

上海大学  
SHANGHAIUNIVERSITY  
毕业设计（论文）  
UNDERGRADUATEPROJECT(THESIS)

题目：面向移动互联网基于 HTML5 和 WebGL 的三维  
渲染技术研究

学院	计算机工程与科学学院
专业	计算机科学与技术
学号	15120777
学生姓名	李 欣
指导教师	刘福岩
起讫日期	2019.02.25 – 2019.06.07

# 目录

<b>摘要</b> .....	IV
<b>ABSTRACT</b> .....	V
<b>第 1 章 绪论</b> .....	1
§1.1 基于移动互联网与 WebGL 的人物建模系统的背景及意义.....	1
§1.2 Web 端人物建模研究现状及存在的问题.....	1
§1.2.1 研究现状 .....	2
§1.2.2 WebGL 技术存在的问题 .....	3
§1.2.3 Web 端人物建模研究的难点.....	3
§1.3 本文研究内容及目标 .....	3
§1.3.1 研究内容 .....	4
§1.3.2 研究目标 .....	4
§1.4 本文组织结构 .....	4
<b>第 2 章 React 特性及 WebGL 技术简述</b> .....	6
§2.1 React 概述 .....	6
§2.1.1 React 与 HTML 5 的关系.....	6
§2.1.2 React 的背景和意义 .....	6
§2.1.3 React 与 WebGL 的关系 .....	7
§2.2 WebGL 技术简述.....	7
§2.2.1 WebGL 简介 .....	7
§2.2.2 WebGL 和 OpenGL 的联系与区别 .....	8
§2.2.3 three.js 框架概述 .....	8
§2.2.4 WebGL 绘图流程 .....	9
§2.3 开发环境简介 .....	10
§2.3.1 JetBrains WebStorm .....	10
§2.3.2 Chrome 浏览器 .....	10
§2.4 本章小结 .....	10
<b>第 3 章 人物构造部件模型的构建</b> .....	11
§3.1 3D 模型的导入与创建 .....	11
§3.1.1 内部导入预置 3D 几何体 .....	11
§3.1.2 外部构建 3D 模型并导入 .....	11
§3.2 专业建模软件构人物构造部件模型 .....	11
§3.2.1 常见专业 3D 建模软件介绍 .....	12
§3.2.2 选择 blender 的原因 .....	12
§3.3 使用 blender 建模过程 .....	12
§3.3.1 模型采集 .....	12

---

§3.3.2 导入扫描模型添加骨节点 .....	14
§3.3.3 模型导出 .....	17
§3.4 本章小结 .....	17
<b>第 4 章 基于 WebGL 的 3D 人物建模系统的设计与实现 .....</b>	<b>18</b>
§4.1 使用 React 初始化搭建前端框架 .....	18
§4.1.1 初始化 React App .....	18
§4.1.2 依赖包的安装 .....	19
§4.2 使用 three.js 建立 3D 场景基础.....	20
§4.2.1 导入库文件以及资源文件 .....	20
§4.2.2 创建 three.js 场景 .....	21
§4.2.3 添加摄像机.....	22
§4.2.4 添加坐标格 .....	22
§4.2.5 添加摄像机控制器 .....	23
§4.2.6 添加地板阴影、光源与背景模糊.....	23
§4.2.7 添加天空盒 .....	24
§4.3 React 建立前端界面和部件选择器 .....	25
§4.3.1 前端界面模块划分 .....	25
§4.3.2 React 的 Component 类 .....	26
§4.3.3 使用 props 与 state 传递参数与回调函数.....	27
§4.3.4 使用 React 建立部件选择器 .....	27
§4.4 基于部件选择器动态渲染场景中的模型 .....	29
§4.4.1 预定义模型与骨节点继承关系 .....	29
§4.4.2 动态导入模型并拼装 .....	30
§4.4.3 将模型依附到骨节点 .....	31
§4.5 3D 人物姿态编辑与导出 .....	32
§4.5.1 人物模型动作姿态存储格式 .....	32
§4.5.2 姿态编辑与导出 .....	33
§4.6 导出设计好的 3D 人物模型 .....	34
§4.6.1 STL 文件格式 .....	34
§4.6.2 导出为 STL 文件 .....	34
§4.7 移动端适应性 .....	35
§4.8 本章小结 .....	36
<b>第 5 章 总结与展望 .....</b>	<b>37</b>
§5.1 本文总结 .....	37
§5.1.1 本文的主要工作 .....	37
§5.1.2 本文的主要创新点 .....	37
§5.2 展望 .....	38

致谢	39
参考文献	40
附录：部分源程序清单	41

# 面向移动互联网基于 HTML5 和 WebGL 的三维渲染技术研究

## 摘要

随着社会和科技发展的日新月异，工业 4.0 和中国制造 2025 被提上国家发展日程。3D 打印已经在高端工业制造和民间创客团体之间变得越来越普及。

近年来流行的专业 3D 计算机复制模型设计软件有美国的 Autodesk 3DS Max、Autodesk Maya、法国达索的 SolidWorks 等，而这些专业的大型建模软件都有一些共同的不足：它们的预加载程序对计算机资源占用极高、对计算机配置要求也不断增加、使用和建模过程繁琐、没有跨平台功能、软件本体售价高昂、大量插件也需要付费使用。而且由于是专业软件，它们的学习成本极高，非特定领域专业工作人员较难上手。

大多数大型企业和专业制造业集团对于专业人才有这大量储备，所以他们能够轻松地采用市面上流行的专业方案。然而，大量学生、3D 打印业余爱好者和独立游戏制作人可能无法负担得起这样的解决方案。

随着对 3D 模型的需求量日渐增大，一款无需安装的、免费的、用户友好的 3D 人物建模软件就变成了大多数业余 3D 打印爱好者的迫切需求。

本文使用 HTML5 和 WebGL 进行虚拟角色建模和渲染。我们希望能够为社会提供一个对用户友好的免费 3D 人物模型辅助设计服务。人物的身体模块依旧需要使用专业的 3D 建模软件。本文主要介绍和使用的专业软件叫做 blender。我们使用 HTML 语言、基于 Node.js 的 React 和基于 WebGL 的 three.js 库用于在网页上实现所创建模型的呈现、客制化调节面板并对其进行渲染，进一步可以将建造好的模型导出成可供 3D 打印的 STL 格式。

由于 Apple 和 Google 都支持通过 WebGL 在 iOS 和 Android 设备上调用 GPU 进行图像渲染。即本文研究的成果可以在移动端进行展示与交互，并可以进行跨平台网页分发。故冠以“面向移动互联网”的标题。

**关键词：**HTML5, WebGL, three.js, React, Node.js, WebStorm, blender, electron, 三维渲染, 三维建模, 3D 打印

# Research on 3D Rendering Technology

## Based on HTML5 and WebGL for Mobile Internet

### ABSTRACT

With the development of society and the advancement of science and technology, Industry 4.0 and China Manufacturing 2025 have been put on the national development agenda. 3D printing has become more and more popular among high-end industrial manufacturing and kick starter groups.

The recent popular CAD solutions are Autodesk 3DS Max, Autodesk Maya, and Dassault's SolidWorks. These large-scale modelling solutions have some shortages in common: resources-costly core pre-loading procedures, skyrocketing run-time hardware requirements, cumbersome operating routines and modelling process, no cross-platform capability, expensive software licensing, and most of their plugins are charged. Furthermore, since they are designed for pros, the learning essentials are incredibly high, and it is difficult for those who are not pro in some specific fields to even get started.

Most large companies and professional manufacturing groups have a large reserve of professionals with expertise, so they can quickly adapt popular professional solutions on the market. However, the vast majority of students, 3D printing amateurs, and independent game producers may not be able to afford such a solution.

With the increased demand for 3D models, a free, user-friendly 3D avatar modelling software that does not require installation becomes an urgent need for the majority of 3D printing amateurs.

This article uses HTML5 and WebGL for virtual avatar modelling and rendering. Intending to provide a user-friendly free 3D avatar modelling service. We use professional 3D modelling software to sketch and export the body parts of the avatar. The professional software introduced and used in this article is named blender. The HTML language, React based on Node.js and the WebGL-based three.js library are used to render the created models on the web page. Moreover, you can export the built model as STL file which could be used to 3D print the model.

Since both Apple and Google support image rendering via GPU and GPU calls on

iOS and Android devices. Moreover, the results of this study can be displayed and interacted on the mobile side and can be distributed across platforms. Therefore, this article is titled after the mobile Internet.

**Keywords:** HTML5, WebGL, three.js, React, Node.js, WebStorm, blender, electron, 3D Rendering, 3D Modeling, 3D Printing

## 第1章 緒論

本章主要介绍基于移动互联网与 WebGL 人物建模系统的背景和意义。分析了国内外相关主题的研究现状，并提出了本文的内容和目标。

### § 1.1 基于移动互联网与 WebGL 的人物建模系统的背景及意义

随着社会和科技发展的日新月异，工业 4.0 和中国制造 2025 被提上国家发展日程。3D 打印已经在高端工业制造和民间创客团体之间变得越来越普及。

近年来流行的专业 3D 计算机复制模型设计软件有美国的 Autodesk 3DS Max、Autodesk Maya、法国达索的 SolidWorks 等。而这些专业的大型建模软件都有一些共同的不足：它们的预加载程序对计算机资源占用极高、对计算机配置要求也不断增加、使用和建模过程繁琐、没有跨平台功能、软件本体售价高昂、大量插件也需要付费使用；而且由于是专业软件，它们的学习成本极高，非特定领域专业工作人员较难上手。

大多数大型企业和专业制造业集团对于专业人才有着大量储备，所以他们能够轻松地采用市面上流行的专业方案；然而，大量学生、3D 打印业余爱好者和独立游戏制作人可能无法负担得起这样的解决方案。

随着对 3D 模型的需求量日渐增大，一款无需安装的、免费的、对用户友好的人物建模软件就变成了大多数业余 3D 打印爱好者的迫切需求。

并且，5G 时代即将到来，移动互联网用户数量与日俱增。对于需要使用移动设备进行虚拟人物建模和演示的用户而言，blender、3DMax 和 SolidWorks 等专业 3D 软件对移动设备硬件有更高的要求，而且这些 3D 建模软件中的大多数都不提供跨平台功能。WebGL 的出现使得从前那些需要使用专业建模和设计软件完成的特殊功能（诸如：姿势变化、身体部位替换、在线身体部位库等），都可以在浏览器中实现。

随着 WebGL 这种技术变得越来越流行，采用这种技术能够让企业节约大量的人力、物力成本，移动办公的应用又为企业创造出更多的价值，这就造就了一种正向反馈。截止 2019 年 5 月，几乎所有浏览器都已支持 WebGL 2.0。与此同时，在 iOS 和 Android 系统中，WebKit 已经成为浏览器内核标准库。

更高的开发效率与程序执行效率使得将大型 3D 建模工具以 Web 服务的形式进行分发和交付变为可能。

### § 1.2 Web 端人物建模研究现状及存在的问题

近年来国内外学者对 Web 端 3D 渲染的研究做了大量的工作，而基于 WebGL

的 3D 场景渲染技术也开始逐步成熟起来，然而基于移动互联网与 WebGL 的人物建模系统尚属首创，而与 3D 打印领域的结合更是极具前瞻性。下面具体阐述研究现状以及存在的问题。

### § 1.2.1 研究现状

The Khronos Group 于 2011 年 3 月首次向开发者社区发布了 WebGL 1.0 技术规范。经过几年的开发与论证，The Khronos Group 最近推出了 WebGL 2.0 技术规范，绝大多数主流浏览器已全部支持 WebGL 2.0 技术规范。几乎所有的主流浏览器都支持 WebGL 1.0 技术规范，越来越多的国产主流浏览器也正在快步前进于加入 WebGL 2.0 阵营的路上。

HTML 5 的初步提案于 2008 年 1 月 22 日首次向公众发布，其主要更新和“W3C 推荐”状态于 2014 年 10 月发布。其目标是通过支持最新的多媒体和其他新功能来改进 HTML 语言；保持其易于人类阅读，并且不受计算机和设备（如 Web 浏览器、解析器等）的平台制约，而不用严格符合 XHTML 标准的限制，并保持向后兼容旧软件。最新的 HTML 5.2 标准也延续了 HTML 5 强大的向后兼容功能，并在此基础上派生出更多的 DOM 接口。

在这样激进的技术进步下，目前大多数浏览器都已经支持 HTML 5 技术。HTML 页面 3D 渲染具有传统客户端计算机辅助设计软件无法替代的便利、跨平台和卓越的用户体验。

最近国内新兴独角兽企业酷家乐已经涉足 Web 3D 领域在线装修实时预览。其主要技术 WebGL 2.0 的 API 也均支持最新的 DOM 规范和 W3C 推荐的 HTML5 规范。

但是，目前国内外还没有基于 HTML5 和 WebGL 的 3D 人物模型设计平台。只有少数 3D 和 WebGL 技术的激进开发者在互联网上发布与共享了一些 Demo 以供参考，其中大多数都没有建模功能。虽然官方教程和相关说明文档已经对 HTML5 和 WebGL 技术作了详细的解释说明，但是互联网上很少有权威网站和课程可以学习 WebGL 技术，涉及该技术的书籍刊物等学习资料种类和数量也不是十分地充裕。技术只有在使用的时候才能发挥它的价值，尽管有关 HTML5 和 WebGL 技术的论文和期刊文章层出不穷，但是真正用到实处的却是少之又少。基于 HTML5 和 WebGL 技术的 3D 人物模型设计平台的开发具有极好的市场潜力，同时也积累了大量的技术和经验，但其应用在国内外尚属首创。

### § 1.2.2 WebGL 技术存在的问题

近年来, WebGL 技术取得了前所未有的发展, 相关技术不断更新且日臻成熟, 越来越多的用途被开发出来, 越来越多的开发者和走在时代前列的公司开始致力于研究相关技术并将其付诸应用, 但 WebGL 技术的生态发展却仍然存在一些较为棘手的问题。首先 WebGL 技术演进发展至今, 仍未形成大型的、从产业标准指定到库开, 再到应用开发, 最后到应用落地盈利的上下游成熟产业链。其次, 由于微软的 DirectX、苹果的 Metal 2、Khronos Group 的 Vulkan 的三家分晋, 又由于 WebGL 技术基于 OpenGL|ES 技术, 这些因素就导致离 WebGL 统一全平台这一目标越来越远。再者, 不同用户对计算机性能需求不同, 用户的计算机硬件性能也是参差不齐, 而 WebGL 技术对计算机硬件尤其是显卡有较高的要求, 这就使得技术的应用层面和范围难以覆盖到全部用户, 技术更新和优化变得较为困难与繁杂。

### § 1.2.3 Web 端人物建模研究的难点

虽然 WebGL 技术处于高速发展期, 但由于本课题尚属首创, 故存在以下难点:

- (1) 设计面板界面和 3D 渲染界面的分离解耦, 彼此通信只保留基本 API 接口, 并采用不同技术开发 (3D 渲染采用基于 WebGL 的 three.js 库开发, 设计面板和 App 容器采用基于 Node.js 的 React 库开发)。
- (2) 人物身体部位采用专业建模软件单独建模, 并需要按照控制面板的选择, 动态加载到 3D 渲染程序中。
- (3) 人物渲染和导出时需要更具骨节点将各个部件拼接在一起, 并将他们计算成为一个整体 (最小几何体)。
- (4) 最小几何体的姿势需要根据控制面板中的设定而变化 (动态渲染骨节点与模型形变)。

## § 1.3 本文研究内容及目标

本文希望能够为社会提供一个对用户友好的免费 3D 人物模型辅助设计服务。

我们使用 *blender* 构建人物身体各个部位。

使用基于 WebGL 的 three.js 库构建 3D 人物模型辅助设计场景并动态渲染人物模型, 实现所见即所得的用户体验。

再使用基于 Node.js 的 React 构建 Web App 基础框架和人物设计面板, 并针对用户的设备差异进行响应式 UI 布局。

最终将提供基于 three.js 扩展开发的 3D 人物模型导出为可供 3D 打印的 STL

格式的功能。

### § 1.3.1 研究内容

本文研究基于网页端的 3D 人物模型外表与姿态的动态设计与实时渲染，和 3D 人物模型导出为 STL 文件的理论背景和技术实践方案，具体研究内容有以下几个方面：

- (1) 3D 人物组成部件的建模；
- (2) 3D 扫描人物模型添加 3D 人物骨节点；
- (3) 使用基于 WebGL 的 three.js 库构建和渲染 3D 人物模型渲染场景；
- (4) 使用基于 Node.js 的 React 库构建 Web App 基础框架和人物模型设计面板，并针对用户的设备差异进行响应式 UI 布局；
- (5) 使用基于 three.js 扩展开发的 3D 人物模型导出为可供 3D 打印的 STL 格式的功能。

### § 1.3.2 研究目标

依据本文研究的内容的指导，论证制定了如下技术功能点：

- (1) 对 3D 人物组成部件进行建模；
- (2) 构建可视化 3D 场景与光结构；
- (3) 动态加载 3D 人物模型部件并计算最小几何体进行渲染呈现；
- (4) 基于 React 将本服务开发为可跨平台分发的 Web App；
- (5) 实现 Web App 网页对不同设备的动态响应与适配优化。

## § 1.4 本文组织结构

本文整体结构可分为五个章节来叙述。

第一章，首先介绍本文研究的背景、探索的意义，随后细化分析了 Web 端人物建模研究的现状及其存在的问题与难点，论证制定本文研究的技术目标及功能点。

第二章，首先介绍了本文采用之 HTML 5 及 WebGL 技术的发展背景及其使用的必要性，旋即于不懈地探索研究后提出使用基于 Node.js 的 React 库开发 Web App 作为服务载体并使用网页分发，使用基于 WebGL 的 three.js 库开发 3D 人物模型动态渲染服务核心用以提高 WebGL 开发的效率的前瞻方案。

第三章，首先介绍了常用的 3D 建模软件，并阐述了本文选择 blender 作为 3D 人物身体部件建模软件的原因，随后介绍了使用 blender 与 Capture3D 扫描构建 3D 人物身体部件及其骨骼的方法。

第四章，主要介绍了 3D 人物模型建模系统的设计与实现，详细介绍说明了 three.js 构建场景于使用 React 构建模块化动态响应前端系统。随后阐述了人体部件模型的导入方法。详细介绍了模型蒙皮依附至骨节点的算法。之后介绍了通过旋转模型骨节点来为人物做出动作的功能。在此之后介绍了 STL 的文件格式，与导出为 STL 文件时候计算法向量的算法。最后展示了基于 React 开发的响应式前端的移动设备兼容性。

第五章，总结全文的论述，立足本文主要的工作、新颖的创意进行总结与对未来的展望。充分发掘本文的技术于社会中应用发展的可能。最后基于其客观存在的不足，提出其改进的方向。

## 第2章 React 特性及 WebGL 技术简述

本章介绍了本文采用的 HTML 5 及 WebGL 技术的发展背景及其使用的必要性，旋即于不懈探索研究后提出使用基于 Node.js 之 React 库开发 Web App 作为服务载体并使用网页分发，使用基于 WebGL 之 three.js 库开发 3D 人物模型动态渲染服务核心用以提高 WebGL 开发的效率的前瞻方案。

### § 2. 1 React 概述

React 是用于编写泛前端的 JavaScript 库，是对 HTML 5 的、以 JavaScript 为表现形式的一层抽象。它由 Facebook 公司研发并开源，而维护这一“重担”就落到了光荣的开源社区 GitHub 上千千万万的开发者肩上。

React 可以用作网页或移动应用程序开发的基础，因为它最适合用于获取需要记录的快速变化数据的最快方法。但是，获取数据只是网页上发生的事情的开始，这就是为什么复杂的 React 应用程序通常需要使用额外的库来进行状态管理、路由和与 API 的交互。

#### § 2. 1. 1 React 与 HTML 5 的关系

HTML 5 是构建网站的基础，而 React 是在 HTML 5、CSS 和 JavaScript 上基于 JavaScript 借鉴 JSX 语法的高级动态渲染抽象层，能够让开发者使用 HTML 和 CSS 结合 JavaScript 逻辑开发出动态响应式泛前端界面，并且可以有泛后端进行 HTTP Socket 通信。这是一种十分高效而流行的开发手段。

#### § 2. 1. 2 React 的背景和意义

2012 年，React 被 Facebook 发布并开源。

2015 年 2 月，Facebook 发布了 React Native，并于 2015 年 3 月开源并托管至 GitHub。由于可以使用 React 的语法来开发原生移动 App，全世界的 iOS、Android 和 UWP 开发者都对此表示了近乎疯狂的喜悦。

2017 年 4 月 19 日，React 360 V1.0.0 向公众发布。这使得具有使用反应经验的开发人员能够进入 VR 开发阶段。

2017 年 9 月 26 日，React 16.0 向公众发布。

2019 年 2 月 16 日，React 16.8 向公众发布。该版本推出了 React Hooks。

归功于 React 独特的设计理念、革命性的 Web 开发创新、出色的性能、直截了当的代码逻辑和行业巨头的站台支撑，越来越多的人开始关注并在自家的产品上使用它，并认可它可能是未来泛前端（网页、iOS/Android App、桌面 App）开

发的主流工具。

### § 2.1.3 React 与 WebGL 的关系

React 是基于 HTML 5 上的一层抽象化接口，它的出现使得 3D 展示界面变得更加简洁高效，同时使页面渲染变得更加灵活。React 与 WebGL 并没有直接的技术关联，它们之间的联系实际上是建立在 HTML5 之上的，React 是 HTML5 的一个界面展示接口，WebGL 同样也是 HTML5 渲染 3D 图形的接口，它们共同为 HTML5 能够产生更加出色的视觉效果而发挥自己的优势。React 仅仅是建立在软件层面上的，而 WebGL 语言可以直接调用计算机底层硬件，这就弥补了 HTML5 无法调用 3D 硬件加速的缺陷，使得 HTML5 能够支持 3D 图形展示。WebGL 为 HTML 5 三维效果的实现提供了强大的支持，HTML5 为 WebGL 提供了一个用于展现自身技术实力并发挥自身用途的广阔平台，二者相辅相成，不可分割。

## § 2.2 WebGL 技术简述

WebGL 可以在不使用浏览器插件或安装其他本地软件的情况下调用底层硬件。时间进入 2019 年，HTML 5 已经走进了千万家，Web 端能够完成的任务越来越复杂，也越来越强大。现今几乎所有主流浏览器都已经在 WebKit 的基础上发展支持了 WebGL，这样开发者们就能通过浏览器调用底层 GPU 接口开发网页 3D 应用。而 three.js 的出现让开发者们的研发旅途上的道路变得更加平直与通畅。

### § 2.2.1 WebGL 简介

现在人们的生活和网络密不可分，我们几乎每时每刻都在和网络打交道。随着移动互联网从生活中的各个方面和入口进入到我们的生活，想要畅游移动互联网，并体验最新颖、最前瞻的技术，就需要强大的网络技术支持。在网络刚出现的那段时代，人们通过浏览器仅仅能够获得文字、图片、动画和音视频等多媒体信息，仅能与网络实现更加丰富多样而且较为复杂的交互，WebGL 技术正是网络技术与计算机图形技术共同发展相互融合后得以实现。WebGL 是由 Khronos 协会研究出的一个新的规范，它是一个网络浏览器接口，可以在浏览器上绘制 3D 模型、渲染出 3D 效果。WebGL 是免费的开源软件，任何人都可以使用，并且其跨平台的特性也使得它广受欢迎。

现有主流的浏览器 3D 渲染支持诸如 ActiveX 之流需要浏览器插件支持，并且没有提供跨平台支持，而 WebGL 不需要任何浏览器插件支持，原生浏览器即可运行。

WebGL 1.0 基于 OpenGL ES 2.0，为 3D 图形提供 API。它使用 HTML5 canvas

元素，并使用文档对象模型（DOM）接口进行访问。WebGL 2.0 基于 OpenGL ES 3.0，在保证了向后兼容 WebGL1.0 后，Khronos Group 的开发人员们在 WebGL 2.0 中加入了更多、更新、更强的应用程序编程接口，并使用 JavaScript 隐式地提供自动内存管理。与 OpenGL ES 2.0 一样，WebGL 没有 OpenGL 1.0 中引入的固定功能 API，也没有在 OpenGL 3.0 中弃用。如果需要，此功能必须由最终开发人员通过提供着色器代码并在 JavaScript 中配置数据绑定来实现。WebGL 中的 Shaders 直接在 GLSL 中表示，并作为文本字符串传递给 WebGL API。WebGL 将这些 API 调用编译成 GPU 着色器代码，对于每个通过 API 发送到 GPU 流处理器的顶点进行光栅化操作并渲染到屏幕。

### § 2.2.2 WebGL 和 OpenGL 的联系与区别

OpenGL 从出现至今经过数十年发展，在台式终端发挥着巨大的作用，许多出色的三维计算机辅助设计软件上都能看到它的身影，unity 上集成的 OpenGL 接口正在帮助动画与影视后期设计人员设计更多的场面绚丽的特效。万物都有两面性，OpenGL 虽然有着强大的三维图形渲染功能，但是因为它有大量的高级渲染函数和复杂的算法功能，使得它体态过于臃肿，根本无法运用到移动设备终端上。Khronos Group 对 OpenGL 进行精简升级，去掉了许多复杂的高级函数和非必需功能，逐渐演变为 OpenGL | ES，随后又对其进行进一步精简，使其能更好地适用于网页，这才有了最终的 WebGL 标准。OpenGL 可以在桌面软件上大显神威，而 WebGL 是 OpenGL 的衍生品，为了实现 OpenGL 在网络平台上的应用而实现的，多用于网页和移动端的三维图形编程。

### § 2.2.3 three.js 框架概述

使用 WebGL 原生的 API 来写 3D 程序非常复杂，需要相对较多的数学知识，对于前端开发者来说学习成本较高，直接基于 WebGL 开发三维网页虚拟环境也有很大难度。开发者需要对 OpenGL 三维图形渲染和 WebGL 底层接口十分熟悉，尽管如此，开发者在进行 WebGL 底层开发时也极易出错，同时在开发三维场景时也需要丰富的计算机图形学知识作为辅助支持，这就极大地提高了开发者的入手门槛，也导致许多三维专业软件开发者不愿放弃基于桌面的三维计算机辅助设计软件而去尝试 WebGL 开发，使得 WebGL 尽管流行这么多年仍然无法成为开发主流。

为了处理好这些困难，降低 WebGL 开发门槛，让更多爱好三维场景开发的开发者们能够更轻松地上手网页三维场景开发，有一些开发人员在 WebGL 的基础上，将 WebGL 进行适当的封装，隐藏了 WebGL 底层语言的复杂接口细节，开

发出了种类繁多的 WebGL 框架，每个框架都有自己的优势和适用的地方，而最独特的还是要属 three.js。

2010年4月，程序员 Richard Cabello 首次在 GitHub 上发布了 three.js 并开源。three.js 对 WebGL 的底层 API 进行了现代化封装，经过了众多 GitHub 上杰出程序员的摸索与开发，其已经日臻完善，并被用在了很多商用的项目当中。它对 WebGL 的 API 进行了更高层的封装，使原来复杂的调用过程变得简单，开发人员可以在它的基础上开发 Web 3D 应用，并可以在其基础上自由扩展功能及函数库，如果有所思所悟的新功能，还可以提交到 GitHub 上，方便了他人，也方便了自己。这几点显著降低了 WebGL 的学习、上手与开发难度。与此同时，three.js 是目前应用较为广泛的 WebGL 框架，学习资料和官方文档都比较齐全，十分适合三维虚拟环境开发。

three.js 的三个重要组件分别为：场景、摄像机以及渲染器。场景是一种空间，定义一个场景就相当于在网页中使用 three.js 开辟出一块空间。场景中有雾化、强制渲染、背景和自动更新属性，只有在创建好场景后才能在场景中加入模型。相机有透视相机和正交相机，因为透视相机视觉效果更像人的眼睛，所以比较常用。渲染器负责渲染出场景中的模型和材质纹理，场景中的元素都需要渲染器进行渲染才能出现在页面上。除此之外，场景中还可以加入灯光、辅助等元素，不同的灯光效果可以使模型渲染更加逼真和绚丽多彩；辅助元素是为帮助开发者进行场景和模型开发制定的，辅助元素可以极大地方便场景开发。

#### § 2.2.4 WebGL 绘图流程

使用 WebGL 绘制三维图形大致可以分为以下几个步骤：

第一步需要创建一块画布，画布相当于一个容器，用来容纳三维图形，HTML5 中的 Canvas 元素就是个不错的选择；第二步需要获取渲染器上下文，并将其初始化，然后利用其渲染器画出三维图形；第三步需要创建顶点数组，因为 WebGL 三维模型都是由许多顶点组成，每个顶点在三维坐标系中都有自己的坐标，因此需要构建坐标数组来存储模型中的每个点；创建完顶点数组后，第四步就该定义矩阵了，每个顶点都需要进行各种三维坐标变化，而这些变化都可以通过矩阵操作来完成；第五步就该创建着色器了，WebGL 着色器有 Vertex Shader 和 Fragment Shader 两种，着色器程序可以直接调用计算机硬件加速来渲染图形，顶点着色器处理图形顶点坐标数组，片元着色器则用来处理三维模型的材质和纹理；最后是第六步，三维场景模型在经过着色器处理后，就可以通过画布显示在页面上了。

## § 2.3 开发环境简介

本项目运用 React 和 three.js 一起开发，而作为一个前端项目，我使用到了多个软件进行协作。

### § 2.3.1 JetBrains WebStorm

JetBrains WebStorm 是一个建立在 IntelliJ 平台上的智能化 JavaScript 前端开发 IDE。在大一期间，本人参加了百度公司的 IFE 网页前端设计大赛，当时比赛指定 IDE 就是 JetBrains WebStorm，最初本人对这个 IDE 相对本人当时用的 vim 有巨大的不同感到十分惊艳，大量集成化的插件、智能化的代码补全、动态分析代码引用跳转、debug 时的内存空间直接查看、美观而又令人舒适的代码配色、科幻风格的 UI 设计，这让还是一个初级开发者的读大一的本人迅速地爱上了 IntelliJ 平台，并迅速将所有的开发都迁移到了这一套平台上。

所以本文的主力 React 开发平台为 JetBrains WebStorm。

### § 2.3.2 Chrome 浏览器

Chrome 浏览器由 Google 基于 Apple 开发的 WebKit 扩展开发。强大的开发和工具让开发人员的开发体验轻松而又愉快。而现在大多数的浏览器都基于 Google Chrome 开源的 Chromium 开发，所以使用 Google Chrome 作为开发用浏览器是一个绝佳的选择。

## § 2.4 本章小结

经过十几年的研究与开发，React 技术与 HTML 5 实现了重大的变革。React 能够允许用户使用 JavaScript 开发动态响应式前端，而 HTML 5 的 Canvas 标签也让您能通过网页直接调用底层显卡 API 接口，让网页渲染 3D 图形变为可能。WebGL 的 3D 模型渲染技术允许场景中的 3D 模型通过纹理光照渲染并显示在计算机屏幕上，以在网页上获得逼真的体验，而且其广泛的国际大型公司支持让它能够创造出便于 3D 渲染和跨平台的体验，更有 three.js 的出现让开发变得更加轻松而又得心应手。

在本章中，我主要介绍了 React 和 HTML 5 的特性和对 WebGL 的支持，又介绍了 WebGL 技术及其上层封装 three.js 的技术背景和发展历史。稍后我将介绍 Web 端人物建模的具体实现。

## 第3章 人物构造部件模型的构建

本章将 SolidWorks、3DS Max、blender 等主流 3D 建模软件进行对比。对比论证选择 blender 作为本文造型软件的原因，同时介绍建模过程，并在 blender 中展示各模块模型。

### § 3.1 3D 模型的导入与创建

由于本文是以描述 3D 虚拟现实空间建模与姿态构建为主题的研究，故 3D 模型在本文的程序实现中是资源性文件。在现有的 three.js 框架下，模型的导入主要有内部导入预置 3D 几何体和外部构建 3D 模型并导入这两种方法。下文中将讨论与对比分析这两种 3D 模型导入方法。

#### § 3.1.1 内部导入预置 3D 几何体

WebGL 是强大的，但也是复杂的。由于其接近底层的语言特性，使用 WebGL 构建一个 3D 模型对于计算机来说是相对轻松的。然而，对与开发人员而言却是不方便的。打个比方，由于在 WebGL 中 3D 模型都是由三角片构成的，拿数学中最有效论证三角片建模法的四面体为例，在 WebGL 中要表现一个四面体需要 4 个三角片，也就是要传给着色器 12 个顶点信息；而构建一个立方体要 12 个三角片，传给着色器 36 个顶点信息。这样的开发成本是巨大的。

three.js 的优势就在这里体现了出来，由于有着大量低层封装，并且有着自己的内置模型库。我们可以用寥寥几行代码就构建出一个内置的 3D 几何形体，并且可以定义其材质与特性。three.js 中有着很多的预置面、线和几何体，模型库有许多形状，包括规则的多面体、球体、不规则形状、平面、网格和文本等。

#### § 3.1.2 外部构建 3D 模型并导入

如果 three.js 内置的模型库无法满足您的需求，你们也可以使用专业 3D CAD 软件进行建模。再将由专业计算机辅助模型设计软件生成的 3D 模型导入 three.js 的场景中。three.js 为开发者提供了诸如 GLTFLoader 和 STLLoader 之类的模型格式导入库，使导入不同格式的模型不再成为开发者的头疼之事。

### § 3.2 专业建模软件构人物构造部件模型

本文主要的目标是开发一个基于 WebGL 和移动互联网的 3D 人物模型构建服务。然而，人物的部件还是需要在专业建模软件中建立。下面我们将介绍几款主流建模软件并对比说明本文选择 blender 的原因。

### § 3.2.1 常见专业 3D 建模软件介绍

#### (1) SolidWorks

SolidWorks 是法国达索公司开发的一种在 Microsoft Windows 上运行的 3D 建模软件。在工业计算机辅助设计系统领域处于绝对领导地位。

#### (2) Autodesk 3DS Max

Autodesk 3DS Max 是一个专业的 3D 计算机辅助模型设计程序, 多用于 3D 建模和场景动画的制作与自动化演示。在 Windows 平台上运行, 并拥有海量的插件库。强大的功能使其一度成为电影后期、动画与游戏行业建模软件的统治者。

#### (3) blender

blender 是一款开源 3D 计算机辅助设计系统。blender 的上层功能大量覆盖了 OpenGL 的 API, 并且提供 Windows 和 macOS 两个版本。并且没有使用限制。

### § 3.2.2 选择 blender 的原因

由于 SolidWorks 和 Autodesk 3DS Max 都是收费的商业软件, 且只提供 Windows 版本, 而本人使用 Apple macOS 系统, 所以无法使用 SolidWorks 或 Autodesk 3DS Max。

由于 blender 为免费的开源软件, 且提供 macOS 支持, 而本人是一个开源爱好者, 故本人选用了 blender 作为建模软件。

### § 3.3 使用 blender 建模过程

#### § 3.3.1 模型采集

本文的部分模型由 Capture3D 采集实体模型而成, 部分模型来源于网络。下面我们将演示如何使用 Capture3D 来采集实体模型。



图 3-1 True Depth 相机采集模型截图

首先，我们打开 App，并开始采集（并没有采集作者的脸）（如图 3-1），采集过程中保持 True Depth 相机面向模型，并匀速转圈，最终得到该物体的扫描模型。之后将 USDZ 模型 AirDrop 到电脑上。

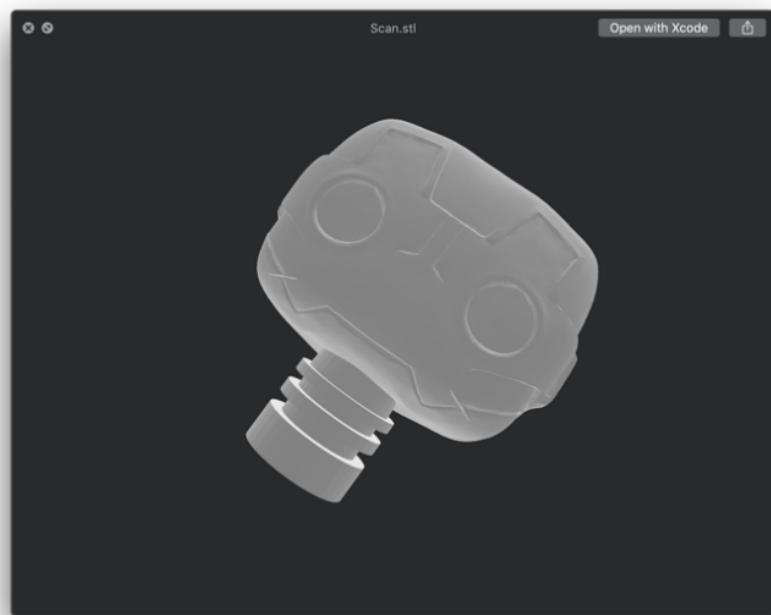


图 3-2 采集完的模型截图

在得到扫描模型后，将其上传到 USDZ 转 stl 网站上进行转换，并下载（如图 3-2）。这就完成了模型的采集。

### § 3.3.2 导入扫描模型添加骨节点

由于本文中需要改变人物姿态形态，故本文将人物的身体部件的组成参考真实的人类建模，而在人物动画中，骨骼是一种建模工具，常用的是改变人物的动作或者姿态。骨骼允许您以比每次重新排列顶点更容易的方式移动角色的肢体。

骨架中的骨骼通常分为两种不同的类型：变形骨骼和控制骨骼。骨骼变形会导致与其相关联的顶点以类似的方式进行变换。变形骨骼直接参与改变与其相关联的顶点的位置。而控制骨骼以类似切换的方式起作用。在这种情况下，它们控制其他骨骼或物体在变化时的反应。例如，当控制骨骼位于左侧的位置时，它可以作为滑动开关控件使用，可以指示其他骨骼在更改时以特定方式作出反应；而当控制骨位被放在右侧时，转换其他骨骼或物体时，会做出一些完全不同的事情。在改变时，骨骼或物体以完全不同的方式作出反应。控制骨骼不直接用于更改顶点的位置。实际上，控制骨骼通常没有与自身直接相关的顶点。

基本原则是将每个骨骼与某些顶点相关联。在姿态模式下，这些顶点将随着骨骼姿势的变化而变化。骨骼是模型渲染时候的幕后工作者，所以我们需要一个模型一同使用它。于是我们选择了刚才扫描到的模型。

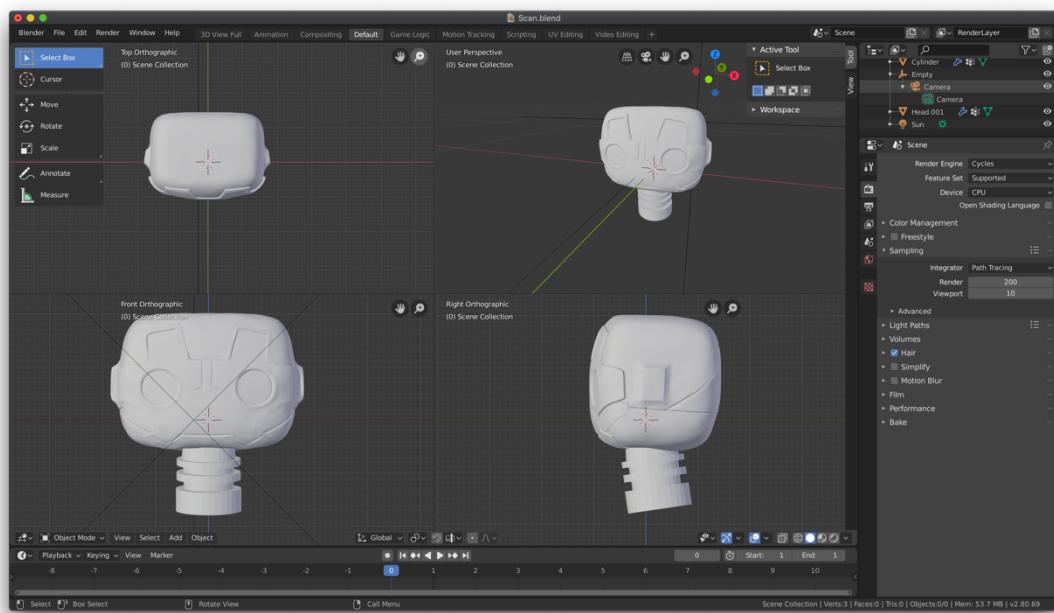


图 3-3 模型导入 blender

我们把刚才扫描到的文件拖入 blender，再打开如图所示的三视图（如图 3-3）。

在添加骨骼之前我们需要先介绍一下 blender 中的骨骼的构成。

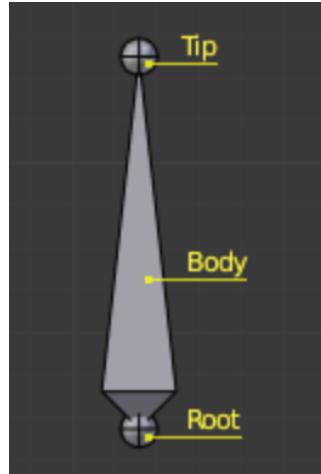


图 3-4 骨节点构成

骨骼有三个基础元素：“开始的关节”叫首端或头部。“身体”部分是骨骼的主体。“结束关节”部分叫顶端或尾端。在编辑模式选择默认的骨架，在这个模式你可以选择首端和尾端，像移动网格顶点一样移动它们。首端和尾端(也叫“关节”)它们各自的位置定义了骨骼。

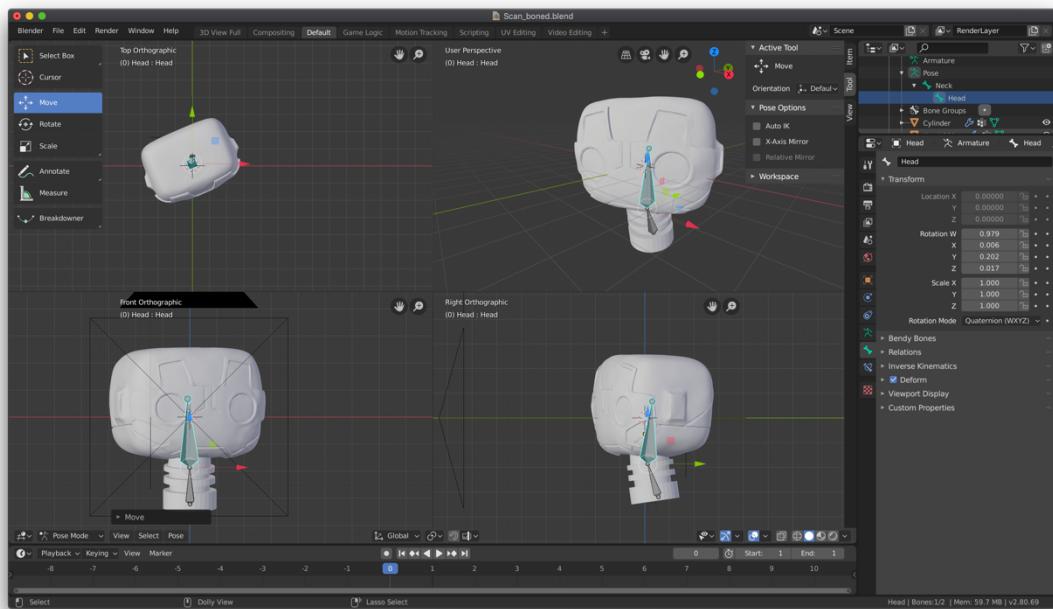


图 3-5 添加完骨节点后的模型

首先我们要在模型的“颈部”添加骨骼并调整到合适的位置，并命名为“Neck”；接着在“Neck”的首部添加一个新的骨节点，新的骨节点会自动将尾部与“Neck”首部相连接，再调整新的骨节点的首部到合适位置后，将其命名为“Head”。头部模块完成后的效果如图所示。

而对于身体其他部位的建模也是同样的流程，本文就不再赘述。在各部位建模完成后，我们需要对人物身体组件的躯干部分进行建模，并指定骨骼接口。

## Bone Listing

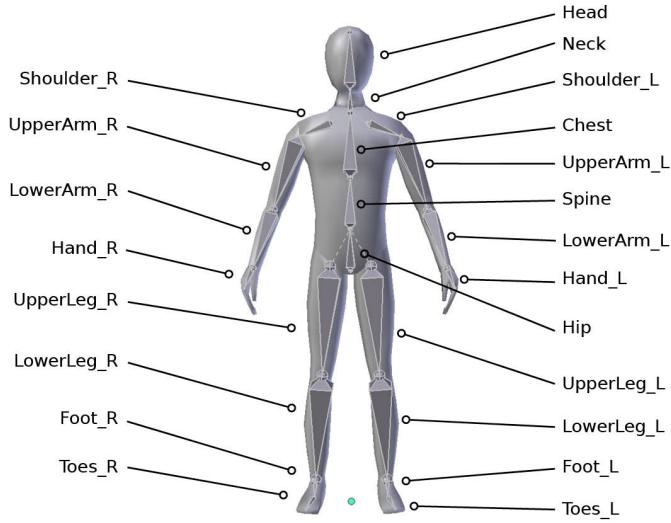


图 3-6 骨节点布局图

如图 3-6 所示，在人体的头部、手臂、手掌、腿部和脚部都只需要建模在摆放骨节点即可完成建模。而身体部位需要额外的操作。

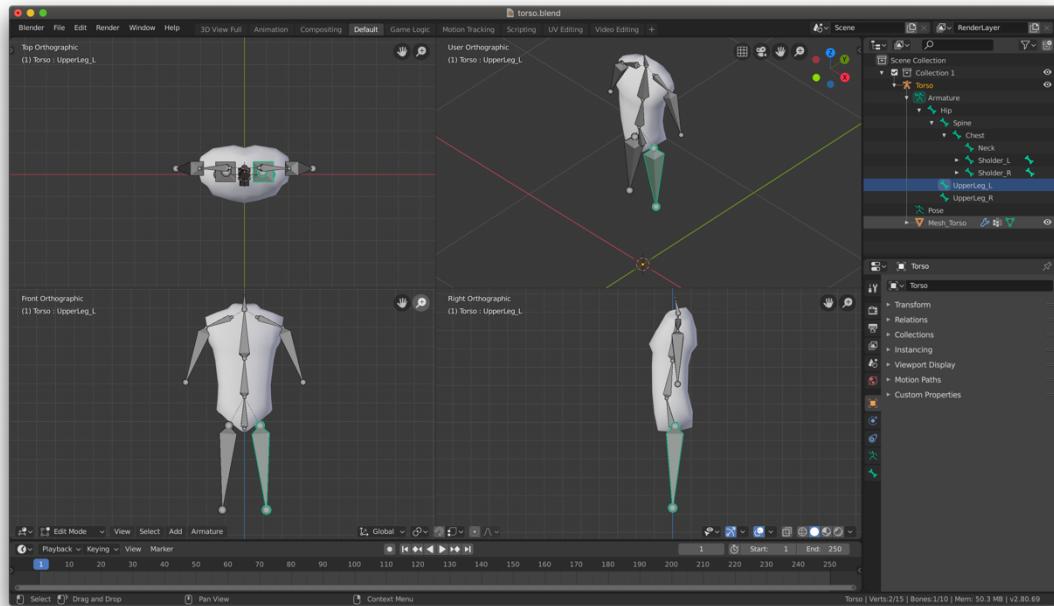


图 3-7 添加完骨节点的躯干部件

在完成身体部分建模和摆放骨节点后，我们并不能就此停止，我们需要摆放一些“冗余”的骨节点，作为拼接模型时的接口。在下一章的代码实现中，我们将详细分析如何使用这些接口来将人物拼接在一起，并实现动作和姿态的渲染，

### § 3.3.3 模型导出

在设计好每一个身体部位时，我们需要将模型导出。`blender` 提供了强大的导入、导出支持，你可以将 `blender` 中内容导出为 `OBJ`、`FBX`、`3DS`、`PLY`、`STL` 等格式，虽然 `three.js` 的推荐模型保存标准是 `JSON` 文件，但是由于本文使用的主要技术是 `WebGL` 技术，且 `blender` 无法不通过插件直接导出 `JSON`。

为了将 3D 模型加载到我们的 3D 人物建模服务中，本人使用了格式 `GLTF`(本人实际上使用了 `GLB`，它只是 `GLTF` 的二进制形式)。本人主要使用这种技术，因为与 `STL` 文件不同，`GLB` 保留的信息不仅仅是模型的几何形状，它还保留了皮肤网格的骨骼、皮肤权重和骨骼名称等信息。

这样，在本人用 `three.js` 实现蒙皮网络时，骨骼上的所有信息已经存在于文件本身而不是外部元数据文件中，这对本人有很大帮助。而且 `glTF` 是 Khronos Group 推出的 3D 模型格式，这和本文的技术核心相切合，并且本文件格式可以通过 `blender` 的内嵌功能进行直接导出。

### § 3.4 本章小结

使用 `WebGL` 进行 3D 人物建模需要大量人体部件。虽然 `WebGL` 也可提供了一些内置的模型，但是简单的球体和立方体等基本几何体无法满足构建复杂人物身体部位模型的要求。`WebGL` 的模型由于 `Mesh` 信息构成，如果通过逐个定点指定位置再构建 `Mesh` 的话，这个工作量是不可想象的，而这样的低效率机械重复对人类来说也是不必要的。而在 `blender` 等专业建模软件中，模型的建立是一个所见即所得的过程，而导出 `Mesh` 也是一件对于用户透明的过程，这样的过程对于开发人员和工程设计人员都是方便的，而本文试图做到的事情也是解放 3D 人物模型构建这的双手，放大他们的生产力。

## 第4章 基于 WebGL 的 3D 人物建模系统的设计与实现

本章阐述了基于 React 和 three.js 开发基于 WebGL 的 3D 人物建模系统之过程与其中除要的技术要点，并简述了部分逻辑实现。

### § 4.1 使用 React 初始化搭建前端框架

本文实现项目使用 React 开发作为容器的 Web App，在开始开发 3D 场景前需要先初始化创建 React 容器来为搭建 three.js 场景创建前提条件，下面将介绍使用 React 初始化搭建前端框架的过程。

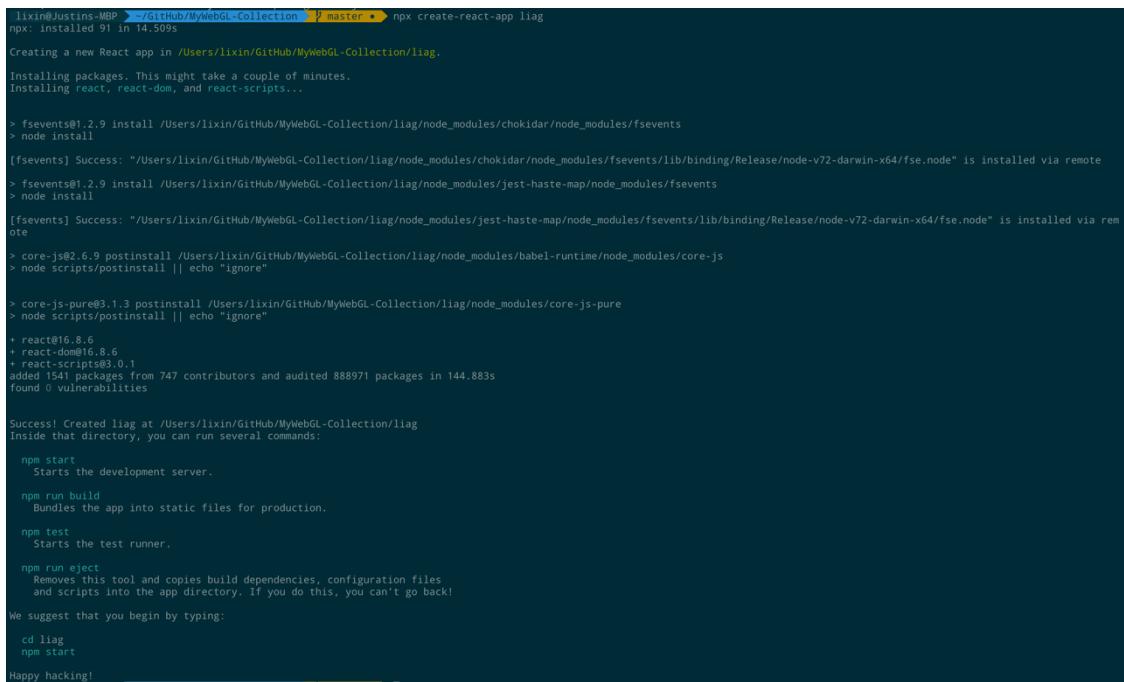
#### § 4.1.1 初始化 React App

使用 React 的第一步是使用 Facebook 在 Node.js 上提供的一个脚手架 App 来创建 React 初始化项目。

在命令行中输入：

```
$ npx create-react-app liag
```

之后 Node.js 就会开始在 CDN 网络中拉取 JavaScript 库文件并保存在本地，由于国内网络环境的原因，这个过程可能会较为漫长。



```
lixin@Justins-MBP ~ % cd GitHub/MyWebGL-Collection / master % npx create-react-app liag
npx: installed 91 in 14.509s
Creating a new React app in /Users/lixin/GitHub/MyWebGL-Collection/liag.
Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts...
> fsevents@1.2.9 install /Users/lixin/GitHub/MyWebGL-Collection/liag/node_modules/chokidar/node_modules/fsevents
> node install
[fsevents] Success: "/Users/lixin/GitHub/MyWebGL-Collection/liag/node_modules/chokidar/node_modules/fsevents/lib/binding/Release/node-v72-darwin-x64/fse.node" is installed via remote
> fsevents@1.2.9 install /Users/lixin/GitHub/MyWebGL-Collection/liag/node_modules/jest-haste-map/node_modules/fsevents
> node install
[fsevents] Success: "/Users/lixin/GitHub/MyWebGL-Collection/liag/node_modules/jest-haste-map/node_modules/fsevents/lib/binding/Release/node-v72-darwin-x64/fse.node" is installed via remote
> core-js@2.6.9 postinstall /Users/lixin/GitHub/MyWebGL-Collection/liag/node_modules/babel-runtime/node_modules/core-js
> node scripts/postinstall || echo "ignore"
> core-js-pure@3.1.3 postinstall /Users/lixin/GitHub/MyWebGL-Collection/liag/node_modules/core-js-pure
> node scripts/postinstall || echo "ignore"
> react@16.8.6
> react-dom@16.8.6
> react-scripts@2.0.1
added 1541 packages from 747 contributors and audited 888971 packages in 144.883s
found 0 vulnerabilities

Success! Created liag at /Users/lixin/GitHub/MyWebGL-Collection/liag
Inside that directory, you can run several commands:
  npm start
    Starts the development server.
  npm run build
    Bundles the app into static files for production.
  npm test
    Starts the test runner.
  npm run eject
    Removes this tool and copies build dependencies, configuration files
    and scripts into the app directory. If you do this, you can't go back!
We suggest that you begin by typing:
  cd liag
  npm start
Happy hacking!
```

图 4-1 创建 React 应用截图

在本地脚本执行完毕后，我们会看到上图 4-1 所示界面，这是和 React 就已经创建了一个名叫“liag”的项目，我们也得到了如下图 4-2 所示的文件目录结构。

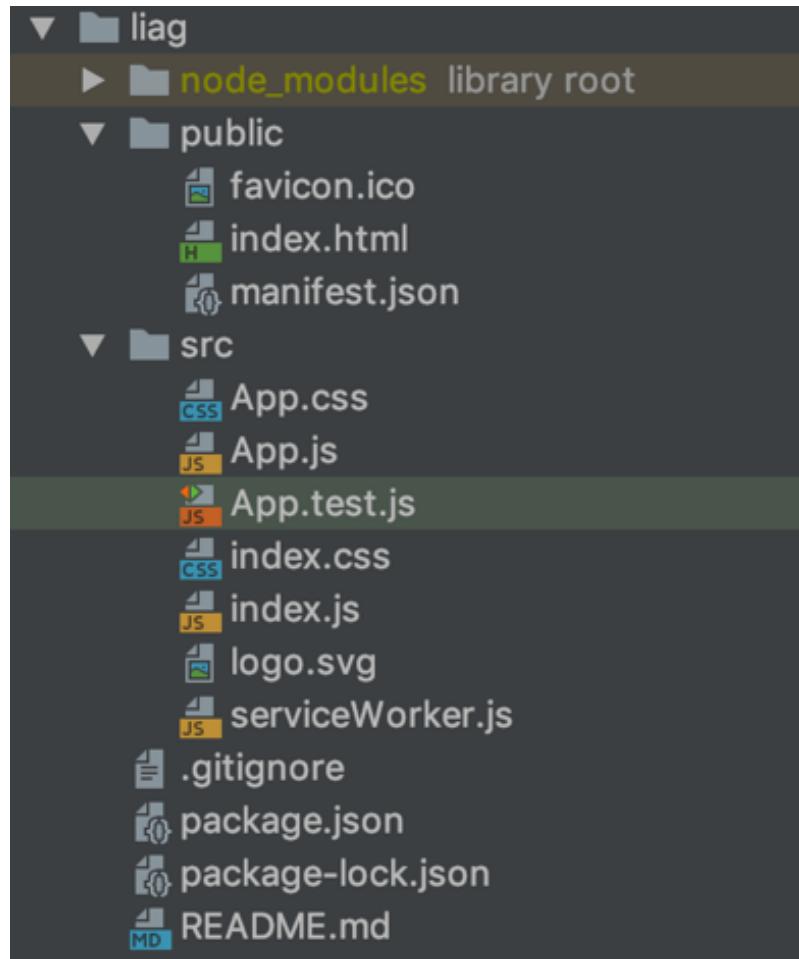


图 4-2 React 应用初始文件列表

如图 4-2 所示，在/liag/node\_modules 目录中存放着的是 Node.js 和 React 的库文件；在 /liag/src 目录中一般存放 JavaScript 代码、React 的框架部件、服务注册库和构成前端界面的各个组成部件；而在/liag/public 目录路中存放前端渲染文件模版，也可以存放第三方 JavaScript 库；/liag/package.json 中存放的是整个 React App 的配置，而依赖列表也是以 JSON 的方式存放其中。

### § 4.1.2 依赖包的安装

在刚开始创建万 React 项目模版时，/liag/package.json 是仅仅存放有 React 基础依赖信息的，之后我们需要按照需求添加依赖包。

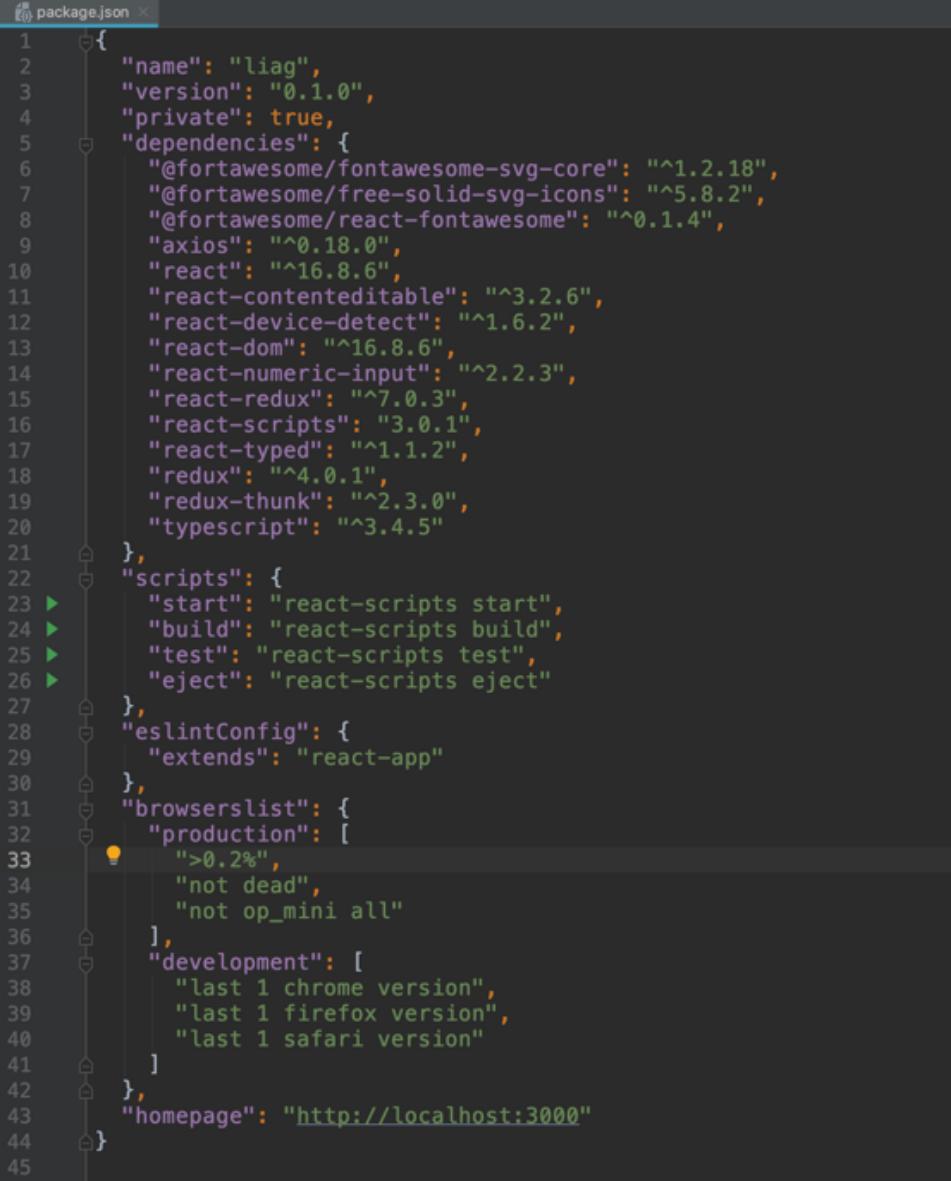
下面我们演示一下如何添加“axios”包：

在命令行中输入：

```
$ npm install --save axios
```

在进度条走完后，/liag/package.json 会自动添加相关依赖，不用手动添加。

在后续开发过程中，如有需要添加更多的包也可以直接在命令行中输入添加，下图 4-3 为本文添加完成依赖包后的/liag/package.json 截图。



```

1  {
2    "name": "liag",
3    "version": "0.1.0",
4    "private": true,
5    "dependencies": {
6      "@fortawesome/fontawesome-svg-core": "^1.2.18",
7      "@fortawesome/free-solid-svg-icons": "^5.8.2",
8      "@fortawesome/react-fontawesome": "^0.1.4",
9      "axios": "^0.18.0",
10     "react": "^16.8.6",
11     "react-contenteditable": "^3.2.6",
12     "react-device-detect": "^1.6.2",
13     "react-dom": "^16.8.6",
14     "react-numeric-input": "^2.2.3",
15     "react-redux": "^7.0.3",
16     "react-scripts": "3.0.1",
17     "react-typed": "^1.1.2",
18     "redux": "^4.0.1",
19     "redux-thunk": "^2.3.0",
20     "typescript": "^3.4.5"
21   },
22   "scripts": {
23     "start": "react-scripts start",
24     "build": "react-scripts build",
25     "test": "react-scripts test",
26     "eject": "react-scripts eject"
27   },
28   "eslintConfig": {
29     "extends": "react-app"
30   },
31   "browserslist": {
32     "production": [
33       ">0.2%",
34       "not dead",
35       "not op_mini all"
36     ],
37     "development": [
38       "last 1 chrome version",
39       "last 1 firefox version",
40       "last 1 safari version"
41     ]
42   },
43   "homepage": "http://localhost:3000"
44 }

```

图 4-3 依赖包列表

## § 4.2 使用 three.js 建立 3D 场景基础

在完成创建 React App 容器后，我们开始使用 three.js 建立 3D 场景基础。

### § 4.2.1 导入库文件以及资源文件

首先我们需要做的是导入 three.js 的库文件，由于 three.js 是开源软件，故其可以在 GitHub 上很轻松地下载到。具体下载流程就不赘述了，这里本文选用的是 three.js\_r104 稳定版，也是最新发布的稳定版本。

因为我们需要重载 three.js 的 STLExporter，故我们选择从外部导入 three.js 库及其相关支持文件，而不是在 React 中添加依赖。

之后我们需要导入在开发过程中需要用到的资源文件，这里就不赘述。

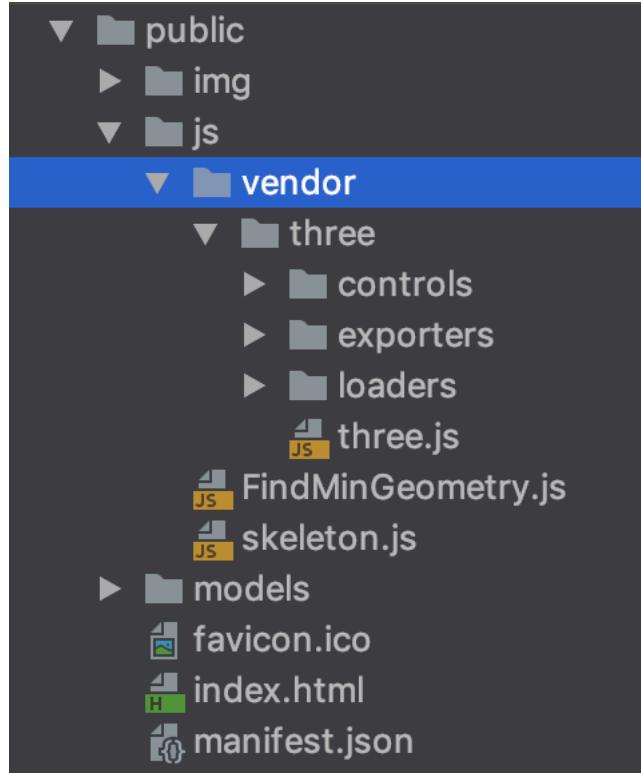


图 4-4 外部 JavaScript 文件列表

如图 4-4 所示，在导入完 three.js 库后目录 /liag/public/ 的文件列表中，/public/js/vendor/ 目录下全部为 three.js 的库文件及其辅助文件，而在 /exporter/ 目录下的 STLExporter 是要自重载的文件。在 /public/models/ 目录下的文件为上一章中导出的 GLB 文件，这些文件存储着一个个加工好的人体部件模型信息。

而在 index.html 中也加入了 skeleton.js 和 FindMinGeometry.js 的引用，这时候 three.js 便会在网页加载时进行渲染。

### § 4.2.2 创建 three.js 场景

场景在 three.js 中是一切的起源，它是一个容器，能够让 three.js 引擎对其子元素进行遍历渲染。

```
let scene = new THREE.Scene();
scene.background = new THREE.Color(0x282c34);
```

图 4-5 代码

上图 4-5 在 skeleton.js 中创建了一个基础场景，并给其设置了藏青色的背景颜色。这时候一切的基础已经具备，但是仅仅是这样的话还是太过于简陋，如果这个时候运行程序，那么迎接您的将是一片没有任何东西的屏幕。

那么我们就必须开始向场景中添加必要的元素。

### § 4.2.3 添加摄像机

场景中的物体不会自动显示到屏幕上，想要 React App 运行时不再是只显示一个纯黑色的页面，你就需要在场景中添加摄像机。three.js 内置了两种摄像机可供选择，他们分别是正交投影摄像机和透视投影摄像机。正交投影摄像机的成像效果类似于数学课上老师们给我们画的几何图形的效果，物体 X 轴和 Y 轴与 Z 轴的夹角为 45 度 ( $\pi/2$ )。如果从一个正立方体的正上方俯瞰此立方体，在正交摄像机中，你只能看到一个正方形。而透视投影摄像机的成像效果更像人类的肉眼，有着近大远小的效果，符合美术概念的透视原理，透视摄像机因此得名。如果从一个正立方体的正上方俯瞰此立方体，在透视摄像机中，你将看到两个嵌套的正方形。

```
function buildCamera() {
    camera = new THREE.PerspectiveCamera(
        75,
        (window.innerWidth / window.innerHeight),
        0.001,
        1000
    );

    camera.position.x = -1;
    camera.position.y = 2;
    camera.position.z = 2;
}
```

图 4-6 代码

如图 4-6 所示，由于本文旨在实现一个所见即所得的 3D 人物建模系统，故本文选择在场景中添加透视摄像机。对于 `fov`，也就是我们常说的视角，本文采取了与人类视角类似的 75 度。

### § 4.2.4 添加坐标格

由于本文要做的是一个 3D 人物建模系统，这时候辅助线的设置就显得尤为重要。

```
let size = 50;
let divisions = 60;
let colorCenterLine = 0x306d7d;
let colorGrid = 0x61dafb;

let gridHelper = new THREE.GridHelper(size, divisions, colorCenterLine, colorGrid);
scene.add(gridHelper);
```

图 4-7 代码

在上图的代码中，我们为场景创建了一个坐标格，并设置了它的大小、分段数、中心轴线颜色以及辅助线颜色。坐标格其实是一个二维线数组，在这种情况下，如果使用 WebGL 原生 API 绘制的话可能会十分麻烦，这里就体现了 three.js 的便利之处。

### § 4.2.5 添加摄像机控制器

为了方便用户在设计角色的时候能够对角色有个 360 度的认识。我需要让摄像机能够围绕角色运动起来，这时候我们就要向场景中添加摄像机控制器。这时候 three.js 提供的 OrbitControls 就是适应当前需求的选择，再结合 TrackballControls 就可以让用户使用鼠标拖动摄像机的位置与滚轮缩放摄像机的视角。

```
function buildControls() {
    controls = new THREE.OrbitControls(camera, renderer.domElement);
    controls.minDistance = 2;
    controls.maxDistance = 7;
    controls.minPolarAngle = 0;
    controls.maxPolarAngle = Math.PI / 2 - 0.1;
    controls.enablePan = false;
}
```

图 4-8 代码

如图 4-8 所示，代码中在创建 OrbitControls 后限定了用户能够观察的角度（不能从底往上看，俯视离垂直差 0.1rad），来减少不必要的渲染。

### § 4.2.6 添加地板阴影、光源与背景模糊

为了能让用户在实时渲染模型时候获得更加逼真的 3D 体验，我们需要模型在实时渲染的时候拥有影子。为了能够显示地板反射，我们需要在场景中添加一个地面模型。而 three.js 中常用的地面模型一般使用 PlaneGeometry 并选择 PI/2 来作为场景中的“地面”。为了使“地面”能够反射光线，我们需要将“地面”的材料改为 ShadowMaterial。

然而这时候还是看不清的，这时候我们需要加入一些光源，由于我们需要照亮地面及辅助坐标格，但是却不想让它们的阴影影响到其他东西，我们需要用到半球光，three.js 中可以调用 HemisphereLight 类来实现。半球光的优势是照射方向上没有影子，可以理解为手术室中的无影灯。再在 Z 轴负方向 20 与 Y 轴正方向 2 处放置点光源，作为背景灯光。点光源可以调用 PointLight 类实现。因为本文将世界坐标系的原点（也就是 (0, 0, 0) 的位置）作为 3D 人物模型的基础渲染点，人物方向面向 Z 轴正方向，所以我们再在角色预定方位的右前方向，也就是初始摄像机方位后方放置另一个点光源来做颜色增强。使得人物的造型更加饱满。

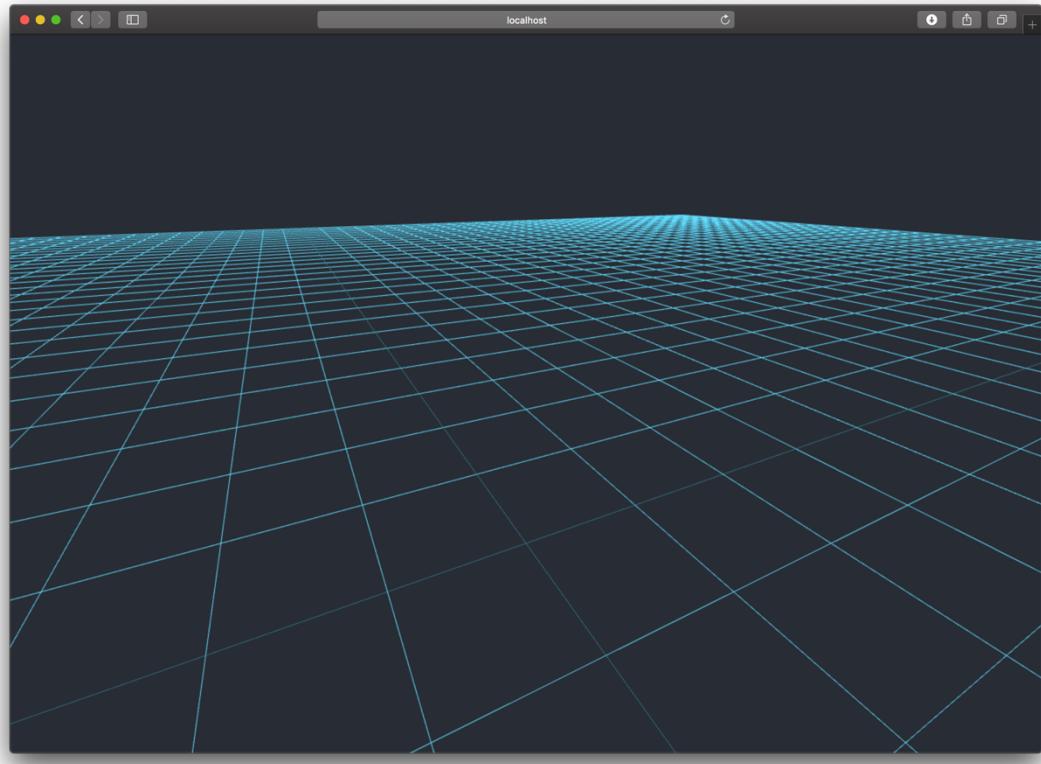


图 4-9 只添加了坐标格的场景

这时候虽然已经可以看见场景已经初具雏形，但是由于是透视相机的缘故，远处的坐标格被渲染得密密麻麻，很是不美观。于是我们要在场景中添加雾化效果，将远处的做出背景模糊的效果。雾化效果可以直接调用 three.js 的 Fog 类实现。

#### § 4.2.7 添加天空盒

下图 4-10 的代码展示了添加天空盒的过程：首先，使用 CubeTextureLoader 将贴图资源中传入的六维地址数列中加载进来；然后指定为 RGB 格式渲染到缓冲区；最后将场景的背景由纯色替换成天空盒。

```

let path = ".../img/library/textures/fantasy/";
let extension = ".jpg";
let urls = [
  path + "px" + extension,
  path + "nx" + extension,
  path + "py" + extension,
  path + "ny" + extension,
  path + "pz" + extension,
  path + "nz" + extension
];
let reflectionCube = new THREE.CubeTextureLoader().load(urls);
reflectionCube.format = THREE.RGBFormat;
scene.background = reflectionCube;

```

图 4-10 代码

而在使用天空盒的过程中，会出现天空盒坐标系与 WebGL 坐标系种类不同的问题。按照由于 Khronos Group 基于 1990 年代立方体贴图中的 RenderMan

规范。在一个坐标系中，正向 x 轴在向正 z 轴看时是向右。换句话说，天空盒使用的坐标系是左手坐标系。与此相对的是，因为技术发展的历史原因，本文中的技术 WebGL 是用了另一种坐标系。由于本文所采用的技术框架使用坐标系统是笛卡尔坐标系。因此，在 three.js 中使用的环境贴图需要交换正 x 轴和负 x 轴（每个 three.js 立方体贴图示例都是这种情况）。

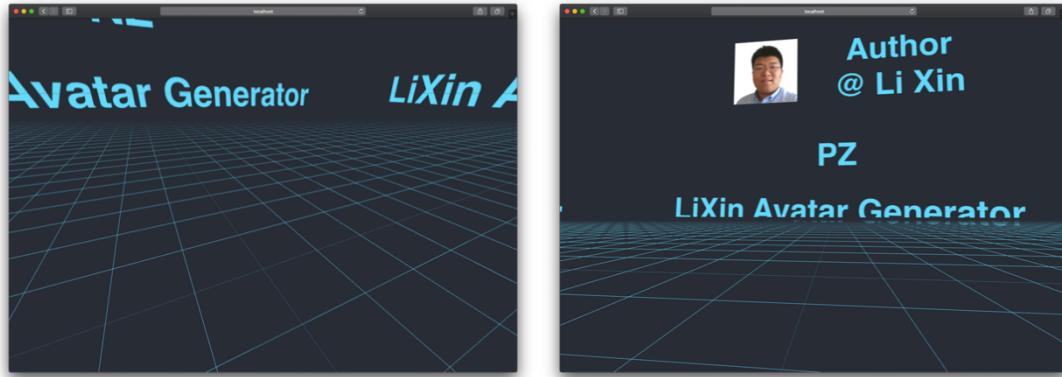


图 4-11 添加完天空盒的场景 1

图 4-12 添加完天空盒的场景 2

在添加好天空盒后，效果如上图（图 4-11、图 4-12）（作者在天空盒中添加了小彩蛋），这里我们也能清晰地看到雾化效果对远处坐标格进行的背景虚化效果十分理想。

### § 4.3 React 建立前端界面和部件选择器

在完成了 three.js 的场景构建后，由于模型是动态渲染的，所以我们不急于、也无法立刻去构建组装人物身体模型。现阶段的第一件事就是建立前端页面和人物身体部件选择器。

#### § 4.3.1 前端界面模块划分

为了方便说明，我们先把完成图（如图 4-13、图 4-14 放上来，并用有颜色的框框出每一个部分。



图 4-13 React 各组件布局 1

图 4-14 React 各组件布局 2

如上图 4-13、图 4-14 所示，为了实现高内聚和低耦合的设计原则，本文将 React 前端组件分为 7 大部分。分别为按键区域、命名输入框、GitHub 开源项目链接（本项目将在完成毕业设计答辩后开源，暂时闭源是为了防止知网误判重复）、Footer 信息栏、身体部件类型选择栏和详细部件选择栏。而详细部件选择栏目细分为搜索栏、部件选择栏和姿势编辑栏目。他们的调用关系如下图 4-15 所示：

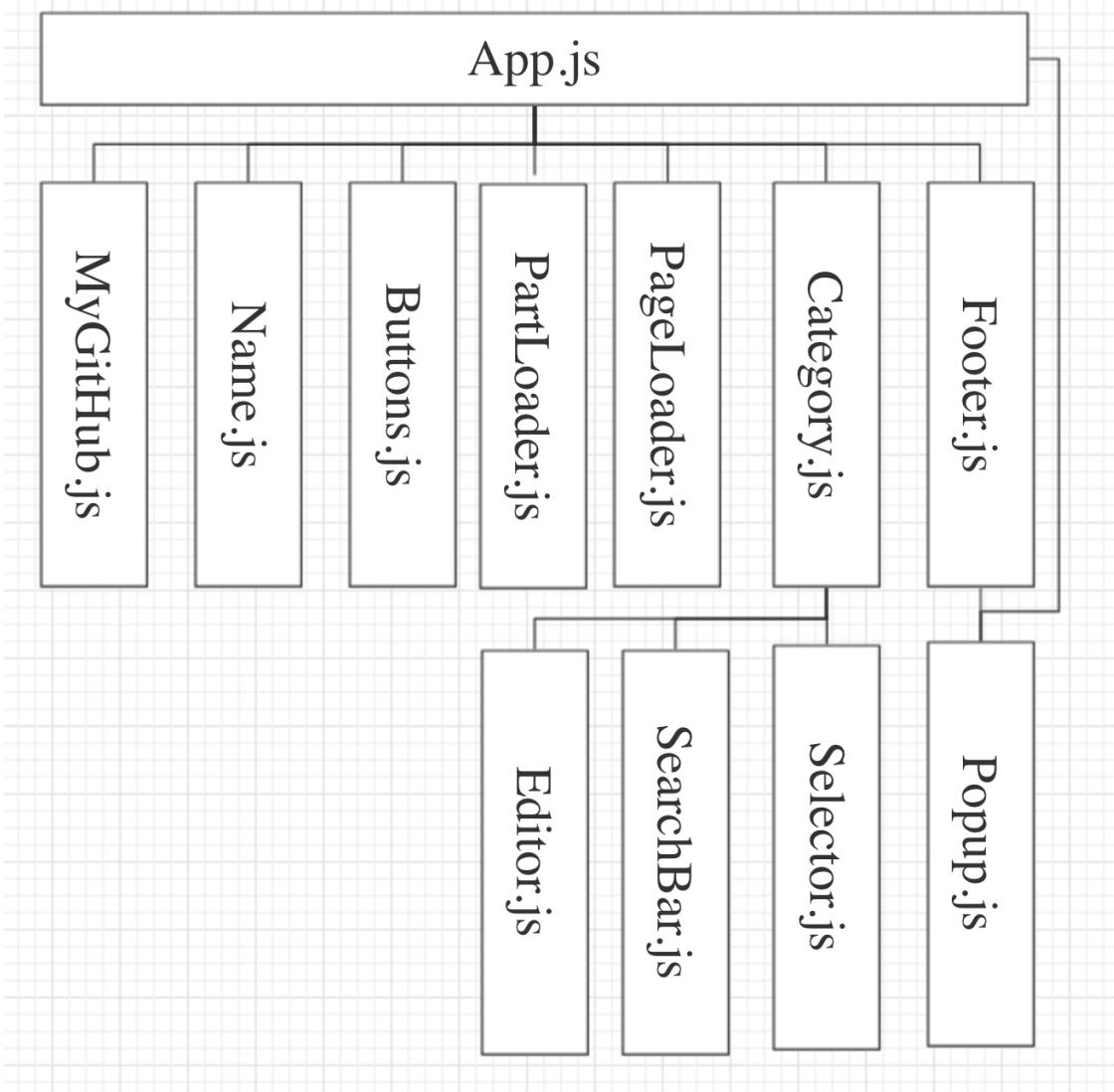


图 4-15 调用关系表

### § 4.3.2 React 的 Component 类

React 的运行机制决定了我们可以通过继承 React 中内置的 Component 类来将构成我们前端页面的组件，并将其定义成一个个的类或者函数。通过这种操作我们就可以完成大量前端代码的复用和组件之间的传值与回调函数通信。

继承 Component 类之后我们必须做的是重载 `render()`方法。由于在 React 生命周期中，当一个 Component 的子类被创建并渲染到前端页面的时候，React 会调

用 render()方法让 Component 的子类动态渲染组件的 HTML 代码并将其按照字符串返回到 App.js 中动态组装。

如果需要在继承了 Component 的类中接收传入的参数 (props) 或者使用自有参 (state) 数需要重载 constructor()方法。

### § 4.3.3 使用 props 与 state 传递参数与回调函数

在 ECMAScript 6 发布之前，在 JavaScript 中定义类的传统方法是定义生成实例对象的构造函数，而引入了“class”关键字的 ECMAScript 6 于 2015 年才姗姗来迟。

由于 React 初代的发布时间早于 2015 年，后期版本在初代版本的基础上迭代更新。所以尽管现在按照 ES6 的标准，JavaScript 已经可以使用“class”关键字，然而 React 中的参数传递与回调函数传递方法依旧需要通过两个重要的概念：props 与 state。

在 Component 子类中定义属性与方法的时，React 需要开发者在其 constructor() 属性中定义 state 属性，并将其他属性与方法定义在 state 中。

在外层 Component 子类中的 render()方法中调用另一个 Component 子类模块的时候，如需向被调用子类传递参数与回调函数，只需要在调用 Component 子类的 HTML 标签中定义属性参数，或将自己的方法函数指向属性参数即可。而 React 可以将 HTML 标签中定义的属性和指向属性参数的回调函数动态指向被引用的 constructor()方法的 props 参数。这样被引用类就可以通过 props 调用引用类 state 中的属性与方法。

### § 4.3.4 使用 React 建立部件选择器

为了能够使部件选择器能够在用户做出操作时候动态加载人物身体部件模型，我们需要使用 JSON 文件定义人物身体部件模型文件与名字和截图的对应关系、骨节点与其“友好名称”的对应关系和预定义人物姿势。定义好的文件列表如图 4-16 所示。

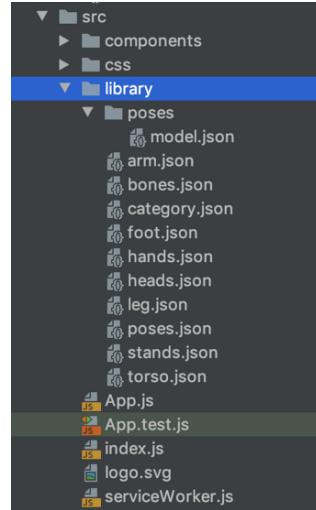


图 4-16 JSON 文件列表

部件选择器 (Selector.js) 继承了 React 的 Component 类，在 render()方法中，部件选择器根据 JSON 文件中的定义加载资源并渲染到前端，在用户做出点选操作后，调用 skeleton.js 中暴露给窗体的 selectedMesh() 这一函数进行模型的动态渲染（下一小节介绍）。

在其引用的搜索框组件 (SearchBar.js) 中，部件选择器通过 state 将自身的 search 属性暴露给搜索框，在用户于搜索框中输入字符的时候，搜索框会通过 props 调用预先暴露的回调函数 updateSearchValue() 来使部件选择器按名称筛选 JSON 内容，并只渲染合适名称的组件选择按钮。

其他组件的开发本文就不赘述。在完成所有组件开发后效果图如下：



图 4-17 开发完成的 React 前端面板

## § 4.4 基于部件选择器动态渲染场景中的模型

本节介绍如何根据部件选择器动态渲染场景中的模型，以及骨节点组装方法。

### § 4.4.1 预定义模型与骨节点继承关系

为了让用户在打开人物模型构造器时看到的不是空空如也，我们需要在 JSON 文件中预定义默认加载的模型。这里我们将默认模型组成定义在 JSON 文件中。如图 4-18 所示，这时候在部件选择器加载的时候会调用 `skeleton.js` 中预先暴露给窗体的 `selectedMesh()` 方法，导入默认模型，最终拼接渲染出一个默认的人物模型。但是现在还没能对模型进行任何的设计与改动。



图 4-18 渲染默认模型

## Bone Listing

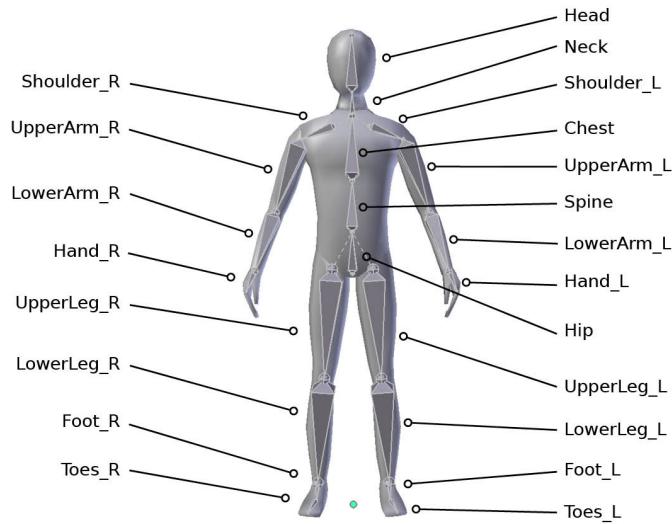


图 4-19 骨节点结构图

在渲染模型之前，我们需要知道哪一块骨头接在哪一块骨头上面，这时候我们只需要参考上图 4-19 中的各个身体部件的连接处来进行关系制定就可以，不需要制定所有的拼接关系。

### § 4.4.2 动态导入模型并拼装

由于指定了默认导入的预定义模型，这时候就可以根据用户在部件选择器上的选择，调用回调函数更改内存中存储的模型组件名，再根据模型组件名通过 React 的 axios 库向服务器发送 HTTP 请求获取 glTF（为了压缩体积，使用了二进制版的 GLB 文件）文件。

在加载组件的时候，我们会把它看成是一个树状结构，这样我们就不仅仅加载了这个模型，也将 Mesh 的骨节点加载了上来。

在我们加载完模型部件后，我们需要将这些加载进来的部件按照骨节点的继承关系拼装在一起成一个树形结构存储与 group 对象中，形成一个整体的模型。

### § 4.4.3 将模型依附到骨节点

```

for (let i = 0, len = bufferIndices.count; i < len; i += 3) {
    const a = bufferIndices.getX(i);
    const b = bufferIndices.getY(i);
    const c = bufferIndices.getZ(i);

    // eslint-disable-next-line no-loop-func
    [a, b, c].forEach(callbackfn: vertexIndex => {
        vector.fromBufferAttribute(bufferPositions, vertexIndex);
        let boneIndices = [];
        boneIndices[0] = bufferSkinIndices.getX(vertexIndex);
        boneIndices[1] = bufferSkinIndices.getY(vertexIndex);
        boneIndices[2] = bufferSkinIndices.getZ(vertexIndex);
        boneIndices[3] = bufferSkinIndices.getW(vertexIndex);

        let weights = [];
        weights[0] = bufferSkinWeights.getX(vertexIndex);
        weights[1] = bufferSkinWeights.getY(vertexIndex);
        weights[2] = bufferSkinWeights.getZ(vertexIndex);
        weights[3] = bufferSkinWeights.getW(vertexIndex);

        let inverses = [];
        inverses[0] = skeleton.boneInverses[boneIndices[0]];
        inverses[1] = skeleton.boneInverses[boneIndices[1]];
        inverses[2] = skeleton.boneInverses[boneIndices[2]];
        inverses[3] = skeleton.boneInverses[boneIndices[3]];

        let skinMatrices = [];
        skinMatrices[0] = skeleton.bones[boneIndices[0]].matrixWorld;
        skinMatrices[1] = skeleton.bones[boneIndices[1]].matrixWorld;
        skinMatrices[2] = skeleton.bones[boneIndices[2]].matrixWorld;
        skinMatrices[3] = skeleton.bones[boneIndices[3]].matrixWorld;

        let finalVector = new THREE.Vector4();
        for (let k = 0; k < 4; k++) {
            let tempVector = new THREE.Vector4(vector.x, vector.y, vector.z);
            tempVector.multiplyScalar(weights[k]);
            tempVector.applyMatrix4(inverses[k]).applyMatrix4(skinMatrices[k]);
            finalVector.add(tempVector);
        }
    })
}

```

图 4-20 代码

如图 4-20 所示的代码中，*i* 是顶点的索引，由于一个三角面由三个顶点确定，所以索引 *i* 以 3 递增。顶点 *i* 的骨骼索引（boneIndices）在几何体中定义为四维向量，由于 three.js 规定每个顶点不支持超过 4 个骨骼，也就是说，如果顶点受到多于 4 个骨骼的影响，顶点就会无效。我们用一个四维向量来存储这四个骨节点对顶点 *i* 的影响。之后我们将存储在 Mesh 中的逆矩阵导出存储在 *inverses* 数组中。这些逆矩阵可以将静态几何体顶点从世界矩阵转换到骨骼的局部矩阵空间。我们

再把存储在 Mesh 中的蒙皮举证导出到 skinMatrixes 这一数组中。这些矩阵与 inverses 中的矩阵作用正好相反，是把骨骼局部空间的点转换到世界坐标系中。最后计算蒙皮举证与逆矩阵的乘积即可。这样就完成了一个顶点的位置运算。

在完成了模型中所有顶点的位置运算后，我们就把模型依附到了骨骼节点的位置。人物模型客制化效果如图。



图 4-21 基于用户选择动态渲染的模型

## § 4.5 3D 人物姿态编辑与导出

仅仅客制化 3D 人物模型组成部分对于一款人物设计服务来说是远远不够的。我们还需要对人物的姿态与动作编辑功能。

### § 4.5.1 人物模型动作姿态存储格式

由于构成人物的组件模型在渲染过程中是依附到骨节点上的，存储人物模型的时候只需要存储各个骨节点的旋转信息就可以了。这时候以 JSON 形式存储是再适合不过的了。为了方便用户创建模型，我们在动作姿态存储文件中添加了一些预定义的动作。

### § 4.5.2 姿态编辑与导出

人物姿态遍编辑面板有两套系统可供选择：

第一套系统，如图 4-16 所示，是在部件选择器界面中复用部分 UI，让用户可以在 JSON 预建的模型中选择一些自己喜欢的直接去打印。



图 4-22 选择预定的姿势

第二套系统，如图 4-22 所示，是允许用户指定任何一个骨节点的旋转信息。由于物体旋转 360 度后的姿势和 0 度是一样的，因此，旋转信息的调节只允许在  $(-\pi, +\pi)$  区间内调节。



图 4-23 自定义姿势

在用户编辑完成心仪的姿势后可以导出为 JSON 文件，以供保存。

## § 4.6 导出设计好的 3D 人物模型

### § 4.6.1 STL 文件格式

STL 是一种 3D 打印时候常用的格式，由于其简单易懂的内容格式，深受广大程序员和 3D 打印爱好者喜爱。

```

solid name

    facet normal ni nj nk
        outer loop
            vertex v1x v1y v1z
            vertex v2x v2y v2z
            vertex v3x v3y v3z
        endloop
    endfacet

endsolid name

```

图 4-24 STL 文件格式

STL 文件格式如图 4-24 所示，每个 facet 表示 WebGL 里的一个三角面， $(ni, nj, nk)$  储存的是该三角面指向模型外部的单位法向量。**vertex** 储存的是该三角面的顶点信息。而 WebGL 使用了三角片建模法，这时候，采用 STL 作为导出模型文件的格式就是一个再适合不过的选择。

### § 4.6.2 导出为 STL 文件

和上文中介绍的将模型依附到骨节点类似，我们只要在针对每个 Mesh 进行相应的格式转换就可以完成。

这时候我们就可以复用大量将模型依附到骨节点的代码，只需要在循环运行的时候于 STL 文件中写入相印格式的信息即可。

```

function computeFaceNormal(a, b, c) {
  const cb = new THREE.Vector3();
  const ab = new THREE.Vector3();

  const vA = new THREE.Vector3().fromBufferAttribute(bufferPositions, a);
  const vB = new THREE.Vector3().fromBufferAttribute(bufferPositions, b);
  const vC = new THREE.Vector3().fromBufferAttribute(bufferPositions, c);

  cb.subVectors(vC, vB);
  ab.subVectors(vA, vB);
  cb.cross(ab);

  cb.normalize();

  return cb;
}

```

图 4-25

唯一需要另外计算的就是三角法向量，图 4-25 中的代码展示了计算平面法向量的过程。将三角面的三个顶点定义为 vA、vB 和 vC；再计算出向量 CB 和向量 AB，之后计算他们两个的叉积，根据右手定则可知结果即为一垂直于三角面的向量；最后对此向量计算单位向量，即可得到三角面的法向量。

导出 STL 文件预览效果如图 4-26 所示：

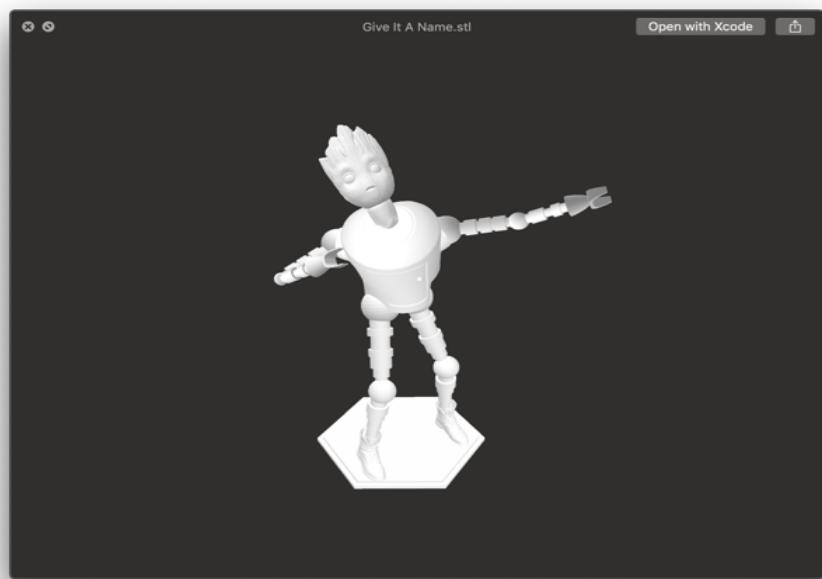


图 4-26 导出的模型

## § 4.7 移动端适应性

由于采用 React 开发，本文中的项目采用响应式布局，在 iPad 等宽屏移动设备上有较好的浏览体验。下图 4-17 为 iPad Pro 端的网页截图。



图 4-27 iPad Pro 端的网页截图

## § 4.8 本章小结

本章主要介绍了 3D 人物模型辅助设计系统的设计与实现，详细介绍了 three.js 构建场景于使用 React 构建模块化动态响应前端系统；随后阐述了人体部件模型的导入方法，详细介绍了模型蒙皮依附至骨节点算法；之后介绍了通过旋转模型骨节点来为人物做出动作的功能；在此之后介绍了 STL 的文件格式，与导出为 STL 文件时候计算法向量的算法；最后展示了基于 React 开发的响应式前端的移动设备兼容性。

## 第5章 总结与展望

### § 5.1 本文总结

本文主要介绍了面向移动互联网基于 HTML5 和 WebGL 的三维渲染技术，设计并实现了基于 WebGL 的 3D 人物模型辅助设计系统。随着 WebGL 这种技术变得越来越流行，采用这种技术的企业能够节约大量的人力、物力成本，移动办公的应用又为企业创造出更多的价值，这就造就了一种正向反馈。而本文中实现的 3D 人物建模系统解决了在创建 3D 人物的时候，大型专业计算机辅助设计系统上手难、收费贵的缺点，让大量学生、3D 打印业余爱好者和独立游戏制作人能够在短时间内得到自己想要的 3D 人物模型。而本文中作品免费开源的特性也使它能够帮助到更多需要它的人。

#### § 5.1.1 本文的主要工作

本文主要介绍了面向移动互联网基于 HTML5 和 WebGL 的三维渲染技术。设计并实现了基于 WebGL 的 3D 人物模型辅助设计系统。在此过程中论述了如下工作：

- (1) 三维度模型的采集与添加骨节点。
- (2) 使用 React 创建响应式前端。
- (3) 根据用户选择动态加载与渲染人物模型。
- (4) 根据用户操作改变模型姿态和动作。
- (5) 导出模型为 STL 文件。

#### § 5.1.2 本文的主要创新点

随着社会和科技发展的日新月异，工业 4.0 和中国制造 2025 被提上国家发展日程。3D 打印已经在高端工业制造和民间创客团体之间变得越来越普及。

近年来流行的专业大型建模软件都有一些共同的不足：它们的预加载程序对计算机资源占用极高，对计算机配置要求也不断增加，使用和建模过程繁琐，没有跨平台功能，软件本体售价高昂，大量插件也需要付费使用；而且由于是专业软件，它们的学习成本极高，非特定领域专业工作人员较难上手。

大多数大型企业和专业制造业集团对于专业人才有着大量储备，所以他们能够轻松地采用市面上流行的专业方案；然而，大量学生、3D 打印业余爱好者和独立游戏制作人可能无法负担得起这样的解决方案。

而本文中实现的 3D 人物建模系统解决了在创建 3D 人物时候，大型专业计算机辅助设计系统上手难、收费贵的缺点。让大量学生、3D 打印业余爱好者和独立游戏制作人能够在短时间内得到自己想要的 3D 人物模型。而本文中作品免费开源的特性也使它能够帮助到更多需要它的人，加之其移动互联网的特性，让人们可以在任何时间、任何地点进行 3D 创作。

## § 5.2 展望

本文主要介绍了面向移动互联网基于 HTML5 和 WebGL 的三维渲染技术。验证了使用 WebGL 开发一款支持全平台的计算机辅助人物模型设计系统的可能，为国、内外同行打下了技术基础。相信在新一轮的技术变革中，这一期会更加完善，日臻完美。

而本文中实现的 3D 人物建模系统虽然解决了在创建 3D 人物时候，大型专业计算机辅助设计系统上手难、收费贵的缺点。但是导出的模型处于 STL 阶段，不利于二次使用，在日后投放到开源社区后，本文中的系统将增加更多文件格式的支持，让这个系统能够帮助到更多的人。