial({map:textura});
```

2. Se executares agora a aplicação, irás ver que o cubo ficou preto. Isto acontece porque não há luzes na cena. Assim, vamos criar uma luz na cena, tal como mostra a imagem abaixo (podes consultar mais sobre os diferentes tipos de luz aqui):

```
25  function Start(){
26
27      cena.add( meshCubo );
28
29      // Criação de luz ambiente com tom branco branco
30      //Atenção: a luz ambiente não interage com texturas pelo que temos que
31      //criar uma luz adicional para iluminar os objetos com textuas (neste caso, o cubo)
32      var luzAmbiente = new THREE.AmbientLight(0x000000);
33      cena.add(luzAmbiente);
34
35      //Cria uma luz direcional branca com intensidade 1
36      var luzDirecional = new THREE.DirectionalLight(0xffffff, 1);
37      //Define a posição da luz no espaço 3D e normaliza a direção da luz
38      luzDirecional.position.set(1, 1, 1).normalize();
39      // Adicionamos a luz à cena.
40      cena.add(luzDirecional);
41
42
43      renderer.render(cena, camaraPerspetiva);
44
45      requestAnimationFrame(loop);
46  }
```

3. Se agora executares o código, verás que o cubo já é afetado pela luz.

2.3. Importação de objetos 3D e animações

Até agora, temos usado o *ThreeJS* de forma nativa sendo que não recorremos a bibliotecas externas complexas. Funcionalidades mais avançadas como a importação de objetos exigem bibliotecas externas que podem ser importadas de forma modular.

1. Como deves ter reparado, a estratégia de importação é através de *importmaps*. Ou seja, vamos importar todas as bibliotecas no ficheiro appThree.js. Para além da biblioteca ThreeJS, vamos agora importar a biblioteca FBXLoader que permite a importação de objetos 3D em formato FBX. Para tal, atualiza o ficheiro index.html, tal como mostra a imagem abaixo (nota: tem em atenção a colocação de uma vírgula entre as duas linhas adicionadas).

```
!DOCTYPE html>
tml>
    <head>
        <title>
            WebGL - Three.js
        </title>
    </head>
    <body>
        <!-- Como alguns browsers não suportam nativamente a importação dinâmica ( importmaps ),
             temos que importar este script de forma a garantir que a importação de bibliotecas
             externas de forma dinâmica -->
        <script async="" src="https://unpkg.com/es-module-shims@2.0.10/dist/es-module-shims.js"></script>

            <!-- Aqui especificamos as bibliotecas externas a serem utilizadas pela nossa aplicação-->
            <script type="importmap">
              {
                  "imports":  {
                                  "three": "https://cdn.jsdelivr.net/npm/three@0.173.0/build/three.module.js",
                                  "FBXLoader": "https://unpkg.com/three@0.173.0/examples/jsm/loaders/FBXLoader.js"
                              }
              }
            </script>
        <script type="module" src="./JavaScript/appThree.js"></script>
    </body>
/html>
```

2. Adiciona as linhas de código abaixo no início do ficheiro *appThree.js*. (nota: a biblioteca FBXLoader é uma de muitas que permitem a importação de ficheiros 3D, podes ver mais sobre este tema aqui).

```
//Importação da biblioteca ThreeJS baseada no importmap
import * as THREE from 'three';

// Importação da biblioteca que nos permite importar objetos 3D em formato FBX baseada no importmap
import { FBXLoader } from 'FBXLoader';

document.addEventListener('DOMContentLoaded', Start);
```

3. Agora, temos de ter um objeto 3D para importar. Para efeitos deste tutorial, descarrega o objeto 3D disponível neste link: https://github.com/mrdoob/three.js/blob/master/examples/models/fbx/Samba%20Dancing.fbx?raw=true. Ser-te-á apresentada uma caixa de diálogo que te irá perguntar onde

pretendes descarregar o ficheiro. Cria uma pasta com o nome "Objetos" na raiz do teu projeto e guarda o ficheiro na pasta que acabaste de criar.

4. Agora que já as bibliotecas necessárias para a importação de objetos no ficheiro *HTML* foram importadas, é necessário modificar o ficheiro "*appThree.js*" para importar o objeto pretendido. Assim, abre o ficheiro "*appThree.js*" e adiciona as seguintes variáveis:

```
// Variável que guardará o objeto importado
var objetoImportado;

// Variável que irá guardar o controlador de animações do objeto importado
var mixerAnimacao;

// Variável que é reponsável por controlar o tempo da aplicação
var relogio = new THREE.Clock();

// Variável com o objeto responsável por importar ficheiros FBX
var importer = new THREE.FBXLoader();
```

5. Abaixo das variáveis, adiciona o código abaixo. Este código irá ser executado assim que a página comece a ser carregada e tratará de importar o objeto passado por parâmetro.

```
importer.load('./Objetos/Samba Dancing.fbx', function (object) {

    // O mixerAnimação é inicializado tendo em conta o objeto importado
    mixerAnimacao = new THREE.AnimationMixer(object);

    // object.animations é um array com todas as animações que o objeto trás quando é importado.
    // O que nós fazemos, é criar uma ação de animação tendo em conta a animação que é pretendida
    // De seguida é inicializada a reprodução da animação.
    var action = mixerAnimacao.clipAction(object.animations[0]);
    action.play();

    // object.traverse é uma função que percorre todos os filhos desse mesmo objeto.
    // O primeiro e único parâmetro da função é uma nova função que deve ser chamada para cada
    // filho. Neste caso, o que nós fazemos é ver se o filho tem uma mesh e, no caso de ter,
    // é indicado a esse objeto que deve permitir projetar e receber sombras, respetivamente.
    object.traverse(function (child) {
        if (child.isMesh) {
            child.castShadow = true;
            child.receiveShadow = true;
        }
    });

    // Adiciona o objeto importado à cena
     cena.add(object);

    // Quando o objeto é importado, este tem uma escala de 1 nos três eixos(XYZ). Uma vez que
    // este é demasiado grande, mudamos a escala deste objeto para ter 0.01 em todos os eixos.
    object.scale.x = 0.01;
    object.scale.z = 0.01;
    object.scale.y = 0.01;

    //Mudamos a posição do objeto importado para que este não fique na mesma posição que o cubo.
    object.position.x = 1.5;
    object.position.y = -0.5;
    object.position.z = -6.0;

    // Guardamos o objeto importado na variável objetoImportado.
    objetoImportado = object;
});
```

6. Se executares o código, irás ver o personagem em "T-Pose" mas sem animação. No entanto, o objeto que importarmos tem uma animação, a qual nós definimos para ser reproduzida (através do código *action.play();*) mas que não está a ser reproduzida. Isto acontece porque não estamos a atualizar o tempo de animação. Para fazê-lo, adiciona o seguinte código à função "*loop()*":

```
function loop() {

    meshCubo.rotateY(Math.PI/180 * 1);

    // Necessário atualizar o mixerAnimação tendo em conta o tempo desde o ultimo update.
    // relogio.getDelta() indica quanto tempo passou desde o último frame renderizado.
    if(mixerAnimacao) {
        mixerAnimacao.update(relogio.getDelta());
    }

    renderer.render(cena, camaraPerspetiva);

    requestAnimationFrame(loop);
}
```

7. Agora, ao abrires a aplicação, deverás ver a personagem a dançar.

Nota: existem diferentes fontes onde podes ir buscar tanto personagens 3D como animações. Um exemplo de referência é o Mixamo.

2.4. Exploração na cena

Como já foi referido, o *ThreeJS* dispõe de diversas bibliotecas que permitem acelerar o processo de desenvolvimento. Podes encontrar a listagem de bibliotecas aqui ou até mesmo exemplos de utilização dessas bibliotecas com o respetivo código-fonte aqui. Assim, em vez de desenvolveres todo o código de raiz, podes tirar proveito das mesmas e alavancar o potencial da tua aplicação. Nesta seção, vamos tirar partido da biblioteca PointerLockControls (biblioteca para a visualização da cena em primeira pessoa) para implementar a exploração da nossa cena através do rato (visualização omnidirecional) Para tal, segue os passos abaixo apresentados.

1. Atualiza o *import map* do ficheiro *index.html* para definir a referência da biblioteca PointerLockControls como ilustra a imagem abaixo (nota: tem em atenção a colocação de uma vírgula entre as duas linhas adicionadas).

```
<!DOCTYPE html>
<html>
    <head>
        <title>
            WebGL - Three.js
        </title>
    </head>
    <body>
        <!-- Como alguns browsers não suportam nativamente a importação dinâmica ( importmaps ),
             temos que importar este script de forma a garantir que a importação de bibliotecas
             externas de forma dinâmica -->
        <script async="" src="https://unpkg.com/es-module-shims@2.0.10/dist/es-module-shims.js"></script>

            <!-- Aqui especificamos as bibliotecas externas a serem utilizadas pela nossa aplicação-->
            <script type="importmap">
              {
                  "imports":  {
                                  "three": "https://cdn.jsdelivr.net/npm/three@0.173.0/build/three.module.js",
                                  "FBXLoader": "https://unpkg.com/three@0.173.0/examples/jsm/loaders/FBXLoader.js",
                                  "PointerLockControls": "https://unpkg.com/three@0.174.0/examples/jsm/controls/PointerLockControls.js"
                              }
              }
            </script>
        <script type="module" src="./JavaScript/appThree.js"></script>
    </body>
</html>
```

2. No ficheiro appThree.js, depois da linha responsável pela importação do ThreeJS, procede à importação da biblioteca PointerLockControls:

```
// Importação da biblioteca que nos permite explorar a nossa cena através do importmap
import { PointerLockControls } from 'PointerLockControls';
```

3. Antes da função Start(), adiciona o seguinte código:

```
//Definição da câmara e do renderer a serem associados ao PointerLockControls
const controls = new PointerLockControls(camaraPerspetiva, renderer.domElement)

controls.addEventListener('lock', function () {
//Possibilidade de programar comportamentos (ThreeJS ou mesmo HTML) quando
//o PointerLockControls é ativado
} );
controls.addEventListener('unlock', function () {
//Possibilidade de programar comportamentos (ThreeJS ou mesmo HTML) quando
//o PointerLockControls é ativado
} );

//Ativação do PointerLockControls através do clique na cena
//Para desativar o PointerLockControls, basta pressionar a tecla Enter
document.addEventListener(
    'click',
    function () {
        controls.lock()
    },
    false
);
```

4. Agora, se executares a aplicação e se clicares no *canvas*, a câmara deverá acompanhar o movimento do rato.

2.5. Navegação na cena

Agora, para além da exploração omnidirecional, vamos adicionar a possibilidade de navegar na cena através do teclado (navegação na cena). Para tal, temos que adicionar um mecanismo que nos permita detetar quando as diferentes teclas são premidas e adicionar movimentação à câmara. Para tal, adiciona o código da imagem abaixo antes da funcção Start(), no ficheiro *appThree.js*. Nota: as teclas são processadas através do seu ID, sendo que podes descobrir facilmente o ID de cada tecla através deste website.

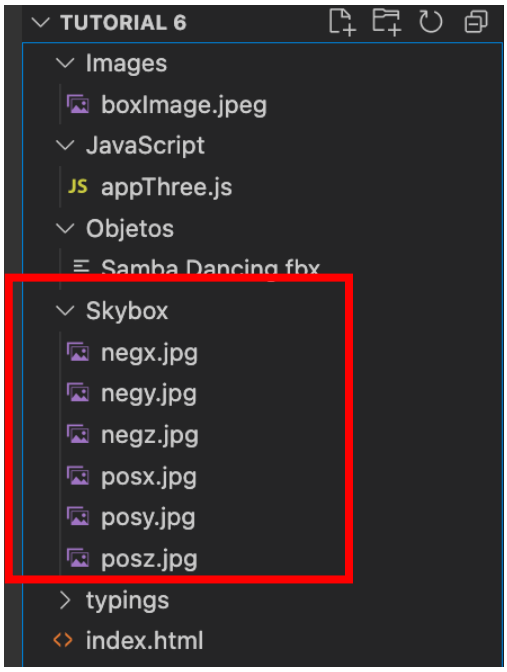
```
//Adiciona o listener que permite detetar quando uma tecla é premida
document.addEventListener("keydown", onDocumentKeyDown, false);

//Função que permite processar o evento de premir teclas e definir
//o seu respetivo comportamento
function onDocumentKeyDown(event) {
    var keyCode = event.which;
    // Comportamento para a tecla W
    if (keyCode == 87) {
        controls.moveForward(0.25)
    }
    // Comportamento para a tecla S
    else if (keyCode == 83) {
        controls.moveForward(-0.25)
    }
    // Comportamento para a tecla A
    else if (keyCode == 65) {
        controls.moveRight(-0.25)
    }
    // Comportamento para a tecla D
     else if (keyCode == 68) {
        controls.moveRight(0.25);
    }
    //Comportamento para a tecla Barra de Espaço
    else if (keyCode == 32){
        //Verificar se o cubo está presente na cena.
        //Caso esteja, removemos. Caso contrário, adicionamos.
        if(meshCubo.parent === cena){
            cena.remove(meshCubo);
        } else {
            cena.add(meshCubo);
        }
    }
};
```

Agora, se executares o código deverás conseguir navegar pela cena usando as teclas *A*, *S*, *D*, *e W*. Para tal, usamos as funções *moveForward()* e *moveRight()* da biblioteca *PointerLockControls*. Para além disso, a título de exemplo, configuramos também a tecla Barra de Espaço para adicionar ou remover o cubo à cena de forma a ilustrar que, através deste bloco de código, podemos definir o comportamento de qualquer tecla e também manipular todos os objetos da cena.

2.6. Criação de Skybox

Até agora, a nossa cena foi sempre construída num espaço "vazio". Ou seja, não há contexto ou paisagem que possa dar uma melhor contextualização do ambiente virtual aos utilizadores. Para conseguirmos fazê-lo de forma otimizada, podemos recorrer à técnica de *Skybox*. O conceito de *Skybox* consiste em criar um cubo gigante, no qual o nosso ambiente virtual vai estar contido. Nesse cubo, aplicamos um conjunto de texturas nas faces interiores do cubo de forma a criar um cenário de fundo para a nossa cena sendo que a câmara deve sempre deslocar-se dentro desse cubo. Este conceito é comum na criação de ambientes virtuais e podes encontrar diferentes packs de texturas que podes usar como Skyboxes como, por exemplo, neste website. Segue os passos seguintes para adicionar uma Skybox à aplicação:

1. Primeiro, vamos descarregar as texturas que nos vão permitir criar a Skybox. Para tal, descarrega este pack de texturas que pode ser encontrado no website acima sugerido. O pack de texturas descarregado contém 7 pastas, cada uma delas corresponde a uma Skybox diferente. Neste caso, vamos usar as texturas existentes na parta denominada "Tenerife4". Para tal, cria no teu projeto uma nova pasta com o nome "Skybox" e copia as imagens da pasta "Tenerife4" para dentro da pasta "Skybox". No final, deverás ter algo deste género (nota que pode haver alguma discrepância entre a imagem abaixo e o teu projeto pois podes ter organizado o projeto de forma diferente ou ter saltado algumas partes deste tutorial):



2. Agora, adiciona o seguinte código antes da função *Start()*, no ficheiro *appThree.js*:

```
//Carregamento das texturas para variáveis
var texture_dir = new THREE.TextureLoader().load( './Skybox/posx.jpg'); //Imagem da direita
var texture_esq = new THREE.TextureLoader().load( './Skybox/negx.jpg'); //Imagem da esquerda
var texture_up = new THREE.TextureLoader().load( './Skybox/posy.jpg'); //Imagem de cima
var texture_dn = new THREE.TextureLoader().load( './Skybox/negy.jpg'); //Imagem de baixo
var texture_bk = new THREE.TextureLoader().load( './Skybox/posz.jpg'); //Imagem da trás
var texture_ft = new THREE.TextureLoader().load( './Skybox/negz.jpg'); //Imagem de frente

// Array que vai armazenar as texturas
var materialArray = [];

// Associar as texturas carregadas ao array
materialArray.push(new THREE.MeshBasicMaterial( { map: texture_dir }));
materialArray.push(new THREE.MeshBasicMaterial( { map: texture_esq }));
materialArray.push(new THREE.MeshBasicMaterial( { map: texture_up }));
materialArray.push(new THREE.MeshBasicMaterial( { map: texture_dn }));
materialArray.push(new THREE.MeshBasicMaterial( { map: texture_bk }));
materialArray.push(new THREE.MeshBasicMaterial( { map: texture_ft }));

//Ciclo para fazer com que todas as texturas do array sejam aplicadas na parte interior do cubo
for (var i = 0; i < 6; i++)
  materialArray[i].side = THREE.BackSide;

//Criação da geometria da skybox
var skyboxGeo = new THREE.BoxGeometry( 100, 100, 100);

//Criação da mesh que vai conter a geometria e as texturas
var skybox = new THREE.Mesh( skyboxGeo, materialArray );

//Adicionar a Skybox à cena
cena.add( skybox );
```

Se executares a tua aplicação, já deverás conseguir olhar em redor e ver o ambiente de fundo na tua aplicação, tal como ilustra a imagem abaixo.