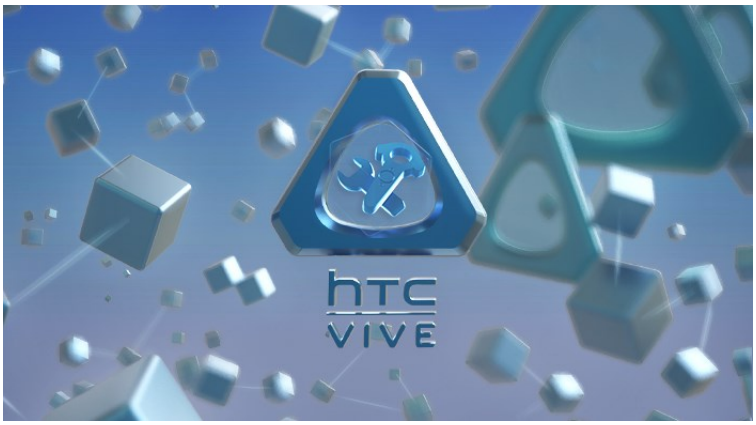


Vive Input Utility Developer Guide

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
        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))
        {
            // ...
        }
    }
}
```

Static class ViveInput provides a simpler API to achieve that.

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

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

- 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.AddPressDown(HandRole.LeftHand, ControllerButton.Trigger, OnTrigger);
    }

    private void OnDestroy()
    {
        ViveInput.RemovePressDown(HandRole.LeftHand, ControllerButton.Trigger, OnTrigger);
    }

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

- Get tracking pose

Static class VivePose provides an API to get a device pose.

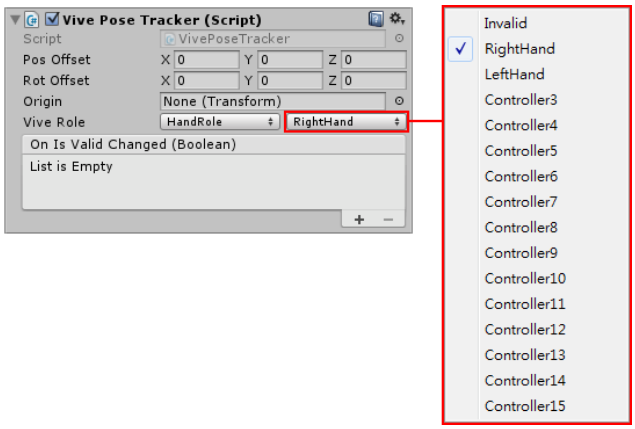
It is easy to identify device role by static class DeviceRole.

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

public class GetPose_VivePose : MonoBehaviour
{
    public Transform deviceOrigin;

    public void ToMidPoint()
    {
        Pose rightHandPose = VivePose.GetPose(HandRole.RightHand);
        Pose leftHandPose = VivePose.GetPose(HandRole.LeftHand);

        Pose.SetPose(transform, Pose.Lerp(rightHandPose, leftHandPose, 0.5f), deviceOrigin);
    }
}
```

Helper Components

- Vive Pose Tracker

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

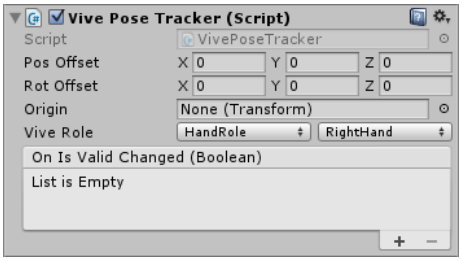


- Pose Modifier

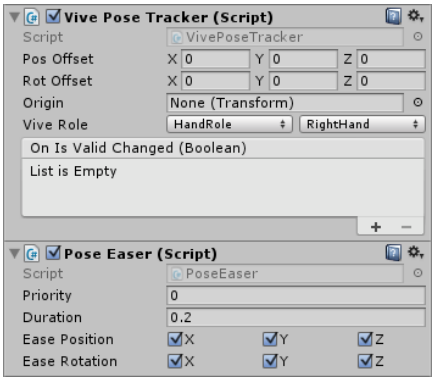
It is a tracking effect script applied to a pose tracker.

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

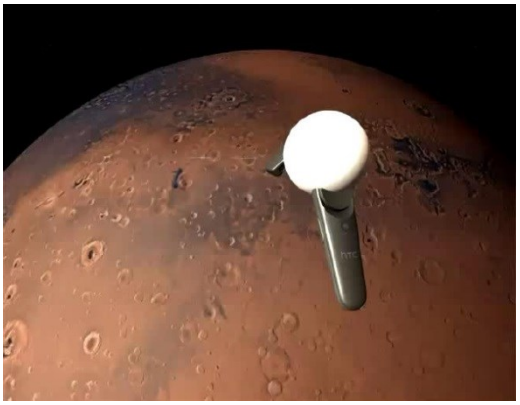
Without pose modifier



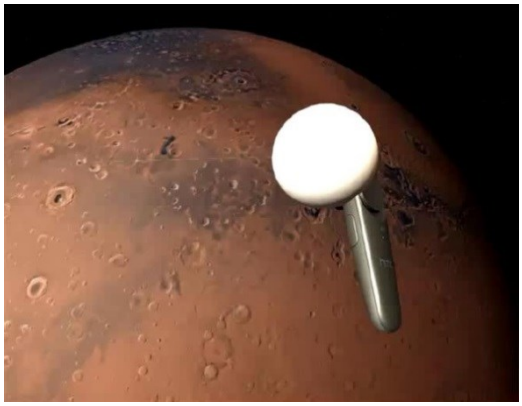
With pose modifier



https://vimeo.com/171724218



https://vimeo.com/171724270



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

- **Pose Stabilizer**

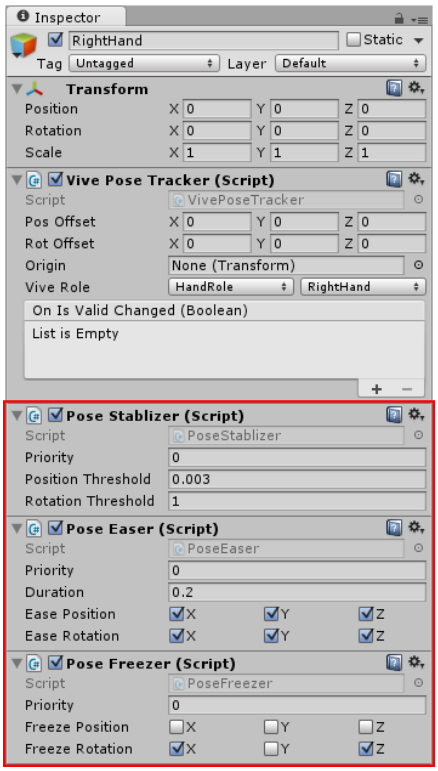
  If controller moves within the threshold, the object stays. Otherwise, the object keeps the offset and follows.

- **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.

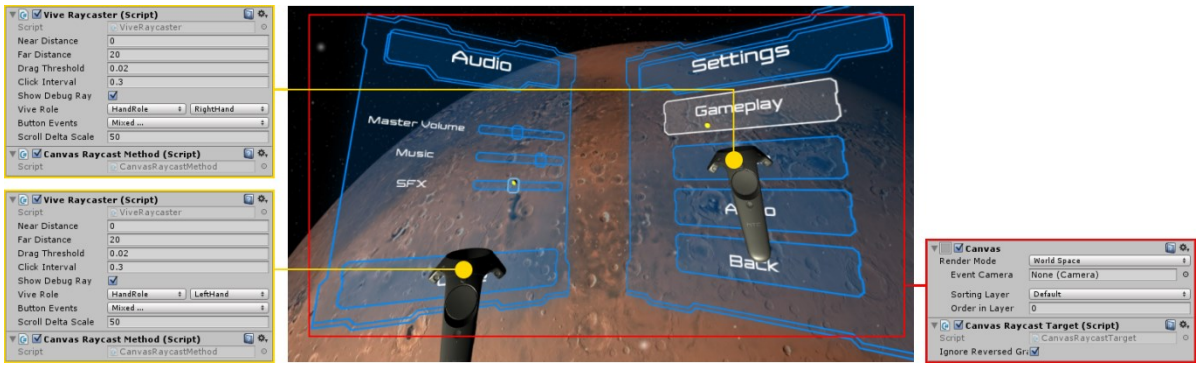
Vive Raycaster is an event raycaster script that sends Vive button's event from its transform.

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.

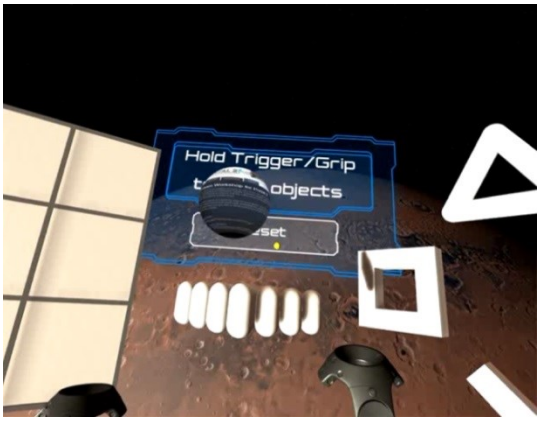
Raycast Method	Against Type
Physics Raycast Method	Collider
Physics 2D Raycast Method	Collider2D
Graphic Raycast Method	Graphic in target Canvas
Canvas Raycast Method	Graphic in all Canvas with CanvasRaycastTarget component

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

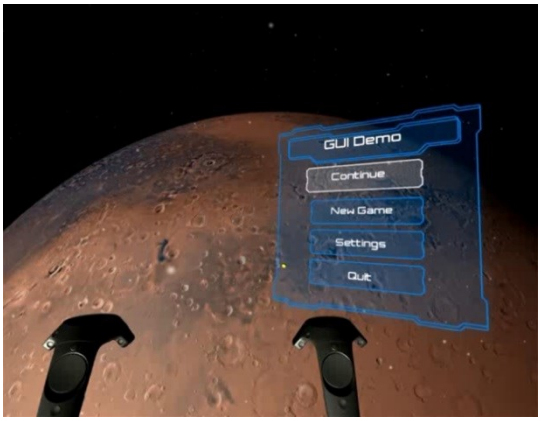


More examples:

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.

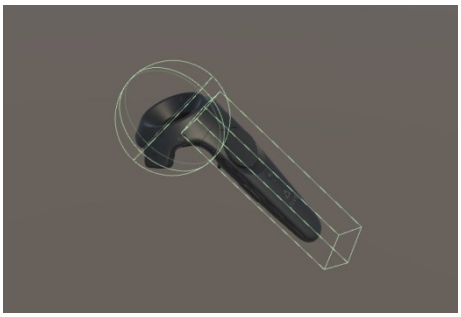
- Add a component that implements event handler interfaces that are supported by the built-in Input Module on an object. See https://docs.unity3d.com/Manual/SupportedEvents.html to learn more about built-in event handlers.
- Add event receiver (Collider/Collider2D/Graphic) to the object or child of the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

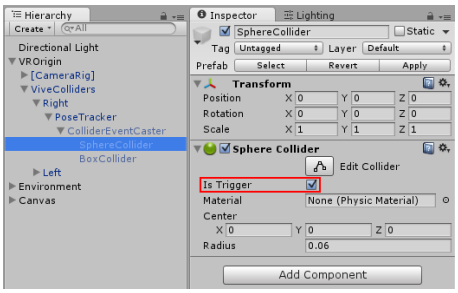
But the Collider Event Caster isn't driven by any input module, so they can work together at the same time if needed.



Instead of using raycast, Collider Event Caster uses 3D physics triggers to "point at" other 3D physics colliders and sends them 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/:

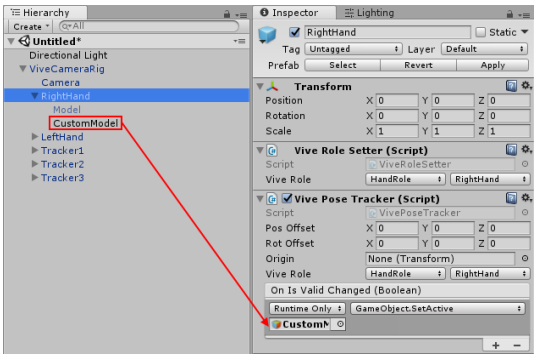
- **[Vive Input Utility] (Collection of Managers)**

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.

API Reference

- uint ViveRole.GetDeviceIndex(DeviceRole role)

  Returns device index of the device identified by the role. Returns OpenVR.k_unTrackedDeviceIndexInvalid if the role doesn't assign to any device.

- bool VivePose.HasFocus()

  Returns true if input focus captured by current process. Usually the process losses focus when player switch to deshboard by clicking Steam button.

- bool VivePose.IsConnected(DeviceRole role)

  Returns true if the device identified by role is connected.

- bool VivePose.HasTracking(DeviceRole role)

  Returns true if tracking data of the device identified by role has valid value.

- Pose VivePose.GetPose(DeviceRole role, Transform origin = null)

  Returns tracking pose of the device identified by role.

- void VivePose.SetPose(Transform target, DeviceRole role, Transform origin = null)

  Set target pose to tracking pose of the device identified by role relative to the origin.

- bool ViveInput.GetPress(HandRole role, ControllerButton button)

  Returns true while the button on the controller identified by role is held down.

- bool ViveInput.GetPressDown(HandRole role, ControllerButton button)

  Returns true during the frame the user pressed down the button on the controller identified by role.

- bool ViveInput.GetPressUp(HandRole role, ControllerButton button)

  Returns true during the frame the user releases the button on the role.

- float ViveInput.GetAxis(HandRole role, ControllerAxis axis)

  Returns raw analog value of the controller identified by role and axis enum.

- int ViveInput.ClickCount(HandRole role, ControllerButton button)

  Return amount of clicks in a row for the button on the controller identified by role. Set ViveInput.clickInterval to configure click interval.

- float ViveInput.LastPressDownTime(HandRole role, ControllerButton button)

  Returns time of the last frame that user pressed down the button on the controller identified by role.

- void ViveInput.AddPress(HandRole role, ControllerButton button, Action callback)
  Add press handler for the button on the controller identified by role.
- void ViveInput.AddPressDown(HandRole role, ControllerButton button, Action callback)
  Add press down handler for the button on the controller identified by role.
- void ViveInput.AddPressUp(HandRole role, ControllerButton button, Action callback)
  Add press up handler for the button on the controller identified by role.
- void ViveInput.AddClick(HandRole role, ControllerButton button, Action callback)
  Add click handler for the button on the controller identified by role. Use ViveInput.ClickCount to get click count.
- void ViveInput.RemovePress(HandRole role, ControllerButton button, Action callback)
  Remove press handler for the button on the controller identified by role.
- void ViveInput.RemovePressDown(HandRole role, ControllerButton button, Action callback)
  Remove press down handler for the button on the controller identified by role.
- void ViveInput.RemovePressUp(HandRole role, ControllerButton button, Action callback)
  Remove press up handler for the button on the controller identified by role.
- void ViveInput.RemoveClick(HandRole role, ControllerButton button, Action callback)
  Remove click handler for the button on the controller identified by role.
- void ViveInput.TriggerHapticPulse(HandRole role, ushort intensity = 500)
  Trigger vibration of the controller identified by role.