计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2018,27(11):241−246 [doi: 10.15888/j.cnki.csa.006628]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于 Three.js 的 3D 磁盘阵列设计①

高齐琦, 江　婷, 田世隆, 林嘉琦

(中电海康集团有限公司, 杭州 312000)
通讯作者: 高齐琦, E-mail: gaoqiqi@hikdata.cn

摘　要: WebGL (Web Graphical Library) 让浏览器在无需安装插件的情况下即可渲染出 3D 图形, 而封装了 WebGL 低级别 API 的 three.js 更是为 3D 图形的高效创建提供了可能. 针对传统方法的磁盘阵列可视化不够直观以及交互功能单一的问题, 本文提出了基于 three.js 的磁盘阵列三维 (3D) 可视化的框架, 从仿真建模和交互两方面阐述了 3D 磁盘阵列的设计思路, 并给出了相应的实现过程. 相对于传统方法, 提出的 3D 可视化框架能够创建真实的三维场景, 给用户带来流畅丰富的视觉以及交互体验.

关键词: 三维可视化; 仿真建模; 交互设计; 用户体验

引用格式: 高齐琦,江婷,田世隆,林嘉琦.基于 Three.js 的 3D 磁盘阵列设计.计算机系统应用,2018,27(11):241–246. http://www.c-s-a.org.cn/1003-3254/6628.html

Design of 3D Raid Based on Three.js

GAO Qi-Qi, JIANG Ting, TIAN Shi-Long, LIN Jia-Qi

(CETHIK Group Co. Ltd., Hangzhou 312000, China)

Abstract: By applying the Web Graphical Library (WebGL), a browser can render Three Dimensional (3D) images without installing any additive plugins. The three.js, a JavaScript library that encapsulates the low-level WebGL API, can even construct a 3D graphic more efficiently. Since existed visualization approaches for Redundant Array of Independent Disk (RAID) are non-intuitive and only with simple interactive functions, a new 3D RAID visualization framework is proposed in this work based on the three.js library. The model simulation and the interactive functions for visualizing a 3D RAID are explained step by step and implemented respectively. Compared to the traditional methods, the proposed 3D visualization framework constructs a more realizable 3D scene, thus provides the users better experiences both at visualization and interaction.

Key words: 3D visualization; simulation modeling; interaction design; user experience

1　引言

随着大数据时代的到来, 存储行业也面临着新的机遇和挑战. 高效、真实且实时的存储设备可视化, 不仅有利于此类设备的设计, 也能为后续的运营和维护提供帮助. 传统磁盘阵列 (Redundant Array of Independent Disk; RAID) 的可视化设计, 只是通过表格罗列信息或是前视图、后视图的方式来实现, 无法在空间位置上展示盘阵的细节, 操作难以上手, 且管理界面单调、互动性差, 影响信息传递和操作效率, 降低决策和响应速度.

WebGL (Web Graphical Library) 的诞生使得浏览器不需要任何插件即可呈现丰富的 3D 图形, 并且提供更友好的交互功能[1], 例如实时动画播放[2]、机房可视化[3]等工作. 然而, 在磁盘阵列的 3D 可视化问题上, 目前国内极少涉及.

① 收稿时间: 2018-03-26; 修改时间: 2018-04-28; 采用时间: 2018-05-08; csa 在线出版时间: 2018-10-24

针对以上问题, 本文提出基于 WebGL 第三方库 three.js 3D 磁盘阵列可视化的设计和实现框架, 同时针对 three.js API 对复杂形态可视化的不足, 还提出了预绘制模型, 并以 OBJ 形式导入的策略. 最后, 在可视化 3D 磁盘基础上, 增加了丰富且实用的交互功能, 为磁盘阵列可视化表达提供了一种新的方法和途径.

本文按如下内容展开, 在第 2 节将简要介绍 WebGL 与 three.js 库; 第 3 节介绍系统框架和页面的主要功能; 在第 4 节提出基于 three.js 的 3D 磁盘阵列可视化设计, 讨论简单和复杂组件的模型建立和导入方法; 在第 5 节介绍客户端的交互功能; 最后进行总结并对后续工作做出展望.

2 关键技术介绍

2.1 WebGL 与 Three.js 库

WebGL 是一种跨平台、免费的 3D 绘图协议, 是 HTML5 规范的组成之一, 通过 HTML5 Canvas 元素对外暴露 DOM (Document Object Model) 编程接口[2]. 基于 OpenGL ES 2.0 标准, WebGL 通过增加对 OpenGL ES 2.0 的 JavaScript 绑定, 为 HTML5 Canvas 提供基于硬件的 3D 加速渲染, 使得浏览器无需第三方插件, 就可以借助系统显卡在浏览器里呈现高性能 3D 图形.

虽然 WebGL API (Application Programming Interface) 的出现使得前端开发者可以直接在页面中绘制 3D 图形, 但是 WebGL 提供的是低级别的、光栅化的 API, 直接编程会面临复杂且易出错的问题. 为了构建一个高等级的、对前端开发者更加友好的 WebGL 开发环境, 许多开源 JavaScript 库被创造出来, 其中 three.js 的应用最广泛, 它不仅提供了简单易懂的 JavaScript API, 并且集轻量级、开源免费等优秀品质于一身[4,5]. 因此, 本文选用 three.js 作为基本的开发工具库.

2.2 Vue.js

前端页面基于 Vue.js 构建 MVVM (Model-View-ViewModel) 模式的渐进式框架, 如图 1 所示, 即采用自底向上增量开发设计, 核心库只关注图层, 拓展了 HTML 功能[6].

2.3 restful API

本文实验系统使用 restful 架构, 这是一种针对网络应用开发的架构, 具有简洁灵活高效的优点. 应用此架构, 前端页面不再需要数据表去保存资源, 所有的资源均通过 restful API 从服务器端获取, 保证前后端分离, 系统结构简洁高效.

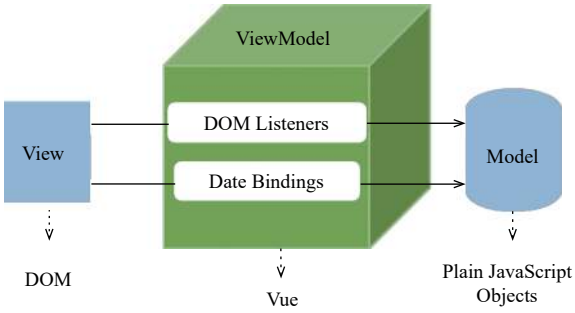


图 1 Vue.js MVVM 架构

2.4 数据库

本文实验系统采用 SQLite 数据库, 这是一种轻型数据库, 易于被集成到系统中, 且具有简洁、开源等优秀性能. SQLite 数据库在系统中用于组织、存储和管理数据, 从而保证数据的可靠性.

2.5 设备管理层

从硬件层直接获取不同类型子组件的信息在数据结构上难以保持一致, 本文实验系统在实际盘阵硬件形态之上封装了设备管理层, 用于实际盘阵硬件形态数据的统一管理, 包括设置参数和获取信息等功能.

3 系统框架及页面功能

本文研究的 3D 磁盘阵列可视化, 以单独页面的形式集成在系统的客户端中, 通过客户端与服务端进行通信, 为用户提供丰富的交互功能.

3.1 系统框架

本文提出的 3D 磁盘阵列可视化系统的框架如图 2 所示, 系统采用 B/S 结构设计, 分为服务端和客户端两大部分, 服务器端负责数据的收集、保存和传输; 客户端则呈现实时的 3D 磁盘阵列并实现信息展示和交互功能.

在服务端, 采用 Restful API+数据库+设备管理层的分层架构设计, 层次明确, 易于理解. 分层的架构使得开发语言多样化, 便于多人协同开发.

在客户端, 整个页面采用 Vue.js 框架, 其构建的 MVVM 模式使模型与视图的双向改变变得简单易行. 其中视图部分, 利用 three.js 的 API 对磁盘阵列进行 3D 仿真建模, 并添加丰富交互功能, 呈现在浏览器页面. 客户端与服务端又采用标准的 HTTP 协议进行数据的传输, 数据以 JSON 的格式进行发送和接收.

3.2 客户端功能介绍

本文实验基于 3.1 所述系统实现, 重点讨论客户端

部分, 3D 磁盘阵列的可视化设计及实现, 即图 2 中上虚线框中的内容. 主要功能包括如下两点.

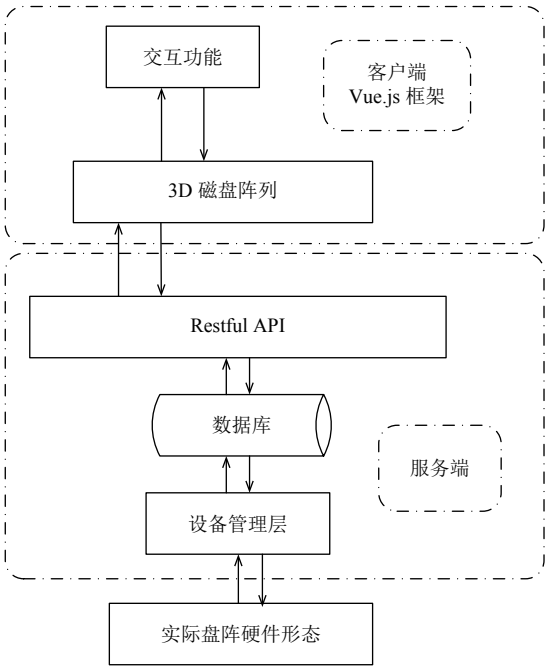


图 2 系统总体框架

1) 对磁盘阵列进行 3D 仿真建模, 并根据实际盘阵硬件形态进行资源映射. 在本文实验系统设计中, 将实际盘阵分解为若干子组件, 如机箱外壳、磁盘、网口、电源、主板和风扇等. 对于规则子组件, 根据实际盘阵的尺寸比例, 使用 three.js 提供的 API 进行绘制; 对于复杂子组件, 通过建模软件创建 OBJ 模型, 然后使用 three.js 提供的 API 导入. 判断实际盘阵硬件形态, 加载对应子组件模型, 并按照对应位置拼接到一起, 实现 3D 仿真建模.

2) 在 3D 磁盘阵列页面集成丰富的交互功能, 包括 3D 场景漫游、子组件选中高亮、单击获取子组件信息、改变实际盘阵硬件形态等交互功能.

4 3D 磁盘阵列的可视化

4.1 规则子组件的建模

针对 3D 磁盘阵列的绘制, 首先将其分解成多个子组件, 并抽象为具有规则形态的几何体, 例如机箱外壳、磁盘、网口、电源和主板外形都可用立方体近似. 通过测量实际盘阵的尺寸和位置参数, 采用 JSON 格式记录所有的尺寸和位置参数, 便于绘制规范和后期维护. 记录的数据形式如图 3 所示, 其中 sizeX、sizeY 和 sizeZ 分别对应组件的长、宽和高; posX、posY 和 posZ 对应组件的位置.

```
{
  "CTL0.0":   {
    "sizeX": 5,     "sizeY": 24, "sizeZ": 33.4,
    "posX" : -52.5, "posY" : 12, "posZ" : 81,
  },
}
```

图 3 磁盘数据示例

除了子组件的尺寸和位置参数, 还需要记录其外观和光源反应等信息, 以获取该组件的材质属性. 本文采用的方法是拍摄实际盘阵图片, 对图片进行处理压缩, 形成 JPG 格式贴图.

在计算机 3D 可视化中, 物体形状通常由三角形面组成, 通常把多个三角形面形成的网格模型叫做 Mesh 模型[3,5]. 根据 three.js 的 API 接口设计, 所有绘制的子组件也以 Mesh 形式存在于页面场景中, 通过 THREE.Mesh (geometry, material) 来创建, 其中需要的添加的参数 geometry 为 Mesh 对象的几何结构, material 为 Mesh 对象的材质.

Three.js 提供多种创建几何结构的 API, 可绘制平面、球体、立方体等等, 本文实验绘制的规则子组件都可抽象为立方体, API 接口为 THREE.BoxGeometry (sizeX, sizeY,sizeZ); three.js 也提供多种创造材质的 API, 本文实验中使用 THREE.MeshPhongMaterial ({map: new THREE.TextureLoader().load(url)}) 增加图片材质, 其中 THREE.MeshPhongMaterial 网格 Phong 式材料, 这种材质会考虑光照的影响, THREE.TextureLoader().load(url) 函数从指定位置加载图片文件, 图片格式可以为 PNG、GIF 或 JPG.

图 4 给出了单个硬盘子组件的 Mesh 样例, 其余子组件根据相同的方法绘制.

图 4 绘制单个磁盘

4.2 复杂子组件的建模

在 3D 磁盘阵列的绘制过程中, 存在一些形态复杂, 无法用基本几何体表现的子组件, 例如风扇等. 本文实验使用建模软件创建 OBJ 模型如图 5 所示.

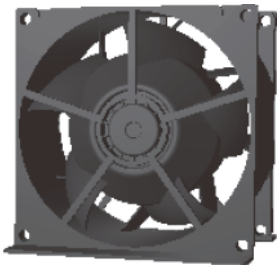


图 5 风扇 OBJ 模型示例

Three.js 可以读取多种三维文件格式, 如 JSON、OBJ 和 STL 等, 对应的格式都要引入一个额外的 JavaScript 文件. 如本文实验中使用的 OBJ 格式模型, 需先在页面中引入 OBJLoader.js, 然后通过 THREE.OBJLoader().load(url,function) 函数完成模型导入. 该函数接收两个参数, 其一是模型路径 url, 另一个为导入完成后的回调函数, 在回调函数中可以设置模型的位置以及大小.

4.3 资源映射

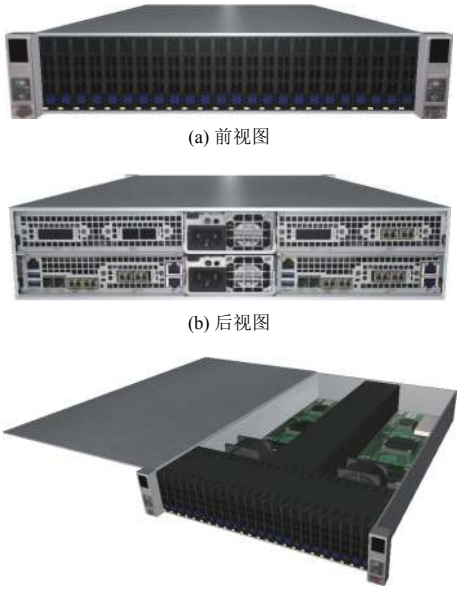
本节描述系统根据实际盘阵硬件形态, 绘制 3D 磁盘阵列的过程, 即资源映射的过程.

首先由客户端下发请求获取实际盘阵硬件形态的命令, 命令下发过程如图 2 自上而下箭头所示流程, 调用 Restful API 后, 在数据库中找到对应的表, 然后下发到设备管理层获取实际盘阵硬件形态信息; 将获取的信息返回客户端的过程如图 2 自下而上箭头所示流程.

在客户端获取到实际盘阵硬件形态信息后, 根据所述信息, 加载对应子组件, 并根据图 3 中的 posX、posY 和 posZ 参数设定位置, 将各部分子组件拼接到一起, 如图 6 所示, 实现资源映射全过程.

5 交互设计

由于磁盘数量较多, 开启定位灯能帮助用户快速定位磁盘. 在 3D 磁盘阵列中点击选中一块磁盘, 点击开启磁盘灯按钮, 命令下发过程如图 2 自上而下箭头所示流程, 调用 Restful API 后, 下发至设备管理层改变实际盘阵硬件形态, 即点亮磁盘灯. 磁盘灯被点亮后, 向客户端返回成功信息, 过程如图 2 自下而上箭头所示流程.



(a) 前视图

(b) 后视图

(c) 俯视图

图 6 3D 磁盘阵列三视图

为了给用户提供良好的交互体验, 在提出的 3D 磁盘阵列可视化页面添加交互功能, 分别为 3D 场景漫游、子组件选中、单击获取子组件信息以及改变实际盘阵硬件形态.

5.1 3D 场景漫游实现技术

当 3D 磁盘阵列按照硬件形态被绘制完成后, 为了全方位多角度的展示模型, 添加 3D 场景漫游功能, 此功能允许用户在场景内自由移动视角.

本文实验使用 three.js 提供的 OrbitControl.js(轨道控件), 首先在页面中引入 OrbitControl.js, 然后使用 THREE.OrbitControls(camera) 创建控件, 并将它绑定到相机上, 通过 orbitcontrols.rotateSpeed、orbitcontrols.zoomSpeed 等属性可分别改变鼠标控制相机的旋转、缩放速度.

通过使用 3D 场景漫游技术, 可全方位多角度的观察 3D 磁盘阵列, 如图 7(a) 为主板子组件的俯视视图, 通过鼠标滚轮滚动即可实现放大缩小, 如图 7(b) 为主板放大细节.

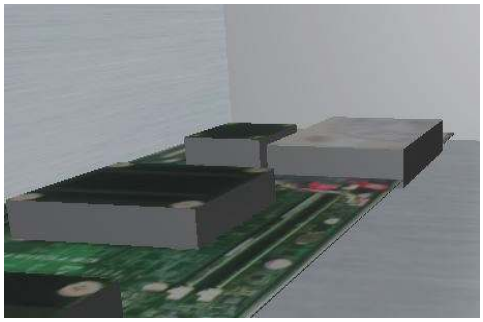
5.2 子组件选中

本节将介绍在页面中选中单个子组件的交互设计中, 如何解决不能进行 DOM 操作和如何在同时拥有 Mesh 和 OBJ 的同一页面进行选择的两个技术难点.

由于三维场景中不能进行 DOM 操作, 无法直接通过对每个子组件添加事件监听来实现交互操作, three.js 中提出了一种射线发射的方式来判断子组件是否被选中的方法[7]. 通过 THREE.Raycaster.intersectObjects(obj-

ects) 方法, 从屏幕上的点击位置向场景中发射一条射线, 在所有 objects 中第一个与射线相交的对象为被选中的子组件.

(a) 主板俯视视图



(b) 主板放大细节图

图 7 漫游功能示例

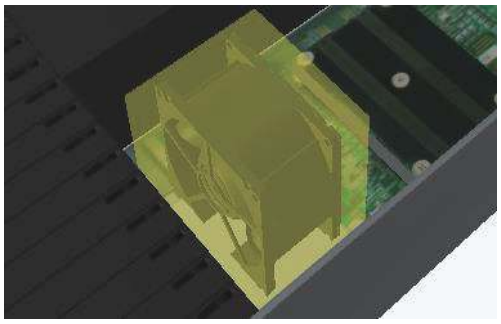
使用上述方法判断选中子组件, 只能检测到 Mesh 格式对象, 即 4.1 节所述的规则组件, 而 OBJ 模型是多个 Mesh 形成的 Group 形式, 无法被检测. 为了实现 OBJ 模型的点选功能, 将 OBJ 模型 Children (Mesh 格式) 存入数组. 遍历其数组, 如果 OBJ 模型任意 Children 被选中, 则判断该 OBJ 模型被选中.

在 3D 磁盘阵列中添加鼠标移入子组件高亮的交互功能, 无论是调用 Three.js 的 API 绘制的子组件电源, 如图 8(a) 所示, 还是导入的 OBJ 模型风扇, 如图 8(b) 所示, 都可以在鼠标移动的过程中被选中.

5.3 子组件信息

在 5.2 的基础上, 将鼠标移入移出的交互方式改为左键单击, 被单击的子组件使用浅蓝色遮罩区分移入选中, 如图 9 所示. 鼠标单击动作绑定获取该子组件详细信息事件, 命令下发过程如图 2 自上而下箭头所示流程, 调用 Restful API 后, 下发至设备管理层获取被点击组件详细信息, 以 JSON 形式传给客户端. 客户端页面使用 Vue.js 框架, 将子组件与表单绑定, 数据成功返回即页面显示其详细信息, 如图 10 所示.

(a) 电源



(b) 风扇

图 8 子组建选中示例

图 9 控制器 B 被选中

图 10 控制器 B 的详细信息

5.4 改变实际盘阵硬件形态

客户端接收到返回成功信息, 通过 THREE.Mesh

BasicMaterial({color:0xff0000}) 创建红色材质, 改变 3D 模型中对应磁盘定位灯的颜色, 如图 11 所示.

图 11 磁盘定位灯示例

6 总结

本文针对于传统磁盘阵列可视化的缺陷, 对其进行升级, 提出了基于 Three.js 的 3D 磁盘阵列可视化框架设计和实现. 针对简单和复杂形态的子组件, 分别采用简单几何体和 OBJ 模型的形式进行建模, 并根据实际盘阵的形态进行资源映射. 在图形界面提供了 3D 场景漫游、子组件选择、单击获取子组件信息和改变实际盘阵硬件形态等功能, 为用户提供良好的交互体验. 二者被有机结合并集成到系统软件中, 提升了系统的易用性. 因此该研究是有理论意义且具有应用价值的.

参考文献

1 Zakas NC. JavaScript 高级程序设计. 李松峰, 曹力, 译. 3 版. 北京: 人民邮电出版社, 2012.

2 杨润斌, 梁文章, 陈姝. 基于 WebGL 的 3D 动画实时播放系统. 计算机系统应用, 2015, 24(11): 58–63. [doi: 10.3969/j.issn.1003-3254.2015.11.009]

3 张玄, 黄蔚. 3D 机房运维可视化系统的设计与实现. 软件, 2016, 37(7): 89–93. [doi: 10.3969/j.issn.1672-7800.2016.07.035]

4 方路平, 李国鹏, 洪文杰, 等. 基于 WebGL 的医学图像三维可视化研究. 计算机系统应用, 2013, 22(9): 25–30. [doi: 10.3969/j.issn.1003-3254.2013.09.005]

5 任宏康, 祝若鑫, 李风光, 等. 基于 Three.js 的真实三维地形可视化设计与实现. 测绘与空间地理信息, 2015, 38(10): 51–54. [doi: 10.3969/j.issn.1672-5867.2015.10.016]

6 王敏, 张昆. 基于 THREE. JS 和 Google Map API 的网页交互可视化技术——以等角航线为例. 测绘与空间地理信息, 2015, 38(7): 158–161. [doi: 10.3969/j.issn.1672-5867.2015.07.054]

7 王双, 廉东本, 陈月. 基于 Unity3D 的仓储可视化管理系统. 计算机系统应用, 2016, 25(7): 72–76. [doi: 10.15888/j.cnki.csa.002543]