天津大学

本科生毕业论文

学　　院　软件学院

专　　业　软件工程

年　　级　2014 级

姓　　名　晏伟豪

指导教师　翁仲铭

2018 年 5 月 27 日

天津大学

本科生毕业设计任务书

题目：基于手势控制设备之虚拟现实交互游戏设计

学生姓名　晏伟豪

学院名称　软件学院

专　　业　软件工程

学　　号　**3014218143**

指导教师　翁仲铭

职　　称　副教授

一、原始依据（包括设计或论文的工作基础、研究条件、应用环境、工作目的等。）

1. 工作基础

面向 PC 以及 Mac 的体感控制器制造公司 Leap 于 2013 年 2 月 27 日宣布，公司旗下产品 Leap Motion 体感控制器将于 5 月 13 日正式上市，随后于 5 月 19 日在美国零售商百思买独家售卖。2013 年 7 月 22 日，新版 Leap Motion 已经开始派送，新版的 Leap Motion 将具有更高的软硬件结合能力。

Leap Motion 体感控制器支持 Windows 7、Windows 8 以及 Mac OS X 10.7 及 10.8，该设备功能类似 Kinect，可以在 PC 及 Mac 上通过手势控制电脑。

Leap Motion 控制器不会替代您的键盘、鼠标、手写笔或触控板，相反，它与它们协同工作。当 Leap Motion 软件运行时，只需将它插入您的 Mac 或 PC 中，一切即准备就绪。

只需挥动一只手指即可浏览网页、阅读文章、翻看照片，还有播放音乐。即使不使用任何画笔或笔刷，用您的指尖即可以绘画，涂鸦和设计。

用您的手指即可切水果、打坏蛋；用您的双手即可飙赛车，打飞机。您可以在 3D 空间进行雕刻、浇铸、拉伸、弯曲以及构建 3D 图像，还可以把他们拆开以及再次拼接。体验一种全新的学习方式，用您的双手探索宇宙，触摸星星，还可以围绕太阳翱翔。一种全新的乐器体验，弹奏空气吉他、空气竖琴和空中的一切乐器，还可以体验全新的采摘和拾起方式。

本次设计中，学生将采用 Leapmotio 作为控制来设计一款 VR 游戏。

2. 研究条件

大体上，Leap 传感器根据内置的两个摄像头从不同角度捕捉的画面，重建出手掌在真实世界三维空间的运动信息。检测范围大提在传感器上方 25mm 到 600mm 之间，检测的空间大提是一个倒四棱锥体。

Leap Motion 传感器会建立一个直角坐标系，坐标的圆点是传感器的中心，坐标的 X 轴平行于传感器，指向屏幕右方。Y 轴指向背离屏幕方向。

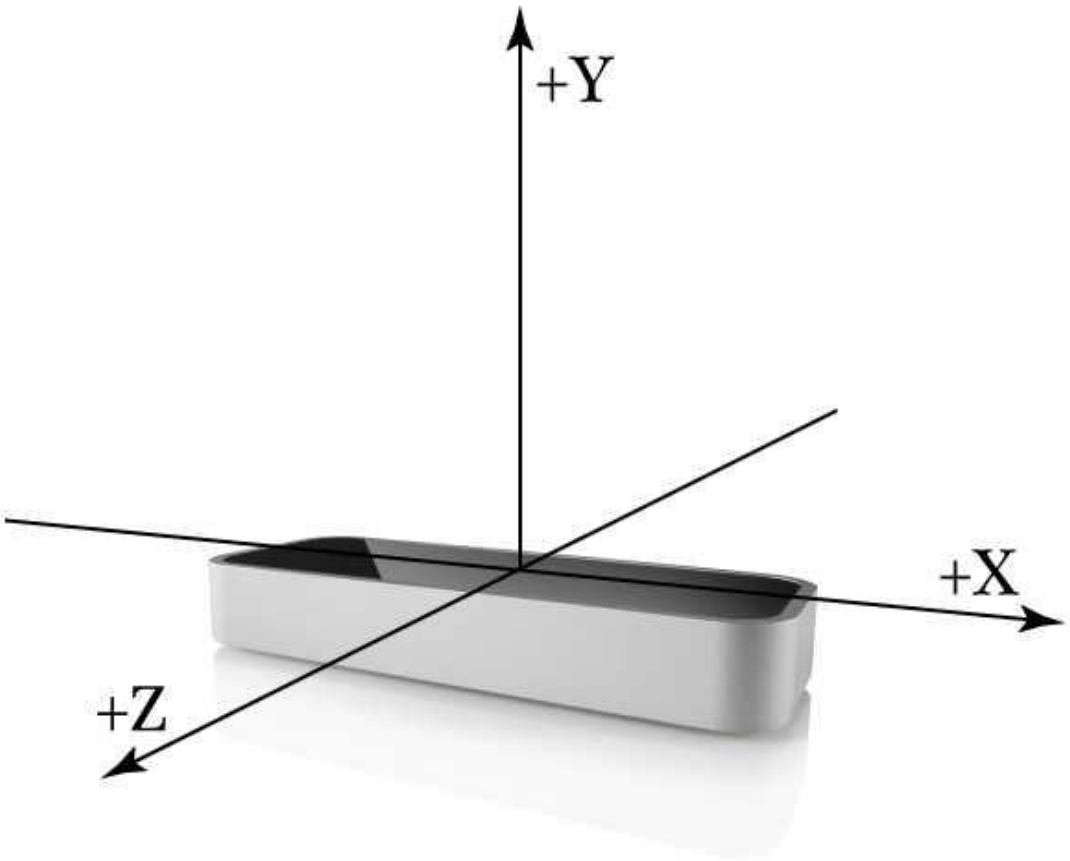
在使用过程中，LeapMotion 传感器会定期发送关于手的运动信息，每份这样的信息成为 frame，每一个这样的 frame 包含检测到的

--所有手掌的列表和信息；

--所有手指的列表和信息；

--手持工具的列表和信息；

--所有可指向对象，即所有手指和工具的列表和信息；



3 应用环境与目的

在翁仲铭老师目前的研究团队中，在 55 楼教学楼内拥有 Oculus 头盔以及 Leapmotion 设备，用 Unity3d 游戏引擎开发出一个基于手势控制的 VR 游戏。

二、参考文献

[1] An immersive acupuncture Training Method based on Machine learning.

[2] Lin S T, Yin C Q. Gesture for Numbers Recognition Based on LeapMotion[J]. Computer Knowledge & Technology, 2015.

[3] Jin H, Chen Q, Chen Z, et al. Multi-LeapMotion sensor based demonstration for robotic refine tabletop object manipulation task[J]. Caai Transactions on Intelligence Technology, 2016, 1(1):104-113.

[4] Zhang Q, Deng F. Dynamic Gesture Recognition based on LeapMotion and HMM-CART Model[C]// 2017:012037.

[5] Lee B, Park K, Ghan S, et al. Designing Canonical Form of Finger Motion Grammar in Leapmotion Contents[C]// International Conference on Mechatronics, Control and Automation Engineering. 2016.

三、设计（研究）内容和要求（包括设计或研究内容，主要指标与技术参数，并根据课题性质对学生提出具体要求）

1. 研究内容

在本毕业设计中，主要的研究内容是对于 Leapmotion 设备进行深入了解，并且与 Oculus 相结合来帮助使用者与虚拟场景交互。在此基础上，通过 Unity3d 设计出一款与 Leapmotion 控制结合的游戏，完成手势控制功能与游戏的结合。

2. 主要指标： 其研究需要经过下列几个步骤

--查阅相关资料与研究成果。
--研究 Leapmotion 控制的原理
--学习并熟练使用 Unity3d 游戏引擎
--基于手势控制与虚拟现实，设计游戏原型（玩法，模型等）
--游戏后期测试，不断改进

3.对学生要求

--需构建一套完成的虚拟现实环境
--设计并实现一款有用可玩性的虚拟现实游戏
--比较 Leapmotion 在不同环境下应用

指导教师（签字）

年　月　日

审题小组组长（签字）

年　月　日

天津大学本科生毕业设计开题报告

课题名称	基于手势操控设备之虚拟现实交互游戏设计		
学院名称	软件学院	专业名称	软件工程
学生姓名	晏伟豪	指导教师	翁仲铭

（内容包括：课题的来源及意义，国内外发展状况，本课题的研究目标、研究内容、研究方法、研究手段和进度安排，实验方案的可行性分析和已具备的实验条件以及主要参考文献等。）

一. 课题的来源及意义

面向 PC 以及 Mac 的体感控制器制造公司 Leap 于 2013 年 2 月 27 日宣布，公司旗下产品 Leap Motion 体感控制器将于 5 月 13 日正式上市，随后于 5 月 19 日在美国零售商百思买独家售卖。2013 年 7 月 22 日，新版 Leap Motion 已经开始派送，新版的 Leap Motion 将具有更高的软硬件结合能力。

Leap Motion 体感控制器支持 Windows 7、Windows 8 以及 Mac OS X 10.7 及 10.8，该设备功能类似 Kinect，可以在 PC 及 Mac 上通过手势控制电脑。

Leap Motion 控制器不会替代您的键盘、鼠标、手写笔或触控板，相反，它与它们协同工作。当 Leap Motion 软件运行时，只需将它插入您的 Mac 或 PC 中，一切即准备就绪。

只需挥动一只手指即可浏览网页、阅读文章、翻看照片，还有播放音乐。即使不使用任何画笔或笔刷，用您的指尖即可以绘画，涂鸦和设计。

用您的手指即可切水果、打坏蛋；用您的双手即可飙赛车，打飞机。您可以在 3D 空间进行雕刻、浇铸、拉伸、弯曲以及构建 3D 图像，还可以把他们拆开以及再次拼接。体验一种全新的学习方式，用您的双手探索宇宙，触摸星星，还可以围绕太阳翱翔。一种全新的乐器体验，弹奏空气吉他、空气竖琴和空中的一切乐器，还可以体验全新的采摘和拾起方式。

二. 国内外发展状况

面向 PC 以及 Mac 的体感控制器制造公司 Leap 于 2013 年 2 月 27 日宣布，公司旗下产品 Leap Motion 体感控制器将于 5 月 13 日正式上市，随后于 5 月 19 日在美国零售商百思买独家售卖。2013 年 7 月 22 日，新版 Leap Motion 已经开始派送，新版的 Leap Motion 将具有更高的软硬件结合能力。

LeapMotion 官网仍旧继续接受美国以及全球其他地区消费者的订单，不过价格将由此前的 69.99 美元，上涨到 79.99 美元，约合人民币 500 元。在 2 月 27 日前进行预订的用户，价格依旧是 69.99 美元。

Leap Motion 体感控制器支持 Windows 7、Windows 8 以及 Mac OS X 10.7 及 10.8，该设备功能类似 Kinect，可以在 PC 及 Mac 上通过手势控制电脑。

该公司也为其发布了名为 Airspace 的应用程序商店，其中包括游戏、音乐、教育、艺术等分类。

已经有包括迪斯尼、Autodesk、Google 在内的公司均已宣称部分旗下软件游戏支持 Leap Motion，其中包括赛车游戏《Wreck-It Ralph: Sugar Rush Speedway》、Autodesk 的 Maya 插件、Google Earth、Cut the Rope（切绳子），以及其他应用，另外流行的事件管理器 Clear Mac 版同样支持 Leap Motion 体感动作操控

三. 研究目标

探讨并研究用 Unity3d 引擎设计出一款与 LeapMotion 结合的 VR 游戏

四. 研究内容

在本毕业设计中，主要的研究内容是对于 Leapmotion 设备进行深入了解，并且与 Oculus 相结合来帮助使用者与虚拟场景交互。在此基础上，通过 Unity3d 设计出一块与 Leapmotion 控制结合的游戏，完成手势控制功能与游戏的结合

五. 研究方法

大体上，Leap 传感器根据内置的两个摄像头从不同角度捕捉的画面，重建出手掌在真实世界三维空间的运动信息。检测范围大提在传感器上方 25mm 到 600mm 之间，检测的空间大提是一个倒四棱锥体。

Leap Motion 传感器会建立一个直角坐标系，坐标的圆点是传感器的中心，坐标的 X 轴平行于传感器，指向屏幕右方。Y 轴指向背离屏幕方向。

在使用过程中，LeapMotion 传感器会定期发送关于手的运动信息，每份这样的信息成为 frame，每一个这样的 frame 包含检测到的

--所有手掌的列表和信息；

--所有手指的列表和信息；

--手持工具的列表和信息；

--所有可指向对象，即所有手指和工具的列表和信息；

个人电脑即可完成。

九．　已具备的实验条件

硬件条件：具有虚拟现实功能的 oculus 头盔，Leapmotion 手势控制设备，个人电脑。

软件条件：游戏设计引擎：Unity3d

十．　参考文献

[1] An immersive acupuncture Training Method based on Machine learning.

[2] Lin S T, Yin C Q. Gesture for Numbers Recognition Based on LeapMotion[J]. Computer Knowledge & Technology, 2015.

[3] Jin H, Chen Q, Chen Z, et al. Multi-LeapMotion sensor based demonstration for robotic refine tabletop object manipulation task[J]. Caai Transactions on Intelligence Technology, 2016, 1(1):104-113.

[4] Zhang Q, Deng F. Dynamic Gesture Recognition based on LeapMotion and HMM-CART Model[C]// 2017:012037.

[5] Lee B, Park K, Ghan S, et al. Designing Canonical Form of Finger Motion Grammar in Leapmotion Contents[C]// International Conference on Mechatronics, Control and Automation Engineering. 2016.

选题是否合适：　是□　否□

课题能否实现：　能□　不能□

指导教师（签字）

年　月　日

选题是否合适：　是□　否□

课题能否实现：　能□　不能□

审题小组组长（签字）

年　月　日

摘　要

LeapMotion 体感控制器是一款通过手势来控制软件或硬件的一种设备，本文探讨的是将 Leap Motion 手势识别与 Unity 跑酷游戏结合的示例。文章将分别介绍 Leap Motion 的基本原理，手势控制的识别与建立以及游戏的完成与实现，并结合实际情况进行说明。最后通过 Leap Motion 控制器操控与键盘鼠标操控游戏的对比，说明该示例的优先与局限性，并对未来的 Leap Motion 发展进行了展望。

关键词：Leap Motion 体感控制器；Unity 游戏引擎；跑酷游戏；手势识别

ABSTRACT

The Leap Motion controller is a device that controls software or hardware through gestures. This paper discusses an example of combining Leap Motion gesture recognition with the Unity Parkour game. The paper will introduce the basic principle of Leap Motion, the recognition and establishment of gesture control, and the completion and realization of the game, and explain the actual situation. Finally, through the comparison between the Leap Motion controller and the keyboard and mouse, the priority and limitations of the example are illustrated and the future development of Leap Motion is prospected.

Key words：Leap Motion；Unity game engine；Parkour game；Gesture recognition

目　　录

第一章　研究背景

1.1 虚拟现实

虚拟现实（Virtual Reality）是一种通过感觉和感知来模拟体验的计算机生成场景。沉浸式环境可以类似于真实世界，或者它可以是幻想，在普通物理试验中创造不可能的体验。增强现实系统也可被视为 VR 一种形式，其通过将实时摄像头馈送到耳机或者通过智能手机或者平板设备的虚拟信息分层，是用户能够观看三维图像。

目前的虚拟现实技术通常使用虚拟现实耳机或多投影环境，有时结合物理环境或道具来生成模拟用户在虚拟或虚拟环境中的物理存在的逼真图像，声音或者其他感觉。使用虚拟现实设备的人能够观察人造世界，在其中一栋并且与虚拟特征或物品进行交互行为。这种效果通常是由头饰 VR 设备产生的，它在眼镜前面有一个小屏幕，但也可以由一个专门设计的房间和多个大屏幕创建。

包括振动和其他感觉在内的 VR 系统及其游戏控制或其他设备，被称为触觉系统。这种触觉信息通常被称为医学、视频游戏和军事训练应用中的力反馈。

1.2 LeapMotion 体感控制器

LeapMotion 控制器是一块小巧的 USB 外围设备，设计用于面朝上的方式放置在桌面上。它也可以安装到虚拟现实的头戴式设备上。使用两个单色红外摄像机和三个红外 LED，该设备可以观察一个大约距设备距离大约 1 米的半球区域。LED 产生图案的 IR 光，相机每秒产生几乎 200 帧的反射数据。然后通过 USB 电缆将其发送到主机，由 LeapMotion 软件通过公司“复杂数学”用以公司尚未公开的方式对其进行分析，以某种方式通过比较两台相机产生的 2D 数据帧来合成 3D 位置数据。在 2013 年的一项研究中，控制器的整体平均精确度显示为 0.7 毫米。

LeapMotion 最初将数千个单元分发给有兴趣为该设备创建应用程序的开发人员。LeapMotion 控制器在 2013 年 7 月首次发货。2016 年 2 月，LeapMotion 发布了其核心测试软件的主要测试版本更新。该软件的测试版本名为 Orion，专门为手部追踪在虚拟现实中的运用而设计。

1.3 Unity

Unity 是一款多用途游戏引擎，支持 2D 和 3D 图形，拖放功能以及使用 C# 编写脚本。支持另外两种编程语言：Boo，在 Unity5 发布时启用，而 JavaScript 在 2017 年 8 月发布的 Unity 后开始弃用。

该引擎针对以下图形 API:Windows 和 Xbox One 上的 Direct3d;Linux, macOS 和 Windows 上的 OpenGL; Android 和 iOS 上的 OpenGL ES; Web 上的 WebGL; 以及视频游戏控制台中的专有 API。此外，Unity 支持 iOS 和 macOS 上的地基 API METAL, Android, Linux 和 Windows 上的 Vulkan 以及 Windows 和 Xbox One 上的

Direct3D 12.

在 2D 游戏中，Unity 允许引入 sprites 和先进的 2D 游戏渲染器。对于 3D 游戏，Unity 允许为游戏引擎支持的每个平台指定纹理压缩，MIPMAP 和分辨率设置，支持反射贴图，视差贴图，屏幕空间环境遮挡，凹凸贴图，动态阴影贴图以及熏染到屏幕的纹理和阴影。开发者还会被提供多种服务，包括 Unity 所支持的写作，性能报告，广告，多人服务，认证，云构建，Everyplay 以及多人游戏和 API。

Unity 支持使用 Cg(Microsoft 的高级着色语言修改版本)创建自定义顶点，片段（或像素），向前计算着色器和 Unity 自己的表面着色器。

第二章　LeapMotion 基本原理

2.1 LeapMotion 的传感器结构

从 API 的角度来看，LeapMotion 传感器的结构如下：

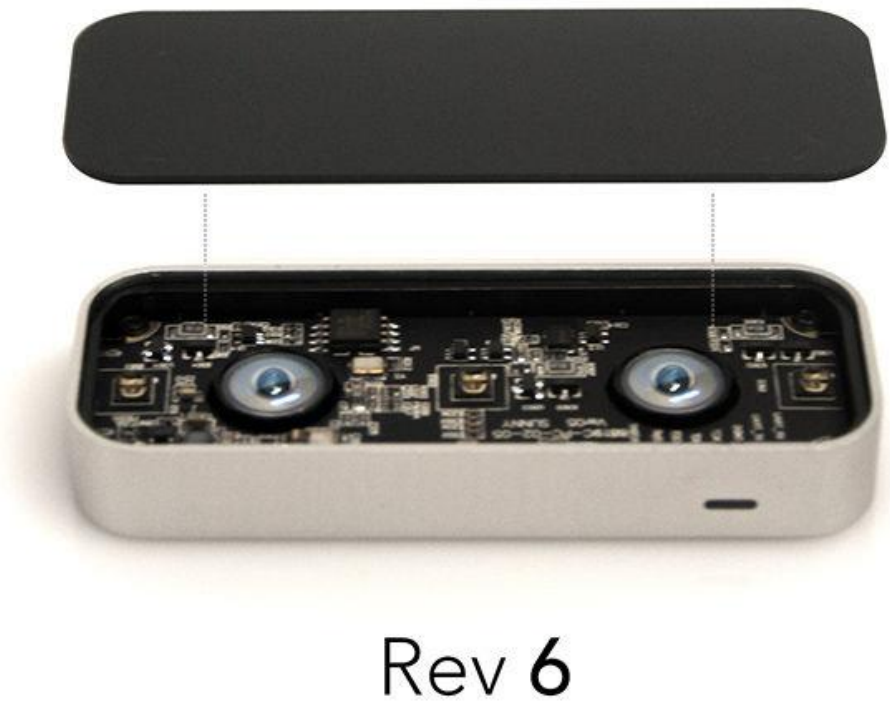


图 2-1-1

通常，Leap Motion 传感器基于内置的两个摄像头从不同角度捕捉前额图像，在三维空间重建手掌的运动信息。Leap Motion 的检测空间大部分为倒四角椎体，检测范围在传感器上方 25mm 至 600mm 之间。

首先，Leap Motion 会建立一个原点在设备中心的直角坐标系，三个轴的方向与平时并无不同，X 轴指向屏幕右方，Y 轴指向垂直地面向上，Z 轴指向垂直屏幕向里，并且他们的单位都为 mm。如图所示：

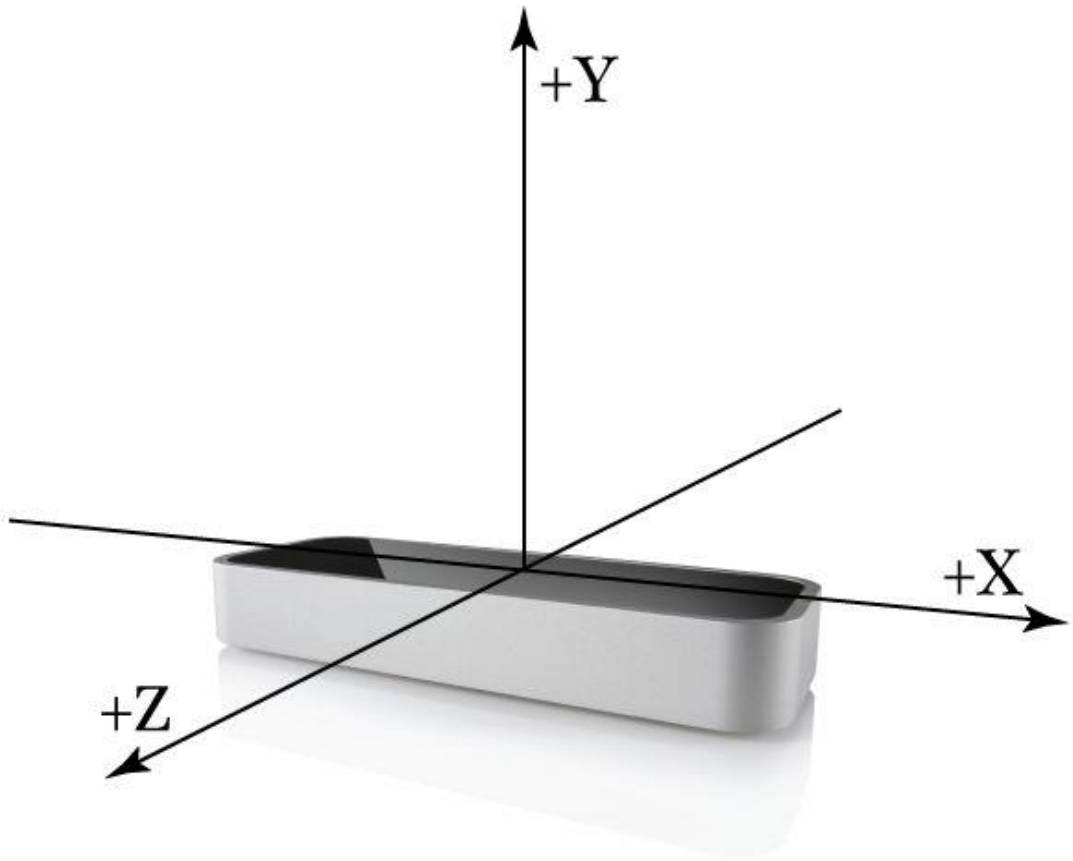


图 2-1-2

在使用期间，Leap Motion 传感器会发送关于手的运动轨迹的信息，并且将这些信息变成帧。每一个这样的帧包含检测到的：

- 所有手掌的列表及信息；
- 所有手指的列表及信息；
- 手持工具（细的，笔直的，比手指长的东西，例如一支笔）的列表及信息
- 所有手指以及和手指一样可以有指向性的工具的列表和信息。

2.2 Leap Motion 的系统架构

Leap Motion 软件作为服务（Windows）或后台进程（Mac 和 Linux）运行。它通过 USB 总线连接到 Leap Motion Controller 设备。手部的运动追踪数据需要启用 Leap 并且连接到 Leap Motion 服务才能获得。该传感器的设备 SDK 提供了两个用于获取跳跃运动数据的 API：本地接口和 WebSocket 接口。本地接口是一个动态库，启用了 Leap 的新应用程序可以被这个接口去创建。而对于 WebSocket 接口而言，Leap 的 Web 应用程序都被支持创建。这些 API 可以让开发者使用几种编程语言创建支持 Leap 的应用程序，包括在浏览器环境中运行的 JavaScript.

2.2.1 本地应用程序接口

本地应用程序接口通过动态加载的库提供。这个库是连接到 Leap Motion 服务，并将跟踪数据提供给开发人员的应用程序。同时，可以直接在 C++和 Objective-C 应用程序中或者通过为 Java，C#和 Python 提供的语言绑定链接到库。

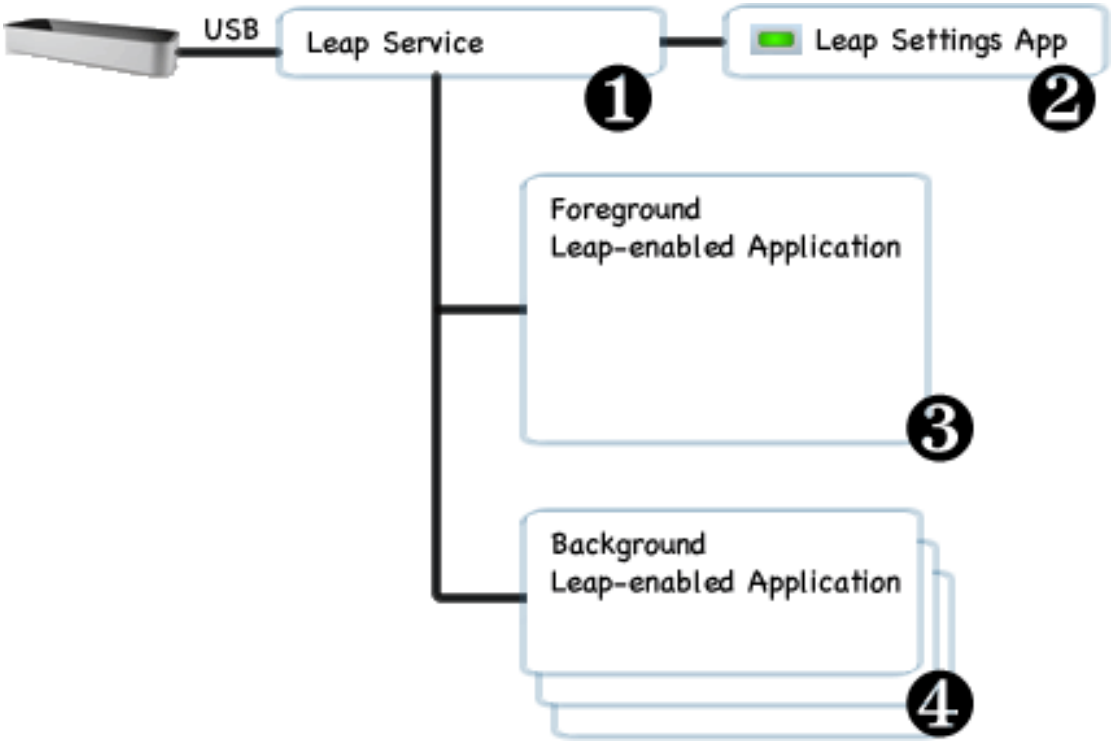


图 2-2-1-1

Leap-enabled applications:

• Leap 的应用程序通过一根专属数据线从设备中获取数据。在电脑中运行的应用程序可以接受这些被处理过后的数据。在大多数的实例中，只有电脑中在运行的服务将会接受到跟踪数据。然而，应用程序可以请求它们在后台接受数据，对于用户来说，可以选择拒绝这些数据。

• Leap Motion 手势控制器在电脑中的程序是单独成块运行的，并不会与其他 Service 产生交集，并且每个开发者即拥有 root 权限的用户都可以自定义它们的安装。Leap Motion 应用程序是 Windows 上的控制面板程序和 Mac OS X 上的菜单栏应用程序。

• Leap 服务可以提供手部运动的跟踪数据供前台的 Leap 应用程序接收。Leap Motion 库的服务与本地库的连接需要被激活的设备应用程序。开发者的应用程序可以直接与 Leap Motion 本地库（C++和 Objective-C）或可用的语言包装库（Java, C#和 Python）之一进行链接。

•当 Leap Motion 的检测区域内没有检测到手部运动信息即失去焦点后，Leap Motion 的服务便不会在发送信息。根据实际情况来看，后台工作的应用程序也可以有接受数据的请求。然而后台服务的配置设置都由前台程序来确定，也可以人工来定义。

2.2.2 WebSocket 接口

Leap Motion 服务在 localhost 域上的端口 6437 上运行 WebSocket 服务。WebSocket 接口以 JSON 消息的形式提供跟踪数据。JavaScript 客户端库可以用于使用 JSON 消息并将跟踪数据显示为常规 JavaScript 对象。

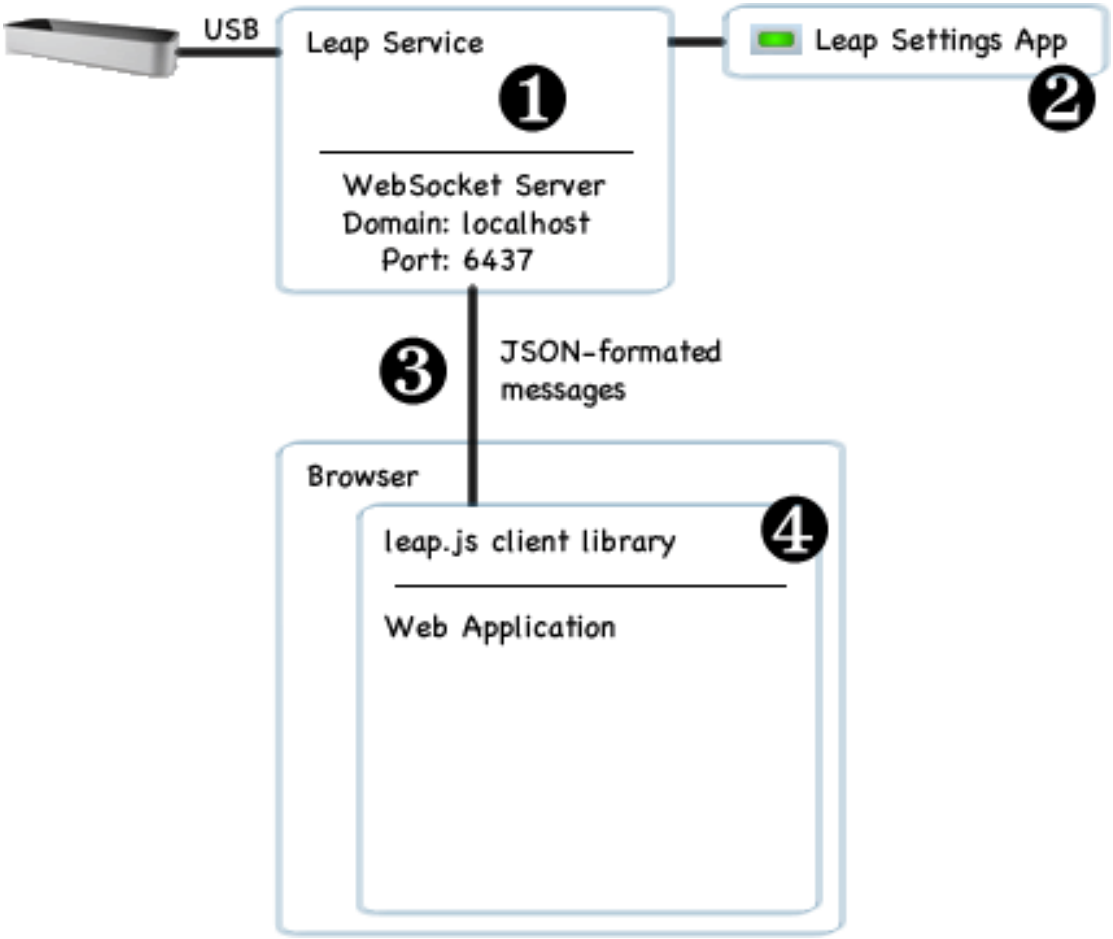


图 2-2-2-1

Leap-enabled web applications:

• 该设备提供的服务器可以被 Leap 传感器的程序所监控

• 可以从控制面板中找到 WebSocket 服务的开关，从而决定是否启用，

• 服务器以 JSOIN 消息的形式发送跟踪数据。应用程序可以将配置消息发送回服务器。

• 应用程序中应该使用 leap.js 客户端的 JavaScript 库。该库建立到服务器的连接并使用 JSON 消息。JavaScript 库提供的 API 在原理和结构方面与本地 API 相似。

2.3 Leap Motion 的数据收集

Leap Motion 收集到的数据在手部位置，手持工具之类都保持在检测范围内时是不会改变的，并且会赋予这些数据一个唯一标志。Leap Motion 里的一些类比如 Frame，可以使用其自带的函数来来获得每个有特别标志的手的运动信息。

运动信息是由 Leap Motion 收集到的帧与帧之间的信息生成的。举例来说，Leap Motion 可以检测到平移，旋转放大和缩放，区别就在于手是否有朝一个方向移动的趋势或者握拳转动以及手部靠近或者分开的动作。所产生的数据包含：

- 旋转的轴向向量
- 旋转的角度（顺时针为正）
- 描述旋转的矩阵
- 缩放因子
- 平移向量

对于每只手，可以检测到如下信息：

- 手掌中心的位置（三维向量，相对于传感器坐标原点，毫米为单位）
- 手掌移动的速度（毫米每秒）
- 手掌的法向量（垂直于手掌平面，从手心向外）
- 根据手掌弯曲的弧度确定虚拟球体的中心；
- 根据手掌弯曲的弧度确定的虚拟球体的半径

其中，手掌的法向量和方向如下图所示：

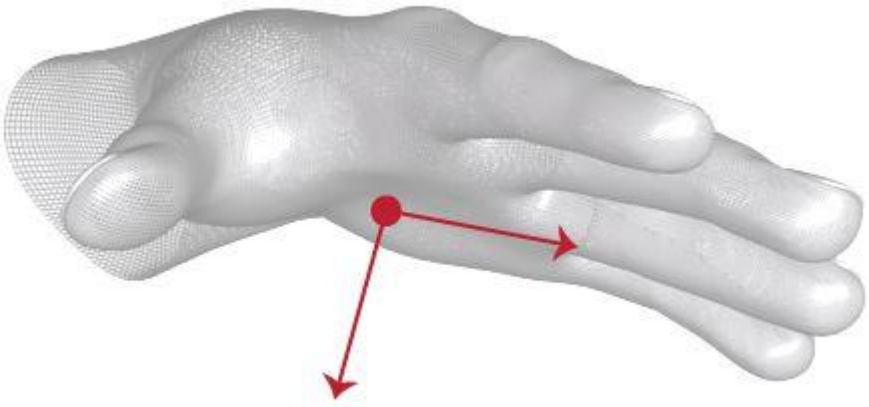


图 2-3-1

[手掌球]的圆心和半径

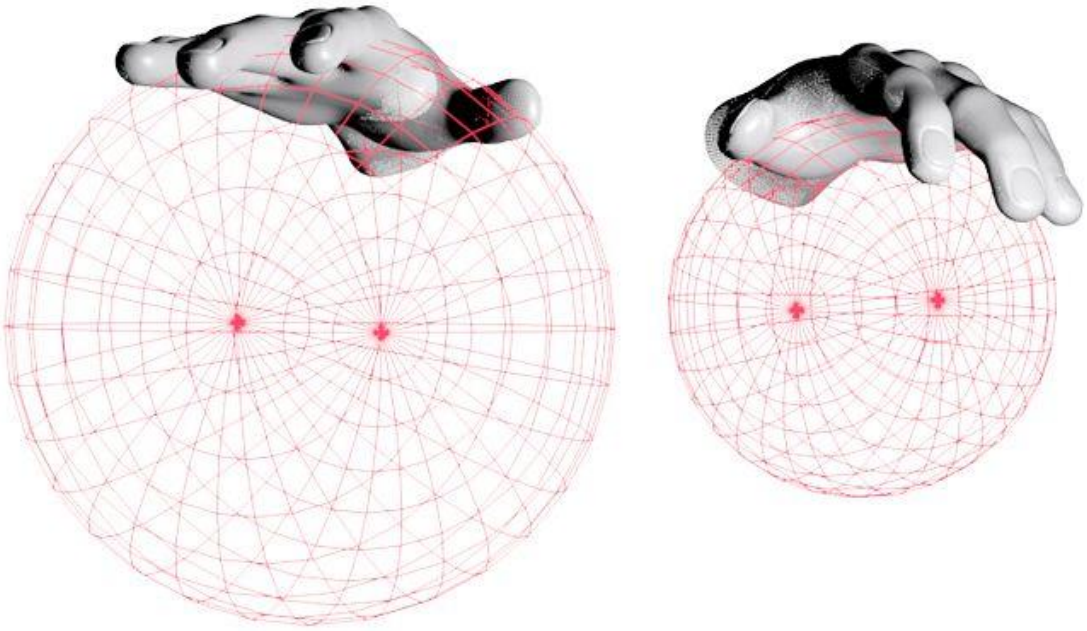


图 2-3-2

手掌的平移、旋转、缩放的信息都可以被检测出来。。检测的数据如全局变换一样，包括：

- 旋转的轴向向量
- 旋转的角度
- 描述旋转的矩阵
- 缩放因子
- 平移向量

除了手指外，LeapMotion 可以检测出手上的工具。比如下图所示：

图 2-3-3

对于手指和工具，会统一地成为可指向对象（Pointable Object），每个 Pointable Object 包含了这些信息：

- 长度
- 宽度
- 方向

- 指尖位置
- 指尖速度

方向和指尖位置如下图

图 2-3-4

用户可以依靠不同的手势被 Leap Motion 识别从而发令。手势和手指，手等其他运动跟踪数据的传回方式一样，每发现一个手势就像帧中自动添加一个手势对象，可以从帧中的手势列表获取手势对象。

Leap Motion 可以识别的运动模式包括：

- circle 画圆，一根手指画一个圆
- swipe 挥扫，手的线性运动
- key tap 击键，敲键盘一样的轻击
- screen tap 触屏，触摸垂直屏幕一样的轻击

Leap Motion 首先识别将其添加到帧中的手势，并且如果这是一个持久动作，则 Leap Motion 将更新的手势对象添加到后续帧。Leap 传感器会在每一次的反馈中更新关于连续动作的信息。相反，敲击动作是不连接的，因此敲击动作只需要一个手势对象。

在实际应用中，可以通过调用 enableGesture()方法启用某个手势的使用。

画圆：你可以使用手指和工具画一个圆。手势开始后，跳跃动作将持续更新过程，直到手势结束或工具离开检测区域或者停止画圆之前，不能被看作是手势的结束。相关 API：CircleGresture

图 2-3-5

挥扫，相关 API：SwipeGesture

图 2-3-6

击键：快速的向下敲击被视为一次击键手势。相关 API：KeyTapGesture

图 2-3-7

触屏：快速向前轻击被视为一次触屏手势，相关 API：ScreenTapGesture

图 2-3-8

开发者可以通过所接受到的全局信息，运动追踪数据以及手和工具的位置信息和变化，来开发 VR 或者自己定义的程序和游戏。

2.4 LeapMotion 的工作原理

2.4.1 LeapMotion 的立体视觉原理

Leap 传感器通过配备双摄像探头，使得其和高精度的人类眼球类似，可以协调空间物体，这里用到了立体视觉原理。为了便于理解，我们可以快速地眨眼睛，并看到物体的位置被移动，这是视差。一旦出现视差，它可以在人脑中产生空间深度感，这是 3D 电影的基本原理。当拍摄电影时，照片被拍摄到左眼和右眼的不同视角，因此产生视觉上的差异，从而在观看平面时会产生三维幻觉。当然，视差不是随意定义的。它必须匹配瞳孔之间的距离，这被称为基线长度。不同年龄、性别和种族的人的基线长度略有不同，因此 3D 电影不一定适合所有群

体。

这种应用于立体视觉的测量方法称为三角测量。三角剖分是确定目标空间位置最常用、最基本的方法。对于普通的 Kinect、激光反求等应用场合，对飞机装配的高精度运行环境等方面的应用较少。Leap 传感器能进行设备校准的前提是，两个摄像探头的基线与距离固定。校准之后的控制器可以精确地计算摄像机目标的空间坐标。

2.4.2 LeapMotion 控制器的工作过程

以上是 Leap Motion 的基本工作原理。其次，介绍了控制器的工作过程。当到达控制器的工作区域时，两个摄像机需要同时捕获目标，并实时计算目标的视差，从而获得空间信息。这里的目标是被过滤的目标信息，如指尖和手掌。在这种情况下，由控制器进行三维扫描的方法是不可能的。控制器的工作区域必须是双摄像机的公共场区域，所以飞跃运动控制器不能被太复杂的多点姿态操作所识别。

2.4.3 LeapMotion 的照明部分与计算效率

为了更容易地识别目标，控制器上的 LED 灯需要照亮目标，增强目标和背景之间的亮度对比，使得识别更容易，并且使设备处于黑暗的环境中。相反，如果在室外有太阳或者红外光比较充足的地方，则会影响控制器的正常使用。

跳跃运动主要采用 TBD 技术在算法中实现。该技术的优点之一是可以精确跟踪目标，但缺点是对存储器有一定的压力，并且还需要一定量的计算，并且还使用具有高帧速率的双摄像机。从官方文献中可以看出，未来的 Leap 将通过优化数学模型来减少 CPU 占用资源。

第三章　LeapMotion 手势控制的建立

3.1 LeapMotion 手势在 Unity 中的建立

3.1.1 Leap Motion 资源以及插件

Leap Motion 的 Unity 资源包提供了一个简单的方式将运动控制的手模型加入到 Unity 游戏去。

图 3-1-1-1

系统要求：

- Leap Motion 版本 2.3.1+
- Unity 5.1+
- Windows 7，Windows 8，Mac OS X

3.1.1.1 安装

- 从 Leap Motion 的官方网站上下载最新的资源包。
- 打开或者创建一个项目。
- 选择 Unity 中的 Assets>Import Package>Custom Package 菜单命令。
- 选中下载的资源包并且打开，之后便可以导入资源包。

3.1.1.2 故障排除

如果在加载 HandController 预制体到场景中并运行后没有看到手模型出现，可以检查一下原因：

- 手在场景摄像机的事业内，并且不被另一个 3D 对象所遮挡，同时足够大以便可见。3D 指针显示在 HandController 预制体的位置上方。
- 暂停游戏并检查手的模型是否在 Unity 编辑器的 Hierachy 结构视图中。如果在的话，在编辑场景视图中找到它们。

• 确保 Unity Editor（或应用程序，如果在编辑器外部运行）具有 OS 的输入焦点。（Leap 的服务仅将数据发送到被激活的应用程序）。

• 检查任务栏中的跳跃动作图标是否变成绿色。如果图标颜色变暗，跳跃运动的服务将不发送数据。在这种情况下，仔细检查是否已经插入了跳跃运动硬件并检查服务是否正在运行。

• 从任务栏图标菜单中打开 Leap Motion Visualizer。如果在 Visualizer 中看到了问题报告，那么问题很可能出现在 Unity 的插件，脚本或者开发者自己的应用程序里。如果 Visualizer 中没有手，那问题是在 Unity 引擎之外的。

3.1.2 Unity 插件概述

Leap Motion 控制器跟踪手部和手指，并报告位置，速度和方向，同时保持低延迟和高精度。该控制器可用于桌面或安装在 VR 耳机上。

上述的 Leap 传感器系统包括硬件设备及其在主机上运行或者再后台上运行的软件组件。软件组件对硬件生成的图像进行分析，并将跟踪信息发送给应用程序。在 Unity 中，Leap Motion 可以连接到此服务获得数据，同时可以通过脚本，使得 Leap Motion 的坐标转换为 Unity 游戏引擎的坐标。这些脚本和附加图形资源可以轻松地讲 3D 动作控制的手部添加到 Unity 场景中。

3.1.2.1 坐标系统

Unity3D 使用左手坐标为笛卡尔坐标系，而 Leap Motion API 使用右手坐标作为笛卡尔坐标系。（本质上，Z 轴指向相反的方向。）Unity 也使用默认的仪表单元，而 Leap 传感器使用的毫米。在 Unity 的内部，会自动将坐标转换为其游戏引擎常用的数据并且缩放单位。

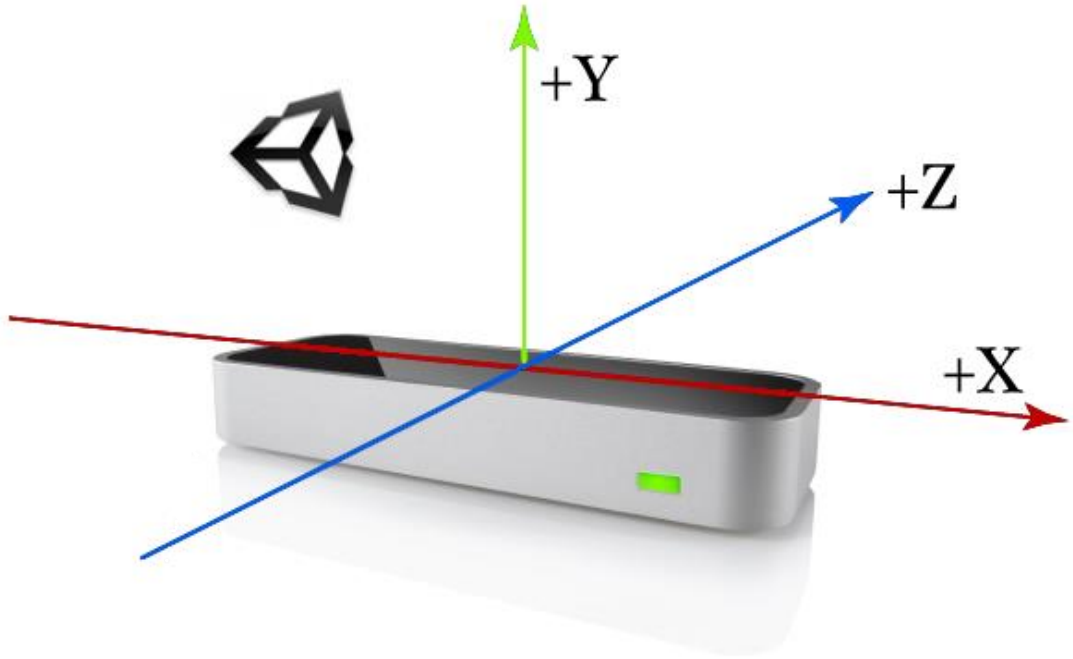


图 3-1-2-1-1

桌面方向上，Unity 左手坐标系统叠加在 Leap Motion 设备上。

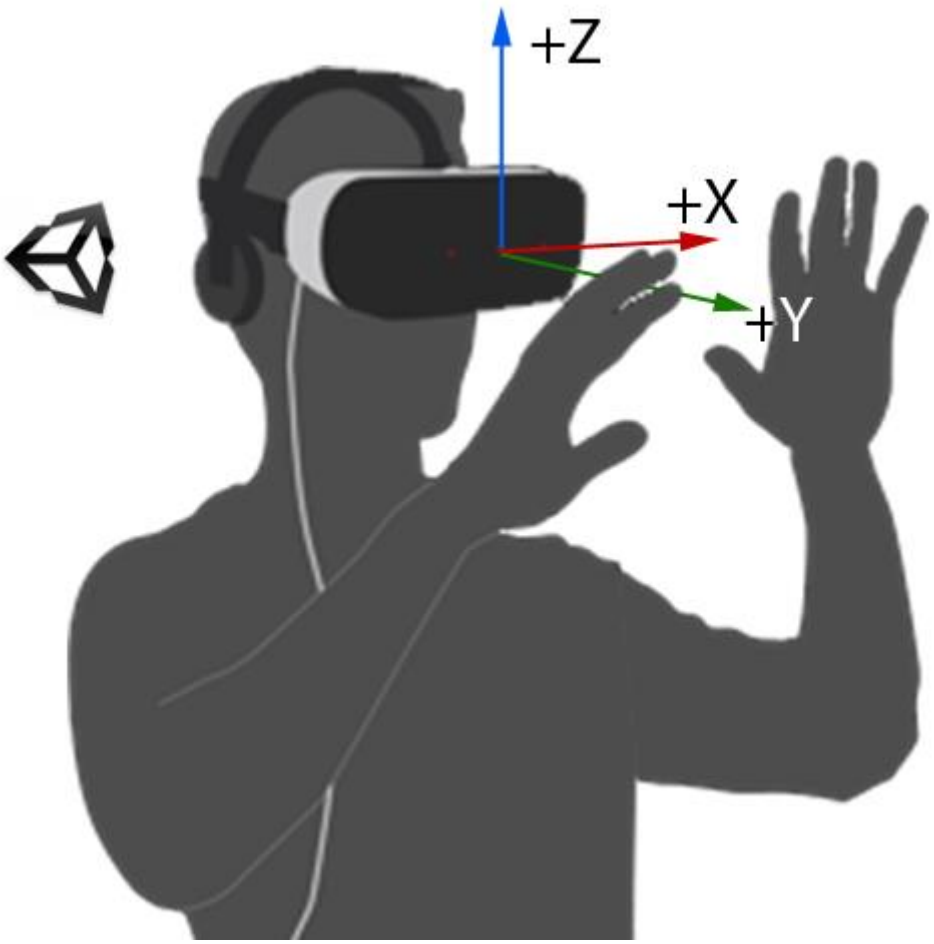


图 3-1-2-1-2

Unity 坐标系统叠加在 HMD 方向上的 Leap Motion 装置上。

直接从 Leap Motion 传感器中获取的运动信息类，并不是 Unity 坐标系，需要通过脚本转化为 Leap 坐标系。开发者可以直接使用 LeapUnityExtensions 文件中的程序功能将 Leap Motion 左边转换为 Unity 坐标。ToUnity（）将轴从右手转换为左手并返回 Unity Vector3 类型的对象。

ToUnityScaled（）也可以将坐标从 milimeters 转换 meters。ToUnity（）通常与方向向量一起使用。

3.1.2.2 手部追踪

Leap Motion 的设备中用到了红外灯和某些光学的传感器。这些传感器具有约 150° 的视野。Leap Motion 控制器的有效范围从设备上方约 0.03 至 0.6 米。

图 3-1-2-2-1
Leap Motion 控制器的桌面模式

当控制器清晰，高对比度地查看物体轮廓时，检测和跟踪效果最佳。为了能够比较轻松地面对具有难度的运动信息收集情况，Leap 中应用程序会结合检测到的手模型与收集到的运动信息。

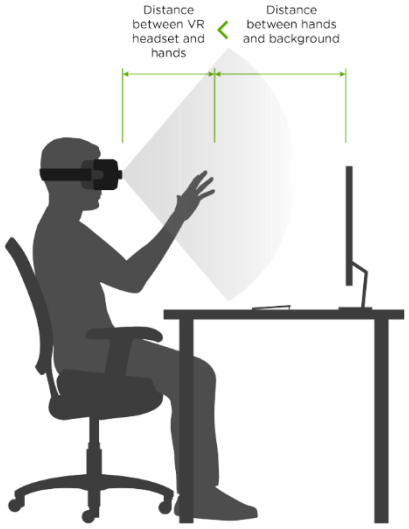


图 3-1-2-2-2
特别是在 HMD 模式下，应该确保传感器与手之间的距离更小

3.1.3 VR 设置

Leap Motion Core Unity Assets 提供的 LMHeadMountedRig 预制件，可以为虚拟场景和增强现实场景提供完整的相机和手持控制器设置。这种预制可以实现简单的配置，可以使用 Leap Motion 相机的图像或 3D 模型来呈现手部模型。

LMHeadMountedRig 中的脚本自动将立体摄像机位置调整到正确的瞳距，并自动补偿 AR 场景中的视频滞后。

要在 Unity 中设置虚拟或者增强现实场景：

- 在 Unity 中打开或创建一个新项目。
- 启用 Unity 虚拟现实支持。
    - 使用 Edit>Project Settings>Player 菜单命令现实播放器设置。
    - 在其他设置类别下，选中虚拟现实支持复选框。

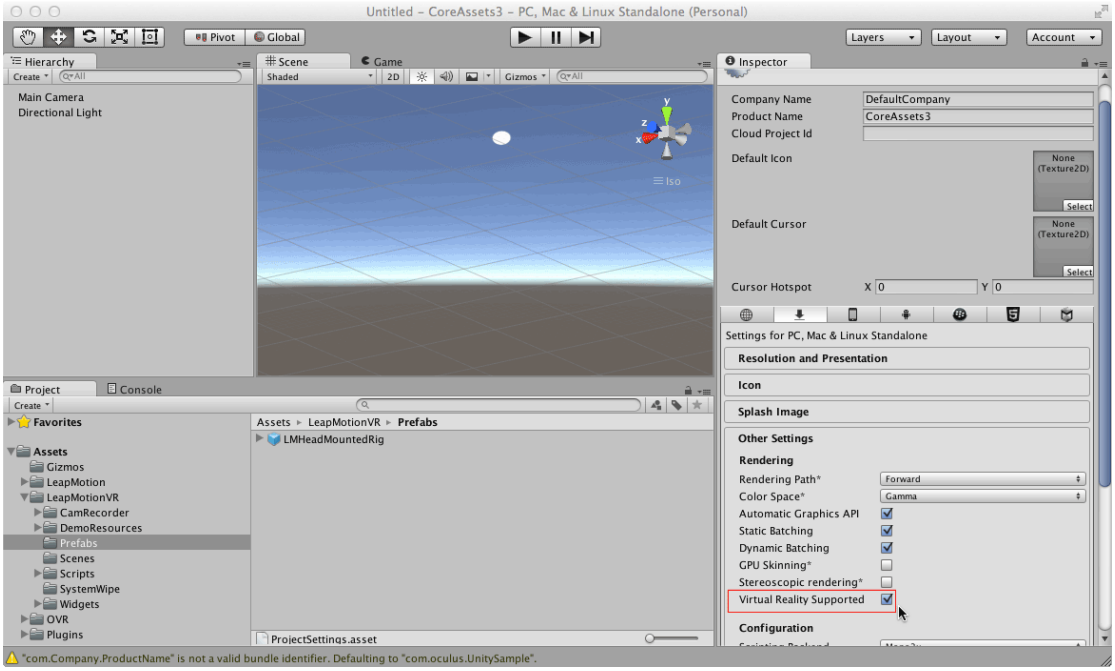


图 3-1-3-1

- 打开或创建一个新场景
- 删除现有的不需要的相机
- 从 Assets/LeapMotionVR/Prefabs 文件夹中将 LMHeadMountedRig 预制件拖到场景中。

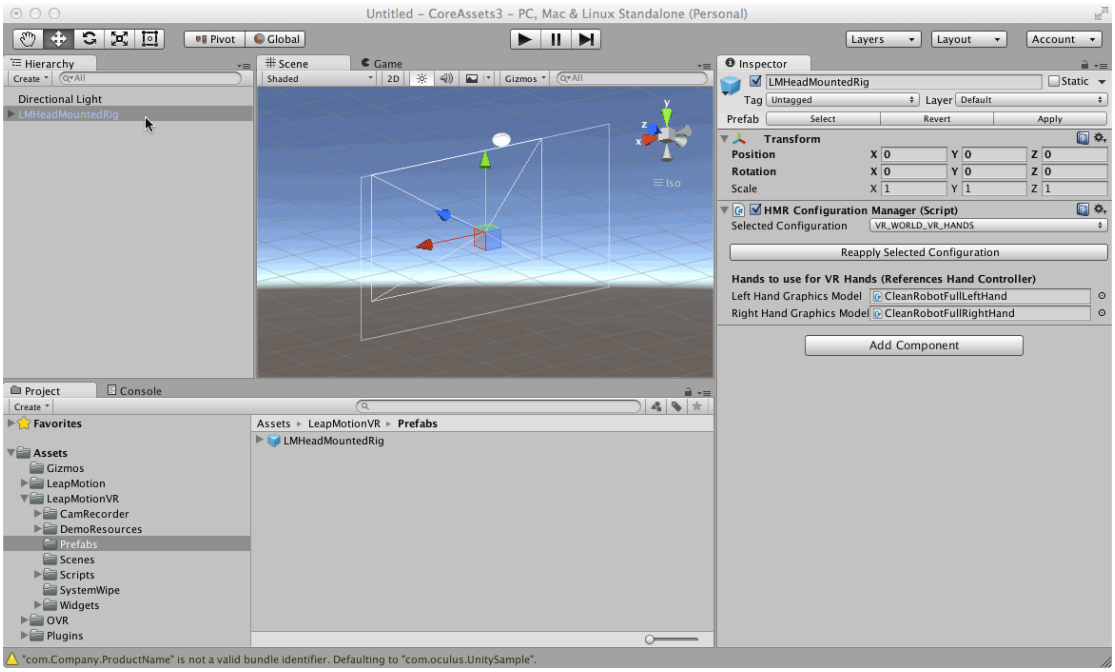


图 3-1-3-2

- 在场景中设置摄像机的方向和位置。
- 根据配置选项，可以选择的有：
  - VR World VR hands 不透明 3D 世界中的 3D 模型手
  - VR World AR hands 带有相机图像指针的不透明的 3D 世界
  - AR World AR hands 相机图像被用作 3D 世界和手部的背景

- 如果使用 VR 指针选项，可以从 Assets/LeapMotion/Prefabs 文件夹中指定所需的预制体
- 直接在 HandController 组件上设置手工预制件以及物理手工预制件，它们深深嵌套在 LMHeadMountedRig 对象内。更改所选配置将重置所有手型号，同样会点击“重新应用所选配置”按钮。
- 按播放按钮进行测试。

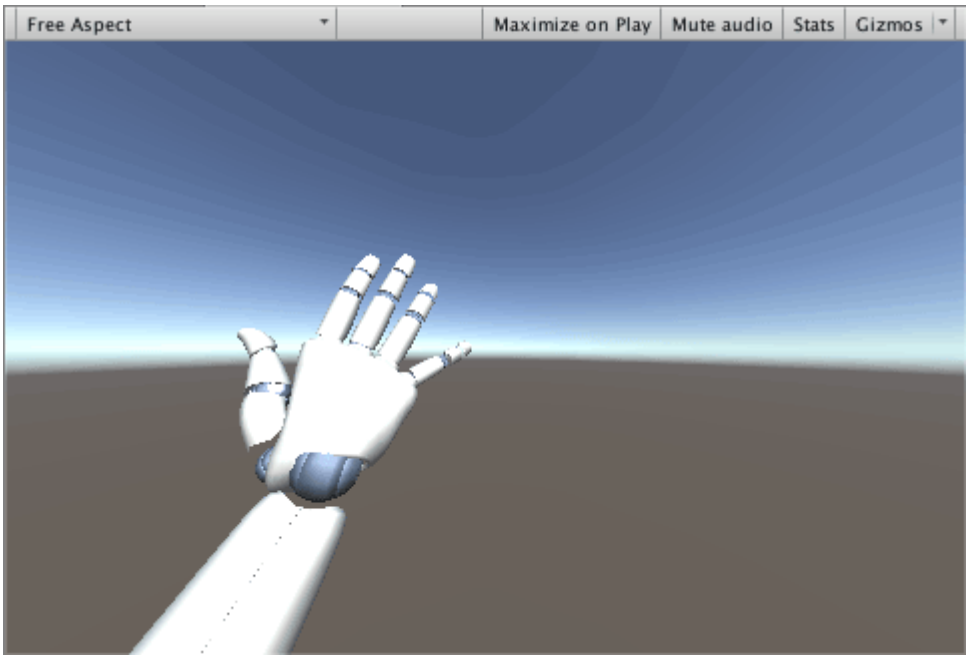


图 3-1-3-3

- 保存场景。

之后就可以按照所希望地向场景里添加其他内容了。

3.1.4 手的资源包

用于 Unity 的 Leap Motion 核心资源包包括大量预制体。开发者可以按照原样使用这些预制体，修改它们或者创建自己的预制件。

有几种不同的创建手的方法，包括为手部分创建独立组件，并分别移动这些部件，创建用骨架操纵的网格手，并通过旋转骨骼是网格变形，以及创建由代码驱动的手部模型后，在其上创建自己的图形。现有的预制件使用这三种方法。

HandController 类协调跟踪数据到手和手指的采集和应用。HandModel 和 FingerModel 类用作手和手指动画的基类。有几个扩展类比如 HandModel 和

FingerModel 类来实现特定类型的动画。这些特定的类，如 SkeletalHand/Finger 和 RiggedHand/Finger，可用于多个手部设计，只要他们所连接的对象遵循相同的基本结构即可。

图形和物理手分成两个不同的预制件。开发者可以使用任何物理手开发成最适合的手模型。例如，RigidFullHand 包括一个刚体和手臂的碰撞体，因此可以将它与包含手臂图形的模型一起使用，当手臂会与场景中其他物体发生碰撞时。

以下是几种手部模型的展示图：

• Image Hands；ImageFullHand；RiggedHand/Finger

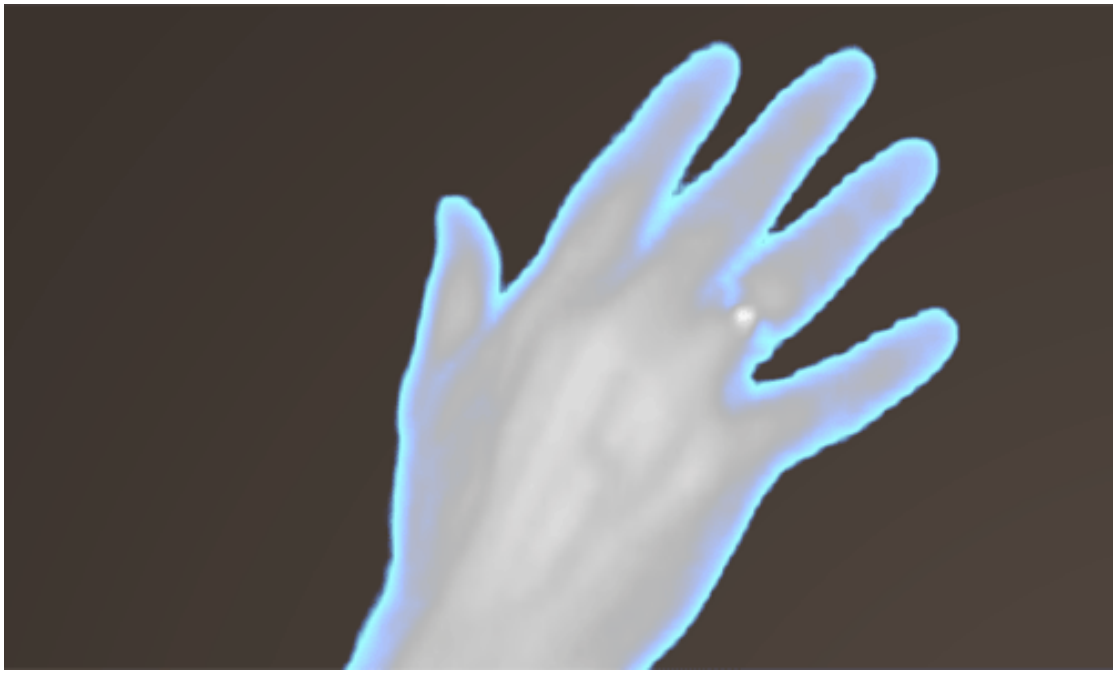


图 3-1-4-1

• Procedural Hands；PolyHand1/2/3，DebugHand；PolyHand/Finger，DebugHand/Finger

图 3-1-4-2

• Component Hands: MinimalHand, CleanRobotHand, GlowRobotHand; SkeletalHand/Finger

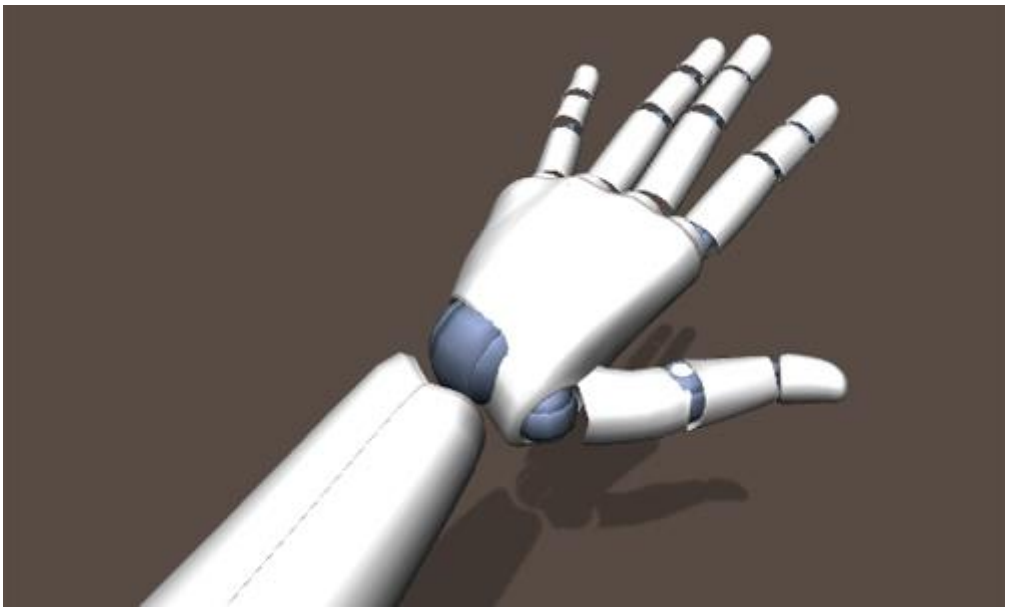


图 3-1-4-3

• Rigged Hands: All the human models; RiggedHand/Finger

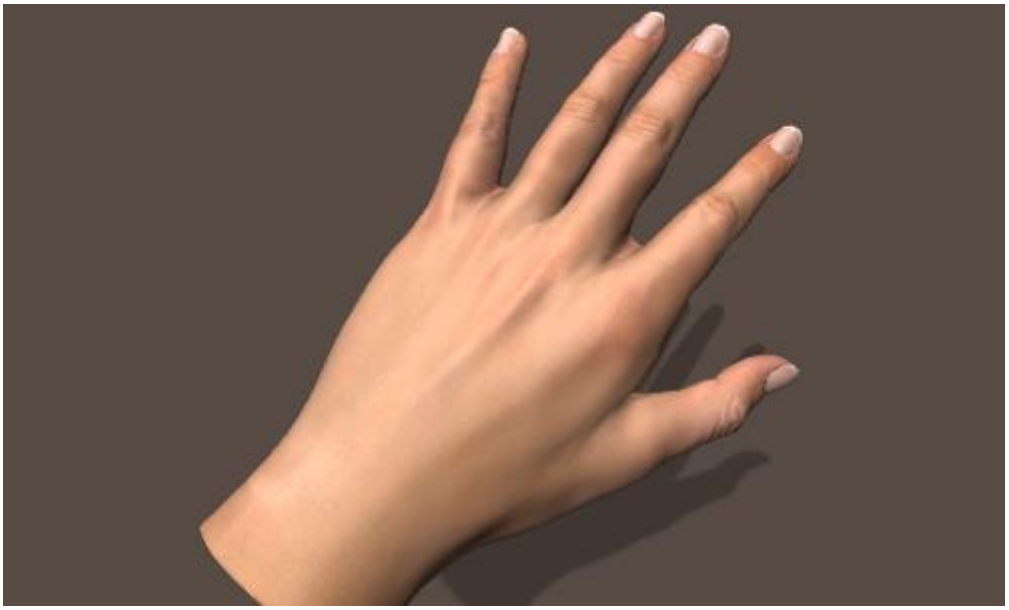


图 3-1-4-4

• Physics Hands: RigidHand, RigidFullHand, RigidRoundHand, ThickRigidHand; RigidHand/Finger

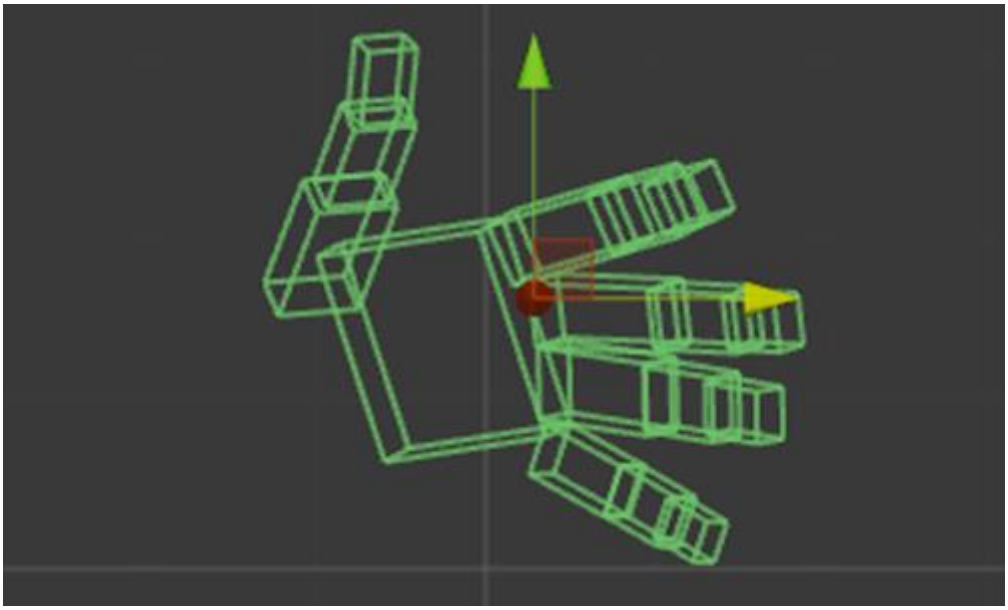


图 3-1-4-5

将 HandController 添加到场景后，可以通过将所需要的手工预制件拖动到 Hand Graphics Model 插槽来设置图形模型。

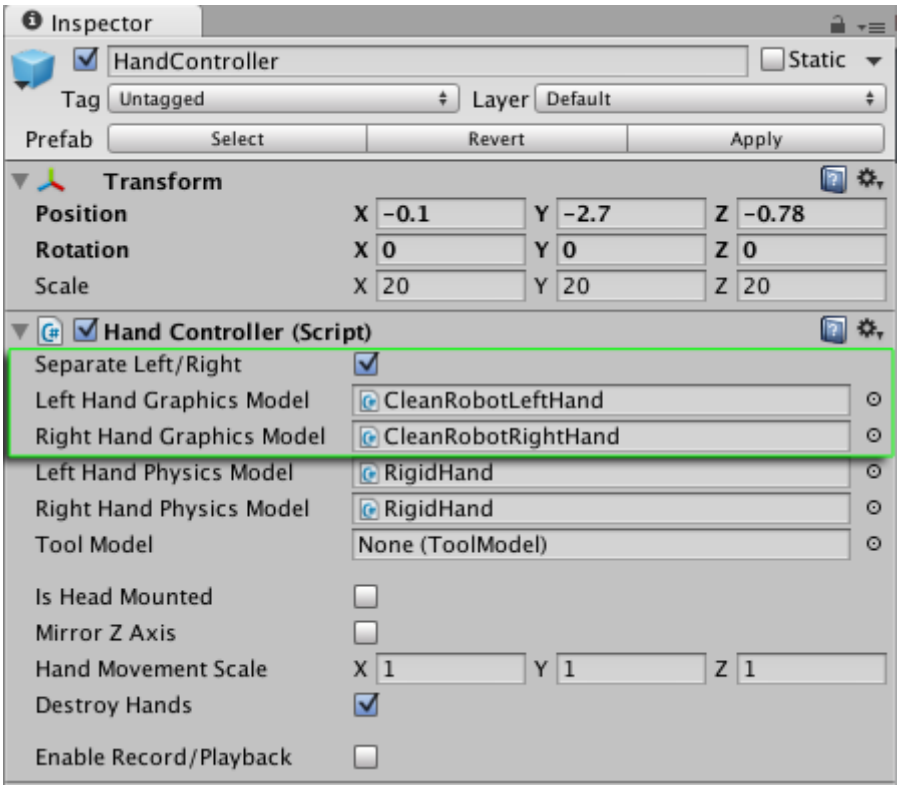


图 3-1-4-6

如果选中了分离的左/右设置，则可以为左右手使用不同的预制件。如果没有选中此选项，HandController 使得双手为同一个镜像模型。现有的预制件在所有标有“右”或“左”的地方都有标注（如果使用错误的手型，错误将会很明显）。

想改变手模型的大小，需要修改 HandController 预制体的 Transform Scale 属性。Scale 值为 1 的话表示为真实大小，但是通常太小，这取决于不同的场景。例如，默认的 Unity 立方体边长 1 米，这样会使手的尺寸看起来很小。将手变得更大也会增加它们的移动范围。通过增加 HandController 手部移动比例值，可以进一步增加动作范围而不会增加手部的大小。

在 VR 和 AR 的场景中，情况会有所不同。在实际情况下应该使用 1:1 比例。由于 Leap Motion 跟踪数据已经使用了现实世界的单位，因此将 HandController 缩放比例设置为 1，并将 HandController 对象放置在相对于相机相同的位置，因为实际上设备是处于两眼之间的相对位置的。对于 VR/AR 应用程序，应该使用 LeapMotionVR 文件夹中的 LMHeadMountedRig 预制体替代 HandController 预制体，并且将这些物体进行自动调整。

设置 Head Mounted 将会提供 Leap Motion 服务的提示，即从 Head Mounted 位置查看手部模型并提高手部识别度。该设置不会自动旋转游戏界面中的手以显示手在 HandController 的前方而不是上方。为此，可以将 HandController 旋转至设置为{x=270，y=180，z=0}。

3.1.5 工具介绍（Widgets）

Leap Motion 核心组件包括的 UI 控件包含以下几种：

- DemoDial – a scrolling list

图 3-1-5-1

• DemoScrollText – a scrolling text box

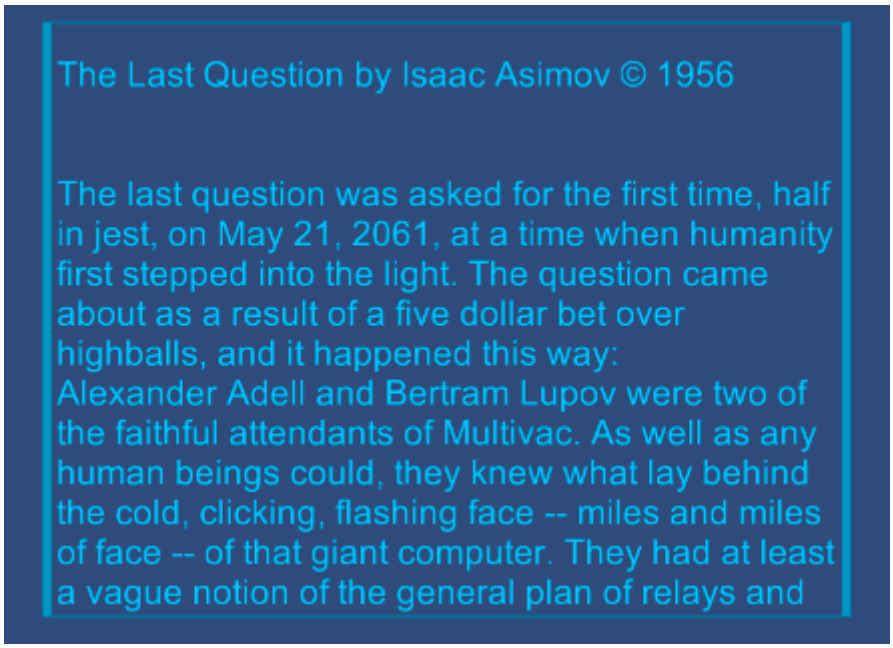


图 3-1-5-2

• DemoSlider – a slider control

图 3-1-5-3

• DemoToggkeButton – a two-state button

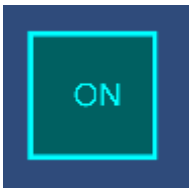


图 3-1-5-4

以上这些小部件的预制件都可以在 Core Assets 中的 LeapMotion/Widgets 中的 Prefabs 文件夹中找到。

3.1.5.1 添加 Widget 到场景

要将小部件（Widget）添加到场景中，只需要将所需的预制件拖动到场景视图中，并根据需要来设置位置和缩放比例。放置和调整窗口小部件的主要考虑因素是，可以使用 Leap Motion 生成的手轻松地达到 UI 的功能部分，并且多个控件足够大并且间隔也足够大，可以方便地使用一个窗口小部件而不同意外触及另

一个窗口。

需要注意的一个技术问题是，一些小部件在内部使用 Unity Canvas 对象。如果将这个小部件放置在另一个画布内，则可能必须手动使用缩放，因为 Unity 仅应用分配给最外面 Canvas 对象的缩放。

有两个场景演示 Core Assets 中包含的小部件。LeapMotion/Widgets 中的场景展示了非 VR 布局。LeapMotion OVR/Widgets/Scenes 中的场景展示了以 VR 头盔 Oculus Rift 的 VR 布局

3.1.5.2 与对象属性挂钩的 Widget

其中 3 个小部件，DemoDial，DemoSlider 和 DemoToggleButton 分别表示的值为：拨号菜单选择，滑块数值以及按钮的切换状态。Widget 的库定义了一个抽象的 DataBinder 类，必须实现该类才能将小部件挂接到场景中对象的属性。

DataBinder 类是一个通用类型；为每个小部件定义一个特定的类的类型。这些在 DataBinder.cs 文件的末尾定义：

```
public abstract class DataBinderSlider : DataBinder<SliderBase, float> {};
public abstract class DataBinderToggle : DataBinder<ButtonToggleBase, bool> {};
public abstract class DataBinderDial : DataBinder<DialGraphics, string> {};
```

图 3-1-5-2-1

在具体的类中，实现了两个函数：一个函数返回的值作为 Widget 的数据模型的属性；另一个更新该属性的值，并且在更改绑定空间的值时调用。例如，DataBinderSlider 的实现会如下所示：

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;
using LMWidgets;

public class SliderDataModel : DataBinderSlider {
    [SerializeField]
    float sliderValue = 0; //Data model

    override protected void setDataModel(float value) {
        sliderValue = value;
    }

    override public float GetCurrentData() {
        return sliderValue;
    }
}
```

图 3-1-5-2-2

本示例定义了数据模型属性本身（sliderValue），但其实可以使用某个场景的现场属性。事实上，数据模型根本不需要简单的属性。它只需要映射到小部件

的数据类型。数据绑定脚本可以附加到任何更方便的对象，例如具有数据模型的属性的对象。

定义数据绑定类之后并将其添加对象后，必须将该小部件拖动到 Unity 的 Inspector 中对象的小部件属性。

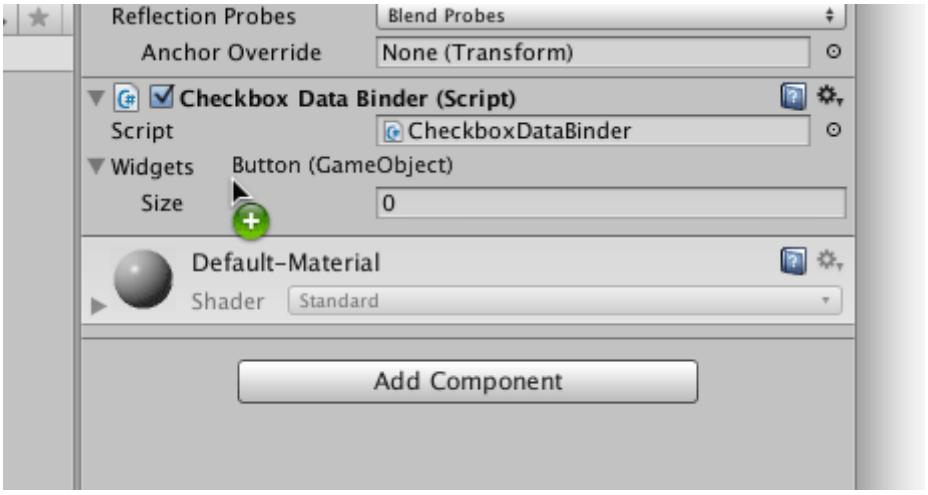


图 3-1-5-2-3

3.1.5.3 与 Events 挂钩

如果对 Widgets 的交互事件有兴趣可以添加事件处理程序。Widget 的库定义了以下句柄：

- StartHandler – buttons, sliders and dials
- EndHandler – buttons, sliders and dials
- ChangeHandler – sliders and dials

要添加事件处理程序，需要使用正确的签名定义事件处理程序函数，并为该 Widget 的事件句柄添加函数引用。事件处理程序的签名如下所示：

```
void FunctionName(object sender, LMWidgets.EventArg<T> arg)
```

图 3-1-5-3-1

其中 T 是与小部件关联的数据的类型。T 对于切换按钮将是 bool 值变量，对于滑块是浮动类型变量，对于拨号是整形变量。

例如，可以定义并注册一个函数来处理滑块小部件的更改时间：

```
public SliderDemo slider = null; //Set in inspector

void Start(){
    slider.ChangeHandler += OnSliderChange;
}

private void OnSliderChange(object sender, LMWidgets.EventArg<float> eventData){
    Debug.Log ("Slider changed to: " + eventData.CurrentValue);
}
```

图 3-1-5-3-2

3.1.6 C# API 的引用

在这一部分介绍一下组成 Leap Motion API 的几种基本类的信息。

这些类中显示的任何坐标，方向和转换都是相对于 Leap Motion 坐标系而非 Unity 游戏世界的标准来的。要将位置向量转换为 Unity 坐标，需要使用 Vector 类中的扩展函数 ToUnityScaled（）。要将方向向量转换为 Unity 坐标，需要使用 ToUnity（）。这些脚本在 Unity Core Assets 中的 LeapUnityExtensions 中包含的 C#脚本中定义。

同时，可以将表示为矩阵的变换从 Leap Motion API 转换为具有 Matrix 类扩展的 Rotation（）的 Unity 四元数旋转和使用 Translation（）的 Vector3 转换。

从 Leap Motion 中手的脚本获得矢量时，这些矢量相对于 HandController 对象已经被转换为 Unity 世界空间。换言之，如果在场景中，旋转控制器的对象，手的位置也会旋转。在条用 ToUnityScaled（）后，可以使用 Unity 中的 Transform.TransformPoint（）函数以相同的方式将点从 Leap Motion 坐标转换为 Unity 世界坐标。

```
Leap.Vector position = finger.TipPosition;
Vector3 unityPosition = position.ToUnityScaled(false);
Vector3 worldPosition = handController.transform.TransformPoint(unityPosition);
```

图 3-1-6-1

开发者也用 TransformDirection（）完成相同的功能。

```
Leap.Vector direction = finger.Direction;
Vector3 unityDirection = direction.ToUnity(false)
Vector3 worldDirection = handController.transform.TransformDirection(unityDirecti
on);
```

图 3-1-6-2

要在场景中转换相对于控制器对象的旋转，需要将控制器的旋转值与对象的旋转值相乘。

```
Quaternion leapRotation = arm.Basis.Rotation(false);
Quaternion finalRotation = controller.transform.rotation * leapRotation;
```

图 3-1-6-3

3.1.7 SDK 库

Leap Motion 库是用 C++编写的。Leap Motion 还使用 SWIG（一种开源工具）为 C#，Java，和 Python 生成语言绑定。SWIG 生成的绑定将使绑定的编程语言编写的调用转换为基本 C++ Leap Motion 库中的调用。 每个 SWIG 绑定使用两个额外的库。Leap 传感器中提供了 WebSocket 服务器和一个 JavaScript 的库来帮忙实现 JS 和 WEB 的开发。

除了 Leap.js 客户端 JavaScript 库外，Leap Motion SDK 还包含了开发支持 Leap 的应用程序和插件所需的所有库，代码和头文件。如果想使用这些文件，需要从 Leap Motion Developer Portal 下载 Leap Motion SDK。每个支持的操作系统由一个 SDK 包。

Unity5 和虚幻引擎 4.9 的插件与主 SDK 分开提供。Unreal 插件包含在虚幻引擎 4.9+。

完整的 Unity 资源包是单独提供的。SDK 中不包含手型，Unity 特定脚本和演示场景。通常情况下，下载和导入整个资源包比较容易，但是插件文件也包含在 SDK lib/UnityAssets 文件夹中。从 Unity5 开始，Pro 和 Personal 都支持插件。

通常在 Unity 中，脚本语言使用的是 C#。C#类定义是为独立库中的.NET 3.5 和.NET 4.0 提供的，使用代码可以引用 LeapCSharp.NET3.5.dll 或者 LeapCSharp.NET4.0.dll（在所有支持的操作系统上使用相同的库名称）。这些库加载 libCSharp.dylib（Mac），LeapCsharp.dll（Windows）或 libLeapCSharp.so（Linux）。中间库加载 libLeap.dylib, Leap.dll 或 libLeap.so。

3.1.8 创建项目

首先，以通常的方式创建一个新的 Unity 项目，即

- 打开 Unity 编辑器
- 选择 File>New Project
- 选择一个名称并保存位置
- 点击保存项目

然后将 Leap Motion 资源包导入项目中。

接下来下载资源包，在下载完成之后，选择 Unity 中的 Asset>Import Package>Custom Package menu command.找到下载的资源包，然后点击打开，资源包将被导入到项目中。在现有的 Unity 项目中有了之前 Leap 文件资源的情况下，通常会先去除之前的文件包。Unity 导入过程指挥添加新文件并覆盖已修改的文件，它并不会删除过时的文件。

3.1.9 添加预制体

将手持控制器预制体放置与希望出现的点的下方。相对于真正的 Leap Motion 谁被，场景中的双手位于与真实双手相同的位置。并且可以通过更改 HandController 对象的缩放比例来更改手的大小，并更改“手部移动比例”设置来允许双手在较大的音量范围内移动。

通过将不同的预制件拖动到左/右图形模型插槽来更改手部图形。

接下来介绍为场景添加手势的基本步骤：

- 在 Project 面板中，找到 Assets/LeapMotionPrefabs 中的 HandController 预制件
- 将 HandController 预制件拖到场景视图中
- 将预制件位置设置到所需要的坐标
- 为了看到手，HandController 必须位于摄像机的视野内
- 设置手部比例尺以适当的数值，将手放入场景中，1 的比例意味着双手呈现其实际尺寸
- 将 HandController 的左右手图形模型属性设置为 HandModerlsHuman 或

HandModelsNonHuman 预制文件夹中所需要的预制体。（一般来说不需要从 RigidHand 中更改物理模型）

• 将所有物体调试完毕后即可开始测试手的模型

当现实中的双手放在 Leap Motion 设备上时，可以看到双手出现在 Hierarchy 面板以及 Game 视图中。如果在 Hierarchy 中看不到手，可以在 Leap Motion 控制面板中打开展示台，确保 Leap Motion 设备已经连接并且发送跟踪数据。在 Unity 中可以暂停游戏界面并使用 Scene 视图来查看具体细节。

3.1.10 访问 Leap Motion 的 API

除了资源包中包含的预制体和脚本以外，开发者还可以编写自己的脚本来访问 Leap Motion API 来追踪数据。Leap Motion 类在 Leap 命名空间中定义。访问 Leap Motion API 的基本 MonoBehavior 类将如下所示：

```
using UnityEngine;
using System.Collections;
using Leap;

public class LeapBehavior : MonoBehaviour {
    Controller controller;

    void Start ()
    {
        controller = new Controller();
    }

    void Update ()
    {
        Frame frame = controller.Frame();
        // do something with the tracking data in the frame...
    }
}
```

图 3-1-10-1

需要注意的是，上图的示例不使用 Leap.Listener 子类。由于 Unity 应用程序具有自然的帧频，因此当 Unity 引擎调用函数时，Update（）函数可以从 Leap Motion 控制器获取当前数据帧。

Assets/LeapMotion/Scripts/Utils 中的脚本 LeapUnityExtensions,cs 提供了将向量和矩阵从 Leap Motion API 类转化为 Unity API 类以及转换比例和坐标系的功能。

3.2 LeapMotion 手势的形成

3.2.1 手固定不动

```
protected bool isStationary (Hand hand)// 固定不动的
{
    return hand.PalmVelocity.Magnitude < smallestVelocity;
}
[Tooltip ("Velocity (m/s) move toward ")]                    //速度（m/s）走向
protected float smallestVelocity = 0.4f;
```

图 3-2-1-1

上图为判定 Leap Motion 手固定不动的代码。图中 PalmVelocity 代表的是手部模型在 X 轴上移动的速度，算法如下，设定一个最小的 X 轴移动速度 smallestVelocity，当手在水平面上的移动速度小于这个值时，认为手此时处于静止不动的情况，并返回一个 bool 值。

3.2.2 手划向左边

```
protected bool isMoveLeft (Hand hand)   // 手划向左边
{
   //x轴移动的速度   deltaVelocity = 0.7f    isStationary (hand)  判断hand是否禁止
    return hand.PalmVelocity.x < -deltaVelocity && !isStationary (hand);
}

[Tooltip ("Velocity (m/s) move toward ")]
protected float deltaVelocity = 0.7f;
```

图 3-2-2-1

上图为判定 Leap Motion 手向左划动的代码，算法如下，如果 Hand 在 X 轴上移动速度小于-deltaVelocity 的值（向左为 X 轴负方向），并且同时 isStationary 函数的返回值为 false，则视为手在向左划，并返回一个 bool 值。

3.2.3 手划向右边

```
protected bool isMoveRight (Hand hand)// 手划向右边
{
    return hand.PalmVelocity.x > deltaVelocity && !isStationary (hand);
}

[Tooltip ("Velocity (m/s) move toward ")]
protected float deltaVelocity = 0.7f;
```

图 3-2-3-1

上图为判定 Leap Motion 手向右划动的代码，算法和向左滑动类似，如果 Hand 在 X 轴上移动速度大于 deltaVelocity 的值（向右为 X 轴正方向），并且同时 isStationary 函数的返回值为 false，则视为手在向右划，并返回一个 bool

值。

3.2.4 手划向上边

```
protected bool isMoveUp (Hand hand)   //手向上
{
    return hand.PalmVelocity.y > deltaVelocity && !isStationary (hand);
}
```

图 3-2-4-1

上图为判定 Leap Motion 手向上滑动的代码，算法与之前相同，只是方向变化，如果 Hand 在 Y 轴上移动速度大于 deltaVeclocity 的值（向上为 Y 轴正方向），同时 isStationary 函数的返回值为 false，则视为手在向上划，并返回一个 bool 值。

3.2.5 手划向下边

```
protected bool isMoveDown (Hand hand) //手向下
{
    return hand.PalmVelocity.y < -deltaVelocity && !isStationary (hand);
}
```

图 3-2-5-1

上图为判定 Leap Motion 手向下滑动的代码，算法同上，不详细叙述了。

3.2.6 手是否抓取

```
protected bool isGrabHand (Hand hand)  //是否抓取
{
    return hand.GrabStrength > 0.8f;    //抓取力
}
```

图 3-2-6-1

Hand.GrabStrength 是 Leap Motion 中自带的函数，函数返回一个手势的抓取力，通过帧与帧之间的不同来判断抓取力的大小，这里定义大于 0.8 手的状态为抓取，并且返回一个 bool 值。

这里还有另一种方法，通过 hand.GrabAngle 来判断是否抓取。GrabAngle 值表示手指与手模型本身构成的角度。通过查看 4 个手指的方向之间的角度来计算一个新的角度。计算角度时不考虑拇指，对于张开的手，角度为 0 弧度，当姿势是握紧拳头时达到 π 弧度。这时可以通过定义一个合适的角度来判断手是否出去抓取状态。

3.2.7 手是否握拳

```
protected bool isCloseHand (Hand hand)     //是否握拳
{
    List<Finger> listOfFingers = hand.Fingers;
    int count = 0;
    for (int f = 0; f < listOfFingers.Count; f++) { //循环遍历所有的手~~
        Finger finger = listOfFingers [f];
        if ((finger.TipPosition - hand.PalmPosition).Magnitude < deltaCloseFinger)    // Magnitude
            //float deltaCloseFinger = 0.05f;
        {
            count++;
                        //if (finger.Type == Finger.FingerType.TYPE_THUMB)
                       //Debug.Log ((finger.TipPosition - hand.PalmPosition).Magnitude);
        }
    }
    return (count == 5);
}
```

图 3-2-7-1

上图为判断手是否握拳的代码，算法如下，创建一个 List，获取当前手的所有手指信息，finger.TipPosition 是当前手指指尖的坐标，每个手的指尖坐标减去 hand.PalmPosition（手掌掌心的坐标）得到的向量的长度如果小于设定值，则视为当前手处于握拳状态，并返回一个 bool 值。

在上一个是否抓取的章节中提到了 hand.GrabAngle，更为简单的方法是直接判断 hand.GrabAngle 的值是否为 π，即可判断是否握拳。

3.2.8 手是否完全张开

```
protected bool isOpenFullHand (Hand hand)         //手掌全展开~
{
    //Debug.Log (hand.GrabStrength + " " + hand.PalmVelocity + " " + hand.PalmVelocity.Magnitude);
    return hand.GrabStrength == 0;
}
```

图 3-2-8-1

上图为判断手是否完全张开的代码，算法十分简单，直接判断手的 GrabStrenth 值是否为 0，为 0 则是完全张开并返回一个 bool 值。除了此方法外，也可以判断 hand.GrabAngle 的值是否为 0.

3.2.9 向量的判断

- 向量转化为角度

```
protected float angle2LeapVectors (Leap.Vector a, Leap.Vector  b)
{
          //向量转化成 角度
           return Vector3.Angle (UnityVectorExtension.ToVector3 (a), UnityVectorExtension.ToVector3
}
```

图 3-2-9-1

这段代码的作用是将 Leap Motion 中的两个向量转化为 Unity 中的角度。

• 判断两个向量是否相同方向

```
protected bool isSameDirection (Vector a, Vector b)
{
            //判断两个向量是否 相同 方向
            //Debug.Log (angle2LeapVectors (a, b) + " " + b);
            return angle2LeapVectors (a, b) < handForwardDegree;
}
```

图 3-2-9-2

设定了一个浮点数 handForwardDegree 为 30F，默认两个向量 a，b 之间的夹角如果小于 30 则判定它们为统一方向。

• 判断两个向量是否相反方向

```
protected bool isOppositeDirection (Vector a, Vector b
{
//判断两个向量是否 相反 方向
        return angle2LeapVectors (a, b) > (180 - handForwardDegree);
}
```

图 3-2-9-3

方法与上面相同，只是判定的角度变为了 180 减去设定的 handForwardDegree 值。

• 判断手掌心的向量方向是否与另一个向量相同

```
protected bool isPalmNormalSameDirectionWith (Hand hand, Vector3 dir)
{
     //判断手的掌心方向于一个  向量   是否方向相同
    return isSameDirection (hand.PalmNormal, UnityVectorExtension.ToVector (dir));
}
    [Tooltip ("Delta degree to check 2 vectors same direction")]  //三角度检查2个向量的方向相同
        protected float handForwardDegree = 30;
```

图 3-2-9-4

算法十分简单，通过之前写的 isSameDirection 函数，函数传参的值变为 hand.PalmNormal 的值和另外一个向量的值即可。

3.2.10 向手掌的方向移动

```
protected bool isHandMoveForward (Hand hand)
       {
           return isSameDirection (hand.PalmNormal, hand.PalmVelocity) && !isStationary (hand);
       }
```

图 3-2-10-1

上图为判断手是否在向掌心向量的方向移动。具体算法如下，用 isSameDirection 判断掌心的向量方向和手掌移动的向量方向是否一致，如果一致并且此时手不处于固定状态，则手在向掌心方向移动，并返回一个 bool 值。

3.2.11 手掌是否垂直（掌心水平）

```
protected bool checkPalmNormalInXZPlane (Hand hand)  //   hand.PalmNormal 垂直于掌心的向量
{
    float anglePalmNormal = angle2LeapVectors (hand.PalmNormal, UnityVectorExtension.ToVector (Vector
    return (anglePalmNormal > 70 && anglePalmNormal < 110);
}
```

图 3-2-11-1

上图为判断手掌是否垂直的代码。具体算法如下，用 angle2LeapVectors 判断 hand.PalmNormal（垂直于掌心的向量）和 vector.up（垂直向上的向量）之间的夹角，当 hand.PalmNormal 与竖直向量成 90 度是，即为手掌水质。但由于在实际操作中不会完全水平，所以给予一个误差上下 20 度，即两个向量夹角的值 anglePalmNormal 在 70 度到 110 度之间都认为是手掌水平。

3.2.12 判断大拇指是否竖直或向下

```
protected bool isThumbDirection (Hand hand, Vector3 dir)
      {
          List<Finger> listOfFingers = hand.Fingers;
          for (int f = 0; f < listOfFingers.Count; f++) {
              Finger finger = listOfFingers [f];
              if (finger.Type == Finger.FingerType.TYPE_THUMB) {
                  float angleThumbFinger = angle2LeapVectors (finger.Direction,
                                                  UnityVectorExtension.ToVector (dir));
                  float angleThumbFinger2 = angle2LeapVectors (
                                                   finger.StabilizedTipPosition - hand.PalmPosition, Uni
                  //Debug.Log (angleThumbFinger + " " + angleThumbFinger2);
                  if (angleThumbFinger < deltaAngleThumb
                     || angleThumbFinger2 < deltaAngleThumb)
                      return true;
                  else
                      return false;
              }
          }
          return false;
      }
```

图 3-2-12-1

从上图的代码中可以看到，用一个 List 将手指的信息全部储存下来，并只对里面有 TYPE_THUMB（大拇指）标记的手指进行处理，采用了两个值，一个 finger.direction，手指的方向与数值方向判断，另一个是手指的坐标减去掌心的坐标形成向量进行判断。当这两个值与竖直方向的夹角小鱼设定的 deltaAngleThumb 时，则可以判断大拇指处于竖直或者向下的状态，并返回一个

bool 值。

3.2.13 判断四指是否靠拢掌心

```
protected bool checkFingerCloseToHand (Hand hand)
{
            List<Finger> listOfFingers = hand.Fingers;
            int count = 0;
            for (int f = 0; f < listOfFingers.Count; f++) {
                Finger finger = listOfFingers [f];
                if ((finger.TipPosition - hand.PalmPosition).Magnitude < deltaCloseFinger) {
                    if (finger.Type == Finger.FingerType.TYPE_THUMB) {
                        return false;
                    } else {
                        count++;
                    }
                }
            }
            //Debug.Log (count);
            return (count == 4);
}
```

图 3-2-13-1

上图为具体代码，具体算法如下，使用一个 List 存储手的手指信息，分别遍历 List 中除了大拇指的手指信息。当 finger.TipPosition（手指的指尖位置）与掌心位置相减得到的向量的大小小于设定的 deltaCloseFinger 值后，即可认为手指此时靠拢掌心，并且返回一个 bool 值。

第四章 LeapMotion 与 Unity 的结合实例

4.1 游戏介绍

本次的 Unity 游戏类型选择了跑酷游戏，与以往在电脑上通过键盘或者手机上通过屏幕滑动人物移动操控不同，这次加入了 Leap Motion 手势控制器，玩家可以通过手势的挥动直接控制游戏中人物的移动，使玩家获得不同的游戏体验。

游戏的基本玩法是玩家控制的角色在一个无限循环的道路上奔跑，玩家通过手势控制角色躲避不同的障碍的同时获得更多的金币以增加更多的分数。在奔跑的同时可以通过获得不同的道具来得到不同的效果，以此增加游戏的趣味性和挑战性。

4.1.1 人物介绍

下图是游戏里的角色

图 4-1-1-1

该角色拥有几个不同的状态，分别是奔跑，左右移动，跳跃翻滚以及死亡状态。

奔跑：

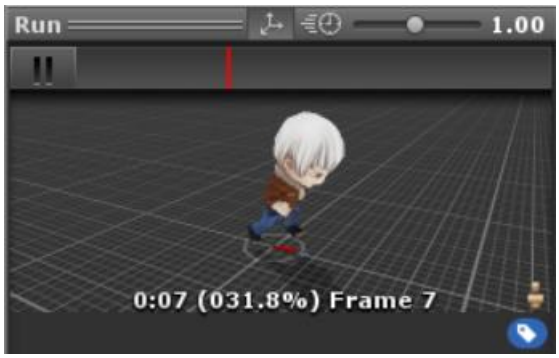


图 4-1-1-2

左右移动：

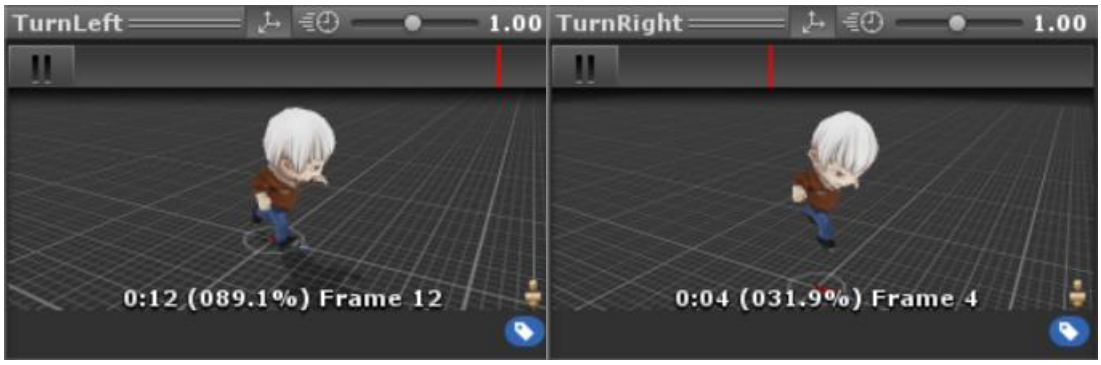


图 4-1-1-2

跳跃翻滚：

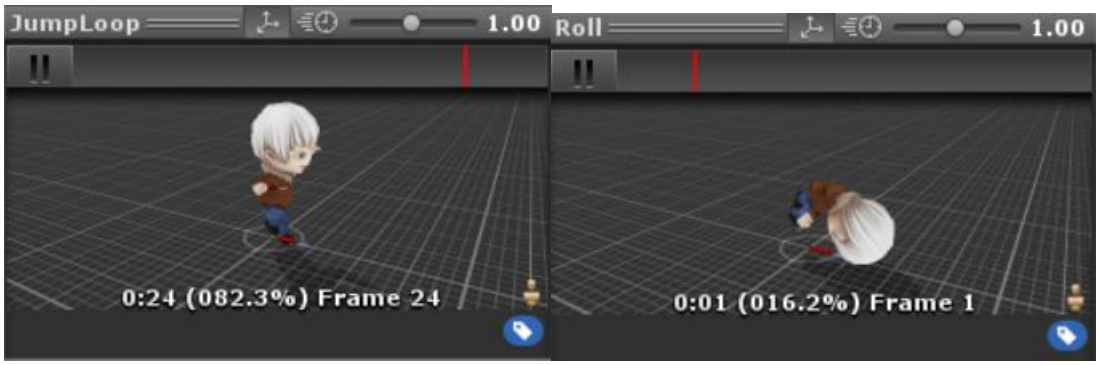


图 4-1-1-3

死亡状态：

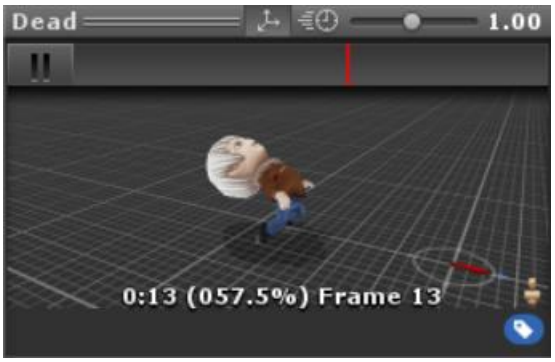


图 4-1-1-4

以上的几种状态构成了角色的移动控制。

4.1.2 道具介绍

游戏中一共有 5 种道具，金币和 4 个状态道具。

图 4-1-2-1

金币在游戏中旋转的，目的是在于使玩家控制的角色获取更多的分数。

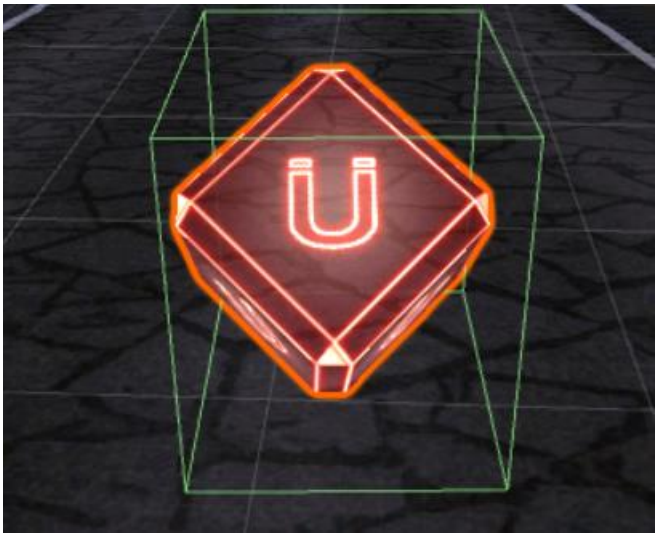


图 4-1-2-2

道具 Magnet（磁铁），作用是吸引身边一定范围内的所有金币，获得分数。

图 4-1-2-3

上图这个跑鞋道具可以是游戏角色获得双连跳效果，即在空中可以进行再一次地跳跃，方便玩家通过一些平常跳跃无法越过的障碍。

图 4-1-2-4

上图是双倍积分，效果是将一定时间内，获得的金币数乘以二，换言之就是获得双倍的分数。

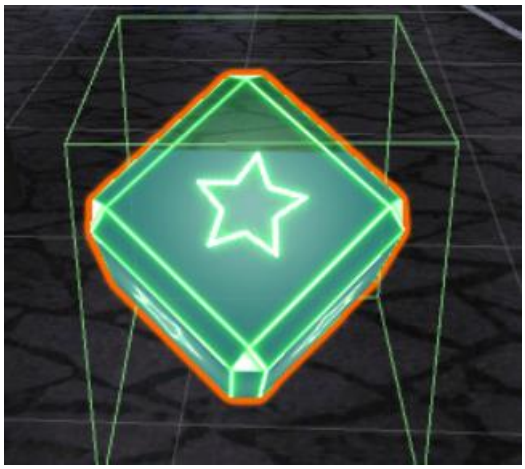


图 4-1-2-5

这是星型道具，效果使角色的移动速度加倍，可以使得游戏获得金币的过程变快，但同时也会增加游戏失败的概率，这个道具是一把双刃剑。

4.2 游戏实现

4.2.1 移动功能的实现

游戏里的人物的移动需要由 Leap Motion 控制器来实现，具体手势是通过上文设定的代码来确定。具体实现代码如下图所示：

```
void GetInputDirection()
{
    //通过foreach获取当前帧的手势，并利用函数获取方向
    inputDirection = InputDirection.NULL;
    Frame frame = provider.CurrentFrame;
    foreach (Hand hand in frame.Hands)
    {

        if (GameObject.Find("LeapHandController").GetComponent<LeapDirection>().isMoveLeft(hand))
        {
            AudioManager.instance.PlaySlide();
            inputDirection = InputDirection.Left;
        }
        else if (GameObject.Find("LeapHandController").GetComponent<LeapDirection>().isMoveRight(hand))
        {
            AudioManager.instance.PlaySlide();
            inputDirection = InputDirection.Right;
        }
        else if (GameObject.Find("LeapHandController").GetComponent<LeapDirection>().isMoveUp(hand))
        {
            AudioManager.instance.PlaySlide();
            inputDirection = InputDirection.Up;
        }
        else if (GameObject.Find("LeapHandController").GetComponent<LeapDirection>().isMoveDown(hand))
        {
            AudioManager.instance.PlaySlide();
            inputDirection = InputDirection.Down;
        }
```

图 4-2-1-1

可以看见代码中有一个名为 GetInputDirection 的函数，这个函数的功能是判断此时人物所处的状态，在 InputDirection 中一共有 5 个状态，分别为 null，left，right，down，up。代表上下左右。判定方法为，获取当前帧的手部信息，再用 foreach 函数遍历所有的 hand 的信息。这时候用 if 语句来判断，当前手的信息是否满足上下左右的条件，比如 isMoveLeft(hand)，判断手是否向左划。如果满足条件，播放滑动的音效并且将状态该为移动方向的状态。

```
IEnumerator UpdateAction()
{
    while (GameAttribute.instance.life > 0)
    {
        if(GameController.instance.isPlay && !GameController.instance.isPause)
        {
            GetInputDirection();//记录手势方向
            //PlayAnimation();//播放动画
            MoveLeftRight();//左右移动
            MoveForward();//上下移动
            UpdateSpeed();
        }
        else
        {
            animation.Stop();
        }

        yield return 0;
    }
    speed = 0;
    AnimationManager.instance.animationHandler = AnimationManager.instance.PlayDead;
    Debug.Log("restart");
    iTween.MoveTo(gameObject, gameObject.transform.position - new Vector3(0, 0, 2), 2.0f);
    yield return new WaitForSeconds(1.5f);
    UIController.instance.ShowRestartUI();
    UIController.instance.HidePauseUI();
}
```

图 4-2-1-2

上图是 C#中的协程，控制动作的更新，如图可知，在玩家生命值大于 0 时，可以进行游戏动作判定。如上图所示，记录手势方向之后，会有左右移动和上下移动的函数分别调用，如果满足条件则会播放左右移动或者上下跳跃翻滚的动画。

```
/*--------------------------人物移动控制函数-----------------*/
void MoveLeft()
{
    if(standPosition != Position.Left)  //判断是否在最左侧，若在无法向左移动
    {
        GetComponent<Animation>().Stop();
        AnimationManager.instance.animationHandler = AnimationManager.instance.PlayTurnLeft;

        xDirection = Vector3.left;//给人物X轴方向的力

        if (standPosition == Position.Middle)
        {
            standPosition = Position.Left;
            fromPosition = Position.Middle;
        }
        else if(standPosition == Position.Right)
        {
            standPosition = Position.Middle;
            fromPosition = Position.Right;
        }
    }
}
```

图 4-2-1-3

上图是向左划动的控制代码，类似于神庙逃亡的三条道路，所以在最左边不能再往左划。采取的方法是当状态变成向 left 时，给人物一个向左的力，对应上时 X 轴反向的力，同时播放划动的动画，并且记录此时的位置和变化之前的位置。

向右划动的控制代码与此类似，所以就不再重复了。同时代码中还有一个 MoveLeftRight 函数，具体函数代码部分如下：

```
if (standPosition == Position.Left)
{
    if (transform.position.x <= -1.7f)
    {
        xDirection = Vector3.zero;
        transform.position = new Vector3(-1.7f, transform.position.y, transform.position.z);
    }
}
else if (standPosition == Position.Right)
{
    if (transform.position.x >= 1.7f)
    {
        xDirection = Vector3.zero;
        transform.position = new Vector3(1.7f, transform.position.y, transform.position.z);
    }
}
else if (standPosition == Position.Middle)
{
    if(fromPosition == Position.Left)
    {
        if (transform.position.x >= 0)
        {
            xDirection = Vector3.zero;
            transform.position = new Vector3(0, transform.position.y, transform.position.z);
        }
    }
```

图 4-2-1-4

主要功能是统一向左向右移动的函数并且在人物移动到指定位置后停止人物收到的 X 轴方向的力，避免人物一直移动出道路外面。

接下来的函数时 MoveForward，函数的代码如下：

```
void MoveForward()
{
    if (inputDirection == InputDirection.Down)
    {
        AnimationManager.instance.animationHandler = AnimationManager.instance.PlayRoll;
    }
    if (characterController.isGrounded)//检测人物是否在地上
    {
        moveDirection = Vector3.zero;
        if (AnimationManager.instance.animationHandler != AnimationManager.instance.PlayRoll
            && AnimationManager.instance.animationHandler != AnimationManager.instance.PlayTurnLeft
            && AnimationManager.instance.animationHandler != AnimationManager.instance.PlayTurnRight)
        {
            AnimationManager.instance.animationHandler = AnimationManager.instance.PlayRun;
        }

        if(inputDirection == InputDirection.Up)
        {
            JumpUp();
            if (canDoubleJump)
            {
                doubleJump = true;
            }
        }
    }
```

图 4-2-1-5

```
else
{
    if(inputDirection == InputDirection.Down)
    {
        QuickGround();
    }

    if(inputDirection == InputDirection.Up)
    {
        if (doubleJump)
        {
            JumpDouble();
            doubleJump = false;
        }
    }

    if(AnimationManager.instance.animationHandler != AnimationManager.instance.PlayJumpUp
        && AnimationManager.instance.animationHandler != AnimationManager.instance.PlayRoll
        && AnimationManager.instance.animationHandler != AnimationManager.instance.PlayDoubleJump)
    {
        AnimationManager.instance.animationHandler = AnimationManager.instance.PlayJumpLoop;
    }
}
moveDirection.z = speed;
moveDirection.y -= gravity * Time.deltaTime;
characterController.Move((xDirection * 10 + moveDirection) * Time.deltaTime);
```

图 4-2-1-6

这个函数的逻辑分为两层，第一是判断人物是否在地上，如果 isGround 为 True，则再判断是状态是 up 还是 down，并播放相应的动画。如果不再地上，则判断是否还能连续跳跃和翻滚直接落到地面上。在空中会有几个阶段，起跳，空中滑翔和落地，这几个时刻的动画都是不同的，所以在函数中都会做判断来确定人物当时的状态。

```
void JumpUp()
{
    AnimationManager.instance.animationHandler = AnimationManager.instance.PlayJumpUp;
    moveDirection.y += jumpValue;
}

void JumpDouble()
{
    AnimationManager.instance.animationHandler = AnimationManager.instance.PlayDoubleJump;
    moveDirection.y += jumpValue * 1.5f;
}

void QuickGround()
{
    moveDirection.y -= jumpValue * 3;
}
```

图 4-2-1-7

具体的跳跃函数如图所示，基本原理都是给人物 Y 轴方向的力，形成一个跳跃的效果。

4.2.2 道具功能的实现

所有可以碰撞的道具都继承一个自定义的 Item 类，类的代码具体如下：

```
void Start()...

// Update is called once per frame
void Update()
{
    gameObject.transform.Rotate(0, rotateSpeed * Time.deltaTime, 0);
}

public virtual void PlayHitAudio()
{
    AudioManager.instance.PlayGetItem();
}

public virtual void HitItem()
{
    PlayHitAudio();
    GameObject effect = Instantiate(hitEffect);
    effect.transform.parent = PlayerController.instance.gameObject.transform;
    effect.transform.localPosition = new Vector3(0, 0.5f, 0);
    Destroy(gameObject);
}

public virtual void OnTriggerEnter(Collider other)
{
    if (other.tag == "Player")
    {
        HitItem();
    }
```

图 4-2-2-1

这个 Item 类中定义了道具的基本属性，rotate 旋转，碰撞时候播放相对应的音效，还有碰撞时候的效果。由于每个道具的音效和碰撞效果不一定一样，所以可以从上面代码发现这两个函数定义为 virtual 函数，使得继承该类的子类可以复写该函数。

4.2.2.1 金币

```
public class Coin : Item {

    /*public override void OnTriggerEnter(Collid
    {
        base.OnTriggerEnter(other);
        if (other.tag == "Player")
        {
            GameAttribute.instance.AddCoin(1);
        }

    }*/

    public override void HitItem()
    {
        base.HitItem();
        GameAttribute.instance.AddCoin(1);
    }

    public override void PlayHitAudio()
    {
        AudioManager.instance.PlayCoinAudio();
    }

}
```

图 4-2-2-1-1

金币效果比较简单，复写的函数中改写了碰撞时候的播放音效和分数加一的效果。

4.2.2.2 Magnet（磁铁道具）

Magnet 的相关类有两个，一个是继承 Item 的 Magnet 类，复写了碰撞音效和碰撞效果。另外一个则是实现磁铁的功能，吸引一定范围内的金币。具体部分代码如下：

```
IEnumerator HitCoin(GameObject coin)
{
    bool isLoop = true;
    while (isLoop)
    {
        if(coin == null)
        {
            isLoop = false;
            continue;
        }
        coin.transform.position = Vector3.Lerp(coin.transform.position, PlayerController.instance.gameObject.transform.position, Time.deltaTime * 10);
        if(Vector3.Distance(coin.transform.position, PlayerController.instance.gameObject.transform.position) < 0.5f)
        {
            coin.GetComponent<Coin>().HitItem();
            isLoop = false;
        }
        yield return null;
    }
}
```

图 4-2-2-2-1

具体实现方法是，给玩家控制的人物增加一个范围确定的圆形碰撞体，当金币碰撞到这个碰撞体后就会向人物移动，碰到人物时发生碰撞事件。在上图的代码中，可以看到使用了 Vector3.Lerp 插值函数，将金币平滑地移动到人物身边。

4.2.2.3 Shoe（跑鞋道具）

跑鞋道具代码上前面的道具类似，不同之处在与会是 PlayController 这个类中的 canDoubleJump 值从 false 变为 true，使得人物可以在空中进行二次跳跃。

4.2.2.4 Multiply（双倍积分）

双倍积分效果是将获得金币的倍数改为 2，并且同时有时间的判定，一定时间后结束双倍积分的效果。

```
IEnumerator MultiplyCoroutine()
{
    multiplyTimeLeft = multiplyDuration;
    GameAttribute.instance.multiply = 2;
    while(multiplyTimeLeft >= 0)
    {
        if (GameController.instance.isPlay && !GameController.instance.isPause)
            multiplyTimeLeft -= Time.deltaTime;
        yield return null;
    }
    GameAttribute.instance.multiply = 1;
}
```

图 4-2-2-4-1

4.2.2.5 Star（暴走鞋）

```
IEnumerator QuickMoveCoroutine()
{
    quickMoveTimeLeft = quickMoveDuration;
    if (!isQuickMoving)
    {
        saveSpeed = speed;
    }
    speed = 20;
    isQuickMoving = true;
    //yield return new WaitForSeconds(quickMoveDuration);
    while(quickMoveTimeLeft >= 0)
    {
        if (GameController.instance.isPlay && !GameController.instance.isPause)
            quickMoveTimeLeft -= Time.deltaTime;
        yield return null;
    }
    speed = saveSpeed;
    isQuickMoving = false;
}
```

图 4-2-2-5-1

在这个代码中有一个计时的问题，在测试时，发现如果在效果暴走效果时间结束之前再次获得了加速效果，会导致效果持续时间的计时发生问题，时间减少变快。这是由于没有用协程的关系，同时接受到两个效果后，Time.deltaTime 的时间减少变快。使用协程（Coroutine）后就能有效地解决这个问题。

4.2.3 场景的设计与完善

4.2.3.1 道路系统的设计

这个由 Leap Motion 手势控制的跑酷游戏是无限模式的，即如果人物不死亡，可以一直跑下去，道路上的障碍物和各种道具也是随机生成的。这里需要介绍的就是如何使道路无限延长。

```
// Update is called once per frame
void Update () {
    if(transform.position.z >= floorOnRunning.transform.position.z + 32)
    {
        RemoveItem(floorOnRunning);
        AddItem(floorOnRunning);
        floorOnRunning.transform.position = new Vector3(0, 0, floorForward.transform.position.z + 32);
        GameObject temp = floorOnRunning;
        floorOnRunning = floorForward;
        floorForward = temp;
    }
}
```

图 4-2-3-1-1

上图为具体代码，基本的算法是当人物跑到了当前道路的起点时，将上一个道路上的东西移除，并添加新的障碍物道具后，改变之前道路的位置，放在当前道路的结尾，如此循环反复，使得路面无限延长。这个算法的要点是需要将 floorOnRunning 和 floorForward 区分清楚，这样才能顺利地进行道路的替换和移动。

接下来要介绍的是道路上的障碍物和道具的随机生成。

```
public class PatternSystemAdd : EditorWindow {

    [MenuItem("Window/AddPatternToSystem")]
    static void AddPatternToSystem()
    {
        var gameManager = GameObject.Find("GameManager");
        if(gameManager != null)
        {
            var patternManager = gameManager.GetComponent <PatternManager> ();
            if(Selection.gameObjects.Length == 1)
            {
                var item = Selection.gameObjects[0].transform.FindChild("Item");
                if(item != null)
                {
                    Pattern pattern = new Pattern();
                    foreach( var child in item)
                    {
                        Transform childTransform = child as Transform;
                        if(childTransform != null)
                        {
                            var prefab = UnityEditor.PrefabUtility.GetPrefabParent(childTransform.gameObject);
                            if(prefab != null)
                            {
                                PatternItem patternItem = new PatternItem
                                {
                                    gameObject = prefab as GameObject,
                                    position = childTransform.transform.localPosition
                                };
                                pattern.PatternItems.Add(patternItem);
                            }
                        }
                    }
                    patternManager.Patterns.Add(pattern);
```

图 4-2-3-1-2

上图所示的代码是完成了一个添加物体的功能。即在设计道路时，完成了一个场景的设计，可以通过如上代码写出来的功能点击 AddPatternToSystem 按钮完成对 GameAttrinbute 中 Pattern 的添加，这样能保存所有添加的场景，并且在之后的道路功能实现中随机地添加在新生成的道路上。

```
void AddItem(GameObject floor)
{
    var item = floor.transform.FindChild("Item");
    if (item != null)
    {
        var patternManager = PatternManager.instance;
        if(patternManager != null && patternManager.Patterns != null && patternManager.Patterns.Count > 0)
        {
            var pattern = patternManager.Patterns[Random.Range(0, patternManager.Patterns.Count)];
            if(pattern != null && pattern.PatternItems != null && pattern.PatternItems.Count > 0)
            {
                foreach(var patternItem in pattern.PatternItems)
                {
                    var newObj = Instantiate(patternItem.gameObject);
                    newObj.transform.parent = item;
                    newObj.transform.localPosition = patternItem.position;
                }
```

图 4-2-3-1-3

上图是 AddItem 的具体代码，在生成新的道路时会执行上面的函数。Pattern 里面的物体会记录类型和位置，所以在添加的时候会完全按照当时设计道路时的摆放位置添加。

4.2.3.2 障碍物与背景的实现

障碍物与背景的房屋是手动放置上去的，同时为了完成游戏功能的实现，需要自己添加碰撞体，并且区分致死碰撞体和正常的碰撞体（只是不让人物通过并不会减少人物生命值）。

背景方面需要注意的是贴图颜色的调配和亮度的选择，因为使用的是 Curve 的贴图渲染方式，会有一定的弯曲折射效果，亮度选择过高后会发生一定的透视效果，影响游戏体验。其次就是方向的放置，在 Unity 中可以较为便捷地实现这些背景的放置。

下面给出障碍物类的部分具体代码，所有游戏中的障碍物均继承与该类。

```
// Update is called once per frame
void Update () {
    transform.Translate(moveDirection * moveSpeed * Time.deltaTime);
}

public virtual void OnTriggerEnter(Collider other)
{
    if(other.tag == "Player")
    {
        AudioManager.instance.PlayHit();
        CameraManager.instance.CameraShake();
        GameAttribute.instance.life -= hurtValue;
    }
    if (other.tag != "Road" && other.tag != "MagnetCollider")
    {
        moveSpeed = 0;
        Debug.Log(other.name);
    }
}
```

图 4-2-3-2-1

图中给出了障碍物的一些功能实现方法，碰撞到玩家后，会触发碰撞音效和相机抖动，也会将玩家的生命值减 1.

在其中值得一提的是，在障碍物 Board（栅栏）的判定方法，障碍物图和碰撞体积如下：

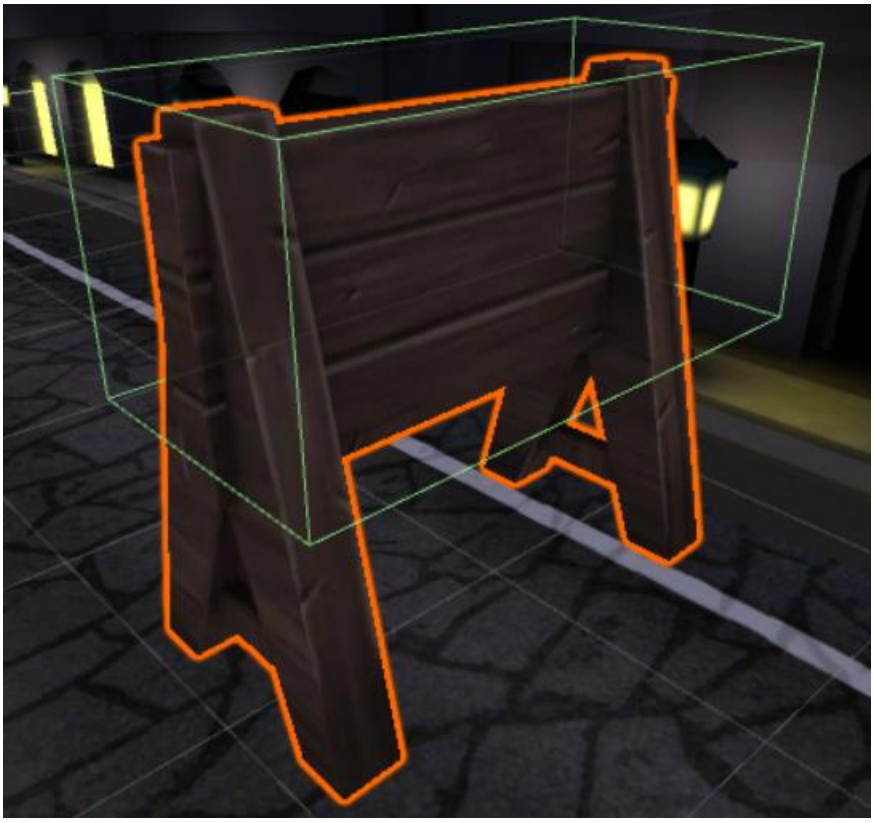


图 4-2-3-2-2

人物在经过这个障碍物时如果不向下划动手使角色产生翻滚的动画，就会碰到栅栏的上方碰撞体，从而结束游戏。而为了实现这个功能，代码如下：

```
public class Board : Obstacle {

    public override void OnTriggerEnter(Collider other)
    {
        if(PlayerController.instance.isRoll != true)
        {
            base.OnTriggerEnter(other);
        }
    }

}
```

图 4-2-3-2-3

实现的方法如上所示，即当玩家的 isRoll 状态不为真的时候，出发碰撞体效果。

4.2.3.3 UI 界面设计

游戏的 UI 界面是一个十分重要的环节，在论文提及的游戏中游戏 UI 大概有一下几个方面，游戏开始，游戏结束，暂停，死亡后的重新开始，音乐开关，左上角的金币获得总数以及右上方的道具状态提示栏。

下图是游戏开始时的画面：

图 4-2-3-3-1

下图是人物死亡时的画面：

图 4-2-3-3-2

里面涉及的逻辑是，在游戏一开始的时候，游戏进入开始画面，此时游戏画面暂停，当玩家点击 Play 按钮后，游戏开始，人物开始跑动，并且右下角出现暂停按钮，点击暂停按钮时，人物画面会停止，并且右上角的道具状态栏的时间保持不变。同时画面中间出现 Resume 按钮和 Exit 按钮。如下图所示，

图 4-2-3-3-3

当点击 Resume 按钮后游戏恢复正常。功能实现是改变 IsPlay 和 IsPause 的 bool 值实现的，在游戏各种功能的判断前都会判断是否处于游戏进行或者暂停亦或是死亡状态，当状态不满足条件时，游戏不会执行相应的功能。

第五章 LeapMotion 手势控制的优点与局限性

5.1 优点

• 小巧方便，只需要一根 Usb 线就可以使用。不同于其他手势识别的设备，例如 Myo 的绑带，需要将其控制带佩戴任何一条胳膊的肘关节上方，十分的繁琐并且不如 Leap Motion 那样携带便捷。

• 支持跨平台。在 Linux，Windows，Mac 上都能使用并且只需要下载不同的 SDK 环境配置包即可，十分方便快捷。

• 支持多种开发语言，例如 C++，C#以及 Java 还有 Python。甚至 Leap Motion 发行公司通过 WebSocket 实现了浏览器中的 JavaScript API。使得该手势控制器在 Web 上也能有所建树。

• 跟踪手指和手掌的精度相当得高。示意图如下图所示：

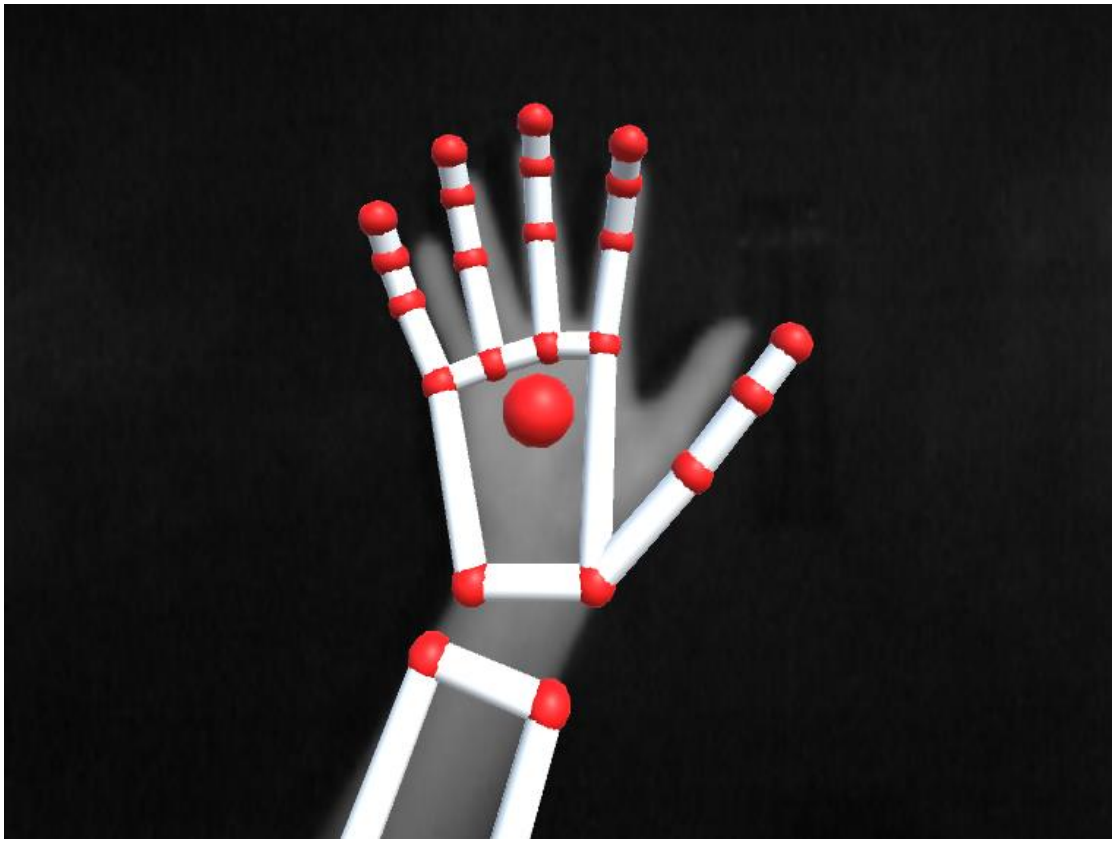


图 5-1-1

相比如前面你提到过的 Myo 绑带，Leap Motion 控制器对于手指和手掌的定位更加得精确，并且不像 Myo 需要那么多的空间，功能也比 Myo 丰富不少。这些都是 Leap Motion 作为手势控制器的优势。

• 能与 Oculus 的 VR 头盔结合在一起。这是许多其他手势控制器无法做到的，手势控制器能与 VR 结合在一起的话可以带给人们更加的虚拟现实体验，在 VR 世界中实现的手势控制使得以后的开发策略和重点变得更加多样性，给予 VR 以更多的可能性。

5.2 局限性

• 检测范围小，容易造成手臂酸痛。在测试当中，稍微较长时间的游戏都会造成手臂酸痛，无法长时间进行游戏。

• 不能检测身体和脸部。虽然这是一个手势控制器，但是无法识别身体和脸部也会对游戏性有一定的影响。

• 某种情况下在正常光照情况下无法正常识别手部运动。这是由于 Leap Motion 自带的两个摄像头中会发出红外光，与正常光相遇后会产生一些影响导致无法正常识别手部的运动。

• 作为生产力工具，完全无法代替键盘和鼠标，并且在很多时候找不到使用场景，这是最致命的一点。举个例子，键盘鼠标，能百分百确定你的按键和输出信息时统一的，但是对于 Leap Motion 来说，即使它十分方便，但是只要有不小的几率会造成按键信息与输出信息不统一，这都将是毁灭性的打击。

第六章 总结与展望

6.1 总结

通过本次研究，我基本掌握了 Leap Motion 的使用方法，了解了 Leap Motion 的基本原理并熟悉了 Leap Motion 在 Unity 环境下的各种 API 的使用。在将 Leap Motion 结合在 Unity 游戏的示例中，我还学会了许多 Unity 游戏引擎和 C#编程语言的使用方法，对于我在今后的游戏设计之路上相当有帮助。通过这次对 Leap Motion 的开发也发现，Leap Motion 精巧易于携带，有较为精确的手势识别能力，在本次 Unity 示例的 3D 跑酷游戏中，展现出了较为优秀的识别能力，但是依旧需要一些代码的调节才能使得游戏开发进行得更为顺畅。

6.2 展望

Leap Motion 的前景拥有许多的可能性，以下是我对于该手势识别控制器的一些展望。

• 医疗科技。将 Leap Motion 集成到高科技的医疗电脑中，医生可以通过这些电脑跨大陆来执行高复杂度和微妙的操作。并且在 AR 的帮助下，医生能看到并控制所有的东西，而不需要跟病人待在同一个房间里。

• 家庭设备。在家中安装少量的 Leap Motion 设备帮助打开和关闭灯光，打开和关闭窗户以及控制电视和其他所有电子设备而无需实际接触任何东西。这对于行动不便的残疾人或在具有污染或者危险的地方十分有用。

• 娱乐。Leap Motion 可以帮助人们控制 AR 游戏中的角色。在无需实际接触的情况下控制无线电设备（例如汽车飞机船只），提供更多的准确性和自由度。

• 三维建模和图形。在之后发展更加精确的情况下，可以使用 Leap Motion 的 3D 控制器使得建筑师和设计师可以自由地操纵 3D 建模和 3D 图形。无需使用物理 2D 控制器例如鼠标键盘触摸板，就可以直接旋转缩放物体。

Leap Motion 的发展还任重道远，并且与 VR 的结合还处于磨合阶段，但是相信在不就的将来我们能进入更加真实立体的虚幻世界。

参考文献

[1]徐崇斌,周明全,沈俊辰,骆岩林,武仲科.一种基于 Leap Motion 的直观体交互技术[J].电子与信息学报,2015,37(02):353-359.

[2]胡弘,晁建刚,杨进,赵再骞,林万洪.Leap Motion 关键点模型手姿态估计方法[J].计算机辅助设计与图形学学报,2015,27(07):1211-1216.

[3]黄俊,景红.基于 Leap Motion 的手势控制技术初探[J].计算机系统应用,2015,24(10):259-263.

[4]I. Oropesa,T.L. de Jong,P. Sánchez-González,J. Dankelman,E.J. Gómez. Feasibility of tracking laparoscopic instruments in a box trainer using a Leap Motion Controller[J]. Measurement,2016,80.

[5]Elyoenai Guerra-Segura,Carlos M. Travieso,Jesús B. Alonso. Study of the variability of the Leap Motion's measures for its use to characterize air strokes[J]. Measurement,2017,10

[6]Smeragliuolo Anna H,Hill N Jeremy,Disla Luis,Putrino David. Validation of the Leap Motion Controller using markered motion capture technology.[J]. Journal of biomechanics,2016,49(9).

外文资料

Abstract Novel 3D acquisition devices like depth cameras and the Leap Motion have recently reached the market. Depth cameras allow to obtain a complete 3D description of the framed scene while the Leap Motion sensor is a device explicitly targeted for hand gesture recognition and provides only a limited set of relevant points. This paper shows how to jointly exploit the two types of sensors for accurate gesture recognition. An ad-hoc solution for the joint calibration of the two devices is firstly presented. Then a set of novel feature descriptors is introduced both for the Leap Motion and for depth data. Various schemes based on the distances of the hand samples from the centroid, on the curvature of the hand contour and on the convex hull of the hand shape are employed and the use of Leap Motion data to aid feature extraction is also considered. The proposed feature sets are fed to two different classifiers, one based on multi-class SVMs and one exploiting Random Forests. Different feature selection algorithms have also been tested in order to reduce the complexity of the approach. Experimental results show that a very high accuracy can be obtained from the proposed method. The current implementation is also able to run in real-time.

Keywords Depth · Gesture recognition · Calibration · Kinect · Leap Motion · SVM

Introduction

Automatic hand gesture recognition is a very intriguing problem that, if efficiently solved, could open the way to many applications in several different fields, e.g. human-computer interaction, computer gaming, robotics and automatic sign-language interpretation. The

---

G. Marin · F. Dominio · P. Zanuttigh ()

Department of Information Engineering, University of Padova, Padova, Italy e-mail: zanuttigh@dei.unipd.it

G. Marin e-mail: maringiu@dei.unipd.it

F. Dominio

e-mail: dominiof@dei.unipd.it

problem can be solved both by using wearable devices and with vision-based approaches. Vision-based hand gesture recognition [27] is less invasive and paves the way for a more natural interaction, but it is also a very challenging problem.

Until a few years ago, all the available approaches were based on the extraction of the information from images and videos [12]. These representations contain a 2D description of the three-dimensional hand pose, which is often difficult to properly understand, and consequently to recognize the performed gesture. This is mainly due to the complex 3D movements that the hand and fingers can do and to the presence of many inter-occlusions between the various hand parts.

The introduction of Time-Of-Flight cameras and of low cost consumer depth cameras based on structured light [6] has made 3D data acquisition available to the mass market, thus opening the way to a new family of computer vision methods that exploit 3D information to recognize the performed gestures. In particular the success of Microsoft's Kinect™ has shown how natural

interfaces based on the acquisition of 3D data can be efficiently employed in commercial applications. However, notice how the standard usage of this device allows to recognize the whole body gestures but not the small details associated to the pose of the fingers. In order to exploit the data of the Kinect and of similar devices for hand gesture recognition, several methods have been proposed. The basic idea behind most of them is to extract relevant features from the depth data and then apply machine-learning techniques to the extracted features. An overview of the various available approaches will be presented in Section 2.

The Leap Motion device is another recently introduced sensor based on vision techniques targeted to the extraction of 3D data, but differently from the Kinect that provides a 3D description of the framed scene, this device is explicitly designed for hand gesture recognition and directly computes the position of the fingertips and the hand orientation. Compared with depth cameras like the Kinect, it produces a far more limited amount of information (only a few keypoints instead of the complete depth description) and works on a smaller 3D region. On the other side, the extracted data are more accurate (according to a recent study [29] its accuracy is of about 200μm) and it is not necessary to use computer vision algorithms to extract the relevant points since they are directly provided by the device. The software provided with the Leap Motion recognizes a few movement patterns only, e.g., swipe or tap, and the exploitation of Leap Motion data for more complex gesture recognition systems is still an almost unexplored field.

Since the Kinect and the Leap Motion have quite complementary characteristics (e.g., a few accurate and relevant keypoints against a large number of less accurate 3D points), it seems reasonable to exploit them together for gesture recognition purposes. If the information provided by the two devices has to be jointly considered, a calibration of the whole system is needed. This paper, following this rationale, presents a novel approach for the combined use of the two devices for hand gesture recognition. Reliable feature extraction schemes from both the Kinect and the Leap Motion data are introduced. The use of joint information from the two devices for more reliable and faster feature extraction is also considered. This is made possible by an ad-hoc approach for the joint calibration of the two devices. Finally, two reliable classification schemes based on Support Vector Machines (SVM) and Random Forests (RF) are proposed. The performances of each of the two devices alone and of their joint exploitation are evaluated. Finally feature selection schemes are considered in order to reduce the dimensionality of the feature vectors. This work has several novel contributions: it presents the first attempt to detect gestures from the data acquired by the Leap Motion proposing reliable approaches for the feature extraction and for the gesture classification based on these features; it shows how to jointly calibrate the Leap Motion with depth cameras like the Kinect, a quite challenging task due to the limited amount of data provided by the Leap Motion; finally it shows how to jointly exploit the two devices for gesture recognition.

The paper is organized in the following way: Section 2 presents a brief overview of the related works, then Section 3 presents the general architecture of the proposed gesture recognition system. The two following sections present the feature descriptors extracted from the Leap Motion data (Section 4) and from depth data (Section 5).

A method for the joint calibration of the 3D measures from the two devices is presented in Section 6.

Then the classification stage is described in Section 7. Experimental results are presented in Section 8 and finally Section 9 draws the conclusions.

Related works

Hand gesture recognition from data acquired by the Kinect or other consumer depth cameras is a novel but very attractive research field. Many approaches have been presented, mostly based on the standard scheme of extracting relevant features from the depth data and then applying machine-learning techniques to the extracted features. In the approach of [14], silhouette and cell occupancy features are extracted from the depth data and used to build a shape descriptor. The descriptor is then used inside a classifier based on action graphs. Other approaches, e.g., [25] and [28] are based on volumetric shape descriptors. The two approaches both exploit a classifier based on Support Vector Machines (SVM). The histograms of the distance of hand edge points from the hand center are instead used in the approaches of [23] and [22]. Another approach based on an SVM classifier is [8], that employs 4 different types of features extracted from the depth data.

Other approaches instead estimate the complete 3D hand pose from depth data. Keskin et Al. [11] try to estimate the pose by segmenting the hand depth map into its different parts, with a variation of the machine learning approach used for full body tracking in [24]. Multi-view setups have also been used for this task [2], since approaches based on a single camera are affected by the large amount of occluded parts, making the pose estimation rather challenging.

Differently from the Kinect, the exploitation of Leap Motion data for gesture recognition systems is still an almost unexplored field. A preliminary study on the usage of this device for sign language recognition has been presented in [21]. The device has been used for Arabic sign language recognition in [18]: in this work the data extracted from the sensor is fed directly to two different machine learning classification algorithms, one based on a Naive Bayes Classifier and one exploiting Multilayer Perceptron Neural Networks. Another recent work [26] analyzes the trajectory of a finger returned by the Leap Motion in order to recognize handwriting. The approach exploits Dynamic Time Warping and a nearest neighbor search. The sensor has also been used for signature recognition using features based on the optical flow and on the trajectories in a recent work [19]. A gesture interface based on the Leap Motion has been presented in [9], where the authors use the device to control a robot arm.

Finally the work that is more related to this one is [16]. In this work we combined the depth-based descriptors of [8] with some ad-hoc descriptors for the Leap Motion data and fed the two sets to an SVM classifier. However, this approach handles the two sensors separately, with two independent processing pipelines and without jointly calibrating them.
The approach proposed in this paper starts from [16] but presents a method for the joint calibration of the two devices and exploits the calibration to extract the features with the combined use of the two sensors. Furthermore the employed feature set has also been updated with a larger feature set for the Leap Motion and new feature descriptors for the depth data

Gesture classification

The approaches of Sections 4 and 5 produce eight different feature vectors, four for the Leap Motion data and four for the depth data. Each vector describes some relevant clues regarding the performed gesture and in order to perform the recognition, two different classification schemes have been used, one based on a multi-class Support Vector Machine classifier and one based on Random Forests. There are 8 feature vectors grouped into the two sets $\mathbf{V}_{leap} = [\mathbf{F}^a,\mathbf{F}^d,\mathbf{F}^e,\mathbf{F}^p]$ that contains all the features extracted from Leap Motion data and $\mathbf{V}_{depth} = [\mathbf{F}^l,\mathbf{F}^\rho,\mathbf{F}^c,\mathbf{F}^{cc}]$ that collects

the features computed from depth information. Feature vectors extracted from the two devices are visually summarized in Fig. 8. Each vector can be used alone or together with any of the other descriptors. The combination of multiple feature descriptors can be obtained by simply concatenating the vectors corresponding to the selected features. The target of the approach is to classify the performed gestures into G classes, one for each gesture in the considered database.

The first classification scheme exploits a multi-class SVM classifier [4] based on the *oneagainst-one* approach. In the employed scheme a set of G(G−1)/2 binary SVM classifiers are used to test each class against each other. The output of each of them is chosen as a *vote* for a certain gesture. For each sample in the test set, the gesture with the maximum number of votes is selected as the output of the classification. In particular a non-linear Gaussian Radial Basis Function (RBF) kernel has been selected and the classifier parameters have been tuned exploiting grid search and cross-validation on the training set. Let us consider a training set containing data from M users. The space of parameters (C,γ) of the RBF kernel is divided by a regular grid. For each couple of parameters the training set is divided into two parts, one containing M−1 users for training and the other with the remaining user for validation and performance evaluation. The procedure is repeated M times changing the user in the validation set. The couple of parameters that gives the best accuracy on average is selected as the output of the grid search. Finally the SVM has been trained on all the M users of the training set with the optimal parameters.

Alternatively we also tested a second classification scheme exploiting Random Forests (RF) [3]. Each tree has been trained on a random sampling of the training set leaving out one third of the sampled vectors for the estimation of the out-of-bag error. The only model parameter to optimize, differently from the pair for the RBF kernel of SVM, is the size m of the feature subset in each node. The parameter controls a trade-off between the tree correlation and the predictive "strength" of each tree, and may be easily found by analyzing the out-of-bag error. The size of the forest is not a critical parameter since the classification error remains relatively stable if a sufficient number of trees is used. In our case we trained a Random Forest of 100 decision trees with a default value of $m = \sqrt{|\mathbf{F}|}$ with $|\mathbf{F}|$ the length of the feature vectors in the dataset ($|\mathbf{F}| = 435$ when all the considered features are used). The implementation of the Random Forest classifier provided by Matlab has been used.

Finally, since the considered vectors contain a large number of elements we also considered the use of feature selection schemes in order to reduce the number of features and avoid the usage of useless or redundant descriptors. Three different feature selection schemes have been tested. The first uses the *F-score* approach [5], i.e., the F-score is computed for each feature and the most discriminative features according to this measure are selected (i.e., the features with an F-score bigger than a pre-defined threshold). Two different thresholds have been used in order to produce two subsets with 16 and 128 features.

The second scheme is based on the Forward Sequential Selection (FSS) algorithm [1]. In this case, starting from the empty set, at each step a new feature is added to the selected ones by choosing the one that allows to obtain the larger improvement in the classification accuracy with respect to the previous step (the SVM classifier previously described has been used to evaluate the classification accuracy).

Finally a third feature selection scheme exploiting Random Forests has been tested. In this

case a classification is performed with the approach of [3] and the out-of-bag error is estimated. Then, in order to measure the importance of the various features, the values of one of the features are permuted and the out-of-bag error is estimated again. The procedure is repeated for each feature and the importance of each feature is given by the normalized average increase of the out-of-bag error after the permutation. This approach is detailed in [5]. The number of selected features is the same of the previous cases in order to allow a fair comparison.

Experimental results

The results have been obtained using the setup depicted in Fig. 6. A Leap Motion device and a Kinect have been used to jointly acquire the data relative to the performed gestures. The first generation Kinect depth camera has been selected due to its large diffusion, however any other depth camera, e.g., Creative's Senz3D or the second generation Kinect can be used in the proposed approach. The two devices have been jointly calibrated using the approach of Section 6 and synchronized in time. A software synchronization scheme has been used: its precision is sufficient for the recognition of gestures based on static poses like the ones considered in this paper.

The considered dataset of gestures contains the 10 different gestures shown in Fig. 9 executed by 14 different people. Each user has repeated each gesture 10 times for a total of 1400 different data samples. Up to our knowledge this is the first database containing both depth data and Leap Motion data and it is available on our website at the url http://lttm.dei.unipd.it/downloads/gesture. In order to compute the results we split the dataset in a train and a test set by using the *leave-one-person-out* approach of Section 7, i.e., we placed in the training set the data from all the users except one and in the test set the data from the remaining user. Since the amount of data associated to a single user (100 samples) is not sufficient for a reliable assessment of the performances we executed 14 completely independent tests changing each time the person in the test set, i.e., as shown in Fig. 10, in each test we used a train set with 13 people and a test set with a single person that is the remaining one. The results of the 14 tests have been averaged to obtain the final accuracy. Note that this is a more challenging test than the standard *leave-one-out* approach, since not only it guarantees that the data in the train set is different from the ones in the test set as in the standard case, but also that the train set does not contain any sample from the user in the test set. This means that the system should be able to classify the data from the user in the test set from what it has *learned* from users different from the one that is using it.

Let us start from the Leap Motion device. Table 1 shows the accuracy obtained using the classification algorithm of Section 7 on the data from this sensor. The 3D positions of the fingertips give a very good representation of the arrangement of the fingers and allow to obtain an accuracy of 81.5 %. They allow to recognize the majority of the gestures even if the recognition of some gestures is not always optimal, as it is possible to see from the confusion matrix in Table 2. For example, gestures G2 and G3 are sometimes confused with gesture G1. This is due mostly to the false positives returned by the Leap Motion sensor that sometimes detects a raised finger in gesture G1.

Fingertip distance features allow to obtain an accuracy of about 76 %: they are able to recognize most gestures but there are some critical issues, e.g., G2 and G3 are easily confused. A relevant issue for this descriptor is the limited accuracy of the hand direction estimation from the

Leap Motion that does not allow a precise match between the fingertips and the corresponding angular regions (i.e., it is not easy to recognize which finger has been raised if a single finger is detected). The other two features have slightly lower performance. The angles allow to obtain an accuracy of 74.2 % and a similar result (73 %) can be obtained.

Conclusions

In this paper an effective gesture recognition pipeline for the Leap Motion, for depth sensors and for their combined usage has been proposed. The different nature of data provided by the Leap Motion (i.e., a higher level but more limited data description) with respect to the depth cameras, poses challenging issues for which effective solutions have been presented. An ad-hoc calibration scheme allowed to jointly calibrate the Leap Motion with depth sensors. The limited number of points computed by the first device makes this task quite challenging but the proposed scheme allows to obtain a good accuracy sufficient for the joint exploitation of the data from the two devices. Several different feature sets have been presented for both sensors. Four different types of features have been extracted from the Leap Motion while different types of descriptors have been computed from the depth data based on different clues likes the distances from the hand centroid, the curvature of the hand contour and the convex hull of the hand shape. It has also been shown how to exploit Leap Motion data to improve the computation time and accuracy of the depth features.

Experimental results have shown how the data provided by Leap Motion, even if not completely reliable, allows to obtain a reasonable overall accuracy with the proposed set of features and classification algorithms. A very good accuracy can be obtained from depth data that is a more complete description of the hand shape, in particular distance and curvature descriptors allow to obtain almost optimal performances. Performances remain very good even when the classification algorithm is changed or feature selection approaches are used to reduce the dimensionality of the feature vectors.

Future work will address the the recognition of dynamic gestures with the proposed setup and improved schemes for the detection and localization of the fingertips jointly exploiting the data from the two sensors.

中文译文

手势识别与联合校准的Leap Motion和深度传感器

摘要:新型3D采集设备（如深度相机和Leap Motion）最近已进入市场。深度摄像机可以获取有框架场景的完整三维描述，而Leap Motion传感器是明确针对手势识别的设备，并且只提供一组有限的相关点。本文展示了如何共同利用这两种类型的传感器进行精确的手势识别。首先介绍了两种设备联合校准的临时解决方案。然后为Leap Motion和深度数据引入一组新的特征描述符。基于手部样本距质心，手部轮廓曲率和手形凸包的距离，采用了各种方案，并考虑使用Leap Motion数据来辅助特征提取。提出的特征集被提供给两个不同的分类器，一个基于多类SVM和一个利用随机森林。为了降低该方法的复杂性，还对不同的特征选择算法进行了测试。实验结果表明，该方法可以获得很高的精度。当前的实现也能够实时运行。

关键词:深度·手势识别·校准·Kinect·Leap Motion·SVM

介绍

自动手势识别是一个非常有趣的问题，如果有效地解决，可以打开在许多不同领域的应用，例如人机交互，计算机游戏，机器人学和自动手语解释。这个问题可以通过使用可穿戴设备和基于视觉的方法来解决。基于视觉的手势识别（27）具有较少的侵入性，为更自然的交互铺平了道路，但也是一个非常具有挑战性的问题。

直到几年前，所有可用的方法都是基于从图像和视频中提取信息[12]。这些表示包含三维手姿态的2D描述，这通常难以正确理解，并因此识别所执行的手势。这主要是由于手和手指可以做的复杂的3D运动，以及在不同的手部之间存在许多遮挡。

基于结构光（6）的飞行时间相机和低成本消费者深度相机的引入使得 3D数据采集可用于大众市场，从而开辟了一种利用3D信息来识别所执行手势的新的计算机视觉方法家族。特别是微软的KiCeTTM成功展示了基于3D数据采集的自然接口如何能有效地应用于商业应用。然而，注意该设备的标准使用如何允许识别整个身体姿势，而不是与手指的姿势相关的小细节。为了利用Kinect的数据和类似的手势识别设备，已经提出了几种方法。它们背后的基本思想是从深度数据中提取相关特征，然后将机器学习技术应用于所提取的特征。将在第2节中介绍各种可用的方法。

跳跃运动装置是另一种最近推出的基于3D数据提取的视觉技术的传感器，

但与提供框架场景的 3D 描述的 Kinect 不同，该装置被明确地设计用于手势识别并直接计算 TH。手指的位置和手的方向。与像 Kinect 这样的深度相机相比，它产生了更为有限的信息量（只有几个关键点而不是完整的深度描述），并且工作在更小的 3D 区域上。另一方面，提取的数据更准确（根据最近的研究（29），其精度约为 200μm），并且不必使用计算机视觉算法来提取相关点，因为它们是由设备直接提供的。具有跳跃运动的软件只识别了一些运动模式，例如，滑动或轻敲，并且对于更复杂的手势识别系统的跳跃运动数据的开发仍然是一个几乎未开发的领域。

由于 Kinect 和跳跃运动具有相当互补的特性（例如，针对大量不太精确的 3D 点的一些精确和相关的关键点），因此将它们一起用于手势识别目的似乎是合理的。如果必须联合考虑由两个设备提供的信息，则需要对整个系统进行校准。本文遵循这一原理，提出了一种新的方法，用于组合使用的两个手势识别装置。介绍了 Kinect 和跳跃运动数据的可靠特征提取方法。还考虑了使用来自两个设备的联合信息以获得更可靠和更快的特征提取。这是可能的一种自组织方法的联合校准的两个设备。最后，提出了基于支持向量机（SVM）和随机森林（RF）的两种可靠分类方案。评价了这两种装置各自的性能和它们的联合开发。最后，考虑特征选择方案，以减少特征向量的维数。这项工作有几个新的贡献：它提出了第一次尝试来检测手势的数据从跳跃运动所获得的，提出了可靠的方法用于特征提取和基于这些特征的手势分类；它展示了如何联合校准跳跃动机。N 的深度相机，如 Kinect，一个相当具有挑战性的任务，由于有限的数据提供的跳跃运动；最后，它展示了如何共同开发两个手势识别设备。

本文以以下方式组织：第 2 节简要介绍了相关的作品，然后第 3 节介绍了所提出的手势识别系统的总体结构。以下两个部分呈现从跳跃运动数据（第 4 节）和深度数据（第 5 节）中提取的特征描述符。

在第 6 节中给出了从两个装置联合校准三维测量的方法。

然后在第 7 节中描述分类阶段。实验结果在第 8 节中给出，最后第 9 节得出结论。

相关工作

来自 Kinect 或其他消费类深度相机采集的数据中的手势识别是一种新颖但非常具有吸引力的研究领域。已经提出了许多方法，主要基于从深度数据提取相关特征的标准方案，然后将机器学习技术应用于所提取的特征。在[14]的方法中，从深度数据中提取轮廓和细胞占用特征，并用于构建形状描述符。然后在基于动作图的分类器内使用描述符。其他方法，例如[25]和[28]则基于体积形状描述符。这两种方法都利用基于支持向量机（SVM）的分类器。在[23]和[22]的方法中使

用手边缘点与手中心距离的直方图。另一种基于 SVM 分类器的方法是[8]，它采用从深度数据中提取的 4 种不同类型的特征。

其他方法则是根据深度数据估算完整的 3D 手势。凯斯金等人。 [11]尝试通过将手深度图分割成不同的部分来估计姿态，其中采用了用于全身跟踪的机器学习方法[24]。多视图设置也被用于这项任务[2]，因为基于单个摄像机的方法受到大量遮挡部件的影响，使得姿态估计相当具有挑战性。

与 Kinect 不同的是，用于手势识别系统的 Leap Motion 数据的开发仍然是一个几乎未被开发的领域。 [21]中已经提出了用于手语识别的这种设备的初步研究。该设备已被用于阿拉伯手语识别[18]：在这项工作中，从传感器提取的数据直接提供给两个不同的机器学习分类算法，一个基于朴素贝叶斯分类器和一个利用多层感知器神经网络。另一项最近的工作[26]分析了 Leap Motion 为了识别手写而返回的手指的轨迹。该方法利用动态时间弯曲和最近邻居搜索。该传感器也被用于使用基于光流特征和最近工作中的轨迹的特征识别[19]。基于 Leap Motion 的手势界面已经在[9]中提出，其中作者使用该设备来控制机器人手臂。最后，与这个更相关的工作是[16]。在这项工作中，我们将[8]的基于深度的描述符与 Leap Motion 数据的一些特定描述符相结合，并将这两个集合提供给 SVM 分类器。但是，这种方法分别处理两个传感器，具有两个独立的处理管线，并且不需要共同校准。

本文提出的方法从[16]开始，但提出了一种联合校准这两种设备的方法，并利用这两种传感器组合使用来提取特征的校准。此外，所采用的特征集还被更新为用于 LeapMotion 的更大特征集以及用于深度数据的新特征描述符

手势分类

第 4 节和第 5 节的方法产生八个不同的特征向量，其中四个用于 Leap Motion 数据，四个用于深度数据。每个矢量描述了一些关于执行的手势的相关线索，并且为了执行识别，已经使用了两种不同的分类方案，一种基于多类支持向量机分类器和一种基于随机森林的分类方案。有 8 个特征向量被分为两组，分别为 Vleap = [Fa，Fd，Fe，Fp]，其中包含从 Leap Motion 数据提取的所有特征，Vdepth = [Fl，Fρ，Fc，Fcc]信息。从这两个设备提取的特征矢量在图 8 中可视化地总结。每个矢量可以单独使用或与任何其他描述符一起使用。多个特征描述符的组合可以通过简单地连接对应于所选特征的矢量来获得。该方法的目标是将执行的手势分类为 G 类，其中一个用于所考虑的数据库中的每个手势。

第一种分类方案利用基于一种方法的多类 SVM 分类器[4]。在所采用的方案中，使用一组 G（G-1）/ 2 二进制 SVM 分类器来测试每个类彼此。它们中的每一个的输出被选择为针对某个手势的投票。对于测试集中的每个样本，具有最大投

票数的手势被选作分类的输出。具体而言，已经选择了非线性高斯径向基函数（RBF）内核，并且已经利用训练集上的网格搜索和交叉验证来调整分类器参数。让我们考虑一个包含来自 M 个用户的数据的训练集。 RBF 内核的参数（C，γ）的空间被规则网格划分。对于每对参数，训练集分为两部分，一部分包含 M-1 用户进行培训，另一部分与剩余用户进行验证和性能评估。该过程重复 M 次，更改验证集中的用户。选择平均得到最佳精度的几个参数作为网格搜索的输出。最后，SVM 已经在训练集的所有 M 个用户中用最优参数进行了训练。

另外我们还测试了利用 Random Forest（RF）[3]的第二种分类方案。每个树已经训练了训练集的随机抽样，留下了三分之一抽样矢量用于估计袋外误差。与 SVM 的 RBF 内核对不同,唯一的优化模型参数是每个节点中特征子集的大小 m。该参数控制树木相关性和每棵树的预测“强度”之间的折衷，并且可以通过分析袋外错误容易地找到该参数。森林的大小并不是一个关键参数，因为如果使用足够数量的树木，分类错误仍然相对稳定。在我们的情况下，

一个具有默认值为 m =√| F |的 100 棵决策树的随机森林与| F |数据集中特征向量的长度（当使用所有考虑的特征时，| F | = 435）。已经使用了由 Matlab 提供的随机森林分类器的实现。

最后，由于所考虑的矢量包含大量元素，我们还考虑使用特征选择方案来减少特征的数量并避免使用无用或冗余描述符。已经测试了三种不同的特征选择方案。第一种方法使用 F 分数方法[5]，即针对每个特征计算 F 分数，并且根据该量度选择最具有判别力的特征（即，具有大于预定义的 F 分数的特征阈）。为了产生两个具有 16 和 128 特征的子集，使用了两个不同的阈值。

第二种方案基于正向顺序选择（FSS）算法[1]。在这种情况下，从空集合开始，在每个步骤中，通过选择允许相对于前一步骤获得分类准确度的更大改进的特征，将新特征添加到所选择的特征（前述 SVM 分类器具有被用来评估分类的准确性）。

最后，测试了利用 Random Forest 的第三个特征选择方案。在这种情况下，采用[3]的方法进行分类，并估算出袋外误差。然后，为了测量各种特征的重要性，其中一个特征的值被置换，并且再次估计出包差错。对每个特征重复该过程，并且每个特征的重要性由置换后的袋外误差的归一化平均增加给出。这种方法详见[5]。所选功能的数量与之前的情况相同，以便进行公平比较。

实验结果

使用图 6 所示的设置已经获得了结果。已经使用 Leap Motion 设备和 Kinect 来共同获取与执行的手势相关的数据。 第一代 Kinect 深度相机由于其较大的漫射而被选中,但是其他任何深度相机（例如 Creative 的 Senz3D 或第二代 Kinect）

均可用于所提出的方法。 这两个装置已经使用第 6 节的方法进行了联合校准，并且在时间上同步。 一种软件同步方案已经被使用：其精度足以用于基于静态姿势的手势识别，如本文中考虑的手势

所考虑的手势数据集包含由 14 个不同的人执行的图 9 中所示的 10 种不同的手势。每个用户重复每个手势 10 次，总共 1400 个不同的数据样本。据我们所知，这是第一个包含深度数据和 Leap Motion 数据的数据库，可以在我们的网站 http://lttm.dei.unipd.it/downloads/gesture 上找到。为了计算结果，我们使用第 7 节的“一人一用”方法将数据集分成一列和一组测试集，也就是说，我们在训练集中放置了除一个用户以外的所有用户的数据测试设置来自剩余用户的数据。由于与单个用户（100 个样本）相关的数据量不足以可靠地评估性能，所以我们执行了 14 次完全独立的测试，每次测试集中的人（即，如图 10 所示）每次测试我们都使用了一套 13 人的火车和一套测试装备，剩下的只有一个人。对 14 次测试的结果进行了平均，以获得最终的准确度。请注意，这是一个比标准离开一次性方法更具挑战性的测试，因为它不仅保证了列车集中的数据与标准情况下的测试集中的数据不同，而且还确保列车集合不包含测试集中用户的任何样本。这意味着系统应该能够将测试集中用户的数据从用户学到的内容中分类出来，这些用户从与用户不同的用户那里学习。

让我们从 Leap Motion 设备开始。表 1 显示了使用第 7 节分类算法获得的来自该传感器数据的精度。指尖的 3D 位置给出了手指排列的非常好的表示，并且允许获得 81.5％的准确度。它们允许识别大部分手势，即使某些手势的识别不总是最佳的，因为从表 2 中的混淆矩阵可以看出。例如，手势 G2 和 G3 有时与手势 G1 混淆。这主要是由于 Leap Motion 传感器返回的误报有时会检测到手势 G1 中的凸起手指。

指尖距离特征可以获得大约 76％的准确度：他们能够识别大多数手势，但也存在一些关键问题，例如 G2 和 G3 容易混淆。该描述符的相关问题是来自 Leap Motion 的手方向估计的有限精度，其不允许指尖与对应角度区域之间的精确匹配（即，如果不能识别哪个手指已经被抬起单指被检测到）。其他两个功能的性能稍低。角度允许获得 74.2％的准确度，并且可以获得类似的结果（73％）

结论

在本文中，已经提出了用于 Leap Motion 的有效手势识别流水线，用于深度传感器及其组合使用。由 Leap Motion 提供的关于深度相机的数据的不同性质（即，更高级别但更有限的数据描述）提出了具有挑战性的问题，已经提出了有效的解决方案。一个专门的校准方案允许用深度传感器联合校准 Leap Motion。由第一个设备计算的有限点数使得这项任务非常具有挑战性，但是所提出的方案

允许获得足够好的准确度，以便共同利用来自两个设备的数据。已经为这两种传感器提供了几种不同的功能集。已经从 Leap Motion 中提取了四种不同类型的特征，而根据不同线索从深度数据计算了不同类型的描述符，例如与手形质心的距离，手形的曲率和手形的凸包的距离。还展示了如何利用 Leap Motion 数据来提高深度特征的计算时间和精度。

实验结果表明，Leap Motion 提供的数据即使不是完全可靠，也可以通过提出的一套特征和分类算法获得合理的总体精度。深度数据可以获得非常好的精确度，这些数据是对手形的更完整描述，特别是距离和曲率描述符可以获得几乎最佳的性能。即使改变分类算法或者使用特征选择方法来降低特征向量的维度，性能仍然非常好。

未来的工作将通过提出的设置和改进的方案来解决动态手势的识别问题，这些改进方案用于联合利用来自两个传感器的数据来检测和定位指尖

致　　谢

首先感谢我的指导老师翁仲铭老师，在过去的半年多的时间里，翁老师给我不少的指导，支持和鼓励，定期开会讨论我的论文进度，在遇到问题时为我解答疑惑，提供给我 Leap Motion 的实验设备并帮助我修改完善论文。在他的引导和帮助下，我顺利完成了本次毕业设计。翁老师一直鼓励同学进行创新创业项目实践，经常带领同学们参与一些前沿的科研项目，如虚拟现实，物联网等，为学校的科研创新做出了很大贡献。这一点也激励我在今后的工作和人生道路中要时刻关注时代潮流，对新兴的前沿科技保持敏感和热情，并不断进行创新。

感谢天津大学软件学院的各位老师，在 4 年的本科学习生活中，给了我很多的帮助和指导，为我打好了扎实的专业基础，让我逐渐掌握了软件工程领域的一些知识与技能。在这里特别感谢王征老师，毕重科老师，因为这两位老师的推荐信我顺利地申请到了美国游戏设计的硕士专业，在接下来的学习中我会更加努力。

感谢我的室友们，从大一到大四给了我许多的帮助和鼓励，相遇不易更是缘分，希望你们以后能获得自己想要的生活，越来越好。

感谢乐团的同学们，每周排练与你们的相遇总能让我放松自己，乐团里，得到的不只是音乐的陪伴，更是一段难忘的回忆与感情。

最后，感谢我的家人，一直无私的支持我，鼓励我，正因为有你们，我才能走到今天。谨以此文先给所有关心、支持和帮助过我的人，谢谢！