廣東工業大學

本科毕业设计（论文）

基于 Leap Motion 的手势交互系统的设计与实现

学　　院　计算机学院

专　　业　软件工程

年级班别　2013 级(1)班

学　　号　3113006215

学生姓名　陈永坤

指导教师　杨　卓

2017 年　5 月

基于 Leap Motion 的手势交互系统的设计与实现　　陈永坤　　计算机学

摘　　要

随着智能设备和人机交互技术的发展，在虚拟现实与增强现实这片竞争日益激烈的市场中，人们开始寻求更为自然的体感交互方式。一些科技公司逐渐将工作重心从硬件制造转移至软件技术的开发，通过软件技术的革新降低成本，以及转型内容制作厂商，来开拓一片新的蓝海。

裸手十指操作作为一种最原始的、最自然的交互形式，有着广阔的市场开发前景。Leap Motion 的出现，则是打开了这一扇大门。它是一款能够侦测人体手部运动数据的体感控制设备，不仅能用于扩展虚拟游戏世界内的操作方式，也为其他应用提供了新的交互思路。

论文将基于Leap Motion这一手势识别设备，探讨其部分关键技术算法，并结合Unity引擎开发出 PC 端上的一款三维体感交互应用。本应用将主题设计成与室内家具电器的体感交互。程序将会结合 MVC 框架，使用并扩展 Leap Motion 提供的 Unity 资源包进行开发；其次，还在 Unity 引擎中搭建若干个室内场景，并提供一套易于操作的用户界面，使用户能够在真实的物理环境下体验徒手操作虚拟的应用世界。

关键词：体感交互，Leap Motion，手势识别，Unity

Abstract

With the development of intelligent devices and human-computer interaction techniques, in the increasingly competitive market of virtual reality and augmented reality, people began to seek a more natural way of somatic interaction. Through the innovation of software technology to reduce costs, and the transformation of content producers, some technology companies are shifting their focus from hardware manufacturing to software technology development to open up a new blue ocean.

As one of the most primitive and natural forms of interaction, the ten finger operation shows a great market prospect. And the emergence of Leap Motion has opened this door. It is a somatic controller that detects the movements of human hands, which not only can be used to extend the mode of operation in virtual game world, but also provides a new thought of interaction for other application.

Based on Leap Motion, the gesture recognition device, this thesis studies some of its key technical algorithms and uses the Unity engine to develop a 3d somatic interaction application on the PC. The application will be designed to interact with indoor furniture appliances. This program will be developed by combining the MVC framework, using and expanding the Unity resource packages which are provided by Leap Motion. Then, it will built several indoor scenes and provide an easy-to-use user interface, so that users are allowed to experience it's virtual world with their hands and fingers in the real physical environment.

Key Words： Somatic interaction, Leap Motion, Gesture recognition, Unity

目录

1 绪论

1.1 人体运动捕捉

人体运动捕捉分为三种：躯体捕捉、手势捕捉和脸部捕捉。其有如下定义：一个通过在时域上跟踪关键点的运动来记录生物动作，并转换成可通过数学表达及合成的三维动作的过程[1]。

1.1.1 光学运动捕捉设备

人体运动捕捉根据所采用的设备可分为：机电式运动捕捉、电磁式运动捕捉和光学运动捕捉。其中，光学运动捕捉是较精确的一种方式。光学运动捕捉利用多个摄像机从不同角度对同一目标进行拍摄，再通过软件计算目标点的坐标，并利用计算机视觉原理进行三维重建，从而得出目标的动作数据[2]。

1、Kinect

Kinect 是微软推出的一款应用于 Xbox 360 主机的周边外设。它是一种 3D 体感摄像机，包含了三个镜头，可捕捉到三维空间人物躯体的各种运动，配合软件可跟踪到人体数十个部位的动作，同时还支持人脸识别和语音辨别。但 Kinect 只能在室内使用，帧速仅有 30 帧/秒，容易发生延迟，且无法识别出一些细节动作如手指动作等。

图 1.1 Kinect 设备

2、RealSense

RealSense 是因特尔推出的 3D 摄像头，它具有和 Kinect 相似的硬件结构。根据工作原理，RealSense 可分为两类：RealSense F200/SR300 和 RealSense R200。前者采用的技术也是与 Kinect 1.0 相同的 Light Coding 技术，但 RealSense 投射的是高速变化的编码

散斑，其帧率更高。后者原理则与 Kinect 不同，采用的是静态散斑辅助的双目视觉技术，不完全受限于散斑的有无，在室外等有其他近红外成分存在的环境下依旧可以工作。

RealSense 能够高精度地识别面部表情、肢体动作以及语音命令，同时支持手势动作识别。

图 1.2 RealSense 设备

3、Leap Motion

Leap Motion 控制器是由同名的 Leap Motion 公司推出的一款利用摄像机及主动红外线对人体手部细节动作进行跟踪的高精度设备。它的硬件结构十分简单，仅依靠普通的灰阶摄像头和红外光源，且相比较前面提及的两者而言，Leap Motion 可以在保持硬件设备不更换的情况下，通过升级软件驱动来增强其功能，因此是一种非常廉价的体感交互设备，再加上其精度之高，以及其最高帧率可达到 240 帧/秒，在开发运动捕捉应用的尝试中具有很高的优先级。

1.1.2 基于 Leap Motion 的手势追踪

当前，大多数体感设备如 Kinect 等对近距离人手精细动作的识别都不理想。不同于这些设备，Leap Motion 传感器的追踪目标仅为人手信息，通过对手部信息包括指尖、指关节、手掌等的位置、方向向量及法向量等信息的追踪和识别，精确提取手势特征，然后直接传输给计算机，由驱动程序去完成后续复杂的处理[3]。Leap Motion 硬件结构简单、追踪量少，因而可以做到高速侦测和构建手势模型。因其对手势的识别精度之高、帧数之快以及传感器的价格亲民，非常适合作为论文在探究体感交互时所采用的设备。

1.2 Leap Motion 工作原理

Leap Motion 是一款由美国 Leap 公司所出的专用于跟踪、捕获人体手部运动数据的

体感控制器，它提供一种能够识别手势动作并进行手势交互的解决方案[4]。

1.2.1 硬件设备

Leap Motion 是一个基于双目视觉的小型手势识别设备，配合一条专用的 USB 数据线，将其连接到 PC 机或虚拟现实头盔，即可实时反馈传感器所捕获并构建的手部数据帧到处理器。

图 1.3 Leap Motion 设备

1.2.2 多角成像技术

Leap Motion 采用基于双目视觉的多角成像技术，在设备上安装有 3 个 LED 红外光源和 2 个灰阶摄像头传感器，可以一次性感知所测物体的所有像素，并使用预置算法进行处理[5]。

当两个摄像头对当前环境进行拍摄，会得到两张具有不同视角的照片，再根据摄像头的各项已知参数和相对位置，以及相同物体在画面中的不同位置，即可计算出物体实际距离摄像头的景深深度[6]。

景深的具体计算方法采用的是“三角测量法”[7]，即利用三角形相似原理计算出单点目标在双摄像头下的景深 z，如图 1.4 所示。

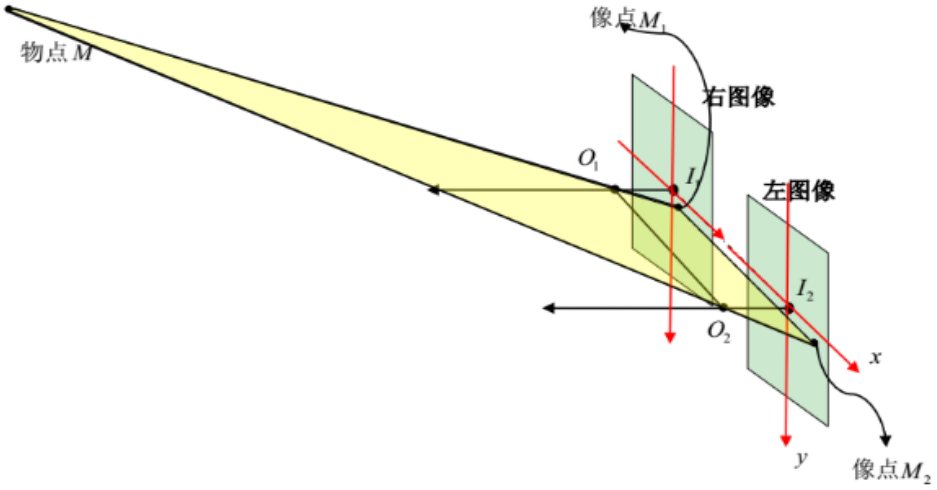


图 1.4 双目视觉计算景深的三角测量法

假设目标在处于 M 位置，M 到左右摄像头光心 O_1、O_2 连线距离即为所需景深 z，而光心后的左右成像面中心点 I_1、I_2 连线为基线距离，可由摄像头参数求得。根据三角形相似，有：

$$\begin{cases} \dfrac{z}{O_1I_1} = \dfrac{MO_1}{O_1M_1} \\ \dfrac{O_1O_2}{M_1M_2} = \dfrac{MO_1}{MM_1} \end{cases}$$

其中已知量有焦距 f（即 O_1I_1、O_2I_2），光心距离 d（即 O_1O_2、I_1I_2）；可测得量为 M 在左右成像面的点 M_1、M_2 与成像面中心 I_1、I_2 的水平偏移量 a 和 b。代入上述式子，最终可得到：

$$\frac{z}{f} = \frac{d}{|b-a|}$$

根据上式，即可求得单点景深 z。得到景深，再次根据三角形相似计算可得到目标点的水平位置 x 和竖直位置 y。

1.2.3 特征点匹配

Leap Motion 在形成深度图像时，仅采用若干个特征点进行手型匹配，在获得这些特征点的信息后，利用 IK 算法即可推算出整个手部模型。若部分特征点被遮挡或干扰而无法侦测到，Leap Motion 会根据内部构建的手模型预估特征点的位置[8]。由于避免了匹配手部所有点，Leap Motion 的处理时间能够减小到 10ms 以内。而在必要时，Leap Motion 的侦测帧率甚至可从 120 帧/秒提升至惊人的 240 帧/秒[9]。

1.2.4 坐标系统

Leap Motion 传感器以其中心为原点建立一个右手笛卡尔坐标系，X 轴与传感器长边平行，指向右方，Y 轴垂直屏幕指向上方，Z 轴背离屏幕指向操作者的方向。若将传感器与头戴设备连接，则 Y 轴指向前方，Z 轴垂直向上。

Leap Motion 传感器的探测视野在屏幕上方形成一个倒四棱锥空间结构，视角约为 150°，有效范围约为 0.3 米到 0.6 米，检测精度可达到毫米级。

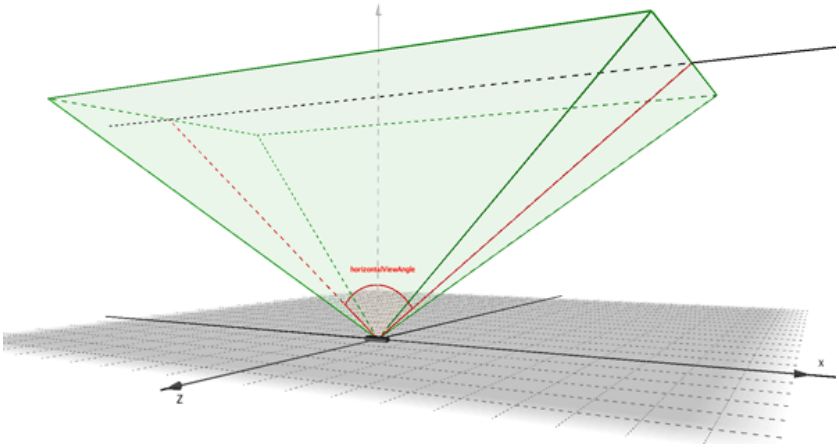


图 1.5 Leap Motion 探测视野

1.2.5 运动数据追踪

Leap Motion 所追踪的具体对象为手、手指、手势以及工具，设备定期发送所侦测到的目标对象的方向、位置等数据，每份这样的数据都被保存到帧（frame）当中，并为每个被其检测到的对象分配唯一 ID，只要设备能够探测到对象，ID 便保持不变。假如设备失去目标，而当目标再次出现时，将重新分配新的 ID。

Leap Motion 视野中的实体发生位移、旋转等变化，都会引起帧信息的变化，通过将当前帧与之前帧的数据进行对比，形成运动信息。

1.2.6 手部模型

手部模型提供被检测手的 ID、位置等其他信息，包括连接这只手的手指和手臂等等。

即使在部分手探测不到的情况下，Leap Motion 也能结合可见部分、内部手部模型以及先前的运动信息来预判该不可见部分接下来最有可能的跟踪数据[10]。

1、手

手对象包含用于描述一只手物理特征的各种属性。

表 1.1 手对象属性表

Palm Position	手掌中心的坐标
Palm Velocity	手掌运动的速度
Palm Normal	垂直于手掌平面的向量
Direction	从掌心指向手指的向量
Sphere Center	适应手掌弧面的一个球的球心
Sphere Radius	适应手掌弧面的一个球的半径

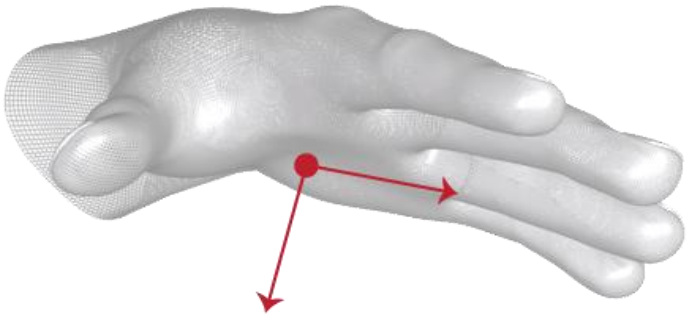


图 1.6 手掌向量示意图

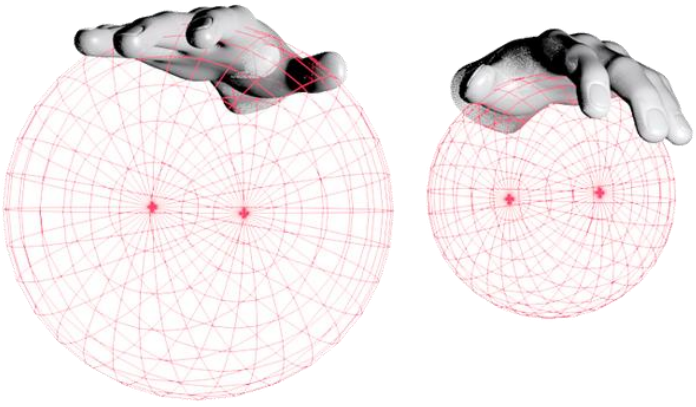


图 1.7 手掌弧面球体示意图

2、手指

Leap Motion 可识别手上每根手指的信息。如果其中某根手指无法被检测到，Leap

Motion 会根据最近观察数据与手掌的解剖模型推测出该手指的特征。

3、工具

Leap Motion 除了可以检测手指外，也可以检测手持的工具，工具被认为是比手指更细、更长而且更直的物件。

图 1.8 工具示意图

手指与工具具有一些相同的属性，Leap Motion 将它们统一称为端点对象（Pointable Object）。

表 1.2 端点对象属性表

Length	可见长度
Width	平均宽度
Direction	与物体指向相同的单位向量
Tip Position	Leap Motion 坐标系下的位置
Tip Velocity	运动速率

图 1.9 端点对象示意图

1.3 论文研究意义

在 VR/AR 火爆的时代，体感交互作为一种新型的交互方式，突破了传统键盘和鼠

标的束缚，让人机操作更加直观、立体化，也让用户得到更为深刻直接的交互体验。虽然市面上的体感交互设备层出不穷，但大多价格昂贵且携带不便。论文通过对 Leap Motion 这一手势交互设备进行初探，并尝试为其设计应用场景，来寻求手势操作这种更自然、更轻便的体感交互方式在计算机应用中的可行性和前景。

1.4 论文主要内容

论文围绕一个 Leap Motion 手势交互系统的开发为基础展开。

绪论大致介绍了人体运动捕捉及几种常见的捕捉设备，阐述了选取 Leap Motion 作为开发设备的原因，并对 Leap Motion 的工作原理进行了较详细的分析陈述。第二章介绍本次开发所需要用的主要软件工具。

第三、四章是论文的重点内容。第三章策划了手势交互系统的主题内容和主要功能，设计、搭建所需的应用场景，并设计了一套用户界面。第四章着重从代码编写角度进行详细的论述，介绍了本次开发所涉及到的 Leap Motion 资源包的 API，程序所选择的主体框架和几种设计模式，以及各功能的主要算法实现。第五章是对发布后的应用程序的测试与分析。最后一章则是对该次毕业设计课题的一个总结。

2 开发工具与环境

2.1 Unity 开发引擎

Unity 是由丹麦 Unity Technologies 公司所开发的一款完整的跨平台游戏开发引擎，可以制作出 3D 或 2D 的优秀游戏，其编辑器可运行在 Windows 和 Mac OS X 下，并支持包括 PC，IOS，Android，Web，XBOX 等多个平台发布[11]。

引擎提供了容易上手的操作界面，内置实用的 UI 系统，并支持物理引擎、多人网络连线、地形处理和角色动作编辑等各项功能，可大幅度降低游戏的开发门槛，缩短游戏的开发时间，并达到降低制作成本的目的，目前已有 80%的手机游戏使用了 Unity 引擎作为开发工具。

Unity 不仅只限于游戏行业，在 3D 设计、工程模拟和虚拟现实等方面也有着广泛的使用。国内常用 Unity 开发虚拟现实三维展示甚至是军事演练模拟等项目。

图 2.1 Unity5 Logo

2.2 Visual Studio 集成开发环境

Microsoft Visual Studio（以下简称 VS）是美国微软公司所推出的一套完整的开发工具集，其包含了软件生命周期所需要的大部分工具，如集成开发环境、代码管理工具和 UML 工具等，其发布出来的产品适用于绝大部分平台，包括 Windows、Windows Phone、Android、IOS、Web。

Visual Studio 是当前 Windows 平台下最流行、最强大的集成开发环境，并可通过安装扩展插件进行更多功能的定制。目前，VS 通过自带扩展便可与 Unity 引擎无缝对接，并支持断点，成为取代 MonoDevelop 编辑器的强大利器。

3 Leap Motion 应用设计

3.1 主题设计

为了较大程度地利用 Leap Motion 及其提供的插件包资源，将开发一个 3D 场景模拟的桌面应用，用于体验 Leap Motion 所带来的手势交互。基于现有资源和网络资源，经反复选材，本应用的主题最终定为室内家居模拟控制。

为了增加该主题的趣味性，本应用将在原有的“家居控制”模式（以下简称“常规模式”）上进行功能的扩展或替换，重点开发出一个更加有趣的“惊悚鬼屋”模式（以下简称“鬼屋模式”）。

3.2 基本功能设计

3.2.1 菜单界面

运行应用后应先进入菜单界面，在菜单界面上有多个基本功能选项供用户操作。

1、操作指引：展示直观的图文介绍，作为使用本应用的操作教程。

2、模式选择：可供用户选择进入常规模式或者鬼屋模式。

3、退出程序：会二次确认用户是否退出应用，防止误操作。

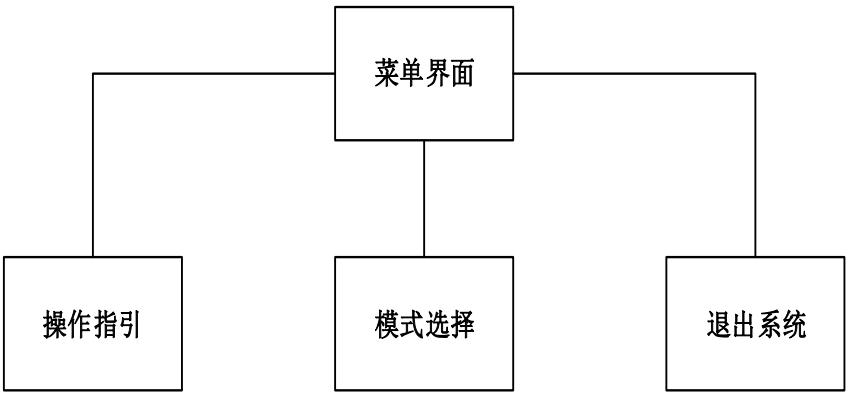


图 3.1 菜单界面功能设计

3.2.2 主题场景

为契合“家居”“鬼屋”等关键字，本应用将搭建若干个室内场景，场景内布置若干可供交互的家居和电器等物品。

1、客厅：从菜单界面进入后的第一个场景，布置“电视”，作为该场景的主要交互

对象。

2、浴室：从走廊进行跳转进入，布置“浴缸”，作为该场景的主要交互对象。

3、厨房：从走廊进行跳转进入，布置“顶灯”、“炉灶”和“冰箱”，作为该场景的主要交互对象。

4、走廊：从客厅进行跳转进入，作为“客厅”、“浴室”和“厨房”场景的中间过渡通道。

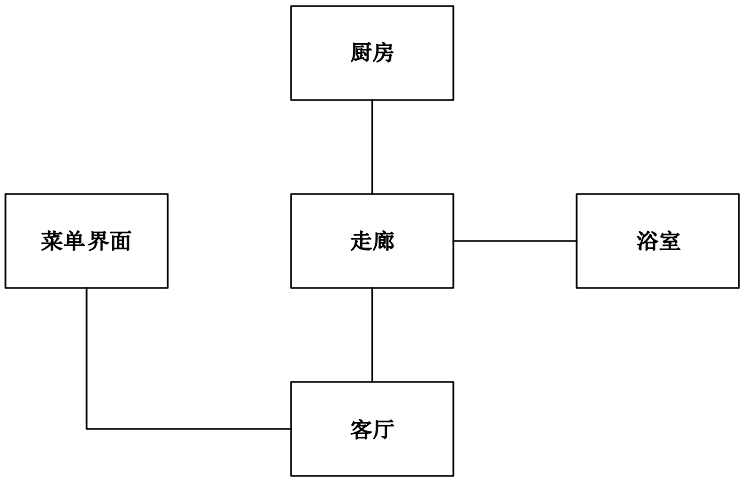


图 3.2 应用场景衔接图

3.2.3 双主题模式

用户选择常规模式或鬼屋模式，可通过同样的控制操作，获得不同的视听体验，具体体现在以下几个方面。

1、灯光：常规模式下，场景均选用暖色调的环境灯光，而鬼屋模式下，当触发了事件时，改用以绿色为主色调的灯光来烘托出一种可怕的气氛。

2、素材：鬼屋模式下选用一些切合氛围的音频、视频和模型素材，如“尖叫声”、“鬼火”或“骷髅头”等。

3、指引：常规模式下，各主要场景内在首次进行某种操作时会有相应功能的操作指引提示。

3.2.4 镜头转换方式

由于 Leap Motion 的交互系统有很大一部分是建立在接触式交互之上的，同时又因为 Unity 以米为单位，而 Leap Motion 的移动是以厘米为单位，故场景中一个微小的镜

头移动对 Leap Motion 而言都是相当大的变化。

为了方便控制，在一个静态场景，最好采用固定视角的方式来设置摄像机，即镜头只在预设的若干个点进行移动和旋转。这样，在搭建场景的过程中，便能忽略场景的空间尺寸，各个模型之间的相对距离可以离得较远，只需在最后选取合适的点作为摄像机的路径点即可。

3.3 构建 3D 场景

3.3.1 Leap Motion 对象配置

新建 Unity 场景，作为应用初始化的入口场景。将下载好的 Leap Motion 核心包和其他资源包一并导入项目中，并在 Hierarchy 面板添加摄像机。

在主摄像机下新建一个物体作为 Leap Motion 对象作为其子物体，这样 Leap Motion 便会跟随摄像机一起移动和旋转。在该物体上还应挂载三个组件：LeapHandController、LeapServiceProvider 和 HandPool。具体参数设置如图。

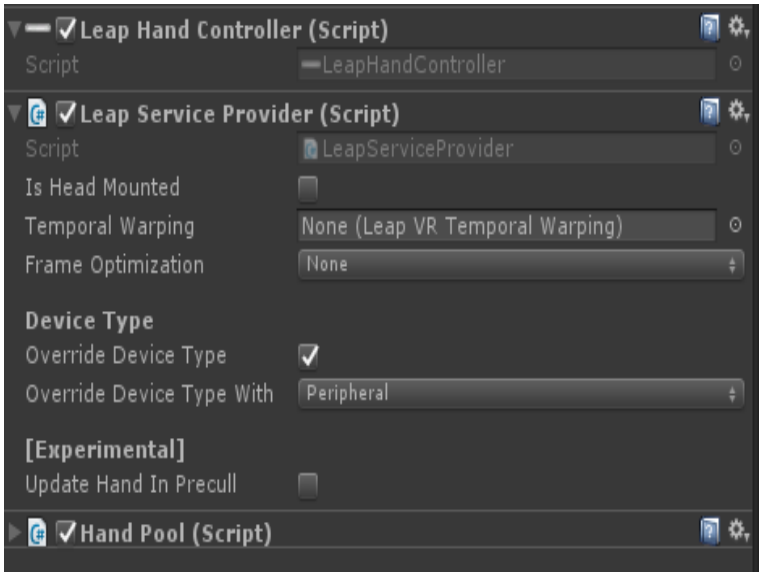


图 3.3 Leap Motion 对象参数配置

在 Leap Motion 物体下再新建一个空物体，作为手部模型的父对象，并从资源包中选择合适的手部模型.prefab 预制体，拉取到父对象之下。同时，在 Leap Motion 物体的 HandPool 脚本上进行引用。

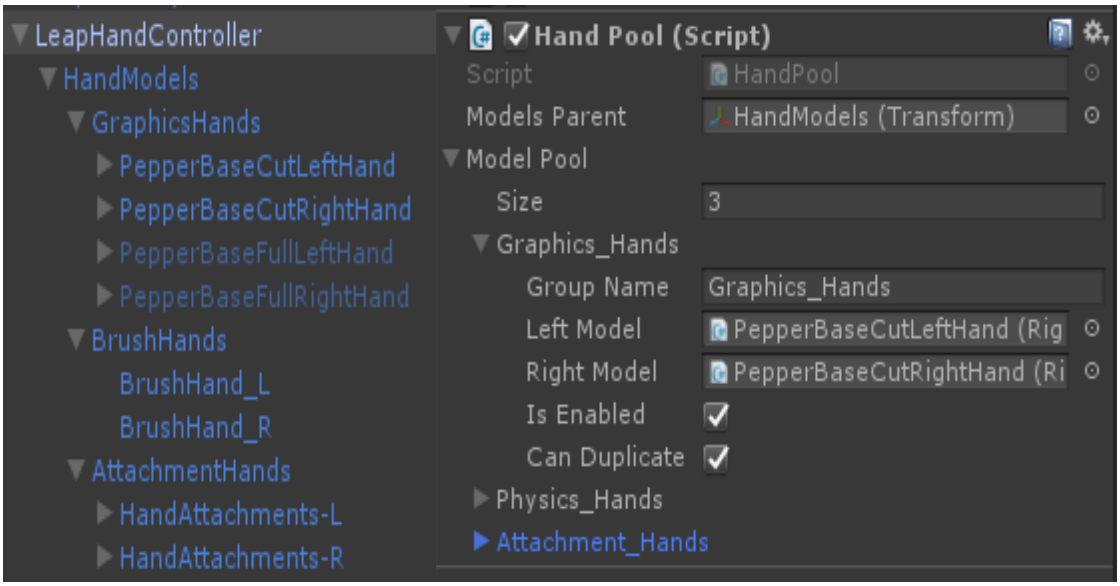


图 3.4 Hand Pool 对象参数配置

从资源包中分别再找出名为 LeapEventSystem 和 InteractionManager 的预制体，挂载在主摄像机下面，即完成了 Leap Motion 对象的搭建。

3.3.2 客厅场景搭建

创建一个作为客厅场景根结点的空物体，此后添加的物体都作为该空物体的子物体。

在场景中添加若干各 Quad（四边形）作为地板和墙壁的面片，并为其附加材质球。按照一定布局添加沙发、茶几、空调、电视以及灯光等物体，最终效果如图。

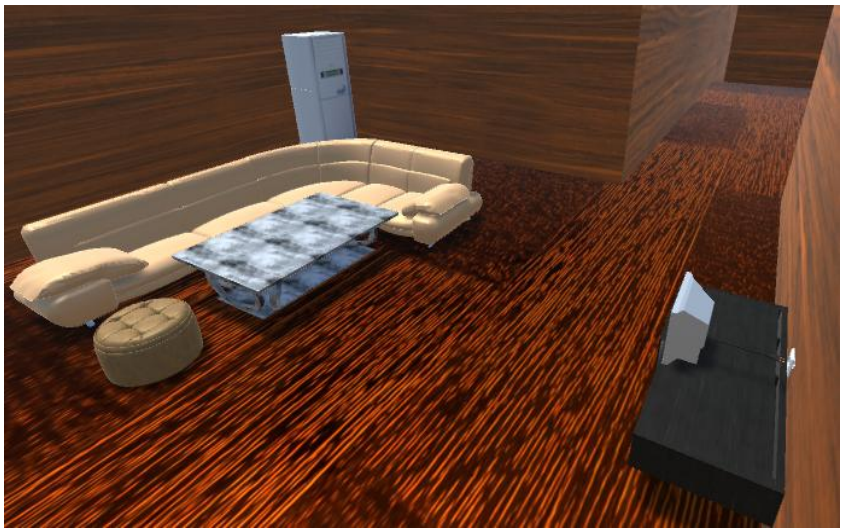


图 3.5 客厅场景预览图

3.3.3 走廊场景搭建

创建一个作为走廊场景根结点的空物体，此后添加的物体都作为该空物体的子物体。

在场景中添加地板和墙壁，参考“3.3.2 客厅场景搭建”。走廊的布局较为简单，最终效果如图。

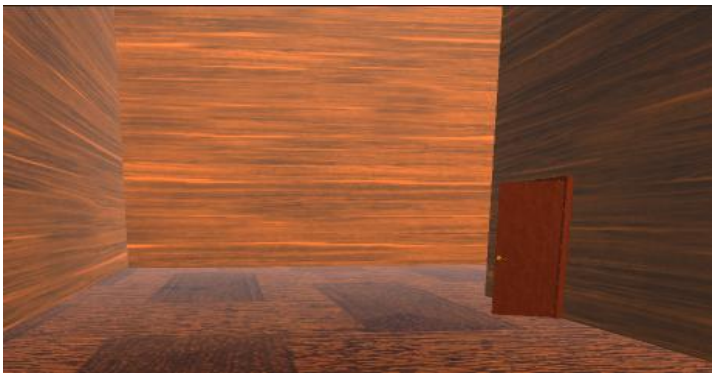


图 3.6 走廊场景预览图

3.3.4 浴室场景搭建

创建一个作为浴室场景根结点的空物体，此后添加的物体都作为该空物体的子物体。

在场景中添加地板和墙壁，参考“3.3.2 客厅场景搭建”。在墙角处放置一个浴缸模型，添加水体并隐藏，最终效果如图。

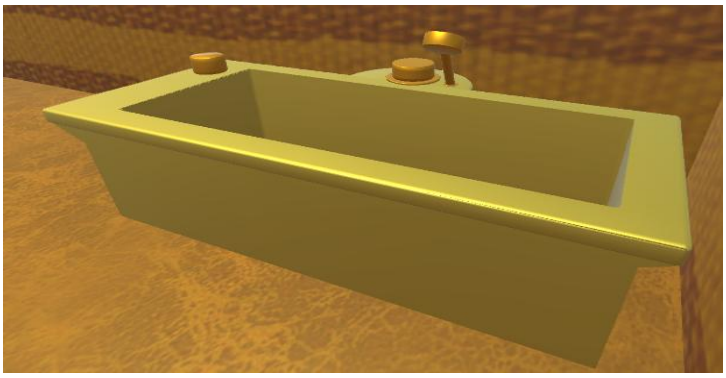


图 3.7 浴室场景预览图

3.3.5 厨房场景搭建

创建一个作为厨房场景根结点的空物体，此后添加的物体都作为该空物体的子物体。

在场景中添加地板和墙壁，参考“3.3.2 客厅场景搭建”。按照一定布局添加餐桌、顶灯、炉灶和冰箱等物体，最终效果如图。

图 3.8 厨房场景预览图

3.3.6 光照烘培

将以上所有场景添加为预制体，然后选择“Window/Lighting”打卡光照设置窗口，选中“Lightmaps”选项卡，取消 Auto 选项的勾选，点击“Build”按钮手动生成光照贴图。

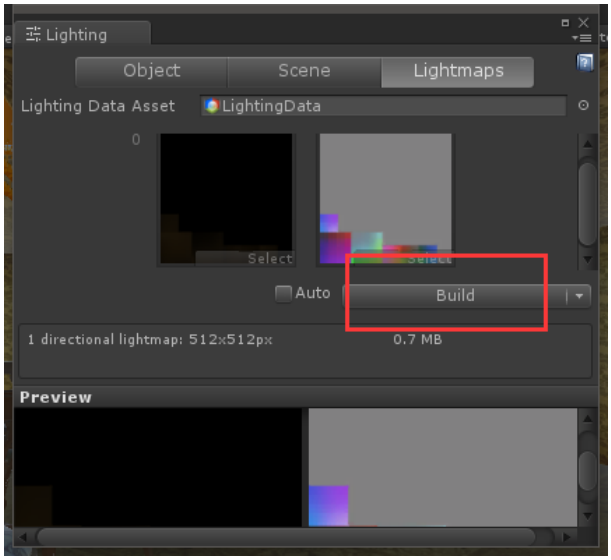


图 3.9 灯光烘焙设置

3.4 界面设计

程序的界面使用引擎自带的 UI 系统 UGUI 来实现。UGUI 支持可视化开发，同时能进行 2DUI 和 3DUI 的开发，是大名鼎鼎的 NGUI（商业游戏中用得最多的 UI 插件）的作者的又一杰作，并且也是开源的系统。相比于 NGUI，使用 UGUI 进行界面开发，将会更方便、更快捷。

3.4.1 UI 效果要求

各个界面采用的 UI 元素统一风格，使用同一套贴图素材，并且 UI 所选用的素材具有一定的立体效果，如按钮素材的边界上有一层下凹效果。

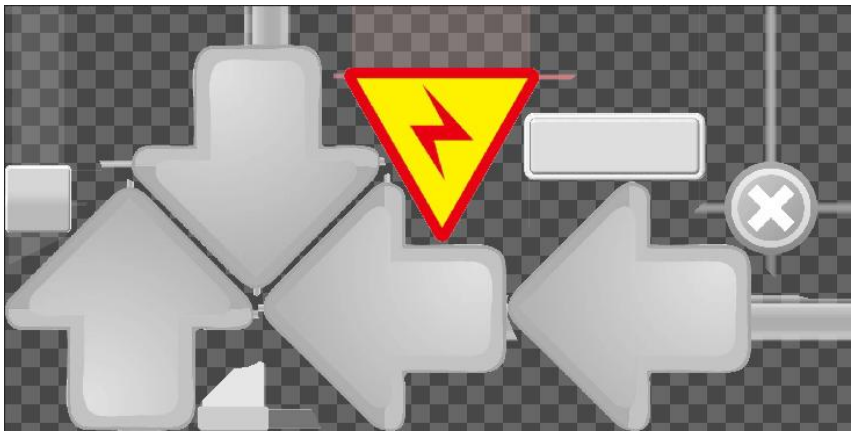


图 3.10 UI 图集预览

为进一步体现立体感，按钮在按下和弹起时应有明显的缩放效果，利用 Leap Motion 提供的 UIInput 资源包中的 CompressibleUI 脚本可实现该效果。

新建一个 Button 并配上合适的贴图作为按钮，调整 Color 属性的 Alpha 值为 30（其他按钮均使用这个配置），在该 Button 下添加一个 Image，使用和 Button 相同的贴图，但其 Alpha 值为 255，在该 Image 下再添加一个 Text。最顶层的 Button 需要挂载 CompressibleUI 脚本，然后将 Image 和 Text 分别拖到 Floating UI Layers 内，设置 Image 的 Shadow 为 Button。为触发缩放效果，在 Button 上添加 EventTrigger 组件，分别配置按下和弹起的事件为 CompressibleUI 的 Retract 和 Expand 方法。

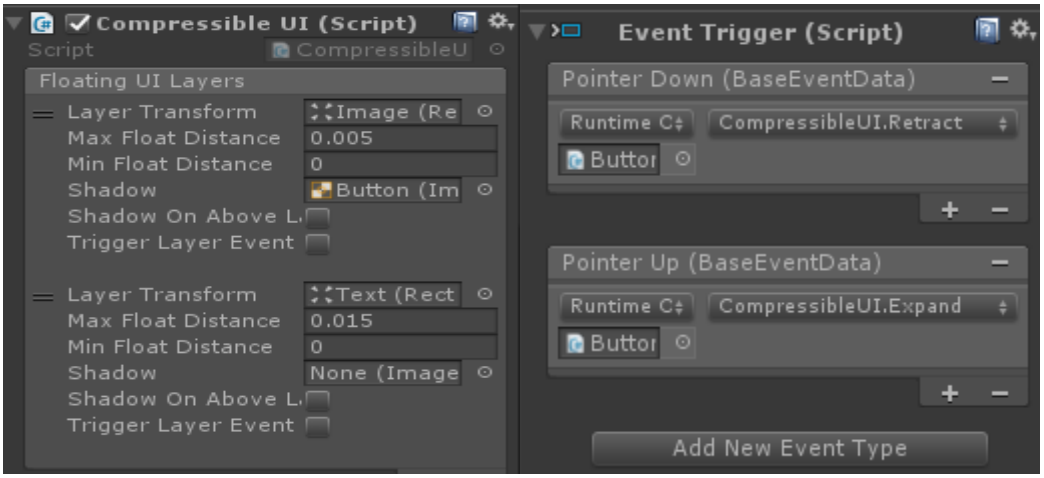


图 3.11 可按压 UI 参数配置

3.4.2 画布属性

UGUI 的 UI 元素都是放在一个名为 Canvas（画布）的物体之下。由于 Leap Motion 的 UI 系统是接触式 UI，为了能触摸到 UI，需要调整各 UI 在摄像机视野内的景深位置。通过直接操作画布属性，将同时影响在该画布下面的所有 UI 元素。

在画布的 Canvas 组件上选择摄像机模式，该模式下的 UI 将一直呈现在摄像机正前方的 Plane Distance 距离位置，距离够近时能够有 3D 触碰效果，且不会因摄像机位置或角度而偏离中央视野。当 Plane Distance 属性的值不大于摄像机 Clipping Planes（裁剪面）的近距离时，UI 界面将会看不到。为了使用 Leap Motion 的操作空间，Plane Distance 的值设为 0.45~0.55 范围内。

为了多个不同分辨率的显示器都能正确显示 UI，画布需要根据开发长宽比和实际长宽比做出拉伸或收缩。在画布下的 CanvasScaler 组件选择“Scale With Screen Size”模式，匹配模式选择适配宽度或高度，其值拉到 Width 端（0 值），即按照屏幕分辨率宽度进行缩放。参考分辨率填如开发时屏幕所用的分辨率，此处是 1400*900。

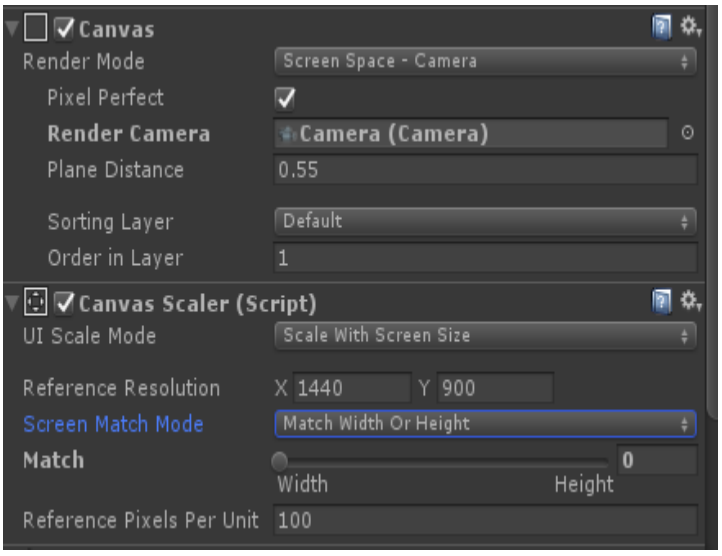


图 3.12 画布参数配置

3.4.3 菜单界面

菜单界面应符合本应用的主题，采用房屋图片作为菜单的背景图片。为吸引用户进行体验，着重强调鬼屋模式，将界面的主色调定为绿色调。为进一步烘托一种阴森恐怖的气氛，在屏幕范围内增加不断浮动的粒子特效。

菜单界面上所使用的 UI 元素严格遵循规定，每个功能面板都需要有返回按钮。面板为半透明，防止遮挡了背景图片的中央主要部分。

菜单的每个面板上的文字在颜色上应有所区分，且本应用只能使用同一种字体。

图 3.13 菜单一级界面设计

在“操作指引”功能面板上，使用图文并茂及拖拽滚动的方案来呈现。

图 3.14 操作指引二级界面设计

3.4.4 场景界面

应用内各场景使用 UI 按钮进行镜头切换，按钮需根据分场景实际位置进行布局。每个按钮都要适应人手以及 Leap Motion 的可操作范围，既不能太拥挤也不能太难触碰到。在切换主要场景的按钮上还应有文本显示。

图 3.15 主场景 UI 设计

鬼屋模式下，当触发场景事件时，UI 的主体颜色切换为和场景环境相符合的绿色调，UI 文字改为红色。当场景恢复正常时，UI 的颜色也一同恢复。

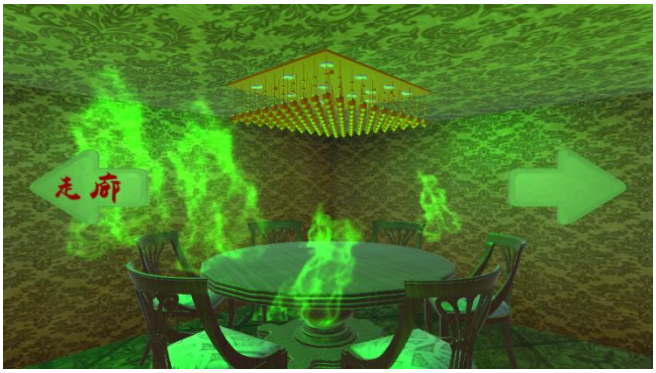


图 3.16 主场景鬼屋模式 UI 设计

4 详细设计与编码

4.1 集成 Leap Motion

Leap Motion 提供了适用于 Unity 应用开发的资源包，其中包括了核心代码脚本和通用模型，通过导入并使用插件资源，可以快速构建出 Leap Motion 应用。

4.1.1 Leap Motion 插件资源

Leap Motion 插件资源包括了以下五个部分：

Core Assets：核心功能包，是使用 Unity 开发 Leap Motion 应用的必要插件。

Interaction Engine：交互引擎模块，是一套支持与场景中物体进行交互的更简便的方案，例如进行抓取、抛掷等动作，要求手部与物体直接接触。

Attachments Module：用于控制场景中物体或 UI 各项属性变换的功能模块，如旋转、缩放、颜色渐变或等，不要求手部与物体直接接触，符合控制条件即可进行插值变换。

Hands Module：手型资源包，内含 Leap Motion 提供的一些手部模型，包括可见的骨骼驱动的人手、程序生成的多边形手、组件构成的机械手以及不可见的物理手等。

Detection Examples：使用 Detection Utilities（核心包中监测具体动作行为的脚本集）开发的 Demo。

本应用开发过程中将会使用到核心包、交互引擎及手型资源包，其余资源可作为参考下载。

下载地址：https://developer.leapmotion.com/unity

4.1.2 主要 API

LeapServiceProvider：LeapProvider 唯一的具体类，其中保留着由传感器传输到应用的帧数据。

表 4.1 LeapServiceProvider 类主要接口

方法或属性	解释
Controller GetLeapController ()	返回 Leap Controller 实例
bool IsConnected ()	是否已正确连接 Leap Motion 设备
bool _isHeadMounted	是否头戴式
Frame CurrentFrame	当前帧数据

HandPool：手部模型的对象池，包含一个 ModelGroup 列表，可配置多种不同的手部模型，它们将会被同时激活或禁用。

ModelGroup：一组手部模型，包括左手和右手。

表 4.2 ModelGroup 类主要接口

方法或属性	解释
string GroupName	组别名称
bool CanDuplicate	是否可以复制新的实例并使用
bool IsEnabled	是否启用这组手模型
IHandModel LeftModel	左手模型
IHandModel RightModel	右手模型

InteractionManager：交互引擎管理器，用于设置接触式交互的各项参数。

表 4.3 InteractionManager 类主要接口

方法或属性	解释
LeapProvider _leapProvider	所使用的 LeapProvider 实例
HandPool _handPool	所使用的 HandPool 实例
string _ldatPath	位于 StreamingAssets 文件夹下的 Ldat 文件路径
string _brushGroupName	HandPool 下 BrushHand 手模型所在组别名
bool _contactEnabled	是否启用接触，禁用将无法对添加了 InteractionBehaviour 组件的物体产生作用
bool _autoGenerateLayers	是否自动创建交互层

InteractionBehaviour：可交互物体，由交互引擎驱动。

表 4.4 InteractionBehaviour 类主要接口

InteractionManager _manager	所使用的 InteractionManager 实例
InteractionMaterial material	所使用的 InteractionMaterial 文件
bool IsBeingGrasped	是否正被抓取中。
bool isKinematic	是否启用运动学
Rigidbody rigidbody	所使用的刚体组件
bool useGravity	是否启用重力
Action<Hand > OnHandGraspedEvent	抓取物体事件
Action<Hand > OnHandReleasedEvent	释放物体事件
InteractionManager _manager	所使用的 InteractionManager 实例

Detector：手势监测器基类，在其派生类内实现监测某一具体的简单手势的方法。

表 4.5 Detector 类主要接口

方法或属性	解释
void Activate ()	手势匹配成功后调用
void Deactivate ()	手势匹配失败后调用
UnityEvent OnActivate	手势匹配成功后一次性触发的事件
UnityEvent OnDeactivate	手势匹配失败后一次性触发的事件
bool IsActive	手势是否匹配成功，成功则处于激活状态

DetectorLogicGate：手势监测逻辑门，通过组合多个简单手势，实现对复杂手势的监测，只有组合逻辑的结果最终为真时才会激活。自身继承自 Detector 类，可以进一步组合，形成激活条件高度复杂的手势。

表 4.6 DetectorLogicGate 类主要接口

方法或属性	解释
void AddAllSiblingDetectors ()	添加同一游戏对象上的其他监测器
void AddDetector (Detector detector)	添加指定监测器
void RemoveDetector (Detector detector)	移除指定监测器
bool AddAllSiblingDetectorsOnAwake	是否默认添加同一游戏对象上的其他监测器
LogicType GateType	逻辑门类型，包括与门和或门
bool Negate	逻辑门结果输出类型，是否取反

4.2 代码整体框架搭建

4.2.1 MVC 框架简介

MVC 是一种软件的架构模式。它把软件系统划分成模型（Model）、视图（View）和控制器（Controller）三个部分，从而将应用程序的输入、处理和输出分离开来。

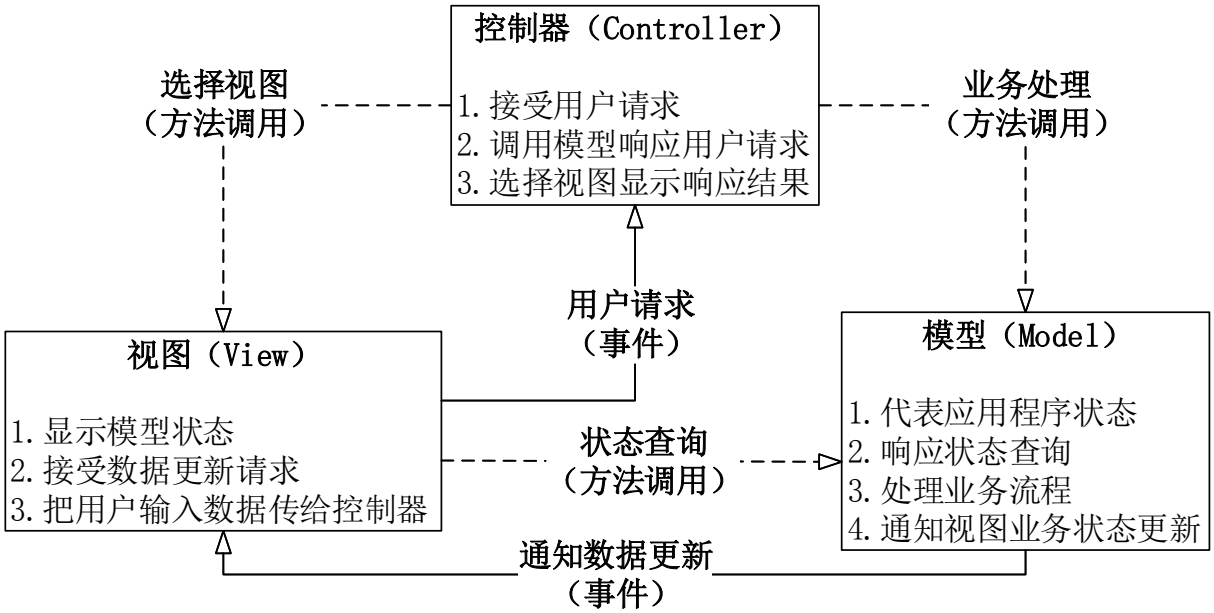


图 4.1 MVC 流程示意图

Model：存储应用程序的核心数据。

View：用界面来显示 Model 数据。

Controller：处理用户输入。

模型、视图和控制器，再加上用户，则成为了程序流程的一个闭环。用户通过视图进行输入操作，向控制器发出请求事件，控制器根据用户请求调用相应方法更改模型数据和状态，模型发生变更后通过事件去通知相关的视图更新界面。这样，便完成了从用户输入开始到数据处理与存储再到返回请求结果的整个过程。

在本程序开发中使用 MVC 模式，旨在提高代码的复用性和可维护性，同时，会进行一定程度的优化以减少使用该模式所带来的负面作用。

4.2.2 观察者模式

模型存储着系统数据，当模型中的数据发生变更时，需要通知视图进行更新。因此，模型宜采用观察者模式实现通知。

观察者模式，又称为订阅-发布模式或模型-视图模式，是常用的软件设计模式之一。当被观察对象的状态发生变更时，会去通知它的所有观察者进行更新。

观察者模式常用“注册——通知——撤销注册”的形式实现。

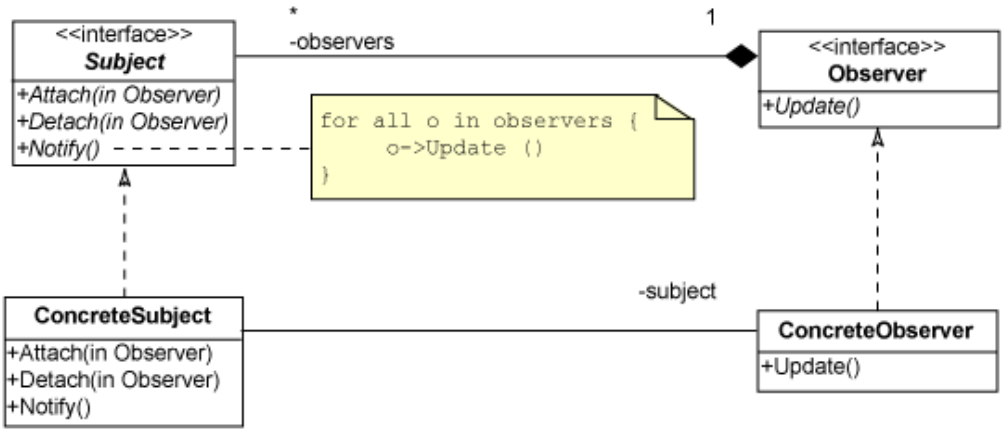


图 4.2 观察者模式 UML 类图

被观察对象保留一份观察者列表，一个被观察对象可以有多个观察者，观察者将自身注册到被观察对象当中。当被观察对象发生了某种变化后，遍历列表内所有已注册的观察者，通知它们更新。

4.2.3 框架实现思路

实现 MVC 框架时结合观察者模式，用户界面作为观察者，而业务数据则是被观察对象。利用 C#委托事件的特性，能够简化观察者模式的设计。通过实现接口方法，完成对事件的注册、调用和注销。由于 C#类实例方法只归属于类的具体实例，指定了实例方法便指定了某个实例，一个委托事件中可注册多个具体实例的多个实例方法，利用这个特性，被观察对象可只保留一个事件清单，不必保留观察者列表，便能根据完成对多个观察者（即实例方法的归属对象）进行特定事件的通知。

在视图层，为了方便，设定成一个视图必须而且只能直接关注一个模型及其状态变更，若需要关注额外的模型，通过一个全局事件管理器到模型中绑定事件以间接实现。视图需要记录自身注册到模型的事件，用于后续的注销操作。结合 Unity 组件式开发方式，视图内的参数需要出现在对象面板上，将其继承至 MonoBehaviour 类，并由 Unity 控制视图的生命周期。

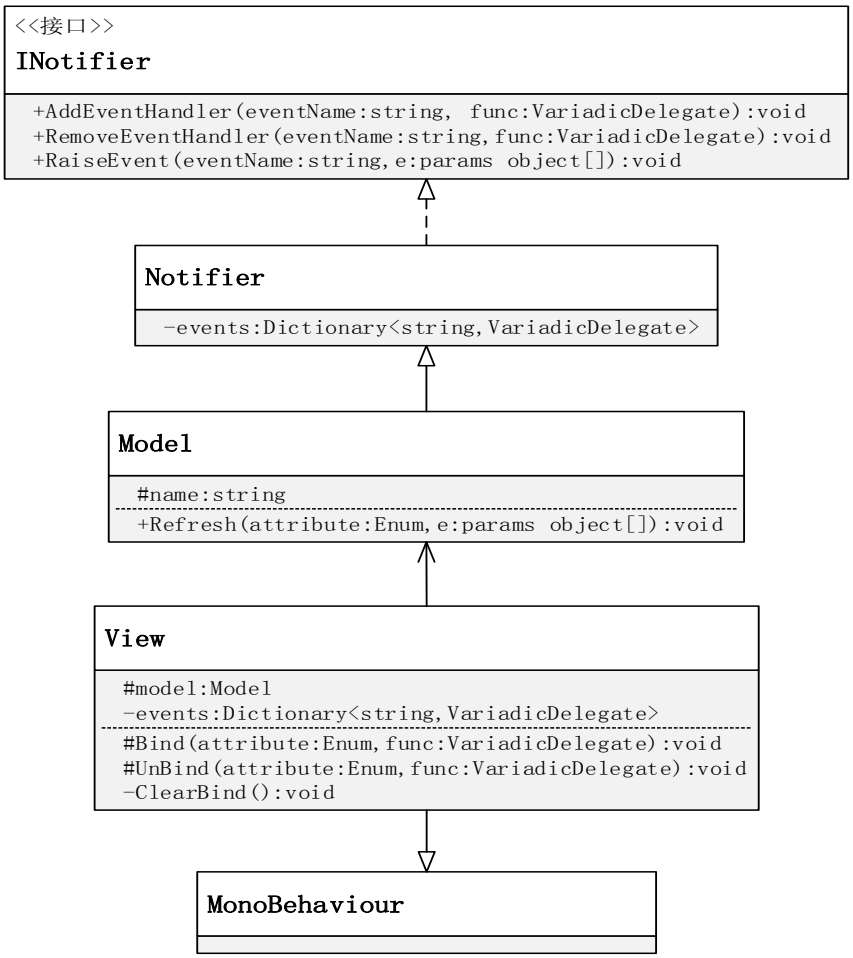


图 4.3 观察者模式 Unity 实现类图

4.3 全局管理器

系统通过一些全局性的管理器去统一控制 MVC 架构未考虑或未涵盖的操作，如系统设置、资源加载等。使用全局管理器，使得 Model、View 和 Controller 均能专注于自身的功能，而一些泛用性强的方法则由管理器完成，达到解耦的目的。这里全局管理器都是静态单例类。

4.3.1 单例模式

单例模式是一种最简单也是使用最广泛的设计模式。它要求所涉及的类负责创建自身的对象，并且有且仅有单个实例被创建。 单例类对外则提供了一种访问其实例对象的静态方法。

单例类的实现方式根据线程安全性和初始化方式划分有饿汉模式、懒汉模式、双重

锁等好几种方法。结合 Unity 单一线程的特点，绝大多数的开发中，单例的线程安全性可不必纳入考虑，因此在本次设计中，可以采用“懒汉模式”的不加锁版本，该实现方式无需同步，免去了传统加锁版本的额外性能开销。

本系统的开发过程中需要使用到多个单例类，为统一结构，提高复用性，将单例类设计成泛型版本 Singleton<T>。由于部分管理器需要结合 Unity 的方法进行使用，可以再添加一个继承自 MonoBehaviour 的泛型版本 MonoSingleton<T>，设计如下：

```
public abstract class MonoSingleton<T> : MonoBehaviour where T : MonoBehaviour
{
    private static T instance;
    public static T Instance
    {
        get
        {
            if (instance == null)
            {
                GameObject go = new GameObject(typeof(T).Name.ToString());
                instance = go.AddComponent<T>();
                DontDestroyOnLoad(go);
            }
            return instance;
        }
    }
    protected virtual void Awake()
    {
        Init();
    }
    protected virtual void Init()
    {
    }
}
```

应用中开发的所有管理器都将继承自单例基类 Singleton<T>或 MonoSingleton<T>，通过单例直接全局调用它们提供的接口。

4.3.2 单例管理器

通常情况下，Unity 开发过程中必要的管理器有：资源管理、UI 管理、声音管理、场景管理、配置管理和事件管理。结合本系统的需要，可另外增添一些管理器，如摄像

机管理器、Leap Motion 管理器、协程管理器，等等。这些管理器提供统一的接口，编写代码时应尽可能使用这些接口。

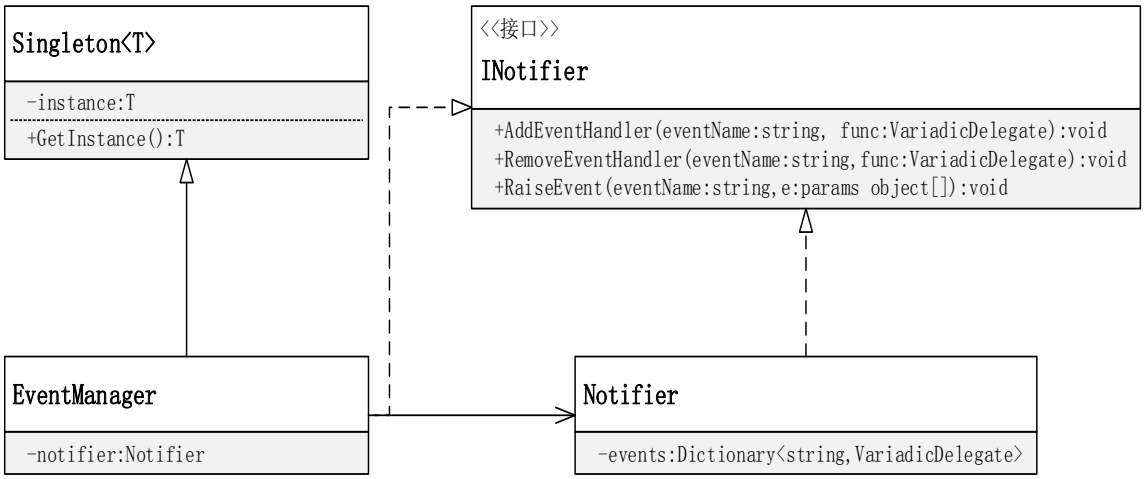


图 5.4 单例与事件管理器继承关系图

表 4.7 各单例管理器说明表

父类	子类	解释
MonoSingleton<T>	CameraManager	摄像机管理器，控制摄像机位置转换。
	CoroutineManager	协程管理器，提供调用协程的统一接口。
	FrameManager	帧管理器，提供逐帧事件的注册和定期执行。
	GlobalManager	全局配置管理器，配置系统参数和方法。
	HandManager	手管理器，提供手模型的访问和更新推送。
	LeapMotionManager	Leap Motion 管理器，提供 Leap Motion 各部分组件的访问方式。
	SoundManager	声音管理器，提供多种控制音频播放的方式。
	UIManager	UI 管理器，管理用户界面的加载和关闭。
Singleton<T>	EventManager	事件管理器，提供通用事件的注册、广播和撤回方法。
	ResourceManager	资源管理器，缓存资源，提供资源获取接口和资源加载的多种方法。
	SceneManager	场景管理器，管理场景切换和方法回调。

4.4 应用场景模块实现

4.4.1 资源管理

应用中出现的所有资源包括 UI、场景文件、场景物体和音频文件。这些资源文件由资源管理器 ResourceManager 统一加载和管理。

Unity 加载资源有多种途径，常见的有使用 Resource.Load 等方法直接由本地文件夹加载现成资源，该文件夹指的是相对路径为 Assets/Resources 的文件夹，需要手动创建。还有一种方式是通过加载 AssetBundle 资源包体后进行解压缩来获取资源，AssetBundle 是以某种方式将一系列的资源文件或者场景文件紧密保存的一种文件，采取此方法可以减小最终发布出来的应用体积，可进行资源的更新替换，并做到按需加载以减少内存占用。

本应用将同时采用以上两类加载方式，在开发过程中采用本地加载，而在发布版本则采用 AssetBundle 包体加载。除此之外，资源加载还有同步和异步方式，在这里，编辑器模式下使用同步加载，发布版本中采用异步加载。在 ResourceManager 脚本下添加判断：

```
#if UNITY_EDITOR
        public bool IsDefaultAsync = false;
        public bool IsDefaultFromServer = false;
#else
        public bool IsDefaultAsync = true;
        public bool IsDefaultFromServer = true;
#endif
```

资源管理器对外提供 GetResource 和 LoadAsset 两个接口，负责获取资源和加载资源，LoadAsset 内部通过 GetResource 判断资源是否已存在内存，不存在时才根据加载方式去加载资源文件，加载所用的路径由 PathHelper 路径工具单例拼接。

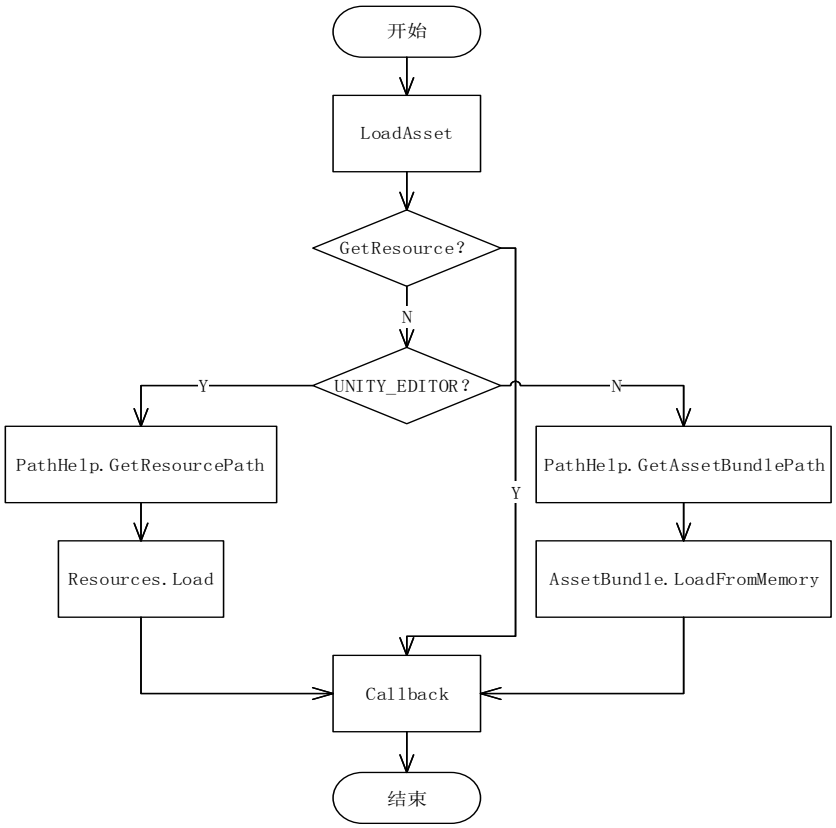


图 4.5 资源加载流程图

4.4.2 UI 界面

控制每个场景 UI 界面（包括菜单界面）的视图类均继承自 UIView 类，UIView 从基类 View 派生而来。UIView 将会自动处理每个 UI 元素的初始化、子界面切换、按键特效和音效等任务。

当没有 Leap Motion 设备连接时，需让鼠标也能够使用。为了支持鼠标，需提供鼠标所使用的辅助 UI，并负责处理它们的显隐。在脚本中添加操作模式变更事件的代码：

```
private void OnConnectedStateChanged(params object[] arg1)
{
    for (int i = 0; i < testBtn.Length; i++)
    {
        testBtn[i].gameObject.SetActive(!(bool)arg1[0]);
    }
}
```

并在事件管理器进行注册：

```
EventManager.Instance.AddEventHandler(Define.EventType.OpModeChanged, OnConnectedStateChanged);
```

每个派生自 UIView 类的具体 UI 界面还需提供该界面上按钮所需要的操作方法，

并在相应的 UI 预制体按钮上逐一绑定。

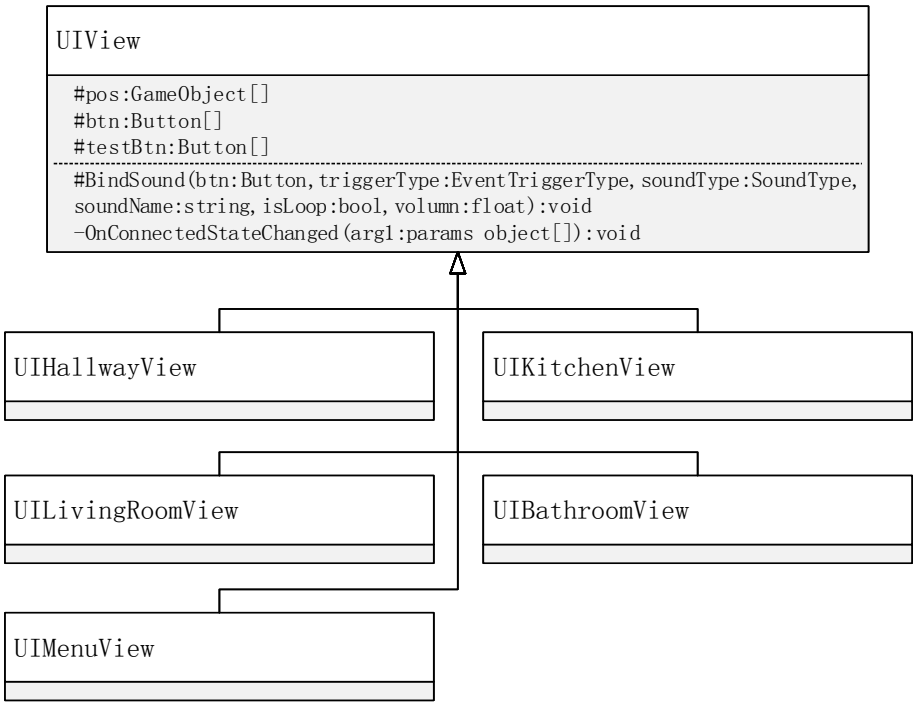


图 4.6 UI 视图继承关系图

4.4.3 镜头控制

1、摄像机镜头转换

在主场景中通过点击界面方位按钮进行镜头切换，按钮触发事件后将获取场景视图的目标位置并调用摄像机管理类 CameraManager 提供的方法。该流程严格按照 MVC 架构进行执行，UIView 必须请求 Controller 去通知相应的 SceneEntityView，场景视图则根据自身的镜头预设点去发起镜头切换，当镜头切换完毕后，还会去执行 UI 视图传入的委托，最终完成场景与界面的同步切换。

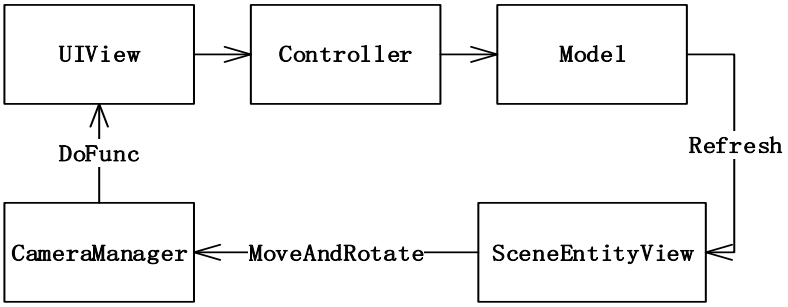


图 4.7 镜头切换控制流程图

当启用镜头切换时，摄像机管理器在每帧对摄像机位置和角度进行平滑处理，在

Update 回调方法内进行实现：

```
if (isStart)
{
    Camera.transform.position = Vector3.SmoothDamp(Camera.transform.position, toPos, ref velocity, smoothTime);
    Camera.transform.rotation = Quaternion.Slerp(Camera.transform.rotation, toRot, 1 / smoothTime * Time.deltaTime);
    if (IsArriveTargetPos(Camera.transform.position,toPos) && IsArriveTargetRot(Camera.transform.rotation, toRot))
    {
        isStart = false;
        if (func != null)
            func();
    }
}
```

脚本需判断是否完成了镜头切换，使用以下两个方法：

bool IsArriveTargetPos(Vector3 current, Vector3 target)；判断是否到达坐标位置。

bool IsArriveTargetRot(Quaternion current, Quaternion target)；判断是否旋转到目标角度。

其中 IsArriveTargetRot 方法将参数的四元数转换为欧拉角，再当成三维向量去调用 IsArriveTargetPos 方法进行判断。

2、场景切换画面渐变过渡

在两个场景间进行切换时，除了摄像机需要对镜头进行转换，还需添加过渡画面，以屏蔽场景加载的卡顿和镜头切换动作。本应用的过渡画面采用黑屏淡入淡出的渐变方式。

在主摄像机下添加一个画布 Canvas，并挂载 Image 组件，选择一张方形的贴图作为背景，颜色初始化为全黑全透明值（0，0，0，0），渐变过程通过改变 Alpha 值对黑背景进行渐变显隐。为了使黑背景遮住所有 UI，将 Canvas 的渲染层级 Order In Layer 提高到 1；同时，为遮住场景物体，调整黑背景的 Z 轴位置，使其称为镜头前最近的物体。

在项目中引入 DOTween 插件，DOTween 是当前一种执行效率和开发效率最高的 Unity 补间动画插件，常用于制作各种 UI 效果。DOTween 通过采用 C#方法链的写法使得复杂动画可以一目了然。处于同一方法链下的方法优先级相同，按照动画的生命周期执行。方法链可通过委托层层嵌套，让每个动作节点可以处理更复杂的任务，具有很强的拓展性。

在 CameraManager 脚本中引用 DG.Tweening 命名空间，然后编写场景切换的背景动画接口：

```
public void ChangeScene(float darkDuration, float pauseDuration, float showDuration, Action onComplete)
{
    blackBG.DOKill();
    ViewToDark(darkDuration)
        .OnPlay(() => IsChanging = true)
        .OnComplete(() =>
        {
            ViewToDark(pauseDuration)
                .OnPlay(() =>
                {
                    if (onComplete != null) onComplete();
                })
                .OnComplete(() =>
                {
                    ShowView(showDuration)
                        .OnComplete(() => IsChanging = false);
                });
        });
}
```

此处代码完成了画面渐变的整个流程：黑屏淡入→保持黑屏→执行场景切换→黑屏淡出。其中，ViewToDark 和 ShowView 方法内使用 DOTween 提供的 DOFade 方法负责处理黑背景的透明度渐变，其参数为目标 Alpha 值和动作的执行时长。

4.4.4 场景实体视图

每个场景都由设置成单独的预制体，并挂载相应的视图脚本，每个脚本都是继承自 SceneEntityView 类，该类也是 View 的派生。SceneEntityView 类主要负责处理场景镜头以及场景声效的切换。

为了方便在 Unity 中配置场景声效，在 SceneEntityView 类中添加一个声效辅助类 SceneSoundGroup 类的实例列表。SceneSoundGroup 类能够对多个音频剪辑进行分组，并设置它们的播放属性。在 Unity 中，当自定义类包含集合元素，且该自定义类型又是另一个类中集合的类型时，需要对它设置[Serializable]标签，才能使其在 Inspector 面板上多层嵌套地显示出来。

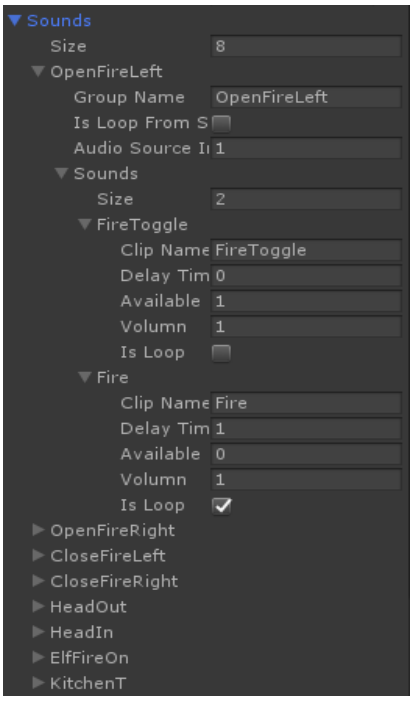


图 4.8 场景声效嵌套配置

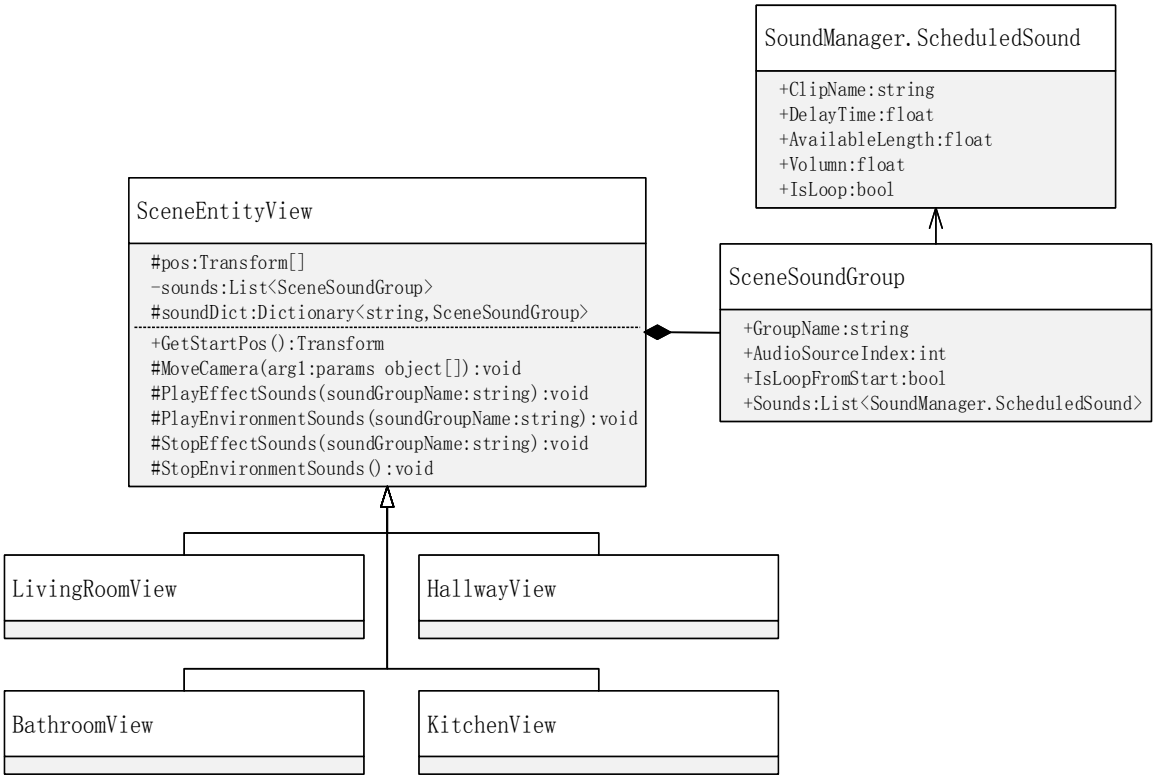


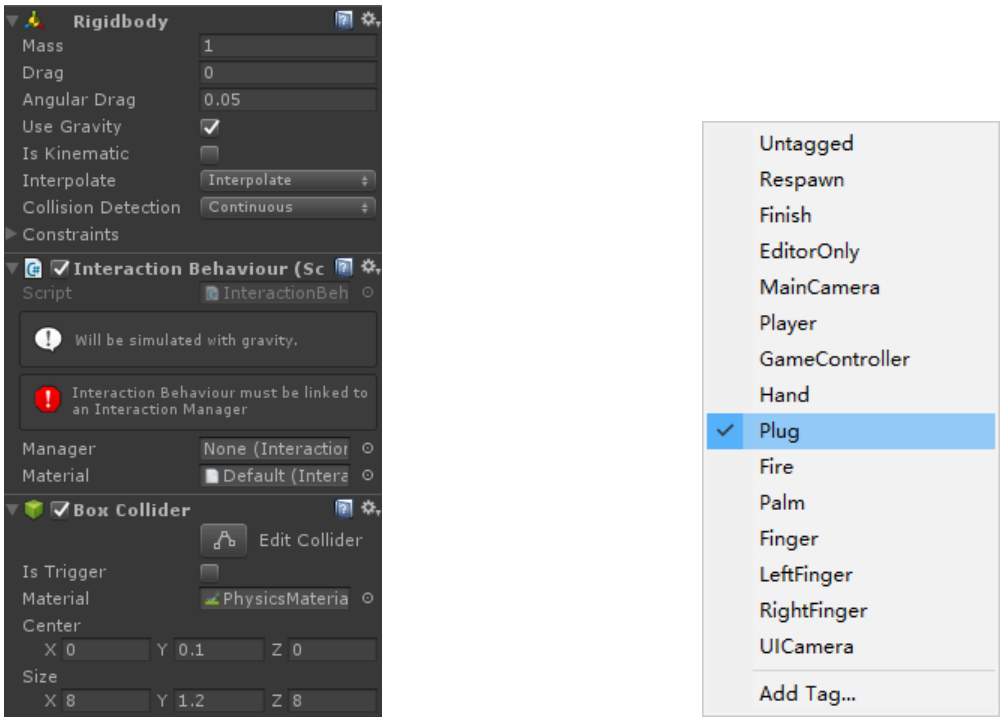
图 4.9 场景实体视图与场景声效继承关系图

4.4.5 客厅场景

客厅的主要交互对象为电视，通过模拟接通电源操作来打开电视并播放节目。

在已布置好的电视电源线添加刚体和碰撞体，并挂载 InteractionBehaviour 脚本，让

它成为 Leap Motion 交互引擎可识别并交互的对象。在插头处也添加一个 Box Collider 组件，插头需勾选 Is Trigger 选项以便成为触发器，更改插头的 Tag 为“Plug”，用于触发标识。



(a) 组件参数配置　　(b) 标签设置

图 4.10 交互引擎对象配置图

在插座上挂载触发器，并新建 SocketView 脚本，编写插座触发器逻辑：

```
void OnTriggerEnter(Collider other)
{
    if (other.tag == "Plug")
        LivingRoomCtrl.Instance.InsertPlug();
}
void OnTriggerExit(Collider other)
{
    if (other.tag == "Plug")
        LivingRoomCtrl.Instance.PutPlugOut();
}
```

当电源线插头部分接触插座时，便会向 LivingRoomCtrl 发出插入电源请求，并经过 LivingRoomModel 通知 LivingRoomView 去操作电线和电视。

```
private void InsertPlug(params object[] arg1)
{
```

```
        if (plugView.IsDrag == false)
            plugInteraction.OnHandReleasedEvent += insertFunc;
        else
            insertFunc.Invoke(null);
    }
```

InsertFunc 是负责将镜头切换到电视机前并播放视频的委托方法。若采用 Leap Motion 交互引擎去抓取电源线插入插座时，需为电源线上 InteractionBehaviour 的回调方法 OnHandReleasedEvent 添加该事件，当检测到手释放了电源线后便会去执行。

PlugView 是挂载在电源线物体上的脚本，当采取鼠标操作时，通过左键点击电源线将触发 Unity 内置回调方法 OnMouseDown，拖动电线平移。

```
    IEnumerator OnMouseDown()
    {
        Vector3 ScreenSpace = Camera.main.WorldToScreenPoint(transform.position);
        Vector3 offset = transform.position - Camera.main.ScreenToWorldPoint(new Vector3(Input.mousePosition.x, Input.mousePosition.y, ScreenSpace.z));
        isDrag = true;
        while (Input.GetMouseButton(0) && isDrag)
        {
            Vector3 curScreenSpace = new Vector3(Input.mousePosition.x, Input.mousePosition.y, ScreenSpace.z);
            Vector3 CurPosition = Camera.main.ScreenToWorldPoint(curScreenSpace) + offset;
            transform.position = CurPosition;
            yield return null;
        }
    }
```

Camera.main.WorldToScreenPoint：将电源线所在位置转换为屏幕空间，由于鼠标的移动范围是一个二维的平面空间，而物体需在三维立体空间内进行移动，这里需要得到电源线在屏幕空间的 z 轴位置，才能使平移方向变为二维。

Camera.main.ScreenToWorldPoint：通过鼠标在屏幕的位置和已求出的 z 轴位置，得到鼠标在世界空间的位置。

代码通过转换鼠标位置到世界空间，加上电源线与鼠标的初始偏移量 offset，得到电源线基于鼠标位置的实时三维坐标。

4.4.6 浴室场景

浴室的主要交互对象是浴缸，通过进水和排水开关来控制浴缸水位。

在浴缸上的两个开关上分别挂载触发器，然后添加触发器脚本 WaterInToggleView

和 WaterOutToggleView，触发后改变浴缸放水状态。

浴缸在注水过程中自动控制水位，以防水体高过浴缸。为浴室场景视图脚本 BathroomView 中的 Update 方法添加以下代码：

```
if(isInDown && wt.localPosition.y<waterMaxHeight)
{
    wt.Translate(Vector3.up* waterInSpeed);
}
if(isOutDown && wt.localPosition.y > waterMinHeight)
{
    wt.Translate(Vector3.down* waterOutSpeed);
}
```

在注水过程中，当水位达到预设最高值时，排水操作也将进行，即使没有按下排水开关，且排水需要略快于注水，保证水位平稳。

当浴缸开关被触碰后，开关会自动下压或者抬起。为避免开关下压或抬起过程中再次接触手部而二次触发了触发器，需要设置一个缓冲期，禁用触发器。在脚本中添加一个标志位 isAnyStateChanged，代表是否处于开关动作阶段，并用计时器复位。

```
if(isAnyStateChanged)
{
    if(cdTimer<interval)
    {
        cdTimer += Time.deltaTime;
    }
    else
    {
        cdTimer = 0;
        isAnyStateChanged = false;
    }
}
```

4.4.7 厨房场景

厨房的交互对象包括了炉灶、顶灯和冰箱，分别代表了接触式非手势操作、非接触式手势操作、和接触式手势操作三种不同情况。

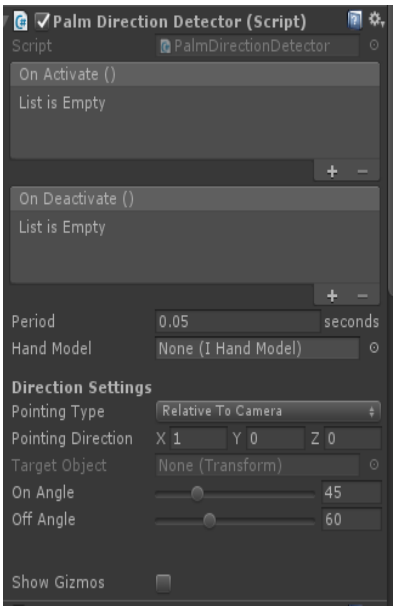
1、炉灶

厨房的炉灶采用了与浴室浴缸开关相同的操作设计，即通过手部直接触发开关上的触发器，点燃或熄灭炉火。

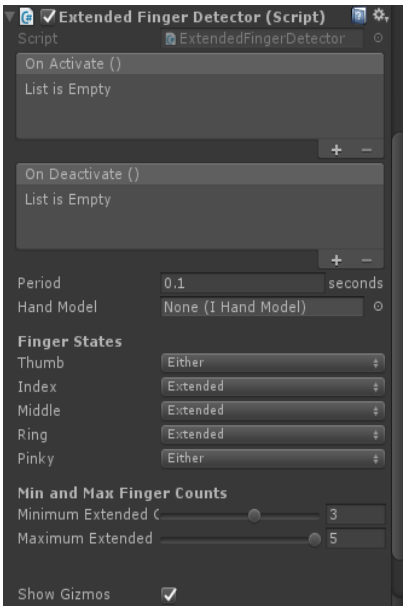
2、顶灯

顶灯的控制使用了 Leap Motion 自带的手势识别脚本 Detector 及其派生类。并通过 Detector Logic Gate 组件组合成一套复合手势，只有复合手势的最终输出为真，才可打开或关闭顶灯。

在厨房对象下新建一个空物体，添加 2 个手掌方向监测脚本 PalmDirectionDetector 和 2 个手指伸展监测脚本 ExtendedFingerDetector，分别监测左右双手的手掌方向和手指的曲直状态。以左手为例，在 PalmDirectionDetector 组件中选择指向类型“Pointing Type”为“Relative To Camera”，设定 Pointing Direction 的 x 值为 1，y 和 z 为 0，代表着朝向摄像机正右方为激活，右手配置反之。接着在每个 ExtendedFingerDetector 组件中都分别设置每个手指的激活条件 Finger States，此处仅要求食指、中指即无名指必须为 Extended，其余两指不作要求，为 Either。



(a) 手掌方向设置



(b) 手指状态设置

图 4.11 顶灯监测器配置

为了满足通过拍掌控制灯光，需要再添加一个监测手掌接触的监测器。由于 Leap Motion 自带的几类监测器都不满足此类需求，需要编写自己的监测器脚本。编写继承自 Detector 的 ObjectProximityDetector 类，实现存在多组目标对象情况下，组与组的物体的接触监测，Detector 采取协程定时监测，其协程部分代码如下：

```
IEnumerator proximityWatcher()
{
```

```
bool proximityState = false;
float onSquared, offSquared;
while (true)
{
    onSquared = OnDistance * OnDistance;
    offSquared = OffDistance * OffDistance;
    if (_currentLhsObj != null && _currentRhsObj != null)
    {
        if (distanceSquared(_currentLhsObj, _currentRhsObj) > offSquared)
        {
            _currentLhsObj = null;
            _currentRhsObj = null;
            proximityState = false;
        }
    }
    else
    {
        if (TargetObjects.Length <= 1)
        {
            for (int i = 0; i < TargetObjects.Length; i++)
            {
                if (TargetObjects[i].Length <= 1)
                {
                    _currentLhsObj = _currentRhsObj = null;
                    proximityState = false;
                    break;
                }

                for (int j = 0; j < TargetObjects[i].Length - 1 && !proximityState; j++)
                {
                    GameObject lhs = TargetObjects[i][j];
                    for (int k = j + 1; j < TargetObjects[i].Length; k++)
                    {
                        GameObject rhs = TargetObjects[i][k];
                        if (distanceSquared(lhs, rhs) < onSquared)
                        {
                            _currentLhsObj = lhs;
                            _currentRhsObj = rhs;
                            proximityState = true;
                            OnProximity.Invoke(_currentLhsObj, _currentRhsObj);
                            break;
```

```
                                }
                            }
                        }
                    }
                }
                else
                    for (int i = 0; i < TargetObjects.Length - 1 && !proximityState; i++)
                        for (int j = i + 1; j < TargetObjects.Length && !proximityState; j++)
                            for (int k = 0; k < TargetObjects[i].Length && !proximityState; k++)
                            {
                                GameObject lhs = TargetObjects[i][k];
                                for (int l = 0; l < TargetObjects[j].Length; l++)
                                {
                                    GameObject rhs = TargetObjects[j][l];
                                    if (distanceSquared(lhs, rhs) < onSquared)
                                    {
                                        _currentLhsObj = lhs;
                                        _currentRhsObj = rhs;
                                        proximityState = true;
                                        OnProximity.Invoke(_currentLhsObj, _currentRhsObj);
                                        break;
                                    }
                                }
                            }
            }
            if (proximityState)
                Activate();
            else
                Deactivate();
            yield return new WaitForSeconds(Period);
        }
    }
```

其中，TargetObjects 为二维数组，代表着若干个目标对象组，每一组内又包含多个子对象。代码段中的对象组接触监测部分采用了多个循环嵌套，在各对象组之间两两匹配，直到有两组对象成功匹配时停止。若有 n 个目标对象组，每组 m 个对象，根据握手定理，最坏匹配次数为 $m^2n(n-1)/2$ 次。当对象组只有一组时，改为组内匹配，最坏匹配次数为 $m(m-1)/2$ 次。

各对象之间的匹配算法由 distanceSquared 方法提供，该方法求得了两个带有碰撞体

的物体分别与其目标对象的最近点间的距离：

```
private float distanceSquared(GameObject lhs, GameObject rhs)
{
    Collider lhsCollider = lhs.GetComponent<Collider>();
    Collider rhsCollider = rhs.GetComponent<Collider>();
    Vector3 lhsClosestPoint, rhsClosestPoint;
    if (lhsCollider != null)
        lhsClosestPoint = lhsCollider.ClosestPointOnBounds(rhs.transform.position);
    else
        lhsClosestPoint = lhs.transform.position;
    if (rhsCollider != null)
        rhsClosestPoint = rhsCollider.ClosestPointOnBounds(rhs.transform.position);
    else
        rhsClosestPoint = rhs.transform.position;
    return (lhsClosestPoint - rhsClosestPoint).sqrMagnitude;
}
```

为所有的监测器添加配置，最后挂载 Detector Logic Gate 组件，设置门类型为与门，当所有关联的监测器输出全为真时，即双手伸直相对拍掌时，逻辑门输出为真，触发回调方法。

3、冰箱

冰箱的主要交互在于冰箱门把手，通过握住把手拉开或合上冰箱门，实现带手势的接触式操作。

新建一个空物体，挂载 2 个 ExtendedFingerDetector 组件，分别代表左右手。根据握拳习惯，将中指、无名指和小拇指配置为弯曲状态。

在每个冰箱门把手上添加 FridgeDoorView 冰箱门视图脚本，以控制冰箱门的协同旋转。为了让冰箱门紧跟着手一起旋转，门的旋转角度需要根据手绕门轴的水平偏移角度来计算。具体做法是：将前后两帧手的所在位置向把手所在的水平面做一个投影，然后计算两个投影向量的角度，即求得所需要的门的旋转角，如图 4.12 所示。

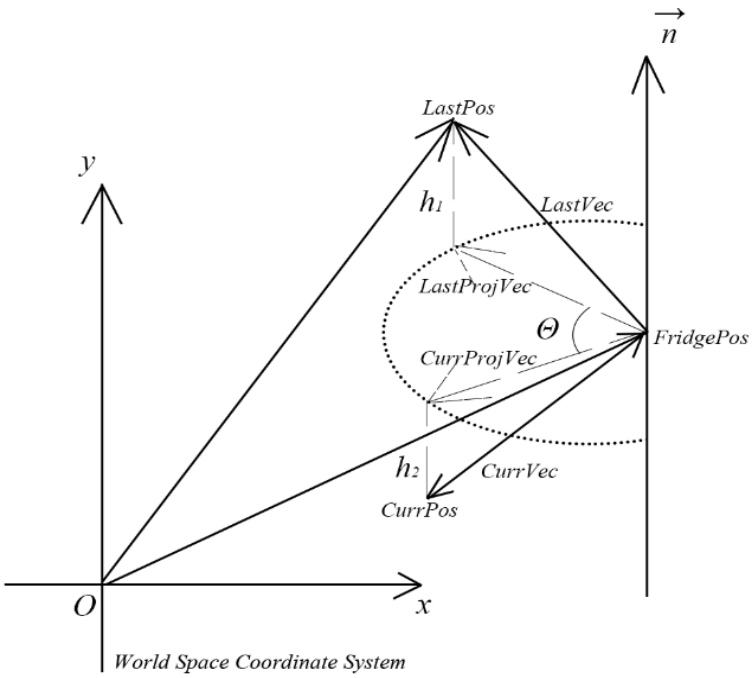


图 4.12 手部绕门轴旋转角求算图解

根据上图图解，可编写出求算旋转角的代码：

```
lastPos = currPos;
currPos = hand.position;
Vector3 lastVec = lastPos - fridgePos.position;
float lastHeight = Vector3.Dot(lastVec, nVec);
Vector3 lastProjVec = new Vector3(lastPos.x, lastPos.y - lastHeight, lastPos.z);
Vector3 currVec = currPos - fridgePos.position;
float currHeight = Vector3.Dot(currVec, nVec);
Vector3 currProjVec = new Vector3(currPos.x, currPos.y - currHeight, currPos.z);
Vector2 from = new Vector2((fridgePos.position - lastProjVec).x, (fridgePos.position - lastProjVec).z);
Vector2 to = new Vector2((fridgePos.position - currProjVec).x, (fridgePos.position - currProjVec).z);
rotateAngle = VectorAngle(from, to);
```

其中求算投影向量时使用了一种简单的技巧：保持原向量的 xz 坐标不变，y 坐标减去或加上原向量到投影平面的距离，即得到目标投影向量。

当采取鼠标操作时，由于只能在 2D 平面移动，缺失了 z 坐标的信息，因而不能够使用原来的算法。通过忽略 y 轴的影响，设把手到门轴的距离为 Length，鼠标点击位置通过 Camera.main.ScreenToWorldPoint 函数转换为三维坐标，再用勾股定理求算未知的 z 轴坐标，得到了由 x、z 和 Length 组成的三边长度，便能利用反三角函数求算出角度。

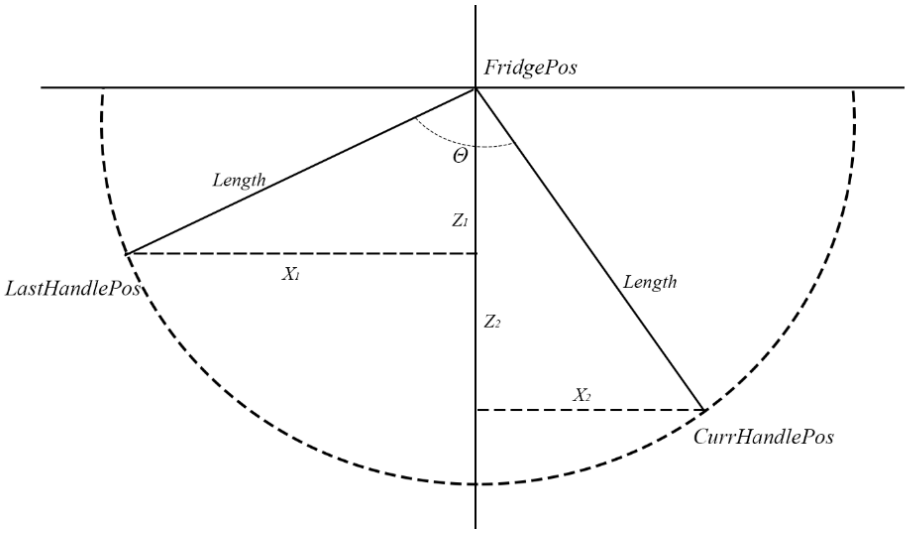


图 4.13 鼠标拉动门把旋转角图解

根据图 4.13 的图解，在 OnMouseDown 回调方法中编写代码求算出鼠标操控把手的旋转角：

```
IEnumerator OnMouseDown()
{
    isMouseDown = true;
    Vector3 friPos = fridgePos.position;
    lastPos = Camera.main.ScreenToWorldPoint(
            new Vector3(Input.mousePosition.x, 0, Camera.main.WorldToScreenPoint(handle.position).z));
    lastPos.y = friPos.y;
    while (Input.GetMouseButton(0))
    {
        currPos = new Vector3(0, friPos.y, handle.position.z);
        currPos.x = Camera.main.ScreenToWorldPoint(
                new Vector3(Input.mousePosition.x, 0, Camera.main.WorldToScreenPoint(handle.position).z)).x;
        float z2 = Mathf.Pow(length, 2) - Mathf.Pow((currPos.x - friPos.x), 2);
        if (z2 < 0)    z2 = 0;
        currPos.z = friPos.z - Mathf.Sqrt(z2);
        float sin1 = Mathf.Abs(lastPos.z - friPos.z) / length;
        if (sin1 > 1f)    sin1 = 1;
        float lastAngle = Mathf.Rad2Deg * Mathf.Asin(sin1);
        if (lastPos.x > friPos.x)    lastAngle = 180 - lastAngle;
        float sin2 = Mathf.Abs(currPos.z - friPos.z) / length;
        if (sin2 > 1f)    sin2 = 1;
        float currAngle = Mathf.Rad2Deg * Mathf.Asin(sin2);
        if (currPos.x > friPos.x)    currAngle = 180 - currAngle;
        rotateAngle = currAngle - lastAngle;
```

```
            lastPos = currPos;
            yield return null;
        }
        isMouseDown = false;
    }
```

脚本编写完毕后，为每个门把手添加触发器以响应来自手的触碰事件或鼠标的点击事件。

4.4.8 走廊场景

走廊只包括 UI 部分，通过操作按钮切换到客厅、浴室或者厨房场景。在走廊的 UI 视图脚本 UIHallwayView 里为切换按钮编写通用加载代码：

```
public void OnClickToOtherScene(string sceneName)
{
    GameObject hallway = FindObjectOfType<HallwayView>().gameObject;
    Object res = ResourceManager.Instance.GetResource(ResourceType.Scene, sceneName);
    if (res != null)
    {
        UIManager.Instance.CloseWindow(SceneType.MainScene, WindowType.Hallway);
        CameraManager.Instance.ChangeScene(0.5f, 0.2f, 0.5f, () =>
        {
            DestroyImmediate(hallway);
            GameObject obj;
            if (res is AssetBundle)
                obj = Instantiate((res as AssetBundle).LoadAsset(sceneName)) as GameObject;
            else
                obj = Instantiate(res) as GameObject;
            Transform startPos = obj.GetComponent<SceneEntityView>().GetStartPos();
            CameraManager.Instance.MoveAndRotate(startPos);
            UIManager.Instance.OpenWindow(
                SceneType.MainScene, EnumDescriptionTool.GetEnum<WindowType>(sceneName),
                null, ResourceManager.Instance.IsDefaultAsync, ResourceManager.Instance.IsDefaultFromServer);
        });
    }
}
```

4.5 资源打包与发布

4.5.1 打包 AssetBundle

在 Unity 中，常将资源文件打包成 Assetbundle 包体以缩小应用包体，并增加灵活

性，见 4.4.1 条。

Unity5 为开发人员提供了极为方便的打包方式，在项目文件夹中选择要打包的资源或预制体，在 Inspector 面板最下方展开可看到 Assetbundle 菜单。在第一个下拉框填写包体名称，可用“/”符号进行分级，便于管理。第二个下拉框填写变体名称，变体主要用了多版本切换所需加载的包体，这里不做考虑，为 None 即可。

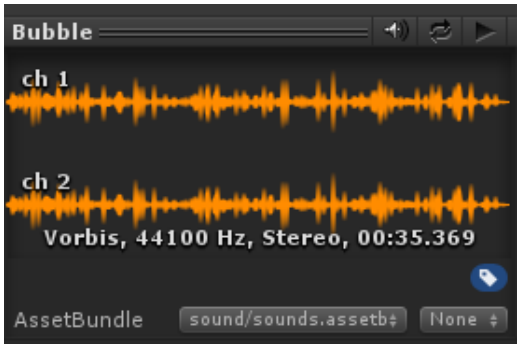


图 4.14 Assetbundle 菜单

为了实现打包，还需编写打包脚本。在 Assets/Editor 文件下下新建名为 BuildAB 的脚本并编写同名工具类 BuildAB：

```
public class BuildAB
{
    [MenuItem("AssetBundle/Build Windows AssetBundle")]
    static void BuildWinAB()
    {
        BuildPipeline.BuildAssetBundles(
                PathHelper.Instance.AssetBundlePath,
                BuildAssetBundleOptions.ForceRebuildAssetBundle |
                BuildAssetBundleOptions.IgnoreTypeTreeChanges |
                BuildAssetBundleOptions.ChunkBasedCompression,
                BuildTarget.StandaloneWindows64);
        AssetDatabase.Refresh();
    }
}
```

编译完成后，在 Unity 菜单栏上便会出现“AssetBundle”选项，选择其中的“Build Windows AssetBundle”进行资源打包。

当所有资源都打包成“.Assetbundle”格式文件后，需要将 Assets/Resources 下使用的资源都移动至其他目录下，以确保接下来发布出来的应用不再包含这些未经压缩且不会从本地文件中被直接访问的资源文件。

4.5.2 发布应用程序

为适应各种不同分辨率的机器使用，程序在发布成 PC 应用时需要将 Supported Aspect Ratios 下的所有比例全部勾选上。Unity 发布的应用默认带有开始动画，这里需要将其去掉，可在 Splash Image 下取消勾选 Show Splash Screen。

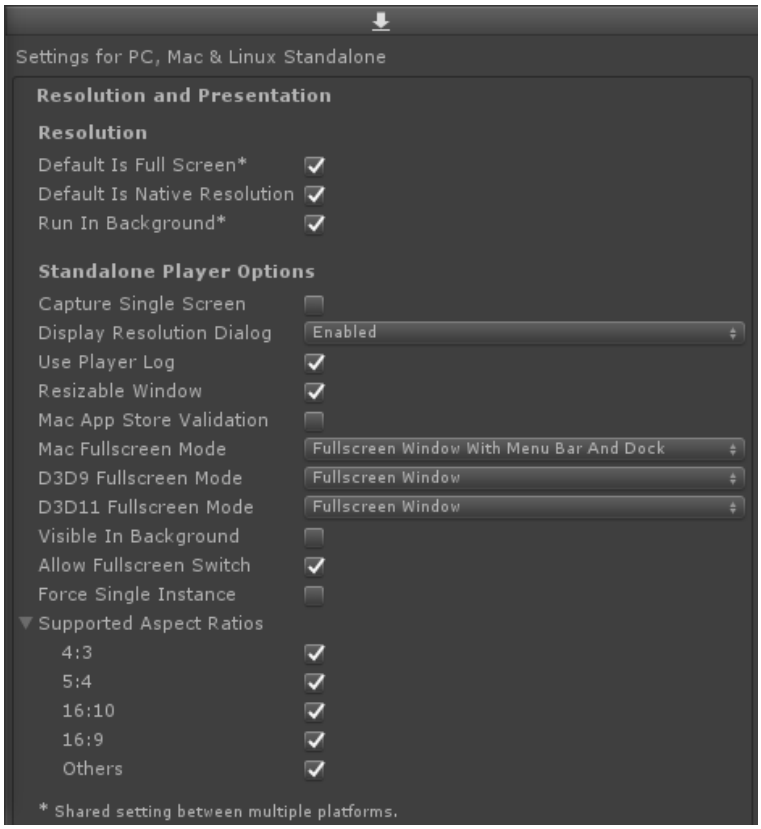


图 4.15 发布程序参数配置图

5 手势交互操作与测试

5.1 Leap Motion 驱动参数调整

使用 Leap Motion 对应用程序进行测试，需要使用 Leap Motion 驱动程序，从官网下载并安装驱动。

图 5.1 安装驱动程序

驱动安装完毕后，打开 Leap Motion 控制面板，在“常规”选项卡中将“自动节能”启用勾选，在“故障排除”选项卡中将“低资源模式”启用勾选，以防设备过热引发异常抖动。

接入设备后检查“Device Status”是否处于绿底显示的“CONNECTED”状态。

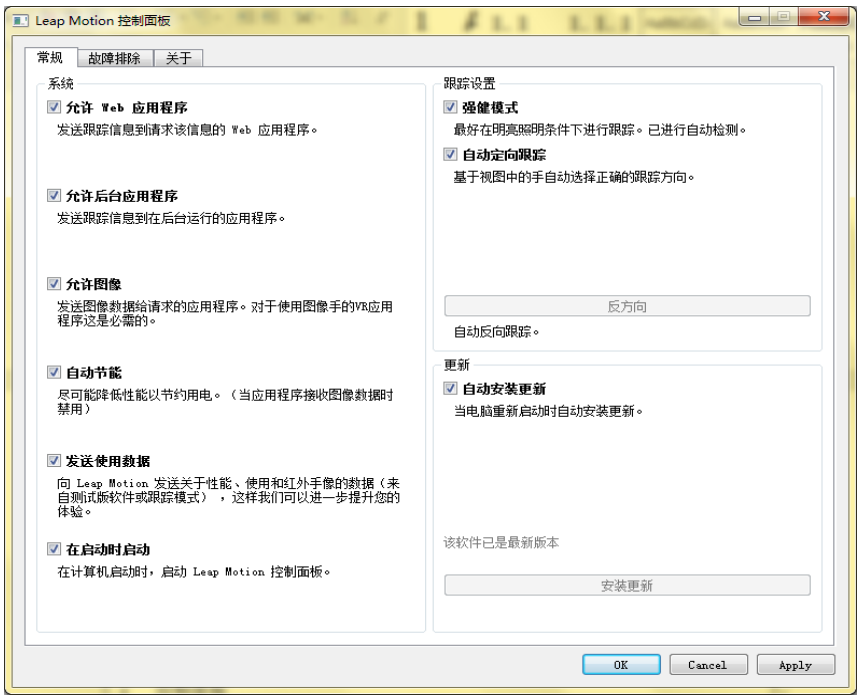


图 5.2 Leap Motion 控制面板

5.2 交互测试与结果

5.2.1 UI 测试

所有 UI 经测试，均能通过鼠标点击或手势触碰两种方式来操控，且当有 Leap Motion 设备连接时，鼠标指针自动隐藏，当 Leap Motion 设备断开连接时，鼠标指针会重新显示。

按钮 UI 在被点击或按压时，会有下压缩小动画，且底色变深，松开后会有弹起放大动画，并恢复底色。

常规模式下，第一次未进行过某种操作将会有 UI 文本提示，操作目标达成后，提示文本在重新加载主场景前不再出现。鬼屋模式下，当触发某些交互事件，如接通电视机电源、浴缸注水或厨房开灯等事件，UI 自动变为符合当前主题的绿底红字。

5.2.2 主场景测试

1、客厅

用手或鼠标可抓起电源线，移动至插座后松手，电线自动接上并切换镜头到电视机前，播放预定视频片段。

图 5.3 常规模式电源线交互

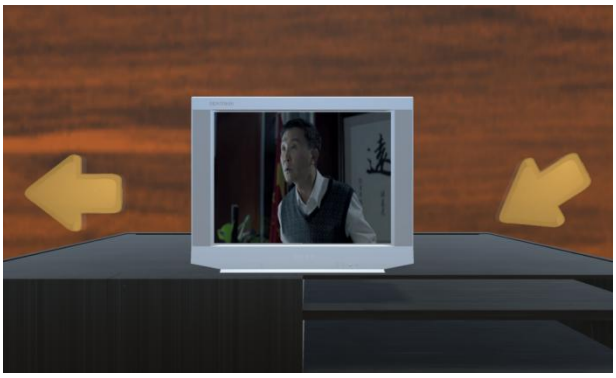


图 5.4 常规模式电视节目

2、浴室

用手或鼠标下压注水开关，浴缸开始注水，水面到一定高度保持平稳，常规模式下冒出气泡，鬼屋模式下冒出毒气。下压排水开关开始排水，排空后气泡或毒气消退。

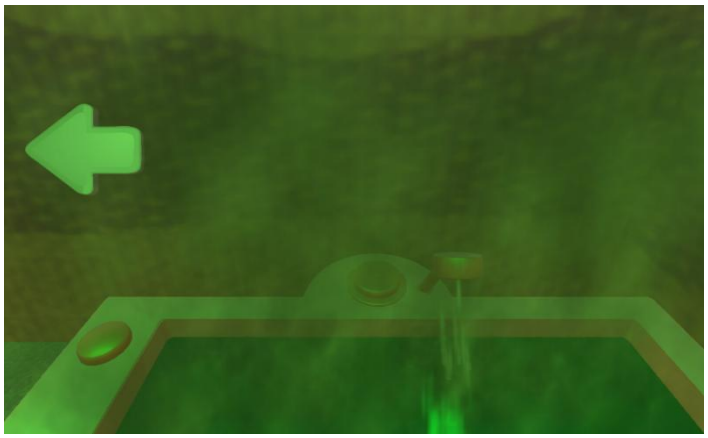


图 5.5 鬼屋模式浴缸交互

3、厨房

双手快速拍掌两次打开灯光，再次拍掌两次关闭灯光，鼠标模式下改为双击屏幕进行操作。鬼屋模式下会不断飘出鬼火。双掌相背拍打手背不起作用，双掌掌心方向沿竖直方向拍掌无效。

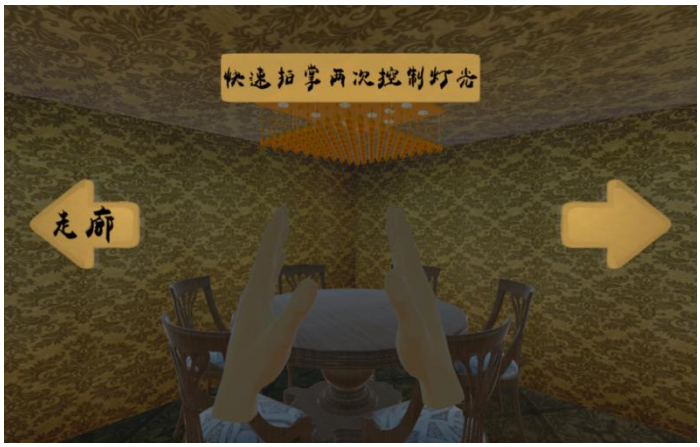


图 5.6 常规模式灯光交互

用手触碰或鼠标点击可以打开或关闭对应炉火，炉火打开时会有燃烧的声音，鬼屋模式下的炉火在开灯的情况下自动转为蓝色火焰。

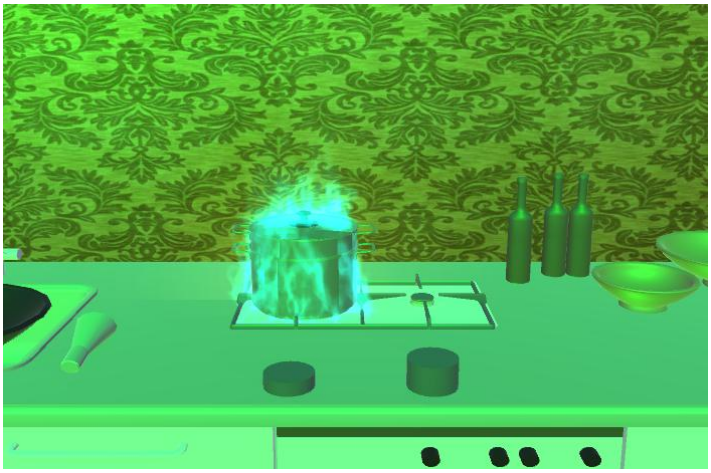


图 5.7 鬼屋模式炉火交互

一手合拢握住冰箱门把手，可以拉动把手，使冰箱门跟着转动。手指伸直，再次拉门把手不起作用。鼠标点击门把手，按住后左右拉动，可使冰箱门跟着转动。

图 5.8 常规模式冰箱门交互

6 项目总结

6.1 设计心得

本次毕业设计开发了一款基于 Leap Motion 的场景模拟桌面应用。万事开头难，在整个过程中，设计应用的内容是一件挺复杂的事情，围绕着“手势交互”这一要求，曾给出好几种不同的主题方案，结合自身对 Leap Motion 和 Unity 的学习，以及可获取到的资源素材，最终定下了室内场景模拟这一主题。

在用户界面上，根据 Leap Motion 的特点，设计了少而精的 UI，并不断优化它们的大小、位置和颜色等属性，进一步简化手势操作的要求。为了充分研究和发挥设备的各种功能，在每个场景镜头下，都有着不一样的交互对象和目的，且交互方式和结果都各具特色，如用抓住电线接通电源可播放电视节目，仅靠双手拍掌来控制灯光开关，等等，见表 6.1。完成基本设计后，在原有的基础上还增添了一种鬼屋模式，该模式在交互结果上进行了符合主题内容的改动，并将初始菜单界面也换成鬼屋主题，大大增强应用的趣味性。

表 6.1 Leap Motion 功能与应用实现对照表

Leap Motion 支持的功能	应用中实现功能的方案
直接触碰物体	控制 UI 与场景中的物体进行接触交互
Interaction Engine：抓起并操作物体	抓起、移动或抛掷电视机电源线
Detection Utilities：自定义组合手势触发事件	控制厨房灯光 触发握住冰箱门把手的条件之一
Attachments Module：根据曲线插值变换物体	未在应用中实现

通过反复探讨、修改本次设计方案，本人得到了一些 Leap Motion 应用设计上的心得。

1、要发挥 Leap Motion 从简出发的手势操作优势，界面的设计也需简明易用，交互方式不宜复杂化。

2、要多方面发挥 Leap Motion 所支持的功能，让用户能够感觉到物尽其用，交互方式的设计在要求简单的原则上又要讲究变化和扩展。

3、当前 Leap Motion 还未被大众所认知，因而在设计一款 Leap Motion 应用时，无论是从界面上，从交互方式上，还是从交互结果上，都要注重新颖性，让用户在体验过后能够印象深刻。

6.2 开发总结

在动手开发的过程中，通过优先搭建出整体开发框架，规范化程序的开发流程，使得整个项目的开发重点和难点能够从编写代码转移到场景设计当中，降低的编程难度。程序中若干处需要使用数学计算的地方，是编程部分的难点，在第五章的详细设计当中均有重点分析。

Leap Motion 在实际的测试使用中，精度很高，但过高的精度容易将人手轻微的颤抖也捕获到，有时会出现异常。在非接触式手势交互中，Leap Motion 可以满足多种复杂手势条件，具有很强的扩展性。而在接触式交互（包括 UI 交互）中，由于 Leap Motion 的坐标系与 Unity 坐标系单位不同，需要将所有物体大小及距离缩小到几十分之一到一百分之一不等才能用于交互，这一类的开发略带难度，需要反复调整。使用 Leap Motion 自带的交互引擎与物体进行接触式交互时，对物体形状和大小也有较多的限制。

将以上使用体验进行归纳，得到初步结论是：Leap Motion 较为适用于开发无接触或弱接触式的手势体感交互系统，例如翻阅电子书、下国际象棋、三维场景漫游等，在这些情景下的体验应当是非常良好的。事实上，本应用所使用到的只是 Leap Motion 部分功能，若深入研究其 API 和提供的资源包，可以制作出如同官方所展示的 Demo 一样强大的应用程序。

6.3 对 Leap Motion 的展望

就目前而言，关于 Leap Motion 应用开发的资料非常稀缺，对于开发出功能丰富的程序还是有一定难度。然而 Leap Motion 凭借着精度高、速度快、价格低、设备轻巧，适用平台广且只需升级驱动等众多优点，依然具有极大的潜力，是研究体感交互的优秀选择。相信未来通过 Leap Motion 公司与 VR-AR-MR 产业链中其他公司的不断深入合作，更多新奇有趣的或实用方便的应用被开发出来，Leap Motion 及其提供的手势交互体验将被众人所熟知，其价值还会体现在智能手机、机器人、汽车等地方。而手势这一交互方式也将成为未来计算的一部分，与鼠标、键盘、触摸、声音等并存。

参 考 文 献

[1] Menache A. Understanding Motion Capture for Computer Animation and Video Games [M]. Morgan Kaufmann. 1999-10.

[2] 黄波士，陈福民. 人体运动捕捉及运动控制的研究 [J]. 计算机工程与应用. 2005-07.

[3] 周开店，谢钧，罗健欣. 基于 Leap Motion 指尖位置的手势提取和识别技术研究 [J] 微型机与应用. 2017-02.

[4] 李映辉，史卓，安亚磊. 基于 LeapMotion 的三维手势识别方法 [J]. 现代计算机. 2016-05.

[5] 林书坦，尹长青. 基于 LeapMotion 的数字手势识别 [J]. 电脑知识与技术. 2015-12.

[6] 王元，高羽. uSens 带你了解世界最先进的手势识别技术[EB/OL]. uSens 凌感科技. 2015-02.

[7] 沈子恒. 三角测量原理与双目视觉景深恢复 [EB/OL]. http://blog.csdn.net /shenziheng1/article/details/53301480. 2016-11.

[8] 佚名. 深度：Leap Motion 手势识别大揭秘 [EB/OL]. 映维网. 2016-05.

[9] 张力钊. Leap Motion:用手指划出动感世界 [J]. 商业观察. 2016-04.

[10] 杜坤. 基于 LeapMotion 与 Unity3D 的体感游戏“Survial&Shoot”的开发 [D]. 云南大学. 2016-04.

[11] Hocking J. Unity in Action: Multiplatform Game Development in C# with Unity 5 [M]. 2015-06.

致　谢

本次毕业设计的从选题到设计过程中，杨卓老师给予了我很多宝贵的意见，并且允许我自行发挥设计主题。杨老师对我们和毕业设计工作十分重视和负责，时刻督促我们要认真、及时完成毕业设计，并为我们提供所需要的设备和资料。老师根据我们能力和喜好的不同因材施教，让我们能有足够的信心来完成整个课题，同时也非常谅解边实习边做毕设的学生，告诉我们要劳逸结合。老师还不断鼓舞有意愿的学生参加大学生毕业设计大赛，并为我们提供了机会和帮助，希望我们能够在毕业之前再获得一份荣誉。在这里，我由衷感谢这位对工作、对学生都尽责尽力的老师。