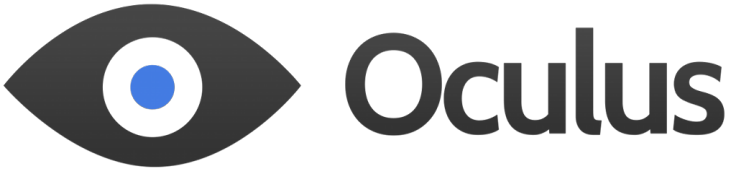

Oculus VR, LLC

Oculus Unity Integration Guide

SDK Version 0.4.4

Authors:
Peter Giokaris, David Borel

Date:
December 2, 2014

Oculus VR, LLC
Irvine, CA

Contents

1 Introduction

Welcome to the Oculus Unity 3D Integration Guide.

This document describes the steps and procedures required to get a Oculus Rift configured and running in Unity. It will also cover 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 the Requirements section in the Oculus SDK Overview documentation to ensure that your hardware and software are able to interface with your Rift.

You must have **Unity Pro 4.5 or later** installed on your system to build the Unity demo.

The included demo scene allows you to move around more easily using a compatible gamepad controller (XBox on Windows, HID-compliant on Mac). This is the recommended controller interface. However, there are also keyboard mappings available for the equivalent movement control.

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

1.2 Direct Mode Display Driver

On Windows, Oculus recommends users install the Oculus Display Driver, which includes a feature known as Direct Display Mode. In direct mode, the driver makes your Rift behave as an appliance instead of a standard Windows display. Applications target the rift by loading the driver and pointing rendering to it before initialization. This is the default behavior. For compatibility, the Rift can also still be used as a Windows monitor. This is known as Extended Display Mode. When extended mode is active, rendering works as usual, with no Oculus intervention. You can choose the mode from the Oculus Configuration Utility's Rift Display Mode screen. The direct mode driver is not yet available for platforms other than Windows.

1.3 Monitor Setup

To get the best experience, you and your users should always configure the Rift correctly.

In Windows 7 and Windows 8, you can change Windows' display settings by right-clicking on the desktop and selecting "Screen resolution".

- It is possible to clone the same image on all of your displays. To ensure each display uses the correct frequency, Oculus recommends extending the desktop instead of cloning it.
- If you are using the Rift in extended mode, it should be set to its native resolution. This is 1920x1080 for DK2 and 1280x800 for DK1.

On Mac systems open **System Preferences**, and then navigate to **Displays**.

- As with Windows, it is possible to mirror the same image on all of your displays. Oculus recommends against mirroring. Click **Arrangement** and ensure **Mirror Displays** is not enabled.
- Some Unity applications will only run on the main display. In the **Arrangement** screen, drag the white bar onto the Rift's blue box to make it the main display.
- Always use the Rift's native resolution and frequency. Click **Gather Windows**. For DK2, the resolution should be **Scaled** to 1080p, the rotation should be 90°and the refresh rate should be 75 Hertz. For DK1, the resolution should be 1280x800, the rotation should be **Standard**, and the refresh rate should be 60 Hertz.

1.4 Recommended Configuration

Version 0.4 of the Oculus Unity integration is beta software. Some configurations work better than others. We recommend the following settings in your project:

- On Windows, enable Direct3D 11. D3D 11 and OpenGL expose the most advanced VR rendering capabilities. In some cases, using D3D 9 may result in slightly reduced visual quality or performance.
- Use the Linear Color Space. Linear lighting is not only more correct for shading, it also causes Unity to perform sRGB read/write to the eye textures. This helps reduce aliasing during VR distortion rendering, where the eye textures are interpolated with slightly greater dynamic range.
- Never clone displays. When the Rift is cloned with another display, the application may not vsync properly. This leads to visible tearing or judder (stuttering or vibrating motion).
- On Windows, always run DirectToRift.exe. Even in extended mode, DirectToRift.exe makes your application run full-screen on the Rift.
- When using the Unity editor, use extended display mode. Direct mode is currently supported only for standalone players. Using it with the Unity editor will result in a black screen on the Rift.

2 First Steps

2.1 Package Contents

Extract the zip file to a directory of choice (for example, **C:\OculusUnity**) You will find the following files located in the **OculusUnityIntegration** directory:

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

2.2 Running the Pre-built Unity demo

Figure 1: Tuscany demo screenshot from within the Unity editor

To run the pre-built demos, download the appropriate demo zip file for the platform you need.

For Windows, download the *demo_win.zip file.

For Mac, download the *demo_mac.zip file.

Run the OculusUnityDemoScene.exe **(Windows)** or OculusUnityDemoScene.app **(Mac)** 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).

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)
Enter	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
M	Toggle screen mirroring on / off (Direct mode only)
R	Reset tracker orientation
Spacebar	Toggle configuration menu
, and .	Decrement / Increment prediction amount (in milliseconds)
[and]	Decrement / Increment vertical field of view (FOV) (in degrees)
- and =	Decrement / Increment interpupillary distance (IPD) (in millimeters)
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 the current configuration
F1	Toggle low-persistence mode on / off
F2	Reset to the default configuration
F3 F4 F5	Recall a saved configuration snapshot
F11	Toggle between full-screen and windowed mode
Esc	Quit application

Table 1: Keyboard mapping for Demo

2.3 Control Layout

2.3.1 Keyboard Control

Table 1 describes the key mappings for the demo that allow the user to move around the environment, as well as changing some Rift device settings:

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 a compliant gamepad controller for your platform, you can control the movement of the player controller with it.
- The left analog stick moves the player around as if you were using the **W,A,S,D** keys.
- The right analog stick rotates the player left and right as if you were using the **Q** and **E** keys.

- The left trigger allows you move faster, or run through the scene.
- The Start button toggles the scene selection. Pressing D-Pad Up and D-Pad Down scrolls through available scenes. Pressing the **A** button starts the currently selected scene.
- If the scene selection is not turned on, Pressing the D-Pad Down resets the orientation of the tracker.

3 Using the Oculus Unity Integration

There are two Unity packages available to use. The **OculusUnityIntegration** is a minimal package that you would use to import for integrating a Rift into your project. However, we recommend viewing the Tuscany demo and reviewing the code to better understand how the integration works.

We also recommend reading through the Oculus Best Practices Guide, which has tips, suggestions, and research oriented around developing great VR experiences. Topics include control schemes, user interfaces, cut-scenes, camera features, and gameplay. The Best Practices Guide should be a go-to reference when designing your Oculus-ready games. For the most up-to-date information on best practices in VR content creation for the Rift, please visit `http://developer.oculusvr.com/best-practices`.

Now you should build a version of 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 before continuing.

Next, create a new project that you can import the Oculus assets into. You do 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 new 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 the Tuscany/Scenes folder. Double-click on the VRDemo_Tuscany scene to load. If you are prompted to save the current scene, select **Don't save**.

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

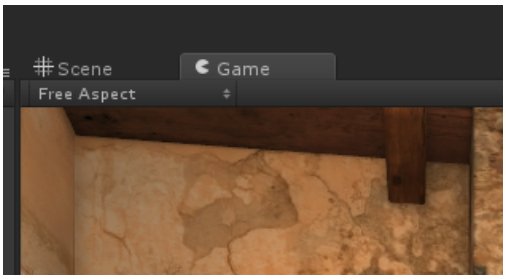

Figure 2: Tuscany demo Game tab

Move the Game view from the editor and drag onto the Rift display. Click the maximize button on the top right of the game window (to the left of the **X** button) to switch to fullscreen. This will let you play the game in fullscreen, while still allowing you to keep the Unity editor open on another monitor. It is recommended that you also automatically hide the taskbar, which is located in the **Taskbar and Start Menu Properties - Taskbar**.

3.2.1 Configuring for Standalone Build

Next, you should build the demo as a standalone fullscreen application. To do this, you will need to change a few project settings to maximize the fidelity of the demo.

Click on **File - Build Settings...** and select one of the following:

- 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**.

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

In the Build Settings pop-up, select **Build and Run**. If prompted, specify a name and location for the build.
If you are building in the same OS, the demo should start to run in fullscreen mode as a standalone application.

3.2.2 Configuring Quality Settings

You may notice that the graphical fidelity is not as high as the pre-built demo. You will need to change some additional project settings to get a better looking scene.

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

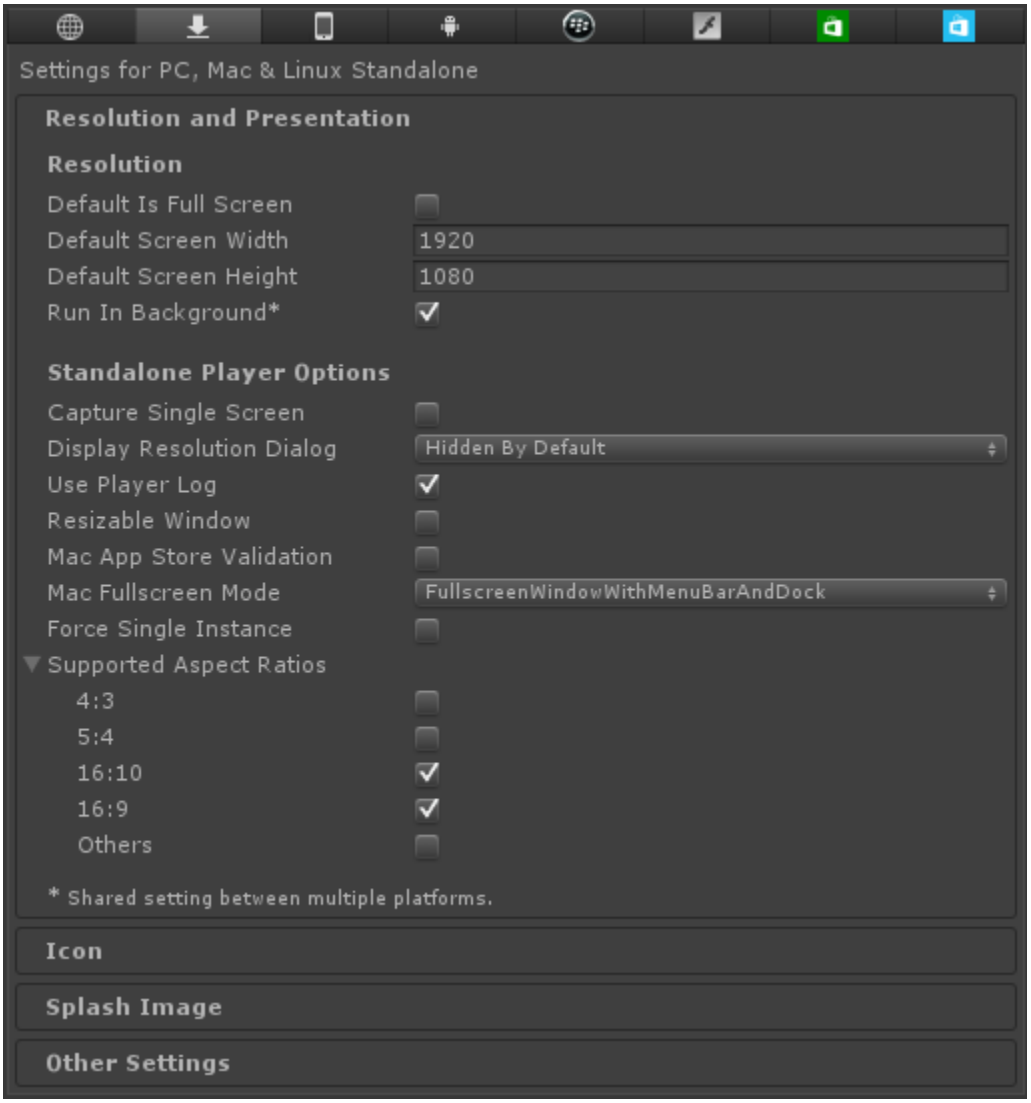

Figure 3: Resolution and Presentation options

Figure 4: Quality settings for Oculus demo

The most important value to modify is **Anti-aliasing**. The anti-aliasing must be increased to compensate for the stereo rendering, which reduces the effective horizontal resolution by 50%. An anti-aliasing value of 4X or higher is ideal. However, if necessary, you can adjust to suit your application needs.

Important: For the 0.4 release, due to a potential performance issue with Unity 4.5 and OSX 10.9, a new quality setting called 'Fastest' has been added. This setting turns off effects and features that may cause the drop in performance.

Now rebuild the project again, and the quality should be at the same level as the pre-built demo.

4 A Detailed Look at the Unity Integration

This section examines some additional information of the Unity integration. The directory structure of the integration is covered. Also, the Unity prefabs are described, as well as several of the key C# scripts.

4.1 Directory Structure

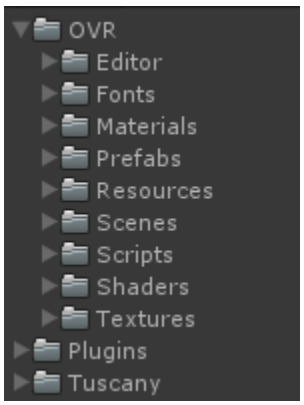

Figure 5: Directory structure of Oculus Unity Integration

4.1.1 OVR

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

The OVR directory contains the following subdirectories:

Editor	Contains scripts that add functionality to the Unity Editor, and enhance several C# component scripts.
Fonts	Contains fonts that are used for graphical components within the integration.
Materials	Contains materials that are used for graphical components within the integration, such as the main GUI display.
Prefabs	Contains the main Unity prefabs that are used to bind the Rift into a Unity scene: **OVRCameraRig** and **OVRPlayerController**.
Resources	Contains prefabs and other objects that are required and instantiated by some OVR scripts, such as the main GUI.
Scenes	Contains sample scenes.
Scripts	Contains the C# files that are used to tie the Rift and Unity components together. Many of these scripts work together within the various Prefabs.
Shaders	Contains the C# files that are related to shader usage.
Textures	Contains image assets that are required by some of the script components.

4.1.2 Plugins

The Plugins folder contains the **OculusPlugin.dll**, which enables the Rift to communicate with Unity on Windows (both 32 and 64-bit versions).

This folder also contains the plugins for other platforms: **OculusPlugin.bundle** for MacOS.

4.1.3 Tuscany

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

4.2 Prefabs

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

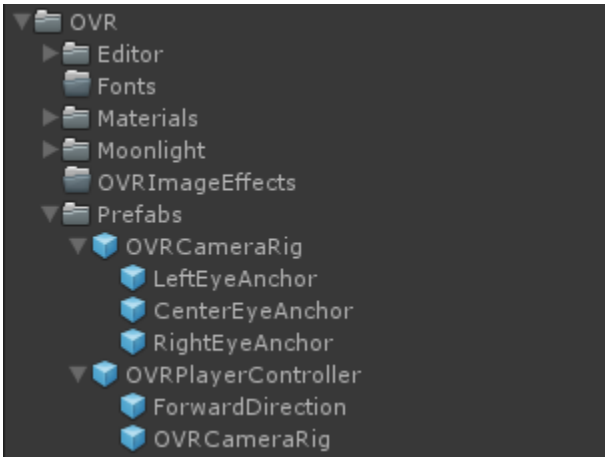

Figure 6: Prefabs: OVRCameraRig and OVRPlayerController

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

Note: You can also add these prefabs into the scene from the **Oculus - Prefabs** menu.

4.2.1 OVRCameraRig

OVRCameraRig replaces the native Unity camera within a scene. You can drag an OVRCameraRig into your scene and you will be able to start viewing the scene with the Rift. **Note:** Make sure to turn off any other camera in the scene to ensure that OVRCameraRig is the only one being used.

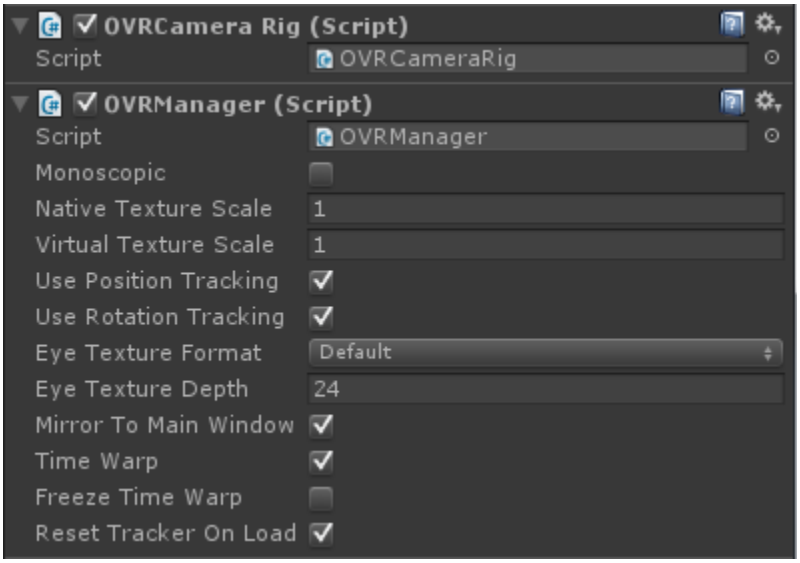

Figure 7: Prefabs: OVRCameraRig, expanded in the inspector

OVRCameraRig contains two 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.) This replaces the conventional camera.

The following scripts (components) are attached to the OVRCameraRig prefab:

- **OVRCameraRig.cs**
- **OVRManager.cs**

Note: OVRMainMenu.cs is a helper class, and is not required for the OVRPlayerController to work. It is available to enable a rudimentary menu on-screen.

4.2.2 OVRPlayerController

The OVRPlayerController is the easiest way to start navigating a virtual environment. It is basically an OVRCameraRig prefab attached to a simple character controller, and includes a physics capsule, a movement system, a simple menu system with stereo rendering of text fields, and a cross-hair component.

To use, drag the player controller into an environment and begin moving around using a gamepad, or a keyboard and mouse **Note:** Make sure that collision detection is active in the environment.

Three scripts (components) are attached to the OVRPlayerController prefab:

- **OVRPlayerController.cs**
- **OVRGamepadController.cs**
- **OVRMainMenu.cs**.

Note: OVRMainMenu.cs is a helper class, and is not required for the OVRPlayerController to work. It is available to enable a rudimentary menu on-screen.

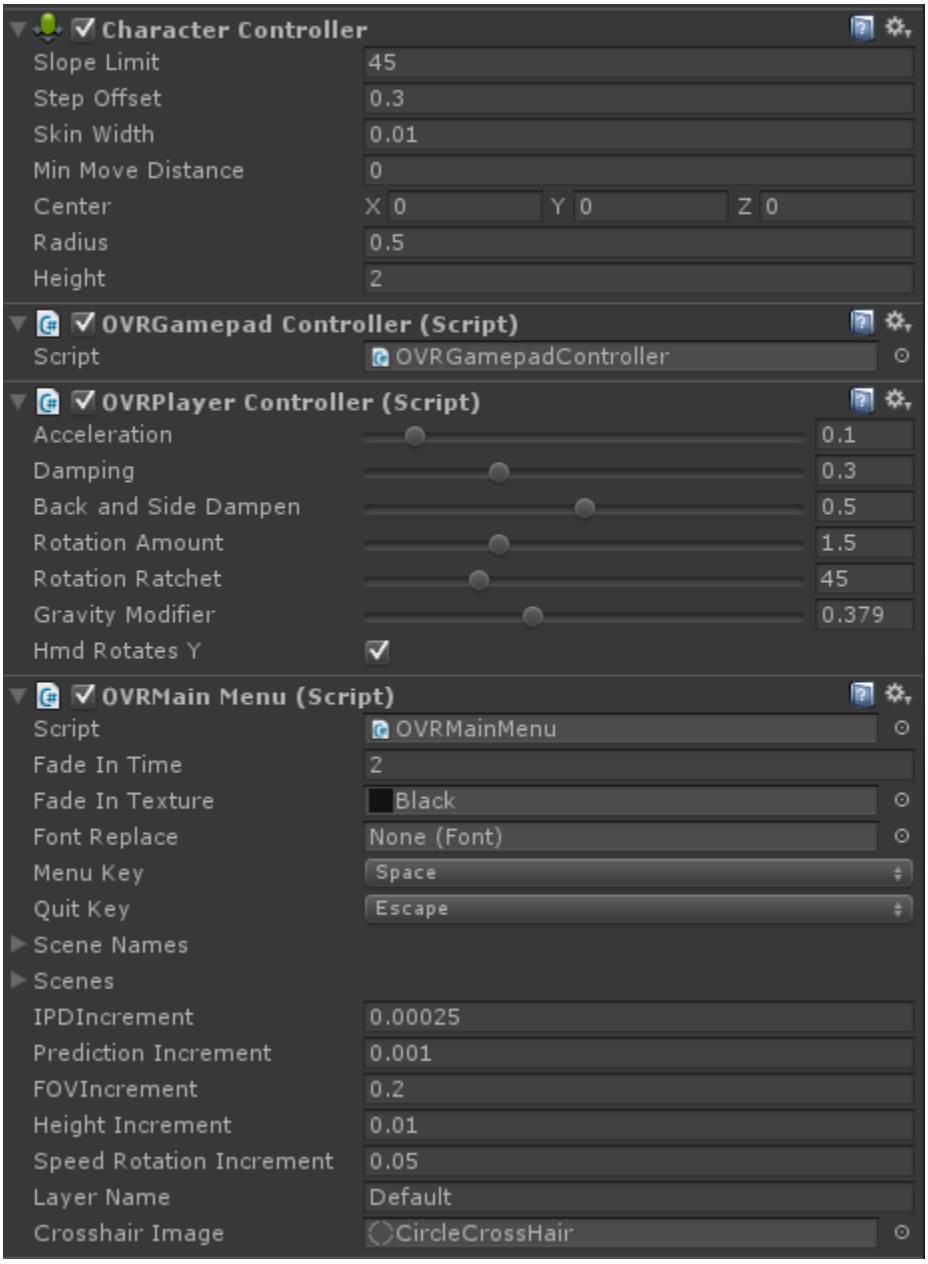

Figure 8: Prefabs: OVRPlayerController, expanded in the inspector

4.3 Unity Components

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

4.3.1 OVRCameraRig

OVRCameraRig is a component that controls stereo rendering and head tracking. It maintains three child "anchor" Transforms at the poses of the left and right eyes, as well as a virtual "center" eye that is half-way between them.

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

Important: All camera control should be done through this component. You should understand this script when integrating a camera into your own controller type.

4.3.2 OVRManager

OVRManager 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 Rift variables to help configure camera behavior.

This component is added to the OVRCameraRig prefab. It can be part of any application object. However, it should only be declared once, because there are public members that allow for changing certain Rift values in the Unity inspector.

OVRManager.cs contains the following public members:

Native Texture Scale	Each camera in the camera controller creates a RenderTexture that is the ideal size for obtaining the ideal pixel fidelity (a 1-to-1 pixel size in the center of the screen post lens distortion). This field can be used to permanently scale the cameras' render targets to any multiple ideal pixel fidelity, which gives you control over the trade-off between performance and quality.
Virtual Texture Scale	This field can be used to dynamically scale the cameras render target to values lower then the ideal pixel fidelity, which can help reduce GPU usage at run-time if necessary.
Use Position Tracking	If disabled, the position detected by the tracker will stop updating the HMD position.
Use Rotation Tracking	If disabled, the orientation detected by the tracker will stop updating the HMD orientation.
Mirror to Display	If the Oculus direct-mode display driver is enabled and this option is set, the rendered output will appear in a window on the desktop in addition to the Rift. Disabling this can slightly improve performance.

Time Warp	Time warp is a technique that adjusts the on-screen position of rendered images based on the latest tracking pose at the time the user will see it. Enabling this will force vertical-sync and make other timing adjustments to minimize latency.
Freeze Time Warp	If enabled, this illustrates the effect of time warp by temporarily freezing the rendered eye pose.
Reset Tracker On Load	This value defaults to **True**. When turned off, subsequent scene loads will not reset the tracker. This will keep the tracker orientation the same from scene to scene, as well as keep magnetometer settings intact.
Monoscopic	If true, rendering will try to optimize for a single viewpoint rather than rendering once for each eye. Not supported on all platforms.
Eye Texture Format	Sets the format of the eye RenderTextures. Normally you should use Default or DefaultHDR for high-dynamic range rendering.
Eye Texture Depth	Sets the depth precision of the eye RenderTextures. May fix z-fighting artifacts at the expense of performance.

4.3.3 Helper Classes

In addition to the above components, your scripts can always access the HMD state via static members of OVRManager.

OVRDisplay Provides the pose and rendering state of the HMD.

OVRTracker Provides the pose, frustum, and tracking status of the infrared tracking camera.

4.3.4 OvrCapi

OvrCapi is a C# wrapper for LibOVR (specifically, CAPI). It exposes all device functionality, allowing you to query and set capabilities for tracking, rendering, and more. Please refer to the Oculus Developer Guide and reference manual for details.

OVRCommon OVRCommon is a collection of reusable static functions, including conversions between Unity and OvrCapi types.

4.3.5 Utilities

The following classes are optional. We provide them to help you make the most of virtual reality, depending on the needs of your application.

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

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

OVRPlayerController contains a few variables attached to sliders that change the physics properties of the controller. This includes **Acceleration** (how fast the player will increase speed), **Dampening** (how fast a player will decrease speed when movement input is not activated), **Back and Side Dampen** (how much to reduce side and back Acceleration), **Rotation Amount** (the amount in degrees per frame to rotate the user in the Y axis) and **Gravity Modifier** (how fast to accelerate player down when in the air). When **HMD Rotates Y** is set, the actual Y rotation of the cameras will set the Y rotation value of the parent transform that it is attached to.

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.

OVRGamepadController OVRGamepadController is an interface class to a gamepad controller.

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

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

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

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 both OVRCameraRig and OVRPlayerController prefabs for convenience, but can be safely removed from them if the functionality is not needed.

OVRCrosshair OVRCrosshair is a helper class that renders and controls an on-screen cross-hair. It is currently used by the OVRMainMenu component.

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).

OVRGridCube OVRGridCube is a helper class that shows a grid of cubes when activated. Its main purpose is to be used as a way to know where the ideal center of location is for the user's eye position. This is especially useful when positional tracking is activated. The cubes will change color

to red when positional data is available, and will remain blue if position tracking is not available, or change back to blue if vision is lost.

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.

OVRVisionGuide OVRVisionGuide implements a helper class that can be used to let a user know if they are falling outside of a safe vision zone. This is especially useful when positional data is available.

4.4 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.

4.4.1 Targeting a Display

To run your application on the Rift in full-screen mode, use the ≪AppName≫_DirectToRift.exe file located next to your standard binary. It works in direct and extended modes. You should include both of these files and the ≪AppName≫_Data folder when publishing builds.

To enable Rift support, the internal OVRShimLoader script forces your builds to 1920x1080 full-screen resolution and suppresses Unity's start-up splash screen by default. You can still access it and change the settings when running your plain executable (≪AppName≫.exe) by holding the ALT key immediately after launch. To disable this behavior, navigate to Edit ≫ Preferences... ≫ Oculus VR and uncheck the "Optimize Builds for Rift" box.

4.4.2 Direct-Mode Display Driver

When the driver is in direct mode, Rift applications run in a window and are mirrored to the Rift. You can disable this behavior by turning off OVRManager.mirrorToMainWindow.

4.4.3 Editor Workflow

If you plan to run your application in the Unity editor, you must use extended mode. A black screen will appear if you run it in direct mode.

The "Build & Run" option (CTRL + B) is not recommended in any driver mode. We recommend you build a standalone player without running it and then run the ≪AppName≫_DirectToRift.exe file produced by the build.

4.4.4 Other Platforms

Linux support will be restored in the near future.

5 Migrating From Earlier Versions

The 0.4.3 Unity Integration's API is significantly different from 0.4.2 and prior versions. This section will help you upgrade.

5.1 API Changes

5.1.1 Unity Components

OVRDevice → OVRManager	Unity foundation singleton.
OVRCameraController → OVRCameraRig	Performs tracking and stereo rendering.
OVRCamera	Removed. Use eye anchor Transforms instead.

5.1.2 Helper Classes

OVRDisplay	HMD pose and rendering status.
OVRTracker	Infrared tracking camera pose and status.
OVR.Hmd → Ovr.Hmd	Pure C# wrapper for LibOVR.

5.1.3 Events

HMD added/removed	Fired from OVRCameraRig.Update() on HMD connect and disconnect.
Tracking acquired/lost	Fired from OVRCameraRig.Update() when entering and exiting camera view.
HSWDismissed	Fired from OVRCameraRig.Update() when the Health and Safety Warning is no longer visible.
Get/Set*(ref *) methods	Replaced by properties.

5.2 Behavior Changes

- OVRCameraRigs position is always the initial center eye position.
- Eye anchor Transforms are tracked in OVRCameraRigs local space.
- OVRPlayerControllers position is always at the users feet.
- IPD and FOV are fully determined by profile.
- Layered rendering: multiple OVRCameraRigs are fully supported.
- OVRCameraRig.*EyeAnchor Transforms give the relevant poses.

5.3 Upgrade Procedure

1. Ensure you didnt modify the structure of the OVRCameraController prefab. If your eye cameras are on GameObjects named CameraLeft and CameraRight which are children of the OVRCameraController GameObject (the default), then the prefab should cleanly upgrade to OVRCameraRig and continue to work properly with the new integration.

2. Write down your settings from the inspectors for OVRCameraController, OVRPlayerController, and OVRDevice. You will have to re-apply them later.

3. Remove the old integration by deleting the following from your project:

    - OVR folder
    - Any file in the Plugins folder with Oculus or OVR in the name
    - Moonlight folder (if applicable)
    - OVR_Internal folder (if applicable)

4. Import the new integration

5. Click Assets ¿ Import Package ¿ Custom Package

6. Open OculusUnityIntegration.unitypackage

7. Click Import All

8. Fix any compiler errors in your scripts. Refer to the API Changes and Behavior Changes sections above.

9. Re-apply your previous settings to OVRCameraRig, OVRPlayerController, and OVRManager.

6 Final Notes

We hope you find making VR worlds within Unity as fun and inspiring as we did 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