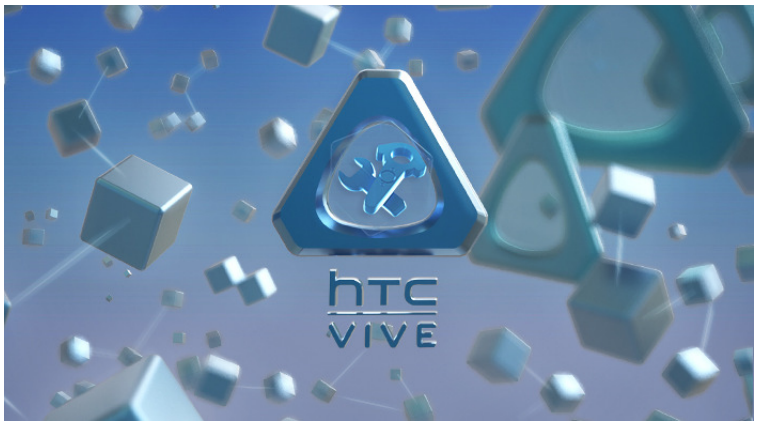


Vive Input Utility Developer Guide

vivesoftware@htc.com

Abstract

Vive Input Utility is a tool based on the SteamVR plugin that allows developers to access Vive device status in handy way.

We also introduce a mouse pointer solution that works in 3D space and is compatible with the Unity Event System.

By importing this utility, developers can save lots of time in writing redundant code to manage 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...

```
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.

```
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.

```
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

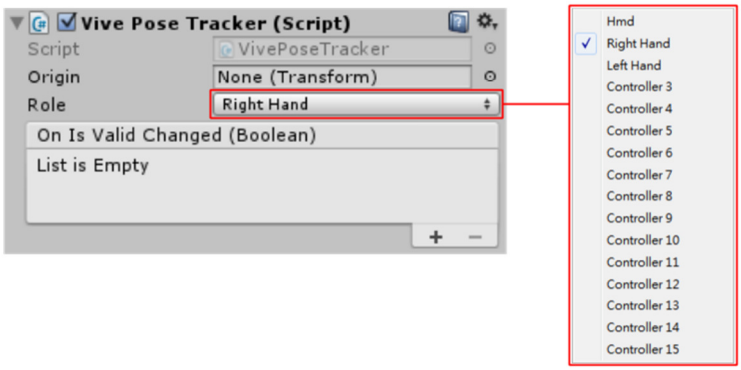
Static class VivePose provides an API to get a device pose.

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

public class GetPoseExample : MonoBehaviour
{
    public Transform deviceOrigin;

    public void SetMidPoint()
    {
        Pose rightHandPose = VivePose.GetPose(DeviceRole.RightHand);
        Pose leftHandPose = VivePose.GetPose(DeviceRole.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.

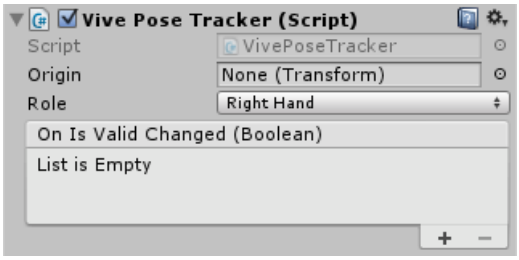


- Pose Modifier

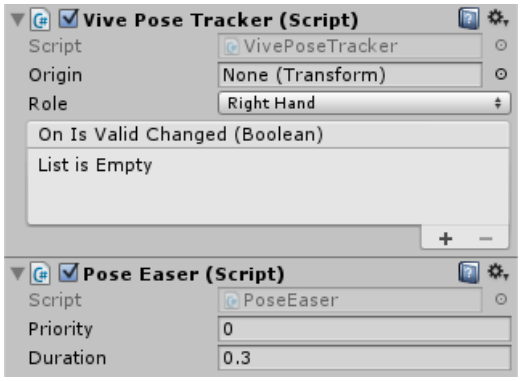
It is a tracking effect script applied to a pose tracker.

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

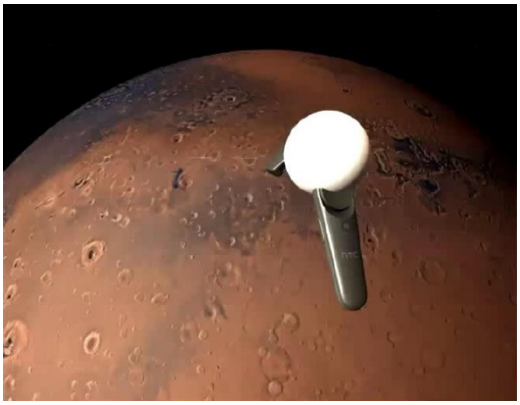
Without pose modifier



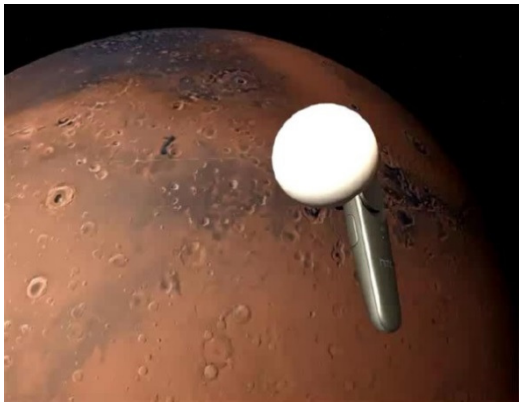
With pose modifier



https://vimeo.com/171724218



https://vimeo.com/171724270



- Vive Raycaster & Raycast Method

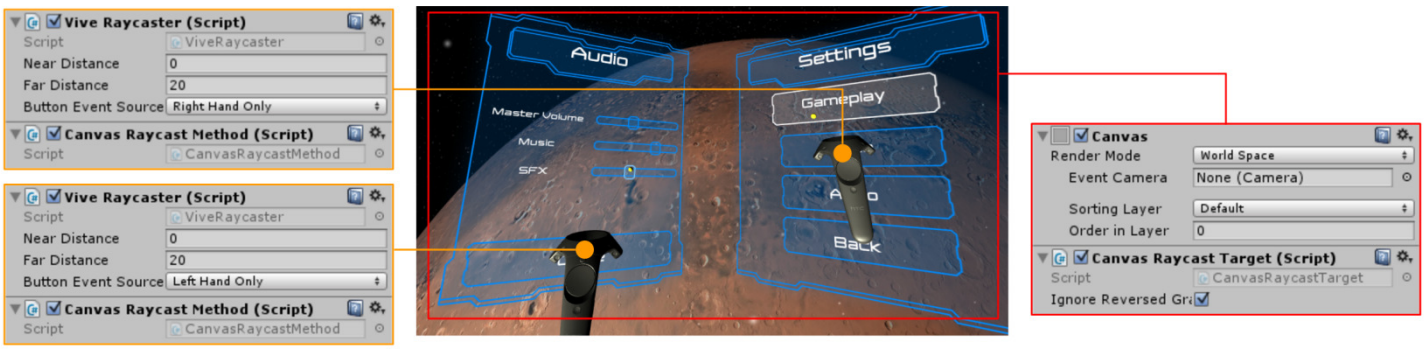
Vive Raycaster is an event raycaster script that sends Vive button's event from its transform.

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.

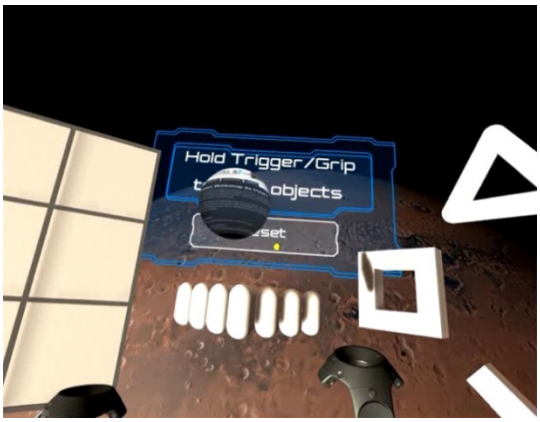
Raycast Method	Against Type
Physics Raycast Method	Collider
Physics 2D Raycast Method	Collider2D
Graphic Raycast Method	Graphic in target Canvas
Canvas Raycast Method	Graphic in all Canvas Raycast Target

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

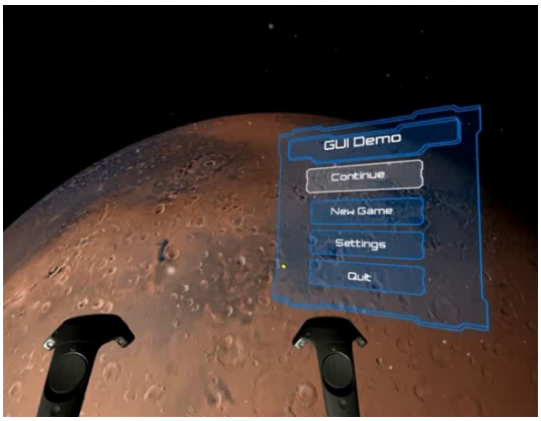


More examples:

https://vimeo.com/169824408



https://vimeo.com/169824438



- Event System Handler

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

Add a component that implements an event handler interface (derives from IEventSystemHandler) on an object.

Add raycast-receiver (Collider/Collider2D/Graphic) to the object or child of the object.

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

public class MyButtonEventHandler : 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)
    {
        var viveEventData = eventData as VivePointerEventData;
        if (viveEventData != null)
        {
            if (viveEventData.viveButton == ControllerButton.Trigger)
            {
                // Vive button triggered!
            }
        }
        else if (eventData != null)
        {
            if (eventData.button == PointerEventData.InputButton.Left)
            {
                // Standalone button triggered!
            }
        }
    }
}
```

API Reference

uint ViveRole.GetDeviceIndex(DeviceRole role)

Returns device index assigned to the role. If target role is not assigned or disconnected, returns OpenVR.k_unTrackedDeviceIndexInvalid.

bool VivePose.HasFocus()

Returns true if input focus captured by current process.

bool VivePose.IsConnected(DeviceRole role)

Returns true if device of role is connected.

bool VivePose.HasTracking(DeviceRole role)

Returns true if device tracking data of role is valid.

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

Returns device tracking data of role relative to origin.

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

Set device tracking data of role into target relative to origin.

bool ViveInput.GetPress(HandRole role, ControllerButton button)

Get button press state.

bool ViveInput.GetPressDown(HandRole role, ControllerButton button)

Get button press down state.

bool ViveInput.GetPressUp(HandRole role, ControllerButton button)

Get button press up state.

float ViveInput.GetTriggerValue(HandRole role)

Get trigger button pressed value.

Vector2 ViveInput.GetPadAxis(HandRole role)

Get track pad touched axis.

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

Get button clicked count. Set ViveInput.clickInterval to configure click interval.

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

Get button last press down frame time.

`void ViveInput.AddPress(HandRole role, ControllerButton button, Action callback)`

Add press handler for button. Triggered every frame if button is pressed.

`void ViveInput.AddPressDown(HandRole role, ControllerButton button, Action callback)`

Add press down handler for button.

`void ViveInput.AddPressUp(HandRole role, ControllerButton button, Action callback)`

Add press up handler for button.

`void ViveInput.AddClick(HandRole role, ControllerButton button, Action callback)`

Add click handler for button. Use ViveInput.ClickCount to get click count.

`void ViveInput.RemovePress(HandRole role, ControllerButton button, Action callback)`

Remove press handler for button.

`void ViveInput.RemovePressDown(HandRole role, ControllerButton button, Action callback)`

Remove press down handler for button.

`void ViveInput.RemovePressUp(HandRole role, ControllerButton button, Action callback)`

Remove press up handler for button.

`void ViveInput.RemoveClick(HandRole role, ControllerButton button, Action callback)`

Remove click handler for button.

`void ViveInput.TriggerHapticPulse(HandRole role, ushort intensity = 500)`

Trigger controller vibration.