分类号_____ 学号 M201176004

学校代码 10487 密级_____

华中科技大学

硕士学位论文

基于 WebGL 实物交互技术及其实现的研究

学位申请人　宁　静

学 科 专 业：软件工程

指 导 教 师：裴小兵　副教授

答 辩 日 期：2014.1.8

A Thesis Submitted in Partial Fulfillment of the Requirements

for the Degree for the Master of Engineering

Research on WebGL-based Physical Interactive Technology and Its Implementation

Candidate : Ning Jing

Major : Software Engineering

Supervisor : Assoc. Prof. Pei Xiaobing

Huazhong University of Science and Technology

Wuhan 430074, P. R. China

January, 2014

独创性声明

本人声明所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除文中已经标明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：

日期：　　年　　月　　日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，即：学校有权保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权华中科技大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本论文属于　保密□，　　在_____年解密后适用本授权书。

　　不保密□。

（请在以上方框内打“√”）

学位论文作者签名：　　指导教师签名：

日期：　　年　　月　　日　　日期：　　年　　月　　日

摘 要

传统的交互方式极大的限制了交互设计的发展，同时对呈现效果及用户体验具有极大的局限性，所以对实物交互技术的研究也就显得尤为重要了。随着 Web 3D 技术的发展，Web 化应用越来越普遍，对 Web 页面上交互技术的研究也越来越重要了。WebGL 的出现，给日新月异的交互技术带来了新的挑战，同时也给 Web 上实物交互的发展带来了很多机遇。目前，基于 WebGL 的实物交互在很多领域已被广泛应用。为顺应时代发展，基于 WebGL 实物交互技术及其实现的研究也就应运而生了。

通过研究 WebGL 及与其相关的一系列知识，并通过实际建模完成了对基于 WebGL 实物交互技术及其实现这一课题的研究和探讨。

对基于 WebGL 实物交互技术及其实现这一课题的研究首先分析对比了 Web 3D 技术的实施方案，发现 WebGL 以其绝对的优势胜出，并对 WebGL 及其相关理论知识进行了研究。由于 WebGL 的原生 API 是非常低等级的，用其进行代码开发会非常复杂，需要选取 WebGL 的引擎技术来减少编程工作量。因此，对当前几种主流的 3D 引擎技术进行了对比，并通过分析选取了 Three.js 作为研究所使用 WebGL 框架。随后给出了基于 WebGL 和基于 Three.js 实物交互过程，并根据该理论基础进行了实物模型的建模。最后，将 LOD 优化算法引入到 WebGL 技术中来，使基于 WebGL 实物模型的渲染速度得以提升。

关键词：实物交互　　WebGL 技术　　Three.js 引擎　　细节层次模型

Abstract

Traditional interactive style greatly limits the development of interaction design, and has great limitations to the rendering effect and the user experience, so the study of the technology of physical interaction it is particularly important.With the development of Web 3D technology,Web applications become more common, the study of interactive technology on the Web pages are increasingly important.The emergence of WebGL brings new challenges to the ever-changing interactive technology, and brings a lot of opportunities to the development of physical interaction on Web pages.Nowadays, WebGL-based physical interaction has been widely applied in many fields. To keep up with the development of The Times, the research of WebGL-based physical interactive technology and its implementation has been produced.

Completed the research and discussion on WebGL-based physical interactive technology and its implementation by learning a series of related knowledge with WebGL and programming the actual modeling.

First of all, the research on WebGL-based physical interactive technology and its implementation, analyzed and compared the implementation plans of Web 3D technology, found that WebGL with absolute advantages, and have a study on WebGL and the related theory with it. Because the native WebGL API is very low level and developing code with it will be very complicated,and so requires us to select a engine technoloy of WebGL to reduce the programming effort. Therefore, compared the several current mainstream engine technologies of 3D , and selected the Three.js as the framework of WebGL. Subsequently, WebGL-based and Three.js-based techniques of physical interaction were described in detail, and have a implementation according to the working principle. In the end, the LOD optimization algorithm is applied into WebGL technology to enable the speed of WebGL rendering can be improved.

Key words： Physical interaction WebGL technology Three.js engine LOD model

目 录

1 绪论

1.1 课题研究的背景与意义

进入信息时代后，2D 平面所营造的单调的虚拟环境越来越难以满足人们的交互体验需求。用户希望能获取更友好的交互体验感。因此，传统的由输入工具所进行的交互，已极大的限制了交互设计的发展，同时，对呈现效果及用户体验具有极大的局限。

随着计算机与人们日常生产生活联系的日益紧密，网络信息媒体空间与物理空间的日益融合，我们将重新思考对人机交互概念的理解。经过一系列的探索和研究，传统的交互方式已经开始向新的实物交互的交互方式发展。新型交互方式的界面呈现使人与计算机之间的交互越来越友好、真实。为了使交互感更加真实，一些研究在交互系统中引入实物模型来构成用户界面。该界面允许用户通过日常的交互经验来操控物理对象以实现与数字空间的交互，从使用户得到更好的交互体验[1]。同时，随着人机交互方式的迅速发展与转变，虚拟界面的呈现形式也从 2D 向 3D 转化，从单调的二维平面向三维空间延伸。于是，虚拟现实技术也就随之产生了。虚拟现实技术使用户仿佛处于真实世界一样，能够直接与虚拟环境中的虚拟对象进行交互，使其得到更逼真的体验感，进而产生沉浸感，以使虚拟操作效率得到提高。同时，虚拟现实技术也能为人们探索世界提供极大的便利，具有很好的经济、社会效益和诱人的发展前景[2]。

信息技术的高速发展，给实物交互的发展带来了很多机遇。随着嵌入式技术和网络信息技术的发展，实物交互越来越趋于被广泛应用。例如，现在的办公设备、家具、汽车、手机等一些产品都带有微处理器，还拥有控制和用户接口功能[3]，一些甚至还带有交流功能，允许通过网络和电话线来连接其它系统。这就可以将实物世界和虚拟世界紧密连接起来。因此，实物交互已然成为了众科研工作者及商业生产者竞相研究的宠儿。

然而，随着 Web 3D 技术的出现，又给日新月异的交互技术带来了新的挑战。

特别是当互联网进入 Web 2.0 时代之后，Web 应用逐渐走进了人们的视线，并正在逐步占据主导地位。随着应用软件的发展，越来越多的应用软件趋于 Web 化，这就直接导致了 Web 3D 技术的迅猛发展[4]。在 WebGL、OpenGL、Java3D、Superscape、blaxxun3D、Cult3d、Vrpie 等众多 Web 3D 技术中，无疑，WebGL 具有最突出的优势：一、HTML5 的 Canvas 元素可以得到由 WebGL 提供的硬件加速，Web 开发人员可以利用这一优势在电脑的浏览器上更流畅地对三维场景和模型进行展示；二、使用 WebGL 进行 3D 图形渲染，可以仅仅只使用 JavaScript 和 OpenGL ES 2.0 语言就能开发出性能良好、效果逼真的图形、动画，甚至是游戏，不需要任何其它的插件（plug-in）；三、WebGL 渲染的内容能够很好的与整个 DOM（Document Object Model）架构融合，因为 WebGL 的 3D 渲染环境是在 HTML5 的 canvas 元素中搭建的，所以通过 WebGL 渲染的图形是在 canvas 元素中显示的[5]。随着实物交互的概念越来越深入人心，实物交互的概念已逐渐渗透到人们生活的方方面面。WebGL 优越的性能能给用户带来流畅的 3D 动画渲染体验。

历史告诉我们，无论应用的内容是多么的丰富有趣，我们任然需要研究一种与之进行实物交互的方式，这样才能够满足用户的需求，吸引人们的注意，顺应时代的要求[6]。

1.2 课题研究现状

随着各行业对计算机以及网络信息依赖性的日益密切，人机交互的发展与创新越来越受到各界人士的广泛关注。实物交互可以使用户沉浸于一系列应用领域的虚拟环境中，包括娱乐、教育、设计、导航等。

虚拟现实（Virtual reality，简称 VR）技术是实物交互最主要的实现手段[7]。实际上，虚拟化技术是一个广义上的概念[8]，它是一种重新配置的过程，该过程可以使资源解决方案得以优化，硬件容量得以增大，软件得以简化[9]。

对于实物交互的研究，发展最快的就是美国了。美国对 VR 技术的研究起源于 20 世纪 40 年代。

美国 UC Berkeley 漫游工作室，对 Berkeley 大学的计算机大楼进行了实时漫游，

构建了 3D 模型，并发现、修正了该建筑中的缺陷[10]；第一个用于建筑设计的三维虚拟建筑系统是由来自北卡罗来纳大学的 Brooks 教授带领的小组研发而成的；麻省理工学院（MIT）1985 年成立了媒体实验室，建立了一个用于不同图形仿真技术的 BOLIO 测试环境；美国宇航局（NASA）Ames 实验室，研究了空间站操纵的实时仿真，并建立了空间站 VR 训练系统[11]。

另外，还有一些欧洲较发达国家都在致力于实物交互技术的研究，并将其进行应用。荷兰海牙 TNO 研究所研发的训练和模拟系统，可以使用户完全沉浸在虚拟环境中[12]；德国的传统制造业中，为了提高工人的生产效率和操作水平，使用了虚拟工厂；由日本千原国宏小组开发的嗅觉模拟器，是 VR 技术在嗅觉研究领域的一项重大突破；法国的达索系统，用于设计制造 3D 的汽车和飞机模型；欧特克的 3D 数字领域，更是引领世界的风向标。

对于实物交互的研究，不仅国外发展迅猛，国内对虚拟现实技术研究与应用正在迅速发展中。不仅国内许多学者和研究机构在对其进行研究并且取得了一些不错的研究成果，而且国家科委国防科工委部还把它列为重点研究项目[13]。

北京航空航天大学计算机系是国内最早进行虚拟显示技术研究的单位，同时也是最具有权威的单位之一，它重点研究了虚拟环境中的实物物体的物理表征，并提出一些可以与视觉相连接的硬件接口的相关算法及实现方法[14]；采用 QuickTime 技术实现的布达拉宫全景的实物虚拟化由清华大学国家光盘工程研究中心完成[15]；浙江大学 CAD＆CG 国家重点实验室开发的虚拟建筑环境实时漫游系统是一套桌面型的模拟真实环境的系统；哈尔滨工业大学计算机系在人工智能方面的虚拟现实技术的研究已取得很大成就，成功地解决了特定人脸合成技术问题，并正在对人说话时语音与动作同步等问题进行研究[16]。

随着时间的推移和信息科技的不断发展，实物交互越来越引起各界人士的广泛关注。在这种情况下，我国对 VR 技术的研究已取得许多成绩，但是与一些发达国家相比，差距任然很大[17]。

WebGL 的崛起，将实物交互的热潮引领到了一个新的高度。

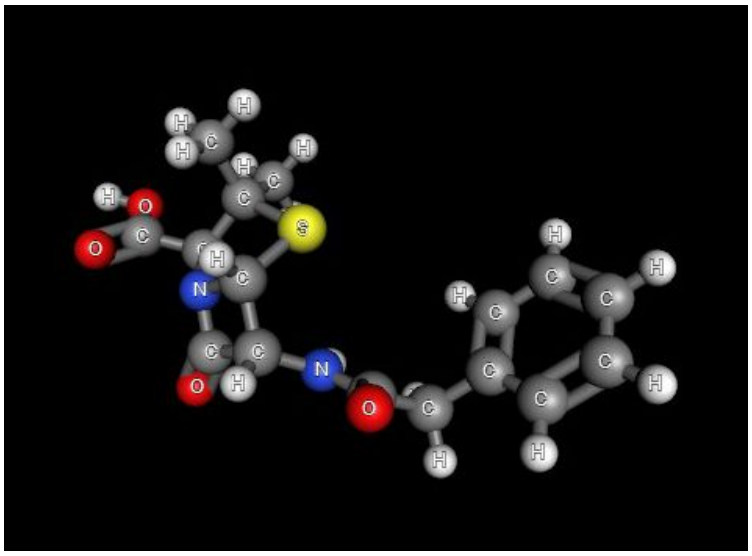


图 1-1　三维化学构造

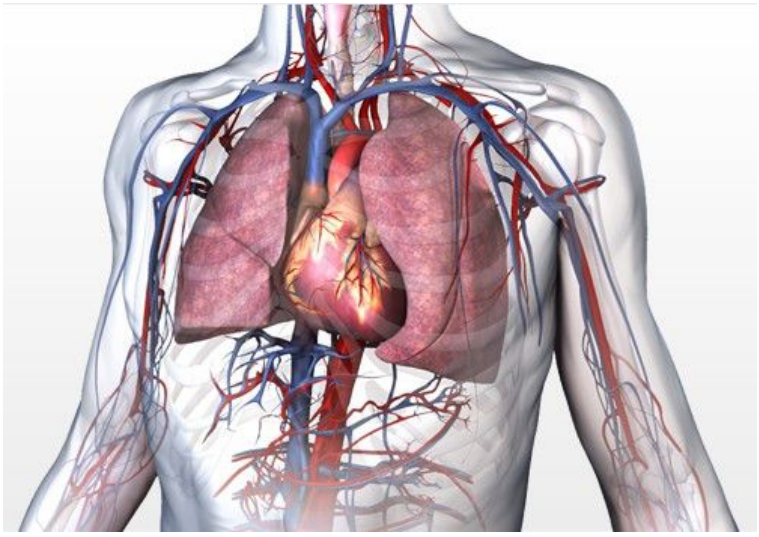


图 1-2　三维人体解剖

如图 1-1、1-2 所示，用 WebGL 制作的实物图形其 3D 画面渲染效果的逼真程度及精致程度令人震撼。

现在，基于 WebGL 的实物交互已被广泛应用于电视电影、游戏、教育、模拟分析、建筑规划等行业。Human Engines 和 Gregg Tavares 开发的 Aquarium 在浏览器中实现了让人惊叹的逼真效果；Aleksandr Rodic 创建的水母模拟（http://chrysaora.com）画面效果足以以假乱真；康威的名为《游戏人生》（Game of

Life）的游戏；由 Tony Parisi 构造的太阳系模型，用 3D 图形模拟了真实的太阳系的运转；来自于 3DBS 的机器人模型，用关键帧创建了机器人的关节动画；另外，还有由 dskfnwn 制作的科迈罗轿车模型，AlteredQualia 开发的街道场景图等等。

虽然，当前的 WebGL 集众多优点于一身，但是它任然还处于不成熟阶段，还有待进一步的研究和完善。

1.3 基于 WebGL 实物交互的发展趋势

目前，基于 WebGL 的实物交互在模拟分析、建筑规划、游戏娱乐、设计、教育等领域上已经有一定的应用。未来，还有望在医疗、家电等领域进行应用。未来虚拟现实技术和 3D 技术的普及将是大势所趋。在实物环境中，生动有效的交互需要以下几点：

（1） 对用户和实物之间关系的控制：在连接物理空间和虚拟信息空间时，对于接口机制之间相互映射的控制和变换成了设计中至关重要的问题。因为用户和实物之间的可能不是很直接的关系。

（2） 交互敏感度的稳健性：人类的行为模式是不能被精确衡量的，除非这些行为是必要的并且可以通过一些手段来加强的。因此，系统应该足够强健来接受用户各种环境状况下不同的交互行为。

（3） 交互呈现模型的灵敏度：人机交互或者人通过虚拟环境与实物进行交互时，需要充分考虑到交互行为的灵敏度。

（4） 合理使用与虚拟环境之间的物理接口：使用物理接口可以使实物对象或者虚拟环境的功能或者交互形态得以扩展。

由于人们对新型交互方式需求的日益增长，实物交互将会渗透到人们生活、生产、教育、娱乐、购物等各方各面。对于传感技术、计算机图形图像等实物交互基础技术的探研，更是会推动实物交互的发展。

随着互联网的发展，单机的实物交互系统已然不能满足现状，人们还要求系统相互之间能够进行通信交流，这在很大程度上需要依赖于网络。而基于 Web 的实物交互系统，不仅能在既满足人们对交互体验的高要求的同时，又能满足系统间的相

互通信，更具优势的是 WebGL 使绘制 3D 图形变得相对简单。在 WebGL 技术出现之后，许多应用中已出现了这一技术的“身影”，例如，三维地理信息平台[18]、家庭娱乐终端[19]、虚拟实验教学[20]、在线多人 3D 协作游戏[21]等。相信在未来，还有更多的基于 WebGL 技术的实物交互系统会如雨后春笋般涌现。因此，对于本课题的研究是很有意义的。

1.4 本文研究内容

随着 Web 3D 技术的出现，特别是当互联网进入 Web 2.0 时代之后，WebGL 技术以其不可阻挡的发展趋势，迅速“占领”了各行各业。为顺应时代的发展要求和如今互联网行业的发展趋势，对基于 WedGL 的实物交互进行了研究。本文的主要研究内容有：

（1） 对 3D 图形绘制基础、实物交互、Web 3D 技术、WebGL、Three.js 引擎、LOD 算法的背景、概念、优缺点以及发展现状等进行了一个系统性的介绍；

（2） 对当前主流的几种 WebGL 引擎技术进行了介绍，并作出了分析对比；

（3） 分别给出了基于 WebGL 和基于 Three.js 的实物交互过程；

（4） 实现了基于 WebGL 和基于 Three.js 的三维实物的建模，并截取了部分主要代码辅助说明；

（5） 详述了使用旋转矩阵实现物体旋转（即坐标系迁移）与物体位移的变换算法，并将编码实现变换算法；

（6） 对实现实物交互的代码运用 LOD 算法进行了优化，并对优化前后进行了测试，同时对测试结果进行了分析对比。

1.5 论文组织结构

本文总共分为五章，其主要内容如下：

第一章，绪论。介绍了课题研究的背景与意义、国内外研究现状以及基于 WebGL 的实物交互的发展趋势。另外，还对本文的主要内容进行了概括，并给出了本文的组织结构。

第二章，关键技术分析。简要介绍了实物交互、Web 3D 技术、WebGL 技术以及与其相关的引擎技术，并对 Web 3D 的主要实施方案和 WebGL 相关的主流引擎技术进行了对比分析。

第三章，基于 WebGL 实物交互的过程。分别给出了基于 WebGL 和基于 Three.js 的实物交互过程。

第四章，基于 WebGL 的优化算法。描述了优化技术的选择过程及 LOD 算法被选择的原因；描述了在当前基于 WebGL 实物模型的基础上使用 LOD 优化算法进行优化的主要思想。

第五章，分别使用 WebGL 和 Three.js 实现了实物模型的建模及交互操作。然后，使用 LOD 算法中的删减法对实物模型进行了优化处理，并对处理结果进行了对比分析。

第六章，总结与展望。对全文内容进行总结，并分析了已完成的工作及研究过程中遇到的问题。随后，对该课题下一步的研究工作做出了规划，并对 WebGL 未来的发展形势进行了预想和推测。

2 关键技术分析

本章分别对实物交互、Web 3D 技术及其主要实施方案进行了一个简单的介绍。对比发现如今 Web 3D 的主要实施方案为 WebGL，于是对 WebGL 及其相关引擎技术作出了概述并进行了分析对比。

2.1 实物交互技术

实际上，实物交互是一种新型的界面交互呈现方式，是网络信息技术高速发展的标志，是一种使人仿佛置身于真实环境中的人机交互方式，能给人带来更好的交互体验感[3]。

实物交互属于人机交互的一种。人机交互（Human-Computer Interaction 或 Human-Machine Interaction，简称 HCI 或 HMI），指的是人与机器之间的互动关系。该互动关系需要通过分析人的行为习惯，探索人与机器之间的互动关系，并将其用直接、易懂的界面展示出来才能得到。然而，实物交互在此基础上，还需要再对实物对象与信息空间的关系进行分析才能得到。实物交互不仅要考虑人机交互问题，还需要在以人为中心的基础上考虑物理空间与媒体空间的交互问题。所以实物交互相对于传统的人机交互更难以实现，因为它与传统的人机交互相比，不仅与计算机图形图像、网络技术等技术有关，还与 3D 技术、嵌入式技术等密切相关。

虽然实物交互在当今社会还没有得以普及，但是随着人们对交互体验越来越高的追求，没有什么可以阻挡它快速发展的步伐。目前，Web 3D 的高速发展和逐渐普及，就是这一现象的印证。

2.2 Web 3D 技术

2.2.1 Web 3D 技术简介

Web 3D 技术是由 Web 2.0 和 3D 技术结合而产生的[22]。Web 3D 有一个独特的技术特点，它在低成本的情况下可以制作出虚拟现实的效果，它不仅可以在虚拟空

间中进行设计和创造，还可以构造虚拟场景[23]。Web 3D 的本质就是对 3D 图形进行实时渲染并且允许与其进行交互的技术，其目的是可以在浏览器的 Web 页面中渲染 3D 虚拟世界[24]。

上世纪 90 年代末，VRML 组织更名为 Web3D 组织。也就是说，Web 3D 技术也是虚拟现实技术的一种。VRML（Virtual Reality Modeling Language）是一种描述虚拟现实世界中交互行为的语言，它通过万维网（WWW）将世界的网络联通[25]。于上世纪 90 年代，VRML 1.0——VRML 标准的第一个版本发布了，之后的 VRML 2.0，也就是后来的 VRML 97 国际标准，由美国计算机协会计算机绘图专业组（Siggraph conference）发布[26]。

Web 3D 技术将虚拟现实技术应用到网络中，它可以通过网络在浏览器上创建虚拟环境[27]。Web 3D 技术可以同时满足 Web 网页和 3D 交互，并且 Web 3D 的实现突破了时间、空间以及地域的限制，随时随地为 Web 网页用户带来了空间感和体验感的全面提升，这种表现方式被越来越多的人认识并接受。随着网络带宽的增长，Web 3D 内容更是得以被广泛传输和使用。

2.2.2 Web 3D 实施方案的对比与选取

至今，已有许多 Web3D 技术的实施方案，其中主要包括[28]：VRML/X3D、Java3D、Cult3D、ShockWare3D。其具体情况如下：

（1）VRML/X3D：VRML 标准使用 C/S 模型：服务器用于存储 VRML 文件，然后再客户端请求时，再将这些文件发送出去。客户端将解释这些文件，然后再虚拟场景下进行渲染。X3D 在 VRML 97 的基础上，更改并添加了一些特性，例如，图层转移、纯色画笔、2D 矢量，尤其是支持 XML，这些特性提高了交互的可行性和扩展性。

（2）Java3D：面对 Web 化的应用程序，Java3D 对其使用 Applet 程序。Applet 是运行在 Java 虚拟机（JVM）上的小程序，一般被安装在普通用户的电脑上。Java3D 只关注场景控制模块和逻辑控制模块，并自动将模块发送到 Web 服务器上，让开发者只用关注用户交互和交互设计。

（3）Cult3D：Cult3D 是一个基于 Java 的 Web3D 实施方案，它是由 Cycore 公司制作的。由 Cult3D 所产生的文件很小，同时与其它的 Web 3D 工具相比，其图像质量却很高。因为 Cult3D 在发送图像时，采用的是一种高效的压缩算法。另外，Cult3D 对初学者来说，很容易学习。

（4）ShockWare3D：ShockWare3D 是由 Intel 和 MacroMedia 两大公司开发出来的。它支持许多特效，如：骨骼变形、细节层次。

在上述所有实施方案中，又以 Java3D 结合 VRML 的实施方案方式为主，其中 VRML 作为描述工具，Java3D 作为渲染实现工具[29]。虽然，在大潮流下 Web 3D 技术已然得到了飞速发展，但是上述所有 Web 3D 的实施方案，都有一些共同的缺陷和弊端，比如，都是以插件形式存在的、发展都很困难。并且与虚拟现实技术相比，Web 3D 在逼真性以及交互性等方面还是有一定的差距的。

然而，WebGL 的发行则为 Web3D 的发展指明了新的方向，为未来三维应用在 Web 上表现的更加完美带来了曙光。目前，WebGL 技术已成为实现 Web 3D 的主流技术。在接下来的章节中，将为大家着重描述 WebGL 技术。

2.3 WebGL 技术

2.3.1 WebGL 简介

WebGL（Web-based Graphics Library）拥有先进的 3D 绘图特性（着色器），尤其是对基于浏览器的结构复杂的 3D 页面的发展很有优势。许多先进的嵌入式技术和便携移动设备都支持 WebGL[30]。因此，WebGL 对于许多设备和系统来说，具有很大的吸引力。

随着科学技术的发展与进步，在上世纪 90 年代，被美国 SGI 公司在跨平台移植 GL 的过程中，逐渐演变成为被人们了解与熟知的 OpenGL 1.0[31]。OpenGL（即开放性图形库 Open Graphics Library），是一个专门用于绘制 3D 模型的开源库[32]。自此之后，OpenGL 陆续发布了 OpenGL 1.X、OpenGL 2.0、OpenGL ES 2.0、OpenGL 3.1 等众多版本。

WebGL 实际上是在 OpenGL ES 2.0 的基础上发展的。WebGL 不仅集成了 OpenGL 的所有优点，而且还对 OpenGL API 予以了简化。WebGL 可以分别对图形

的光线、纹理、景深、粒子系统、碰撞检测和响应[33]进行控制。

图 2-1　OpenGL 发展时间轴

如图 2-1 所示，WebGL 的工作草案在 2009 年才面世，并且它的第一个稳定版本一直到 2011 年 2 月才完成。但是 WebGL 在短的时间内，它迅速得到了绝大多数主流浏览器的支持，其中包括 Safari 浏览器、Chrome 浏览器、Firefox 浏览器以及 Opera 浏览器[5]。换言之，目前除了 Microsoft 的 Internet Explorer 之外，其它主流的最新版本的浏览器都已支持 WebGL，具体支持情况表 2-1 所示[34]：

表 2-1　主流浏览器对 WebGL 的支持情况

IE	Firefox	Safari	Chrome	Opera
6.0 ×	3.5 ×	3.2 ×	10.0 √	10.6 ×
7.0 ×	3.6 ×	4.0 ×	11.0 √	11.0 ×
8.0 ×	4.0 √	5.0 ×	12.0 √	11.1 ×
9.0 ×	5.0 √	6.1 √	13.0 √	11.5 ×
	6.0 √		14.0 √	
	7.0 √			
10.0 ×	8.0 √	6.0 √	15.0 √	12.0 √

2.3.2　WebGL 的优势

3D 精湛的画面及生动的交互体验，对人们产生了巨大的吸引力和震撼感。对于未来，3D 技术有着极大的潜在市场。但是在 WebGL 诞生之前，大家很少能在 Web 上看到特别逼真的 3D 效果，因为在 Web 上很难实现 3D 效果，除非是借助一系列的插件或工具。

现在，随着 HTML5 标准的提出和进化，WebGL 作为 HTML5 标准之一，已经

完美的解决了现存的 Web 交互的两个问题：一、只需通过 JavaScript 便可以在浏览器中实现实物交互，不需要任何其它的插件；二、WebGL 根据其统一的、标准的、跨平台的 Web 图形标准进行图形渲染的同时，可以对硬件加速以加快图形渲染的速度[35]。WebGL 的主要特性如下：

（1）WebGL 实际上是一套 API：WebGL 提供一套 JavaScript 的编程接口，直接用 JavaScript 来进行图形的绘制。其所有的绘画与渲染功能都需要通过 JavaScript 编来进行调用。

（2）可以搭建动态 Web 应用：WebGL 与 JavaScript 配合，可以很好的在浏览器上搭建 3D 图形，效果更加流畅自然。因此，WebGL 对于 Web 交付是非常友好的[36]。

（3）跨平台：WebGL 可以很好的运行于任何操作系统，任何设备。

（4）开源：对于 WebGL 的使用，是完全免费的。

WebGL 产生自 OpenGL ES 2.0，在继承了 OpenGL ES 2.0 优点的同时，还在 OpenGL ES 2.0 的基础上做出了一些改进。因此，WebGL 能得到众多科研工作者和企业的支持是有其一定的道理的。

2.3.3 WebGL 的安全问题

近几年，随着 WebGL 的迅猛发展，业内对 WebGL 的安全问题，也一直存在着各种疑问。目前，就 4 个方面的安全性问题，WebGL 标准官方进行了简要介绍[35]：

1）资源与越界方面的限制

资源初始化与越界方面的限制是指 WebGL 中的任何资源（如顶点、纹理等对象）都必须进行初始化，设置初始化数据。

WebGL 资源通过调用 drawElement 或 drawArray 进行图形绘制时，必须保证没有越界存取的情况，没有对未初始化的内容进行存取的情况。

2）源安全限制

源安全限制（origin restriction）是指必须对 HTML5 canvas 元素的 origin-clean 属性进行设置以保证信息安全。对于以下情况，必须把 origin-clean 的值设置为 false：

（1）当使用的 Document 对象来自不同的 canvas 元素时，其

HTMLImageElement 或 HTMLVideoElement 调用 texImage2D 的情况

（2）当 origin-clean 设置为 false 时，HTMLCanvasElement 对 texImage2D 进行调用的情况

3）对 GLSL 的支持

OpenGL 的着色语言（OpenGL Shading Language），即为 GLSL。它必须按照 OpenGL ES Shading Language 1.0 的标准且不能超越在 OpenGL 标准中附录 A 中强制的最小功能来进行 WebGL 的实现。

4）拒绝服务式攻击的防护

拒绝服务式攻击的防护是指经过组合后的渲染使得绘画时间过长。也就是说，在脚本需要被长时间运行的情况下，用户代理可以进行自我保护。但是，绘图调用的长时间运行可能会导致整个页面，而不仅仅是用户代理，失去交互性，也就是人们口中的“死机”。

用户代理应防止绘画时间和交互损失的时间过长：

（1）尽量采用较小的绘图调用；

（2）若绘图调用时间过长，则禁止后续操作；

（3）使用系统提供的工具限制绘图调用的时间；

（4）把用户代理的图像绘制调用作为一个单独的进程以结束或者重启无损程序状态。

2.3.4 WebGL 引擎技术的对比与选取

1）GLGE

GLGE 是一个用于简化 WebGL 的 JavaScript 库。GLGE 是一个原生的基于浏览器的 JavaScript API，可以直接与 openGL ES2 进行连接。在无需任何插件的情况下，通过使用硬件 GLGE 可以对 2D/3D 应用程序进行加速。其 API 文档非常完整，可直接用于工程项目开发。

目前，GLGE 支持的主要特性包括：关键帧动画；Perpixel 定向照明灯、投光灯和点光源；法线及置换贴图；动画材质；骨骼动画（WIP）；Collada 格式支持；视

差映射；文本渲染；深度阴影；基于着色器的选择；环境映射；反射/折射；Collada 动画。

但是就目前而言，GLGE 并不是很成熟。它还处开发阶段，新功能还不完善，而且 GLGE 没有中文版的 API 文档，使用起来很不方便。

2）PhiloGL

由 Sencha 实验室发布 PhiloGL 是一个 WebGL 的开源开发框架，用于数据可视化、创意编码以及游戏开发。其 API 文档比较完整，可直接用于工程项目开发。它的主要优点有：① 惯用的 JavaScript：PhiloGL 是建立在 JavaScript 良好的习惯用法基础之上的。它提供了功能强大，很有表达力的 API；② 专注于性能：为 WebGL 提供一个清晰的抽象定论的同时，还试图尽可能的接近 gl 调用，同时；③ 完善的模块：PhiloGL 的模块不仅仅涵盖程序和着色器管理，还包含 XHR、JSONP、效果等等；

但是 PhiloGL 也存在着与 GLGE 同样的问题：使用起来很不方便，因为 PhiloGL 目前还没有中文版的参考资料。

3）C3DL

C3DL 是根据 Canvas 3D（一个 Firefox 插件，也就是现在 WebGL 的前身）而得名的。它最早是作为 CATGames 的一部分进行开发的，其开发的主要目的是为了简化 WebGL 的使用。它的主要特性有：在运行时，可以交换纹理作为场景；用视频文件和 canvas 标签作为对象的纹理；关于对象，提供更多可操控的信息，如三角形的数量、对象的大小（缩放到特定大小）、居中的对象等；碰撞检测；可以暂停/停止/启动场景（渲染和更新）；可以标记场景中的静态对象，所以时间不会浪费在计算一个不移动物体的动作上。

但是，C3DL 的技术开发背景是非常复杂的。目前，它也没有较好的中文版的资料文档。

4）Three.js

Three.js 是由西班牙巴塞罗那的程序员 Ricardo Cabello Miguel 所开发的。其主要特点有：掩盖了 3D 渲染的细节；面向对象；功能丰富；速度快；支持交互；拥

有一个强大易用的数学库；扩展性强。

在上述几种 WebGL 的第三方引擎技术中，选择 Three.js 作为 WebGL 的第三方引擎框架。因为 Three.js 跟其他三种第三方框架引擎技术相比，其最大的优势在于入手很容易，很适合初学者使用。不仅如此，在众多引擎中，Three.js 是最受欢迎的，其资料完善，文档和案列也很多。使用 Three.js 可以非常方便的创建各种 3D 场景，另外，为了提高性能，Three.js 中还包含了图形引擎的高级技巧[36]。并且，由于内置了很多操作简易的工具，所以 Three.js 拥有非常强大的功能。

2.5 Three.js 技术

Three.js 的开发目标是创建一个轻量级的，具有非常低复杂度的 3D 图形库。Three.js 是一个跨浏览器的基于 WebGL 的 JavaScript 库。 其脚本可以与 HTML5 的 canvas 元素、SVG 或者 WebGL 结合使用。

当然，Three.js 也有其不完善的地方。目前的 Three.js 还很不成熟，还在持续更新中。Three.js 它缺少游戏引擎和虚拟显示平台等系统中的一些常用功能，如，公告板（billboard）、头像（avatar）和物理引擎等[36]。另外，Three.js 并不支持任何与网络有关的操作。

2.6 本章小结

本章主要对与本课题的研究息息相关的几种关键技术进行了介绍及对比分析。首先，分析了几种主流的 Web 3D 技术的实施方案。其次，主要对 WebGL 技术、Three.js 引擎技术进行了介绍。最后，对比分析了几种现在主流的 WebGL 相关的引擎技术。

3 基于 WEBGL 实物交互的过程

Three.js 是 WebGL 的一个第三方引擎框架。虽然它是在 WebGL 的基础上进行的二次开发，并且本质上也属于 WebGL 的编码，但是二者在建模上还是有一定的差别的。

3.1 三维图形绘制方法

3.1.1 观察视角

在所有 3D 图形中，无一例外经常会涉及到四个参数：长宽比、视野、近平面、远平面。

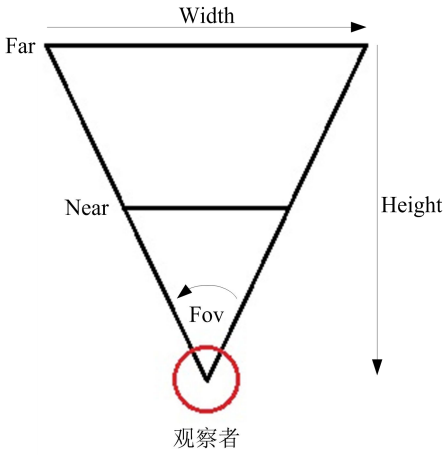


图 3-1 观察视角

图 3-1 中，height/width 表示长宽比；fov 表示观察者视野即视角；near 表示近平面；far 表示远平面。

3.1.2 绘图方法

通常，3D 图形的绘制有多种绘制方法，如顶点、多边形和网格。其中，网格（mesh），也被称为模型（model），是最常用的一种绘制方法。

网格是由一个或多个多边形（通常为三角形和四边形）组成的。三维空间中，

多边形各个顶点的坐标决定了多边形所在的位置。网格模型不仅仅只局限于使用 x，y，z 坐标所定义的形状，它还包括网格表面的其他特征，如颜色、纹理等等。这些网格的表面特征可以通过物体的属性来进行定义。

3.2 WebGL 的实物交互过程

WebGL 与浏览器、电脑硬件之间的交互过程如图 3-2 所示。

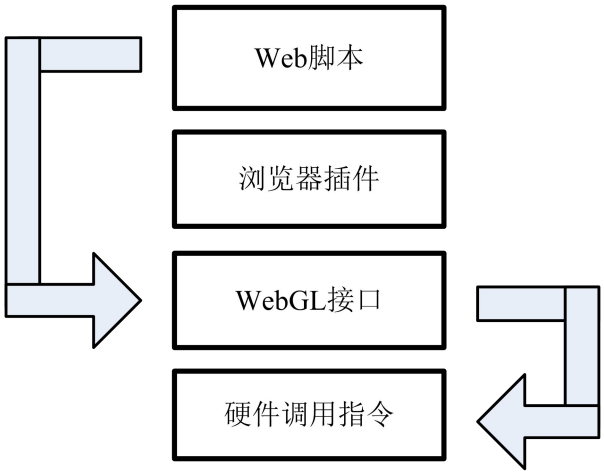


图 3-2　WebGL 与浏览器、电脑硬件之间的交互过程

实现绘制图形的关键就是对 Web 脚本进行编写，将三维物体转换到二维空间[37]中，并通过 WebGL 接口调用电脑硬件进行图形渲染。了解了 WebGL 与浏览器、电脑硬件之间的交互过程之后，还需要对 WebGL 在浏览器上的工作过程进行了解。

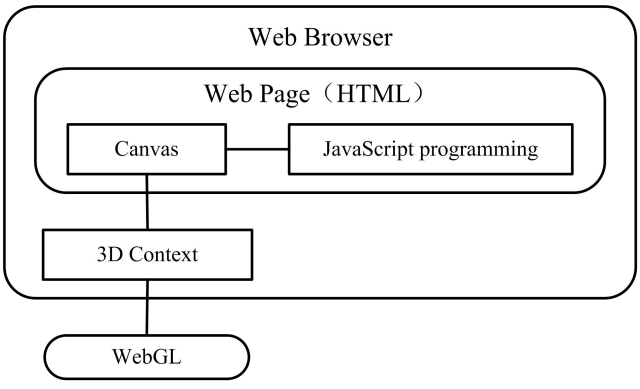


图 3-3　浏览器中 WebGL 关键元素之间的联系

如图 3-3 所示，Web 浏览器应用程序打开一个包含 canvas 元素和 javascript 程序的 Web 页面[38]，javascript 用于对 canvas 上的元素进行操纵，提取出其 3D Context 中的物体并将 3D 场景绘制到 canvas 中去。3D Context 提供了一个与硬件加速器 GLES2 API 的连接。所以只需要对 3D Context 进行设置，并用 JavaScript 进行编程就可完成图形的绘制了。

3.2.1 WebGL 的常用 API

在了解 WebGL 的工作原理之前，还需要对其最重要最常用的一些对象进行一些了解：

（1）WebGLContext：context 相当于是 WebGL 的一个 API，同时也控制着 WebGL 的状态机制。因此，在使用 WebGL API 之前，首先应该创建一个 context；同时，给 DOM 中的 Canvas 元素定义一个 WebGLRenderingContext 对象，用于管理 OpenGL 的状态以及渲染绘画环境。

（2）WebGLBuffer（VBO）：通过在 GPU 显存中创建一块 Buffer，来保存一些图形渲染时需要用到的顶点信息和初始数据。

（3）WebGLFrameBuffer（FBO）：即帧缓存对象，该对象用于将显示信息输出到程序的帧缓存上。

（4）WebGLRenderBuffer：用于渲染缓存，为 FBO 提供渲染绘画的目标 buffer。WebGLRenderBuffer 中包含具体的图像绘画数据。

（5）WebGLTexture：对图像纹理进行的一系列的操作，如：创建纹理、生产纹理数量、存储纹理索引等。

（6）WebGLProgram：编程对象可以被着色器所依附。编程对象提供一种详细的机制用于说明连接于该编程对象的着色对象。同时，它也提供一种检测着色器兼容性的方法（例如：检测顶点着色器和片元着色器的兼容性）。

（7）WebGLShader：即着色器，用于将图形信息转换为可在屏幕上显示的像素信息。该对象指明了在显示器上该如何绘画以及该画些什么。

（8）WebGLUniformLocation：该对象表示在着色程序中，uniform 变量的地址信息。

3.2.2 WebGL 可编程管线流程

WebGL 沿用了 OpenGL ES 2.0 的可编程管线流程。

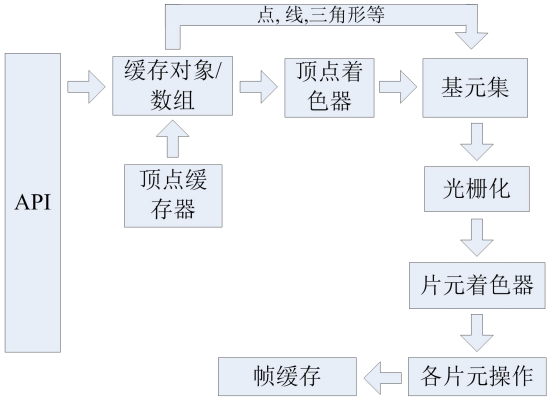


图 3-4　WebGL 可编程管线流程

图 3-4 描述了在 WebGL 中整个 API 的调用过程。这个示意图表达了可编程管线中数据流处理的本质：数据流先到达顶点处理器，然后经过片元处理器，最后到达帧缓存[39]。

WebGL 的可编程管线流程，是使用 WebGL 实现图形绘制的编程核心。对任何图形的绘制，都需要根据以上流程来进行操作。

在绘制图形之前，还需要在 HTML DOM 中，定义一个 Canvas 元素，并定义一个与其相关联的渲染环境。接下来，使用 JavaScript 编程来实现具体实物图形的绘制了。下面，结合图 3-4 对 WebGL 实现图形绘制的具体工作流程，来进行一个详细的阐述：

WebGL 的工作首先是从建立顶点数组（Vertex Array）开始的。通过 JavaScript 所建立的顶点数组中一般包含了顶点信息、颜色信息和光照信息等。然后，在顶点缓冲（Vertex Buffer）中存储这些顶点信息，并同时将信息传递给图形处理器（Graphic Processing Unit，即 GPU）。在将信息提交给 GPU 进行渲染工作的同时，还需要为其提供一个索引数组，用于控制顶点数组。这样，GPU 就可以根据索引数组的描述，

对顶点信息进行读取和处理了。接下来，光栅器（Rasterizer）会对需要绘制的图形进行光栅化处理，将当前的基元信息转换为片元。处理完成之后，片元着色器（Fragment Shader）会对每一个像素的颜色、深度、光照、纹理映射等进行输出。最后，帧缓冲器（Frame Buffer）会根据片元着色器所输出的值对图形进行绘制并输出。

3.2.3 着色器中数据处理过程

着色器是 3D 渲染中最重要也是最不可或缺的一部分，有了着色器对像素的处理，渲染器才知道如何对每个像素进行绘制。同样，在 WebGL 中，着色器必须被定义。有了着色器，开发者们才有能力去控制图形中的每一个顶点和像素，才能开发出栩栩如生的三维图形及动态效果。

着色器中需要自己编写的程序部分，涉及到了一些 GLSL 的编程规范中的数据类型。表 3-1 列举了会用到的矢量类型，表 3-2 和表 3-3 分别列举了 GLSL 的矩阵和纹理类型。

表 3-1　GLSL 向量类型

类　型	描　述
vec2、vec3、vec4	2D、3D 和 4D 的浮点向量
ivec2、ivec3、ivec4	2D、3D 和 4D 的整数向量
bvec2、bvec3、bvec4	2D、3D 和 4D 的布尔向量

表 3-2　GLSL 矩阵类型

类　型	描　述
mat2	2×2 矩阵
mat3	3×3 矩阵
mat4	4×4 矩阵

表 3-3　GLSL 纹理类型

类　型	描　述
sampler1D、sampler2D、sampler3D	1D、2D 和 3D 纹理处理
samplerCube	对立方体的纹理处理

另外，在着色器中还有几种常用到的基本变量类型：

（1）Attribute：经常变更的信息，传递到顶点着色器中的数据；

（2）Uniform：统一的信息，顶点着色器和片元着色器中的数据；

（3）Varying：从顶点着色器到片元着色器的数据[40]。

着色器是由顶点着色器和片元着色器两个部分所组成的。这两个部分各司其职，协作完成图形的渲染工作。

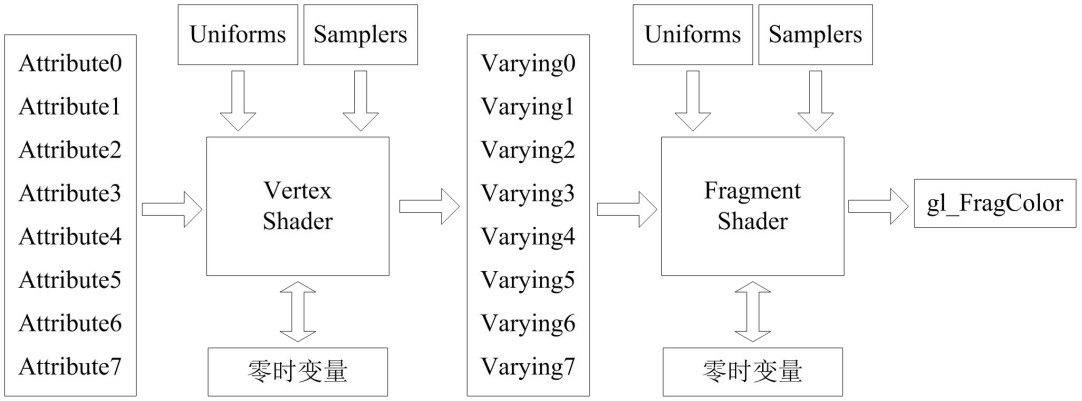


图 3-5　顶点着色器和片元着色器的数据处理流程

顶点着色器和片元着色器中数据处理的方式如图 3-5 所示。顶点着色器（vertex shader）负责利用视图矩阵和投影矩阵将图形的坐标信息转换到 2D 空间；片元着色器（fragment shader），又可以被称为像素着色器（pixel shader），负责生成并输出每个像素的颜色，这个颜色可以是纯色、材质或者纹理等等。

3.2.4　WebGL 实物交互原理

首先实现动画效果，然后对鼠标动作进行监听，根据鼠标动作调用动画效果，从而实现与实物模型之间的交互。

3D 模型的动画变换是指对已渲染模型的移动、缩放、旋转等操作。变换需要对模型 3D 网格的顶点位置进行变更。这通常是由矩阵来操作的，通过矩阵计算出顶点变换以后的位置。

由于 WebGL 是很低层的，它没有实现动画的方法可以直接被调用。因此，想要使用 WebGL 编码实现物体动画的话，就必须要通过矩阵变化来达到这一效果。

3.3 Three.js 的实物交互过程

3.3.1 Three.js 的常用 API

直接使用原生的 WebGL API 来编写代码是一项非常繁琐的工作。Three.js 在 WebGL 的基础上进行了二次开发，对原生 WebGL API 进行了封装和简化。

Three.js 常用的 API 包括：

（1） 渲染器：渲染是将三维图形映射到二维空间的过程，因此渲染器就是指执行渲染操作的代码或软件。

（2） 场景：场景是指一个映射在二维平面上的三维空间，它就相当于是一个容器，将要被显示的内容都需要添加到这个场景当中进行显示。

（3） 相机：在 WebGL 中，三维空间中的物体投影到二维空间的方式有：透视投影、正投影和复合投影。在 Three.js 中，相机就是用来指示三维物体在二维平面上的投影方式的。

（4） 光源：根据一个物体显示效果的不同，可以对场景追加一个或多个不同的光源。Three.js 中有四种光源可以设置，分别是点光源(Point Light)、聚光灯(Spot Light)、平行光源(Direction Light)和环境光(Ambient Light)。

（5） 物体：Three.js 中包含许多已封装好的物体，还包含许多用户可以随意进行设置的属性，如位置、材质、纹理等信息。

除了上述列举的常用 API 之外，Three.js 还提供许多其它丰富的 API 供各种绚丽的 3D 效果的绘制。

3.3.2 Three.js 的编程流程

使用 Three.js 进行实物图形的绘制，其原理与基于 WebGL 的图形绘制原理类似，只不过编程流程要相对精简一些。因为一些繁琐的过程 Three.js 的开发者们已经帮忙完成好了，直接调用就可以了。

如图 3-6 所示，Three.js 的编程流程很简单：首先，分别对渲染器、场景、光源、物体、相机进行初始化设置；然后，将光源和物体添加到场景中去；随后，将场景和相机添加到渲染器中。这样一帧画面就已经绘制好了，最后只需要用渲染器将这帧画面在屏幕上渲染出来就可以了。

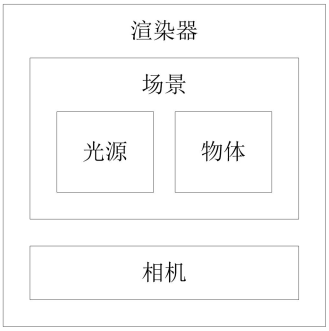


图 3-6　Three.js 的工作原理

三维空间中的实物图形绘制，最主要以及最重要的环节就是使用着色器对图形进行着色并渲染了。Three.js 除了继承了 WebGL 中的顶点着色器和片元着色器（像素着色器）之外，它自身还设置了一个几何着色器。几何着色器，顾名思义，就是对物体的几何元素进行操作，用于添加或删除物体表面几何元素的数量。也就是说，其用途之一就是制作不同 LOD 层级的模型[33]。为基于 Three.js 的物体添加 LOD 效果就是通过几何着色器完成的。

着色器为物体的着色方式又分为四种，分别是：平面着色、朗博着色、高氏着色和 Phong 着色。其中 Phong 着色又是 Three.js 为复杂实物图形着色的重要方式。根据顶点法线，Phong 着色可以通过计算得出每个像素的中间法线值从而为每个像素着色[33]。但是，采用 Phong 着色会使得处理器的开销很大。

3.3.3　Three.js 实物交互原理

使用 Three.js 来实现旋转动画的效果其原理和基于 WebGL 的旋转动画的实现原理是一样的。但是使用 Three.js 实现就容易多了，直接调用 Object.rotation.set()方法就可以了。

在 PC 机的浏览器上，最常见的交互就是实物物体与鼠标之间的交互了。这个交互过程需要用到点击检测，检测实物图形是否位于鼠标指针的下方。实际上，点击检测的过程恰好与实物图形绘制的过程是相反的。实物绘制是将一个三维空间中的实物物体其坐标转换到二维坐标系中进行绘制；而点击检测则是先将已鼠标坐标转换为视口空间坐标（原点在中央，x 轴和 y 轴的坐标范围是-0.5 到+0.5 的区间内），

然后再将视口坐标转换成 3D 空间中的一个点，以此点为原点发射一条射线，则任何与这条拾取射线相交的物体都被认为处于鼠标指针下方。

3.4 本章小结

本章分别从 WebGL 和 Three.js 的角度出发，针对实物交互的过程进行了阐述。本章第一小节描述了 3D 图形的绘制方法与观察视角。在本章第二小节重点研究了针对 WebGL 编程管线流程及 WebGL 两个着色器中的数据处理的过程。第三小节涉及了 Three.js 的工作流程之外，还与基于 WebGL 的实物图形的实现方法进行了分析对比，重点说明了其与基于 WebGL 的实现方法的不同之处。

4 基于 WebGL 的优化算法

由于 Three.js 只是从工作量上对 WebGL 进行了一个简化，性能上并没有得到提升。为了在使用 WebGL 建模时，模型的运行速度也能得到提升，将把 LOD 优化算法引用到基于 WebGL 的实物交互中。

4.1 优化算法选择

LOD(Level of detail)，即细节层次模型算法是一种有效的提高三维模型实时渲染速度的表达方式，它在虚拟现实、地理信息、计算机图形图像学、医学图像系统等领域有广泛的应用。现今在各个领域中，对需要构造和使用的各种实物交互模型的要求都越来越高了，而描绘出一个精细的模型不但是对计算机的存储容量、处理速度的一个巨大挑战，而且也已经成为了 3D 图形绘制、实时网络传输的一个难题了。为了解决这个难题，一般都会选择使用 LOD 模型简化算法来对实时渲染的 3D 模型进行简化。

一个复杂的 3D 模型却拥有很多个基元需要被操作，所以现有的绝大多数 GPU 都达不到要求[41]。当进行一个复杂 3D 模型的渲染时，其渲染速度及其渲染效果很难达到平衡。当要求渲染图形的效果清晰时，那么必然非常消耗 GPU 内存，相应的其渲染速度必定很慢；反之，若要求图形显示流畅时，那么图形的清晰度很有可能达不到要求。这时，细节层次（Levels of Detail，LOD）模型的使用意义就凸显出来了。使用 LOD 模型对当前 3D 渲染算法进行简化，既可以保证不影响画面效果，又可以保证在原有的基础上相对提高图形渲染速度。

在这里，基于 WebGL 的实物交互也存在同样的问题，为了使模型的运行性能得到提升，在此将 LOD 简化算法引入到 WebGL 的建模中。

4.2 细节层次模型优化思想

实际上，计算机实时的处理的能力远远低于真实场景的复杂程度。想要有效提

高图形渲染速度，就必须降低图形的复杂度以适应计算机的处理能力。LOD 模型就是根据这个核心思想来完成对图形的简化的。

LOD 模型，是由 Clarknl 于 1976 年提出来的[42]。他认为当物在屏幕上渲染区域较小时，完全可以使用较粗略的模型描述该物体，为了更快的对场景进行绘制，还给出了一个判断可见区域的算法[43]。LOD 模型一改以往人们对“图形质量越精细越好”的认知，它根据物体距离视点的距离、物体在屏幕上的投影区域等因素来实时描绘实物模型不同精细程度的层次模型[44]。在理论上，这是一种全新的建立 3D 模型的方法。

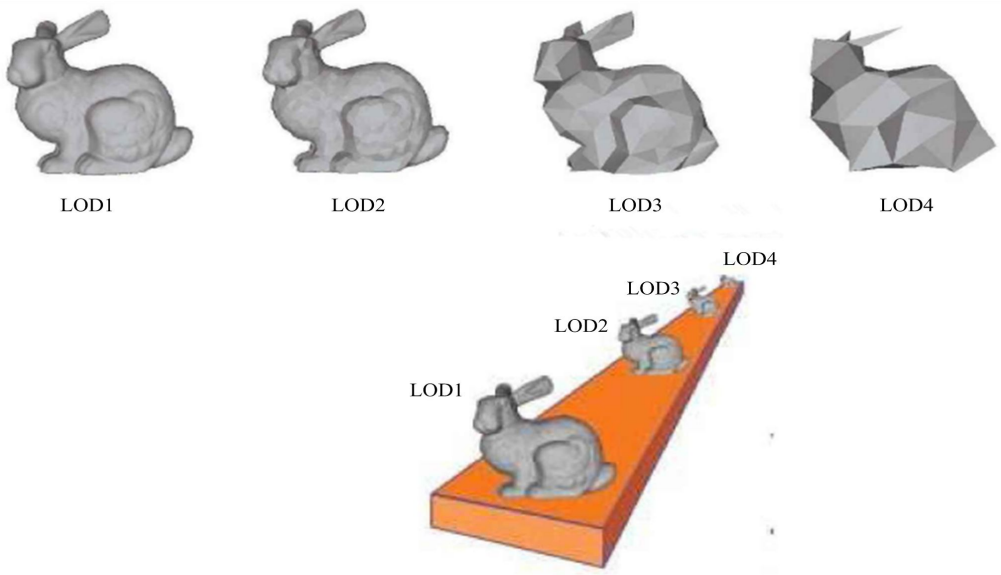


图 4-1　细节层次模型

如图 4-1 所示，当物体距离视点很近，或者物体在画面上投影区域越大，表明当前物体所占的像素较多，那么就需要详细描绘该物体；反之，则采用较粗糙的绘制。但需要注意的是，当对相邻两个层级之间的物体细节描绘差别较大时，可能会导致画面显示存在跳跃感，这就需要对相邻层级的图形之间做好过度，进行平滑处理，以消除不同层级之间模型进行切换时所存在的跳跃感。

4.3　细节层次模型优化方法

基于 LOD 算法的简化方法一般分为两种，即静态简化和动态简化。

4.3.1 静态简化算法

静态简化方法只与图形自身有关，它是指根据一定的精简原则，使用重构出的简单模型来描绘复杂模型。静态化简也可以满足构造多分辨率模型的需求，但是需要事先构造并存储多个近似分辨率的模型，而且当在不同分辨率的模型之间进行切换时，由于相邻两层模型表面之间的几何元素数目差别较大，会由此产生明显的跳跃的感。

静态简化方法有两大缺点：一、由于要存储多个近似分辨率的模型，所以会很耗费内存；二、由于使用的是新构建的简化模型来描绘原模型，所以不能恢复原模型的信息。

简单来讲，LOD 优化算法静态简化方法就是通过减少点、线或多边形等物体的细节描述，来降低图形的数据量和复杂度，以提高处理速度。接下来，将分别对 LOD 的几种静态简化方法进行探究：

（1）顶点聚类法（Vertex Clustering）：如图 4-2 所示，将物体表面分成许多个小格（小格尺寸小于用户规定的误差），在每个小格中，分别计算出一个平均顶点，并将同一小格中的其它顶点都合并到该顶点上，这样就对现有模型进行了简化。

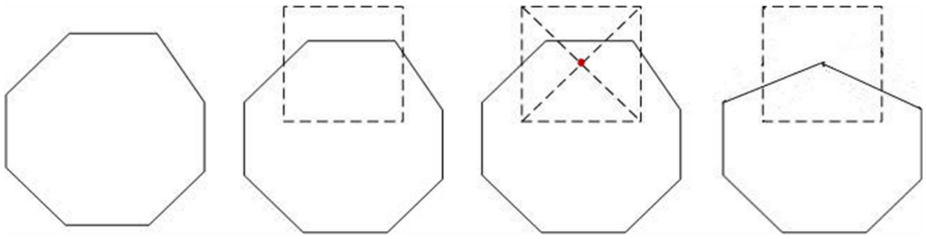


图 4-2　顶点聚类法

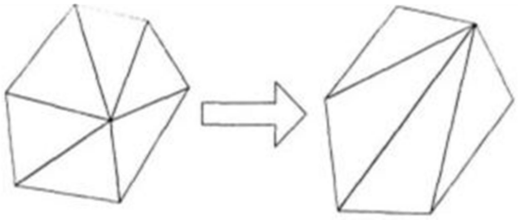
使用该方法进行编程实现非常简单且非常高效，并且对输入网格的拓扑结构没有特定要求。但是，使用该方法在简化过程中有可能导致模型表面的拓扑结构被修改。

（2）采样法（Sampling）：采样算法是指在原模型的表面重新选取采样点，然后三角化原模型表面上的新采样点并删除原顶点，根据简化后的网格，通过算法建立一个和原模型相似度最高的简化模型。

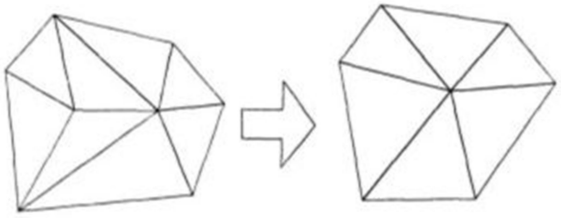
使用该方法进行简化，用户可以根据需求对采样点数量和局部顶点的密度进行

实时调整，以控制简化后近似模型的质量。但是，该方法不易获得质量较高的简化模型，且编程实现比较复杂。

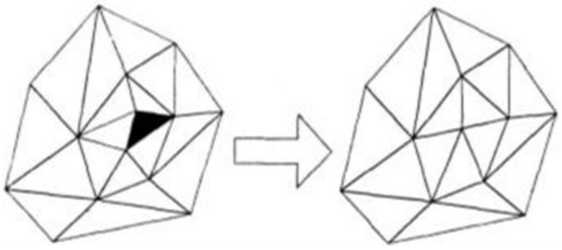
（3）删减法（Decimation）：如图 4-3 所示，主要通过对物体的几何单元删除来对物体局部进行一个更新操作及简化。删减法又包括对物体表面的顶点（如图 4-3（a）所示）、边（如图 4-3（b）所示）以及三角形（如图 4-3（c）所示）等的删减。



（a）点删除



（b）边删除



（c）三角形删除

图 4-3　删减法

使用该方法可以对模型表面冗余的几何元素进行删除，生成的简化模型的质量比较好。但是大幅度简化能力有限。

4.3.2　动态简化算法

动态简化是根据用户分辨率，实时生成近似模型，动态简化与用户视点等因素有关。其基本思想是在模型简化过程中，可以实时得到具有所需要的分辨率的近似

模型[45]。每个模型所需要简化的程度由模型之外的因素决定，例如视点、模型在屏幕上投影区域的大小。

动态简化是通过一些简单的局部的几何变换来实现的，它可以生成不同分辨率下的连续的近似模型。其最具代表性，也最常用的方法就是细分法。

细分法（Refinement）就是先选取一个网格作为基网格，然后基网格开始，逐步进行迭代细分。每细分一次，向模型添加一些细节，直到细节模型和原始模型之间的相似度满足用户规定的要求为止。但是细分法并不使用所有的情况，这类方法一般只适用于以下情况：一、基网格容易获取的情况下。因为对于多边形模型来说，创建基模型是一件很困难的事；二、基网格必须与原始模型保持同样的拓扑结构，这样一来，对简化幅度就有了很大的限制；三、生成的模型的底层细节一旦发生改变，会影响到高层上。

4.3.3 简化方法选取

考虑到本文中的实际情况：① 不需要构造太多近似分辨率的模型；② 层级模型与层级模型之间不能相互影响;③ 没有一个严格的有规律可循的拓扑结构;④ 编程开发易实现，且易维护。在后面的优化具体实施中将选择删减法来对原模型进行优化处理，同时构造多个近似分辨率模型。

4.4 本章小结

为了提升计算机对精细复杂模型的描绘速度，一般都会选择使用 LOD 模型简化算法来对实时渲染的 3D 模型进行简化。而对基于 WebGL 的实物模型描绘而言，也需要选择一种方法对其实时渲染速度进行优化。由此，本文尝试将 LOD 优化算法也引用到 WebGL 的编程中来。

本章第一节，描述了优化技术的选择过程及 LOD 算法被选择的原因，并对使用 LOD 算法的重要性及必要性予以说明；第二节，描述了在当前基于 WebGL 的实物模型的基础上使用 LOD 优化算法进行优化的主要思想；第三节，分析了 LOD 优化算法其包含的优化方法，其主要分为静态简化方法和动态简化方法，并分析了各种方法的优缺点，并选取了一种最适用于本文的优化处理方法。

5 基于 WebGL 实物交互的实现

本章实现了具体实物模型的建模，并截取部分主要实现代码对基于 WebGL 实物交互实现的具体实现过程予以说明。还使用了 LOD 算法对实物模型进行了优化，并还对实物模型实现优化前后，进行了测试、对比、分析。

5.1 开发环境

开发代码是由少量的 HTML5 代码和大量的 JavaScript 代码实现的，主要工作是编写 Web 脚本。所以基于 WebGL 实物交互开发环境为 Eclipse；调试及测试环境为：chrome 浏览器和 Tomcat 服务器。

5.2 实物交互的实现

5.2.1 基于 WebGL 实物交互的实现

在本小节中，通过编码实现了与实物模型的交互操作及动画效果。

图 5-1 展示了实物模型绘制完成并加入动画及鼠标交互后，最终实物模型渲染的结果。

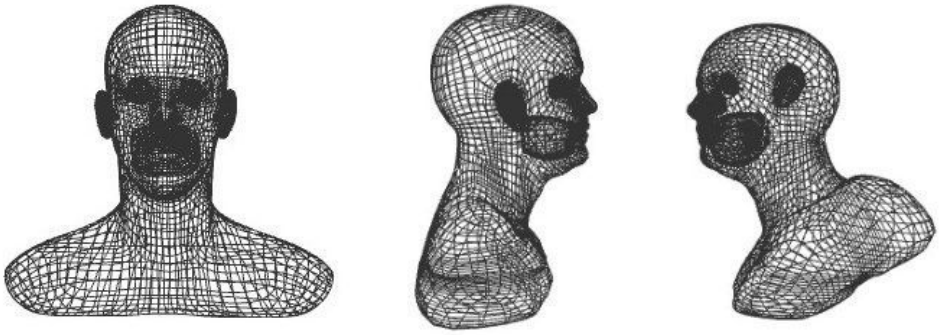


图 5-1 实物交互实现结果展示

用鼠标拖动图 5-1 中的实物模型，实物模型会实现旋转动画。下面以该项目实例为例，详细说明实物交互的实现过程。

第一步，先要绘制出实物模型，才能给模型添加交互功能。

1）基于 WebGL 实物图形的实现

（1）着色器的实现

在绘制图形之前，首先要对其着色器进行设置。着色器分为顶点着色器和片元着色器，对这两个着色器的程序需要分别进行编写。先来对顶点着色器（vertex shader）进行编写：

```
attribute vec3 vertexPosition;     //定义只读 3D 顶点向量存放顶点坐标
attribute vec3 vertexNormal;       //定义只读 3D 顶点向量存放法线向量

uniform mat4 mMatrix;          //定义统一的 4×4 矩阵作为模型视图矩阵
uniform mat4 pMatrix;          //定义统一的 4×4 矩阵作为投影变换矩阵

//定义一个 4D 浮点向量存放需要向片元着色器传递的数据
varying vec4 worldSpaceNormal;
//定义物体顶点坐标和法线向量的计算方法
void main() {
        gl_Position = pMatrix * mMatrix * vec4(vertexPosition, 1.0);
        worldSpaceNormal = mMatrix * vec4(vertexNormal, 0.0);
}
```

接下来，对片元着色器（vertex shader）进行编写，代码如下所示。

```
precision mediump float;     //声明为低精度画面
varying vec4 vWorldSpaceNormal;
//生成并输出每个像素的颜色
void main() {
        float diff = dot(vec3(0,0,1), vWorldSpaceNormal.xyz);
        diff *= 0.8;
        gl_FragColor = vec4(diff, diff, 0, 1);
}
```

可以直接将 GLSL 着色器的代码作为长字符串被引用。所以，在记事本中就可以编写着色器代码。

（2）图形绘制的实现

接下来，以绘制一个实物图形为例，并截取部分主要代码，来展示说明使用 WebGL 是如何进行图形绘制的。

首先，获取画布上下文环境并对当前视图进行初始化。

```
var c = document.getElementById('canvas');
//WebGL 目前还不成熟，参数只能设置为 experimental-webgl
var gl = c.getContext("experimental-webgl");
gl.viewport(0,0,c.width,c.height);        //初始化 WebGL 视图
gl.clearColor(0,0,0,1);            //用黑色清空背景
```

然后，定义缓存器，并对其进行初始化及绑定。

```
//创建一个顶点缓存器
this.vertexPosBuffer = gl.createBuffer();
//将 vertexPosBuffer 设定为接下来的操作所使用的缓存器
gl.bindBuffer(gl.ARRAY_BUFFER, this.vertexPosBuffer);
//使用 vertexPositions 列表创建 Float32Array 对象，并将其填充到顶点缓存器上
gl.bufferData(gl.ARRAY_BUFFER,  new  Float32Array(mesh.vertexPositions),
gl.STATIC_DRAW);
//创建一个索引缓存器
this.indexBuffer = gl.createBuffer();
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, this.indexBuffer);
gl.bufferData(gl.ELEMENT_ARRAY_BUFFER,  new  Uint16Array(mesh.indices),
gl.STATIC_DRAW);
```

最后，开始绘制图形。

```
gl.useProgram(program);       //设置着色器
//指定需要从中读取信息的缓存器
```

```
gl.enableVertexAttribArray(program.vertexPositionAttribute);
//设置着色器参数：点点坐标、模型矩阵、投影矩阵和视图矩阵
gl.vertexAttribPointer(program.vertexPositionAttribute, 3, gl.FLOAT, false, 0, 0);
this.setMatrixUniforms(program);
//绘制物体
gl.drawElements(gl.TRIANGLES, program.numindices, gl.UNSIGNED_SHORT, start *
2);
```

上述工作完成之后，调用 JSON 文件，将实物模型的材质、顶点坐标数组、顶点法线数组、顶点材质纬度等信息加载进来。根据顶点法线值对顶点进行相应的处理，根据材质信息，给每个像素添加显示效果。

2）基于 Three.js 实物图形的实现

使用 Three.js 来完成同样图形的绘制就方便多了。Three.js 有现成的 API 可供我们调用，具体实现过程如下：

（1）初始化渲染器

```
width = document.getElementById('c').clientWidth;
height = document.getElementById('c').clientHeight;
renderer = new THREE.WebGLRenderer({antialias: true});      //新建渲染器
renderer.setSize(width, height);       //设置视图尺寸
document.getElementById('c').appendChild(renderer.domElement);      //添加渲染器
renderer.setClearColorHex(0x000000, 1.0);       //设置画布背景色及透明度
```

（2）初始化场景

```
scene = new THREE.Scene();
```

（3）初始化相机

```
cameraInit = new THREE.PerspectiveCamera(45,width/height,1,10000);
cameraInit .up.set = (0,0,1);
cameraInit .position.set(400 , 400 , 400);
```

```
cameraInit .lookAt( {x:0, y:0, z:0 } );
```

（4）初始化灯光

```
DLight = new THREE.DirectionalLight(0xffffff, 1.0, 0);   //设定光源颜色
DLight .position.set( 50, 50, 200 );        //指定从该点到原点的方向
scene.add(DLight );
ALight = new THREE.AmbientLight(0x555555);
scene.add(ALight );
```

（5）物体初始化

```
yellowCube = new THREE.Mesh(new THREE.CubeGeometry(125,125,125),
new THREE.MeshLambertMaterial({color: 0xffff00, ambient: 0xffff00}) );
scene.add(yellowCube );
yellowCube .position.set(0,0,0);
```

对渲染器、场景、相机、灯光和物体初始化完成之后，将灯光、物体等添加到场景里面去，并将相机场景添加到渲染器中去即可。然后，同样加载保存了模型顶点和材质等信息的 JSON 文件。

另外，给模型添加投影，只需要以下四个步骤：

（1）在初始化渲染器时加入代码：

```
renderer.shadowMapEnabled = true;      //允许渲染映射投影
```

（2）在初始化灯光时加入代码：light.castShadow = true;

（3）在初始化物体时加入代码：

```
Object1.castShadow = true;      //允许物体 1 投影
Object2.receiveShadow = true;       //允许物体 2 接收物体 1 的投影
```

第二步，编写动画效果。

1）变换动画的实现

（1）基于 WebGL 动画的实现

在 WebGL 中，由于没有任何现成方法或 API 给我们调用，所以，物体的旋转

等变换的实现必须依靠自己来编码完成。

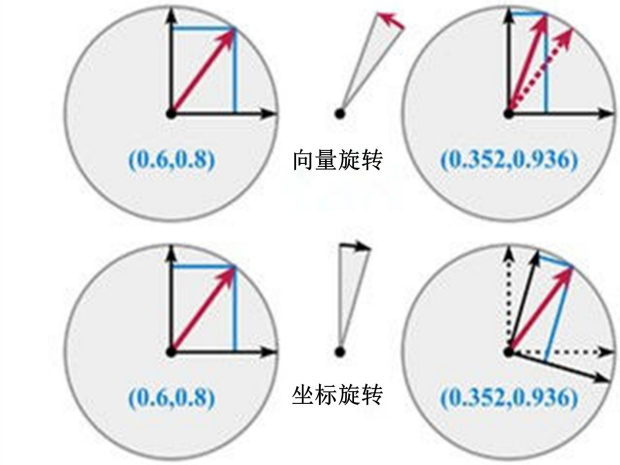


图 5-2　旋转原理

这里所说的旋转是相对于某一参照物的旋转。如图 5-2 所示，在三维空间中，某一物体旋转，可以理解为是物体相对于坐标系的旋转；也可以理解为坐标系相对于物体的旋转。

在线性代数中，旋转矩阵[46]是可以用于在欧式空间（Euclidean space）[47]执行旋转的矩阵。例如对于矩阵 R 来说：

$$R=\begin{bmatrix}\cos\theta & -\sin\theta\\ \sin\theta & \cos\theta\end{bmatrix} \tag{5-1}$$

若要将笛卡尔平面上的点（x，y）逆时针旋转θ度，就需要使用旋转矩阵 R 来完成该旋转操作。当使用旋转矩阵 R 进行旋转时，各点的位置必须用列向量 v 来表示，通过使用矩阵乘法 Rv 来得到旋转矢量。旋转矩阵是方阵，更具体的讲，旋转矩阵可表征为正交矩阵。旋转矩阵对旋转提供了一种代数描述，它被广泛地用于几何、物理和计算机图形的计算。

基本旋转指的是物体绕某一坐标轴所进行的旋转。以下三个矩阵分别表示在三维空间中绕 x，y 或 z 轴旋转角度θ的基本旋转矩阵：

$$R_x(\theta)=\begin{bmatrix}1 & 0 & 0\\ 0 & \cos\theta & -\sin\theta\\ 0 & \sin\theta & \cos\theta\end{bmatrix} \tag{5-2}$$

$$R_y(\theta)=\begin{bmatrix}\cos\theta & 0 & \sin\theta\\ 0 & 1 & 0\\ -\sin\theta & 0 & \cos\theta\end{bmatrix} \tag{5-3}$$

$$R_z(\theta)=\begin{bmatrix}\cos\theta & -\sin\theta & 0\\ \sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix} \tag{5-4}$$

在三维空间中，一般默认采用的是右手坐标系，即 X 轴指向右，Y 轴指向上，Z 轴指向前方。以上旋转矩阵很容易验证其正确性，例如，若将列向量（1，0，0）绕 Z 轴逆时针旋转 90°，该向量就变为（0,1,0），如图 5-3 所示。

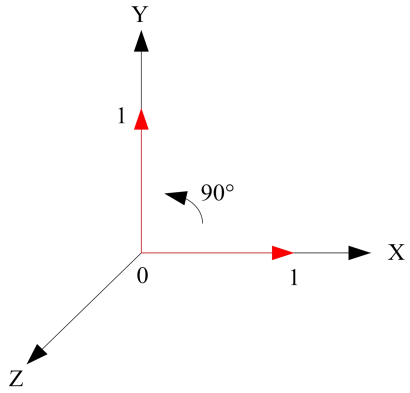


图 5-3　矢量旋转示例

$$R_z(\theta)\begin{bmatrix}1\\0\\0\end{bmatrix}=\begin{bmatrix}\cos90° & -\sin90° & 0\\ \sin90° & \cos90° & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1\\0\\0\end{bmatrix}=\begin{bmatrix}0 & -1 & 0\\ 1 & 0 & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1\\0\\0\end{bmatrix}=\begin{bmatrix}0\\1\\0\end{bmatrix} \tag{5-5}$$

其它旋转矩阵都可以通过这三个基本旋转矩阵相乘获得。

若从角度和坐标轴变量来描述旋转矩阵的话，可以使物体实现绕给定坐标轴旋转。

首先，给定一个单位向量 $u=(u_x，u_y，u_z)$，其中 $u_x^2+u_y^2+u_z^2=1$，矩阵绕 u 向量所在的方向逆时针旋转θ角：

$$R=\begin{bmatrix}\cos\theta-u_x^2(1-\cos\theta) & u_xu_y(1-\cos\theta)-u_z\sin\theta & u_xu_z(1-\cos\theta)+u_y\sin\theta\\ u_yu_x(1-\cos\theta)+u_z\sin\theta & \cos\theta+u_y^2(1-\cos\theta) & u_yu_z(1-\cos\theta)-u_x\sin\theta\\ u_zu_x(1-\cos\theta)-u_y\sin\theta & u_zu_y(1-\cos\theta)+u_x\sin\theta & \cos\theta+u_z^2(1-\cos\theta)\end{bmatrix} \tag{5-6}$$

将公式 5-6，用代码进行实现。其具体实现代码如下，代码中涉及四个变量即：angle、x、y、z。其中 angle 表示物体逆时针方向的旋转角度，x、y、z 表示一个向量。

```
//首先将（x，y，z）向量转化为单位向量
var invlen = 1 / Math.sqrt(x*x+y*y+z*z);
var n = { x : invlen * x, y : invlen * y, z : invlen * z };

var s = Math.sin(angle);
var c = Math.cos(angle);
var t = 1 - c;
//变换矩阵实现
this.d[0] = t*n.x*n.x+c;
this.d[1] = t*n.x*n.y + s*n.z;
this.d[2] = t*n.x*n.z - s*n.y;
this.d[3] = t*n.x*n.y - s*n.z;
this.d[4] = t*n.y*n.y + c;
this.d[5] = t*n.y*n.z + s*n.x;
this.d[6] = t*n.x*n.z + s*n.y;
this.d[7] = t*n.y*n.z - s*n.x;
this.d[8] = t*n.z*n.z + c;
```

根据旋转矩阵操作旋转原理，旋转矩阵与物体初始坐标系相乘得出物体旋转后坐标系，这里 m 表示物体原始坐标系，也就是说要完成物体旋转操作（也即完成坐标系的迁移），只需要完成公式 5-7 的矩阵乘法 Rm 即可，在此给出计算公式，如公式 5-7 所示，代码就不一一给出了。

$$Rm=\begin{bmatrix} this.d[0] & this.d[3] & this.d[6] \\ this.d[1] & this.d[4] & this.d[7] \\ this.d[2] & this.d[5] & this.d[8] \end{bmatrix}\begin{bmatrix} m.d[0] & m.d[3] & m.d[6] \\ m.d[1] & m.d[4] & m.d[7] \\ m.d[2] & m.d[5] & m.d[8] \end{bmatrix} \quad (5\text{-}7)$$

在旋转变换的基础上实现位移变换很简单，例如，若要对一个矢量（x，y，z）进行一个（x’，y’，z’）的位移操作，那么把两个矢量相加即可。那么同样的道理，若要计算物体的旋转位移，那么首先得计算出进行旋转操作后的坐标系，也即物体相对于初始位置旋转后的位置，然后再加上位移向量。

2）基于 Three.js 动画的实现

直接调用 Object.rotation.set()方法实现旋转，在旋转编程实现完成之后，直接在绘制某帧画面时，调用 requestAnimationFram()函数，进行循环重绘实现动画，代码如下：

```
//加入物体旋转动画
cube.rotation.set(0,0,t/100);          //设定绕 z 轴旋转及物体旋转速度
renderer.clear();
renderer.render(scene, camera);
window.requestAnimationFrame(loop);          //调用动画
```

第三步，调用鼠标动作实现交互操作。

在 Three.js 中，提供了 THREE.Projector()方法，用于将鼠标坐标转换为视口空间坐标；提供了 THREE.Ray()方法来找出位于鼠标指针下方的物体，具体实现过程如下所示：

（1）把鼠标坐标转换为视口空间坐标

```
var x = ((pagex-offset.left)/this.container.offsetWidth)*2-1;
var y = -((pagey-offset.top)/this.container.offsetHeight)*2+1;
var vector = new THREE.Vector(x,y,0.5);
this.projector.unprojectVector(vector,this.camera);
```

（2）判断物体是否处于鼠标指针下

```
var ray = new THREE.Ray(this.camera.position,
vector.subSelf(this.camera.position).normalize());
```

var intersects = ray.intersectScene(this.scene);

此时，对 intersects.length 的值进行判定。当 intersects.length 的值大于 1 时，即表明当前鼠标指针下有物体，可以对其进行相应的操作；反之，则没有。

然后需要对轨迹球进行调用 THREE.TrackballControls()，对鼠标动作进行控制，这个函数里面已经封装好了鼠标动作，可以直接进行调用：鼠标左击表示物体旋转、右击表示物体平移、滚轮表示视图远近。

最后可通过事件监听器给渲染器渲染的对象绑定一个监听事件来监听鼠标动作 renderer.domElement.addEventListener()，同时通过该监听事件可以添加 mousemove，mousedown，mouseup 等鼠标动作。

5.2.2 基于 LOD 优化算法的实现

在第 4 章中已经对 LOD 算法做出了详细描述，并在 3.2.2 小节中对 Three.js 的几何着色器作了简要介绍。下面，在 Three.js 绘制的实物图形的基础上，调用其几何着色器，并结合 LOD 算法对图形绘制程序进行优化。

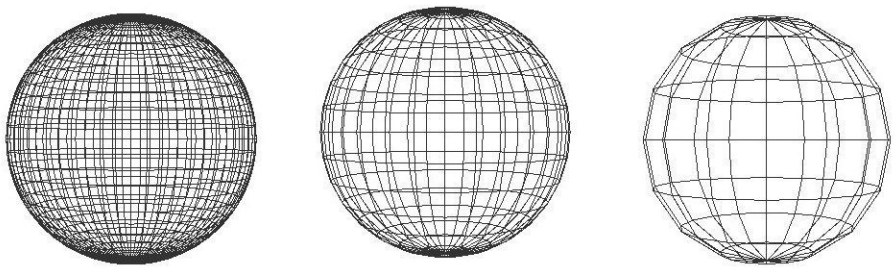


图 5-4 使用删除法优化后的效果

处理完成之后，实物模型效果如图 5-4 所示。这里为了能更明显的表现对图形进行优化处理前后的效果，物体纹理采用网格进行渲染。以简单球体为例，首先对图形使用删减法，对模型表面的几何元素进行删减。模型表面所有的几何元素，如顶点、面等，都保存在一个数组当中，直接对数组进行操控即可。

然后，把上面这三个实物模型分别作为 LOD1 层、LOD2 层、LOD3 层，来建立 LOD 模型。实验中，相机位置设置为：

Camera.position.set(400*Math.cos(t/100),400*Math.sin(t/200),400*Math.cos(t/50))。

计算矢量$(\cos(t/100),\sin(t/200),\cos(t/50))$距离原点(0,0,0)的距离 S：

$$S=\sqrt{\cos^2(t/100)+\sin^2(t/200)+\cos^2(t/50)} \quad (5\text{-}8)$$

在 Matlab 中根据公式 5-8 计算出 S 的最大值 S_{max} 和最小值 S_{min}，$D=S_{max}-S_{min}$。将 S 分为三段，分别为：S_{min}~（S_{min}+D/3），（S_{min}+D/3）~（S_{min}+2D/3），（S_{min}+2D/3）~S_{max}。当相机位于距离原点最近的那段距离（S_{min}~S_{min}+D/3），即模型离视点最近时，使用 LOD1 的模型；当相机位于距离原点最远的那段距离（（S_{min}+2D/3）~S_{max}），即模型离视点最远时，使用 LOD3 的模型；其它处于中间距离（2S/3-S）时，使用 LOD2 的模型，如图 5-5 所示。

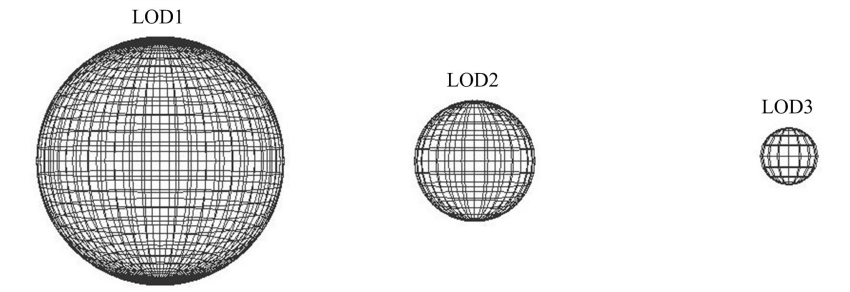


图 5-5　LOD 模型

从上图可以明显看出，虽然对实物图形进行了一些几何元素的删减，但是并不影响其视觉效果。

5.3　实现结果分析

5.3.1　WebGL 实物交互实现结果分析

在小节之前的内容，分别用 WebGL 和 Three.js 实现了实物模型的建模与交互操作。

WebGL 尚不成熟，还在发展阶段，所以它也存在一些缺陷。对于开发人员来说，WebGL 最大的缺陷就是使用其原生 API 进行图形绘制以及实现交互操作，其过程相当的复杂和繁琐。在这里，选用了第三方引擎技术——Three.js 作为 WebGL 的实施

框架，对基于 WebGL 实物交互的编程工作进行简化。

Three.js 是在 WebGL 的基础上进行二次开发而产生的，实际上其本质和 WebGL 是一样的，只不过用其来实现实物交互，一些原本需要自己编程完成的东西别人已经完成了，只需要直接调用就可以了。使用 Three.js 其达到的效果、运行的速度等等一些画面质量因素和使用 WebGL 都是一样的。

为了从运行性能方面对实物模型进行进一步的优化，将 LOD 算法引用到这里来，实现了对实物模型的优化。下面将对使用 LOD 优化算法优化前后的模型进行测试，并对测试结果进行了对比分析。

5.3.2 优化算法实现结果分析

1）测试结果分析与对比

在接下来的内容中，主要从实物图形渲染的几何元素的数目和画面的流畅性（即运行速度）出发，粗略的对使用优化算法处理前后的实物模型的渲染进行测试、对比和分析。

（1）几何元素数目

首先，对 LOD1、LOD2、LOD3 三个层级模型的几何元素数目进行了统计，结果如表 5-1 所示。

表 5-1 几何元素数目统计结果

	顶点数目	平面数目
LOD1	10201	10000
LOD2	1089	1024
LOD3	121	100

对上表进行分析，可以看出从 LOD1 层级的模型到 LOD2 层级的模型再到 LOD3 层级的模型，渲染实物图形时，需要渲染的顶点数目和平面数目明显减少了。这就表明渲染器渲染 LOD3 层级模型的工作量明显要比渲染 LOD1 层级模型的工作量要少很多。那么，可以推断：层级 LOD3 模型的渲染速度肯定会比层级 LOD1 模型的渲染速度要快。为了证明该推断，下面从画面的流畅性（即运行速度）出发，对优化实验前后实物模型的 FPS（fram per second，帧/秒）分别进行了测试。

（2）画面流畅性

由于实际环境有限，所以只能从众多采样数据中抽取一些具有代表性的数据来进行说明，测试结果如表 5-2 所示。

表 5-2　LOD 模型 FPS 值抽样数据

	采样数据（单位：FPS）									
LOD1	50	54	52	49	45	44	46	53	49	44
	51	43	45	51	46	43	45	50	55	51
LOD2	57	58	60	58	60	59	60	59	60	58
	60	57	52	59	56	60	58	59	58	60
LOD3	59	58	59	55	60	59	58	60	59	58
	60	59	60	58	59	60	60	60	60	59

虽然上表中通过测试所得出的实验数据比较乱，但是，已经已经隐约可以从上表中看出一些规律了，下面先对上表中的数据进行一个统计整理，如表 5-3 所示。

表 5-3　LOD 模型 FPS 值统计结果

	最大值（单位：FPS）	最小值（单位：FPS）	平均值（单位：FPS）
LOD1	54	43	48.3
LOD2	60	52	58.4
LOD3	60	55	59

将上述表格中的数据进行对比分析，可以看出 LOD1 层级模型运行的 FPS 值明显要小于 LOD3 层级模型运行的 FPS 值。LOD3 层级模型的运行流程性要相对好一些，这就很好的证明了之前所推断的内容。

以上是对 LOD1、LOD2、LOD3 三个层级的实物模型渲染分别作出的对比与分析。经过对比分析之后，发现其在需要渲染的几何元素数目和运行流程性上，确实存在着一定的差异。如上一小节图 5-5 中所示，将 LOD1、LOD2、LOD3 三个层级放在一个模型中，这样建立的最终的 LOD 模型与原模型之间也一定存在着差异。

所以，最终的 LOD 模型，即使用 LOD 算法实现优化后的模型，也对其 FPS 值进行了抽样，并选取了具有代表性的 20 组数据，分别为：60、59、49、59、55、48、59、60、54、52、59、54、45、60、52、57、45、59、46、57。接下来，分别对原模型和使用 LOD 算法实现优化以后的模型的 FPS 值进行抽样、统计，并进行对比分析。

表 5-4　FPS 值抽样数据统计结果

	最大值（单位：FPS）	最小值（单位：FPS）	平均值（单位：FPS）
实现优化前	54	43	48.3
实现优化后	60	45	54.45

从表 5-4 中可以看出，虽然由于人眼的视觉暂留功能，原模型和使用 LOD 算法实现优化后模型的运行速度差异人眼没办法识别，但是使用 LOD 算法优化后的模型的运行速度比原模型要快这一事实是不能被否认的。

原本基于 WebGL 的 3D 图形在运行流程性上就有很大的优势，在其基础上再使用 LOD 算法进行优化，为来 WebGL 一定会有更大的发展空间！

2）缺陷说明

在本实验中，还有一些问题和缺陷没有解决。

对优化前后实物图形渲染的测试，不仅仅与优化效果有关，还与实际测试环境有关。由于实际测试环境所导致的一系列客观原因（如：CPU、浏览器、其他进程等）的影响，测试结果存在一些误差是必然的。本文中的测试是在尽可能保持优化处理前后测试环境一致的前提下完成的。这样即使存在误差，测试结果的误差区间也会尽可能的保持一致。

另外，在本实验中没有对所建立的 LOD 模型进行平滑处理，所以在运行时画面会有明显的跳跃感。

5.4　本章小结

本章主要描述了实物图形绘制以及实物交互的具体实现过程。首先，本章第一

小节介绍了实际开发需要的开发环境；然后，在本章第二小节实现了基于 WebGL 相关技术的实物模型的建模，并在建模完成后的模型上添加了交互动作及动画效果，还使用 LOD 优化算法对实物模型实现了优化；第三小节对使用 LOD 算法实现优化前后的模型进行了测试，并对测试结果进行了对比分析。

6 总结与展望

6.1 全文总结

在基于 WebGL 的实物交互实例的开发中，主要运用了互联网领域一项很新颖的技术——WebGL。不得不说，3D 是很有吸引力的一种技术，在学习中就会情不自禁的投入进去，每当有一些进展都会给你进一步研究下去的动力，因为每一点进步都可以在页面中很好的体现出来。不论是对算法改进而使得动画 fps 的提升、对系统资源消耗的降低，还是场景上更好的表现力、更方便的人机交互方式，都能让我感受到 Web 3D 技术的魅力。

当前基于课题研究的建模，是带有研究性、验证性和实验性的开发，是为了研究 WebGL 技术在实物交互上的应用而进行的开发。经过实际的研究及开发，发现 WebGL 不愧为当下 Web 3D 技术的主要实施方案，其精致程度、逼真程度及运行的流畅性都很好的证明了这一点。

在对本课题的研究中，主要完成了一下工作：

（1）理论研究：首先，查阅各种资料文献，对本文中所用到的关键技术和算法进行了了解。然后，从多个角度出发，对 WebGL 和 Three.js 实现绘图的原理以及其它相关理论基础知识进行了研究；

（2）3D 实物绘制：分别使用 WebGL 和 Three.js 完成了 3D 实物图形的建模。在 WebGL 绘制的图形中，由于需要使用矩阵变换算法实现动画，遂将旋转矩阵引用于此；

（3）LOD 算法实现：分析了在建模中使用 LOD 算法的方法，并选取了 LOD 算法中的删减法建立 LOD 模型实现优化；

（4）测试及对比分析：在完成上述所有工作之后，分别对模型优化前后进行了测试并对测试结果从模型几何元素数目及模型运行流畅性两方面进行了分析对比。

对于本课题的研究中的建模，主要是基于 WebGL 这一先进技术完成的。这一技术的发展时间还不是很长，而且其自身也还处于一个尚不成熟的初级发展阶段。

所以，在对本课题研究开发的过程中遇到了很多问题，也存在着许多的缺陷和不足。

（1）环境限制

对复杂精致的 3D 图形的渲染是一项非常浩大的工作。其对电脑 CPU 处理速度和显卡的要求都是很高的。在实际操作的电脑上，这些客观环境要求都还达不到，因此对于一些国外研究学者制作的优秀的实例，没办法运行观赏，而单靠查看代码，其学习效率是很低下的。

（2）理论基础知识不够扎实

在研究过程中，发现入门很快，但是要想真正掌握就没那么容易了，导致理论基础知识薄弱的原因主要有三个方面的原因：一、不仅时间很紧凑，而且 WebGL 是属于最底层的编程，用其来进行编程是一项非常复杂繁琐的工作；二、用 WebGL 实现精致的实物图形绘制并且为图形添加各种交互功能不仅需要深入掌握 WebGL，还需要了解图形图像学以及高等数学知识；三、国内有关于该方面研究的资料文献比较少，且比较单一。

（3）实物模型难以获取

在对一个物体进行编程绘制之前，首先得将物体在三维坐标系中进行绘制。而想要绘制一个真实逼真的物体，其顶点数量是非常之多的，可能至少有上万个。在实际操作中，不可能在三维坐标系中将模型的每一个顶点坐标都计算出来。所以，只能借助于第三方的三维模型制作软件或者去网上找一些现成的坐标模板。

（4）对代码的优化比较片面

在使用 LOD 算法对代码进行优化时，只选取了其删减法进行了优化，而实际中，还有许多其它好的优化方法。并且在对模型优化时，没有进行平滑处理，画面之间切换时有明显的跳跃感。

在对本课题研究的前期，看到网上许许多多基于 WebGL 的精致绚丽的作品，不由得让我对 WebGL 的 3D 实物实现进行了很多的畅想和规划。但是在实际进行了部分编码之后，发现 WebGL 很深奥，要想真正的琢磨透彻仅仅靠这么短暂的时间是完全不可能的。因此，在实际编码的过程中遇到了许多困难，并且还有因为某些克服不了的困难而中途放弃开发的功能。即便如此，我还是因此而收获了很多。

6.2 展望

WebGL 目前在国内的发展尚在起步阶段，这样一个前沿的新兴技术对我有着致命的吸引力。虽然，最终对该课题的研究还是完成了，但是本文的研究内容，还只是基于 WebGL 实物交互的冰山一角。在以后的时间里，我也还会将我对 WebGL 的学习热情保持下去。就到目前为止的研究工作而言，还需要我进一步研究和完善的还有以下内容：

（1）对使用 LOD 算法优化后的模型进行平滑处理；

（2）深入研究 LOD 算法在 WebGL 中的运用；

（3）研究移动终端上基于 WebGL 实物交互技术的应用及实现。

目前，基于 WebGL 的实物交互，已经被广泛应用到了游戏娱乐、电视电影、教育教学、模拟分析等领域。对于 WebGL 这样一个尚不成熟的技术来说，这已经是相当了不起的成绩了。未来，随着对该技术的不断完善，基于 WebGL 的实物交互还极有可能被应用于军事、医疗、家电等领域。依照这样的发展趋势来看，相信未来的实物交互应用领域一定是 WebGL 的天下！

致 谢

终于，对该课题的研究接近尾声了，这同时也预示着两年多研究生的学习生涯也要画上句号了。在这两年多的时间里，我学到了很多、积累了很多、成长了很多，同时在技术上也有了很大的提升。这也为我完成本论文奠定了良好的基础。回首这两年多的时候，我需要对太多人表示感谢了。

首先，我要感谢我的导师——裴小兵老师。在裴老师悉心的指导下，本论文经过选题、开题、撰写、修改、定稿才得以完成。裴老师常年从事于互联网行业的前沿科学研究，他严谨的治学作风、敏锐的科学洞察力和忘我的工作精神令人尊敬和敬佩。裴老师在这两年多的时间里为我提供了良好的科学实验环境和项目实践平台，并且一直对我进行着孜孜不倦的教诲，才使得我有今天巨大的进步。这一切都使我终生难忘。借此论文完成之际，我要裴老师表达我最诚挚的谢意和祝福。

同时，我还要感谢长期和我一起学习奋斗的赵岚同学。赵岚同学在我为论文进行前期准备工作期间，给我提供了很大的帮助。并且，在我平时的学习生活中，赵岚同学给予了我很大的鼓励，正式这种鼓励给了我前进的动力。

其次，还要对李宜彬同学、胡晓赟同学、黄江玮同学以及丁永康学弟表示感谢，感谢他们在我撰写论文遇到困难时给我提供的帮助。

最后，再一次对所有曾经帮助过我的人表示最衷心的感谢！

参考文献

[1] 陈雅茜. 交互式桌面及其相关技术研究. 计算机工程与应用, 2012, 48(32): 147-152

[2] 张小瑞, 孙伟, 宋爱国. 虚拟物体力/触觉交互算法的研究进展. 系统仿真学报, 2011, 23(4): 637-647

[3] Keiichi Sato, Youn-kyung Lim. Physical Interaction and Multi-Aspect Representation for Information Intensive Environments. In: Proceedings of the 2000 IEEE International Workshop on Robot and Human Interactive Communication, Japan, 2000: 436-443

[4] 殷周平, 吴勇. 基于 WebGL 和 AJAX 的 WEB3D 应用研究. 安庆师范学院学报(自然科学版), 2013, 19(1): 58-61

[5] 陆凌牛. HTML5 开发精要与实例详解. 北京: 机械工业出版社, 2012: 100-485

[6] Konstantinos Moustakas, Michael G. Strintzis, DimitriosTzovaras, et al. Masterpiece: Physical Interaction and 3D Content-Based Search in VR Applications. In: Institute for Infocomm Research, 2006: 92-100

[7] 姜岩. 虚拟现实技术在 Internet 上的应用: [硕士学位论文]. 沈阳: 沈阳工业大学图书馆, 2002

[8] 金岳辉, 熊潇, 吴祯. 虚拟化技术在绿色机房建设中的应用. 中国金融电脑, 2011, 3: 80-82

[9] 朱晓菊, 张礼坚. 实物虚拟化——交互设计的未来. 艺术与设计(理论), 2009, 6: 205-207

[10] 杨炳祥. 实时三维漫游系统中关键技术研究与实现: [硕士学位论文]. 西安: 西安电子科技大学图书馆, 2009

[11] 龚江林, 刘廷章, 谢雁. 快速成型过程的实时仿真动画技术研究. 设计与研究, 2005, 1: 7-9

[12] 徐良军, 章建, 蒋毅等. 基于虚拟现实技术的电力安监仿真培训与考试系统研发. 计算机系统应用, 2010, 19(11): 162-165

[13] 许微. 虚拟现实技术的国内外研究现状与发展. 现代商贸工业, 2009, 2: 279-280

[14] 罗为君. 基于 XNA 的虚拟现实三维引擎设计与实现: [硕士学位论文]. 长沙: 湖南大学图书馆, 2010

[15] 朱连毅. 基于虚拟现实的火电厂事故人员培训系统设计: [硕士学位论文]. 哈尔滨: 哈尔滨工程大学图书馆, 2009

[16] 李文. 基于 EON 的虚拟现实技术在建筑设计中的应用研究: [硕士学位论文]. 西安: 西安建筑科技大学图书馆, 2012

[17] 牟长军, 童志伟. 综述虚拟现实技术及其应用. 装备制造技术, 2007, 1: 64-72

[18] 欧阳洪原, 孙敬杰. 基于HTML5的三维地理信息平台环境搭建技术探讨. 测绘, 2013, 36(3): 102-103

[19] 苏雪. 基于 WebGL 标准的家庭娱乐终端上 Web 3D 渲染的实现. 长江大学学报(自然科学版), 2011, 8(12): 102-104

[20] 程新丽. 基于 WebGL 的虚拟实验教学研究. 科技经济市场, 2011, 9: 7-9

[21] Bijin Chen, Zhiqi Xu. A Framework for Browser-based Multiplayer Online Games using WebGL and WebSocket. In: Multimedia Technology (ICMT), 2011: 471-471

[22] 王维敏. Web3D技术探索及几种Web3D技术的比较选择: [硕士学位论文]. 武汉: 武汉大学图书馆, 2004

[23] Limin Chen, Chaoyun Peng, Jianping Zhang. The Study Of Anchored Instruction's Application Based On Web3D. In: 2010 2nd International Conference on Education Technology and Computer, 2010, 4: 526-529

[24] 韩义. Web3D 及 Web 三维可视化新发展——以 WebGL 和 O3D 为例. 科技广场, 2010, 5: 86-90

[25] Ma Yadi, Wu Zheng, Zhang Zhiqiang, et al. Web3D Technologies and Motion Estimation in Education. In: 2008 International Workshop on Education

Technology and Training & 2008 International Workshop on Geoscience and Remote Sensing, 2008: 69-72

[26] Yuhui Yang, Jianping Zhang. Web3D technology and its application in the cultural relic protection. In: Institute for Electrical and Electronic Engineers, 2009: 10-13

[27] Yanxing OU, Ruomei Wang, Guanwei Xiao. Design and Implementation of Virtual Prototype Apartment Based on Web3D. In: 2012 Fourth International Conference on Digital Home, 2012: 161-166

[28] 陈煜, 林玮. Web3D 引擎中三维图形对象拾取的算法与实现. 工程图学学报, 2011, 6: 82-88

[29] 李统乾, 刘凤荣. 网络三维交互技术(Web3D)概述. 科技信息, 2010, 1: 45-46

[30] Kostas Kapetanakis, Spyros Panagiotakis. Evaluation of Techniques for web 3D Graphics Animation on Portable Devices. In: 2012 International Conference on Telecommunications and Multimedia (TEMU), 2012: 152-157

[31] Eero Aho, Kimmo Kuusilinna, Tomi Aarnio, et al. Towards Real-time Applications in Mobile Web Browsers. In: TIVIT Cloud Software Program, 2012: 57-66

[32] Lei Feng, Chaoliang Wang, Chuanrong Li, et al. A Research for 3D WebGIS Based on WebGL. In: 2011 International Conference on Computer Science and Network Technology, 2011: 348-351

[33] Williams J L. HTML5 游戏开发实践指南. 黄敏译. 北京: 机械工业出版社, 2012: 67-112

[34] Dorde Golubovic, Goran Miljkovic, Svetozar Miucin, et al. WebGL Implementation in WebKit based web browser on Android platform. In: 19th Telecommunications forum TELFOR 2011, 2011: 1139-1142

[35] 秀野堂主, 蒋宇捷, 罗睿. 论道 HTML5. 北京: 人民邮电出版社, 2012: 108-297

[36] Tony Parisi. WebGL 入门指南. 郝稼力译. 北京: 人民邮电出版社, 2013: 3-82

[37] Yutaka J. Kanayama, Gary W. Krahn. Theory of Two-Dimensional Transformations. In: IEEE Transactions on Robotics and Automation, 1998, 14(5): 827-843

[38] Maged N Kamel Boulos, Jeffrey Warren, Jianya Gong, et al. Web GIS in Practice VIII: HTML5 and the Canvas Element for Interactive Online Mapping. In: International Journal of Health Geographics, 2010: 1-13

[39] 刘运增. 互联网上的三维技术: Web 3D. 计算机与网络, 2003(10): 32- 35

[40] 谭文文, 丁世勇, 李桂英. 基于 WebGL 和 HTML5 的网页 3D 动画的设计与实现. 电脑知识与技术, 2011, 7: 6981-6983

[41] Chao Peng, Peng Mi, Yong Cao. Load Balanced Parallel GPU Out-of-Core for Continuous LOD Model Visualization. In: Companion: High Performance Computing, Networking Storage and Analysis, 2012, 37: 215-223

[42] Muhammad Hussain, Yoshihiro Okada, and Koichi Niijima. A Fast and Memory-efficient Method for LOD Modelling of Polygonal Models. In: Companion: Proceedings of the 2003 International Conference on Geometric Modeling and Graphics (GMAG' 03), 2003: 1-6

[43] 冯良波, 罗大庸. 3D 多层次模型简化算法的研究. 计算机技术与发展, 2010, 20(4): 97-100

[44] 崔镭. 虚拟现实中 LOD 技术的研究: [硕士学位论文]. 西安: 西安电子科技大学图书馆, 2008

[45] 武凯华. LOD 模型简化算法研究: [硕士学位论文]. 沈阳: 沈阳航空工业学院图书馆, 2009

[46] 郑军. 四元数和旋转矩阵相互转化的算法实现. 阴山学刊, 2012, 26(3): 11-14

[47] Isidro B. Nieto, Jose Refugio Vallejo. A Decision Boundary Hyperplane For The Vector Space Of Conics Using A Polinomial Kernel Inm-Euclidean Space. In: 2008 International Joint Conference on Neural Networks, 2008: 1273-1278