Relatório técnico: Primeiros testes com o Three.js.

Introdução.

Este relatório comenta sobre um estudo introdutório sobre o Three.js. O Three.js é uma biblioteca e API Javascript que cria animações em 3D usando WebGL. Neste estudo executaremos um código para criar um site num server local com um cubo tridimensional se movendo, para fins de nos familiarizar com a ferramenta.

Execução do código.

Primeiramente temos que inicializar com projeto Node digitando no Terminal:

```
npm init -y
```

Em seguida instalaremos via Terminal também o Three.js e o Vite.js que é um servidor de desenvolvimento local.

```
npm install --save three
```

```
npm install --save-dev vite
```

Se todos os comandos foram executados corretamente será criado um arquivo `package.json`. Neste arquivo dentro de `scripts` adicione:

```
"scripts": {
    "dev": "vite"
}
```

Após isso criaremos um arquivo `index.html` com o seguinte código:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>My first three.js app</title>
    <style>
      body { margin: 0; }
    </style>
  </head>
  <body>
```

```
    <script type="module" src="/main.js"></script>
  </body>
</html>
```

Adicionaremos um arquivo chamado `main.js` contendo o código:

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

E finalmente criaremos uma pasta `/public` para armazenar dados de textura, áudio e modelos 3D.

Criando o Cubo em terceira dimensão.

No próximo trecho de código será criada a cena, a câmera e o renderizador:

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

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

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

Depois será criado o cubo em si:

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

camera.position.z = 5;
```

Para renderizar a cena, adicionaremos:

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

O código seguinte irá adicionar animações ao cubo, e será escrito dentro da função `animate` criada acima, em cima do `renderer.render`.

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

O código final no arquivo `main.js` deverá ser organizado assim:

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

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

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

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

camera.position.z = 5;

function animate() {

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

    renderer.render( scene, camera );

}
```

Por final para executar o código no server local faremos:

```
npx vite
```

Aparecerá uma URL como `http://localhost:5173`, clique nela para observar o cubo em ação.

Modificações no código main.js

O código original do `main.js` foi substituído pelo seguinte código, que carrega um modelo 3D externo (arquivos .obj e .mtl) em vez de criar um cubo primitivo. Isso permite exibir a maçã fotogrametrada com mais detalhes e realismo. As bibliotecas `OBJLoader` e `MTLLoader` foram adicionadas para permitir o carregamento dos arquivos.

Para entender melhor as modificações, observe os seguintes trechos do código:

1. Importação das bibliotecas:

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

import { MTLLoader } from 'three/examples/jsm/loaders/MTLLoader.js';
import { OBJLoader } from 'three/examples/jsm/loaders/OBJLoader.js';
```

Aqui, importamos as bibliotecas `MTLLoader` e `OBJLoader` para carregar os arquivos .mtl (material) e .obj (geometria do modelo).

2. Carregamento do modelo:

```
const mtlLoader = new MTLLoader();
mtlLoader.load('apple.mtl', (materials) => {
    materials.preload();
    const objLoader = new OBJLoader();
    objLoader.setMaterials(materials);
    objLoader.load('apple.obj', (object) => {
        scene.add(object);
    });
});
```

Este trecho mostra como os arquivos .mtl e .obj são carregados. Primeiro, o `MTLLoader` carrega o arquivo .mtl, que contém as informações de material do modelo. Em seguida, o `OBJLoader` carrega o arquivo .obj, que contém a geometria do modelo. O material carregado é aplicado ao objeto antes de adicioná-lo à cena.

3. Ajustes na animação (se necessário):
Caso o modelo precise de ajustes na animação, essa parte do código seria modificada para adequar a animação ao novo modelo. Por exemplo, se o modelo não precisar de rotação, a função `animate` pode ser ajustada para:

```
function animate() {
    // Remova ou ajuste as linhas de rotação, se necessário
    // cube.rotation.x += 0.01;
    // cube.rotation.y += 0.01;

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

Com essas modificações, o código agora carrega um modelo 3D externo, permitindo uma representação mais detalhada e realista na cena.

4. Configuração da Iluminação:

```
const ambientLight = new THREE.AmbientLight(0x404040); // soft white light
scene.add(ambientLight);

const directionalLight = new THREE.DirectionalLight(0xffffff, 0.5);
directionalLight.position.set(5, 5, 5);
scene.add(directionalLight);
```

Aqui, adicionamos dois tipos de luz: `AmbientLight` e `DirectionalLight`. A `AmbientLight` ilumina todos os objetos na cena igualmente, enquanto a `DirectionalLight` emite luz em uma direção específica. A posição e a intensidade da `DirectionalLight` são ajustadas para criar um efeito de iluminação mais interessante.

5. Configuração da Câmera:

```
const camera = new THREE.PerspectiveCamera(75, window.innerWidth /
window.innerHeight, 0.1, 1000);
camera.position.z = 5;
camera.position.y = 2;
camera.lookAt(0,0,0);
```

A câmera é configurada com um ângulo de visão de 75 graus, uma proporção de tela que corresponde à janela do navegador e distâncias de renderização mínima e máxima de 0.1 e 1000, respectivamente. A posição da câmera é definida para `z = 5` e `y = 2`, e a câmera é apontada para a origem da cena (0,0,0) usando `lookAt`.

A câmera é configurada com um ângulo de visão de 75 graus, uma proporção de tela que corresponde à janela do navegador e distâncias de renderização mínima e máxima de 0.1 e 1000, respectivamente. A posição da câmera é definida para `z = 5` e `y = 2`, e a câmera é apontada para a origem da cena (0,0,0) usando `lookAt`.

Conclusão

Fontes e Referências

- Documentação do Three.js
- Documentação do Vite
- Documentação do npm Docs