Vive Input Utility 开发向导

Email: vivesoftware@htc.com

Forum: http://community.viveport.com

GitHub: https://github.com/ViveSoftware/ViveInputUtility-Unity

Wiki: https://github.com/ViveSoftware/ViveInputUtility-Unity/wiki

翻译:
Github:
Email:me@ms0ng.com
翻译说明:
部分我看的不太明白的地方会用 加粗中文(原句) 的方式进行翻译,有纰漏或者错误在QQ群里或者发Email,这份文档可能会发布在Github上,届时也可以在Github中提交issue.

About

Vive Input Utility(一下简称VIU) 是一个Unity插件,它允许开发者们获取Vive设备的信息,包括Vive Tracker

VIU提供了在3D空间中实现手柄操作的方法,并且用Unity Event System实现. 除此之外,VIU还提供了一个设备绑定系统来管理多个**跟踪设备(tracking devices)**

我们的目标是加快Unity开发者门开发VR的速度,发掘优秀的VR体验,以及省略写大量冗余的代码来管理Vive设备的时间

Motivation

SteamVR插件提供了C#接口给开发者用于与Vive设备的交互

但是,要掌握玩意儿需要写很长很烦的代码:

- 当手柄连接时,你必须持续不断地用SteamVR_ControllerManager取得正确的**设备序号(device index)**
- 查找SteamVR_ControllerManager也需要废很多时间

所以咱们的目标是提供更便捷的接口以减少代码冗余

主要特性

- 使用静态方法检索设备输入:
  - 按钮的事件
  - **姿势跟踪(Tracking pose)**
- 使用ViveRaycaster组件与 Unity Event System实现3D空间中的手柄操作

静态接口(Static Interface)

- 获取按钮事件

用**SteamVR** 来实现已经是一种CXK行为...

```
using UnityEngine;
using Valve.VR;

public class GetPressDown_SteamVR : MonoBehaviour
{
    public SteamVR_ControllerManager manager;
    private void Update()
    {
        // 扣下扳机
        SteamVR_TrackedObject trackedObj = manager.right.GetComponent<SteamVR_TrackedObject>();
        SteamVR_Controller.Device rightDevice = SteamVR_Controller.Input((int)trackedObj.index);
        if (rightDevice.GetPressDown(EVRButtonId.k_EButton_SteamVR_Trigger))
        {
            // ...
        }
    }
}
```

ViveInput 类(在 **HTC.UnityPlugin.Vive** 下)提供了一种更简单的方式去实现:

```
using UnityEngine;
using HTC.UnityPlugin.Vive;

public class GetPressDown_ViveInput : MonoBehaviour
{
    private void Update()
    {
        // 扣下扳机
        if (ViveInput.GetPressDownEx(HandRole.RightHand, ControllerButton.Trigger))
        {
            // ...
        }
    }
}
```

也可以获取手柄更详细的操作信息,比如扳机的**扣动程度(axis values)**

```
using UnityEngine;
using HTC.UnityPlugin.Vive;

public class GetAxisValue_ViveInput : MonoBehaviour
{
    private void Update()
    {
        // 取扳机扣动程度 axis value
        if (ViveInput.GetAxisEx(HandRole.RightHand, ControllerAxis.Trigger) > 0.5f)
        {
            // ...
        }
    }
}
```

- 监听按钮的事件

ViveInput 同样提供了**通过回调的监听器**(callback style listener).

```
using UnityEngine;
using HTC.UnityPlugin.Vive;

public class GetPressDown_ViveInputHandler : MonoBehaviour
{
    private void Awake()
    {
        ViveInput.AddListenerEx(HandRole.RightHand, ControllerButton.Trigger, ButtonEventType.Down, OnTrigger);
    }

    private void OnDestroy()
    {
        ViveInput.RemoveListenerEx(HandRole.RightHand, ControllerButton.Trigger, ButtonEventType.Down, OnTrigger);
    }

    private void OnTrigger()
    {
        // ...
    }
}
```

- 取**追踪姿势**(**tracking pose**)

VivePose 类(在 **HTC.UnityPlugin.Vive** 下)提供了一个API用于取得 device pose.

其返回值类型为 **RigidPose**（在 **HTC.UnityPlugin.Utility**下）.

```
using UnityEngine;
using HTC.UnityPlugin.Vive;
using HTC.UnityPlugin.Utility;

public class GetPose_VivePose : MonoBehaviour
{
    private void Update()
    {
        RigidPose pose1 = VivePose.GetPoseEx(HandRole.RightHand);
        RigidPose pose2 = VivePose.GetPoseEx(TrackerRole.Tracker1);
        // 设置 transform 的值为两个pose中间的点
        if (VivePose.IsValidEx(HandRole.RightHand) && VivePose.IsValidEx(TrackerRole.Tracker1))
        {
            transform.localPosition = Vector3.Lerp(pose1.pos, pose2.pos, 0.5f);
            transform.localRotation = Quaternion.Lerp(pose1.rot, pose2.rot, 0.5f);
        }
    }
}
```

(译注:这里的pose应该指的是Vive的手柄...吧???我之前好像把它直译成姿势了)

- 识别设备

通过给设备定义**HandRole, TrackerRole**或者 **BodyRole**（在**HTC.UnityPlugin.Vive**下）来对设备进行识别和分类

- **HandRole:** 大多数情况下映射的是带有按钮的手柄, 除了 ExternalCamera. ExternalCamera是映射给第三个手柄 或者是第一个tracker用的. (选项中有 RightHand, LeftHand, ExternalCamera, Controller4~15)
- **TrackerRole:** 映射的是通用的 tracker, 比如 **Vive Tracker**. (选项中有Tracker1~13)
- **BodyRole:** 映射的是手柄和 tracker, 这些手柄和tracker 是与玩家的头, 身体和四肢绑定的(**mapping to controllers and trackers depends on their position related to player's head, body and limbs**). (选项中有Head, RightHand, LeftHand, RightFoot, LeftFoot, Hip)

- Get more device information detail

Class **VRModule** under **HTC.UnityPlugin.VRModuleManagement** provides an API to get more about device info besides button state and tracking pose, such as connecting state, serial number, model number, device class…

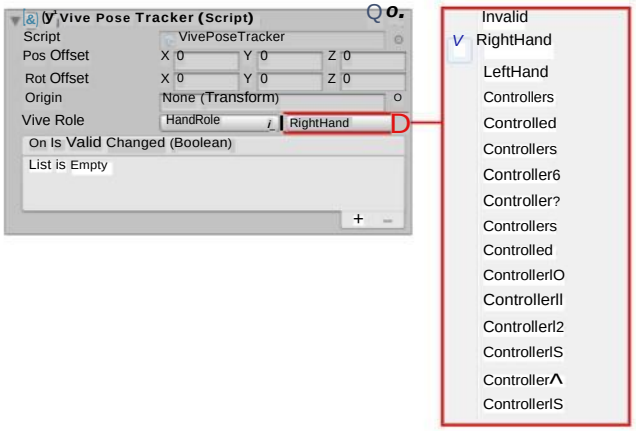
Notice that class **ViveRole** under **HTC.UnityPlugin.Vive** is responsible for mapping role to device index, and **VRModule.GetDeviceIndex** requires device index as its argument, not role enum.

```
using UnityEngine;
using HTC.UnityPlugin.Vive;
using HTC.UnityPlugin.VRModuleManagement;

public class PrintDeviceState_VRModule : MonoBehaviour
{
    private uint m_deviceIndex;
    private void Update()
    {
        var deviceIndex = ViveRole.GetDeviceIndexEx(HandRole.RightHand);
        if (m_deviceIndex != deviceIndex)
        {
            m_deviceIndex = deviceIndex;
            if (VRModule.IsValidDeviceIndex(deviceIndex))
            {
                var deviceState = VRModule.GetDeviceState(deviceIndex);
                Debug.Log("HandRole.RightHand is now mapped to device " + deviceIndex);
                Debug.Log("serialNumber=" + deviceState.serialNumber);
                Debug.Log("modelNumber=" + deviceState.modelNumber);
                Debug.Log("renderModelName=" + deviceState.renderModelName);
                Debug.Log("deviceClass=" + deviceState.deviceClass);
                Debug.Log("deviceModel=" + deviceState.deviceModel);
            }
            else
            {
                Debug.Log("HandRole.RightHand is now mapped to invalid device");
            }
        }
        else
        {
            if (VRModule.IsValidDeviceIndex(deviceIndex))
            {
                var deviceState = VRModule.GetDeviceState(deviceIndex);
                Debug.Log("velocity=" + deviceState.velocity);
                Debug.Log("angularVelocity=" + deviceState.angularVelocity);
                Debug.Log("position=" + deviceState.position);
                Debug.Log("rotation=" + deviceState.rotation);
                Debug.Log("buttonPressed=" + deviceState.buttonPressed);
                Debug.Log("buttonTouched=" + deviceState.buttonTouched);
            }
        }
    }
}
```

Helper Components

- **Vive Pose Tracker**

It works like SteamVR_TrackedObject,but targets device by using ViveRole.DeviceRole instead of device index.

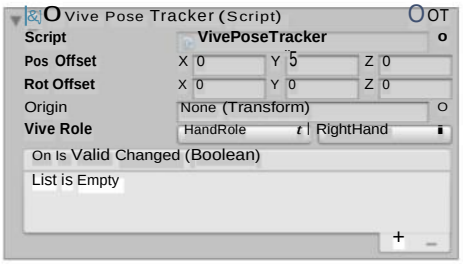


- **Pose Modifier**

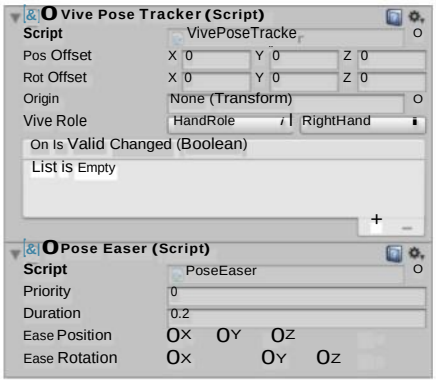
It is a tracking effect script applied to a pose tracker.

Implement abstract class PoseTracker.BasePoseModifierto write custom tracking effect.

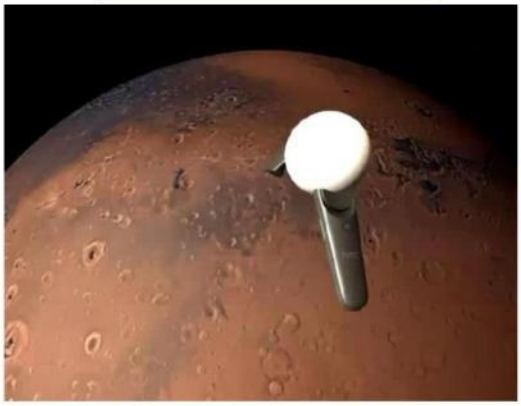
Without pose modifier



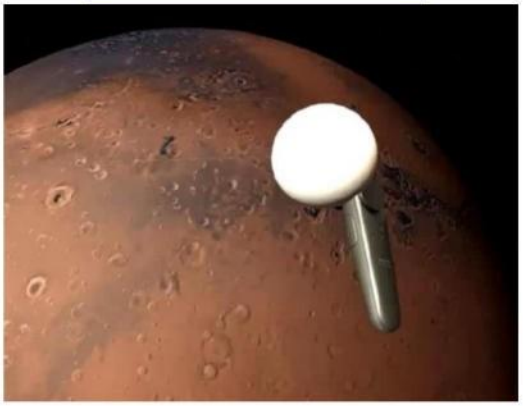
With pose modifier



https://vimeo.com/171724218



https://vimeo.com/171724270



Multiple modifiers are allowed. Priority determines the pose modified order.

- Pose Stabilizer

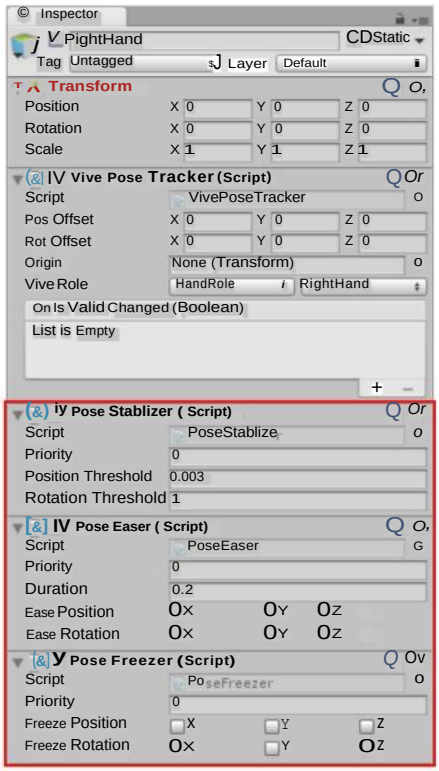
  If controller moves within the threshold, the object stays. Otherwise, the object keeps the offset and follows.

- Pose Easer

  Let the object follows controller using easing curve.

- Pose Freezer

  It constrains the object following movement by setting axis flags.



- **Vive Raycaster & Raycast Method**

Custom Pointer3DRaycaster implement for Vive controller to achieve 3D-space-pointer that compatible with Unity Event System.

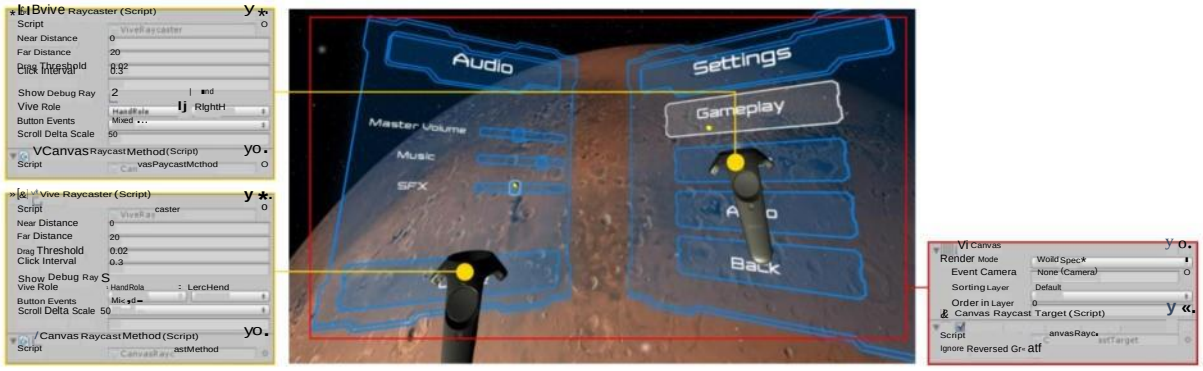
Vive Raycaster is an event raycaster script that sends Vive button's event from its transform.

That means your controller can act like a 3D mouse by combining Vive Pose Tracker and Vive Raycaster.

A Vive Raycaster must works with Raycast Method to raycast against different types of objects.

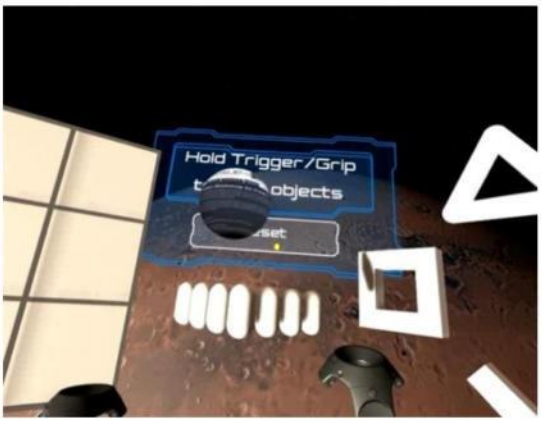
Raycast Method	Against Type
Physics Raycast Method	Collider
Physics 2D Raycast Method	Collider2D
Graphic Raycast Method	Graphic in target Canvas
Canvas Raycast Method	Graphic in all Canvas with CanvasRaycastTarget component

For example, you can arrange them like this to interact with UGUI menu:



More examples:

https://vimeo.com/169824408



https://vimeo.com/169824438

- Raycaster Event Handler

You must implement an Event System Handler to catch an event sent by an event raycaster.

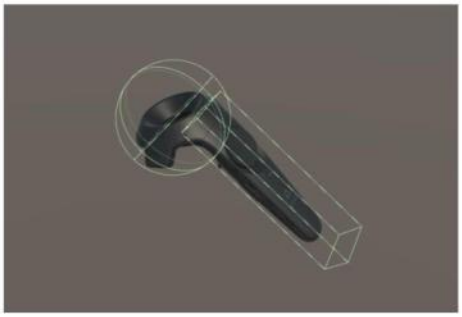
- Add a component that implements event handler interfaces that are supported by the built-in Input Module on an object. See https://docs.unity3d.com/Manual/SupportedEvents.htmlto learn more about built-in event handlers.
- Add event receiver (Collider/Collider2D/Graphic) to the object or child of the object.

```
using UnityEngine;
using UnityEngine.EventSystems;
using System.Collections.Generic;
using HTC.UnityPlugin.Vive;

public class MyPointerEventHandler : MonoBehaviour
    , IPointerEnterHandler
    , IPointerExitHandler
    , IPointerClickHandler
{
    private HashSet<PointerEventData> hovers = new HashSet<PointerEventData>();

    public void OnPointerEnter(PointerEventData eventData)
    {
        if (hovers.Add(eventData) && hovers.Count == 1)
        {
            // turn to highlight state
        }
    }

    public void OnPointerExit(PointerEventData eventData)
    {
        if (hovers.Remove(eventData) && hovers.Count == 0)
        {
            // turn to normal state
        }
    }

    public void OnPointerClick(PointerEventData eventData)
    {
        if (eventData.IsViveButton(ControllerButton.Trigger))
        {
            // Vive button triggered!
        }
        else if (eventData.button == PointerEventData.InputButton.Left)
        {
            // Standalone button triggered!
        }
    }
}
```

- **Vive Collider Event Caster**

Like Vive Raycaster, Vive Collider Event Caster is also an extension of Unity Event System.

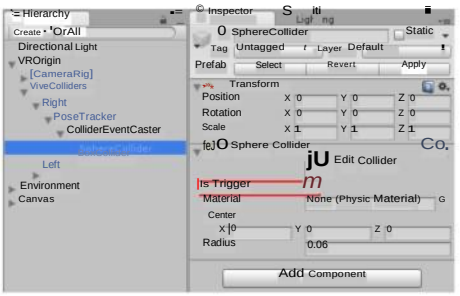
But the Collider Event Caster isn't driven by any input module, so they can work together at the same time if needed.



Instead of using raycast, Collider Event Caster uses 3D physics triggers to "touch at" other 3D physics colliders and sends them hover events and button events.

As a result, when setting up Collider Event Caster, remember to set child colliders as a trigger.

To setup Vive Collider Event Caster in short cut, follow the tutorial document, and replace VivePointers with ViveColliders prefab in step 2.



- **Collider Event Handler**

Because Collider Event System works based on trigger message, the event receiver can only be colliders that able to send trigger message. (Box/Sphere/Capsule/Mesh colliders)

Collider Event doesn't supports built-in event handlers, only the followings:

- **IColliderEventHoverEnterHandler**
  Called when a controller enters the object.
- **IColliderEventHoverExitHandler**
  Called when a controller exits the object.
- **IColliderEventPressDownHandler**
  Called when a controller button is pressed on the object.
- **IColliderEventPressUpHandler**
  Called when a controller button is released on the object.
- **IColliderEventPressEnterHandler**
  Called when a controller enters the object with pressed button or when a controller button is pressed on the object.

- **IColliderEventPressExitHandler**
  Called when a controller exits the object with pressed button or when a controller button is released on the object.
- **IColliderEventClickHandler**
  Called when a controller is pressed and released on the same object without leaving it.
- **IColliderEventDragStartHandler**
  Called on the drag object when dragging is about to begin.
- **IColliderEventDragFixedUpdateHandler**
  Called on the drag object when a drag is happening in that fixed frame (called on the original drag start object).
- **IColliderEventDragUpdateHandler**
  Called on the drag object when a drag is happening in that frame (called on the original drag start object).
- **IColliderEventDragEndHandler**
  Called on the drag object when a drag finishes (called on the original drag start object).
- IColliderEventDropEndHandler
  Called on the object where a drag finishes.
- **IColliderEventAxisChangedHandler**
  Called when the touch pad scrolls or trigger button moves on the object.

- **Collider Event Data**

There are three kinds of event data sent to the event handler, each of them delivered with different properties and status, except eventCaster, the owner of the event data.

- **ColliderHoverEventData**
  An empty event data without any properties except eventCaster.
- **ColliderButtonEventData**
  Represents a button on a controller. You can get button status from its properties like isPressed, isDragging, pressPosition and pressRotation.
- **ColliderAxisEventData**
  Stored with scroll delta values.

Prefabs

There are some prefabs prepared for setting up scene conveniently placed at Assets/HTC.UnityPlugin/Prefabs/:

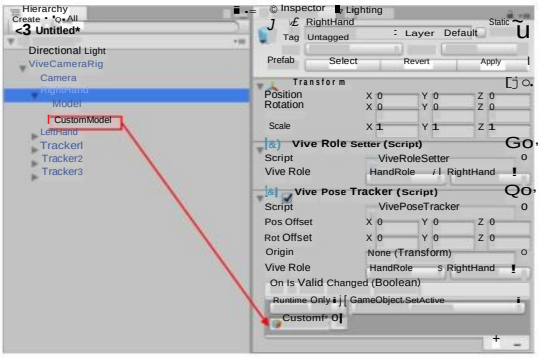
- **[Vive Input Utility] (Collection of Managers)**

This prefab include all the runtime managers used by Vive Input Utility. By adding it into the scene manually, you can override their default properties.

- **Vive CameraRig**

This prefab is like the [CameraRig] in SteamVR plugin, but manage device tracking and models in ViveRole's way. It includes a VR camera and 5 tracking devices with default render models.

You can simply replace the default model by putting your custom model under the specific device object. For example, if you want to replace RightHand model, just put your custom model under RightHand object. Additionally, you can register the model's GameObjcet.SetActive function under OnIsValidChanged event to hide the model when the device is not connected.



- **Vive Pointers**

This prefab creates two event raycasters tracked along each hands (without device model). The raycast can interact with the built-in UGUI elements or physics colliders, just like a 3D mouse pointer.

- **Vive Curve Pointers**

This prefabs creates two "curved" event raycasters (without device model).

- **Vive Colliders**

This prefab creates two grabbers tracked along each hands (without device model) that can grab physics objects with grabbable component (BasicGrabbable, StickyGrabbable).

- **Vive Rig**

This prefab is a combination of all four prefabs **Vive Camera Rig**, **Vive Pointers**, **Vive Curve Pointers**, **ViveColliders**, with a **ControllerManagerSample** script. The script is a basic demonstration of how to control all these functions, it should always be customized on demend.

Common Used API Reference

- **using HTC.UnityPlugin.Vive**
  - **static bool ViveInput.GetPressEx<TRole>(TRole role, ControllerButton button)**
    - Returns true while the button is pressed.
  - **static bool ViveInput.GetPressDownEx<TRole>(TRole role, ControllerButton button)**
    - Returns true during the frame the user starts pressing down the button.
  - **static bool ViveInput.GetPressUpEx<TRole>(TRole role, ControllerButton button)**
    - Returns true during the frame the user releases the button.
  - **static float ViveInput.LastPressDownTimeEx<TRole>(TRole role, ControllerButton button)**
    - Returns last press down unscaled time.
  - **static int ViveInput.ClickCountEx<TRole>(TRole role, ControllerButton button)**
    - Returns the button click count. Click count will increase only if the button pressed down/up duration less than **ViveInpu.clickInterval** (default is 0.3 second).
  - **static float ViveInput.GetAxisEx<TRole>(TRole role, ControllerAxis axis)**
    - Returns the value of the axis.
  - **static void ViveInput.AddListenerEx<TRole>(TRole role, ControllerButton button, ButtonEventType eventType, System.Action callback)**
    - Adds listener to handle the specific button event.
  - **static void ViveInput.RemoveListenerEx<TRole>(TRole role, ControllerButton button, ButtonEventType eventType, System.Action callback)**
    - Removes listener that handles the specific button event.
  - **static void ViveInput.TriggerHapticPulseEx<TRole>(TRole role, ushort durationMicroSec = 500)**
    - Triggers a haptic pulse. Usually called once in each frame to make a long vibration. Currently only works with Vive Controller.
  - **static HTC.UnityPlugin.Utility.RigidPose VivePose.GetPoseEx<TRole>(TRole role)**
    - Returns current position and rotation of the device identified by role.

- `static uint ViveRole.GetDeviceIndexEx<TRole>(TRole role)`
  - Returns the device index that the role is mapping to.

- `using HTC.UnityPlugin.VRModuleManagement`
  - `static bool VRModule.HasInputFocus()`
    - Returns true if input focus captured by current process. Usually the process losses focus when player switch to dashboard by clicking Steam button.
  - `static IVRModuleDeviceState VRModule.GetDeviceState(uint deviceIndex, bool usePrevious = false)`
    - Returns the readonly device state.