Vive Input Utility Developer Guide

Email: vivesoftware@htc.com

Forum: http://community.viveport.com

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

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

About

Vive Input Utility is an Unity plugin that allows developers to access Vive device status, including Vive Tracker.

We also introduce a mouse pointer solution that works in 3D space and is compatible with the Unity Event System, and a device binding system to manage multiple tracking devices.

Our goal is to accelerate Unity developers making new VR apps and discovering new VR experience by saving their time in writing redundant code managing Vive devices.

Motivation

The SteamVR plugin provides a C# interface to let Unity developers interact with Vive devices.

But getting the controller input status or device pose causes lots of redundant code:

- You must continuously get the correct device index, which is determined by SteamVR_ControllerManager whenever a controller is connected.
- Locating SteamVR_ControllerManager also causes more effort.

So our main goal is to provide handy interface and reduce the redundancy.

Main Features

- Using static function to retrieve device input:
  - Button's event
  - Tracking pose.
- Using ViveRaycaster component to achieve 3D-space-pointer that compatible with Unity Event System.

Static Interface

- Get button's event

Instead of finding device through **SteamVR** scripts...

```
using UnityEngine;
using Valve.VR;

public class GetPressDown_SteamVR : MonoBehaviour
{
    public SteamVR_ControllerManager manager;
    private void Update()
    {
        // get trigger down
        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))
        {
            // ...
        }
    }
}
```

Class **ViveInput** under **HTC.UnityPlugin.Vive** provides a simpler API to achieve that.

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

public class GetPressDown_ViveInput : MonoBehaviour
{
    private void Update()
    {
        // get trigger down
        if (ViveInput.GetPressDownEx(HandRole.RightHand, ControllerButton.Trigger))
        {
            // ...
        }
    }
}
```

Getting axis values as well.

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

public class GetAxisValue_ViveInput : MonoBehaviour
{
    private void Update()
    {
        // get trigger axis value
        if (ViveInput.GetAxisEx(HandRole.RightHand, ControllerAxis.Trigger) > 0.5f)
        {
            // ...
        }
    }
}
```

- Listen to button's event

ViveInput also provides 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()
    {
        // ...
    }
}
```

- Get tracking pose

Class **VivePose** also under **HTC.UnityPlugin.Vive** provides an API to get a device pose.

Its return type is **RigidPose** under **HTC.UnityPlugin.Utility**.

```
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);
        // set transform to the mid point between them
        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);
        }
    }
}
```

- Easily identify devices by defined roles

Currently there are **HandRole**, **TrackerRole**, **BodyRole** defined under **HTC.UnityPlugin.Vive**.

  - **HandRole**: Mostly mapping to controllers with buttons, except ExternalCamera is mapping to third found controller or first found tracker. (RightHand, LeftHand, ExternalCamera, Controller4~15)
  - **TrackerRole**: Mapping to generic trackers such as **Vive Tracker**. (Tracker1~13)
  - **BodyRole**: Mapping to controllers and trackers depends on their position related to player's head, body and limbs. (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…

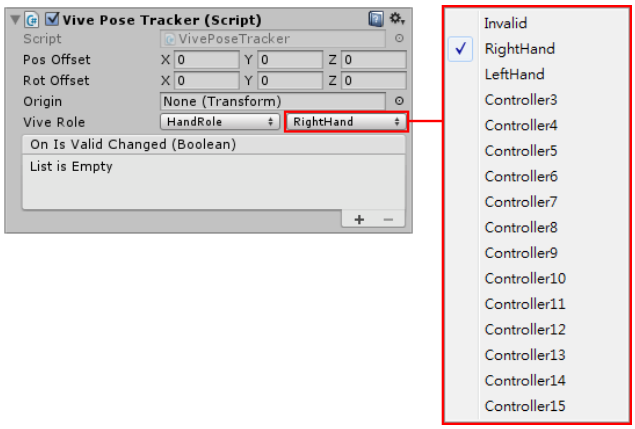
Notice that class **ViveRole** under **HTC.UnityPlugin.Vive** is responsible for mapping role to device index, and **VRModule.GetDeviceIndex** requires device index as its argument, not role enum.

```
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.

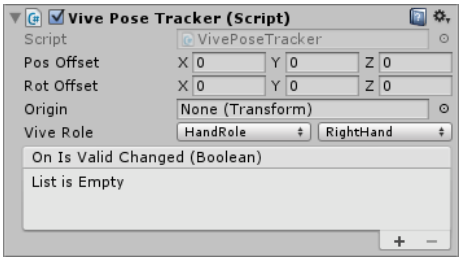


- Pose Modifier

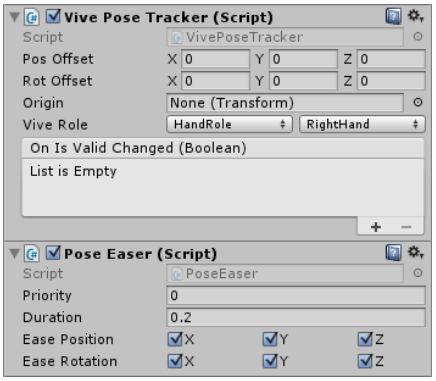
It is a tracking effect script applied to a pose tracker.

Implement abstract class PoseTracker.BasePoseModifier to write custom tracking effect.

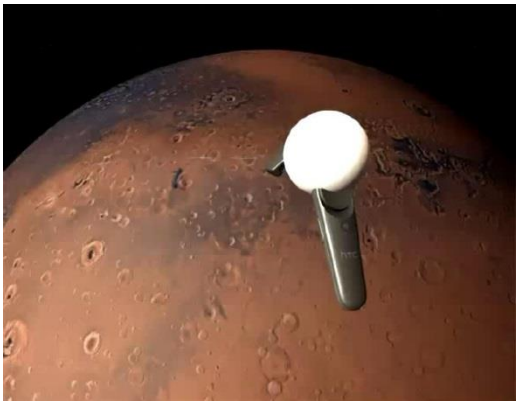
Without pose modifier



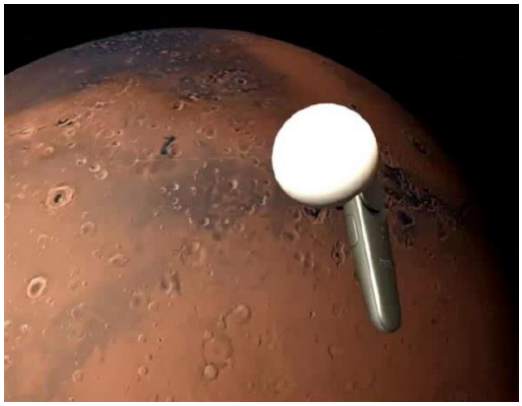
With pose modifier



https://vimeo.com/171724218



https://vimeo.com/171724270



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

- **Pose Stabilizer**

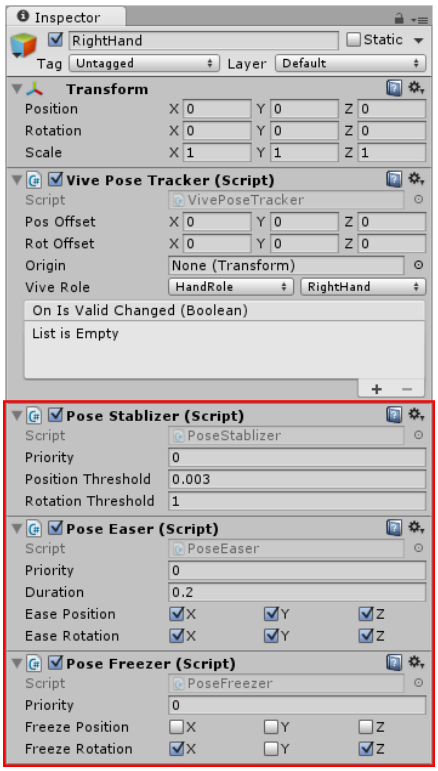
  If controller moves within the threshold, the object stays. Otherwise, the object keeps the offset and follows.

- **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.

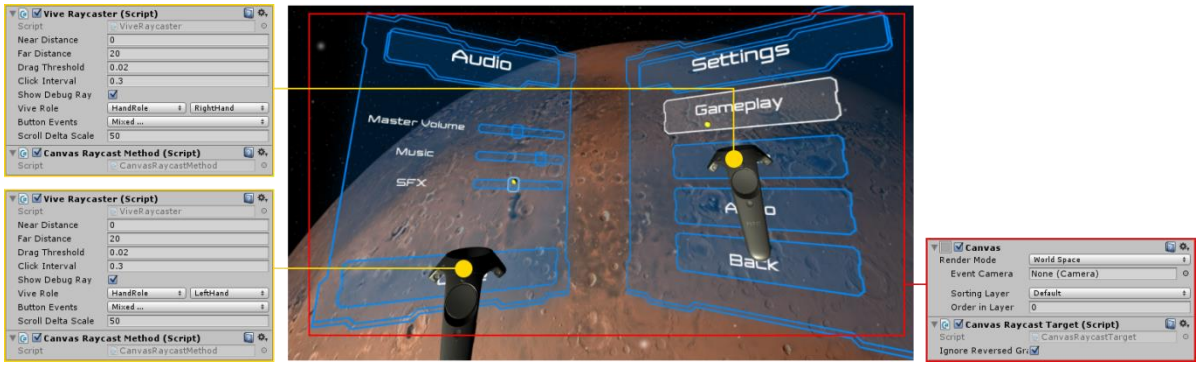
Vive Raycaster is an event raycaster script that sends Vive button's event from its transform.

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.

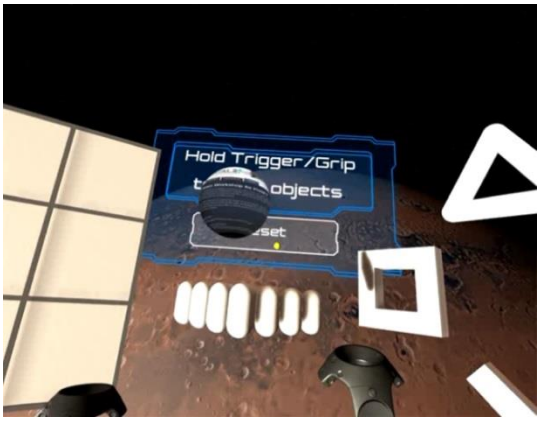
Raycast Method	Against Type
Physics Raycast Method	Collider
Physics 2D Raycast Method	Collider2D
Graphic Raycast Method	Graphic in target Canvas
Canvas Raycast Method	Graphic in all Canvas with CanvasRaycastTarget component

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

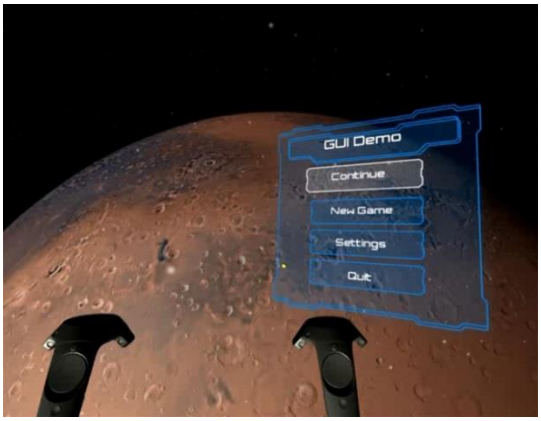


More examples:

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.

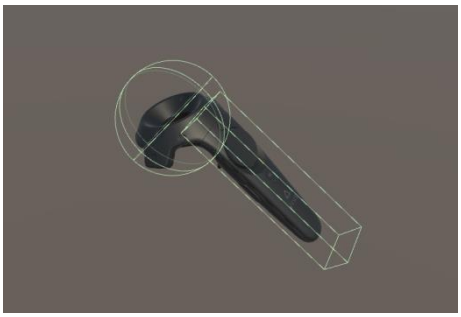
- Add a component that implements event handler interfaces that are supported by the built-in Input Module on an object. See https://docs.unity3d.com/Manual/SupportedEvents.html to learn more about built-in event handlers.
- Add event receiver (Collider/Collider2D/Graphic) to the object or child of the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

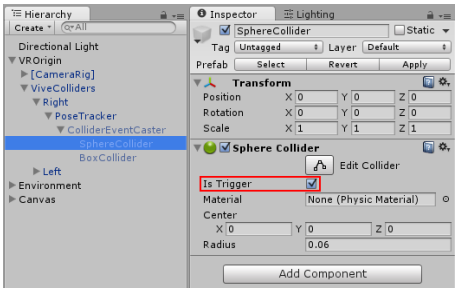
But the Collider Event Caster isn't driven by any input module, so they can work together at the same time if needed.



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/:

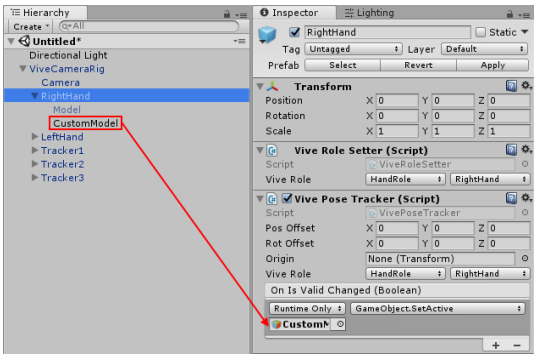
- **[Vive Input Utility] (Collection of Managers)**

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 Camera Rig**

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.