## 1、单反、双反都是相机<

在第一课里，我向您承诺过会对相机进行一些介绍，那么今天就是我履行诺言的时刻了。嘿，这一刻，有点激动，想到相机，大学时，一直想买一个单反，但是要1万多。工作后，当一个月的工资就能买一个单反时，内心充满了骄傲和自豪。所以，各位大大们，技术还是有用的，至少技术可以用来挣钱。

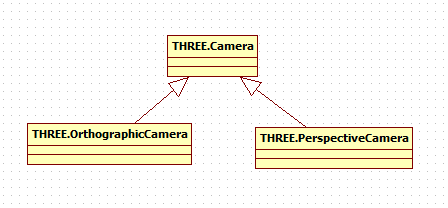
家有万贯，不如一技在手，加油。

对WebGL感兴趣，还不知道如何入门的大大们，看看本教程吧，我有信心看完本教程，特别是看完本教程的中级和高级篇，你应该对3D世界有一个自己的理解了。使用你做的绚丽的demo去找一份不做的工作，就应该没有问题。

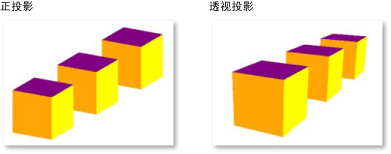
那我们今天的主题是三维空间的观察。

### **1、 认识相机**

在Threejs中相机的表示是THREE.Camera，它是相机的抽象基类，其子类有两种相机，分别是正投影相机THREE.OrthographicCamera和透视投影相机THREE.PerspectiveCamera。类图如下所示：



正投影相机有时候也叫正交投影摄像机，下图显示了正交摄像机投影和透视投影之间的差别。



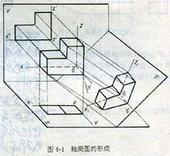
### **2、 两者的区别**

正投影和透视投影的区别是：透视投影有一个基本点，就是远处的物体比近处的物体小。我们看看著名的蒙娜丽莎的微笑就是典型的透视作品：



观察上图，对，没错是凤姐，主要不是叫您看它，是看它后面的树木，它是不是越来越小呢？对，这就是透视。

在工程建筑领域，正投影的例子很多，例如下面就是一个正投影的例子：



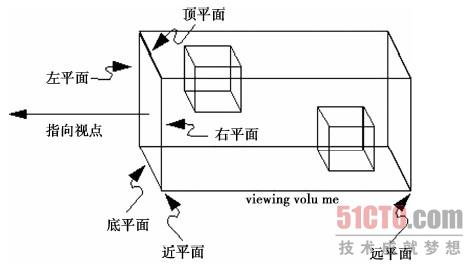
其特点是，远近高低比例都相同。

## 2 正投影相机

下面我们来介绍正投影相机，正投影的构造函数如下所示：

OrthographicCamera( left, right, top, bottom, near, far )

结合下面一个图，我们来看看，各个参数的意思。



介绍参数之前，先假定一个相机中心点，相机中心点可以想成是镜头的中心点。为了让大家能更容易的明白，我还是上一幅图吧，虽然这样会多花我一点时间。



图中红点就是我们假设的相机中心点。下面介绍一下构造函数的参数：

1、 left参数

left：左平面距离相机中心点的垂直距离。从图中可以看出，左平面是屏幕里面的那个平面。

2、 right参数

right：右平面距离相机中心点的垂直距离。从图中可以看出，右平面是屏幕稍微外面一点的那个平面。

3、 top参数

top：顶平面距离相机中心点的垂直距离。上图中的顶平面，是长方体头朝天的平面。

4、 bottom参数

bottom：底平面距离相机中心点的垂直距离。底平面是头朝地的平面。

5、near参数

near：近平面距离相机中心点的垂直距离。近平面是左边竖着的那个平面。

6、far参数

far：远平面距离相机中心点的垂直距离。远平面是右边竖着的那个平面。

有了这些参数和相机中心点，我们这里将相机的中心点又定义为相机的位置。通过这些参数，我们就能够在三维空间中唯一的确定上图的一个长方体。这个长方体也叫做视景体。

投影变换的目的就是定义一个视景体，使得视景体外多余的部分裁剪掉，最终图像只是视景体内的有关部分。

好了，看一个简单的例子：

var camera = new THREE.OrthographicCamera( width / - 2, width / 2, height / 2, height / - 2, 1, 1000 );

scene.add( camera );

这个例子将浏览器窗口的宽度和高度作为了视景体的高度和宽度，相机正好在窗口的中心点上。这也是我们一般的设置方法，基本上为了方便，我们不会设置其他的值。

## 3、透视投影相机

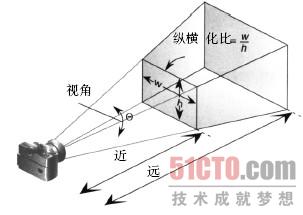
透视投影是更符合我们视觉的投影，当你凝视一个女人时，就是因为远近高低各不同，女人才显得美丽，叫你看一个正投影的女人，估计连胸部都分不清，那么也没什么看头。

正因为，透视相机这么有魅力，所以在各种应用中运用非常广泛。

透视投影相机的构造函数如下所示：

PerspectiveCamera( fov, aspect, near, far )

我们来欣赏一幅图来看看这个函数的各个参数的意思：



先来明确这个图里涉及的概念。很多作者都认为这些概念很简单，不需要讲解，但是其实正是这些简单的东西，让很多初学者不明白。所以我一直想把这些简单的内容给讲清楚，以至于大家不在这个上面花费过多的时间，毕竟多的时间可以去挣钱，可以去陪女朋友，去做超级奶爸。

1、视角fov：这个最难理解,我的理解是,眼睛睁开的角度,即,视角的大小,如果设置为0,相当你闭上眼睛了,所以什么也看不到,如果为180,那么可以认为你的视界很广阔,但是在180度的时候，往往物体很小，因为他在你的整个可视区域中的比例变小了。

2、近平面near：这个呢，表示你近处的裁面的距离。补充一下，也可以认为是眼睛距离近处的距离，假设为10米远，请不要设置为负值，Three.js就傻了,不知道怎么算了,

3、远平面far：这个呢，表示你远处的裁面,

4、纵横比aspect：实际窗口的纵横比，即宽度除以高度。这个值越大，说明你宽度越大，那么你可能看的是宽银幕电影了，如果这个值小于1，那么可能你看到的是如下的图中的LED屏幕了。



好了，看看下面一个简单的例子：

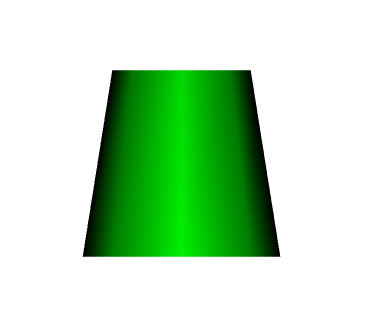
var camera = new THREE.PerspectiveCamera( 45, width / height, 1, 1000 );

scene.add( camera );

这个例子，很明了，就不在解释了，如有不明白，那请积极问我QQ吧。

## 4、实例

接下来，结合上面讲的两种相机，我们来看一个实例。这个实例首先使用正投影相机，然后在使用透视相机。先看看正投影相机的效果：



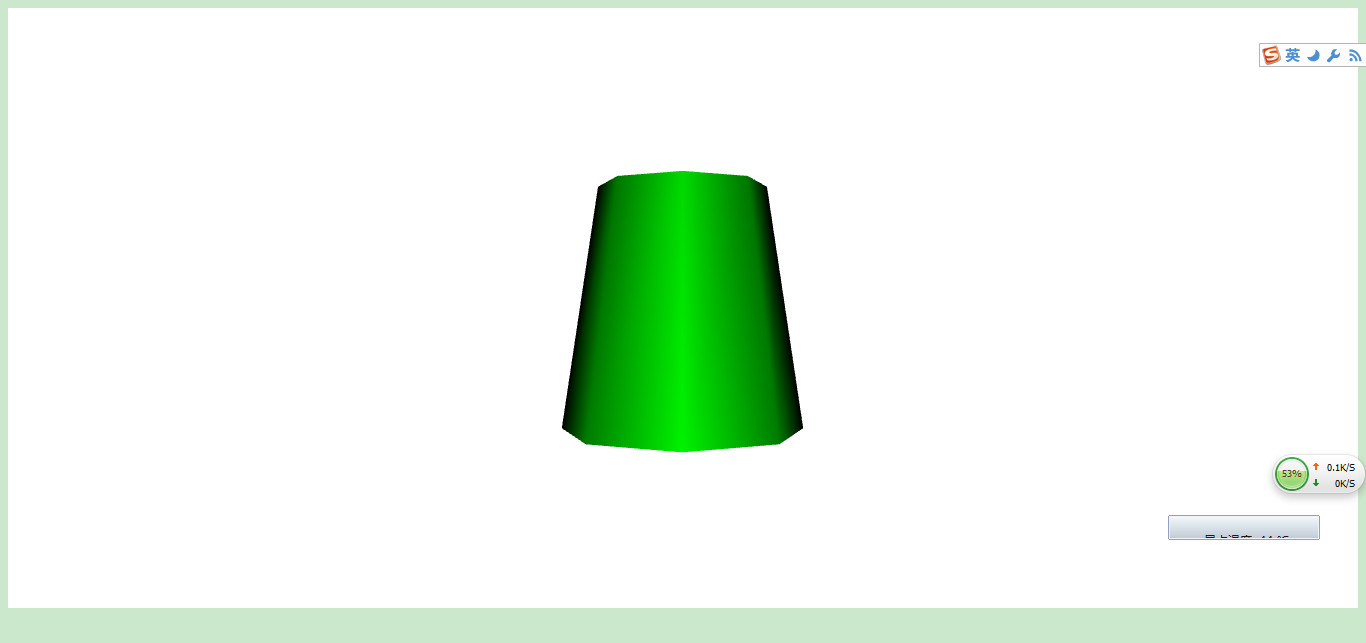
从图中可以看出，它基本上各个方向大小都相同，没有透视的效果。 我们来看看这一段代码，你可以从4-1.html下载本实例。

|  |
| --- |
| <!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 = new THREE.OrthographicCamera( window.innerWidth / - 2, window.innerWidth / 2, window.innerHeight / 2, window.innerHeight / - 2, 10, 1000 ); |
| 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; |
| function initObject() { |
| var geometry = new THREE.CylinderGeometry( 70,100,200); |
| var material = new THREE.MeshLambertMaterial( { color:0xFFFFFF} ); |
| 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() |
| { |
| changeFov(); |
| renderer.render(scene, camera); |
| requestAnimationFrame(animation); |
| } |
|  |
| function setCameraFov(fov) |
| { |
| camera.fov = fov; |
| camera.updateProjectionMatrix(); |
| } |
|  |
| function changeFov() |
| { |
| var txtFov = document.getElementById("txtFov").value; |
| var val = parseFloat(txtFov); |
| setCameraFov(val); |
| } |
| </script> |
| </head> |
|  |
| <body onload="threeStart();"> |
| <div id="canvas-frame"></div> |
| <div> |
| Fov:<input type="text" value="45" id="txtFov"/>(0到180的值) |
| </div> |
| </body> |
| </html> |
|  |

明白了正投影的效果，我们现在将相机变成透视投影，只要更改上面关于相机的代码，就可以了，这里我们变成如下的代码：

camera = new THREE.PerspectiveCamera(45, width / height, 1, 10000);

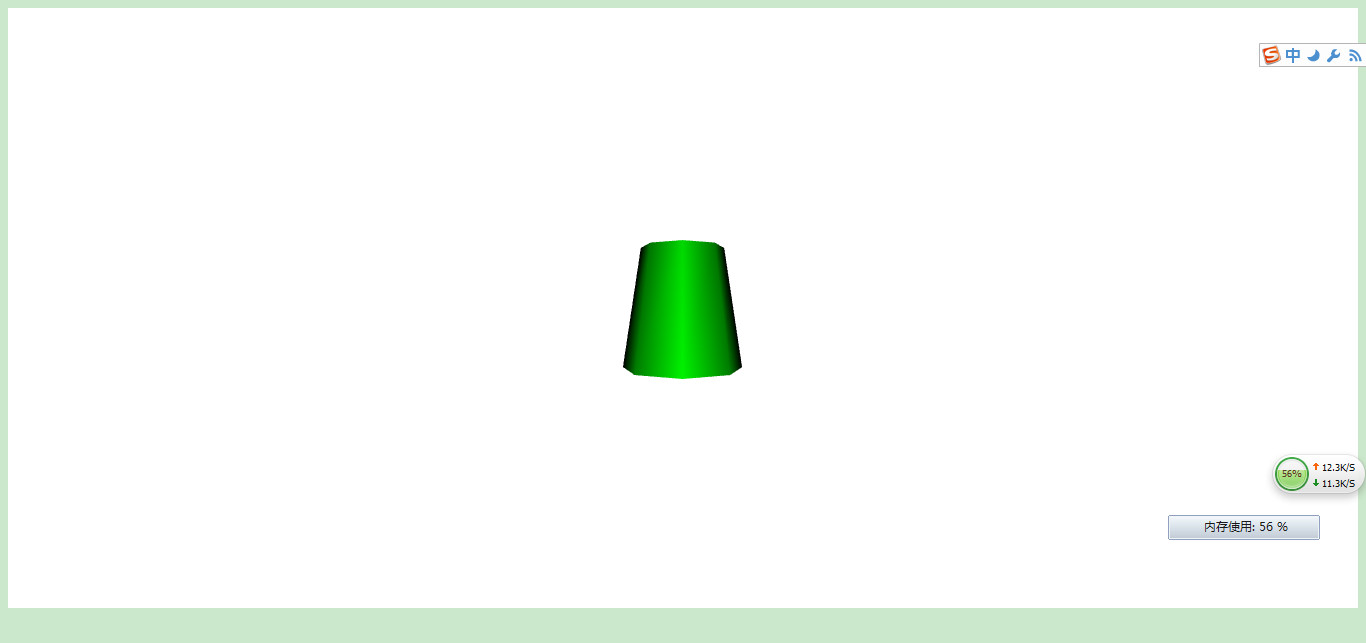
效果如下:



这是视角为45度的情况，也就是眼睛睁开45度的情况。人类的正常视角是120度左右，但是要集中注意力看清楚东西，那么眼睛的视角在30-40度比较好。

这里我们分别展示视角设置为80度，100度，120度，160度和179度时，看到场景的情况：

80度视角效果图如下:



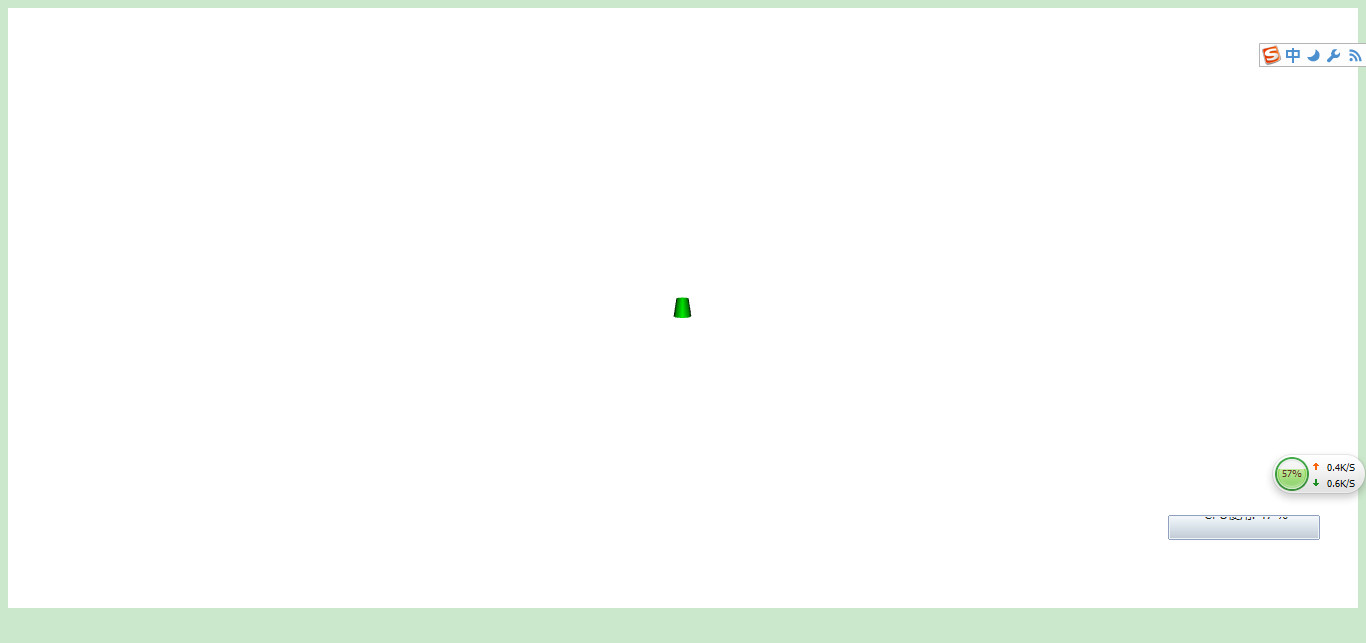
100度视角效果图如下:



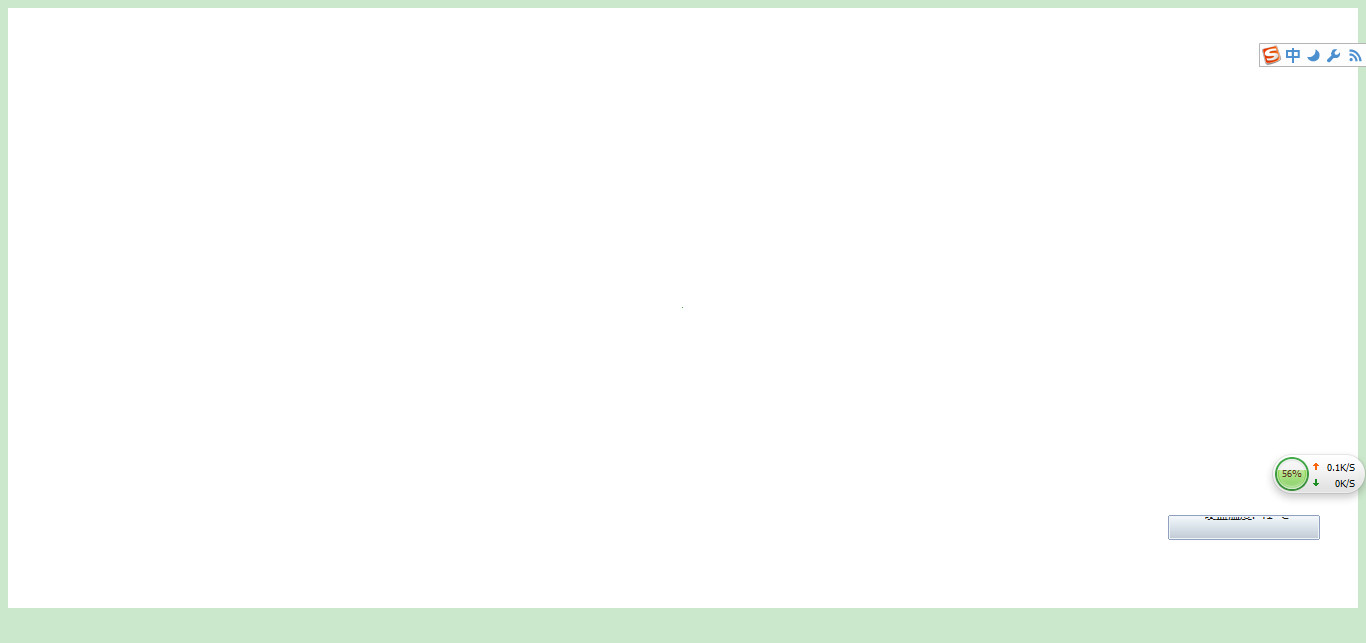
120度视角效果图如下:



160度视角效果图如下:



179度视角效果图如下:



ok，我们已经将主要的视角大小给搞定了。反复对照上面的图，你会发现，视角越大，中间的物体越小，这是因为，视角越大，看到的场景越大，那么中间的物体相对于整个场景来说，就越小了。

你还可以试一试睁大您的眼睛，努力挣得最大，你发现周围的物体看不清了，这就是眼大不清的原理，你无法集中注意力，而且你视图看到你前面的所有物体，你的焦距无法固定，所以场景非常模糊。

虽然你也许感觉不了非常明显，你前面的某一件物体确实缩小了，但在计算机固定大小的屏幕上，显示更多更大的场景，毫无疑问，每一件物体显示是缩小了。

当到达179度的时候，three.js真的傻了，他已经完全不明白你要看什么了，他已经将你要看的场景设为无穷大了，所以每一件物体相对于无穷大来说，基本在屏幕中无法显示了。