

HTC Vive追踪器

开发指南

Ver. 1.3

历史版本

版本号	版本日期	版本说明
1.0	2016.09.26	初版
1.1	2016.12.05	1. 用例图片修正 2. 重排Pogo pin 3. 删除USB数据线连接方式. 4. 修改数据格式
1.2	2017.01.09	1. 修改SteamVR监控软件图像
1.3	2017.01.19	1. 修改了Pogo pin的设计 2. 修改了固件升级

©2016 HTC Corporation. All Rights Reserved. HTC, the HTC logo, Vive, the Vive logo, and all other HTC product and services names are the trademarks or registered trademarks of HTC Corporation and its affiliates in the U.S. and other countries.

目录

序言	1
用例	1
硬件需求	4
接口	5
射频 (RF)	6
电源	8
光学	8
底座	9
底座设计	9
机械方面的考虑	12
外壳尺寸	13
主要特性	14
安置机制	15
用标准的三角架云台安置 (不接电)	15
使用侧面的转轮拧紧 (有需要的话可以接电)	16
外设设计	17
弹簧针(Pogo pin)面板的设计	18
坐标系统	24
软件部分	26
系统需求	26
数据格式	28
外设整合	30
Unity整合	34
固件升级	42

序言

这个文档为VR外设的开发者和内容开发者提供了指导方案。主要介绍了如何使用Vive追踪器来追踪位置和传输特定数据(在用以及不用HTC Vive VR系统的情况下)。

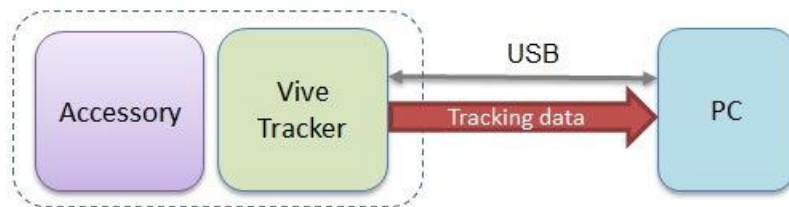
Vive追踪器可以通过和HTC的无线dongle配对 ,也可以通过USB接口把定位数据传给PC。而一个附加到Vive追踪器上的外设可以：

- 通过追踪器底部的的弹簧针 (Pogo pin) 来模拟Vive手柄的按键
- 向PC发送特定数据。可以用追踪器的USB接口, 也可以用外设原有的方式

用例

Vive追踪器用例有5种：

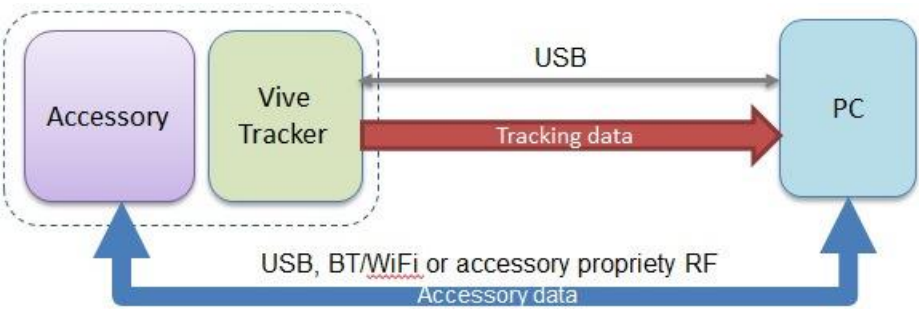
用例1：通过USB数据线追踪被动物体。这种情况下不需要dongle，Vive和PC之间直接用USB数据线来传递定位数据。



图：Vive追踪器用例1

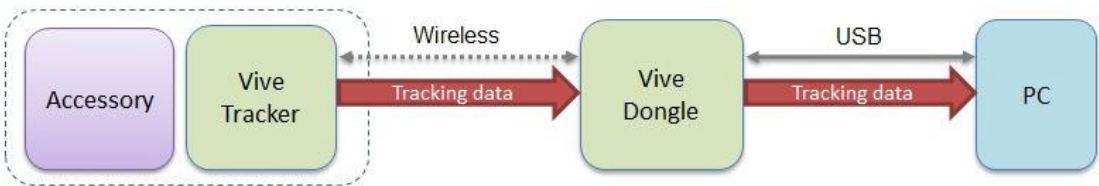
用例2：通过USB接口来定位被动物体，同时，外设和电脑之间用USB、蓝牙、Wi-Fi或者专属无线技术 (Proprietary RF) 传输数据。这种情况和用例1类似，不过外设直

接和PC连接来传递它所产生的数据。



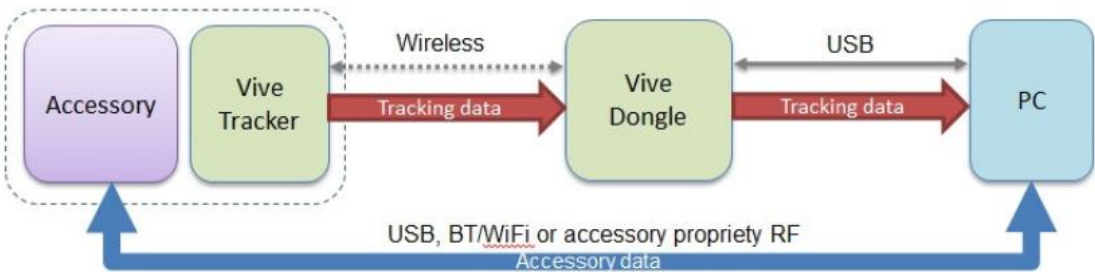
图：Vive追踪器用例2

用例3：通过无线接口来定位移动。在这种情况下，需要用到dongle来从Vive追踪器向PC传递定位数据。



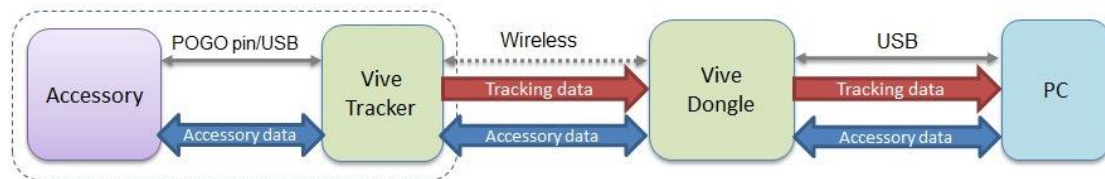
图：追踪器用例3

用例4：通过无线接口来定位移动，同时，外设和电脑之间用USB、蓝牙、Wi-Fi或者专属无线技术 (Proprietary RF) 传输数据。这种情况和用例3类似，不过外设直接和PC连接来传递它所产生的数据。



图：追踪器用例4

用例5：通过无线接口来定位移动，同时，外设通过追踪器来模拟Vive手柄按键或者传输数据。这种情况和用例3类似，不同的是此时外设通过弹簧针(Pogo pin)或者USB来和追踪器连接，以此向间接向PC传输数据。

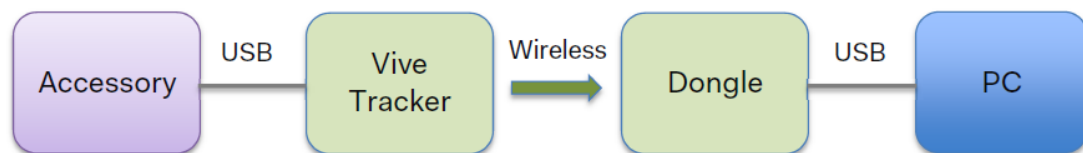


图：追踪器用例5

硬件要求

这一节是硬件要求。满足这些条件，外设才能在HTC Vive VR系统中用Vive追踪器来定位和传递特定数据。

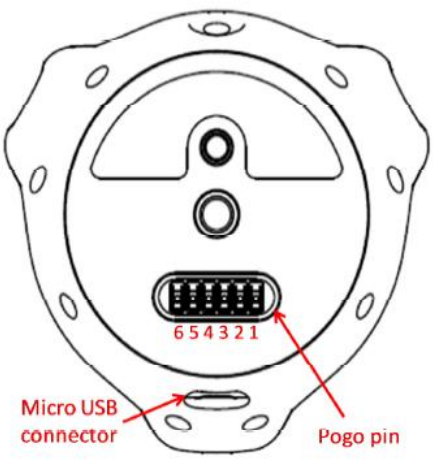
兼容的外设设备可以通过连接Vive追踪器上的USB接口来向PC间接传递数据。同时，Vive追踪器需要和dongle配对来传递数据。下图是这种架设的概念图：



图：Vive 追踪器架设的概念图

接口

通过 micro USB connector 连接达到 USB 2.0 满速。

	Pin 编号	类型	描述
	1	数字输出	通用数字输出
	2	GND	0线
	3	数字/电源输入	通用输入pin：内部上拉电阻 连接到VDD ,低电平有效(手柄侧键) 电源输入pin
	4	数字输入	通用输入pin：内部上拉电阻 连接到VDD ,低电平有效(手柄扳机键)
	5	数字输入	通用输入pin：内部上拉电阻 连接到VDD ,低电平有效(手柄触摸板键)
	6	数字输入	通用输入pin：内部上拉电阻 连接到VDD ,低电平有效(菜单键)

绝对最大额定值

符号	参数	最小值	最大值	单位
V_I	输入电压	-0.5	3.6	V
V_{ESD}	静电放电电压，人体模型	--	4000	V

电特性(供电电压 $V_{DD} = 3.3V$)

符号	Parameter	最小值	典型值	最大值	单位
V_{OH}	高电平输出电压	$V_{DD} - 0.4$	--	--	V
V_{OL}	低电平输出电压	--	--	0.4	V
V_{IH}	高电平输入电压	0.7 V_{DD}	--	--	V
V_{IL}	低电平输入电压	--	--	0.3 V_{DD}	V
I_{OH}	高电平输出电流	20	--	--	mA
I_{OL}	低电平输出电流	4	--	--	mA
I_{IH}	高电平输入电流	--	0.5	10	nA
I_{IL}	低电平输入电流	--	0.5	10	nA

射频 (RF)

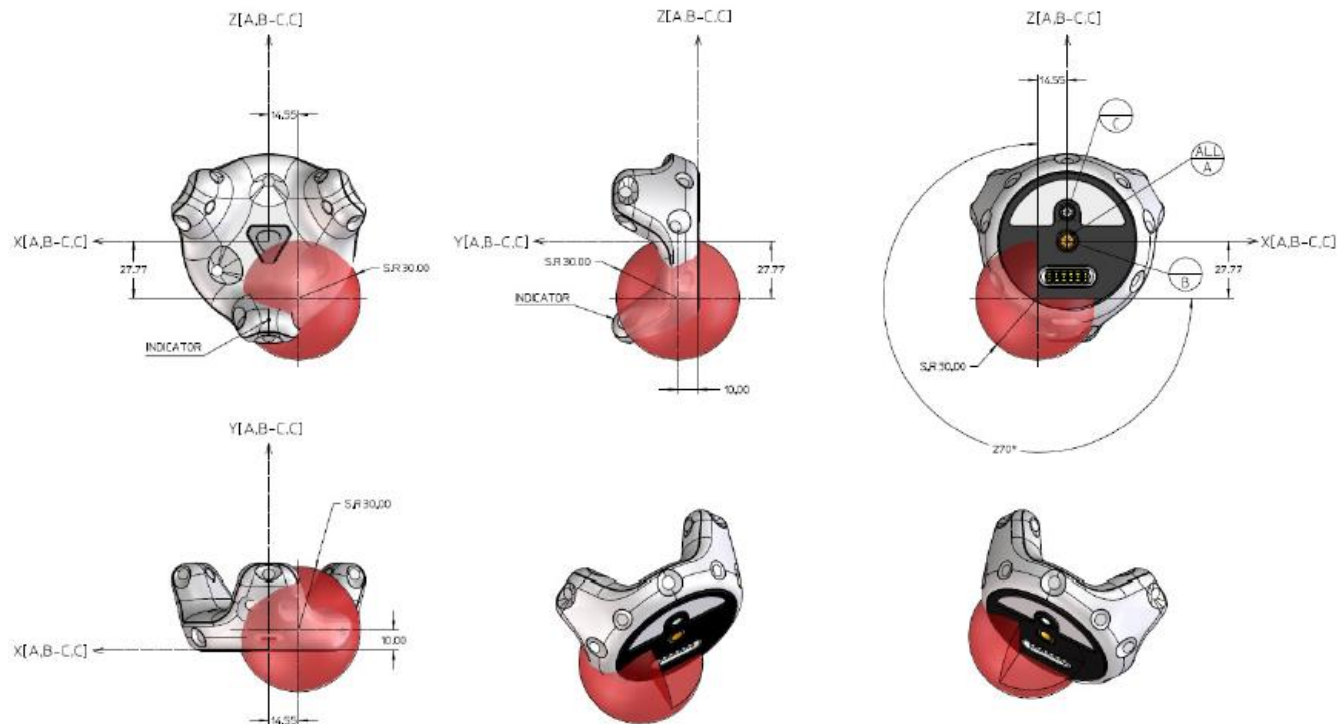
为了保证Vive追踪器和dongle的连接稳定，在附加上外设后，追踪器的OTA性能不能降低到使噪音大于3dB。

为了提高射频性能，建议采取以下建议：

除了必要的部件，比如1/4英寸的螺丝、连接弹簧针 (Pogo pin) 的电板和相关电路，

外设的金属部件应该和天线保持30mm以上的距离以防止影响OTA性能。

下图展示了”远离区”，在远离区里只应该放非金属部件（球半径=30mm，以天线馈电点为球心）。



图：天线的受限制区域

电源

Micro USB connector	电压要求	最大充电电流	最大充电时间
AC	5V +/- 5%	1000 mA	1.5 hrs
PC	5V +/- 5%	500 mA	3 hrs

弹簧针 3	电压要求	最大充电电流	最大充电时间
PC	5V +/- 5%	500 mA	3 hrs

注

AC: D+ short to DPC:

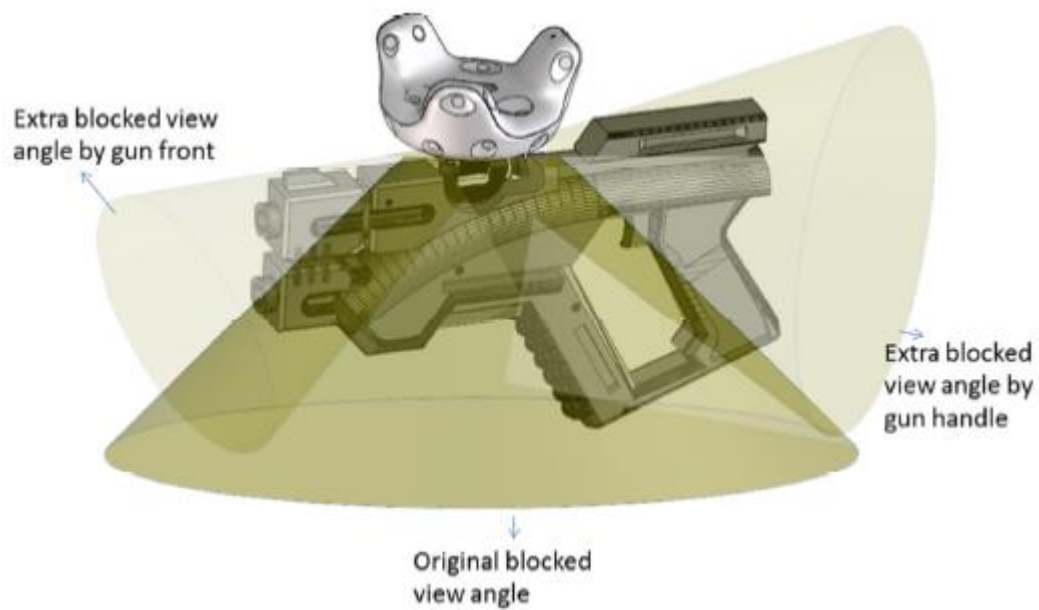
D+/D- communication

表: Micro USB connector 和弹簧针的充电参数

光学

Vive追踪器的视场角是270度。尽量不要放东西在这个视场角内，因为那会阻挡传感器。

如果底座部分伸展出了建议放置区，会有更大的视野盲区。



图：底座伸出了建议放置区

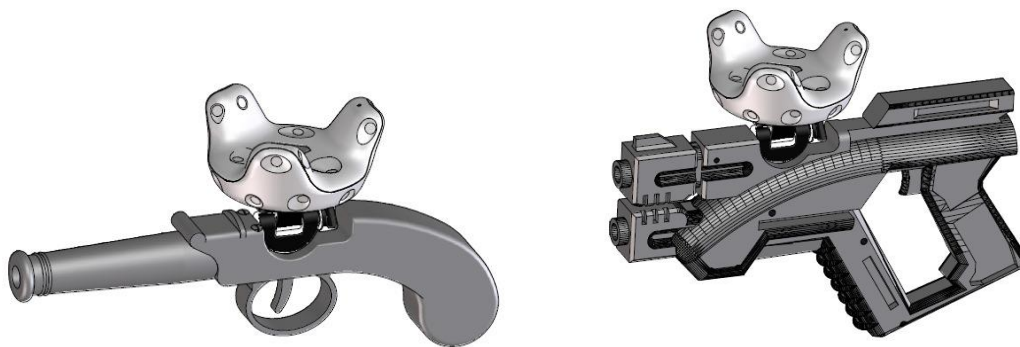
底座

以下是底座部分需要兼容的地方：

- a. 底座需要符合ISO标准 (ISO 1222:2010)。另外Vive追踪器有一些物理限制，比如不能拧进过长的螺丝。
- b. 用户应该能用双手轻松的装拆Vive追踪器，一手拿着追踪器，一手拿着外设
- c. 用户在装拆Vive追踪器时手不会受伤
- d. 用户装拆Vive追踪器时不会不舒服
- e. 外设的外形应该设计成不容易在使用中撞到使用者
- f. 追踪器的定位信号不应该被挡住
- g. 外设的外壳的光学特性必须不影响追踪器使用

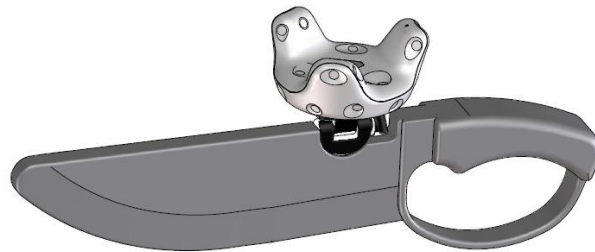
底座设计

枪



剑

建议把结合部位设计得靠近手持区，并且在VR程序中设置长度。

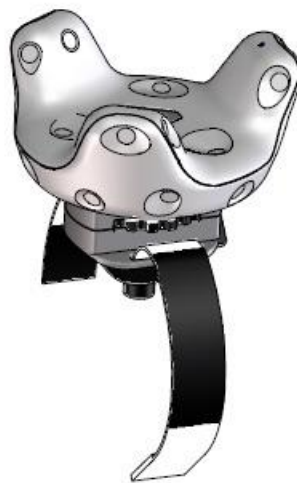


多用途的底座

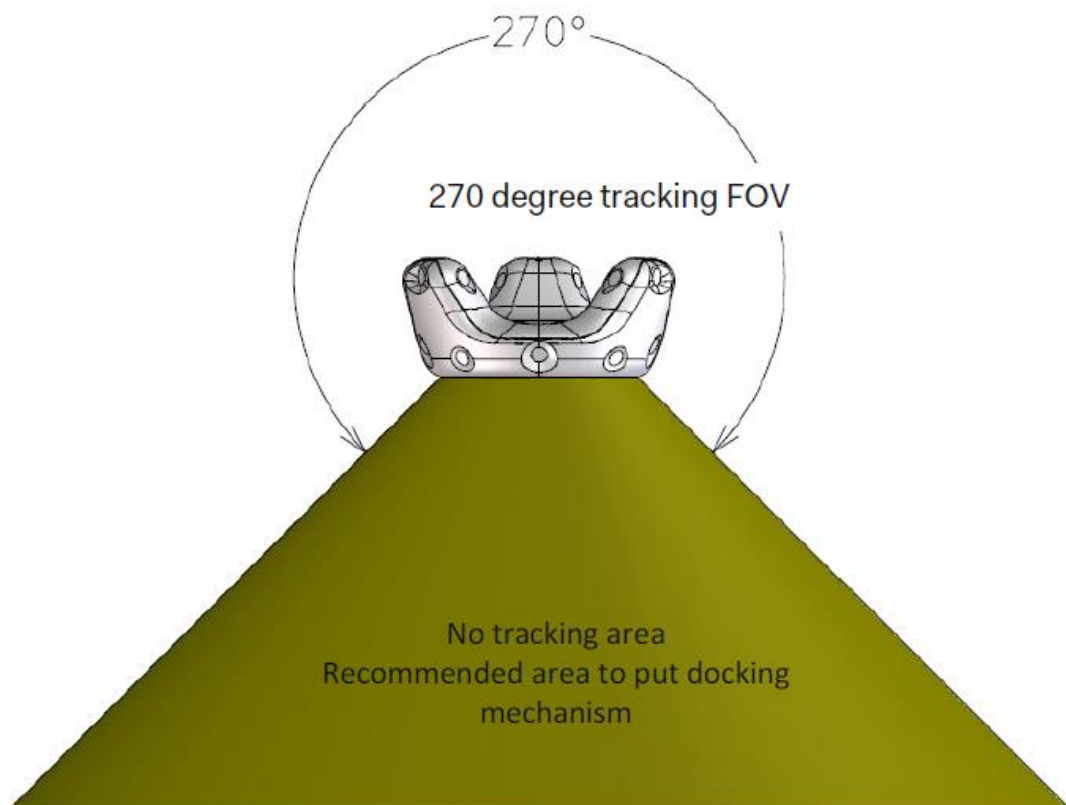
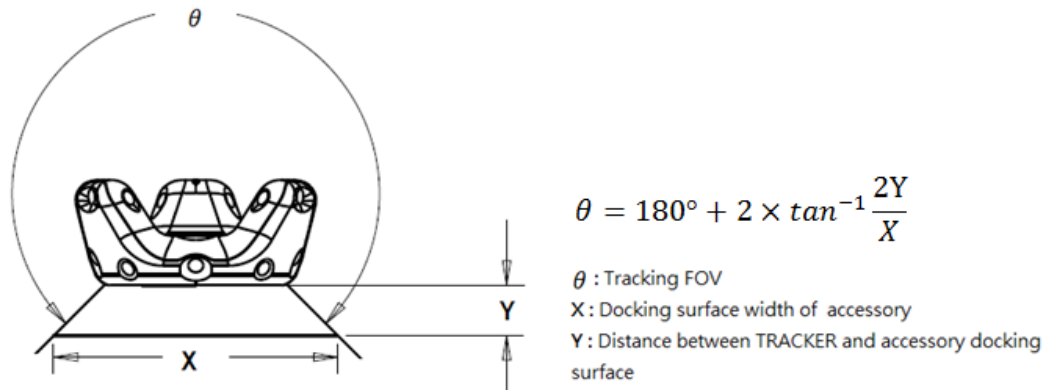
用户可以把Vive追踪器依附到任何要定位的物体和表面。

*如果物体/表面是光滑、坚硬的，建议使用强力胶带把底座粘贴到上面（比如 3m VHB胶带）。

*如果物体/表面是粗糙、软的，建议使用带子把它们捆起来。

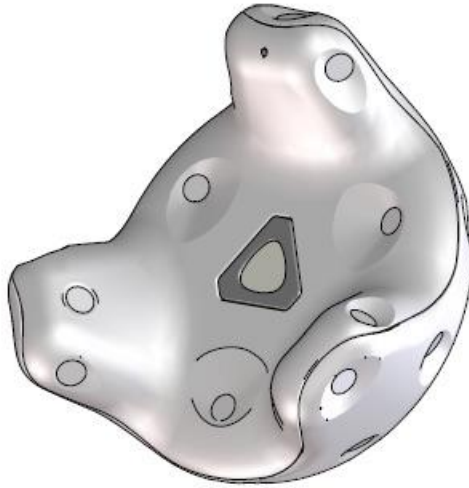


错误的放置方式会让外设挡住Vive追踪器，影响使用。追踪器的视场角大小、底座上表面宽度、追踪器到外设的距离 三者的几何关系如下：



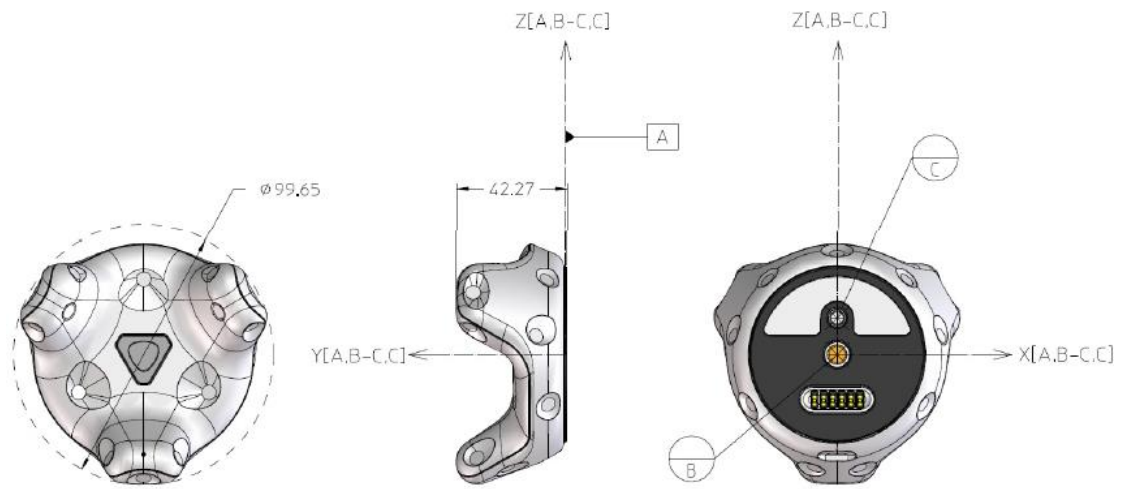
机械方面的考虑

为了方便开发者为Vive追踪器开发兼容的外设,这一节将列举一些机械方面的考虑。



图：Vive追踪器

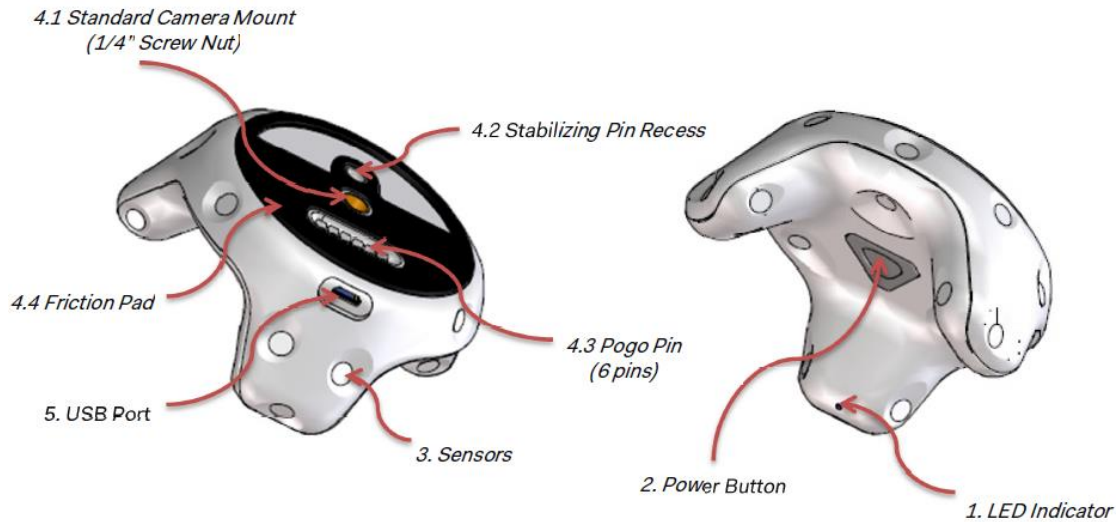
外壳尺寸



图：不同角度下的Vive追踪器

Vive追踪器的总大小是： $\varnothing 9.65\text{mm} * 42.27\text{mm (H)}$ 。

主要特性



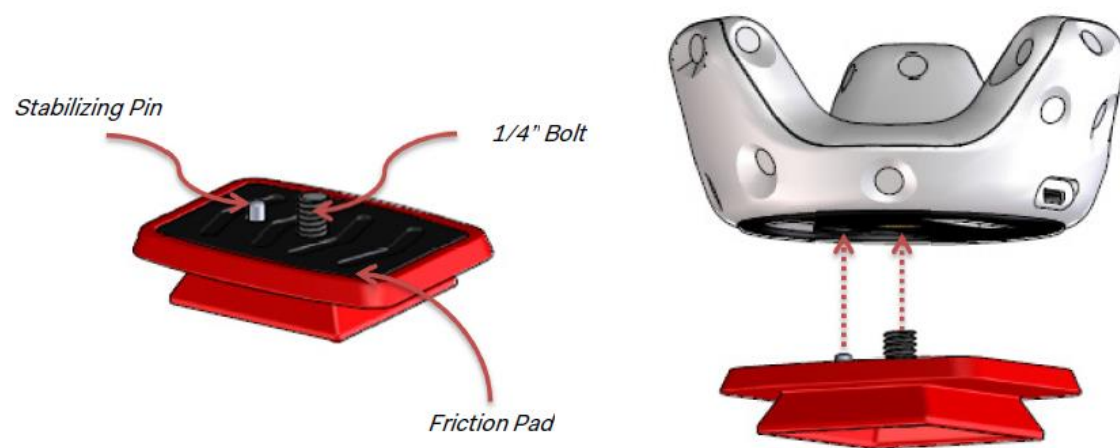
1. **LED灯**：显示Vive追踪器状态
2. **电源键**：用来开/关电源，低能耗蓝牙配对（BLE）
3. **传感器**：接受基站信号。VR系统会用接受到的信号来计算当前位置。外设要尽可能减少表面反射（比如：不用白色外壳）因为那会产生干扰信号进而影响性能。**推荐使用防反射涂料。**
4. **安放到底座**：追踪器使用了标准的三角架放置机制。主要特性包括以下几点：
 - 4.1 1/4英寸的螺母用于固定外设
 - 4.2 一个针槽用于稳定，防止旋转
 - 4.3 弹簧针（Pogo pin）可以在开发者需要的时候连接外设
 - 4.4 防滑衬垫用于进一步稳定追踪器和外设
5. **USB接口**：用于通过micro USB线和外设相连

安置机制

Vive追踪器采用了常用的相机三角架安放方法，符合ISO标准（ISO 1222:2010）。

下图是介绍Vive追踪器如何安置的一些示意图。

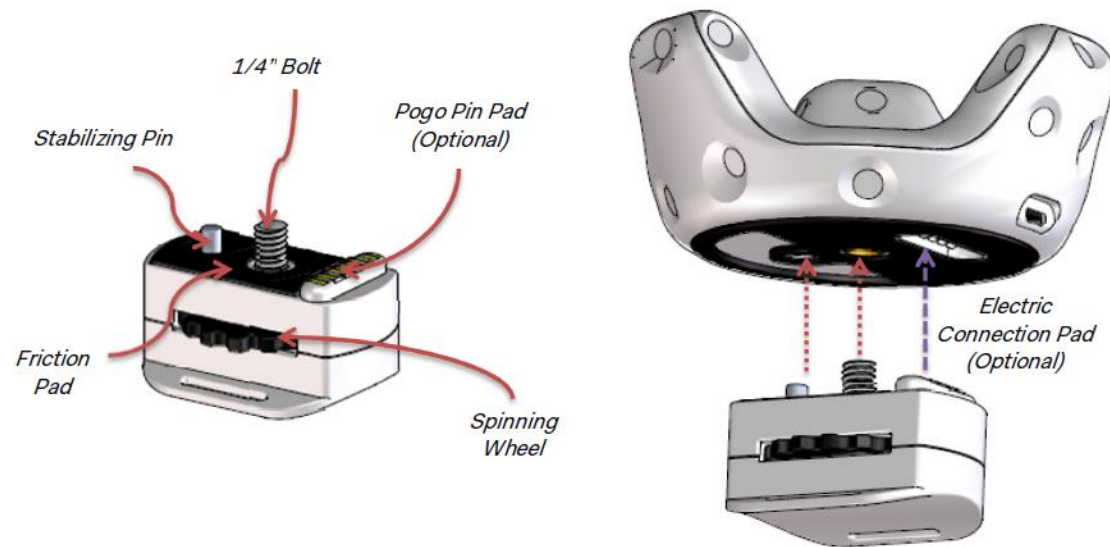
用标准三角架云台安置（不接电）



图：用标准三角架云台安置

和装相机一样，先把Vive追踪器装到云台上，再把云台装到外设上。

使用侧面的转轮拧紧 (有需要的话可以接电)



图：使用侧面的转轮拧紧

开发者可以用侧面的转轮来拧紧底座的螺丝。为了方便操作，侧轮最好直径大于25 mm。用这种机制就可以使用弹簧针 (Pogo pin) 了。

外设设计

以下是不同的外设机制，符合ISO标准：

- 1/4英寸螺栓设计

请参考ISO 1222-2010, 第1页图1

- 稳定针设计

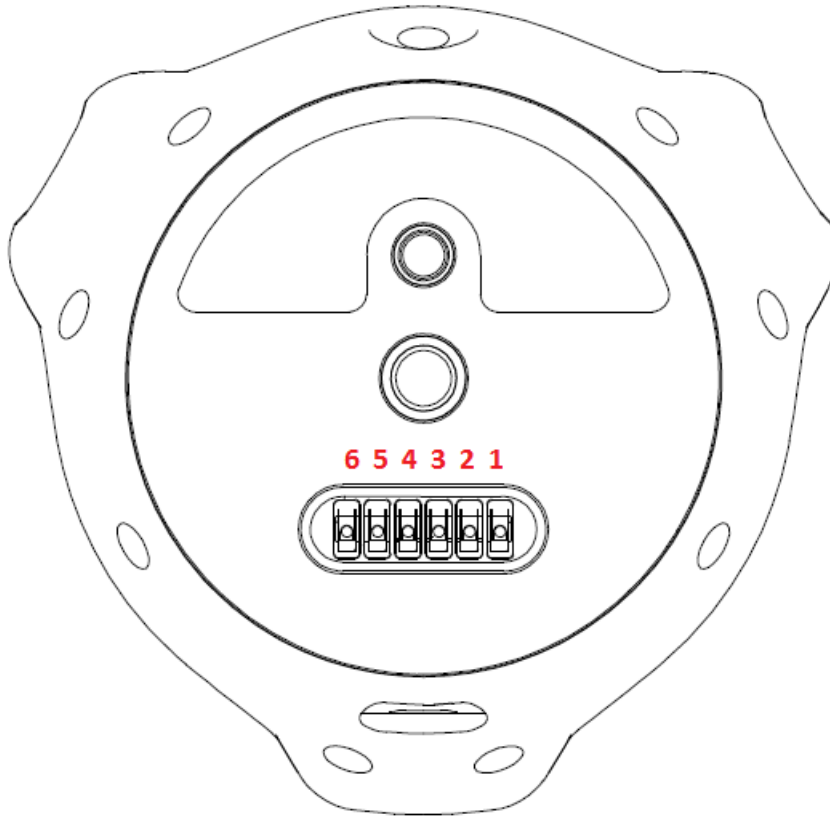
Vive追踪器使用了 ISO 1222-2010 第3页图5的设计。想要查看具体的尺寸和允许范围，请参照 13到17页。建议使用稳定针，这样能让追踪性能更好。

- 螺纹设计

Vive追踪器使用的螺丝是1/4英寸的 ,螺距是1.27 mm。详情请参考ISO 1222-2010 , 第3到第5页。

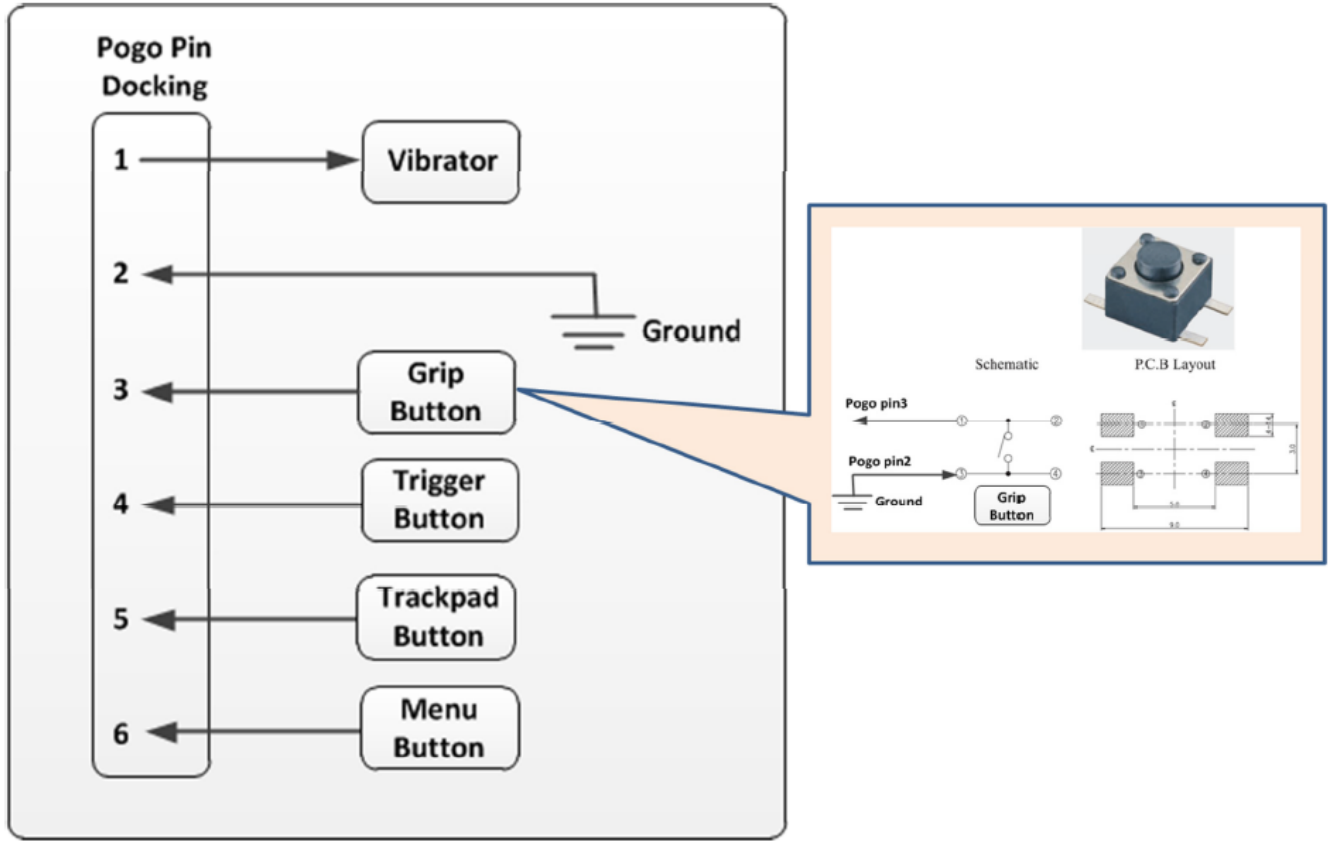
弹簧针 (Pogo pin) 面板的设计

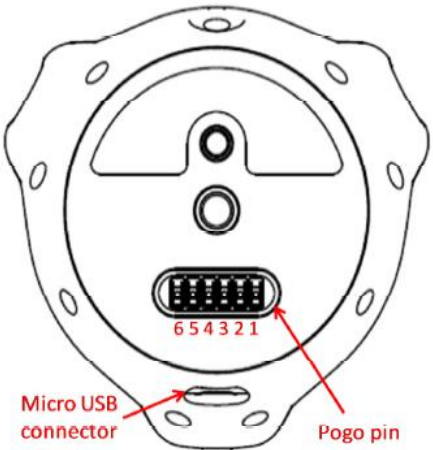
a. 各Pin的定义(Vive追踪器)



b. 弹簧针面板的参考设计

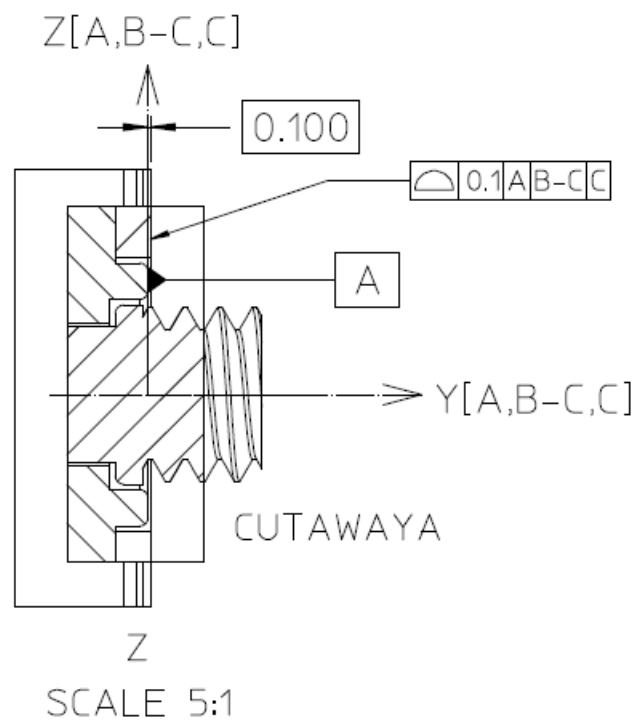
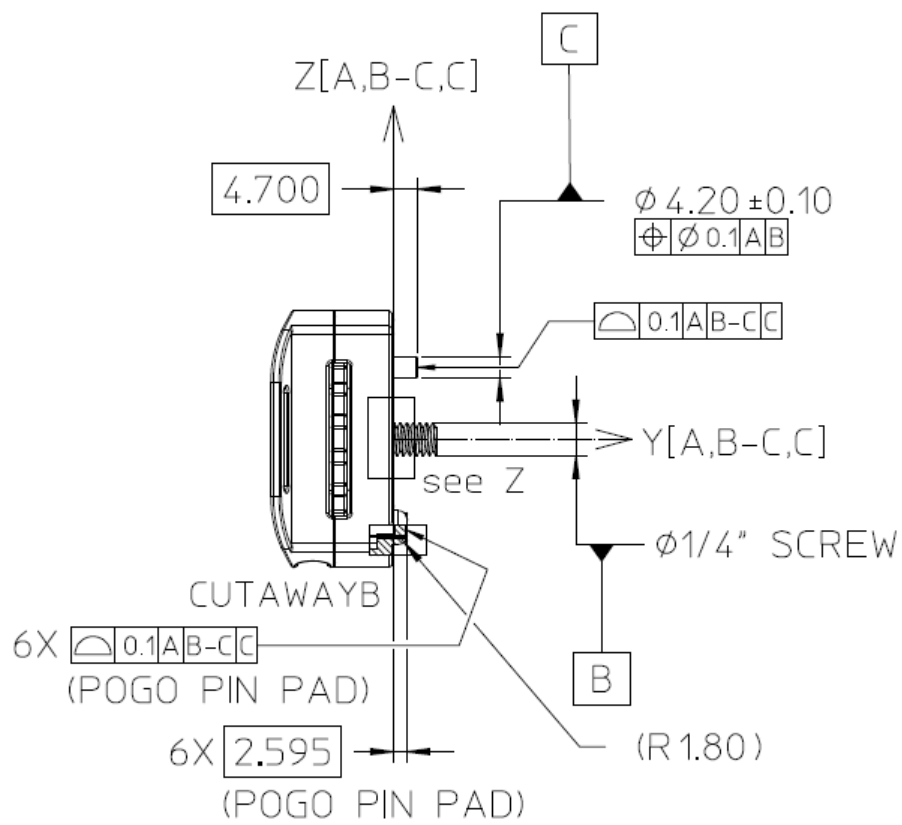
电学

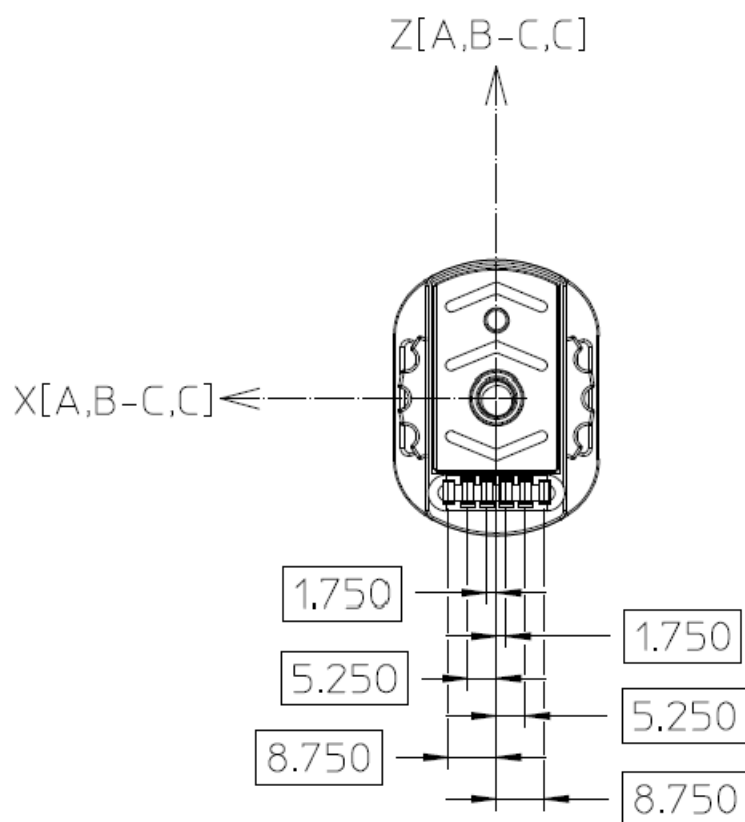
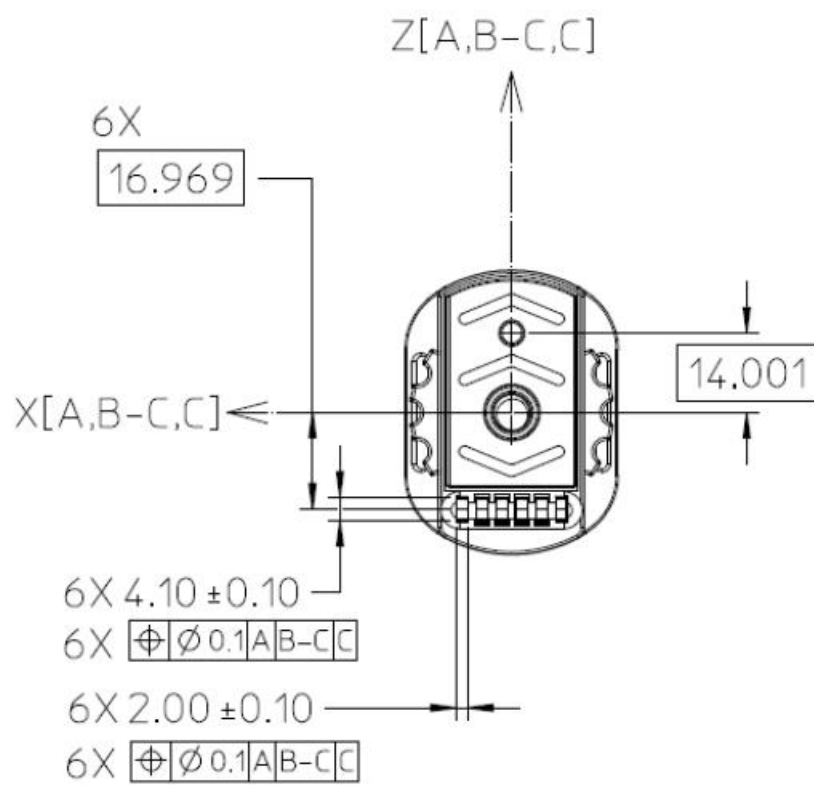


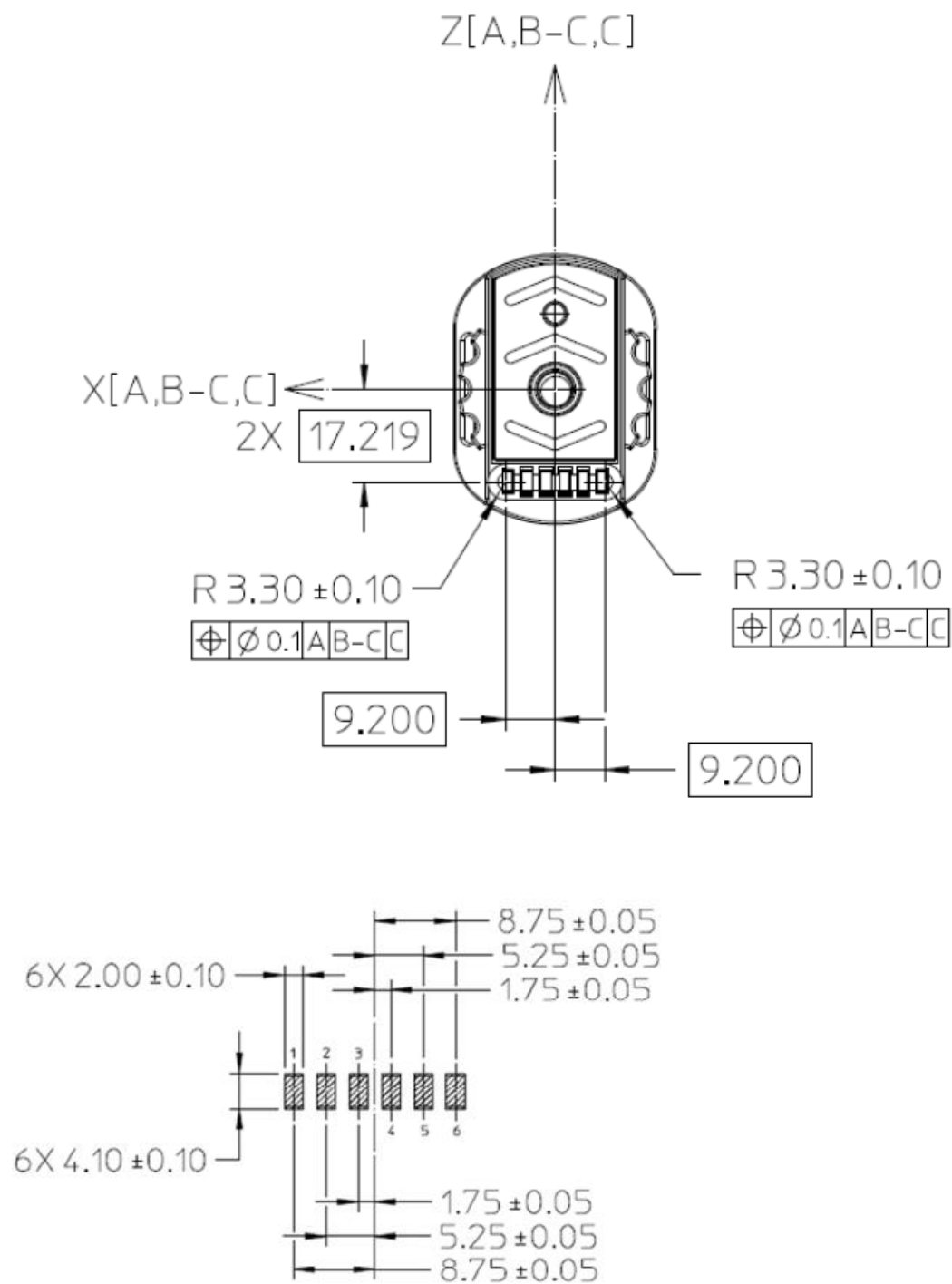
	Pin 编 号	类型	描述
	1	数字输出	通用数字输出
	2	GND	0线
	3	数字/电源输入	通用输入pin：内部上拉电阻 连接到VDD，低电平有效(手柄侧键)

			电源输入pin
	4	数字输入	通用输入pin：内部上拉电阻 连接到VDD ,低电平有效(手柄扳机键)
	5	数字输入	通用输入pin：内部上拉电阻 连接到VDD ,低电平有效(手柄触摸板键)
	6	数字输入	通用输入pin：内部上拉电阻 连接到VDD ,低电平有效(菜单键)

机械







接触电阻：

30 mΩ (Max) 初始值 (在 20mV (Max) 、 100mA 断路时测得)

额定接触电流：

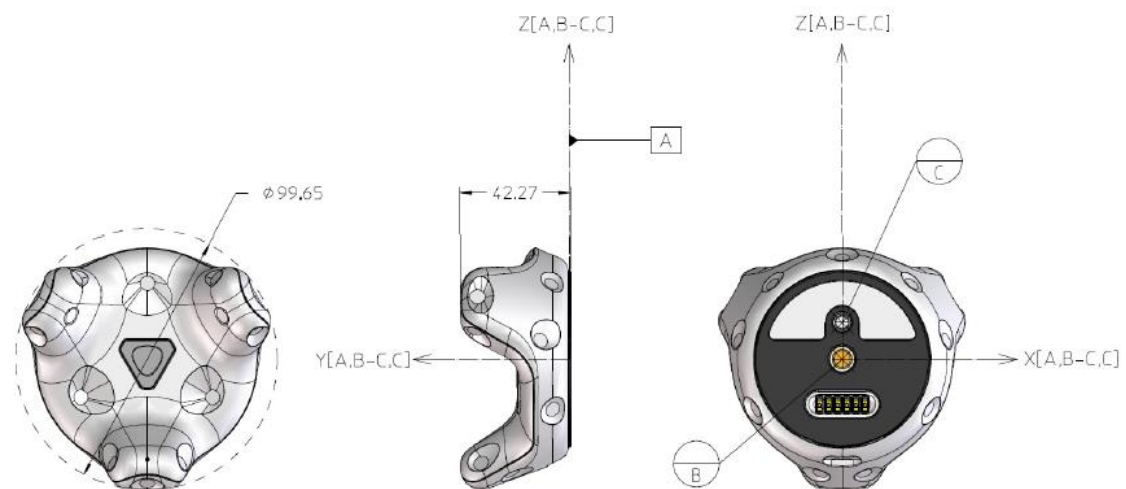
1.8 A (Min)

坐标系统

Vive追踪器的坐标系统是右手坐标系

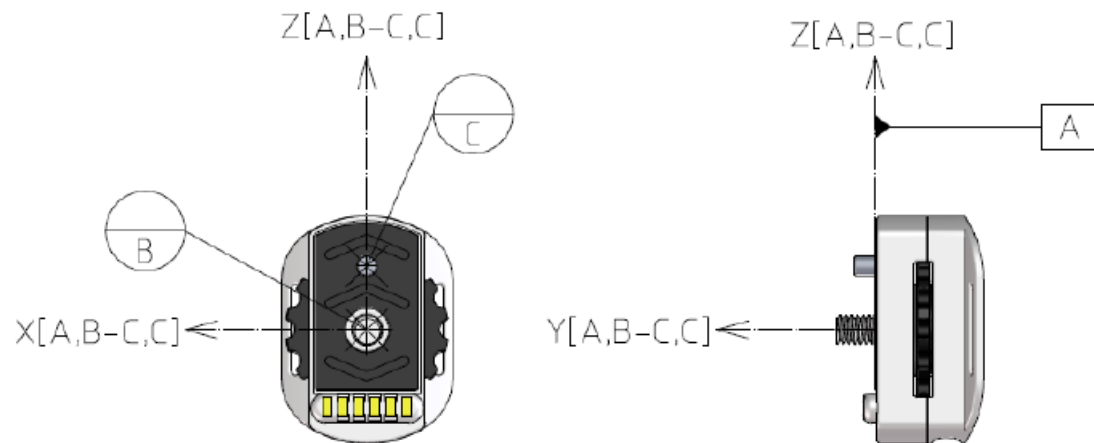
Vive追踪器

- 定义基准面A为：顶部的圆环面，在1/4英寸螺母附近
- 定义基准点B为：标准相机安放点（1/4英寸螺丝）中心线和A的交点
- 定义基准点C为：稳定针槽的中心线与A的交点
- 坐标系统由A的基准框架、BC连线、和C组成



外设

- 定义基准面A为：顶部的圆环面，在1/4英寸螺丝附近
- 定义基准点B为：1/4英寸螺丝中心线和A的交点
- 定义基准点C为：稳定针的中心线与A的交点
- 坐标系统由A的基准框架、BC连线、和C组成



软件部件

这部分描述了 HTC Vive 追踪器的软件部件。

如果你是外设制造商，你可以通过 Vive 跟踪器传输数据。你可以参考本章的数据格式部分找到外设和跟踪器之间数据传输的详细格式。

如果你是内容开发商，可以参考 Unity 整合和外设整合部分来为加载了 Vive 跟踪器的外设制作虚拟现实的内容。

当 Vive 跟踪器发布新的固件版本时，你需要进行更新。通过 USB 线连接 Vive Tracker 后，运行电脑上自带的工具即可。你可以在固件升级部分找到详细的步骤。

系统需求

对于外设制造商和内容开发商：

1. 为了测试 Vive 跟踪器在你的内容或外设上的使用，你需要一套 HTC Vive 头盔和相关的可运行软硬件环境。你可以在 www.htcvive.com 上找到更多信息；
2. 你的电脑至少有一个可用的 USB2.0 接口。一是为了驱动 dongle (之前提到需要用到 dongle 的情况)，二是为了升级 Vive 跟踪器固件版本。

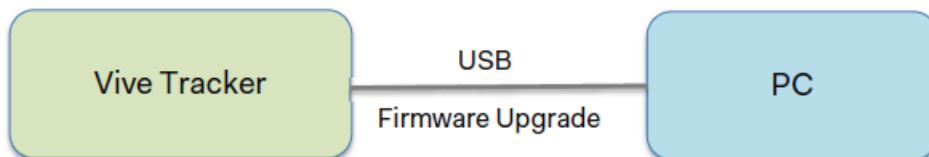


图 Vive 跟踪器和电脑

对于外设制造商：

如果你的外设需要模拟和 Vive 控制器一样的按键功能或者想通过 Vive 跟踪器传输数据到电脑上，你的外设必须分别支持下面的接口：

1. Pogo pin 接口

请参见硬件需求部分寻找按键模拟的详细信息。

2. 高速 USB 接口和 HID (人机交互设备) 协议类

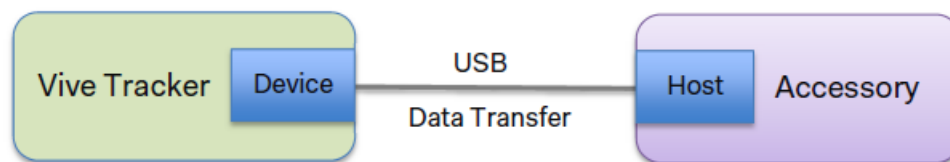


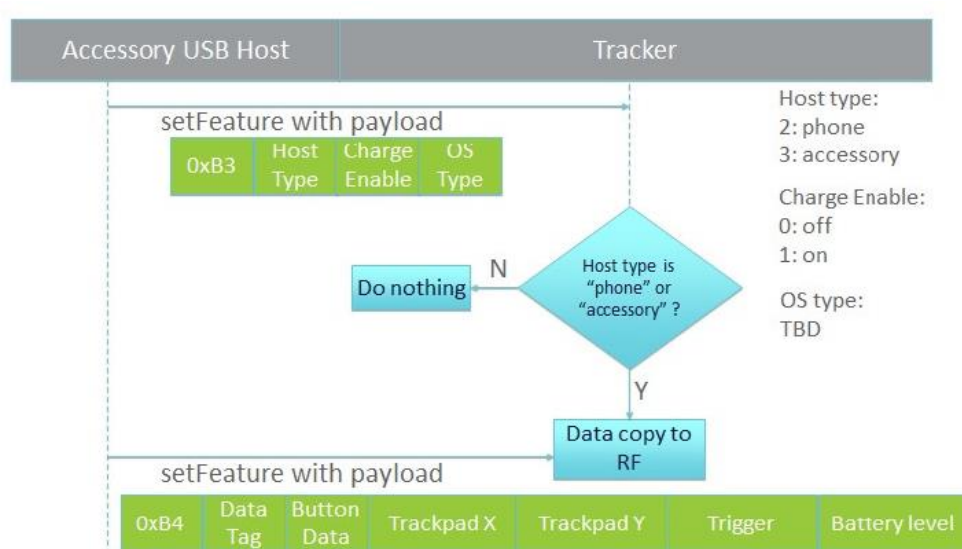
图 Vive 跟踪器和外设

数据格式

这部分描述了当 USB 接口被使用时，外设制造商通过 Vive 跟踪器在电脑和外设间传输数据所需的数据格式。

从外设到 Vive 跟踪器传输的数据格式是通过 USB HID 特征报告传输的。类似于 Vive 控制器的用户接口。**数据传输间隔应该超过 10ms。**

参见下图来了解外设和 Vive 跟踪器之间的 USB 命令流。



SetFeature 0xB3 的数据格式如下：

字节编号	数据	备注
0	主机类型	2：手机 3：外设
1	是否充电	预留
2	OS 类型	预留

SetFeature 0xB4 的数据格式如下：

字节编号	数据	备注
0	标签编号	表示送出数据版本。当前数据格式版本默认值是 0
1	按键	<div>扳手键 0x01</div> <div>缓冲键 0x02</div> <div>菜单键 0x04</div> <div>系统键 0x08</div> <div>平板键 0x10</div> <div>平板键按下 0x20</div> <div>预留 0x40</div> <div>预留 0x80</div>
2	平板 X 值	int16 值范围：-32768~32767
3		
4	平板 Y 值	int16 值范围：-32768~32767
5		
6	扳机键原始值	uint16 值范围：0~65535
7		
8	电量水平	预留
9		

表 数据格式 (Vive 跟踪器外设)

外设整合

这部分描述了外设和 Vive 跟踪器之间位置转换的信息。内容开发者在使用像 Unity 这样的游戏引擎时，可以为使用外设的内容创建正确的旋转和平移操作。

外设的局部坐标系假设 z 轴朝向前方（左手坐标系），Vive 跟踪器连接在外设上的方式如下图。在整合过程中，与 Vive 跟踪器相关的外设旋转角和平移距离通过六个变量单独描述：Roll、Yaw、Pitch 以及 D_x ， D_y ， D_z 。

在设计决定了外设的中心后，基于实际的整合条件可以测量出下面的角度和距离。要了解 Vive 跟踪器中心的详细信息，参考硬件及机械设计相关的指导手册。

下面描述的是一个枪外设的例子：

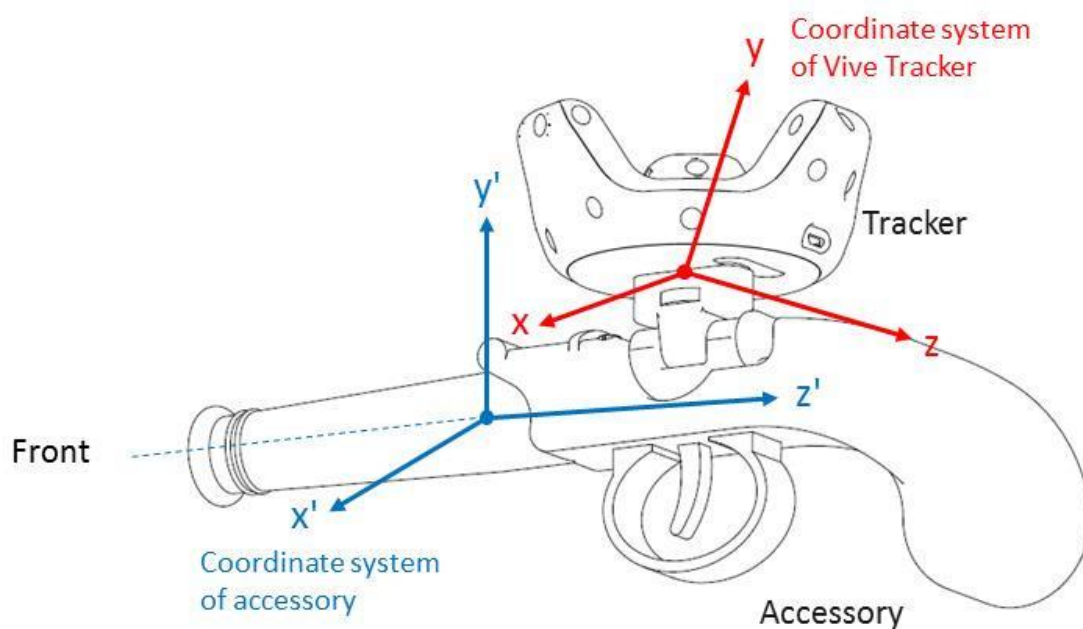


图 整合 Vive 跟踪器到外设的例子

Pitch : 绕 x 轴的角度

Yaw : 绕 y 轴的角度

Roll : 绕 z 轴的角度

D_x : 外设和跟踪器之间 x 轴向距离

D_y : 外设和跟踪器之间 y 轴向距离

D_z : 外设和跟踪器之间 z 轴向距离

内容开发者可以采集上面的信息将跟踪器姿态转换为外设姿态。

假设跟踪器旋转角度矩阵是 $R_{Tracker}$, 外设旋转矩阵 $R_{Accessory} = R_{Pitch_Yaw_Roll} * R_{Tracker}$.

外设位置 $V_{Accessory} = V_{Tracker} + R_{Accessory} * Distance$

下面是一个 Unity 的示例代码 (距离的单位是米 , 角度单位是度):

```
public class Accessory : MonoBehaviour {

    const float dX = 0.0100224f;
    const float dY = -0.07616526f;
    const float dZ = 0.4884118f;

    const float roll = 10.854305f;
    const float yaw = 91.8736f;
    const float pitch = 78.805113f;

    void Update () {

        //Collect delta rotation and displacement between Tracker and Accessory
        Vector3 delta_displacement = new Vector3(dX, dY, dZ);
        Quaternion delta_rotation = Quaternion.Euler(roll, yaw, pitch);

        //Get current Tracker pose
        Vector3 tracker_position = SteamVR_Controller.Input(3).transform.pos;
        Quaternion tracker_rotation = SteamVR_Controller.Input(3).transform.rot;

        //Transform current Tracker pose to Accessory pose
        GameObject.Find("Accessory").transform.rotation = tracker_rotation * delta_rotation;
        GameObject.Find("Accessory").transform.position = tracker_position + (tracker_rotation *
        delta_rotation) * delta_displacement;

    }

}
```

图 外设整合的 Unity 示例代码(1)

另外一个 Unity 示例代码显示了如何通过比较跟踪器 (下面例子中的 AxisY_Tracker, AxisZ_Tracker) 和外设 (下面例子中的 AxisY_Accessory, AxisZ_Accessory) 的平行于 y 轴和 z 轴的矢量来转换外设。

```
public class Accessory : MonoBehaviour {

    const Vector3 AxisY_Tracker = new Vectors(AxisY_Tracker_X, AxisY_Tracker_Y,
AxisY_Tracker_Z);
    const Vector3 AxisZ_Tracker = new Vectors(AxisZ_Tracker_X, AxisZ_Tracker_Y, AxisZ_Tracker_Z);

    const Vector3 AxisY_Accessory = new Vectors(AxisY_Accessory_X, AxisY_Accessory_Y, AxisY_Accessory_Z);
    const Vector3 AxisZ_Accessory = new Vectors(AxisZ_Accessory_X, AxisZ_Accessory_Y, AxisZ_Accessory_Z);

    void Update () {

        //Calculate delta rotation by comparing vectors parallel to Y axes of Tracker and the accessory
        Quaternion delta_rotY = Quaternion.FromToRotation(AxisY_Tracker, AxisY_Accessory);
        AxisZ_Tracker = delta_rotY * AxisZ_Tracker;
        Quaternion delta_rotZ = Quaternion.FromToRotation(AxisZ_Tracker, AxisZ_Accessory);

        //Collect delta rotation and displacement between Tracker and Accessory
        Vector3 delta_displacement = new Vector3(dX, dY, dZ);
        Quaternion delta_rotation = delta_rotZ * delta_rotY;

        //Get current Tracker pose
        Vector3 tracker_position = SteamVR_Controller.Input(3).transform.pos;
        Quaternion tracker_rotation = SteamVR_Controller.Input(3).transform.rot;

        //Transform current Tracker pose to Accessory pose
        GameObject.Find("Accessory").transform.rotation = delta_rotation * tracker_rotation;
        GameObject.Find("Accessory").transform.position = tracker_position + (delta_rotation *
tracker_rotation) * delta_displacement;
    }
}
```

图 外设整合的 Unity 示例代码 (2)

Unity整合

这部分为内容开发者提供了一个在 VR 内容激活 Vive 跟踪器的示例，以 Unity 游戏引擎为例。

首先，你需要让 Vive 跟踪器被 SteamVR 识别。假设你已经有两个 Vive 控制器而且已经在电脑的 USB 口上插入了 dongle。右键点击已存在的控制器图标其中一个，在弹出菜单中选择“配对控制器”（如下图所示）。按住 Vive 跟踪器电源键 2 秒，之后松开进入配对模式。

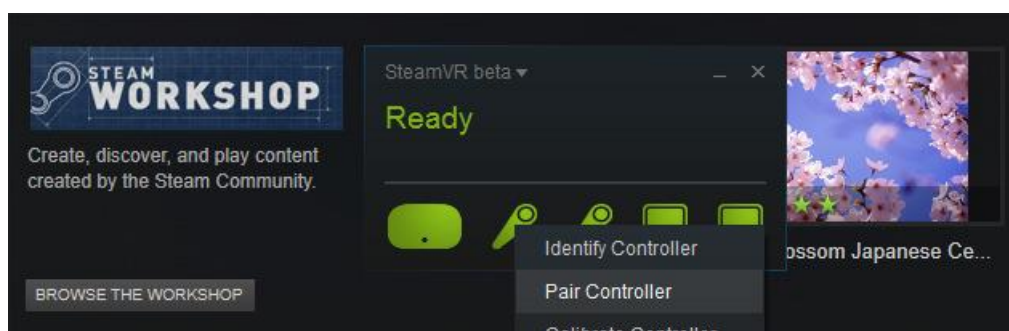


图 配对 Vive 控制器

在 Vive 跟踪器和 dongle 之间配对成功后，你可以在 SteamVR 图形界面上看到 Vive 跟踪器已被识别。



图 Vive 跟踪器被添加至 SteamVR

你可以从下面的网址下载 Unity 最新的个人版本。

<https://store.unity.com/download?ref=personal>



图 示例所使用的 Unity 版本

首先你需要导入 SteamVR 插件至你的项目。可以在 Unity 的资源商店中下载。



图 Unity 资源商店

在 Vive 跟踪器的开发者版本中将使用类似于你为 Vive 控制器创建内容时用的方法和命名规则。使用 Vive 跟踪器创建内容的步骤如下：

步骤一：添加“Camera Rig”至层级界面。

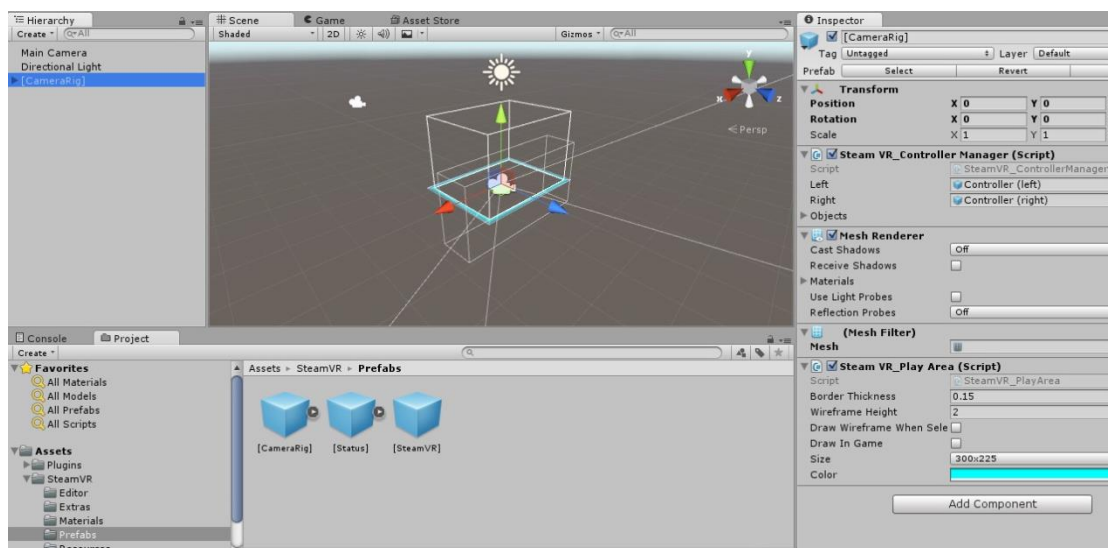


图 添加“Camera Rig”

步骤二：为 Vive 跟踪器创建 3D 物体。在此例子中使用“Capsule”。

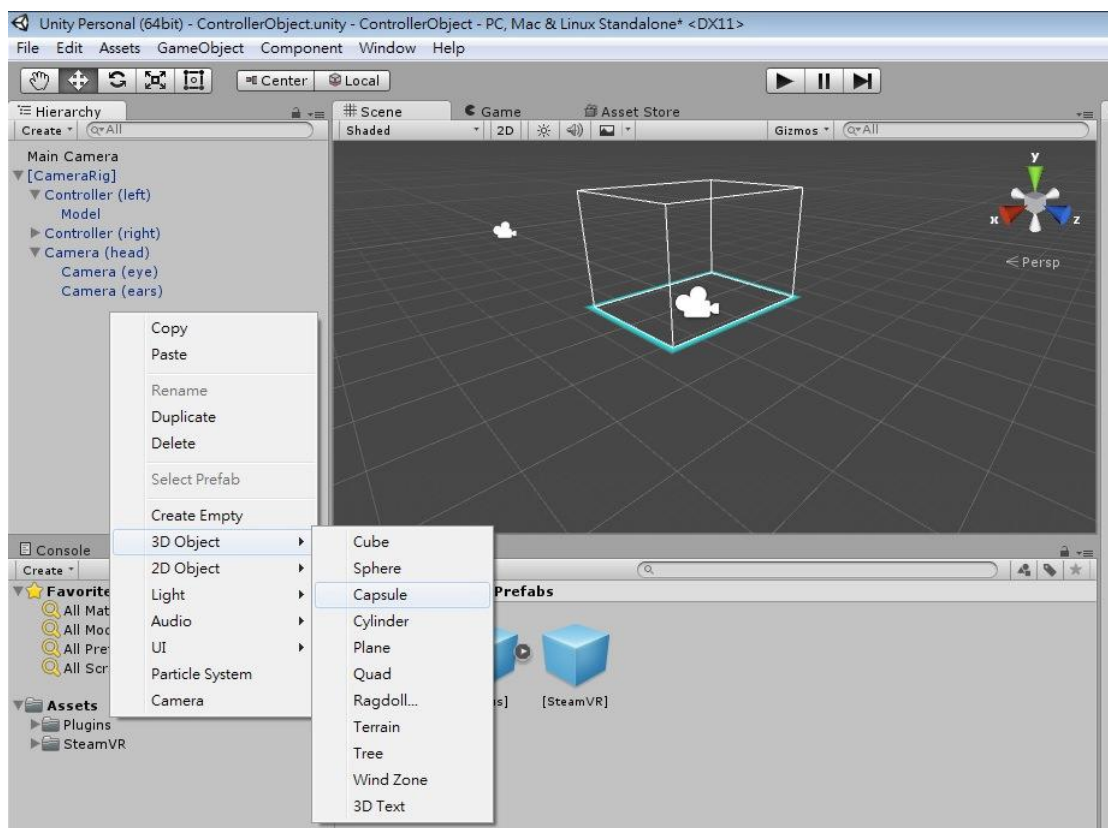


图 创建 3D 物体

步骤三：添加“SteamVR_Tracked_Object”脚本至“Capsule”物体。

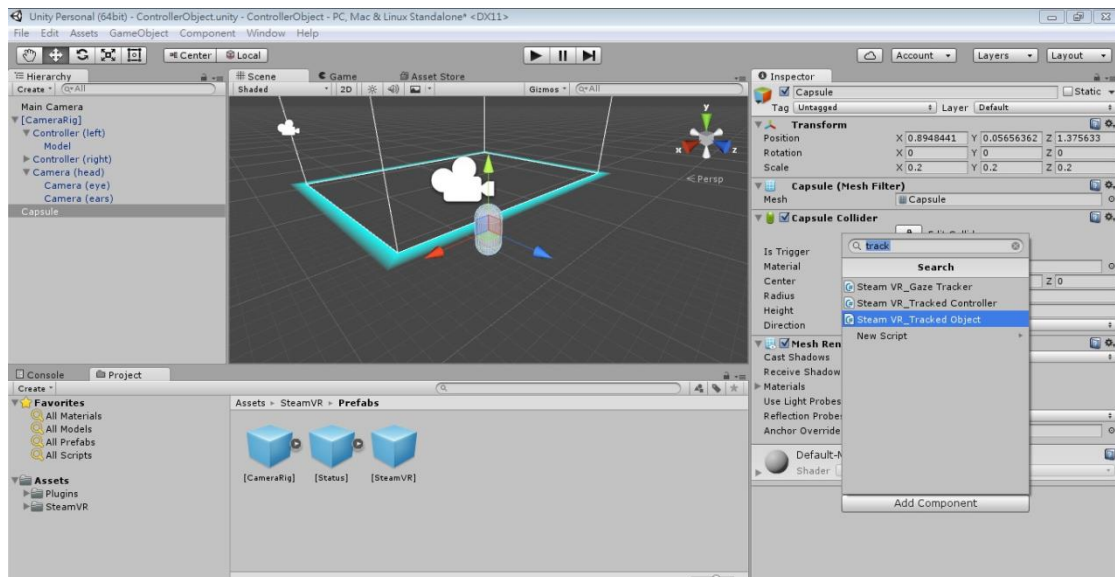


图 添加“SteamVR_Tracked_Object”脚本

步骤四：设置“SteamVR Controller Manager”中的物体数量。在此例子中，设置过程只使用了一个 Vive 跟踪器。

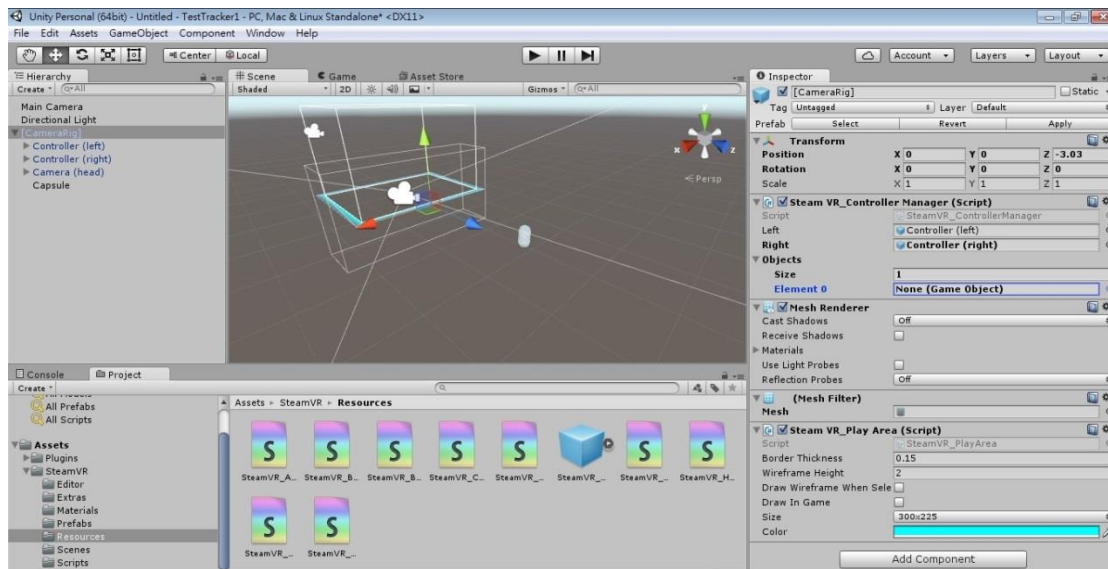


图 设置“SteamVR Controller Manager”中的物体数量

步骤五：设置“Capsule”物体至“SteamVR Controller Manager”中的物体：“Element0”。

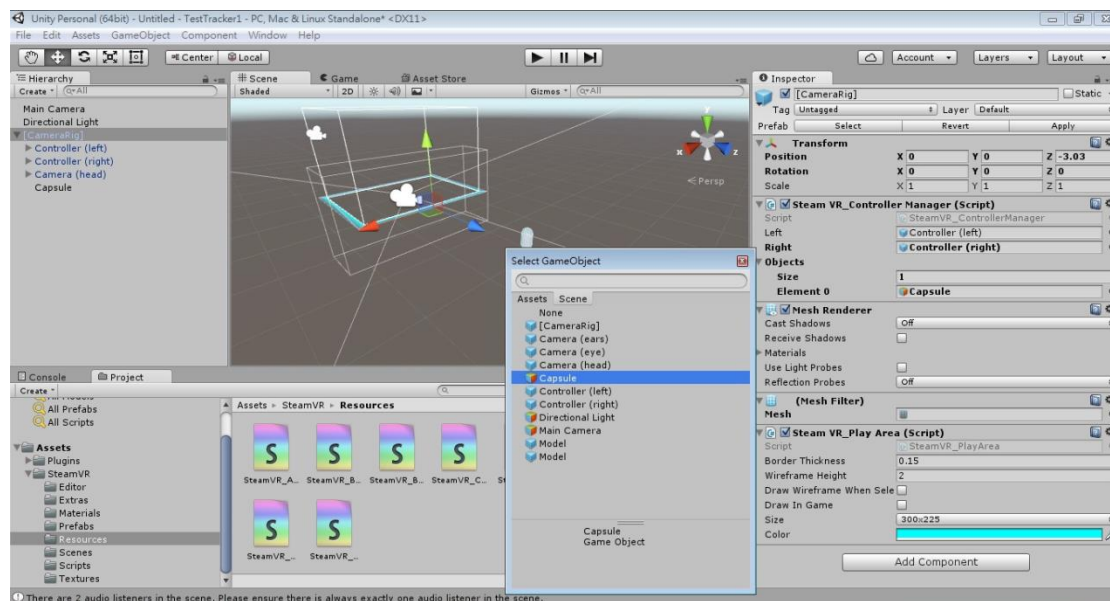


图 设置“SteamVR Controller Manager”中的物体

步骤六：完成上面的步骤后，按 Unity 中的运行按钮。当你移动 Vive 跟踪器时，你将会看到 Capsule 跟随着移动。



图 运行 Unity

固件升级

在 Vive 跟踪器的开发者版本中 ,用于固件升级的 ROM 文件由 HTC 提供。使用 SteamVR 自带的“lighthouse_watchman_update.exe”程序 , 文件目录是 “SteamVR\tools\lighthouse\bin\win32\”。如果你正确安装了 HTC Vive 头盔的话应该可以正常使用。

对于在设计验证测试阶段 (DVT) 的设备 , 请先按照下面的步骤更新固件。固件文件为 upgradeFirmware.zip。

*注意 : 请检查你的 Vive Tracker 电源键上的设备号。如果设备号以 D 开头 , 它就是处于 DVT 阶段的设备。

1. 如果电脑上连接了 Vive 控制器 , 先拔出它。
2. 通过 USB 线链接 Vive Tracker。
3. 解压 upgradeFirmware.zip 文件。
4. 执行 upgradeFirmware.exe 文件。
5. 执行完毕 , 移除 tracker 连接线 , 重启设备。
6. 你将会在 SteamVR beta 界面看到新的 Vive Tracker 图标。

对于在 DVT 以及通过 DVT 的设备 , 根据 Steam 的提示或按照下面的步骤更新 Vive Tracker 的固件 :

1. 复制由 HTC 提供的固件二进制文件 (包括 MCU , FPGA 和 RF) 至 “lighthouse_watchman_update.exe”文件同目录。

2. 如果 Vive 控制器插在电脑上的话，拔出。

3. 通过 USB 线连接 Vive 跟踪器。

4. 在命令行窗口执行下面的命令升级固件。

a. 升级 MCU 固件版本：

```
lighthouse_watchman_update -mn tracker_mcu_filename.bin
```

b. 升级 FPGA 的固件版本：

```
lighthouse_watchman_update -fn tracker_fpga_filename.bin
```

c. 升级 RF 的固件版本：

```
lighthouse_watchman_update -rn tracker_rf_filename.bin
```