# 1.1 Three.js基本框架

在本网站[第1课 绘制一个三角形](http://shiba.hpe.sh.cn/jiaoyanzu/wuli/ShowArticle.aspx?articleId=1248)等一系列文章中，使用的都是WebGL原生API来编程。但之后就应该自己编写一个代码库，以用于随后的通用3D编程，这样就可以将相对复杂的原生API封装起来，提高编程效率。

现在有很多不错的WebGL开源框架，Three.js就是其中的一个，它是一个3D JavaScript库，封装了底层的图形接口，使得程序员能够在无需掌握繁冗的图形学知识的情况下，也能用简单的代码实现三维场景的渲染。它是由西班牙程序员Ricardo Cabello Miguel开发的，此人更出名的网名为Mr.doob。

Three.js在地址为<http://mrdoob.github.io/three.js/>，在那儿你可以下载到全部源代码、文档和示例，我使用的版本为r71。

本文会建立一个Three.js基本框架，以后的示例都是在这个框架的基础上实现的。我们会将“第1课 绘制一个三角形”的步骤用这个框架再实现一次。接下来的绝大多数示例都参考自《Learning Three.js：The Javascript 3D Library for WebGL》，在本网站可下载到此书的电子英文版和源代码。

## 基本框架

在WebGL原生API中，要在页面上生成内容，大致需要以下几步：

* 创建WebGL上下文
* 创建顶点缓冲（和索引缓冲）
* 创建顶点着色器和片段着色器
* 编译着色器
* 创建程序对象和链接着色器
* 绘制场景

而要构建一个Three.js框架，必须拥有三样东西：

* + 渲染器（renderer）对象：负责创建WebGL上下文，绘制场景。
	+ 场景（scene）对象：场景是最上级的对象，用于容纳所有的其他图形对象（网格、相机等）。
	+ 相机（camera）对象：定义我们在渲染好的scene中能够看到些什么。

下面就是网页代码：

<!DOCTYPE html>

<html>

<head>

 <title>示例01.01 – 基本框架</title>

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

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

 <style>

 body{

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

 margin: 0;

 overflow: hidden;

 }

 </style>

</head>

<body>

 <!-- 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(0x00204d);

 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);

 // 设置相机位置，观察目标设置为场景中央

 camera.position.x = 0;

 camera.position.y = 0;

 camera.position.z = 3;

 camera.lookAt(scene.position);

 renderer.render(scene, camera);

 });

</script>

</body>

</html>

以上代码还使用了jquery库，这不是必须的，但可以让某些DOM操作更加便捷。

若现在运行程序，只会显示墨绿色的背景，我们需要在场景中添加一个黄色的三角形。

## 添加网格（Mesh）对象

Three.js中绘制的绝大多数对象都是网格（Mesh），一个Mesh通常是由一个几何体（Geometry）对象和一个材质（Material）对象组合而成。

其中Geometry对象保存了Mesh对象的顶点信息和索引信息，而Material对象保存与渲染效果相关的属性。通过设置材质可以改变物体的颜色、纹理贴图、光照模式等。

Three.js并没有内置三角形的几何体对象，所以我们需要从几何体基类Geometry自己定义顶点和索引。代码如下：

var triangleGeometry = new THREE.Geometry();

// 三角形的三个顶点坐标

triangleGeometry.vertices.push(new THREE.Vector3(0.0, 0.5, 0.0));

triangleGeometry.vertices.push(new THREE.Vector3(-0.5, -0.5, 0.0));

triangleGeometry.vertices.push(new THREE.Vector3(0.5, -0.5, 0.0));

// 其实是设置顶点索引

triangleGeometry.faces.push(new THREE.Face3(0, 1, 2));

我们只需要将这个三角形设置为黄色，所以只需要用到最简单的材质——MeshBasicMaterial，代码如下：

// 设置材质，只是简单将颜色设置为黄色

var triangleMaterial = new THREE.MeshBasicMaterial({ color: 0xffff00 });

最后由以上两个对象生成网格对象，并添加到场景中。

// 由几何体和材质创建网格，并添加到场景中

var triangle = new THREE.Mesh(triangleGeometry, triangleMaterial);

scene.add(triangle);

## 添加动画

接来下添加动画。requestAnimationFrame()函数是专门为创建脚本式动画而设计的，它比传统的setInterval()和setTimeout()方法更适合于创建动画。用setlnterval()或setTimeout()方法创建动画时，需要确定动画更新的最佳频率。但是这个最佳频率对于动画的的设计人员来说是很难确定的。但是，浏览器可以比较容易确定这个最佳频率的。浏览器上可能会同时运行多个动画，这可能会影响这个帧频。在这种情形下，浏览器会降低所有动画的帧频，这样它们就以流畅但稍低的频率执行动画。

在代码中，需要将renderer.render(scene, camera);替换为以下代码：

render();

function render() {

 // 让三角形绕y轴旋转

 triangle.rotation.y += 0.02;

 requestAnimationFrame(render);

 renderer.render(scene, camera);

}

如果现在运行代码，就可以看到一个黄色三角形正在绕竖直的y轴旋转。你会发现有一半时间三角形会变得不可见，这是因为没有在材质中没有将side属性设置为THREE.DoubleSide的缘故。

这里还引入了一个小的辅助库，它可以检测出动画的帧频。要显示这个统计图形，需要先引入这个库

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

然后添加一个<div>元素作为容器：

<div id="Stats-output"></div>

最后初始化统计对象并添加到<div>元素中：

function initStats() {

 var stats = new Stats();

 stats.setMode(0); // 0: fps, 1: ms

 // 将状态信息放置到屏幕左上角

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

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

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

 $("#Stats-output").append(stats.domElement);

 return stats;

}

别忘了初始化对象：

$(function () {

var stats = initStats();

…

}

以及在render()方法中调用它的update()方法

function render() {

 stats.update();

 …

}

## 使用dat.GUI库

最后我们还要使用一个名为dat.GUI的库创建一个简单的用户界面，用以修改代码中的变量。

首先要在<header>元素中添加这个库：

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

接下来要定义一个JavaScript对象，用来保存我们想要通过dat.GUI库修改的那些变量。本例中添加了2个变量，用来控制三角形的颜色和是否以线框模式绘制三角形。

var controls = new function () {

 this.color = triangleMaterial.color.getStyle();

 this.wireframe = triangleMaterial.wireframe;

}

接下来把这个对象传递给dat.GUI对象：

var gui = new dat.GUI();

gui.addColor(controls, 'color').onChange(function (e) {

 triangle.material.color.setStyle(e);

});

gui.add(controls, 'wireframe').onChange(function (e) {

 triangle.material.wireframe = e;

});

至此我们已经完成了全部的任务。程序截图如下：



## 完整代码

<!DOCTYPE html>

<html>

<head>

 <title>示例01.01 - 基本框架</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>

 body{

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

 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(0x00204d);

 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);

 // 设置相机位置，观察目标设置为场景中央

 camera.position.x = 0;

 camera.position.y = 0;

 camera.position.z = 3;

 camera.lookAt(scene.position);

 var triangleGeometry = new THREE.Geometry();

 // 三角形的三个顶点坐标

 triangleGeometry.vertices.push(new THREE.Vector3(0.0, 0.5, 0.0));

 triangleGeometry.vertices.push(new THREE.Vector3(-0.5, -0.5, 0.0));

 triangleGeometry.vertices.push(new THREE.Vector3(0.5, -0.5, 0.0));

 // 其实是设置顶点索引

 triangleGeometry.faces.push(new THREE.Face3(0, 1, 2));

 // 设置材质，只是简单将颜色设置为黄色

 var triangleMaterial = new THREE.MeshBasicMaterial({ color: 0xffff00 });

 // 由几何体和材质创建网格，并添加到场景中

 var triangle = new THREE.Mesh(triangleGeometry, triangleMaterial);

 scene.add(triangle);

 // 设置三角形颜色和线框模式的js对象

 var controls = new function () {

 this.color = triangleMaterial.color.getStyle();

 this.wireframe = triangleMaterial.wireframe;

 }

 // 添加用户界面

 var gui = new dat.GUI();

 gui.addColor(controls, 'color').onChange(function (e) {

 triangle.material.color.setStyle(e);

 });

 gui.add(controls, 'wireframe').onChange(function (e) {

 triangle.material.wireframe = e;

 });

 render();

 function render() {

 stats.update();

 // 让三角形绕y轴旋转

 //triangle.rotation.y += 0.02;

 requestAnimationFrame(render);

 renderer.render(scene, camera);

 }

 function initStats() {

 var stats = new Stats();

 stats.setMode(0); // 0: fps, 1: ms

 // 将状态信息放置到屏幕左上角

 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>