**Leap Motion 官方中文开发文档**

目录

[Leap Motion 官方中文开发文档 1](#_Toc18205)

[目录 1](#_Toc8600)

[Leap Motion概述 3](#_Toc27455)

[坐标系统 3](#_Toc22003)

[运动追踪数据 4](#_Toc1552)

[帧 4](#_Toc3969)

[手模型 5](#_Toc9860)

[手指和工具模型 7](#_Toc1381)

[手势 7](#_Toc815)

[点击 9](#_Toc26621)

[Leap Motion结构 10](#_Toc22212)

[程序编程接口 11](#_Toc19403)

[Leap Motion的应用接口 11](#_Toc7634)

[Leap Motion网页套接字接口 11](#_Toc29910)

[编程语言支持 12](#_Toc10034)

[C++ 12](#_Toc19924)

[Objective-C 12](#_Toc13561)

[C# 13](#_Toc29147)

[Java 13](#_Toc6640)

[Python 13](#_Toc9329)

[JavaScript 13](#_Toc24545)

[其它编程语言 13](#_Toc13097)

[操作系统的支持 13](#_Toc9530)

[Leap Motion用户体验指南 13](#_Toc21817)

[使用Leap Motion可视化工具 14](#_Toc5333)

[概述 15](#_Toc169)

[一般控制 15](#_Toc12514)

[摄像头控制 16](#_Toc10763)

[追踪信息 16](#_Toc12871)

[Leap Moition体感器设置对话框的使用 16](#_Toc19162)

[概述 17](#_Toc2044)

[菜单命令 17](#_Toc11253)

[Leap Motion设置 17](#_Toc7711)

[高级设置 18](#_Toc7377)

[操作模式 18](#_Toc4826)

[系统交互 18](#_Toc17748)

[设备校准 19](#_Toc28499)

[汇报漏洞 19](#_Toc28542)

[自动漏洞汇报 20](#_Toc27963)

[应用日志 20](#_Toc20285)

[获取帧数据Getting Frame Data 20](#_Toc21944)

[概述 20](#_Toc16944)

[通过轮询获取帧 21](#_Toc11455)

[通过回调函数获取帧 21](#_Toc3612)

[从帧中获取数据 22](#_Toc13336)

[使用IDs来追踪帧间实体 22](#_Toc7259)

[追踪手、手指和工具 23](#_Toc27550)

[概述 23](#_Toc24987)

[手和尖端列表 23](#_Toc8324)

[手 25](#_Toc23706)

[获取手特征 25](#_Toc27546)

[获取指尖和工具 26](#_Toc11206)

[计算手的朝向 26](#_Toc16107)

[将手指坐标变换到手的参照帧（以手为参照对象） 26](#_Toc30960)

[尖端 27](#_Toc25906)

[将尖端对象转化成手指或者工具 28](#_Toc13164)

[计算手指基本位置 28](#_Toc10732)

[触摸仿真 28](#_Toc5203)

[概述 28](#_Toc22297)

[获取触摸地带 29](#_Toc1480)

[获取触摸距离 29](#_Toc24041)

[获取尖端的稳定坐标 29](#_Toc3435)

[从Leap Motion坐标系转换到应用坐标系 30](#_Toc27110)

[触点例子 30](#_Toc7008)

[屏幕定位 Screen Location 32](#_Toc23133)

[Leap Motion 2.x API的变化 33](#_Toc27287)

[增加直接获取Leap近红外图像的API函数，Frame.Images()。 33](#_Toc15373)

[全新的骨骼追踪模型 34](#_Toc17454)

[关于Hand的改变 34](#_Toc28015)

[关于Finger的改变 35](#_Toc18225)

[新增Bone类 35](#_Toc10766)

[新增Arm类 35](#_Toc7934)

这是C++语言下的Leap Motion引导文章，也适用于其他语言，原文地址：<https://developer.leapmotion.com/documentation/Languages/C++/Guides/Leap_Overview.html>，我使用中括号来标记自己的想法和揣测。由于本人英语水平凡，翻译这篇文章来锻炼英语，有能力的人最好阅读英文全文。

由于LeapMotion官网越来越卡，我将最新的SDK下载后，上传到百度网盘里，你可以[http://pan.baidu.com/s/1gfQeLPp](http://guoming.me/sdk) 或 [https://www.leapmotion.com/setup/desktop/windows](http://pan.baidu.com/s/1jGrCHds#path=%252FLeapMotion) 下载最新的驱动及SDK。

初学者下载SDK后，在samples文件夹下就有各种LeapMotion的代码，如C++、Java、Python、CSharp、js等等。而在docs文件夹下就有完整的英文参考文档，本文就是取自该文

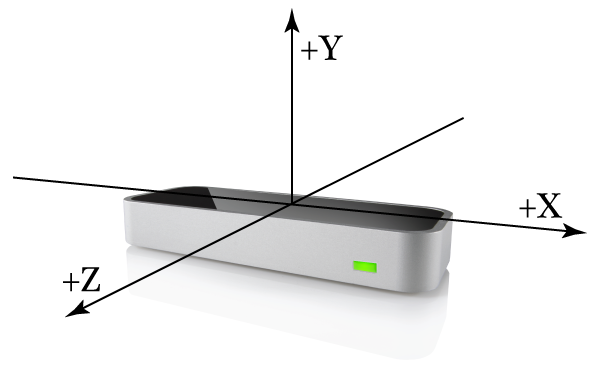
# Leap Motion概述

Leap Motion系统可以检测并跟踪手、手指和类似手指的工具。这个器件可以在高精确度和高跟踪帧率下工作。

Leap Motion软件分析在器件可视范围内的物体。它识别手、手指和工具，可以实时获取它们的位置、手势和动作。Leap Motion的可视范围是一个倒金字塔，塔尖在设备中心。[这个很好理解，传感器一般都这样]Leap Motion的可工作范围大约在设备前方的从25到600毫米，也就是1英寸到2英尺。[2.5厘米到0.6米，果然弥补了Kinect近距离的不足啊，十分适坐在电脑前操作]

## 坐标系统

Leap Motion的系统采用了右手笛卡尔坐标系。返回的数值都是以真实世界的毫米为单位[和Kinect的深度数据一样一样的]。原点在Leap Motion 控制器的中心。x轴和z轴在器件的水平面上，x轴和设备的长边平行[z轴和短边平行，挺好记]。y轴是垂直的，以正值增加形式朝上（与朝向下的计算机图形学的坐标系相反）。距离计算机屏幕越远，z轴正值不断增加。[看到那个小绿灯吧，得确保让它朝着我们这个坐标系才对，摆放时得注意！]



上图：Leap Motion的右手坐标系统

## 运动追踪数据

由于Leap Motion设备最终在它视野中的手、手指和工具，它提供一组数据集更新，或者是帧，或者是数据。每帧数据包含一个基本追踪数据列表，如手、手指和工具，也包括识别出的手势和描述场景中的运动因素。

但检测到手、手指和工具或手势时，Leap Motion'软件为它分配一个唯一的ID指示符。只要这个实体一直存在于设备可视范围内，这个ID指示符就保持不变[和Kinect的骨骼追踪的ID是一致的]。如果追踪目标丢失或者失而复得，Leap Motion软件会分配一个新的ID（软件无法知道手、手指是否和之前看到的一样）[也就是说，不包含手指识别啦，和Kinect的骨骼追踪在丢失后情况完全一致]。

### 帧

一个帧对象提供追踪数据、手势和在Leap Motion可视范围内的整体运动因素的列表。

#### 追踪数据列表

Hands手---所有的手。

Pointables有端点的---手指和有段点的工具。

手指---所有的手指。

工具----所有的工具。

手势---所有手势的开始、结束或者哪个进行了更新。

这三个具有端点的列表（Pointables，Fingers，Tools）包含了任何在数据帧检测出的有端点的对象。你可以通过访问手的列表，来获取手中物体的信息。需要注意的是，如果用户的手只在Leap Motion的视野中出现一部分，那么手指或者工具都无法与手关联。[也就说，注意的手的摆放，确保手和工具同事都在可视范围内]

如果你正在通过一帧又一帧的来追踪一个单一的物体，例如手指，你可以使用与物体关联的ID，在新的帧中查询它。通过ID，你可以查询手、手指、工具、端点物体和手势。如果这个物体在当前帧存在，那么查询函数返回一个关于物体的引用。如果物体不存在了，那么返回一个特别的无效对象。无效对象被很好定义，但不包含任何有效的追踪数据。这个技术使得我们在使用Leap Motion追踪数据时，简化了大量的对于空指针的检测。

#### 帧运动

Leap Motion软件分析总体运动，只要之前帧数据发生了位移、旋转、尺度变化等。例如，如果你把双手同时移动到Leap Motion的左侧视野，帧就包含了位移变化。如果你扭动你的双手，好像旋转一个球，帧就包含旋转。如果你将双手靠近或者远离，帧就包含缩放信息。Leap Motion软件会使用视野范围内的物体，只要分析到发生了运动。如果它只检测到一只手，那么Leap Motion程序就会基于那只手的运动，给出帧运动因素。如果它检测出一双手，它就将双手的运动趋势结合起来，给出运动因素。通过每只手对应的手对象，你也可以获取单独运动参数。

帧运动信息的产生是通过比较当前帧与之前一个特别的帧。描述合成运动的属性包含：

1.旋转坐标Rotation Axis----一个方向向量来描述坐标的旋转。

2.旋转角度Rotation Angle----相对于旋转坐标（笛卡尔坐标系）的顺时针方向的旋转角度。

3.旋转矩阵Rotation Matrix-----一个旋转的矩阵变换。

4.缩放因子Scale Factor------一个因子来描述膨胀和收缩。

5.位移Translation-----------一个向量来描述线性运动。

你可以通过运动因子来操控在你应用场景中的物体，而不必在多帧数据中跟踪单独的手和手指。[也就是factors可以手动修改]

帧结构包含一个估计某个方向上的特定运动。例如，如果缩放因子很大，难么你可以忽略在帧中的旋转、或位移（如果这可以让你的应用更实际）。通过概率方法滤去除掉不想要的运动信息可以使你的程序更容易被使用。[也就是说，只关注最主要的运动，忽视那些次要的变化，毕竟人不是机器，例如做旋转可定发生一定位移]

### 手模型

手模型提供关于手，或者其它被检测出来的手指、工具的坐标、特征和运动。

Leap Motion的接口函数尽可能多的提供关于一个手的信息。当时，Leap Motion程序可能无法计算出在每个帧下的手部信息。例如，当一个手攥成一个拳头，它的手指则无法被Leap Motion看到，所以手指的信息就是空。你的应用应该能处理到这些模型消失的状态。

Leap Motion程序也不判定一只手到底是左手还是右手[Kinect判断~]。在手的列表中，可以出现多余2只手的信息，因为可以不止一个人也可以出现一个类似手的物体在Leap Motion的视野范围里。但是，我们建议最多让两只手同时出现在Leap Motion控制器的视野里，这样可以确保最佳追踪质量。

#### 手属性

手对象提供一些属性来反映一只被检测到手的物理特征。

1. 手掌坐标Palm Position-------------在Leap Motion的坐标系下，手掌中心的坐标以毫米为单位被衡量。

2.手掌速率Palm Velocity-------------手掌毫米/每秒的运动的速度。

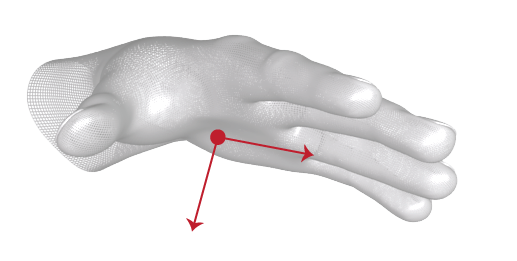
3.手掌法线方向Palm Normal--------------与手掌所形成的平面的垂直向量，向量方向指向手掌内侧。

4.方向Direction-------------------由手掌中心指向手指的向量。

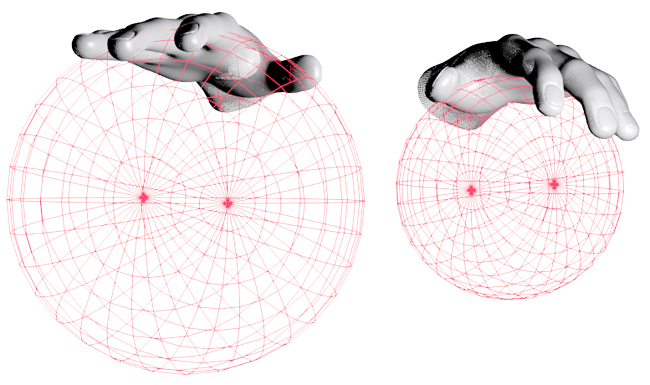
5.球心Sphere Center------------可以适合手掌内侧弧面的一个球心。（假设握着一个球）

6.球半径Sphere Radius------------同上，这个是球半径。当手形状变化，半径跟着变化。

方向和手掌法线方向是在Leap Motion坐标系下，描述手的方向的向量。



上图：手掌标准垂直于手掌往外，方向朝着手指方向。



上图：当手卷曲，球变小

球心和球半径描述了一个球，这个球满足手掌的曲率，正好可以被手掌的握着。

#### 手运动

手对象还提供了一些用于描述手运动的属性。Leap Motion程序分析手的运动，包括那些与手关联的手指、工具的位移、旋转和缩放。将你的手绕着Leap Motion的运动，会产生位移。张开、扭曲和倾斜你的手，可以产生旋转。将你的手势或者工具对着Leap Motion靠近或远离，可以产生缩放。

手的运动是通过当前帧与之前特定帧对比得到的。描述手运动的属性有和Frame Motion一样。

1.旋转坐标Rotation Axis

2.旋转角度Rotation Angle

3.旋转矩阵Rotation Matrix

4.缩放因子Scale Factor

5.位移Translation

手对象同样包含了对于手运动最重要的估计。[后面一样了，不说了]

#### 手指和工具列表

你可以通过下面三个中的一个，访问与手联系的手指和工具：

1.端点Pointables------手指和工具都是端点物体。

2.手指Fingers----------仅仅手指。

3.工具Tools-------------仅仅工具。

你还可以通过之前帧中获取的ID来访问单独的手指和工具。使用Hand::finger()，hand::tool()，或者你不需要区分手指和工具，使用Hand::pointable()函数。这些函数返回当前帧中物体的引用。如果手指和工具在此帧中，没有和手关联，那么返回一个无效的对象。

### 手指和工具模型

Leap Motion设备不但检测和追踪在视野范围里的手指还追踪工具。Leap Motion程序通过形状对手指类似物体进行分类。一个工具相对于手指来说是更长、更瘦还是更直。

在Leap Motion模型里，手指和工具的物理特征被抽象到一个端点对象中。手指和工具是一类端点对象，端点对象的物理特征包括：

1.长度Length-------物体的可视长度（从手长出来的部分）。

2.宽度Width-------物体的平均宽度。[注意平均哦]

3.方向Direction---一个单位朝向向量，方向与物体指向相同。（例如从底座到尖端）

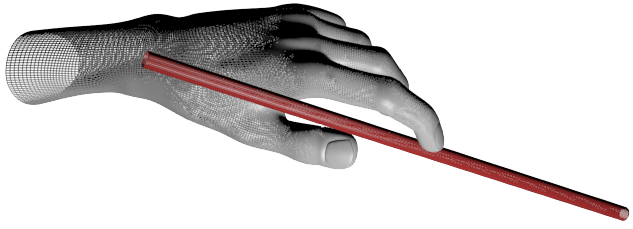
4.尖坐标Tip Position----在Leap Motion坐标系下，尖的位置。

5.尖点速率Tip Velocity-----尖的运动毫米/秒。



上图：红点还是尖坐标，指针式尖方向。

Leap Motion程序把检测到的端点物体识别为手指或者是工具。使用Pointable::isTool()函数来确定是哪一类端点对象。



这个工具，比手指更长、更细和更直。

### 手势

Leap Motion程序把特定的运动模式识别为手势，可以猜测用户的意图或指令。Leap Motion对于一帧数据中的手势访问方法和其它的手指和手势一致的。对于每个被检测到的手势，Leap Motion都将一个手势对象Gesture object添加到帧数据中。你可以通过帧手势列表来获取手势对象。

以下是Leap Motion可以识别的运动模式。[重点来了啊，看不懂下面会有图像示意~]

1.圈Circle----一个手指头画圆。

2.挥动Swipe----手的线性运动。

3.按键点击按键Key Tap-----手指点击运动，就像按下键盘一样。

4.屏幕点击Screen Tap----对电脑屏幕方向进行垂直点击。

但Leap Motion把一个运动识别测很难过一个手势模式，它把手势对象加入帧中。如果手势重复数次，Leap Motion会把更新手势对象不断添加到随后的帧中。画圈和挥手的手势都是持续的。Leap Motion为程序在每帧中持续更新这些手势。点击时离散的手势。Leap Motion把每次点击作为独立的手势对象报告。

重要：在你的应用使用手势之前，你必须把每种所需要识别的手势开启。控制类有一个enableGesture()方法让你开启特定的手势识别。[也就是一开始都是关闭的，不开没法用]

#### 画圈

Leap Motion可以识别手指在空中以圆的方式运动，返回一个圈手势Circle gesture。



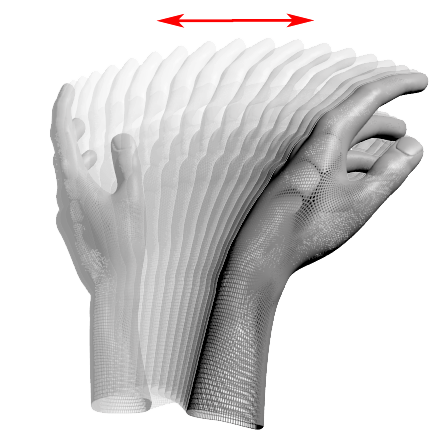
上图：食指的圈手势

你可以用任何手指和工具画圈。圈手势是持续的。一旦这个手势开始，Leap Motion就会持续更新状态直到停止。但手指或者工具远离了轨迹或者运动太缓慢，圈手势则终止。

可以从CircleGesture的接口函数参考中看到更多的信息。

#### 挥手

Leap Motion把手指的线性运动识别为挥手手势。



上图：一个水平挥手的手势。

你可以用任意手指在任意方向上作挥手手势。挥手手势是持续的。一旦手势开始，Leap Motion机会更新状态直到手势结束。当手指变换了方向或者运动太缓慢时，挥手手势结束。

更多信息可以参考SwipeGesture的接口函数文档。

### 点击

Leap Motion可以识别2种点击，按键点击Key Tap和向前的屏幕点击Screen Tap。

#### 按键点击

Leap Motion把一个快速的、往下一根手指或工具的运动识别为一个按键手势Key Tap。



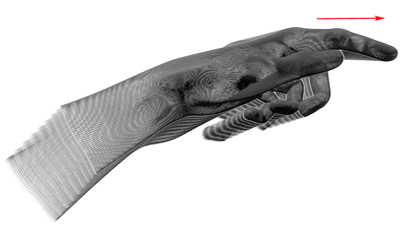
上图：食指的按键手势

你可以像按下钢琴键那样产生一个按键点击手势。按键点击手势时离散的。只有一个独立的手势对象会被添加到点击手势。

从KeyTapGesture接口函数可以看到更多的信息。

#### 屏幕点击

Leap Motion可以把一个手指或者工具做一个快速的、朝前的点击识别为一个屏幕点击手势。



上图：食指的屏幕点击手势

你可以往前点击或者把手推向前方来产生一个屏幕点击，就像触摸一个与你垂直的屏幕。点击手势是离散的，只有一个独立的手势对象会被添加到点击手势。

可以在ScreenTapGesture的接口函数看到更多的信息。

【十分不建议使用屏幕点击动作，根本太难触发了！！】

# Leap Motion结构

Leap Motion体感控制器支持最流行的桌面操作系统。Leap Motion程序作为一个服务（在Windows中）或一个守护进程（在Mac和Linux中）。这个程序通过USB总线和Leap Motion体感控制器相连。基于Leap的应用程序通过Leap Motion服务来获取运动追踪数据。Leap Motion的SDK提供两类API来获取Leap Motion的数据。这些API可以让你在多个语言下开发基于Leap的应用程序，包括在浏览器中运行JavaScript语言。

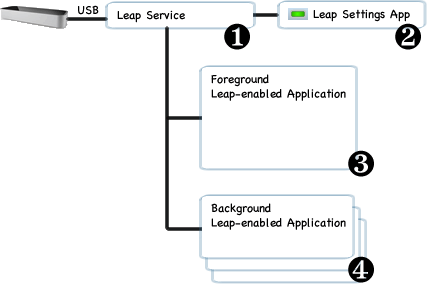
注意：Leap Motion 服务/守护进程通过TCP端口：127.0.0.1:6437/6438/6439来与应用程序通信。这个地址-端口必须不是被防火墙或其它程序屏蔽的。

## 程序编程接口

Leap Motion SDK提供两类API来从Leap Motion服务中来获取追踪数据，一个是原生接口，一个是网页套接字WebSocket。原生接口是一个可以让你创建新的Leap应用程序的动态库。而网页套接字接口和JavaScript客户端库使你可以创建基于Leap的应用。

### Leap Motion的应用接口

应用程序接口通过动态链接库提供。这个库连接到Leap Motion服务，并且为您的程序提供追踪数据。你在C++和Objective-C开发的应用中可以直接链接到库中，或者通过Java，C#和Python的语言绑定链接。



上图是：基于Leap的应用程序

1.Leap Motion的服务通过USB总线，从Leap Motion控制器接收数据。它[这个服务]处理这些信息，然后发送到正在运行的Leap应用。默认情况下，服务仅仅向前台应用程序发送追踪数据。但是，应用程序可以通过询问方式，使得他们从后台也可接收数据（询问可以被用户否定）。

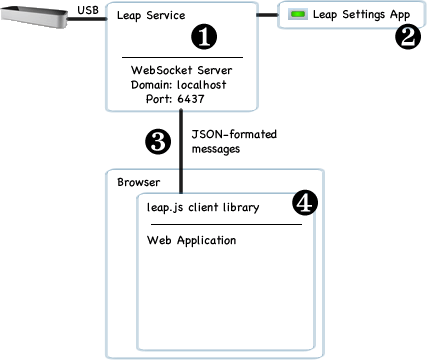
2.Leap Motion的设置对话框是和服务独立的，它允许计算机用户配置Leap Motion的安装。在Windows下，这个配置程序是个控制面板小程序，而在Mac X操作系统下是个菜单条。

3.前台Leap Motion应用程序从服务接受运动追踪数据。一个Leap应用程序可以通过原生Leap Motion库链接到Leap Motion服务。你的应用程序既可以通过C++和Objective-C直接连到原生库上，也可以通过语言包装库（Java，C#和Python）链接。

4.当一个Leap应用程序失去了操作系统焦点，Leap Motion服务则停止向它发送数据。经过请求许可，应用程序可以在后台接收数据。当运行在后台时，由前台程序配置。

### Leap Motion网页套接字接口

Leap Motion的服务以网页套接字（WebSocket）在本地域6437端口下运行。网页套接字接口通过JSON信息提供追踪数据。一个JavaScript本地库可以接受JSON消息，并且把追踪数据作为普通的JavaScript对象。



上图：Leap网页应用程序

1.Leap Motion服务提供网页套接字服务，监听[http://127.0.0.1:6437](http://127.0.0.1:6437/)。

2.Leap Motion的设置对话框运行用户启动或禁止网页套接字服务。

3.服务端以JSON信息形式发送追踪数据。一个应用程序可以向服务器返回配置信息。

4.leap.js客户端JavaScript库应该在网页应用中使用。这个库建立与服务器的链接，并且接受JSON信息。有JavaScript库提供的API与原生API的结构类似。

这个接口主要目的是用于网页应用，但是任何可以建立网页套接字的应用程序都能使用它。这个服务符合[RFC6455](http://tools.ietf.org/html/rfc6455)。

## 编程语言支持

Leap Motion库是由C++编码的。Leap Motion还是用了SWIG，一个开源工具，来生成C#，Java和Python的语言绑定。SWIG生成的绑定可以翻译绑定语言对于C++ Leap Motion库的调用。每个SWIG绑定都使用两个附加的库。对于JavaScript和网页程序的开发，Leap Motion提供了一个网页套接字服务和一个客户端JavaScript库。

在开发Leap应用程序和插件时，需要的所有的库、代码和头文件都包含在Leap Motion的SDK里，除了leap.js的客户端JavaScript库。你尅通过Leap Motion开发门户网站来下载Leap Motion SDK。对于任何支持的操作系统，SDK压缩包都有提供。JavaScript客户库是独立的，可以从[LeapJS GitHub repository](https://github.com/leapmotion/leapjs)下载。

### C++

用C++开发Leap Motion控制器应用，你需要在你的程序中，加入API头文件并且链接上Leap Motion库，根据操作系统，可能是libLeap.dylib，Leap.dll或者是libLeap.so等。

### Objective-C

Objective-C应用由手写的封装代码提供。为了建立一个Objective-C的Leap应用程序，你需要在程序中包含C++头文件，然后是封装头文件和Objective-C++代码文件。你可以使用在封装中定义的纯Objective-C类。将你的程序链接上libLeap.dylib并且导入在你应用程序包下的库。

### C#

C#类的定义由独立的库分别提供.NET 3.5和.NET 4.0环境。你的代码既可以参考LeapCSharp.NET3.5.dll，也可以参考LeapCSharp.NET4.0.dll（在所有支持的操作系统下，都使用同样名称的库）。这些库加载libCSharp.dylib（Mac），LeapCSharp.dll（Windows）或者libLeapCSharp.so（Linux）。中间库加载libLeap.dylib，Leap.dll或者libLeap.so（依赖于系统）。

使用.NET3.5库来进行Unity 3D游戏的程序开发。Leap Motion SDK提供了一个Unity开发环境的”Pro”版本的Unity插件。查看知识库的文章，确认各个版本的Unity editor是否支持关于开发Leap应用的插件。

### Java

Leap.jar包含Leap Motion的Java类。这个代码载入libLeapJava.dylib（Mac），LeapJaval.dll（Windows）或者libLeapJava.so（Linxu）。这些包含原生代码的库可以将Java函数的翻译成在libLeap.dylib，Leap.dll或者libLeap.so的基本Leap Motion API。基本Leap Motion动态库可以通过中间库载入。

### Python

Leap.py包含Python模块，你可以在你的Python应用程序中使用。这个模块加载LeapPython.so（Mac和Linxu）或者LeapPython.dll（Windows）。这些库加载libLeap.dylib，Leap.dll或者libLeap.so。

### JavaScript

Leap Motion JavaScript有两个重要组成部分。第一个部分是网页套接字服务器，由Leap Motion服务提供。这个服务器允许网页应用（或者任何可以链接到网页套接字的程序）来获取以JSON格式信息的Leap Motion追踪信息。第二个部分是JavaScript客户端库，Leap.js。Leap.js是一个开源JavaScript API可以接受网页套接字JSON输出，并且以物理上和结构上都与原生Leap MotionAPI类似的形式呈现。

### 其它编程语言

其它社会创建的语言绑定版本可以从[链接](https://developer.leapmotion.com/links)看到。

## 操作系统的支持

Leap Motion程序目前支持OS X 10.6和Windows 7+。还有一些试验中的alpha版本的Linux SDK提供给开发者。

# Leap Motion用户体验指南

接下来的用户体验指南，包含了我们对于你的Leap Motion应用的建议，确保它更容易学习和使用。

#请记住，符号可能很难学习和记忆。

@避免强迫用户去学习复杂的手势来与你的应用交互。

#反而，从物理交互和现实世界的行为中汲取灵感。

@越多的模仿物理交互，用户需要学习的越少，你的应用也会被认为更直观和自然。

#不要觉得受到现实世界的限制和不便之处----这只是你的世界。[既要基于现实世界，又要有合理的超越。]

@交互不需要像以前一直那样。它可以如你想象的那样。为什么强迫用户伸手出去，自己抓取到物体？为什么不让物体自己过来？让它们“有动力”。

#用户应该觉得他们的意念被放大了，而不是被压迫和屏蔽。

@例如，但使用鼠标时，用户经常希望他们的动作被增强（比如，他们不需要把鼠标移动10英尺，使得它们在屏幕上同样也移动10英尺）。对于手势交互，放大或者夸大这些反馈可以获得更积极的结果。牢记，有些人会比其他人更敏感，所以把这种夸大的反馈放入敏感设置中，这样用户可以根据他们自己的体验调整。[好吧，我觉得更应该直接让用户做一些手势，程序主动计算出这个用户适合的敏感值。再附带这个敏感值修改~]

#专注于如何提供给用户动作的实时动态反馈。他们越是收到越多的反馈，他们越能与你的软件交互。

@例如，当“前推”一个按钮，用户是应该知道他们按下了。但是，他们若知道什么时候把手悬浮在一个按钮上，或者能知道自己到底按下多少，这更有效。[这个前推动作，微软的Kinect做的很好哦]

#在屏幕上的视觉效果（比如手、工具或数字反馈的呈现）应该简约、实用、非侵入[不需要触碰]。

@用户不应该受到在任务中他们的工具或环境的影响。视觉效果不应该让他们分心。

#对于破坏性的或不可逆的行为的思考应该多于有害行为。

@微妙的手势应该保留细微的动作。相反，诸如关闭应用或者删除一个文件的行为是不可逆的，需要一个谨慎的手势。当对于用户的反馈不是很确定时，需要反复考量，比如如何提示确认。

#在导航和交互行为之间，提供一个明确的区分，除非它们都很简单，或者一个是自动处理的（或者在帮助下）。把它们混合在一而过复杂的情景下，会导致用户的迷惑和迷失。

@例如，在一个三维环境下，移动一个物体同时定位他的视角，本质上是很难的。但是，如果根据用户的运动，视角自动调整[日本的3D游戏大多都这样，比如波斯王子、鬼武者、鬼泣等]，那么操作这个物体就更简单了。同样的，当导航一个大的数据集时，用户希望有一个方便的视角。但是，当突出数据的某一部分时，视角应该保持静止。

#总体上，设想你的用户在没有任何指示或教程下是用你的应用。

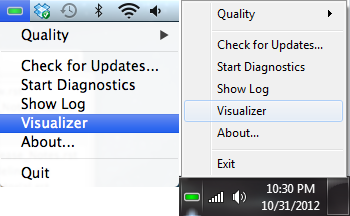
@不惜一切代价，让他们的第一直觉的猜测是正确的。在适当的情况下，创建更多的方式来做其他事情。

# 使用Leap Motion可视化工具

这篇文章描述Leap Motion可视化工具应用程序（Visualizer），它可以让你看到Leap Motion控制器捕获的运动追踪数据。

## 概述

可视化工具应用程序展示指尖追踪数据，并且是一个感受Leap如何产生数据的好途径。在Windows开始菜单或者Mac的查找条中，可以打开Leap Motion菜单中的可视化工具。



上图：从Leap Motion的图标菜单里运行可视化工具。

现在你可以在Leap Motion控制器可视范围里移动下自己的手指。你可以看到的手指被显示成彩色笔的样子，以你的指尖端点。

## 一般控制

你可以控制键盘命令控制可视化工具。按下H键显示可以操作的命令。

#Esc --- 退出程序。

#H --- 显示可视化工具的帧率、Leap Motion的帧率以及按键命令。

#S --- 切换窗口状态或全屏状态。

#Z --- 反转z坐标轴的正方向和负方向。

#G --- 开启和关闭坐标网格。

#J --- 在黑色和蓝色或者白色与灰色（Jony状态）之间切换。

#I --- 是否对之间路径进行插值。

#Y --- 是否以艺术手法对指尖路径进行绘制。

#L --- 显示追踪信息，包括上标签的坐标轴，可视化区域，指尖坐标和处理延时。

#T --- 切换手指和工具使用线条绘制或者柱形绘制。在柱形绘制下，柱形的颜色取决于Leap Motion软件将物体识别出一个工具还是手指。在线条模式下，每个物体再被识别后颜色是随机分配的。

#N --- 在手指、手掌面和手球模式之间循环切换手掌追踪数据。[可以参考第一篇概述中的演示]

#O --- 是否打开手势指标的显示。

#P --- 是否暂停可视化工具的显示。

#B --- 显示屏幕位置可视效果。这些视觉效果包含对桌面计算面积的显示帧，手指的投影以及手指和工具指向与屏幕的交点。使用Leap Motion设置对话框[这个近两天翻译]来设置屏幕位置。

#. --- 对可视化窗口中的屏幕内容进行镜面显示（为了展示镜面显示器，屏幕位置可视效果要开启）。

#D --- 切换触摸绘图仿真模式。

## 摄像头控制

可视化工具在三维空间中显示指尖。你可以使用以下的按键控制摄像头：

#= ---放大。

#- --- 缩小。

#V --- 在正投影、前方、上方和侧面视角之间循环切换摄像头。

#A --- 但他们移动到边缘时，自动平移摄像头使得追踪中的手指在视野中。

#C --- 把摄像头回到中心位置，使得最终的手指在视角中心。

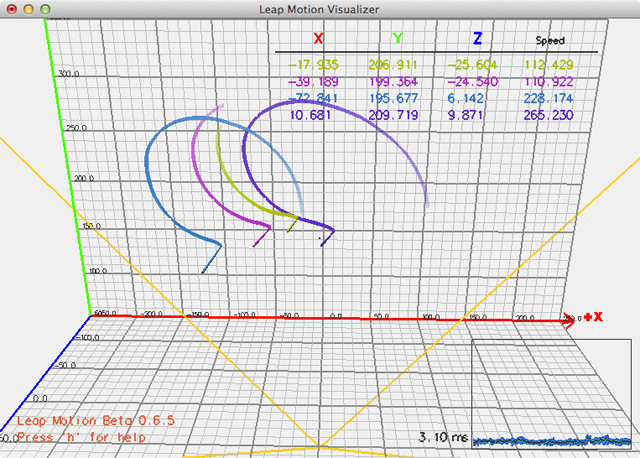
#方向键左和右 --- 开始和停止在正像投影下视角的旋转。

#鼠标点击和拖拉 --- 在正投影下旋转摄像头。

#鼠标右击和拖拉 --- 平移摄像头。

## 追踪信息

按下L键，你可以观测到更多的关于追踪数据的信息。



上图：Leap Motion可视化工具显示额外的追踪数据。

在这个模式下：

#带标签的坐标轴添加到网格显示中，以毫米为衡量单位。

#一个倒金字塔形状表示对Leap Motion控制器视野的估计。

#每个追踪到的指尖的x，y，z坐标都以毫米显示。

#一个以毫秒为单位的图表显示了随着时间推移的处理延时时间。

# Leap Moition体感器设置对话框的使用

这篇文章描述如何使用Leap Motion体感器控制对话框，对现有的选项进行设置。

## 概述

Leap Motion软件与Leap Motion体感器硬件进行通讯，将手的追踪数据发送到应用中。因此，当你想在应用中获取Leap Motion的数据时，Leap Motion服务（Windows）或者守卫进程（Mac和Linux）必须是运行着的。一个独立的Leap Motion任务栏或者菜单栏应用允许用户对TA的Leap Motion软件进行控制。

当Leap Motion应用运行时，它会在Windows的任务栏或者Mac的搜索栏中显示一个图标提示。但Leap设备插入了并且成功运行，图标变成绿色。但软件在初始化会闪黄光，而在错误发生时变成红色。

Leap Motion体感器未连接上电脑（或者Leap Motion软件还没检测到她）。



Leap Motion体感器和软件运行正常。



Leap Motion体感器运行在可靠模式，之后的运行模式章节里有详细解释。



短暂显示，这个图标表示Leap Motion镜头脏了。



发生了一个错误，并且Leap Motion无法恢复。使用显示日志菜单来打开应用日志对错误进行更详细的了解。



当你第一次运行Leap Motion软件时，它会检测你是否通过开发商凭据登陆、你是否有Leap Motion设备的校准文件，这些对于Leap的运行时很必要的。Leap Motion软件在你登陆后会自动下载校准文件。

## 菜单命令

Leap Motion的任务栏或者菜单栏应用有一系列按钮和输入框，它们可以修改追踪数据如何产生、如何向你提供设备状态信息。通过应用图标可找到这些菜单里的按钮。

Leap Motion应用提供了一下菜单命令：

#运行Airspace --- 打开Airspace应用。

#设置--- 打开Leap Motion控制器设置对话框。

#可视化工具--- 运行可视化工具应用。链接：Leap Motion可视化工具使用。

#校准--- 运行设备校准工具，链接：设备校准工具。

#漏洞日志--- 打开漏洞日志对话框。

#启动日志--- 显示Leap Motion软件产生的事件日志。

#关于--- 提供Leap Motion软件的版本号，还包含指向我们网站的链接以及技术支持电子邮箱。

#暂停/回复追踪 --- 暂停/回复Leap Motion追踪数据的产生。

## Leap Motion设置

你可以使用Leap Motion设置对话框来调整Leap Motion系统的表现。

在追踪设置页面，你可以设置一些影响最终的选项，比如操作模式和点击颠倒方向（Reverse Orientation）按钮来进行水平颠倒。

在日志（Logging）页面，你可以设置日志文件选项，将当前日志保存到文件中，并且可以选择自动漏洞汇报功能。

在屏幕（Screen）页面你可以注册Leap Motion设备与已经选择的显示器的关系，这样Leap Motion软件可以准确计算出指尖和工具的射线与屏幕的交点。可以参考屏幕位置章节。

在操作系统交互（OS Interaction）页面，你可以设置针对操作系统的触摸和鼠标输入的仿真。

在网络套接字（WebSocket）页面，你可以打开或者关闭网络套接字服务器。网络套接字服务器向网页或者其它通过网络套接字协议的应用程序，提供JSON格式的追踪数据。

在高级（Advanced）页面，你可以设置一些杂项。可以查看高级设置章节。

## 高级设置

Leap Motion控制器设置对话框的高级设置页面，提供了下面的设置：

#范围灵敏度 --- 收缩Leap Motion视野。

#允许后台应用 --- 运行应用在后台请求Leap Motion数据并且接收到他。

#开启自动省电 --- 允许Leap Motion软件根据视野中运动的目标来调整追踪帧率，达到减少电力使用的效果。

#启用面部移除 --- 激活面部拒绝逻辑，有助于防止出现在Leap Motion追踪数据的人脸被认为是手。

## 操作模式

Leap Motion系统有多个操作模式，可以决定系统如何分析数据。下面的模式是用户可以在Leap Motion应用的设置对话框中可选取的：

#精度（Precision） --- 精度优先于速度。

#平衡（Balanced） --- 权衡速度与精度。

#高速（High Speed） – 速度优先于精度。

#稳健模式（Robust Mode） --- 最终稳定性优先于速度。

#削弱资源（Low Resource） --- 减少Leap Motion使用的USB带宽。

稳健模式提升在强烈光照条件下追踪数据的可靠性。稳健模式允许Leap Motion控制器在一个宽泛的环境下运行；但是，其它的表现特性会有所降低。对于表现效果影响最重要的是，更多的处理延迟，以及用户运动太快会导致追踪数据的丢失。

当光照变弱时，Leap Motion控制器自动进入稳健模式。当光照变强时，Leap Motion又恢复到之前用户选择的操作模式[真智能啊]，但是至少会保持在稳健模式30秒。在这个状态下，Leap Motion的任务栏或者菜单栏图标会在绿色与黄色之间互相变换。

当Leap Motion进入稳健模式时，说明它检测到较恶劣的光照情况，它会把情况保存到一个诊断文件。如果光照问题不明显，你可以将这个诊断文件发送给Leap Motion网站。这个信息将帮助我们改进光照分析算法。

## 系统交互

系统水平上的输入 将用户的手和手指运动翻译成系统触摸和鼠标输入命令。Leap Motion软件根据你的手指运动和手的姿势可以计算出一个合适的的虚拟触摸平面。当你的手指靠近这个平面，Leap Motion软件会在你可能触碰的地方画一个透明的圆。当手指触碰到虚拟平面时，覆盖的圆形区域会变成绿色，Leap Motion通过操作系统模仿一个触摸事件。

你可以根据下面的设置，来调整系统交互行为：

#交互高度 --- Leap Motion软件将一个直线围着的立体（a rectilinear volume）投影到电脑屏幕上。通过移动这个立体中心往上或往下，可以改变交互高度。

#点击速度 – 调整Leap Motion软件以多快的速度响应你运动时的触摸位置。

注意：当一个Leap原生应用获得系统输入焦点时，输入仿真丝失效的。[它的意思应该是获得焦点的才可以接受数据吧]

## 设备校准

如果Leap Motion控制器的传感器的初始对齐效果被破坏了，设备必须重新校准。一些说明Leap Motion需要校准的症状：

#持续的跳动

#追踪数据经常不连续

#在特定视野区域发生追踪偏差

#只能对小范围追踪

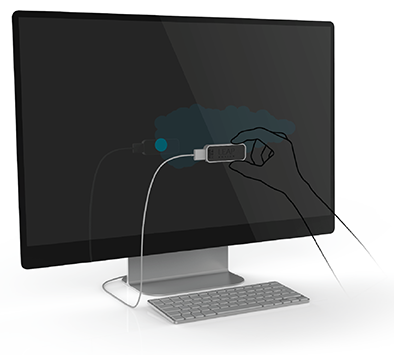
你可以使用Leap Motion可视化工具来验证这些症状。

Leap Motion控制器重新校准的方法：

1.使用再Windows任务栏或者Mac查询栏中的图标，打开Leap Motion应用菜单。

2.选择重新校准（Recalibrate）命令。

3.根据屏幕指示来完成校准步骤。



上图：你需要一个平坦的、反光的平面。使用镜子是最理想的，而像平坦的显示器之类的反光平面也可以接受的。在步骤中，抓住Leap Motion控制器，使得LED光在反光平面上可以反射到设备的顶端。校准窗口显示一个移动的圆盘，依据你对于平面和设备之间角度的调整。旋转控制器，让它“画”窗口。移动Leap Motion控制器往上和往下或者到边上，不会太影响校准，但是校准还是需要旋转的。

## 汇报漏洞

如果你对Leap Motion软件或者硬件有问题，请联系我们，这样才可以帮你解决它。你可以通过Leap Motion开发入口网页或者使用Leap Motion应用中的漏洞汇报。

我们会请你重新在自己的设备上调出这个问题，并记录一些诊断信息，这样你可以将它发送到Leap Motion网站帮助我们了解问题。你可以启动，也可以随时停止对Leap Motion的漏洞诊断信息录制。这些数据是压缩和加密的，对于你本地应用程序的调试没有价值。记录的数据随着时间会越来越大，所以我们不建议长时间记录诊断数据。

汇报一个漏洞：

1.使用在Windows工具栏和Mac的查询栏里的图标打开Leap Motion应用菜单。

2.选择漏洞报告按钮。

3.在漏洞报告窗体里的下拉列表中，选择一个能描述你漏洞的最佳原因。

4.添加任何重要细节，在细节填空中添加诸如如何产生你的漏洞。

5.对于某些类型的漏洞，你会被请求录制诊断信息。

6.当结束时，点击发送按钮。你的漏洞信息和诊断记录将被发送到Leap Motion。

如果你被要求记录诊断信息：

1.请点击记录（Record）按钮。

2.重新产生这个问题（使用Leap Motion可视化工具会有所帮助）。

3.过一小段时间后，点击停止按钮（Stop）。

4.点击发送按钮（Send）按钮来上传诊断信息到Leap Motion以供分析。

### 自动漏洞汇报

Leap Motion应用提供了一个，当Leap Motion系统发生特定类型错误时，可以自动向我们发送诊断信息的选项。这个选项默认是关闭的，但是我们建议你打开它，帮助我们改进Leap Motion软件。

打开自动漏洞汇报：

1.使用Windows任务栏和Mac查询栏的图标打开Leap Motion应用菜单。

2.选择设置命令。

3.在Leap Motion设置对话框，点击日志按钮。

4.检查自动发送漏洞汇报选项。

5.点击确定按钮。

## 应用日志

在Leap Motion应用中打开应用日志，观察Leap Motion应用中相关事件。如果你遇到问题，我们希望你可以保存这个日志，并将它发送到[Leap Motion](https://developer.leapmotion.com/links)以帮助我们捕获这个问题。你可以在设置对话框的日志页面（Logging）进行日志的保存。

# 获取帧数据Getting Frame Data

Leap Motion的API为你的程序提供以一系列被称为帧对象的快照的运动追踪数据。追踪数据的每一帧都包含度量位置和其它在快照中关于检测到的实体的信息。这篇文章详细讨论从Leap Motion控制器中捕获帧物体（Frame objects）。

话题：

## 概述

从链接好的控制器获取一帧包含追踪信息的对象。无论什么时候，只要你的应用程序准备使用控制器（Controller）类中的frame()方法来操作，就可以获取一个帧对象：

if( controller.isConnected()) //controller is a Controller object

{

Frame frame = controller.frame(); //The latest frame

Frame previous = controller.frame(1); //The previous frame

}

frame()方法输入一个历史（history）参数标示可以重新得到多少以前的帧。[应该就是缓存空间大小吧]一般的，过去的60帧内容都保存在这个历史缓存中。

## 通过轮询获取帧

通过对控制器物体的轮询获得帧对象的方式，是当你应用运行在自然帧率下最简单以及往往最好的策略。当你设备处理一帧数据时，你只需要调用一下控制器的frame()函数。

但你使用轮询方法，有一定概率你会在同一行内（如果应用个帧率快于Leap的帧率）获取同一帧的内容，又或者跳过一帧（当Leap帧率快于你应用的帧率）。在大部分情况下，丢失或者重复的帧对象都是不重要的。比如，如果你在屏幕前用手移动物体，物体的运动依旧是平滑的（假设你应用的总体帧率是足够高的）。[不知道它表达的什么意思，我觉得只要它数据是平滑变化的就好]

为了检测你是否正在处理了一个已经处理过的帧对象了，将已保存最后处理过帧的ID数值与当前帧进行比较：

int64\_t lastFrameID = 0;

void processFrame( Leap::Frame frame )

{

if( frame.id() == lastFrameID ) return;

//...

lastFrameID = frame.id();

}

如果你的应用丢失了一些帧，你可以使用frame()函数里的历史参数来访问丢失的帧（只要这个帧对象还在历史缓存中）：

int64\_t lastProcessedFrameID = 0;

void nextFrame( Leap::Controller controller )

{

int64\_t currentID = controller.frame().id();

for( int history = 0; history < currentID - lastProcessedFrameID; history++)

{

processFrame( controller.frame(history) );

}

lastProcessedFrameID = currentID;

}

void processNextFrame( Leap::Frame frame )

{

if( frame.isValid() )

{

//...

}

}

## 通过回调函数获取帧

同样的，你还可以使用一个监听（Listener）对象来获取Leap Motion控制器的帧对象。只要一个新的帧到来，这个控制器对象就会调用监听者的onFrame()函数。在onFrame处理中，你可以调用控制器（Controller）的frame()函数来获取他自己的帧（Frame）对象。

使用监听者回调函数比较复杂，因为回调是多线程的。每个回调都来自一个独立的线程。你必须保证来自多线程中的数据都是以线程安全方式获取的。

接下来的例子定义了一个最小的监听子类，它可以获取新的数据帧：

class FrameListener : Leap::Listener

{

void onFrame(Controller &controller)

{

Frame frame = controller.frame(); //The latest frame

Frame previous = controller.frame(1); //The previous frame

//...

}

};

正如你所见，通过一个监听对象获取追踪数据和轮询模式几乎一样。

注意，跳过一帧是很必要的，即使当我们使用监听回调函数。如果你的onFrame回调函数用了过多的时间才执行完毕，那么下一帧被被添加到历史记录中，但是onFrame回调函数跳过去了。不太相同的，如果一个Leap软件不能及时处理一帧数据，那个帧会被抛弃，并且不会被加入到历史中。当计算机由于需要运行太多任务导致停滞，就会导致这个问题。

## 从帧中获取数据

帧类定义了一些如何访问帧中数据的函数。例如，下面的代码描述了如何获取Leap Motion系统追踪到的基本对象。

Leap::Controller controller = Leap::Controller();

// wait until Controller.isConnected() evaluates to true

//...

Leap::Frame frame = controller.frame();

Leap::HandList hands = frame.hands();

Leap::PointableList pointables = frame.pointables();

Leap::FingerList fingers = frame.fingers();

Leap::ToolList tools = frame.tools();

这个从帧对象中返回的对象都是只读的。你可以安全的保存它们并在未来使用它们。它们是线程安全的。内部的，这些对象使用[C++ Boost库共享指针类](http://www.boost.org/doc/libs/1_51_0/libs/smart_ptr/shared_ptr.htm" \l "ThreadSafety)。

## 使用IDs来追踪帧间实体

如果你获取到了一个来自不同帧下实体的ID，你可以获取对象在当前帧的描述信息。将ID传递给合适类型的帧函数：

Hand hand = frame.hand(handID);

Pointable pointable = frame.pointable(pointableID);

Finger finger = frame.finger(fingerID);

Tool tool = frame.tool(toolID);

如果具有相同ID的对象无法被找到----也许一只手或者手指移出了Leap的视野。----那么，取而代之，会返回一个特别的无效对象。无效对象也是类的实例，但是他们所有的成员都返回0数值、0数组或者其他的无效对象。这个技术使得将方法连接在一起调用更方便。例如，下列的代码计算多帧下指尖的平均位置。

//Average a finger position for the last 10 frames

int count = 0;

Leap::Vector average = Leap::Vector();

Leap::Finger fingerToAverage = frame.fingers()[0];

for( int i = 0; i < 10; i++ )

{

Leap::Finger fingerFromFrame = controller.frame(i).finger(fingerToAverage.id());

if( fingerFromFrame.isValid() )

{

average += fingerFromFrame.tipPosition();

count++;

}

}

average /= count;

如果没有无效的对象，那么这段代码需要每次在返回手指对象时检测这个帧对象。

# 追踪手、手指和工具

手、手指和工具是Leap Motion系统的基本追踪实体。这篇文章将详细讨论如何获取和使用这些实体对象。

## 概述

Leap的API定义了一个可以表示每个基本追踪到的物体的类。

帧对象提供了一个可以访问手、手指和工具的列表。指尖和工具都是尖端物体，并且他们可以被同时放入尖端列表PointableList中，或者分别使用手指列表FingerList和工具列表类ToolList。手对象提供访问他们的手指和工具的途径（都一起在尖端列表中或者分开）。

手、手指和工具的物体特点在Leap的坐标系统下（以毫米为单位衡量）有报告。Leap的SDK提供了向量类（Vector）来描述点和朝向。向量类提供了一些有用的向量运算的数学函数[这个挺不错，避免自己写了]。

## 手和尖端列表

列表类都都拥有近似的结构。他们被设计成向量形式矩阵并且支持迭代器。你不能移除或者改变从Leap API接收的列表中的成员变量，但你可以组合列表中的同一个类型的对象。

对某一个列表使用迭代器：

for(HandList::const\_iterator hl = handList.begin(); hl != handList.end();)

std::cout << \*hl << std::endl;

手、尖端物体、手指和工具列表基于在Leap坐标系统下的相对位置，定义了额外的函数来获取列表中的成员变量。这些函数包含leftmost()，rightmost()，和frontmost()。接下来的小片段代码说明了使用这些函数的方法：

Leap::Finger farLeft = frame.fingers().leftmost();

Leap::Finger mostForwardOnHand = frame.hands()[0].fingers().frontmost();

Leap::Tool rightTool = frame.tools().rightmost();

一个更复杂的例子是计算一个包含检测到的所有尖端物体箱子的矩形体。因为在API中没有提供，这个例子定义了它自己的函数来获取顶部、底部和后方的尖端。

float left = frame.pointables().leftmost().tipPosition().x;

float right = frame.pointables().rightmost().tipPosition().x;

float front = frame.pointables().frontmost().tipPosition().z;

float back = backmost(frame.pointables()).tipPosition().z;

float top = topmost(frame.pointables()).tipPosition().y;

float bottom = bottommost(frame.pointables()).tipPosition().y;

Leap::Pointable backmost(PointableList pointables)

{

if(pointables.count() == 0) return Leap::Pointable::invalid();

Leap::Pointable backmost = pointables[0];

for( int p = 1; p < pointables.count(); p++ )

{

if( pointables[p].tipPosition().z > backmost.tipPosition().z)

backmost = pointables[p];

}

return backmost;

}

Leap::Pointable topmost(PointableList pointables)

{

if(pointables.count() == 0) return Leap::Pointable::invalid();

Leap::Pointable topmost = pointables[0];

for( int p = 1; p < pointables.count(); p++ )

{

if( pointables[p].tipPosition().y > topmost.tipPosition().y)

topmost = pointables[p];

}

return topmost;

}

Leap::Pointable bottommost(PointableList pointables)

{

if(pointables.count() == 0) return Leap::Pointable::invalid();

Leap::Pointable bottommost = pointables[0];

for( int p = 1; p < pointables.count(); p++ )

{

if( pointables[p].tipPosition().y < bottommost.tipPosition().y )

bottommost = pointables[p];

}

return bottommost;

}

[也就是简单计算下所有物体三维坐标的最小外接矩形，很容易理解~代码如出一辙，看懂一个后面都懂。看了下API，以及我自己猜测，left,front,right这几个方位，Leap Motion是比较关注的，而对于高Leap可能无法准确探测，但对于其它几个方向为什么不直接提供函数呢？]

## 手

手类描述了一个被Leap追踪到的物理形式的手。一个手对象提供了访问它自己尖端（手指）的列表，以及一个描述的手的坐标、朝向和运动的属性。从一帧中获取手对象：

Leap::Frame frame = controller.frame(); // controller is a Leap::Controller object

Leap::HandList hands = frame.hands();

Leap::Hand firstHand = hands[0];

或者，如果你从前一帧中已知ID：

Leap::Hand knownHand = frame.hand(handID);

你还可以通过它们在帧中相对位置获取手对象：

Leap::Frame frame = controller.frame(); // controller is a Leap::Controller object

Leap::HandList hands = frame.hands();

Leap::Hand leftmost = hands.leftmost();

Leap::Hand rightmost = hands.rightmost();

Leap::Hand frontmost = hands.frontmost();

注意，rightmost()和rightmost()函数仅仅区分哪只手是最右侧或者最左侧。这个方法不区分这只手是左手还是右手。

### 获取手特征

一只手被通过它的坐标、朝向和运动得以描述。

手的坐标是通过手掌坐标属性得到的，提供了一个以Leap Motion为原点，毫米为单位的手掌中心三维坐标的向量。手的朝向由2个向量给出：方向---从手掌中心指向手指方向，手掌垂直面---指向手掌外侧，垂直于手的平面。

手的运动通过运动速度属性给出，它是一个描述手在毫米每秒瞬时运动的向量。你还可以获得运动因子，它可以描述一只手在两帧时的位移、旋转和缩放值的变化。

下面的代码块描述了如何在一帧中获取一个手对象和它的基本属性：

Leap::Hand hand = frame.hands().rightmost();

Leap::Vector position = hand.palmPosition();

Leap::Vector velocity = hand.palmVelocity();

Leap::Vector direction = hand.direction();

### 获取指尖和工具

你可以通过列表或仅仅通过之前帧的ID两种方式，获取与手关联的手指和工具。

通过列表方法：

// hand is a Leap::Hand object

Leap::PointableList pointables = hand.pointables(); // Both fingers and tools

Leap::FingerList fingers = hand.fingers();

Leap::ToolList tools = hand.tools();

通过之前帧ID方法：

Leap::Pointable knownPointable = hand.pointable(pointableID);

为了获取手指或者工具在Leap视野的相对位置，可以使用匹配列表类中right-，left-和frontmost函数。

// hand is a Leap::Hand object

Leap::Pointable leftPointable = hand.pointables().leftmost();

Leap::Finger rightFinger = hand.fingers().rightmost();

Leap::Tool frontTool = hand.tools().frontmost();

注意，这些函数都是相对于Leap Motion原点位置的，而不是手它自己的。为了获取相对于手的手指，你可以使用Leap矩阵类来讲手指坐标转换到参考帧的手。[不太懂]

### 计算手的朝向

使用手的方向和垂直向量，你可以计算出手的朝向角度（相对于东方）。向量类定义了俯仰Pitch旋转（围绕x轴旋转）、左右Yaw旋转（围绕y轴的旋转）和平面Roll旋转（围绕z轴旋转）：

float pitch = hand.direction().pitch();

float yaw = hand.direction().yaw();

float roll = hand.palmNormal().roll();

注意，roll函数仅仅提供使用垂直向量时的预判的角度。[看来roll判断比较困难吧]

### 将手指坐标变换到手的参照帧（以手为参照对象）

有时，通过手的参照帧获取在手上的手指坐标是常用的方法。这种方法可以让你对手指在空间上进行排序，并且可以简单的分析手指的坐标[很有用啊]。你可以使用Leap矩阵类构建一个转换矩阵来转换手指坐标和方向坐标。参照中的手帧可以通过手的朝向和手掌的垂直向量（由与两个坐标轴正交的第三个坐标轴）被有效的定义。这种方法使得x轴沿着手侧面，而z轴指向前方，y轴与手掌平面平行。

Leap::Frame frame = leap.frame();

for( int h = 0; h < frame.hands().count(); h++ )

{

Leap::Hand leapHand = frame.hands()[h];

Leap::Vector handXBasis = leapHand.palmNormal().cross(leapHand.direction()).normalized();

Leap::Vector handYBasis = -leapHand.palmNormal();

Leap::Vector handZBasis = -leapHand.direction();

Leap::Vector handOrigin = leapHand.palmPosition();

Leap::Matrix handTransform = Leap::Matrix(handXBasis, handYBasis, handZBasis, handOrigin);

handTransform = handTransform.rigidInverse();

for( int f = 0; f < leapHand.fingers().count(); f++ )

{

Leap::Finger leapFinger = leapHand.fingers()[f];

Leap::Vector transformedPosition = handTransform.transformPoint(leapFinger.tipPosition());

Leap::Vector transformedDirection = handTransform.transformDirection(leapFinger.direction());

// Do something with the transformed fingers

}

}

## 尖端

尖端对象可以被表示为手指和工具 ---- 也就是可以指出方向的东西。

你可以通过手对象中特定的手，获取与手关联的手指和工具。尖端未必要和手对象关联 --- 物理上的手也许不在视野中或者被另一只手挡住了。因此帧中尖端列表可以包含哪些和手不关联的手指和工具[即增加了适用性，同时也引入了复杂性呀]。

尖端对象有着许多描述手指和工具的属性：

#尖端坐标 --- 在Leap Motion坐标系下以毫米为单位的瞬时的坐标。

#尖端速度 --- 以毫米每秒为单位的瞬时速度。

#指向 --- 当前的指向朝向向量。

#长度 --- 手指和工具显现的长度。

#宽度 --- 平均宽度。

#触摸距离 --- 在虚拟触摸平面中归一后的距离。

#触摸区域 --- 当前尖端和虚拟触摸平面的关系。

下面的例子说明如何从帧中获取简短对象和基本访问属性的方法：

Leap::Pointable pointable = frame.pointables().frontmost();

Leap::Vector direction = pointable.direction();

float length = pointable.length();

float width = pointable.width();

Leap::Vector stabilizedPosition = pointable.stabilizedTipPosition();

Leap::Vector position = pointable.tipPosition();

Leap::Vector speed = pointable.tipVelocity();

float touchDistance = pointable.touchDistance();

Leap::Pointable::Zone zone = pointable.touchZone();

### 将尖端对象转化成手指或者工具

为了将尖端对象转化到合适的手指和工具子类，需要使用合适的手指和工具构造函数（某些时候你应该使用Leap类的构造函数）[不太明白]。

if (pointable.isTool()) {

Leap::Tool tool = Leap::Tool(pointable);

} else {

Leap::Finger finger = Leap::Finger(pointable);

}

### 计算手指基本位置

如果你需要计算手指的基本位置，你可以如下使用手指尖位置和朝向：

Leap::Vector basePosition = -pointable.direction() \* pointable.length();

basePosition += pointable.tipPosition();

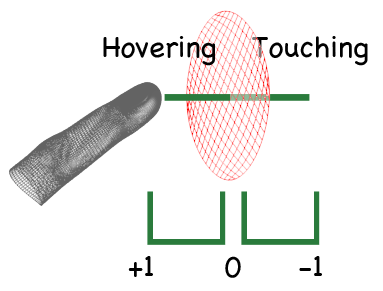
# 触摸仿真

Leap Motion的API提供了在你的应用中可以进行仿真的信息。触摸信息是通过尖端类提供的。

## 概述

Leap 定义了一个自适应的触摸平面，你可以将它与你应用中的二维元素美妙的结合在一起。这个平面粗糙的旋转到与x-y平面平行，但是根据用户的手指和手的位置动态调整。但用户的手或者工具从前方到达平面，Leap会报告尖端物体是接近还是触碰到这个虚拟的平面。API通过2个值汇报与平面相关的信息：触摸平面地带和到触摸平面的距离。

上图：虚拟触摸平面



触摸地带是识别Leap Motion软件是否把尖端当作悬浮在触摸平面，是穿透触摸平面还是相对于屏幕较远（或者指向错误的方向）。地带包含“悬浮”、“触摸”和“无”。触摸地带的转换根据触摸距离变换有一定滞后。这个滞后用于避免突然和反复的变换。如果你在应用中使用这个触摸交互，你可能不需要经常考虑地带。

只有当尖端在悬浮和触摸地带，触摸距离才是有效的。这个距离是归一到[-1,+1]的数值。当一个尖端物体第一次进入悬浮地带，此时触摸距离为+1，而当距离不断减少到0，意味着尖端物体接近触摸平面。但尖端物体穿入平面，距离变为0。因为尖端物体继续往触摸地带推进，距离不断靠近但不会超过-1（永远趋向于-1）。

你可以使用地带数值，根据悬浮还是触摸来决定何时更新界面元素。你还可以使用距离，根据是否靠近平面进一步改变界面元素[一个粗，一个细]。例如，当手指在控制体上并且在悬浮地带，你可以让控制体高亮显示，并且依据用户到底多接近控制体的距离，改变光标形态。

作为触摸仿真API的一部分，Leap Motion为尖端物体，提供了一个额外的相对于标准坐标的稳定坐标。Leap Motion软件使用自适应滤波器来稳定这个位置，可以平滑和降低运动，最终使得它在屏幕上小区域内（例如按钮和链接）更容易地与用户交互。但运动缓慢是平滑效果也越强，这样用户可以调整距离并且更容易的触碰到特定的点[这个效果很赞，和人们在现实世界操作一样]。

## 获取触摸地带

触摸地带通过尖端类的touchZone属性描述。这些地带都是使用枚举类型标识的，一共有下面三个状态：

#NONE --- 尖端距离触摸屏幕太远，大于考虑它触摸的距离。又或者它指向反方向的用户（也就是物体默认在用户正面前，用户手指着自己是一种不合适的操作）

#HOVERING --- 尖端物体的顶端已经到达悬浮地带，但不被认为触摸。

#TOUCHING --- 尖端到达了虚拟平面内。

下面的代码片段说明如何取回最前面手指的地带标识：

Leap::Frame frame = leap.frame();

Leap::Pointable pointable = frame.pointables().frontmost();

Leap::Pointable::Zone zone = pointable.touchZone();

## 获取触摸距离

触摸距离是通过尖端类的touchDistance属性描述的。这个距离范围是+1到-1，对应了手指移向和穿过虚拟触摸平面。这个距离没有物理意义，但却是Leap Motion软件认为的到底有多靠近触碰。

下面的代码段说明如何取出最前端手指的触摸距离：

Leap::Frame frame = leap.frame();

Leap::Pointable pointable = frame.pointables().frontmost();

float distance = pointable.touchDistance();

## 获取尖端的稳定坐标

稳定坐标是尖端类的stabilizedTipPosition属性描述的。这个位置根据标准Leap Motion坐标系统参照得出的，但是具有一个上下文相关大量数据的滤波器，所以很稳定。

下面的代码片段描述了如何得到最前端手指的稳定位置：

Leap::Frame frame = leap.frame();

Leap::Pointable pointable = frame.pointables().frontmost();

Leap::Vector stabilizedPosition = pointable.stabilizedTipPosition();

## 从Leap Motion坐标系转换到应用坐标系

但使用触摸仿真时，你必须把Leap Motion的坐标空间转化到应用的屏幕空间。为了使得这个操作更简单，Leap Motion的API提供了一个交互箱子类[交互盒子类]（IneractionBox）。这个交互箱子类描述Leap Motion视野中的线性物体运动。这个类提供了一个把物体范围中的坐标归一化到[0,1]范围内。你可以归一化一个距离，并且根据应用尺寸，把结果坐标进行缩放，来获取一个在应用中的坐标。

例如，如果你在客户区域中有个具有windowWidth和windowHeight两个度量的窗口,你可以使用以下代码，取得在窗口中触摸点的二维像素坐标：

Leap::Frame frame = leap.frame();

Leap::Finger finger = frame.fingers().frontmost();

Leap::Vector stabilizedPosition = finger.stabilizedTipPosition();

Leap::InteractionBox iBox = leap.frame().interactionBox();

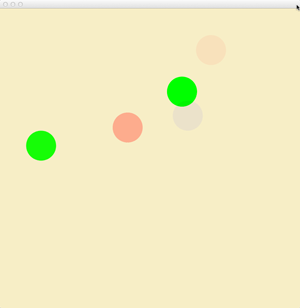
Leap::Vector normalizedPosition = iBox.normalizePoint(stabilizedPosition);

float x = normalizedPosition.x \* windowWidth;

float y = windowHeight - normalizedPosition.y \* windowHeight;

## 触点例子

下面的例子使用触摸仿真的API，来显示所有在应用窗口下，检测到的尖端物体。这个例子使用了触摸地带来设置点的颜色，使用触摸距离来设置alpha数值。使用了交互盒子类爸稳定的顶点位置被映射到应用窗口中。



上图：触摸点例子

#include "cinder/app/AppNative.h"

#include "cinder/gl/gl.h"

#include "Leap.h"

#include "LeapMath.h"

using namespace ci;

using namespace ci::app;

using namespace std;

class TouchPointsApp : public AppNative {

public:

void setup();

void draw();

private:

int windowWidth = 800;

int windowHeight = 800;

Leap::Controller leap;

};

void TouchPointsApp::setup()

{

this->setWindowSize(windowWidth, windowHeight);

this->setFrameRate(120);

gl::enableAlphaBlending();

}

void TouchPointsApp::draw()

{

gl::clear( Color( .97, .93, .79 ) );

Leap::PointableList pointables = leap.frame().pointables();

Leap::InteractionBox iBox = leap.frame().interactionBox();

for( int p = 0; p < pointables.count(); p++ )

{

Leap::Pointable pointable = pointables[p];

Leap::Vector normalizedPosition = iBox.normalizePoint(pointable.stabilizedTipPosition());

float x = normalizedPosition.x \* windowWidth;

float y = windowHeight - normalizedPosition.y \* windowHeight;

if(pointable.touchDistance() > 0 && pointable.touchZone() != Leap::Pointable::Zone::ZONE\_NONE)

{

gl::color(0, 1, 0, 1 - pointable.touchDistance());

}

else if(pointable.touchDistance() <= 0)

{

gl::color(1, 0, 0, -pointable.touchDistance());

}

else

{

gl::color(0, 0, 1, .05);

}

gl::drawSolidCircle(Vec2f(x,y), 40);

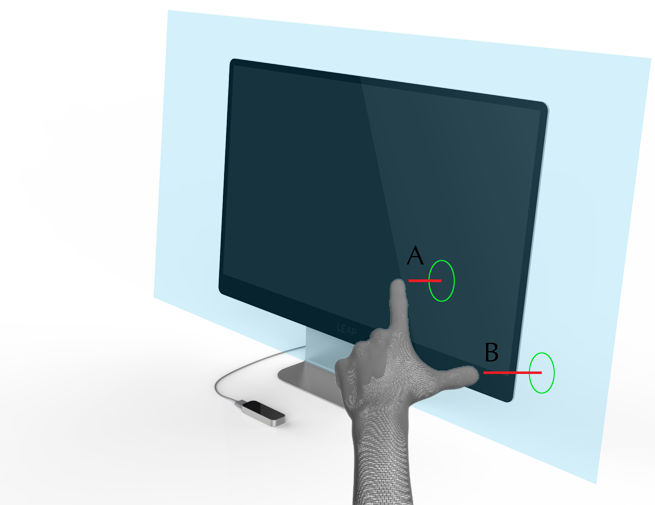
}

}

CINDER\_APP\_NATIVE( TouchPointsApp, RendererGl )

这个例子使用了[Cinder](http://libcinder.org/)库来创建用于绘画的应用窗口。[例子这么简单，怎么玩运行啊]

# 屏幕定位 Screen Location



上图：2个点与相对于的投影点

@点A直接投影到屏幕上。

@点B投影到了屏幕外侧。

这里原来的内容大多失效，只举例如何将手指投影到我们的二维应用的代码，下面的代码需要好好的研读。**特别需要注意的是stabilizedTipPosition()，通过这个函数可以获取平滑后的手指或手掌等的坐标，避免在应用中的抖动现象！**

将手指坐标映射到二维应用中

int appWidth = 800;

int appHeight = 600;\_x000D\_InteractionBox iBox = controller.frame().interactionBox();

Pointable pointable = controller.frame().pointables().frontmost();

Vector leapPoint = pointable.stabilizedTipPosition();\_x000D\_Vector normalizedPoint = iBox.normalizePoint(leapPoint, false);\_x000D\_int appX = normalizedPoint.x \* appWidth;

int appY = (1 - normalizedPoint.y) \* appHeight;*//The z-coordinate is not used*

将手指坐标映射到二维应用中（更加灵敏）

int appWidth = 800;

int appHeight = 600;\_x000D\_InteractionBox iBox = controller.frame().interactionBox();

Pointable pointable = controller.frame().pointables().frontmost();

Vector leapPoint = pointable.stabilizedTipPosition();

Vector normalizedPoint = iBox.normalizePoint(leapPoint, false);\_x000D\_normalizedPoint \*= 1.5; *//scale*

normalizedPoint -= Leap::Vector(.25, .25, .25); *// re-center*int appX = normalizedPoint.x \* appWidth;

int appY = (1 - normalizedPoint.y) \* appHeight;*//The z-coordinate is not used*

# Leap Motion 2.x API的变化

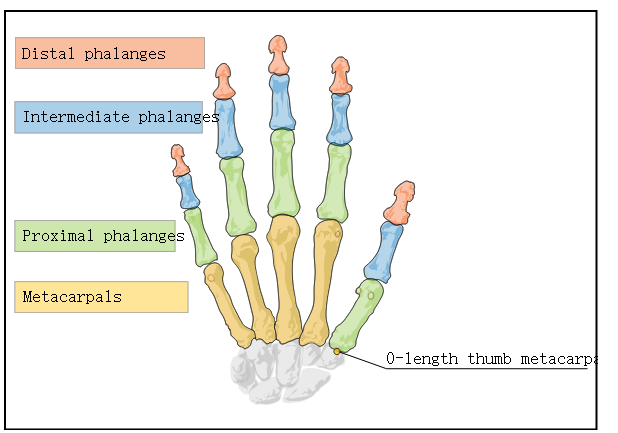
## 增加直接获取Leap近红外图像的API函数，Frame.Images()。

使用方法如下（更具体的描述参见LeapSDK/docs/cpp/devguide/Leap\_Images.html，描述了如何矫正、转换成rgb图像等等）：

//Uses Cinder OpenGL wrapper\_x000D\_Frame frame = controller.*[frame](E:/Knowledge/LeapMotion/%E5%AE%89%E8%A3%85%20SDK/LeapDeveloperKit_2.2.0+23475_win/LeapSDK/docs/cpp/api/Leap.Controller.html" \l "cppclass_leap_1_1_controller_1a1a25f3894cf82667f423dad20d7d6b62)*();\_x000D\_\_x000D\_ImageList images = frame.*[images](E:/Knowledge/LeapMotion/%E5%AE%89%E8%A3%85%20SDK/LeapDeveloperKit_2.2.0+23475_win/LeapSDK/docs/cpp/api/Leap.Frame.html" \l "cppclass_leap_1_1_frame_1a85588dbb02a8ff793cf2ced9067f8263)*();\_x000D\_for(int i = 0; i < 2; i++){\_x000D\_ *[Image](file:///E:/Knowledge/LeapMotion/%E5%AE%89%E8%A3%85%20SDK/LeapDeveloperKit_2.2.0+23475_win/LeapSDK/docs/cpp/api/Leap.Image.html" \l "cppclass_leap_1_1_image#cppclass_leap_1_1_image_1a66d0d6da3b89580352c28d37b57df5bb)* image = images[i];\_x000D\_\_x000D\_ const unsigned char\* image\_buffer = image.data();\_x000D\_\_x000D\_ //Draw the raw image data as a greyscale bitmap\_x000D\_ Surface surface(image.width(), image.height(), image.width() \* 4, SurfaceChannelOrder::RGBA);\_x000D\_ int cursor = 0;\_x000D\_ Surface::Iter iter = surface.getIter();\_x000D\_ while( iter.line() ) {\_x000D\_ while( iter.pixel() ) {\_x000D\_ iter.r() = image\_buffer[cursor];\_x000D\_ iter.g() = iter.b() = iter.r();\_x000D\_ iter.a() = 255;\_x000D\_ cursor++;\_x000D\_ }\_x000D\_ }

## 全新的骨骼追踪模型

在2.x中Leap提供了更多的关于手和手指的信息，并且可以在手在视野中不清晰时更好的预测手指和手的位置。一个手上永远有5个手指，而且即使手指交叉也可以被追踪（当然你得提供视野，完全遮挡住它肯定无法预测）。



### 关于Hand的改变

增加了识别左右手的参数 Hand.isLeft() 、 Hand.isRight() 【当然了，leap得看到几乎全部的手指才能做出正确判断，如果你握拳，Leap往往要识别错误的】

增加了每个关节的位置和朝向

增加了一个抓取强度Hand.gripStrength()和捏的强度Hand.pinchStrength()来标示用户正在捏或者抓。【数据都是0~1范围之内的浮点数，在应用中自己设定阈值（比如0.5）来触发一些功能】

实时输出每只手各5个手指的信息

还有Hand.confidence()，Hand.palmWidth()，Hand.basis()，Hand.arm()是新增的函数。

basic()函数在很多类中也都有，貌似是一个相对于手的统一的基本向量，可以参见Matrx basis()的定义，xBasis是相对于小拇指，yBasis是相对于手的上方，zBasis是相对于手腕的正方向，我还不太明白他的作用。

### 关于Finger的改变

Finger.type()当前是什么手指，也就是食指、拇指、中指之类的枚举

Finger.bone()当前手指的骨头，都有4个骨头（大拇指0号骨头没有），使用如下：

*[Leap::FingerList](E:/Knowledge/LeapMotion/%E5%AE%89%E8%A3%85%20SDK/LeapDeveloperKit_2.2.0+23475_win/LeapSDK/docs/cpp/api/Leap.FingerList.html" \l "cppclass_leap_1_1_finger_list)* fingers = frame.*[hands](E:/Knowledge/LeapMotion/%E5%AE%89%E8%A3%85%20SDK/LeapDeveloperKit_2.2.0+23475_win/LeapSDK/docs/cpp/api/Leap.Frame.html" \l "cppclass_leap_1_1_frame_1a788e21d8ff5d1f54209e75f14a27110b)*()[0].fingers();\_x000D\_for(Leap::FingerList::const\_iterator fl = fingers.*[begin](E:/Knowledge/LeapMotion/%E5%AE%89%E8%A3%85%20SDK/LeapDeveloperKit_2.2.0+23475_win/LeapSDK/docs/cpp/api/Leap.FingerList.html" \l "cppclass_leap_1_1_finger_list_1aa4b02d4f1a8500fb07a551069060709f)*(); fl != fingers.*[end](E:/Knowledge/LeapMotion/%E5%AE%89%E8%A3%85%20SDK/LeapDeveloperKit_2.2.0+23475_win/LeapSDK/docs/cpp/api/Leap.FingerList.html" \l "cppclass_leap_1_1_finger_list_1a350132543d80a1c1e5be844e6d2878ea)*(); fl++){\_x000D\_ *[Leap::Bone](E:/Knowledge/LeapMotion/%E5%AE%89%E8%A3%85%20SDK/LeapDeveloperKit_2.2.0+23475_win/LeapSDK/docs/cpp/api/Leap.Bone.html" \l "cppclass_leap_1_1_bone)* *[bone](file:///E:/Knowledge/LeapMotion/%E5%AE%89%E8%A3%85%20SDK/LeapDeveloperKit_2.2.0+23475_win/LeapSDK/docs/cpp/api/Leap.Finger.html" \l "cppclass_leap_1_1_finger_1a1c96d488ec00ab0c488c8e9153ead9c7#cppclass_leap_1_1_finger_1a1c96d488ec00ab0c488c8e9153ead9c7)*;\_x000D\_ *[Leap::Bone::Type](E:/Knowledge/LeapMotion/%E5%AE%89%E8%A3%85%20SDK/LeapDeveloperKit_2.2.0+23475_win/LeapSDK/docs/cpp/api/Leap.Bone.html" \l "cppclass_leap_1_1_bone_1a1d1cfd8ffb84e947f82999c682b666a7)* boneType;\_x000D\_ for(int b = 0; b < 4; b++)\_x000D\_ {\_x000D\_ boneType = static\_cast<*[Leap::Bone::Type](E:/Knowledge/LeapMotion/%E5%AE%89%E8%A3%85%20SDK/LeapDeveloperKit_2.2.0+23475_win/LeapSDK/docs/cpp/api/Leap.Bone.html" \l "cppclass_leap_1_1_bone_1a1d1cfd8ffb84e947f82999c682b666a7)*>(b);\_x000D\_ bone = (\*fl).bone(boneType);\_x000D\_ std::cout << "Finger index: " << (\*fl).*[type](E:/Knowledge/LeapMotion/%E5%AE%89%E8%A3%85%20SDK/LeapDeveloperKit_2.2.0+23475_win/LeapSDK/docs/cpp/api/Leap.Bone.html" \l "cppclass_leap_1_1_bone_1afbd0fa31db28593e9669c3c56711c0a7)*() << " " << bone << std::endl;\_x000D\_ }\_x000D\_}

实时输出某个手指手否是伸直的Finger.isExtended()

【小明：在1.x时代，大家用的比较多的获取当前有多少只手指是伸展开的功能fingers.count()，已经永远是5了。现在大家需要使用isExtended()函数来判断每个手指是否是伸直的还是弯曲的。】

### 新增Bone类

增加了一堆方法，如basis()、direction()、invalid()、isValid()、prevJoint()、nextJoint()、type()、width()

### 新增Arm类

增加了一堆方法，如basic()、direction()、elbowPosition()、invalid()、isValid()、width()、wristPosition()