Explicações dos Exemplos

Checkers

Prof. Forster

Continuando a explicação dos exemplos para a realização das atividades.

O próximo exemplo é uma cena completa com animação simples e criação procedural da geometria. Toda a geometria é descrita no código. Requer um pouco de tentativa e observação para conseguir a cena do jeito que queremos.

Mais adiante, este exemplo vai ser utilizado novamente para exemplificar a interação com o usuário.

Neste exemplo, é construído um arquivo HTML único, com o script e os estilos já embutidos.

Parte HTML do código

```
<!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>
```

Como já foi explicado anteriormente, colocamos elementos típicos na parte HTML do código.

- **DOCTYPE** - para informar alguns editores que se trata de um arquivo HTML
- **meta charset=utf-8** - para o navegador utilizar o conjunto correto de caracteres internacionais (e podermos colocá-los no código sem dar erro)
- **script src=three.js** - para carregar a biblioteca THREE.JS
- **seçao script** - para colocarmos a parte JS do código
- **style** - definição de opções de layout das tags, que poderia estar num arquivo CSS separado

```
var scene = new THREE.Scene();

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

var renderer = new THREE.WebGLRenderer({antialias: true});
renderer.setSize( window.innerWidth, window.innerHeight );
renderer.setClearColor(0x220000);
document.body.appendChild( renderer.domElement );
```

Definimos os três elemento essenciais para o THREE.JS desenhar os gráficos: cena, câmera e renderer. Utilizamos a razão de aspecto obtida das dimensões da janela do navegador.

Desta vez produzimos a canvas, que é a superfície em que a animação será gerada, no código JS ao invés de definí-la no HTML.

Definimos também uma cor de fundo para o renderizador.

```
var geometry = new THREE.BoxGeometry( 1, 1, 1 );
var material = new THREE.MeshNormalMaterial(
      { color: 0x00ffff, wireframe: false } );
var cube = new THREE.Mesh( geometry, material );
```

Criamos um cubo que ficará flutuando sobre o tabuleiro, representando uma fonte de luz. O cubo é criado através de THREE.BoxGeometry com centro na origem e dimensões (1,1,1).

Usamos como material o THREE.MeshNormalMaterial. Trata-se de uma função que colore cada ponto do cubo de acordo com a direção normal na superfície. Podemos observar que cada face do cubo tem uma cor que varia conforme este rotaciona.
Deixamos um valor de cor, mas este não é utilizado, a não ser que se troque a classe do material.

Por fim, construímos o Mesh combinando geometria e material.

Bom, agora vamos construir o tabuleiro e as peças.

Como todas as casas do tabuleiro são iguais em geometria e, também, por sua vez, as peças do tabuleiro são todas iguais em geometria, vamos manter cada geometria em uma única variável.

```
var slot_geo=new THREE.BoxGeometry(1, 1, 0.2)

var piece_geo = new THREE.CylinderGeometry(
       0.46, 0.46, 0.4, 32)
```

slot_geo é a geometria da casa do tabuleiro (vão ser 64 delas).
piece_geo é a geometria de cada peça do tabuleiro (vão ser 32).

No próximo slide, criamos esses elementos.

```
for (var i=0; i<8; i++)
  for (var j=0; j<8; j++)
  {
    var slot= new THREE.Mesh( slot_geo,
          new THREE.MeshLambertMaterial(
            { color: (i+j)%2?0xffff88:0x665522 } ));
        slot.position.x= i;
        slot.position.y= j;
        scene.add(slot);

        if((i+j+1)%2 && (j==0 || j==1|| j==2 || j==5||
                         j==6 || j==7))
        {
          var piece = new THREE.Mesh(piece_geo,
             new THREE.MeshLambertMaterial(
             { color: j<5?0xffffff:0x334411 } ));
          piece.position.x= i;
          piece.position.y= j;
          piece.position.z=0.4
          piece.rotation.x= Math.PI/2.0;
          scene.add(piece)
        }

  }
```

Obervando parte por parte o código:
fazemos um loop pelas 8x8 casas do tabuleiro:

```
for (var i=0; i<8; i++)
  for (var j=0; j<8; j++)
```

A cor da casa depende se é uma casa par ou uma casa ímpar:

```
    var slot= new THREE.Mesh( slot_geo,
          new THREE.MeshLambertMaterial(
            { color: (i+j)%2?0xffff88:0x665522 } ));
```

Todas as casa vão utilizar a mesma geometria `slot_geo` e o material de reflexão de Lambert, com cor `0XFFFF88` para casa ímpares e cor 0XFFFF88 para casas pares.

Quanto à peças, estas só são colocadas nas casas pares e apenas nas 3 primeiras e 3 últimas fileiras.

```
        if((i+j+1)%2 &&
           (j==0 || j==1|| j==2 || j==5|| j==6 || j==7))
```

A cor delas depende de que lado do tabuleiro estão (fileira j)

```
      var piece = new THREE.Mesh(piece_geo,
        new THREE.MeshLambertMaterial(
        { color: j<5?0xffffff:0x334411 } ));
```

Posicionamento das casas

```
        slot.position.x= i;
        slot.position.y= j;
        scene.add(slot);
```

Posicionamento das peças

```
          piece.position.x= i;
          piece.position.y= j;
          piece.position.z=0.4
          piece.rotation.x= Math.PI/2.0;
          scene.add(piece)
```

Cada cas tem 1m, então báiscamente atribui-se os valores de i e j.

A rotação é para "deitar" o cilindro.

Aqui criamos a fonte de luz.
Trata-se de uma fonte de luz puntual, de forma que se alterarmos a posição, veremos que o ponto mais brilhante do tabuleiro se desloca.

O atributo `target` não faz parte da definição da luz puntual, mas é útil no caso de testarmos uma luz direcional.

A luz é branca 0xFFFFFF com intensidade 1.0

```
var light =  new THREE.PointLight( 0xffffff, 1.0 );
light.position.set(4,4,2);
light.target=cube;

cube.position.set(4,4,2);

scene.add( cube );
scene.add(light);
```

Definimos aqui os parâmetros da câmera.
Inicialmente haviamos definido a transformação de projeção da câmera com `new THREE.PerspectiveCamera( 75, window.innerWidth / window.innerHeight, 0.1, 1000 );` onde 75 é o fov (field of view), em seguida o aspecto e as coordenadas dos planos de corte mínimo 0.1 e máximo 1000.

Isto significa que a projeção perspectiva vai transformar o espaço visualizado (uma pirâmide com ângulo definido pelo fov e correção pelo aspecto dado) em um paralelepípedo e projetá-lo no plano de imagem. Os pontos com coordenada z inferior a 0.1 são cortados (muito próximos) e também os de z superior a 1000 (muito distantes).

Aqui definimos a visualização através da posição da câmera:

`camera.position`

da direção para cima:

`camera.up`

e da direção de observação:

`camera.lookAt`

```
camera.position.x = 4;
camera.position.z = 5;
camera.position.y = -3;
camera.up = new THREE.Vector3(0,0,1);
camera.lookAt(new THREE.Vector3(4, 4, 0));
```

A matriz de visualização é calculada conforme visto na teoria.

Segue o código da animação da cena do tabuleiro. Na verdade, há a rotação do cubo e uma movimentação da câmera.

```
var t=0;

function animate() {
  requestAnimationFrame( animate );
  cube.rotation.x += 0.01; cube.rotation.y += 0.01;
  renderer.render( scene, camera );
  t=t+0.01;
  camera.position.x = 5*Math.cos(t)+4;
  camera.position.y = 5*Math.sin(t)+4;
  camera.position.z = 5+Math.sin(t/2.0)+2*Math.sin(t*2.0);
  camera.up = new THREE.Vector3(0,0,1);
  camera.lookAt(new THREE.Vector3(4, 4, 0));
}

animate();
```

Explicando agora por partes:

```
var t=0;

function animate() {
  requestAnimationFrame( animate );
  renderer.render( scene, camera );
  t=t+0.01;
...

}

animate();
```

A variável t representa o tempo e é incrementada de 0.01 a cada quadro. Veja que existem outras formas para conseguir uma medida precisa do tempo, sendo esta uma simplificação.

A função animate() é chamada uma vez no script e através do `requestAnimationFrame` avisa ao navegador que deve ser chamada novamente na próxima atualização do quadro.

Aqui atualizamos a rotação do cubo, incrementado seus ângulos em relação aos eixos x e y. A matriz de transformação é calculada automaticamente

```
...
  cube.rotation.x += 0.01; cube.rotation.y += 0.01;
  renderer.render( scene, camera );
...
```

O movimento da câmera descreve uma trajetória em 3 dimensões, mas esta se direciona sempre à posição 4,4,0.

```
...
  camera.position.x = 5*Math.cos(t)+4;
  camera.position.y = 5*Math.sin(t)+4;
  camera.position.z = 5+Math.sin(t/2.0)+2*Math.sin(t*2.0);
  camera.up = new THREE.Vector3(0,0,1);
  camera.lookAt(new THREE.Vector3(4, 4, 0));
...
```

Com isso, conseguimos construir a animação do tabuleiro de damas.