InstantVR Edge Support

Version: 3.5
Date: July 29, 2016

Features

The Edge version of InstantVR supports the following input:

- keyboard/mouse: using standard WASD input and mouse look functions
- game controllers: support for the followin game controllers is builtin: Xbox, PS4, Steelseries XI, GameSmart and Sweex GA100 (yes it is obscure, but I happen to have one)
- Oculus Rift DK2, CV1
- HTC Vive / SteamVR controllers
- Samsung Gear VR
- Google Cardboard for Android
- Razer Hydra
- Microsoft Kinect 2: 6 point body tracking is supported with inverse kinematics to complete the body movements.
- Leap Motion
- VicoVR on Android

The following networking options are supported in InstantVR Edge:

- Unity Networking
- Photon/PUN (Photon Server is not yet supported)

For code-free programming, InstantVR also support PlayMaker with dedicated actionscripts.

Prerequisites

- The current distribution only supports development on Windows and Android.
- Unity 5.3.3p1 or higher (Unity 5.3.5 or higher is required for Rift CV1)
- Oculus Runtime for Windows version 0.7.0.0 or higher (1.3 or higher for Rift CV1) is required. The runtime can be downloaded from the Oculus VR site.
- The Kinect for Windows SDK v2.0 or higher is required for Microsoft Kinect 2 support. It can be downloaded from the Microsoft Download Center.

- Leap Motion software 2.3.1 or higher is required for Leap Motion support. Leap Motion Orion is supported and strongly recommended. It can be downloaded from the Leap Motion setup page.
- Optionally, PlayMaker 1.7.8 or higher is supported

Getting started

To use InstantVR you need an environment first. The minimum is a flat terrain with a directional light, but you can make it as complex as you want with lots of meshes, rigidbodies and colliders. You are only bound by the limits of Unity and the computer used to drive the game.

You should not include a camera of other first or third person objects in your game. This is fully handled by InstantVR.

To complete the scene you should drag one of the prefabs named 'MH_…' from the folder InstantVR into your scene.

The MH_VR prefabs support Oculus Rift, HTC Vive and Gear VR with various tracking devices. Note that MH_VRSteamVR is needed to use HTC Vive in combination with the SteamVR controllers.

The MH_Cardboard supports Google Cardboard, optionally with VicoVR.

The MH_Basic does not include support for any tracking input.

MH_ThirdPerson is included for networking purposes.

Video

An introductional tutorial can be found here: https://www.youtube.com/watch?v=ObtPdsUx9sg

General configuration

InstantVR

The instantVR contains references to the 6 transforms which move the avatar. These transforms can be placed anywhere within the Hierarchy.

IVR_Body Movements

This script translates the movements of the targets to the actual body positions.

It has the following options:

- Enable torso: enables body movements of the upper body
- Enable left: enables leg movements

Extensions

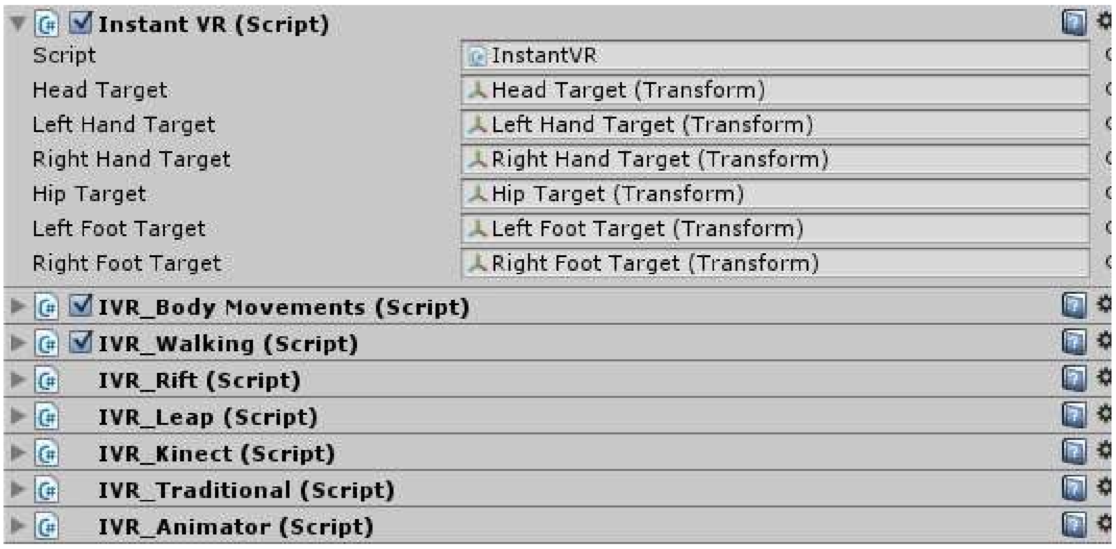
A number of extensions is included which can be added to the GameObject with InstantVR which adds support for one or more input devices or other target controllers.

Extensions can be added to every GameObject with an InstantVR script attached to extend the tracking functionality. Extensions will typically implement support for specific input devices like the Oculus Rift or Microsoft Kinect, but may also include extensions for animations or networking.

Multiple extensions can be added to a single InstantVR gameObject. The extensions which will be used at play time depends on:

- availability: is the hardware associated with the extension present?
- priority: the extensions with the highest priority are chosen over lower priority extensions.
- tracking: is the hardware currently tracking?

In the example below, we have added 5 extensions: Oculus Rift, Leap Motion, Kinect 2, Traditional (Game controller/mouse/keyboard) and Animator.



Instant VR (Script)

Script	InstantVR
Head Target	Head Target (Transform)
Left Hand Target	Left Hand Target (Transform)
Right Hand Target	Right Hand Target (Transform)
Hip Target	Hip Target (Transform)
Left Foot Target	Left Foot Target (Transform)
Right Foot Target	Right Foot Target (Transform)

IVR_Body Movements (Script)
IVR_Walking (Script)
IVR_Rift (Script)
IVR_Leap (Script)
IVR_Kinect (Script)
IVR_Traditional (Script)
IVR_Animator (Script)

Dynamic behaviour

During gameplay with all supported hardware available the behaviour is as follows:

- Oculus Rift will always be used for head tracking. It has the highest priority and is always tracking.
- Leap Motion is used for tracking the hands when the hands are in the field of view of the Leap Motion cameras. When the hands are outside the view they cannot be tracked by the Leap Motion so it will not be used then. Hand tracking will drop down to the next hand tracker in the hierarchy
- Kinect 2 supports tracking of all 6 tracking targets, but in this case it will not be used for head tracking, because the Oculus Rift has a higher priority. It will

be used for hand tracking when the hands are outside the view of the Leap Motion camera. All other body parts are tracked by Kinect 2.

- Traditional input can be used for walking around

- The animator is only used with target are outside the tracking area of the Kinect.

When the game is played with just the Oculus Rift available, it will behave like this:

- Oculus Rift will always be used for head tracking. It has the highest priority and is always tracking.

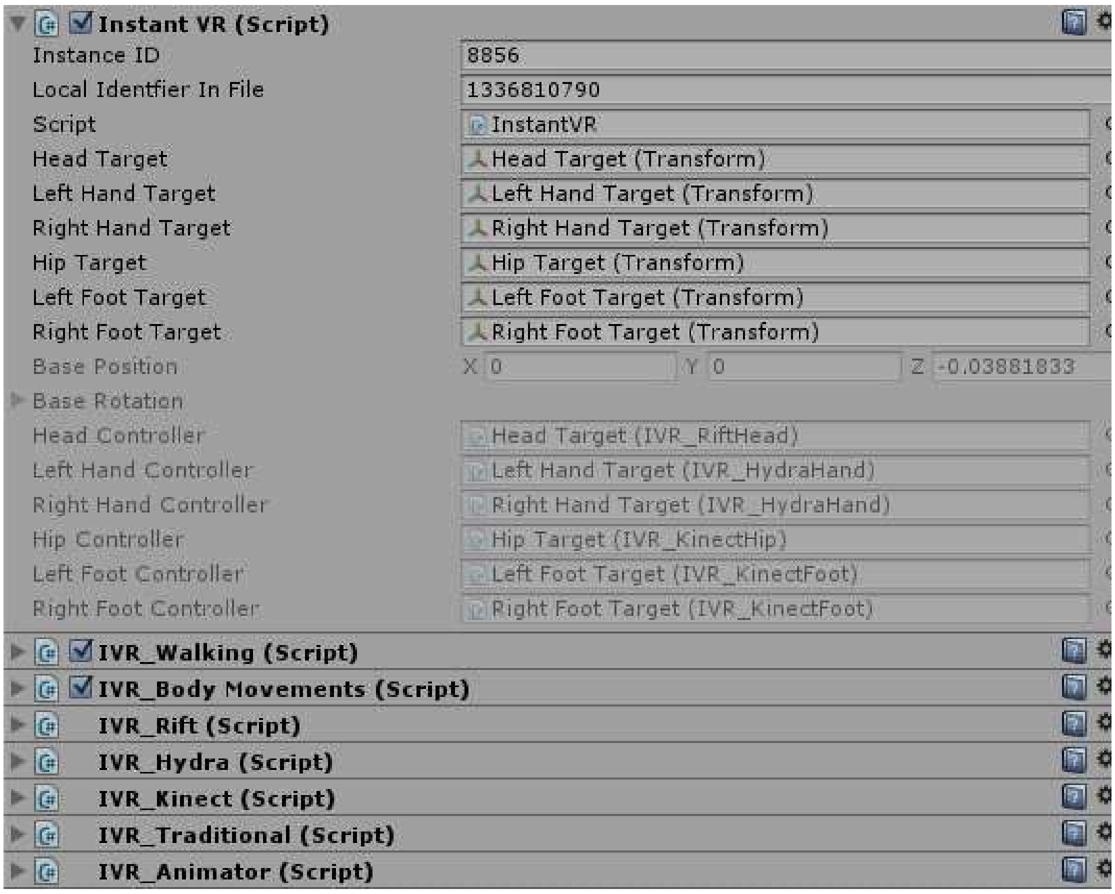
- Leap Motion and Kinect 2 are not available, so will not be used

- Traditional input can be used for walking around

- The animator is used to move all targets except the head, which is tracked by the Oculus Rift. This results in leg and arm movements during walking and rotation movements.

Traditional and Animator are typically used when no or limited VR hardware are available and it is good practice to have them as the lowest 2 priority spots as it is shown here.

During play mode in the editor it is possible to view which extensions are currently used for each target by switching to debug view:



Instant VR (Script)

Instance ID	8856
Local Identfier In File	1336810790
Script	InstantVR
Head Target	Head Target (Transform)
Left Hand Target	Left Hand Target (Transform)
Right Hand Target	Right Hand Target (Transform)
Hip Target	Hip Target (Transform)
Left Foot Target	Left Foot Target (Transform)
Right Foot Target	Right Foot Target (Transform)
Base Position	X 0 Y 0 Z -0.03881833
Base Rotation	
Head Controller	Head Target (IVR_RiftHead)
Left Hand Controller	Left Hand Target (IVR_HydraHand)
Right Hand Controller	Right Hand Target (IVR_HydraHand)
Hip Controller	Hip Target (IVR_KinectHip)
Left Foot Controller	Left Foot Target (IVR_KinectFoot)
Right Foot Controller	Right Foot Target (IVR_KinectFoot)

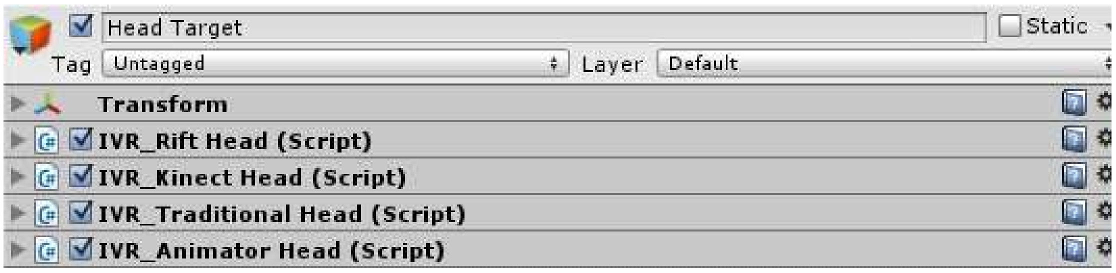
IVR_Walking (Script)
IVR_Body Movements (Script)
IVR_Rift (Script)
IVR_Hydra (Script)
IVR_Kinect (Script)
IVR_Traditional (Script)
IVR_Animator (Script)

In this case we have the extensions for Oculus Rift, Razer Hydra, Microsoft Kinect, the traditional and animator extensions configured. In the debug view of InstantVR you can see the various controllers used: Rift for the Head Target, Hydra for the Hand Targets and Kinect for the Hip and Feet Targets.

Controllers

Every extension implements one or more controllers for targets. The Oculus Rift extension has one controller: the Rift Head Controller which is used solely for the head target. The Leap Motion and Razer Hydra extensions just have a Hand Controller which is used for the left and right hands, while the Kinect extensions have controllers for every target as they can track the full body.

If we look into the Targets themselves we can see the controllers currently associated with that target. Here we have an example of the Head Target:


Head Target
Static
Tag Untagged
Layer Default
Transform
IVR_Rift Head (Script)
IVR_Kinect Head (Script)
IVR_Traditional Head (Script)
IVR_Animator Head (Script)

You can see the Rift and Kinect Head Controllers. Like the priority for extensions the order in which the controllers appear determines the priority. This order is in fact controlled by the ordering of the extensions. So you need to reorder the extension when you want to change the priority of the Head Controllers. Direct changes in the order of controllers will be undone automatically.

Note: the controller priorities and positions are not updated when the inspector is in Debug mode. The priorities and ordering will be corrected when the inspector is switched back to Normal mode.

It is possible to disable specific controllers. For instance: if you do not want to use Kinect Head tracking when the Rift is not available, but you do want to use Kinect for the rest of the body, you can disable the Kinect Head Controller. This is done by disabling the IVR_Kinect Head by unchecking the script in the example above. If you want to disable all controllers of an extension, it is better to remove the extension altogether.

Calibration

Certain extensions need calibration to work correctly. Examples are the Oculus Rift, Microsoft Kinect and Razer Hydra.

The calibration information can be configured in the extensions, but it is also possible to calibrate during gameplay.

Calibration configuration is done by setting the Tracker Position in the extensions. The tracker position is the position of the tracker relative to the player's position. For instance if the tracker is 1 meter in front of you at a height of 1.8 meter from the ground, 20cm to the left, the Tracker Position should be set to x = 0, y = 1.8, z = -0.2

The following extensions can have their tracker positions configured:

Oculus Rift: the position of the IR camera

Microsoft Kinect: the position of the Kinect sensor bar

Razer Hydra: the position of the Base station

The Kinect Tracker Position will be determined automatically when it is used in combination with the Oculus Rift and the Rift has a higher priority.

The calibration can also be done manually during gameplay using the calibration keys implemented the various input controllers:

Pressing Tab on the keyboard

Pressing Back and Start simultaneously on an Xbox controller

Pressing both option buttons on the Hydra

IVR_UnityVR (Rift, HTC Vive, Gear VR)

Adds support for the Oculus Rift DK2/CV1 or HTC Vive on Windows and Gear VR on Android.

Important: the option 'Virtual Reality Supported' needs to be checked in the Edit menu, Project Settings, Player.

Head Target
Rotational tracking is supported for the Oculus Rift, HTC Vive and the Gear VR. Positional tracking is only supported for the Oculus Rift DK2/CV1 and HTC Vive. Note that positional tracking is only possible when the headset is in range of the Oculus IR camera or Lighthouse.

IVR_Cardboard

Google Cardboard enables a wide range of mobile phones to be used as a head mounted display. Cardboard is only supported on Android.

Important: the option 'Virtual Reality Supported' needs to be unchecked in the Edit menu, Project Settings, Player. This is in contrast to the IVR_Rift setting.

Head Target

Rotational tracking is supported for all mobile phones compatible with Google Cardboard.

IVR_Hydra

Adds support for the Razer Hydra.

Hand Targets

Full positional and rotational tracking of hand targets is supported. Note however that the range of the trackers is quite limited and that the tracking will get significantly less accurate at a distance of 1 meter from the base station.

The targets tracking can be recalibrated by pressing the start button on both Hydra controllers simultaneously.

Controller Buttons

The Hydra controller buttons are supported as game controller input. The buttons are mapped as follows:

Joystick	ControllerInputSide.horizontal & .vertical, .up, .down, .left & .right
Joystick press	ControllerInputSide.stickButton
Trigger	ControllerInputSide.trigger
Bumper	ControllerInputSide.bumper
Button 1	ControllerInputSide.button[0]
Button 2	ControllerInputSide.button[1]
Button 3	ControllerInputSide.button[2]
Button 4	ControllerInputSide.button[3]
Option	ControllerInputSide.option

For more information see IVR_Traditional.

IVR_Kinect2

Adds support for the Microsoft Kinect 2 or ONE. Kinect is only supported on Windows.

Head Target

Only positional tracking is enabled by default. Rotational tracking can be enabled by checking ‘Head Rotation’ in the IVR_Kinect 2 Head script on the Head Target.

Hand Targets

Positional and limited rotation tracking is supported. Due to the limitation of the

Kinect, only the X and Y rotation axis are supported.

Hip Target
Only positional tracking is supported.

Foot Targets
Only positional tracking is possible. When Kinect is enabled on the feet, the walking animation is disabled. This is noticeable when using thumb stick walking.

Hand Movements
Hand movement input is limited to closing, opening the whole hand and the ‘lasso’ position which is pointing with index and middle finger. Note that this is not completely stable so you may be dropping objects accidentally. More elaborate hand movements can be achieved by adding Leap Motion tracking.

IVR_Leap

Adds support for the Leap Motion.

Hand Target
Hands are fully tracked (positional and rotational) while the hands are in the tracking range of the Leap Motion.

Hand Movements
Full hand movements are supported with bending values for each finger individually while the hands are in the tracking range of the Leap Motion. Reliable grabbing of objects can only be achieved using the Leap Motion Orion runtime.

IVR_Traditional (Mouse, Keyboard, Game Controllers)

Adds support for game controllers and mouse/keyboard.

To support Game controllers, you need to update the InputManager Settings. The package does contain an archive called ‘GameControllerInputSettings.zip’ which contains universal InputManager settings for a the game controllers. Move this to the ProjectSettings folder to get maximum support for your controller.

Game Controller Input
InstantVR natively supports a growing number of game controllers. The following controllers are currently supported:

- Microsoft Xbox controller (360, One)

- Playstation4 controller
- Steelseries XL controller
- GameSmart controllers (e.g. MadCatz C.T.R.L.R.)
- Sweex GA100

Additionally, a number of hand trackers are also supported like game controllers:

- Razer Hydra
- SteamVR Controllers (HTC Vive)

Controller Input Sides
Controllers are split in a left and right side which support the same buttons. On each side the following buttons are supported:

- Thumbstick horizontal and vertical (float values)
- Thumbstick button press (boolean)
- Up, down, left & right (boolean)
- Buttons 0..3 (boolean)
- Bumper (float value)
- Trigger (float value)
- Option (boolean)

For each game controller most buttons can be mapped to these buttons.

Multiple Controllers
Currently, the game controller input is limited to one game controller. All available game controllers will be mapped to same input.

Scripting
The game controller input can be retrieved using:

```
Controllers.GetController(0)
```

IVR_Animator

The animator implements procedural animations for legs and arms. It is typically used as the last extension in the list of extensions as a fallback when these are not driven by a input device

Hand Targets

Follow head: when enabled, the hands will follow the X/Z position of the hips. Needs to be enabled for physical walking using the Rift and when using the IVR_Walking script

Arm swing: enables the arm swing animationg while walking

Hip Target

Follow head: when enabled, the hip will follow the X/Z position of the head. Needs to be enabled for physical walking using the Rift.

Rotation method determins how the rotation along the Y axis is determined:

- NoRotation: speaks for itself
- LookRotation: the body rotates to the direction in which the player is looking. Good option when using the Oculus Rift.
- HandOrientation: is not supported in the free version. It will behave the same as LookRotation
- Auto: chooses HandOrientation when Hydra is used and LookRotation otherwise

Walking

Physical walking is supported in combination with the Oculus Rift DK2, HTC Vive, Microsoft Kinect or VicoVR.

Interactive Hands

InstantVR Edge supports interactive hands which means that the hand and the objects thet grab collide with the environment instead of going through static objects. For this to work the IVR_KinematicPhysics scripts needs to be added to the Left and Right Hand Targets.

When the hands or the objects they are holding collide, the difference between the real hand position and the virtual hand position is translated into forces. These forces are applied to the object the virtual hand is colliding with. The amount of force depends on the strength parameter set in the IVR_KinematicPhysics script.

When there is no collision, the hands will follow the trackers directly in a kinematic way. When the parameter *Continuous NK* (non-kinematic) is selected, the hand will be moved around using physics. This will have the result that the hands and objects will have inertia which may be preceived as latency. The reason is that it take a certain amount of force before an object starts to move (which is inertia). As the force is calculated based on a difference between the virtual and real hand, the object will only start to move when the real hand is a certain distance away from the virtual hand. Normally you want to uncheck *Continuous NK* for this reason.

Grabbing Objects

Using the Razer Hydra, Kinect 2 or Leap Motion it is possible to grab any non-kinematic rigidbody with your hands.

Additionally some objects may have Handles attached which predetermine a position and orientation of that object when it is grabbed. This ensures that a sword or gun is always in the right position and orientation when it is grabbed. The range of the handle determines the working range of the handle. If the object is grabbed outside the range of the handle, the normal grabbing functionality is used. This enables you to grab a sword at the cutting end if you want.

Gaze Selection

With InstantVR you can select objects by looking at them. This is done by attaching the HeadMovements script to the head target. The field ‘Focus Point’ will then contain the current point of focus in space. This point can be made visible by enabling ‘Show Focus Point’. The default object used for the focus point is a sphere, but you can use a custom object using the ‘Focus Point Obj’ field.

When the avatar is looking at a Rigidbody, the field ‘Look Transform’ will contain the transform of the object which enables you to do something with that object.

Lastly, gaze selection supports Unity’s Event System when ‘Use Event System’ is checked. When the avatar is looking at an object with an Event Trigger attached and when the fire button is pressed, it will send the appropriate events to that object.

Custom avatars

A video showing how to use your own avatars and how to overcome any issues can be found here: https://www.youtube.com/watch?v=mB5VIc9k2m8

PlayMaker

InstantVR Advanced has builtin PlayMaker support. The following actions are implemented:

Character Move

Moves the character around

Game Object	the Game Object of the character which should be moved. The Game Object should have the InstantVR script attached.
Move Vector	the direction vector of the movement.

Character Rotate

Rotates the character

Game Object	the Game Object of the character which should be rotated. The Game Object should have the InstantVR script attached.
Angles	the desired rotation along the Y-axis

Character Collision

Detects collisions of the character with the static environment.

Game Object	the Game Object of the character for which collisions should be detected. The Game Object should have the InstantVR script attached.
Collision Start Event	the event which should be sent when the character collides with static objects without rigidbodies.
Collision End Event	the event which should be sent when the character no longer collides with static objects without rigidbodies.
Collision State	boolean value: is the character currently colliding?

Get Bone Information

Retrieves positions and rotations of bones

Game Object	the Game Object of the character for which information about the bones is requested. The Game Object should have the InstantVR script attached.
Bone	the bone for which information is requested
Position	position of the bone
Rotation	orientation of the bone

Get Focus Point

The position where the character is looking at.

Game Object	the Game Object of the character. The Game Object should have the InstantVR script attached.
Focus point	the position in world space to where the character is looking.
Every frame	should the focus point be updated for every frame

Get Hand Pose

Retrieves the amount each finger is bent.

Hand Object	the Game Object of the hand target. The Game Object

	should have the IVR_HandMovements script attached.
Thumb Curl	The amount (0..1) the thumb is bent.
Index Curl	The amount (0..1) the index finger is bent.
Middle Curl	The amount (0..1) the middle finger is bent.
Ring Curl	The amount (0..1) the ring finger is bent.
Little Curl	The amount (0..1) the little finger is bent.
Every frame	should the hand pose be updated for every frame

Set Hand Pose

Sets the amount each finger is bent.

Hand Object	the Game Object of the hand target. The Game Object should have the IVR_HandMovements script attached.
Thumb Curl	The amount (0..1) the thumb is bent.
Index Curl	The amount (0..1) the index finger is bent.
Middle Curl	The amount (0..1) the middle finger is bent.
Ring Curl	The amount (0..1) the ring finger is bent.
Little Curl	The amount (0..1) the little finger is bent.

Hand Grabbing

Retrieves information about objects which have been grabbed.

Hand Object	the Game Object of the hand target. The Game Object should have the IVR_HandMovements script attached.
Grabbed Event	the event to sent when an object has been grabbed.
Let Go Event	the event to sent when an object has been dropped.
Grabbed Object	the object which has been grabbed.

Get Controller Axis

Gets the values of the horizontal and vertical thumbstick axis of a game controller

Controller Side	left or right side of the controller
Store Vector	the Vector3 which should store the values of the thumbstick input

Get Controller Button

Get the status of one of the game controller buttons

Controller Side	left or right side of the controller.
Button	the button which we want to read.
Store Bool	stores the value of the button as a boolean. All buttons statuses are converted to booleans.
Store Float	stores the value of the button as a float. All button statuses are converted to float values.
Button Pressed	event to send when the button is pressed.
Button Released	event to send when the button is released.

Video

An instructional video about using PlayMaker with InstantVR can be found here: https://www.youtube.com/watch?v=JMV1ydKjEH8

Networking

Unity Networking

The recommended way to instantiate an InstantVR Avatar on Unity Networking is to use an IVR_Unet Avatar as the player prefab in the Network Manager. An example of this is found in the 'Grocerey Store Unet' demo. Alternative you can instantiate the avatar using Networking.Instantiate.

In the Networking map of the Prefab directory you can find an example IVR_Unet Avatar prefab called MH_VR_Unet. This is basically a MH_VR prefab supporting Unity Networking.

Besides the first-person MH_VR prefab (or other standard prefab of you choice), it also includes a reference to the MH_ThirdPerson prefab which is used on remote clients. The first person avatar should include the tracking options and can use a headless avatar, while the third person avatar can use a full body avatar and should not include any tracking options.

Setting up a network with a Server/Host and Clients is done in the regular way so we refer to the available Unity documentation for that.

Photon Networking

When using Photon networking, an avatar is instantiated by calling PhotonNetwork.Instantiate using the name of a IVR_Photon Avatar. Note that the avatar prefab should be located in a Resources map. An example scene using this is found in the 'Grocery Store PUN' demo.

In the Networking/Resources map of the Prefab directory you can find an example IVR_Photon Avatar prefab called MH_VR_Photon. This is basically a MH_VR prefab supoorting Photon Networking. Like with Unity Networking, it has references to first-person and third person avatars.

Setting up Photon networking is done in the regular way so we refer to the available Photon documentation for that. Note that Photon Server has not yet been tested with InstantVR and is for that reason not supported at the moment.

Known issues and limitations

The following issues are known to the current version of InstantVR:

In InstantVR v3, the priority of the controllers is not updated from the extensions priority when the inspector is in Debug Mode. The ordering will be corrected again when the inspector is switched back to Normal mode.

Leg walking animation does not work when Kinect is selected for the foot targets

Version history

Version 1.0

- Initial release

Version 1.1

- Unity Free support
- Objects can now be throwed
- Included options to enable/disable controllers
- Look rotation using Rift is now working properly
- Improved forearm IK
- Leg orientation improved with high foot positions
- Free and Advanced code bases merged

Version 2.0

- Added Microsoft Kinect support
- Physical drift correction
- Free walking support
- Side stepping
- Improved Razer Hydra calibration
- Removed Character Motor and Character Controller
- Hands no longer collide with body if they use the same layer

Version 2.1

- Proximity based walking speed
- Arm swing animation added when Hydra or Rift are not present

- Improved leg animation with thumbstick walking
- Improved Hydra and Rift calibration. No need to face the Rift camera anymore.
- Body rotation is now supported with Kinect
- The combination of Hydra and Kinect with body rotation is now supoorted

Version 3.0

- Unity 5 support
- Leap Motion (Rift mounted) support
- Kinect 2 support
- Modular architecture
- Body movements in editor
- Realtime switching between input controllers

Version 3.1

- Unity 5.1 support
- Google Cardboard support

Version 3.2

- Gear VR support
- Redesigned calibration

Version 3.3

- VicoVR support
- PlayMaker support
- Leap table mounted support
- Gaze selection

Version 3.4

- HTC Vive/SteamVR support
- Builtin game controller support

Version 3.5

- Native HTC Vive/SteamVR support (unity 5.4)

More information

The lastest and detailed support information can be found at the Passer VR website: http://serrarens.nl/passervr/support/instantvr-support/

Contact

You can contact me via email: support@passervr.com or use the contact form on my website: http://serrarens.nl/passervr/contact/

Thank you for supporting my work!
Pascal.