

北京信息科技大学

毕业设计（论文）

题 目：视频驱动虚拟角色动作的自动生成系统的设计与实现

学 院：_____ 计算机学院 _____

专 业：_____ 软件工程 SP _____

学生姓名： 王衡飞 班级/学号： 软工 1801-1/2018011479

指导老师/督导老师：_____ 宋文凤 _____

起止时间：_____ 2022 年 2 月 28 日至 2022 年 6 月 10 日 _____

摘要

目前，虚拟主播、动画制作、元宇宙等领域发展的如火如荼，前景十分广阔。但以往虚拟形象的动画制作是个庞大的工程，需要耗费大量的时间和人力进行拍摄、重建等工作，即便是现有的视频捕捉方案，也需要额外的硬件成本和较高的硬件要求。于是本课题设计并实现了视频驱动虚拟角色动作的自动生成系统，该系统致力于使用普通的视频素材，对比了多个视频驱动的动作捕捉算法并选择了为实时性能优化的 BlazePose GHUM 3D 算法，对人物动作进行分析实时提取并生成三维骨骼坐标信息，其次设计了骨骼转换算法、适用于多种虚拟形象映射绑定算法，将其应用于驱动虚拟形象。此外还设计了实时通信模块将生成的虚拟形象用于视频直播和 AR/VR 直播。编写了一套桌面图形用户界面软件用于实现这些功能并管理虚拟形象，该系统跨平台、易于安装和使用且对计算机配置要求较低，与现有的解决方案相比大大降低了虚拟主播及虚拟形象生成的门槛，并创新性的实现了 AR/VR 实时虚拟形象转播。

关键词： 动作捕捉； 虚拟形象； 虚拟主播； 元宇宙；
增强现实； 骨骼映射；

Abstract

Virtual reality technologies, including Virtual Character, are widely applied in VTuber, animation production, metaverse, and other fields, and the prospects are extensive. However, the animation production of virtual characters is a massive project, which requires a lot of time and workforce for refactoring and drawing. Even the existing video capture solutions require additional hardware costs and higher hardware requirements. Therefore, this project designs and implements an automatic generation system for video-driven virtual character animation. The system is dedicated to using common video materials, compares multiple video-driven motion capture algorithms, and selects BlazePose GHUM 3D, which is optimized for real-time performance. The algorithm analyzes the character's action and extracts and generates the three-dimensional skeleton coordinate information in real-time. Secondly, the skeleton transformation algorithm is designed, suitable for a variety of avatar mapping and binding algorithms and is applied to drive the virtual characters. In addition, a real-time communication module is designed to use for live video, and AR/VR live broadcasts. A set of desktop graphical user interface software is written to realize these functions and manage virtual characters. The system is cross-platform, easy to install and use, and requires less computer configuration. Compared with existing solutions, it greatly reduces the number of VTuber and the threshold for virtual characters generation, and innovatively realizes AR/VR real-time virtual characters live broadcasts.

Keywords: Motion Capture; Virtual Characters; VTuber; Metaverse;
Augmented Reality; Bones binding;

目 录

摘要.....	I
Abstract	II
目录.....	III
第一章 概述.....	1
1.1 研究背景	1
1.2 相关技术的国内外文献综述简析	2
1.3 研究内容与主要贡献	3
1.4 论文结构	3
第二章 相关技术与开发环境.....	4
2.1 TensorFlow Lite、MediaPipe 机器学习框架.....	4
2.2 虚拟形象模型格式	4
2.2.1 VRM 格式.....	4
2.2.2 GLB/GLTF 格式.....	4
2.2.3 FBX 格式.....	5
2.3 Electron 跨平台桌面 GUI 框架.....	5
2.4 Three. js 基于 WebGL 的 3D 渲染引擎.....	5
2.5 WebSocket 基于 TCP 的全双工网络传输协议	5
2.6 OBS 开源直播软件	5
2.7 WebXR API	5
2.8 开发环境 VSCode 与 Copilot	6
第三章 视频驱动虚拟角色动作的自动生成系统需求分析与设计.....	7
3.1 视频驱动虚拟角色动作的自动生成系统需求分析	7
3.1.1 视频驱动虚拟角色动作的自动生成系统功能需求	7
3.1.2 视频驱动虚拟角色动作的自动生成系统非功能性需求	9
3.2 运行环境要求	9
3.3 视频驱动虚拟角色动作的自动生成系统总体设计	10
3.4 动作捕捉模块设计	10
3.5 虚拟形象管理及展示模块设计与实现	11
3.6 渲染模块设计与实现	15
3.7 图形用户界面设计与实现	16
3.8 通信模块设计与实现	18
第四章 动作捕捉算法及虚拟形象绑定算法.....	21
4.1 视频驱动的动作捕捉算法研究	21
4.2 BlazePoze GHUM 3D 算法及性能评估	22
4.3 动作捕捉骨骼架构转换算法	26

4.4 虚拟形象骨骼映射驱动算法	28
第五章 视频驱动虚拟角色动作的自动生成系统的实际应用.....	32
5.1 用于 OBS 软件的直播、录制接口设计与实现	32
5.2 AR/VR 的虚拟形象直播功能的设计与实现.....	33
第六章 跨平台发布及优化.....	36
6.1 发布跨平台应用程序	36
6.1.1 Windows 平台下打包及优化.....	36
6.1.2 macOS 平台下打包及优化	37
6.2 使用 CI/CD 系统进行自动打包	38
6.3 以 B/S 模式运行系统	39
6.4 性能优化	40
6.4.1 在双显卡设备上使用高性能 GPU.....	40
6.4.2 后台渲染优化	41
6.4.3 多线程优化	41
结束语.....	42
致谢.....	43
参考文献.....	44

第一章 概述

本论文针对视频驱动虚拟角色动作的自动生成，从课题来源与研究意义出发，通过研究国内外文献综述及现有的相关商业解决方案，学习课题相关技术。本课题的任务是设计并实现一个视频驱动的虚拟形象动作生成系统的适用于 Windows/macOS/Linux 平台的桌面应用程序，主要功能是导入和管理虚拟形象，从实时或非实时的视频数据中自动获取人体的动作信息，将获取到的动作信息用于驱动导入的虚拟形象 3D 模型并可用于直播、录制、AR/VR 实时展示等操作。且将该系统作为一个开箱即用的软件，用户无需配置 Python/CUDA 等依赖程序，在全新的 Windows/macOS 计算机上直接双击即可运行整套系统。

1.1 研究背景

目前直播、动画制作领域发展的如火如荼，虚拟主播/虚拟偶像等应用，除了互联网大厂早已布局外，诸多传统知名品牌也逐渐尝试引入虚拟+内容到自身品牌宣传活动中。虚拟偶像俨然成为互联网上广泛议论的话题之一。截止至 2022 年 5 月，目前国内人气较高的虚拟偶像“嘉然”在 Bilibili 上粉丝已经超过 164.2 万，在抖音上的粉丝数更是达到了 215.9 万。此外，许多大型活动也邀请了虚拟主播/虚拟偶像进行参加，2021 年 8 月，乐元素在 B 站举办了以虚拟偶像为主的名为 NBF 音乐节的活动，并且推广了自家的新虚拟偶像“吉诺儿”。2021 年清华大学主办的“华语辩论世界杯”辩论赛上，更是出现了“虚拟偶像与真实偶像，谁更能满足当代人的精神需求”这一辩题，足以看出虚拟形象和虚拟主播的未来的广阔前景。

虚拟偶像技术的发展离不开动作捕捉技术，在国家政策和互联网浪潮的推动下，中国动作捕捉系统行业发展迅速，取得了巨大成就，应用领域不断扩大。近年来，国内动作捕捉系统的行业快速增长，产值从 2013 年的 23.8 亿元增长到 2017 年的 44.5 亿元。知名动作捕捉厂商诺亦腾（Noitom）此前曾于 2015 年底获得超过 2 千万美元的 B 轮融资，估值超过 2 亿美元，年销售额达数千万。

在我国 3D 电影、游戏互动、虚拟现实的出现，不仅丰富了人们的娱乐活动，也带动了动作捕捉系统行业的发展。并且虚拟形象、虚拟偶像在电影、娱乐等行业都有着极其广阔的前景^[1]。

而元宇宙的概念的提出，如 Meta（原 Facebook）、Epic、腾讯等各大厂家分分布局元宇宙，有上百亿美元的资本融入元宇宙行业；以及未来 VR/AR 设备的普及，动作捕捉产生虚拟形象并应用于 AR/VR 势必是元宇宙中不可或缺的一部分^[2]。

但以往虚拟形象的动画设计及制作是个庞大的工程，需要耗费大量的时间和人力进行拍摄、绘制等工作；虚拟主播、虚拟偶像也有着很高的门槛，使得现在虚拟主播往往都由大公司进行运营，虚拟主播进行实时直播更是难上加难。

而该项目致力于使用普通的视频素材或摄像头，以及普通的计算机配置（无需价格高昂的独立显卡），对人物动作进行分析提取，并根据人物动作对虚拟角色形象进行实时的生成，并可应用于直播、动画视频制作等领域，甚至创新性的提出了 AR/VR 虚拟主播的概念并将其实现，且大大降低了这方面制作的成本，节省了时间。

1.2 相关技术的国内外文献综述简析

目前，整个系统的技术难点主要集中在两个方面：一是如何捕捉人体的骨骼动作信息，在业内通常将此任务称作“动作捕捉”；二是如何使用捕捉到的动作信息来驱动虚拟形象，使虚拟形象可以根据捕捉到的动作信息来进行运动，并借助渲染软件进行渲染。通常这两个方面分为二维和三维两个版本，二维版本要更为简单，无需获得深度信息；而三维版本则是进一步升级，让任务形象可以进行立体旋转。本项目需要使用三维动作捕捉和驱动三维虚拟形象。

二维的动作捕捉已经发展的十分成熟也有许多成熟的解决方案（如 OpenPose^[3]），该部分主要讨论三维动作捕捉。但其实，视频源的三维动作捕捉是依赖二维动作捕捉的，这个结论在下文中也会进行具体的阐述。

穿戴式的三维动作捕捉技术路线上分为光学动捕和惯性动捕两个方向。光学动作捕捉基于光线从身上所穿戴的标记点反射到不同位置的摄像机，对人体上的标记点进行绝对定位来构建三维模型还原运动。惯性动作捕捉^[5]通过穿戴惯性传感器来捕捉人的关键骨骼旋转信息，通过加速度传感器及陀螺仪传感器，对人体的运动进行测量，最后将数据传输到计算机上去还原人体运动。这类的解决方案需要使用者佩戴专门的动作捕捉硬件才能实现，虽然的准确度较高^[4]，但是其价格不菲，而且穿戴专门的动作捕捉设备往往大幅度增加了动作捕捉的门槛，通常只用于专业领域。

非穿戴式视频方案的三维动作捕捉通常可以用是否包含深度信息来划分，通常可以分为使用 RGB 单视频源（既只有一个角度视频源）、RGB 双目或多角度视频源（既有多个视频源，可以从中更好的判断深度信息）、RGBD 视频源（其中 D 代表深度）。使用 RGBD 视频源的解决方案目前有用于全身动作捕捉的带有红外测距摄像头 Kinect^[7]，及 iPhone X 及之后使用带有 3D 结构光的传感器(Face ID) 用于面部捕捉^[8]，他们利用深度信息可以直接判断骨骼关键点的二维位置后加入该二维点所在像素的深度信息即可，准确度更高计算量更小，但是缺点是具备可以用于全身及面部同时进行捕捉的 RGBD 摄像机也比较昂贵。基于多个 RGB 视频源的动作捕捉方案^[6]可以利用多个视频源中的视角差来大致评估深度信息。

基于 RGB 单目相机视频的三维动作捕捉方案，通常是使用神经网络技术将已经通过其他动作捕捉方式(如穿戴式动作捕捉设备)提取过动作信息的视频与原视频作为数据集进行训练后进行使用。首先提取出 2D 的骨骼信息，然后生成三维信息（可通过对人体的三维重建然后进行投影^[14]），图 1-1 展示了大致的流程。

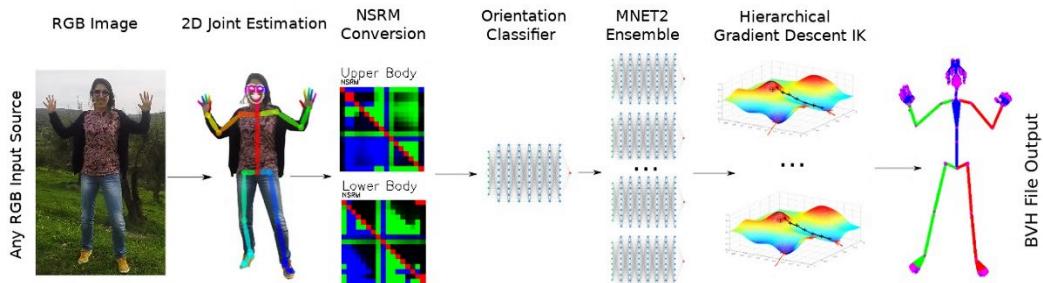


图 1-1 RGB 视频三维动作捕捉流程^[16]

1.3 研究内容与主要贡献

该项目研究内容为如何利用相关的动作捕捉算法及编写虚拟形象驱动的骨骼绑定和映射算法进行虚拟角色动作的自动生成，并编写一套可开箱即用的桌面软件（不需要用户配置如 Python、CUDA 等环境，详见“3.1.4 运行环境要求”部分），支持 Windows/macOS 等主流平台，可自行导入不同种类多种格式的虚拟形象，同时也内置了多种虚拟形象，并由本软件完成骨骼映射，通过用户输入的视频文件或摄像头实时驱动他们并输出到屏幕上，并且编写了动作转发和输出转发实时通信系统使其可以连接到直播系统，如开源直播系统（Open Broadcast Software，OBS）及 VR/AR 设备。

主要贡献包括但不限于以下方面：

- (1) 分析了多种动作捕捉算法，选择了适合该系统的算法并对其进行了优化及性能评估。
- (2) 编写了适用于多种主流虚拟形象 3D 模型格式的骨骼映射算法。
- (3) 研究了如何将实时生成的虚拟形象用于直播、录制、AR/VR 的解决方案。
- (4) 编写了该系统的桌面端和可运行在浏览器上的软件，将其集成在到一个可开箱即用的跨平台桌面端环境内。
- (5) 并编写了动作转发服务器，并研究了如何将 HTTP/HTTPS 协议使用同一端口进行监听。
- (6) 使用 CI/CD 系统进行从源代码自动编译及打包，将整套系统及算法以 Mozilla Public License, version 2.0 协议开源。

1.4 论文结构

该论文第一章阐述了背景及综述，以及对于当前已有论文的阅读和分析状况；第二章阐述了相关技术开发环境，并阐述了这些技术在该项目中的应用；第三章对系统的整个设计进行了设计与实现，讲述了每个模块的作用；第四章详细介绍了本系统使用和设计的关键算法；第五章介绍了该系统的应用场景及与其他设备、软件的接口；第六章介绍了对系统的优化以及编译打包交付等细节。

第二章 相关技术与开发环境

本章节讲述了开发所使用到的框架、技术、格式标准等，这些内容在设计本系统时都有所使用。

2.1 TensorFlow Lite、MediaPipe 机器学习框架

TensorFlow Lite 是一种运行在设备端的开源深度学习框架，它通常被用于在移动设备、嵌入式设备和物联网设备上运行模型^[9]。相比传统的在服务器端使用 GPU 运行 TensorFlow 模型的模式，它具有离线、高效、轻量等特点，这使得它更加适合于应用在对实时性更高的场景。

TensorFlow Lite 模型以称为 Flat Buffers 的专用、高效的可移植格式（由文件扩展名 “.tflite”）表示。相比 TensorFlow 的 Protocol Buffers 模型格式，这种格式的体积更小和推理速度更快有优势。

而 MediaPipe 是一款由 Google Research 开发的开源多媒体机器学习模型应用框架，为直播和流媒体提供跨平台、可定制的机器学习解决方案。任何 TensorFlow 和 TensorFlow Lite 的模型都可以在 MediaPipe 上使用。

该系统使用 Mediapipe 运行 BlazePose GHUM 3D 的 Tensorflow Lite 模型进行视频驱动的实时动作捕捉处理，该部分详见第四章。

2.2 虚拟形象模型格式

该系统支持导入 VRM、GLB/GLTF、FBX 格式的虚拟形象，并支持对这些格式应用本文第五章所提到的骨骼映射和绑定算法。

2.2.1 VRM 格式

VRM 是专注于人型的 3D 文件格式，是由日本的十三家厂商（IVR、XVI、S-court、cluster、Crypton Future Media、SHOWROOM、DUO、DWANGO、Virtual Cast、Pixiv、Mirrativ、Unity Technologies Japan、Wright Flyer Live Entertainment）联合发起设立 VRM 联盟推出的专门用于 3D 虚拟偶像的文件格式标准。该格式具有统一的骨骼标准、命名方式、坐标系等，是目前大多数虚拟主播软件支持的格式。

2.2.2 GLB/GLTF 格式

图形语言传输格式（Graphics Language Transmission Format，GLTF），其有.gltf 和.glb 两种扩展名。glTF 是一种免版税规范，用于通过引擎和应用程序高效传输和加载 3D 场景和模型。其支持储存三维模型、外观、场景及动画，旨在成为一种用于 3D 资产的简化的、可互操作的格式，同时最大限度地减少应用程序的文件大小和处理难度。通常这种格式在 Web 中使用较为广泛。

2.2.3 FBX 格式

FBX 是由 Kaydara 开发的一种专有文件格式，自 2006 年以来一直归 Autodesk 所有，较多的专业建模软件往往使用该格式。这是一种 3D 资产交换格式，在 Adobe Mixamo 网站上主要使用 FBX 格式进行下载和上传。

2.3 Electron 跨平台桌面 GUI 框架

Electron 是 GitHub 开发的一个开源框架。它通过使用 Node.js（一种通常用于后端开发的框架）和 Chromium 的渲染引擎完成跨平台的桌面 GUI 应用程序的开发。使用具有强大生态的 Web 技术进行开发，同时可以直接在 UI 上使用 Node.js 模块，开发成本低，可扩展性强，有着更炫酷的 UI。并且其跨平台，一套代码可打包为 Windows、Linux、Mac 三套软件，且能编译快速。

该项目的 GUI 基于 Electron 框架开发。

2.4 Three.js 基于 WebGL 的 3D 渲染引擎

Three.js 是一个开源的 JavaScript 库，可让使用者轻松创建 3D 内容。借助于 Three.js，只需了解 JavaScript 即可轻松创建 3D 内容，还可以轻松处理它们的关系及运动效果。

该项目的虚拟形象渲染部分基于 Three.js 完成。

2.5 WebSocket 基于 TCP 的全双工网络传输协议

WebSocket 是一种计算机通信协议，通过单个 TCP 连接提供全双工通信通道，在 2011 年被 IETF 标准化为 RFC 6455^[10]。其中 ws 与 wss 分别代表非加密的 WebSocket 和加密的 WebSocket，并作为协议标识符使用。

该项目的动作数据和虚拟形象转发部分采用基于 WebSocket 设计的通信协议完成。

2.6 OBS 开源直播软件

开源直播软件(Open Broadcast Software, OBS)是一个专为高效捕捉、合成、编码、录制和流式传输视频内容而设计的软件。它是一个跨平台软件，适用于 Windows、macOS、Linux 和 BSD。OBS 使用 C / C++ 编写并使用 Qt 构建，通过实时消息协议(Real-Time Messaging Protocol, RTMP)提供实时捕获、场景合成、录制、编码和广播。它可以将视频流式传输到任何支持 RTMP 的目的地，包括 YouTube、Twitch、Instagram、Facebook 和国内的 Bilibili、斗鱼、虎牙等。

该项目为 OBS 制定了接口，可以实时地将产生的虚拟形象导入至 OBS 用于录制、直播使用。

2.7 WebXR API

WebXR 是一个用于访问 XR 设备（其中的 X 代表任意的，R 为 Reality 的简写）的应用程序编程接口(Application Programming Interface, API)，它提供了访问增强现实(Augmented Reality, AR)和虚拟现实(Virtual Reality, VR)设备（例如 HTC Vive, Oculus Rift, Google Cardboard, HoloLens, Magic Leap 或开源虚拟现实）及其传感器的支持。它是 Immersive Web Community Group

的产品，该社区的贡献者来自 Google、Microsoft、Mozilla 等。其中，WebXR 的增强现实模块允许虚拟内容在显示给用户之前与现实世界环境保持一致。WebXR 使用 Google ARCore 为 Android 设备上的 Google Chrome 浏览器提供增强现实体验。

本论文的 4.4 章节将会使用 WebXR 接口实现 AR/VR 并展示在小米 9 手机上使用 AR 展示虚拟形象及其动作的效果。

2.8 开发环境 **VSCode** 与 **Copilot**

Visual Studio Code，通常也称为 VS Code，是 Microsoft 为 Windows、Linux 和 macOS 制作的源代码编辑器。GitHub Copilot 是由 Github 和 OpenAI 创造的 AI 工具^[12]，其基于 OpenAI Codex 模型，经过自然语言和数十亿行公共源码的训练，其中来源包含 Github 上的项目。该工具通过自动代码补全来帮助程序员们编写代码。并且作为 VSCode 等编辑器的插件形式供用户使用。

该项目的整个开发过程使用 VSCode 完成并使用 Copilot 作为智能自动补全工具，大大的提高了开发效率。

第三章 视频驱动虚拟角色动作的自动生成系统需求分析与设计

本章从工程化的角度，讲述了从需求分析到架构设计与系统实现的过程，并具体描述了如何实现需求分析中的需求，如何将算法进行应用，以及跨平台桌面应用程序的用户图形界面系统的设计。

3.1 视频驱动虚拟角色动作的自动生成系统需求分析

该部分将从功能需求、非功能性需求（性能及易用性）方面对该系统进行分析，并且给出了该系统对运行环境的要求。

3.1.1 视频驱动虚拟角色动作的自动生成系统功能需求

该系统分为虚拟形象管理，虚拟形象动作生成，虚拟形象动作转发三大功能模块。

其中虚拟形象管理模块功能如下：

- 导入不同格式不同骨骼结构的虚拟形象，如 VRM/GLB/GTLF/FBX 格式的虚拟形象 3D 模型文件。
- 修改虚拟形象的信息或删除虚拟形象，如重命名、修改图片等。
- 查看及编辑虚拟形象骨骼位置信息，如查看虚拟形象都包含有哪些骨骼并手动改变他们的姿态等。
- 对虚拟形象骨骼绑定所使用的骨骼映射表和绑定函数进行查看和修改，可用于导入非标准的骨骼样式的虚拟形象
- 在三维空间内查看 3D 虚拟形象的外观

虚拟形象动作生成模块功能需求如下：

- 从视频中逐帧提取人物的三维骨骼信息，可使用摄像头或视频文件作为视频源
- 将视频中提取到的骨骼关键点位置信息转换为骨骼姿态信息
- 使用提取到的姿态信息驱动用户所选择的虚拟形象
- 将虚拟形象实时渲染到图形界面上

虚拟形象动作转发模块功能需求如下：

- 将生成的虚拟形象通过网络转发浏览器、AR/VR、直播软件
- 可以使用浏览器查看生成的虚拟形象及动作，可用于通过网络远程查看虚拟形象及动作
- 可以在 OBS 直播系统中读取生成的虚拟形象及动作，可用于在国内外主流视频直播平台上将产生的虚拟形象及动作实时直播。
- 可以在 AR/VR 设备上使用生成的虚拟形象及动作

图 3-1 为对上述功能需求进行分析后得到的系统用例图。

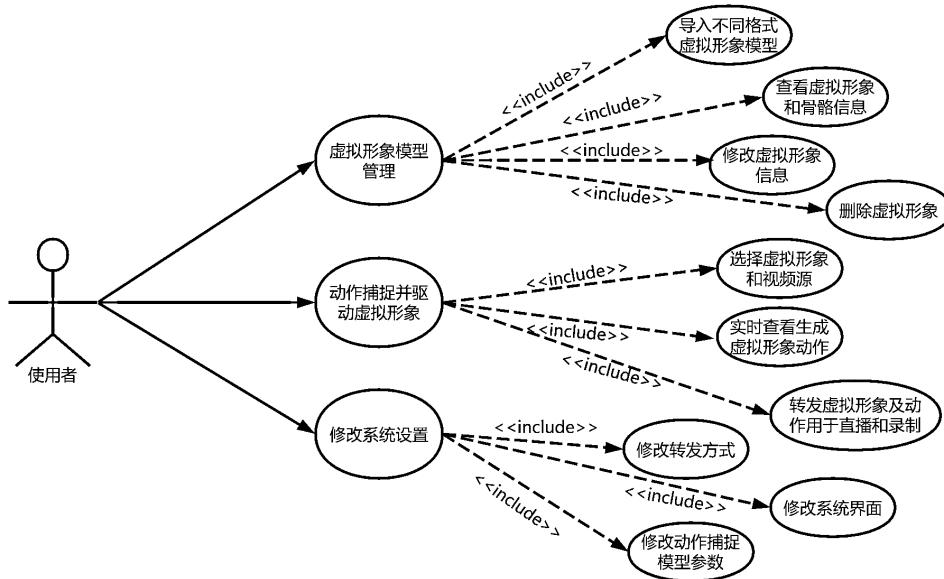


图 3-1 系统用例图

对于导入虚拟形象而言，表 3-1 中展示了其用例规约。

表 3-1 导入虚拟形象用例规约

用例规约	
用例名称	导入虚拟形象
主要参与者	用户
前置条件	下载好或创建好符合格式规定的虚拟形象模型
后置条件	添加成功后在页面中显示该虚拟形象
基本事件流	1) 进入添加虚拟形象的界面 2) 选择虚拟形象模型文件 3) 输入虚拟形象名称 4) 选择是否需要添加封面图片并选择封面图片
扩展事件流	步骤 2 中若虚拟形象格式不符合标准，系统提示模型文件格式错误
非功能性需求	界面美观，操作简单
优先级	高

对于动作捕捉并驱动虚拟形象而言，表 3-2 中展示了其用例规约。

表 3-2 动作捕捉并驱动虚拟形象用例规约

用例规约	
用例名称	动作捕捉并驱动虚拟形象
主要参与者	用户
前置条件	用户具备摄像头或准备好含人物的视频文件
后置条件	在界面中显示虚拟形象并展示实时动作

基本事件流	1) 选择内建的虚拟形象或用户导入的虚拟形象 2) 选择摄像头或视频文件 3) 开始动作捕捉 4) 驱动虚拟形象
扩展事件流	步骤 2 中若操作系统需要摄像头权限，则弹出授权对话框
非功能性需求	界面美观，操作简单
优先级	高

3.1.2 视频驱动虚拟角色动作的自动生成系统非功能性需求

该系统由于可以用于直播使用，所以需要用于实时视频的性能。当延迟大于 600ms 的时人们能明显感受到延时和卡顿，故系统的实时视频处理延时应小于 600ms。所以在下文中选择算法时，本项目选择了为实时性能优化的 CNN（循环神经网络）的 TensorFlow Lite 模型，并使用 Mediapipe 对视频流进行实时机器学习处理。而大多数将 RGB 视频转换为三维动作数据的机器学习模型都依赖于 Python、CUDA 等环境，这无疑在无形中的大大提升了用户的使用门槛，为此针对易用性方面，本项目提出了可开箱即用、配备有好的用户图形界面的需求。

3.2 运行环境要求

该软件为跨平台的桌面软件，即可运行在 Windows、macOS、Linux 系统中，对于 Windows 和 macOS 该系统提供了打包好的程序包，无需安装依赖及运行环境库，即可直接使用。对于 Linux 系统，由于其版本众多，难以提供打包好的可直接使用的程序包，但该系统可以在开发环境中进行运行，仅在 Ubuntu 22.04 版本上通过测试。

对于 Windows 平台，系统要求如下：

- OS: Windows 7 / 8.1 / 10 / 11
- CPU: Intel Core i5 6th gen 或更高 / AMD Ryzen 5 3rd gen 或更高
- 内存: 8 GB RAM 或更高
- GPU: Intel Iris Graphics 540 (AMD / NVIDIA 同等规格) 或更高
- 存储: 不少于 10 GB 可用空间

对于 macOS 平台，系统要求如下：

- OS: macOS 10.13 或更高
- 型号: 2015 年及之后
- CPU: Intel Core i5 6th gen 或更高 / Apple Silicon M-Series
- 内存: 8 GB RAM 或更高
- 存储: 不少于 10 GB 可用空间

对于 Windows / macOS / Linux 平台，需要开发（修改代码、编译打包等），需要安装 Node.js 14 或更高版本。为了更好的使用体验，建议使用配备独立显卡的设备运行。无论何种系统，都应该正确安装各种硬件的驱动程序（尤其是显卡驱动）来确保本系统可以正常运行。

3.3 视频驱动虚拟角色动作的自动生成系统总体设计

为了开发的便捷性及系统的可移植性考虑，并满足 3.1.1 中的各模块功能需求，本系统设计了一个具备了可扩展性和可移植性的模块化结构。图 3-2 展示了该系统各个模块及各模块之间的调用关系。

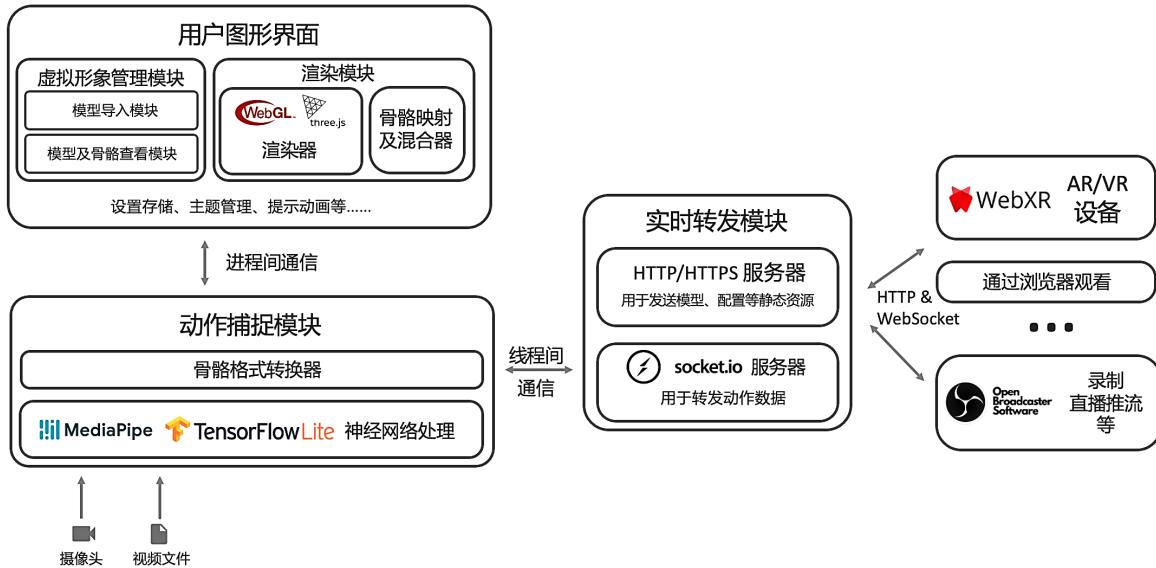


图 3-2 总体设计示意图

其中“动作捕捉模块”设计与实现在 3.4 中进行阐述，所使用的算法及准确度评估在第五章中进行阐述；“实时转发模块”设计与实现在 3.8 中进行阐述；“图形用户界面”中的“虚拟形象管理模块”部分在 3.5 中进行阐述，“渲染模块”部分在 3.6 中进行阐述，其余界面设计在 3.7 中进行阐述。

3.4 动作捕捉模块设计

动作捕捉模块采用为实时性能优化的事件驱动设计。摄像头和视频文件解码器作为事件源，每获取到一帧画面后会产生一个事件，将事件发送给神经网络进行计算并提取其中的每个骨骼点的位置。由于为了实时性考虑，神经网络只读取最近一帧画面，如果由于性能不足等情况，未能处理的视频帧将被丢弃。然后将 BlazePose 标准的骨骼节点的三维坐标通过插值算法映射到中间骨骼节点，经过去抖动及时域帧间插值（为了弥补之前丢弃的视频帧所带来的不连贯，通常人体动作是连贯的，此处使用帧内线性插值）后将动作数据以 JSON 格式发送给渲染引擎和动作转发服务器，此部分算法在第四章会进行详细讲解。图 3-3 展示了模块的整个工作流程。

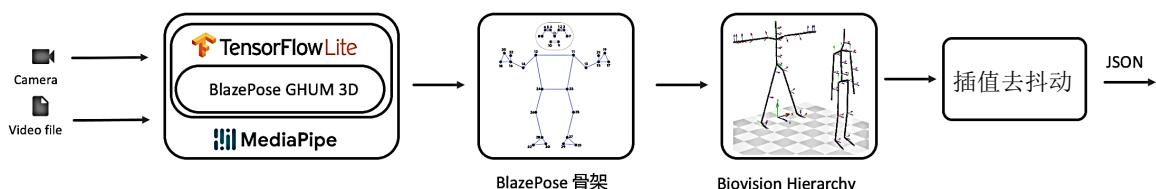


图 3-3 动作捕捉模块工作流程

该模块的工作过程可用事件驱动的有限状态机模型表示。图 3-4 为该状态机示意图，当启动动作捕捉模块时，模块会进入初始化状态，初始化成功后进入就绪状态。当收到视频帧数据后，进入处理状态，此时进入处理中状态。在动作捕捉模块处于处理中状态时，如模块工作状态为实时模式时，新发来的视频帧数据会被丢弃，直到处理完成后重新回到就绪态才会处理新的视频帧。

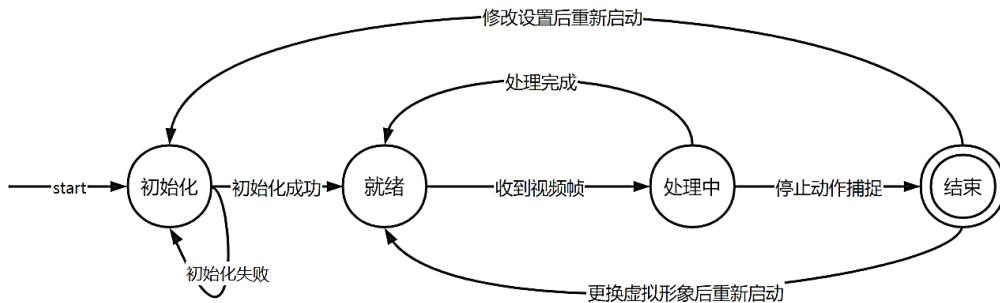


图 3-4 动作捕捉模块状态转移图

3.5 虚拟形象管理及展示模块设计与实现

虚拟形象管理模块支持导入及删除形象、查看和控制虚拟形象的骨骼、在非标准骨骼情况下手动修改骨骼映射、修改装饰物等功能。

其中导入部分支持导入 VRM、GLB/GLTF、FBX 格式的模型文件，要求模型文件具有骨骼信息。由于目前人型虚拟形象的骨骼命名方式没有统一的标准，各个厂商各个软件有各自的命名格式，所以本项目选择了几种有代表性的命名规范和格式，对这几种骨骼格式和命名方案进行了适配。

首先适配的是从 VRoid Studio¹ 软件创建的虚拟形象模型，VRoid Studio 是一款可以制作人形虚拟形象（角色）的 3D 模型的软件。该款软件兼容于 Windows 与 Mac，并且任何人都可以免费使用。VRoid 的主要特征就是通过类似绘画、捏脸、换装的方式进行人物的建模，使用者无需专业的三维建模知识，只需要从各种预设中选择自己喜欢的预设并调整参数即可。图 3-5 为用户使用 VRoid Studio 创建虚拟形象的画面。其真正的降低了创建属于自己的虚拟形象的门槛，这个理念和本课题中所设计的项目理念一致，所以最优先且最好的支持了该骨骼格式和命名方案。

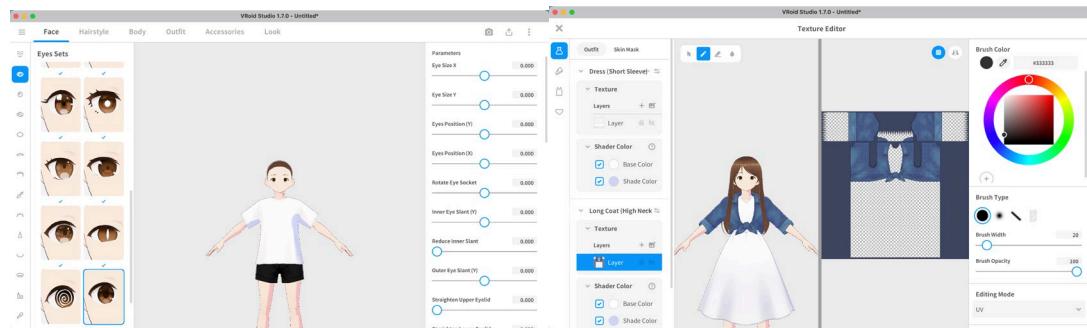


图 3-5 使用 VRoid Studio 创建虚拟形象

¹ <https://vroid.com/en/studio>

其次适配了 Mixamo 网站²生成的 FBX 格式。Mixamo 使用机器学习方法来自动执行角色动画处理的步骤，包括从 3D 建模到绑定和 3D 动画。他可以帮助用户快速的为用户创建的 T 字型 3D 形象自动添加骨骼信息，用户只需要简简单单的选择关节等关键点即可。图 3-6 为用户将自定义 3D 形象模型文件上传至 Mixamo 网站进行骨骼绑定的画面。

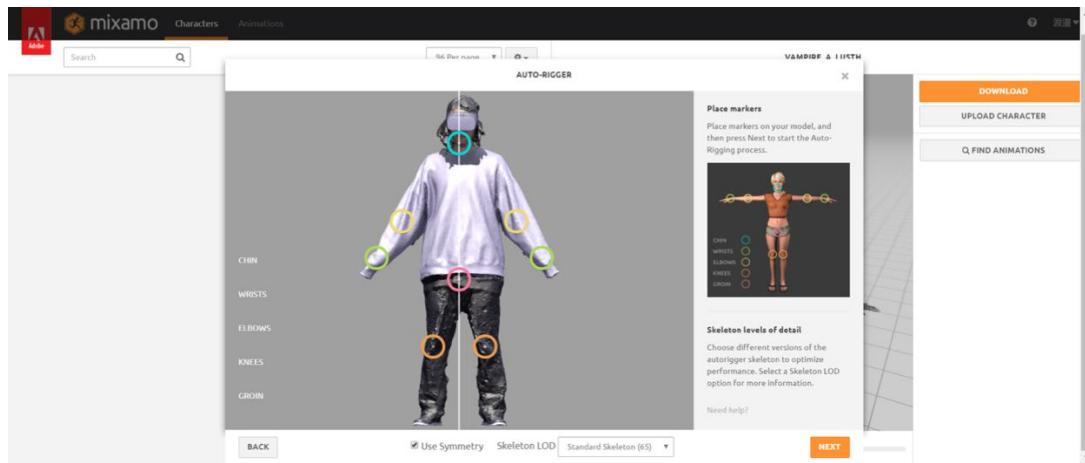


图 3-6 使用 Mixamo 进行自动骨骼绑定

此外，Mixamo 网站上还提供了大量可供用户免费使用的虚拟形象模型文件，用户只需创建 Adobe 账号，即可下载这些虚拟形象的 fbx 文件，方便用户获得到更多不同类型的专业虚拟形象，图 3-7 为 Mixamo 网站上可供下载的虚拟形象页面。

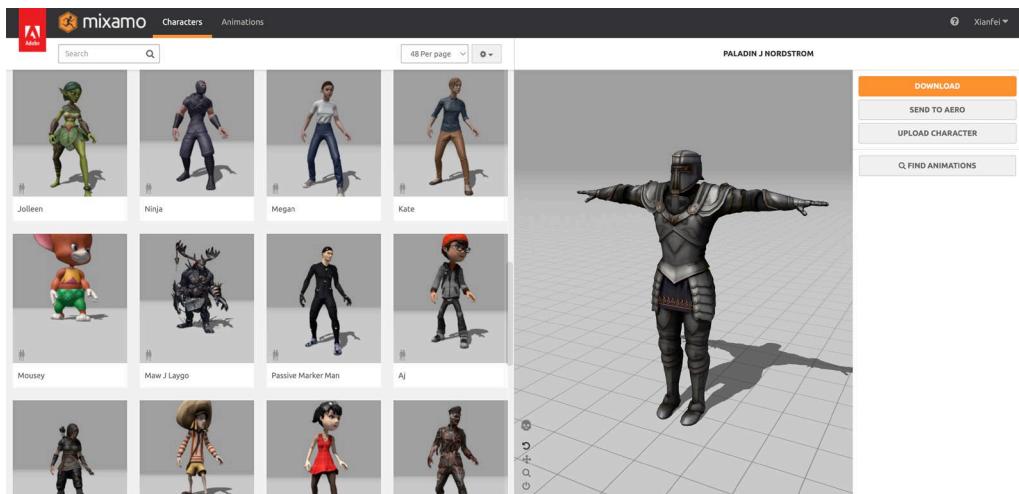


图 3-7 从 Mixamo 上获取虚拟形象

对于其他的虚拟形象格式文件，本项目也提供了手动骨骼映射工具，只需要在导入模型后点击虚拟形象进入虚拟形象展示模块，然后进入编辑骨骼绑定信息选项，即可进入骨骼映射页面，方便用户使用更多的从其他渠道获得的虚拟形象，提高了兼容性，图 3-8 展示了此过程。

² <https://www.mixamo.com/>

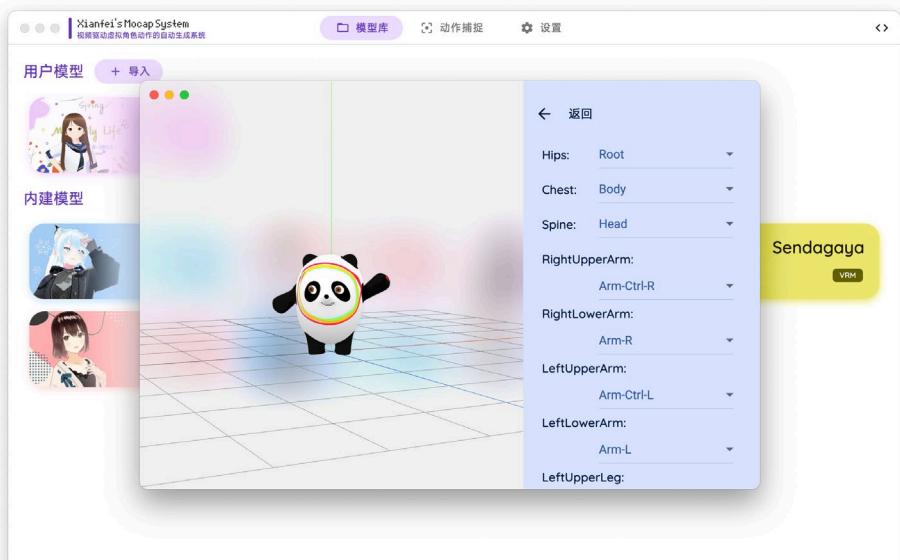


图 3-8 使用本项目自带的骨骼映射器手动映射

用户导入虚拟形象模型后，会将用户导入的虚拟形象以 JSON 的格式存储在用户目录 /sysmocap/profile.json 文件中。对于每个虚拟形象，表 3-3 展示了其所存储的信息。

表 3-3 模型文件中存储的信息

属性名	类型	作用
name	String	存储虚拟形象模型名称
type	String	存储虚拟形象模型类型
bgPic (可选)	String	存储虚拟形象模型封面的图片
path	String	存储虚拟形象模型的路径
accessories (可选)	[{String, String}]	用于存储换装信息，存储键值对表，键为装饰物名称，值为图层名称
cameraTarget (可选)	Vector3	用于存储默认虚拟相机所拍摄位置，用于绕物体移动相机
cameraRotation (可选)	Euler (Vector3)	用于存储默认虚拟相机姿态
cameraPosition (可选)	Vector3	用于存储默认虚拟相机位置
init (可选)	[{String, Vector3}]	用于存储模型的某些骨骼默认位置或姿态，存储键值对表，键为骨骼名
binding (可选)	[{String, {String, {Lambda,Lambda,Lambda}}}]	用于存储绑定信息，存储键值对表，键为中间骨骼格式名，值为映射骨骼名和三个坐标维度映射函数，见本文 4.4

同时该系统提供了一个可视化的 JSON 编辑器，方便用户更为直观方便的编辑虚拟形象的信息。图 3-9 展示了编辑器及模型包含的数据存储形式。

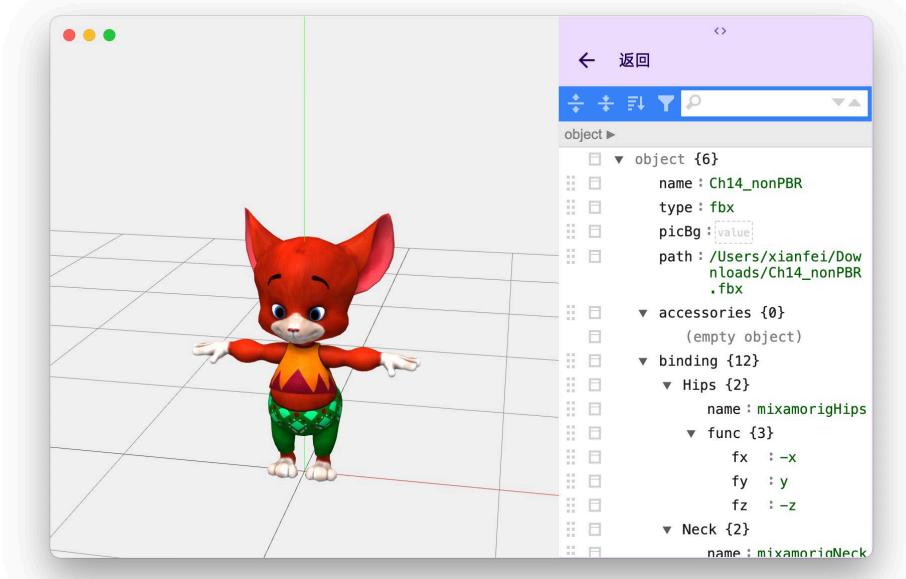


图 3-9 模型信息编辑器

除此之外，虚拟形象展示模块还具有查看和控制骨骼列表和位置，编辑模型三维图层和装饰物等功能，为用户提供了更高的可玩性。图 3-10 为打开虚拟形象后在三维图层装饰物编辑器中隐藏鞋子和外套的画面。中间黑色部分为骨骼的查看和控制器，用户可以了解到模型有哪些骨骼信息并手动操作它们。

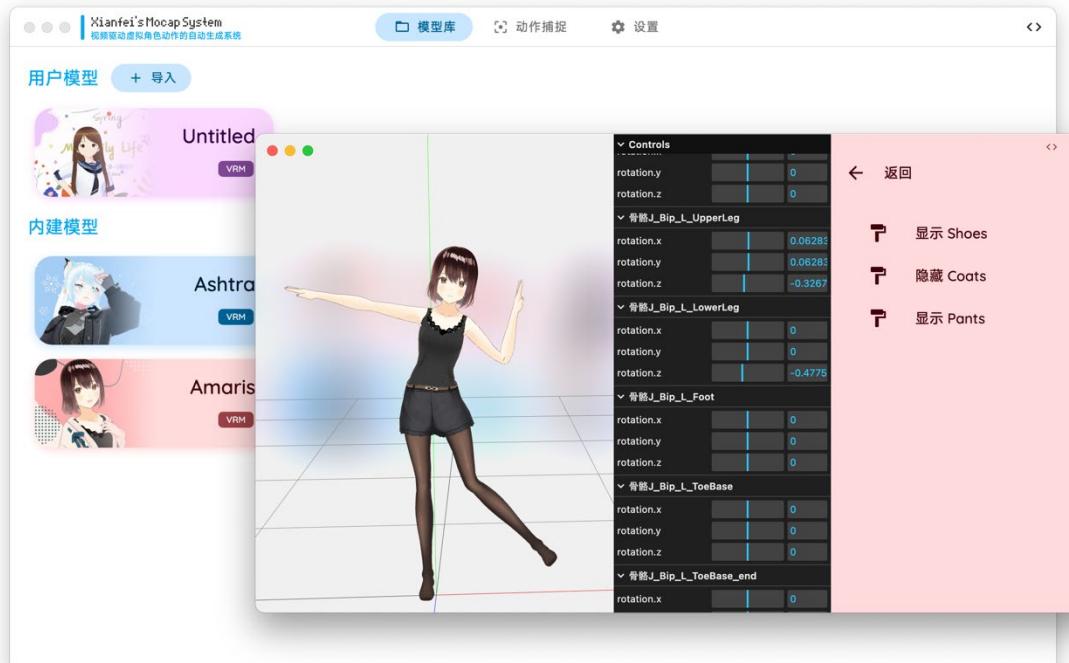


图 3-10 虚拟形象骨骼控制及三维图层控制页面

3.6 渲染模块设计与实现

对于渲染模块，该系统在选题初期做了许多探索性尝试，最早的时候尝试了虚幻引擎（Unreal Engine, UE），这是一个跨平台的主要用于游戏制作渲染引擎，它渲染的画面具有很好的画质，但是由于它太过庞大以及不易于嵌入至整个项目中（与虚拟形象管理等模块结合），加上其开发难度较大，于是在初步了解 UE 之后放弃了它。图 3-11 为在 UE 导入 Metahuman 虚拟形象并让其播放动画的画面。

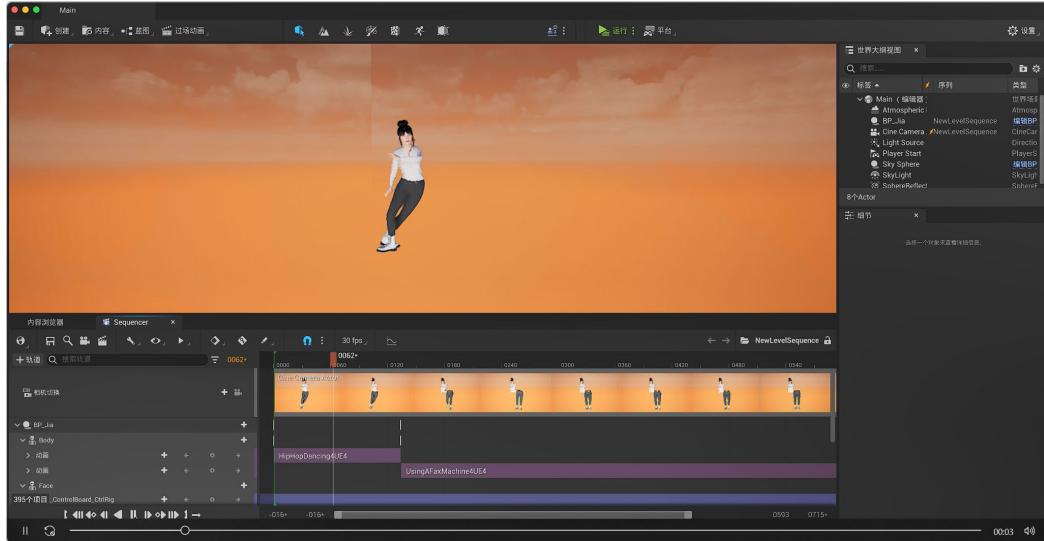


图 3-11 在 UE 中导入 Metahuman

此外，还尝试使用 Qt 作为整个 GUI 程序及渲染器的开发。但是 Qt 对于 3D 渲染的能力有限，图 3-12 为在 Qt 上渲染动作动画，但是其无法处理 3D 模型渲染的任务，所以放弃了基于 Qt 开发的方案。并且最终采用了 Three.js 作为虚拟形象渲染器的解决方案。

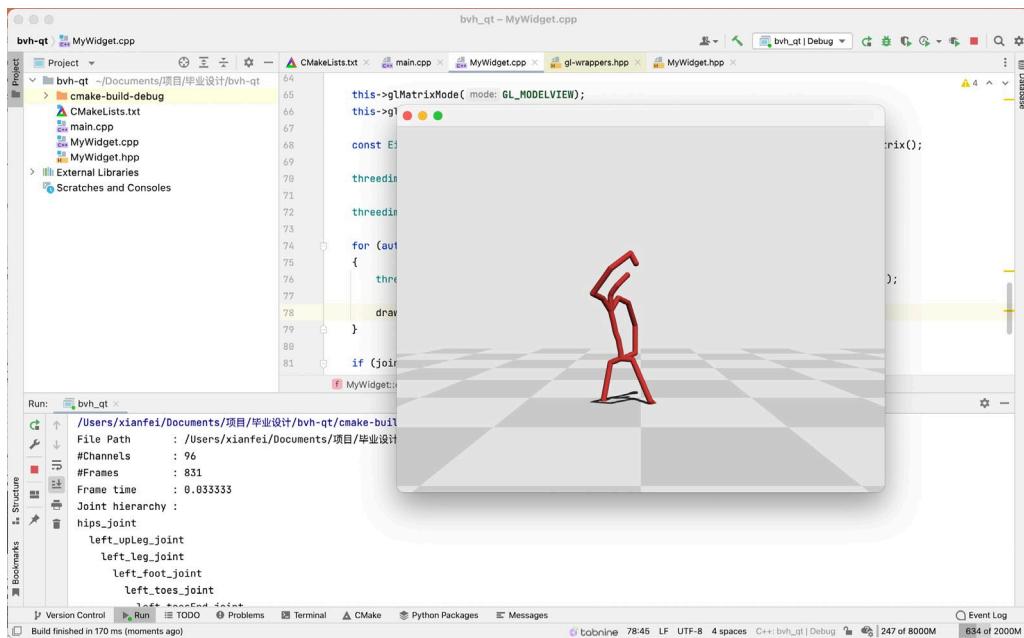


图 3-12 在 Qt 中渲染人体动作动画

Three. js 配合上 GLTFLoader、FBXLoader、Three-VRM 等模块，可以实现多种 3D 格式虚拟形象的导入（但是需要手动编写骨骼映射算法），并且其提供了多种灯光、环境模块，可以在仅需编写较少代码的情况下实现多种灯光效果。

上一小节提到的虚拟形象展示模块与下一小节提到的动作捕捉页面的虚拟形象实时预览都是基于 Three. js 进行渲染的。得益于它属于 Web 技术有着十分良好的可移植性及兼容性，后文中所介绍的 OBS 插件及 VR/AR 展示也是基于 Three. js 作为渲染器完成的。此外，还为渲染器设置了虚拟摄像机，可以通过鼠标键盘更改虚拟摄像机位置，实现调节虚拟形象的显示视角。

3.7 图形用户界面设计与实现

该系统在用户图形界面方面使用 Material Design 设计语言，借助于 Material Design Color Utils 色彩搭配及自动取色算法，以及毛玻璃窗口背景效果，在保持了界面干净简洁的同时，制造出了色彩感和科技感。且 UI 部分进行了大量的适配工作，确保在 macOS 和 Windows 系统上都能有着近乎一致、自然的视觉效果（如 macOS 中关闭、最小化、最大化窗口按钮在左上角，而 Windows 中则在右上角）。主界面采用了标签页样式的设计，分成了“模型库”、“动作捕捉”、“设置”三个页面，整体程序的主色调根据设置页面中用户设置的主色调决定。以白色为底色的页面配上不同颜色的浅色系色块，突出了虚拟形象的主题色，让用户更轻松更容易的选择到想使用的虚拟形象。

在虚拟形象模型查看窗口中，整个窗口采取了以毛玻璃作为虚拟形象部分背景、以虚拟形象主色调作为控制器和信息页背景的设计，配合上主页面的色块设计，凸显出色彩感及科技所带来的美感。图 3-13 为该系统主页面并打开了两个虚拟形象预览窗口。

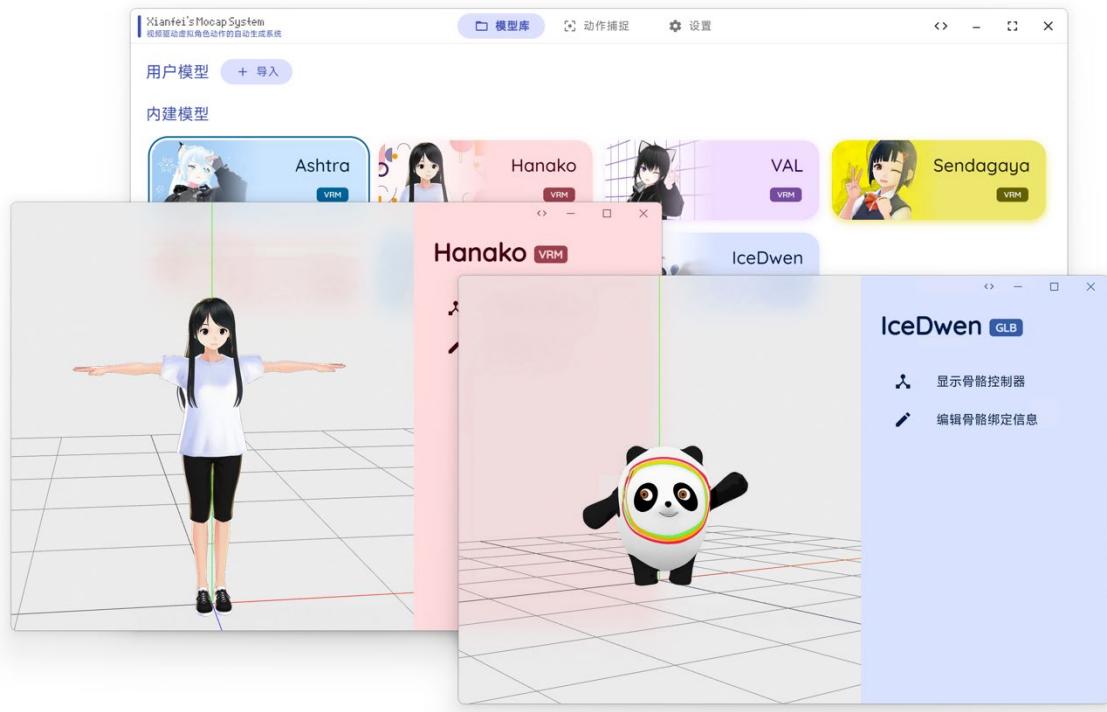


图 3-13 UI 设计

此外，该系统支持多语言，且目前支持了中文和英文两种语言，可在设置页面中自由切换无需

重启软件，方便了国际友人使用该软件。图 3-14 展示了不同语言下的设置页面。

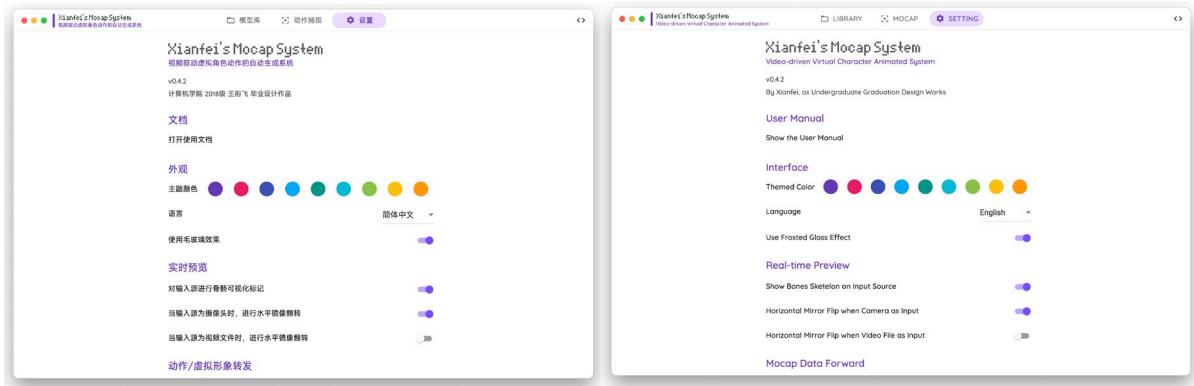


图 3-14 中文页面（左）与英文页面（右）

图 3-15 为动作捕捉页面，在动作捕捉页面中，渲染时可以实时显示当前系统的动作捕捉帧率及渲染帧率，以便于用户及时了解到当前系统的运行状态及性能消耗，该功能可以根据需要在设置中进行开启或关闭。此外在虚拟形象显示框中，用户可以使用鼠标拖拽虚拟形象改变虚拟摄像机的视角，也可以使用鼠标滚轮改变虚拟形象的大小，还可以使用键盘上的 W、A、S、D 按钮改变虚拟摄像机的位置，实现对虚拟形象的展示角度、大小等因素的自由控制。



图 3-15 动作捕捉页面

在模型库页面添加虚拟形象时，本项目设计了拖拽添加虚拟形象的方式，用户只需要将虚拟形象模型文件拖入程序中，然后修改名称（默认是文件名）、封面图片（可选）即可，大大简化了导入虚拟形象的流程。图 3-16 为导入虚拟形象页面。导入虚拟形象后，自动取色算法会根据导入的形象

封面计算出图片的主色调，用作该虚拟形象的主色调。此外该系统还使用了贝塞尔曲线设计了非线性动画，整个操作显得更加流畅和丝滑。



图 3-16 导入用户模型页面

并且，为模型库页面特别定制了专属的右键菜单，图 3-17 为右键菜单页面，通过右键可以对虚拟形象进行删除、在系统文件管理器中查看模型文件、设为动作捕捉时默认形象等操作，方便了用户的使用。右键菜单背景采用毛玻璃设计，打开和关闭时伴有非线性动画，更加的美观和丝滑。



图 3-17 右键菜单优化

该系统的大部分页面使用了 Vue.js 进行辅助开发，实现了 MVVM (Model View View Model) 模式的数据和视图之间的双向绑定。例如在设置页面，用户只需要更改设置，当 Vue 检测到设置更改会自动调用持久化函数来保存设置(设置以 JSON 文件的形式存储在用户目录的 sysmocap 文件夹下)，由于 MVVM 特性使得设置对 UI 的操控也随之生效。这不仅方便了开发，同时也提升了用户体验。UI 设计部分绝大多数效果、动画都是使用层叠样式表 (Cascading Style Sheets, CSS) 原生实现的 (该项目共编写了超过五百行 CSS 代码)。搭配上 Material Design Color Utils 的自动取色算法，实现了更美观更丰富的视觉效果。此外用户还可以在设置页面中指定该程序的主题色，未导入图片的模型将会使用主题色作为配色。

3.8 通信模块设计与实现

该部分设计了一个用于虚拟形象模型及虚拟形象动作的实时转发模块，并将其进行实现。采用 HTTP/HTTPS 作为模型文件及静态文件/配置文件的传输协议，使用基于 WebSocket 的 Socket.io 作为动作数据实时通信协议，实现了低延时高可靠的实时级性能需求，并且具备了一定的可扩展性，为下文中的适用于主流直播平台实时直播方案及自主开发的 AR/VR 虚拟形象实时直播功能打下了基础。图 3-18 为相关部分的设置页面。



图 3-18 动作及虚拟形象转发部分相关设置

表 3-4 中为所有静态文件/配置文件接口，可使用 HTTP 和 HTTPS 调用。

表 3-4 动作及形象转发 HTTP/HTTPS API 接口

路径	方法	返回数据
/	GET	含渲染器应用程序的页面
/model	GET	返回模型文件
/modelInfo	GET	返回的其他模型信息，见表 3-3
/useWebXR	GET	返回 WebXR 配置信息

该模块的动作转发协议使用 WebSocket 传输 JSON 格式的自己设计的可扩展数据结构进行通信，数据结构包含了数据包类型（分为动作数据、控制数据等），数据包类型使用“type”关键字进行表示。其中当“type”为“xf-sysmocap-data”（xf 为本人名字的首字母，sysmocap 为软件名，下同）时，传输的是动作数据；当“type”为“xf-sysmocap-control”是，传输的是控制数据。

图 3-19 中左侧为动作数据包的一级结构，动作数据包中包含人体骨骼信息“riggedPose”、面部信息及眼睛信息“riggedFace”、手部信息“riggedLeftHand”和“riggedRightHand”。右侧为控制数据包的一级结构，包含了虚拟摄像机的位置“cameraPosition”、姿态“cameraRotation”、缩放倍率“cameraZoom”。

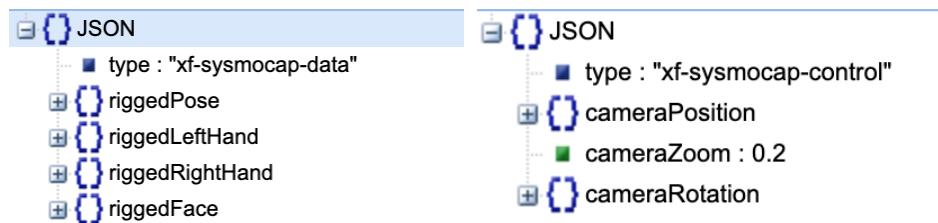


图 3-19 通信数据包一级结构

其中人体骨骼信息包含了身体中四肢和躯干的各个骨骼名称、旋转欧拉角信息；面部表情信息包含嘴巴形状、眼睛张开程度、瞳孔朝向、头部位置及旋转信息；手部包含了各个左手和右手的各个关节及其旋转信息。图 3-20 中展示了身体、手部、面部动作信息详细数据格式。



图 3-20 身体、手部、面部动作信息详细数据格式

通过这些通信数据，可以实时的控制虚拟形象的各个骨骼的位置及面部信息，以及虚拟摄像机的位置姿态及缩放。

由于 AR/VR 所使用的 WebXR 接口要求传输必须是安全连接（HTTPS & WSS），所以需要设计一款同时支持 HTTP/HTTPS/WS/WSS 的服务器软件（由于 HTTPS 会遇到证书错误需要手动确认，且 HTTPS 不能访问非安全资源），且希望能在只占用一个端口的情况下完成这些功能。因此，图 4-2 左侧为使用 Wireshark 抓取 HTTPS 握手数据包，其中图中偏下的蓝色部分为 TLS 握手部分，可以看出，HTTPS 握手数据包在 TCP 报文的内容的第一个字节是 0x16。而 HTTP 请求则通过明文传输 HTTP 请求头，HTTP 请求头的第一部分为请求协议（GET、POST 等），由于明文传输字符串的特性，其第一位不会是 ASCII 码小于 0x20 的控制字符。所以可以编写判断逻辑，如果第一个字节是 0x16（十进制 22）则建立 HTTPS 连接，如果第一位在 0x20 至 0x7F 之间则建立 HTTP 连接。图 4-2 右侧为使用代码的实现过程。此外，该系统内置了一个使用 OpenSSL 创建 CA（Certification Authority，认证机构）证书并创建自签名证书用于 HTTPS 证书，但是连接时需要手动确认该证书无效或将 CA 证书安装至系统。

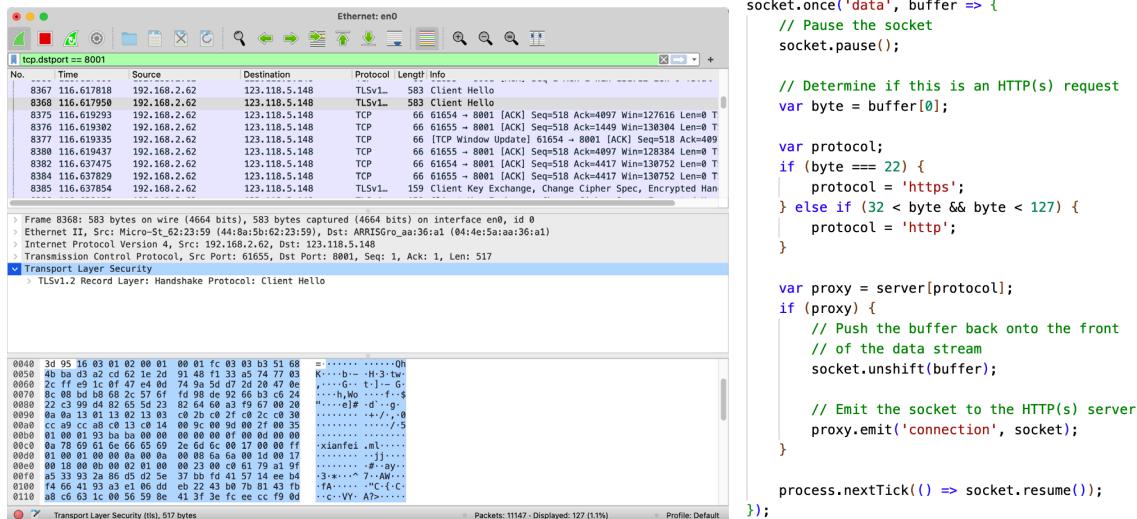


图 3-21 对 HTTPS 数据包分析（左）及服务器实现（右）

第四章 动作捕捉算法及虚拟形象绑定算法

本章节分为算法研究和算法实现两个部分，4.1 与 4.2 小节主要讲述了对于动作捕捉算法性能、效果等方面进行研究，通过对比实验的方式选择了最适合本系统使用的视频驱动无穿戴式动作捕捉算法并对其进行了不同视频场景不同设备不同操作系统环境下的性能测试；4.3 与 4.4 小节讨论了如何将动作捕捉算法所产生的骨骼点的动作数据应用于驱动虚拟形象，并设计了相应的算法。

4.1 视频驱动的动作捕捉算法研究

视频驱动的动作捕捉算法用于将视频数据转换为骨骼的动作数据。目前 Github 上开源的视频驱动的三维动作捕捉算法有不少，如 TCMR (Beyond Static Features for Temporally Consistent 3D Human Pose and Shape from a Video)^[13]、PyMAF (3D Human Pose and Shape Regression with Pyramidal Mesh Alignment Feedback Loop)^[15]、MocapNET^[16]及该项目中使用的 BlazePose GHUM 3D (下文简写为 BlazePose 3D)。

下面对上文提到的四种算法进行测试与对比。其中 TCMR 与 PyMAF 算法为非实时算法，只能通过输入视频文件后得到动作信息，而 MocapNET 和 BlazePose 3D 算法为实时算法，可以使用摄像头或视频文件作为输入源。

在算法选择测试中，TCMR 与 PyMAF 的测试环境为 Google Colab 在线机器学习平台，BlazePose 3D 与 MocapNET 测试环境为本机，其中 BlazePose 3D 运行 Heavy 模型。本机的硬件配置为 CPU: Intel Core i7-12700H / RAM: 16G / GPU: Intel Iris Xe Graphics / OS: Win11 & Ubuntu 18.04 WSL。此处使用了一段 Bilibili UP 主“慕慕有奶糖”的长度为 1 分 22 秒的舞蹈视频³作为测试，因为舞蹈视频通常包含了较多的复杂动作，能够更好的测试算法的性能。对于非实时算法以总耗时记录，对于实时算法以帧率记录，测试进行五次取平均值作为结果。图 4-1 为在本机运行 MocapNET 算法测试时的实时输出。

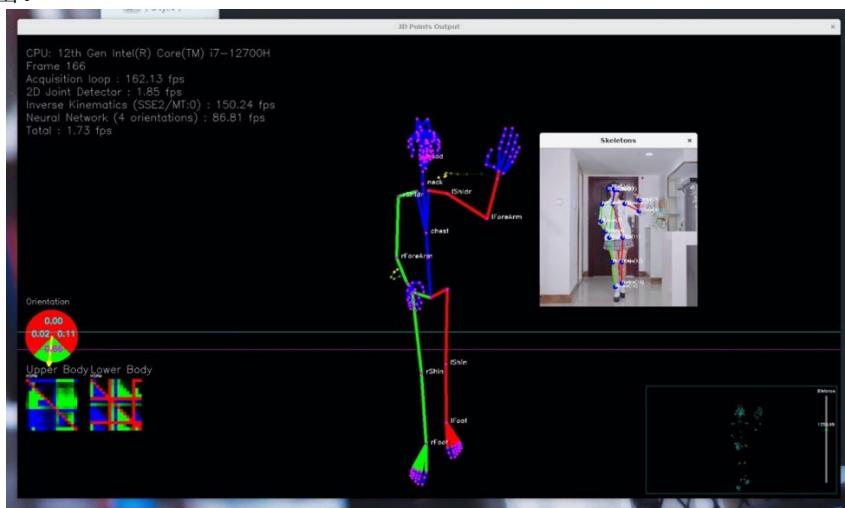


图 4-1 MocapNet 实时输出

³ <https://www.bilibili.com/video/av766736553>

测试结果见表 4-1，从中可以看出，TCMR 与 PyMAF 非实时算法所消耗的时间远远长于实时算法（此测试结果也可能与 Colab 免费版性能有关，但都有 NVIDIA Tesla 专业 GPU 加速），但是即使这两种算法消耗了更长的时间，在舞蹈中的复杂动作中对于四肢与躯干的前后位置关系也会有一些错误（但好于两种实时算法），图 4-2 中右手与身体的前后位置关系推断出现一些问题，视频中的左手在身体后面，而生成的人物模型的左手在身体前面，而图 4-3 中在左手位置推断也有些许问题，原视频中人物左臂位于身后。这四种算法均在不同程度上遇到此类问题。

表 4-1 不同种算法的速度对比

算法	TCMR	PyMAF	MocapNET	BlazePose 3D
运行环境	Google Colab			本机
耗时/帧率	19.82 min	23.26 min	2.3 fps	12.8 fps



图 4-2 右手与身体前后关系推断不准确示意图



图 4-3 左手位置推断不准确示意图

通常情况下，算法的准确度、速度和硬件需求是无法兼得的三个方面，能够较为准确的识别动作的算法通常需要花费更多的处理时间，而精准的实时算法有需要更好的硬件（比如 GPU）。该项目希望降低用户的使用门槛，在当下的数字货币计算的热潮中，高性能独立显卡是一种稀缺资源，价格较高，且现阶段购买如 RTX30xx 系列独立显卡通常会购买到翻新的重度使用过的显卡（俗称“矿卡”，此处的“矿”指的是用于数字货币计算的挖矿行为），恰好本人手里也没有较新的高性能独立显卡，所以本课题希望选择一种使用普通的集成显卡或一般的独立显卡就可以运行的算法。

4.2 BlazePose GHUM 3D 算法及性能评估

综上所述，BlazePose GHUM 3D 算法是目前在普通非专业计算机中综合性能最好的算法。

BlazePose GHUM 3D 由 BlazePose 算法和 GHUM 3D 算法组合而成，前者用于评估二维图像中的关节点信息，如图 4-4 右侧图片；后者用于从二维视频中创建三维人体模型（三维重建），如图 4-4 左侧图片。该算法所生成的骨骼关键点信息的二维数据（x、y 坐标）由 BlazePose 算法活动，深度信息（z 坐标）通过 GHUM 3D 数据投影到 2D 点的得到。该算法包含三个复杂度的卷积神经网络（Convolutional Neural Network, CNN）模型，分别为 Lite、Full、Heavy。

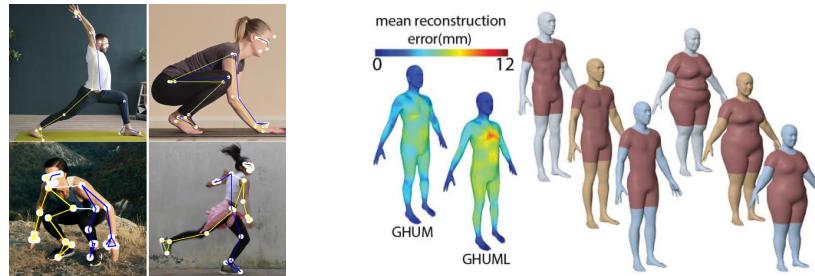


图 4-4 BlazePose 示意图（左）与 GHUM 示意图（右）

此外，BlazePose 算法还可与实时手部检测算法 BlazePalm^[17]、实时面部检测算法 BlazeFace^[18]一同使用，实现对虚拟形象的面部、手部动作生成。BlazePalm 算法可以实时检测手部各个关节信息，BlazeFace 算法可以实时的检测面部轮廓及关节点。通常还需要辅以表情识别算法来确定视频中人物的表情来达到更好虚拟形象的面部效果^[19]。图 4-5 为这两种算法的示意图。



图 4-5 BlazePalm 示意图（左）与 BlazeFace 示意图（右）

根据 Google Research 对 BlazePose GHUM 3D 通过 1400 个样本均匀分布跨越 14 个地理区域的准确性分析⁴，该模型对各个地区的人的识别准确率都很高。表 4-2 表示了不同复杂度的模型（Lite 为较低复杂度，Full 为中等复杂度，Heavy 为较高复杂度）其对不同地区的人的检测点准确度百分比（Percentage of Detected Joints, PDJ）及标准差。公式 4-1 为 PDJ（记为 \bar{x} ）计算公式，PDJ 由每个人识别正确的关节数（记为 b_i ）比上所有识别的关节数（记为 a_i ）的平均值得到，其中 n 为样本个数（ $n = 1400$ ）；公式 4-2 为识别准确度标准差（记为 S_x ）计算公式，每个人识别正确的关节数（记为 b_i ）比上所有识别的关节点数（记为 a_i ）减去 PDJ(\bar{x}) 的平方之和除以样本总数后开平方得到。

$$\bar{x} = \frac{\sum_{i=1}^n \frac{b_i}{a_i}}{n} \quad (\text{公式 4-1})$$

$$S_x = \sqrt{\frac{\sum_{i=1}^n (\frac{b_i}{a_i} - \bar{x})^2}{n-1}} \quad (\text{公式 4-2})$$

⁴ https://drive.google.com/file/d/10WlcTvrQnR_R2TdTmKw0nkyRLqrwNkWU/

表 4-2 BlazePose GHUM 3D 对 14 中不同地区人的识别准确度

地区	Lite model		Full model		Heavy model	
	PDJ	标准差	PDJ	标准差	PDJ	标准差
澳洲	94.1	8.3	95.4	7.6	98	3.9
加勒比海	94.8	8.3	97.7	4.9	98.9	2.9
欧洲	90.3	12.6	95.1	8.4	97.8	4.8
北非	94.7	8.2	97.7	5.2	98.9	3.3
南美洲	95	8.3	97.8	4.8	99	3
东南亚	94.5	8.1	97.1	5.1	98.5	3.9
西亚	94.9	7.8	97.7	5.6	99	2.9
中美洲	95.4	6.5	97.6	4.5	98.6	2.9
中亚	94.3	8.7	96.8	5.4	97.9	4.7
东亚	91.3	12.5	94.6	8.3	97.4	5.1
中非	94.6	9.5	96.6	6.6	98.3	4.6
北美	93.4	8.6	96.2	6.3	98.7	3.3
南非	92.9	10	95.1	6.9	97	6.3
南亚	93.4	9	96.8	6.2	98.2	4.5
平均值	93.8		96.6		98.3	
波动范围	5.1		3.2		2	

从测试结果可知，使用更高复杂度的模型能获得更高检测点准确度，且能获得更小的波动范围，且该模型对于中国人所属的东亚地区人种识别准确度均低于平均值，所以在本项目中默认使用较高复杂度模型。

此外，其还对该模型对于不同肤色、不同性别的人群进行准确度测试。表 5-3 为对不同种肤色的人进行准确度测试，色调数值越大肤色越深，数值越小肤色越浅。可以从中得出不同肤色的人对于该算法都能较好的识别，且从识别结果来看肤色的深浅与识别准确度并无明确的关系，但似乎肤色色调为 2 的检测点准确度最高，且模型复杂度越高的算法对不同种色调的肤色的人检测点准确度波动范围越小。表 4-4 为对不同性别的人进行准确度测试，可以看出无论在何种复杂度的模型下，对于女性的检测点准确度要低于男性，但差距并不明显，平均波动范围百分比在 1%-1.6% 之间。

表 4-3 BlazePose GHUM 3D 对不同肤色的人识别准确度

肤色色调	所占比例	Lite model		Full model		Heavy model	
		PDJ	标准差	PDJ	标准差	PDJ	标准差
1	1.3	96.3	2.5	95.1	5.5	98.8	1.4
2	9.5	91.4	10.1	94.7	7.7	97.7	4.2
3	34.3	93.9	9.3	96.7	6.1	98.2	4.4
4	36.2	94.3	9	97	6.2	98.6	3.9
5	14.2	94.5	9.3	97.2	5.5	98.5	3.9
6	4.5	96.1	7.1	97.1	5.8	98.7	3.7
平均值		94.2		96.3		98.2	
波动范围		4.9		2.5		1.1	

表 4-4 BlazePose GHUM 3D 对不同性别的人识别准确度

性别	所占比例	Lite model		Full model		Heavy model	
		PDJ	标准差	PDJ	标准差	PDJ	标准差
男	45.9	94.7	9	97.3	5.67	98.9	3.5
女	54.1	93.1	9.5	96	6.8	97.9	4.7
平均值		93.9		96.7		98.4	
波动范围		1.6		1.3		1	

由此可见，该模型的准确率相当在当前的性能下有着较好的表现。由于本系统驱动虚拟形象骨骼时只使用旋转信息（因为模型与视频中人物身高、臂长、腿长等均不相同，使用位置信息驱动虚拟形象可能导致其变形、扭曲），所以即使动作捕捉的精度足够高，驱动虚拟形象时也仅能还原其大致位置。而第四章所提到的虚拟形象转发及相关应用需要依赖于实时性，且为了更好的展示效果还需要加入面部和手部检测算法，所以本课题需要对该模型（加入面部和手部检测算法）的组合性能进行评估。本文选取了三组不同硬件配置和操作系统的计算机对该软件性能进行对比，模型复杂度使用较高复杂度模型，具体参数如表 4-5 所示。

表 4-5 不同测试组的配置参数

	测试组 1	测试组 2	测试组 3	测试组 4
CPU	Intel Core i7-4790k	Intel Core i7-4790k	Intel Core i7-12700H	Intel Core i7-12700H
GPU	NVIDIA GTX 770	NVIDIA GTX 770	NVIDIA RTX 3060	Intel Iris Xe
RAM	16G DDR3	16G DDR3	16G DDR5	16G DDR5
OS	macOS 11.6.5	Windows 10	Windows 11	Windows 11

在本文所设计的程序中分别以 5.1 中所提及的舞蹈视频⁵（舞蹈视频通常包含有大量且复杂的动作，对模型的性能通常要求很高）、刘畊宏《本草纲目》健身操视频⁶（健身操相对于舞蹈的动作要更为简单，但是还是包含有大量动作，对模型的性能通常要求较高）、Bilibili 博主“浅影阿_”的唱歌视频⁷（唱歌视频主要在于上半身的面部动作，不包含下半身，为大多数虚拟主播常用的使用场景）作为测试数据进行测试，使用本程序自带的 FPS 性能监视器记录帧率，进行十次采样取平均值，测试过程如图 4-6，Mocap 下方的折线图上的 fps 数值即为动作捕捉帧数。

⁵ <https://www.bilibili.com/video/av766736553>

⁶ <https://www.bilibili.com/video/av640082825>

⁷ <https://www.bilibili.com/video/av683893544>



图 4-6 使用 FPS 监视器记录帧率

从表 4-6 的测试结果可以看出，在全身视频（健身操与跳舞）上性能没有明显差异。但是在半身视频上性能明显高于全身视频。并且运行时性能在 Windows 上要高于在 macOS 上（这可能与 macOS 上 NVIDIA 显卡也需要使用苹果 Metal API 有关，不能使用 NVIDIA 官方驱动及 CUDA）。总体效果配合上输出渲染时所使用的时域帧内差值算法（球面线性插值）后显示效果基本令人满意。

表 4-6 BlazePose 3D 算法在本程序中的性能

	测试组 1	测试组 2	测试组 3	测试组 4
《本草纲目》健身操	14.9fps	8.6fps	22.1fps	13.7fps
舞蹈视频	15.2fps	7.8fps	22.3fps	13.9fps
唱歌视频	18.1fps	12fps	25.6fps	15.2fps

4.3 动作捕捉骨骼架构转换算法

通过上文中图 4-4 可知 BlazePose 3D 算法输出的骨骼样式是不包含脊柱的，而绝大多数虚拟形象的骨骼是包含脊柱的，且通常只有骨干节点包含位置信息，其余节点只记录相对于父节点的旋转信息。而 Mediapipe 输出的人体骨骼信息是不包含脊柱，如图 4-7 中蓝色直线及以数字命名的点为 Mediapipe 的输出骨骼信息，红色虚线与蓝色虚线标出的就是我们需要估算出的脊柱骨骼，为了方便下文的表达，我们不妨将这两条虚线所创建的三个点记作 A、B、C。其中 AB 为点 11 与点 12 构成线段的垂直平分线，AC 为点 23 与点 24 构成线段的垂直平分线，此处设 AB 的长度等于 AC。点 A、点 B、点 11、点 12 共面，点 B、点 C、点 23、点 24 共面。由于 BlazePose 3D 在躯干上只有四个检测点(11, 12, 23, 24)，这肯定会导致无法检测类似于弯腰时腰背是否挺直的细节信息。

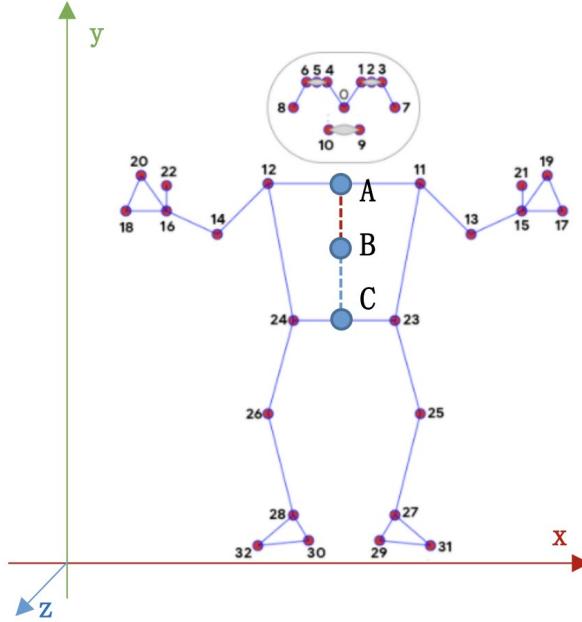


图 4-7 BlazePose 骨骼结构

在本论文中，我们令虚拟形象二维投影所处的平面为 xOy 平面，面朝方向为 z 轴正方向。如图 4-7 中的红蓝绿坐标轴。相对姿态使用欧拉角 Roll（横摆角或者翻滚角）、Pitch（俯仰角）、Yaw（偏航角或者航向角）表示。其与坐标轴的关系如图 4-8。

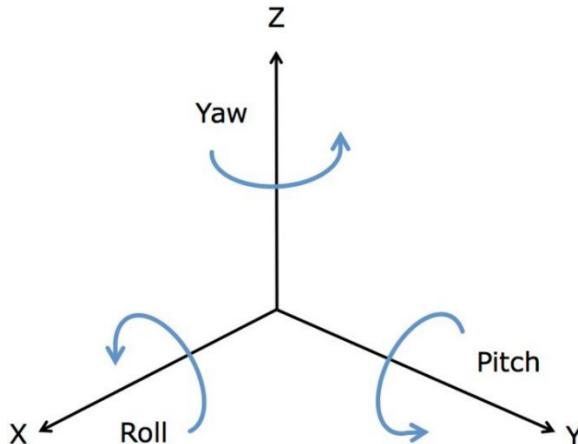


图 4-8 欧拉角关系

上文中已经提到，虚拟形象驱动时主要使用的是骨骼的相对旋转信息。所以，如何计算出这两个骨骼的相对旋转信息是能够使用 BlazePose 3D 可以用于驱动虚拟形象的关键。首先，根据 A、B 点是中点的信息，可以很容易的获得 A、B 点的坐标。在该转换算法中，令 BC 段骨骼为根 (Root) 骨骼。将点 23 与点 24 所构成的线段与 x 轴夹角记为 θ_x ，与 y 轴夹角记为 θ_y ，与 z 轴夹角记为 θ_z ；将点 11 与点 12 所构成的线段与 x 轴夹角记为 ϑ_x ，与 y 轴夹角记为 ϑ_y ，与 z 轴夹角记为 ϑ_z ；那么 BC 骨骼的姿态 Roll 为 $\frac{\pi}{2} - \theta_x$ ，Pitch 等于 $\vartheta_y - \theta_y$ 的一半，Yaw 为 θ_z 。骨骼 AB 相对于骨骼 BC 的姿态 Roll 为点 11 与点 12 所构成的线段与点 23 与点 24 所构成的线段投影到 xOy 平面的夹角为 0（由于躯干无法分析弯腰的特性），Pitch 为 $\vartheta_y - \theta_y$ 的一半，Yaw 为 $\vartheta_z - \theta_z$ 。通过此种计算即可得到脊柱骨骼信息。该算法使用伪代码表示如下：

骨骼架构转换算法 伪代码**算法 1：骨骼架构转换算法**

```

1. horizontalShoulder.rotation = getEuler(output[11], output[12]) // 获得点 11, 12 构成线段的姿态
2. horizontalHips.rotation = getEuler(output[23], output[24]) // 获得点 23, 24 构成线段的姿态
3. // 计算 BC 骨骼 (Hips) 姿态
4. Hips.rotation.roll = tan((pointA.position.z - pointC.position.z) / (pointA.position.y - pointC.position.y))
5. Hips.rotation.pitch = (horizontalShoulder.rotation.pitch - horizontalShoulder.rotation.pitch) / 2
6. Hips.rotation.yaw = PI / 2 - horizontalShoulder.rotation.yaw
7. // 计算 AB 骨骼 (Spine) 姿态
8. Spine.rotation.roll = 0
9. Spine.rotation.pitch = (horizontalShoulder.rotation.pitch - horizontalShoulder.rotation.pitch) / 2
10. Spine.rotation.yaw = horizontalShoulder.rotation.yaw - horizontalHips.rotation.yaw

```

此外，为了完成下一步的骨骼映射算法，我们规定此步骤输出一种中间骨骼形式。图 4-9 表示了这种骨骼形式，这种骨骼形式包含了 12 个关键骨骼，分别为 Hips（臀部）、Neck（颈部）、Chest（胸部）、Spine（脊柱）、RightUpperArm（右上臂）、RightLowerArm（右下臂）、LeftUpperArm（左上臂）、LeftLowerArm（左下臂）、LeftUpperLeg（左大腿）、LeftLowerLeg（左小腿）、RightUpperLeg（右大腿）、RightLowerLeg（右小腿），使用英文对他们进行命名。

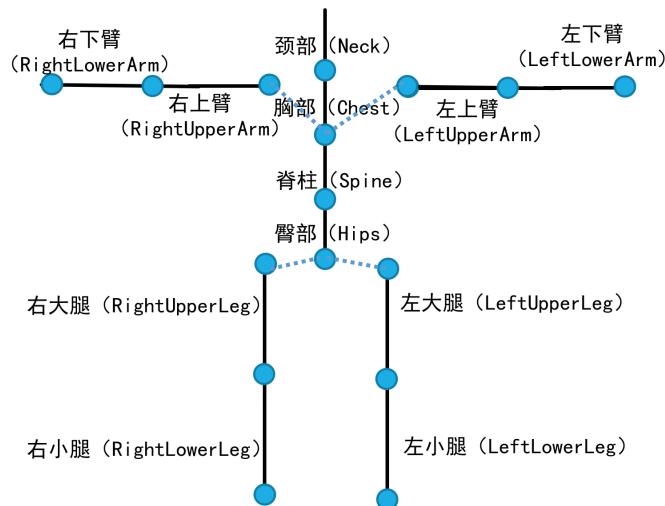


图 4-9 输出骨骼结构示意图

4.4 虚拟形象骨骼映射驱动算法

本论文 3.4 中曾提到，该系统支持导入 VRM、GLB/GLTF、FBX 格式，并且支持对 VRoid Studio 及 Mixamo 创建的带有标准骨骼的虚拟形象支持全自动绑定。在对虚拟形象的骨骼进行进一步研究

时，发现了不同虚拟形象的不同骨骼可能有着不一样的坐标系。图 4-10 中是三个具有不同骨骼类型但初始姿态相同（均为 T 字形）的虚拟角色，当试图将其右侧小臂向前伸时，发现对于不同的虚拟形象所需要执行的操作是完全不同的，分别是增加 y 轴夹角、减小 z 轴夹角和减少 y 轴夹角，究其原因是因为它们有着不同的坐标系。

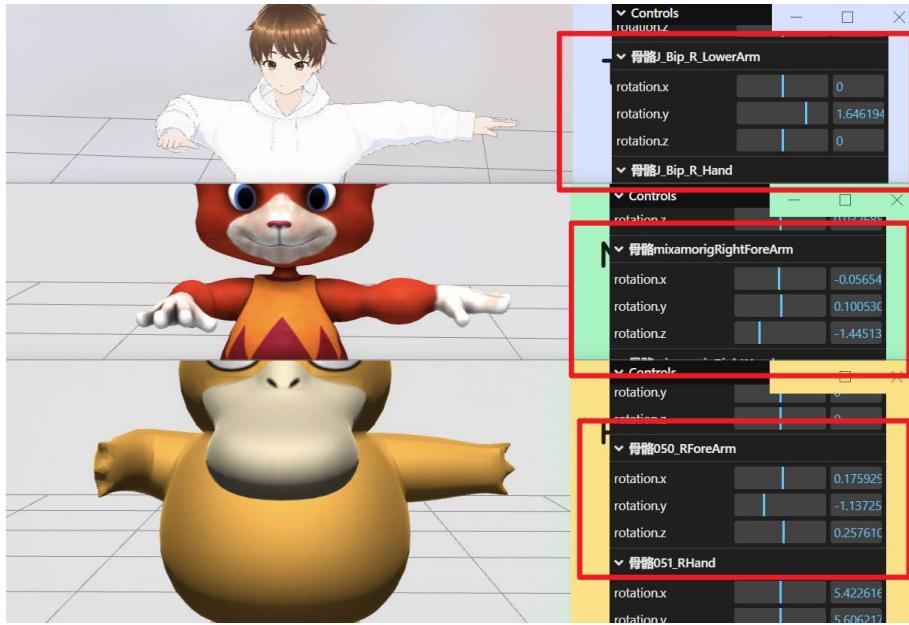


图 4-10 不同虚拟角色的同一动作的不同操作

为了实现对不同格式的不同骨骼架构的绑定，在上一小节我们设定了骨骼中间格式。而该小节主要讨论的是如何用骨骼中间形式驱动不同类型的虚拟形象。面对不同种骨骼名称和坐标系，本论文提出了一种基于键值对（Key-Value Pairs）的解决方案，用于将模型特有的骨骼形式映射到骨骼中间格式。其包含需要映射的骨骼名，以及三个映射函数 $f_1(x, y, z)$, $f_2(x, y, z)$, $f_3(x, y, z)$ ，用于坐标系重映射和基变换。对于不同类型的骨骼需要不同的映射表，其中表 4-7 为部分用于映射 Mixamo 标准骨骼的映射表。

表 4-7 将中间骨骼转换为 Mixamo 标准骨骼映射表

中间格式	目标骨骼	f_1	f_2	f_3
Hips	mixamorigHips	-x	y	-z
Neck	mixamorigNeck	-x	y	-z
Chest	mixamorigSpine2	-x	y	-z
Spine	mixamorigSpine	-x	y	-z
RightUpperArm	mixamorigRightArm	-z	x	-y
RightLowerArm	mixamorigRightForeArm	-z	x	-y
LeftUpperArm	mixamorigLeftArm	z	-x	-y
LeftLowerArm	mixamorigLeftForeArm	z	-x	-y
LeftUpperLeg	mixamorigLeftUpLeg	-x	y	-z
LeftLowerLeg	mixamorigLeftLeg	-x	y	-z

通过上述操作，可以将不同骨骼命名方式的模型统一成一致的命名规则和相同的坐标系。下面将讨论如何驱动不同格式的虚拟形象 3D 模型。对于 VRM 格式的虚拟形象 3D 模型，使用 VRM 对象中 humanoid.getBoneNode 方法来查找骨骼；对于 GLB/GLTF/FBX 格式的虚拟形象 3D 模型，首先创建 SkeletonHelper 对象用于管理虚拟形象 3D 模型的骨骼信息，然后遍历整个表通过名字匹配来查找驱动骨骼。根据其对应的骨骼名在映射表中查找到相应的骨骼，然后使用中缀表达式解析函数解析设定的函数表达式（表 4-7 中的 f1, f2, f3）后对目标骨骼的旋转角使用函数表达式计算。此外为了优化流畅度，解决动作捕捉性能通常低于渲染性能的问题，在修改目标骨骼的旋转角前，还进行了球面线性插值(Spherical linear interpolation, Slerp) 处理。公式 4-3 为球面线性插值公式。

$$qt = \text{slerp}(q_0, q_1, t) = a * q_0 + b * q_1 \quad (\text{公式 4-3})$$

其中 q_0, q_1 是相邻的两帧骨骼旋转角所对应的四元数，参数 $t \in [0,1]$ 为动作捕捉帧速率与渲染帧速率的比值（动作捕捉帧速率不会低于渲染帧速率）， $a = \sin(\alpha(1-t))/\sin\alpha$, $b = \sin(\alpha * t/\sin\alpha)$, $\cos\alpha = q_0 \cdot q_1$. 整个过程使用伪代码表示该算法如下：

骨骼映射驱动算法 伪代码

算法 2：骨骼映射驱动算法

```

1.   for outputBone in output.skeleton do // 遍历动作捕捉输出的转换后的骨骼信息
2.       targetBoneInfo = model.lookUpTable[targetBone.name] // 从查找表中读取骨骼
3.       for bones in virtualCharacter.bones do // 遍历虚拟形象具有的骨骼
4.           if bones.name == targetBoneInfo.name then // 如果骨骼匹配则调用中缀表达式进行映射
5.               euler.rotation.x = invoke(targetBoneInfo.func.f1) // 创建临时欧拉角
6.               euler.rotation.y = invoke(targetBoneInfo.func.f2)
7.               euler.rotation.z = invoke(targetBoneInfo.func.f3)
8.               // 将欧拉角转换为四元数后进行球面线性插值，t 为动作捕捉帧速率与渲染帧速率的比值
9.               bones.quaternion = Slerp(bones.quaternion, eulerToQuaternion(euler.rotation), t)
10.          end if
11.      end for
12.  end for

```

图 4-12 为驱动虚拟形象骨骼映射流程图。

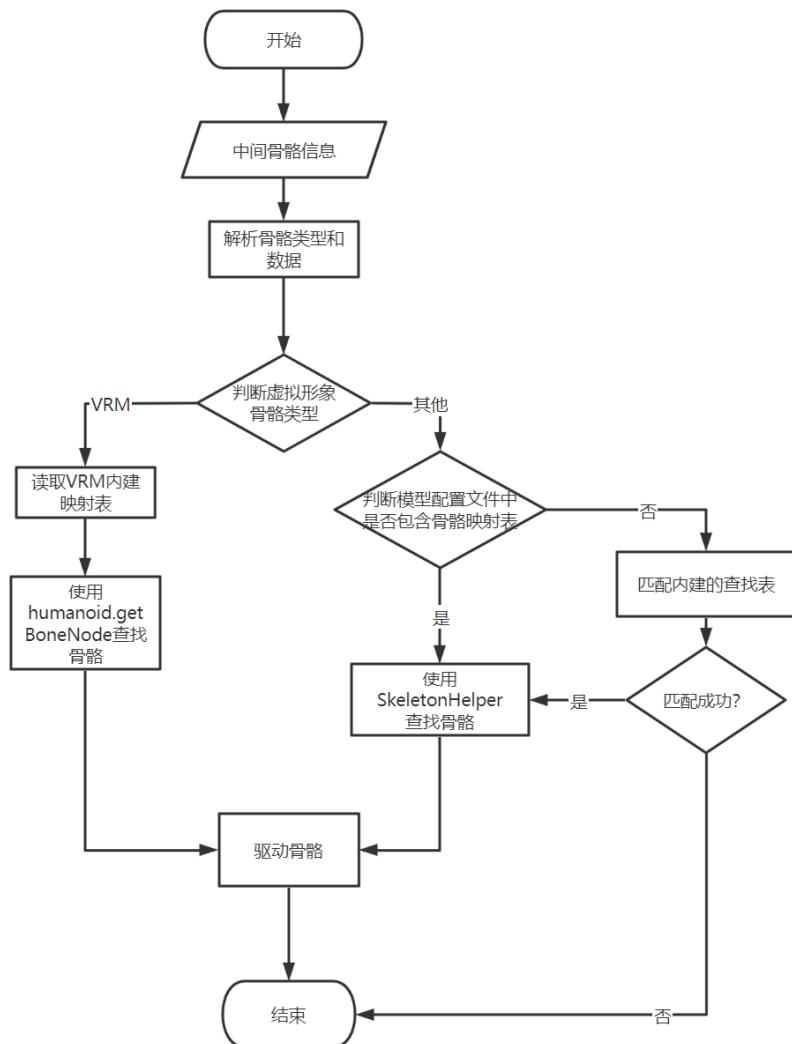


图 4-12 骨骼映射驱动流程图

第五章 视频驱动虚拟角色动作的自动生成系统的实际应用

本论文 1.1 小节中提到了该系统的应用前景，本章节具体讨论如何设计并实现相应的接口、模块等，可将该系统用于直播、影视制作等，并设计了前沿的在 VR/AR 设备上的三维虚拟形象直播功能。此部分需要在该系统设置页面中打开虚拟形象动作转发和 WebXR 支持选项。

5.1 用于 OBS 软件的直播、录制接口设计与实现

当下视频直播行业十分火爆，尤其是在疫情期间，视频直播的人数大增。而 OBS 作为一款功能齐全的开源直播软件，其不仅能输出到主流的视频直播平台，还可以当做虚拟摄像头使用。其支持将摄像头、视频采集卡、视频文件、屏幕捕捉等视频来源作为视频源，方便直播及录制使用。得益于 OBS 内置了 CEF (Chromium Embedded Framework) 浏览器框架，可以直接使用 HTTP 作为视频输入源。图 5-1 为此部分的流程图。

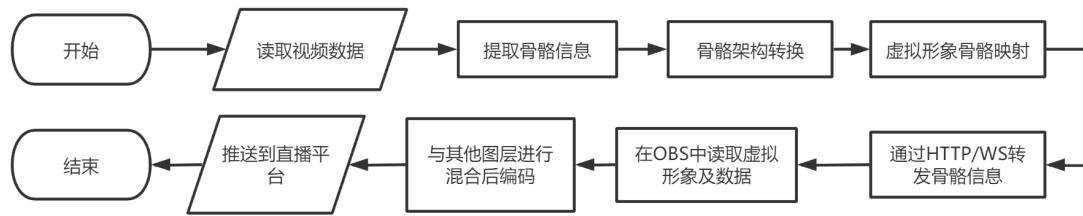


图 5-1 将虚拟形象及动作发送至 OBS 的流程图

使用方法十分简单，在该系统中打开通过网络转发动作数据及虚拟形象功能后，设定好端口号，打开软件后在 OBS 中添加来源并选择浏览器，输入本机地址和端口号，并且点击“互动”按钮可以进行拖动调整位置、角度和滑动滚轮调整大小。图 5-2 为在 OBS 中调用本程序接口添加输入源和获取生成的虚拟形象的画面。

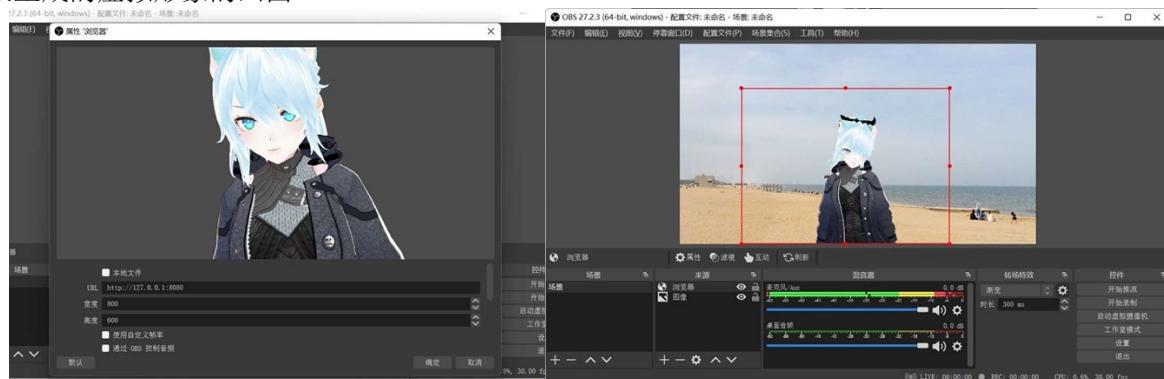


图 5-2 在 OBS 中调用本程序接口

此外，在设计此接口时专门为 OBS 适配了透明背景，无需进行抠像即可直接将虚拟形象融合至 OBS 设定好的场景中，进一步降低了使用难度并提升了画质。

5.2 AR/VR 的虚拟形象直播功能的设计与实现

AR/VR 虚拟形象直播还没有相应的直播平台，但是随着元宇宙概念的普及，AR/VR 直播会成为未来的趋势，所以本论文该部分设计了一套该系统借助于 WebXR 接口实现 AR/VR 动作直播的系统。由于 Three.js 目前支持 WebXR API，所以只需要稍加修改 Three.js 渲染器即可。图 5-3 为对 Three.js 的相关修改使其可以支持 WebXR API 来使用 AR/VR 环境，首先会访问本文 3.7 中提到的 WebXR 配置文件 API 读取是否开启 WebXR 的配置参数，然后分别添加进入 AR 和 VR 模式的按钮。

```
import { ARButton } from "/node_modules/three/examples/jsm/webxr/ARButton.js";
import { VRButton } from "/node_modules/three/examples/jsm/webxr/VRButton.js";

const useXRres = await fetch("/useWebXR");
const useXR = await useXRres.json();

if (useXR) {
  renderer.xr.enabled = true;
  const ar = ARButton.createButton(renderer);
  const vr = VRButton.createButton(renderer);
  ar.style.marginLeft = '-75px';
  vr.style.marginLeft = '75px';
  document.body.appendChild(ar);
  document.body.appendChild(vr);
}

if (useXR) renderer.setAnimationLoop( function () {
  if (currentVrm) {
    // Update model to render physics
    currentVrm.update(clock.getDelta());
  }
  renderer.render(scene, orbitCamera);
});
```

图 5-3 启用 WebXR AR 和 VR 相关支持代码

对于 AR 而言，需要带有 Google AR 框架的手机且使用最新版 Chrome 即可。详细需求可以参见 Google AR Core 文档⁸。该部分使用带有完整 Google 服务（Google Mobile Services, GMS）的小米 9 手机进行测试。当使用 Chrome 以 HTTPS 协议访问本软件时，在支持的设备上会出现“Start AR”按钮，点击后系统提示需要在周边环境创建 3D 地图，点击允许即可进入 AR 模式。图 5-4 左侧是该软件画面，右侧为手机屏幕。

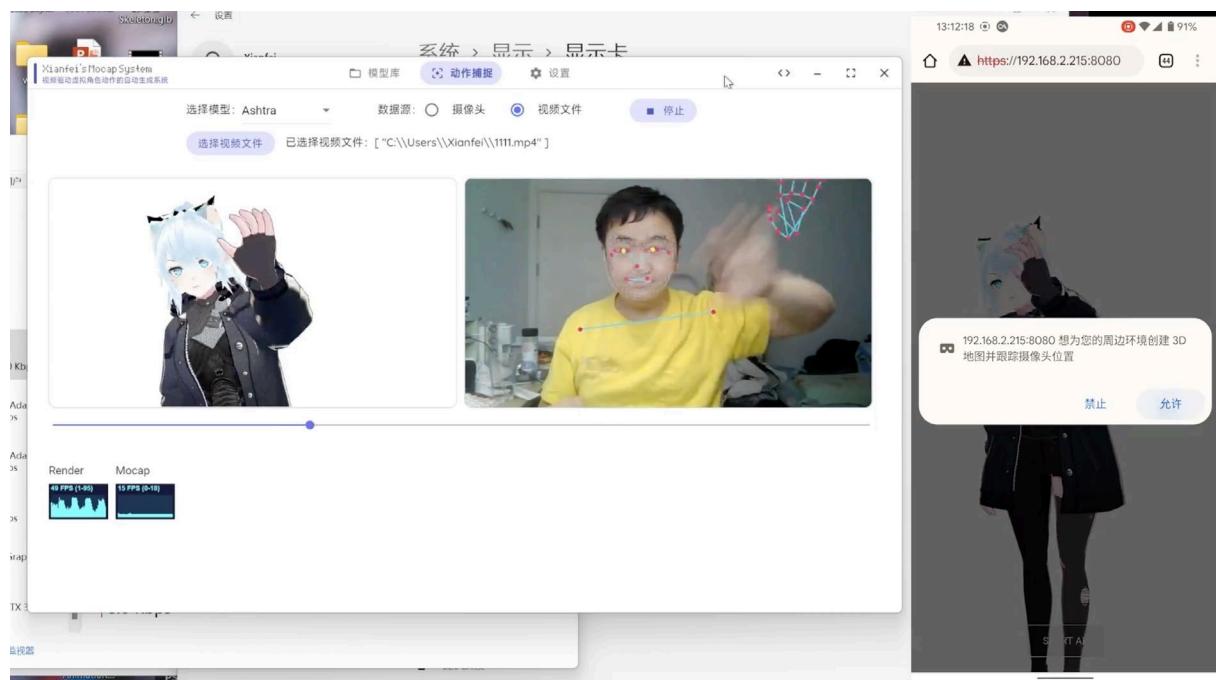


图 5-4 进入 AR 模式

⁸ <https://developers.google.com/ar/develop/webxr/requirements>

进入 AR 模式后，可以看到虚拟形象站立在当前环境内。无论怎样移动手机，虚拟形象都会保持原地不动且做着由该系统捕捉到的动作，在内网环境下测试整个过程没有明显延时。图 5-5 和图 5-6 为不同角度不同距离观察虚拟形象。

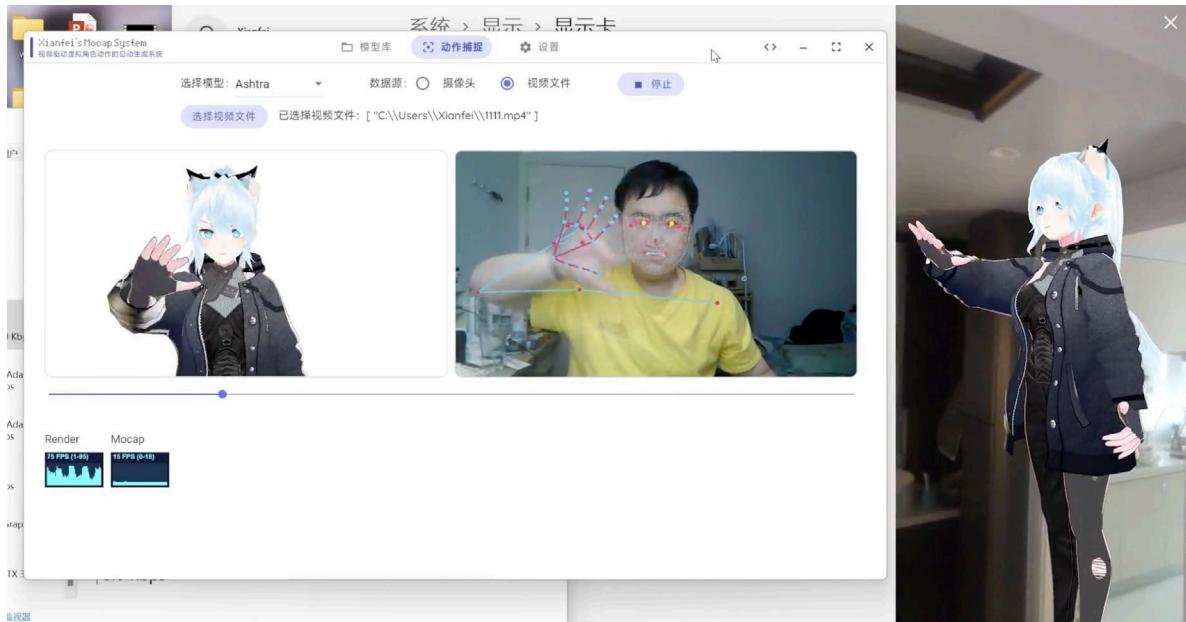


图 5-5 在 AR 模式中在虚拟形象右前方观察



图 5-6 在 AR 模式中在虚拟形象左前方近距离观察

为了更好的令虚拟形象显示在地面上，还需要对地面高度进行校准，通过调节 y 轴偏移量并计算模型最低点位置可让虚拟形象始终处于站立在地面上的状态^[21]。图 5-7 为在不同环境下使用 AR 观察虚拟形象及其动作，本文选择了不同地形（如柏油路、楼梯、土坡）、不同光线环境（如树荫下、阳光下、室内）的测试环境，均有较为良好的效果。



图 5-7 在 AR 模式下不同环境进行观察

同时，该系统还支持 VR 设备，如 HTC Vive，Oculus Rift，Google Cardboard，HoloLens 等。为了进一步的降低用户使用门槛，本文选择了其中成本最低的 Cardboard 解决方案进行描述。Cardboard 由纸板、凸透镜、皮筋、魔术贴等便宜且常见的材料组成^[20]，需要将手机放入 Cardboard 中进行使用。

在 Google Play 应用商店安装“Google VR 服务”软件后，使用 Chrome 以 HTTPS 协议访问本系统（同 AR 显示部分），点击“Entry VR”按钮后将手机插入 Cardboard 中即可在 AR 中实时浏览虚拟形象及时动作。图 5-8 为使用小米 9 手机进入 VR 模式过程截图，图 5-9 为把手机插入 Cardboard 中显示的画面。



图 5-8 在手机上进入 VR 模式截图



图 5-9 Cardboard VR 画面

该系统在 AR/VR 方面的应用创新不仅为虚拟形象直播提供了全新的方式，也为元宇宙下虚拟人物动作交互场景打下了基础。

第六章 跨平台发布及优化

6.1 发布跨平台应用程序

得益于使用 Electron 框架进行桌面 GUI 开发，并使用 Node.js 插件，该系统大部分代码都可以无需修改的适应于不同平台。该系统可在 Windows、macOS、Linux 下以开发环境运行，但是为了创造一个开箱即用的软件，我们需要对不同系统创建不同的部署环境。

6.1.1 Windows 平台下打包及优化

在 Windows 平台上，实现毛玻璃背景效果需要使用 Node.js 依赖 electron-acrylic-window 来创建具有半透明背景的窗口，所以需要代码对其进行控制。图 6-1 中为相关代码实现。

```
// Enable Acrylic Effect on Windows by default
if (platform === "win32") {
  try {
    blurBrowserWindow =
      require("electron-acrylic-window").BrowserWindow;
  } catch (e) {
    blurBrowserWindow = BrowserWindow;
  }
  // if not on Windows, use electron window
  else blurBrowserWindow = BrowserWindow;
}
```

图 6-1 为 Windows 启用毛玻璃背景效果

在打包时，选择了使用 electron-packager 加上 electron-installer-zip 的组合，前者可以将程序主体和相关依赖、数据（如机器学习框架、渲染器、内建模型等）打包到一个含可执行文件的文件夹中，后者可以将打包产物压缩成一个 zip 压缩包。图 6-2 为打包生成的产物，其中 SysMocap.exe 为可执行文件。

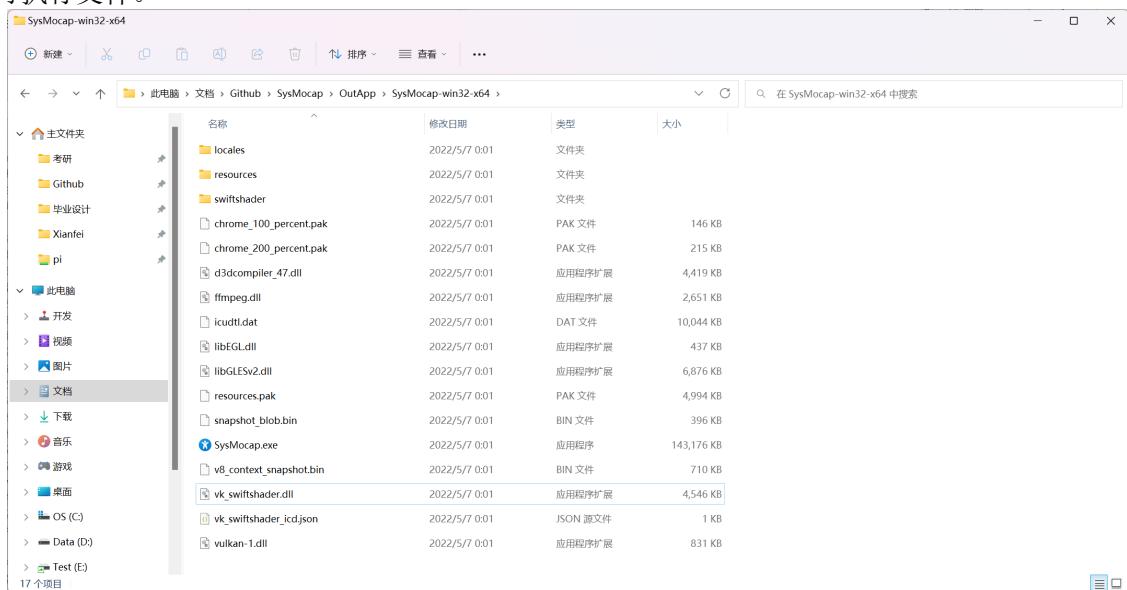


图 6-2 Windows 平台上打包产物

Windows 系统下用户配置文件(设置、用户模型等)位于%UserProfile%\sysmocap\profile.json 文件。

6.1.2 macOS 平台下打包及优化

在 macOS 系统下, 由于该系统控制窗口最小化、最大化和关闭的按钮位于左上角(将其称之为“红绿灯”按钮), 这与大多数系统都不一样, 所以我们需要对其进行单独布局。在创建窗口时, 首先要声明使用系统默认的“红绿灯”按钮, 如图 6-3 中代码是用于判断系统为 macOS(代号 darwin)时使用特定的标题栏样式。

```
var viewer = new myBrowserWindow({
  width: 820,
  height: 540,
  titleBarStyle: platform === "darwin" ? "hiddenInset" : "hidden",
  autoHideMenuBar: true,
  ...addtionalArgs,
  titleBarOverlay: {
    color: args.backgroundColor,
    symbolColor: args.color,
  },
},
```

图 6-3 针对 macOS 系统标题栏优化 1

此外, 在布局 UI 时, 也需要对左上角标题文字的边距进行调整, 留出位置让给“红绿灯”按钮, 图 6-4 左为在样式方面对系统进行的判断, 右为最终效果。



图 6-4 针对 macOS 系统标题栏优化 2

由于 macOS 对系统权限管理较为严格, 而本系统在实时动作捕捉时需要访问摄像头, 在 macOS 中, 访问摄像头需要申请权限。图 6-5 左侧为申请摄像头权限代码, 右侧为启动动作捕捉时系统提示授予摄像头访问权限。



图 6-5 在 macOS 系统上申请系统权限

在打包方面, 选择了使用 electron-packager 加上 electron-installer-dmg 的组合, 前者同 6.1.1 中的 electron-packager, 后者可以打包成 macOS 上特色的 dmg 镜像, 如图 6-6, 只需要拖动即可完成 APP 的安装, 也可以直接双击运行。

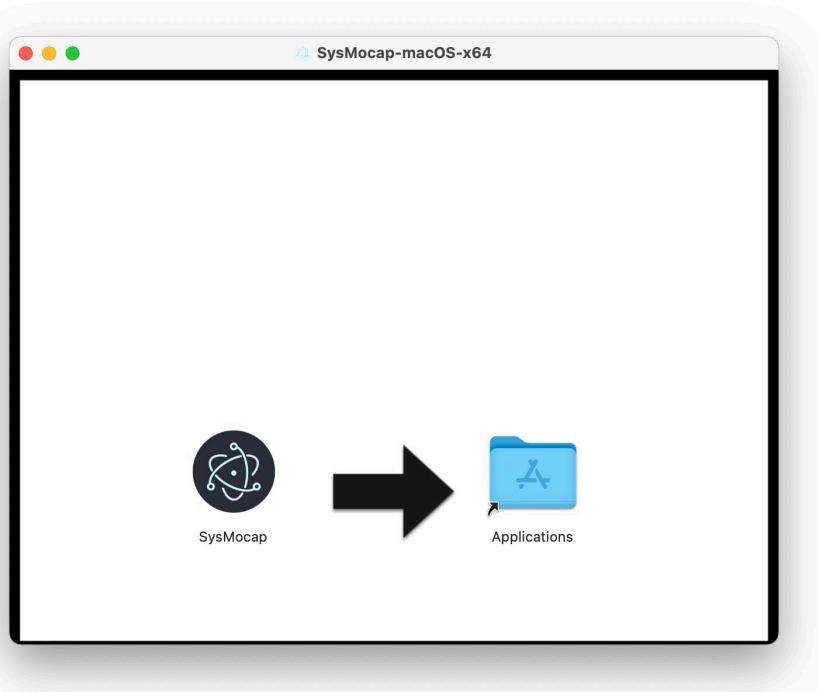


图 6-6 在 macOS 系统上安装镜像 dmg 文件

macOS 系统下用户配置文件（设置、用户模型等）位于 `~/sysmocap/profile.json` 文件。

6.2 使用 CI/CD 系统进行自动打包

上一小节展示了如何在 macOS/Windows 平台上发布可执行程序包。而开发过程中，每次修改代码后都需要手动运行编译打包的过程过于繁琐，所以需要引用持续集成/持续交付（Continuous Integration/ Continuous Delivery, CI/CD）系统自动完成编译打包和交付操作。此处我们选择 GitHub Actions 平台作为 CI/CD 环境。首先需要将上一小节所说的几个步骤都配置为 node package manager (npm) 命令，图 6-7 中配置了对不同平台的编译打包命令。

```
"scripts": {
  "start": "electron .",
  "start-nogpu": "electron . --disable-gpu",
  "package:mac64": "electron-packager ./ SysMocap --platform=darwin --arch=x64 --out ./OutApp --download.mirror=https://github.com/atom/electron/releases/download/v1.7.0/Electron-v1.7.0-darwin-x64.zip",
  "package:win64": "electron-packager ./ SysMocap --icon=sysmocap.ico --platform=win32 --arch=x64 --out ./OutApp --download.mirror=https://github.com/atom/electron/releases/download/v1.7.0/Electron-v1.7.0-win32.zip",
  "zip:win64": "electron-installer-zip ./OutApp/SysMocap-win32-x64/ ./OutApp/packages/SysMocap-Windows-x64.zip",
  "zip:mac64": "electron-installer-zip ./OutApp/SysMocap-darwin-x64/ ./OutApp/packages/SysMocap-macOS-x64.zip",
  "dmg": "electron-installer-dmg ./OutApp/SysMocap-darwin-x64/SysMocap.app SysMocap-macOS-x64 --out ./OutApp"
},
```

图 6-7 npm scripts

配置好本地打包命令后，编写 GitHub Actions 配置文件，需要执行的过程为拉取 Git->安装 Node.js->安装 Node.js 依赖->编译->打包->上传产物->交付。

配置好 GitHub Action 环境后，只需要 Push 带有版本号 tag 的 Commit，即可触发工作流运行。图 6-8 为其产生的工作流，产生的可执行文件包会自动推送到 Release 页面。其中 macOS 程序包大小为 278MB，Windows 程序包大小为 301MB。

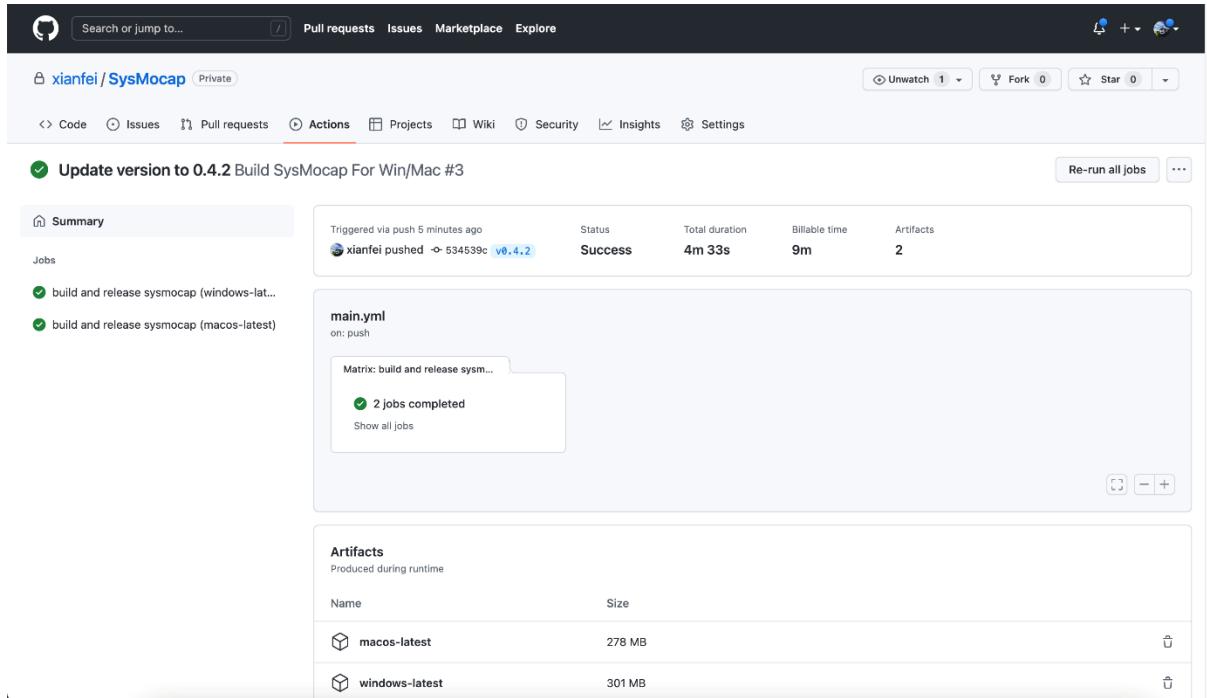


图 6-8 Github Actions 工作页面

6.3 以 B/S 模式运行系统

由于该系统渲染器部分使用基于 WebGL 的 Three.js 部分完成，第五章动作转发部分也展示了完整的动作转发流程，所以借助网络实时通信（Web Real-Time Communication，WebRTC）技术，可以将整个系统运行在 BS (Browser/Server，浏览器/服务器) 模式下。但是运行在该模式下延时较高且目前无法使用动作转发（因为存在多用户问题）及模型导入功能，所以该技术当前阶段仅供演示使用。

为了能使这套系统能运行在 BS 模式下，我们为它添加了行命令接口(Command-Line Interface, CLI)，在行命令中运行该程序的可执行文件加上--help 命令可以查看所有可用的参数，图 6-9 为在 CLI 模式下运行截图。

```
→ MacOS git:(main) ✘ ./SysMocap --help
Options
--help          Print this usage guide.
--bsmode        Run SysMocap on B/S mode. (experimental)
--port number  The HTTP port when running on B/S mode.
--reset         Reset preferences and run sysmocap
--disable-acrylic  (Windows Only) disable acrylic effects
```

图 6-9 以 CLI 传递 help 参数运行该系统

要想以 BS 模式运行该系统，只需传入 bsmode 参数并设置端口号（不设置默认使用 8080），然后在浏览器访问即可，图 6-10 中为使用 5500 作为端口号，在浏览器中访问 <http://127.0.0.1:5500/> 的画面。

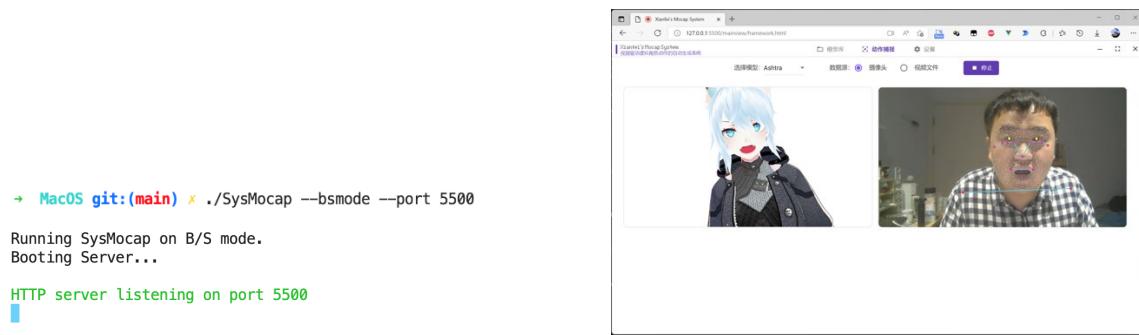


图 6-10 以 BS 模式运行该系统

6.4 性能优化

6.4.1 在双显卡设备上使用高性能 GPU

(该部分仅限于 Windows 系统)

由于目前绝大多数包含独立显卡的笔记本电脑均使用双显卡技术（同时使用 CPU 内建的核心显卡与独立显卡），除非启用显卡直连技术，否则默认情况下程序会在核心显卡上运行。在 Windows 上，在程序参数加上 SHIM_MCCOMPAT 0x800000001 参数即可强制 Windows 为该进程调用独立显卡。

此外，还在设置页面加入了相关参数和选项，可以让用户自行选择是否使用独立显卡进行计算（不使用独立显卡会更省电）。图 6-11 为在具有双显卡的笔记本上不启用独显和启用独显的区别。



图 6-11 使用独立显卡设置页面

使用独立显卡时，系统只会把计算任务交给独立显卡进行处理，渲染任务由与屏幕相连的显卡处理（在笔记本上通常是集成显卡）。图 6-12 为在普通具有独显的笔记本上，两个显卡都有一定的负载，分别处理渲染和计算任务，进一步提升了性能和资源利用率。

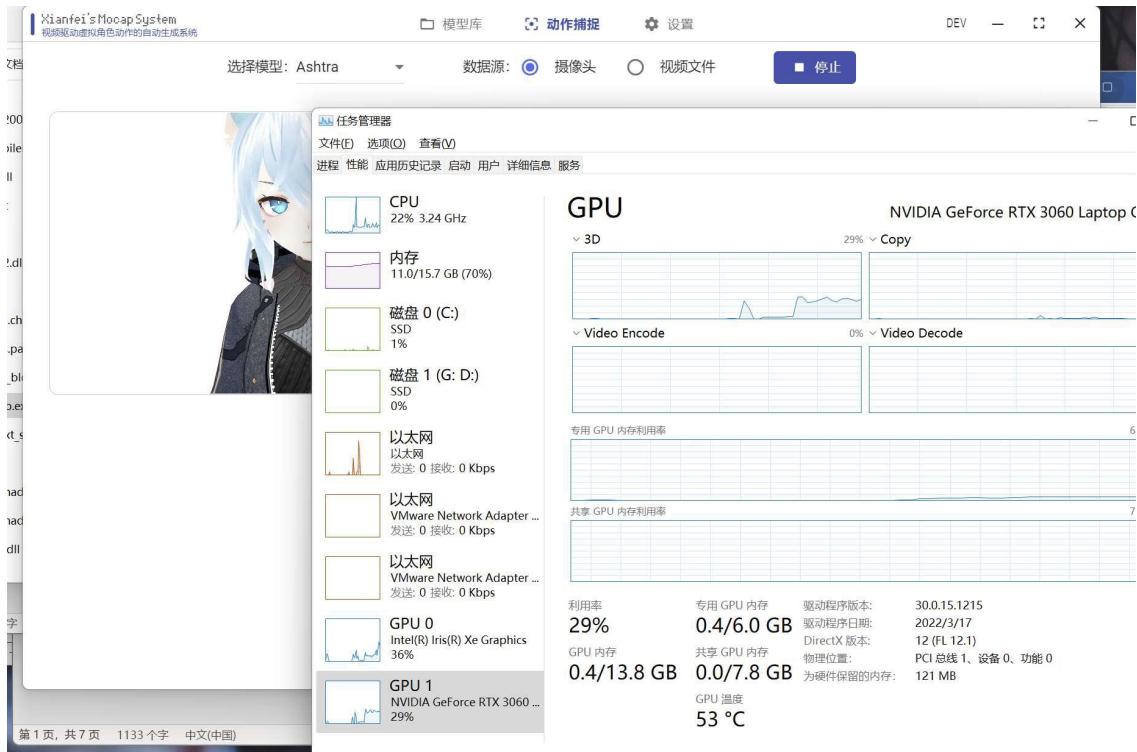


图 6-12 显卡资源利用率

6.4.2 后台渲染优化

由于 Chromium 的节能特性，在当程序处于后台时，将会降低程序渲染性能及减少计数器 Tick 频次，这对动作捕捉时进行虚拟形象转发不利。如果不阻止渲染器进入后台，那么会明显感觉在启用动作转发时，将本系统放入后台后转发的虚拟形象会卡顿。所以需要增加 disable-renderer-backgrounding 参数，图 6-13 中为相应的代码。

```
// Prevents Chromium from lowering the priority of invisible pages' renderer processes.
// Improve performance when Mocap is running and forward motion data in background
app.commandLine.appendSwitch("disable-renderer-backgrounding", true);
```

图 6-13 阻止后台降低优先级的代码

6.4.3 多线程优化

为了让动作和虚拟形象转发模块可以获得更高的网络通信优先级和带宽同时不影响到动作捕捉及渲染的运行，我们让动作捕捉数据通过进程间通信（Inter-process communication, ipc）转发到 Node.js 主进程，然后由 Node.js 主进程创建线程。动作捕捉模块将动作数据通过进程间通信转发至主进程后，通过线程间通信转发至服务器线程，动作捕捉数据每秒钟小于 30 次的进程间通信不会带来较大的开销，但是却能较好的提高网络转发性能。

良好的通信性能为实时转发打下了坚实基础，在 OBS 转发及 AR/VR 转发方面，使用无线局域网下几乎感觉不到延时，且几乎不会出现丢包、抖动过大等情况。

结束语

本系统以视频驱动的动作捕捉算法为基础，以驱动虚拟形象作为应用形式，研究了其在直播、虚拟主播、VR/AR 领域的应用。并且为了降低这些应用的门槛，设计了一套功能齐全、易于安装、简单易用的系统，同时设计并实现了从导入虚拟形象、驱动虚拟形象、进行直播和 VR/AR 展示的一站式解决方案，用户仅仅需要一台主流配置的带有摄像头的计算机，即可成为 3D 虚拟主播。在算法调研方面，本文对比了当下主流的视频驱动的三维动作捕捉算法，并将其进行对比，选出了性能高、对配置要求低的算法；在算法研究及实现方面，本文自主设计了骨骼转换及骨骼绑定算法，将动作捕捉数据可用于驱动不同类型的虚拟形象；在工程应用方面，本文设计了桌面端程序、后端服务器、前端页面，并且将机器学习框架、算法模型、虚拟形象等封装在了一个应用程序内，实现开箱即用；在应用创新方面，本文设计了为元宇宙时代准备的 AR/VR 虚拟形象实施转发工作流，可应用于 AR/VR 中的虚拟形象直播、通话等。

在算法调研时，发现许多算法都只能运行在特定版本的操作系统上（Ubuntu 16.04/18.04），且需要借助 CUDA 和独立显卡才能高效的运行，于是在开发过程中，研究了许多方案，试图将机器学习运行在生产环境下，即无需用户进行特殊配置，就像使用办公软件一样简单。最终我发现了可以运行在 WASM (Web Assembly, 一种用于虚拟机的汇编语言) 环境下的 MediaPipe，加上为实时性能优化的 BlazePose 3D，实现了像办公软件一样简单易用。

在日常生活中，往往人们对好看的东西有着较好的第一印象，在我日常下载软件时也会优先选择界面较为美观的软件进行使用。不只是在计算机上，在日常生活中买东西时，也会因为一个东西长得好看而买它。所以在编写用户界面时，运用了大量前端和美术知识，做了更好看的配色和动画效果。

在开发时，本项目尽可能的选择了现代化开发工具和框架，将本科四年所学到的知识尽可能的运用，如计算机网络、通信协议设计、前后端开发、GUI 开发等。Electron 是我十分喜欢的桌面 GUI 开发框架，在两个大创项目中都有使用其开发桌面 GUI 的经验；Vue 是十分流行的前端框架，将 Vue 与 Electron 相结合开发桌面 GUI 可以高效的开发好看又好用的程序；WebXR 在以前研究 AR 展示的时候有所接触；在“服务外包大赛”中也尝试过将机器学习框架作为后端用于推荐和预测；但将机器学习集成在桌面端应用程序中，这还是第一次。曾经一直觉得，眼界和想法，有时候甚至比所掌握的技术还要重要，这次毕设也实现了很多自己无意间想出来的点子，可能以后工作中也没有这种自己设计自己实现一个作品的机会了。

如果您在阅读本论文后，恰好对该系统感兴趣或想尝试、了解该项目，或是想成为虚拟主播，可以在本项目 Github Release 页面中下载到可执行文件压缩包，也欢迎浏览项目源代码并提出宝贵的建议，<https://github.com/xianfei/SysMocap>。

致谢

首先大学四年，感谢北京市的双培计划，让我同时感受和学习到了北京信息科技大学和北京邮电大学的氛围和相关课程，对于两个学校的核心课程和技术类社团、交流群都有涉及，这让我在技术领域上获得了很大的进步，了解到了许多前沿技术和应用。

在毕设中，感谢宋文凤老师对毕设方面的指导，尤其是对虚拟形象、动作捕捉、骨骼绑定方面，告诉了我许多基础的知识和一些研究方向。刚开始选择这个题目时，感觉这是一个很深奥很难的方向，通过与老师的一番交流才有了下手的方向。

其次在开发和搜索资料的过程中，感谢 Google 为开源社区和学术界贡献了许多代码和算法，其开发的 BlazePose 算法、TensorFlow Lite 框架、MediaPipe 框架、ARCore 引擎、Material Color 色彩科学、Chromium 项目及 V8 引擎在该项目中起到了巨大的作用，同时 Google 学术也为本人在撰写论文查阅英文文献时起到了至关重要的作用。感谢所有开源项目的作者，本系统的实现离不开这些开源项目的支持，也希望本人在将来的科研和工作中能为开源社区和学术界贡献自己的一份力量。

感谢各位老师和评委及读者阅读本论文，感谢老师和同学给予我的帮助，因为有大家的帮助才能让我在学术的道路上走的更远。

参考文献

- [1] 何国威. 从动作捕捉到虚拟偶像：计算机技术对演员及电影娱乐生态的发生与重构[J]. 当代电影, 2022(01):72–81.
- [2] 元宇宙虚拟人产业链[J]. 中国科技信息, 2022(06):6–7.
- [3] Cao Z, Simon T, Wei S E, et al. Realtime multi-person 2d pose estimation using part affinity fields[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2017: 7291–7299.
- [4] 王涵, 连家斌. NOKOV 红外光学动作捕捉系统精度测试[J]. 现代信息科技, 2021, 5(15):113–115, 118. DOI:10.19850/j.cnki.2096-4706.2021.15.029.
- [5] 周瑞文. 基于惯性测量单元的三维动作捕捉系统关键技术研究[D]. 哈尔滨工业大学, 2020. DOI:10.27061/d.cnki.ghgdu.2020.002157.
- [6] Remelli E, Han S, Honari S, et al. Lightweight multi-view 3d pose estimation through camera-disentangled representation[C]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2020: 6040–6049.
- [7] Zhang Z. Microsoft kinect sensor and its effect[J]. IEEE multimedia, 2012, 19(2): 4–10.
- [8] Ward L M, Morison G, Simpson W A, et al. Using functional near infrared spectroscopy (fNIRS) to study dynamic stereoscopic depth perception[J]. Brain Topography, 2016, 29(4): 515–523.
- [9] 李双峰. TensorFlow Lite: 端侧机器学习框架[J]. 计算机研究与发展, 2020, 57(9): 1839–1853.
- [10] 高锐, 闫光辉, 罗浩, 严天峰. 基于 WebSocket 技术无线频谱大数据实时监测系统设计与实现[J]. 兰州交通大学学报, 2022, 41(01):52–60.
- [11] Lugaresi C, Tang J, Nash H, et al. Mediapipe: A framework for building perception pipelines[J]. arXiv preprint arXiv:1906.08172, 2019.
- [12] Chen M, Tworek J, Jun H, et al. Evaluating large language models trained on code[J]. arXiv preprint arXiv:2107.03374, 2021.
- [13] Choi H, Moon G, Chang J Y, et al. Beyond static features for temporally consistent 3d human pose and shape from a video[C]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2021: 1964–1973.
- [14] Zanfir A, Bazavan E G, Xu H, et al. Weakly supervised 3d human pose and shape reconstruction with normalizing flows[C]//European Conference on Computer Vision. Springer, Cham, 2020: 465–481.
- [15] Zhang H, Tian Y, Zhou X, et al. Pymaf: 3d human pose and shape regression with pyramidal mesh alignment feedback loop[C]//Proceedings of the IEEE/CVF International

- Conference on Computer Vision. 2021: 11446–11456.
- [16]Qammaz A, Argyros A. Occlusion-tolerant and personalized 3D human pose estimation in RGB images[C]//2020 25th International Conference on Pattern Recognition (ICPR). IEEE, 2021: 6904–6911.
- [17]Zhang F, Bazarevsky V, Vakunov A, et al. Mediapipe hands: On-device real-time hand tracking[J]. arXiv preprint arXiv:2006.10214, 2020.
- [18]Bazarevsky V, Kartynnik Y, Vakunov A, et al. Blazeface: Sub-millisecond neural face detection on mobile gpus[J]. arXiv preprint arXiv:1907.05047, 2019.
- [19]Choutas V, Pavlakos G, Bolkart T, et al. Monocular expressive body regression through body-driven attention[C]//European Conference on Computer Vision. 2020: 20–40.
- [20]Perla R, Hebbalaguppe R. Google cardboard dates augmented reality: issues, challenges and future opportunities[J]. arXiv preprint arXiv:1706.03851, 2017.
- [21]Barone Rodrigues A, Dias D R C, Martins V F, et al. WebAR: A web-augmented reality-based authoring tool with experience API support for educational applications[C]//International Conference on Universal Access in Human-computer Interaction. Springer, Cham, 2017: 118–12

