# 3.1 使用几何和网格对象

Three.js库附带了很多现成的几何体，你可以在三维场景中使用它们。只要加上材质、创建一个mesh变量，基本上就算完成了。

本文主要介绍如何用这些几何体的基类Geometry手工创建几何体。

Three.js库中的geometry和其他大多数三维库中的一样，基本上是三维空间中的点集，以及一些将这些点连接起来的面。举例来说，一个方块：

* 一个方块有8个角。每个角都可以定义为x、y和z坐标的一个组合。所以每个方块都是三维空间中的8个点。在Three.js库中，这些点称为顶点（vertice）。
* 一个方块有6个侧面，每个角有一个顶点。在Three.js库里，每个侧面称为面（face）。

当你使用Three.js库提供的这些几何体时，你不必亲自定义所有的这些顶点和面。对于一个方块来讲，你只要定义长宽高即可。Three.js库会利用这些信息，在正确的位置创建一个拥有8个顶点的几何体，并用正确的面连接起来。

尽管你可以使用Three.js库提供的几何体，或者自动生成，但是你仍然可以通过定义顶点和面，手工创建几何体。创建方法可以参考下面的代码片段：

var vertices = [

 new THREE.Vector3(1,3,1),

 new THREE.Vector3(1,3,-1),

 new THREE.Vector3(1,-1,1),

 new THREE.Vector3(1,-1,-1),

 new THREE.Vector3(-1,3,-1),

 new THREE.Vector3(-1,3,1),

 new THREE.Vector3(-1,-1,-1),

 new THREE.Vector3(-1,-1,1)

];

var faces = [

 new THREE.Face3(0,2,1),

 new THREE.Face3(2,3,1),

 new THREE.Face3(4,6,5),

 new THREE.Face3(6,7,5),

 new THREE.Face3(4,5,1),

 new THREE.Face3(5,0,1),

 new THREE.Face3(7,6,2),

 new THREE.Face3(6,3,2),

 new THREE.Face3(5,7,0),

 new THREE.Face3(7,2,0),

 new THREE.Face3(1,3,4),

 new THREE.Face3(3,6,4),

];

这段代码展示的是如何创建一个简单的方块。在vertices数组里定义了构成这些方块的点。将这些点连接起来，创建三角面片，并保存在faces数组里。例如，元素new THREE.Face3(0，2，1)就是用vertices数组里的点0、2和1创建的一个三角面片。

本示例的截图如下。你可以修改这个方块的所有顶点。



对于一个顶点，无论何时修改了该顶点下拉列表框里的属性，这个方块都会按照修改后的位置正确地渲染出来，但这需要一些额外的设置。出于效率方面的考虑，Three.js库假设一个网格的几何体在其生命周期内不会改变。为了使我们的示例能够工作，我们需要将下面的代码添加到render循环里：

mesh.geometry.vertices=vertices;

mesh.geometry.verticesNeedUpdate=true;

mesh.geometry.computeFaceNormals();

在上述这段代码的第一行里，我们将mesh的vertices属性指向一个更新后的顶点数组。我们不需要重新配置侧面，因为它们跟以前一样仍然连接到那些点。设置好这些更新的顶点之后，我们需要告诉geometry对象，这些顶点需要更新。为此我们可以将geometry的verticesNeedUpdate属性设为true。最后我们需要通过调用computeFaceNormals()函数重新计算侧面法线，从而完成整个模型的更新。

## 组合材质

本示例的材质定义比前面的复杂，代码如下：

var materials = [

 new THREE.MeshLambertMaterial( { opacity:0.6, color: 0x44ff44, transparent:true } ),

 new THREE.MeshBasicMaterial( { color: 0x000000, wireframe: true } )

];

我们使用的不是一个单一的材质，而是有两个元素的材质数组。原因是除了显示一个绿色透明的方块之外，我还想显示一个线框。因为使用线框的话，更容易找出顶点和面的位置。在创建网格的时候，Three.js库也支持使用多种材质。你可以使用SceneUtils.createMultiMaterialObject()函数来达到这个目的，如下所示：

var mesh = THREE.SceneUtils.createMultiMaterialObject(geom,materials);

在这个函数里，Three.js库创建的不是一个THREE.Mesh实例，而是为每个你指定的材质创建一个实例，并把这些实例存放在一个组里。你可以像使用Scene对象那样使用这个组。你可以添加网格、通过名字查找网格等。例如，要为组中所有子对象添加阴影，我们可以这么做：

mesh.children.forEach(function(e) {e.castShadow=true});

## Geometry对象的clone方法

Geometry对象定义的是物体的形式、形状，赋予一定材质之后，我们就可以创建出一可以添加到场景中并由Three.js库渲染的物体。通过clone()函数我们可以创建出一个geometry对象的副本。赋予不同的材质后，我们就可以用这个副本创建出一个不同的网格对象。你可以在控制界面的顶部找到一个**clone**按钮，如果你点击这个按钮，你就可以按照geometry对象当前的状态创建出一个副本，而且这个新对象被赋予了新的材质，并被添加到了场景中。clone()函数代码如下：

var cloned = mesh.children[0].geometry.clone();

var materials = [

 new THREE.MeshLambertMaterial( { opacity:0.6, color: 0xff44ff, transparent:true } ),

 new THREE.MeshBasicMaterial( { color: 0x000000, wireframe: true } )

];

var mesh2 = THREE.SceneUtils.createMultiMaterialObject(cloned,materials);

mesh2.children.forEach(function(e) {e.castShadow=true});

mesh2.translateX(5);

mesh2.translateZ(5);

mesh2.name="clone";

scene.remove(scene.getChildByName("clone"));

scene.add(mesh2);

这段JavaScript代码会在点击**clone**按钮时调用。我们复制的是方块的第一个子对象的geometry。mesh变量有两个子对象：一个mesh用的是MeshLambertMaterial材质，另一个mesh用的是MeshBasicMaterial材质。基于这个复制出来的几何体，我们将创建一个新的网格，并命名为mesh2。我们可以使用translate()函数移动这个新建的网格，删除之前的副本（如果可见的话），并把这个副本添加到场景中。

## 完整代码

<!DOCTYPE html>

<html>

<head>

 <title>示例03.01 - 自定义几何体</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">

 $(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);

 renderer.shadowMapEnabled = true;

 // 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 = -20;

 camera.position.y = 25;

 camera.position.z = 20;

 camera.lookAt(new THREE.Vector3(5, 0, 0));

 // 创建一个平面作为地面

 var planeGeometry = new THREE.PlaneBufferGeometry(60,40,1,1);

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

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

 plane.receiveShadow = true;

 plane.rotation.x=-0.5\*Math.PI;

 plane.position.x=0

 plane.position.y=0

 plane.position.z = 0

 scene.add(plane);

 // 添加一个聚光灯光源

 var spotLight = new THREE.SpotLight( 0xffffff );

 spotLight.position.set( -40, 60, 10 );

 spotLight.castShadow = true;

 scene.add( spotLight );

 // 自定义几何体

 var vertices = [

 new THREE.Vector3(1,3,1),

 new THREE.Vector3(1,3,-1),

 new THREE.Vector3(1,-1,1),

 new THREE.Vector3(1,-1,-1),

 new THREE.Vector3(-1,3,-1),

 new THREE.Vector3(-1,3,1),

 new THREE.Vector3(-1,-1,-1),

 new THREE.Vector3(-1,-1,1)

 ];

 var faces = [

 new THREE.Face3(0,2,1),

 new THREE.Face3(2,3,1),

 new THREE.Face3(4,6,5),

 new THREE.Face3(6,7,5),

 new THREE.Face3(4,5,1),

 new THREE.Face3(5,0,1),

 new THREE.Face3(7,6,2),

 new THREE.Face3(6,3,2),

 new THREE.Face3(5,7,0),

 new THREE.Face3(7,2,0),

 new THREE.Face3(1,3,4),

 new THREE.Face3(3,6,4),

 ];

 var geom = new THREE.Geometry();

 geom.vertices = vertices;

 geom.faces = faces;

 geom.mergeVertices();

 // 组合材质

 var materials = [

 new THREE.MeshLambertMaterial( { opacity:0.6, color: 0x44ff44, transparent:true } ),

 new THREE.MeshBasicMaterial( { color: 0x000000, wireframe: true } )

 ];

 var mesh = THREE.SceneUtils.createMultiMaterialObject(geom,materials);

 mesh.children.forEach(function(e) {e.castShadow=true});

 scene.add(mesh);

 function addControl(x,y,z) {

 var controls = new function() {

 this.x = x;

 this.y = y;

 this.z = z;

 }

 return controls;

 }

 var controlPoints = [];

 controlPoints.push(addControl(3,5,3));

 controlPoints.push(addControl(3,5,0));

 controlPoints.push(addControl(3,0,3));

 controlPoints.push(addControl(3,0,0));

 controlPoints.push(addControl(0,5,0));

 controlPoints.push(addControl(0,5,3));

 controlPoints.push(addControl(0,0,0));

 controlPoints.push(addControl(0,0,3));

 var gui = new dat.GUI();

 gui.add(new function() {

 this.clone = function() {

 var cloned = mesh.children[0].geometry.clone();

 var materials = [

 new THREE.MeshLambertMaterial( { opacity:0.6, color: 0xff44ff, transparent:true } ),

 new THREE.MeshBasicMaterial( { color: 0x000000, wireframe: true } )

 ];

 var mesh2 = THREE.SceneUtils.createMultiMaterialObject(cloned,materials);

 mesh2.children.forEach(function(e) {e.castShadow=true});

 mesh2.translateX(5);

 mesh2.translateZ(5);

 mesh2.name="clone";

 scene.remove(scene.getObjectByName("clone"));

 scene.add(mesh2);

 //console.log(scene.children);

 }

 },'clone');

 for (var i = 0 ; i < 8 ; i++) {

 f1 = gui.addFolder('顶点' + (i+1));

 f1.add(controlPoints[i], 'x',-10,10);

 f1.add(controlPoints[i], 'y',-10,10);

 f1.add(controlPoints[i], 'z',-10,10);

 }

 render();

 function render() {

 stats.update();

 var vertices = [];

 for (var i = 0 ; i < 8 ; i++) {

 vertices.push(new THREE.Vector3(controlPoints[i].x, controlPoints[i].y,controlPoints[i].z));

 }

 mesh.children.forEach(function(e) {

 e.geometry.vertices=vertices;

 e.geometry.verticesNeedUpdate=true;

 e.geometry.computeFaceNormals();

 });

 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>