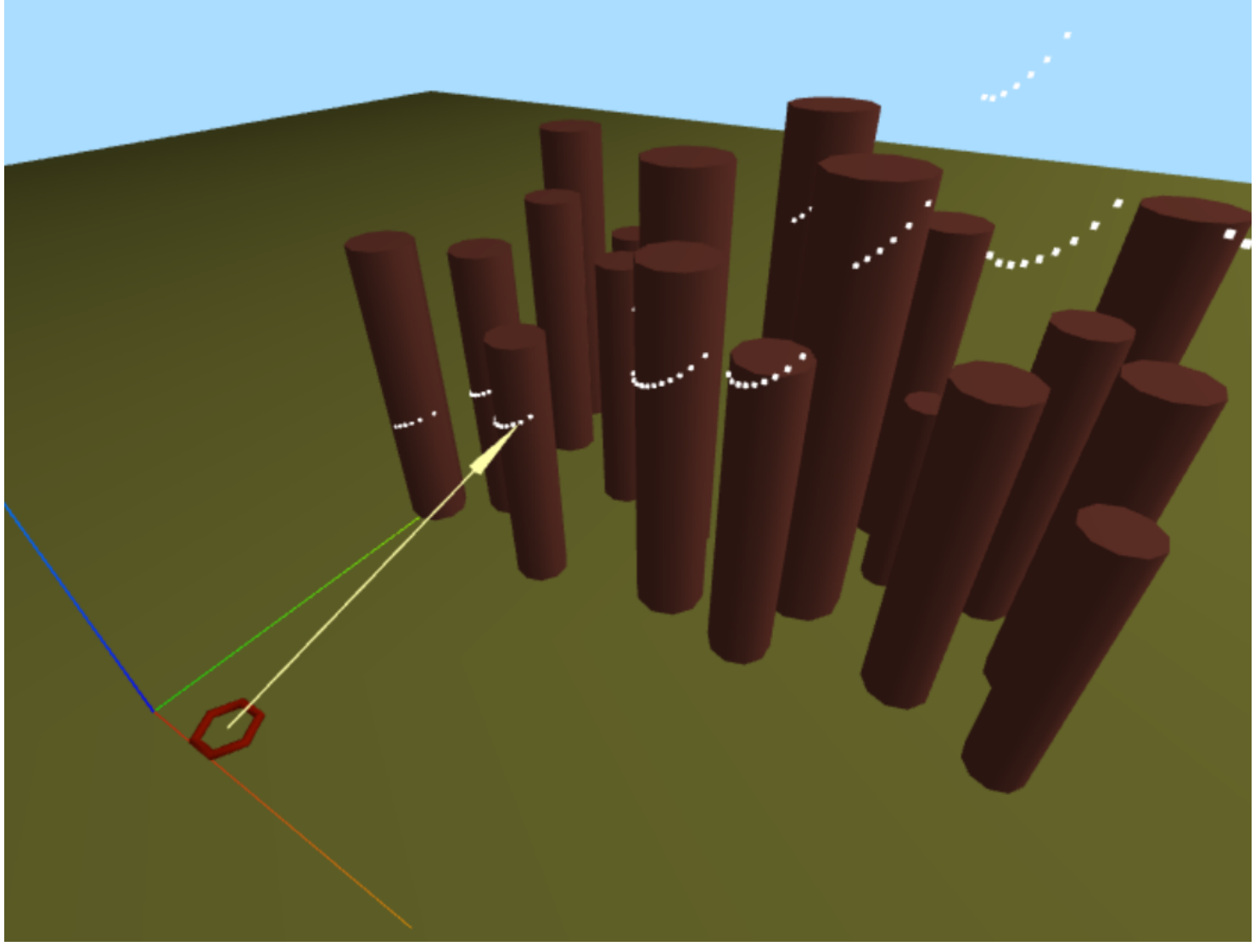
Explicações dos Exemplos

# Field Visualization

**Prof. Forster**

Vamos explicar o código da "Visualização de campo" que seria uma simulação de um Drone com um sensor de distância que navega por uma plantação para contar troncos de árvore.



Código HTML básico:

```
<!DOCTYPE html>
<html>
<head>
<meta charset=utf-8>
<title> Field visualization</title>
<style>
  body { margin: 0; overflow: hidden;}
  canvas { width: 100%; height: 100%; overflow: hidden;}
</style>
</head>
<body>

<script>
...
</script>

</body>
</html>
```

No código HTML observar a opção `overflow:hidden` que esconde as barras de rolagem se o tamanho da canvas for maior que a área cliente da janela do navegador.

Também a tag `<title>` que define o nome que aparecerá no título da janela do navegador e serve para identificar a página em outras situações.

```
var drone_loc = {"x": [0.0, 0.1010101010101010101,
0.2020202020202020202, 0.3030303030303030304, ...],
"y": [0.0, 0.0, 0.0, ...],
"z": [2.0, 2.0, 2.0, ...],
"n": [[[0.0], [1.0], [0.0]],...]}

var trees = [[8.488176972685787, 5.894479624604992,
0.4844091466117287, 4.844091466117287, 0.0], ...]

var ray_lengths=[[4.862631140902231],[4.869768855004858],...]
```

Esta variáveis contém dados gerados em outro programa (que faz a simulação do movimento e do traçado do raio) e foram colocadas como objetos do JS.

Elas poderiam ser consideradas como armazenadoras de valores de entrada, captados de uma fontes de dados.

```
<script src="js/three.js"></script>
<script src="js/OrbitControls.js"></script>
```

Inclimos a biblioteca THREE.JS para construir a visualização e o módulo OrbitControls que permite-nos alterar a visão interativamente através do mouse.

Para usar o OrbitControls, passamos a câmera para o módulo com `var controls = new THREE.OrbitControls( camera );` e atualizamos a cada quadro com `controls.update()`.

```
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 );
document.body.appendChild( renderer.domElement );
renderer.setClearColor( new THREE.Color(0xAADDFF));


camera.position.y = -5;
camera.position.z = 10;
camera.position.x = 5;
camera.up.set(0,0,1);
camera.lookAt(5,5,5);
```

Nada de novo aqui, apenas a cor do céu 0xAADDFF.

Para funcionar o controle de órbita de visualização com o mouse, colocamos:

```
var controls = new THREE.OrbitControls( camera );
```

Para visualizar os eixos X, Y e Z, acrescentamos como objeto o AxesHelper:

```
var axesHelper = new THREE.AxesHelper( 5 );
scene.add( axesHelper );
```

Aqui definimos o plano do chão. Um quadrado de 80x80.
A cor definida por reflexão difusa (Lambert) é 0xCCCC55.
Combinamos a geometri e o material no mesh `plane` e o acrescentamos à cena após o posicionar.

```
planegeom=new THREE.PlaneGeometry(80,80)
planemat= new THREE.MeshLambertMaterial({color:0xCCCC55})
plane= new THREE.Mesh(planegeom, planemat)


plane.position.x=2
plane.position.y=2


scene.add(plane)
```

Utilizamos duas fontes de luz. Uma luz ambiente (branco fraco) para dar uma iluminação mínima a cada objeto (o lado escuro não fica tão escuro) e, outra luz que é puntual, que ajuda a percepção visual das formas.

```
var light = new THREE.AmbientLight( 0x404040 );
scene.add( light );

var light2 = new THREE.PointLight( 0xffffff, 1, 100 );
light2.position.set( 50, 20, 50 );
scene.add( light2 );
```

Na luz pontual definimos cor, intensidade e distância máxima de alcance.

O nosso drone é construído com a classe TorusGeometry. O parâmetro `tubularSegments` é deixado no valor padrão 6 e, portanto, a figura fica com forma hexagonal.

O material é novamente o Lambert, com cor 0xFF2200.

```
var drone_geom = new THREE.TorusGeometry
                                    (radius=0.3, tube=0.05)
drone=new THREE.Mesh(drone_geom,
       new THREE.MeshLambertMaterial({color:0xFF2200}))

scene.add(drone)
```

O objeto ArrowHelper é utilizado para representar o sensor Laser do drone. Os parâmetro são direçao, origem, comprimento e cor.

```
var arrow=new THREE.ArrowHelper(
  new THREE.Vector3(0,1,0),
  new THREE.Vector3(0,0,2),1,0xFFFFAA)

scene.add(arrow)
```

A floresta é definida como um Group e adicionada à cena.

```
var forest= new THREE.Group()
scene.add(forest)
for (i=0; i<trees.length; i++)
{
  x=trees[i][0]
  y=trees[i][1]
  r=trees[i][2]
  h=trees[i][3]
  a=trees[i][4]
  var tree_geom=new THREE.CylinderGeometry(
     r-Math.tan(a)*h, r, h, 10, 3, false)
  var tree_mesh=new THREE.Mesh(tree_geom,
     new THREE.MeshLambertMaterial({color:0xAA5544}))
  tree_mesh.rotation.x=Math.PI/2.0
  tree_mesh.position.x=x
  tree_mesh.position.y=y
  tree_mesh.position.z=h/2
  forest.add(tree_mesh)
}
```

Os dados para a construção das árvores estão na variável `trees` definida anteriormente e foram produzidas por um simulador.

Essa variável contém uma lista de vetores da forma $(x, y, r, h, a)$ que são posição do centro, raio, altura e inclinação (o raio da base e do topo são distintos, o "Cylinder" é na verdade um tronco de cone) de cada uma das árvores.

Mesmo com apenas 10 segmentos radiais, as árvores não parecem poligonais.

O material da árvore é Lambert. Seria mais eficiente ter um único objeto material para todas as árvores, mas não é preocupante. (Compartilhar geometria é mais importante, mas aqui no caso as geometrias são distintas)

As árvores são adicionadas ao grupo `forest`.

Colocamos agora os pontos calculados como intersecção dos raios do sensor com os troncos das árvores.

```
var spot_geometry = new THREE.BoxGeometry(0.05,0.05,0.05)
var spot_material =
    new THREE.MeshBasicMaterial({color:0xFFFFFF})
for (i=0; i<drone_loc.x.length; i++)
{
  spot_mesh = new THREE.Mesh(spot_geometry,spot_material)
  spot_mesh.position.set(
     drone_loc.x[i]+drone_loc.n[i][0][0]*ray_lengths[i][0],
     drone_loc.y[i]+drone_loc.n[i][1][0]*ray_lengths[i][0],
     drone_loc.z[i]+drone_loc.n[i][2][0]*ray_lengths[i][0])
  scene.add(spot_mesh)
}
```

Os pontos são acrescentados um a um na cena. São minicubos de cor branca (MeshBasicMaterial) independente das fontes de luz. Os pontos são criados no loop e a posição é obtida calculando a partir dos dados inseridos nas variáveis `drone_loc` e `ray_lengths`.

Observamos o código da animação:

```
var t=0
var tm=0;

var animate = function () {
    requestAnimationFrame( animate );
    controls.update();
    tm=(tm+1)%5;
    if (tm==0) t=(t+1)%drone_loc.x.length;

    renderer.render( scene, camera );

    drone.position.set(drone_loc.x[t],drone_loc.y[t],
          drone_loc.z[t],);
    arrow.position.set(drone_loc.x[t],drone_loc.y[t],
          drone_loc.z[t],);
    arrow.setDirection(new THREE.Vector3(
          drone_loc.n[t][0][0], drone_loc.n[t][1][0],
          drone_loc.n[t][2][0]).normalize());
    arrow.setLength(ray_lengths[t][0]);
};


animate()
```

Olhando por partes:

```
var t=0
var tm=0;

var animate = function () {
    requestAnimationFrame( animate );
    controls.update();
    tm=(tm+1)%5;
    if (tm==0) t=(t+1)%drone_loc.x.length;

    renderer.render( scene, camera );

...
};


animate()
```

Aqui temos `controls.update()` para usarmos a interface do mouse para manipular a câmera. Temos dois contadores de tempo, a cada 5 quadros, `t` é incrementada.

Neste pedaço de código, o drone é reposicionado para cada elemento de entrada no tempo `t`. A seta é posicionada com origem no drone e orientada com a direção do raio do sensor e seu comprimento (obtidos dos dados de entrada). O método `normalize` é para obter um vetor unitário.

```
    drone.position.set(drone_loc.x[t],drone_loc.y[t],
          drone_loc.z[t],);
    arrow.position.set(drone_loc.x[t],drone_loc.y[t],
          drone_loc.z[t],);
    arrow.setDirection(new THREE.Vector3(
          drone_loc.n[t][0][0], drone_loc.n[t][1][0],
          drone_loc.n[t][2][0]).normalize());
    arrow.setLength(ray_lengths[t][0]);
```

Observe que `t` é volta a zero sempre que a lista de dados for processada por completo.