## 1、因为精彩，所以值得

还记得2000年左右有一个很不错的OpenGL教程《Nehe OpenGL》，大学时，我才有机会读到那么好的教程。那时，废寝忘食，花了一个月的时间看完了整个教程。觉得不过瘾，又读了一遍师兄翻译的中文教程。在当时，Nehe教程实为经典，第一个原因是当时OpenGL的资料非常罕见，无论是新华书店，还是互联网上都很难找到一本详尽的OpenGL教程。第二个原因是NeHe教程的作者确实非常用心，写出来很精彩的实例，并浅显易懂的讲解了它。

我一直认为，我写《WebGL中文网》的目的是为3D爱好者们提供一套精彩的入门和精通资料，这套资料不仅包含基础知识，而且包含工程级别的实例，更重要的是，那能让大家理解图形学知识。长此以来，很多同学在学习了计算机图形学后，对图形学的知识还是不能应用。有很大的原因，就是缺少理论化的实践，所以，这点，我们也考虑到了，一定在讲解示例的过程中，将其中的图形学原理也给阐述清楚。

以使大家认真学习后，能够在较短的时间学到我所学习的知识。

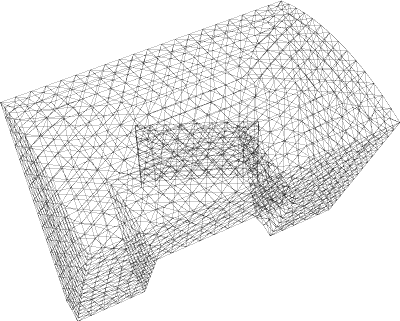
关于图形学这些知识，我是用了很多时间去摸索的，在资料匮乏的时候去掌握一门新的知识，确实是很费时间和脑力的事情，但是现在，有了本教程，你可以直通光明顶，学到可以胜任WebGL工作的技能。我想，这也是《WebGL中文网》的宗旨吧。

## 2、不从画鸡蛋开始，我们从画点开始

睁开您的眼睛，看看你的周围，它是一个多么美妙的3D世界啊。3D世界由什么组成，除了上帝，还有谁能够回答呢？

是的，没有人能够回答这个问题。但是在计算机世界里，3D世界由什么组成就很好回答。

### **1、3D世界的组成**

在计算机世界里，3D世界是由点组成，两个点能够组成一条直线，三个不在一条直线上的点就能够组成一个三角形面，无数三角形面就能够组成各种形状的物体，如下图：

我们通常把这种网格模型叫做Mesh模型。给物体贴上皮肤，或者专业点就叫做纹理，那么这个物体就活灵活现了。最后无数的物体就组成了我们的3D世界。

那么3D世界的组成，是否真的这样简单？是的，从编程的角度，目前为此，你只需要知道这些。下一节，我们从点说起。

### **2、在Threejs中定义一个点**

在三维空间中的某一个点可以用一个坐标点来表示。一个坐标点由x,y,z三个分量构成。在three.js中，点可以在右手坐标系中表示：

空间几何中，点可以用一个向量来表示，在Three.js中也是用一个向量来表示的，代码如下所示：

|  |
| --- |
| THREE.Vector3 = function ( x, y, z ) { |
|  |
| this.x = x || 0; |
| this.y = y || 0; |
| this.z = z || 0; |
|  |
| }; |

我们来分析这段代码：前面我们已经知道了THREE是Three.js引擎的一个全局变量。只要你想用它，就可以在任何地方用它。有点充气娃娃的意思，不需要你同意，你想用就用吧。

那么THREE.Vector3呢，就是表示Vector3是定义在THREE下面的一个类。以后要用Vector3，就必须要加THREE前缀。当然Three.js的设计者，也可以不加THREE这个前缀，但是他们预见到，Three.js引擎中会有很多类型，最好给这些类型加一个前缀，以免与开发者的代码产生冲突。

THREE.Vector3被赋值为一个函数。这个函数有3个参数，分别代表x坐标，y坐标和z坐标的分量。函数体内的代码将他们分别赋值给成员变量x，y，z。看看上面的代码，中间使用了一个“||”（或）运算符，就是当x=null或者undefine时，this.x的值应该取0。

### **3、点的操作**

在3D世界中点可以用THREE.Vector3D来表示。对应源码为/src/math/Vector3.js（注意：源码所在的位置，可能不同版本不一样，请自己搜索Vector3关键词来确定）。在您继续学习之前，你可以打开该文件浏览一下，推荐使用WebStorm，如果没有，你也可以用NotePad++。

现在来看看怎么定义个点，假设有一个点x=4，y=8，z=9。你可以这样定义它：

var point1 = new THREE.Vecotr3(4,8,9);

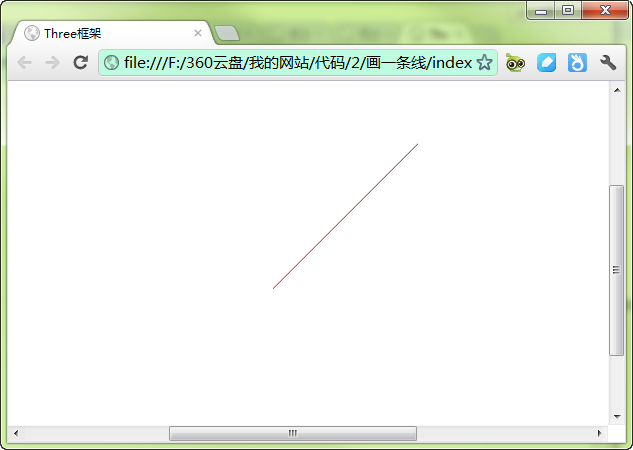
另外你也可以使用set方法，代码如下：

var point1 = new THREE.Vector3();

point1.set(4,8,9);

我们这里使用了set方法，为了以后深入学习的方便，这里将Vector3的常用方法列出如下，为了不影响文章的连贯性，我们专门列出了一个网页来介绍它。

## 3、实例：画一条彩色线

初中数学中有一个定理：两个不重合的点能够决定一条直线。在three.js中，也可以通过定义两个点，来画一条直线。我们先看看，这一节，我们要完成实例的效果图：

例子说明：这是一条每个点不同颜色的线条

这个例子的代码如下所示，可以在“【初级教程\chapter2\2-1.html】”中找到这份代码：[在线浏览](http://hewebgl.com/code/primary/chapter2/a2-1.html)

|  |
| --- |
| <!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 = 1000; |
| camera.position.z = 0; |
| camera.up.x = 0; |
| camera.up.y = 0; |
| camera.up.z = 1; |
| camera.lookAt({ |
| x : 0, |
| y : 0, |
| z : 0 |
| }); |
| } |
|  |
| var scene; |
| function initScene() { |
| scene = new THREE.Scene(); |
| } |
|  |
| var light; |
| function initLight() { |
| light = new THREE.DirectionalLight(0xFF0000, 1.0, 0); |
| light.position.set(100, 100, 200); |
| scene.add(light); |
| } |
|  |
| var cube; |
| function initObject() { |
|  |
| var geometry = new THREE.Geometry(); |
| var material = new THREE.LineBasicMaterial( { vertexColors: true } ); |
| var color1 = new THREE.Color( 0x444444 ), color2 = new THREE.Color( 0xFF0000 ); |
|  |
| // 线的材质可以由2点的颜色决定 |
| var p1 = new THREE.Vector3( -100, 0, 100 ); |
| var p2 = new THREE.Vector3( 100, 0, -100 ); |
| geometry.vertices.push(p1); |
| geometry.vertices.push(p2); |
| geometry.colors.push( color1, color2 ); |
|  |
| var line = new THREE.Line( geometry, material, THREE.LinePieces ); |
| scene.add(line); |
| } |
|  |
| function threeStart() { |
| initThree(); |
| initCamera(); |
| initScene(); |
| initLight(); |
| initObject(); |
| renderer.clear(); |
| renderer.render(scene, camera); |
| } |
|  |
| </script> |
| </head> |
|  |
| <body onload="threeStart();"> |
| <div id="canvas-frame"></div> |
| </body> |
| </html> |

看看A begin 和A end之间的代码，就是画线的代码。

### **1、首先，我们声明了一个几何体geometry，如下：**

var geometry = new THREE.Geometry();

几何体里面有一个vertices变量，可以用来存放点。

### **2、定义一种线条的材质，使用THREE.LineBasicMaterial类型来定义，它接受一个集合作为参数，其原型如下：**

LineBasicMaterial( parameters )

Parameters是一个定义材质外观的对象，它包含多个属性来定义材质，这些属性是：

Color：线条的颜色，用16进制来表示，默认的颜色是白色。

Linewidth：线条的宽度，默认时候1个单位宽度。

Linecap：线条两端的外观，默认是圆角端点，当线条较粗的时候才看得出效果，如果线条很细，那么你几乎看不出效果了。

Linejoin：两个线条的连接点处的外观，默认是“round”，表示圆角。

VertexColors：定义线条材质是否使用顶点颜色，这是一个boolean值。意思是，线条各部分的颜色会根据顶点的颜色来进行插值。（如果关于插值不是很明白，可以QQ问我，QQ在前言中你一定能够找到，嘿嘿，虽然没有明确写出）。

Fog：定义材质的颜色是否受全局雾效的影响。

好了，介绍完这些参数，你可以试一试了，在课后，我们会展示不同同学的杰出作品。下面，接着上面的讲，我们这里使用了顶点颜色vertexColors: THREE.VertexColors，就是线条的颜色会根据顶点来计算。

var material = new THREE.LineBasicMaterial( { vertexColors: THREE.VertexColors } );

### **3、接下来，定义两种颜色，分别表示线条两个端点的颜色，如下所示：**

var color1 = new THREE.Color( 0x444444 ),

color2 = new THREE.Color( 0xFF0000 );

### **4、定义2个顶点的位置，并放到geometry中，代码如下：**

var p1 = new THREE.Vector3( -100, 0, 100 );

var p2 = new THREE.Vector3( 100, 0, -100 );

geometry.vertices.push(p1);

geometry.vertices.push(p2);

### **5、为4中定义的2个顶点，设置不同的颜色，代码如下所示：**

geometry.colors.push( color1, color2 );

geometry中colors表示顶点的颜色，必须材质中vertexColors等于THREE.VertexColors 时，颜色才有效，如果vertexColors等于THREE.NoColors时，颜色就没有效果了。那么就会去取材质中color的值，这个很重要，大家一定记住。

### **6、定义一条线**

定义线条，使用THREE.Line类，代码如下所示：

var line = new THREE.Line( geometry, material, THREE.LinePieces );

第一个参数是几何体geometry，里面包含了2个顶点和顶点的颜色。第二个参数是线条的材质，或者是线条的属性，表示线条以哪种方式取色。第三个参数是一组点的连接方式，我们会在后面详细讲解。

然后，将这条线加入到场景中，代码如下：

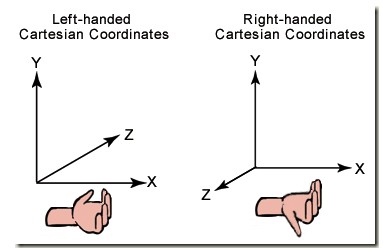
scene.add(line);

这样，场景中就会出现刚才的那条线段了。

## 4、知识补充：坐标系

我们下面会学习使用直线画一个网格出来，为了更好的理解这个网格在空间中的位置，我们是时候，讲一讲空间坐标系了。

### **1、右手坐标系**

Threejs使用的是右手坐标系，这源于opengl默认情况下，也是右手坐标系。下面是右手坐标系的图例，如果对这个概念不理解，可以百度一下，我保证你伸出手比划的那一瞬间你就明白了，如果不明白请给作者留言，我会尽快补上关于坐标系的知识。

图中右边那个手对应的坐标系，就是右手坐标系。在Threejs中，坐标和右边的坐标完全一样。x轴正方向向右，y轴正方向向上，z轴由屏幕从里向外。

## 5、线条的深入理解

在Threejs中，一条线由点，材质和颜色组成。

点由THREE.Vector3表示，Threejs中没有提供单独画点的函数，它必须被放到一个THREE.Geometry形状中，这个结构中包含一个数组vertices，这个vertices就是存放无数的点（THREE.Vector3）的数组。这个表示可以如下图所示：three.js向量

为了绘制一条直线，首先我们需要定义两个点，如下代码所示：

var p1 = new THREE.Vector3( -100, 0, 100 );

var p2 = new THREE.Vector3( 100, 0, -100 );

请大家思考一下，这两个点在坐标系的什么位置，然后我们声明一个THREE.Geometry，并把点加进入，代码如下所示：

var geometry = new THREE.Geometry();

geometry.vertices.push(p1);

geometry.vertices.push(p2);

geometry.vertices的能够使用push方法，是因为geometry.vertices是一个数组。这样geometry 中就有了2个点了。

然后我们需要给线加一种材质，可以使用专为线准备的材质，THREE.LineBasicMaterial。

最终我们通过THREE.Line绘制了一条线，如下代码所示:

var line = new THREE.Line( geometry, material, THREE.LinePieces );

ok，line就是我们要的线条了。

## 6、画高中时深爱的坐标平面

我还深爱着高中时的那个坐标平面，它勾起了我关于前排同学的细细长发的回忆…

这个平面的效果如下所示：

它横竖分别绘制了20条线段，在摄像机的照射下，就形成了这般模样。你可以在[初级教程\chapter2\2-2.html]发现这些代码：

|  |
| --- |
| <!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 = 1000; |
| camera.position.z = 0; |
| camera.up.x = 0; |
| camera.up.y = 0; |
| camera.up.z = 1; |
| camera.lookAt({ |
| x : 0, |
| y : 0, |
| z : 0 |
| }); |
| } |
|  |
| var scene; |
| function initScene() { |
| scene = new THREE.Scene(); |
| } |
|  |
| var light; |
| function initLight() { |
| light = new THREE.DirectionalLight(0xFF0000, 1.0, 0); |
| light.position.set(100, 100, 200); |
| scene.add(light); |
| } |
|  |
| var cube; |
| function initObject() { |
| var geometry = new THREE.Geometry(); |
| geometry.vertices.push( new THREE.Vector3( - 500, 0, 0 ) ); |
| geometry.vertices.push( new THREE.Vector3( 500, 0, 0 ) ); |
|  |
| for ( var i = 0; i <= 20; i ++ ) { |
|  |
| var line = new THREE.Line( geometry, new THREE.LineBasicMaterial( { color: 0x000000, opacity: 0.2 } ) ); |
| line.position.z = ( i \* 50 ) - 500; |
| scene.add( line ); |
|  |
| var line = new THREE.Line( geometry, new THREE.LineBasicMaterial( { color: 0x000000, opacity: 0.2 } ) ); |
| line.position.x = ( i \* 50 ) - 500; |
| line.rotation.y = 90 \* Math.PI / 180; |
| scene.add( line ); |
|  |
| } |
| } |
|  |
| function threeStart() { |
| initThree(); |
| initCamera(); |
| initScene(); |
| initLight(); |
| initObject(); |
| renderer.clear(); |
| renderer.render(scene, camera); |
| } |
|  |
| </script> |
| </head> |
|  |
| <body onload="threeStart();"> |
| <div id="canvas-frame"></div> |
| </body> |
| </html> |

画网格关键之处在上面的A begin 和A end 之间，我们不浪费纸，但是浪费一些点，在下面重复一下上面的代码：

|  |
| --- |
| var cube; |
| function initObject() { |
| var geometry = new THREE.Geometry(); |
| *// B begin* |
| geometry.vertices.push( new THREE.Vector3( - 500, 0, 0 ) ); |
| geometry.vertices.push( new THREE.Vector3( 500, 0, 0 ) ); |
| *// B end* |
|  |
| for ( var i = 0; i <-= 20; i ++ ) { |
|  |
| var line = new THREE.Line( geometry, new THREE.LineBasicMaterial( { color: 0x000000, opacity: 0.2 } ) ); |
| line.position.z = ( i \* 50 ) - 500; |
| scene.add( line ); |
|  |
| var line = new THREE.Line( geometry, new THREE.LineBasicMaterial( { color: 0x000000, opacity: 0.2 } ) ); |
| line.position.x = ( i \* 50 ) - 500; |
| line.rotation.y = 90 \* Math.PI / 180; |
| scene.add( line ); |
|  |
| } |
| } |

思路：我们要画一个网格的坐标，那么我们就应该找到线的点。把网格虚拟成正方形，在正方形边界上找到几个等分点，用这些点两两连接，就能够画出整个网格来。

### **1、定义2个点**

在x轴上定义两个点p1(-500,0,0)，p2(500,0,0)。

geometry.vertices.push( new THREE.Vector3( - 500, 0, 0 ) );

geometry.vertices.push( new THREE.Vector3( 500, 0, 0 ) );

### **2、算法**

这两个点决定了x轴上的一条线段，将这条线段复制20次，分别平行移动到z轴的不同位置，就能够形成一组平行的线段。

同理，将p1p2这条线先围绕y轴旋转90度，然后再复制20份，平行于z轴移动到不同的位置，也能形成一组平行线。

经过上面的步骤，就能够得到坐标网格了。代码如下：

|  |
| --- |
| for ( var i = 0; i <-= 20; i ++ ) { |
|  |
| var line = new THREE.Line( geometry, new THREE.LineBasicMaterial( { color: 0x000000, opacity: 0.2 } ) ); |
| line.position.z = ( i \* 50 ) - 500; |
| scene.add( line ); |
|  |
| var line = new THREE.Line( geometry, new THREE.LineBasicMaterial( { color: 0x000000, opacity: 0.2 } ) ); |
| line.position.x = ( i \* 50 ) - 500; |
| line.rotation.y = 90 \* Math.PI / 180; // 旋转90度 |
| scene.add( line ); |
|  |
| } |