基于云计算的体感交互游戏

——基于LeapMotion和树莓派的实现探索

姓名：杜佳昆

学号：55100210

学院：软件学院

专业：软件工程

导师：房至一

职称：教授

基于云计算的体感交互游戏

——使用Leap Motion的体感游戏设计

摘要

当今计算机科学正在进入云计算时代，随着互联网的发展壮大，云计算的概念逐渐深入了人们的生活中。而随着电脑的普及，人们已经不满足于用鼠标和键盘这些传统输入设备来操作电脑，越来越多的人转向了对体感识别的研究，开发。它则涉及了很多的识别算法，需要大量的运算，需要一定的硬件作为支持。相比之下，云计算比传统的硬件堆积有更大的优势。因此，本论文提出了一种将数据获取和数据解析分离，运用云计算进行体感识别的实现方式。

方案具体内容为：使用类似树莓派这种开源硬件作为终端，来连接体感设备（比如Leap Motion），并将设备通过终端虚拟化映射到云端，使其成为云端的一个虚拟硬件，这样数据就能直接发送到云端，云端分析后形成可利用的结果数据反馈给体感应用进行后续操作。

这套方案所需工作包括：

<1> 支持跨平台的 USB over IP 驱动。

<2> 在终端平台上初始化体感设备的驱动。

<3> 体感交互应用

本论文着重探讨体感应用这一部分工作。

关键词：云计算，体感识别，手势识别，树莓派，Leap Motion，Linux ，Raspberry Pi，Arm，USB，USB over IP

The Somatosensory cloud-based interactive game

——Somatosensory Game Design using Leap Motion

**Abstract**

Nowadays, computer science is entering the cloud computing era, with the development and expansion of the Internet, the concept of cloud computing gradually penetrated people's lives. With the popularity of computers, people are not satisfied that operate computer with traditional input devices like mouse and keyboard, more and more people turned to the somatosensory recognition research and development. It involves a lot of recognition algorithm and computation, need some hardware for support. In contrast, cloud computing has more advantage than traditional hardware. Therefore, this paper proposes a cloud-computing way to implementations somatosensory recognition that separate data acquisition and data analysis.

The details of implementations: it uses open hardware like Raspberry Pi as terminal to connect somatosensory device (such as Leap Motion), and the device was mapped to cloud by terminal virtualization. The device become a virtual hardware on cloud, so data can be sent directly to the cloud, the cloud formed the results after analysis and send to somatosensory application to do others.

Work to do:

(1) A cross-platform USB over IP driver

(2) A driver that initialize somatosensory device on terminal

(3) Somatosensory application

This paper focuses the work of somatosensory application.

Keywords: Cloud Computing, Somatosensory Recognition, Gesture Recognition, Raspberry Pi, Leap Motion, USBIP, Linux, Arm, USB

第1章 绪论

1.1 课题来源

本课题源于作者在淘宝实习期间的一个工作需求。作者是淘宝 UED 的体感交互小组的一员，该小组设计开发了一款使用 Leap Motion 的体感游戏，出于推广的需要，计划将这款游戏部署在园区的一些位置，一方面供大家娱乐，一方面也能让大家更多的接触体感。

体感识别需要进行大量的运算，因此 Leap Motion 对于硬件的要求较高，部署时使用 PC 或者 mini PC 成本较高。因为园区内有内部使用的私有云服务，于是想到可以将体感识别与云计算结合，来达到部署的目的。

1.2 研究背景及意义

作者尝试解决以上需求的过程是一个探索的过程，所使用的方案运用了云计算的优势，提出了一种将体感交互和云计算结合的方式。

通过这种方式，减少体感交互应用的部署当中的障碍，使体感交互更容易的走进用户的生活。同时，也降低了体感交互研究的成本，提高其研究效率。这种方式，减少了一些体感交互使用的限制条件，使其的应用能够有更多的创意。

1.3 国内外研究现状

1.3.1 云计算

国外由于计算机发展较早，有很多厂商已经具备非常强大的开发和业务能力，同时具备在该领域传统的技术实力，因此，国外云计算发展的比较早，且有相对完整的模式。在国外，云计算的使用率比较高，服务提供商相对较多，质量也很好，像 Amazon，Linode等都经历了市场的考验。国内现在还处于成长期，预计在2015年之后，才真正进入成熟期，云计算服务模式才能被大众接受。处于安全等因素的考虑，国内云计算的使用率低于其他国家，大多是倾向于私有云，而不是公有云服务。国内的云服务商还在发展出去，阿里云，盛大云，SAE，BAE，美团云等，总体体验上跟国外的服务商还有些差距。云计算的普及，与网络条件的关系很大，国内的网络条件还有所欠缺。现在世界范围内，云计算的概念正应用于各种传统行业，为其带来了新的生机。

1.3.2 体感交互

体感交互方面，在国内外已经有很多相关的研究，也有很多的成果，Kinect、Leapmotion 等，但离大规模普及还有些距离。对于各种部位，动作的识别已经有很多成熟的技术，现在很多的设备都已经集成了体感识别的功能，手机，电脑，游戏机，家电等。对于识别准确率，复杂环境的识别等，是现在主要研究的方向。这种新型的输入方式，相比传统的键盘、鼠标、触控等设备，更接近人的本能，有良好的体验，发展势不可挡。

1.3.3 云计算与体感交互结合

对于两者的结合，已有基于云计算和体感网络的运动训练平台，NBA也将引入体感追踪技术，来进行大数据分析。另外，微软推出的Xbox One，将需要大量硬件资源，但对于延迟不敏感的图形计算放到云端去，比如流体和物理计算，画⾯面背景中的图形元素等，主机只接收计算结果，并将主要硬件资源用来计算比如人物的动作，击中时的反应等更关键的内容，但感觉对网络要求较高。

总体来说，两者都是未来发展的趋势，体感需要大量运算，将其与云计算结合，也是可行的方案，特别是在网络条件良好的环境内。

1.4 论文章节内容介绍

第一章绪论，主要介绍课题来源，研究背景及意义，国内外研究现状等3个方面的内容，具体的阐述了以上3点，是对全文的概括和导读。

第二章相关技术，主要介绍整套方案中所使用到的相关技术和理论知识。

第三章基础架构的设计和实现，主要介绍整套方案中USB over IP和体感设备在终端初始化的驱动的设计和实现。

第四章游戏设计，主要介绍这套方案中使用的体感交互游戏的设计过程，包括流程，资源，概念等方面的设计。

第五章游戏的实现和测试，主要介绍了游戏的具体实现过程，以及相关的测试。

第六章结束语，主要介绍了在开发过程中遇到的问题和心得。

第2章 相关技术

2.1 Linux驱动模型

Linux 系统中设备和驱动都统一由设备驱动模型来管理。

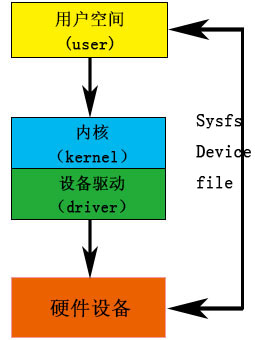
设备驱动模型将硬件抽象化，以便能够更好的进行操作。在Linux下，一切都是文件，所以每个设备在Linux的文件系统上都对应这一个文件。这些文件存在于/dev目录下，但这里面并不都是物理上存在的设备，有些是挂载这的udev文件。模型的硬件抽象包括：电源管理、即插即用设备支持、与Userspace的通信。Linux驱动分为设备驱动和总线驱动。驱动层次如图1。

图1 Linux 设备驱动层次

驱动程序的功能如下：

1. 对硬件设备初始化和释放
2. 进行设备管理，包括参数设置，提供对外的设备统一操作接口
3. 读取应用程序请求的数据或写入应用程序传递的数据
4. 检测或处理设备的错误

2.1.1 基础数据结构模型：kobject, kset, subsystem

Linux设备驱动模型有几个基本数据结构模型：kobject, kset, subsystem。

kobject：这是设备驱动模型的基础，sysfs就是以它为基础搭建起来的。类似于C++中的基类。它会嵌入到更大的对象中，即容器——用来描述设备模型组件，比如：bus、devices、drivers等。当kobject加入容器之后，就允许内核：

1. 保存改容器的引用次数
2. 保存各容器之前的层次关系
3. 提供每人容器的用户视图

kobject定义如下：

struct kobject

{

　　　　const char \*name; //显示在sysfs中的名称

　　　　struct list\_head entry; 　　//下一个kobject结构

　　　　struct kobject \*parent;　　　//指向父kobject结构体，如果存在

　　　　struct kset 　　\*kset;　　　　//指向kset集合

　　　　struct kobj\_type　　\*ktype; //指向kobject类型描述符

　　　　struct sysfs\_dirent \*sd; //对应sysfs的文件目录

　　　　struct kref kref;　　　　　　　　//kobject引用计数

　　　　unsigned int state\_initialized:1; //是否初始化

　　　　unsigned int state\_in\_sysfs:1; //是否加入sysfs

　　　　unsigned int state\_add\_uevent\_sent:1; //是否支持热插

　　　　unsigned int state\_remove\_uevent\_sent:1; //是否支持热拔

}

kset：它是相同类型结构的kobject的集合。

定义如下：

struct kset{

struct subsystem \* subsys;

struct kobj\_type ktype;

struct list\_head list;

struct kobject kobj;

struct kset\_hotplug\_ops \* hotplug\_ops;

};

subsystem：由一系列的kset组成，可以包含不同类型的kset。

定义如下：

struct subsystem{

struct kset kset;

struct rw\_semaphore rwsem;

}

2.1.2 设备

设备数据device一般是一个包含具体数据的设备数据结构体，定义如下：

struct device

{

　　 struct klist klist\_children; //连接子设备的链表

　　 struct device \*parent; //指向父设备的指针

　　 struct kobject kobj; //内嵌的kobject

　　 char bus\_id[BUS\_ID\_SIZE]; //连接到总线上的位置

　　 unsigned uevent\_supress:1; //是否支持热插拔事件

　　 const char \*init\_name; //设备的初始化名字

　　 struct device\_type \*type; //设备相关的特殊处理函数

　　 struct bus\_type \*bus; //指向连接的总线指针

　　 struct device\_driver \*driver; //指向该设备的驱动程序

　　 void \*driver\_data; //指向驱动程序私有数据的指针

　　 struct dev\_pm\_info power; //电源管理信息

　　 dev\_t devt; //设备号

　　 struct class \*class; //指向设备所属类

　　 struct attribute\_group \*\*groups; //设备的组属性

　　 void (\*release)(struct device \*dev); //释放设备描述符的回调函数

};

2.1.3 驱动

Linux下通用的驱动数据结构是device\_driver，不针对具体设备类型。目前，特定的设备驱动数据结构都会包含这个类型的成员。其定义如下：

struct device\_driver

{

　　 const char \*name;　　//设备驱动名字

　　 struct bus\_type \*bus; //指向驱动属于的总线，总线上有很多设备

　　 struct module \*owner; //设备驱动自身模块

　　 const char \*mod\_name; //设备驱动名字

　　 int (\*probe)(struct device \*dev); /探测函数

　　 int (\*remove)(struct device \*dev);

　　 void (\*shutdown)(struct device \*dev);

　　 int (\*suspend)(struct device \*dev,pm\_message\_t state);

　　 int (\*resume)(struct device \*dev);

　　 struct attribute\_group \*\*group;

　　 struct dev\_pm\_ops \*pm;

　　 struct driver\_private \*p;

};

2.2 Linux USB驱动

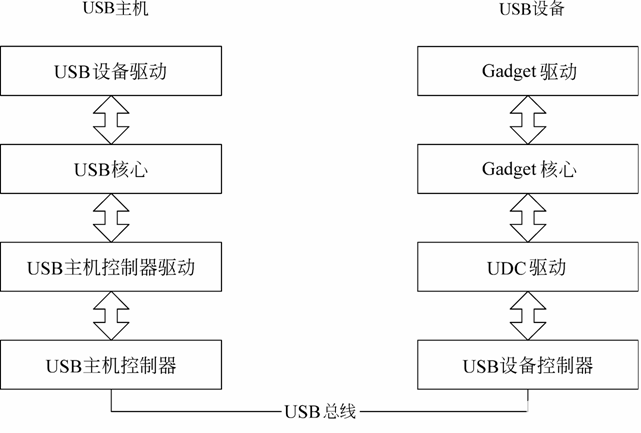
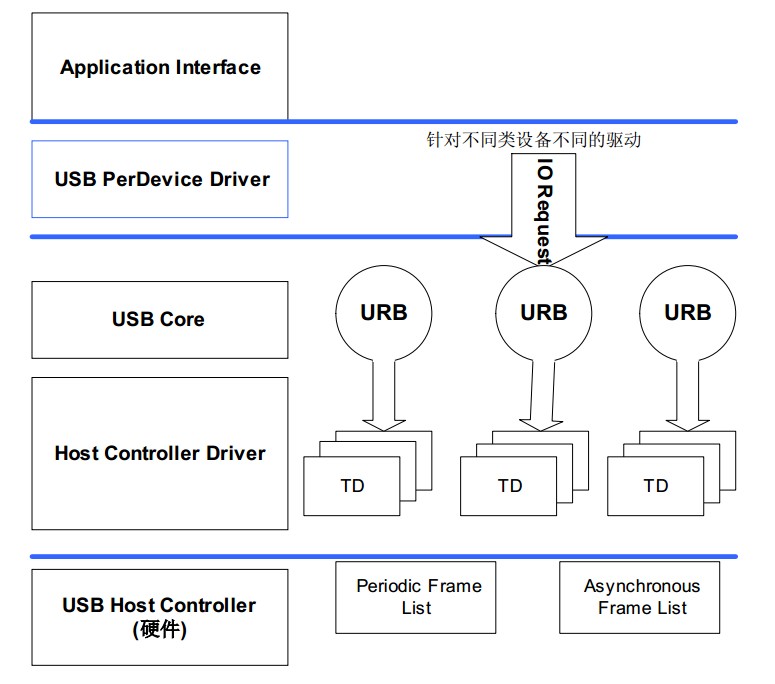
Linux内核几乎支持所有的USB设备，包括键盘，鼠标，打印机，扫描仪，modem等。在编译Linux内核时，有一个“Support for USB”选项，可以选择是否编译相关模块。Linux的usb驱动分为主机驱动和gadget驱动，前者是设备连接到计算机上，通过主机驱动来控制所连接设备。而后者用于嵌入式设备上，控制嵌入式设备。Linux的usb驱动两种类型图如下：

图2 Linux 的usb驱动类型

本论文主要涉及到的是主机驱动。主机驱动分为三层：USB PerDevice Driver,，USB core，USB HCD。对应的就是上图中的USB设备驱动，USB核心，USB主机控制器驱动。USB设备驱动位于USB核心上层，是连接用户应用程序和USB核心的桥梁，是最终与用户应用程序交互的软件模块，为其提供接口，视USB设备的不同而不同。比如键盘的usbhid驱动，U盘的usb-storage驱动。USB核心是Linux USB驱动的核心层，它将URB包传递给USB主机控制器驱动。最下面是USB主机控制驱动层，是USB主机驱动程序直接与硬件通信的软件模块，主要负责：主机控制器硬件初始化，为核心层提供接口函数，对根HUB设备进行配置和控制，完成四种类型数据的传输等。其流程如下图所示：

图3 Linux主机驱动工作流程

2.2.1 Linux USB驱动开发

2.3 LibUSB介绍

2.4 WireShark 介绍

2.5 NodeJS 介绍

2.6 LeapMotion

2.7 HTML5

2.8 CSS3