第二节课 threejs三要素学习并通过最简单的MESH创建地板

1、搭建threejs三要素

让我们先看一下一套最简单的threejs代码，类似于我们学习任何一种语言的helloWord。

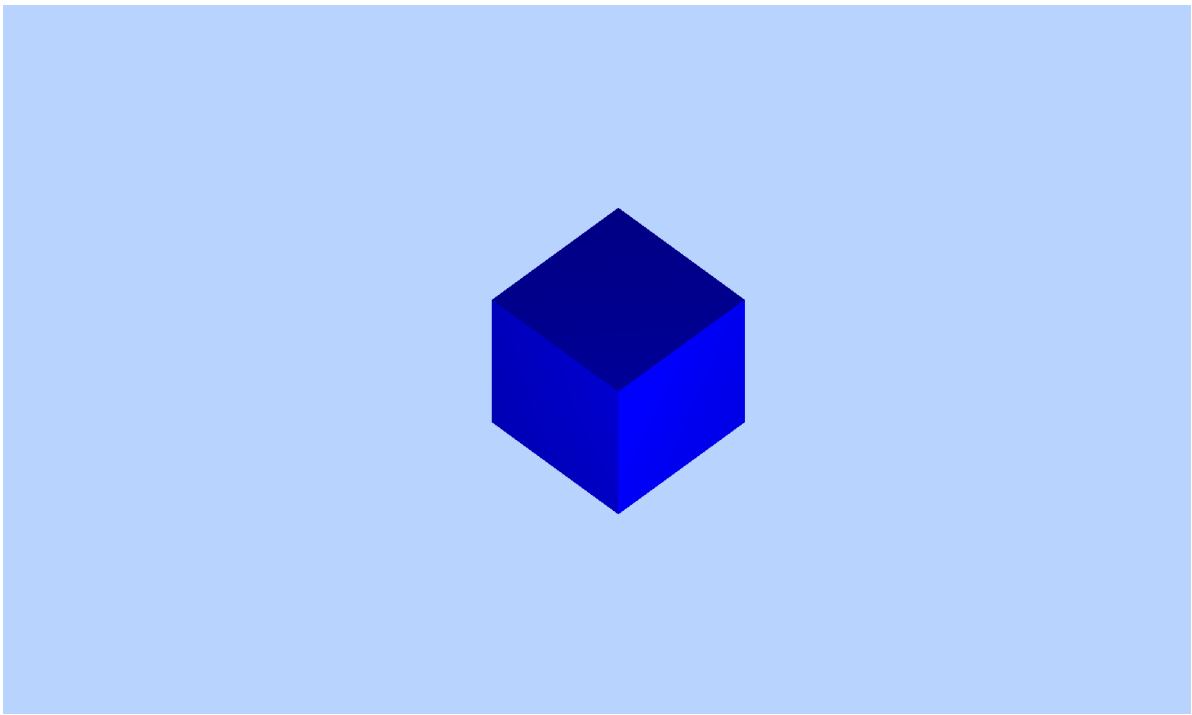
```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <title>第一个three.js文件_WebGL三维场景</title>
  <style>
    body {
      margin: 0;
      overflow: hidden;
      /* 隐藏body窗口区域滚动条 */
    }
  </style>
  <!--引入three.js三维引擎-->
  <script
src="http://www.yanhuangxueyuan.com/versions/threejsR92/build/three.js">
</script>
  <!-- <script src="./three.js"></script> -->
  <!-- <script src="http://www.yanhuangxueyuan.com/threejs/build/three.js">
</script> -->
</head>

<body>
  <script>
    /**
     * 创建场景对象Scene
     */
    var scene = new THREE.Scene();
    /**
     * 创建网格模型
     */
    // var geometry = new THREE.SphereGeometry(60, 40, 40); //创建一个球体几何对象
    var geometry = new THREE.BoxGeometry(100, 100, 100); //创建一个立方体几何对象
Geometry
    var material = new THREE.MeshLambertMaterial({
      color: 0x0000ff
    }); //材质对象Material
    var mesh = new THREE.Mesh(geometry, material); //网格模型对象Mesh
    scene.add(mesh); //网格模型添加到场景中
    /**
     * 光源设置
     */
    //点光源
    var point = new THREE.PointLight(0xffffff);
    point.position.set(400, 200, 300); //点光源位置
    scene.add(point); //点光源添加到场景中
```

```
    //环境光
    var ambient = new THREE.AmbientLight(0x444444);
    scene.add(ambient);
    // console.log(scene)
    // console.log(scene.children)
    /**
     * 相机设置
     */
    var width = window.innerWidth; //窗口宽度
    var height = window.innerHeight; //窗口高度
    var k = width / height; //窗口宽高比
    var s = 200; //三维场景显示范围控制系数，系数越大，显示的范围越大
    //创建相机对象
    var camera = new THREE.OrthographicCamera(-s * k, s * k, s, -s, 1, 1000);
    camera.position.set(200, 300, 200); //设置相机位置
    camera.lookAt(scene.position); //设置相机方向(指向的场景对象)
    /**
     * 创建渲染器对象
     */
    var renderer = new THREE.WebGLRenderer();
    renderer.setSize(width, height);//设置渲染区域尺寸
    renderer.setClearColor(0xb9d3ff, 1); //设置背景颜色
    document.body.appendChild(renderer.domElement); //body元素中插入canvas对象
    //执行渲染操作    指定场景、相机作为参数
    renderer.render(scene, camera);
  </script>
</body>
</html>
```

结果如下图所示：



我们知道THREEJS中三要素是：场景（scene）、相机（camera）和渲染器（renderer）。有了这三样东西，才能将物体渲染到网页中去。

2、场景

在Threejs中场景就只有一种，用THREE.Scene来表示，要构件一个场景也很简单，只要new一个对象就可以了，代码如下：

```
var scene = new THREE.Scene();
```

场景是所有物体的容器，如果要显示一个苹果，就需要将苹果对象加入场景中。只需要记住如下几个方面就可以：

属性

属性	描述
children	数组，用于存储添加到场景中的所有对象
fog	雾化，雾化效果的特点是场景中的物体离得越远就会变得越模糊，有三个参数：雾的颜色，最近距离，最远距离
overrideMaterial	材质覆盖，当设置了该属性后，场景中所有的物体都会使用该属性指向的材质，即使物体本身也设置了材质

详细效果请参加项目示例代码。

方法

方法	描述
Add()	向场景中添加对象
Remove()	移除场景中的对象
getObjectByName()	获取场景中指定名称的对象
tranverse()	以一个方法作为参数，这个方法将会在每一个子对象上执行。如果子对象本身还有子对象，该方法将会在所有的子对象上执行，直到遍历完场景树中的所有对象为止

3、相机

相机决定了场景中那个角度的景色会显示出来。相机就像人的眼睛一样，人站在不同位置，抬头或者低头都能够看到不同的景色。

场景只有一种，但是相机却又很多种。和现实中一样，不同的相机确定了呈相的各个方面。比如有的相机适合人像，有的相机适合风景，专业的摄影师根据实际用途不一样，选择不同的相机。对程序员来说，只要设置不同的相机参数，就能够让相机产生不一样的效果。当前我只需要关注两种相机即可。分别是正投影相机THREE.OrthographicCamera和透视投影相机THREE.PerspectiveCamera。

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。

正投影

透视投影

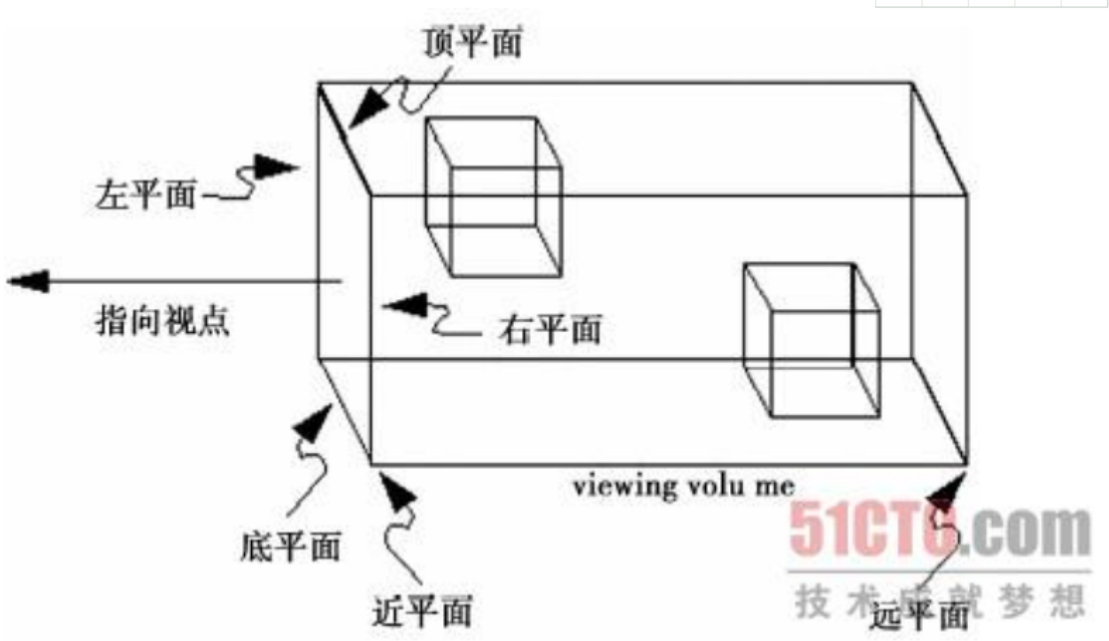
详细效果请参加项目示例代码。

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。



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： 渲染空间的左边界

2、 right参数

right：渲染空间的右边界

3、 top参数

top：渲染空间的上边界

4、 bottom参数

bottom：渲染空间的下边界

5、near参数

near：near属性表示的是从距离相机多远的位置开始渲染，一般情况会设置一个很小的值。 默认值0.1

6、far参数

far：far属性表示的是距离相机多远的位置截止渲染，如果设置的值偏小小，会有部分场景看不到。 默认值1000

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 );
```

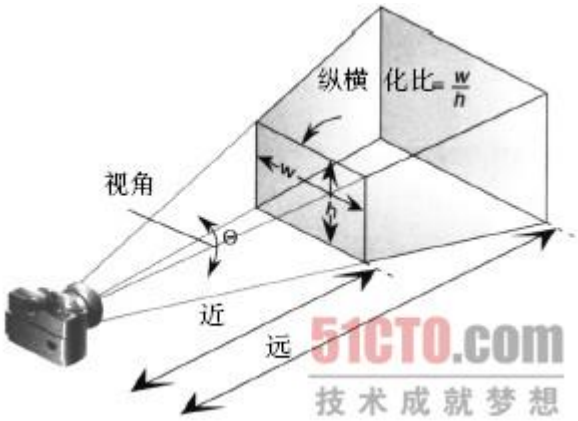
这个例子将浏览器窗口的宽度和高度作为了视景体的高度和宽度，相机正好在窗口的中心点上。这也是我们一般的设置方法，基本上为了方便，我们不会设置其他的值。

透视投影相机

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：



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，

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

scene.add( camera );
```

4、渲染器

Three.js中的场景是一个物体的容器，开发者可以将需要的角色放入场景中，例如苹果，葡萄。同时，角色自身也管理着其在场景中的位置。

相机的作用就是面对场景，在场景中取一个合适的景，把它拍下来。

渲染器的作用就是将相机拍摄下来的图片，放到浏览器中去显示。他们三者的关系如下图所示：

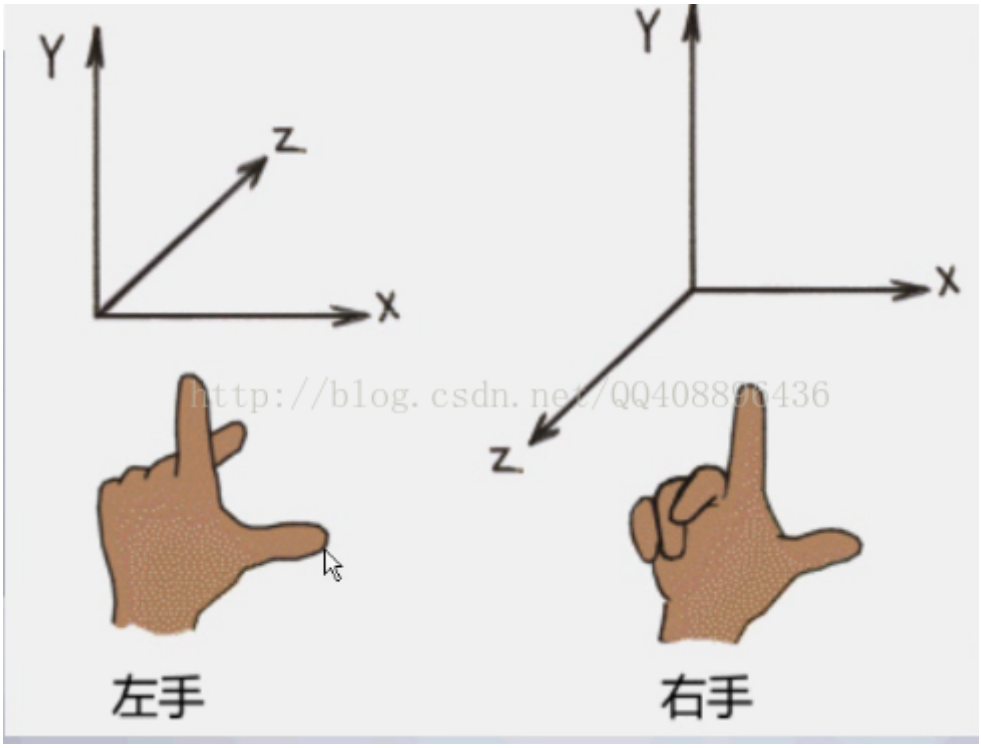
我们用new THREE.WebGLRenderer()来新建一个WebGL渲染器。

属性	值	含义
antialias	true/false	是否开启反锯齿，设置为true开启反锯齿。
alpha	true/false	是否可以设置背景色透明。
maxLights	数值int	最大灯光数，我们的场景中最多能够添加多少个灯光。
logarithmicDepthBuffer	true/false	模型的重叠部位不停的闪烁。这便是Z-Fighting问题，为解决这个问题，我们可以采用该种方法(详细解释大家感兴趣参照https://www.cnblogs.com/lst619247/p/9098845.html)

方法	含义
setSize	制定渲染器的宽高，renderer.setSize(width,height)
setClearColor	设置canvas背景色(clearColor)和背景色透明度（clearAlpha）
setPixelRatio	设置分辨率，解决场景模糊，抗锯齿的一种很好的方法

5、关于threejs中的坐标系

三维坐标系分两种，左手坐标系和右手坐标系。如下图：



左手坐标系和右手坐标系的区别只是Z轴的方向不同而已。而threejs中采用的是右手坐标系。

6、关于threejs中的灯光

光源种类	含义
环境光(AmbientLight)	笼罩在整个空间无处不在的光，环境光可以说是场景的整体基调，由于环境光无处不在，也就是说它是没有方向的，当然不能产生阴影。而且，它也不能作为环境中唯一的光源。后果就是所有的物体都是黑的（效果见实例），具体作用就是弱化阴影或者给场景添加一些颜色，所以设置的时候只需要设置一些颜色即可。
点光源(PointLight)	向四面八方发射的单点光源，可以将点光源想象成萤火虫一样发出的光。由于它的光线也发射到四面八方，在ThreeJS中它也是不能产生阴影的。如何理解不产生阴影？知道了什么是PointLight，你就能理解为什么不产生阴影了，原因是这样的光源会朝着所有的方向发射光线，在这种情况下计算阴影对GPU来讲是一个非常沉重的负担，所以不能产生阴影，你愿意你的应用慢如蜗牛吗，当然有其他光源可以替代，就比如SpotLight
聚光灯(SpotLight)	锥形效果的光源，能够产生阴影
平行光(DirectinalLight)	平行光，类似太阳光，距离很远的光，如太阳般，照射到地球上每一束光都是平行的。所有对象接收的光强都是一样的，会产生阴影，与聚点光源的主要差别是：聚点光源光距离目标越远光越暗淡，而平行光光强都是一样的。用direction(方向)，color(颜色)，intensity(强度)来计算属性和阴影，形成的不是光锥而是一个方块，很重要

各种光源的使用方法和属性

光源种类	属性	代码示范
环境光(AmbientLight)	颜色	var ambientLight = new THREE.AmbientLight("#0c0c0c");// *设置颜色*

点光源(PointLight)	color:光源颜色
	intensity：强度，光照的强度，默认为1
	distance：距离， 光照照射得到的距离，决定光可以照射多远
	position：光源所在的位置
	visible：可见性，如果为true，光源就会打开
		var pointColor = "#ccffcc";
		var pointLight = new THREE.PointLight(pointColor);
		pointLight.distance = 100;
		pointLight.intensity = 1;
		scene.add(pointLight);

聚光灯(SpotLight)	**制造产生阴影的几个步骤,让球体和方块将阴影投影到地上，哪些物体投射阴影，哪些物体接受阴影**
	render.shadowMapEnabled=true;//告诉render我们需要阴影(允许阴影隐射)
	plane.receiveShadow=true;//地面接受阴影
	cude.castShadow=true;//cast投射，就是方块投射阴影
	spotLight.castShadow=true;不是所有的光源都可以投射阴影 ,这里使用聚点光源可以产生阴影
	castShadow：为true则该光源会产生阴影
	color — 光的颜色值，十六进制，默认值为0xffffff.
	intensity — 光的强度，默认值为1.
	distance — 光照距离，默认为0，表示无穷远都能照到.
	angle — 圆椎体的半顶角角度，最大不超过90度，默认为最大值。
	penumbra — 光照边缘的模糊化程度，范围0-1，默认为0，不模糊
	decay — 随着光的距离，强度衰减的程度，默认为1，为模拟真实效果，建议设置为2
		var pointColor = "#ffffff";
		var spotLight = new THREE.SpotLight(pointColor);
		spotLight.position.set(-40, 60, -10);
		spotLight.castShadow = true;
		spotLight.shadowCameraNear = 2;
		spotLight.shadowCameraFar = 200;
		spotLight.shadowCameraFov = 30;
		spotLight.target = plane;//光照照向地面
		spotLight.distance = 0;
		spotLight.angle = 0.4;
		scene.add(spotLight);

平行光(DirectinalLight)	color — 光的颜色值，十六进制，默认值为0xffffff.
	intensity — 光的强度，默认值为1.
	target - 平行光的方向
		var pointColor = "#ff5808";
		var directionalLight = new THREE.DirectionalLight(pointColor);
		directionalLight.intensity = 0.5;

7、通过我们学到的知识来搭建地板模型

首先我们添加一个可以绘制立方体的类

```
function Cube(option){
    this.length = option.length || 1;
    this.width = option.width || 1;
    this.height = option.height || 1;

    this.Name = option.objName;
```

```
    this.positionX = option.position.x || 0;
    this.positionY = option.position.y || 0;
    this.positionZ = option.position.z || 0;

    this.style=option.style||{color:0xFF0000};
    let
curmaterial=CommonFunction.createMaterial(this.width,this.height,this.style);

    let cubeGeometry = new THREE.BoxGeometry(this.length, this.height,
this.width);

    let cube = new THREE.Mesh( cubeGeometry, curmaterial );
    cube.name=this.Name;
    cube.position.x=this.positionX;
    cube.position.y=this.positionY;
    cube.position.z=this.positionZ;
    return cube;
}
```

可以通过该类设置具体立方体的长宽高，位置，以及颜色等等。

初始化我们对应的三要素

场景：

```
/**
 初始化场景，仅仅需要有句话就可以生命一个场景，非常简单
 **/
Store3D.prototype.initScene = function () {
    this.scene = new THREE.Scene();
},
```

相机：

```
/**
 初始化场景，因为我们做的工厂模型，尽可能的接近于真实情景，采用透视相机
 **/
Store3D.prototype.initCamera = function () {
    //声明一个透视相机，
    // 视角：60，
    // 纵横比aspect:全屏，使用的是浏览器的宽度/高度
    //近平面near：0.1
    //远平面视角far:10000
    this.camera = new THREE.PerspectiveCamera(60, window.innerWidth /
window.innerHeight, 0.1, 10000);
    /*
    设置相机位置，注意threejs中的坐标系采用的是右手坐标系
     */
    this.camera.position.x = 0;
    this.camera.position.y = 1600;
    this.camera.position.z = 1000;
    //相机的朝向
    this.camera.lookAt(0, 0, 0);
    //将相机放到场景中
    this.scene.add(this.camera);
},
```

渲染器

```
/**
 声名渲染器
 **/
Store3D.prototype.initRenderer = function () {
    this.renderer = new THREE.WebGLRenderer(
        {
            antialias: true,//是否开启反锯齿，设置为true开启反锯齿。
            alpha: true,//是否可以设置背景色透明。
            logarithmicDepthBuffer: true//模型的重叠部位便不停的闪烁起来。这便是Z-Fighting问题，为解决这个问题，我们可以采用该种方法
        }
    );
    this.renderer.setSize(window.innerWidth, window.innerHeight);//渲染器的尺寸与windows的尺寸相同
    this.renderer.setClearColor(0x39609B);//设置渲染的背景颜色
    this.renderer.setPixelRatio(window.devicePixelRatio);//设置渲染器的分辨率与浏览器电脑本身的分辨率相同
    //将渲染器添加到我们的网页中，可以将渲染的内容在网页中显示出来
    let container = document.getElementById("container");
    container.appendChild(this.renderer.domElement);
},
```

添加灯光

```
Store3D.prototype.initLight=function(){
    //首先添加个环境光
    let ambient = new THREE.AmbientLight(0xffffff, 1); //AmbientLight,影响整个场景的光源
    ambient.position.set(0, 0, 0);
    this.addObject(ambient);
    //添加平行光,平行光类似于太阳光
    let directionalLight = new THREE.DirectionalLight(0xffffff, 0.3);//模拟远处类似太阳的光源
    directionalLight.position.set(0, 200, 0);
    this.addObject(directionalLight);
    //设置点光源
    let pointLight1=new THREE.PointLight(0xffffff, 0.3);
    pointLight1.position.set(-500,200,0);
    this.addObject(pointLight1);
    let pointLight2=new THREE.PointLight(0xffffff, 0.3);
    pointLight2.position.set(500,200,0);
    this.addObject(pointLight2);
},
```

最后将我们新建的cube类，加入到我们的场景中，当然高度设置的比较低，类似于一个薄片，这样就可以了。