Visualização de Informação - Guião Nº 1

# Three.js - Introdução

Miguel Filipe Rodrigues Almeida de Matos Fazenda - N. 110877

Universidade de Aveiro

16 de janeiro de 2022

## Resumo

Este relatório visa documentar o processo de realização do guião, assim como também as dificuldades encontradas e suas respetivas soluções.

O guião diz respeito à introdução da biblioteca **Three.js**, focando nas funcionalidades básicas tais como os três componentes principais (**camera**, **scene**, **renderer**) e geometrias básicas.

**Keywords**

*camera*, *scene*, *renderer*, *geometrias*

## 1 Introdução

O **Three.js** é uma biblioteca e API cross-browser para **JavaScript** utilizada para **criar** e **visualizar** gráficos 3D animados num browser, com recurso a WebGL.

De maneira a facilitar a familiarização com a biblioteca, foram feitos vários exercícios com o intuito de aprender e solidificar os básicos da biblioteca.

## 2 Three.js

O Three.js de maneira a possibilitar a criação e visualização de gráficos, tem como base três ferramentas fundamentais:

- **camera**, podemos considerar a camera como se fosse o olho humano. Simboliza a posição de onde se irá ver a scene;
- **scene**, espaço tridimensional onde os gráficos são criados e armazenados;
- **renderer**, renderiza/processa os gráficos presentes na scene para serem vistos pela camera;

As **Geometry** e **Mesh** são ferramentas que permite, respetivamente, definir e personalizar figuras.

## 3 Ferramentas e Bibliotecas

**Ferramentas**

- VS Code, editor de texto que fornece todas as funcionalidades necessárias para criar o código;
- browser, para podermos visualizar os exercícios;

**Bibliotecas**

- Three.js, biblioteca que permite criar os gráficos 3D;

## 4 Código

O código realizado, é referente a três exercícios. A parte inicial dos exercícios é igual entre eles, os exercícios apenas diferem nas figuras criadas. O processo de "inicializaçõ" consiste em instanciar os três componentes principais (**camera**, **scene** e **renderer**). As figuras são compostas sempre por uma **Geometry** (forma geométrica do objeto) e uma **Mesh** (aspeto da figura).

```
//js aqui...
// creation of scene, camera and renderer
const scene = new THREE.Scene();
const camera = new THREE.PerspectiveCamera( 75, window.innerWidth / window.innerHeight, 0.1, 1000 );
const renderer = new THREE.WebGLRenderer();
```

Figura 1: inicialização dos componentes principais

### 4.1 1º Exercício

O primeiro exercício consiste na familiarização da camera, scene e renderer, e na criação de uma figura básica, nomeadamente um cubo.

O processo passa pela instanciação dos três componentes principais. Tem a caraterística única de definir uma cor para background, que é possível realizar através do renderer.

A criação do cubo é feita através de uma Geometry já existente. As caraterísticas como a cor

```
renderer.setClearColor( new THREE.Color("rgb(255, 0, 0)"));
```

Figura 2: definição da cor do background

e a opção de se ver os vértices são definidas na Mesh. Depois de criado o cubo, é sempre necessário adicioná-lo à scene para poder ser renderizado quando pretendido.

```
// definition of an object/gemoetry and camera position
const geometry = new THREE.BoxGeometry(1,1,1);
const material = new THREE.MeshBasicMaterial( { color: 0x000000, wireframe: true } );
const cube = new THREE.Mesh( geometry, material );
scene.add( cube );
camera.position.z = 5;
```

Figura 3: criação do cubo

De maneira a lidar com a alteração das dimensões da janela do browser, para que o cubo possua o comportamento pretendido, é necessário criar uma função que permita que o renderer e a camera atualizem as suas dimensões para as da janela.

```
//scene rendering
function render() {
    requestAnimationFrame(render);

    //scene animation
    cube.rotation.x += 0.01;
    cube.rotation.y += 0.01;

    renderer.render(scene, camera);
}
render();

window.addEventListener('resize', function(){
    renderer.setSize(window.innerWidth, window.innerHeight);
    camera.aspect = window.innerWidth / window.innerHeight;
    camera.updateProjectionMatrix();
});
```

Figura 4: update das dimensões da janela

O resultado final pode ser visto no fichiero "first_example.html". No entanto, segue-se uma "preview"do mesmo:

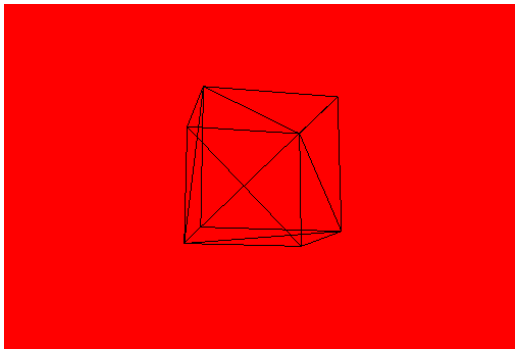

Figura 5: preview exercício 1 final

## 4.2 2º Exercício

O segundo exercício aborda primitivas bidimensionais, figuras bidimensionais que compõem as figuras tridimensionais, e adição de cores às primitivas.

As primitivas dependem de vários aspetos importantes, que são necessários para serem criadas:

- normal da figura, os pontos devem ser criados no sentido dos ponteiros do relógio de acordo com a posição da camera;
- quantidade de figuras depende do resultado da divisão entre a quantidade de pontos criada pelo tamanho máximo do conjunto de pontos;
- a quantidade de lados de uma figura depende do tamanho máximo do conjunto de pontos;
- as cores são associadas a um ponto específico através de valores "rgb";

```
const vertices2 = new Float32Array([
    //9 points for 3 triangles
    0.0,  0.0,  0.0,
    -0.7,  0.25, 0.0,
    -0.35,-1.0, 0.0,

    -0.2, 0.15, 0.0,
    0.35, 0.65, 0.0,
    -0.85, 0.9,  0.0,

    0.15,-0.95, 0.0,
    0.90,-0.7,  0.0,
    0.65, 0.10, 0.0,
]);

// 9/3 = 3 triangles that will be created
geometry2.setAttribute('position', new THREE.BufferAttribute(vertices2, 3));
```

Figura 6: criação de pontos

```
var colors2 = new Uint8Array([
    // 9 rgb values for 9 created points
    255, 0, 0,
    255, 0, 0,
    255, 0, 0,

    0, 255, 0,
    0, 255, 0,
    0, 255, 0,

    0, 0, 255,
    0, 0, 255,
    0, 0, 255,
]);

// colors are each defined for a point
geometry2.setAttribute('color', new THREE.BufferAttribute(colors2, 3, true));
```

Figura 7: definição de cores

Na figura seguinte podemos ver o resultado final:

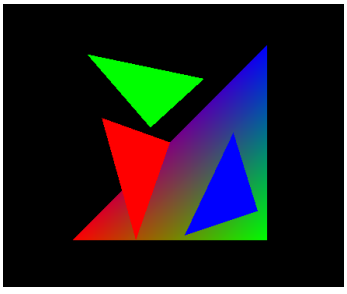

Figura 8: exercício 2 final

## 4.3 3º Exercício

O terceiro exercício dá liberdade para explorar os tipos de Geometry já existentes que a biblioteca fornece, pelo que foram criadas quatro figuras diferentes, cada uma com uma rotação diferente e parâmetros diferentes dos default.

### 4.3.1 Box

A primeira figura é uma BoxGeometry que possui as seguintes caraterísticas:

- comprimento: 1;
- largura: 2;
- altura: 3;
- posição: (X: -3, Y: 0, Z: 0);
- rotação: (X: 0.01, Y: 0.01, Z: 0);

### 4.3.2 TorusKnot

A segunda figura é uma TorusKnotGeometry que possui as seguintes caraterísticas:

- raio do torus: 0.7;
- raio do tubo: 0.1;
- segmentos tubulares: 100;
- segmentos radiais: 16;
- quantidade de vezes que a figura gira sobre o seu eixo:4;
- quantidade de vezes que a figura gira sobre o círculo no centro: 5;
- posição: (X: 0, Y: -1.5, Z: 0);
- rotação: (X: 0.02, Y: 0, Z: 0.01);

### 4.3.3 Sphere

A terceira figura é uma SphereGeometry que possui as seguintes caraterísticas:

- raio: 0.8;
- raio do tubo: 20;
- segmentos tubulares: 20;
- posição: (X: 3, Y: 0, Z: 0);
- rotação: (X: -0.01, Y: 0, Z: 0.02);

### 4.3.4 Shape

A quarta e última figura é uma ShapeGeometry. Ao contrário das outras geometrias já disponíveis, esta permite criar uma geometria com um único lado ao nosso gosto. A figura é criada através de caminhos estipulados.

```
// shape(heart)
const x = 0, y = 0;
const heartShape = new THREE.Shape();
heartShape.moveTo( x + 0.5, y + 0.5 );
heartShape.bezierCurveTo( x + 0.5, y + 0.5, x + 0.4, y, x, y );
heartShape.bezierCurveTo( x - 0.6, y, x - 0.6, y + 0.7,x - 0.6, y + 0.7 );
heartShape.bezierCurveTo( x - 0.6, y + 1.1, x - 0.3, y + 1.54, x + 0.5, y + 1.9 );
heartShape.bezierCurveTo( x + 1.2, y + 1.54, x + 1.6, y + 1.1, x + 1.6, y + 0.7 );
heartShape.bezierCurveTo( x + 1.6, y + 0.7, x + 1.6, y, x + 1.0, y );
heartShape.bezierCurveTo( x + 0.7, y, x + 0.5, y + 0.5, x + 0.5, y + 0.5 );
const shape_geometry = new THREE.ShapeGeometry(heartShape);
const shape_material = new THREE.MeshBasicMaterial({color: 0x000000, wireframe: true});
const shape = new THREE.Mesh(shape_geometry, shape_material);
shape.position.x = 0;
shape.position.y = -1.5;
scene.add(shape);
```

Figura 9: Caminhos da ShapeGeometry

### 4.3.5 Resultado final

O resultado final pode ser visto na seguinte imagen:

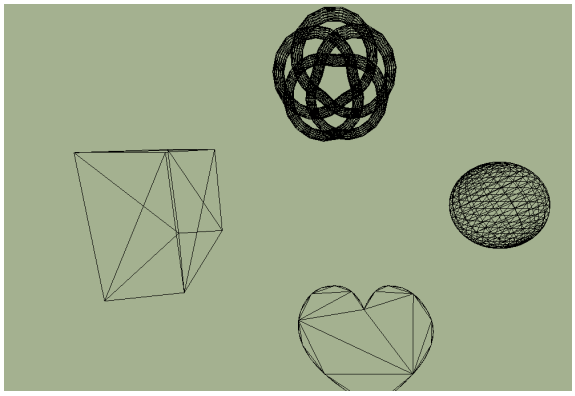

Figura 10: exercício 3 final

# Conclusão

Após a realização destes exercícios podemos afirmar que foram adquiridas competências relacionadas com:

- configuração do ambiente;
- atualização do viewport;
- visualização de geometrias e mesh's;
- personalização de mesh's e geometrias;