

TONGJI UNIVERSITY

本科毕业设计（论文）

|  |  |
| --- | --- |
| 课题名称 | 基于Web3D智能可编辑  UI模板库 |
| 学 院(系) | 软件学院 |
| 专 业 | 软件工程 |
| 学 号 | 1352894 |
| 学生姓名 | 豆雨桐 |
| 指导教师 | 贾金原 |
| 日 期 | 2017/6/1 |

**基于Web3D的智能可编辑UI模板库**

摘 要

3D技术近年来飞速发展，无论是游戏行业，电影行业还是制造业，3D技术已经为人们的生活带来了前所唯有的便利和感官体验的飞跃。而由于三维场景与3D模型构建复杂，数据量大，难以交互，需要专业的渲染器和集成环境，3D场景与模型现在多存在于有限的平台，比如桌面应用程序，移动应用程序上，这意味着人们必须下载应用程序或者插件才能展示3D场景。这为3D场景的展示和发展造成了极大的不便和阻碍。互联网+是近几年发展最为迅速的行业，其发展迅猛的不可或缺因素即为其便利性，人们不需要下载任何插件或者应用程序，无需存储空间和注册账户就可以浏览各类数据。因此3D与互联网的结合即web3D也就应运而生，能在网页上随时随地，不限于设备与存储空间就可以看到渲染精美的三维模型和场景，并能与之交互，是十分奇妙与鼓舞人心的。

而因缺乏网站开发与界面设计经验，想要将三维模型和场景展示在网页上对于一个不熟悉前端语言与脚本语言的人是几乎不可能完成的。本文将阐述一个Web3D智能可编辑UI模板库的实现过程与使用方法，旨在为无编程经验的人提供图形化的3D场景编辑功能，也为工程师节省前端开发工作，使人们可以通过简单地参数输入，鼠标点击和拖拽就能生成美观合理的自定义三维场景，将自己的三维模型展示在互联网上。

**关键词：**Web3D，Three.js，Vue.js，Bootstrap，UI模板库，可视化编辑

**The Editable UI Templates Library based on Web3D**

**ABSTRACT**

**Key words：**WebGL, Three.js, Bootstrap, UI templates,

目 录

1.引言

1.1 系统的背景与意义

1.2 本文工作及创新性

1.3 本文章节安排

2.研究背景及相关技术

. 2.1 WebGL

2.2 Three.js

2.3 Dat.gui

2.4 Bootstrap

3概要设计

3.1 需求分析

3.1.1 功能性需求分析

3.1.2 非功能性需求分析

3.2 平台开发可行性分析

3.3 系统总体架构

3.4 UI设计理念

4 详细设计及关键技术

4.1 模型的上传与加载

4.1.1模型的本地上传

4.1.2 OBJ格式模型加载

4.1.3 STL格式模型加载

4.1.4 VTK格式模型加载

4.2场景光照生成

4.2.1 场景点光实现

4.2.2 场景面光实现

4.2.3 场景聚光灯实现

4.3 场景背景生成

4.3.2 Shader背景实现

4.3.3 二维背景图片实现

4.3.4 天空盒背景实现

4.4 交互方式实现

4.4.1 固定拖拽交互模式实现

4.4.2 自由拖拽交互模式实现

4.4.3 第一人称控制交互模式实现

4.4.4 自由漫游交互模式实现

4.5 场景渲染

4.5.1 场景雾化效果实现

4.5.2 场景网格系统实现

4.5.3 地面的实现

4.6 可拖拽小组件的实现

4.7 可编辑侧边栏实现

4.8 轻量化网站构建

4.9 网站操作引导实现

\*4.9 不同模型的上传 （未做）

\*4.10 自定义界面的保存（未做）

5.平台集成开发与测试

[6．结果和展望 27](#_Toc451698799)

[6.1 结果 27](#_Toc451698800)

[6.2 展望 27](#_Toc451698801)

[参考文献 28](#_Toc451698802)

[谢 辞 29](#_Toc451698803)

# 第一章 引 言

1.1 本课题的研究背景与意义

1.1. 1传统WebVR开发流程及问题

传统的WebVR应用程序开发流程一种是需要UI设计人员首先根据用户需求设计出系统原型，再交由程序员开发实现，第二种是程序员首先根据用户需求实现系统功能，再将整个系统交给UI设计人员美化。

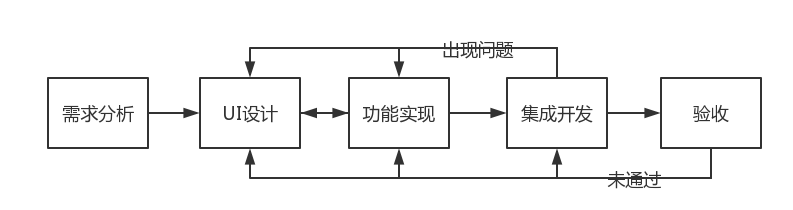


图1 传统开发流程1

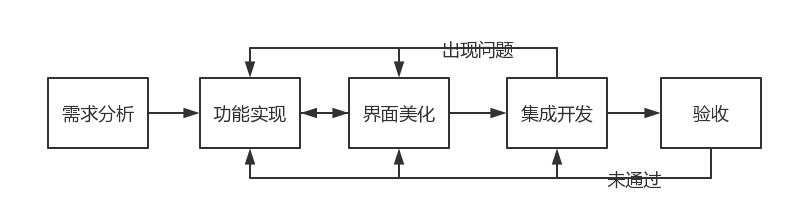
****

图2 传统开发流程2

传统WebVR应用程序开发流程造成的问题：

1)开发周期长，程序员或者UI设计人员必须得等对方完成所有工作后才能开始进行工作，开发工作不能并行发生，造成人力资源的浪费。

2)影响敏捷开发的效率，如果甲方对结果不够满意，提出很多改进意见，如果涉及到即与逻辑相关又与界面相关的问题，可能导致程序员看不懂UI设计人员改过的代码，又需要程序员与UI设计人员再次交流沟通，大大降低了敏捷开发的周期和效率。

3)UI设计人员与程序员之间交流不畅，若为第一种传统开发流程，会面临UI设计人员设计的很复杂花哨，但是技术上非常难以实现的问题，程序员就需要花大量的时间完成一个并不必要的UI设计，或者联系UI设计人员让其返工。若为第二种传统开发流程，UI设计人员又需要花费大量的时间精力去熟悉整个项目才能着手美化界面，这是非常耗时且完全不必要的。

1.1.2本文的研究目标与意义

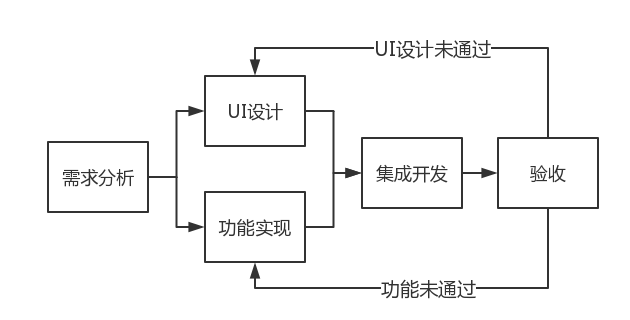


图3本文实现的WebVR开发流程

本课题旨在构造一套专门面向WebVR的UI模板库，使得UI设计与功能实现可以相对独立

如果不需要专门的UI设计，程序员可以使用系统提供的默认风格样式，使对界面与UI设计毫无经验的程序员能够使用当下最流行的前端样式库Bootstrap定义自己页面的样式风格。构造出界面简洁美观的Web3D应用程序，

如果需要UI设计人员对项目进行样式定制设计，那么本系统也能使UI设计人员不必学习webGL相关知识及底层逻辑，只需具有前端设计开发经验即可。并能使得程序员与UI设计人员可以相互独立，并行开发，高效互通，大大降低交流成本。

1.1.3本文工作

本文主要的工作就是在HTML5上创建一个基于Three.js的3D网页的UI模板，使得用户可以通过简单地点击，拖拽等操作，就可以自定义web3D场景。具体工作就是在webGL的技术基础上，整合Three.js的demo，将封装好的demo中的javaScript参数提取出来与Dat.gui对应，并实现一些Three.js的案例中未给出的渲染和显示功能，将参数提取与Dat.gui对应，从而使用户可以在网页上实时的操作这些参数来改变三维场景。提供一套相对科学美观的UI风格，并提供对部分UI模块样式的可选功能，系统贯穿有统一的设计风格和基本的设计理念，使得用户可以不用对UI设计有相关经验就能创造合理美观的UI界面。

由于Web3D是一门相对新的技术，虽然3d相关工作者可以熟练地构建三维场景与模型，但不能将3d模型以自定义的方式合理美观地展示在网页上。本文就结合最流行的bootstrap前端框架，整合出由多次web3D项目经验总结出的合理模板，提供最常用的3D场景渲染元素，结合dat.gui变量实时控制功能，使使用者可以简单地生成美观个性的web3D页面。

1.1.4 本文章节安排

本文将阐述基于web3D的一个UI模板库的实现，第一章为引言与概述，阐述系统搭建的背景与意义，综述本文所做工作。第二章将概述web3d相关技术知识及开发技术背景。第三章总结需求，从需求分析出发搭建系统架构，设计系统，并阐述本系统所涉及的UI理念。第四章为系统的实现，讲解在系统实现过程中学习到的新知识和解决的技术难题。第五章将使用本系统构建一个具有美观UI的web3d界面，并分析测试系统的功能和性能。第六章将总结该系统的开发的收获，并指出系统不足，对系统的后续完善做出展望。

# 第二章 研究综述及相关技术

2.1 WebVR研究现状.

2.1.1 WebGL

目前，主流的Web3D引擎主要有两个，首先是Adobe公司的Flash3D技术。基于该技术的可以在网页上绘制、呈现2D/3D图形，以及动画。但是，浏览器本身是不能直接显示Flash3D内容的，需要适时的根据提示，下载到内存，并实时安装网页版的Flash Player插件到网页上。

其次是微软的Silverlight3D技术，俗称“银光”，银光技术是一个跨浏览器，跨平台，并提供下一代基于.NET的媒体体验和丰富的交互式应用程序跨设备插件[[7]](http://en.wikipedia.org/wiki/WebGL#cite_note-WebGLWebsite-4)，遗憾的是，银光技术同样的需要在浏览器上安装插件。

除此之外，还有一些非主流的技术，这些技术都不如Flash3D或者Silverlight3D，而且同样的都有需要浏览器安装插件的缺点，因此就不再一一列举。

针对Flash3D、Silverlight3D以及WebGL的技术特点以及功能，做了如下的对比：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Web3D绘制技术** | **插件大小** | **需要安装** | **可持续性** | **是否支持Html5** | **性能** |
| Flash3D | 16.8M | 否 | 低 | 不支持 | Flash在2D性能最优越。但是3D上三者性能几乎相同。 |
| Silverlight3D | 6.6M | 是 | 低 | 不支持 |
| WebGL | 无 | 否 | 高 | 支持 |

**表格 1 主流网页3D技术对比**

2009 年 8 月 Khronos 提出 WebGL 绘图技术，它是一个跨平台、免费的、用于在 Web 浏览器创建三维图形的 API。WebGL基于 OpenGL ES 2.0 标准，并使用OpenGL 着色语言 GLSL，而且还提供了类似于标准的 OpenGL 的 API。WebGL 可以直接在 HTML5 的 Canvas 元素中绘制三维动画并提供硬件三维加速渲染，利用 WebGL 实现 Web3D 不需要安装浏览器插件，只需要编写网页代码即可实现三维图像的展示。

WebGL 技术的提出完美地解决了现有 Web交互式三维的问题，WebGL 技术标准免去了开发网页专用渲染插件的麻烦，可被用于创建具有复杂三维结构的网站页面，甚至可以用来设计三维网页游戏等。相对其他 Web3D 的实现方式，WebGL 的优势主要表现在：

1）WebGL 是通过 JavaScript 实现网络交互式三维 动画制作。基于此技术，客户端不需要安装插件即可支持三维图形展示及 GPU 硬件加速；

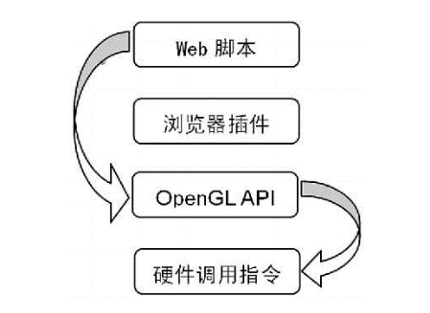
2）WebGL 具有开放性。目前，虽然 Adobe Flash Player 11、Silver light 3.0 都已经支持 GPU 加速，但它们都是私有的、不透明的；

3）WebGL 利用底层的图形硬件加速功能进行图形渲染，促使 Web 开发人员借助系统显卡来在浏览器里更流畅地展示三维场景和模型，同时还能创建复杂的导航和数据视觉化。

2.1.2 免插件的WebVR编程技术

现阶段主流的 Web 交互式动画是由浏览器插件调用 DirectX、OpenGL 等操作系统图形接口实现图形渲染，虽然这种方案实现了硬件加速，但是客户端必须下载插件。在网络环境不理想的情况下，一个插件的下载和安装至少需要几分钟的时间，有些用户因担心插件会对电脑带来危害而不愿下载相关插件，有些则因网络传输速度的限制需要用户等待较长时间，用户的兴趣就会大大降低，进而导致用户量的减少。WebGL 技术为 Web3D 免插件的实现带来了福音。WebGL 直接以 OpenGL 接口实现 HTML5 的 canvas 标签调用，以统 一的 OpenGL 标准，从 Web 脚本生成利用硬件加速功能的 Web 交互式三维的图形渲染（如图 所示）。

2011 年 3 月，多媒体技术标准化组织 Khronos 在 美国洛杉矶举办的游戏开发大会上发布WebGL标准规范 R 1.0，支持 WebGL 的浏览器不借助任何插件便可提供硬件图形加速从而提供高质量的三维体验。说明WebGL已经渐渐走向成熟，是非常值得信赖和非常有发展前景的新兴技术。



WebGL免插件图形渲染

2.1.3 流行的WebGL开源框架

使用原生WebGL接口开发是十分繁琐而且复杂的，而现在已经有很多不错的WebGL开源框架。有PhiloGL、Babylon.js、X3DOM和Three.js等。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **名称** | **数据流** | **场景绘制** | **支持格式** | **API** | **编辑器** |
| Three.js | 海量点云数据流（与Potree拓展组合） | 支持大型场景，性能良好 | 标准网络文件格式 | 唯一的充当WEBGL顶层的架构，能够直接访问WebGL的功能，直接实现复杂功能 | 官方demo，类似于Unity3d，图形界面提供控件控制式开发 |
| PhiloGL | 普通的数据流，数据对象包含维度与纹理等 | 支持大型场景，性能一般 | 标准网络文件格式 | 较为成熟的API | 无 |
| Babylon.js | 普通数据流，并通过储存本地数据库保证运行效率 | 支持大型场景，性能一般 | 标准网络文件格式 | 较为成熟的API | 在线编译器，纯代码编程但是能看到运行结果 |
| X3DOM | 支持逐行多边形流（有限制）[[17]](http://en.wikipedia.org/wiki/WebGL#cite_note-WebGLWebsite-4) | 支持大型场景，性能一般 | 基于ISO标准交换格式，允许长期储存 | 完整的API；且能绘制高水平的三维场景 | 无 |

**表格 2 WebGL框架综合比**

由上表也不难看出，相对于其它现有的WebGL框架，Three.js是最为成熟和健全的，因此Three.js正是其中最受欢迎的一个。它掩盖了3D渲染的细节，将原生的API细节抽象化，把3D场景拆解为一些对象种类。它是面向对象的，功能相对于原生WebGL也更加丰富，同时保持着极高的性能，本身也支持了WebGL本身不具有的picking功能，可以方便的为应用添加WebGL本身并不包含用户交互功能。可以方便的用于虚拟现实呈现、3D游戏开发，动画制作，以及特殊的视觉效果制作等。难得的是Three.js还内置了文件格式的支持，可以通过主流的建模软件导出文本格式文件通过three.js加载，也可以使用Three.js自己的JSON格式或二进制格式，这一功能对我们项目的开发也是十分重要。

Three.js从2010年4月第一次发布以来至今，已经更新过七十余次。官方网站上也有丰富的demo和活跃的论坛，其功能与API都趋于完整和健全，因此选择Three.js作为项目框架是十分合理和必要的。

2.3 面向WebVR的界面设计工具

Dat.gui同Three.js一样也是托管在github上的开源库，它能以图形化的界面实时的操作javascript里的变量，即在网页运行时就可通过拖拽、点击、输入等操作对javascript变量进行更改。三维场景模型庞大，参数复杂，如能在页面上就对模型与场景参数进行监控与操作，无疑大大降低了web3d页面创建的复杂程度与不可控性。因此将Dat.gui与Three.js结合是顺理成章的，将参数图形化地显示在页面上后，使用者无需每次都在IDE中修改各类参数，重跑项目并在调试窗内观测变量值和刷新三维场景渲染效果，而只需在页面上可视化的实时更改参数即可明确的观察

2.4 WebVR界面样式库

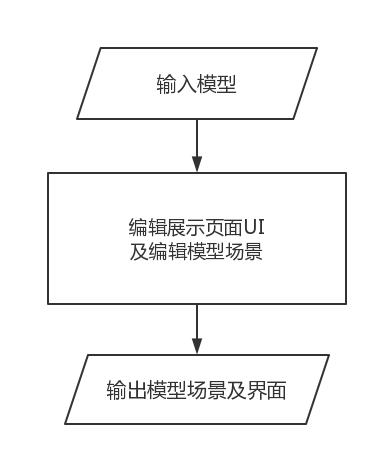
Bootstrap 框架来自 Twitter，是一个 CSS /HTML 框架，由设计师 Mark Otto 和 Jacob Thornton 合作开发。它基于 HTML、CSS、JavaScript 技术，提供一套直观的web设计工具包，可以用来开发跨浏览器并且美观大气的页面; 还提供很多流行的、样式简洁 的 UI 组件、栅格系统以及一些常用的 JavaScript 插件，使得 Web开发更加快捷。

Bootstrap 框架自推出后颇受欢迎，一直是GitHub上的热门开源项目，包括 NASA 的 MSNBC( 微软全国广播公司) 的 Breaking News 都使用了该项目，是目前欧美国家中最流行的前端框架。Bootstrap 框架具有以下重要特性:

1. 一套完整的基础 CSS 插件；丰富的预定义样式表
2. 一组基于jQuery 的 JavaScript 插件集;
3. 一个非常灵活的响应式( Responsive) 栅格系统，并且崇尚移动先行( Mobil First) 的思想。 Bootstrap 框架包含丰富的组件(即插件) ，包括下拉菜单、按钮组、按钮式下拉菜单、导航条、分页、排版、缩略图、警告对话框、进度条等，根据这些组件，可快速搭建一个风格简约、功能完备的网站。
4. Bootstrap 框架自带一组 jQuery 交互插件，包括模式对话框、标签页、滚动条、弹出框等，不但功能完善，而且十分精致，正在成为众多 jQuery 项目的默认设计标准。而这些模块化的组件也易于修改，通过修改变量就可以形成自己的独特风格

# 第三章 概要设计

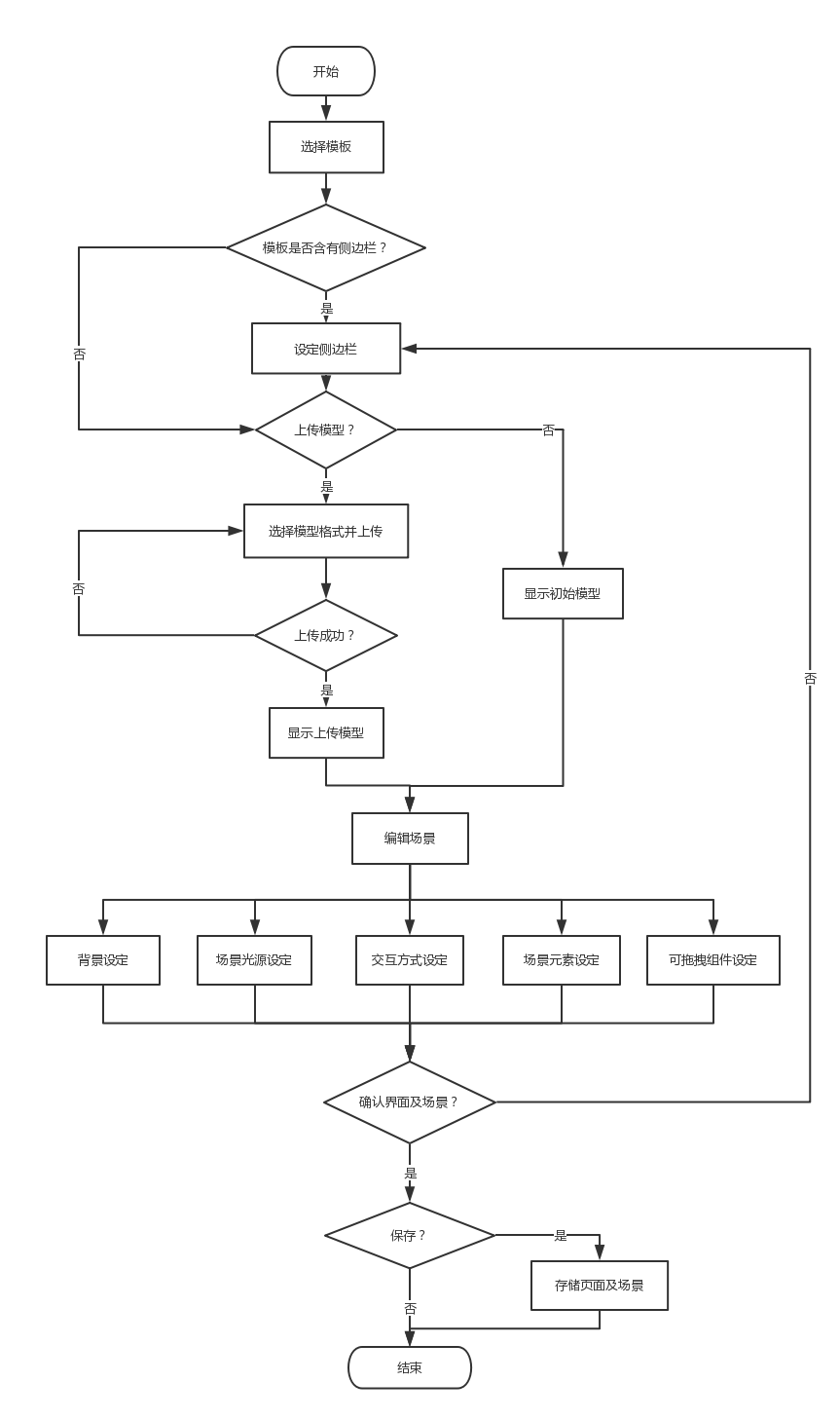
3.1需求分析  
3.1.1功能性需求   
 用户可上传各类型格式的模型文件，用户在选择好上传的模型格式后，可以通过点击或者拖拽的方式上传相关模型文件，选定后时显示上传进度与结果。  
 上传成功后上传界面界面消失，显示不同风格的模型加载进度条样式，选择好后转入模型模板编辑主页面，选择好页面模板后，在canvas窗口中显示刚才选择的加载进度条，进度条加载完成后自动消失，以用户选择的模板显示上传的模型。  
 对上传模型进行可视化编辑，包括模型的展示背景，模型的场景光照，模型的交互方式和模型场景的附加组件等。并可生成并拖拽含有对三维场景展示方式基本变换操作的小组件。  
 在用户编辑好自己的三维场景后，可点击保存按钮，保存html页面及其包含的三维场景



需求分析

3.1.2非功能性需求

1. 网站不会因为模型过大或者元素较多等因素崩坏。无论如何给场景中增加元素，都可以保证用户可以流畅的与模型交互，没有延迟与卡顿。
2. 可以较快的上传较大的模型，较快保存场景。
3. 操作页面简单美观，风格统一。



系统流程图

本项目是一个单页应用系统，提供两种基本模板，全屏模板和带侧边栏的模板。用户选好模板之后，可以选择观察默认的三维模型，也可上传自己的三维模型，就可以在模板的基础上编辑自己的三维场景。本文提供的可自定义三维场景元素有：背景、灯光、交互方式、场景渲染和可拖拽小组件五种。其中背景包括纯色背景、Shader背景、二维静态背景和天空盒背景四种选项，灯光包括点光、面光和聚光灯三种，交互方式包括固定拖拽交互，全局拖拽交互，第一人称视角交互和自由漫游视角交互四种，场景渲染分为雾化效果、网格元素和地面三种，可拖拽小组件分为样式不同的5种。用户可以在这五大类自定元素中自由选择需要的三维元素，这五类自定元素可以互不干扰的独立存在，大大增加了场景的丰富性和多样性。

3.2 UI设计理念

本系统遵循“KISS”UI设计理念，即“Keep It Simple and Stupid”，摒弃了花哨的Jquery插件库和各类动画，而是统一使用Bootstrap提供的动画和样式，摒弃了设计纷繁的icon和logo，统一使用FontAwesome提供的字体图标。并尽量保证所有网站元素风格统一，包括吸引视点的主题颜色都是使用rgb值完全相同的朱红色，所有block采用圆角为3像素带3像素灰色投影。网页采用响应式布局，均以百分比、相对布局等布局方式保证网站无论在任何分辨率的设备上都可以正常显示。

# 第四章 详细设计及关键技术

4.1模型的上传与加载

4.1.1 模型的上传

//创建输入类型为文件的输入框

**var fileInput** = **document**.createElement( **'input'** );  
**fileInput**.**type** = **'file'**;  
//读取上传的文件

//此处loader为各种loader的总称，具体类别将在下文介绍**fileInput**.addEventListener( **'change'**, **function** ( event ) {

***loader***.loadFile( **fileInput**.**files**[ 0 ] );  
});

4.4.1 STL格式模型加载

STL 为 stereolithography 的缩写，含义为光固化立体造型术。 是一种为快速原型制造技术服务的三维图形文件格式。 由于其具有解析简单，文件体积较小等优点，被广泛用于工业领域。 STL 将三角形作为基础图形，对模型表面进行组织，因此十分适合在电子设备上进行绘制。同时三角形的面可以表现流畅的曲线，使模型外观更加细腻， 所以说使用STL文件对构建高质量模型发挥重要作用。 正是由于其具有上述特性，最近发展异常火热的 3D 打印技术都是以 STL 作为标准格式。目前的 STL 文件格式包括二进制文件 (BINARY) 和文本文件 (ASCII) 两种。下面将分别对这两种格式的解析方法进行介绍。

1) STL 文本文件的解析方法

STL 文本文件以行为单位顺序给出三角形面片的相关数据。每行的开头为一到两个关键字，表明后面数据的相关含义。STL文件中的关键字 facet 代表一个三角形面片的信息单元。 此单元中包括关键字 vertex 和 normal， 分别代表三角形的顶点坐标和法向量数据。 若干个三角形面片相互组织就形成了最终的 STL 三维模型，具体格式说明如下所示：

solid filename stl //自定义文件头

facet normal x y z //三角面片的法向量

outer loop vertex x y z //第一个顶点坐标

vertex x y z //第二个顶点坐标

vertex x y z //第三个顶点坐标

endloop endfacet //完成一个三角面片定义

…

endsolid filename stl //定义结束

2) STL 二进制文件的解析方法

二进制 STL 文件通过固定的字节数来描述三角形面片的相关信息。文件的起始 80 字节是文件头部分，此部分内容中存储模型名，开发人员可以在其中存储描述模型的任何文字信息。 后面紧随一个 4 字节的整数，其代表此模型中的三角形面片的总个数。此整数后面存储是每个三角形面片的几何信息。每个三角形面片占用固定的 50 字节，其中包含的信息依 次是3个4字节浮点数，代表三角形面片的法向量信息。后存储的9个4字节的浮点数，每3个浮点数作为一组，分别代表三角形面片顶点的 XYZ 坐标。每个三角面片的最后 2 个字节用来描述三角面片的属性信息。由此可以出一个二进制 STL文件的大小为三角形面片数乘以 50 再加上 84字节。

3) 通过 Three.js 引擎加载 STL 模型的具体步骤

了解了STL文件的格式我们可以对Three.js的loader进行解读，也更方便自定义我们想要自定义的loader参数。目前本课题只提供默认的加载方式。

首先要找到 STL 文件对应的加载器文件 STLLoader.js。 此文件中封装了读取 STL 二进制文件和 STL 文本文件数据并初始化网格体的方法，导入STLLoader.js。

加载 STL 文件 渲染场景初始化工作完成之后，就可以对 STL 模型进行加载了。主要思路为： 先创建一个 STL 格式的加载器， 创建完成后利用加载器的 load 方法对模型进行加载。 load 方法中第一个参数为 STL 模型文件所在位置和名称。 第二个参数为 STL 文件读取完成后，程序需要进行的一系列操作，其中通常包括设置模型的材质和设置缩放倍数等。 具体的代码部分如下所示：

//新建 STL 格式的加载器

**var *loader*** = **new** THREE.STLLoader();  
***loader***.load(**"model/model.stl"**, **function** (geometry){ ***group*** =  
**new** THREE.Mesh(geometry, ***mat***); });

上述代码 中 render 方法为渲染一次画面的方法。为了展示模型的每一个细节，程序在两次绘制之间的间隔中不断改变模型的旋转角度，使运行效果更加流畅多样。

4.1.2 OBJ格式模型加载

OBJ 文件格式简介 OBJ 文件是 Alias/Wavefront 公司开发的三维动画软件 Maya 的一种数据输出类型,它定义了对象的几何和其它的一些特性。由于它的文件结构非常简单，所以适合在应用程序中读取或进行 3D 文件格式转换。OBJ文件和STL文件一样，可以是二进制文件，也可以是 ASCII 文件，其中二进制格式文件的扩展名是.mod，ASCII格式文件的扩展名是.obj。

OBJ 文件格式结构也相对简单，与STL使用三角形构成面的做法不同，它使用不限制边数的多边形构成面，因此相对STL格式来说，可以展示结构更为复杂和精妙的模型，同时也带来模型太重等弊端。OBJ文件保存一些多边形的信息，诸如顶点点几何坐标、纹理坐标，点与点如何连线组成线、面等等。用一个一个的多边形面片拼合构造出三维物体。

下面简要的介绍obj 文件的文件格式格式和基本结构。 OBJ 文件不需要任何种文件头(File Header)，尽管经常使用几行文件信息的注释作为文件的开头。 OBJ 文件由一行行文本组成，重要的数据有顶点数据（Vertex data）： v：几何体顶点(Geometric vertices)。 表示本行指定一个顶点。 此前缀后跟着 3 个单精度浮点数，分别表示该定点的 X、Y、Z 坐标值； Vt：贴图坐标点(Texture vertices)。 表示本行指定一个纹理坐标。 此前缀后跟着两个单精度浮点数。 分别表示此纹理坐标的 U、V 值。 Vn：顶点法线(Vertex normals)。 表示本行指定一个法线向量。 此前缀后跟着 3 个单精度浮点数，分别表示该法向量的 X、Y、Z 坐标值。

OBJ 文件不包含面的颜色定义信息，不过可以引用材质库，材质库信息储存在一个后缀是".mtl"的独立文件中（关键字"mtllib" 即材质库的意思）。 材质库中包含材质的漫射(diffuse)，环境(ambient)，光泽(specular)的 RGB(红绿蓝)的定义值，以及反射(specularity)，折射(refraction)， 透明度(transparency)等其它特征。

在Three.js中加载OBJ格式的三维模型时，首先要找到 OBJ文件对应的加载器文件 OBJLoader.js和MTLLoader.js。然后创建一个 OBJ格式的加载器， 创建完成后利用加载器的 load 方法对模型进行加载。具体的代码部分如下所示：

//首先创建MTLLoader  
**var** mtlLoader = **new THREE**.MTLLoader();

//找到存放关于这个OBJ模型所有文件的文件根目录路径  
mtlLoader.setPath( **'../../'** );  
mtlLoader.load( **'xxx.mtl'**, **function**( materials ) {

//将导入的模型材质传入函数内部，  
 materials.preload();

//在MTLLoader内部创建OBJLoader  
 **var** objLoader = **new THREE**.OBJLoader();

//设置OBJ的材质为MTLLoader传入的材质  
 objLoader.setMaterials( materials );   
 objLoader.load( **'xxx.obj'**, **function** ( object ) {  
 …//设置模型位置，大小，角度等基本属性  
 ***scene***.add( object );  
 });  
});

4.1.3 VTK格式模型加载

VTK全称Visualization Toolkit ，具有强大的[三维](http://baike.baidu.com/item/%E4%B8%89%E7%BB%B4)图形功能。VTK既支持基于体素Voxel-basedrendering 的[体绘制](http://baike.baidu.com/item/%E4%BD%93%E7%BB%98%E5%88%B6)Volume Rendering又保留了传统的面绘制，从而在极大的改善可视化效果的同时又可以充分利用现有的图形库和图形硬件，且具有设备无关性，使其代码具有良好的可移植性。VTK格式的文件着重于展示模型的网格。同上两种模型文件格式一样，有ASCII与二进制两种保存形式，文件格式也与OBJ格式大同小异，不做赘述。

Three.js也支持VTK格式的解析，过程如下：   
//在接收到vtk文件之后，在回调函数设置了位置，材质等元素。

**var *loader*** = **new** THREE.VTKLoader();  
loader.load( **"../../xxx.vtk"**, **function** ( geometry ) {  
 **var** mesh = **new** THREE.Mesh( geometry, material );  
 …//设置mesh位置，大小，角度等基本属性  
 scene.add( mesh );  
} );

4.1.4 JSON格式文件的加载

JSON格式[7]是一种轻量级的数据交换格式，目前在数据传输领域非常常见。three.js对blender最为友好，官方提供了插件可以在blender中导出three.js可以用的json格式。将生成的json文件可以直接导入three.js。具体实现方式如下：

//创建一个loader  
 **var objectLoader** = **new** THREE.ObjectLoader();  
 // loader引入json文件即可

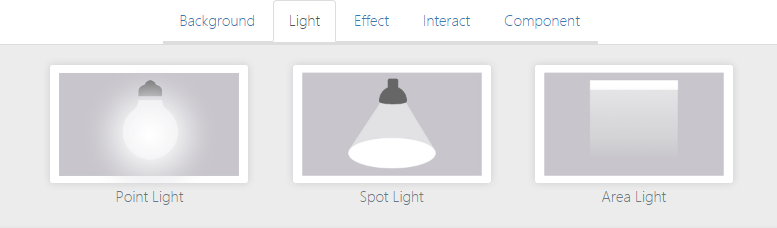
objectLoader.load(**"javascript/js/teapot-claraio.json"**, **function** ( obj ) {  
 **scene**.add( obj );  
} );

4.1.1导出场景关键代码：

**var output** = **scene**.toJSON();  
 **output** = **JSON**.stringify( **output**, **null**, **'\t'** );  
 **output** = **output**.*replace*( /[\n\t]+([\d\.e\-\[\]]+)/g, **'$1'** );  
 **output** = **JSON**.stringify( **output** );   
 saveString( **output**, **'scene.json'** );

4.1场景光照生成

本文提供三种在三维场景中最常用的光源模式，分别为点光源，面光源与聚光灯光源，如下图：

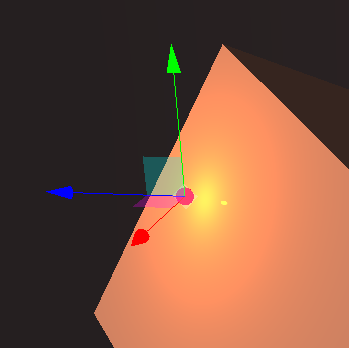
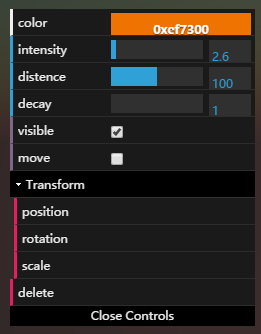


4.1.1 场景点光实现

点光源指的是可以向各个方向发光的光源。它不产生阴影，因为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，严重影响渲染效率，使帧数大幅降低，所以不能产生阴影。

Three.js提供的点光源接口有：color——光源颜色； intensity——强度，也可以理解为亮度；distance——距离，光线能到达的最远的距离；decay——光源的衰弱速度；visible：可见性，如果为true，光源本体就会以一个球形显示出来，否则就只能看到光线对周围环境造成的影响，而看不到光源本身。

本文为光源增加了move属性，使得光源可以绕物体不断的旋转。效果如图：

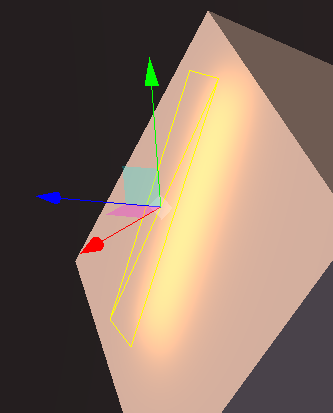
点光源效果图 点光源可控参数

4.1.2 场景区域光

区域光也称面光源，可以对周围对象产生阴影。

Three.js提供的区域光接口有：width——光源宽度； height——光源高度，color——光源颜色；intensity——光源强度；visible：可见性，如果为true，光源本体就会以一个带有颜色的矩形显示出来，否则就只能看到光线对周围环境造成的影响，而看不到光源本身。

本文为面光源也增加了move属性，使得光源可以绕物体不断的旋转。效果如图：

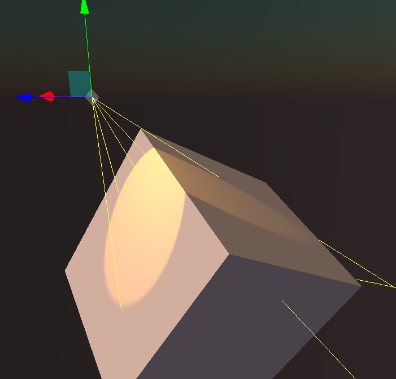
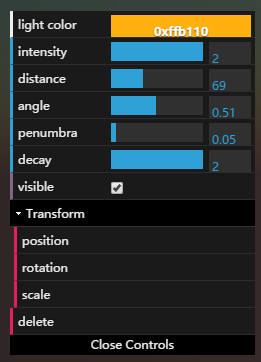
 

面光源效果图 面光源可控参数

4.1.3 场景聚光灯实现

场景聚光灯效果和点光源类似，但是限制了光源散射的方向，因此可以使用户将光投射在特定的物体上，这样就可以在渲染器能承受的范围内让特定的物体产生阴影。

Three.js提供的接口非常丰富，本系统抽离出较为常用的部分参数，包括color——颜色；intensity——光源强度；distance——光源可达最远距离；angle——聚光灯顶角的角度；decay——光强度衰减速率；visible：可见性，如果为true，光源本体就会以一个锥形线框显示出来，方便观测与调试，否则就只能看到光线对周围环境造成的影响，而看不到光源本身。

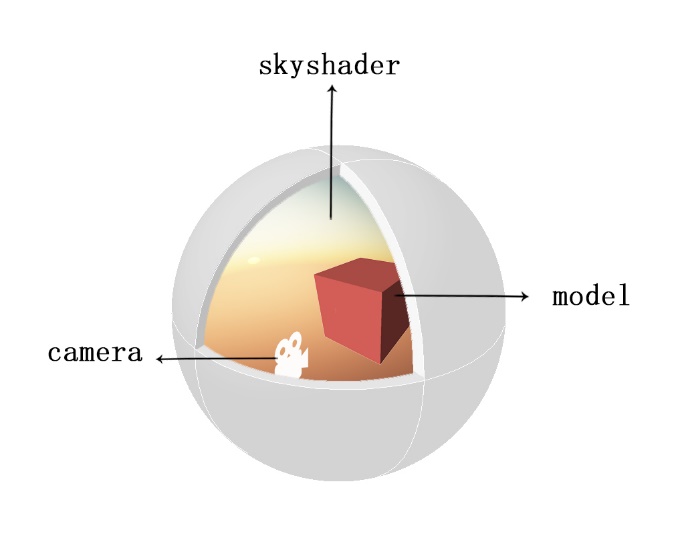
聚光灯效果图 聚光灯可控参数

3.2 场景背景生成

3.2.1 Shader背景实现

Shader就是一种能把线框图转变为漂亮的 3D 形象的系统。当设置了所有的属性与参数，如高光属性、纹理与凹凸度等时， Shader系统就会工作。它将计算出我们给出的表面外观。Shader以许多类型与风格出现，而且许多种相似的Shader有不同的名字，这都容易使人混淆不清。一个3D程序有不同种的Shader，因为在真实世界里有数以百万计的不同表面，它们对光都有不同的反应。Shader 的一个任务就是去模拟对象与光之间的相互作用，并用最精确的方式把它们展现在观众面前。 从技术的角度看，Shader的作用是指出一个表面上的每一个点是怎样表现的，它对光会做出什么样的反应。有的 Shader描绘出光线的散射，有的指出颜色，有的设置凹凸度，其它的则设置透明度。有的Shader使一个表面看上去像皮毛，有的让它闪闪发光。因此实在是很难把所有Shader加以分类，因为有许多种做各式各样不同工作的Shader。

本文所用到的skyshader原理也是如此，我们定义一个巨大的球体包裹在场景外部，然后使用shader渲染球体的内部材质，形成形似天空的均匀渐变。原理如图：



SkyShader原理图

首先我们定义一些影响shader最终渲染效果的重要参数，比如天空亮度，天空通透度等，代码如下：

**var** skyShader = {  
 **uniforms**: {

//提取出的重要参数  
 },

**vertexShader**: [

//顶点渲染代码

]

**fragmentShader**: [

//面片渲染代码

]

//将skyshader中定义的重要参数整体克隆出来，变为父对象Sky的变量。

**var** skyUniforms =**THREE**.**UniformsUtils**.clone( skyShader.**uniforms** );  
//定义sky的渲染材质，材质为上文定义好的skyshader  
**var** skyMat = **new THREE**.ShaderMaterial( {  
 **fragmentShader**: skyShader.**fragmentShader**,  
 **vertexShader**: skyShader.**vertexShader**,  
 **uniforms**: skyUniforms,

//材质贴在球体内部

**side**: **THREE**.**BackSide**

} );  
//定义sky的线框，即一个半径为450000的球体  
**var** skyGeo = **new THREE**.SphereBufferGeometry( 450000, 32, 15 );

//定义sky的线框，即一个半径为450000的球体  
  
**var** skyMesh = **new THREE**.**Mesh**( skyGeo, skyMat );

//暴露sky对象及其内部接口，即之前定义的重要参数，使得在其他地方也可以对//skyshader的参数做引用和操作

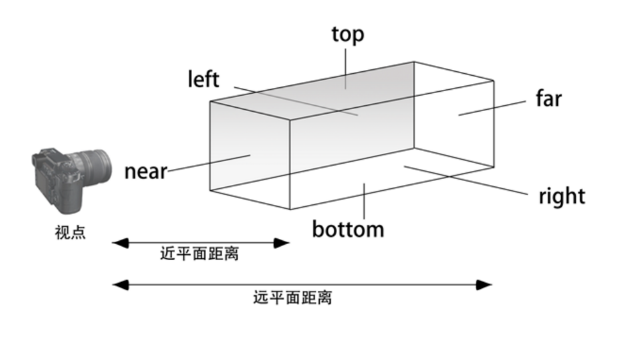
**this**.**mesh** = skyMesh;  
**this**.**uniforms** = skyUniforms;

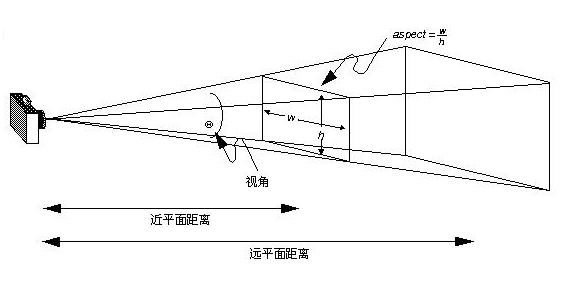
}

3.2.2二维背景图片实现

由于整个场景是三维的，用户可以通过改变相机的位置与视角与场景进行交互，如果只是在相机面前放一块贴有图片的平面作为背景的话，无论该平面有多大，难免在交互时会看到图片的边缘。 而本文构思在用户与三维模型交互时，二维的背景图片应当是保持固定的，即无论用户如何与场景交互，如何移动和旋转相机，都看到的是固定的背景图片，不会看到背景的边缘。

在三维场景中，camera有两种主要的类型，一种是透视相机(perspectiveCamera)，如图1，这种相机就如同人眼与一般相机一样，看到的物体近大远小，有着明显的透视效果，这种相机主要有四个参数，视角大小，视角纵横比，能看到的最近距离与最远距离。通过定义这四个参数以及该相机的位置与角度，就可以模拟人眼在场景中的任何变换。而另一种叫做正交投影相机(OrthographicCamera),该相机垂直于投影面进行投影，因此物体在投影之后的比例不会发生改变，如图二所示。这种投影在建筑制图，工业设计领域运用非常广泛，因为人们可以不受视角与距离约束的看清模型的各个组件。正交投影照相机的构造函数为OrthographicCamera(left, right, top, bottom, near, far)包含了六个参数，left就是整个投影线区域的左侧面，right就是投影线区域的右侧面，top就是投影线区域的上侧面，而bottom就是投影线区域的下侧面。 near是到距离相机较近的那一面的距离，far是离距离相机较远的那一面的距离，这六个投影面围成的区域就是相机投影的可见区域。在三维空间中，只有这个区域中的物体才能被照相机所观测到。





因此想出给场景中再加一个camera的想法，场景主相机是用户看到的视角，为透视相机，如果我们再给场景中加一个正交投影相机，该相机与主相机位置相同，用来观测二维图片，即二维背景图片只会被正交投影相机观测而不会在主相机内观测到，那么此时二维图片就不会因为相机的前后左右移动而产生透视效果，而是固定不动了。代码如下：

//二维背景图片路径

**var** texture1 = **THREE**.**ImageUtils**.loadTexture( **'images/image-backgrounds/imagebg-'**+imageNum+**'.jpg'** );

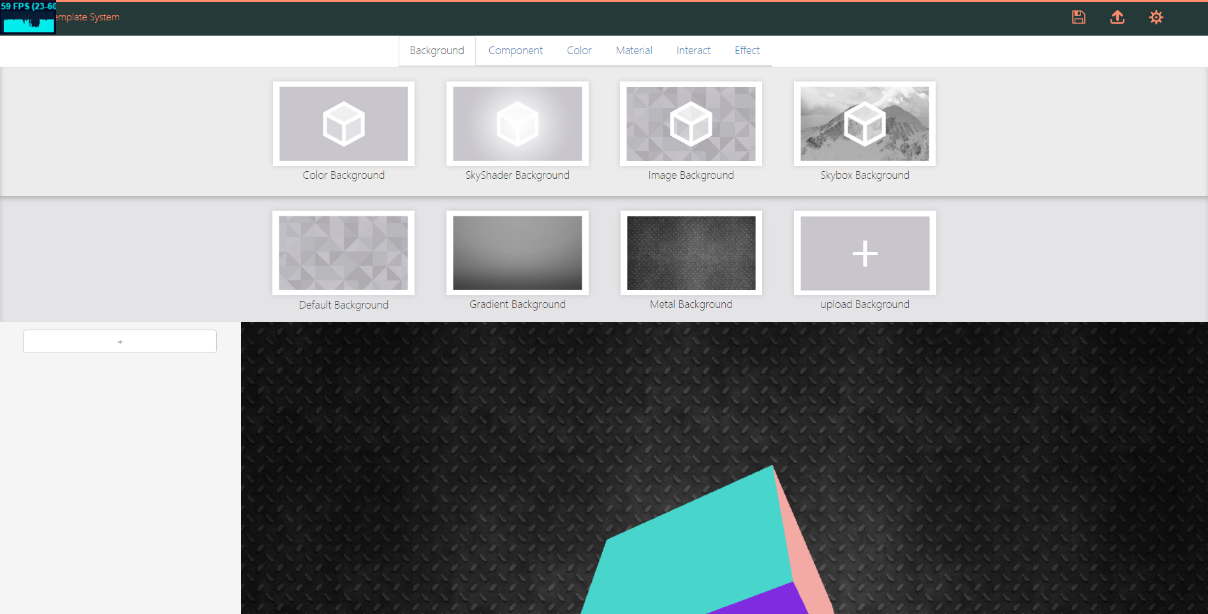
//贴有背景图片的平面**var** backgroundMesh = **new THREE**.**Mesh**(  
 **new THREE**.PlaneGeometry(2, 2, 0),  
 **new THREE**.MeshBasicMaterial({  
 **map**: texture1  
 }));  
//贴有背景图片的平面  
***backgroundScene*** = **new THREE**.Scene();//正交投影相机

//正交投影相机  
***backgroundCamera*** = **new THREE**.OrthographicCamera(0,1000);

//将正交投影相机添加到场景中  
***backgroundScene*** .add(***backgroundCamera*** );

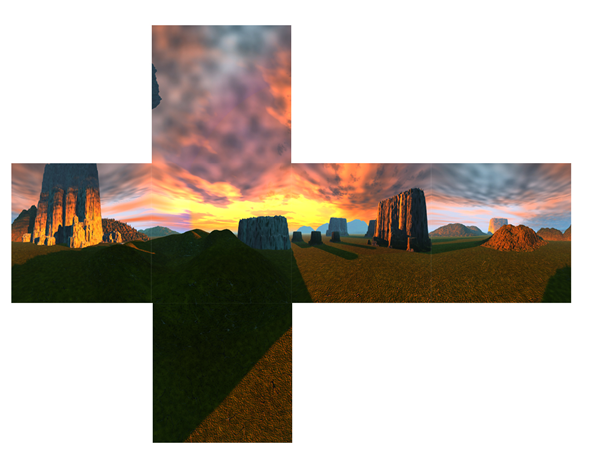
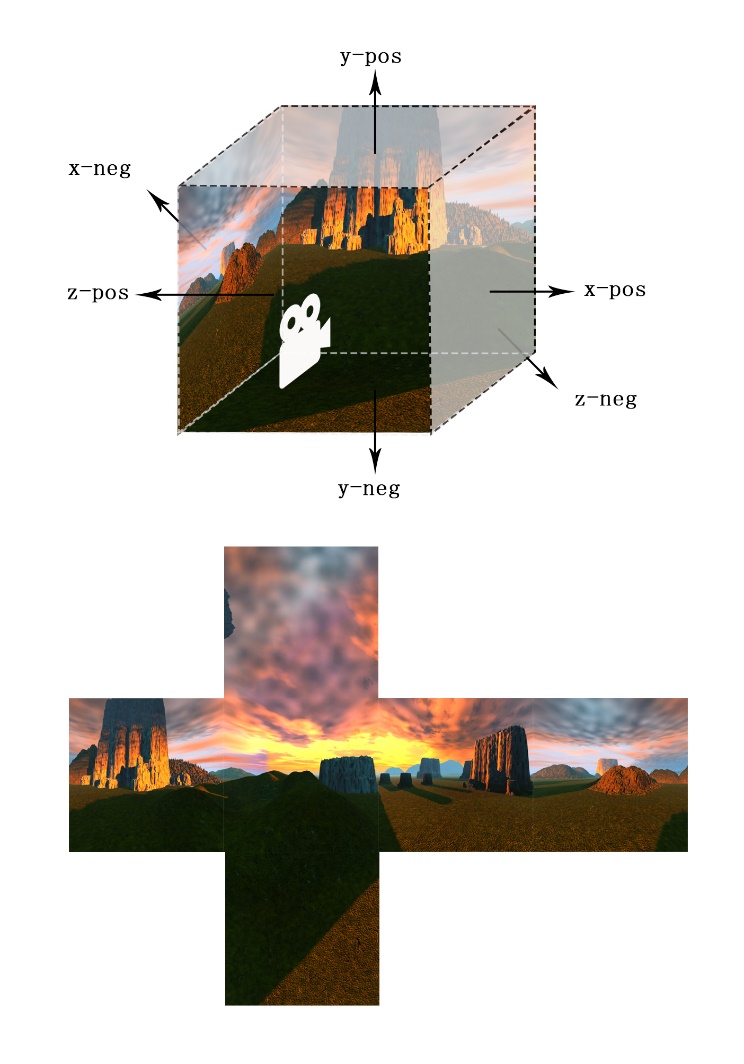
//将贴有背景图片的平面添加到场景中  
***backgroundScene*** .add(backgroundMesh );

效果如图：



* + 1. 天空盒背景实现

天空盒同现在非常流行的全景图片生成的原理一样，是在相机外围覆盖一个内部贴有场景图片长方体，如图所示



在这个盒子的各个面上贴有表示周围环境纹理而无缝衔接的图片，从而生成有真实感的场景，由于本文需要提供不同的天空盒以供用户选择，因此将不同的天空盒放在同一个文件目录中，通过统一的命名方式，即同一个天空盒的六张贴图放在一个文件目录下，六张贴图命名前缀相同，后缀以**"-xpos"**, **"-xneg"**, **"-ypos"**, **"-yneg"**, **"-zpos"**,**"-zneg"**命名，从而区别六张贴图的方向和次序，在加载天空盒贴图时，只需根据用户点击传入天空盒的名字，获得该天空盒对应的六张贴图，重新以这六张贴图生成天空盒，即可达到点击生成不同天空盒的效果，并且整个场景任何时刻只存在一个天空盒，保证了网页的运行速率和场景的轻量化。代码如下。

//天空盒贴图图片的路径

**var** imagePrefix = **"images/skyboxs/"**+skyboxName+**"/"**+skyboxName+**"-"**;

//天空盒6张贴图图片分别对应的天空盒字的6个面的方向

**var** directions = [**"xpos"**, **"xneg"**, **"ypos"**, **"yneg"**, **"zpos"**, **"zneg"**];

//天空盒贴图图片的格式

**var** imageSuffix = **".png"**;

//生成6面体的线框  
**var** skyGeometry = **new THREE**.CubeGeometry( 5000, 5000, 5000 );  
//天空盒的材质  
**var** materialArray = [];

//为6个面生成相同的材质，定义其贴图贴在盒子内部  
**for** (**var** i = 0; i < 6; i++)  
 materialArray.push( **new THREE**.MeshBasicMaterial({  
 **map**: **THREE**.**ImageUtils**.loadTexture( imagePrefix + directions[i] + imageSuffix ),  
 **side**: **THREE**.**BackSide** }));  
**var** skyMaterial = **new THREE**.MeshFaceMaterial( materialArray );

//以上述线框和材质的定义生成天空盒实例  
**skyBox** = **new THREE**.**Mesh**( skyGeometry, skyMaterial );

//将天空盒加入到场景中  
**scene**.add( **skyBox** );

3.3 交互方式实现

本文提供四种基本交互模式，用户可根据自己的喜好和方便在四种交互方式中任意切换，如图：



四种交互方式

4.3.1 固定拖拽交互模式实现

因为许多情况下用户是需要专注的展示和观察3D模型，而不需要对背景有过多的渲染，因此能只对模型操作的交互方式是必不可少的。本文实现了一种可以对模型进行拖拽来旋转模型的交互方式。具体实现方式是监听html上的鼠标事件，获取鼠标的移动距离来对模型进行对应的旋转操作，因为三维场景是按帧渲染的，如果我们对模型有了操作，就应当对整个场景进行渲染，而拖拽旋转事件要求物体可以流畅的连续旋转，因此让animate函数调用three.js中的requestAnimationFrame(*animate*)函数，即每一帧animate都会调用自己，那么如果将三维场景渲染函数放在animate中，就可以做到每一帧都可以更新三维场景，也即实现了拖拽交互。代码如下：

//为html添加鼠标点击事件监听，如果点击了鼠标则调用*onDocuemntMouseDown函数。*

**document**.addEventListener( **'mousedown'**,*onDocuemntMouseDown*,**false**)

**function** *onDocuemntMouseDown*(event) {

//阻止执行点击事件默认触发的函数   
 event.**preventDefault**();

//判断鼠标点击范围是否在三维场景内，如果不在，就取消监听，如果在，就增加鼠标移

//动，鼠标松开和鼠标是否处于浏览器外三种监听   
 **if**(event.**target**.**tagName** != **"CANVAS"**){

**document**.removeEventListener( **'mousemove'**, *onDocumentMouseMove*, **false** );  
**document**.removeEventListener( **'mouseup'**, *onDocumentMouseUp*, **false** );  
**document**.removeEventListener( **'mouseout'**, *onDocumentMouseOut*, **false** );  
}**else**{  
**document**.addEventListener(**'mousemove'**,*onDocumentMouseMove*,**false**)  
 **document**.addEventListener(**'mouseup'**,*onDocumentMouseUp*,**false**)  
 **document**.addEventListener(**'mouseout'**,*onDocumentMouseOut*,**false**)  
 }

}

//如果鼠标在场景内做了点击并移动，那么就检测鼠标的移动距离，将其距离乘以旋转速

//度获得模型的旋转角度

**function** *onDocumentMouseMove*(event) {

//鼠标横向移动距离

***mouseX*** = event.**clientX**-***windowHalfX***;  
 ***targetRotation*** = ***targetRotationOnMouseDown***+(**mouseX**-***mouseXOnMouseDown***)\*0.02;

//鼠标横向移动距离  
 ***mouseY*** = event.**clientX**-***windowHalfX***;  
 ***targetYRotation*** = ***targetYRotationOnMouseDown***+(**mouseY**-***mouseYOnMouseDown***)\*0.02;  
}

4.3.2 自由拖拽交互模式实现

因考虑到用户在拖拽时可能不仅想要看到模型的旋转，也希望看到背景场景的其他角度，从而增设自由拖拽交互模式，该模式与固定拖拽交互模式实现非常相似，唯一不同的是在监听到鼠标拖拽事件时，不是对模型的角度做变化而是对camera的角度做变化，从而实现整个场景旋转的效果，不做赘述。

4.3.3 自由漫游交互模式实现

自由漫游交互模式包括鼠标控制视角和键盘控制移动两部分，鼠标控制视角与前两种交互模式相同，即通过给鼠标增加监听，获得鼠标位移从而改变camera的rotation实现。键盘控制部分原理也是大同小异，通过给键盘增加监听，判断用户按键的keycode，使用switch语句来对相机进行各个方向的移动，代码如下。

**switch** ( event.**keyCode** ) {  
 **case** 87: */\*W\*/* **this**.**moveState**.**forward** = 0; **break**;  
 **case** 83: */\*S\*/* **this**.**moveState**.**back** = 0; **break**;  
  
 **case** 65: */\*A\*/* **this**.**moveState**.**left** = 0; **break**;  
 **case** 68: */\*D\*/* **this**.**moveState**.**right** = 0; **break**;  
  
 **case** 82: */\*R\*/* **this**.**moveState**.**up** = 0; **break**;  
 **case** 70: */\*F\*/* **this**.**moveState**.**down** = 0; **break**;  
}

因为鼠标监听与键盘监听都是消耗非常大的，会较大程度的影响场景渲染的频率，就会造成FPS的变化幅度较大，也就是渲染一帧所花的时间时长时短，而这时如果相机的移动速度和旋转速度不变的话，相机每一帧移动的距离就因为FPS而产生很大的差异，造成相机的移动的不均匀，甚至卡顿。为了使自由漫游时的体验更为平滑，定义一个系统时钟，在调用自由漫游交互模式生成函数的时候就启动，代码如下

**var** clock = **new THREE**.Clock();

***clock***.start();

然后定义一个变量delta,值为clock.getDelta(), getDelta()函数返回调用该函数时距离上一次调用该函数时的时间长度，如果是第一次调用该函数，则返回此时距离开始计时时的时间长度，也就是开始调用自由漫游交互模式生成函数到现在的时间长度。将这个delta值作为参数传给控制相机移动的函数，用delta乘以默认的移动和旋转速度，从而相机每一帧的移动距离和角度现在不仅取决于先前设定好的速度，还与每一帧的间隔时间成正相关。那么在animate()函数每一帧不断调用render()的时候，delta值就等于每一帧渲染所需的时间，如果一帧渲染的较慢，那么delta值就较大，在这一帧里相机的移动距离和角度也越大，反之如果这一帧很快就渲染完了，delta值就很小，用其乘以速度，那么相机的移动距离和旋转角度也就较小了。这样就可以实现无论FPS如何变化，每一帧的渲染时间如何浮动，相机的运行位移和角度都是稳定的。代码如下：

**function** *render*() {  
 **var** delta = ***clock***.getDelta();

//调用改变相机位移与角度的函数update()，将delta值传入  
 ***FlyControl***.update( delta );

//刷新场景  
 ***renderer***.*render*(***scene***,***camera***);  
}

//flyControl的update函数

**this**.update = **function**( delta ) {  
 **var** moveMult = delta \* **this**.**movementSpeed**;  
 **var** rotMult = delta \* **this**.**rollSpeed**;

……

}

3.5 可视化三维场景编辑

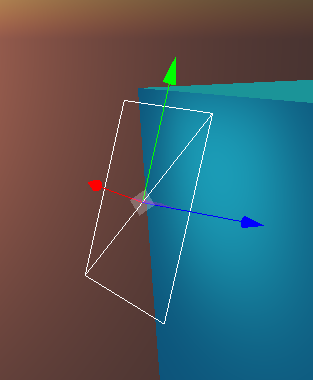
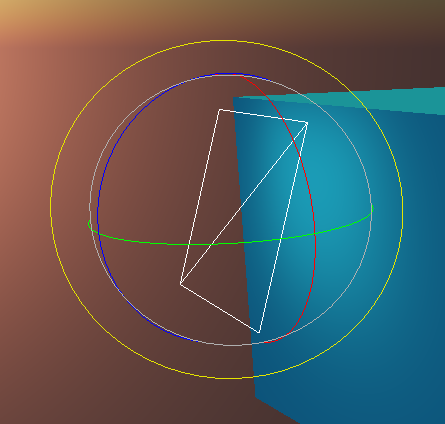
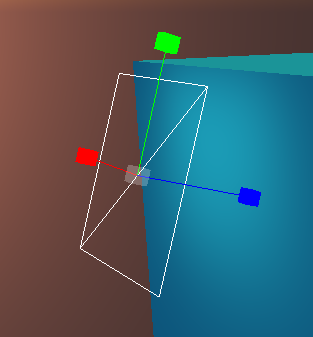
本平台最大的特点与优势就是可以使用户图形化可视的编辑web3D场景，几乎每一种场景元素都可以通过改变少量参数来达到不同的展示效果，这些场景元素的可视化操作的实现方式都是大同小异的，本节将选出具有代表性的点光源场景元素对可视化编辑功能加以说明。

无论什么三维元素，在场景中都有自己的transform属性，即其position，rotation和scale，这三个属性不受限于对象的类型，因此我们对可能需要改变这三个属性的三维元素增加鼠标拖拽改变效果。以点光源为例，新建Three.js给出的TranformControls控件，将其参数设置为该点光源，并在场景中添加该控件即可，代码如下：

**pointLightControl**= **new THREE**.TransformControls(***camera***,***renderer***.**domElement**);  
**pointLightControl**.attach( pointlight );

**scene**.add( **lightControls**.**pointLightControl** );

TranformControls控件有三个部分，即上文提到的position control，rotation control和scale control，可以通过其setMode()方法来指明需要何种control，下文将通过GUI的按钮来在这三种control中自由切换。效果如下：

平移操作 旋转操作 缩放操作

首先在定义一个三维元素的时候需要将其需要改变的参数封装在接口内，方便外部函数调用并对其进行操作，如点光源的构造函数为：

pointlight = **new THREE**.PointLight( 0xff0040, 6, 100,1 );

其中四个参数的含义依次为：光源颜色，光源强度，光照最远距离和光照在这个距离内的衰减值。因为光源自身是不可见的，因此我们增设一个和光源位置一样的球体用以可视化光源，就像是加了一个用户可见的灯泡一样，这样可以方便用户对光源各类参数的可视化调整。代码如下：

//创建一个半径为0.5，经度分割为16份，纬度分割为8份的多面体网格，近似球体

**var** sphere = **new THREE**.SphereGeometry( 0.5, 16, 8);  
//以上述多面体为网格，光源颜色为材质创建光源球体

**var** pointlightHelper = **new THREE**.**Mesh**( sphere,

**new THREE**.**MeshBasicMaterial**( {

**color**: pointlight.**color**.getHex()} )

)；

//将光源球体作为一个子元素加到光源中，这样光源球就可以始终和光源的位置信息保持//一致

pointlight.add( pointlightHelper);

生成光源后，我们将所有需要对该光源所做的操作创建一个类，代码如下：

**var** *pointLightData* = **function**(){  
 **this**.**color** = pointlight.**color**.getHex();  
 **this**.**intensity** = pointlight.**intensity**;  
 **this**.**move** = **true**;  
 **this**.position = **function** () {

**pointLightControl**.setMode(**"translate"**)};  
 **this**.rotation = **function** ()

**pointLightControl**.setMode(**"rotate"**)};  
 **this**.scale = **function** () {

**pointLightControl**.setMode(**"scale"**)};  
 **this**.delete = **function** () {  
 removePointLight();

};  
}

然后生成该操作集的对象

**var** param = **new** *pointLightData*();

现在就可以新建一个dat.gui库的GUI对象

**PointLightGui** = **new dat**.**GUI**();

向GUI控件中添加操作集对象的变量，添加变量类型为颜色、整型、bool型、函数型的GUI子菜单，在该子菜单的值发生改变时，就对模型的该参数进行调整，从而达到通过GUI控制模型参数的效果。

//十六进制，控制光源的颜色

**PointLightGui**.addColor( param, **'color'** ).onChange(

**function** ( val ) {pointlight.**color**.setHex( val );});

//整型，控制光源的强度  
**PointLightGui**.add( param, **'intensity'**, 0,50 ).onChange(

**function** ( val ) {pointlight.**intensity** = val;} );

//bool型，控制光源是否自动移动  
**PointLightGui**.add( param, **'move'**).onChange(

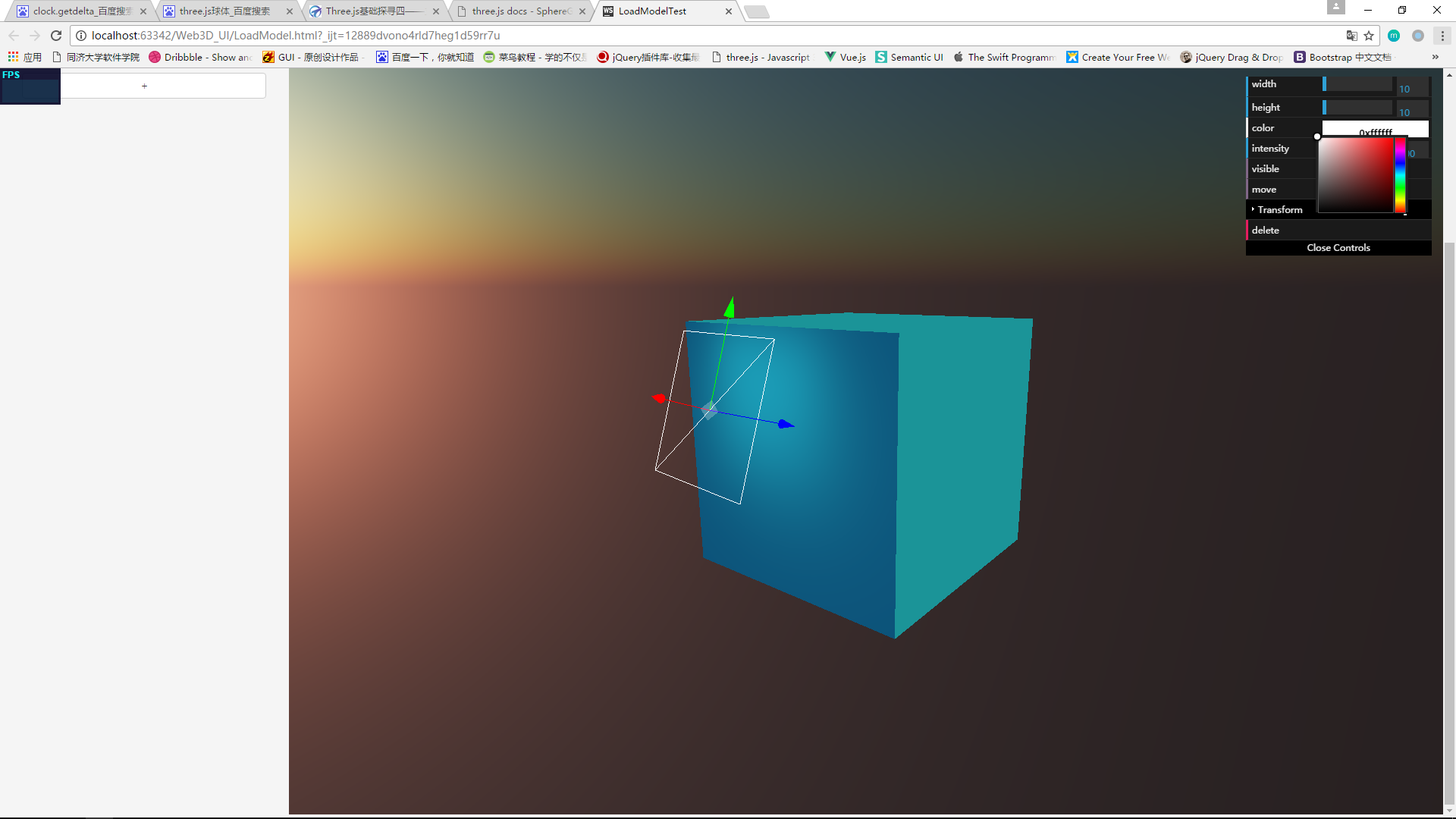
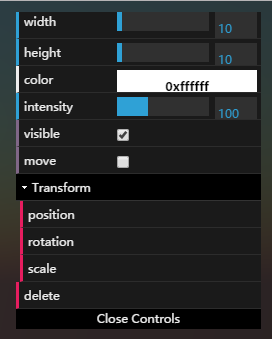
**function** ( val ) {***movePointLight*** = val;});

//在GUI控件中添加名为Transform的文件夹，其有三个子文件，分别为**'position','rotation'** //和**'scale'.**这三个选项分别对应操作集对象中定义的三个函数。即TranformControls里对应的三//种操作模式  
**var** transform = **PointLightGui**.addFolder(**'Transform'**);  
 transform.add(param, **'position'**);  
 transform.add(param, **'rotation'**);  
 transform.add(param, **'scale'**);

//在GUI控件中名为delete的button，该button调用removePointLight()函数。

**PointLightGui**.add(param,**'delete'**);

GUI效果如下：

另为了使光源更加多样，为光源增设了自动移动属性，代码如下：

**function** *update*() {  
 **var** time = Date.now() \* 0.0005;   
 pointlight.position.**x** = **Math**.sin( time \* 0.7 ) \* 20;  
 pointlight.position.**y** = **Math**.cos( time \* 0.5 ) \* 20;  
 pointlight.position.**z** = **Math**.cos( time \* 0.3 ) \* 20;  
}

3.6 可拖拽小组件的实现

可拖拽小组件主要是通过Jquery库和Jquery UI实现的，这5种不同布局的可拖拽小组件都是共用同一段HTML代码，只是通过使用Jquery改变该Html的class动态的修改组件的样式。这样不仅可以使网站的内容丰富的同时代码精简，而且可以保证一个场景之中永远都只存在一个可拖拽小组件，

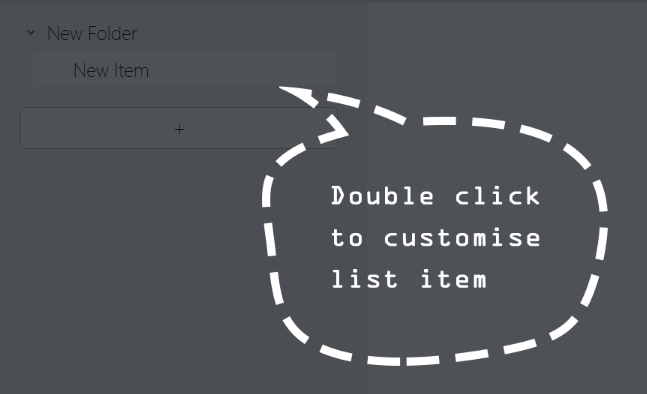
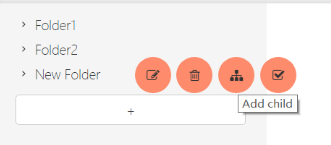
可拖拽效果源自Jquery UI的Draggable()功能。为该可拖拽组件增加draggable，droppable等特性，设置相关参数，如拖拽容器，拖拽显示光标等。并在拖拽结束时监听光标位置，如果拖拽垂直位置大于屏幕的一半，即拖拽的更靠近屏幕底部时，组件打开不再默认地向下弹出，而是变为向上展开，这样就可以允许用户把可拖拽组件放在屏幕底部了。如图，同时监听h5的dblclick事件，如果鼠标在可拖拽组件上触发双击事件，则打开关闭组件。

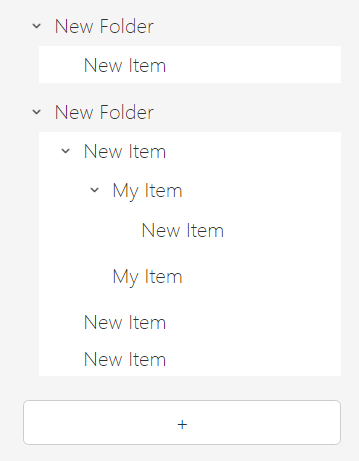
   

可拖拽组件 默认组件 向下展开 向上展开

3.7 可编辑侧边栏实现

可编辑侧边栏在Jquery提供的树状文件结构基础上改进而成。



3.8 不同模板间的独立性与轻量化网站构建

为了监控网站的性能，观察网页渲染的帧数，本文调用了stat.js提供javascript性能监控器，如图所示，如果将stat的begin()与end()函数分别放在上文所述animate内渲染函数的前后，就可以得到渲染所花的时间，从而观察网页性能。代码如下：

**function** *animate*() {  
 requestAnimationFrame(*animate*);  
 ***stats***.begin();  
 *render*();  
 ***stats***.end();  
}

C:\Users\admin\AppData\Local\Temp\1495691214(1).png

为了保证用户在进行了一系列添加、切换、删除等操作后，场景中没有冗余元素，仍能流畅正常的渲染运行，本系统给不同三维元素的切换时设定了不同的清除方式。部分是在切换后仍保留在场景中而不显示元素及其控制块，部分在切换后保留元素而隐藏控制块，部分因为开销较大或用户需要而直接删除。下面就以这三种方式分别举例说明。

1) 保留而不显示的如场景雾，这类场景元素的新建与生成比渲染与保留的开销要大得多，那么如果用户不想再显示场景雾，最科学的做法就是将雾的浓度变为零，然后隐藏雾的控制块。这样就相当于是场景中其实有雾，不过是浓度为0而已，而renderer对雾的渲染开销非常小，几乎可以忽略不计。如果用户稍后想再次给场景中加雾化效果，只需再次显示控制块，调整雾的浓度即可。比再次调用雾的生成函数带来的渲染开销要小得多。这种控制方式还用于纯色背景，网格这两种场景元素。

2) 用于用户想要保存这种元素设定，继而操作其他场景元素的情况，比如用户在设置好了skyshader背景场景，需要接着向场景中添加灯光。这时我们暂时不需要对skyshader的相关参数做操作，那么只需将skyshader的控制块隐藏起来，这样就算用户稍后想返回继续对skyshader做操作，系统只需再次显示保存了用户操作的控制块，用户也无需再从头设置参数了。这种方式使用在5类场景元素相互切换，以及5类中的子元素相互切换中。

3) 直接删除该场景元素及其控制块，如二维图片背景在上文中我们提过是通过再新建一个orbitcamera与贴有此二维图片的平面实现的，为了显示这个二维图片，渲染器需要渲染两个相机视角，大大增加了渲染压力。如果此时用户将背景切换为天空盒，那么我们就可以毫不犹豫的将OrbitCamera与平面对象从场景中删除掉，从而大幅度降低渲染器的渲染压力。等到用于想要再次设置二维图片背景时系统再重新生成相关元素即可。实现方式是找到所有与该元素相关的对象，调用scene.remove()函数，再刷新三维场景即可。

由于三维场景编辑选项多，逻辑复杂，而不同三维元素在场景中的开销也大小不一，将不同的三维元素区分对待是非常科学的，也是使页面流畅运行的必要手段

3.8.1 dat.gui控制器的显示与隐藏

上文中多次提到显示隐藏控制块，这里的控制块指的是使用dat.gui库生成的GUI对象，而dat.gui库只提供GUI菜单的展开与折叠，不提供显示与隐藏，因此扩展dat.gui接口。代码如下：  
dat.**GUI**.**prototype**.hide = **function**() {  
 **this**.**domElement**.setAttribute(**"hidden"**, **true**);  
};  
dat.**GUI**.**prototype**.show = **function**() {  
 **this**.**domElement**.removeAttribute(**"hidden"**);  
};

这样在之后的代码中只需调用.hide()与.show()函数即可隐藏于显示GUI列表。

# 第五章 平台集成开发与测试

# 参考文献

1. Web3D官方网站[DB/OL]. (2017)[2017-03]. [www.web3d.org](http://www.web3d.org)
2. Blender官方网站[DB/OL]. (2017)[2017-03]. [www.blender.org](http://www.blender.org)
3. Three.js官方网站[DB/OL]. (2017)[2017-03]. [threejs.org](https://threejs.org)
4. XML3D官方网站[DB/OL]. (2017)[2017-03] <http://xml3d.org>
5. X3DOM官方网站[DB/OL]. (2017)[2017-03] <https://www.x3dom.org>
6. BootStrap官方文档[DB/OL].<http://v3.bootcss.com>
7. Json文格式官方文档<http://www.json.org/>
8. Dirksen J. Three. js essentials[M]. Packt Publishing Ltd, 2014.
9. Rego N, Koes D. 3Dmol. js: Molecular visualization with WebGL[J]. Bioinformatics, 2015,

31(8): 1322-1324.

1. Mayhew D J. Principles and guidelines in software user interface design[M]. Prentice-Hall, Inc.,2001.
2. Cooper A. About face: The essentials of user interface design[M]. John Wiley & Sons, Inc., 2005.
3. Fielding R T, Taylor R N. Principled design of the modern Web architecture[J]. ACM Transactions on Internet Technology (TOIT), 2002, 2(2): 115-150.
4. 基于WebGL技术的网络三维可视化研究与实现 刘爱华 韩勇 张小垒 陈戈 中国海洋大学信息科学与工程学院海洋信息技术教育部工程研究中心 2016
5. 三维模型文件中的OBJ格式在OpenGL中的输入与处理 王金峰 姚国清中国地质大学

(北京) 2011

1. 基于Bootstrap框架的响应式网页设计与实现 舒后 熊一帆 葛雪娇 北京印刷学院 学报 2016

# 谢 辞