# 2.1 选择合适的相机

Three.js库里有两种不同的相机：正交投影相机和透视相机。

本例中的截图如下：



这是一个透视视图，也是最自然的视图。距离相视越远的方块，被渲染得越小。

如果使用右上方的按钮切换到正交投影相机，程序截图如下：



使用正交投影相机时，所有方块渲染出来的大小都一样；对象和相机之间的距离不会影响渲染结果。这种相机通常用在二维游戏中。

在三维游戏中，应尽量使用透视相机，因为它最贴近真实世界。切换相机的代码如下：

this.switchCamera = function() {

 if (camera instanceof THREE.PerspectiveCamera) {

 camera = new THREE.OrthographicCamera( window.innerWidth / - 16, window.innerWidth / 16, window.innerHeight / 16, window.innerHeight / - 16, -200, 500 );

 camera.position.x = 2;

 camera.position.y = 1;

 camera.position.z = 3;

 camera.lookAt(scene.position);

 this.perspective = "Orthographic";

 } else {

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

 camera.position.x = 120;

 camera.position.y = 60;

 camera.position.z = 180;

 camera.lookAt(scene.position);

 this.perspective = "Perspective";

 }

};

在上面这个代码片段里你可以看到，我们创建THREE.PerscpectiveCamera的方法跟创建THREE.OrthographicCamera的方法有些不一样的地方．我们先来看一下THREE.PerspectiveCamera。它接受如下这些参数：

|  |  |
| --- | --- |
| **参数** | **描述** |
| fov（视场） | fov表示视场（field of view）。这是从相机位置能够看到的部分场景。例如，人类有差不多180度的视场，而一些鸟类差不多会有一个完整的、360度的视场。由于普通的显示器不能完全显示我们看到的景象，所以一般会选择一块较小的区域。对大多数情况下会用60度到90度左右的视场。推荐默认值：45 |
| aspect（长宽比） | 这是渲染结果输出区的横向长度和纵向长度的比值。在我们的例子中，由于我们会使用整个窗口作为输出界面，所以会使用这个窗口的长宽比。这个长宽比决定了水平视场和垂直视场之间的比例关系。推荐默认值：window.innerWidth/window.innerHeight |
| near（近裁平面） | 从距离相机多近的地方开始渲染场景。通常情况下我们会为这个属性设置一个很小的值，从而可以渲染从相机位置可以看到的所有物体。默认值：0.1。 |
| far（远裁平面） | 相机可以从它所处的位置看多远。如果我们将这个值设得太低，那么场景中的一部分可能不会被渲染；如果设得太高，在某些情况下会影响渲染的效率。默认值：1000。 |

下图展示了这些属性：



要配置正交投影相机，我们得使用其他一些属性。正投影不关心使用什么样的长宽比，或者以什么样的视角观察场景。所有对象渲染的尺寸都一样。下表是正交投影相机的属性：

|  |  |
| --- | --- |
| **参数** | **描述** |
| left（左边界） | 可视范围的左平面。你可以将它当做是可渲染部分的左侧边界。如果我们把这个值设为-100，那么你就不会看到任何比这个左侧边界更远的对象 |
| right（右边界） | 跟left属性一样，不过这次是界面的另外一侧。比这个右侧边界更远的对象不会被渲染 |
| top（上边界） | 可被渲染空间的最上面 |
| bottom（下边界） | 可被渲染空间的最下面 |
| near（近裁平面） | 基于相机所在的位置，从这一点开始渲染场景 |
| far（远裁平面） | 基于相机所在的位置，一直渲染到场景中的这一点 |

所有这些属性可以在下图中找到：



## 设定相机的观察目标

我们也可以使用下面的代码设定相机的观察目标：

camera.lookAt(new THREE.Vector3(x,y,z));

在这个例子中，相机的观察目标可以移动，它所指向的位置用一个红点标识了出来。你会看到场景正在从左向右移动。场景并不是真的在移动，而是相机的观察目标在移动（屏幕中央的那个红点），其效果就是场景在从左向右移动。你也可以在这个例子将相机换成正交投影相机。你会看到改变相机观察目标的效果跟使用透视相机时有所不同。

## 完整代码

<!DOCTYPE html>

<html>

<head>

 <title>示例02.01 - 相机</title>

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

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

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

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

 <style>

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

 body {

 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);

 // 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();

 // 创建相机

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

 camera.position.x = 120;

 camera.position.y = 60;

 camera.position.z = 180;

 // 创建一个Plane作为地面

 var planeGeometry = new THREE.PlaneBufferGeometry(180,180);

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

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

 // 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);

 // 添加绿色盒子

 var boxGeometry = new THREE.BoxGeometry(8,4,8);

 var boxMaterial = new THREE.MeshLambertMaterial({ color: 0x00ee22 });

 for (var j = 0 ; j < (planeGeometry.parameters.height / 9) ; j++) {

 for (var i = 0 ; i < planeGeometry.parameters.width / 9 ; i++) {

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

 box.position.z = -((planeGeometry.parameters.height) / 2) + 2 + (j \* 9);

 box.position.x = -((planeGeometry.parameters.width) / 2) + 2 + (i \* 9);

 box.position.y=2;

 scene.add(box);

 }

 }

 // 添加一个红色球作为相机的观察目标

 var lookAtGeom = new THREE.SphereGeometry(2);

 var lookAtMesh = new THREE.Mesh(lookAtGeom, new THREE.MeshLambertMaterial({ color: 0xff0000 }));

 scene.add(lookAtMesh);

 // 添加平行光

 var directionalLight = new THREE.DirectionalLight( 0xffffff, 0.7 );

 directionalLight.position.set( -20, 40, 60 );

 scene.add(directionalLight);

 // 添加环境光

 var ambientLight = new THREE.AmbientLight(0x292929);

 scene.add(ambientLight);

 // 切换相机

 var controls = new function() {

 this.perspective = "Perspective";

 this.switchCamera = function() {

 if (camera instanceof THREE.PerspectiveCamera) {

 camera = new THREE.OrthographicCamera( window.innerWidth / - 16, window.innerWidth / 16, window.innerHeight / 16, window.innerHeight / - 16, -200, 500 );

 camera.position.x = 2;

 camera.position.y = 1;

 camera.position.z = 3;

 camera.lookAt(scene.position);

 this.perspective = "Orthographic";

 } else {

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

 camera.position.x = 120;

 camera.position.y = 60;

 camera.position.z = 180;

 camera.lookAt(scene.position);

 this.perspective = "Perspective";

 }

 };

 }

 var gui = new dat.GUI();

 gui.add(controls, 'switchCamera');

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

 render();

 var step = 0;

 function render() {

 stats.update();

 step += 0.02;

 // 移动相机的观察目标

 if (camera instanceof THREE.PerspectiveCamera) {

 var x = 10 + (100 \* (Math.sin(step)));

 camera.lookAt(new THREE.Vector3(x, 30, 0));

 lookAtMesh.position.set(x, 10, 0);

 } else {

 var x = ((Math.cos(step)));

 camera.lookAt(new THREE.Vector3(x, 0, 0));

 lookAtMesh.position.set(x, 10, 0);

 }

 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>