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，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。

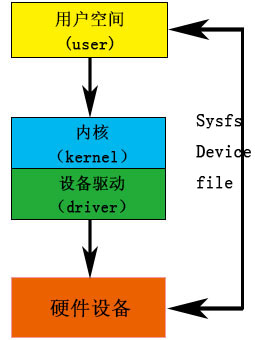
设备驱动模型将硬件抽象化，以便能够更好的进行操作。在Linux下，一切都是文件，所以每个设备在Linux的文件系统上都对应这一个文件。这些文件存在于/dev目录下，但这里面并不都是物理上存在的设备，有些是挂载这的udev文件。模型的硬件抽象包括：电源管理、即插即用设备支持、与Userspace的通信。Linux驱动分为设备驱动和总线驱动。驱动层次如图1。

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

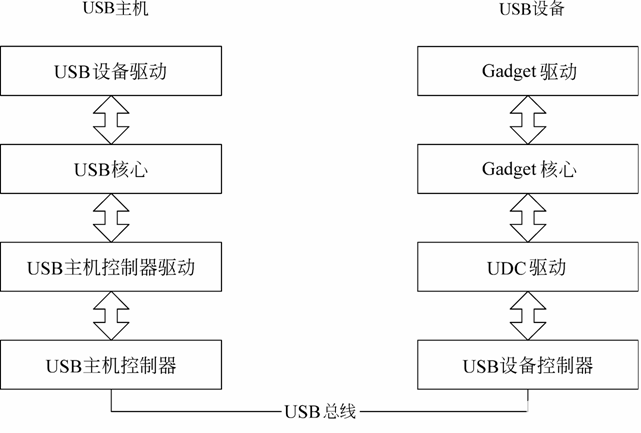
Linux内核几乎支持所有的USB设备，包括键盘，鼠标，打印机，扫描仪，modem等。在编译Linux内核时，有一个“Support for USB”选项，可以选择是否编译相关模块。Linux的usb驱动分为主机驱动和gadget驱动，前者是设备连接到计算机上，通过主机驱动来控制所连接设备。而后者用于嵌入式设备上，控制嵌入式设备。Linux的usb驱动两种类型图如下：

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：

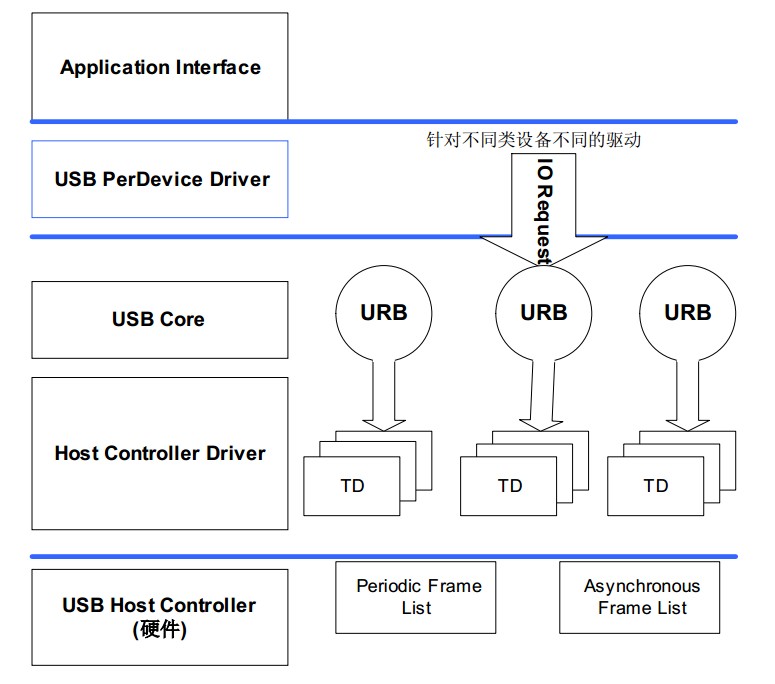


图 Linux主机驱动工作流程

2.2.1 Linux USB驱动开发

在Linux kernel源码driver/usb/目录下的usb-skeleton.c文件，是一个最基础的USB驱动程序，成为USB骨架。我们可以通过修改这个通用USB驱动骨架，来完成自己的USB驱动程序。

一个完整的USB驱动包括设备的注册和注销，读写操作的响应，释放设备文件句柄等功能。在骨架中这些被这样表示：

static struct usb\_driver skel\_driver = {

name: "skeleton",

probe: skel\_probe,

disconnect: skel\_disconnect,

fops: &skel\_fops,

minor: USB\_SKEL\_MINOR\_BASE,

id\_table: skel\_table,

};

驱动可以作为一个模块被载入内核，所以驱动必须提供一个初始化函数供模块载入内核时调用，可以写在初始化里。

static int \_\_init usb\_skel\_init (void)

{

usb\_register(&skel\_driver);

}

module\_init(usb\_skel\_init);

以上代码包含了驱动设备的注册。当卸载时，代码如下：

static void \_\_exit usb\_skel\_exit(void)

{

/\* deregister this driver with the USB subsystem \*/

usb\_deregister(&skel\_driver);

}

module\_exit(usb\_skel\_exit);

因为Linux支持热插拔，为了使系统能够自动装载驱动，需要创建一个MODULE\_DEVICE\_TABLE，代码如下：

/\* table of devices that work with this driver \*/

static struct usb\_device\_id skel\_table [] = {

{ USB\_DEVICE(USB\_SKEL\_VENDOR\_ID,

USB\_SKEL\_PRODUCT\_ID) },

{ } /\* Terminating entry \*/

};

MODULE\_DEVICE\_TABLE (usb, skel\_table);

Discount函数会在释放文件句柄是执行，主要是确认当前是否有其他程序访问这个设备，并减少计数器，操作如下：

/\* decrement our usage count for the device \*/

--skel->open\_count;

if (skel->open\_count <= 0) {

/\* shutdown any bulk writes that might be

going on \*/

usb\_unlink\_urb (skel->write\_urb);

skel->open\_count = 0;

}

/\* decrement our usage count for the module \*/

MOD\_DEC\_USE\_COUNT;

Skel的write和read函数是完成驱动对读写操作的响应。在skel\_write中，一个FILL\_BULK\_URB函数，就完成了urb 系统callbak和我们的skel\_write\_bulk\_callback之间的联系。值得注意的是，skel\_write\_bulk\_callback是中断执行的，所以不能使用太久。read 函数与write 函数稍有不同在于：程序并没有用urb 将数据从设备传送到驱动程序，而是我们用usb\_bulk\_msg 函数代替，这个函数能够不需要创建urbs 和操作urb函数的情况下，来发送数据给设备，或者从设备来接收数据。我们调用usb\_bulk\_msg函数并传提一个存储空间，用来缓冲和放置驱动收到的数据，若没有收到数据，就失败并返回一个错误信息。

2.3 LibUSB介绍

Linux驱动可以编写成内核驱动，虽然有优点，但是应为Linux内核版本太多，有官方的，也有自编译的，所以一个内核驱动会被在各种内核下编译，导致大部分的工作时间是在编译。因此，就诞生了LibUSB这个C库。它提供了一系列的外部API为应用程序所用，应用程序通过这些API来操作硬件。由于LibUSB接近USB规范，所以使用它开发驱动节约时间，降低成本。

2.3.1 初始化设备接口

1. void usb\_init(void); 是用来初始化相关数据。

2. int usb\_find\_busses(void); 寻找系统上的usb总线，返回总线数。

3. int usb\_find\_devices(void); 寻找总线上的usb设备，在调用usb\_find\_busses()后使用，返回设备数量。

4. struct usb\_bus \*usb\_get\_busses(void); 返回总线的列表。

2.3.2 操作设备接口

1. usb\_dev\_handle \*usb\_open(struct \*usb\_device dev);

打开要使用的设备，在对硬件进行操作前必须要调用usb\_open 来打开设备。

2. int usb\_close(usb\_dev\_handle \*dev);

与usb\_open相对应，关闭设备，是必须调用的, 返回0成功，小于0 失败。

3. int usb\_set\_configuration(usb\_dev\_handle \*dev, int configuration);

设置当前设备使用的configuration，参数configuration 是要使用的configurtation descriptoes中的bConfigurationValue, 返回0成功，小于0失败

4. int usb\_set\_altinterface(usb\_dev\_handle \*dev, int alternate);

设置当前设备配置的interface descriptor，参数alternate是指interface descriptor中的bAlternateSetting。返回0成功，小于0失败

5. int usb\_resetep(usb\_dev\_handle \*dev, unsigned int ep);

复位指定的endpoint，参数ep 是指bEndpointAddress,。

6. int usb\_clear\_halt (usb\_dev\_handle \*dev, unsigned int ep);

复位指定的endpoint，参数ep 是指bEndpointAddress。

7. int usb\_reset(usb\_dev\_handle \*dev);

reset设备，因为重启设备后还是要重新打开设备，所以用usb\_close就已经可以满足要求了。

8. int usb\_claim\_interface(usb\_dev\_handle \*dev, int interface);

注册与操作系统通信的接口，只有注册接口，才能做相应的操作。Interface 指 bInterfaceNumber。

9. int usb\_release\_interface(usb\_dev\_handle \*dev, int interface);

注销被usb\_claim\_interface函数调用后的接口，释放资源，和usb\_claim\_interface对应使用。

2.3.3 控制传输接口

1. int usb\_control\_msg(usb\_dev\_handle \*dev, int requesttype, int request, int value, int index, char \*bytes, int size, int timeout);

从默认的管道发送和接受控制数据

2. int usb\_get\_string(usb\_dev\_handle \*dev, int index, int langid, char \*buf, size\_t buflen);

3. int usb\_get\_string\_simple(usb\_dev\_handle \*dev, int index, char \*buf, size\_t buflen);

4. int usb\_get\_descriptor(usb\_dev\_handle \*dev, unsigned char type, unsigned char index, void \*buf, int size);

5. int usb\_get\_descriptor\_by\_endpoint(usb\_dev\_handle \*dev, int ep, unsigned char type, unsigned char index, void \*buf, int size);

2.3.4 批传输接口

1. int usb\_bulk\_write(usb\_dev\_handle \*dev, int ep, char \*bytes, int size, int timeout);

2. int usb\_interrupt\_read(usb\_dev\_handle \*dev, int ep, char \*bytes, int size, int timeout);

2.3.5 中断传输接口

1. int usb\_bulk\_write(usb\_dev\_handle \*dev, int ep, char \*bytes, int size, int timeout);

2. int usb\_interrupt\_read(usb\_dev\_handle \*dev, int ep, char \*bytes, int size, int timeout);

2.4 WireShark 介绍

开发闭源硬件的USB驱动，需要了解其自有的协议和传输方式，需要用到USB数据采集器，不过一般这种仪器都价格昂贵，个人购买不太适当。而WireShark在Linux下可以对USB协议进行抓包，正满足我们的需要。

WireShark是一款非常出名的网络封包分析软件，前身是Ethereal。它图形界面的版本叫WireShark，命令行界面的叫TShark。使用WinPCAP作为接口，网卡进行数据报文交换。

WireShark会列出说抓取的每个网络封包尽可能详尽的信息，供开发人员，网络工程师等进行参考。其Windows版本界面如图：

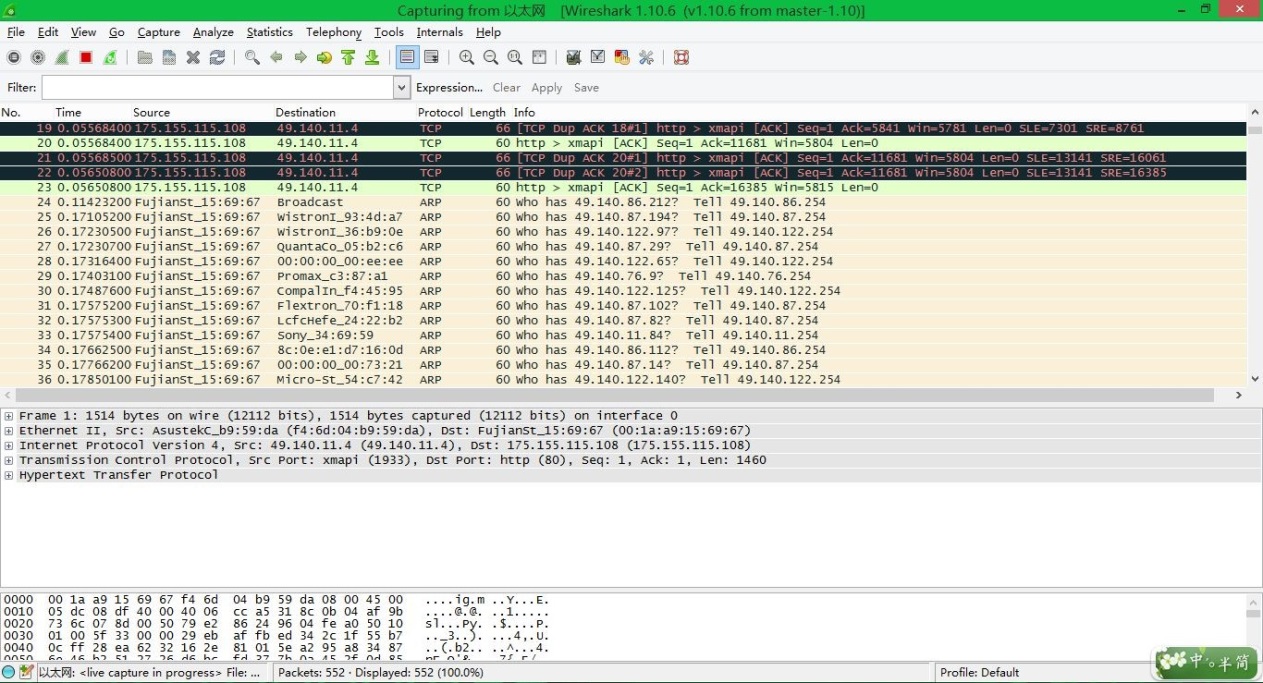


图 4 WireShark界面

2.5 NodeJS 介绍

NodeJS是一个新兴的使用Javascript进行开发的一个平台。它基于Chrome的Javascript运行时建立的，使用Google V8引擎执行Javascript，速度很快，性能非常好。

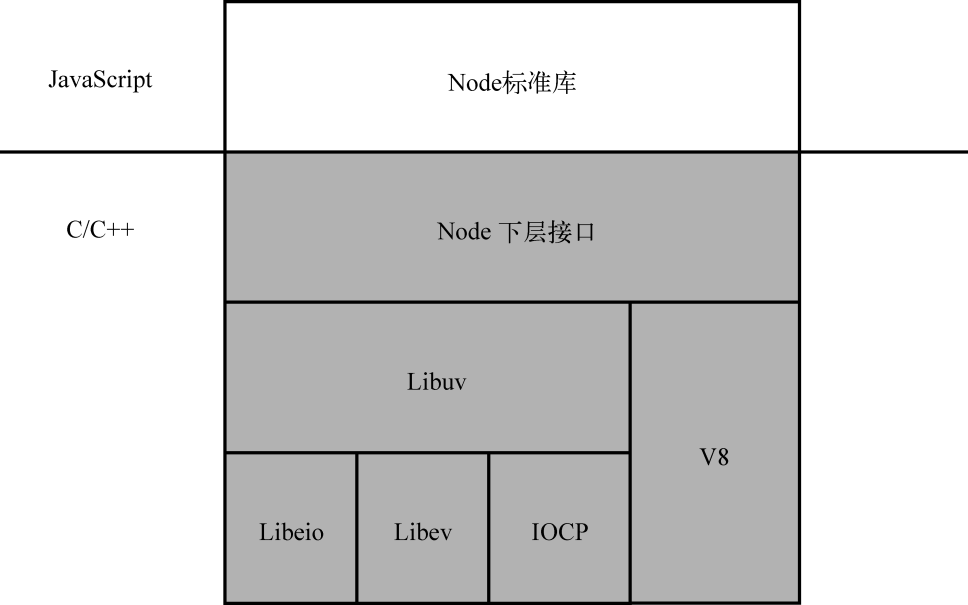
NodeJS虽然使用Javascript进行开发，但它仍能进行一些系统级的操作，包括I/O，网络等操作。它使用异步事件驱动，非阻塞的I/O模型，因此本身很轻量，高效，非常适合数据敏感，实时，I/O密集型的应用开发。虽然是单线程运行，但可以通过分布式部署的方式，实现良好的并行操作。底层使用libuv来实现针对不同平台的兼容，至于事件模型，在Linux下使用libev和libeio，在Windows下使用IOCP来完成，因此具有良好的跨平台性。

图 5 NodeJS 架构

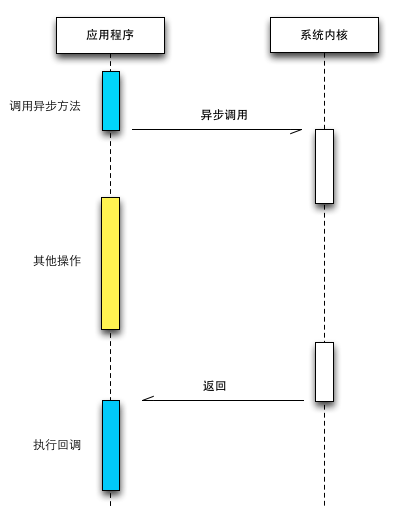


图 NodeJS 事件模型

同时，它拥有良好的包管理机制，使用模块对代码区分和封装。有NPM提供统一的包管理，对于开发者很友好，有强大的社区力量支持。

作者本人是前端工程师，所以采用NodeJS作为开发平台，前后端语言一致，开发时效率较高。

下面是NodeJS的一个代码片段：

var http = require('http');

server = http.createServer(function (req, res) {

res.writeHeader(200, {"Content-Type": "text/plain"});

res.end("Hello World\n");

});

server.listen(8000);

console.log("httpd start @8000");

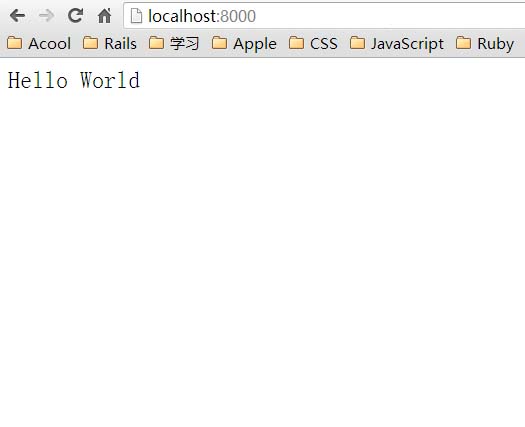
这段代码的作用是在本地开启一个http服务器，监听8000端口，当有访问时，返回"Hello World"。它通过回调的方式，来保证事件的执行，达到异步的效果。

图 7 NodeJS 示例

2.6 LeapMotion

LeapMotion是体感控制器制造公司Leap旗下的产品。它是一个手势识别控制器，具有较高的识别精度，同时也为开发者提供了良好的SDK。

图 8 Leap Motion

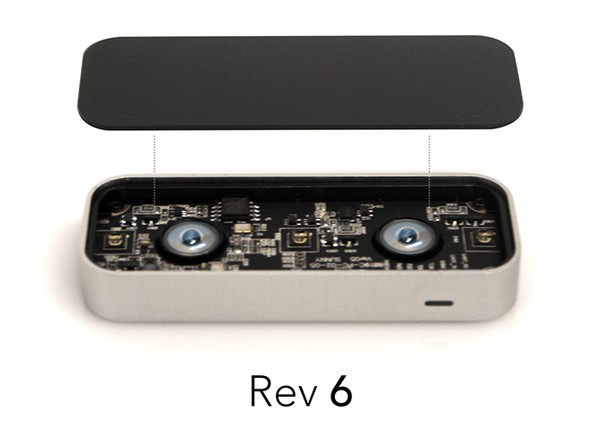
Leap Motion内置了两个灰阶摄像头和红外LED，从不同角度捕捉画面，重建出手掌在三维空间里的运动信息，其检测空间大体是一个倒四棱锥，位于传感器上方25mm到600mm之间。

图 9 Leap Motion传感器结构

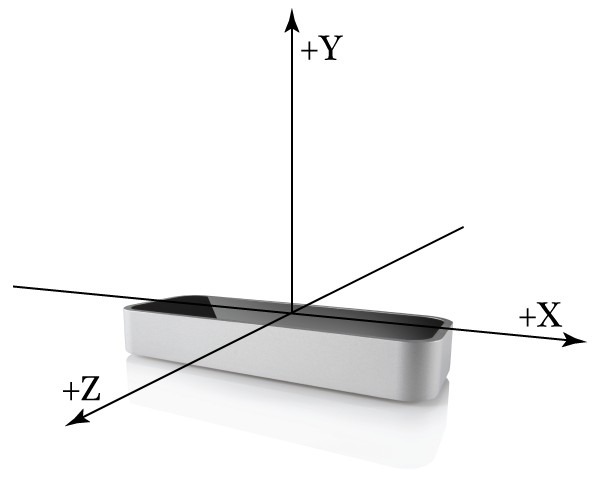
它的传感器会建立一个直角坐标系，以传感器的中心为坐标原点，X轴向右平行于传感器，Y轴指向上方，Z轴为背离屏幕的方向，识别单位为毫米。如图：

图 10 Leap Motion坐标系

在使用时，Leap Motion会定期向主机发送手的运动信息，成为帧。每一帧包含：

1. 所有手掌的列表及信息

2. 所有手指的列表及信息

3. 手指工具（比如笔，尺等）的列表及信息

4. 所有Pointable Object，即所有手指和工具的列表及信息

Leap Motion会给这些分配唯一的ID，当这些存在于它的视野范围内时，不会该变对应的ID。通过这些ID，才能使用Frame::hand()，Frame::finger()等函数来查询每个运动对象的信息。

Leap Motion可以根据当前帧和前帧所检测到的数据，生成运动信息。例如：如果检测到两只手，且都向一个方向移动，则视为平移；如果像是握球一样转动，则为旋转；如果两只手分开或者靠近，则为缩放。产生的数据包含：

1. 旋转的轴向量

2. 旋转的角度（顺时针为正）

3. 描述旋转的矩阵

4. 缩放因子

5. 平移向量

针对每只手，有如下信息：

1. 手掌中心的位置（三维向量）

2. 手掌的移动速度（mm/s）

3. 手掌的法向量（垂直于手掌平面，从手心向外）

4. 手掌的朝向

5. 根据手掌弯曲的弧度确定的虚拟球体的中心

6. 根据手掌弯曲的弧度确定的虚拟球体的半径

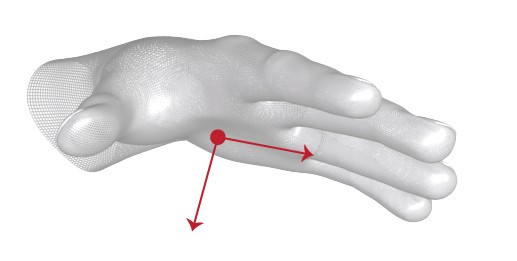
 其中，手掌的法向量和方向如图：

图 11 手掌法向量和方向

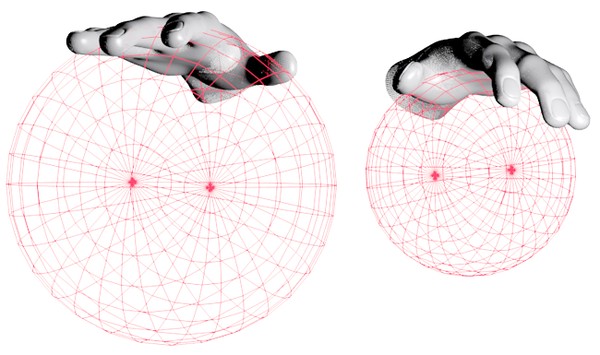
手掌虚拟球的圆心和半径：

图 12 虚拟球半径和圆心

针对每个手掌，也可检测出平移，旋转，缩放（手指张合）的信息，检测数据域全局变换一样，就不一一列举。

Leap Motion除了可以检测手指外，也可以检测手持的工具，如图：

图 13 手持工具检测

针对手指和工具，称为Pointable Object，每个Pointable Object包含：

1. 长度

2. 宽度

3. 方向

4. 指尖位置

5. 指尖速度

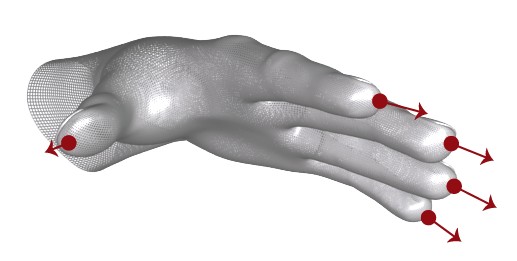
方向和指尖位置如图：

图 14 手指检测

根据全局的信息，运动变换，手掌，手指和工具的信息及变换，开发者就可以开发响应的使用手势识别的应用。

Leap Motion的数据采集是通过摄像头实现的，其主要核心部分就在于其附带的软件上，包括设备的初始化，原始数据的解析，API等，下文会讨论到。

2.7 HTML5

超文本标记语言的第五重大修改版本，是一个正在进行中的草案，不过已经有很多现代浏览器提供了相应了支持。相比于HTML4来说，HTML5提供了更多的多媒体标签，加强了标签语义化的感念，推动了Web标准化运动的发展。主要有以下几个特性：

2.7.1 语义特性

HTML5 完善了很多的标签，赋予了网页良好的结构和意义。增加的标签随着对数据，格式，残障，RDFa等方面的支持，构建了更加有价值的数据驱动的Web。

2.7.2 本地存储特性

使用HTML5开发的Web APP因为有HTML5 APP Cache和本地存储功能，所以能够更快的启动，更快的响应。

2.7.3 设备兼容特性

HTML5支持Geolocation功能，即地理位置定位，这种功能的加入，为开发者提供了更多的选择，为应用增添更多的体验。HTML5提供了数据域应用接入的开放接口，使得外部应用可以与浏览器内部的数据相连，比如视频可以与摄像头相连。

2.7.4 连接特性

HTML5拥有更好的服务器推送技术，Server-Sent Event和WebSocket就是其中的两个特性，它们能够实现服务器将数据“推送”到客户端的功能，使得基于Web的实时聊天，网页游戏的体验更良好，快速。

2.7.5 网页多媒体特性

HTML5提供了对音乐，视频支持的Audio，Video标签，同时可以调用APPS，摄像头等，使得用户体验更加丰富。

2.7.6 三维、图形及特效特性

HTML5提供基于SVG、WebGL、Canvas及CSS3的3D功能，能在浏览器中为用户呈现令人惊叹的视觉效果。

2.7.7 性能与集成特性

HTML5通过XMLHttpRequest2等技术，减少用户的等待时间，新的XHR对象，能够解决以前AJAX的问题，使得网站更加多样化，快速。

2.7.8 CSS3特性

在结构和性能不变的情况下，CSS3增加了更多的风格和更强的效果，Web排版变得更加轻松，绚丽。WebFont，Web开放字体的应用，使网站更加多样化。

2.8 CSS3

CSS（层叠样式表），是在制作网页过程中，对网页的布局、字体、颜色、背景和其他效果进行精确控制的技术。将网页表现和结构分开，通过代码对网页外观进行控制。CSS3是CSS的最新版本，拥有模块化的概念，更多的功能和对网页更好的控制。新增了：盒子模型、列表模块、超链接方式、语言模块、背景和边框、文字特效、多栏布局、弹性盒、选择器、2D/3D转换、动画等。在对元素的选择上，CSS3有着更灵活，准确的语法，能够方便准确的控制特定的元素。

CSS3的2D/3D转换和动画功能，使网页更加具有表现力，能够展现出更多的特效，更准确的突出主题。这些动画效果，也大大减少了网页对于图片，flash等这些元素的依赖，使网页更小，更轻。同时也减少了对Javascript的依赖，很多的效果用CSS3就能实现，这样网页能够变的更快速。

以下的代码实现了一个不用图片的圆角矩形（仅在Chrome中测试）：

<div class="box">BOX</div>

.box{

width: 200px;

height: 200px;

border-radius: 20px;

border: 1px #000 solid;

background-image: -webkit-gradient(linear, left top, left bottom, color-stop(0, #02CCFF), color-stop(1, #00508F));

box-shadow: 5px 5px 10px rgb(139, 136, 136);

text-align: center;

line-height: 200px;

font-size: 40px;

color: #fff;

text-shadow: #29FF00 0px 0px 10px;}

.box::first-letter {

text-transform: lowercase;

}

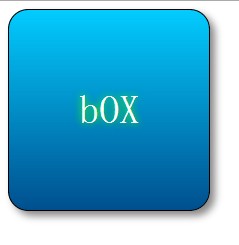
效果如下图：

图 15 CSS3示例

使用Css3 Transform：

-webkit-transform:rotate(45deg);

-webkit-transform:skew(20deg, 20deg);

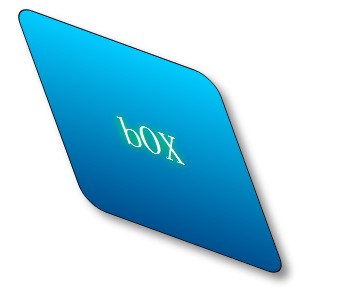


图 16 CSS3 transform

本课题的游戏中会大量运用到CSS3的特性，下文会详细讨论。

第3章 基础架构的设计

3.1 USB over IP 设计

3.1.1 系统架构

这个实现方案中的USB over IP部分组件，从功能上可以分为服务端和客户端两个部分。服务端将设备共享到网络上，客户端通过搜索服务端，即可得到共享设备的列表，然后做映射，就可以在客户端上使用服务端的设备。其中，底层的URB（USB数据请求包）被封装成IP包，在网络中流通。

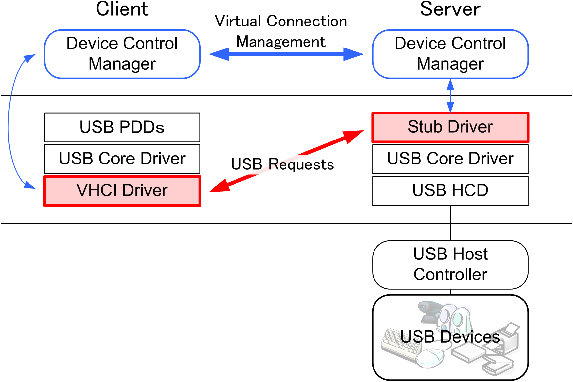
系统结构图如下：

图 17 USB over IP系统设计

两端的通信还有由TCP/IP协议完成的，这种稳定性的连接，能够尽力保证数据的完整性。服务器端通过增加一个Stub Driver来将自己的设备映射到客户端，客户端则通过增加一个VHCI Driver，来完成URB的发送和反馈，对系统原来的USB core不做改动。

3.1.2 IP封装策略

在这个系统中，我们增加了一个VHCI（虚拟主机控制器接口）驱动，它是USB HCD的等价物，负责处理排队的URB。URB被转换成USB/IP请求块，有VHCI发送到远程机器上。Stub Driver等价于一个USB PDD，将传来的USB/IP数据包进行解包，提取URB，然后交给本地USB设备。

3.1.3 服务端设计

3.1.4 客户端设计

3.2 Leap Motion Initialize driver