THREE.JS: CÂMERAS

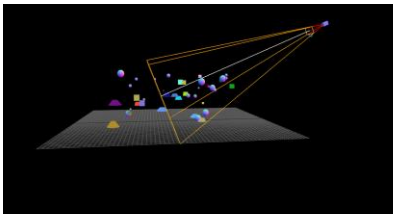
Motores Gráficos

Prof. João Paulo Lima

Universidade Federal Rural de Pernambuco
Departamento de Computação

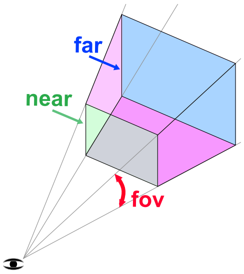
Câmera :: Projeção em Perspectiva

- Câmera mais comum no Three.js: `PerspectiveCamera`
- Visão 3D onde objetos mais distantes aparecem menores
- Define tronco de pirâmide



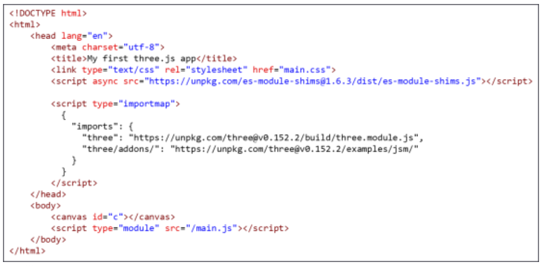
Câmera :: Projeção em Perspectiva

- `PerspectiveCamera` define tronco de pirâmide baseada em 4 propriedades:
  - `near`: plano próximo
  - `far`: plano distante
  - `fov`: ângulo do campo de vista na direção y
  - `aspect`: proporção de tela (largura ÷ altura)



Câmera :: Projeção em Perspectiva

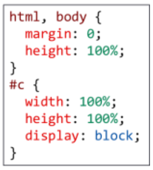
- 1. Alterar arquivo index.html conforme a seguir



```
<!DOCTYPE html>
<html>
    <head lang="en">
        <meta charset="utf-8">
        <title>My first three.js app</title>
        <link type="text/css" rel="stylesheet" href="main.css">
        <script async src="https://unpkg.com/es-module-shims@1.6.3/dist/es-module-shims.js"></script>

        <script type="importmap">
          {
            "imports": {
              "three": "https://unpkg.com/three@v0.152.2/build/three.module.js",
              "three/addons/": "https://unpkg.com/three@v0.152.2/examples/jsm/"
            }
          }
        </script>
    </head>
    <body>
        <canvas id="c"></canvas>
        <script type="module" src="/main.js"></script>
    </body>
</html>
```

Câmera :: Projeção em Perspectiva

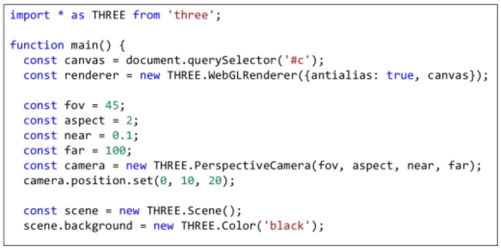
- 2. Criar arquivo main.css com conteúdo a seguir



```
html, body {
   margin: 0;
   height: 100%;
}
#c {
   width: 100%;
   height: 100%;
   display: block;
}
```

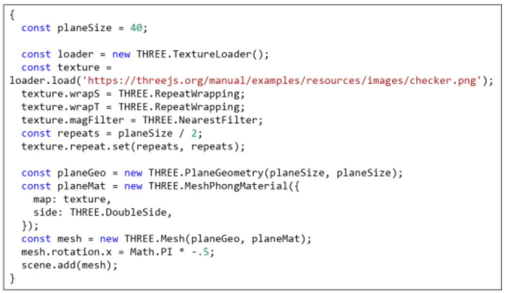
Câmera :: Projeção em Perspectiva

- 3. Alterar arquivo main.js conforme a seguir



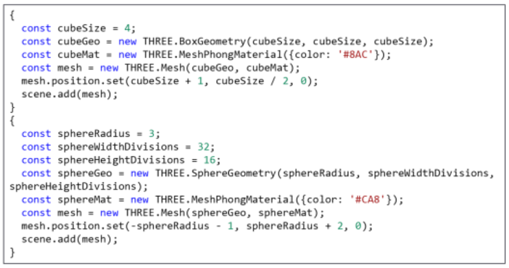
```
import * as THREE from 'three';

function main() {
  const canvas = document.querySelector('#c');
  const renderer = new THREE.WebGLRenderer({antialias: true, canvas});

  const fov = 45;
  const aspect = 2;
  const near = 0.1;
  const far = 100;
  const camera = new THREE.PerspectiveCamera(fov, aspect, near, far);
  camera.position.set(0, 10, 20);

  const scene = new THREE.Scene();
  scene.background = new THREE.Color('black');
```

Câmera :: Projeção em Perspectiva



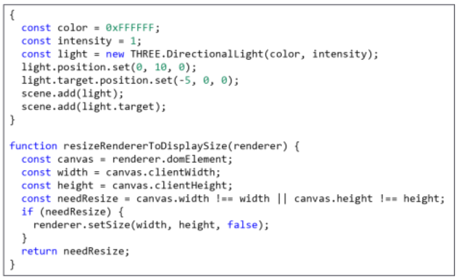
```
{
  const planeSize = 40;

  const loader = new THREE.TextureLoader();
  const texture =
loader.load('https://threejs.org/manual/examples/resources/images/checker.png');
  texture.wrapS = THREE.RepeatWrapping;
  texture.wrapT = THREE.RepeatWrapping;
  texture.magFilter = THREE.NearestFilter;
  const repeats = planeSize / 2;
  texture.repeat.set(repeats, repeats);

  const planeGeo = new THREE.PlaneGeometry(planeSize, planeSize);
  const planeMat = new THREE.MeshPhongMaterial({
    map: texture,
    side: THREE.DoubleSide,
  });
  const mesh = new THREE.Mesh(planeGeo, planeMat);
  mesh.rotation.x = Math.PI * -.5;
  scene.add(mesh);
}
```

Câmera :: Projeção em Perspectiva



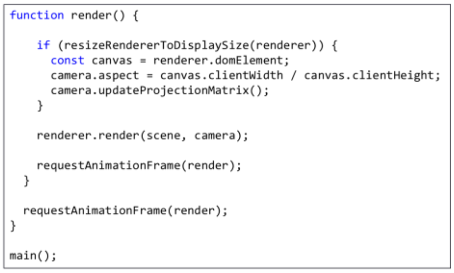
```
{
  const cubeSize = 4;
  const cubeGeo = new THREE.BoxGeometry(cubeSize, cubeSize, cubeSize);
  const cubeMat = new THREE.MeshPhongMaterial({color: '#8AC'});
  const mesh = new THREE.Mesh(cubeGeo, cubeMat);
  mesh.position.set(cubeSize + 1, cubeSize / 2, 0);
  scene.add(mesh);
}
{
  const sphereRadius = 3;
  const sphereWidthDivisions = 32;
  const sphereHeightDivisions = 16;
  const sphereGeo = new THREE.SphereGeometry(sphereRadius, sphereWidthDivisions,
sphereHeightDivisions);
  const sphereMat = new THREE.MeshPhongMaterial({color: '#CA8'});
  const mesh = new THREE.Mesh(sphereGeo, sphereMat);
  mesh.position.set(-sphereRadius - 1, sphereRadius + 2, 0);
  scene.add(mesh);
}
```

Câmera :: Projeção em Perspectiva



```
{
  const color = 0xFFFFFF;
  const intensity = 1;
  const light = new THREE.DirectionalLight(color, intensity);
  light.position.set(0, 10, 0);
  light.target.position.set(-5, 0, 0);
  scene.add(light);
  scene.add(light.target);
}

function resizeRendererToDisplaySize(renderer) {
  const canvas = renderer.domElement;
  const width = canvas.clientWidth;
  const height = canvas.clientHeight;
  const needResize = canvas.width !== width || canvas.height !== height;
  if (needResize) {
    renderer.setSize(width, height, false);
  }
  return needResize;
}
```

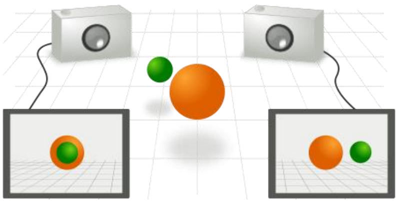
Câmera :: Projeção em Perspectiva



```
function render() {

    if (resizeRendererToDisplaySize(renderer)) {
      const canvas = renderer.domElement;
      camera.aspect = canvas.clientWidth / canvas.clientHeight;
      camera.updateProjectionMatrix();
    }

    renderer.render(scene, camera);

    requestAnimationFrame(render);
  }

  requestAnimationFrame(render);
}

main();
```

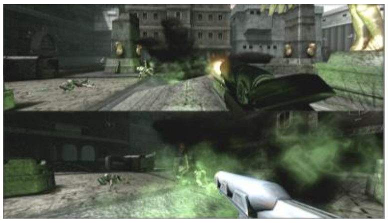
Múltiplas Câmeras

- Diferentes vistas de uma mesma cena



Múltiplas Câmeras em Janela Única

- Cada câmera virtual em diferentes porções da janela



Múltiplas Câmeras em Janela Única

- 1. Alterar elemento `<body>` de index.html

```html
<body>
    <canvas id="c"></canvas>
        <div class="split">
            <div id="view1" tabindex="1"></div>
            <div id="view2" tabindex="2"></div>
        </div>
    <script type="module" src="/main.js"></script>
</body>
```

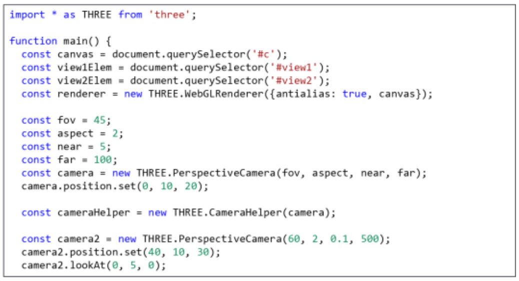
Múltiplas Câmeras em Janela Única

- 2. Adicionar linhas a seguir no fim do arquivo main.css

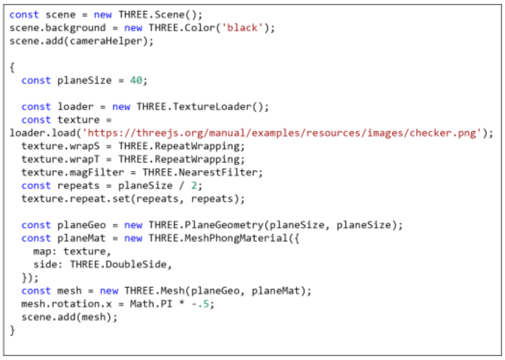
```css
.split {
  position: absolute;
  left: 0;
  top: 0;
  width: 100%;
  height: 100%;
  display: flex;
}
.split>div {
  width: 100%;
  height: 100%;
}
```

Múltiplas Câmeras em Janela Única

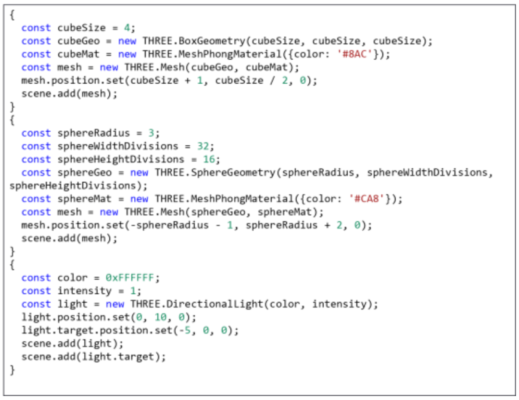
- 3. Alterar arquivo main.js conforme a seguir



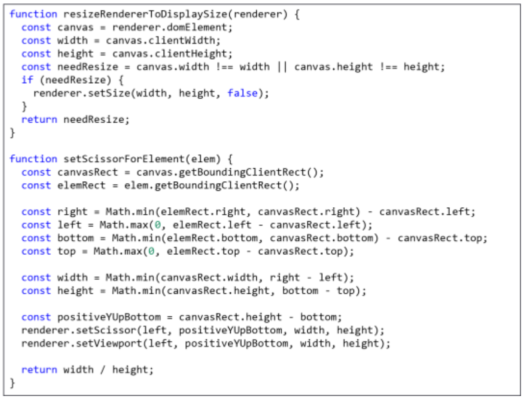
```javascript
import * as THREE from 'three';

function main() {
  const canvas = document.querySelector('#c');
  const view1Elem = document.querySelector('#view1');
  const view2Elem = document.querySelector('#view2');
  const renderer = new THREE.WebGLRenderer({antialias: true, canvas});

  const fov = 45;
  const aspect = 2;
  const near = 5;
  const far = 100;
  const camera = new THREE.PerspectiveCamera(fov, aspect, near, far);
  camera.position.set(0, 10, 20);

  const cameraHelper = new THREE.CameraHelper(camera);

  const camera2 = new THREE.PerspectiveCamera(60, 2, 0.1, 500);
  camera2.position.set(40, 10, 30);
  camera2.lookAt(0, 5, 0);
```



```javascript
const scene = new THREE.Scene();
scene.background = new THREE.Color('black');
scene.add(cameraHelper);

{
  const planeSize = 40;

  const loader = new THREE.TextureLoader();
  const texture =
loader.load('https://threejs.org/manual/examples/resources/images/checker.png');
  texture.wrapS = THREE.RepeatWrapping;
  texture.wrapT = THREE.RepeatWrapping;
  texture.magFilter = THREE.NearestFilter;
  const repeats = planeSize / 2;
  texture.repeat.set(repeats, repeats);

  const planeGeo = new THREE.PlaneGeometry(planeSize, planeSize);
  const planeMat = new THREE.MeshPhongMaterial({
    map: texture,
    side: THREE.DoubleSide,
  });
  const mesh = new THREE.Mesh(planeGeo, planeMat);
  mesh.rotation.x = Math.PI * -.5;
  scene.add(mesh);
}
```

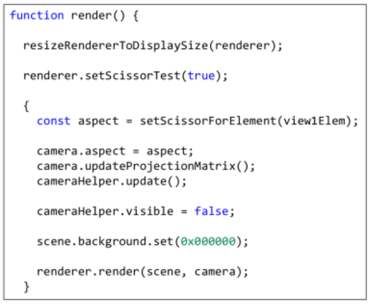


```javascript
{
  const cubeSize = 4;
  const cubeGeo = new THREE.BoxGeometry(cubeSize, cubeSize, cubeSize);
  const cubeMat = new THREE.MeshPhongMaterial({color: '#8AC'});
  const mesh = new THREE.Mesh(cubeGeo, cubeMat);
  mesh.position.set(cubeSize + 1, cubeSize / 2, 0);
  scene.add(mesh);
}
{
  const sphereRadius = 3;
  const sphereWidthDivisions = 32;
  const sphereHeightDivisions = 16;
  const sphereGeo = new THREE.SphereGeometry(sphereRadius, sphereWidthDivisions,
sphereHeightDivisions);
  const sphereMat = new THREE.MeshPhongMaterial({color: '#CA8'});
  const mesh = new THREE.Mesh(sphereGeo, sphereMat);
  mesh.position.set(-sphereRadius - 1, sphereRadius + 2, 0);
  scene.add(mesh);
}
{
  const color = 0xFFFFFF;
  const intensity = 1;
  const light = new THREE.DirectionalLight(color, intensity);
  light.position.set(0, 10, 0);
  light.target.position.set(-5, 0, 0);
  scene.add(light);
  scene.add(light.target);
}
```



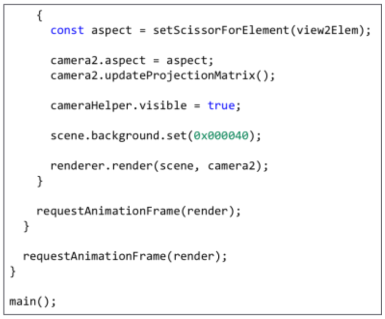
```javascript
function resizeRendererToDisplaySize(renderer) {
  const canvas = renderer.domElement;
  const width = canvas.clientWidth;
  const height = canvas.clientHeight;
  const needResize = canvas.width !== width || canvas.height !== height;
  if (needResize) {
    renderer.setSize(width, height, false);
  }
  return needResize;
}

function setScissorForElement(elem) {
  const canvasRect = canvas.getBoundingClientRect();
  const elemRect = elem.getBoundingClientRect();

  const right = Math.min(elemRect.right, canvasRect.right) - canvasRect.left;
  const left = Math.max(0, elemRect.left - canvasRect.left);
  const bottom = Math.min(elemRect.bottom, canvasRect.bottom) - canvasRect.top;
  const top = Math.max(0, elemRect.top - canvasRect.top);

  const width = Math.min(canvasRect.width, right - left);
  const height = Math.min(canvasRect.height, bottom - top);

  const positiveYUpBottom = canvasRect.height - bottom;
  renderer.setScissor(left, positiveYUpBottom, width, height);
  renderer.setViewport(left, positiveYUpBottom, width, height);

  return width / height;
}
```

Múltiplas Câmeras em Janela Única



```
function render() {

  resizeRendererToDisplaySize(renderer);

  renderer.setScissorTest(true);

  {
    const aspect = setScissorForElement(view1Elem);

    camera.aspect = aspect;
    camera.updateProjectionMatrix();
    cameraHelper.update();

    cameraHelper.visible = false;

    scene.background.set(0x000000);

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

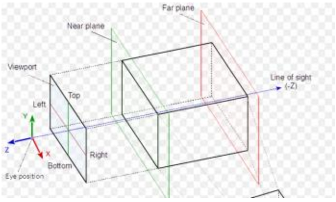
Múltiplas Câmeras em Janela Única



```
    {
      const aspect = setScissorForElement(view2Elem);

      camera2.aspect = aspect;
      camera2.updateProjectionMatrix();

      cameraHelper.visible = true;

      scene.background.set(0x000040);

      renderer.render(scene, camera2);
    }

    requestAnimationFrame(render);
  }

  requestAnimationFrame(render);
}

main();
```

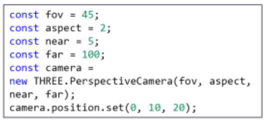
Câmera :: Projeção Ortogonal

- Segunda câmera mais comum
- Sem distorção de perspectiva
- Caixa no lugar do tronco de pirâmide
  - left
  - right
  - top
  - bottom
  - near
  - far



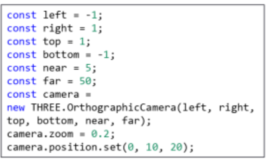
Câmera :: Projeção Ortogonal

- Em main.js, trocar código da esquerda pelo da direita

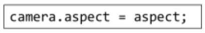


```
const fov = 45;
const aspect = 2;
const near = 5;
const far = 100;
const camera =
new THREE.PerspectiveCamera(fov, aspect,
near, far);
camera.position.set(0, 10, 20);
```

→

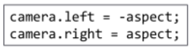


```
const left = -1;
const right = 1;
const top = 1;
const bottom = -1;
const near = 5;
const far = 50;
const camera =
new THREE.OrthographicCamera(left, right,
top, bottom, near, far);
camera.zoom = 0.2;
camera.position.set(0, 10, 20);
```



```
camera.aspect = aspect;
```

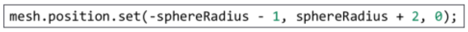
→



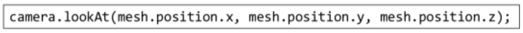
```
camera.left = -aspect;
camera.right = aspect;
```

Câmera :: Projeção Ortogonal

- Em main.js, adicionar linha de baixo após linha de cima



```
mesh.position.set(-sphereRadius - 1, sphereRadius + 2, 0);
```



```
camera.lookAt(mesh.position.x, mesh.position.y, mesh.position.z);
```

Exercício

- Mover câmera ao redor da esfera
- Dica: usar grafo de cena