# 1.2 场景的基本功能

本文主要介绍Scene类的几个重要函数和属性，如下表所示：

|  |  |
| --- | --- |
| **函数/属性** | **描述** |
| add(object) | 在场景中添加对象。 |
| children | 返回场景中所有对象的列表，包括相机和光源。 |
| getChildrenByName(name) | 创建对象时，可以通过name属性为它指定一个唯一的名称，然后你就可以使用这个方法根据对象名称返回这个对象。 |
| remove(object) | 从场景中移除对象。 |
| traverse(function) | 传入一个回调函数遍历scene的所有子对象。 |
| fog | 通过该属性可以设置场景的雾化效果。它可以渲染出一层雾气，隐藏远处的物体。 |
| overrideMaterial | 强制场景中的所有物体都使用相同的材质。 |

我们可以使用add(object)函数在场景中添加一个THREE.Mesh对象（即plane，作为地面）、一个THREE.AmbientLight（环境光）对象、一个THREE.SpotLight（聚光灯）对象。THREE.Camera（相机）对象会自动添加，无需手工添加。代码如下：

// 创建一个Plane作为地面

var planeGeometry = new THREE.PlaneGeometry(60, 40, 1, 1);

var planeMaterial = new THREE.MeshLambertMaterial({color: 0xffffff});

var plane = new THREE.Mesh(planeGeometry,planeMaterial);

plane.receiveShadow = true;

// Plane默认在xy平面，需要将它旋转到xz平面

plane.rotation.x=-0.5\*Math.PI;

plane.position.x=0

plane.position.y=0

plane.position.z=0

// 将这个Plane添加到场景中

**scene.add(plane);**

// 添加环境光

var ambientLight = new THREE.AmbientLight(0x0c0c0c);

**scene.add(ambientLight);**

// 添加一个聚光灯用于产生阴影

var spotLight = new THREE.SpotLight(0xffffff);

spotLight.position.set( -40, 60, -10 );

spotLight.castShadow = true;

**scene.add(spotLight);**

使用dat.GUI控件可以在场景中添加方块、从场景中移除最后添加的方块，以及显示场景中的所有物体。在这个控件区中的最后一项用于显示场景中物体的数量。你会发现场景启动的时候场景中就已经有4个物体了。就是我们前面提到的地面、环境光、点光源和相机。添加方块的addBox()函数的代码如下：

this.addBox = function() {

var boxSize = Math.ceil((Math.random() \* 3));

var boxGeometry = new THREE.BoxGeometry(boxSize,boxSize,boxSize);

var boxMaterial = new THREE.MeshLambertMaterial({color: Math.random() \* 0xffffff });

var box = new THREE.Mesh(boxGeometry, boxMaterial);

box.castShadow = true;

**box.name = "box-" + scene.children.length;**

// 随机放置立方体

box.position.x = -30 + Math.round((Math.random() \* planeGeometry.parameters.width));

box.position.y = Math.round((Math.random() \* 5));

box.position.z = -20 + Math.round((Math.random() \* planeGeometry.parameters.height));

// 将立方体添加到场景中

scene.add(box);

**this.numberOfObjects = scene.children.length;**

};

当你点击addBox按钮时，一个新的THREE.BoxGeometry实例就会创建出来，其尺寸是0到3之间的一个随机数。除了随机大小尺寸之外，这个方块在场景中的颜色和位置也是随机的。

这段代码里的一个新东西是我们使用name属性为这个方块指定了一个名字。方块的名字是在box-后面加上当前场景中对象的数量（即显示的scene.children.length属性）。所以会得到诸如box-1、box-2、box-3等类似的名字。名字在调试的时候很有用，可以使用Scene.getChildByName(name)函数直接获取指定的对象，然后可以执行一些操作，例如改变它的位置。最后一行代码的目的是在控制界面使用numberOfObjects变量显示场景中的对象。所以无论什么时候添加或删除对象，我们都要将这个变量设置为更新后的数量。

在控制界面上调用的下一个函数是removeBox，点击这个按钮可以把最后添加的方块从场景中移除。下面的代码片段即该函数的定义：

this.removeBox = function () {

var allChildren = scene.children;

var lastObject = allChildren[allChildren.length - 1];

if (lastObject instanceof THREE.Mesh) {

scene.remove(lastObject);

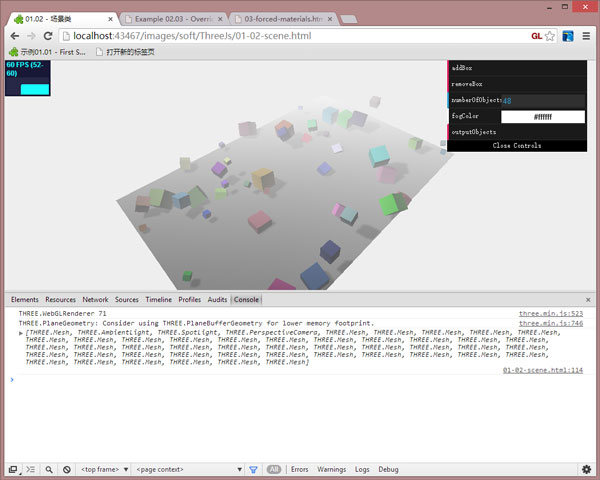
this.numberOfObjects = scene.children.length;

}

}

在这段代码里我们通过THREE.Scene()对象的children属性获取最后添加的对象。我们还需要检查一下这个对象是不是一个Mesh对象，以防移除相机和光源。删掉这个对象之后，我们将再一次更新控制界面上那个表示场景中对象数量的属性。

控制界面上的最后一个按钮的标签是outputObjects。这个按钮的功能是打印出场景中的所有对象，并将结果输出到网络浏览器的Console（控制台），如下图所示：



往Console输出信息用的是内置的console对象，代码如下：

this.outputObjects = function() {

console.log(scene.children);

}

这对调试来讲非常有用，可以帮助你查找场景中某个特定物体的相关的问题或错误。

在代码中，我们还用到了THREE.Scene.traverse()函数。我们可以将一个函数作为参数传递给traverser()函数。这个传递来的函数将会在场景的每一个子对象上调用一次：在render()函数里，我们将使用traverse()函数类更新每个方块的旋转弧度（我们特意忽略了表示地面的plane对象）。我们也可以使用for循环遍历children这个属性数组来达到同样的目的。代码如下：

function render() {

stats.update();

// 旋转立方体

scene.traverse(function(e) {

if (e instanceof THREE.Mesh && e != plane ) {

e.rotation.x+=0.02;

e.rotation.y+=0.02;

e.rotation.z+=0.02;

}

});

requestAnimationFrame(render);

renderer.render(scene, camera);

}

## 在场景中添加雾化效果

通过fog属性可以为整个场景添加一种雾化效果。一个物体离得越远，就越模糊。

在Three.js库里打开雾化效果很简单。只要在定义完场景后加上如下一行代码即可：

scene.fog = new THREE.Fog(0xffffff, 0.015, 100);

我们在这里定义了一个白色的雾化效果（0xffffff）。后面的两个属性用来调节雾的显示。0.015是near（近处）属性的值，而100设置的是far（远处）属性的值。通过这两个属性你可以决定雾从什么地方开始，以及浓度加深的程度。还有另外一个方法可以设置场景中雾的浓度；你可以使用如下的定义：

scene.fog=new THREE.FogExp2( 0xffffff, 0.015 );

这次我们不指定near属性和far属性，只给出颜色和浓度。要想获得理想的效果，你最好亲手试验一下这些属性。

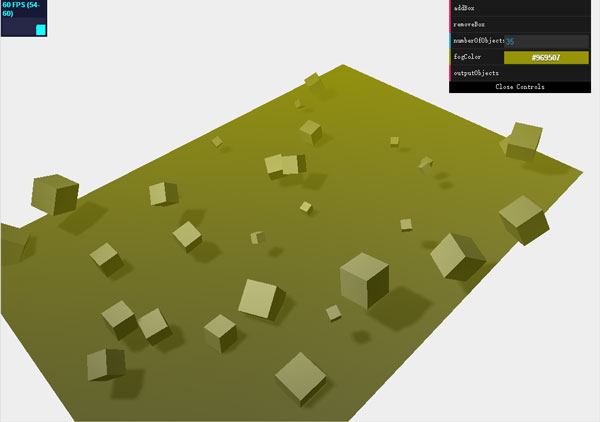
## 使用材质覆盖属性

场景的overrideMaterial属性用来设置场景中所有物体的材质。当使用下面这段代码那样使用这个属性时，所有添加到场景中的物体都会使用同样的材质：

scene.overrideMaterial = new THREE.MeshLambertMaterial({ color: 0xffffff });

你可以看到所有的方块都是用相同的材质和颜色渲染的。你可以将这行代码注释掉看看会发生什么。

最终的程序截图如下：



## 完整代码

<!DOCTYPE html>

<html>

<head>

<title>01.02 - 场景类</title>

<script src="../../../Scripts/jquery-2.1.3.min.js"></script>

<script src="../../../Scripts/Threejs/three.min.js"></script>

<script src="../../../Scripts/Threejs/dat.gui.min.js"></script>

<script src="../../../Scripts/Threejs/stats.js"></script>

<style>

body {

/\* 将margin设置为0，overflow设置为hidden，可让浏览器显示全屏 \*/

margin: 0;

overflow: hidden;

}

</style>

</head>

<body>

<div id="Stats-output">

</div>

<!-- 作为Canvas容器的div -->

<div id="WebGL-output">

</div>

<script type="text/javascript">

// 页面加载完毕后，就可以运行Three.js了。

$(function () {

var stats = initStats();

// 创建渲染器，并设置视口大小和清除色

var renderer = new THREE.WebGLRenderer();

renderer.setClearColor(0xEEEEEE, 1.0);

renderer.setSize(window.innerWidth, window.innerHeight);

renderer.shadowMapEnabled = true;

// 将WebGL的输出canvas放置到div中

$("#WebGL-output").append(renderer.domElement);

// 创建scene对象，用来容纳网格、相机、光源等对象

var scene = new THREE.Scene();

//scene.fog=new THREE.FogExp2( 0xffffff, 0.015 );

scene.fog = new THREE.Fog(0xffffff, 0.015, 100);

scene.overrideMaterial = new THREE.MeshLambertMaterial({ color: 0xffffff });

// 场景相机

var camera = new THREE.PerspectiveCamera(45, window.innerWidth / window.innerHeight, 0.1, 1000);

// 设置相机

camera.position.x = -30;

camera.position.y = 40;

camera.position.z = 30;

camera.lookAt(scene.position);

// 创建一个Plane作为地面

var planeGeometry = new THREE.PlaneGeometry(60, 40, 1, 1);

var planeMaterial = new THREE.MeshLambertMaterial({ color: 0xffffff });

var plane = new THREE.Mesh(planeGeometry, planeMaterial);

plane.receiveShadow = true;

// Plane默认在xy平面，需要将它旋转到xz平面

plane.rotation.x = -0.5 \* Math.PI;

plane.position.x = 0

plane.position.y = 0

plane.position.z = 0

// 将这个Plane添加到场景中

scene.add(plane);

// 添加环境光

var ambientLight = new THREE.AmbientLight(0x0c0c0c);

scene.add(ambientLight);

// 添加一个聚光灯用于产生阴影

var spotLight = new THREE.SpotLight(0xffffff);

spotLight.position.set(-40, 60, -10);

spotLight.castShadow = true;

scene.add(spotLight);

// GUI界面

var controls = new function () {

this.numberOfObjects = scene.children.length;

this.fogColor = scene.fog.color.getStyle();

this.addBox = function () {

var boxSize = Math.ceil((Math.random() \* 3));

var boxGeometry = new THREE.BoxGeometry(boxSize, boxSize, boxSize);

var boxMaterial = new THREE.MeshLambertMaterial({ color: Math.random() \* 0xffffff });

var box = new THREE.Mesh(boxGeometry, boxMaterial);

box.castShadow = true;

box.name = "box-" + scene.children.length;

// 随机放置立方体

box.position.x = -30 + Math.round((Math.random() \* planeGeometry.parameters.width));

box.position.y = Math.round((Math.random() \* 5));

box.position.z = -20 + Math.round((Math.random() \* planeGeometry.parameters.height));

// 将立方体添加到场景中

scene.add(box);

this.numberOfObjects = scene.children.length;

};

this.removeBox = function () {

var allChildren = scene.children;

var lastObject = allChildren[allChildren.length - 1];

if (lastObject instanceof THREE.Mesh) {

scene.remove(lastObject);

this.numberOfObjects = scene.children.length;

}

}

this.outputObjects = function () {

console.log(scene.children);

}

}

var gui = new dat.GUI();

gui.add(controls, 'addBox');

gui.add(controls, 'removeBox');

gui.add(controls, 'numberOfObjects').listen();

gui.addColor(controls, 'fogColor').onChange(function (e) {

scene.fog.color.setStyle(e);

});

gui.add(controls, 'outputObjects');

// 渲染

render();

function render() {

stats.update();

// 旋转立方体

scene.traverse(function (e) {

if (e instanceof THREE.Mesh && e != plane) {

e.rotation.x += 0.02;

e.rotation.y += 0.02;

e.rotation.z += 0.02;

}

});

requestAnimationFrame(render);

renderer.render(scene, camera);

}

function initStats() {

var stats = new Stats();

stats.setMode(0); // 0: fps, 1: ms

stats.domElement.style.position = 'absolute';

stats.domElement.style.left = '0px';

stats.domElement.style.top = '0px';

$("#Stats-output").append(stats.domElement);

return stats;

}

});

</script>

</body>

</html>