## 1、让场景动起来

这一节课，我们要让场景动起来，不禁想到了郭富城的一首歌《动起来》。心中有很多感慨，觉得时间过得太快，自己还没有多大的成功。以淡淡的感伤开始这节课的讲解。

不过我会继续努力，寻找自己更多的成功。

大家也动起来，沉静下来，仔细的研究Three.js的每一个细节，终将成为这个领域的高手。不仅是成为three.js的高手，更重要的是理解图形学的概念，轻易掌握其他3D图形库。不是吹牛，大家阅读完这套课程（包括中级，高级），能够轻易的实现艳丽的粒子系统、模拟多种物理现象（如衣服在风中飘动），让浏览器中2D和3D混合等令人大饱眼福的效果。

以往的例子中，我们很少让物体动起来，即使动起来了，也很少讲这方面的知识。这里我们对让场景动起来做一些解释。

### **1、场景怎么会动起来**

场景中的物体怎么才能运动起来。我们这里从《古兰经》讲起，这样，你永远不会忘记。

《古兰经》上有这样一个故事：一天穆罕穆德告诉人们说大山会向我们走来。于是人们就远望大山，看它怎么走过来，可是等了好长时间大山还是纹丝不动的在那里，人们就问穆罕穆德，大山也没向我们走来啊。默罕默德告诉人们：既然大山没向我们走来，那我们就向大山走去吧。于是人们来到了大山的山顶，人们征服了那座大山。

这个故事揭示了场景动起来的方法，第一种方法是让物体在坐标系里面移动，摄像机不动。第二种方法是让摄像机在坐标系里面移动，物体不动。这样场景就能够动起来了。

摄像机可以理解我们自己的眼睛。

### **2、渲染循环**

物体运动还有一个关键点，就是要渲染物体运动的每一个过程，让它显示给观众。渲染的时候，我们调用的是渲染器的render() 函数。代码如下：

renderer.render( scene, camera );

如果我们改变了物体的位置或者颜色之类的属性，就必须重新调用render()函数，才能够将新的场景绘制到浏览器中去。不然浏览器是不会自动刷新场景的。

如果不断的改变物体的颜色，那么就需要不断的绘制新的场景，所以我们最好的方式，是让画面执行一个循环，不断的调用render来重绘，这个循环就是渲染循环，在游戏中，也叫游戏循环。

为了实现循环，我们需要javascript的一个特殊函数，这个函数是requestAnimationFrame。

调用requestAnimationFrame函数，传递一个callback参数，则在下一个动画帧时，会调用callback这个函数。

于是，我们的游戏循环会这样写。

function animate() {

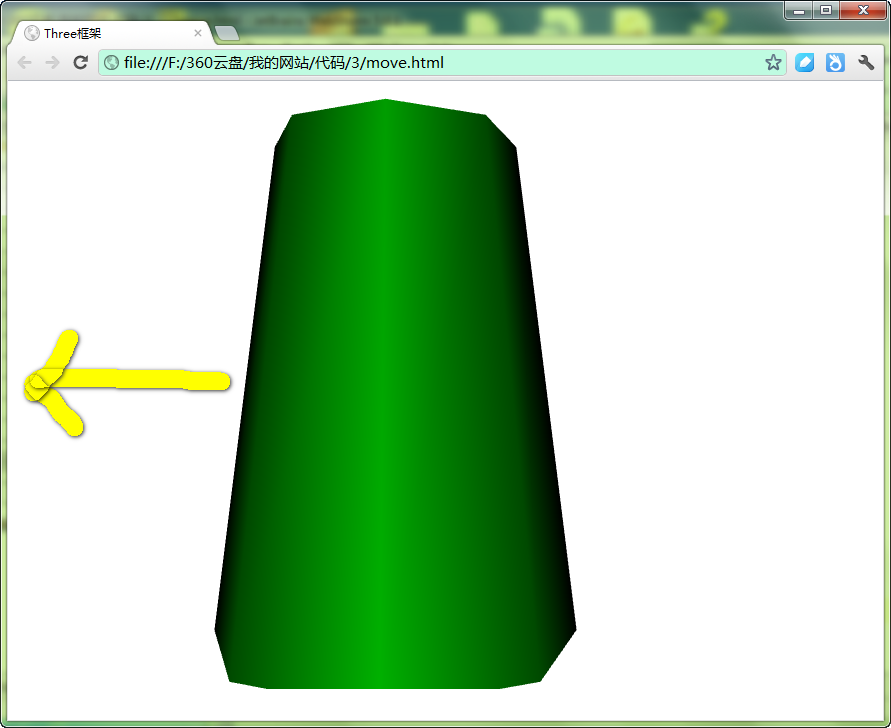
render();

requestAnimationFrame( animate );

}

这样就会不断的执行animate这个函数。也就是不断的执行render()函数。在render()函数中不断的改变物体或者摄像机的位置，并渲染它们，就能够实现动画了。

## 2、改变相机的位置，让物体移动

有了这些简单的基础知识，我们来实现一个动画效果。它的效果如下所示：

看箭头的方向，你会发现这个物体在向左边移动。你可以在“初级教程\chapter3\3-1.html”这个源文件中发现代码：

|  |
| --- |
|  |
| <!DOCTYPE html> |
| <html> |
| <head> |
| <meta charset="UTF-8"> |
| <title>Three框架</title> |
| <script src="js/Three.js"></script> |
| <style type="text/css"> |
| div#canvas-frame { |
| border: none; |
| cursor: pointer; |
| width: 100%; |
| height: 600px; |
| background-color: #EEEEEE; |
| } |
|  |
| </style> |
| <script> |
| var renderer; |
| function initThree() { |
| width = document.getElementById('canvas-frame').clientWidth; |
| height = document.getElementById('canvas-frame').clientHeight; |
| renderer = new THREE.WebGLRenderer({ |
| antialias : true |
| }); |
| renderer.setSize(width, height); |
| document.getElementById('canvas-frame').appendChild(renderer.domElement); |
| renderer.setClearColor(0xFFFFFF, 1.0); |
| } |
|  |
| var camera; |
| function initCamera() { |
| camera = new THREE.PerspectiveCamera(45, width / height, 1, 10000); |
| camera.position.x = 0; |
| camera.position.y = 0; |
| camera.position.z = 600; |
| camera.up.x = 0; |
| camera.up.y = 1; |
| camera.up.z = 0; |
| camera.lookAt({ |
| x : 0, |
| y : 0, |
| z : 0 |
| }); |
| } |
|  |
| var scene; |
| function initScene() { |
| scene = new THREE.Scene(); |
| } |
|  |
| var light; |
| function initLight() { |
| light = new THREE.AmbientLight(0xFFFFFF); |
| light.position.set(100, 100, 200); |
| scene.add(light); |
| light = new THREE.PointLight(0x00FF00); |
| light.position.set(0, 0,300); |
| scene.add(light); |
| } |
|  |
| var cube; |
| function initObject() { |
| var geometry = new THREE.CylinderGeometry( 100,150,400); |
| var material = new THREE.MeshLambertMaterial( { color:0xFFFF00} ); |
| var mesh = new THREE.Mesh( geometry,material); |
| mesh.position = new THREE.Vector3(0,0,0); |
| scene.add(mesh); |
| } |
|  |
| function threeStart() { |
| initThree(); |
| initCamera(); |
| initScene(); |
| initLight(); |
| initObject(); |
| animation(); |
|  |
| } |
| function animation() |
| { |
| //renderer.clear(); |
| camera.position.x =camera.position.x +1; |
| renderer.render(scene, camera); |
| requestAnimationFrame(animation); |
| } |
|  |
| </script> |
| </head> |
|  |
| <body onload="threeStart();"> |
| <div id="canvas-frame"></div> |
| </body> |
| </html> |

我们将重点放在A begin和A end 处的代码，它将不断的通过下面的代码改变相机的位置：

camera.position.x =camera.position.x +1;

将相机不断的沿着x轴移动1个单位，也就是相机向右移动。相机向右移动，那么想一想相机中的物体，是怎么移动的呢？毫无疑问，它是反方向移动的，是向左移动的。

设置完相机的位置后，我们调用requestAnimationFrame(animation)函数，这个函数又会在下一个动画帧出发animation()函数，这样就不断改变了相机的位置，从而物体看上去在移动了。

另外，必须要重视render函数，这个函数是重新绘制渲染结果，如果不调用这个函数，那么即使相机的位置变化了，但是没有重新绘制，仍然显示的是上一帧的动画。Render函数调用如下：

renderer.render(scene, camera);

## 3、改变物体自身的位置，让物体移动

第二种方式，就是让物体动起来，只要改变物体的位置就可以了。代码如下，你可以在“初级教程\chapter3\3-2.html”中找到代码：

|  |
| --- |
| <!DOCTYPE html> |
| <html> |
| <head> |
| <meta charset="UTF-8"> |
| <title>Three框架</title> |
| <script src="js/Three.js"></script> |
| <style type="text/css"> |
| div#canvas-frame { |
| border: none; |
| cursor: pointer; |
| width: 100%; |
| height: 600px; |
| background-color: #EEEEEE; |
| } |
|  |
| </style> |
| <script> |
| var renderer; |
| function initThree() { |
| width = document.getElementById('canvas-frame').clientWidth; |
| height = document.getElementById('canvas-frame').clientHeight; |
| renderer = new THREE.WebGLRenderer({ |
| antialias : true |
| }); |
| renderer.setSize(width, height); |
| document.getElementById('canvas-frame').appendChild(renderer.domElement); |
| renderer.setClearColor(0xFFFFFF, 1.0); |
| } |
|  |
| var camera; |
| function initCamera() { |
| camera = new THREE.PerspectiveCamera(45, width / height, 1, 10000); |
| camera.position.x = 0; |
| camera.position.y = 0; |
| camera.position.z = 600; |
| camera.up.x = 0; |
| camera.up.y = 1; |
| camera.up.z = 0; |
| camera.lookAt({ |
| x : 0, |
| y : 0, |
| z : 0 |
| }); |
| } |
|  |
| var scene; |
| function initScene() { |
| scene = new THREE.Scene(); |
| } |
|  |
| var light; |
| function initLight() { |
| light = new THREE.AmbientLight(0xFF0000); |
| light.position.set(100, 100, 200); |
| scene.add(light); |
| light = new THREE.PointLight(0x00FF00); |
| light.position.set(0, 0,300); |
| scene.add(light); |
| } |
|  |
| var cube; |
| var mesh; |
| function initObject() { |
| var geometry = new THREE.CylinderGeometry( 100,150,400); |
| var material = new THREE.MeshLambertMaterial( { color:0xFFFFFF} ); |
| mesh = new THREE.Mesh( geometry,material); |
| mesh.position = new THREE.Vector3(0,0,0); |
| scene.add(mesh); |
| } |
|  |
| function threeStart() { |
| initThree(); |
| initCamera(); |
| initScene(); |
| initLight(); |
| initObject(); |
| animation(); |
|  |
| } |
| function animation() |
| { |
| mesh.position.x-=1; |
| renderer.render(scene, camera); |
| requestAnimationFrame(animation); |
| } |
|  |
| </script> |
| </head> |
|  |
| <body onload="threeStart();"> |
| <div id="canvas-frame"></div> |
| </body> |
| </html> |
|  |

关注A begin 和A end 处的代码，其中有一句，这也是和前一个例子唯一不同的一句：

mesh.position.x-=1;

其中mesh就是指的物体，它有一个位置属性position，这个position是一个THREE.Vector3类型变量，所以你要把它向左移动，只需要将x的值不断的减少就可以了。这里我们减去的是1个单位。

Ok，分析完毕，很轻松吧。

## 4、物体运动后，怎么评估程序的性能

关于性能：测试一个程序，性能上是否有瓶颈，在3D世界里，经常使用帧数的概念，首先我们来定义一下帧数的意义。

帧数：图形处理器每秒钟能够刷新几次，通常用fps（Frames Per Second）来表示。如下是每秒钟59次刷新的应用：

当物体在快速运动时,当人眼所看到的影像消失后，人眼仍能继续保留其影像1/24秒左右的图像，这种现象被称为视觉暂留现象。是人眼具有的一种性质。人眼观看物体时，成像于视网膜上，并由视神经输入人脑，感觉到物体的像。一帧一帧的图像进入人脑，人脑就会将这些图像给连接起来，形成动画。

毫无疑问，帧数越高，画面的感觉就会越好。所以大多数游戏都会有超过30的FPS。为了监视FPS，看看你的程序哪里占用了很多的CPU时间，就需要学习一下性能监视器。

### **1、性能监视器Stats**

在Three.js中，性能由一个性能监视器来管理，它的介绍在https://github.com/mrdoob/stats.js 可以看到。性能监视器的截图如下所示:IMG_258

其中FPS表示：上一秒的帧数，这个值越大越好，一般都为60左右。点击上面的图，就会变成下面的另一个视图。IMG_259

MS表示渲染一帧需要的毫秒数，这个数字是越小越好。再次点击又可以回到FPS视图中。

### **2、性能监视器Stats的使用**

在Three.js中，性能监视器被封装在一个类中，这个类叫做Stats，下面是一段伪代码，表示Stats的使用。

|  |
| --- |
| var stats = new Stats(); |
| stats.setMode(1); *// 0: fps, 1: ms* |
| *// 将stats的界面对应左上角* |
| stats.domElement.style.position = 'absolute'; |
| stats.domElement.style.left = '0px'; |
| stats.domElement.style.top = '0px'; |
| document.body.appendChild( stats.domElement ); |
| setInterval( function () { |
| stats.begin(); |
| *// 你的每一帧的代码* |
| stats.end(); |
| }, 1000 / 60 ); |

你现在可以自己写一段代码，来验证一下，你的程序的帧数了。

Stats到底做了什么事情呢？我们来分析一下。

1、setMode函数

参数为0的时候，表示显示的是FPS界面，参数为1的时候，表示显示的是MS界面。

2、stats的domElement

stats的domElement表示绘制的目的地（DOM），波形图就绘制在这上面。

3、begin

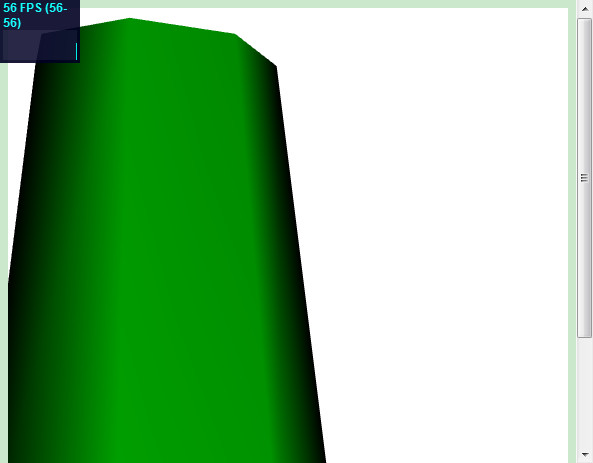
begin，在你要测试的代码前面调用begin函数，在你代码执行完后调用end()函数，这样就能够统计出这段代码执行的平均帧数了。

## 5、性能测试实例

好了，有了上面关于Stats类的基础知识后，我们来讲一个使用这个类的实例。

Stats的begin和end 函数本质上是在统计代码执行的时间和帧数，然后用公式fps=帧数/时间 ，就能够得到FPS。Stats的这个功能，被封装成了一个更好的函数update，只需要在渲染循环中调用就可以了。请看下面的代码[初级教程\chapter3\3-3.html]，这份代码是在3-2.html的基础上，加上了stats统计得到的。

|  |
| --- |
|  |
| <!DOCTYPE html> |
| <html> |
| <head> |
| <meta charset="UTF-8"> |
| <title>Three框架</title> |
| <script src="js/Three.js"></script> |
| <script src="js/Stats.js"></script> |
|  |
| <style type="text/css"> |
| div#canvas-frame { |
| border: none; |
| cursor: pointer; |
| width: 100%; |
| height: 600px; |
| background-color: #EEEEEE; |
| } |
|  |
| </style> |
| <script> |
| var renderer; |
| var stats; |
| function initThree() { |
| width = document.getElementById('canvas-frame').clientWidth; |
| height = document.getElementById('canvas-frame').clientHeight; |
| renderer = new THREE.WebGLRenderer({ |
| antialias : true |
| }); |
| renderer.setSize(width, height); |
| document.getElementById('canvas-frame').appendChild(renderer.domElement); |
| renderer.setClearColor(0xFFFFFF, 1.0); |
|  |
| stats = new Stats(); |
| stats.domElement.style.position = 'absolute'; |
| stats.domElement.style.left = '0px'; |
| stats.domElement.style.top = '0px'; |
| document.getElementById('canvas-frame').appendChild(stats.domElement); |
| } |
|  |
| var camera; |
| function initCamera() { |
| camera = new THREE.PerspectiveCamera(45, width / height, 1, 10000); |
| camera.position.x = 0; |
| camera.position.y = 0; |
| camera.position.z = 600; |
| camera.up.x = 0; |
| camera.up.y = 1; |
| camera.up.z = 0; |
| camera.lookAt({ |
| x : 0, |
| y : 0, |
| z : 0 |
| }); |
| } |
|  |
| var scene; |
| function initScene() { |
| scene = new THREE.Scene(); |
| } |
|  |
| var light; |
| function initLight() { |
| light = new THREE.AmbientLight(0xFF0000); |
| light.position.set(100, 100, 200); |
| scene.add(light); |
| light = new THREE.PointLight(0x00FF00); |
| light.position.set(0, 0,300); |
| scene.add(light); |
| } |
|  |
| var cube; |
| var mesh; |
| function initObject() { |
| var geometry = new THREE.CylinderGeometry( 100,150,400); |
| var material = new THREE.MeshLambertMaterial( { color:0xFFFFFF} ); |
| mesh = new THREE.Mesh( geometry,material); |
| mesh.position = new THREE.Vector3(0,0,0); |
| scene.add(mesh); |
| } |
|  |
| function threeStart() { |
| initThree(); |
| initCamera(); |
| initScene(); |
| initLight(); |
| initObject(); |
| animation(); |
|  |
| } |
| function animation() |
| { |
| //renderer.clear(); |
| //camera.position.x =camera.position.x +1; |
| mesh.position.x-=1; |
| renderer.render(scene, camera); |
| requestAnimationFrame(animation); |
|  |
| stats.update(); |
| } |
|  |
| </script> |
| </head> |
|  |
| <body onload="threeStart();"> |
| <div id="canvas-frame"></div> |
| </body> |
| </html> |
|  |

代码的运行效果如下图所示，左上部，就是显示的帧计数器。

以上代码一共只有几个步骤：

### **1、new 一个stats对象，代码如下**

stats = new Stats();

### **2、将这个对象加入到html网页中去，代码如下**

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

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

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

### **3、调用stats.update()函数来统计时间和帧数。代码如下**

stats.update();

好了，这节课就讲到这里了，我们讲了重要的游戏循环和性能测试的方法。当帧数较低的时候，你就要注意了，可能是你的代码性能太低了造成的。一般情况下，帧数都可以跑到60的。

## 6、使用动画引擎Tween.js来创建动画

上面介绍了通过移动相机和移动物体来产生动画的效果。使用的方法是在渲染循环里去移动相机或者物体的位置。如果动画稍微复杂一些，这种方式实现起来就比较麻烦一些了。

为了使程序编写更容易一些，我们可以使用动画引擎来实现动画效果。和three.js紧密结合的动画引擎是Tween.js,你可以再https://github.com/sole下载。

对于快速构件动画来说，Tween.js是一个容易上手的工具。首先，你需要引擎js文件，如下：

<-script src="../js/tween.min.js" data-ke-src="../js/tween.min.js"><-/script>

第二步，就是构件一个Tween对象，对Tween进行初始化，本例的代码是:

|  |
| --- |
| function initTween() |
| { |
| new TWEEN.Tween( mesh.position) |
| .to( { x: -400 }, 3000 ).repeat( Infinity ).start(); |
| } |

TWEEN.Tween的构造函数接受的是要改变属性的对象，这里传入的是mesh的位置。Tween的任何一个函数返回的都是自身，所以可以用串联的方式直接调用各个函数。

to函数，接受两个参数，第一个参数是一个集合，里面存放的键值对，键x表示mesh.position的x属性，值-400表示，动画结束的时候需要移动到的位置。第二个参数，是完成动画需要的时间，这里是3000ms。

repeat( Infinity )表示重复无穷次，也可以接受一个整形数值，例如5次。

Start表示开始动画，默认情况下是匀速的将mesh.position.x移动到-400的位置。

第三步是，需要在渲染函数中去不断的更新Tween，这样才能够让mesh.position.x移动位置:

|  |
| --- |
| function animation() |
| { |
| renderer.render(scene, camera); |
| requestAnimationFrame(animation); |
| stats.update(); |
| TWEEN.update(); |
| } |

其中的TWEEN.update()完成了让动画动起来到目标。如果不调用这个函数场景就不能动起来了。

你可以再[初级教程\chapter3\3-4.html]和[初级教程\chapter3\3-5.html]找到这节的代码，3-4.html是让物体动起来，3-5.html是让相机动起来。

## **7、使用动画引擎Tween.js来创建不规则动画**

本节是一扩展的小结。如果对此不敢兴趣，可以跳过此节，也不影响学习。

上面讲的运动是直线运动，有时候我们需要曲线运动，例如下面图中的运动轨迹：

点击这里可以看到曲线运动方式。在实际工作中，经常是曲线运动，所以你有必要去快速掌握这些知识。