

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

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 3 月 1 日至 2022 年 6 月 6 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、绘制等工作，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基于 CNN 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

Nowadays, VTuber, animation production, metaverse and other fields is in full swing, and the prospects are very broad. However, in the past, the animation production of virtual characters was a huge project, which required a lot of time and manpower for shooting and drawing. Even the existing video capture solutions required 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 based on CNN optimized for real-time performance. The algorithm analyzes the action of the character and extracts and generates the three-dimensional skeleton coordinate information in real time. Secondly, the skeleton transformation algorithm is designed, which is 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 broadcast.

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

# 目 录

<b>摘要</b> .....	I
<b>Abstract</b> .....	II
<b>目录</b> .....	III
<b>第一章 概述</b> .....	1
1.1 研究背景 .....	1
1.2 相关技术的国内外文献综述简析 .....	1
1.3 研究内容与主要贡献 .....	2
1.4 论文结构 .....	3
<b>第二章 开发环境与相关技术</b> .....	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 .....	5
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 .....	6
2.7 WebXR AR 及 VR API .....	6
2.8 开发环境 VSCode 与 Copilot .....	6
<b>第三章 虚拟形象动作生成系统系统设计</b> .....	7
3.1 系统需求分析 .....	7
3. 1. 1 功能需求 .....	7
3. 1. 2 非功能性需求 .....	7
3. 1. 3 运行环境要求 .....	7
3.2 总体设计 .....	8
3.3 动作捕捉模块设计 .....	9
3.4 虚拟形象管理及展示模块设计 .....	9
3.5 渲染模块设计 .....	12
3.6 图形用户界面设计 .....	13
<b>第四章 实时动作数据及虚拟形象转发</b> .....	16
4.1 转发系统概述 .....	16
4.2 动作转发通信模块 .....	17
4.3 用于 OBS 直播软件的接口设计与实现 .....	18

4.4 AR/VR 的虚拟形象直播功能的设计与实现 .....	18
<b>第五章 动作捕捉算法及虚拟形象绑定算法 .....</b>	<b>21</b>
5.1 视频驱动的动作捕捉算法研究 .....	21
5.2 BlazePoze GHUM 3D 算法及性能评估 .....	23
5.3 动作捕捉骨骼架构转换算法 .....	26
5.4 虚拟形象骨骼映射算法 .....	28
<b>第六章 跨平台发布及优化 .....</b>	<b>30</b>
6.1 发布跨平台应用程序 .....	30
6.1.1 Windows 平台下打包及优化 .....	30
6.1.2 macOS 平台下打包及优化 .....	31
6.2 使用 CI/CD 系统进行自动打包 .....	32
6.3 以 B/S 模式运行系统 .....	33
6.4 性能优化 .....	34
6.4.1 在双显卡设备上使用高性能 GPU .....	34
6.4.2 后台渲染优化 .....	35
6.4.3 多线程优化 .....	35
<b>结束语 .....</b>	<b>36</b>
<b>致谢 .....</b>	<b>37</b>
<b>参考文献 .....</b>	<b>38</b>

# 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。

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，市场知名度大大提高，应用领域不断扩大。但总体来说，我国动作捕捉系统行业还处于发展初期，在目前还存在着很多问题。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。2013 年至 2017 年，国内市场成交了大量的动作捕捉系统。总体而言，国内动作捕捉系统行业产值在过去几年呈现快速增长趋势，产值从 2013 年的 23.8 亿元增长到 2017 年的 44.5 亿元。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前景<sup>[1]</sup>。

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<sup>[2]</sup>。

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。而且受限于设备的要求，需要价格不菲的设备进行采集及加工处理。而该项目致力于使用普通的视频素材，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生成，并应用于直播、动画视频制作等领域，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<sup>[3]</sup>），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。

非视频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。光学动作捕捉基于红外光从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。惯性动作捕捉通过惯性传感器来捕捉人的关键骨骼旋转信息，对人体运动进行测量，最后通过无线方式将数据传输到计算机上去还原人体运动。通常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。该类型方案的准确度较高<sup>[4]</sup>，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。而基于惯性的动作捕捉<sup>[5]</sup>是也是需要用者佩戴专门的动作捕捉硬件，通过加速度传感器及陀螺仪传感器，来计算每个传感器的位置，从而知道使用者的各个关节所在的位置。

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<sup>[7]</sup>，及 iPhone X 及之后的带有 3D 结构光传感器（Face ID）用于面部捕捉的 Animoji<sup>[8]</sup>，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<sup>[6]</sup>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<sup>[14]</sup>），图 1-1 展示了大致的流程。

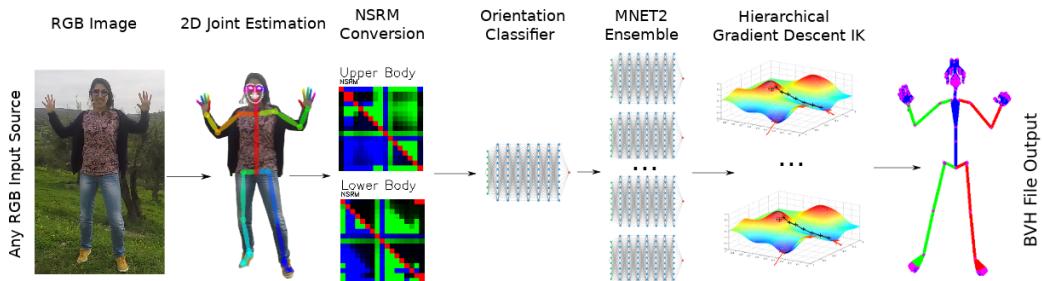


图 1-1 RGB 视频三维动作捕捉流程

### 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处理。
- (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 MPLv2 (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；第二章展示了开发环境，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分析与描述；第四章对动作及虚拟形象转发模块的详细设计及基于它的应用；第五章详细介绍了关键算法；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 开发环境与相关技术

### 2.1 TensorFlow Lite、MediaPipe 机器学习框架

TensorFlow Lite 是一种在设备端运行 TensorFlow 模型的开源深度学习框架，它允许开发人员在移动、嵌入式和物联网设备上运行模型，从而在这些设备上实现机器学习。相比传统的在服务器端使用 GPU 运行 TensorFlow 模型的模式，TensorFlow Lite 在以下五个方面更具有优势：低延迟（无需往返服务器）、隐私性（无需从设备泄露个人数据）、连接性（无需互联网连接）、大小（缩小模型和二进制大小）、低功耗（推理效率高）。

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” 标识）表示。相比 TensorFlow 的 Protocol Buffers 模型格式，这种格式的体积更小（代码占用空间更小）和推理速度更快（无需额外的解析和解压步骤即可直接访问数据）有优势。这允许 TensorFlow Lite 在计算和内存资源有限的设备上高效运行。

MediaPipe 是一款由 Google Research 开发并开源的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。它是一个基于图的数据处理管线，且任何 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 运行进行视频驱动的实时动作捕捉处理，该部分详见第五章。

### 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，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，意在统一 3D 虚拟偶像的文件格式标准，再将这个格式标准推向全球。人形 3D 模型格式多种多样的不统一，坐标系不同，尺度不同，初始姿势不同，表情不同，骨骼导入程序的方法也不同。而 VRM 统一了各模型的坐标系、骨骼、表情等信息，使得模型使用起来更加方便，模型间的互换性也更好。除此之外，VRM 还提供了弹簧骨骼以及碰撞等组件，易于处理，适用于各种应用。

## 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。这种跨平台格式已成为 Web 上的 3D 对象标准。

## 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。这是一种 3D 资产交换格式，有助于在 3ds Max、Maya、MotionBuilder、Mudbox 和其他专有软件和第三方软件之间进行更高保真度的数据交换。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。Electron 现已被多个开源 Web 应用程序用于前端与后端的开发，著名项目包括 GitHub 的 Atom 和微软的 Visual Studio Code。

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 是一个商业可用的开源 JavaScript 库，可让使用者轻松创建 3D 内容。在此前，为了仅使用 WebGL 表示 3D 内容，即使只显示一个立方体也需要编写大量 JavaScript 和 GLSL 代码，并且还需要专业知识。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。WebSocket 协议在 2011 年被 IETF 标准化为 RFC 6455。它由主要浏览器供应商 ( Apple、Google、Mozilla 和 Microsoft ) 组成的联盟 Web 超文本应用技术工作组(WWHATWG)维护。

RFC 6455 声明 WebSocket “旨在通过 HTTP 端口 443 和 80 工作以及支持 HTTP 代理和中介”，从而使其与 HTTP 兼容。为了实现兼容性，WebSocket 握手使用 HTTP Upgrade 标头从 HTTP 协议更改为 WebSocket 协议。与 HTTP 轮询等半双工替代方案相比，WebSocket 协议支持 Web 浏览器（或其他客户端应用程序）和 Web 服务器之间的交互，从而促进与服务器之间的实时数据传输。

这是通过为服务器提供一种标准化的方式来实现的，即无需客户端首先请求即可向客户端发送内容，并允许在保持连接打开的同时来回传递消息。协议规范将 ws (WebSocket) 和 wss (WebSocket Secure) 定义为两个新的统一资源标识符 (URI) 方案，分别用于未加密和加密连接。

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

OBS (Open Broadcast Software) 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 Studio 用 C / C++ 编写并使用 Qt 构建，通过实时消息协议 (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 AR 及 VR API

WebXR Device API 是一个 Web 应用程序编程接口，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。XR 中的“X”代表沉浸式体验范围内的任何事物。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。它基于 Electron 框架开发（与本论文中所讨论的系统使用的同样的 GUI 框架），其功能包括对调试、语法突出显示、智能代码完成、片段、代码重构和嵌入式 Git 的支持。用户可以更改主题、键盘快捷键、首选项，并安装添加附加功能的扩展。在 Stack Overflow 2021 开发人员调查中，Visual Studio Code 被评为最受欢迎的开发人员环境工具，82,000 名受访者中有 70% 的人表示他们使用过它。

GitHub Copilot 是由 Github 和 OpenAI 创造的 AI 工具<sup>[12]</sup>，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。

## 第三章 虚拟形象动作生成系统系统设计

### 3.1 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 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：

- 增加、删除、修改虚拟形象
- 查看及编辑虚拟形象骨骼位置信息
- 对虚拟形象骨骼绑定进行修改
- 查看 3D 虚拟形象

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
- 将视频中提取到的骨骼信息转换为姿态信息
- 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：

- 将生成的虚拟形象进行转发
- 可以使用浏览器查看生成的虚拟形象及动作
- 可以在 OBS 直播系统中读取生成的虚拟形象及动作
- 可以在 AR/VR 设备上使用生成的虚拟形象及动作

#### 3.1.2 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。

#### 3.1.3 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
- Memory: 8 GB RAM 或更高
- Graphics: Intel Iris Graphics 540 (AMD / NVIDIA 同等规格) 或更高
- Storage: 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
- Memory: 8 GB RAM 或更高
- Storage: 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。

## 3.2 总体设计

为了开发的便捷性及系统的可移植性考虑，该系统在实际开发中的模块划分与需求分析中的功能模块划分略有不同。图 3-1 展示了整个系统各个模块及各模块之间的通信。

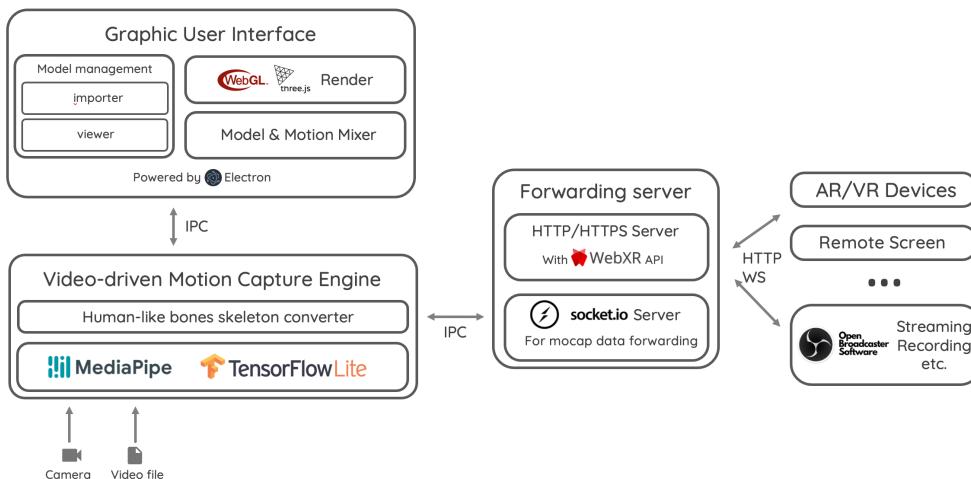


图 3-1 各个模块功能及模块间通信示意图

其中“动作模型部分”模块设计在 3.3 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；“转发服务器”模块设计在第四章中进行阐述；“图形用户界面”及其中包含的渲染器及模型管理在 3.4 和 3.5 中进行阐述。

### 3.3 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 Biovision Hierarchy (BVH) 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，此部分算法在第五章会进行详细讲解。图 3-2 展示了模块的整个工作流程。

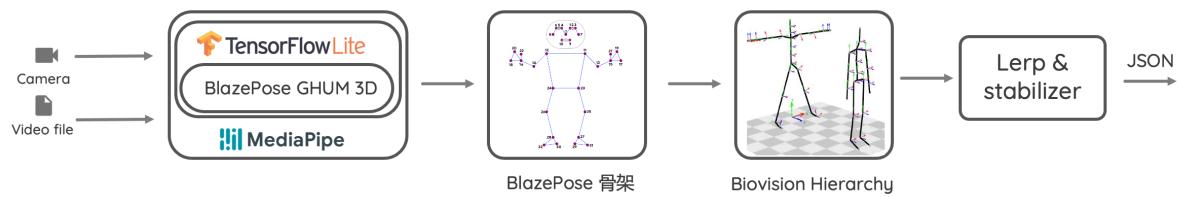


图 3-2 动作捕捉模块工作流程

### 3.4 虚拟形象管理及展示模块设计

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 Studio 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 3-3 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。

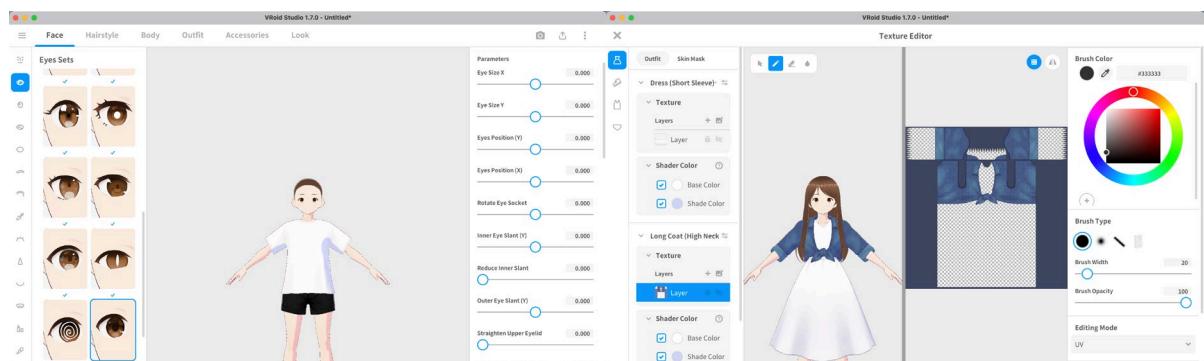


图 3-3 使用 VRoid Studio 创建虚拟形象

其次适配了 Mixamo 网站生成的 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 3-4 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。

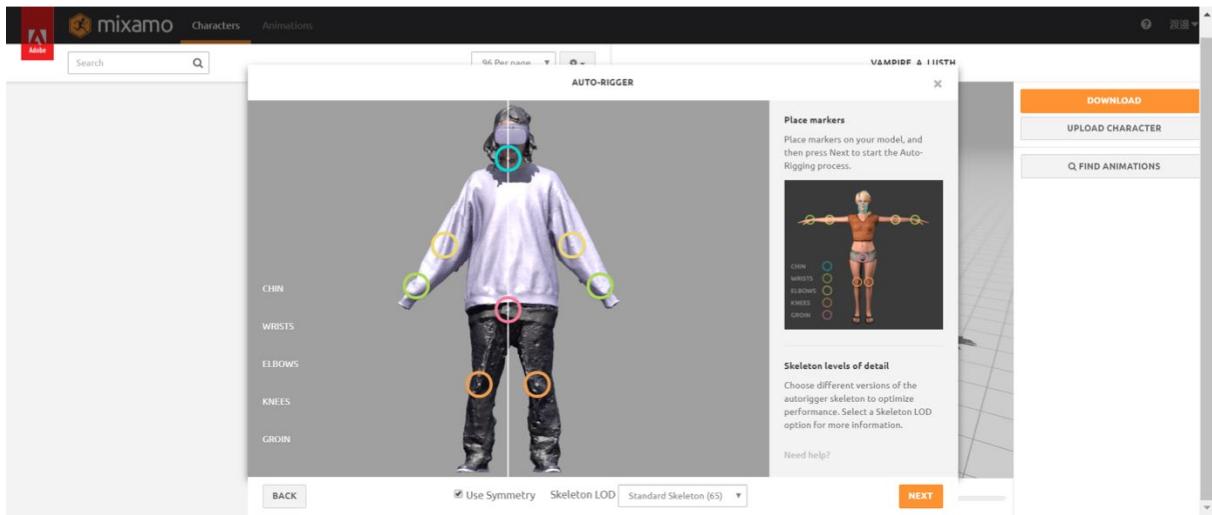


图 3-4 使用 Mixamo 进行自动骨骼绑定

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 3-5 为 Mixamo 网站上可供下载的虚拟形象页面。

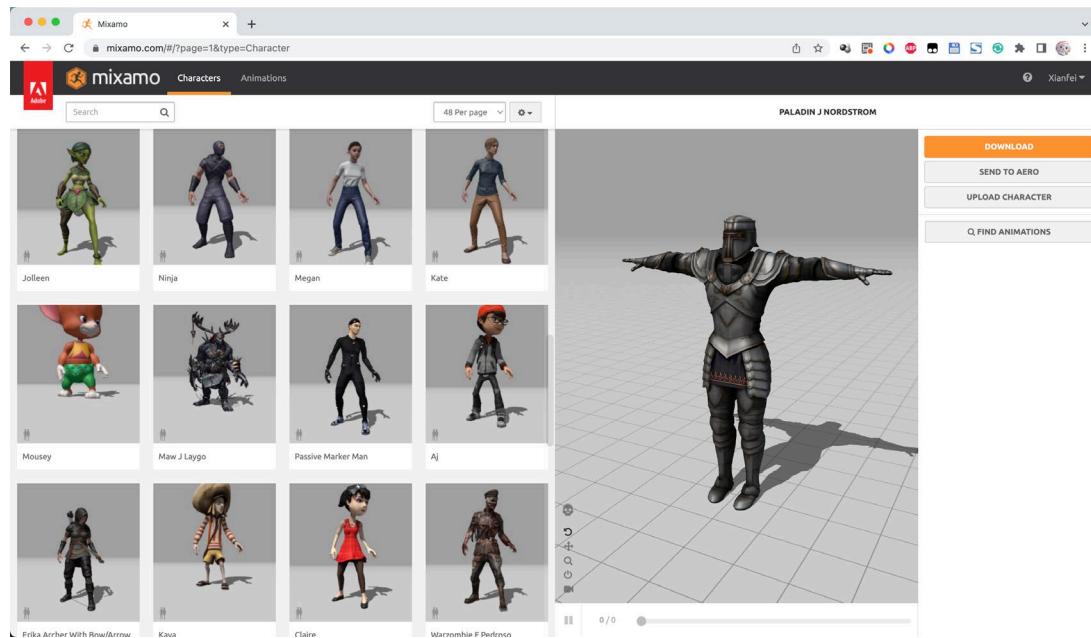


图 3-5 从 Mixamo 上获取虚拟形象

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 3-6 展示了此过程。

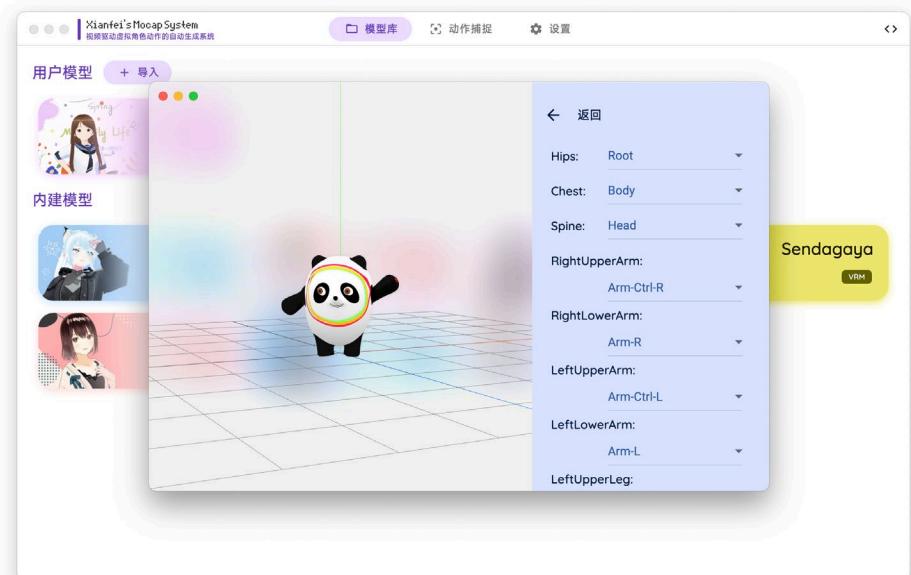


图 3-6 使用本项目自带的骨骼映射器手动映射

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 xxx 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。

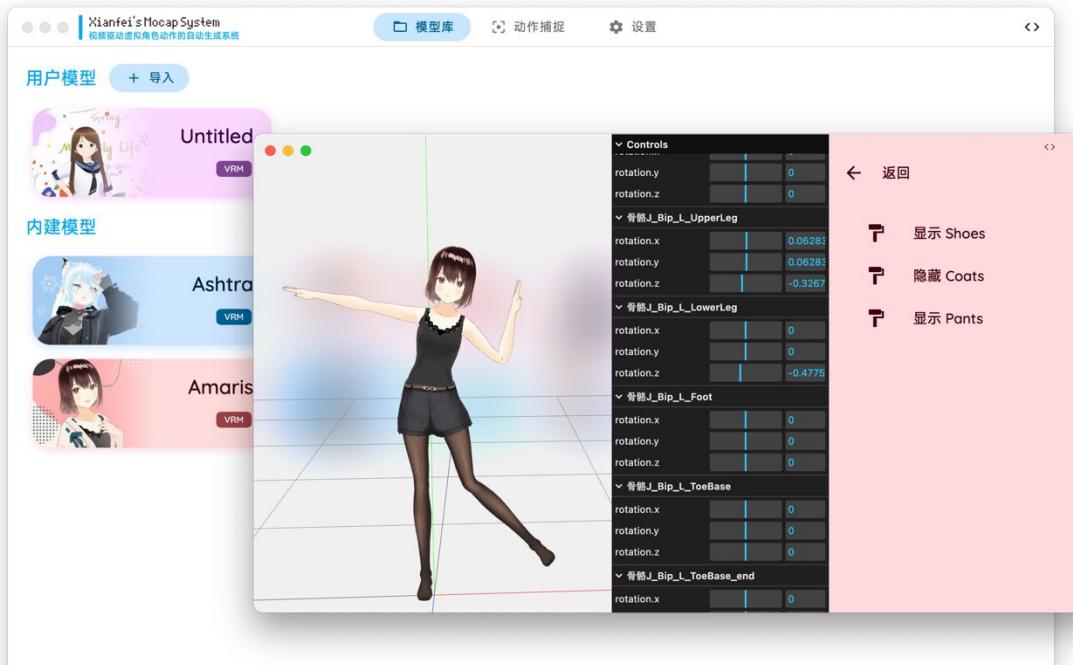


图 3-7 虚拟形象骨骼控制及三维图层控制页面

### 3.5 渲染模块设计

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 3-8 为在 UE 导入 Metahuman 虚拟形象并让其播放动作动画的画面。

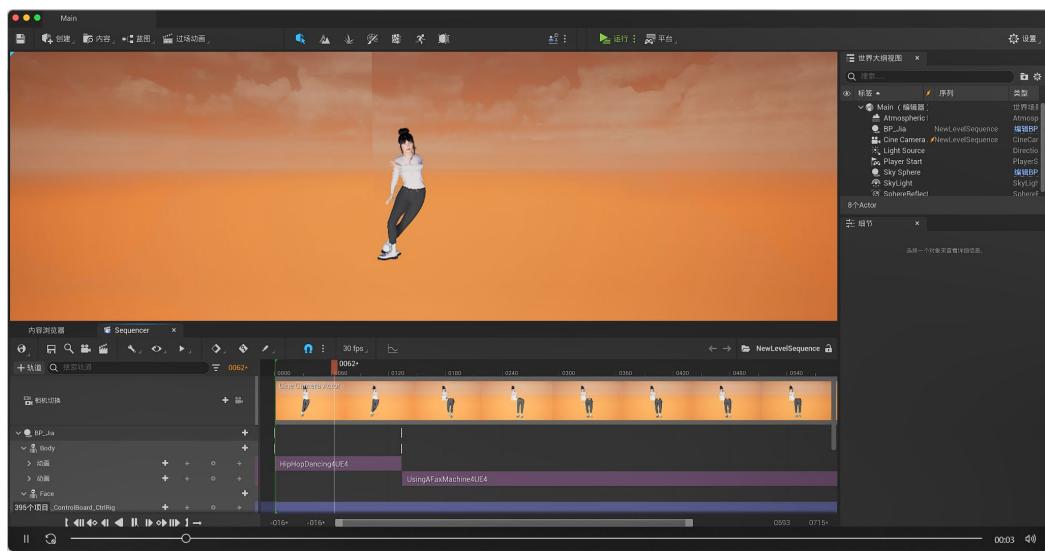


图 3-8 在 UE 中导入 Metahuman

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 3-9 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。

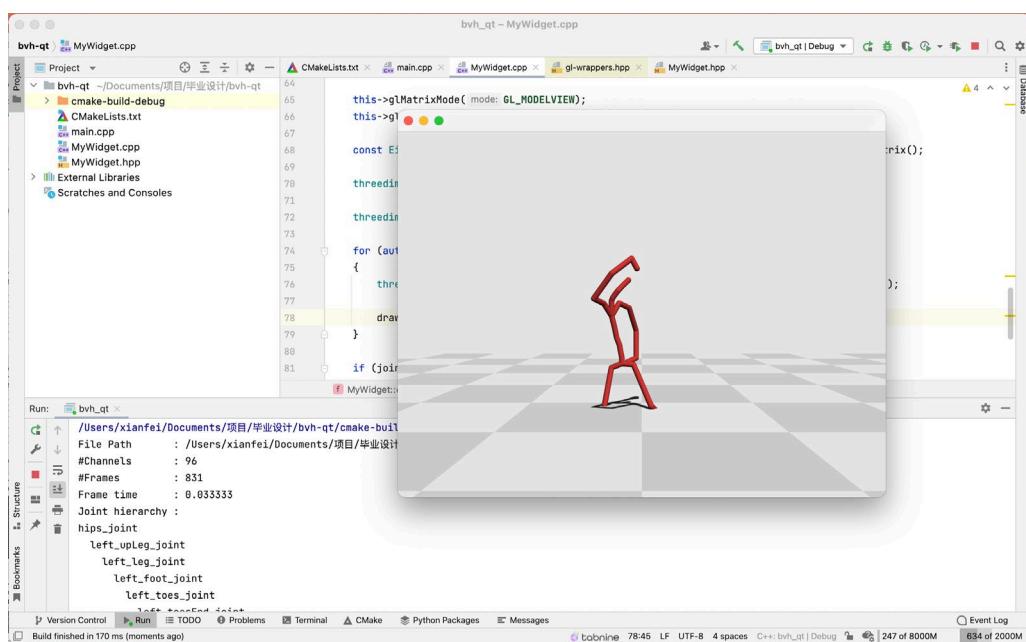


图 3-9 在 Qt 中渲染人体动作动画

在整个系统上，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 进行预览的。得益于它属于 Web 技术有着十分良好的可移植性及兼容性，第四章所介绍的 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。

### 3.6 图形用户界面设计

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 3-10 为该系统主页面并打开了两个虚拟形象预览窗口。



图 3-10 UI 设计

此外，整个系统采用了多语言设计，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且无需重启软件，大大的方便国际友人使用该软件。图 3-11 展示了不同语言下的设置页面。

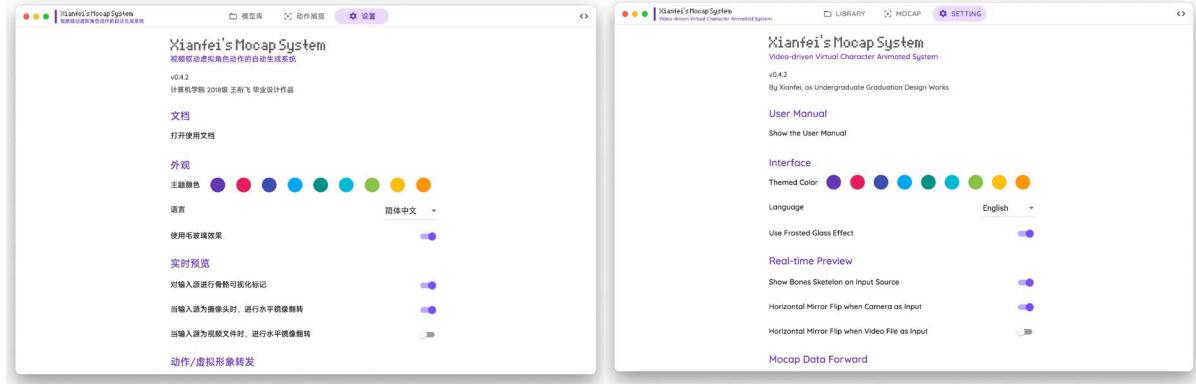


图 3-11 中文页面（左）与英文页面（右）

图 3-12 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 WASD 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 3-12 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 3-13 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 3-13 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 3-14 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 3-14 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)。该项目使用了可变字体 (Variable fonts, VF) 技术，字体的粗细程度可以无极调节，用于实现点击后字体变粗的效果。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。

## 第四章 实时动作数据及虚拟形象转发

### 4.1 转发系统概述

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系统，并将其进行实现。该系统采用 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 4-1 为相关部分的设置页面。



图 4-1 动作及虚拟形象转发部分相关设置

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。

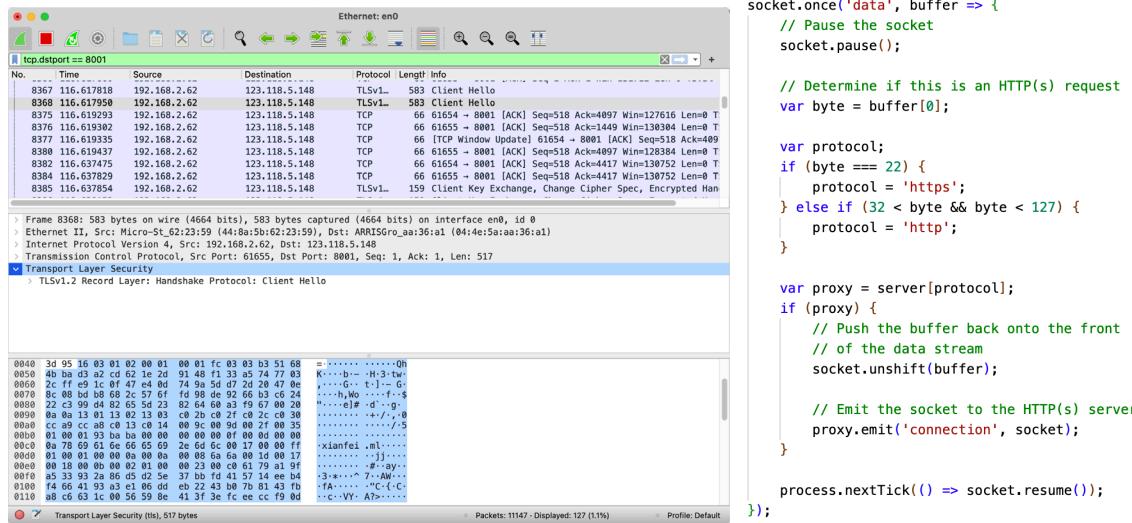


图 4-2 对 HTTPS 数据包分析（左）及服务器实现（右）

## 4.2 动作转发通信模块

该系统的动作转发协议使用 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”时，传输的是控制数据。

图 4-3 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”。

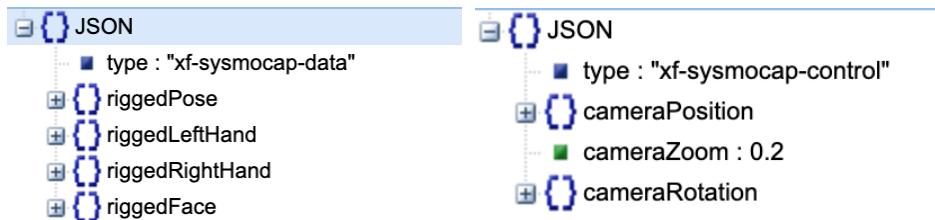


图 4-3 通信数据包一级结构

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 4-4 中展示了身体、手部、面部动作信息详细数据格式。

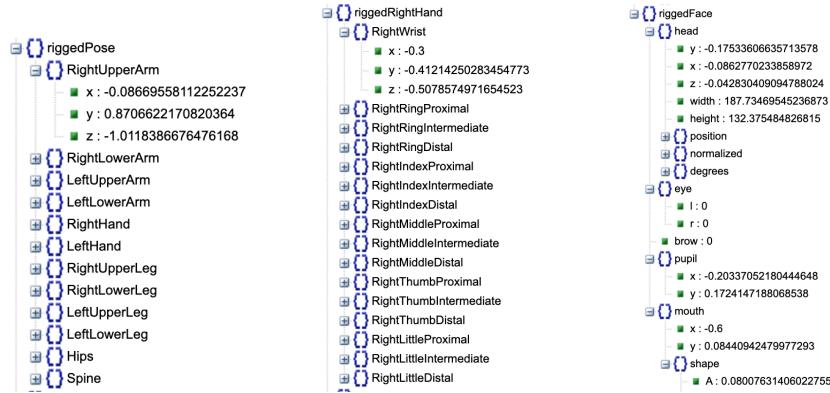


图 4-4 身体、手部、面部动作信息详细数据格式

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，以及虚拟摄像头的位置姿态及缩放。

### 4.3 用于 OBS 直播软件的接口设计与实现

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。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 `http://127.0.0.1:8080` (8080 为在设置内设置的端口号)，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 4-5 为在 OBS 中调用本程序接口获取生成的虚拟形象的画面。

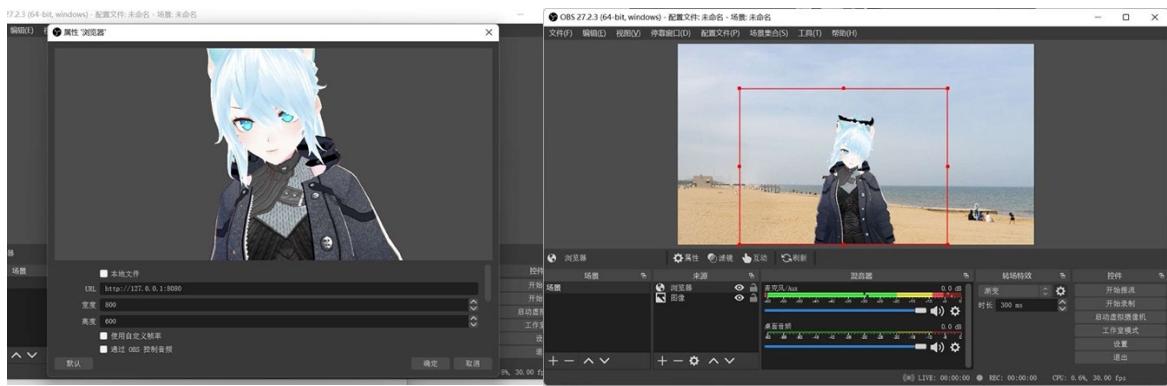


图 4-5 在 OBS 中调用本程序接口

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。

### 4.4 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 VR 需要专门的设备，所以仅在 VR 模拟器中完成了该部分的测试；而 AR 对于设备要求相对较低，需要带有 Google AR 框架的手机且通过 Google 认证，使用最新版 Chrome 即可。详细需求可以参见 Google AR Core 文档 (<https://developers.google.com/ar/develop/webxr/requirements>)。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 4-6 左侧是该软件画面，右侧为手机屏幕。

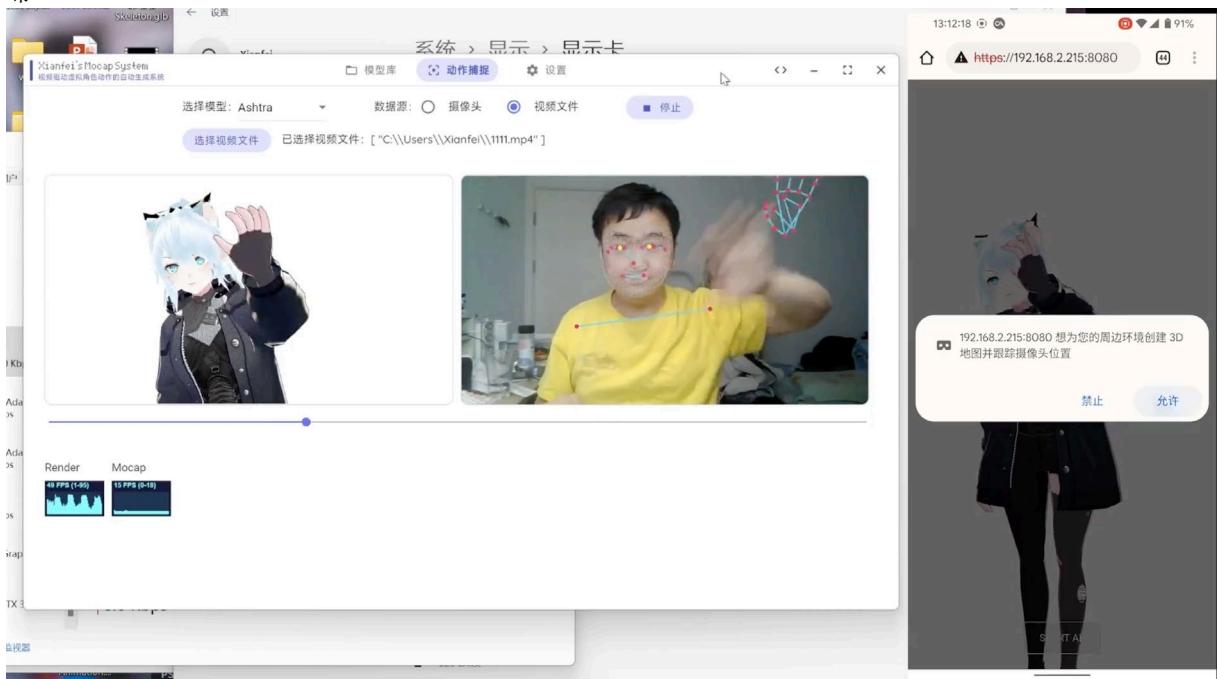


图 4-6 进入 AR 模式

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延时不超过 500ms。图 4-7 和图 4-8 为不同角度不同距离观察虚拟形象。

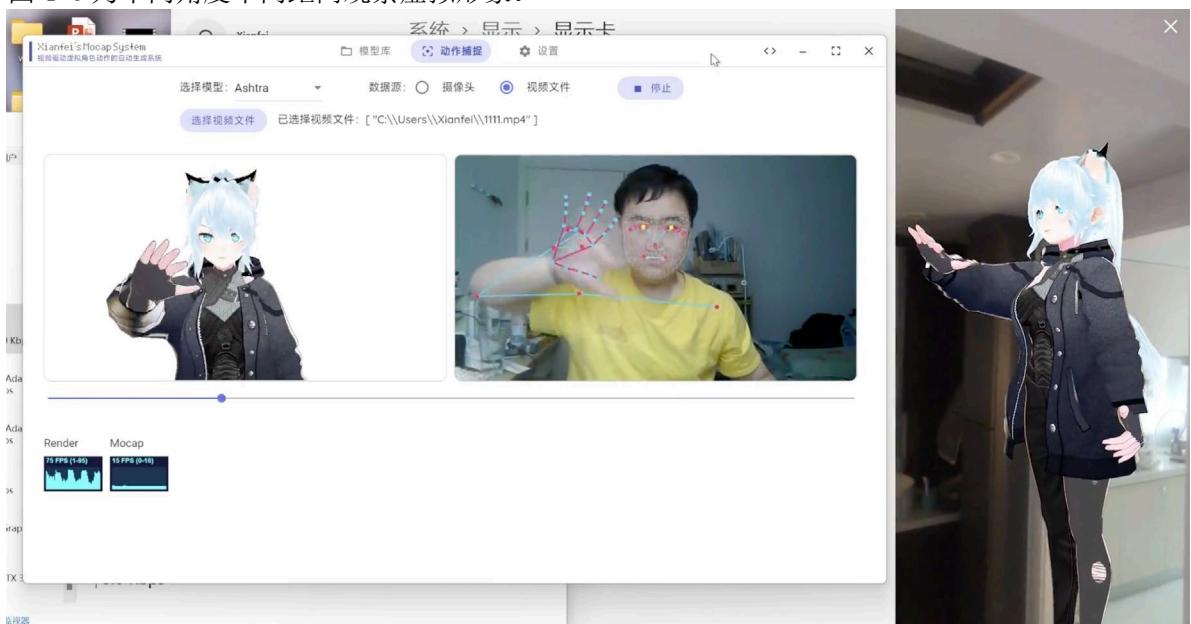


图 4-7 在 AR 模式中在虚拟形象右前方观察



图 4-8 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。<sup>[20]</sup>

## 第五章 动作捕捉算法及虚拟形象绑定算法

### 5.1 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)<sup>[13]</sup>、PyMAF (3D Human Pose and Shape Regression with Pyramidal Mesh Alignment Feedback Loop)<sup>[15]</sup>、MocapNET<sup>[16]</sup>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 TMCR 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，选择了舞蹈视频、健身视频进行测试，因为舞蹈和健身视频中所出现的动作比较复杂，能更好的测试算法的准确性。TMCR 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 (<https://www.bilibili.com/video/av766736553>) 作为测试，测试了这几种算法的性能，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 5-1 为 MocapNET 实时输出。

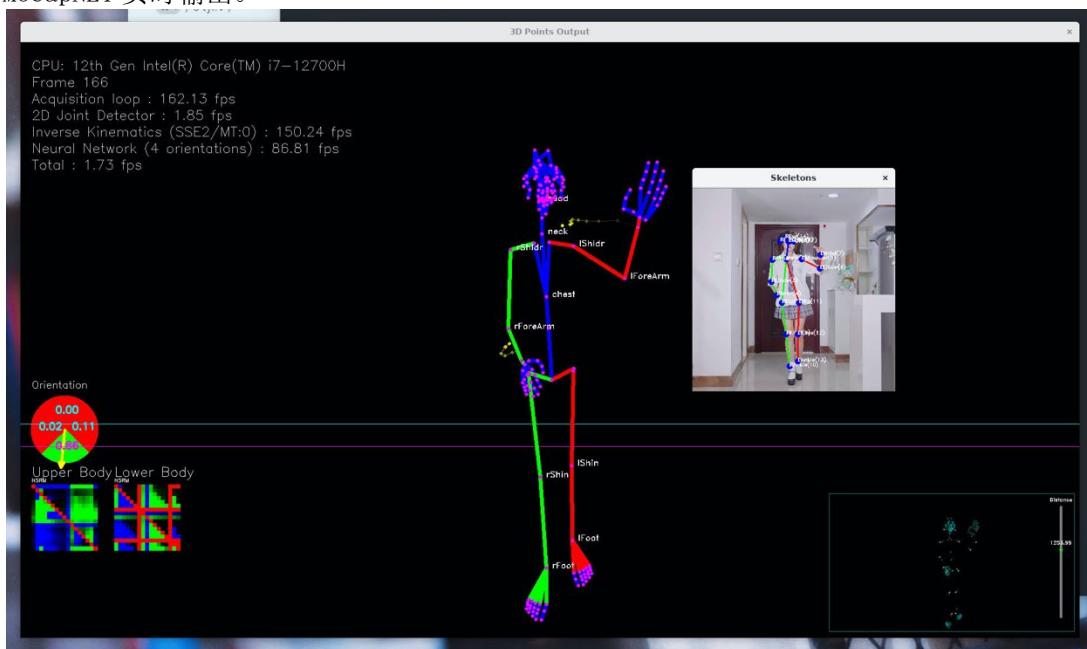


图 5-1 MocapNet 实时输出

测试结果见表 5-1，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 5-2 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 5-3 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 5-1 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 5-2 右手与身体前后关系推断不准确示意图



图 5-3 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。

## 5.2 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 5-4 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 5-4 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 算法获得，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模型，分别为 Lite、Full、Heavy。

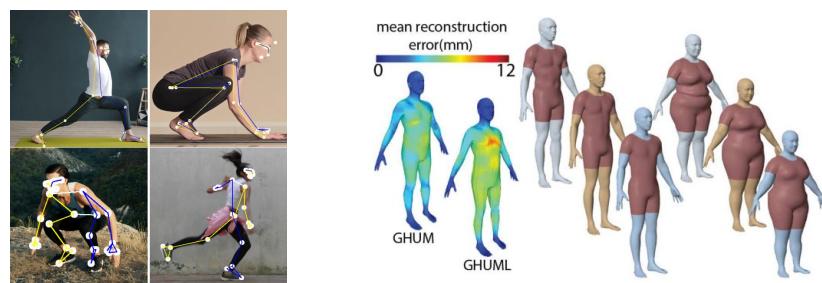


图 5-4 BlazePose 示意图（左）与 GHUM 示意图（右）

此外，BlazePose 算法还可与实时手部检测算法 BlazePalm<sup>[17]</sup>、实时面部检测算法 BlazeFace<sup>[18]</sup>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<sup>[19]</sup>。



图 5-5 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，该模型对各个地区的人的识别准确率都很高。表 5-2 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（Percentage of Detected Joints, PDJ）及标准差（Standard deviation）。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，且能获得更小的差值区间（Range），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。

表 5-2 BlazePose GHUM 3D 对 14 中不同地区人的识别准确度

Region	Lite model		Full model		Heavy model	
	PDJ	Standard deviation	PDJ	Standard deviation	PDJ	Standard deviation
Australia and New Zealand	94.1	8.3	95.4	7.6	98.0	3.9
Caribbean	94.8	8.3	97.7	4.9	98.9	2.9
Europe	90.3	12.6	95.1	8.4	97.8	4.8
Northern Africa	94.7	8.2	97.7	5.2	98.9	3.3
South America	95.0	8.3	97.8	4.8	99.0	3.0
Southeastern Asia	94.5	8.1	97.1	5.1	98.5	3.9
Western Asia	94.9	7.8	97.7	5.6	99.0	2.9
Central America	95.4	6.5	97.6	4.5	98.6	2.9
Central Asia	94.3	8.7	96.8	5.4	97.9	4.7
Eastern Asia	91.3	12.5	94.6	8.3	97.4	5.1
Middle Africa	94.6	9.5	96.6	6.6	98.3	4.6
Northern America	93.4	8.6	96.2	6.3	98.7	3.3
Southern Africa	92.9	10.0	95.1	6.9	97.0	6.3
Southern Asia	93.4	9.0	96.8	6.2	98.2	4.5
<b>Average</b>	<b>93.8</b>		<b>96.6</b>		<b>98.3</b>	
<b>Range</b>	<b>5.1</b>		<b>3.2</b>		<b>2.0</b>	

此外，其还对该模型对于不同肤色、不同性别的群体进行准确度测试。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差距区间越小。表 5-4 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，平均相差区间百分比在 1%-1.6% 之间。

表 5-3 BlazePose GHUM 3D 对不同肤色的人识别准确度

Skin tone type	% of dataset	Lite model		Full model		Heavy model	
		PDJ	Standard deviation	PDJ	Standard deviation	PDJ	Standard deviation
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	9.0	97.0	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
<b>Average</b>		<b>94.2</b>		<b>96.3</b>		<b>98.2</b>	
<b>Range</b>		<b>4.9</b>		<b>2.5</b>		<b>1.1</b>	

表 5-4 BlazePose GHUM 3D 对不同性别的人识别准确度

Gender	% of dataset	Lite model		Full model		Heavy model	
		PDJ	Standard deviation	PDJ	Standard deviation	PDJ	Standard deviation
Male	45.9	94.7	9.0	97.3	5.67	98.9	3.5
Female	54.1	93.1	9.5	96.0	6.80	97.9	4.7
<b>Average</b>		<b>93.9</b>		<b>96.7</b>		<b>98.4</b>	
<b>Range</b>		<b>1.6</b>		<b>1.3</b>		<b>1.0</b>	

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，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 5-5 所示。

表 5-5 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（代表大量复杂动作的视频，<https://www.bilibili.com/video/av766736553>）、刘畊宏《本草纲目》健身操视频（代表大量简单动作的视频，<https://www.bilibili.com/video/av596060641>）、Bilibili 博主“浅影阿\_”的唱歌视频（代表只包含上半身的视频，<https://www.bilibili.com/video/av683893544>）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 5-6，Mocap 下方图标上的 fps 数值即为动作捕捉帧数。

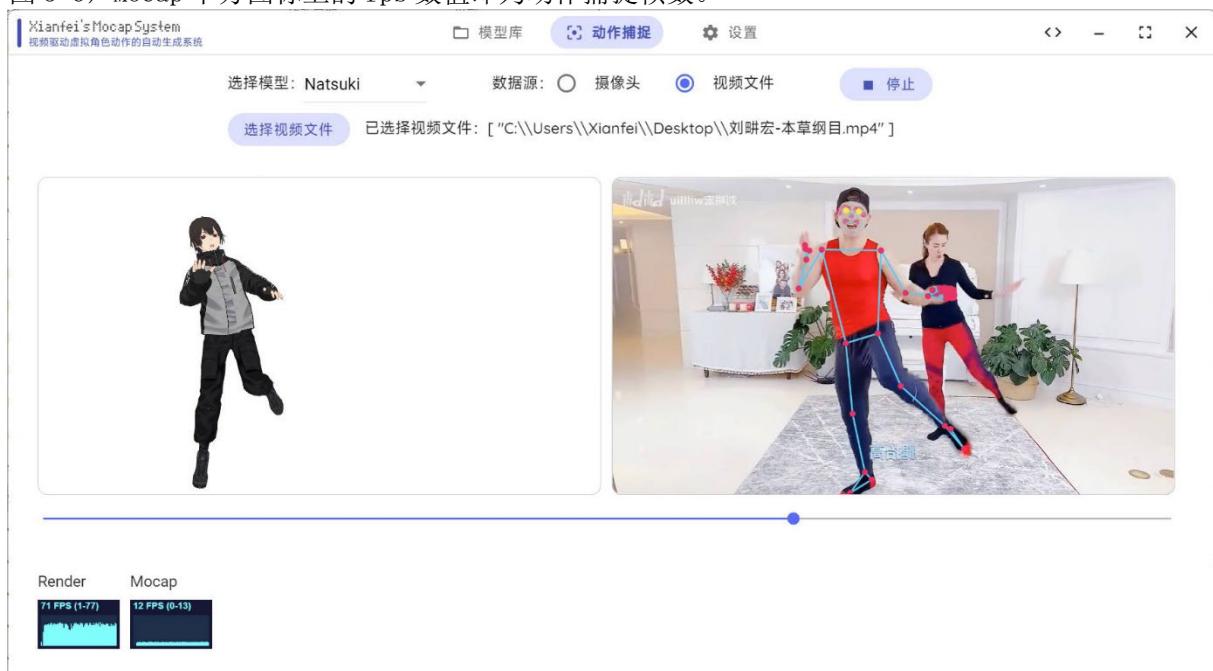


图 5-6 使用 FPS 监视器记录帧率

从表 5-6 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）。总体效果配合上时域帧内差值算法后显示效果基本令人满意。

表 5-6 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

### 5.3 动作捕捉骨骼架构转换算法

通过上文中图 5-4 可知 BlazePose 3D 算法输出的骨骼样式是不包含脊柱的，而大多数虚拟形象的动作骨架都包含脊柱节点，如图 5-7 所示为 Biovision Hierarchy（下文简写为 BVH）结构，其包含 18 个节点。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。该格式骨骼类型和数量与大多数虚拟形象 3D 模型文件相似（通常只是命名不同）。

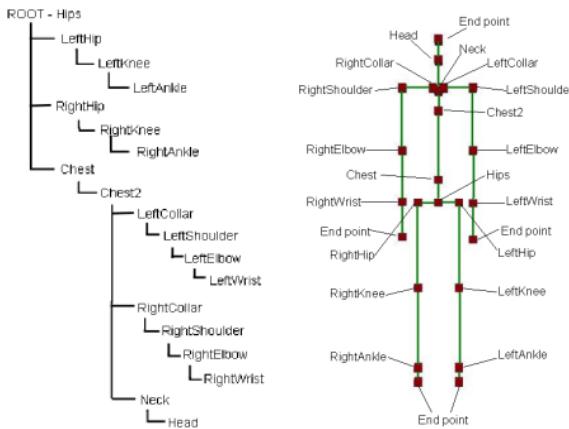
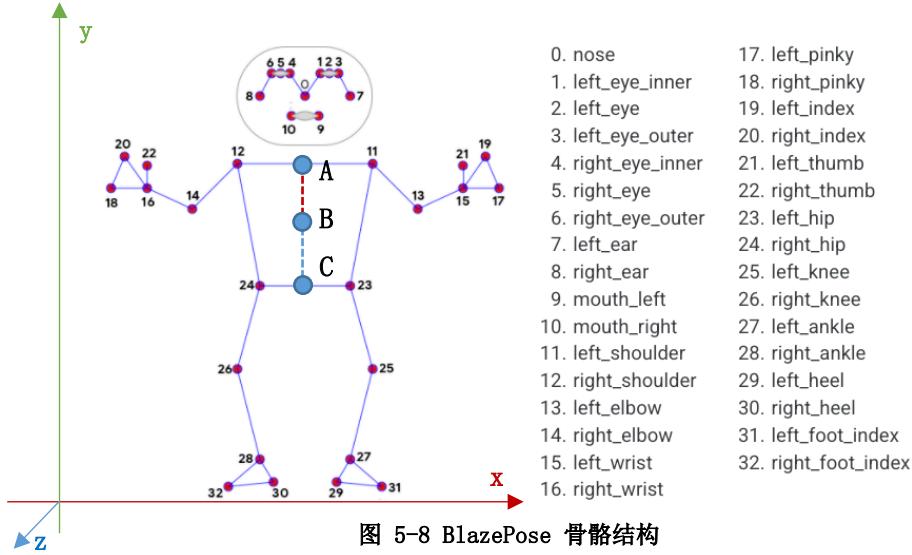


图 5-7 BVH 骨骼结构

而 Mediapipe 输出的人体骨骼信息是不包含脊柱的，其余的四肢信息结构和 BVH 基本一致。如图 5-8 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 (left\_shoulder) 与点 12 (right\_shoulder) 构成线段的垂直平分线，AC 为点 23 (left\_hip) 与点 24 (right\_hip) 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。



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 5-8 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 5-9。

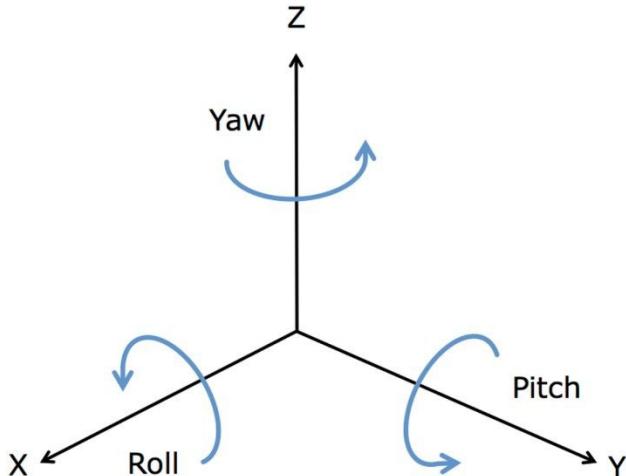


图 5-9 欧拉角关系

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  $\theta_x$ ，与 y 轴夹角记为  $\theta_y$ ，与 z 轴夹角记为  $\theta_z$ ；将点 11 与点 12 所构成的线段与 x 轴夹角记为  $\vartheta_x$ ，与 y 轴夹角记为  $\vartheta_y$ ，与 z 轴夹角记为  $\vartheta_z$ ；那么 BC 骨骼的姿态 Roll 为  $\frac{\pi}{2} - \theta_x$ ，Pitch 等于 AC 与 y 轴的夹角，Yaw 为  $\theta_z$ 。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  $\vartheta_x - \theta_x$ ，Pitch 为 0 (由于躯干无法分析弯腰的特性)，Yaw 为  $\vartheta_z - \theta_z$ 。通过此种计算即可得到脊柱骨骼信息。

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。这种骨骼形式骨架与 BVH 几乎一致，但比 BVH 要更加简洁，仅包含了 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）。

## 5.4 虚拟形象骨骼映射算法

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（由于其他建模软件可以让用户自定义骨骼名称，实现绑定较为困难，这两个软件的骨骼架构和名称用户不能自行更改，使用较为简易并且比较流行）。

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，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。图 5-10 为用于映射 Mixamo 标准骨骼的映射表。

```
"binding": {
    "Hips": "mixamorigHips",
    "Neck": "mixamorigNeck",
    "Chest": "mixamorigSpine2",
    "Spine": "mixamorigSpine",
    "RightUpperArm": "mixamorigRightArm",
    "RightLowerArm": "mixamorigRightForeArm",
    "LeftUpperArm": "mixamorigLeftArm",
    "LeftLowerArm": "mixamorigLeftForeArm",
    "LeftUpperLeg": "mixamorigLeftUpLeg",
    "LeftLowerLeg": "mixamorigLeftLeg",
    "RightUpperLeg": "mixamorigRightUpLeg",
    "RightLowerLeg": "mixamorigRightLeg"
}
```

图 5-10 Mixamo 骨骼映射表

对于 VRM 格式模型，VRM 提供了标准查找表 VRMSchema.HumanoidBoneName，其命名格式兼容上一小节提出的中间格式（中间格式是 VRMSchema 的子集），图 5-11 中节选了部分 VRM 标准查找表，可以看得出，该表的 Key 部分与本论文定义的中间格式命名方式一致。

```
> THREE.VRMSchema.HumanoidBoneName
< ▾ {Chest: 'chest', Head: 'head', Hips: 'hips', Jaw: 'jaw', LeftEye: 'leftEye', ...} ⓘ
  Chest: "chest"
  Head: "head"
  Hips: "hips"
  Jaw: "jaw"
  LeftEye: "leftEye"
  LeftFoot: "leftFoot"
  LeftHand: "leftHand"
  LeftIndexDistal: "leftIndexDistal"
  LeftIndexIntermediate: "leftIndexIntermediate"
  LeftIndexProximal: "leftIndexProximal"
  LeftLittleDistal: "leftLittleDistal"
  LeftLittleIntermediate: "leftLittleIntermediate"
```

图 5-11 VRMSchema 骨骼映射表

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。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。为了方便开发，该系统创建了 rotationBones 函数，接受骨骼名称 name 参数和旋转欧拉角 rotation 参数。图 5-12 中左侧为针对 VRM 格式的模型使用 VRMSchema.HumanoidBoneName 标准查找表，右侧为针对 GLB/GLTF/FBX 格式的模型使用自定义查找表(modelObj.binding)进行骨骼映射并使用 lambda 表达式进行查找。算法大体思路一直，均为查找到对应骨骼并更改他的旋转信息（VRM 骨骼使用四元数表示旋转）。

```
const b = currentVrm.humanoid.getBoneNode(
  THREE.VRMSchema.HumanoidBoneName[name]
);
if (b) {
  let euler = new THREE.Euler(
    rotation.x,
    rotation.y,
    rotation.z,
    rotation.rotationOrder || "XYZ"
  );
  b.quaternion = new THREE.Quaternion().setFromEuler(euler);
}

// convert name with model json binding info
name = modelObj.binding[name];
// find bone in bones by name
var b = skeletonHelper.bones.find((bone) => bone.name == name);
if (b) {
  b.rotation.x = rotation.x;
  b.rotation.y = rotation.y;
  b.rotation.z = rotation.z;
} else {
  console.log("Can not found bone " + name);
}
```

图 5-12 rotationBones 函数中骨骼映射算法关键部分

## 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 平台下，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。

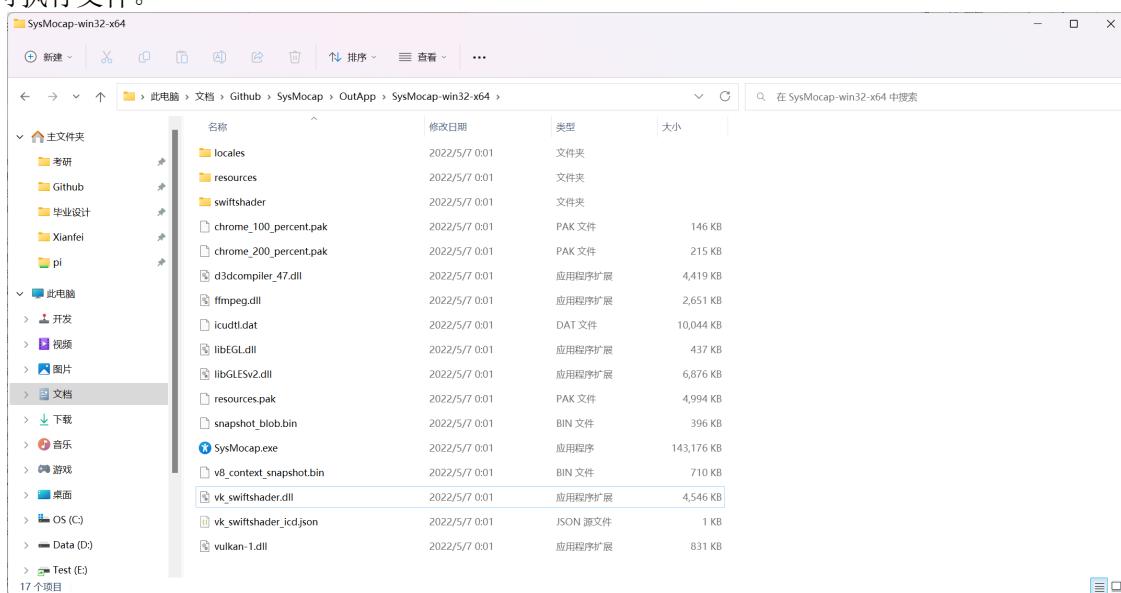


图 6-2 Windows 平台上打包产物

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。

```
window.startMocap = async function (e) {
  if (process.platform == "darwin" && app.videoSource == "camera") {
    if (
      remote.systemPreferences.getMediaAccessStatus("camera") !==
      "granted"
    ) {
      if (!(await remote.systemPreferences.askForMediaAccess("camera"))) {
        alert("需要授予摄像头使用权限");
        return;
      }
    }
  }
}
```



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。

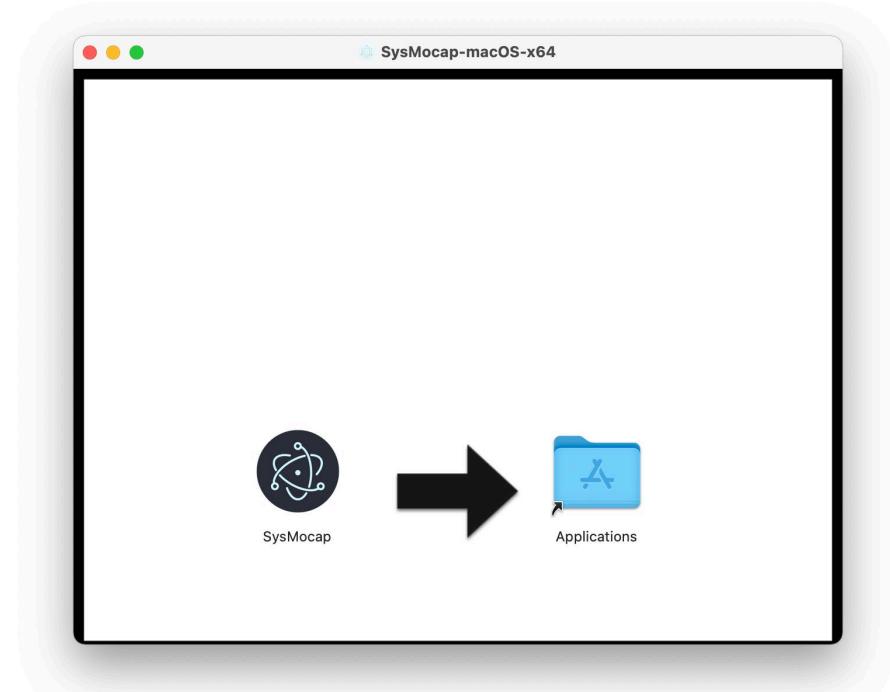


图 6-6 在 macOS 系统上安装镜像 dmg 文件

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。

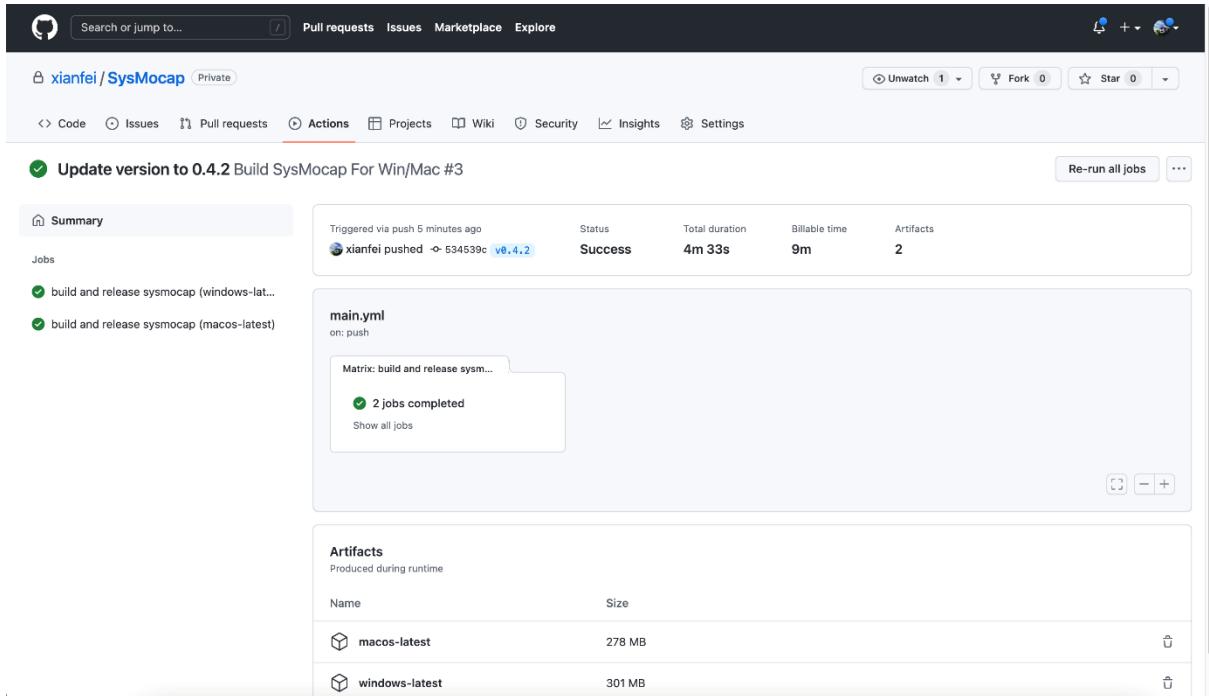


图 6-8 Github Actions 工作页面

### 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
          (A grid of 16 small video frames showing various character poses)

Video-based Motion Capture & Character Animation System
by xianfei

SysMocap
A real-time video-based motion capture system for virtual character animating
and rendering. It's a free and open-sourced software.

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 0x8000000001 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 worker\_threads 工作线程。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，使用千兆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。最终我们发现了可以运行在 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，告诉我了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，也希望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，感谢你们，因为有你们，温暖了四季。



## 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] Z. Cao, G. Hidalgo, T. Simon, S. Wei and Y. Sheikh, "OpenPose: Realtime Multi-Person 2D Pose Estimation Using Part Affinity Fields" in IEEE Transactions on Pattern Analysis & Machine Intelligence, vol. 43, no. 01, pp. 172–186, 2021.
- [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] E. Remelli, S. Han, S. Honari, P. Fua and R. Wang, "Lightweight Multi-View 3D Pose Estimation Through Camera-Disentangled Representation," in 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, 2020 pp. 6039–6048.
- [7] Zhang, Zhengyou. (2012). Microsoft Kinect Sensor and Its Effect. IEEE Multimedia – IEEEMM. 19. 4–10. 10.1109/MMUL.2012.24.
- [8] Herring, Susan & Dainas, Ashley & Lopez Long, Holly & Tang, Ying. (2020). Animoji Adoption and Use: Gender Associations with an Emergent Technology. 10.36190/2020.03.
- [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] 贺怀清;洪炳熔. 一种虚拟人运动拟合算法[J]. 电子学报, 2001, 29(8): 1107–1109.
- [11] Lugaressi, Camillo & Tang, Jiuqiang & Nash, Hadon & McClanahan, Chris & Ubweja, Esha & Hays, Michael & Zhang, Fan & Chang, Chuo-Ling & Yong, Ming & Lee, Juhyun & Chang, Wan-Teh & Hua, Wei & Georg, Manfred & Grundmann, Matthias. (2019). MediaPipe: A Framework for Building Perception Pipelines. arXiv:1906.08172
- [12] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Heben Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford,

- Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, Wojciech Zaremba. (2021) Evaluating Large Language Models Trained on Code. arXiv:2107.03374
- [13]Hongsuk Choi, Gyeongsik Moon, Ju Yong Chang, Kyoung Mu Lee. (2021) Beyond Static Features for Temporally Consistent 3D Human Pose and Shape from a Video. arXiv:2011.08627
- [14]Andrei Zanfir, Eduard Gabriel Bazavan, Hongyi Xu, Bill Freeman, Rahul Sukthankar, and Cristian Sminchisescu. Weakly supervised 3d human pose and shape reconstruction with normalizing flows. In European Conference on Computer Vision, 6(2): pages 465 - 481, 2020.
- [15]Zhang, Hongwen & Tian, Yating & Xinch, Zhou & Ouyang, Wanli & Liu, Yebin & Wang, Limin & Sun, Zhenjun. (2021). PyMAF: 3D Human Pose and Shape Regression with Pyramidal Mesh Alignment Feedback Loop. 11426–11436. 10.1109/ICCV48922.2021.01125.
- [16]Qammaz, Ammar and ammarkov. (2021) Towards Holistic Real-time Human 3D Pose Estimation using MocapNETs. British Machine Vision Conference (BMVC 2021)
- [17]Fan Zhang, Valentin Bazarevsky, Andrey Vakunov, Andrei Tkachenka, George Sung, Chuo-Ling Chang, Matthias Grundmann. (2020) MediaPipe Hands: On-device Real-time Hand Tracking. arXiv:2006.10214
- [18]Valentin Bazarevsky, Yury Kartynnik, Andrey Vakunov, Karthik Raveendran, Matthias Grundmann. (2019) BlazeFace: Sub-millisecond Neural Face Detection on Mobile GPUs. arXiv:1907.05047
- [19]Vasileios Choutas, Georgios Pavlakos, Timo Bolkart, Dimitrios Tzionas, and Michael J. Black. Monocular expressive body regression through body-driven attention. In European Conference on Computer Vision, 3(1): pages 20 - 40, 2020.
- [20]Lee, D. ; Shim, W. ; Lee, M. ;Lee, S. ; Jung, K.-D. ; Kwon, S. Performance Evaluation of Ground AR Anchor with WebXR Device API. Appl. Sci. 2021, 11, 7877.

