经常有网友在QQ群里面问，three.js如何加载模型。有的人是为了做游戏需要加载一些人物模型，有的人是为了毕业设计（模拟一些物理运动，周围需要一些物体做陪衬），无论基于什么样的需求，他们都想在自己的应用中，加载3D模型。有了3D模型才能让他们的程序更有创造力。

下面就有一个网友在咨询怎么加载模型：



估计有朋友有类似的疑惑，那么这一课，我们就来仔细的讲一讲，关于模型的加载。这是一个有趣的事情，那现在我们开始吧。

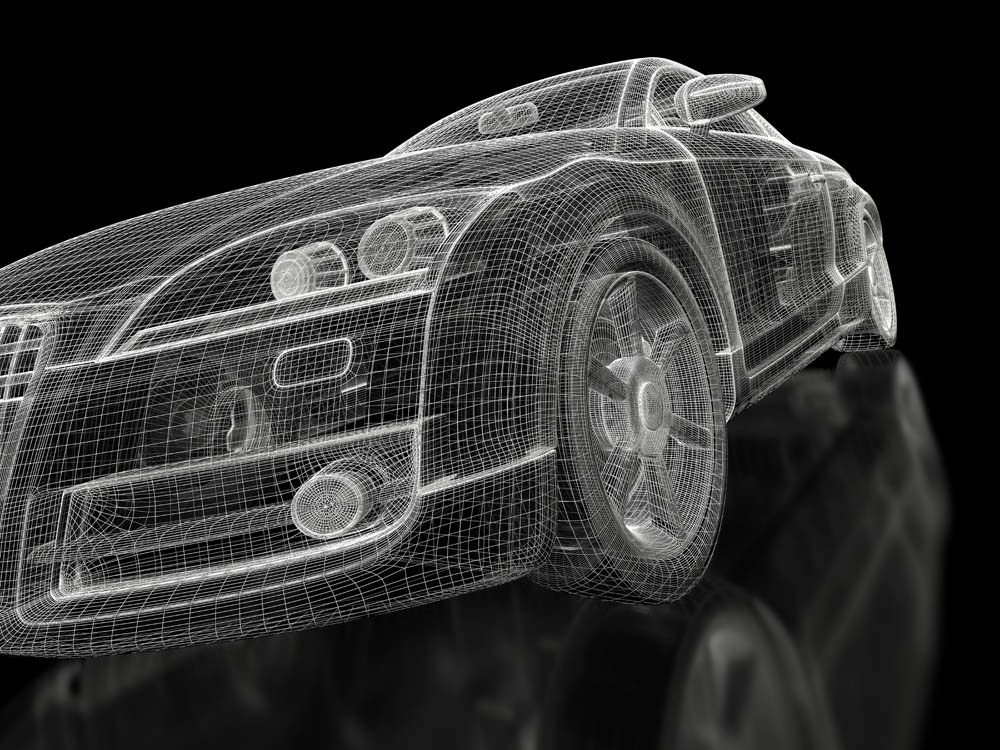
## 3、关于模型的基础知识

既然在讲3D世界，那么我们这里提到的模型就是3D模型。我不想将一些书上的定义摘抄给大家，那些概念对大家的理解帮助并不大，而接下来，我会将我最直接通俗的理解告诉大家。

我们知道，在3DMAX，MAYA等软件（这是一些三维编辑软件）中，可以制作出3D模型。这些模型可用于室内设计，三维影视，三维游戏等领域。那么3D模型是怎么定义的呢？看看下面我的定义：

3D模型由顶点(vertex)组成，顶点之间连成三角形或四边形（在一个平面上）,多个三角形或者四边形就能够组成复杂的立体模型.

如下图所示：



上图就是一辆汽车的3D模型（立体模型），因为是由一个个网格组成，所以，我们也叫其为网格模型。

网格模型很像我们小时候学习的素描，想想我们画素描是不是先描点，然后画线，由线组成面，最后由面组成现实生活中的物体呢？

在这里，概念总是那么难以理解，我们不做深抠概念的学究，我们需要感性的去认识3D模型。

首先，我们来欣赏一些3D模型，这里收集了的一些3D模型，当然其中一些因为加上了纹理，非常诱人。



除此之外，你可以在google的3D模型库找到更多的模型，网址是：http://sketchup.google.com/3dwarehouse/（这个网站可能被墙，无法访问，使用代理吧），这是一个很有用的网站，请大家惠存。

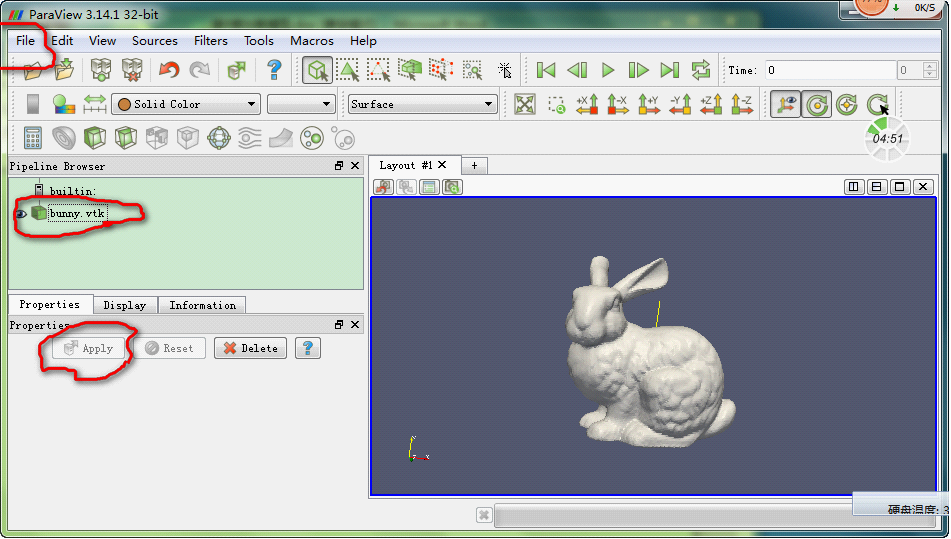
## 4、怎么看模型？模型查看器

有了3D模型，我们怎么去看它。我们在编写程序之前，总是希望先看到一些实际的场景，这样，当我们写起程序来，才会有一些感觉，依葫芦画瓢的感觉。所以，这里我们先教大家怎么观察3D模型。

你可以下载一个3D-Max，或者Maya之类的软件，然后安装上它。但是他们太重量级，动辄就是几个G，不太实用，我们也没有耐心去安装这样一个巨大的程序，而且这些程序过于专业，您并不一定能很快使用它。

所以，在工程领域，我们一般使用一些轻量级的查看器，如Blender、ParaView。这些都是一个几十M的软件，且功能足够可用。大家可以到网上下载一个ParaView，各大网站均有下载。

ParaView是一个模型查看器，打开软件，会看到如下的界面，当然这只兔子是我们打开一个模型文件后的结果。



例如，我们要打开一个vtk模型文件。你可以在three.js的源码包中找到一个bunny.vtk的文件(在源代码中搜索一下这个文件)，也可以在我们提供的代码中找到【初级教程\chapter7A\models\vtk】，然后使用ParaView的File菜单打开，就会在Pipeline Browser中看到一个bunny.vtk的文件，选中它，然后点Apply，这样就可以再右边看到一个兔子模型了。

ParaView的使用不是本章重点，这里就不累述了。你可以在这里看到ParaView的使用，地址如下：http://wenku.baidu.com/view/f360cc7102768e9951e7386a.html

简单的使用一下，你就知道怎么观察模型了，并且你也应该学会像保存图片一样，将一种3D模型的格式转换为另一种3D模型的格式，如obj、stl等。

## 5、模型在three.js中的表示

我们的最终目的是要讲解怎么将模型导入three.js中，让three.js能够显示我们的模型。

模型是由面组成，面分为三角形和四边形面。三角形和四边形面组成了网格模型。在Three.js中用THREE.Mesh来表示网格模型。THREE.Mesh可以和THREE.Line相提并论，区别是THREE.Line表示的是线条。THREE.Mesh表示面的集合。

Ok，让我们来认识一下THREE.Mesh，它的构造函数是：

THREE.Mesh = function ( geometry, material )

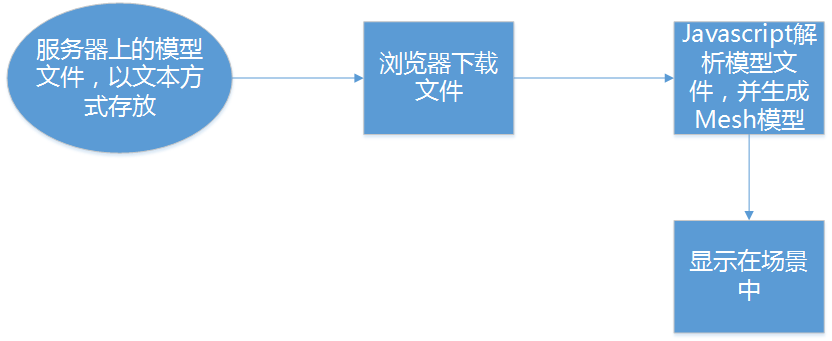
其中第一个参数geometry：是一个THREE.Geometry类型的对象,他是一个包含顶点和顶点之间连接关系的对象。

第二个参数Material：就是定义的材质。有了材质就能够让模型更好看，材质会影响光照、纹理对Mesh的作用效果。

## 6、模型的加载

在three.js中，模型是怎么加载到浏览器中的呢？

为了让大家更明白原理，我们首先来看看three.js加载一个简单模型的过程。这个过程是这样的：



上图的顺序是：

1、服务器上的模型文件以文本的方式存储，除了以three.js自定义的文本方式存储之外，当然也可以以二进制的方式存储，不过这里暂时不讲。

2、浏览器下载文件到本地

3、Javascript解析模型文件，生成Mesh网格模型

4、显示在场景中。

对照上面这幅图，我们对需要注意的几点重点说明一下：

1、服务器上的模型文件大多是存储的模型的顶点信息，这些信息可以以文本的方式存储的（并不一定需要用文本的方式存储）。Three.js支持很多种3D模型格式，例如ply，stl，obj，vtk等等。随着three.js的升级，会支持越来越多的文件格式，到目前为止，three.js已经能够支持市面上大多数3D模型格式了。

同时需要重点说明的是，如果认真理解完three.js对模型的加载、解析方法，那么写一种自己的3D文件解析器是非常便利的。

2、第二步是浏览器下载文本文件，这是一件很普通的事情，只需要使用javascript的异步请求就可以实现了。

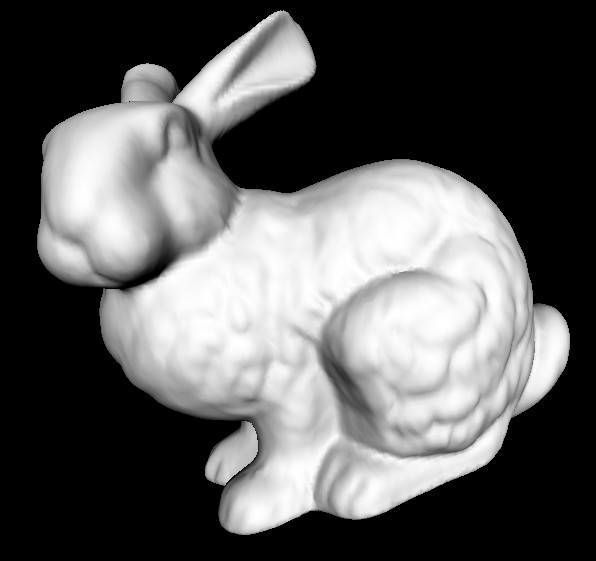
3、Javascript解析文本文件并生成一个geometry，最终生成Mesh，也是一件简单的事情。我们会在后面介绍这个过程。

4、当产生Mesh后，将其加入到场景中，那就非常简单了。

Ok，整个模型的加载思路就是这样。

## 7、一个简单的实例

本课，我们先来加载一个简单的模型，这个模型是一只兔子，如下图所示，你能在代码清单【初级教程\chapter7A】中找到这课的源码。建议你先运行一下，如果因为交叉域访问错误，请按照扩展阅读的第二课设置一下浏览器，或者搭建一个Web服务器，将源码放到Web服务器中。



这个兔子的模型文件是【初级教程\chapter7A\models\vtk\bunny.vtk】这个文件，他是一个后缀名为vtk的文件，表示这是vtk格式。可能很多童鞋都不知道什么是vtk，那么这里给大家补一下vtk文件模式吧。

至于为什么选择vtk，因为vtk是一种简单的3D文件格式，而且是以文本的方式存储的，所以理解起来非常方便。你可以打开刚才那个vtk文件，看一下是不是很简单。

## 8、VTK文件格式

什么是vtk文件？

Vtk模型是一种以文本方式表示的3D模型文件，其能够表示点面信息，而且能够以人类易读易懂的方式以文本的形式存储下来。在科学研究中，这种文件格式使用得非常多，我们这里详细的讲解一下，这种文件格式。

vtk是3D模型的一种格式，现在版本已经到4.0了。你可以在网上找到这种格式的详细解释，当然最好去google搜索。

打开bunny.vtk文件，我们以它来解释vtk文件的格式，其中#是注释的开始

|  |
| --- |
| # 这里表示使用的是vtk的3.0版本。虽然4.0版本已经出来了，不过目前广泛使用的仍然是3.0 |
| # vtk DataFile Version 3.0 |
| # 这一行是输出vtk文件的软件写的文字，无论什么都可以。 |
| vtk output |
| # ASCII，表示这份vtk使用的标准ASCII码字符集 |
| ASCII |
| # “DATASET POLYDATA”表示多边形面集，面是由一个个点组成的 |
| DATASET POLYDATA |
| # 这里表示这个模型由35947个点组成，每个坐标的分量是一个浮点型 |
| POINTS 35947 float |
| # 下面是35947个点的数据 |
| -0.0378297 0.12794 0.00447467 -0.0447794 0.128887 0.00190497 -0.0680095 0.151244 0.0371953 |
| -0.00228741 0.13015 0.0232201 -0.0226054 0.126675 0.00715587 -0.0251078 0.125921 0.00624226 |
| -0.0371209 0.127449 0.0017956 0.033213 0.112692 0.0276861 0.0380425 0.109755 0.0161689 |
| -0.0255083 0.112568 0.0366767 -0.0245306 0.112636 0.0373469 0.0274031 0.12156 0.0212208 |
| -0.0628961 0.158419 -0.0175871 0.0400813 0.104202 0.0221684 0.0451532 0.0931968 0.0111604 |
| .......................................... |
| .......................................... |
| POLYGONS 69451 277804 |
| 3 21216 21215 20399 |
| 3 9186 9280 14838 |
| 3 16020 13433 5187 |
| 3 16021 16020 5187 |
| 3 20919 20920 21003 |
| 3 23418 15239 23127 |
| 3 30553 27378 30502 |
| .................. |
| .................... |
| CELL\_DATA 69451 |
| POINT\_DATA 35947 |

你可以在 http://wenku.baidu.com/view/a67cdad049649b6648d747fc.htm 这个网站找到vtk 4.0版本的解释。下面对几个重要的段落进行一下解释。

1、# vtk DataFile Version 3.0表示这个vtk文件的版本是3.0。最新版本是4.0，不过改变不大。

2、vtk output表示该文件是名字，一般写成vtk output就可以了，基本上，你没有必要去改变它。

3、ASCII表示该文件的格式，是ascii版本，该位置也可以写binary，那么这个文件就是二进制格式的了。

4、DATASET POLYDATA中的DATASET是关键字表示数据集的意思，POLYDATA表示数据的类型，可以取STRUCTED\_POINTS、STRUCTURED\_GRID、UNSTRUCTURED\_GRID、POLYDATA、FIELD等。这里取的是POLYDATA，表示三角形或者四边形数据。

5、POINTS 35947 float 表示这个模型由35947个点组成，每个点的分量，其数据类型是浮点型。这一行后面就是35947\*3个float型数字了。每三个数字表示一个点。

6、POLYGONS 69451 277804，POLYGONS是关键字，69451表示模型有69451个多边形组成，后面行的3 21216 21215 20399中的3表示每个多边形由三个顶点组成，如果等于4，那么每个多边形有4个顶点组成。277804表示整个POLYGONS占据的数组的长度，长度计算公式是69451\*4 = 277804，乘数4是3 21216 21215 20399这组元素的长度（一共有4个元素），也就每一行元素的个数，这主要是用来计算存储空间的。

7、接下来后面是69451行数据，每一行是一个多边形面。每个面由3个顶点组成，如3 21216 21215 20399这一行，后面的21216 21215 20399这三个数字，表示在上面的POINTS 35947 float段的顶点索引。

8、CELL\_DATA 69451 表示面的个数，和上面定义的面数目必须一致。

9、POINT\_DATA 35947表示点的个数，和“POINTS 35947 float”定义的也必须相同。 Ok，vtk格式就这么多精髓了，虽然有些关键字这里我们并没有讲到，但是已经足够了，在以后遇到如果不明白，可以质询我们，也可以查查网上的文档，用一下google。

了解了VTK模型的格式，我们下面来写一个解析器，解析这个模型。

## 9、顶点和面索引之间的关系

加载vtk模型，主要分为2步：

1、将vtk文件中的点，转换为geometry的vertices数组中。

2、将vtk文件中每个点的索引，转换到geometry的faces中。

### **1、点和索引之间的关系**

上面提到点和索引，如果对点和索引的关系不太理解，那么后面学习起来将非常痛苦，所以，我们这里先简单的讲一讲点和索引之间的密切关系。

我们先出一道题目，大家准备纸和笔了，如果你没有动手，在纸和笔上画出来，没有理解，就不好怪我了。

题目：6个点，要求每个三角形不重用点，那么最多可以组成多少个三角形？

好了，画出来了吗？是不是只能画出2个来。Ok，这是点不能重复的情况。在点不能重复的情况下，画2个三角形，需要6个点。

如果在点能够重复的情况下，6个点可以画多少三角形呢？

这个数学题，我们就不解释了，但是只要你在纸上认真的画一下，你就会发现，在允许点重复的情况下，6个点能画出远远超过2个三角形。

结论：同样的数目的点，如果允许重用，那么能够画出更多的三角形。能够画出更多三角形，是不是就能形成更复杂的几何体呢？

再反过来想一想，同样数目的点，是不是占用的存储空间一样呢？同样个数的三角形，是不是重复利用点的三角形，会比不重复利用点的三角形占用的存储空间多呢？这里主要指的内存空间。

讲到这里，我们为大家揭穿最后一点秘密，索引就是为了重复利用顶点而诞生的。索引数组中存放的整形数，每三个整形数，就能够决定一个三角形。

索引中存放的整形数表示顶点在geometry.vertices数组中的位置。例如，geometry的faces的第0,1,2个元素分别是6,100,62，那么表示取geometry.vertices中第6,100,62个点，组成一个三角形。是不是恍然大悟了。Ok，画一下geometry.vertices和geometry.faces这两个数据结构，并将里面的关系画出来，你就明白了。

## 10、vtk文件的加载

我们本课要加载一只兔子模型，这只兔子使用的是vtk模型，大家可以打开这个文件看一下。

Vtk的加载需要使用VTKLoader.js这个文件，你可以在three.js-master\examples\js\loaders文件夹或者在【初级教程\chapter7A\js\loaders】中找到。由于源码不太长，我们将其列在这里：

|  |
| --- |
| *// 构造函数* |
| THREE.VTKLoader = function () { |
| THREE.EventDispatcher.call( this ); *// 继承自监听器，使这个类有监听的功能* |
| }; |
|  |
| *// VTKLoader的原型函数，里面包含了VTKloader的成员函数，成员变量的定义* |
| THREE.VTKLoader.prototype = { |
| *// 构造函数* |
| constructor: THREE.VTKLoader, |
| *// 加载函数，url表示要加载的vtk文件的url路径，callback表示加载完成后要调用的后续处理函数，这里是异步操作，加载需要一个过程，不能将程序阻塞在这里，所以需要异步回调* |
| load: function ( url, callback ) { |
| *// 将类自身保存在scope中，scope表示域的意思，这里是为了避免this的歧义，因为，每一个地方使用this，其意义不一样。* |
| var scope = this; |
| *// ajax 异步请求* |
| var request = new XMLHttpRequest(); |
| *// 加载完成的监听器，加载完成后，将调用第二个参数定义的回调函数* |
| request.addEventListener( 'load', function ( event ) { |
| *// 对服务器加载下来的数据进行解析，后面详细解释* |
| var geometry = scope.parse( event.target.responseText ); |
| *// 解析完成后，发一个load事件，表示数据解析完成* |
| scope.dispatchEvent( { type: 'load', content: geometry } ); |
| *// 如果设置了回调函数，那么调用回调函数* |
| if ( callback ) callback( geometry ); |
| }, false ); |
| *// 加载过程中，向自身发送进度progress信息，信息中包含已经加载的数据的字节数和文件总共的字节数，通过两者的百分比能够了解加载进度。* |
| request.addEventListener( 'progress', function ( event ) { |
| *// 发送正在加载的消息，两个参数分别是已经加载了多少字节，总共多少字节* |
| scope.dispatchEvent( { type: 'progress', loaded: event.loaded, total: event.total } ); |
| }, false ); |
| *// 加载出错的监听器，加载的过程也可能出错，这里如果出错，进行错误处理，* |
| request.addEventListener( 'error', function () { |
| *// 加载出错之后需要发布的错误消息，* |
| scope.dispatchEvent( { type: 'error', message: 'Couldn\'t load URL [' + url + ']' } ); |
|  |
| }, false ); |
| *// 初始化 HTTP 请求参数，例如 URL 和 HTTP 方法，但是并不发送请求。* |
| request.open( 'GET', url, true ); |
| *//发送 HTTP 请求，开始下载* |
| request.send( null ); |
| }, |
|  |
| *// parse函数在上面调用过，主要负责解析数据的功能，我们将在后面详细介绍解析函数，这里就不介绍了。* |
| parse: function ( data ) { |
| var geometry = new THREE.Geometry(); |
| function vertex( x, y, z ) { |
| geometry.vertices.push( new THREE.Vector3( x, y, z ) ); |
| } |
|  |
| function face3( a, b, c ) { |
| geometry.faces.push( new THREE.Face3( a, b, c ) ); |
| } |
|  |
| function face4( a, b, c, d ) { |
| geometry.faces.push( new THREE.Face4( a, b, c, d ) ); |
| } |
|  |
| var pattern, result; |
|  |
| *// float float float* |
|  |
| pattern = /([\+|\-]?[\d]+[\.][\d|\-|e]+)[ ]+([\+|\-]?[\d]+[\.][\d|\-|e]+)[ ]+([\+|\-]?[\d]+[\.][\d|\-|e]+)/g; |
|  |
| while ( ( result = pattern.exec( data ) ) != null ) { |
|  |
| *// ["1.0 2.0 3.0", "1.0", "2.0", "3.0"]* |
|  |
| vertex( parseFloat( result[ 1 ] ), parseFloat( result[ 2 ] ), parseFloat( result[ 3 ] ) ); |
|  |
| } |
|  |
| *// 3 int int int* |
|  |
| pattern = /3[ ]+([\d]+)[ ]+([\d]+)[ ]+([\d]+)/g; |
|  |
| while ( ( result = pattern.exec( data ) ) != null ) { |
|  |
| *// ["3 1 2 3", "1", "2", "3"]* |
|  |
| face3( parseInt( result[ 1 ] ), parseInt( result[ 2 ] ), parseInt( result[ 3 ] ) ); |
|  |
| } |
|  |
| *// 4 int int int int* |
|  |
| pattern = /4[ ]+([\d]+)[ ]+([\d]+)[ ]+([\d]+)[ ]+([\d]+)/g; |
|  |
| while ( ( result = pattern.exec( data ) ) != null ) { |
|  |
| *// ["4 1 2 3 4", "1", "2", "3", "4"]* |
|  |
| face4( parseInt( result[ 1 ] ), parseInt( result[ 2 ] ), parseInt( result[ 3 ] ), parseInt( result[ 4 ] ) ); |
|  |
| } |
|  |
| geometry.computeCentroids(); |
| geometry.computeFaceNormals(); |
| geometry.computeVertexNormals(); |
| geometry.computeBoundingSphere(); |
|  |
| return geometry; |
|  |
| } |
|  |
| } |

在注释中，我们解释了大部分代码，请仔细阅读。在上面的代码中，以下的部分代码需要注意：

|  |
| --- |
| request.addEventListener( 'load', function ( event ) { |
|  |
| var geometry = scope.parse( event.target.responseText ); |
|  |
| scope.dispatchEvent( { type: 'load', content: geometry } ); |
|  |
| if ( callback ) callback( geometry ); |
|  |
| }, false ); |

这里event.target.responseText是服务器返回的文本数据，也就是vtk文件里的所有数据，我们通过scope.parse方法将其转换为geometry。

转换完后，我们会通过dispathEvent向自身发送一个加载完成的消息，消息中返回了geometry几何体。这个几何体是可以和Mesh合体，最终显示在场景中的。

最后，如果callback不为null的话，那么我们就调用这个回调函数。在这个回调函数中，会做一些模型加载完成后，应该做的事情，例如，将模型放到某一个位置。

接下来的重点就是parse函数如何实现的了。

## 11、伟大的解析函数parse

上一节，我们讲了THREE.VTKLoader这个类，那么接着我们将parse给补充一下，parse函数主要完成了从vtk到geometry的转换。我们将代码复制下来重新解释一下：

|  |
| --- |
| *// data是从服务器传过来的数据，其实就是vtk文件中的文本数据 ，打开平【初级教程\chapter7A\models\vtk\bunny.vtk】看一下，你就知道是什么了？一定打开这个文件哦。* |
| parse: function ( data ) { |
| *// new 一个几何体* |
| var geometry = new THREE.Geometry(); |
| *// 定义一个内部函数vertex，用参数x,y,z生成一个顶点，并放入geometry的vertices数组中，* |
| function vertex( x, y, z ) { |
| geometry.vertices.push( new THREE.Vector3( x, y, z ) ); |
| } |
|  |
| *// 定义一个面索引函数face3，将面的3个点的索引放入geometry的faces数组中。* |
| function face3( a, b, c ) { |
| geometry.faces.push( new THREE.Face3( a, b, c ) ); |
| } |
| *// 定义一个面索引函数。如果一个面由4个顶点组成，那么我们构造一个Face4放入faces中，注意面可以由在同一平面的3个点组成，也可以由同一平面的4个顶点组成。* |
| function face4( a, b, c, d ) { |
| geometry.faces.push( new THREE.Face4( a, b, c, d ) ); |
| } |
| *// pattern存放模式字符串，result是临时变量* |
| var pattern, result; |
|  |
| *// float float float* |
| *// pattern是一个正则表达式，能够匹配3个空格隔开的float，如-0.0378297 0.12794 0.00447467都是pattern的菜。对正则表达式不了解，请一定补习一下哦。* |
| pattern = /([\+|\-]?[\d]+[\.][\d|\-|e]+)[ ]+([\+|\-]?[\d]+[\.][\d|\-|e]+)[ ]+([\+|\-]?[\d]+[\.][\d|\-|e]+)/g; |
| *// exec是正则表达式的执行匹配函数，result返回一个包含3个字符串的数组，如果data读到了最后，那么result将返回null* |
| *// while 循环在data中，寻找符合正则表示式的数据，将符合条件的数据，转换为一个顶点* |
| while ( ( result = pattern.exec( data ) ) != null ) { |
| *// ["1.0 2.0 3.0", "1.0", "2.0", "3.0"]* |
| *// 将字符串转换为float，并放入geometry中* |
| vertex( parseFloat( result[ 1 ] ), parseFloat( result[ 2 ] ), parseFloat( result[ 3 ] ) ); |
| } |
|  |
| *// 3 int int int* |
| *// 这里匹配面数据，如3 21216 21215 20399，这类数据是面索引数据* |
| pattern = /3[ ]+([\d]+)[ ]+([\d]+)[ ]+([\d]+)/g; |
| *// 取出data中的所有面索引数据，* |
| while ( ( result = pattern.exec( data ) ) != null ) { |
| *// ["3 1 2 3", "1", "2", "3"]* |
| *// 将面数据放入geometry的faces中* |
| face3( parseInt( result[ 1 ] ), parseInt( result[ 2 ] ), parseInt( result[ 3 ] ) ); |
|  |
| } |
|  |
| *// 4 int int int int* |
| *// 这里是4个顶点一个面的情况，本例的vtk文件，没有这种情况* |
| pattern = /4[ ]+([\d]+)[ ]+([\d]+)[ ]+([\d]+)[ ]+([\d]+)/g; |
|  |
| while ( ( result = pattern.exec( data ) ) != null ) { |
| *// ["4 1 2 3 4", "1", "2", "3", "4"]* |
| face4( parseInt( result[ 1 ] ), parseInt( result[ 2 ] ), parseInt( result[ 3 ] ), parseInt( result[ 4 ] ) ); |
|  |
| } |
|  |
| *// 这里的4个函数，在后面解释* |
| geometry.computeCentroids(); |
| geometry.computeFaceNormals(); |
| geometry.computeVertexNormals(); |
| geometry.computeBoundingSphere(); |
|  |
| return geometry; |
|  |
| } |

结合源代码和注释，我们能容易的理解上面的代码。这个pasrse函数主要用到了正则表达式，有一本书书名叫《程序员的6种武器》，其中就有一种是正则表达式。所以，如果不明白的可以问我，或者自己买本书看看。记住，当你年轻时，投资教育是最合算的。进入社会后，你就没那么多时间来投资教育了，现在开始换学习吧。

除了最后的4行，我们没有解释之外，其他的我们都清楚的解释了。下面来看看最后的4个函数：

### **1、geometry.computeCentroids()**

这个函数是算geometry中每一个面的重心，无论在平面坐标系还是空间坐标系中，重心可以求坐标的平均值来得到，如A点（X1，Y1，Z1），B点（X2，Y2，Z2）和C点（X3，Y3，Z3），他们形成的三角面的中心是：

重心的横坐标：(X1+X2+X3)/3

重心的纵坐标：(Y1+Y2+Y3)/3

重心的竖坐标：（z1+z2+z3）/3

computeCentroids函数内部就这样处理的，重心计算出来了，被存储在face.centroid这个变量中。

computeCentroids的函数代码如下：

|  |
| --- |
| computeCentroids: function () { |
| var f, fl, face; |
| for ( f = 0, fl = this.faces.length; f < fl; f ++ ) { |
| face = this.faces[ f ]; |
| face.centroid.set( 0, 0, 0 ); |
|  |
| face.centroid.add( this.vertices[ face.a ] ); |
| face.centroid.add( this.vertices[ face.b ] ); |
| face.centroid.add( this.vertices[ face.c ] ); |
| face.centroid.divideScalar( 3 ); |
| } |
| } |

从以上的代码可以看出，每一个面都有一个重心。

### **2、geometry. computeFaceNormals ()**

这个函数用来计算每一个面归一化后的法向量，法向量垂直于面。计算之后的法向量被存放在了face.normal中。法向量与模型受光情况有关，目前，我们还没有对关照原理进行详细讲解，这里先略过。

### **3、geometry. computeVertexNormals ()**

这个函数计算每一个顶点的法向量。

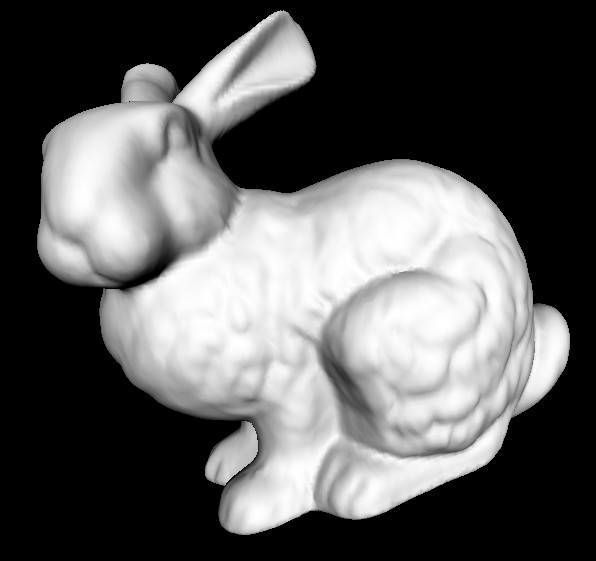
### **4、geometry. computeBoundingSphere ()**

这个函数这个计算一个可以包围geometry的一个椭圆，中心点就是geometry的中心，sphere的最大半径是离中心点最远的那一个点。

这4个函数的计算结果，在threejs引擎中，会用到，所以，你自己写加载某种模型的解析器时，最好也调用一下，这几个函数。

至此，关于parse函数，我们已经清晰的讲完了，parse函数完整的返回了一个geometry对象。你可以在Mesh中使用了。

## 12、完整的加载兔子vtk模型

最后，讲一个实例。和前面课程的程序大同小异，这里的不同是我们需要从服务器加载一些模型数据，然后转换为Mesh网格模型，最后显示在场景中。同样，按照以前的讲课思路，我们先看看本科最终的效果图，你可以打开本课的代码看看效果，下面是效果图：

这是一只可爱的兔子，在图形学的很多硕士论文中都会出现这只可爱的兔子。我师兄当年就是研究的将这只兔子签名，三维加密，是一个很不错的课题。有兴趣的童鞋可以研究一下。

Ok，接下来，我们展示一下全部的代码，代码不是很多，请不要害怕。

|  |
| --- |
| if ( ! Detector.webgl ) Detector.addGetWebGLMessage(); |
| var container, stats; |
| var camera, controls, scene, renderer; |
| var cross; |
| init(); |
| animate(); |
|  |
| function init() { |
| camera = new THREE.PerspectiveCamera( 60, window.innerWidth / window.innerHeight, 0.01, 1e10 ); |
| camera.position.z = 0.2; |
| controls = new THREE.TrackballControls( camera ); |
|  |
| controls.rotateSpeed = 5.0; |
| controls.zoomSpeed = 5; |
| controls.panSpeed = 2; |
|  |
| controls.noZoom = false; |
| controls.noPan = false; |
|  |
| controls.staticMoving = true; |
| controls.dynamicDampingFactor = 0.3; |
|  |
| scene = new THREE.Scene(); |
| scene.add( camera ); |
|  |
| // light |
| var dirLight = new THREE.DirectionalLight( 0xffffff ); |
| dirLight.position.set( 200, 200, 1000 ).normalize(); |
|  |
| camera.add( dirLight ); |
| camera.add( dirLight.target ); |
| // A begin |
| var material = new THREE.MeshLambertMaterial( { color:0xffffff, side: THREE.DoubleSide } ); |
| var loader = new THREE.VTKLoader(); |
| loader.addEventListener( 'load', function ( event ) { |
| var geometry = event.content; |
| var mesh = new THREE.Mesh( geometry, material ); |
| mesh.position.setY( - 0.09 ); |
| scene.add( mesh ); |
| } ); |
| loader.load( "models/vtk/bunny.vtk" ); |
| *// A end* |
| *// renderer* |
| renderer = new THREE.WebGLRenderer( { antialias: false } ); |
| renderer.setClearColorHex( 0x000000, 1 ); |
| renderer.setSize( window.innerWidth, window.innerHeight ); |
| container = document.createElement( 'div' ); |
| document.body.appendChild( container ); |
| container.appendChild( renderer.domElement ); |
|  |
| stats = new Stats(); |
| stats.domElement.style.position = 'absolute'; |
| stats.domElement.style.top = '0px'; |
| container.appendChild( stats.domElement ); |
| // |
| window.addEventListener( 'resize', onWindowResize, false ); |
| } |
|  |
| function onWindowResize() { |
| camera.aspect = window.innerWidth / window.innerHeight; |
| camera.updateProjectionMatrix(); |
| renderer.setSize( window.innerWidth, window.innerHeight ); |
| controls.handleResize(); |
| } |
|  |
| function animate() { |
| requestAnimationFrame( animate ); |
| controls.update(); |
| renderer.render( scene, camera ); |
| stats.update(); |
| } |

请大家仔细阅读一下代码，我想大多数的代码大家都能明白意思。本节课的重点集中在A bengin和A end之处。

最后，在强调一下模型加载完成后的回调函数，如下：

模型加载完成后，通过Mesh函数生成一个Mesh，并设置位置，最终显示在scene中。现在，你可以移动鼠标，全方位的看一下模型了。