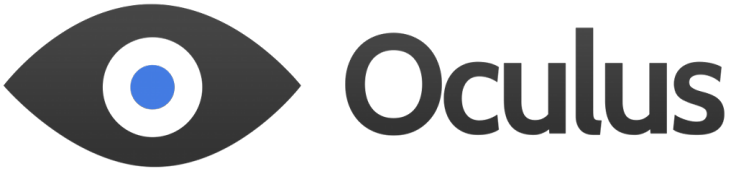

Oculus VR, Inc

Oculus Unity Integration Guide

SDK Version 0.2.5

Authors:
Peter Giokaris

Date:
November 27, 2013

Oculus Vr, Inc.
19800 MacArthur Blvd
Suite 450
Irvine, CA 92612

Contents

1 Introduction **4**
- 1.1 Requirements . . . 4
- 1.2 Monitor Setup . . . 4

2 First Steps **5**
- 2.1 Package Contents . . . 5
  - 2.1.1 Oculus Config Util . . . 5
- 2.2 Running the Pre-built Unity demo . . . 5
- 2.3 Control Layout . . . 6
  - 2.3.1 Keyboard Control . . . 6
  - 2.3.2 Mouse Control . . . 7
  - 2.3.3 Gamepad Control . . . 7

3 Using the Oculus Unity Integration **9**
- 3.1 Installation . . . 9
- 3.2 Working with the Unity Integration . . . 9
  - 3.2.1 Configuring for Standalone Build . . . 10
  - 3.2.2 Configuring Quality Settings . . . 11

4 A Detailed Look at the Unity Integration **13**
- 4.1 Directory Structure . . . 13
  - 4.1.1 OVR . . . 13
  - 4.1.2 Plugins . . . 14
  - 4.1.3 Tuscany . . . 14
- 4.2 Prefabs . . . 15
  - 4.2.1 OVRCameraController . . . 15
  - 4.2.2 OVRPlayerController . . . 17
- 4.3 C# Scripts . . . 19
  - 4.3.1 OVRDevice . . . 19
  - 4.3.2 OVRComponent . . . 19
  - 4.3.3 OVRCamera . . . 19
  - 4.3.4 OVRCameraController . . . 19
  - 4.3.5 OVRPlayerController . . . 22
  - 4.3.6 OVRMainMenu . . . 22
  - 4.3.7 OVRGamepadController . . . 23
  - 4.3.8 OVRPresetManager . . . 23
  - 4.3.9 OVRCrosshair . . . 23
  - 4.3.10 OVRGUI . . . 23
  - 4.3.11 OVRMagCalibration . . . 23
  - 4.3.12 OVRMessenger . . . 23
  - 4.3.13 OVRUtils . . . 23
- 4.4 Calibration Procedures . . . 24
  - 4.4.1 Magnetometer Yaw-Drift Correction . . . 24
- 4.5 Known Issues . . . 25
  - 4.5.1 Targeting a Display . . . 25
  - 4.5.2 Skyboxes . . . 25

4.5.3 Location of Tracker Query 25
4.5.4 MSAA Render Targets 25

5 Final Notes **26**

1 Introduction

Welcome to the Oculus Unity 3D integration!

This document will outline the steps required to get a Rift up and running in Unity. It will then go through some of the details within the integration so you have a better understanding of what's going on under the hood.

We hope that the process of getting your Rift integrated into your Unity environment is a fun and easy experience.

1.1 Requirements

Please see requirements in the Oculus SDK documentation to ensure you are able to interface with your Rift.

You must have **Unity Pro 4.2 or higher** installed on your machine to build the Unity demo.

The included demo scene allows you to move around more easily using a compatible gamepad controller (XBox on Windows and Linux, HID-compliant on Mac), so it is good to have one available. However, it is not required since there are keyboard mappings for the equivalent movements.

- *It is recommended that you read the Oculus SDK documentation first and are able to run the included SDK demo.*

1.2 Monitor Setup

For Unity, you must ensure that the Rift is set as the main display, or that it is duplicating the monitor which is set to main display.

In Windows 7 and 8, these settings can be found under Screen Resolution dialog when you right click on desktop.

On Mac with multiple displays, open up System Preferences and then navigate to Hardware / Displays. From there, select Arrangement tab, and check Mirror Displays.

When duplicating the displays it is recommended that you set the resolution to 1280 x 800, the resolution of the Developer Kit 1 (DK1) Rift. Optionally, a second monitor will allow you to have the Unity editor open within a higher-res screen.

2 First Steps

2.1 Package Contents

Extract the zip file to a directory of choice (e.g. C:\OculusUnity) You will find the following files under the **OculusUnityIntegration** directory:

- **OculusUnityIntegrationGuide.pdf** - This file.
- **ReleaseNotes.txt** - This file contains current and previous release information.
- **OculusUnityIntegration.unitypackage** - This package will install the minimum required files into Unity for Rift integration.
- **OculusUnityIntegrationTuscanyDemo.unitypackage** - This package will install the required files into Unity for Rift integration, along with the demo scene and all of the demo scene assets.

2.1.1 Oculus Config Util

You will also find a directory called **OculusConfigUtil**. This contains a configuration utility that allows one to set profiles which can be used to personalize various attributes for a given user (like **IPD** and **Player Eye Height**).

2.2 Running the Pre-built Unity demo

Figure 1: Tuscany demo screen-shot from within the Unity editor

To run the pre-built demos, please download the appropriate demo zip file for the platform that you would like to see it on.

For Windows, please download *demo_win.zip file.

For Mac, please download the *demo_mac.zip file.

For Linux (Ubuntu), please download the *demo_linux.zip file.

Run the OculusUnityDemoScene.exe **(Windows)**, OculusUnityDemoScene.app **(Mac)** or OculusUnityDemoScene **(Linux)** pre-built demo. If prompted with a display resolution dialog, hit the **Play!** button. The demo will launch in full-screen mode (note: if you are duplicating monitors, you will be able to see the stereo image on your 2D display as well).

2.3 Control Layout

2.3.1 Keyboard Control

Use the controls in **Table 1** to move around the environment and to change some key Rift settings:

KEYS	USAGE
W or Up arrow	Move player forward
A or Left arrow	Strafe player left
S or Down arrow	Move player back
D or Right arrow	Strafe player right
Left Shift	When held down, player will run
Q and E	Rotate player left / right
Right Shift	Toggles scene selection (Up and Down arrow will scroll through scenes)
Return	If scene selection is up, will load currently selected scene
X	Enable / Disable yaw-drift correction
C	Toggle mouse cursor on / off
P	Toggle prediction mode on / off
B	Reset orientation of tracker
Spacebar	Toggle configuration menu
, and .	Decrement / Increment prediction amount (in milliseconds)
[and]	Decrement / Increment vertical field of view (FOV) (in degrees)
- and =	Decrement / Increment interpuillary distance (IPD) (in millimeters)
1 and 2	Decrement / Increment distortion constant 1 (K1)
3 and 4	Decrement / Increment distortion constant 2 (K2)
5 and 6	Decrement / Increment player height (in meters)
7 and 8	Decrement / Increment player movement speed multiplier
9 and 0	Decrement / Increment player rotation speed multiplier
Tab	Holding down Tab and pressing F3 - F5 will save a snapshot of current configuration
F2	Reset to default configuration
F3 F4 F5	Recall saved configuration snapshot
F6	Toggle Mag Compass on / off if Mag reference orientation set
F11	Toggle between full-screen and windowed mode
Esc	Quit application

Table 1: Keyboard mapping for Demo

2.3.2 Mouse Control

Using the mouse will rotate the player left and right. If the cursor is enabled, The mouse will track the cursor and not rotate the player until the cursor is off screen.

2.3.3 Gamepad Control

- If you have compliant gamepad controller for your platform, you can control the movement of the player controller with it.
- The left analog stick will move the player around as if you were using the **W,A,S,D** keys.
- The right analog stick will rotate the player left and right as if you were using the **Q** and **E** keys.
- The left trigger will allow you move faster, or run through the scene.
- The start button will toggle scene selection. Pressing D-Pad Up and D-Pad Down will scroll through available scenes. Pressing the A-Button will start the currently selected scene.

- If scene selection is not toggled, Pressing D-Pad Down will reset orientation of tracker.

3 Using the Oculus Unity Integration

We have included two Unity packages for you to use. The **OculuUnityIntegration** is a minimal package that you would use to import into into a project you want to integrate the Rift into. However, we recommend walking through the Tuscany demo to get a feel for how the integration works.

It's time to build the demo that you just played. To do this, you will be importing the **OculusUnityIntegrationTuscanyDemo** Unity package into a new Unity project.

3.1 Installation

If you already have Unity open, you should first save your work.

Create new empty project that will be used to import the Oculus assets into. You will not need to import any standard or pro Unity asset packages; the Oculus Unity Integration is fully self-contained.

Double-click on the **OculusUnityIntegrationTuscanyDemo.unitypackage** file. This will import the assets into your empty project. The import process may take a few minutes to complete.

3.2 Working with the Unity Integration

Click on the project browser tab (usually found on the bottom left-hand corner) and navigate into Tuscany/Scenes. Double-click on the VRDemo_Tuscany scene to load it as the current scene (if you are prompted to save the current scene, select **Don't save**).

Press the **PLAY** button at the top of editor. The scene will start and the Rift will be controlling the camera's view. The game window will start in the main editor application as a tabbed view:

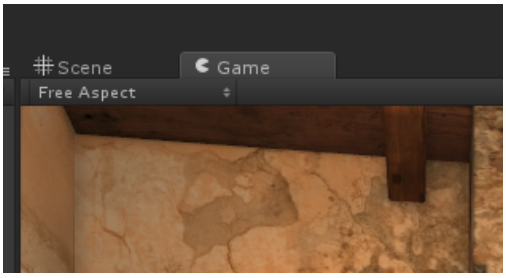

Figure 2: Tuscany demo Game tab

Tear the Game view from the editor and drag onto the Rift display. Click the square (maximize) button on the top right of the game window (to the left of the 'X' button) to go full-screen. This will let you play the game full screen, while still allowing you to keep the Unity editor open on another monitor. It is recommended that you also 'Auto-hide the taskbar', which you can find under **Taskbar and Start Menu Properties - Taskbar**.

3.2.1 Configuring for Standalone Build

Now it's time to build the demo as a standalone full-screen application. We will need to make a few project setting changes to maximize the fidelity of the demo.

Click on **File - Build Settings...** For **Windows**, set **Target Platform** to 'Windows' and set **Architecture** to either 'x86' or 'x86_64'. For **Mac**, set **Target Platform** to 'Mac OS X'. For **Linux**, set **Target Platform** to 'Linux' and set **Architecture** to either 'x86' or 'x86_64'.

Within the Build Settings pop-up, Click on **Player Settings**. Under **Resolution and Presentation**, set the values to the following:

Figure 3: Resolution and Presentation options

In the Build Settings pop-up, hit **Build and Run**, and if asked, specify a location and name for the build.

Provided you are building in the same OS, the game should start to run in full-screen mode as a stand-alone application.

3.2.2 Configuring Quality Settings

You may notice that the graphical fidelity is not as high as the pre-built demo. We must change some more Project settings to get a better looking scene.

Navigate to **Edit - Project Settings - Quality**. Set the values in this menu to be the same as the following:

Figure 4: Quality settings for Oculus demo

The most notable quality Setting value to set is **Anti-aliasing**. We need to increase the anti-aliasing amount to compensate for the stereo rendering, which is effectively reducing the horizontal resolution by 50%. A value of 2X or higher is ideal.

OK, now re-build again and the quality should be at the same level as the pre-built demo.

4 A Detailed Look at the Unity Integration

We will now examine some of the finer details of the integration. We will take a look at the directory structure of the integration first. We will then examine the prefabs and also at some of the key C# scripts.

4.1 Directory Structure

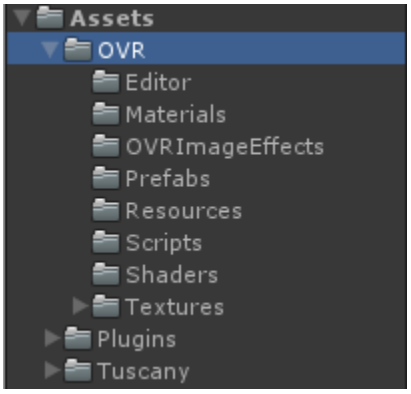

Figure 5: Directory structure of Oculus Unity Integration

4.1.1 OVR

OVR is the top-level folder of the Oculus Unity integration. Everything under OVR should be name-safe when importing the minimal integration package into a new project (using **OculusUnityIntegration.unitypackage**).

The OVR directory contains the following directories:

Editor:	Contains scripts that add functionality to the Unity Editor, and enhance several C# component scripts.
Materials:	Contains materials that are used for graphical components within the integration, such as the Magnetometer compass and the main GUI display.
OVRImageEffects:	Contains scripts and the lens correction shader required to warp the images within the rift display.
Prefabs:	Contains the main prefabs used to bind the Rift into a Unity scene: **OVRCameraControler** and **OVRPlayerController**.
Resources:	Contains prefabs and other objects that are required and instantiated by some OVR scripts, such as the magnetometer compass geometry.
Scripts:	Contains the C# files that are used to tie the Rift and Unity components together. Many of these scripts work together within the Prefabs.
Shaders:	Contains the C# files that are used to tie the Rift and Unity components together. Many of these scripts work together within the Prefabs.
Textures:	Contains image assets required by some of the script components.

4.1.2 Plugins

The Plugins directory contains the **OculusPlugin.dll** which allow for the Rift to communicate with Unity on WIndows (both 32 and 64 bit versions).

This folder also contains the plugins for other platforms (i.e. **OculusPlugin.bundle** for MacOS).

4.1.3 Tuscany

Tuscany contains all of the Tuscany assets needed for the demo.

4.2 Prefabs

The current integration of the Rift into Unity revolves around two prefabs that can be added into a scene:

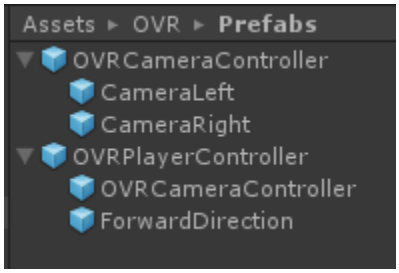

Figure 6: Prefabs: OVRCameraController and OVRPlayerController

To use, you simply need to drag and drop one of the prefabs into your scene.

As well, you can add these prefabs into the scene from the **Oculus - Prefabs** menu.

4.2.1 OVRCameraController

OVRCameraController is meant to be the replacement stereo camera for a single Unity camera within a scene. You can drag an OVRCameraController into your scene (make sure to turn off any other camera in the scene to ensure that OVRCameraController is the only one being used) and you will be able to start viewing the scene with the Rift.

OVRCameraController contains 2 Unity cameras, one for each eye. It is meant to be attached to a moving object (such as a character walking around, a car, a gun turret etc.), replacing the conventional camera.

Two scripts (components) are attached to the OVRCameraController prefab: **OVRCameraController.cs** and **OVRDevice**.

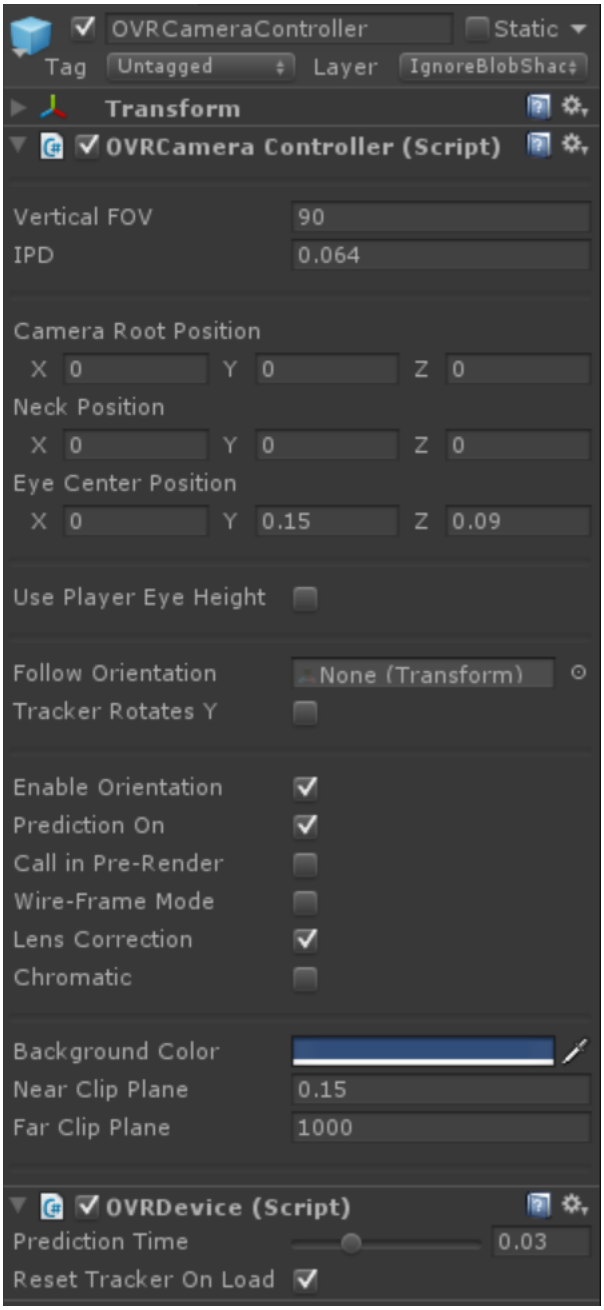

Figure 7: Prefabs: OVRCameraController, expanded in the inspector

4.2.2 OVRPlayerController

The OVRPlayerController is the easiest way to start 'navigating' a virtual environment. It is basically an OVRCameraController prefab attached to a simple character controller, including a physics capsule, movement, a simple menu system with stereo rendering of text fields and a crosshair component.

To use, drag the player controller into an environment and begin moving around using a gamepad and / or a keyboard and mouse (make sure that collisions are present in the environment :)).

Four scripts (components) are attached to the OVRPlayerController prefab: **OVRPlayerController.cs**, **OVRGamepadController.cs**, **OVRMainMenu.cs** and **OVRCrosshair.cs**.

NOTE: OVRMainMenu.cs and OVRCrosshair.cs are helper classes, and are not required for the OVRPlayerController to work. They are available to help get a rudimentary menu and a cursor on-screen.

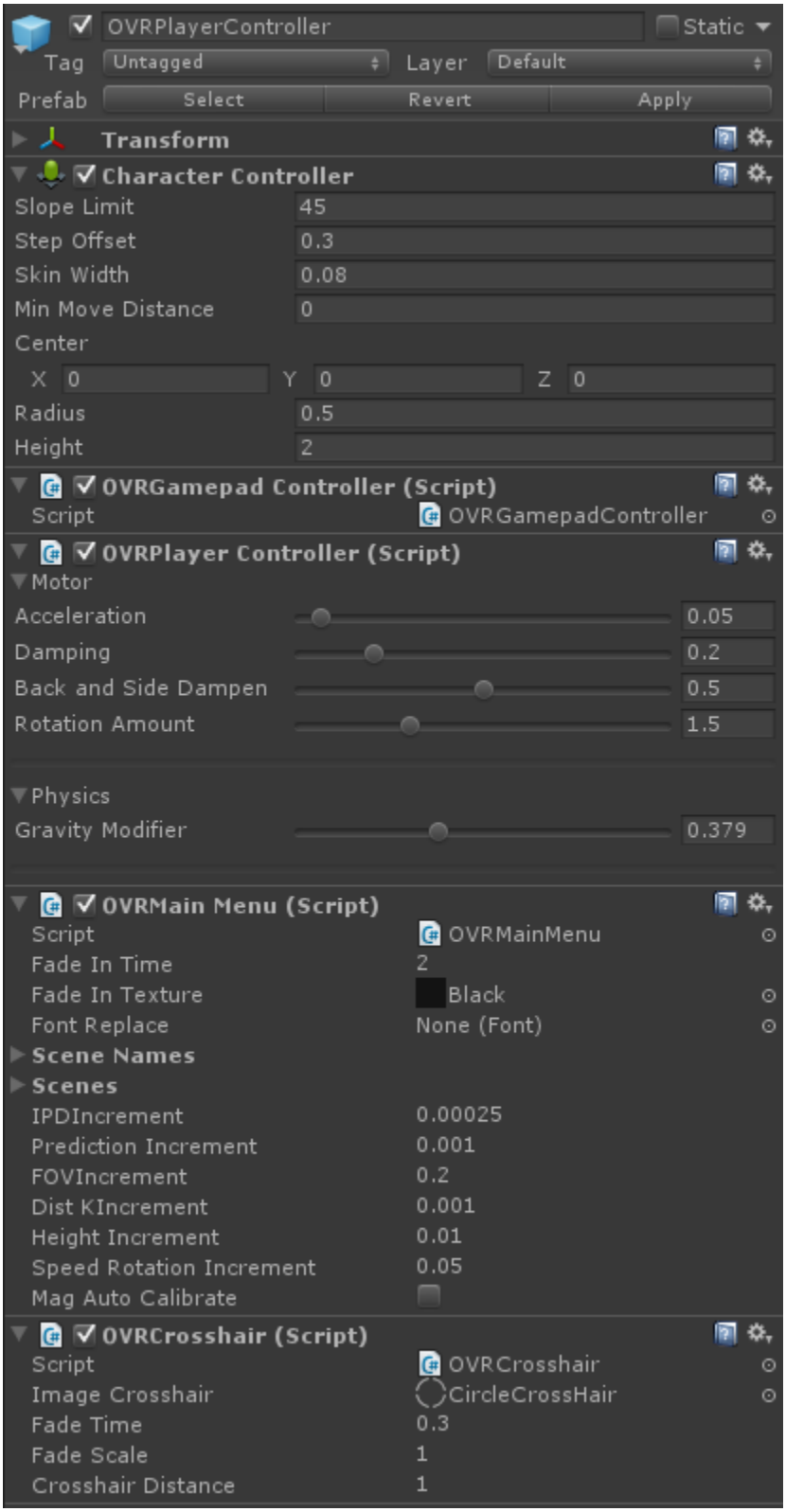

Figure 8: Prefabs: OVRPlayerController, expanded in the inspector

4.3 C# Scripts

The following section gives a general overview of what each of the scripts within the **Scripts** folder does.

4.3.1 OVRDevice

OVRDevice is the main interface to the Oculus Rift hardware. It includes wrapper functions for all exported C++ functions, as well as helper functions that use the stored Oculus variables to help set up camera behavior.

This component is added to the OVRCameraController prefab. It can be part of any game object that one sees fit to place it. However, it should only be declared once, since there are public members that allow for tweaking certain Rift values in the Unity inspector.

OVRDevice.cs contains the following public members:

Prediction Time: This field adjusts how far ahead in time the tracker will predict the players head position will be. It is effective in covering input to video latency. The default value is 50 milliseconds, but can be adjusted accordingly.

Reset Tracker On Load: This value defaults to true. When turned off, subsequent scene loads will keep the tracker from resetting. This will keep the tracker orientation the same from scene to scene, as well as keep magnetometer settings intact.

4.3.2 OVRComponent

OVRComponent is the base class for many of the OVR classes. It is used to provide base functionality to classes derived from it.

4.3.3 OVRCamera

OVRCamera is used to render into a Unity Camera class.

This component handles reading the Rift tracker and positioning the camera position and rotation. It also is responsible for properly rendering the final output, which also the final lens correction pass.

4.3.4 OVRCameraController

OVRCameraController is a component that allows for easy handling of the lower level cameras.

It is the main interface between Unity and the cameras. This is attached to a prefab that makes it easy to add a Rift into a scene.

NOTE: All camera control should be done through this component.

It is important to understand this script when integrating a camera into your own controller type.

OVRCameraController.cs contains the following public members:

Vertical FOV: This field sets up the vertical Field Of View (FOV) for the cameras, in degrees. This is calculated by the SDK for optimal viewing, but one can override this value if desired.

IPD: This field sets up the Inter-Pupillary Distance for the cameras, in meters. The default is set to either the average IPD value or the value that has been set by the Oculus Config Tool. One can also override this value if desired.

Camera Root Position: This field sets up where the camera is situated in relation to the root of the object which it is attached to. In OVRPlayerController, this value is set to 1.0f, effectively at the center of the default collision capsule which is set to 2.0 meters.

Neck Position: This field sets up where the neck is located in relation to the gameObject which the script is attached to. This field should be modified when requiring the head to animate (either procedurally, such as bobbing, or when attached to a game object).

Use Player Eye Height: This field is used to set the actual player eye height from the Oculus Config Tool. If set, the eye height will be calculated from the default player profile. When on, the Neck Position will be adjusted in relation to the Camera Root Position and Eye Center Position so that the total of all three equals the Player Eye Height. Note that while this is enabled, Writing to the Neck Position via 'Set Neck Position' will not be allowed. Please see OVRCameraController.cs script for more details on this implementation.

Eye Center Position: This field is used to set where the center of the eyes are located for head modeling. Note that the Interpupillary Distance (IPD) value which is used to separate the left and right eyes are set automatically, but this value can be set as well (see IPD field, above).

Follow Orientation: If this set, the Cameras will always follow the rotation component and the head-tracking will be an offset of this. Useful for mounting the Camera on a rigid object (like a car or a body).

Tracker Rotates Y: When set, the actual Y rotation of the Cameras will set the Y rotation value of the parent transform which it is attached to. *This may be expanded to include X and Z values as well.* This is useful for giving other components or other game objects feedback on what the actual orientation of the camera is. Currently this is used by OVRPlayerController to allow head tracking to influence forward direction of the controller. **Setting 'Follow Orientation' with the parent transform of this component and setting Tracker Rotates Y to true will cause a feedback loop in orientation.**

Enable Orientation: When disabled, the Orientation of the tracker will stop updating the camera orientation.

PredictionOn: When set, prediction correction on the tracker will be set. The amount of prediction time can be adjusted through the OVRDevice component located in the OVRCameraController prefab.

CallInPreRender: When set, rendering will take place in the 'OnPreRender' function within OVRCamera.cs This has the effect of reading the tracker closer to the render (and hence, lower actual latency). However, please see section **Known Issues** for an explanation of issues that occur with this setting.

WireMode: When set, rendering will of scene will be in wire-frame.

LensCorrection: When set, Lens Correction pass will be turned on.

Chromatic: When set, the shader to fix Chromatic Aberration will be used. Chromatic Aberration is caused by the lenses, in which the colors of the image are radially spread out, causing color fringes to occur on certain parts of the rendered image.

Background Color: Sets the background color of both cameras attached to the OVRCameraController prefab.

Near Clip Plane: Sets the near clip plane value of both cameras.

Far Clip Plane: Sets the far clip plane of both cameras.

4.3.5 OVRPlayerController

OVRPlayerController implements a basic first person controller for the Rift. It is attached to the OVRPlayerController prefab, which has an OVRCameraController attached to it.

The controller will interact properly with a Unity scene, provided that the scene has collision assigned to it.

The OVRPlayerController prefab has an empty GameObject attached to it called ForwardDirection. This game object contains the matrix which motor control bases it direction on. This game object should also house the body geometry which will be seen by the player.

4.3.6 OVRMainMenu

OVRMainMenu is used to control the loading of different scenes. It also renders out a menu that allows a user to modify various Rift settings, and allow for storing these settings for recall later.

A user of this component can add as many scenes that they would like to be able to have access to.

OVRMainMenu is currently attached to the OVRPlayerController prefab for convenience, but can safely removed from it and added to another GameObject that is used for general Unity logic.

4.3.7 OVRGamepadController

OVRGamepadController is an interface class to a gamepad controller.

On Windows machines, the gamepad must be XInput-compliant.

Currently native XInput-compliant gamepads are not supported on MacOS. Please use the conventional Unity Input methods for gamepad input.

4.3.8 OVRPresetManager

OVRPresetManager is a helper class to allow for a set of variables to be saved and recalled using the Unity PlayerPrefs class.

OVRPresetManager is currently being used by the OVRMainMenu component.

4.3.9 OVRCrosshair

OVRCrosshair is a component that adds a stereoscopic cross-hair into a scene. NOTE: Crosshair currently draws into the scene after the view has been rendered, therefore there is no distortion correction on it.

4.3.10 OVRGUI

OVRGUI is a helper class that encapsulates basic rendering of text in either 2D or 3D. The 2D version of the code will be deprecated in favor of rendering to a 3D element (currently used in OVRMainMenu).

4.3.11 OVRMagCalibration

OVRMagCalibration is a helper class that allows for calibrating the magnetometer for y-drift correction.

4.3.12 OVRMessenger

OVRMessenger is a class that allows for advanced messages between different classes. It is a direct copy of the 'Advanced CSharp Messenger' that is available on the Unity Unify Community Wiki under 'General Concepts'.

4.3.13 OVRUtils

OVRUtils is a collection of reusable static functions.

4.4 Calibration Procedures

The following section outlines basic calibration procedures that a developer may want to include in their application. Mainly it will describe the calibration process that has been set up in the Unity integration; for a more detailed understanding of a particular calibration, please see the Oculus SDK documentation.

4.4.1 Magnetometer Yaw-Drift Correction

In order for yaw-drift calibration to take place, the magnetometer must have been pre-calibrated in the Calibration tool.

To toggle yaw-drift correction on and off, press 'X'.

Provided that calibration was successful, pressing 'F6' will' toggle a visual 'compass' that will indicate how far the sensor has drifted. You will be able to see the world along with the compass lines rotate while this is happening.

4.5 Known Issues

The following section outlines some currently known issues with Unity and the Rift that will either be fixed in later releases of Unity or the Rift Integration package.

A work-around may be available to compensate for some of the issues at this time.

4.5.1 Targeting a Display

Future support for targeting a display for full-screen mode will be included in a future Unity release.

4.5.2 Skyboxes

Unity skyboxes are not affected by the perspective matrix calculations done on the cameras. Therefore, if a skybox is needed, you will need to provide a custom skybox.

For an example of how to implement a custom skybox, please look at the skybox that has been included with the Tuscany demo level.

4.5.3 Location of Tracker Query

Calling the tracker for orientation can be done in one of two places with OVRCamera.

Calling it within the function **OnPreCull** ensures that the integrity of many Unity sub-systems that require camera orientation to be done before camera culling is called is maintained. However, there is an added latency that occurs by calling it here.

Querying the tracker within the function **OnPreRender** reduces the latency, but the integrity of certain systems cannot be guranteed. It is left to the user to decide which location best suits the application.

4.5.4 MSAA Render Targets

Unity 4.2 and on supports MSAA Render Targets. However, there are a few known issues when substituting a Unity camera with a render target with anti-aliasing turned on:

- In Forward rendering, dynamic lights will not work correctly. If the the render target is larger then the final target, you will see a potential 'cut-out' effect on the screen. As well, if dynamic shadows are turned on, rendering will not occur, or a subset of the geometry will render.
- Deferred rendering will not work if anti-aliasing is turned on (a value of 2, 4 or 8).

5 Final Notes

We hope you find making VR worlds within Unity as fun and inspiring as we had bringing the Rift to you.

Now, go make some amazing new realities!

Questions?

Please visit our developer support forums at:

`https://developer.oculusvr.com`

or send us a message at support@oculusvr.com