

TONGJI UNIVERSITY

本科毕业设计（论文）

|  |  |
| --- | --- |
| 课题名称 | 基于Web3D的智能可编辑  UI模板库 |
| 学 院(系) | 软件学院 |
| 专 业 | 软件工程 |
| 学 号 | 1362894 |
| 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 libriary based on Web3D**

**ABSTRACT**

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

目 录

1.引言

1.1 系统的背景与意义

1.2 研究背景及相关技术

. 1.2.1 WebGL

1.2.2 Three.js

1.2.3 Dat.gui

1.2.4 Bootstrap

1.3 平台开发可行性

1.4 本文工作及创新性

2．系统总体架构

3．关键技术

3.1场景光照生成

3.1.1 场景点光实现

3.1.2 场景面光实现

3.1.3 场景聚光灯实现

3.2 场景背景生成

3.2.1 纯色背景实现

3.2.2 Shader背景实现

3.2.3 二维背景图片实现

3.2.4 天空盒背景实现

3.3 交互方式实现

3.3.1 固定拖拽交互模式实现

3.3.2 自由拖拽交互模式实现

3.3.3 第一人称控制交互模式实现

3.3.4 自由漫游交互模式实现

3.4 场景渲染

3.4.1 场景雾化效果实现

3.4.2 场景网格系统实现

3.4.3 地面的实现

3.5 不同格式3d模型的加载

3.5.1 OBJ格式模型加载

3.5.2 STL格式模型加载

3.5.3 VTK格式模型加载

……

3.5 可拖拽小组件的实现

3.6 可编辑侧边栏实现

3.7 轻量化网站构建

\*3.5 场景小地图的拖拽生成（未做）

\*3.6 Web VR体验（未做）

\*3.8网站的登录与上传（未做）

\*3.9自定义界面的保存与再加载（未做）

4.平台集成开发与测试

4.1 首钢

4.2 上汽

[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)

# 引 言

1.1 系统的背景与意义

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。

1.2 研究背景及相关技术

. 1.2.1 WebGL

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 是基于 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 [4]。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。 2.1 WebGL 的优势 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。 2.2 WebGL 免插件的实现 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 min 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，进而导致客户量的减少。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 1 所示）。 图 1 WebGL 免插件图形渲染 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。

1.2.2 Three.js

Three.js是使用JavaScript编写的WebGL第三方库，该库提供了相对健全的3D场景与模型的显示与渲染功能，它是运行在浏览器中的 Web3D 引擎，通过将WebGL原生API的细节抽象化，内置图形编程的常用功能函数，从而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。而且提供了WebGL本身不具有的picking功能，可以方便的为应用添加用户交互功能。Three.js是开源代码，完整的代码托管在github上面[[9,10,11]](http://en.wikipedia.org/wiki/WebGL#cite_note-WebGLWebsite-4) 。

1.2.3 Bootstrap

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 Git Hub 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; ②丰富的预定义样式表; ③一组 基于 jQuery 的 JavaScript 插件集; ④一个非常灵活 的响应式( Ｒesponsive) 栅格系统，并且崇尚移动先 行( Mobil First) 的思想［2－4］。 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。 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

1.3 平台开发可行性

1.3.1 Three.js框架平台分析

相对于其它现有的WebGL框架，Three.js是最为成熟和健全的，支持大部分主流Web3D引擎的功能，且拥有自己的一套编辑器框架，不论是学习还是研究都能较为方便的进行。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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**

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。

1.3.2 Dat.gui

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

1.4 本文工作及创新性

1.4.1 本文工作

本文主要的工作就是创建一个基于Three.js的3D网页的UI模板，使得用户可以通过简单地长传，点击，拖拽等操作，就可以自定义web3D场景。具体工作就是整合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。

1.4.2 本文创新性

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。

第二章 系统总体架构

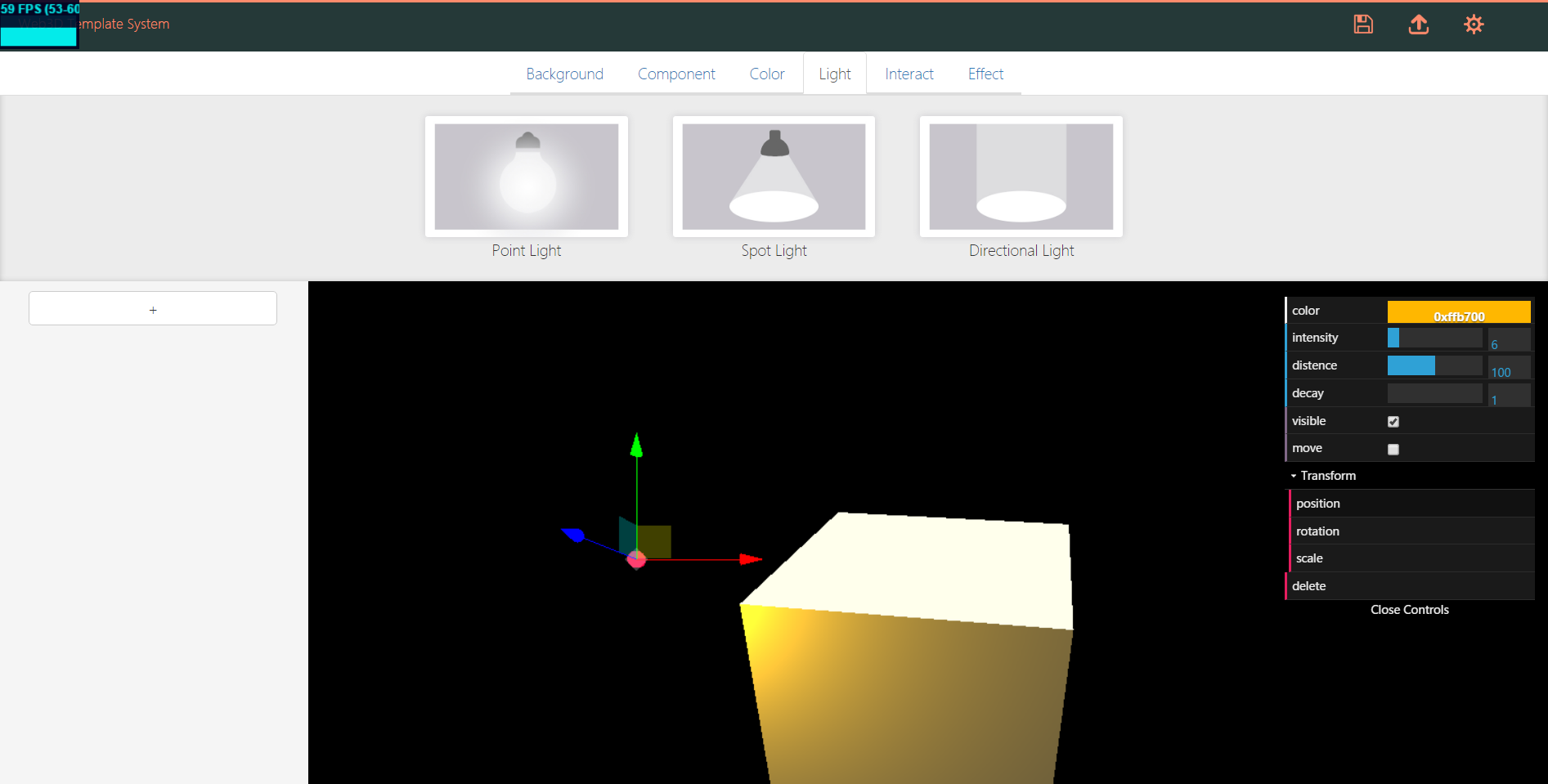
本项目是一个单页应用系统，提供两种基本模板，全屏模板和带侧边栏的模板。用户选好模板之后就可以在模板的基础上编辑自己的三维场景。本文提供的可自定义三维场景元素有：背景、灯光、交互方式、场景渲染和可拖拽小组件五种。其中背景包括纯色背景、Shader背景、二维静态背景和天空盒背景四种选项，灯光包括点光、面光和聚光灯三种，交互方式包括固定拖拽交互，全局拖拽交互，第一人称视角交互和自由漫游视角交互四种，场景渲染分为雾化效果、网格元素和地面三种，可拖拽小组件分为样式不同的5种。用户可以在这五大类自定元素中自由选择需要的三维元素，这五类自定元素可以互不干扰的独立存在，大大增加了场景的丰富性和多样性。

项目主要场景与功能都在同一个html中，五个自定义模块分别有对应的js文件。

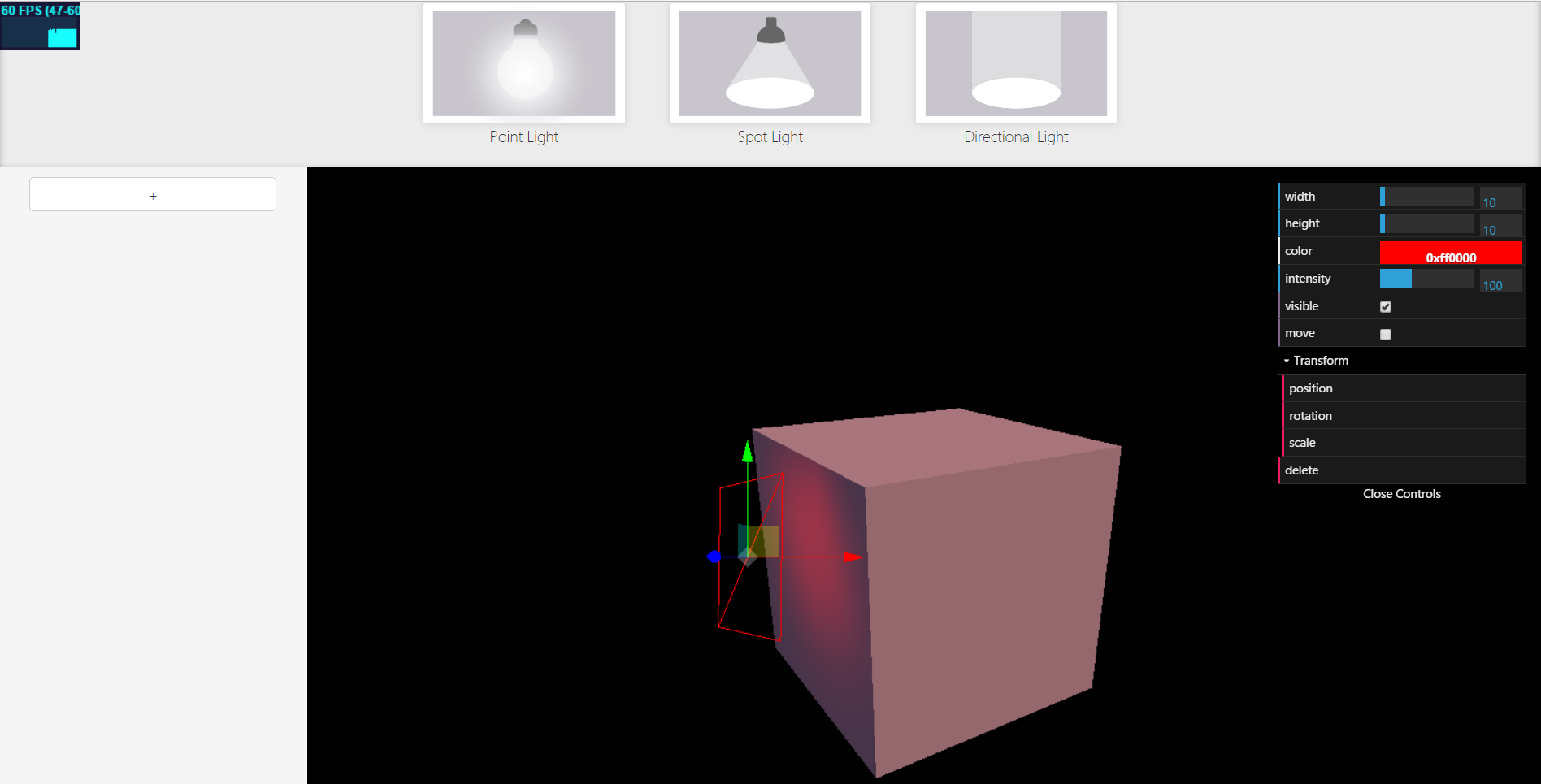
# 第三章 关键技术

3.1场景光照生成

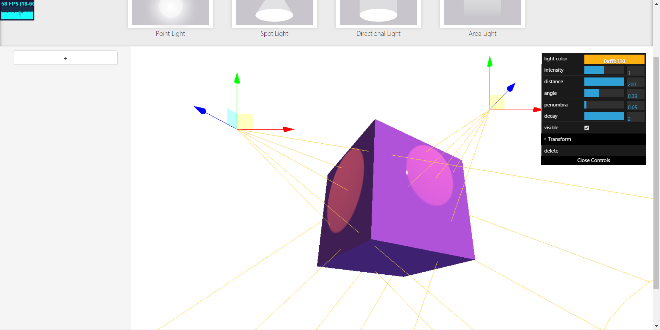
3.1.1 场景点光实现



3.1.2 场景面光实现



3.1.3 场景聚光灯实现



3.2 场景背景生成

3.2.1 Shader背景实现

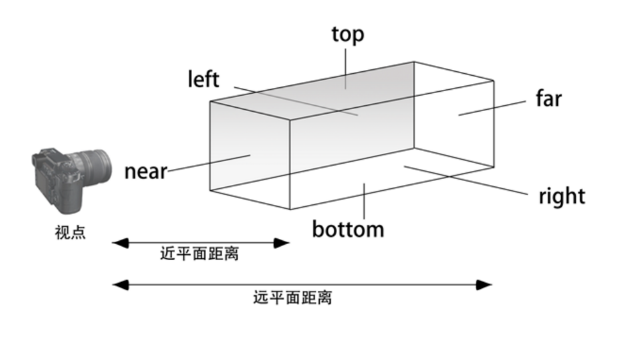
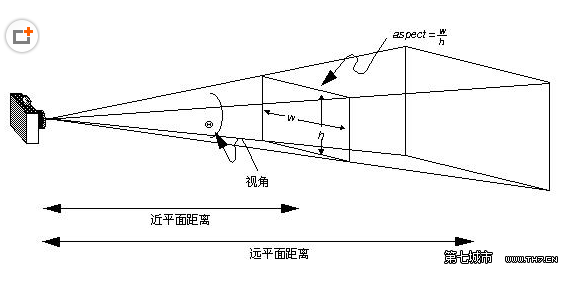
什么是Shader呢？Shader就是一种能把线框图转变为漂亮的 3D 形象的系统。当设置了所有的属性与参数，如高光属性、纹理与凹凸度等时， Shader系统就会工作。它将计算出我们给出的表面外观。Shader以许多类型与风格出现，而且许多种相似的Shader有不同的名字，这都容易使人混淆不清。一个3D程序有不同种的Shader，因为在真实世界里有数以百万计的不同表面，它们对光都有不同的反应。Shader 的一个任务就是去模拟对象与光之间的相互作用，并用最精确的方式把它们展现在观众面前。 从技术的角度看，Shader的作用是指出一个表面上的每一个点是怎样表现的，它对光会做出什么样的反应。有的 Shader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。

而我用到的shader是…………………………

* + 1. 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'** );

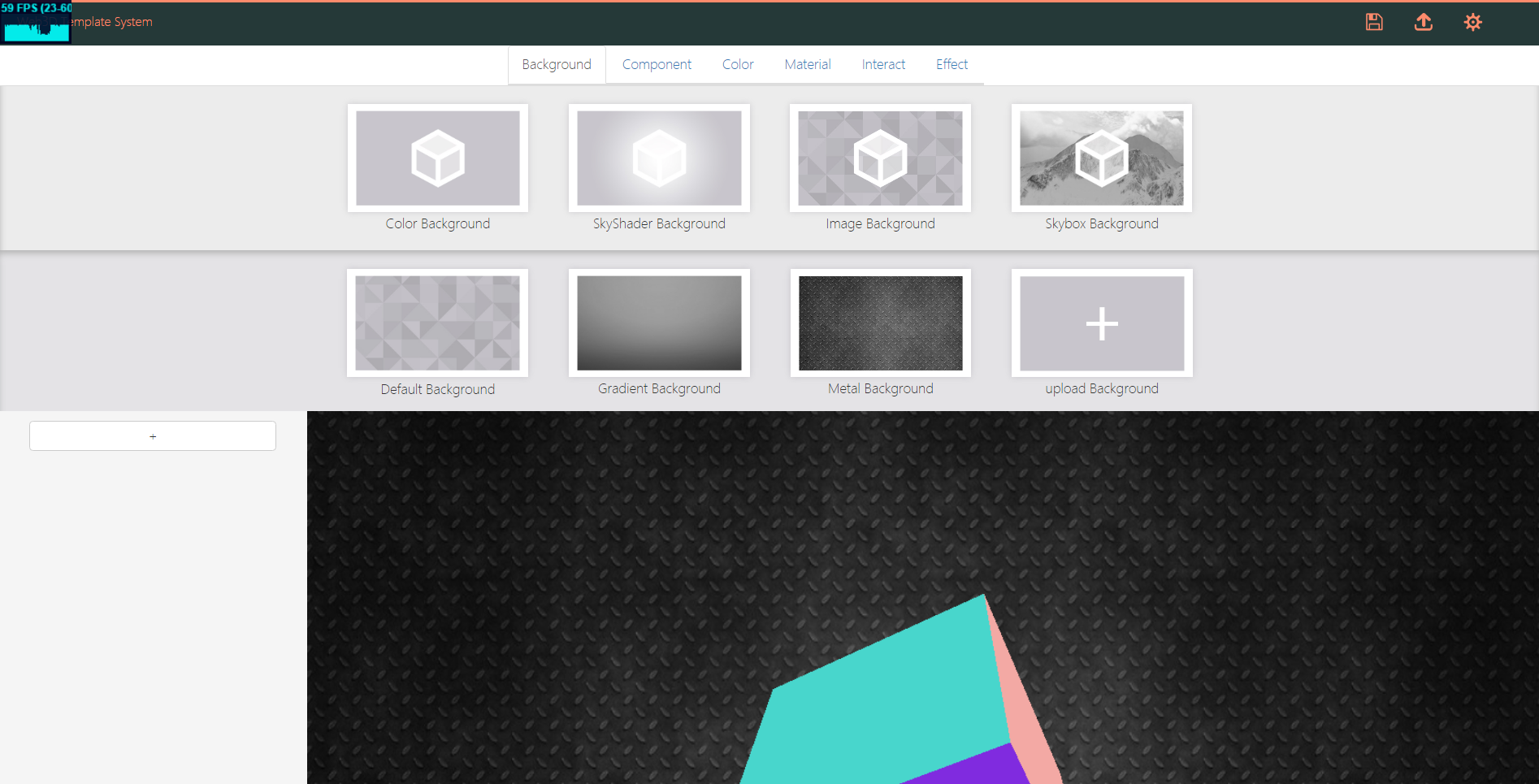
//贴有背景图片的平面**var** backgroundMesh = **new THREE**.**Mesh**(  
 **new THREE**.PlaneGeometry(2, 2, 0),  
 **new THREE**.MeshBasicMaterial({  
 **map**: texture1  
 }));  
//贴有背景图片的平面  
***backgroundScene*** = **new THREE**.Scene();//正交投影相机

//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*** );

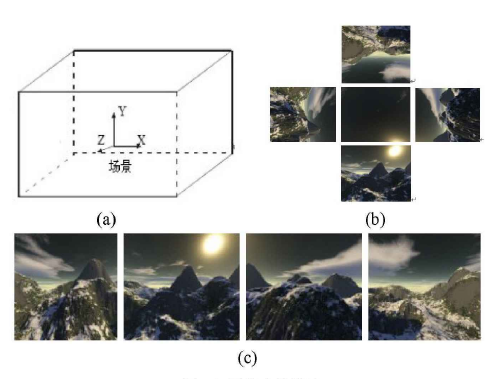
//将贴有背景图片的平面添加到场景中  
***backgroundScene*** .add(backgroundMesh );

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

3.3.1 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;  
}

另外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

3.3.2 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。

3.3.3 第一人称控制交互模式实现

3.3.4 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**;

……

}

3.4 不同格式3d模型的加载

3.4.1 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。 这里的 OBJ[2]文件不是我们常见的程序编译生产的目标文件，而是一种定义物体几何模型的 3D 文件，最初是 Wavefront 的标准 3D 模型文件，后来被广泛应用到各种应用软件中。 OBJ 文件可以是二进制文件，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。 3.1 OBJ 文件格式 Maya 的开放系统输出数据文件类型有 MEL，DXF，Explore，Geo，OBJ， IGES， Aliaswire，Aiff，Image，RIB，MOV (Aaciimotion)， VRML2。 其中 OBJ 模型文件为文本类型，结构比较简单，它保存的是一些多边形的信息，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就可以构造出三维物体。 下面简要的介绍一个 obj 文件格式[3]。 OBJ 文件的基本结构[4]。 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，注释行以符号“#”为开头，空格和空行可以随意加到文件中以增加文件的可读性。 有字的行都由一两个标 记字母也就是关键字(Keyword)开头，关键字可以说明这一行是什么样的数据。 多行可以逻辑地连接在一起表示一行，方法是在每一 行最后添加一个连接符(\)。 注意连接符(\)后面不能出现空格或 Tab 格，否则将导致文件出错。 下列关键字可以在 OBJ 文件使用。 在这个列表中, 关键字根据数据类型排列，每个关键字有一段简短描述。 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。 Vp：参数空格顶点 (Parameter space vertices)。 自由形态曲线(Free-form curve)/表面属性(surface attributes): Deg：度(Degree)。 Bmat：基础矩阵(Basis matrix)。 Step：步尺寸(Step size)。 cstype 曲线或表面类型 (Curve or surface type)： 元素(Elements): P：点(Point) L：线(Line) F：面(Face) Curv：曲线(Curve) curv2 2D：曲线(2D curve) surf：表面(Surface) 自由形态曲线(Free-form curve)/表面主体陈述(surface body statements): Parm：参数值(Parameter values ) Trim：外部修剪循环(Outer trimming loop) Hole：内部整修循环(Inner trimming loop) Scrv：特殊曲线(Special curve) sp：特殊的点(Special point) end：结束陈述(End statement) 自由形态表面之间的连接(Connectivity between free-form surfaces): Con：连接 (Connect) 成组(Grouping): G：组名称(Group name) S：光滑组(Smoothing group) Mg：合并组(Merging group) O： 对象名称(Object name) 显示(Display)/渲染属性(render attributes): Bevel：导角插值(Bevel interpolation) c\_interp：颜色插值(Color interpolation) d\_interp：溶解插值(Dissolve interpolation) lod：细节层次(Level of detail) usemtl：材质名称(Material name) mtllib：材质库(Material library) shadow\_obj：投射阴影(Shadow casting) trace\_obj：光线跟踪(Ray tracing) ctech：曲线近似技术(Curve approximation technique) stech：表面近似技术 (Surface approximation technique) 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[5]。 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。 "usemtl"指定了材质之后，以后的面都是使用这一材质，直到遇到下一个"usemtl"来指定新的材质。

3.4.2 STL格式模型加载

STL 模型的基本知识

STL 为 stereolithography 的缩写， 含义为光固化立体造型 术。 1988 年 3D SYSTEMS 公司将其制定为接口协议。 STL 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。 STL 格式模型在 3DS Max 中的展示效果如图 1 所示。 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。 1.1 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 … //省略了其他数据信息 endsolid filename stl //定义结束 1.2 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。 之 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 看 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 个 字节。

2 通过 Three.js 引擎加载 STL 模型的具体步骤 了解了 STL 的基本情况之后， 下面就可以简单介绍一下 通过 Three.js 引擎加载 STL 模型的开发思路。 Three.js 为用 于 3D 场景渲染的 JavaScript 开源库， 其中对底层 API 进行封装， 在保证渲染效果的情况下， 使开发过 程 尽 量 简 化。 并 且 由 于其具有极强兼容性， 支持加载多种格式的文件， 所以以Three.js 引擎作为工具对 STL 模型进行加载和渲染。 2.1 引入 Three.js 库文件 Three.js 库中包含许多个工具文件， 每一个文件中都封装 着完成特定功能的方法。 three.js 文件是 Three.js 引擎中使用频 率最高的工具文件， 其中封装了调 整 场 景、 摄像机和材质的 相关方法， 可以方便快捷地对渲染场景进行初始化。 Three.js 引擎发展至今已经有了多种加载器 ， 可以 支 持 JSON、 OBJ、 STL、 CTM 等 多 种 格 式 。 使 用 Three.js 引 擎 对 STL 模型进行读取之前， 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。 进行实际的开发之前， 首先将 Three.js 库文件下载到开发 设备中。 使用时， 开发人员需要开发一个简单的 HTML 格式 的文件， 并将 three.js 和 STLLoader.js 作为外部 JavaScript 文件 导入程序中。 具体标签如下： 2.2 初始化渲染场景 加载 STL 模型之前， 需要使用 Three.js 对整个渲染场景进 行初始化， 主要包括新建场景、 设置摄像机位置和焦点、 光 源位置等相关操作。 具体的代码部分如下所示：

var scene = new THREE.Scene();//新建场景

var webGLRenderer = new THREE. WebGLRenderer();//新建渲染器

webGLRenderer.setClearColor

(new THREE.Color(0x000, 1.0));

webGLRenderer.setSize(window. innerWidth, window.innerHeight);

camera.position.x = 150;//摄像机位置

camera.position.y = 150; camera.position.z = 150;

camera.lookAt(new THREE.Vector3(0, 40, 0));

//设置光源位置

spotLight.position.set(150, 150, 150);

scene.add(spotLight);

//添加光源到场景

2.3 加载 STL 文件 渲染场景初始化工作完成之后， 接下来就可以对 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);

scene.add(group);//添加模型进场景 }); render();//进行绘制的方法

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， 使运行效果更加绚丽。 通过 Three.js 引擎 加载 STL 模型的运行效果如图 2 所示。 从图 2 中可 以 看 出 程 序可以很好地表现出模型的细节部分， 并且在光和材质的作 用下， 画面体现出很强的立体感和真实感， 给人以美的享受。

3.4.3 VTK格式模型加载

……

3.5 可视化三维场景编辑

3.6 可拖拽小组件的实现

3.7 可编辑侧边栏实现

3.8 不同模板间的独立性与轻量化网站构建

3.7.1 dat.gui控制器的显示与隐藏

3.7.2模板的删除与隐藏

# 第4章 平台集成开发与测试

# 参考文献

1. Web3D官方网站[DB/OL]. (2016)[2016-03]. [www.web3d.org](http://www.web3d.org)
2. Blender官方网站[DB/OL]. (2016)[2016-03]. [www.blender.org](http://www.blender.org)
3. Three.js官方网站[DB/OL]. (2016)[2016-03]. threejs.org
4. Vue.js开源网站[DB/OL]. (2016)[2016-03]. cn.vuejs.org
5. BootStrap官方文档[DB/OL].v3.bootcss.com

**[6]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

## [7]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

## 通过Three.js引擎加载STL模型

苏亚光王冬刘佳

华北理工大学 10.16184/j.cnki.comprg.2016.10.002

## 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

## 虚拟场景实时生成技术研究

孙文静

西安电子科技大学

# 谢 辞

正文内容