Oculus Unity Guide

Version 1.13.0

Copyrights and Trademarks

Contents

Getting Started Guide

This guide describes initial setup, importing the optional Utilities for Unity package, and building Oculus apps using Unity's first-party support.

Topics include:

- Information on compatibility between Unity Editor versions and Utilities for Unity version
- Environment setup for Rift or mobile development
- How to build Rift or mobile applications from the Unity editor
- A simple Hello World! Unity VR tutorial
- Other Oculus-provided resources and tools for Unity developers
- A FAQ to help you get started.

Compatibility and Version Requirements

System and Hardware Requirements

To verify that you are using supported hardware, please review the relevant PC or mobile setup documentation:

- PC SDK: *Getting Started with the SDK*
- Mobile SDK: *System and Hardware Requirements*

Unity Version Compatibility

Unity 5.1 or later provide built-in support for Rift and Gear VR development. The optional Oculus Utilities for Unity package offers additional developer resources.

All developers should use Unity 5. Legacy support is available for Unity 4 - see our *Unity 4.x Legacy Integration Developer Guide* on page 106 for more information.

Unity Version	Recommended Release
Unity 5.4	Unity 5.4.5f1 and later
Unity 5.5	Unity 5.5.2p3 and later
Unity 5.6	Unity 5.6.0f3 and later
Unity 4	Unity 4.7.0f1 (legacy support only)

For complete details Oculus SDK or Integration version compatibility with Unity, see *Unity-SDK Version Compatibility*.

OS Compatibility

- **Windows**: Windows 7, 8, 10
- **Mac**: OS X Yosemite, El Capitan, Sierra

OS X development requires the Oculus Rift Runtime for OS X, available from our *Downloads page*. Note that runtime support for OS X is legacy only. It does not support consumer versions of Rift.

Unity Personal and Professional Licenses

The Unity Personal and Professional licenses both provide built-in Rift support. Mobile developers using the Unity Free license receive basic Android support automatically. Mobile developers using a Professional license require an Android Pro Unity license.

For more information, see *License Comparisons* in Unity's documentation.

Controllers

You may wish to have a controller for development or to use with the supplied demo applications. Available controllers include the Oculus Touch or Xbox 360 controller for Rift, and the Gear VR Controller for mobile development.

Preparing for Rift Development

Unity 5.1 or later offers built-in Rift support. The Oculus SDK is not required.

To enable VR support is enabled in the Unity Editor, check the *Virtual Reality Supported* checkbox in *Player Settings*. Applications targeting the *PC, Mac & Linux* platform in *Build Settings* will now run on the Rift.

Unity automatically applies position and orientation tracking, stereoscopic rendering, and distortion correction to your main camera when VR support is enabled. For more details, see *Unity VR Support* on page 26.

If you have already set up your Oculus Rift for regular use, you are ready to begin Unity development.

Unity development for Rift requires the Oculus App (the PC application normally installed during Rift setup). If you have not already installed it, download it from the *Setup page* and install it. When Unity requires the Oculus runtime, it will automatically launch the Oculus App.

You may develop Rift apps on PCs that do not meet our minimum specifications or have a Rift connected. We strongly recommend having a Rift available for development to preview your apps as you go, but it is not required as long as the Oculus App is installed.

Advanced developers may find it useful to review our Oculus SDK documentation for more insight into the rendering pipeline and underlying logic. If that interests you, we recommend *Intro to VR* and the *PC Developer Guide* for a deeper dive into core Rift development concepts.

If you are interested in submitting an application to the Oculus Store, please see our *Publishing Guide*. We recommend doing so before beginning development in earnest so you have a realistic sense of our guidelines and requirements.

Preparing for Mobile Development

To prepare for Unity mobile development for Gear VR, you must set up the Unity Editor for Android development and install the Android SDK. The Oculus Mobile SDK is not required.

Note: If you are developing with a Unity Professional license, you will need an Android Pro Unity license to build Android applications with Unity. The Free license includes basic Android support. For more information, see *License Comparisons* Unity's documentation.

We recommend reviewing Unity's *Getting started with Android development* for general information on Android development, but the essential setup steps are described below.

Once you have set up the Unity Editor for Android development, VR support is enabled by checking the *Virtual Reality Supported* checkbox in *Player Settings*. Applications targeting the Android Platform will then run on Gear VR.

Unity automatically applies orientation tracking, stereoscopic rendering, and distortion correction to your main camera when VR support is enabled. For more details, see *Unity VR Support*.

If you are already prepared for Unity Android development, you are nearly ready to begin developing for Gear VR.

Mobile Device Setup

Follow the instructions in the *Device Setup* section of our Mobile SDK Developer Guide to prepare to run, debug, and test your Gear VR applications.

Android SDK

The Android SDK is required for mobile development with Unity. For setup instructions, see *Standalone Android SDK Tool (Windows)* or *Standalone Android SDK Tool (OS X)* in our Mobile SDK Developer Guide. Most Unity developers do not need to install Android Studio or NDK.

Once you have installed the Standalone Android SDK tools, you may continue with this guide.

Once you have installed the Android SDK, you may wish to familiarize yourself with adb (Android Debug Bridge), a useful tool used for communicating with your Android phone. For more information, see *Adb* in our Mobile Developer Guide.

Sign your App with an Oculus Signature File

All Gear VR applications must be signed with an Oculus Signature File (osig) during development to access low-level VR functionality on your mobile device. This signature comes in the form of an Oculus-issued file that you include in your application.

Each signature file is associated with a specific mobile device, so you will need an osig file for each device that you use for development.

Please see our osig self-service portal for more information and instructions on how to request an osig for development: *https://dashboard.oculus.com/tools/osig-generator/*

Once you have downloaded an osig, be sure to keep a copy in a convenient location for reuse. You will only need one osig per device for any number of applications.

Make a copy of the osig and copy it to the following directory:

```
<project>/Assets/Plugins/Android/assets/
```

If that directory does not exist, create it and copy the osig file to it. Note that the folder names are **caps sensitive** and must be exactly as stated above.

If you attempt to run a Gear VR APK that has not been correctly signed with an osig, you will get the error message "thread priority security exception make sure the apk is signed".

We recommend removing your osig before building a final APK for submission to the Oculus Store. When your application is approved, Oculus will modify the APK so that it can be used on all devices. See *Building Mobile Apps for Submission to the Oculus Store* on page 17 for more information.

Application Entitlement Checking

Entitlement checking is used to protect apps from unauthorized distribution. It is disabled by default in Unity. Entitlement checking is not required for development, but it is required for submitting an application to the Oculus Store.

For more information and instructions, see *Getting Started with the SDK* in our Platform guide.

Getting Started with Mobile Development

Mobile applications are subject to more stringent limitations and requirements and computational limitations than Rift applications.

We strongly recommend carefully reviewing *Mobile Development* on page 54 and *Best Practices for Rift and Gear VR* on page 62 in our Developer Guide to be sure you understand our performance targets and recommendations for mobile development. These sections contain important information that can help you avoid mistakes that we see frequently.

Previewing Gear VR apps in Rift

You may find it useful to preview mobile applications using the Oculus Rift during development, but use caution when doing so. The rendering path for Android applications differs substantially from the rendering path used for Rift application previews and builds, and you may notice important differences in the look-and-feel and performance.

Designing Apps for Both Rift and Mobile

When developing for both Rift and mobile platforms, keep in mind that their requirements differ substantially. If you would like to generate builds for both PC and mobile from a single project, it is important to follow the more stringent mobile development best practices, as well as meeting the required 90 FPS required by the Rift. This approach is not often taken in practice.

Additional Sources of Information

For information on Oculus tools to assist with mobile development, such as our Sample Framework or the Oculus Remote Monitor for performance debugging, please see *Other Oculus Resources for Unity Developers* on page 24.

For information on core VR development concepts, see the *Intro to VR* in our PC Developer Guide. It is written from the perspective of Rift development, but many contents apply equally well to mobile design.

Most Unity developers do not need to install the Oculus Mobile SDK. However, advanced developers may find it useful to review our Mobile SDK Developer Guide for insight into the underlying logic. Developers interested in the Android lifecycle and rendering path of Oculus mobile applications should review our documentation on *VrApi*. *Mobile Best Practices* and *General Recommendations* may also be of interest.

If you are interested in submitting an application to the Oculus Store, please see our *Publishing Guide*. We recommend doing so before beginning development in earnest so you have a realistic sense of our guidelines and requirements.

Importing the Oculus Utilities Package

Oculus Utilities for Unity 5 is an optional Unity Package that includes scripts, prefabs, and other resources to assist with development.

For more details, see *Oculus Utilities for Unity* on page 26.

Importing Oculus Utilities for Unity 5 has three steps:

1. If you have previously imported an earlier Utilities version into your project, you must delete its content before importing a new version.
2. Open the project you would like to import the Utilities into, or create a new project.
3. Import the Utilities Unity Package.

1. Delete Previously-Imported Assets If Necessary

If you have previously imported another version of the Utilities package, delete all Oculus content from your Unity project before importing the new version.

Be sure to close the Unity Editor, navigate to your Unity project folder, and delete the following:

Folder	Content to Delete
Assets/Plugins	Oculus.* OVR.*
Assets/Plugins/ Android/	*Oculus* AndroidManifest.xml *vrapi* *vrlib* *vrplatlib*
Assets/Plugins/x86/	Oculus.* OVR.*
Assets/Plugins/ x86_64/	Oculus.* OVR.*

Note: Be sure to delete OculusPlugin.dll in Assets/Plugins/x86_64 and OculusUnityPatcher*.exe in Assets/OVR/Editor in addition to all other specified files.

2. Open or Create a Unity Project

If you are already working in a Unity project, save your work before beginning. Otherwise, create a new project:

1. From the Unity menu, select *File > New Project*.
2. Click the *Browse* button and select the folder where the Unity project will be located.
3. Make sure that the *Setup defaults for:* field is set to *3D*.
4. You do not need to import any standard or pro Unity asset packages, as the Oculus Utilities package is fully self-contained.
5. Click the *Create* button. Unity will reopen with the new project loaded.

3. Import the Unity Package

Select *Assets > Custom Package...* and select the Utilities Unity Package to import it.

Alternately, you can locate the .unityPackage file in your file system and double-click it to launch.

When the *Importing package* dialog box opens, leave all of the boxes checked and select *Import*. The import process may take a few minutes to complete.

Building Rift Applications

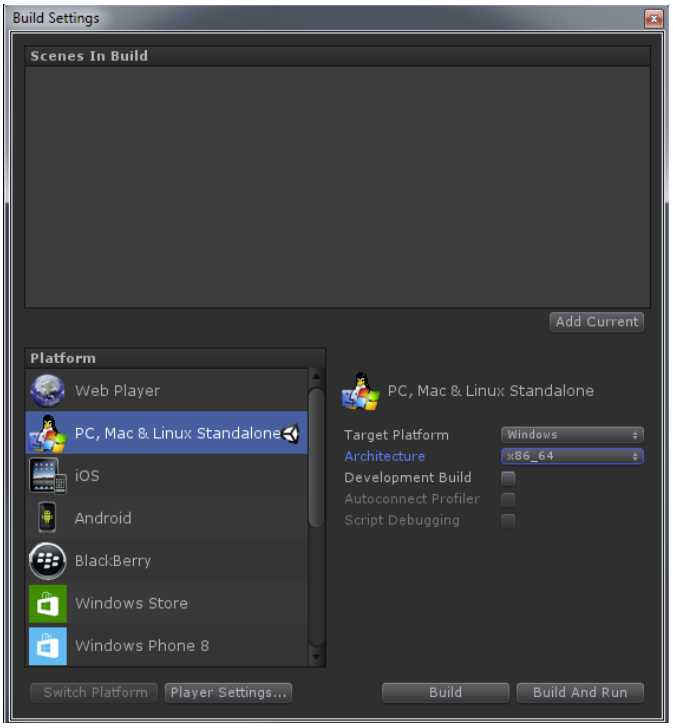
This section describes the steps necessary for building Rift apps in Unity.

Build Settings

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

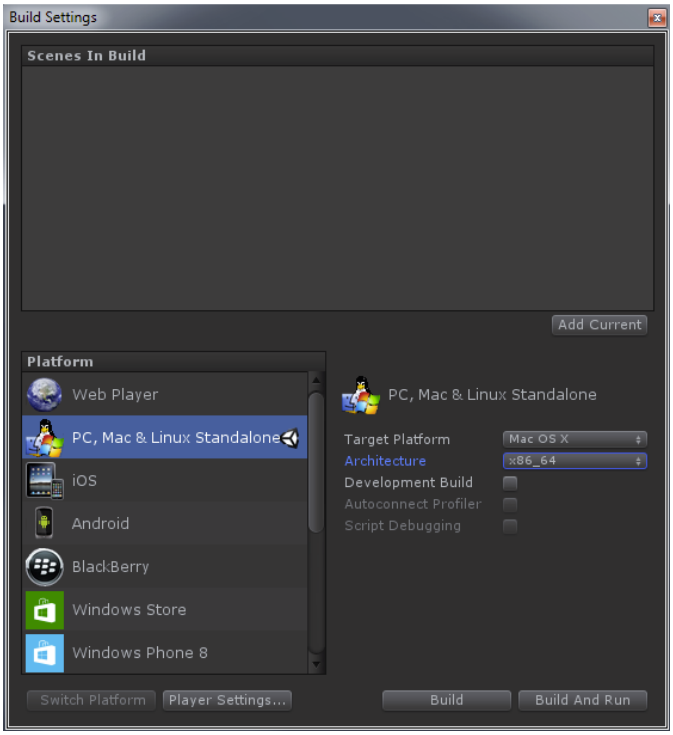
For Windows, set *Target Platform* to *Windows* and set *Architecture* to either *x86* or *x86 64*.

Figure 1: Build Settings: PC



For Mac, set *Target Platform* to *Mac OS X*.

Figure 2: Build Settings: Mac

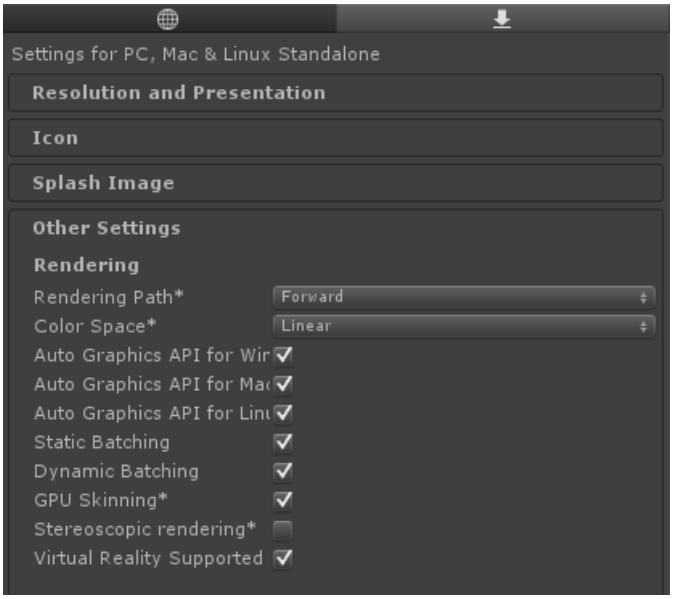


Note: Be sure to add any scenes you wish to include in your build to *Scenes In Build*..

Player Settings

Within the *Build Settings* pop-up, click *Player Settings*. In the *Other Settings* frame, verify that *Virtual Reality Supported* is checked. All additional required settings are enforced automatically.

Figure 3: Player Settings



Quality Settings

Navigate to *Edit > Project Settings > Quality*. We recommend the following settings:

Pixel Light Count	3
Texture Quality	Full Res
Anisotropic Textures	Per Texture
Anti Aliasing	2x or 4x Multi Sampling
Soft Particles	Deselect
Realtime Reflections Probes	Select
Bilboards Face Camera	Select

The *Anti-aliasing* setting is particularly important. It must be increased to compensate for stereo rendering, which reduces the effective horizontal resolution by 50%. An anti-aliasing value of 2X is ideal, 4x may be used if you have performance to spare. We do not recommend 8x.

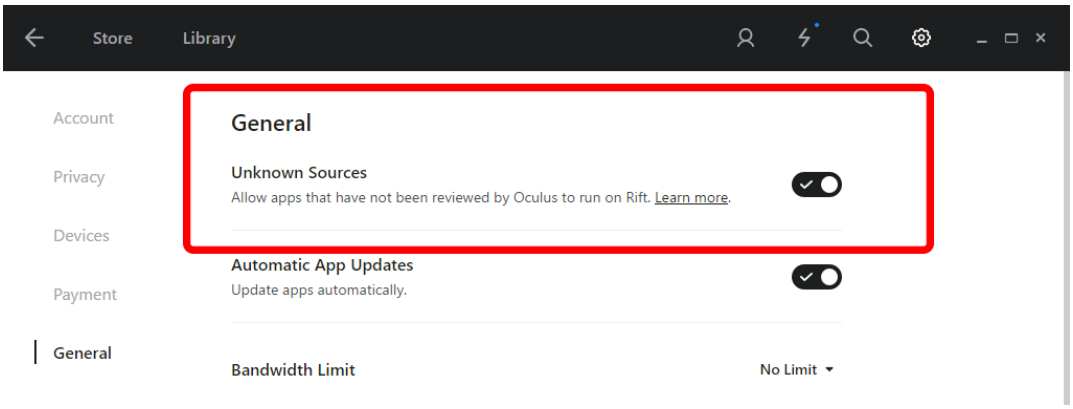
For more information on our recommended settings, see *Best Practices for Rift and Gear VR* on page 62.

Build and Run the Application

In the *Build Settings* dialog, click select *Build*. If prompted, specify a name and location for the build.

To run your application, you must allow apps that have not been reviewed by Oculus to run on your Rift:

1. Launch the Oculus App
2. Click the "gear" icon in the upper-right
3. Select *Settings* > *General* and set *Unknown Sources* to allow. When prompted for confirmation, select *Allow* (check mark).



You may wish to disable the *Unknown Sources* option when you are not doing development work.

To run your application, navigate to the target folder of your build and launch the executable.

Building Mobile Applications

This section describes the steps necessary for building Gear VR apps in Unity.

Android Manifest

During the build process, Unity projects with VR support enabled are packaged with an automatically-generated manifest which is configured to meet our requirements (landscape orientation, vr_only, et cetera). All other values, such as Entitlement Check settings, are not modified. **Do not** add the noHistory attribute to your manifest.

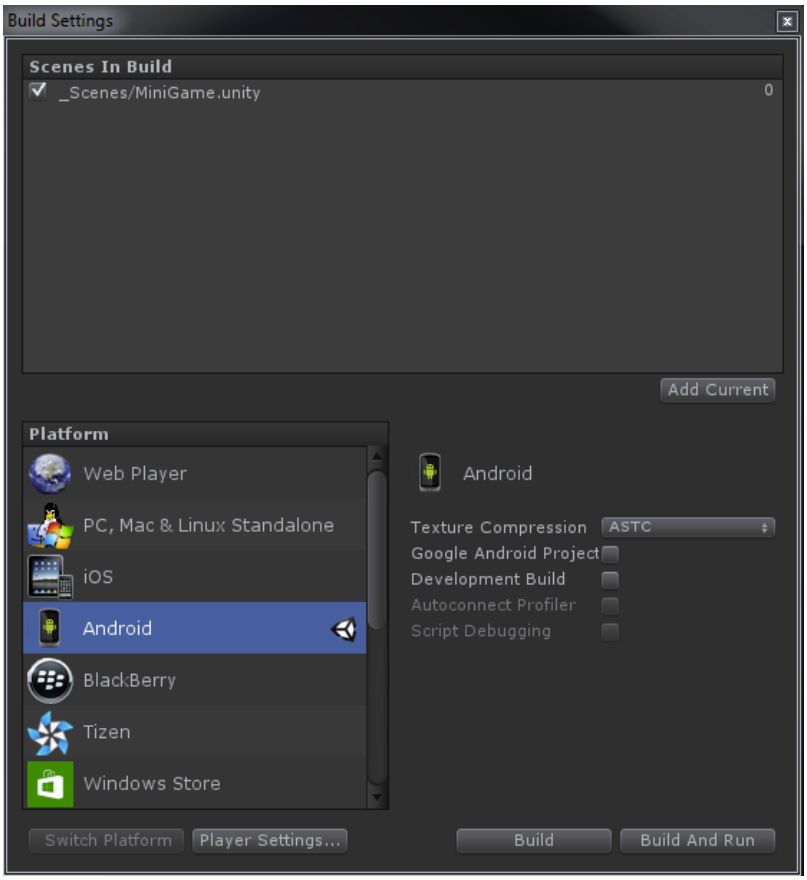
To build an application for submission to the Oculus Store, you must build a custom manifest using the Oculus Utilities for Unity. See *Building Mobile Apps for Submission to the Oculus Store* on page 17 for details.

Oculus Signature File

Your application must include an Oculus Signature File or osig. See "Sign your App with an Oculus Signature File" in *Preparing for Mobile Development* on page 6 for more information.

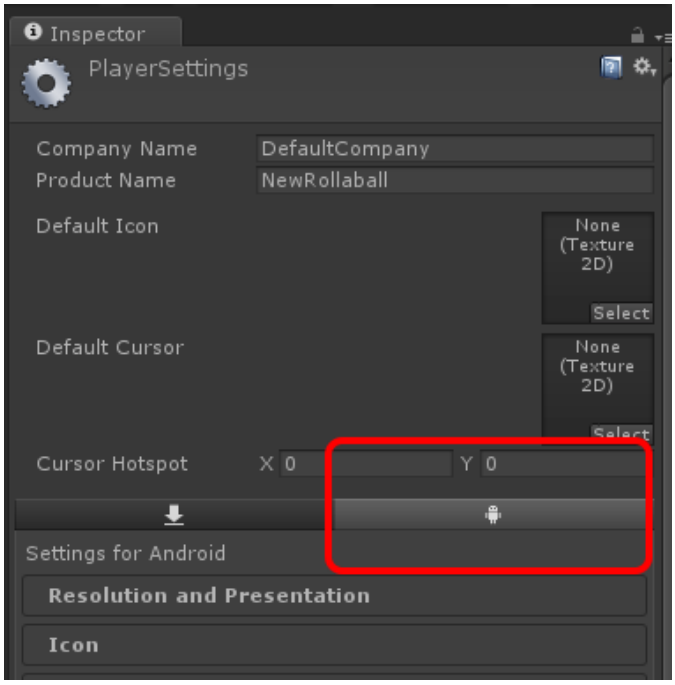
Build Settings

From the *File* menu, select *Build Settings…*. Select *Android* as the platform. Set *Texture Compression* to *ASTC*.



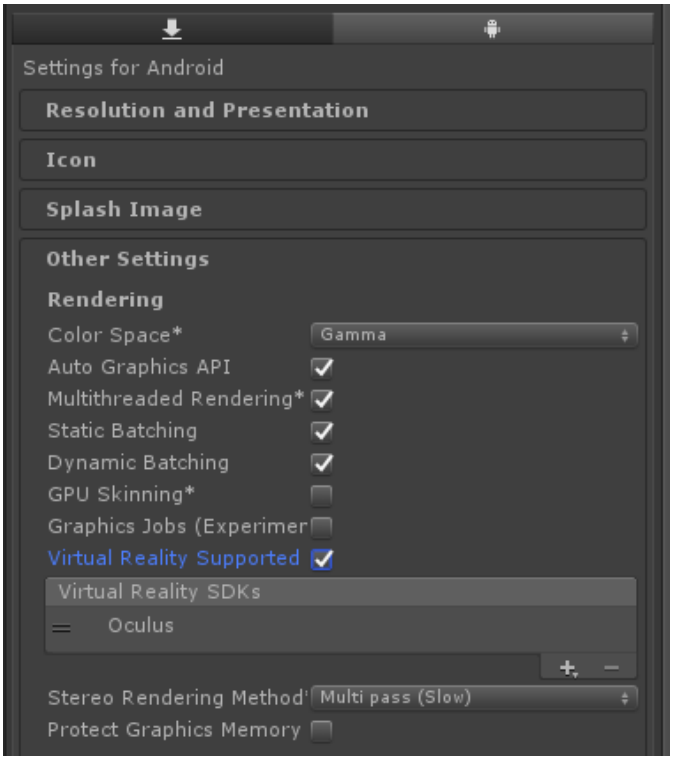
Player Settings

1. Click the *Player Settings...* button and select the *Android* tab.



2. Set a Bundle Identifier under *Identification* in *Other Settings*.
3. Select *Virtual Reality Supported* under *Rendering* in *Other Settings*.

All required settings are enforced automatically, but you may wish to make additional settings as appropriate, such as enabling *Multithreaded Rendering*. For more information on our recommended settings, see the *Best Practices for Rift and Gear VR* on page 62 section.



Quality Settings

Navigate to *Edit* > *Project Settings* > *Quality*. We recommend the following settings:

Pixel Light Count	1
Texture Quality	Full Res
Anisotropic Textures	Per Texture
Anti Aliasing	2x or 4x Multi Sampling
Soft Particles	Deselect
Realtime Reflections Probes	Select
Bilboards Face Camera	Select

The *Anti-aliasing* setting is particularly important. It must be increased to compensate for stereo rendering, which reduces the effective horizontal resolution by 50%. An anti-aliasing value of 2X is ideal, 4x may be used if you have performance to spare. We do not recommend 8x.

For more information on our recommended settings, see *Best Practices for Rift and Gear VR* on page 62.

Build and Run the Application

Build your application for the Android Platform and load it onto your phone. When you launch the application, you will be prompted to insert your phone into the Gear VR headset.

1. Save the project before continuing. If you are not already connected to your phone via USB, connect now. Unlock the phone lock screen.
2. On some Samsung models, you must set the default USB connection from *Connected for charging* or similar to *Software installation* or similar in the Samsung pulldown menu.
3. From the *File* menu in the Unity Editor, select *Build Settings…*. While in the Build Settings menu, add your scenes to *Scenes in Build* if necessary.
4. Verify that Android is selected as your *Target Platform* and select *Build and Run*. If asked, specify a name and location for the APK.
5. The APK will be installed and launched on your Android device.

To run your application later, remove your phone from the Gear VR headset and launch the app from the phone desktop or Apps folder. Then insert the device into the Gear VR when prompted to do so.

Note that your will not see your application listed in your Oculus Home Library - only applications approved and published by Oculus are visible there.

Sideloading Unity Applications

You may build an APK locally and sideload it to your phone.

To do so, follow all of the instructions above, except in the final steps, select *Build* instead of *Build and Run*.

Once you have built an APK on your local system, you may copy it to your phone by following the instructions in *Using adb to Install Applications* in our Mobile SDK Developer Guide.

Running Mobile Apps Outside of the Gear VR Headset

When Developer Mode is enabled on your phone, Oculus applications will run without inserting the phone into the Gear VR headset. This can be convenient during development.

VR applications run in Developer Mode play with distortion and stereoscopic rendering applied, and with limited orientation tracking using the phone's sensors.

For instructions on setting your device to Developer Mode, see *Developer Mode: Running Apps Outside of the Gear VR Headset* in our Mobile SDK Developer Guide.

Building Mobile Apps for Submission to the Oculus Store

If you are building an application for the Oculus Store, you will need to take a few extra steps.

1. Create a custom manifest using the Oculus Utilities for Unity
2. Remove your osig
3. Sign your application with an Android Keystore
4. Enable Entitlement Checking with the Oculus Platform SDK

For more information on the submission process, see our Publishing Guide.

1. Build a Custom Manifest for Submission

To build an APK that includes a manifest with the values required by the Oculus Store, you must download the Oculus Utilities for Unity 5 package and import it into your project as described in *Importing the Oculus Utilities Package* on page 8.

Once you have a done so, in the Unity Editor, select *Tools > Oculus > Create store-compatible AndroidManfiest.xml*. Then build your project normally.

2. Remove your osig

We recommend removing your Oculus Signature File before building your application for submission.

3. Sign your application with an Android Keystore

Mobile applications must be signed with an Android signature for submission. Unity automatically signs applications with a temporary Android debug certificate by default. Before building your final release build, create a new Android keystore by following the "Sign Your App Manually" instructions in Android's *Sign your Applications* guide. Then assign it with the *Use Existing Keystore* option, found in *Edit > Project Settings > Player > Publishing Options*.

For more information, see the *Android section* of Unity's documentation.

4. Enable Entitlement Checking with the Oculus Platform SDK

Entitlement checking is used to protect apps from unauthorized distribution. It is disabled by default in Unity. Entitlement checking is not required for development, but it is required for submitting an application to the Oculus Store.

For more information and instructions, see *Getting Started with the SDK* in our Platform guide.

Tutorial: Build Your First VR App

This short tutorial walks you through creating and running a simple Unity app for Rift or Gear VR.

When you finish, you will have a working VR application that you can play on your Rift or Gear VR device, to the amazement of your friends and loved ones.

Requirements

- Unity 5 (see *Compatibility and Version Requirements* on page 5 for version recommendations)
- Rift or Gear VR
- Compatible gamepad: optional for Rift, but a Bluetooth gamepad is required to control the player on Gear VR.

Preparation

Before beginning, you will need to set up your development environment.

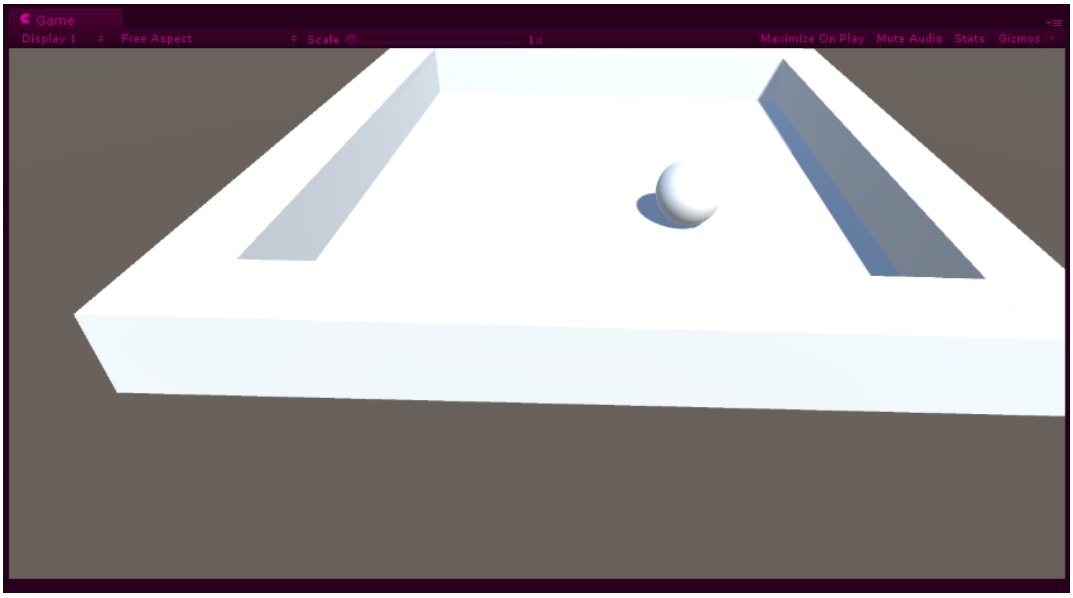
If you are building for Rift, please follow the instructions in *Preparing for Rift Development* on page 6. Be sure to configure the Oculus App to run apps from unknown sources, as described in that section.

If you are building for Gear VR, please follow the instructions in *Preparing for Mobile Development* on page 6. You should be able to communicate with your Samsung phone via USB before beginning. To verify this, retrieve the device ID from your phone by connecting via USB and sending the command `adb devices` from a command prompt. The phone should return its device ID. Make a note of it - you will need it later to request a Oculus Signature File.

Build Your Simple Application

We are going to build a simple game with a play area that consists of a floor and some walls. Then we'll add a ball and attach a controller script so we can move it around the play area with a keyboard or gamepad. Finally, we'll add VR support so we can run the game on Rift or Gear VR.

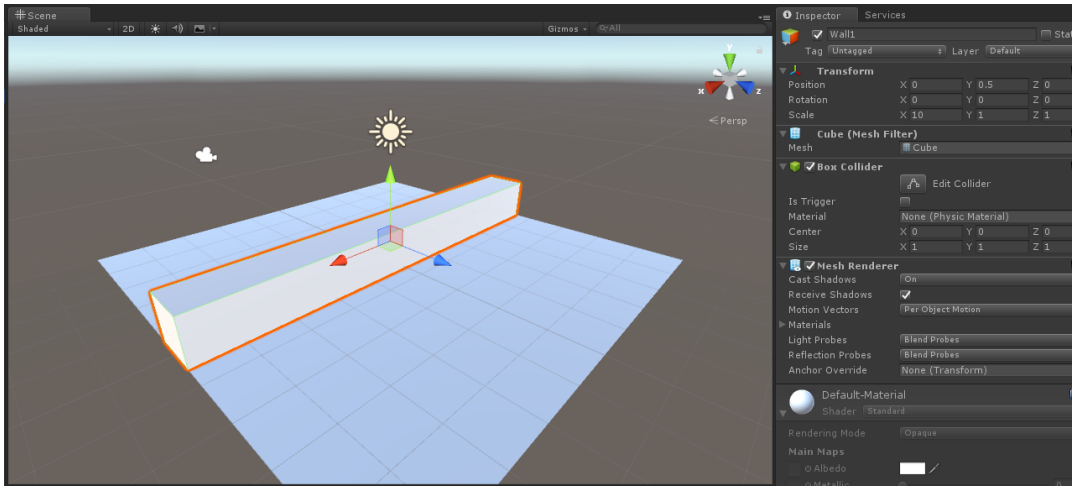
So let's get started without any further ado.



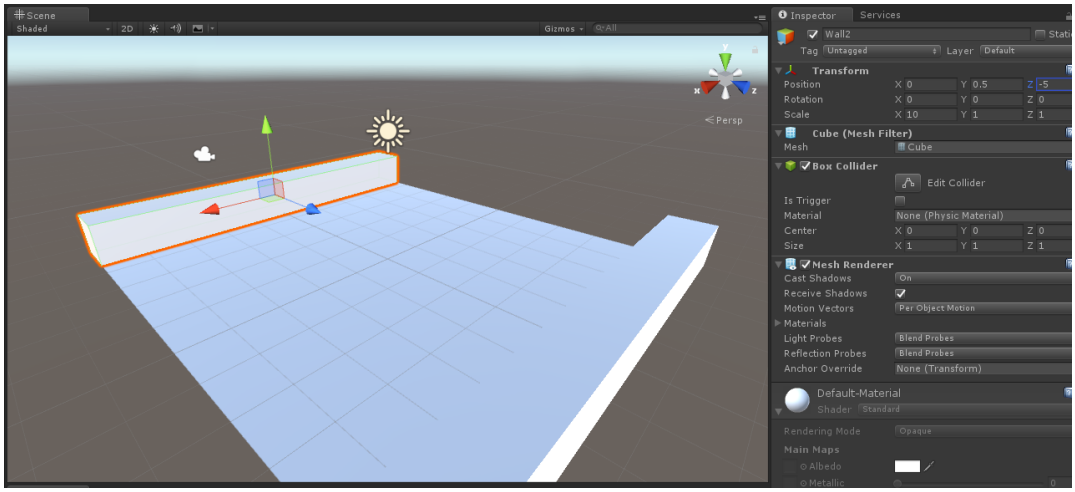
Part 1: Build a Play Area and Add a Player

In this part of the tutorial, we'll build a simple play area consisting of a floor and four walls, and add a sphere as a player.

1. Launch the Unity Editor and, in the initial launch dialog, create a new project by clicking *New*. Give it a creative name like VRProject, and select a location for the files to live in. Verify that the 3D option is selected, and click *Create Project*.
2. Save the Scene.
    a. Select the File pulldown menu, and click *Save Scene as…*.
    b. Give the scene a creative name like VRScene.
3. Let's create a floor.
    a. In the GameObject pulldown menu, select *3D Object* > *Plane*.
    b. In the Inspector, verify that the Position is set to origin (0,0,0). If not, set it to origin by selecting the gear icon pulldown in the upper-right of the Transform section of the Inspector and clicking *Reset*.
    c. Find Plane in the Hierarchy View and right-click it. Select *Rename* in the context menu, and rename it *Floor*.
4. Now we'll create the first wall.
    a. In the GameObject pulldown menu, select *3D Object* > *Cube*.
    b. If the cube isn't at origin, reset its Transform like we did in the preceding step.
    c. Longify the cube by setting the X value of Scale to *10* under Transform.
    d. Move it up slightly by setting the Y value of Position to *.5* under Transform. It should now rest upon the floor.
    e. Find Cube in the Hierarchy View and right-click it. Select *Rename* in the context menu, and name it *Wall1*.



5. Move the wall to the outer edge of the play area.
    a. Select Wall1 in the Hierarchy View or in the Scene View.
    b. In the Inspector, set the Z value of Position to *5* under Transform.
6. Make a second wall and move it to the opposite edge of the play area.
    a. Right-click Wall1 in the Hierarchy View and select *Duplicate* in the context menu. You will see a second wall named *Wall1 (1)*.
    b. Right-click Wall1 (1) in the Hierarchy View. Select *Rename* in the context menu, and rename it *Wall2*.
    c. Select Wall2 in the Hierarchy View or in the Scene View.
    d. In the Inspector, set the Z value of Position to *-5* under Transform.

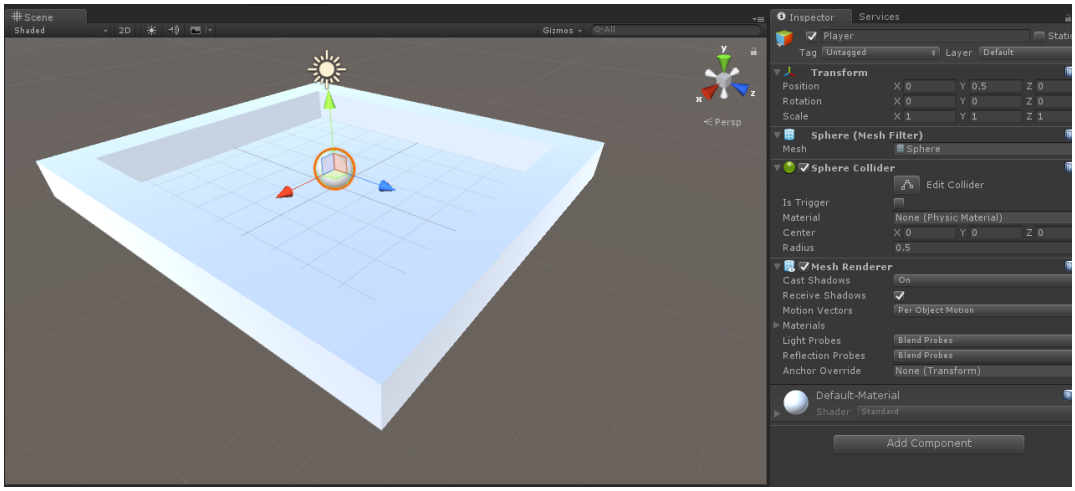


7. Make a third wall, rotate it, and move it into place.

    a. Right-click Wall1 in the Hierarchy View and select *Duplicate* in the context menu. You will see a wall named *Wall1 (1)*.
    b. Right-click Wall1 (1) in the Hierarchy View and select *Rename* in the context menu, and rename it *Wall3*.
    c. Select Wall3 in the Hierarchy View or in the Scene View.
    d. In the Inspector, set the *Y* value of Rotation to *90* under Transform.
    e. Select Wall3 in the Hierarchy View or in the Scene View.
    f. In the Inspector, set the X value to *4.5* and the Z value to *0* in Position, under Transform.
8. Make a fourth wall and move it into place.

    a. Right-click Wall3 in the Hierarchy View and select *Duplicate* in the context menu. You will see a wall named *Wall3 (1)*.
    b. Right-click Wall3 (1) in the Hierarchy View and select *Rename* in the context menu. Name it *Wall4*.
    c. Select Wall4 in the Hierarchy View or in the Scene View.
    d. In the Inspector, set the X value of Position to *-4.5* under Transform.
9. Now we have a play area with a floor surrounded by walls. Let's add a sphere player.

    a. In the GameObject pulldown menu, select *3D Object > Sphere*.
    b. In the Inspector, set the Position to *0, .5, 0*.
    c. Right-click Sphere in the Hierarchy View and select *Rename* in the context menu. Name it *Player*.



10. Adjust your camera so we can see the play area better.
    a. Select Main Camera in the Hierarchy View.
    b. In the Inspector, set Position to *0, 5, 0* and Rotation to *20, 0, 0* under Transform.

Part 2: Add a control script to your Player

In this part of the tutorial, we will prepare the Player so we can control its movement programmatically, based on user input from keyboard or gamepad.

1. Add a RigidBody component to the Player. This will allow us to move it (for more details, *RigidBody* in Unity's manual).
    a. Select Player in the Hierarchy View or in the Scene View.
    b. In the Inspector, click Add Component at the bottom. Select the *Physics* category, then select *RigidBody*.
2. Create a new script, which we will use to control the Player.
    a. Select Player in the Hierarchy View or in the Scene View.
    b. In the Inspector, click Add Component at the bottom. Select the *New Script* category.
    c. Set the name of our new script to PlayerController, and set the language to *C Sharp*. Click *Create and Add*.
    d. Right-click Player Controller (Script) in the Inspector, and select *Edit Script* in the context menu. This will launch the editor Unity associates with editing C# scripts (MonoDevelop by default). You should see the following:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class temp : MonoBehaviour {

// Use this for initialization
void Start () {

}

// Update is called once per frame
void Update () {

}
}
```

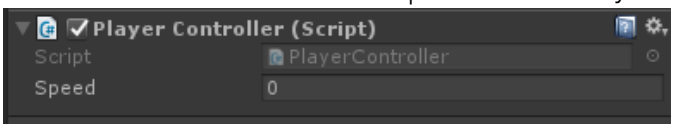
3. Add a new function to move your Player. Add the text in bold:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class temp : MonoBehaviour {

// Use this for initialization
void Start () {

}

// Update is called once per frame
void Update () {

 // get input data from keyboard or controller
float moveHorizontal = Input.GetAxis("Horizontal");
float moveVertical = Input.GetAxis("Vertical");

// update player position based on input
Vector3 position = transform.position;
position.x += moveHorizontal * speed * Time.deltaTime;
position.z += moveVertical * speed * Time.deltaTime;
transform.position = position;
```

```
    }
}
```

4. Attach the Player Controller script to your Player.

    a. Select Player in the Hierarchy View or in the Scene View.
    b. In the Project View, select *All Scripts*. You should see PlayerController on the right side of the Project View.
    c. Click on PlayerController in the Project View and drag it into the Inspector, below *Add Component*. You should see it added as a new component to the Player.



    d. Set speed to *3*.

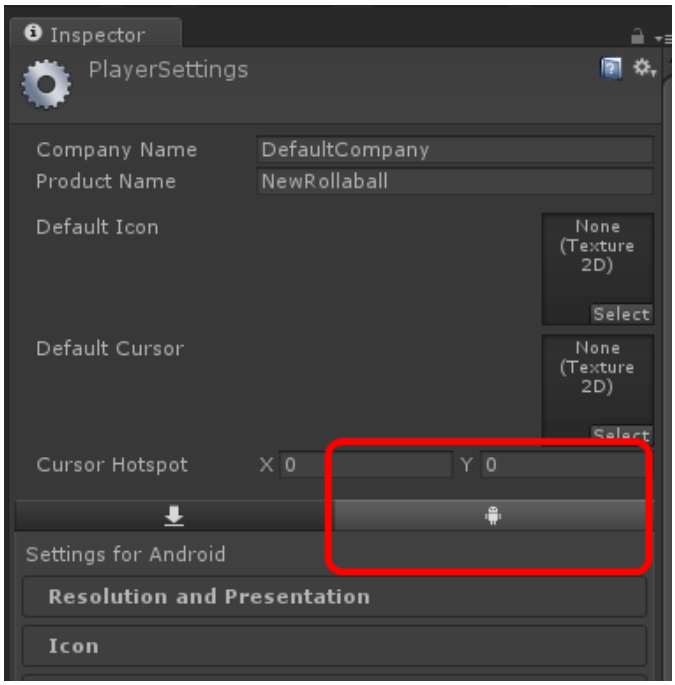
At this point if you preview your game in the Game View by pressing the Play button, you'll find you can control the Player with the arrow keys or W-A-S-D on your keyboard. If you have a Unity-compatible gamepad controller you can control it with a joystick. Try it out!

Part 3: Modify your project for VR

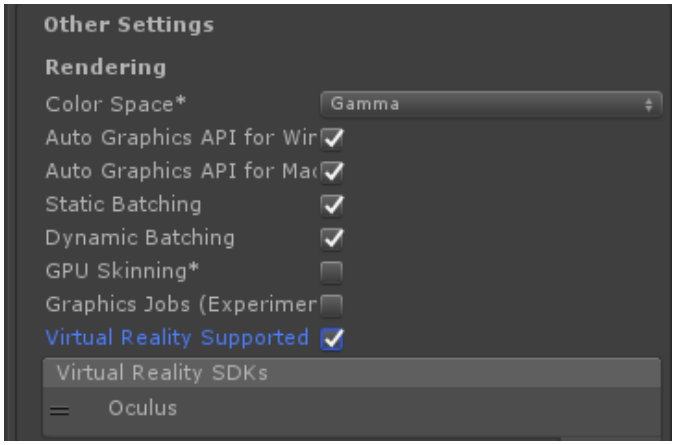
In this step we'll enable VR support in Player Settings.

1. In the Edit menu, select *Project Settings* > *Player*.
2. In the Inspector window, locate the platform selection tabs. If developing for the PC, select the PC platform tab. It looks like a download icon. If developing for Android, select the Android platform tab. It looks like an Android icon.

Figure 4: Android Platform



3. In *Other Settings* > *Rendering*, select the *Virtual Reality Supported* check box.

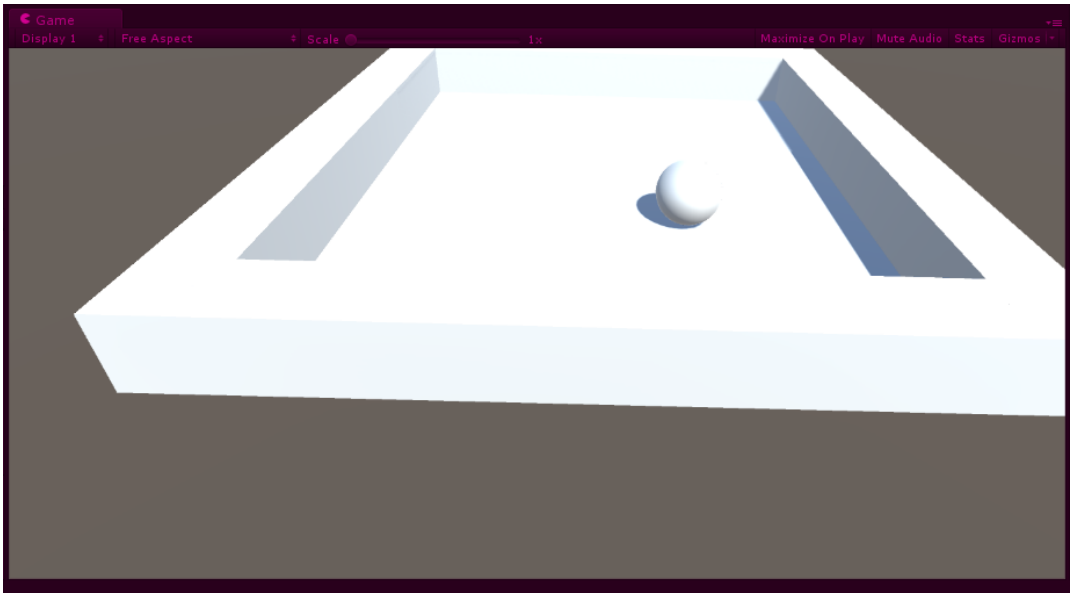


That's it! That's all you need to do to make your game into a VR application.

If you have a Rift, go ahead and try it out. Press the Play button to launch the application in the Play View. If the Oculus App is not already running, it will launch automatically, as it must be running for a VR application to play.

Go ahead and put on the Rift and accept the Health and Safety Warnings. The game will be viewable in the Rift, and the Unity Game View will show the undistorted left eye buffer image.

Save your scene and project before proceeding.



Play

If you are developing for Rift, follow the *Building Rift Applications* on page 10 to build an executable file.

If you are developing for mobile, follow the *Preparing for Mobile Development* on page 6 instructions to build an APK and load it onto your phone, then insert the phone into the Gear VR to launch your game. Be sure to copy your osig to the specified folder as described in that section, or you will not be able to run your application.

To re-launch the applications after they auto-run, you will need to launch them from the Android desktop or your PC file system. Only applications accepted by the Oculus Store are visible in the Oculus Home Library.

Other Oculus Resources for Unity Developers

All Oculus Unity development materials are available for download at our Developer site: *https://developer.oculus.com/downloads/*

Utilities Package Contents

- A Utilities package containing scripts, assets, sample scenes, et cetera. See *Oculus Utilities for Unity* on page 26 for more information.

Samples

- Oculus Sample Framework for Unity 5 - several sample scenes and guidelines for common and important features. See *Unity Sample Framework* for more information.

Mobile Resources

- Oculus Remote Monitor debugging client for mobile development. See *Oculus Remote Monitor (Mobile)* on page 66 for more information).

Platform SDK

- The Oculus Platform supports features related to security, community, revenue, and engagement such as matchmaking, in-app purchase, entitlement checking, VoIP, and cloudsaves. For more information on the Platform Unity plugin, see our *Platform Developer Guide*.

Avatar SDK

- The Oculus Avatar SDK includes a Unity package to assist developers with implementing first-person hand presence for the Rift and Touch controllers. It includes avatar hand and body assets viewable by other users in social applications for Rift and Gear VR. The first-person hand models and third-person hand and body models supported by the Avatar SDK automatically pull the avatar configuration choices the user has made in Oculus Home to provide a consistent sense of identity across applications. The SDK includes a Unity package with scripts, prefabs, art assets, and sample scenes. For more information, see our *Avatar SDK Developer Guide*.

Audio Resources

- Oculus Native Spatializer Plugin (ONSP) for Unity provides easy-to-use high-quality audio spatialization. The ONSP is included with the Audio SDK Plugins. See *Oculus Native Spatializer for Unity* for more information. The Unity Editor includes a built-in basic version of the ONSP with a subset of features. See *Unity Audio* on page 58 for more information.
- Oculus OVRLipSync: is a Unity 5 plugin for animating avatar mouths to match speech sounds.
- Oculus OVRVoiceMod: a Unity 5 plugin used to modify audio signals, making a person's voice sound like a robot, for example, or changing their voice from male to female, or vice-versa.

Getting Started FAQ

If you're new to Oculus development with Unity, this FAQ can answer the most common beginner questions.

Question: What's the best way to get started if you're a beginner?

Answer: Browse through this FAQ, and check out our *Getting Started Guide* on page 5. Read through Unity's excellent *documentation* and try out some of their introductory *tutorials* to get acquainted with Unity development.

When you're ready to get into the trenches, find out what the version of Unity 5 we recommend at our *Compatibility and Requirements* page, then download and install it. Next, build your own simple VR game by following the instructions in our *Tutorial: Build Your First VR App* on page 18.

You can also browse around on our *Unity Developer Forum*.

Question: What are the system requirements for Unity development for Oculus? What operating systems are supported for Unity development?

Answer: For the most up-to-date information, see *Unity Compatibility and Requirements*. We currently support Windows and OS X for development. The Oculus Rift requires Windows 7, 8 or 10.

Question: What version of Unity should I use?

Answer: Our latest version recommendations may be found in our *Unity Compatibility and Requirements* document. Be sure to check back regularly, as we update it frequently as new SDKs and Unity versions are released. You can find an archive of information in our *Unity-SDK Version Compatibility* list.

Question: What other tools and resources do you provide to Unity developers?

Answer: To find the latest tools we provide, check out our *Other Oculus Resources for Unity Developers* on page 24.

Question: What do I need to run Rift applications that I build with Unity?

Answer: You will need a compatible Windows PC, a Rift, and the Oculus software. For more details, see *Preparing for Rift Development* on page 6

Question: I want to focus on mobile development for the Samsung Gear VR. What do I need to do to get started? Do I need to download the Oculus Mobile SDK?

Answer: The Android SDK is required for mobile development with Unity. However, most Unity developers do not need to download the Oculus Mobile SDK, or to install Android Studio or NDK. For more details, see *Preparing for Mobile Development* on page 6.

Question: Can I develop a single application for both Samsung Gear VR and the Oculus Rift?

Answer: Yes, but when developing for both Rift and mobile platforms, keep in mind that the requirements for PC and mobile VR applications differ substantially. If you would like to generate builds for both PC and mobile from a single project, it is important to follow the more stringent mobile development best practices, as well as meeting the required 90 fps required by the Rift.

Question: What is the difference between the Oculus Unity 4 Legacy Integration and the Oculus Utilities for Unity 5? How do the differences between Unity 4 and 5 affect development?

Answer: All developers should use Unity 5 and the optional Oculus Utilities package. The Unity 4 Integration is maintained for legacy purposes only.

In Unity 4, you must import our Legacy Integration for Unity 4 unitypackage and add the supplied VR camera and player controller to your application to add Rift or Gear VR support.

Question: Where can I ask questions or get help?

Answer: Visit our developer support forums at *https://developer.oculus.com*. Our Support Center can be accessed at *https://support.oculus.com*.

Unity Developer Guide

Welcome to the Oculus Unity Developer Guide.

This guide describes development using Unity's first-party Oculus support, and the contents and features of the Oculus Utilities for Unity 5 package.

Unity VR Support

Unity 5.1 and later offer first-party virtual reality support, enabled with the *Virtual Reality Supported* checkbox in the *Other Settings* > *Configuration* tab of *Player Settings*.

When Unity virtual reality support is enabled, any camera with no render texture is automatically rendered in stereo to your device. Positional and head tracking are automatically applied to your camera, overriding your camera's transform.

Unity applies head tracking to the VR camera within the reference frame of the camera's local pose when the application starts. If you are using OVRCameraRig, that reference frame is defined by the TrackingSpace GameObject, which is the parent of the CenterEyeAnchor GameObject that has the Camera component.

The Unity Game View does not apply lens distortion. The image corresponds to the left eye buffer and uses simple pan-and-scan logic to correct the aspect ratio.

For more information and instructions for using Unity's VR support, see the *Virtual Reality section* of the Unity Manual.

Note: Unity's VR support is not compatible with Oculus Legacy Integrations for Unity 4.

Oculus Utilities for Unity

This section provides an overview of the Utilities package, including its directory structure, the supplied prefabs, and several key C# scripts.

Contents

OVR

The contents of the OVR folder in OculusUtilities.unitypackage are uniquely named and should be safe to import into an existing project.

The OVR directory contains the following subdirectories:

Editor	Scripts that add functionality to the Unity Editor and enhance several C# component scripts.
Materials	Materials used for graphical components within the Utilities package, such as the main GUI display.

Meshes	Meshes required by some OVR scripts, such as the TrackerBounds.
Prefabs	The main Unity prefabs used to provide the VR support for a Unity scene: OVRCameraRig and OVRPlayerController.
Scenes	Sample scenes illustrating common concepts.
Scripts	C# files used to tie the VR framework and Unity components together. Many of these scripts work together within the various Prefabs.
Textures	Image assets required by some script components.

Note: We strongly recommend that developers not directly modify the included OVR scripts.

Plugins

The Plugins folder contains the OVRGamepad.dll, which enables scripts to communicate with the Xbox gamepad on Windows (both 32 and 64-bit versions).

This folder also contains the plugin for Mac OS: OVRGamepad.bundle.

Prefabs

Utilities for Unity 5 provides prefabs in Assets/OVR/Prefabs:

- OVRCameraRig
- OVRPlayerController
- OVRCubemapCaptureProbe

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

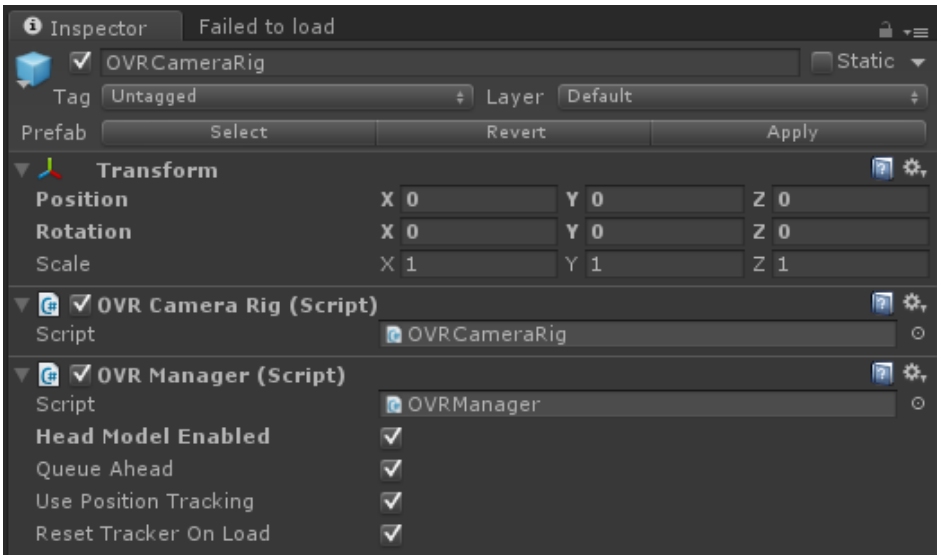
OVRCameraRig

OVRCameraRig is a custom VR camera that may be used to replace the regular Unity Camera in a scene. Drag an OVRCameraRig into your scene and you will be able to start viewing the scene with the Gear VR and Rift.

The primary benefit to using OVRCameraRig is that it provides access to OVRManager, which provides the main interface to the VR hardware. If you do not need such access, a standard Unity camera may be easily configured to add basic VR support; see *Unity VR Support* on page 26 for more information.

Note: Make sure to turn off any other Camera in the scene to ensure that OVRCameraRig is the only one being used.

Figure 5: OVRCameraRig and OVRManager



OVRCameraRig contains one Unity camera, the pose of which is controlled by head tracking; two "anchor" Game Objects for the left and right eyes; and one "tracking space" Game Object that allows you to fine-tune the relationship between the head tracking reference frame and your world. The rig is meant to be attached to a moving object, such as a character walking around, a car, a gun turret, et cetera. This replaces the conventional Camera.

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

- OVRCameraRig.cs
- OVRManager.cs

Learn more about the OVRCameraRig and OVRManager components in *Unity Components* on page 30.

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. It includes a physics capsule, a movement system, a simple menu system with stereo rendering of text fields, and a cross-hair component.

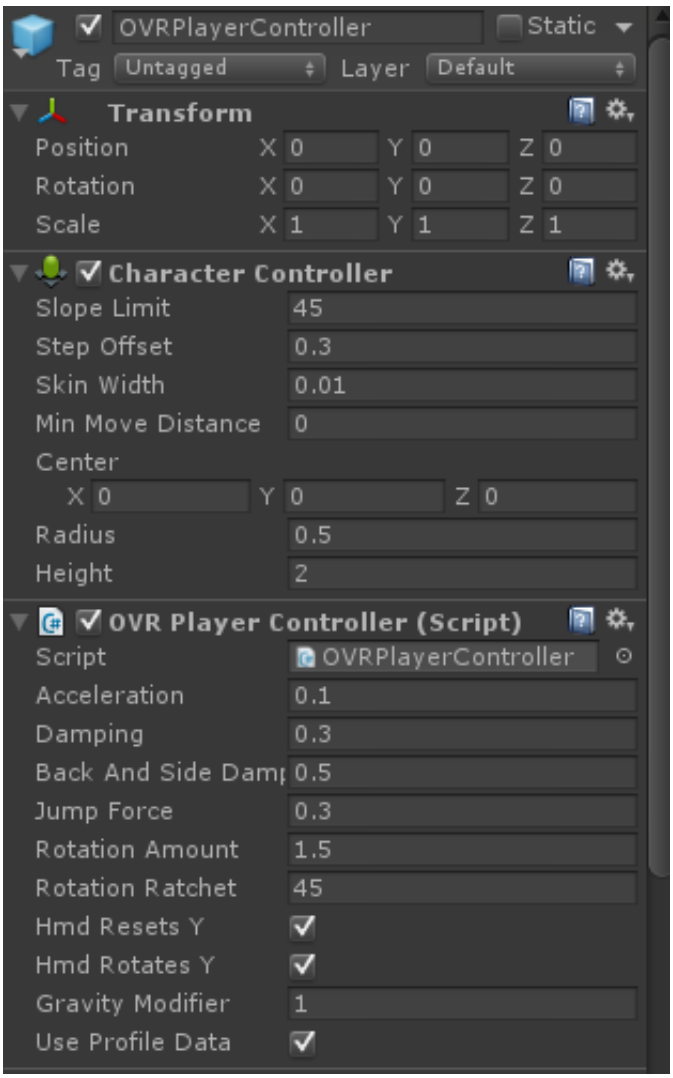
To use, drag the player controller into an environment and begin moving around using a gamepad, or a keyboard and mouse.

Note: Make sure that collision detection is active in the environment.

One script (Components) is attached to the OVRPlayerController prefab:

- OVRPlayerController.cs

Figure 6: OVRPlayerController



OVRCubemapCaptureProbe

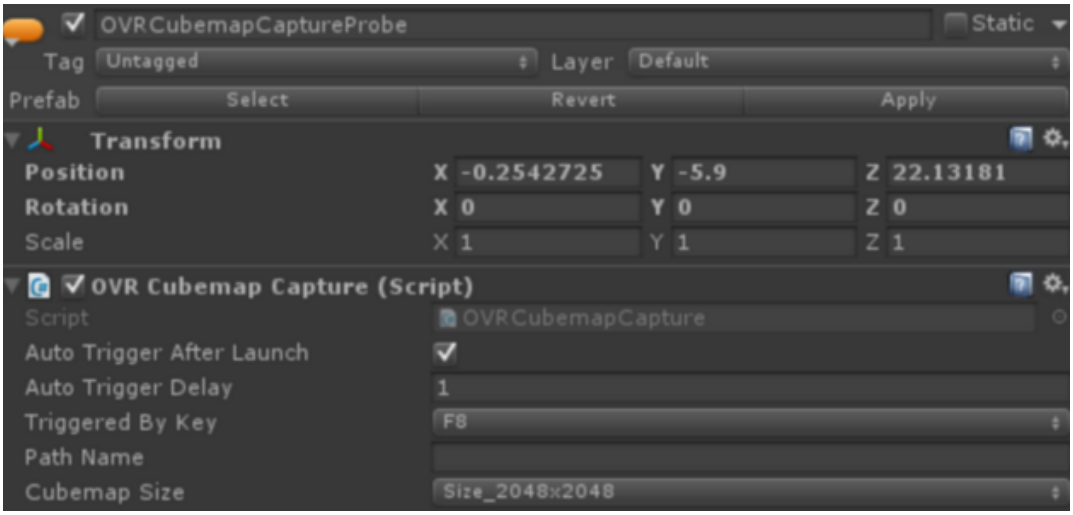
This prefab allows you to capture a static 360 screenshot of your application while it is running, either at a specified time after launch, when a specified key is pressed, or when the static function OVRCubemapCapture.TriggerCubemapCapture is called. For more information on this function, see our Unity Developer Reference.

OVRCubemapCaptureProbe is based on OVR Screenshot (see *Cubemap Screenshots* on page 59 for more information).

Screenshots are taken from the perspective of your scene camera. They are written to a specified directory and may be either JPEG or PNG. File type is specified by the file extension entered in the Path Name field; default is PNG. Resolution is configurable.

Basic Use

Drag OVRCubemapCaptureProbe into the scene and set the parameters as desired in the Inspector view.



Parameters

Auto Trigger After Launch	Select to enable capture after a delay specified in Auto Trigger Delay. Otherwise capture is triggered by the keypress specified in Triggered by Key.
Auto Trigger Delay	Specify delay after application launch before cubemap is taken. (requires Auto Trigger After Launch selected).
Triggered By Key	Specifies key to trigger image capture (requires Auto Trigger After Launch not selected).
Path Name	Specifies directory, file name, and file type (JPEG or PNG) for screen capture. Windows default path: C:\Users\$username\AppData\LocalLow\Sample\$yourAppName\OVR_ScreenShot306\ Android default path: /storage/emulated/0/Android/data/com.unity3d.$yourAppName/files/OVR_ScreenShot360/ Default file type: PNG.
Cubemap Size	Specify size (2048 x 2048 is default, and is the resolution required for preview cubemaps submitted to the Oculus Store).

Unity Components

This section gives a general overview of the Components provided by the Utilities package.

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 halfway between them.

This Component is the main interface between Unity and the cameras. It is attached to a prefab that makes it easy to add comfortable VR support to a scene.

Important: All camera control should be done through this component. You should understand this script when implementing your own camera control mechanism.

Mobile and Rift Public Members

Updated Anchors	Allows clients to filter the poses set by tracking. Used to modify or ignore positional tracking.

Game Object Structure

TrackingSpace	A Game Object that defines the reference frame used by tracking. You can move this relative to the OVRCameraRig for use cases in which the rig needs to respond to tracker input. For example, OVRPlayerController changes the position and rotation of TrackingSpace to make the character controller follow the yaw of the current head pose.

OVRManager

OVRManager is the main interface to the VR hardware. It is a singleton that exposes the Oculus SDK to Unity, and includes helper functions that use the stored Oculus 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 it includes public members that allow for changing certain values in the Unity Inspector.

OVRManager.cs contains the following public members:

Table 1: Mobile and Rift Public Members

Monoscopic	If true, rendering will try to optimize for a single viewpoint rather than rendering once for each eye. Not supported on all platforms.

Table 2: Rift-Only Public Members

Queue Ahead (Deprecated)	When enabled, distortion rendering work is submitted a quarter-frame early to avoid pipeline stalls and increase CPU-GPU parallelism.
Use Recommended MSAA Level	When enabled, Unity will use the optimal antialiasing level for quality/performance on the current hardware.
Enable Adaptive Resolution (Rift only)	Enable to configure app resolution to scale down as GPU exceeds 85% utilization, and to scale up as it falls below 85% (range 0.5 - 2.0; 1 = normal density). Requires Unity 5.4 or later. To minimize the perceived artifacts from changing resolution, there is a two-second minimum delay between every resolution change.
Max Render Scale (Rift only)	Sets minimum bound for Adaptive Resolution (default = 0.7).
Min Render Scale (Rift only)	Sets maximum bound for Adaptive Resolution (default = 1.0).

Tracking Origin Type	Set to `Eye Level` to track the position and orientation y-axis relative to the HMD's position. Set to `Floor Level` to track position and orientation relative to the floor, based on the user's standing height as specified in the Oculus Configuration Utility. Default is `Eye Level`.
Use Position Tracking	Disables the IR tracker and causes head position to be inferred from the current rotation using the head model.
Use IPD in Position Tracking	If enabled, the distance between the user's eyes will affect the position of each OVRCameraRig's cameras.
Reset Tracker On Load	When disabled, 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.

Helper Classes

In addition to the above components, your scripts can always access the HMD state via static members of OVRManager. For detailed information, see our *Unity Scripting Reference* on page 79.

OVRDisplay	Provides the pose and rendering state of the HMD.
OVRTracker	Provides the pose, frustum, and tracking status of the infrared tracking sensor.

Rift Recentering

`OVRManager.display.RecenterPose()` recenters the head pose and the tracked controller pose, if present (see *OVRInput Unified Input API* on page 35 for more information on tracking controllers).

If `Tracking Origin Type` is set to `Floor Level`, `OVRManager.display.RecenterPose()` resets the x-, y-, and z-axis position to origin. If it is set to Eye Level, the x-, y-, and z-axis are all reset to origin, with the y-value corresponding to the height calibration performed with the Oculus Configuration Utility. In both cases, the y rotation is reset to 0, but the x and z rotation are unchanged to maintain a consistent ground plane.

Recenter requests are passed to the Oculus C API. For a more detailed description of what happens subsequently, please see *VR Focus Management* in our PC SDK Developer Guide.

Utilities

OVRPlayerController	OVRPlayerController implements a basic first-person controller for the VR framework. 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 Game Object 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.
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.

Game Object Structure

ForwardDirection	An empty Game Object attached to the OVRPlayerController prefab containing the matrix upon which motor control bases its direction. This Game Object should also house the body geometry which will be seen by the player. See TrackingSpace in "OVRCameraRig" for more information

For more information on OVRInput, see *OVRInput Unified Input API* on page 35.

Oculus Scripts and Scenes

A handful of rudimentary sample scenes are provided in Assets/OVR/Scenes. They illustrate simple implementations of basic components and may be useful for verifying that VR functionality is working properly. For much more detailed samples including scripts and assets that you may re-use in your own applications, see *Unity Sample Framework* on page 72.

Trivial	An empty scene with one cube and a plain Unity camera. If this scene fails to render normally, Unity's VR support is not working.
Cubes	A 3D array of cubes and an OVRCameraRig from the Utilities package.
Room	A cubic room formed from six cubes enclosing an OVRPlayerController. Includes the scripts OVRGrabber and OVRGrabable, enabling users to pick up cubes with Touch controllers.

Scripts for assisting with mobile development are located in Assets/OVR/Scripts/. Scripts included in the folder that are not intended for developers to use are omitted from this list.

OVRBoundary	Exposes exposes an API for interacting with the Oculus Guardian System. For more information, see *OVRBoundary Guardian System API* on page 46.
OVRGrabber	Allows grabbing and throwing of objects with the OVRGrabbable component on them using Oculus Touch. Requires OVRGrabbable to use.

OVRGrabbable	Attach to objects to allow them to be grabbed and thrown with the Oculus Touch. Requires OVRGrabber to use.
OVRHaptics	Programmatically controls haptic feedback to Touch controllers. For more information, see *OVRHaptics for Oculus Touch* on page 48.
OVRHapticsClip	Programmatically controls haptic feedback to Touch controllers. For more information, see *OVRHaptics for Oculus Touch* on page 48.
OVRInput	Exposes a unified input API for multiple controller types. For more information, see *OVRInput Unified Input API* on page 35.
OVROverlay.cs	Add to a Game Object to render as a VR Compositor Layer instead by drawing it into the eye buffer. For more information, see *VR Compositor Layers* on page 49
OVRPlatformMenu.cs	Helper component for detecting Back Key long-press to bring-up the Universal Menu and Back Key short-press to bring up the Confirm-Quit to Home Menu. Additionally implements a Wait Timer for displaying Long Press Time. For more information on interface guidelines and requirements, please review *Interface Guidelines* and *Universal Menu* in the *Mobile SDK documentation*.
OVRTouchpad.cs	Interface class to a touchpad.

Simple scripts for assisting with mobile development are located in Assets/OVR/Scripts/Util. Scripts included in the folder that are not intended for developers to use are omitted from this list.

OVRChromaticAberration.cs	Drop-in component for toggling chromatic aberration correction on and off for Android.
OVRDebugGraph.cs	Drop-in component for toggling the TimeWarp debug graph on and off. Information regarding the TimeWarp Debug Graph may be found in the *TimeWarp technical note* in the *Mobile SDK documentation*.
OVRModeParms.cs	Example code for de-clocking your application to reduce power and thermal load as well as how to query the current power level state.
OVRMonoscopic.cs	Drop-in component for toggling Monoscopic rendering on and off for Android.
OVRResetOrientation.cs	Drop-in component for resetting the camera orientation.
OVRWaitCursor.cs	Helper component for auto-rotating a wait cursor.

See our *Oculus Utilities for Unity Reference Manual* for a more detailed look at these and other C# scripts. Undocumented scripts may be considered internal, and should generally never be modified.

Input

This guide describes Unity input features supported by the Oculus integration.

Topics include:

- OVRInput unified input API
- OVRBoundary API for interacting with the Rift Guardian System API
- OVRHaptics for Touch

OVRInput Unified Input API

OVRInput exposes a unified input API for multiple controller types. It may be used to query virtual or raw controller state, such as buttons, thumbsticks, triggers, and capacitive touch data. It currently supports the Oculus Touch, Microsoft Xbox controllers, and the Oculus remote on desktop platforms. For mobile development, it supports the Gear VR Controller as well as the touchpad and back button on the Gear VR headset. Gear VR gamepads must be Android compatible and support Bluetooth 3.0.

For keyboard and mouse control, we recommend using the UnityEngine.Input scripting API (see Unity's *Input scripting reference* for more information).

Mobile input bindings are automatically added to InputManager.asset if they do not already exist.

For more information, see *OVRInput* in the *Unity Scripting Reference* on page 79. For more information on Unity's input system and Input Manager, documented here: *http://docs.unity3d.com/Manual/Input.html* and *http://docs.unity3d.com/ScriptReference/Input.html*.

`SetControllerVibration()` support for Oculus Touch is now deprecated; please use *OVRHaptics for Oculus Touch* on page 48 instead.

Oculus Touch Tracking

OVRInput provides Touch position and orientation data through `GetLocalControllerPosition()` and `GetLocalControllerRotation()`, which return a Vector3 and Quaternion, respectively.

Controller poses are returned by the constellation tracking system and are predicted simultaneously with the headset. These poses are reported in the same coordinate frame as the headset, relative to the initial center eye pose, and may be used for rendering hands or objects in the 3D world. They are also reset by `OVRManager.display.RecenterPose()`, similar to the head and eye poses.

Gear VR Controller

Gear VR Controller provides orientation data through `GetLocalControllerRotation()`, which returns a quaternion.

Gear VR positions the controller relative to the user by using a body model to estimate the controller's position. Whether the controller is visualized on the left or right side of the body is determined by left-handedness versus right-handedness, which is specified by users during controller pairing.

To query handedness of a paired controller, use `IsControllerConnected()` or `GetActiveController()` to query for `RTrackedRemote` or `LTrackedRemote`.

For example:

```
// returns true if right-handed controller connected
OVRInput.IsControllerConnected(OVRInput.Controller.RTrackedRemote);
```

Use `OVRInput.Get()` to query controller touchpad input. You may query the input position with Axis2D:

```
OVRInput.Get(OVRInput.Axis2D.PrimaryTouchpad, OVRInput.Controller.RTrackedRemote);
```

A touchpad touch occurs when the user's finger makes contact with the touchpad without actively clicking it. Touches may be queried with `OVRInput.Get(OVRInput.Touch.PrimaryTouchpad)`. Touchpad clicks are alias to virtual button One clicks, and may be queried with `OVRInput.Get(OVRInput.Button.PrimaryTouchpad)`.

The volume and home buttons are reserved.

OVRInput Usage

The primary usage of OVRInput is to access controller input state through `Get()`, `GetDown()`, and `GetUp()`.

- `Get()` queries the current state of a control.
- `GetDown()` queries if a control was pressed this frame.
- `GetUp()` queries if a control was released this frame.

Control Input Enumerations

There are multiple variations of `Get()` that provide access to different sets of controls. These sets of controls are exposed through enumerations defined by OVRInput as follows:

Control	Enumerates
OVRInput.Button	Traditional buttons found on gamepads, Touch controllers, the Gear VR Controller touchpad and back button, and the Gear VR headset touchpad and back button.
OVRInput.Touch	Capacitive-sensitive control surfaces found on the Oculus Touch and Gear VR Controller.
OVRInput.NearTouch	Proximity-sensitive control surfaces found on the Oculus Touch controllers.
OVRInput.Axis1D	One-dimensional controls such as triggers that report a floating point state.
OVRInput.Axis2D	Two-dimensional controls including thumbsticks and the Gear VR Controller touchpad. Report a Vector2 state.

A secondary set of enumerations mirror the first, defined as follows:

OVRInput.RawButton
OVRInput.RawTouch
OVRInput.RawNearTouch
OVRInput.RawAxis1D

OVRInput.RawAxis2D

The first set of enumerations provides a virtualized input mapping that is intended to assist developers with creating control schemes that work across different types of controllers. The second set of enumerations provides raw unmodified access to the underlying state of the controllers. We recommend using the first set of enumerations, since the virtual mapping provides useful functionality, as demonstrated below.

Button, Touch, and NearTouch

In addition to traditional gamepad buttons, the Oculus Touch controllers feature capacitive-sensitive control surfaces which detect when the user's fingers or thumbs make physical contact (a "touch"), as well as when they are in close proximity (a "near touch"). This allows for detecting several distinct states of a user's interaction with a specific control surface. For example, if a user's index finger is fully removed from a control surface, the NearTouch for that control will report false. As the user's finger approaches the control and gets within close proximity to it, the NearTouch will report true prior to the user making physical contact. When the user makes physical contact, the Touch for that control will report true. When the user pushes the index trigger down, the Button for that control will report true. These distinct states can be used to accurately detect the user's interaction with the controller and enable a variety of control schemes.

The Gear VR Controller touchpad may be queried for both touch status and click status, where "touch" refers to the user's finger making contact with the touchpad without actively clicking it.

Example Usage

```
// returns true if the primary button (typically "A") is currently pressed.
OVRInput.Get(OVRInput.Button.One);

// returns true if the primary button (typically "A") was pressed this frame.
OVRInput.GetDown(OVRInput.Button.One);

// returns true if the "X" button was released this frame.
OVRInput.GetUp(OVRInput.RawButton.X);

// returns a Vector2 of the primary (typically the Left) thumbstick's current state.
// (X/Y range of -1.0f to 1.0f)
OVRInput.Get(OVRInput.Axis2D.PrimaryThumbstick);

// returns true if the primary thumbstick is currently pressed (clicked as a button)
OVRInput.Get(OVRInput.Button.PrimaryThumbstick);

// returns true if the primary thumbstick has been moved upwards more than halfway.
// (Up/Down/Left/Right - Interpret the thumbstick as a D-pad).
OVRInput.Get(OVRInput.Button.PrimaryThumbstickUp);

// returns a float of the secondary (typically the Right) index finger trigger's current state.
// (range of 0.0f to 1.0f)
OVRInput.Get(OVRInput.Axis1D.SecondaryIndexTrigger);

// returns a float of the left index finger trigger's current state.
// (range of 0.0f to 1.0f)
OVRInput.Get(OVRInput.RawAxis1D.LIndexTrigger);

// returns true if the left index finger trigger has been pressed more than halfway.
// (Interpret the trigger as a button).
OVRInput.Get(OVRInput.RawButton.LIndexTrigger);

// returns true if the secondary gamepad button, typically "B", is currently touched by the user.
OVRInput.Get(OVRInput.Touch.Two);

// returns true after a Gear VR touchpad tap
OVRInput.GetDown(OVRInput.Button.One);

// returns true on the frame when a user's finger pulled off Gear VR touchpad controller on a swipe
 down
OVRInput.GetDown(OVRInput.Button.DpadDown);

// returns true the frame AFTER user's finger pulled off Gear VR touchpad controller on a swipe right
OVRInput.GetUp(OVRInput.RawButton.DpadRight);
```

```
// returns true if the Gear VR back button is pressed
OVRInput.Get(OVRInput.Button.Two);

// Returns true if the the Gear VR Controller trigger is pressed down
OVRInput.Get(OVRInput.Button.PrimaryIndexTrigger);

// Queries active Gear VR Controller touchpad click position
// (normalized to a -1.0, 1.0 range, where -1.0, -1.0 is the lower-left corner)
OVRInput.Get(OVRInput.Axis2D.PrimaryTouchpad, OVRInput.Controller.RTrackedRemote);

// If no controller is specified, queries the touchpad position of the active Gear VR Controller
OVRInput.Get(OVRInput.Axis2D.PrimaryTouchpad);

// returns true if the Gear VR Controller back button is pressed
OVRInput.Get(OVRInput.Button.Back);

// returns true on the frame when a user's finger pulled off Gear VR Controller back button
OVRInput.GetDown(OVRInput.Button.Back);
```

In addition to specifying a control, `Get()` also takes an optional controller parameter. The list of supported controllers is defined by the OVRInput.Controller enumeration (for details, refer to *OVRInput* in the *Unity Scripting Reference* on page 79.

Specifying a controller can be used if a particular control scheme is intended only for a certain controller type. If no controller parameter is provided to `Get()`, the default is to use the `Active` controller, which corresponds to the controller that most recently reported user input. For example, a user may use a pair of Oculus Touch controllers, set them down, and pick up an Xbox controller, in which case the Active controller will switch to the Xbox controller once the user provides input with it. The current Active controller can be queried with `OVRInput.GetActiveController()` and a bitmask of all the connected Controllers can be queried with `OVRInput.GetConnectedControllers()`.

Example Usage:

```
// returns true if the Xbox controller's D-pad is pressed up.
OVRInput.Get(OVRInput.Button.DpadUp, OVRInput.Controller.Gamepad);

// returns a float of the Hand Trigger's current state on the Left Oculus Touch controller.
OVRInput.Get(OVRInput.Axis1D.PrimaryHandTrigger, OVRInput.Controller.Touch);

// returns a float of the Hand Trigger's current state on the Right Oculus Touch controller.
OVRInput.Get(OVRInput.Axis1D.SecondaryHandTrigger, OVRInput.Controller.Touch);
```

Querying the controller type can also be useful for distinguishing between equivalent buttons on different controllers. For example, if you want code to execute on input from a gamepad or Touch controller, but not on a Gear VR Touchpad, you could implement it as follows:

```
if (OVRInput.GetActiveController() != OVRInput.Controller.Touchpad) { /* do input handling */ }
```

Note that the Oculus Touch controllers may be specified either as the combined pair (with `OVRInput.Controller.Touch`), or individually (with `OVRInput.Controller.LTouch` and `RTouch`). This is significant because specifying LTouch or RTouch uses a different set of virtual input mappings that allow more convenient development of hand-agnostic input code. See the virtual mapping diagrams in *Touch Input Mapping* for an illustration.

Example Usage:

```
// returns a float of the Hand Trigger's current state on the Left Oculus Touch controller.
OVRInput.Get(OVRInput.Axis1D.PrimaryHandTrigger, OVRInput.Controller.LTouch);

// returns a float of the Hand Trigger's current state on the Right Oculus Touch controller.
OVRInput.Get(OVRInput.Axis1D.PrimaryHandTrigger, OVRInput.Controller.RTouch);
```

This can be taken a step further to allow the same code to be used for either hand by specifying the controller in a variable that is set externally, such as on a public variable in the Unity Editor.

Example Usage:

```
// public variable that can be set to LTouch or RTouch in the Unity Inspector
public Controller controller;
…
// returns a float of the Hand Trigger's current state on the Oculus Touch controller
// specified by the controller variable.
OVRInput.Get(OVRInput.Axis1D.PrimaryHandTrigger, controller);

// returns true if the primary button ("A" or "X") is pressed on the Oculus Touch controller
// specified by the controller variable.
OVRInput.Get(OVRInput.Button.One, controller);
```

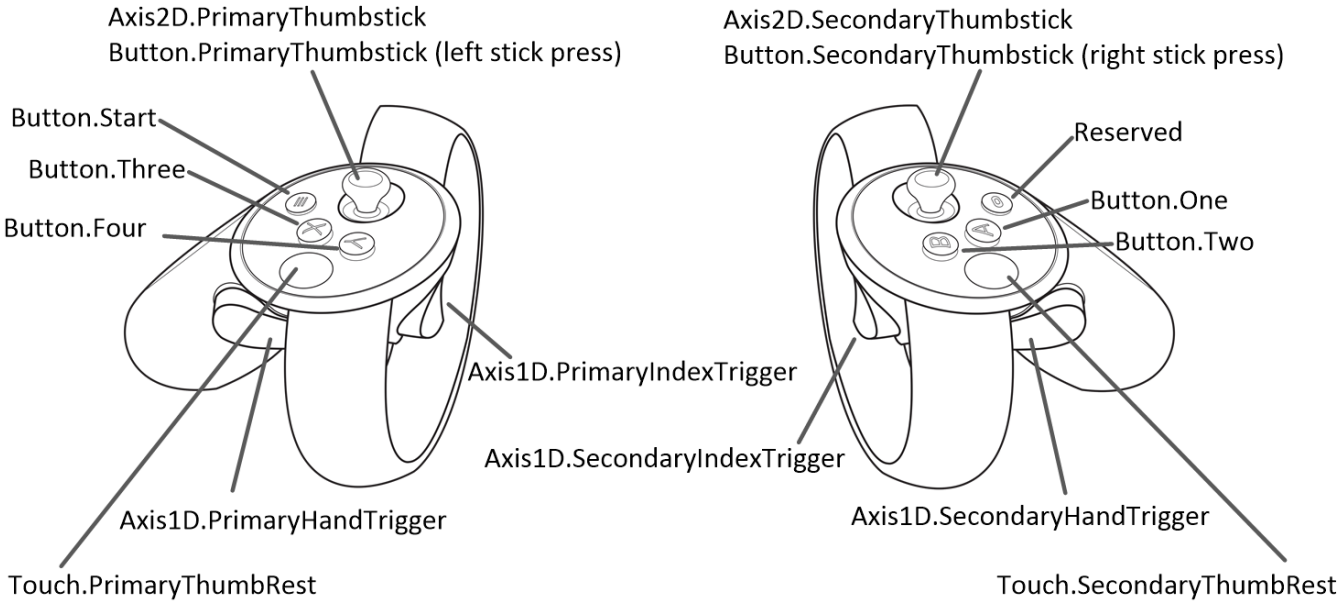
This is convenient since it avoids the common pattern of if/else checks for Left/Right hand input mappings.

Touch Input Mapping

The following diagrams illustrate common input mappings for Oculus Touch controllers. For more information on additional mappings that are available, refer to *OVRInput* in the *Unity Scripting Reference* on page 79.

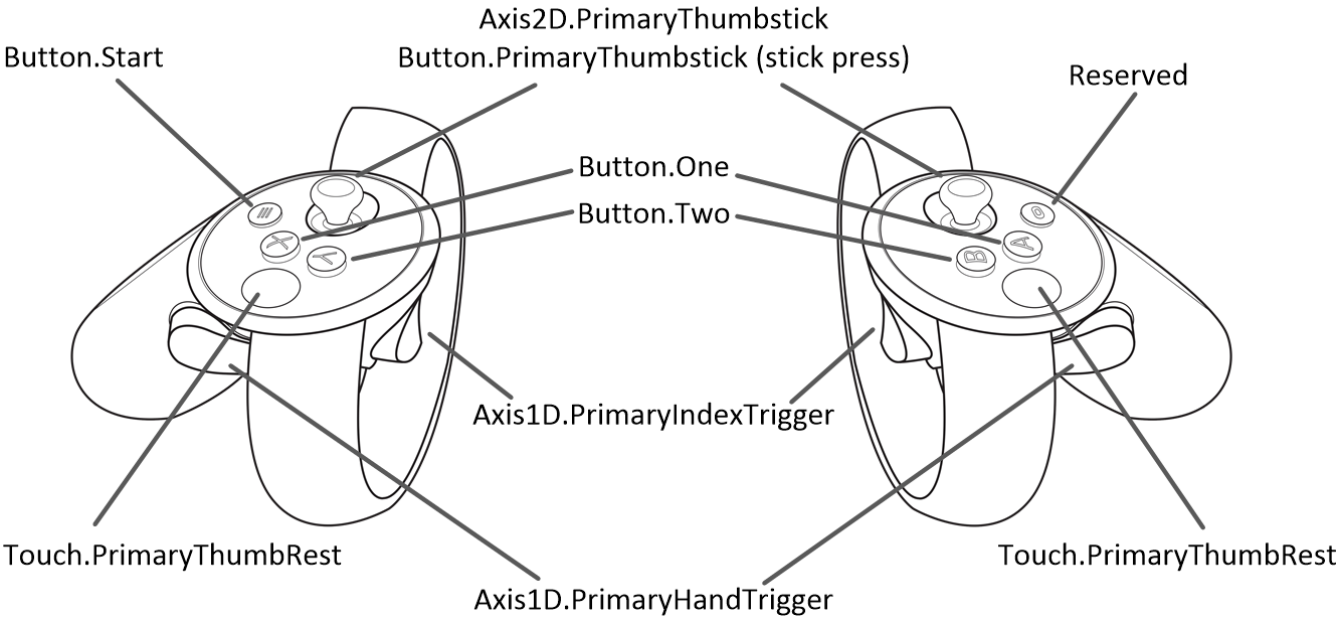
Virtual Mapping (Accessed as a Combined Controller)

When accessing the Touch controllers as a combined pair with OVRInput.Controller.Touch, the virtual mapping closely matches the layout of a typical gamepad split across the Left and Right hands.



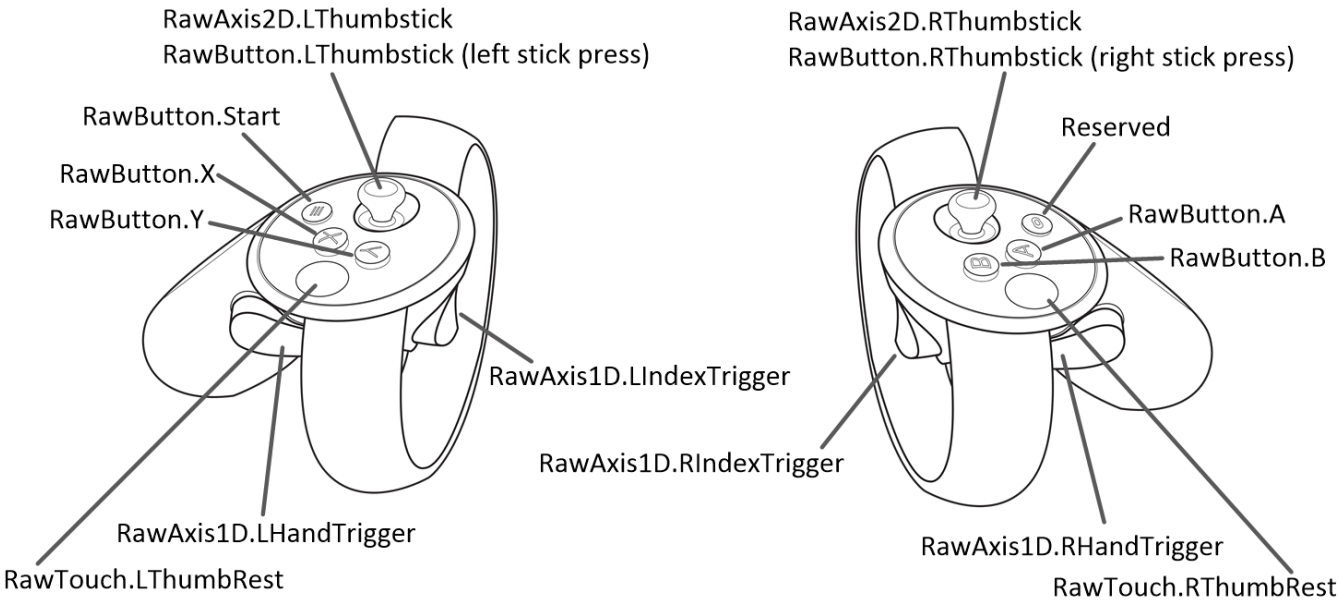
Virtual Mapping (Accessed as Individual Controllers)

When accessing the Left or Right Touch controllers individually with OVRInput.Controller.LTouch or OVRInput.Controller.RTouch, the virtual mapping changes to allow for hand-agnostic input bindings. For example, the same script can dynamically query the Left or Right Touch controller depending on which hand it is attached to, and Button.One will be mapped appropriately to either the A or X button.



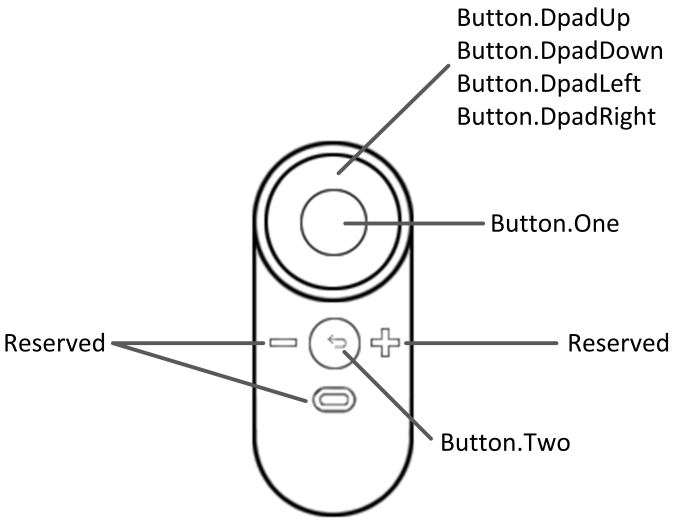
Raw Mapping

The raw mapping directly exposes the Touch controllers. The layout of the Touch controllers closely matches the layout of a typical gamepad split across the Left and Right hands.

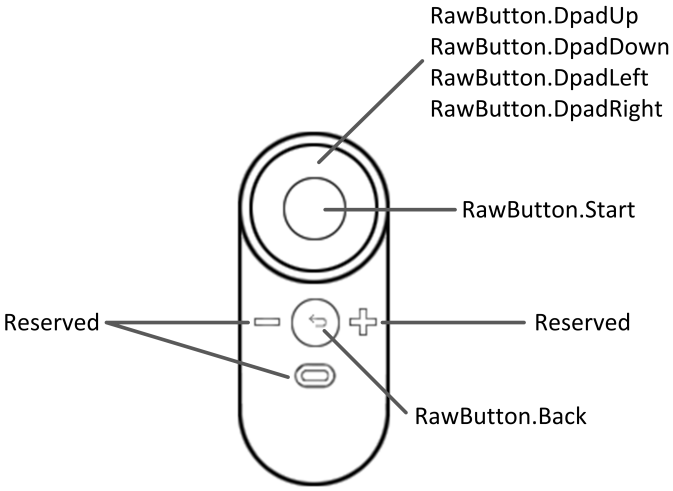


Rift Remote Input Mapping

Virtual Mapping



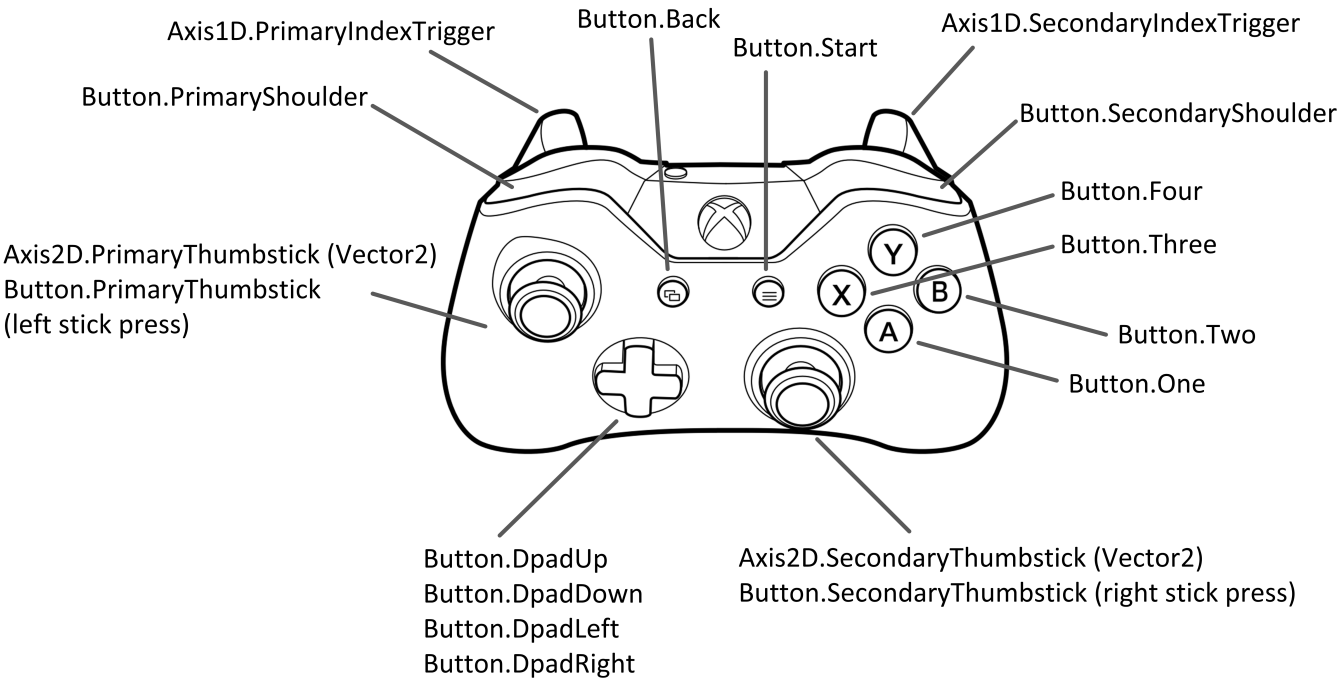
Raw Mapping



Xbox Input Handling

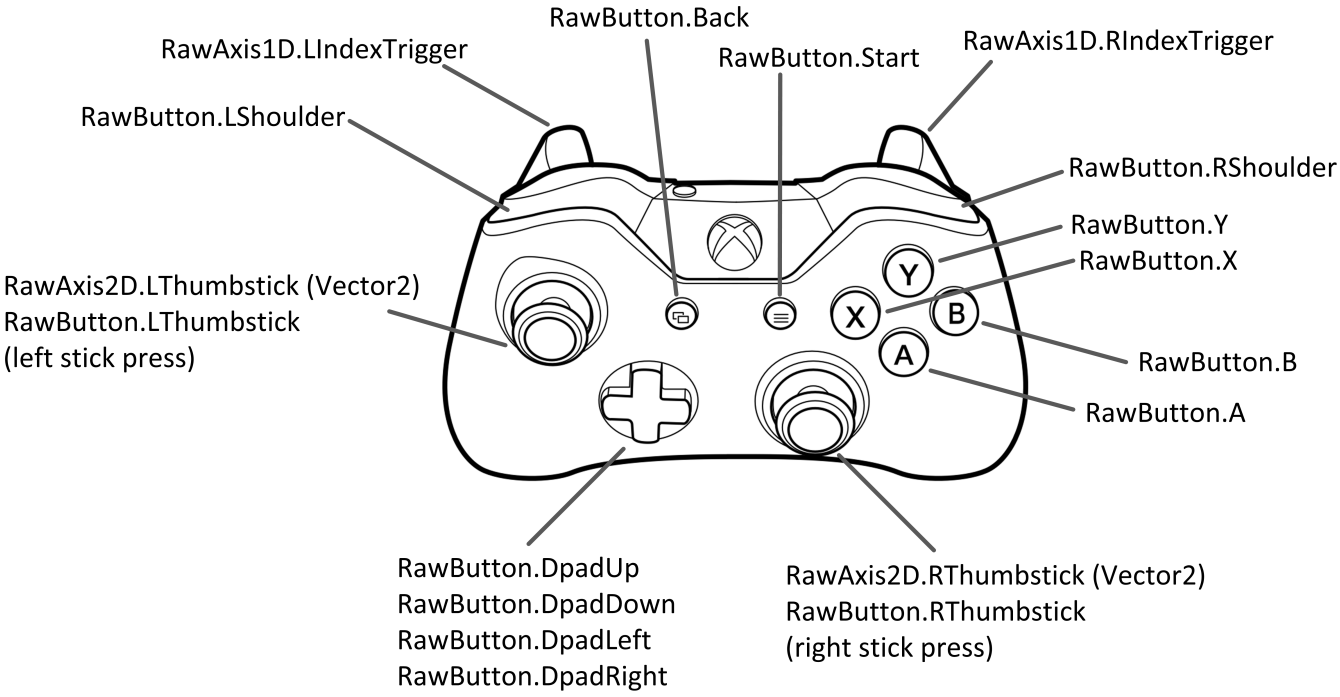
Virtual Mapping

This diagram shows a common implementation of Xbox controller input bindings using OVRInput.Controller.Gamepad.



Raw Mapping

The raw mapping directly exposes the Xbox controller.



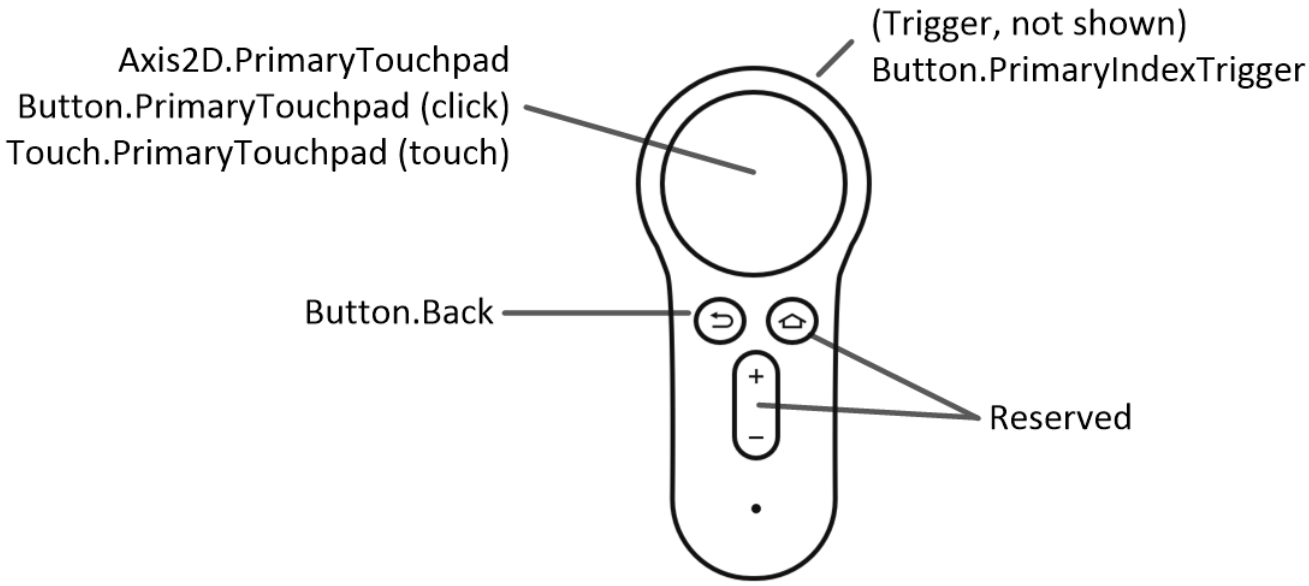
Gear VR Controller Input Handling

Virtual Mapping

This diagram shows a common implementation of Gear VR Controller input bindings using OVRInput.Controller.RTrackedRemote.

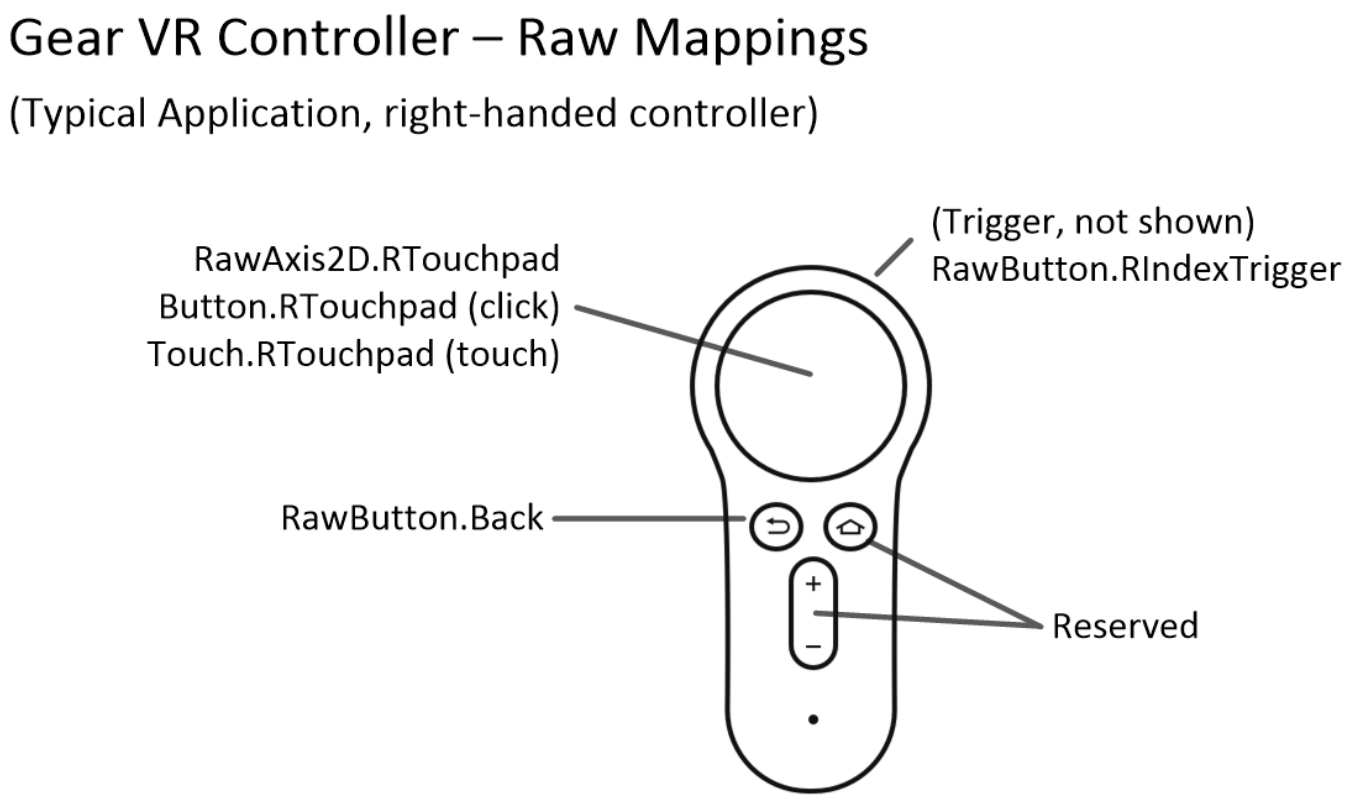
Gear VR Controller – Virtual Mappings

(Typical Application)



Raw Mapping

The raw mapping directly exposes the Gear VR Controller. Note that this assumes a right-handed controller.

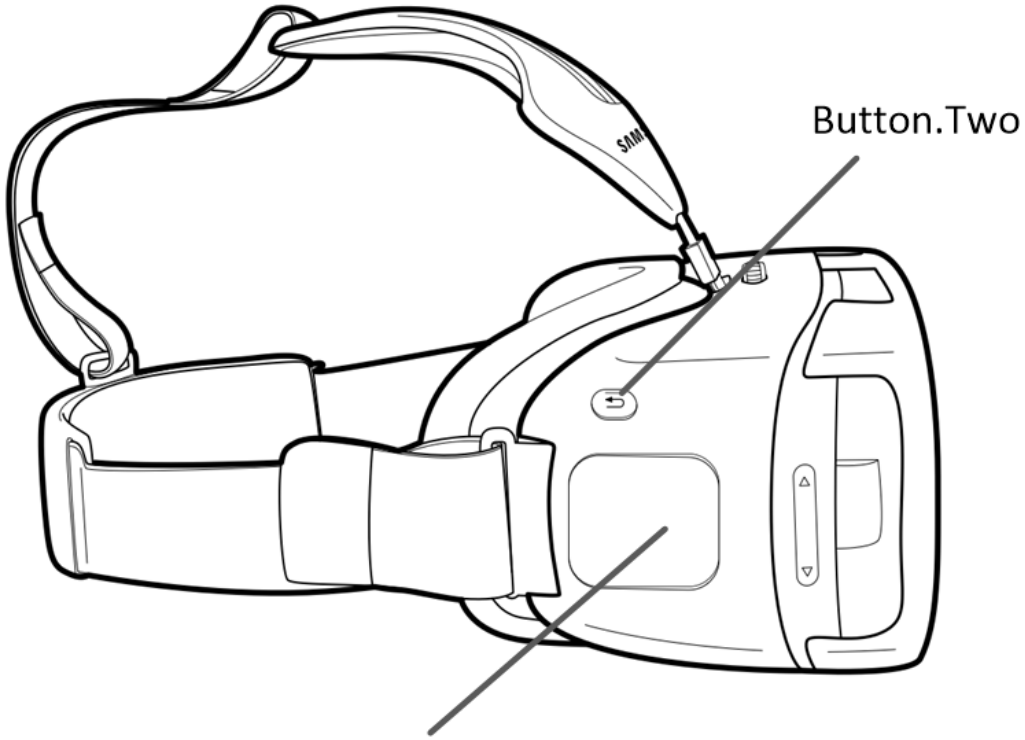


Gear VR Input Handling

A Gear VR touchpad swipe is not defined until the user removes their finger from the touchpad. `Get()` and `GetDown()` will return true on the frame that the user's finger is pulled off, and `GetUp()` will return true the next frame. A Gear VR touchpad tap may be queried with Button.One/RawButton.Start, and a back button press may be queried with Button.Two/RawButton.Back.

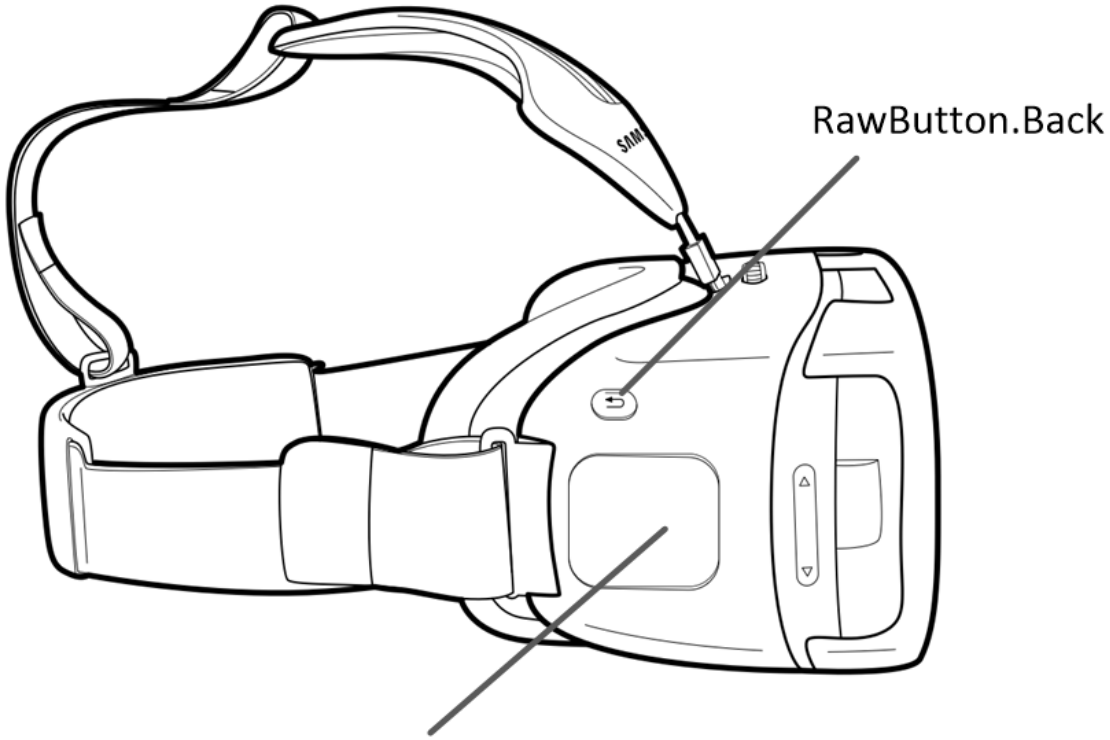
Note that a back-button long-press is reserved and is automatically handled by the Gear VR VrApi. For more information, see *Universal Menu and Volume* in our Mobile SDK Developer Guide.

Virtual Mapping



Button.DpadUp
Button.DpadDown
Button.DpadLeft
Button.DpadRight
Button.One (tap)

Raw Mapping

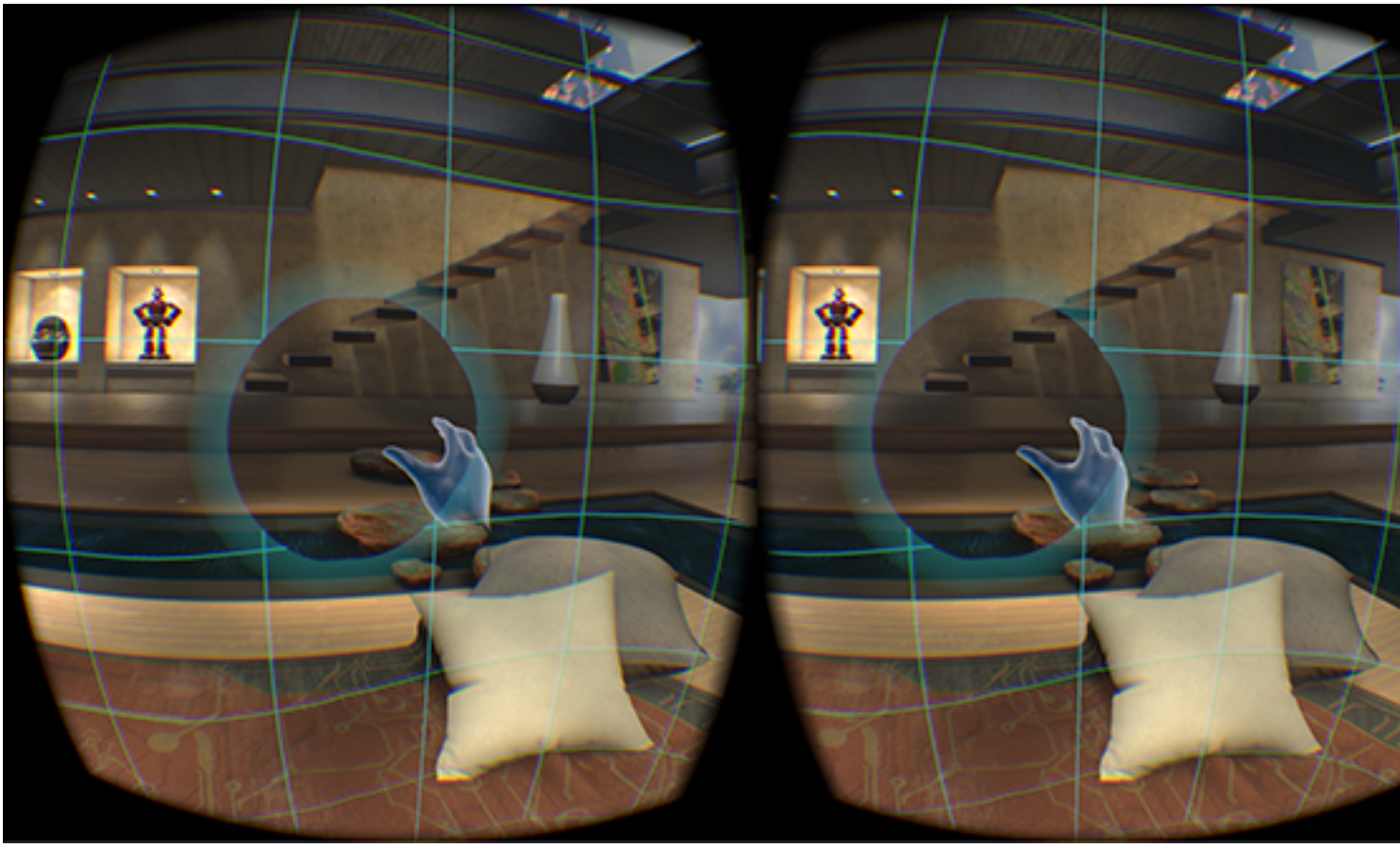


OVRBoundary Guardian System API

OVRBoundary exposes an API for interacting with the Rift Guardian System for Touch.

The Oculus Guardian System is an in-VR visualization of Play Area bounds for Touch users. The boundary visualization is handled automatically by Oculus software, but developers may interact with the Guardian System in various ways using the OVRBoundary API. Possible use cases include pausing the game if the user leaves the Play Area, or placing geometry in the world based on boundary points to create a "natural" integrated barrier with in-scene objects.

During Touch setup, users define an interaction area by drawing a perimeter called the Outer Boundary in space with the controller. An axis-aligned bounding box called the Play Area is calculated from this perimeter.



When tracked devices approach the Outer Boundary, the Oculus runtime automatically provides visual cues to the user demarcating the Outer Boundary. This behavior may not be disabled or superseded by applications, though the Guardian System visualization may be disabled via user configuration in the Oculus App.

Note: The Guardian System visualization is not visible in the Play View of the Editor, but behaves normally

See OVRBoundary in our *Developer Reference* for additional details.

Basic Use

Boundaries are `BoundaryType.OuterBoundary` and `BoundaryType.PlayArea`.

Node types are `Node.HandLeft`, `Node.HandRight`, and `Node.Head`.

Applications may query the location of nodes relative to the Outer Boundary or Play Area by using `OVRBoundary.BoundaryTestResult TestNode()`, which takes the node and boundary type as arguments.

Applications may also query arbitrary points relative to the Play Area or Outer Boundary using `OVRBoundary.BoundaryTestResult TestPoint()`, which takes the point coordinates in the tracking space as a Vector3 and boundary type as arguments.

Results are returned as a struct called `OVRBoundary.BoundaryTestResult`, which includes the following members:

Member	Type	Description
IsTriggering	bool	Returns true if the node or point triggers the queried boundary type.
ClosestDistance	float	Distance between the node or point and the closest point of the test area.
ClosestPoint	Vector3	Describes the location in tracking space of the closest boundary point to the queried node or point.
ClosestPointNormal	Vector3	Describes the normal of the boundary point that is closest to the queried node or point.

Applications may request that boundaries be displayed or hidden using `OVRBoundary.SetVisible()`. Note that the Oculus runtime will override application requests under certain conditions. For example, setting Boundary Area visibility to false will fail if a tracked device is close enough to trigger the boundary's automatic display, and setting the visibility to true will fail if the user has disabled the visual display of the boundary system.

Applications may query the current state of the boundary system using `OVRBoundary.GetVisible()`.

Additional Features

You may set the boundary color of the automated Guardian System visualization using `OVRBoundary.SetLookAndFeel()`. Alpha is unaffected. Use `ResetLookAndFeel()` to reset.

`OVRBoundary.GetGeometry()` returns an array of up to 256 points that define the Boundary Area or Play Area in clockwise order at floor level. You may query the dimensions of a Boundary Area or Play Area using `OVRBoundary.GetDimensions()`, which returns a Vector3 containing the width, height, and depth in tracking space units, with height always returning 0.

OVRHaptics for Oculus Touch

This guide reviews OVRHaptics and OVRHapticsClip, two C# scripts that programmatically control haptics feedback for the Oculus Touch controller.

Haptics Clips

Haptics clips specify the data used to control haptic vibrations in Touch controllers.

Vibrations are specified by an array of bytes or "samples," which specify vibration strength from 0-255. This data can be sent to the left and right touch controllers independently, which process the amplitudes at a sample rate of 320 Hz. The duration of vibration is determined by the number of bytes sent to the devices.

Haptics clips may be created in different ways, depending on your needs. For example, you may manually create a clip with a pre-allocated fixed size buffer, and then write in bytes procedurally. This allows you to generate vibrations on a frame-by-frame basis.

The OVRHaptics class is used to produce the actual vibrations. It defines a LeftChannel and a RightChannel. You can also access these channels through the aliased Channels property, where Channels[0] maps to LeftChannel, and Channels[1] maps to RightChannel. This alias is useful when using a variable for the channel index in a script that can be associated with either hand..

Once you have selected a haptics channel, you may perform four operations with the following OVRHapticsChannel member functions:

- Queue(OVRHapticsClip clip): Queues up a clip.
- Preempt(OVRHapticsClip clip): Removes any previously existing clips already in the queue, and queues up the provided clip; useful for per-frame scenarios.
- Mix(OVRHapticsClip clip): Performs a simple sum and clip mix of the provided clip with any existing clips already in the queue. Can be useful to play multiple clips simultaneously. For example, firing a shotgun in a scene while a dinosaur is stomping by.
- Clear(): Removes all pending clips in the queue and stops haptics for the current channel.

See our *Developer Reference* for API documentation and details on the relevant classes and members.

Generating Haptics Clips from AudioClips

The OVRHapticsClip(AudioClip audioClip, int channel = 0) constructor allows applications to read in a audio clip and generate haptics clips that correspond in strength to the audio clip's amplitude (i.e., volume). You may use monophonic audio clips, or access the left or right channel of a stereo audio clip with the optional channel parameter (default 0 = left stereo/mono, 1 = right stereo). See scripting reference *here*.

OVRHapticsClip reads in an audio clip, downsamples the audio data to a sequence of bytes with the expected sample rate and amplitude range, and feeds that data into the clip's internal amplitude buffer.

We generally recommend AudioClip-generated haptics clips for static sound effects such as gunshots or music that do not vary at runtime. However, you may wish to write your own code to pipe the audio output of an audio source in realtime to a OVRHapticsClip, allowing you near-realtime conversion of audio into corresponding haptics data.

Best Practices

The Rift must be worn in order for haptics to work, as the Oculus runtime only allows the currently-focused VR app to receive Touch haptics.

It is important to keep your sample pipeline at around the right size. Assuming a haptic frequency of 320 Hz and an application frame rate of 90 Hz, we recommend targeting a buffer size of around 10 clips per frame. This allows you to play 3-4 haptics clips per frame, while preserving a buffer zone to account for any asynchronous interruptions. The more bytes you queue, the safer you are from interruptions, but you add additional latency before newly queued vibrations will be played.

Note: For use with Oculus Touch only.

Advanced Rendering Features

This guide describes advanced rendering features that can assist performance.

VR Compositor Layers

OVROverlay is a script in OVR/Scripts that allows you to render Game Objects as VR Compositor Layers instead of drawing them to the eye buffer.

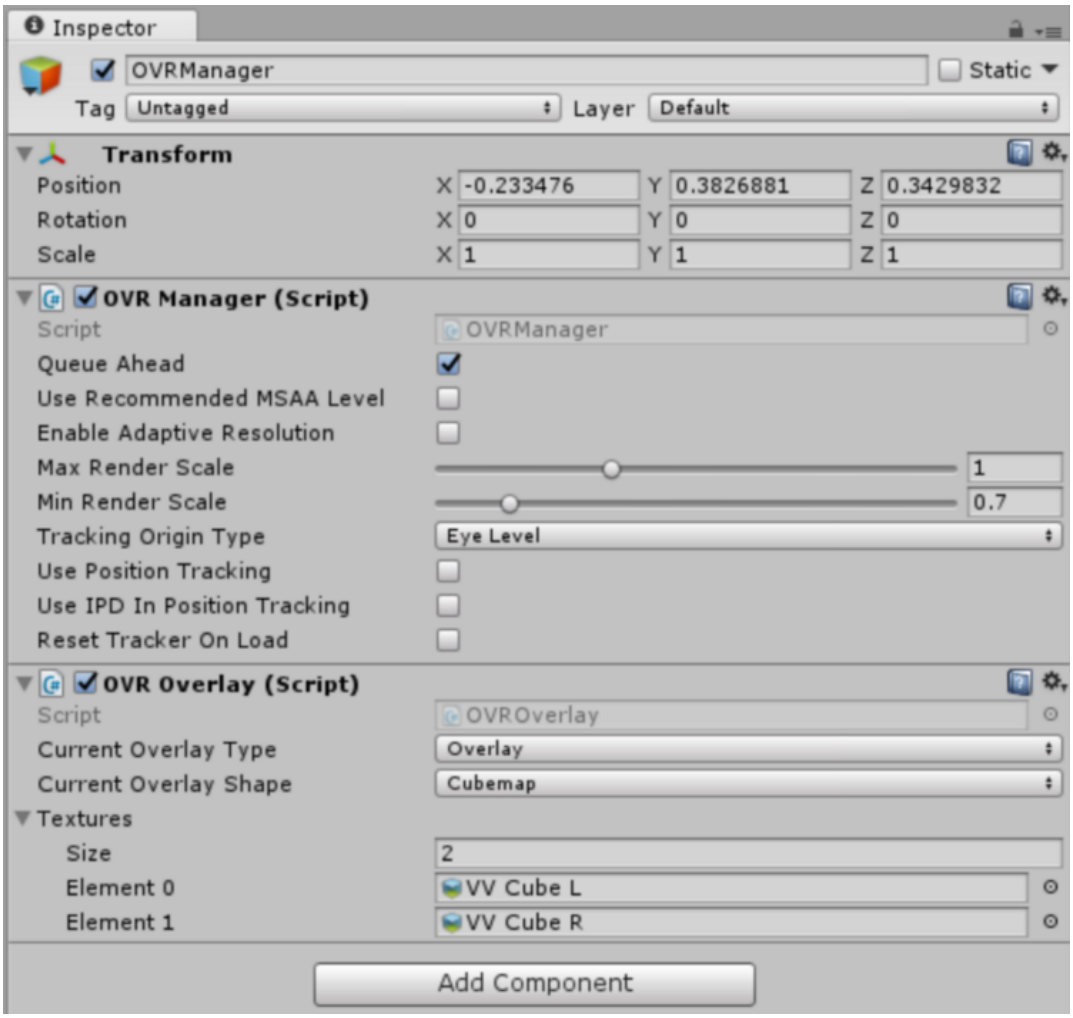
OVROverlay

Game Objects rendered as VR compositor layers render at the frame rate of the compositor instead of rendering at the application frame rate. They are less prone to judder, and they are raytraced through the lenses, improving the clarity of textures displayed on them. This is useful for displaying easily-readable text.

Quadrilateral and cubemap compositor layers are currently supported by Rift and mobile. Cylinder and offset cubemap compositor layers are currently available in mobile only.

Overlay sample: A sample illustrating the use of quad and cylinder VR Compositor Layers for a UI is included with the rendering samples of our Unity Sample Framework. See *Unity Sample Framework* on page 72 for more information.

All layer types support both stereoscopic and monoscopic rendering, though stereoscopic rendering only makes sense for cubemaps in most cases. Stereoscopically-rendered overlays require two textures, specified by setting Size to 2 in the Textures field of OVROverlay in the Inspector.



Gaze cursors and UIs are good candidates for rendering as quadrilateral compositor layers. Cylinders may be useful for smooth-curve UI interfaces. Cubemaps may be used for startup scenes or skyboxes.

We recommend using a cubemap compositor layer for your loading scene, so it will always display at a steady minimum frame rate, even if the application performs no updates whatsoever.

Applications may add three compositor layers to a scene. You may use no more than one cylinder and one cubemap compositor layer per scene.

Note that if a compositor layer fails to render (e.g., you attempt to render more than three compositor layers), only quads will currently fall back and be rendered as scene geometry. Cubemaps and cylinders will not display at all, but similar results can be achieved with scene geometry such as Unity's Skybox component or Cylinder MeshFilter.

You may use OVRRTOverlayConnector to render textures to a compositor layer. See OVRRTOverlayConnector below for more information.

Ordering and Transparency

The depth ordering of compositor layers is controlled by two factors:

1. Whether objects are rendered in front of or behind the scene geometry rendered to the eye buffer, and
2. The sequence in which the compositor layers are enabled in the scene.

By default, VR compositor layers are displayed as overlays in front of the eye buffer. To place them behind the eye buffer, set *Current Overlay Type* to Underlay in the Inspector. Note that underlay compositor layers are more bandwidth-intensive, as the compositor must "punch a hole" in the eye buffer with an alpha mask so that underlays are visible. Texture bandwidth is often a VR bottleneck, so use them with caution and be sure to assess their impact on your application.

Underlays depend on the alpha channel of the render target. If a scene object that should occlude an underlay is opaque, set its alpha to 1. If the occluder is transparent, you must use the OVRUnderlayTransparentOccluder shader provided in the Utilities in Assets/OVR/Shaders. Overlays do not require any special handling for transparency.

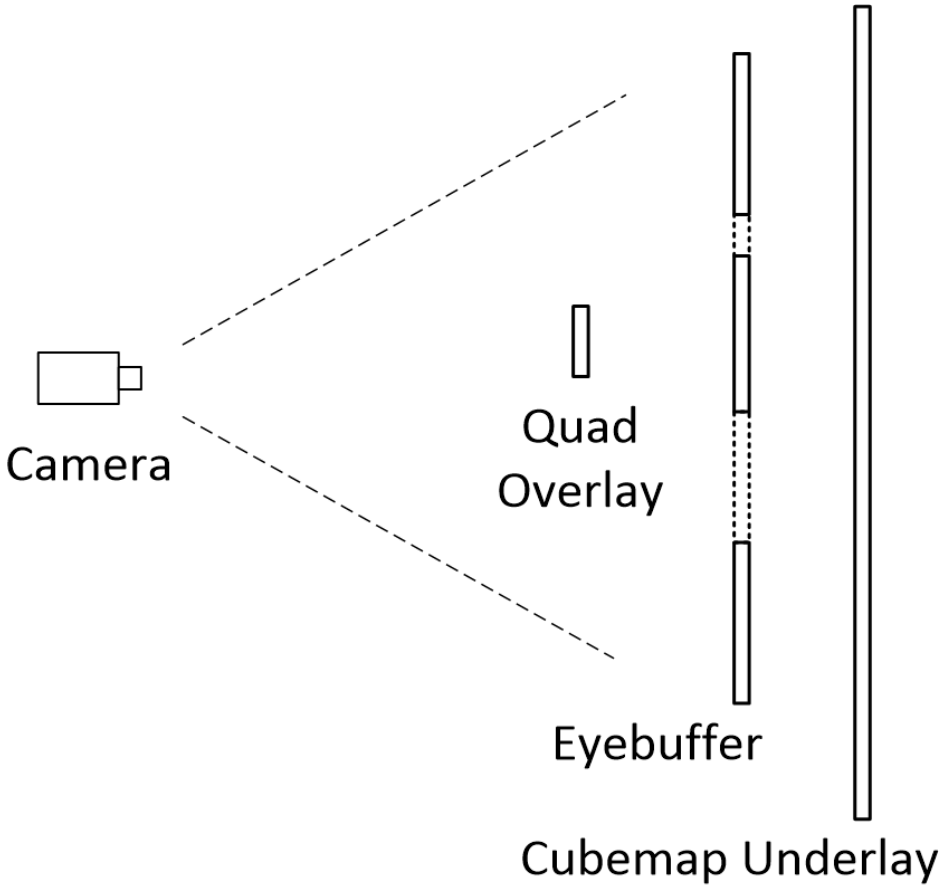
Compositor layers are depth ordered by the sequence in which they are enabled in the scene, but the order is reversed for overlays and underlays. Underlays should be enabled in the scene in the sequence in which you want them to appear, enabling the underlays in front first and the layers in the back last. Overlays should be enabled in the opposite order.

Basic usage

1. Attach OVROverlay.cs to a Game Object.
2. Specify the *Current Overlay Shape*:
    - Quad (Rift and Mobile)
    - Cubemap (Rift and Mobile)
    - Cylinder (Mobile only)
3. Specify the OVROverlay Texture. If you leave it as *None* (default), it will use the Renderer material's main texture, if available. See OVRRTOverlayConnector below for more information on rendering textures to an OVROverlay Game Object.
4. Disable all compositor layers (both overlays and underlays),
5. Enable them sequentially to set the order in which you wish them to appear, enabling overlays in front last and underlays in front first.

Example

In this example, most of the scene geometry is rendered to the eye buffer. The application adds a gaze cursor as a quadrilateral monoscopic overlay and a skybox as a monoscopic cubemap underlay behind the scene.



Note the dotted sections of the eye buffer, indicating where OVROverlay has "punched a hole" to make the skybox visible behind scene geometry.

In this scene, the quad would be set to *Current Overlay Type: Overlay* and the cubemap would be set to *Current Overlay Type: Underlay*. Both would be disabled, then the quad overlay enabled, then the skybox enabled.

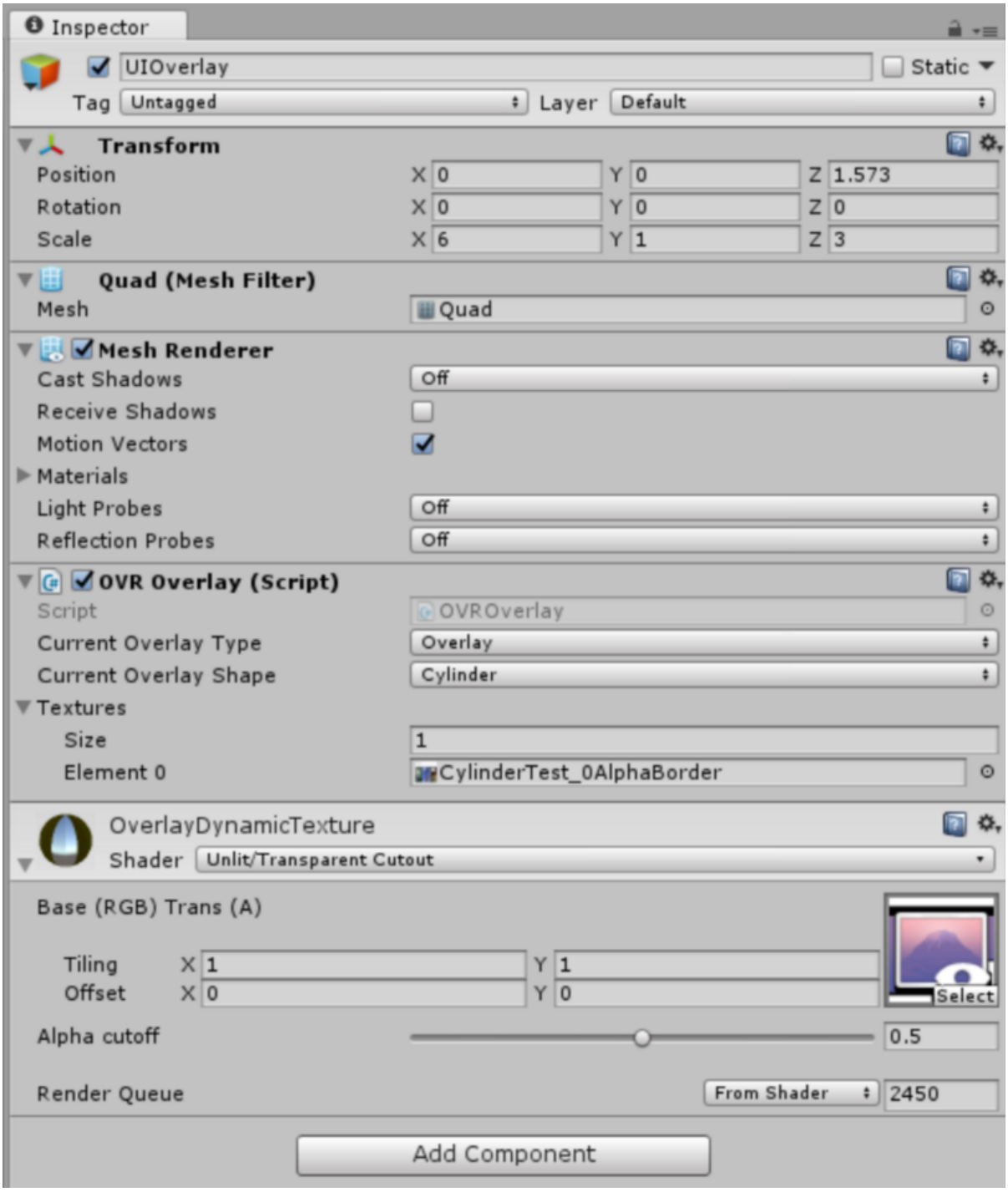
Note that if the cubemap in our scene were transparent, we would need to use the OVRUnderlayTransparentOccluder, which is required for any underlay with alpha less than 1. If it were stereoscopic, we would need to specify two textures and set Size to 2.

Cylinder and Offset Cubemap Overlays (Mobile Only)

The center of a cylinder overlay Game Objects is used as the cylinder's center. The dimensions of the cylinder are encoded in transform.scale as follows:

- [scale.z] cylinder radius
- [scale.y] cylinder height
- [scale.x] length of the cylinder arc

To use a cylinder overlay, your camera must be placed inside the inscribed sphere of the cylinder. The overlay will fade out automatically when the camera approaches to the inscribed sphere's surface.



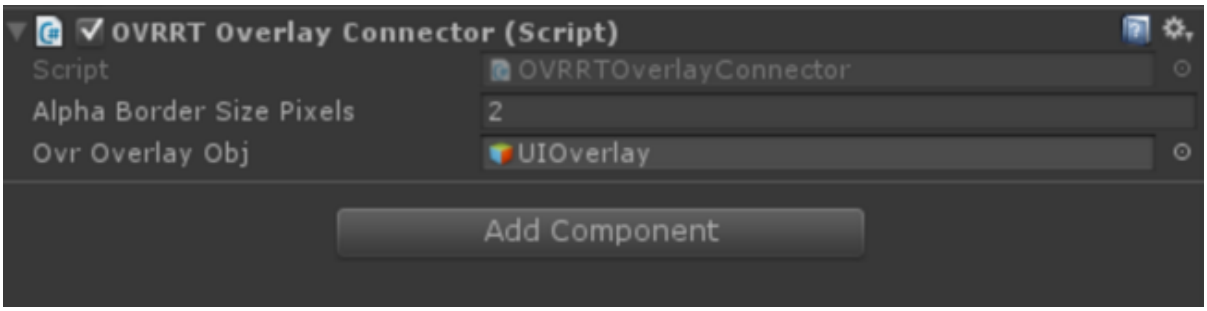
Only half of the cylinder may be displayed, so the arc angle must be smaller than 180 degrees.

Offset cubemap compositor layers are useful for increasing resolution for areas of interest/visible areas by offsetting the cubemap sampling coordinate.

They are similar to the same as standard cubemap compositor layers. Attach the OVROverlay script to an Empty Game Object, and specify the texture coordinate offset in the Position Transform. For more information, see OVROverlay in our *Unity Scripting Reference* on page 79.

OVRRTOverlayConnector

OVRRTOverlayConnector is a helper class in OVR/Scripts/Util used to link a Render Texture to an OVROverlay Game Object. Attach this script to your camera object, and specify your overlay owner object in `Ovr Overlay Obj`.



The overlay camera must use Render Texture, and must be rendered before the Main Camera (e.g., using camera depth), so the Render Texture will be available before being used.

OVRRTOverlayConnector triple-buffers the render results before sending them to the overlay, which is a requirement for time warping a render target. It also clears the render Texture's border to `alpha = 0` to avoid artifacts on mobile.

For more information, see "OVRRTOverlayConnector" in our *Unity Scripting Reference* on page 79.

Mobile Development

This section provides guidelines to help your Unity app perform well with Samsung Gear VR.

Good performance is critical for all VR applications, but the limitations inherent to mobile development warrant special consideration.

Other Resources

Related resources:

- *Performance Targets* on page 63
- *Rendering Guidelines* in our Mobile SDK Developer Guide

Rendering Optimization

Be conservative on performance from the start.

- Keep draw calls down.
- Be mindful of texture usage and bandwidth.
- Keep geometric complexity to a minimum.
- Be mindful of fillrate.

Reducing Draw Calls

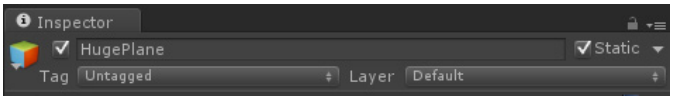
Keep the total number of draw calls to a minimum. A conservative target would be less than **100 draw calls per frame**.

Unity provides several built-in features to help reduce draw calls such as batching and culling.

Draw Call Batching

Unity attempts to combine objects at runtime and draw them in a single draw call. This helps reduce overhead on the CPU. There are two types of draw call batching: Static and Dynamic.

Static batching is used for objects that will not move, rotate or scale, and must be set explicitly per object. To mark an object static, select the Static checkbox in the object Inspector.



Dynamic batching is used for moving objects and is applied automatically when objects meet certain criteria, such as sharing the same material, not using real-time shadows, or not using multipass shaders. More information on dynamic batching criteria may be found here: *https://docs.unity3d.com/Documentation/Manual/DrawCallBatching.html*

Culling

Unity offers the ability to set manual per-layer culling distances on the camera via Per-Layer Cull Distance. This may be useful for culling small objects that do not contribute to the scene when viewed from a given distance. More information about how to set up culling distances may be found here: *https://docs.unity3d.com/Documentation/ScriptReference/Camera-layerCullDistances.html*.

Unity also has an integrated Occlusion Culling system. The advice to early VR titles is to favor modest "scenes" instead of "open worlds," and Occlusion Culling may be overkill for modest scenes. More information about the Occlusion Culling system can be found here: *https://docs.unity3d.com/Manual/OcclusionCulling.html*.

Reducing Memory Bandwidth

- **Texture Compression**: Texture compression offers a significant performance benefit. Favor ASTC compressed texture formats.
- **Texture Mipmaps**: Always use mipmaps for in-game textures. Fortunately, Unity automatically generates mipmaps for textures on import. To see the available mipmapping options, switch *Texture Type* to *Advanced* in the texture inspector.
- **Texture Filtering**: Trilinear filtering is often a good idea for VR. It does have a performance cost, but it is worth it. Anisotropic filtering may be used as well, but keep it to a single anisotropic texture lookup per fragment.
- **Texture Sizes**: Favor texture detail over geometric detail, e.g., use high-resolution textures over more triangles. We have a lot of texture memory, and it is pretty much free from a performance standpoint. That said, textures from the Asset Store often come at resolutions which are wasteful for mobile. You can often reduce the size of these textures with no appreciable difference.
- **Framebuffer Format**: Most scenes should be built to work with a 16 bit depth buffer resolution. Additionally, if your world is mostly pre-lit to compressed textures, a 16 bit color buffer may be used.
- **Screen Resolution**: Setting Screen.Resolution to a lower resolution may provide a sizeable speedup for most Unity apps.

Reduce Geometric Complexity

Keep geometric complexity to a minimum. 50,000 static triangles per-eye per-view is a conservative target.

Verify model vert counts are mobile-friendly. Typically, assets from the Asset Store are high-fidelity and will need tuning for mobile.

Unity Pro provides a built-in **Level of Detail** System (not available in Unity Free), allowing lower-resolution meshes to be displayed when an object is viewed from a certain distance. For more information on how

to set up a LODGroup for a model, see the following: *https://docs.unity3d.com/Documentation/Manual/LevelOfDetail.html*

Verify your vertex shaders are mobile friendly. And, when using built-in shaders, favor the Mobile or Unlit version of the shader.

Bake as much detail into the textures as possible to reduce the computation per vertex: *https://docs.unity3d.com/430/Documentation/Manual/iphone-PracticalRenderingOptimizations.html*

Be mindful of Game Object counts when constructing your scenes. The more Game Objects and Renderers in the scene, the more memory consumed and the longer it will take Unity to cull and render your scene.

Reduce Pixel Complexity and Overdraw

Pixel Complexity: Reduce per-pixel calculations by baking as much detail into the textures as possible. For example, bake specular highlights into the texture to avoid having to compute the highlight in the fragment shader.

Verify your fragment shaders are mobile friendly. And, when using built-in shaders, favor the Mobile or Unlit version of the shader.

Overdraw: Objects in the Unity opaque queue are rendered in front to back order using depth-testing to minimize overdraw. However, objects in the transparent queue are rendered in a back to front order without depth testing and are subject to overdraw.

Avoid overlapping alpha-blended geometry (e.g., dense particle effects) and full-screen post processing effects.

Best Practices

- Be batch friendly. Share materials and use a texture atlas when possible.
- Prefer lightmapped, static geometry.
- Prefer lightprobes instead of dynamic lighting for characters and moving objects.
- Always use baked lightmaps rather than precomputed GI.
- Use Non-Directional Lightmapping.
- Bake as much detail into the textures as possible. E.g., specular reflections, ambient occlusion.
- Only render one view per eye. No shadow buffers, reflections, multi-camera setups, et cetera.
- Keep the number of rendering passes to a minimum. No dynamic lighting, no post effects, don't resolve buffers, don’t use grabpass in a shader, et cetera.
- Avoid alpha tested / pixel discard transparency. Alpha-testing incurs a high performance overhead. Replace with alpha-blended if possible.
- Keep alpha blended transparency to a minimum.
- Be sure to use texture compression. We recommend using ASTC texture compression as a global setting.
- Check the *Disable Depth and Stencil** checkbox in the *Resolution and Presentation* pane in *Player Settings*.
- Be sure to initialize GPU throttling, and avoid dangerous values (-1 or >3) See *Power Management* in our Mobile SDK Developer Guide for more information.
- Avoid full screen image effects.
- Be careful using multiple cameras with clears - doing so may lead to excessive fill cost.
- Avoid use of LoadLevelAsync or LoadLevelAdditiveAsync. This has a dramatic impact on framerate, and it is generally better to go to black and load synchronously.
- Avoid use of Standard shader or Standard Specular shader.
- Avoid using Projectors, or use with caution - they can be very expensive.

- Avoid Unity's Default-Skybox, which is computationally expensive for mobile. We recommend setting Skybox to None (Material), and Ambient Source to Color in Window > Lighting. You may also wish to set Camera.clearFlags to SolidColor (never Skybox).

CPU Optimizations

- Be mindful of the total number of Game Objects and components your scenes use.
- Model your game data and objects efficiently. You will generally have plenty of memory.
- Minimize the number of objects that actually perform calculations in `Update()` or `FixedUpdate()`.
- Reduce or eliminate physics simulations when they are not actually needed.
- Use object pools to respawn frequently-used effects or objects instead of allocating new ones at runtime.
- Use pooled AudioSources versus PlayOneShot sounds, as the latter allocate a Game Object and destroy it when the sound is done playing.
- Avoid expensive mathematical operations whenever possible.
- Cache frequently-used components and transforms to avoid lookups each frame.
- Use the Unity Profiler to:
  - Identify expensive code and optimize as needed.
  - Identify and eliminate Garbage Collection allocations that occur each frame.
  - Identify and eliminate any spikes in performance during normal play.
- Do not use Unity's `OnGUI()` calls.
- Do not enable gyro or the accelerometer. In current versions of Unity, these features trigger calls to expensive display calls.
- All best practices for mobile app and game development generally apply.

Startup Sequence and Reserved Interactions

Startup Sequence

For good VR experiences, all graphics should be rendered such that the user is always viewing a proper three-dimensional stereoscopic image. Additionally, head-tracking must be maintained at all times. We recommend considering using a cubemap overlay for your startup screen (see *VR Compositor Layers* on page 49), which will render at a consistent frame rate even if the application is unavailable to update the scene.

An example of how to do this during application start up is demonstrated in the SDKExamples Startup_Sample scene:

- Solid black splash image is shown for the minimum time possible.
- A small test scene with 3D logo and 3D rotating widget or progress meter is immediately loaded.
- While the small start up scene is active, the main scene is loaded in the background.
- Once the main scene is fully loaded, the start scene transitions to the main scene using a fade.

Universal Menu

Applications must handle Back Key short-press action, which are generally treated as generic back-actions appropriate to the application's state. Back Key long-presses are reserved, and always open the Universal Menu. This is handled automatically by the VrAPI as of Mobile SDK 1.0.4, supported by Utilities for Unity version 1.0.9 and later. For more information, see *Universal Menu and Reserved User Interactions* in our Mobile Developer Guide.

For sample implementation of Universal Menu support for mobile, see the GlobalMenu_Sample in the Utilities folder Assets/Scenes.

See the class description of OVRPlatformMenu in our *Unity Scripting Reference* on page 79 for details about the relevant public members.

Volume

The volume buttons are reserved, and volume adjustment on the Samsung device is handled automatically. Volume control dialog is also handled automatically by the VrApi as of Mobile SDK 1.0.3, supported by Utilities for Unity versions 1.5.0 and later. Do not implement your own volume handling display, or users will see two juxtaposed displays.

It is possible to override automatic volume display handling by setting VRAPI_FRAME_FLAG_INHIBIT_VOLUME_LAYER as an ovrFrameParm flag.

Unity Audio

This guide describes guidelines and resources for creating a compelling VR audio experience in Unity.

If you're unfamiliar with Unity's audio handling, we recommend starting with the *Unity Audio guide*.

General Best Practices

- Do not use more than 16 audio sources.
- Avoid using Decompress on Load for audio clips.
- Do not use ONSP reflections for mobile applications.
- Disable Preload Audio Data for all individual audio clips.

Unity Audio and Rift

Audio input and output automatically use the Rift microphone and headphones unless configured to use the Windows default audio device by the user in the Oculus app. Events OVRManager.AudioOutChanged and AudioInChanged occur when audio devices change, making audio playback impossible without a restart.

For instructions on using Unity 5 and Wwise with the Rift, see *Rift Audio* in the PC SDK Developer Guide.

The Oculus Audio SDK and Audio Spatialization

The Oculus Audio SDK provides free, easy-to-use spatializer plugins for engines and middleware including Unity 5. Our spatialization features support both Rift and Gear VR development.

Our ability to localize audio sources in three-dimensional space is a fundamental part of how we experience sound. Spatialization is the process of modifying sounds to make them localizable, so they seem to originate from distinct locations relative to the listener. It is a key part of creating presence in virtual reality games and applications.

For a detailed discussion of audio spatialization and virtual reality audio, we recommend reviewing our *Introduction to Virtual Reality Audio* guide.

Oculus Native Spatializer Plugin for Unity (ONSP)

The Oculus Native Spatializer Plugin (ONSP) is an plugin for Unity that allows monophonic sound sources to be spatialized in 3D relative to the user's head location.

The ONSP is built on Unity's Native Audio Plugin, which removes redundant spatialization logic and provides a first-party HRTF.

The ONSP Audio SDK also supports early reflections and late reverberations using a simple 'shoebox model,' consisting of a virtual room centered around the listener's head, with four parallel walls, a floor, and a ceiling at varying distances, each with its own distinct reflection coefficient.

The ONSP is available with the Audio SDK Plugins package from our *Downloads page*. To learn more about it, see our *Oculus Audio SDK Guide* and our *Oculus Native Spatializer for Unity Guide*.

Built-in audio spatialization

Unity versions 5.4.1p1 and later include a basic version of our ONSP built into the editor, which includes HRTF spatialization, but not early reflections and late reverberations. This makes it trivially easy to add basic spatialization to audio point sources in your Unity project.

For more information, see *VR Audio Spatializers* in the Unity Manual, or *First-Party Audio Spatialization (Beta)* in our Oculus Native Spatializer for Unity guide.

Other Features

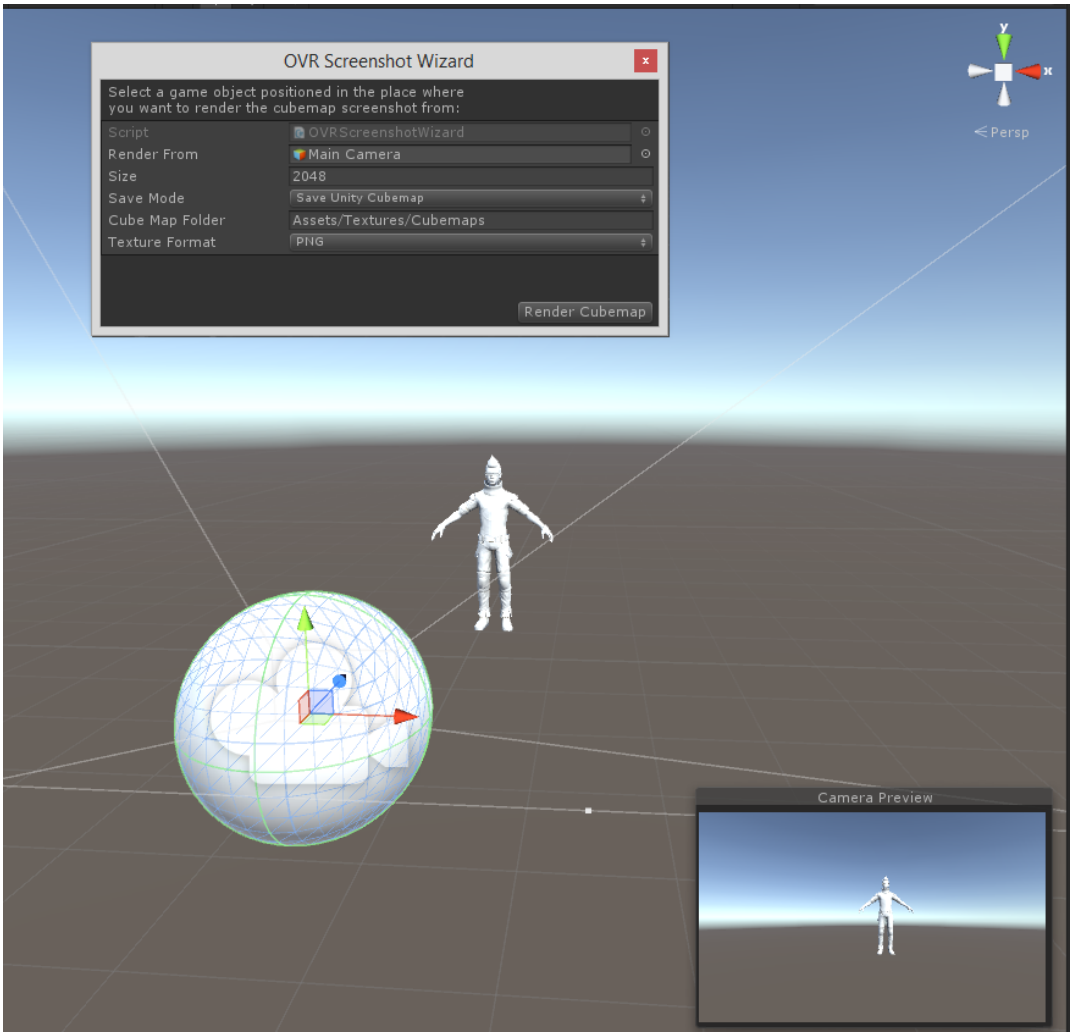
This guide describes miscellaneous tools and features of the Utilities for Unity 5 and Oculus Integration.

Cubemap Screenshots

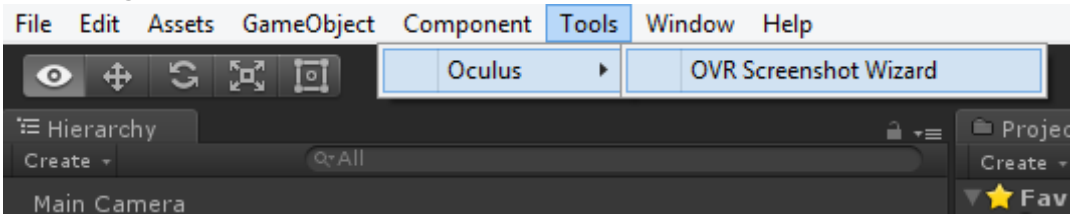
The OVR Screenshot Wizard allows you to easily export a 360 screenshot in cubemap format.

Cubemap previews may be submitted with applications to provide a static in-VR preview for the Oculus Store. For more information, see *Oculus Store Art Guidelines* (PDF).

You may also use OVRCubemapCaptureProbe to take a 360 screenshot from a running Unity app. (see *Prefabs* on page 27 for more information).



Basic Usage

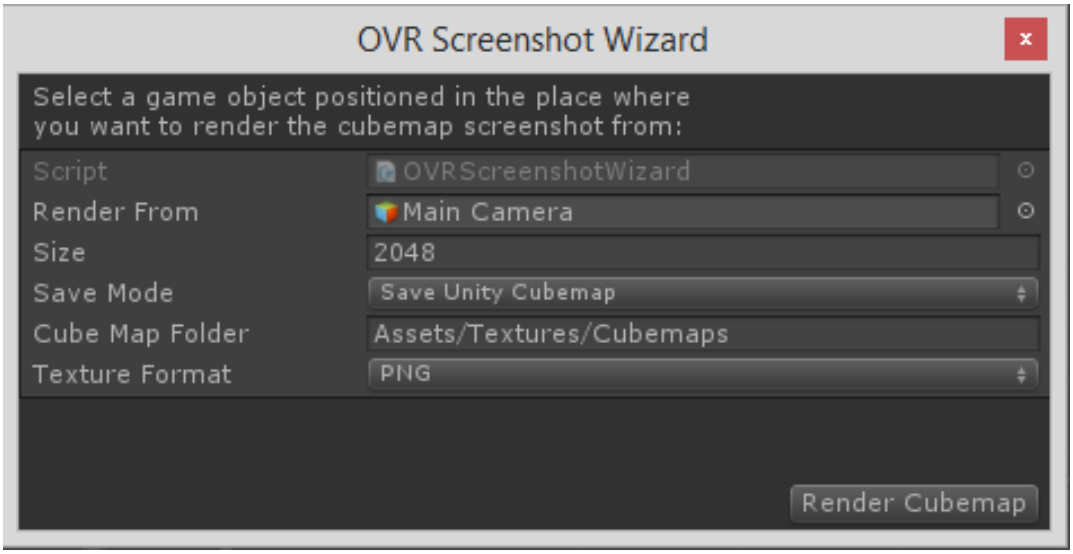


When you import the Oculus Utilities OVRScreenshotWizard into your project, it will add a new *Tools* pull-down menu to your menu bar. Select *Tools* > *Oculus* > *OVR Screenshot Wizard* to launch the tool.

By default, the screenshot will be taken from the perspective of your Main Camera. To set the perspective to a different position, assign any Game Object to the *Render From* field in the Wizard and click *Render Cubemap* to save.

The generated cubemap may be saved either as a Unity Game Object, or as a horizontal 2D atlas texture in PNG or JPG format with the following face order (Horizontal left to right): +x, -x, +y, -y, +z, -z.

Options



Render From: You may use any Game Object as the "camera" that defines the position from which the cubemap will be captured.

To assign a Game Object to function as the origin perspective, select any instantiated Game Object in the Hierarchy View and drag it here to set it as the rendering position in the scene. You may then position the Game Object anywhere in the scene.

If you do not specify a Game Object in this field, the screenshot will be taken from the Main Camera.

Note: If the Game Object extends into the visible area of the scene, it will be included in the capture. This may be useful if you wish to lock art to the origin point, e.g., if you wished to show looking out on the scene from a cage, for example. If you do not want the Game Object to be visible, be sure to use a simple object like a cube or a sphere, or simply use the scene Main Camera.

Size: Sets the resolution for each "tile" of the cubemap face. For submission to the Oculus Store, select 2048 (default, see *Oculus Store Art Guidelines* for more details).

Save Mode

- Save Cube Map: Generate Unity format Cubemap
- Save Cube Map Screenshot: Generate a horizontal 2D atlas texture, resolution: (6 * Size) * Size
- Both: Save both Unity Cubemap and 2D atlas texture

Cube Map Folder: The directory where OVR Screenshot Wizard creates the Unity format Cubemap. The path must be under the root asset folder "Assets"

Texture Format: Sets the image format of 2D atlas texture (PNG or JPEG).

Render Cubemap: Click the button to generate cubemap.

Note: If Save Mode is set to Save Cube Map Screenshot or Both, a pop-up dialog allows you to specify the destination folder where the 2D atlas texture will be generated. You may save it outside of Assets folder if you wish.

Best Practices for Rift and Gear VR

This section describes performance targets and offers recommendations for developers.

General Best Practices

- Use trilinear or anisotropic filtering on textures. See *Textures* in the Unity Manual for more information.
- Use mesh-based occlusion culling (see *Occlusion Culling* in the Unity Manual).
- Always use the Forward Rendering path (see *Forward Rendering Path Details* in the Unity Manual).
- Enable Use Recommended MSAA Levels in OVRManager (see *OVRManager* for more information).
- Watch for excessive texture resolution after LOD bias (greater than 4k by 4k on PC, greater than 2k by 2k on mobile).
- Verify that non-static objects with colliders are not missing rigidbodies in themselves or in the parent chain.
- Avoid inefficient effects such as SSAO, motion blur, global fog, parallax mapping.
- Avoid slow physics settings such as Sleep Threshold values of less than 0.005, Default Contact Offset values of less than 0.01, or Solver Iteration Count settings greater than 6.
- Avoid excessive use of multipass shaders (e.g., legacy specular).
- Avoid large textures or using a lot of prefabs in startup scenes (for bootstrap optimization). When using large textures, compress them when possible.
- Avoid realtime global illumination.
- Disable shadows when approaching the geometry or draw call limits.
- Avoid excessive pixel lights (>1 on Gear VR; >3 on Rift).
- Avoid excessive render scale (>1.2).
- Avoid excessive shader passes (>2).
- Be cautious using Unity WWW and avoid for large file downloads. It may be acceptable for very small files.

Testing and Performance Analysis

In this guide, we'll review baseline targets, recommendations, tools, resources, and common workflows for performance analysis and bug squashing for Unity VR applications.

General Tips

VR application debugging is a matter of getting insight into how the application is structured and executed, gathering data to evaluate actual performance, evaluating it against expectation, then methodically isolating and eliminating problems.

When analyzing or debugging, it is crucial to proceed in a controlled way so that you know specifically what change results in a different outcome. Focus on bottlenecks first. Only compare apples to apples, and change one thing at a time (e.g., resolution, hardware, quality, configuration).

Always be sure to profile, as systems are full of surprises. We recommend starting with simple code, and optimizing as you go - don't try to optimize too early.

We recommend creating a 2D, non-VR version of your camera rig so you can swap between VR and non-VR perspectives. This allows you to spot check your scenes, and it may be useful if you want to do profiling with third-party tools (e.g., Adreno Profiler).

It can be useful to disable *Multithreaded Rendering* in *Player Settings* during performance debugging. This will slow down the renderer, but also give you a clearer view of where your frame time is going. Be sure to turn it back on when you're done!

Performance Targets

Before debugging performance problems, establish clear targets to use as a baseline for calibrating your performance.

These targets can give you a sense of where to aim, and what to look at if you're not making frame rate or are having performance problems.

Below you will find some general guidelines for establishing your baselines, given as approximate ranges unless otherwise noted.

Mobile

- 60 FPS (required by Oculus)
- 50-100 draw calls per frame
- 50,000-100,000 triangles or vertices per frame

PC

- 90 FPS (required by Oculus)
- 500-1,000 draw calls per frame
- 1-2 million triangles or vertices per frame

For more information, see:

- *PC SDK Developer Guide*
- *Mobile Development* on page 54

Unity Profiling Tools

This section details tools provided by Unity to help you diagnose application problems and bottlenecks.

Unity Profiler

Unity comes with a built-in profiler (see Unity's *Profiler manual*). The Unity Profiler provides per-frame performance metrics, which can be used to help identify bottlenecks.

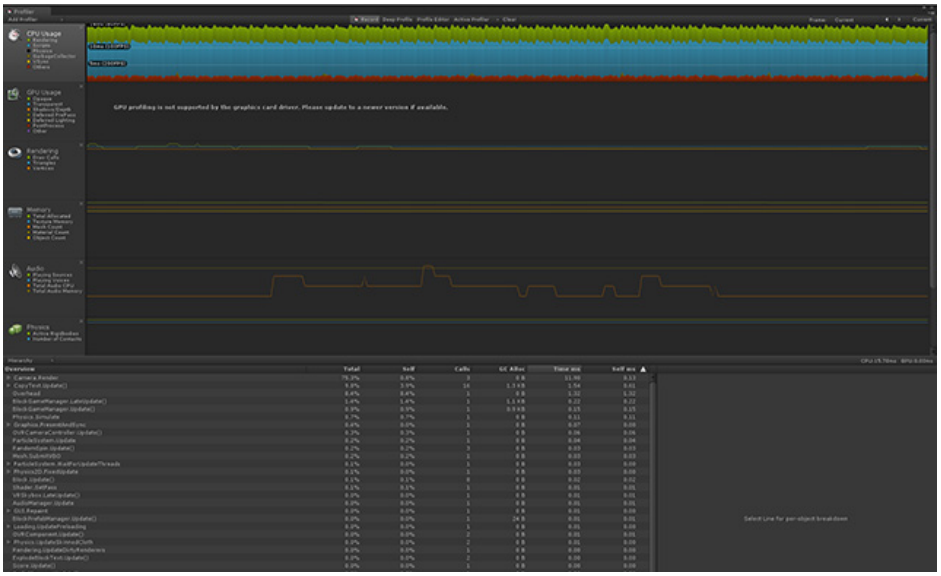
Unity Pro comes with a built-in profiler. The profiler provides per-frame performance metrics, which can be used to help identify bottlenecks.

PC Setup

To use Unity Profiler with a Rift application, select *Development Build* and *Autoconnect Profiler* in *Build Settings* and build your application. When you launch your application, the Profiler will automatically open.

Mobile Setup

You may profile your application as it is running on your Android device using adb or Wi-Fi. For steps on how to set up remote profiling for your device, please refer to the Android section of the following Unity documentation: *https://docs.unity3d.com/Documentation/Manual/Profiler.html*.



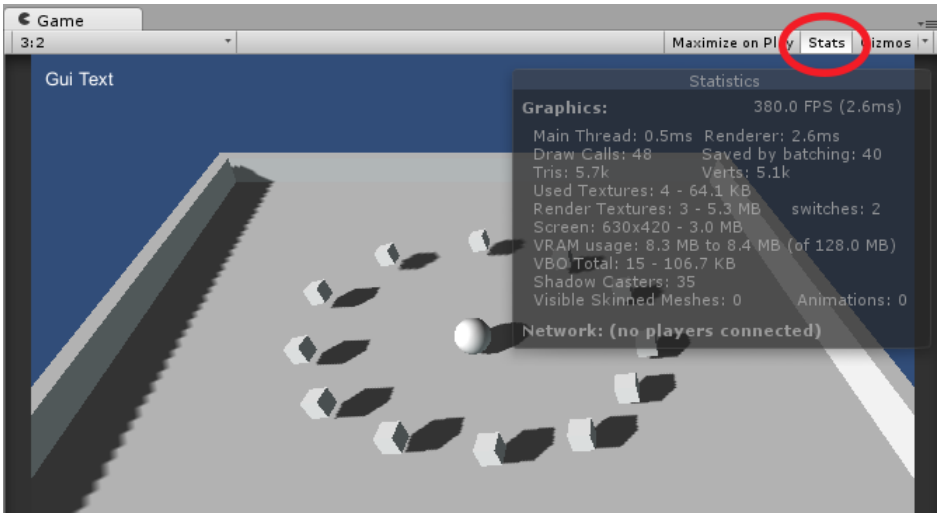
The Unity Profiler displays CPU utilization for the following categories: Rendering, Scripts, Physics, GarbageCollector, and Vsync. It also provides detailed information regarding Rendering Statistics, Memory Usage (including a breakdown of per-object type memory usage), Audio and Physics Simulation statistics.

GPU Usage data for Android is not available at this time.

The Unity profiler only displays performance metrics for your application. If your app isn't performing as expected, you may need to gather information on what the entire system is doing.

Show Rendering Statistics

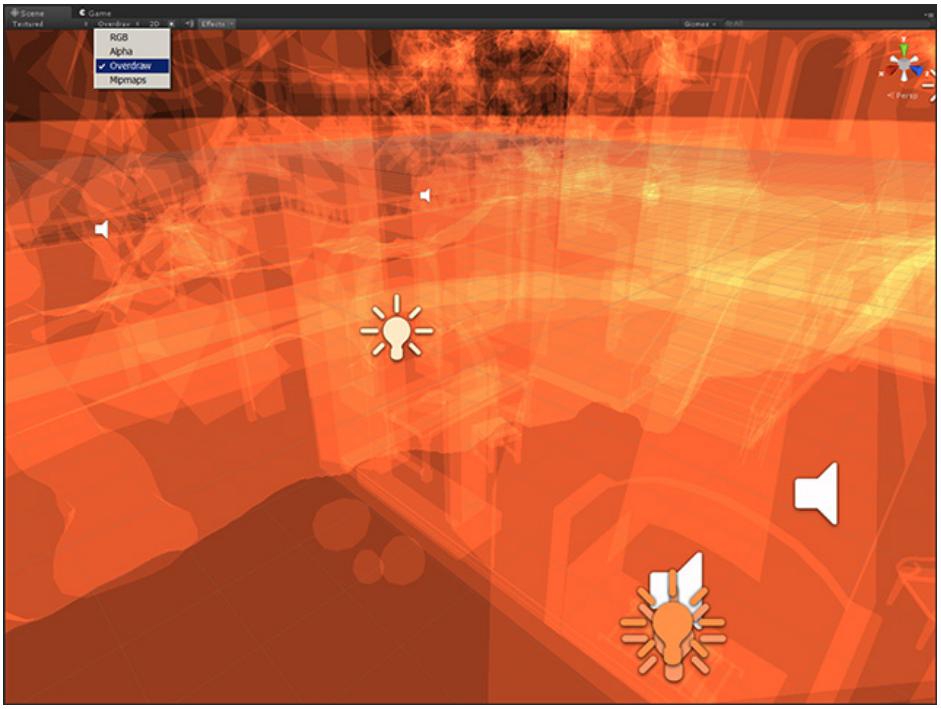
Unity provides an option to display real-time rendering statistics, such as FPS, Draw Calls, Tri and Vert Counts, VRAM usage. While in the Game View, pressing the Stats button above the Game View will display an overlay showing realtime render statistics. Viewing stats in the Editor can help analyze and improve batching for your scene by indicating how many draw calls are being issued and how many are being saved by batching (the OverDraw render mode is helpful for this as well).



Show GPU Overdraw

Unity provides a specific render mode for viewing overdraw in a scene. From the Scene View Control Bar, select OverDraw in the drop-down Render Mode selection box.

In this mode, translucent colors will accumulate providing an overdraw "heat map" where more saturated colors represent areas with the most overdraw.



Unity Frame Debugger

Unity Frame Debugger lets you walk through the order of draw calls for any scene. Even if you're not actively debugging, it can be very useful for understanding how Unity is putting your scene together. This can be very helpful in debugging pipeline problems.

For more information, see *Frame Debugger in Unity 5.0*.

Unity Built-in Profiler

Unity Built-in Profiler (not to be confused with Unity Profiler) provides frame rate statistics through logcat, including the number of draw calls, min/max frametime, number of tris and verts, et cetera.

To use this profiler, connect to your device over Wi-Fi using ADB over TCPIP as described in the *Wireless usage* section of Android's adb documentation. Then run `adb logcat` while the device is docked in the headset.

See *Unity's Measuring Performance with the Built-in Profiler* for more information. For more on using adb and logcat, see *Android Debugging* in the Mobile SDK documentation.

Performance Auditing Tool

The performance auditing tool may be used to verify that your Rift or mobile VR project configuration and settings are consistent with our recommendations.

For example, after running the Performance Auditing Tool, you may be prompted to use ASTC compression, or to disable the built-in Unity Skybox. For a look at many of the recommendations we used to establish the auditing baseline, see *Best Practices for Rift and Gear VR* on page 62.

This tool is intended to help verify that your application is performant, and will not specifically evaluate it for submission to the Oculus Store.

Use with Care

Applying recommendations may substantially affect the look and feel of your game. We strongly recommend that you only accept changes that you fully understand. For example, if you accept our recommendation to deactivate the Unity skybox, you will need to replace it with a cube mapped box or sphere and a shader that supports depth testing.

Rift and Mobile Auditing

Performance recommendations differ substantially for Rift and mobile applications. This tool may be used for both platforms. It will evaluate your project based on the target platform selected in *Build Settings*.

To audit a Rift project, select *File* > *Build Settings…*, and under *Platform*, select *PC, Mac, & Linux Standalone*. If *Switch Platform* is not grayed out, click it.

To audit a mobile project, select *File* > *Build Settings…*, and under *Platform*, select *Android*. If *Switch Platform* is not grayed out, click it.

Use

Once you have verified your Build Settings are configured properly, run the Performance Auditing Tool.

1. Open the VR project you want to audit in the Unity editor.
2. If you have not already imported the Utilities for Unity package, do so now (for further instructions, see *Getting Started Guide* on page 5).
3. In the Unity toolbar, select *Tools* > *Oculus* > *Audit Project for VR Performance Issues*.

For each issue the tool finds, you will be provided with the option of automatically updating your settings to bring them in line with our recommendations.

Rift Performance Tools

Oculus Performance Head-Up Display (HUD)

The Oculus Performance Head-Up Display (HUD) is an important, easy-to-use tool for viewing timings for render, latency, and performance headroom in real-time as you run an application in the Oculus Rift. The HUD is easily accessible through the Oculus Debug Tool provided with the PC SDK. For more details, see the *Performance Head-Up Display* and *Oculus Debug Tool* sections of the Oculus Rift Developers Guide.

Compositor Mirror

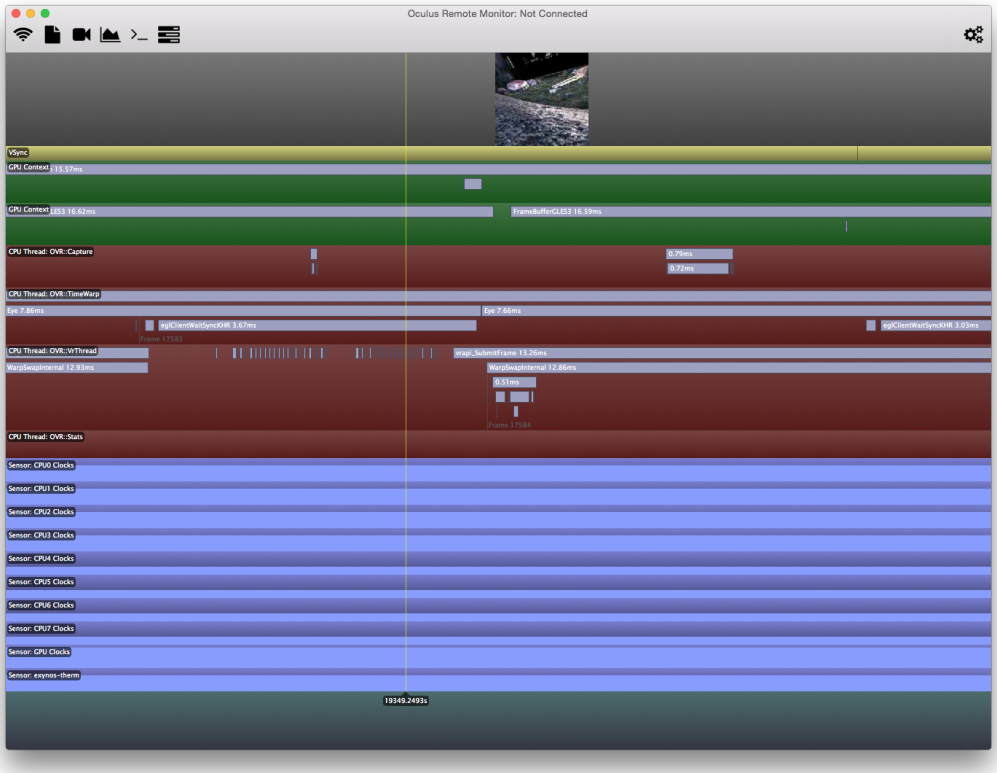
The compositor mirror is an experimental tool for viewing exactly what appears in the headset, with Asynchronous TimeWarp and distortion applied.

The compositor mirror is useful for development and troubleshooting without having to wear the headset. Everything that appears in the headset will appear, including Oculus Home, Guardian boundaries, in-game notifications, and transition fades. The compositor mirror is compatible with any game or experience, regardless of whether it was developed using the native PC SDK or a game engine.

For more details, see the *Compositor Mirror* section of the PC SDK Guide.

Oculus Remote Monitor (Mobile)

The Oculus Remote Monitor client (available for Windows and Mac OS X) connects to VR applications running on remote mobile devices to capture, store, and display the streamed-in data. The VrCapture library is automatically included in Unity 5 projects, so setup and use of the Oculus Remote Monitor is easy.



Oculus Remote Monitor is available from our *Downloads* page. For more information about setup, features, and use, see *Oculus Remote Monitor* in our Mobile SDK guide.

Feature Highlights

- The Frame Buffer Viewer provides a mechanism for inspecting the frame buffer as the data is received in real-time, which is particularly useful for monitoring play test sessions. When enabled, the Capture library will stream a downscaled pre-distortion eye buffer across the network.
- The Performance Data Viewer provides real-time and offline inspection of the following on a single, contiguous timeline:
  - CPU/GPU events
  - Sensor readings
  - Console messages, warnings, and errors
  - Frame buffer captures
- The Logging Viewer provides raw access to various messages and errors tracked by thread IDs.
- Nearly any constant in your code may be turned into a knob that can be updated in real-time during a play test.

Additional Third-Party Tools

ETW + GPUView

Event Tracing for Windows (ETW) is a trace utility provided by Windows for performance analysis. *GPUView* view provides a window into both GPU and CPU performance with DirectX applications. It is precise, has low overhead, and covers the whole Windows system. Custom event manifests.

ETW profiles the whole system, not just the GPU. For a sample debug workflow using ETW to investigate queuing and system-level contention, see Example Workflow: PC below.

Windows 10 replaces ETW with *Tracelogging*.

Systrace

Reports complete Android system utilization. Available here: *http://developer.android.com/tools/help/systrace.html*

NVIDIA NSight

NSight is a CPU/GPU debug tool for NVIDIA users, available in a *Visual Studio version* and an *Eclipse version*.

Mac OpenGL Monitor

An OpenGL debugging and optimizing tool for OS X. Available here: *https://developer.apple.com/library/mac/technotes/tn2178/_index.html#//apple_ref/doc/uid/DTS40007990*

APITrace

https://apitrace.github.io/

Analyzing Slowdown

In this guide, we take a look at three of the areas commonly involved with slow application performance: pixel fill, draw call overhead, and slow script execution.

Pixel Fill

Pixel fill is a function of overdraw and of fragment shader complexity. Unity shaders are often implemented as multiple passes (draw diffuse part, draw specular part, and so forth). This can cause the same pixel to be touched multiple times. Transparency does this as well. Your goal is to touch almost all pixels on the screen only one time per frame.

Unity's Frame Debugger (described in *Unity Profiling Tools* on page 63) is very useful for getting a sense of how your scene is drawn. Watch out for large sections of the screen that are drawn and then covered, or for objects that are drawn multiple times (e.g., because they are touched by multiple lights).

Z-testing is faster than drawing a pixel. Unity does culling and opaque sorting via bounding box. Therefore, large background objects (like your Skybox or ground plane) may end up being drawn first (because the bounding box is large) and filling a lot of pixels that will not be visible. If you see this happen, you can move those objects to the end of the queue manually. See *Material.renderQueue* in Unity's Scripting API Reference for more information.

Frame Debugger will clearly show you shadows, offscreen render targets, et cetera.

Draw Calls

Modern PC hardware can push a lot of draw calls at 90 fps, but the overhead of each call is still high enough that you should try to reduce them. On mobile, draw call optimization is your primary scene optimization.

Draw call optimization is usually about batching multiple meshes together into a single VBO with the same material. This is key in Unity because the state change related to selecting a new VBO is relatively slow. If you select a single VBO and then draw different meshes out of it with multiple draw calls, only the first draw call is slow.

Unity batches well when given properly formatted source data. Generally:

- Batching is only possible for objects that share the same material pointer.
- Batching doesn't work on objects that have multiple materials.

- Implicit state changes (e.g. lightmap index) can cause batching to end early.

Here is a quick checklist for maximizing batching:

- Use as few textures in the scene as possible. Fewer textures require fewer unique materials, so they are easier to batch. Use texture atlases.
- Bake lightmaps at the largest atlas size possible. Fewer lightmaps require fewer material state changes. Gear VR can push 4096 lightmaps without too much trouble, but watch your memory footprint.
- Be careful not to accidentally instance materials. Note that accessing `Renderer.material` automatically creates an instance (!) and opts that object of batching. Use `Renderer.sharedMaterial` instead whenever possible.
- Watch out for multi-pass shaders. Add `noforwardadd` to your shaders whenever you can to prevent more than one directional from applying. Multiple directionals generally break batching.
- Mark all mesh that never moves as `Static` in the editor. Note that this will cause the mesh to be combined into a mega mesh at build time, which can increase load time and app size on disk, though usually not in a material way. You can also create a static batch at runtime (e.g., after generating a procedural level out of static parts) using StaticBatchingUtility.
- Watch your static and dynamic batch count vs the total draw call count using the Profiler, internal profiler log, or stats gizmo.

Script Performance

Unity's C# implementation is fast, and slowdown from script is usually the result of a mistake and/or an inadvertent block on slow external operations such as memory allocation. The Unity Profiler can help you find and fix these scripts.

Try to avoid foreach, lamda, and LINQ structures as these allocate memory needlessly at runtime. Use a for loop instead. Also, be wary of loops that concatenate strings.

Game Object creation and destruction takes time. If you have a lot of objects to create and destroy (say, several hundred in a frame), we recommend pooling them.

Don't move colliders unless they have a rigidbody on them. Creating a rigidbody and setting `isKinematic` will stop physics from doing anything but will make that collider cheap to move. This is because Unity maintains two collider structures, a static tree and a dynamic tree, and the static tree has to be completely rebuilt every time any static object moves.

Note that coroutines execute in the main thread, and you can have multiple instances of the same coroutine running on the same script.

We recommend targeting around 1-2 ms maximum for all Mono execution time.

PC Debug Workflow

In this guide, we'll use the example of a hypothetical stuttering app scene and walk through basic steps debugging steps.

Where to Start

Begin by running the scene with the *Oculus Performance HUD*.

If the scene drops more than one frame every five seconds, check the render time. If it's more than 8 ms, have a look at GPU utilization. Otherwise, look at optimizing CPU utilization. If observed latency is greater than 30 ms, have a look at queuing.

CPU Profiling (Unity Profiler)

Look for the tallest bars in the CPU Usage graph in the Unity Profiler. Sort hierarchy by Total CPU time, and expand to see which objects and calls take the most time.

If you find garbage collection spikes, don't allocate memory each frame.

GPU Profiling (Unity Profiler)

Are your rendering stats too high? (For reference baselines, see *Performance Targets*.

Check for hogs in your hierarchy or timeline view, such as any single object that takes 8 ms to render. The GPU may also wait for long stalls on CPU. Other potential problem areas are mesh rendering, shadows, vsync, and subsystems.

Mobile Tips

Use *Oculus Remote Monitor (Mobile)* on page 66 for VRAPI, render times, and latency. Systrace shows CPU queueing.

It is a common problem to see Gfx.WaitForPresent appear frequently in Oculus Remote Monitor. This reports the amount of time the render pipeline is stalled, so begin troubleshooting by understanding your scene is assembled by Unity - the Unity Frame Debugger is a good starting place. See *Unity Profiling Tools* on page 63 for more information.

Troubleshooting and Known Issues

This section outlines some currently known issues with the Oculus Unity Integration and the Oculus Utilities for Unity.

Rift

The app does not launch as a VR app.

Verify that you have installed the Oculus App and completed setup as described in *Preparing for Rift Development* on page 6.

Verify that you have selected *Virtual Reality Supported* in *Player Settings*.

are using a compatible runtime - see *Compatibility and Requirements* for more details.

Verify that the HMD is plugged in and working normally.

Verify that you have not selected D3D 9 or Windows GL as the renderer (Legacy Integration only).

Mobile

The app does not launch as a VR app.

Verify that you selected *Virtual Reality Supported* in *Player Settings* before building your APK.

Applications fail to launch on Gear VR with error message "thread priority security exception make sure the apk is signed".

You must sign your application with an Oculus Signature File (osig). See "Sign your App with an Oculus Signature File" in *Preparing for Mobile Development* on page 6 for instructions.

General Issues

Unity 5 hangs while importing assets from SDKExamples.

Unity 5 is known to import ETC2-compressed assets very slowly.

Receiving OVRPlugin console errors after importing a new version of Utilities.

Be sure to delete any previously-imported Utilities packages from your Unity project before importing a new version. If you are receiving errors and have not done so, delete the relevant folders in your project and re-import Utilities. For more information, please see *Importing the Oculus Utilities Package* on page 8.

Contact Information

Questions?

Visit our developer support forums at *https://developer.oculus.com*.

Our Support Center can be accessed at *https://support.oculus.com*.

Unity Sample Framework

The Oculus Unity Sample Framework provides sample scenes and guidelines for common VR-specific features such as hand presence with Oculus Touch, crosshairs, driving, hybrid mono rendering, and video rendering to a 2D textured quad.

The Unity Sample Framework can guide developers in producing reliable, comfortable applications and avoiding common mistakes. The assets and scripts included with the Sample Framework may be reused in your applications per the terms of our *SDK 3.4 license*.

It is available as a Unity Package for developers who wish to examine how the sample scenes were implemented, and as binaries for the Rift and Gear VR for developers to explore the sample scenes entirely in VR. The Rift executable is available from our *Downloads Center*, and the Gear VR application is available through the Oculus Store in the Gallery section.

Figure 7: Sample Framework UI Scene

The Unity Sample Framework requires **Unity v 5.4** or later. Please check *Compatibility and Requirements* for up-to-date version recommendations.

Sample Scenes

In the Unity project, the following scenes are found in /Assets/SampleScenes:

Scene	Directory	Concept Illustrated
Multiple Cameras	Cameras/	Switching between cameras in a scene.
Per-Eye Cameras	Cameras/	Using different cameras for each eye for a specific object in the scene.
Crosshairs	First Person/	Using crosshairs to aim a weapon in VR and different configuration options.
Teleport	First Person/ Locomotion/	A teleportation locomotion scene that reduces the risk of discomfort.
Mirror	First Person/	A simple mirror effect.

Scene	Directory	Concept Illustrated
Outdoor Motion	First Person/	Basic forms of movement, and the effects a variety of design choices may have on comfort.
Scale	First Person/	How various scale factors interact.
Stairs	First Person/	Factors affecting comfort in first-person stairs movement.
AvatarWithGrab	Hands/	Uses the Unity Avatar SDK and the scripts OVRGrabber and OVRGrabbable to illustrate hands presence with Touch. Pick up and throw blocks from a table using the Touch grip buttons. This sample requires importing the Oculus Avatar SDK.
CustomControllers	Hands/	A simple sample displaying tracked Touch models in a scene.
CustomHands	Hands/	Uses low-resolution custom hand models and the scripts OVRGrabber and OVRGrabbable to illustrate hands presence with Touch. Pick up and throw blocks from a table using the Touch grip buttons. May be used as a reference for implementing your own hand models.
Input Tester	Input/	This scene assists with testing input devices, displaying axis values in real time.
Keyboard	Input/	A virtual keyboard.
Movie Player	Rendering/	Video rendering to a 2D textured quad using the Android Media Surface Plugin. Source for the plugin ships with the Mobile SDK in \VrAppSupport\MediaSurfacePlugin.
Surface Detail	Rendering/	Different ways to create surface detail with normal, specular, parallax, and displacement mapping.
PerfTest	Rendering/	Demonstrates hybrid mono rendering, in which near content is rendered stereoscopically and distant content is rendered monoscopically using scripts and shaders.
StereoMonoRoom	Rendering/	Another implementation of the hybrid mono rendering feature also demonstrated by PerfTest (see above).
OverlayUIDemo	UI/	Demonstrates creating a UI with a VR Compositor Layer to improve image quality and anti-aliasing. Includes a quad overlay for Rift, and a quad and a cylinder overlay for mobile.
Pointers	UI/	How UI elements can be embedded in a scene and interact with different gaze controllers.
Pointers - Gaze Click	UI/	An extension of the Pointers scene, with gaze selection.
Tracking Volume	UI/	Different ways to indicate the user is about to leave the position tracking volume.

A Note on Comfort

These samples are intended to be tools for exploring design ideas in VR, and should not necessarily be construed as design recommendations. The Sample Framework allows you to set some parameters to values that will reliably cause discomfort in most users - they are available precisely to give developers an opportunity to find out how much is too much.

It is as important to play test your game on a range of players throughout development to ensure your game is a comfortable experience. We have provided in-game warnings to alert you to potentially uncomfortable scenes.

Downloading and Installation

To download the Oculus Sample Framework Unity Project or Rift binary, visit our *Downloads Center*. The Gear VR Sample Framework application may be downloaded for free from the Gallery Apps section of the Oculus Store.

To run the PC Binary

Note: You will need to enable running applications from unknown sources in the Oculus app settings. Launch the Oculus app, and in the "gear" pull-down menu in the upper right, select *Settings* > *General* and toggle *Unknown Sources* on to allow. You may wish to disable this setting after use for security reasons.

1. Download the Unity Sample Framework PC Binary.
2. Unzip the contents.
3. Run the OculusSampleFramework.exe. Note that it must be run from a directory location that also includes the OculusSampleFramework_Data folder.

To open the project in Unity Editor:

1. Verify that you have installed the latest-recommended version of Unity 5 (see *Compatibility and Requirements* for up-to-date information).
2. Download the Unity Sample Framework Project from our *Downloads Center*.
3. Launch the Unity Editor and create a new project.
4. Import the Unity Sample Framework Unity Package by selecting *Assets* > *Import Package* > *Custom Package…* and selecting the Unity Sample Framework.

Building the Unity Project

This is only necessary if you want to experiment with the project, as application binaries are provided by Oculus for free download.

To build the Unity Project for Rift:

Note: You will need to enable running applications from unknown sources in the Oculus app settings. Launch the Oculus app, and in the "gear" pull-down menu in the upper right, select *Settings* > *General* and toggle *Unknown Sources* on to allow. You may wish to disable this setting after use for security reasons.

1. Open the Sample Framework project as described above.
2. From the Editor menu bar, select *OVR* > *Samples Build Config* > *Configure Rift Build*.
3. Build and run the project normally.

To build the Unity Project for the Gear VR:

1. Open the Sample Framework project as described above.
2. From Editor menu bar, select *OVR* > *Samples Build Config* > *Configure Gear VR Build*.
3. Ensure the project contains an Oculus signature file and that you have an Android keystore configured. See *Application Signing* in our Mobile SDK documentation for more details.
4. Sample Framework Android builds use a custom manifest and are not visible from Applications, and cannot be launched from Oculus Home or the Android Application Launcher. To launch:
    a. Install the APK to your phone.
    b. Open *Settings* > *Applications* > *Application Manager* > *Gear VR Service*.
    c. Select *Storage*.
    d. Select *Manage Storage*.
    e. Toggle *Add icon to app list to On*.

f. Close *Settings*.
g. Open *Apps*.
h. Select *Gear VR Service*.
i. Select *Oculus Sample Framework* to launch.

Exploring the Sample Framework in VR

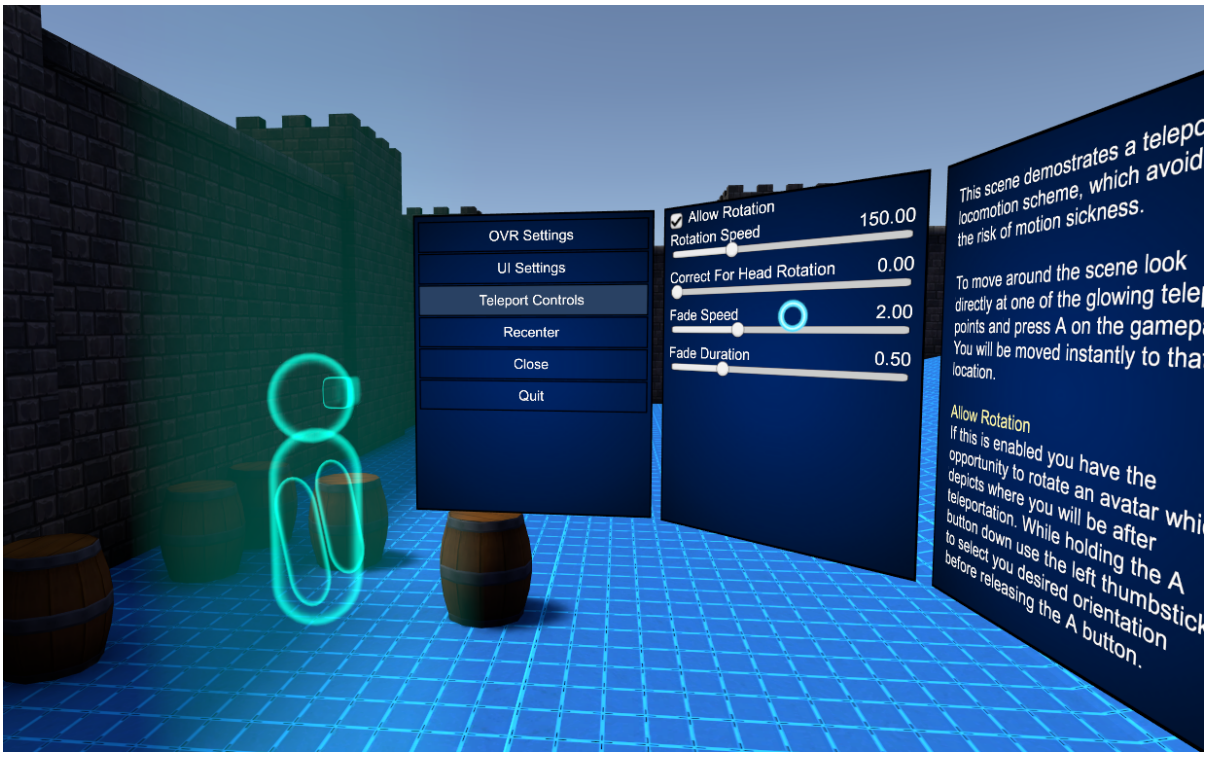
Sample scenes are browsed and controlled with a simple UI which provides in-app explanatory notes. Parameter controls allow users to adjust settings, providing an immediate, direct experience of the impact of different design decisions. The Sample Framework control panel itself is an example of in-VR control and navigation, and may be used as a model for your own applications.

We have provided a Windows executable for use with the Oculus Rift or DK2. A Samsung Gear VR may be downloaded for free from the Gallery Apps section of the Oculus Store. These applications are simply builds of the Unity project.

Navigation

Launch the Sample Framework on Rift or Gear VR to load the startup scene. You will see the *Inspector*, a three-pane interface providing controls for scene settings, documentation, and navigation controls for browsing to other scenes. Making selections on the top-level menu on the left panel changes the content of the other two panels. The center panel is a contextual menu, and the right panel displays notes and instructions for the current scene.

Figure 8: Sample Framework Controls in VR



Inspector navigation is primarily gaze-controlled, supplemented by a mouse and keyboard, a gamepad (PC or Gear VR), or the Gear VR touchpad.

To launch a scene from the center panel, you may select and click the scene with a mouse, gaze at the scene name and press the A button on a gamepad, or tap the Gear VR touchpad.

Some scenes are grouped into folders (displayed as buttons). When browsing from a folder, select the ".." button to navigate one level up in the scene hierarchy.

Scrolling

Some panels support vertical scrolling. Several methods of scrolling are supported in order to illustrate some of the available options for implementing this feature. The following methods are supported:

1. Gaze at the bottom or top of the panel.
2. Gaze over the panel and swipe up or down on the Gear VR touch pad.
3. Position the mouse pointer over the panel and use the mouse scroll wheel.
4. Click and drag the panel using the mouse pointer.
5. Click and drag by using the gaze pointer and "clicking" with the Gear VR touch pad, the gamepad A button, or your space bar.
6. Gaze over the panel and scroll with the right thumbstick.

Unity Reference Content

This section contains reference material, including compatibility information for Unity and Utilities for Unity versions, a complete scripting reference for the C sharp scripts included with Utilities for Unity, release notes, a migration guide for Unity 4 users, and a legacy guide describing our Unity 4 integration..

Unity-SDK Version Compatibility

This reference describes the relationship between Unity versions, Oculus PC and Mobile SDKs, and Oculus Unity Integration and Utilities packages.

Utilities for Unity 5.x Versions

We are currently supporting the 5.4, 5.5, and 5.6 branches of the Unity editor, which are all under active development. Our Utilities for Unity 5 package no longer supports Unity versions prior to 5.3.4p5.

Release Date	Utilities	PC SDK	Mobile SDK	Unity
3/30/2017	1.13.0	1.12.0	1.0.4.5	5.6.0f3 or later
3/30/2017	1.13.0	1.12.0	1.0.4.5	5.5.2p3 or later
3/30/2017	1.13.0	1.12.0	1.0.4.5	5.4.5f1 or later
3/10/2017	1.12.0	1.12.0	1.0.4.5	5.5.2p3 or later
3/10/2017	1.12.0	1.12.0	1.0.4.5	5.4.5f1 or later
3/10/2017	1.12.0	1.12.0	1.0.4.5	5.3.8f1 or later (deprecated)
2/1/2017	1.11.0	1.11.0	1.0.4.5	5.5.1p2 or later
2/8/2017	1.11.0	1.11.0	1.0.4.5	5.4.4p3 or later
2/1/2017	1.11.0	1.11.0	1.0.4.5	5.3.7p4 or later (deprecated)
11/28/2016	1.10.0	1.10.0	1.0.4	5.5.0p1 or later
11/28/2016	1.10.0	1.10.0	1.0.4	5.4.3p3 or later
11/28/2016	1.10.0	1.10.0	1.0.4	5.3.7p2 or later
11/1/2016	1.9.0	1.9.0	1.0.4	5.4.2p2 or later
11/1/2016	1.9.0	1.9.0	1.0.4	5.3.6p8 or later
9/15/2016	1.8.0	1.8.0	1.0.3	5.4.1p1 or later
9/15/2016	1.8.0	1.8.0	1.0.3	5.3.6p5 or later
8/18/2016	1.7.0	1.7.0	1.0.3	5.4.0p3 or later
8/18/2016	1.7.0	1.7.0	1.0.3	5.3.6p3 or later
7/28/2016	1.6.0	1.5.0	1.0.3	5.4.0b16 or later
7/28/2016	1.6.0	1.6.0	1.0.3	5.3.6p1 or later
6/30/2016	1.5.0	1.5.0	1.0.3	5.4.0b16 and later

Release Date	Utilities	PC SDK	Mobile SDK	Unity
6/30/2016	1.5.0	1.5.0	1.0.3	5.3.4p5 and later
4/20/2016	1.3.2	1.3.2	1.0.0.1	5.4.0b11 and later
4/20/2016	1.3.2	1.3.2	1.0.0.1	5.3.3p3 and later
3/28/2016	1.3.0	1.3.0	1.0.0.1	5.3.4p1
10/30/2015	0.1.3.0 Beta	0.8.0.0	0.6.2.0	5.2.2p2
10/1/2015	0.1.2.0 Beta	0.7.0.0	0.6.2.0	5.2.1p2
7/6/2015	0.1.0.0 Beta	0.6.0.1	0.5.0	5.1.1

Unity 4.x Integration Versions

Release Date	Integration	PC SDK	Mobile SDK	Unity
3/28/2016	1.3.0	1.3.0	1.0.0.1	4.7.0f1
10/30/2015	0.8.0.0	0.8.0.0	1.0.0	4.6.9p1
9/08/2015	0.6.2.0	0.7.0.0	0.6.2.0	4.6.7+
8/14/2015	0.6.1.0	0.6.0.1	0.6.1.0	4.6.7+
8/7/2015	0.6.0.2	0.6.0.1	0.6.0.1	4.6.7
6/25/2015	D 0.6.0.1	0.5.0.1	0.6.0.1	4.6
6/12/2015	M 0.6.0.1	0.5.0.1	0.6.0.1	4.6
5/15/2015	D 0.6.0.0	0.5.0.1	0.5.0	4.6
3/31/2015	M 0.5.0	0.5.0.1	0.5.0	4.6
3/26/2015	D 0.5.0.1	0.5.0.1	0.5.0	4.6
3/20/2015	M 0.4.3.1	0.4.4	0.4.3.1	4.5
2/27/2015	M 0.4.3	0.4.4	0.4.3	4.5
1/23/2015	M 0.4.2	0.3.2	0.4.2	4.5
1/7/2015	M 0.4.1	0.3.2	0.4.1	4.4
12/4/2014	D 0.4.4	0.4.4	0.4.0	4.6
11/12/2014	M 0.4.0	0.3.2	0.4.0	4.4
11/10/2014	D 0.4.3.1	0.4.3.1	N/A	4.5
10/24/2014	D 0.4.3	0.4.3	N/A	4.5
9/4/2014	D 0.4.2	0.4.2	N/A	4.5
8/11/2014	D 0.4.1	0.4.1	N/A	4.4
7/24/2014	D 0.4.0	0.4.0	N/A	4.4
5/22/2014	D 0.3.2	0.3.2	N/A	4.3
4/14/2014	D 0.3.1	0.3.1	N/A	4.3
10/10/2013	D 0.2.5	0.2.5	N/A	4.3

Unity Scripting Reference

The Unity Scripting Reference contains detailed information about the data structures and files included with the Utilities and Legacy Integration packages.

Utilities for Unity 5.x Developer Reference

- *Oculus Unity Utilities Reference Manual 1.13*
- *Oculus Unity Utilities Reference Manual 1.12*
- *Oculus Unity Utilities Reference Manual 1.11*
- *Oculus Unity Utilities Reference Manual 1.10*
- *Oculus Unity Utilities Reference Manual 1.9*
- *Oculus Unity Utilities Reference Manual 1.8*
- *Oculus Unity Utilities Reference Manual 1.7*
- *Oculus Unity Utilities Reference Manual 1.6*
- *Oculus Unity Utilities Reference Manual 1.5*
- *Oculus Unity Utilities Reference Manual 1.3*
- *Oculus Unity Utilities Reference Manual 0.1.3.0*
- *Oculus Unity Utilities Reference Manual 0.1.2.0*
- *Oculus Unity Utilities Reference Manual 0.1.0.0*

Unity 4.x Legacy Integration Developer Reference

- *Oculus Unity 4.x Legacy Integration 1.3.0*
- *Oculus Unity 4.x Legacy Integration 0.8.0*
- *Oculus Unity 4.x Legacy Integration 0.6.2*
- *Oculus Unity 4.x Legacy Integration 0.6.1*

Release Notes

This section describes changes for each version release.

1.13 Oculus Utilities for Unity 5 Release Notes

Oculus Utilities for Unity 5 version 1.13.0

This document provides an overview of new features, improvements, and fixes included in the latest version of the Oculus Utilities for Unity 5. For information on first-party changes to Unity VR support for Oculus, see the Unity Release Notes for the appropriate version. You will find a scripting reference for the included C# scripts in our *Unity Reference Content* on page 77.

Unity 5.6.0f3 is the only supported version of 5.6 at this time. Earlier versions of 5.6.0 are not supported, and 5.6.0p1 has a crashing bug affecting Gear VR applications.

New Features

- Added android:installLocation="auto" to store-compatible AndroidManifest.xml.

Bug Fixes

- Fixed double-counting of orientation when recentering OVRPlayerController.
- Fixed NullReferenceExceptions on edit-and-continue.

Known Issues

- Unity 5.6.0f3 is the only supported version of 5.6 at this time. Gear VR applications built with Unity 5.6.0f2 crash immediately upon launch, and Gear VR applications built with 5.6.0p1 may crash when Multi-View is enabled.
- The following versions of Unity require the *Visual C++ Redistributable for Visual Studio 2015* or Rift builds will fail to run in VR, and the error "Security error. This plugin is invalid!" will be reported in output_log.txt:
  - 5.3.6p3-5.3.6p7
  - 5.4.0f1-5.4.2p1
  - 5.5.0b1-5.5.0b8
- Unity has a known issue such that parenting one VR camera to another will compound tracking twice. As a workaround, make them siblings in the GameObject hierarchy.
- Rift
  - All Unity versions prior to 5.4.3p3 leak 5MB/s if you have a Canvas in your scene, enable *Run In Background*, and dismount the Rift. You can check OVRManager.hasVrFocus in an Update function to disable your Canvases while the HMD is dismounted.
  - Transparent VR Compositor Layers do not currently support multiple layers of occlusion.
- Gear VR
  - Do not use Utilities 1.11.0 due to a crash when returning to focus from Universal Menu or Quit to Home dialog.
  - Due to a Unity bug, the Camera pose can be corrupted by scripts in the first frame after being enabled with VR support. As a workaround, use the latest Utilities version or zero out the eye anchor poses when a new OVRCameraRig is spawned and the first frame after usePerEyeAnchors changes.
  - With Unity 5.3, the world may appear tilted. As a workaround, use the latest Utilities version or disable the virtual reality splash image.
  - Mobile developers should not use Unity versions 5.3.6p1-2 and 5.4.0p1-2 due to incorrect positional movement of the head.
  - Unity 5.3.4-5.3.6p3 and Unity 5.4.0b16-Unity 5.4.0p3: Do not set *DSP Buffer Size* to *Best* in Audio Manager in the Inspector for now or you will encounter audio distortion. Set it to *Good* or *Default* instead.
- Mobile App Submission to Oculus Store
  - All mobile applications using Utilities 1.9 and 1.10 will fail Oculus Store submission due to a bug affecting reserved interaction handling for the Universal Menu. Please remove previously-imported project files as described in *Importing the Oculus Utilities Package* on page 8 and import the latest Utilities version, and update your Unity editor to a *compatible version* if necessary.

1.12 Oculus Utilities for Unity 5 Release Notes

Oculus Utilities for Unity 5 version 1.12.0

This document provides an overview of new features, improvements, and fixes included in the latest version of the Oculus Utilities for Unity 5. For information on first-party changes to Unity VR support for Oculus, see the Unity Release Notes for the appropriate version. You will find a scripting reference for the included C# scripts in our *Unity Reference Content* on page 77.

Version Compatibility

On initial release, Utilities v 1.12.0 is compatible with our recommended Unity 5.3.8f1 and 5.4.5f1. For up-to-date compatibility information, see *Compatibility and Version Requirements* on page 5.

New Features

- Added support for the Gear VR Controller to OVRInput. For more information, see *OVRInput Unified Input API* on page 35.

API Changes

- Added `OVRPlugin.GetAppFramerate()` to OVRDisplay.cs; returns frame rate reported by Oculus plugin (Rift and Gear VR). Requires a Unity Editor version we recommend for use with Utilities 1.12 - see *Compatibility and Version Requirements* on page 5 for details.

Bug Fixes

- Changed OVRInput.GetAngularVelocity(..) and OVRInput.GetAngularAcceleration(..) to return Vector3 instead of Quaternion, avoiding issues for rates above 2*pi. Developers who need the old behavior for any reason may use Quaternion.Euler(..).
- Gear VR:
  - Fixed bug causing mobile applications built with Unity versions compatible with Utilities 1.11.0 to crash when returning to focus from Universal Menu or Quit to Home dialog, or when the Gear VR is taken off for several seconds, then put back on.
  - Fixed black screen during launch with developer mode enabled in some Unity 5.3 versions using our 1.11 integration.
  - Fixed bug causing mobile apps using Utilities 1.11.0 to appear tilted when enabling a virtual reality splash screen.
- VR Compositor Layers:
  1. Added OVRUnderlayTransparentOccluder, which was missing in previous versions.
  2. Fixed issue causing layer colors to appear washed out when using render targets as input textures on PC.
  3. Fixed issue where right-side textures were lost when using stereo pairs of OVROverlays.

Known Issues

- The following versions of Unity require the *Visual C++ Redistributable for Visual Studio 2015* or Rift builds will fail to run in VR, and the error "Security error. This plugin is invalid!" will be reported in output_log.txt:
  - 5.3.6p3-5.3.6p7
  - 5.4.0f1-5.4.2p1
  - 5.5.0b1-5.5.0b8
- Unity has a known issue such that parenting one VR camera to another will compound tracking twice. As a workaround, make them siblings in the GameObject hierarchy.
- Rift
  - All Unity versions prior to 5.4.3p3 leak 5MB/s if you have a Canvas in your scene, enable *Run In Background*, and dismount the Rift. You can check OVRManager.hasVrFocus in an Update function to disable your Canvases while the HMD is dismounted.
  - Transparent VR Compositor Layers do not currently support multiple layers of occlusion.
- Gear VR

  - Do not use Utilities 1.11.0 due to a crash when returning to focus from Universal Menu or Quit to Home dialog.
  - Due to a Unity bug, the Camera pose can be corrupted by scripts in the first frame after being enabled with VR support. As a workaround, use Utilities 1.12 or zero out the eye anchor poses when a new OVRCameraRig is spawned and the first frame after usePerEyeAnchors changes.
  - With Unity 5.3, the world may appear tilted. As a workaround, use Utilities 1.12 or disable the virtual reality splash image.
  - Mobile developers should not use Unity versions 5.3.6p1-2 and 5.4.0p1-2 due to incorrect positional movement of the head.
  - Unity 5.3.4-5.3.6p3 and Unity 5.4.0b16-Unity 5.4.0p3: Do not set *DSP Buffer Size* to *Best* in Audio Manager in the Inspector for now or you will encounter audio distortion. Set it to *Good* or *Default* instead.
- Mobile App Submission to Oculus Store
  - All mobile applications using Utilities 1.9 and 1.10 will fail Oculus Store submission due to a bug affecting reserved interaction handling for the Universal Menu. Please remove previously-imported project files as described in *Importing the Oculus Utilities Package* on page 8 and import Utilities version 1.12, and update your Unity editor to a *compatible version* if necessary.
  - When building a mobile application for submission to the Oculus Store, you must set *Install Location* to *Auto* in addition to generating a custom manifest as described in *Building Mobile Applications* on page 14.

    1. Click *Edit* > *Project Settings* > *Player*.
    2. Expand the *Other Settings* properties.
    3. Set *Install Location* to *Auto*.

1.11 Oculus Utilities for Unity 5 Release Notes

Oculus Utilities for Unity 5 version 1.11.0

This document provides an overview of new features, improvements, and fixes included in the latest version of the Oculus Utilities for Unity 5. For information on first-party changes to Unity VR support for Oculus, see the Unity Release Notes for the appropriate version. You will find a scripting reference for the included C# scripts in our *Unity Reference Content* on page 77.

If you have previously imported a Unity integration package, you must delete all Oculus Integration content before importing the new Unity package. For more information, see *Importing the Oculus Utilities Package* on page 8.

Unity 5.3 and 5.6 Support

Unity 5.3 support is deprecated and not all features are guaranteed to work. Please update to 5.4 or 5.5. We do not currently support preview versions of 5.6.

New Features

- Added Performance Auditing Tool for Rift and mobile development. This tool verifies that your VR project configuration and settings are consistent with our recommendations. For more information, see *Performance Auditing Tool* on page 65.
- Added OVRGrabber and OVRGrabbable scripts for Oculus Touch to the Room sample in Assets/Scenes/. For details, see our *Unity Reference Content* on page 77.

- (Mobile only) Added offcenter cubemap support to OVROverlay, allowing you to display an overlay as a cubemap with a texture coordinate offset to increase resolution for areas of interest. For more information, see OVROverlay in our *Unity Reference Content* on page 77.

API Changes

- Deprecated OVRProfile.

Bug Fixes

- Fixed bug affecting reserved interaction handling for the Universal Menu that caused all mobile applications using Utilities 1.9 and 1.10 to fail Oculus Store submission.

Known Issues

- Adaptive Resolution is not currently working and should be disabled. If applications using Adaptive Resolution reach 45 Hz, they will remain stuck at that frame rate until relaunched. A fix is planned for the Rift 1.12 runtime release and should not require any application changes.
- The following versions of Unity require the *Visual C++ Redistributable for Visual Studio 2015* or Rift builds will fail to run in VR, and the error "Security error. This plugin is invalid!" will be reported in output_log.txt:
  - 5.3.6p3-5.3.6p7
  - 5.4.0f1-5.4.2p1
  - 5.5.0b1-5.5.0b8
- Rift
  - All Unity versions prior to 5.4.3p3 leak 5MB/s if you have a Canvas in your scene, enable *Run In Background*, and dismount the Rift. You can check OVRManager.hasVrFocus in an Update function to disable your Canvases while the HMD is dismounted.
- Gear VR
  - Due to a Unity bug, the Camera pose can be corrupted by scripts in the first frame after being enabled with VR support. As a workaround, use Utilities 1.11 or zero out the eye anchor poses when a new OVRCameraRig is spawned and the first frame after usePerEyeAnchors changes.
  - With Unity 5.3, the world may appear tilted. As a workaround, use Utilities 1.10 or disable the virtual reality splash image.
  - All mobile applications using Utilities 1.9 and 1.10 will fail Oculus Store submission due to a bug affecting reserved interaction handling for the Universal Menu. Please remove previously-imported project files as described in *Importing the Oculus Utilities Package* on page 8 and import Utilities version 1.11, and update your Unity editor to a *compatible version* if necessary.
  - Mobile developers should not use Unity versions 5.3.6p1-2 and 5.4.0p1-2 due to incorrect positional movement of the head.
  - Unity 5.3.4-5.3.6p3 and Unity 5.4.0b16-Unity 5.4.0p3: Do not set *DSP Buffer Size* to *Best* in Audio Manager in the Inspector for now or you will encounter audio distortion. Set it to *Good* or *Default* instead.

1.10 Oculus Utilities for Unity 5 Release Notes

Oculus Utilities for Unity 5 version 1.10.0

This document provides an overview of new features, improvements, and fixes included in the latest version of the Oculus Utilities for Unity 5. For information on first-party changes to Unity VR support for Oculus, see the Unity Release Notes for the appropriate version. You will find a scripting reference for the included C# scripts in our *Unity Developer Reference*.

If you have previously imported a Unity integration package, you must delete all Oculus Integration content before importing the new Unity package. For more information, see *Importing the Oculus Utilities Package* on page 8.

Utilities 1.10 adds an option to build in APK for submission to the Oculus Store in *Tools* > *Oculus*. It also includes a fix for an issue that caused poor performance or freezing when using multiple VR cameras or VR Compositor underlays with Gear VR. Any mobile application using either of these should update to this version.

New Features

- Added option to *Tools* > *Oculus* to build APK for submission to Oculus Store.
- Added Rift support for cubemap overlays to VR Compositor Layers.

Bug Fixes

- Fixed poor performance or freezing bug when using multiple VR cameras or VR Compositor underlays with Gear VR.
- Fixed memory leak in OVROverlay.
- Fixed uncommon issue in which setting mirror to full screen caused app rendering to freeze in Rift.

Known Issues

- The following versions of Unity require the *Visual C++ Redistributable for Visual Studio 2015* or Rift builds will fail to run in VR, and the error "Security error. This plugin is invalid!" will be reported in output_log.txt:
  - 5.3.6p3-5.3.6p7
  - 5.4.0f1-5.4.2p1
  - 5.5.0b1-5.5.0b8
- Rift
  - All Unity versions leak 5MB/s if you have a Canvas in your scene, enable Run In Background, and dismount the Rift. You can check OVRManager.hasVrFocus in an Update function to disable your Canvases while the HMD is dismounted.
- Gear VR
  - Mobile developers should not use Unity versions 5.3.6p1-2 and 5.4.0p1-2 due to incorrect positional movement of the head.
  - Unity 5.3.4-5.3.6p3 and Unity 5.4.0b16-Unity 5.4.0p3: Do not set DSP Buffer Size to Best in Audio Manager in the Inspector for now or you will encounter audio distortion. Set it to Good or Default instead.
- Touch
  - For PCs using Oculus runtime 1.10, `OVRInput.GetConnectedControllers()` does not mark Touch controllers as disconnected when batteries are removed, and the input mask returns Touch (Left+Right) active when only one controller is on. This issue will resolve automatically when runtime 1.11 is released.

1.9 Oculus Utilities for Unity 5 Release Notes

Oculus Utilities for Unity 5 version 1.9.0

This document provides an overview of new features, improvements, and fixes included in the latest version of the Oculus Utilities for Unity 5. For information on first-party changes to Unity VR support for Oculus, see the Unity Release Notes for the appropriate version. You will find a scripting reference for the included C# scripts in our *Unity Developer Reference*.

If you have previously imported a Unity integration package, you must delete all Oculus Integration content before importing the new Unity package. For more information, see *Importing the Oculus Utilities Package* on page 8.

This release adds Gear VR touchpad and back button support to OVRInput.

Be sure to check out the new Mono Optimization sample included in *Unity Sample Framework* on page 72 v1.5.1. Monoscopically rendering distant content in a scene can offer significant rendering performance improvements.

New Features

- Added Gear VR Touchpad and back button support to OVRInput.
- OVRInput.Controller.Active now automatically switches away from Touch if the user is not holding it.
- OVRBoundary now supports more than 256 Guardian System bounds points.
- Improved image quality for higher values in VRSettings.renderScale due to mipmapping. (Rift)

API Changes

- OVRInput.Button and OVRInput.RawButton events now report Gear VR touchpad swipes and back button presses.

Bug Fixes

- Unity 5.3.6p8, 5.4.2p2, and 5.5.0b9 correct a failure to report shoulder button events in OVRInput when used with Utilities 1.9.0.
- Fixed dependency on the Visual C++ Redistributable for Visual Studio 2015 causing Rift builds to fail to run in VR in some versions of Unity (see Known Issues for more information).
- Fixed 5MB/s memory leak when using OVROverlay.

Known Issues

- OVRInput fails to report shoulder button events when Utilities 1.9.0 is used with Unity versions 5.4.2p1 and 5.5.0b8 or earlier.
- The following versions of Unity require the *Visual C++ Redistributable for Visual Studio 2015* or Rift builds will fail to run in VR, and the error "Security error. This plugin is invalid!" will be reported in output_log.txt:
  - 5.3.6p3-5.3.6p7
  - 5.4.0f1-5.4.2p1
  - 5.5.0b1-5.5.0b8
- Rift
  - All Unity versions leak 5MB/s if you have a Canvas in your scene, enable Run In Background, and dismount the Rift. You can check OVRManager.hasVrFocus in an Update function to disable your Canvases while the HMD is dismounted.
- Gear VR
  - Mobile developers should not use Unity versions 5.3.6p1-2 and 5.4.0p1-2 due to incorrect positional movement of the head.
  - Unity 5 automatically generates manifest files with Android builds that will cause them to be automatically rejected by the Oculus Store submission portal. If this is blocking your application submission, please let us know on our *Developer forum* and we will work with you on a temporary workaround.

- Unity 5.3.4-5.3.6p3 and Unity 5.4.0b16-Unity 5.4.0p3: Do not set DSP Buffer Size to Best in Audio Manager in the Inspector for now or you will encounter audio distortion. Set it to Good or Default instead.

1.8 Oculus Utilities for Unity 5 Release Notes

Oculus Utilities for Unity 5 version 1.8.0

This document provides an overview of new features, improvements, and fixes included in the latest version of the Oculus Utilities for Unity 5. For information on first-party changes to Unity VR support for Oculus, see the Unity Release Notes for the appropriate version. You will find an updated scripting reference for the included C# scripts in our *Unity Developer Reference*.

If you have previously imported a Unity integration package, you must delete all Oculus Integration content before importing the new Unity package. For more information, see *Importing the Oculus Utilities Package* on page 8.

Mobile SDK Examples has been deprecated; use the *Unity Sample Framework* on page 72instead.

Note: Due to issues with earlier releases, we now recommend all developers update to 5.3.6p5 or version 5.4.1p1 or later.

New Features

- Added support for the Oculus Guardian System, which visualizes the bounds of a user-defined Play Area. Note that it is currently unsupported by public versions of the Oculus runtime. See *OVRBoundary Guardian System API* on page 46 for more information.
- Added underlay support allowing VR compositor layers to be rendered behind the eye buffer.
- Added support for stereoscopic cubemap VR compositor layers (mobile only).

API Changes

- Added OVRBoundary API for interacting with the Oculus Guardian System.
- Removed OVRTrackerBounds.

Bug Fixes

- Fixed blackscreen issue related to unplugging the HDMI cable and re-plugging it back in.
- Fixed Touch judder issue.

Known Issues

- Due to issues with earlier releases, we now recommend all developers update to 5.3.6p5 or version 5.4.1p1 or later. 5.3.6p3-5.3.6p4 are also known to work.
- Gear VR
  - Mobile developers should not use Unity versions 5.3.6p1-2 and 5.4.0p1-2 due to incorrect positional movement of the head.
  - Unity 5 automatically generates manifest files with Android builds that will cause them to be automatically rejected by the Oculus Store submission portal. If this is blocking your application submission, please let us know on our *Developer Forum* and we will work with you on a temporary workaround.

- Gear VR developers using Unity 5.3.4 or later, or using Unity 5.4.0b16 and later: Do not set DSP Buffer Size to Best in Audio Manager in the Inspector for now or you will encounter audio distortion. Set it to Good or Default instead.

1.7 Oculus Utilities for Unity 5 Release Notes

Oculus Utilities for Unity 5 version 1.7.0

This document provides an overview of new features, improvements, and fixes included in the latest version of the Oculus Utilities for Unity 5. For information on first-party changes to Unity VR support for Oculus, see the Unity Release Notes for the appropriate version. You will find an updated scripting reference for the included C# scripts in our *Unity Developer Reference*.

If you have previously imported a Unity integration package, you must delete all Oculus Integration content before importing the new Unity package. For more information, see *Importing the Oculus Utilities Package* on page 8.

Note: Unity versions prior to 5.3.4p5 (or 5.3.3p3 + OVRPlugin 1.3) are no longer supported. In addition, Mobile developers should not use Unity versions 5.3.6p1-2 and 5.4.0p1-2 due to incorrect positional movement of the head.

New Features

- Updated OVROverlay.cs to support cubemap (skybox) and hemicylinder overlay shapes (mobile only) in addition to the existing quadrilateral Game Object shape.
- Added OVRRTOverlayConnector to stream Render Texture contents to an OVROverlay. For more information, see *VR Compositor Layers* on page 49.
- Added runtime support for Adaptive Resolution (see description in *OVRManager*).

API Changes

- Unity versions prior to 5.3.4p5 (or 5.3.3p3 + OVRPlugin 1.3) are no longer supported.
- OVRInput.SetControllerVibration may no longer address a pair of Touch controllers with OVRinput.Controller.Touch; you must address each controller individually using OVRinput.Controller.LTouch or OVRinput.Controller.RTouch.
- Removed OVRDebugGraph.cs from Assets/OVR/Scripts/.

Bug Fixes

- Fixed backwards head-neck model z translation on Gear VR.
- When targeting Oculus Touch, OVRInput.SetControllerVibration calls are now limited to 30 per second due to performance issues; additional calls are discarded. (Note: we recommend using OVRHaptics to control Touch vibrations - it provides better haptics quality without the performance issues of OVRInput.SetControllerVibration().
- Fixed crash in CreateDirect3D11SurfaceFromDXGISurface after eye buffer re-allocation on Rift.
- Fixed OVRInput Xbox controller detection with Unity on Windows 10 Anniversary Edition.
- Fixed delay when loading with "Run in Background" enabled.
- Fixed missing runtime support for adaptive viewport scaling.

Known Issues

- Gear VR

- Mobile developers should not use Unity versions 5.3.6p1-2 and 5.4.0p1-2 due to incorrect positional movement of the head.
- Unity 5 automatically generates manifest files with Android builds that will cause them to be automatically rejected by the Oculus Store submission portal. If this is blocking your application submission, please let us know on our *Developer Forum* and we will work with you on a temporary workaround.
- Gear VR developers using Unity 5.3.4 or later, or using Unity 5.4.0b16 and later: Do not set DSP Buffer Size to Best in Audio Manager in the Inspector for now or you will encounter audio distortion. Set it to Good or Default instead.

1.6 Oculus Utilities for Unity 5 Release Notes

Oculus Utilities for Unity 5 version 1.6.0

This document provides an overview of new features, improvements, and fixes included in the latest version of the Oculus Utilities for Unity 5. For information on first-party changes to Unity VR support for Oculus, see the Unity Release Notes for the appropriate version. You will find an updated scripting reference for the included C# scripts in our *Unity Developer Reference*.

New Features

- Added Adaptive Resolution to OVRManager, which automatically scales down app resolution when GPU utilization exceeds 85%. See "OVRManager" in *Unity Components* for details. (Rift only, requires Unity v 5.4 or later)
- OVR Screenshot Wizard size parameter is now freeform instead of dropdown selection for greater flexibility.
- Added recommended anti-aliasing level to help applications choose the right balance between performance and quality.
- Added support for more than one simultaneous OVROverlay. Now apps can show up to 3 overlay quads on Gear VR and 15 on Rift.

API Changes

- Added OVRHaptics.cs and OVRHapticsClip.cs to programmatically control haptics for Oculus Touch controller. See *OVRHaptics for Oculus Touch* on page 48 for more information.
- Added public members Enable Adaptive Resolution, Max Render Scale, and Min Render Scale to OVRManager.
- Added OVRManager.useRecommendedMSAALevel to enable auto-selection of anti-aliasing level based on device performance.
- Added OVRManager.useIPDInPositionTracking to allow apps to separately disable head position tracking (see OVRManager.usePositionTracking) and stereopsis.

Bug Fixes

- Fixed bug preventing power save from activating on Gear VR.
- Fixed counter-intuitive behavior where disabling OVRManager.usePositionTracking prevented proper eye separation by freezing the eye camera positions at their original offset.

Known Issues

- Gear VR

- Unity 5 automatically generates manifest files with Android builds that will cause them to be automatically rejected by the Oculus Store submission portal. If this is blocking your application submission, please let us know on our *Developer Forum* and we will work with you on a temporary workaround.
- Gear VR developers using Unity 5.3.4 or later, or using Unity 5.4.0b16 and later: Do not set DSP Buffer Size to Best in Audio Manager in the Inspector for now or you will encounter audio distortion. Set it to Good or Default instead.

1.5 Oculus Utilities for Unity 5 Release Notes

Oculus Utilities for Unity 5 version 1.5.0

This document provides an overview of new features, improvements, and fixes included in the latest version of the Oculus Utilities for Unity 5. For information on first-party changes to Unity VR support for Oculus, see the Unity Release Notes for the appropriate version. You will find an updated scripting reference for the included C# scripts in our *Unity Developer Reference*.

New Features

- Added OVR Screenshot and OVR Capture Probe tools, which exports a 360 screenshot of game scenes in cube map format. See *Cubemap Screenshots* on page 59 for more information.
- Switched to built-in volume indicator on mobile.
- Exposed OVRManager.vsyncCount to allow half or third-frame rate rendering on mobile.
- Added bool OVRManager.instance.isPowerSavingActive (Gear VR).

Bug Fixes

- Repeatedly changing resolution or MSAA level no longer causes slowdown or crashing.
- Fixed scale of OVRManager.batteryLevel and OVRManager.batteryTemperature.
- Fixed race condition leading to black screens on Rift in some CPU-heavy cases.
- Fixed memory bloat due to unpooled buffers when using MSAA.

Known Issues

- **Gear VR developers** using Unity 5.3.4 or later, or using Unity 5.4.0b16 and later: Do not set *DSP Buffer Size* to *Best* in *Audio Manager* in the Inspector for now or you will encounter audio distortion. Set it to *Good* or *Default* instead.

1.3 Oculus Utilities for Unity 5 Release Notes

Oculus Utilities for Unity 5 version 1.3.2

This document provides an overview of new features, improvements, and fixes included in the latest version of the Oculus Utilities for Unity 5. For information on first-party changes to Unity VR support for Oculus, see the Unity Release Notes for the appropriate version. You will find an updated scripting reference for the included C# scripts in our *Unity Developer Reference*.

Unity Versions and OVRPlugin

Oculus Utilities for Unity 5 version 1.3.2 is for use with Unity 5.3 or Unity 5.4. It may only be used with **Unity 5.3.3p3 or later, or 5.4.0b11 or later**. You **must** download and install our OVRPlugin for Unity 1.3.2, available

from our Downloads Page, to use these versions. For more information, see *Utilities 1.3.2 and OVRPlugin* on page 92.

New Features

- OVRInput may now be used without an OVRManager instance in the scene.

API Changes

- Restored OVRVolumeControl.

Bug Fixes

- OVRManager.instance.usePositionTracking now toggles the head model on Gear VR.
- Fixed incorrect fog interaction with transparent UI shader.
- Fixed crash on start with Unity 5.4.0b14 and b15 on Gear VR.
- Restored OVRVolumeControl, which was accidentally removed in 1.3.0.

Known Issues

- **Utilities 1.3.0**: Volume control will be missing on mobile applications until the release of Mobile SDK 1.0.2. OVRVolumeControl is avaialble with Utilities v 0.1.3.0 and earlier. It was also restored in Utilities v 1.3.2.

Oculus Utilities for Unity 5 version 1.3.0

Note: Oculus Utilities for Unity 5 version 1.3.0 is for use with **Unity 5.3.4p1 only**. To use Unity 5.3.4p1 with the Oculus Rift or Samsung Gear VR, **you must download and install our OVRPlugin for Unity 1.3.0**, available from our *Downloads page*. For more information, see *Unity 5.3.4p1 and OVRPlugin 1.3.0*.

Overview of Major Changes

This public release incorporates all changes from Utilities private releases 0.2.0 through 1.3.0.

We no longer include the deprecated InputManager.asset file.

Rift Floor-Level Tracking

OVR Manager now includes Tracking Origin Type, which defaults to Eye Level (i.e., tracking origin is handled as in previous releases). When set to Floor Level, tracking origin will be calculated relative to the user's standing height as specified during Rift setup or with the Oculus app.

Note: Floor-level tracking will often be used with standing experiences, but there may be situations in which eye-level tracking is a better fit for a standing experience, or floor-level tracking is a better fit for a seated experience.

Any application running Unity should now be able to pull the correct height information.

New Features

- Added support for PC SDK 1.3, including support for Rift consumer version hardware.
- Added support for Asynchronous TimeWarp and Phase Sync.
- Added Rift Remote controller support.
- Added application lifecycle management, including VR-initiated backgrounding and exit.
- Exposed proximity sensor.
- Added support for multiple trackers.

- Exposed velocity and acceleration for all tracked nodes.
- Added support for EyeLevel and FloorLevel tracking origins.
- Audio input and output now automatically use the Rift microphone and headphones (if enabled in the Oculus app).
- Rift's inter-axial distance slider now affects the distance between Unity's eye poses.
- Splash screen now uses Asynchronous TimeWarp for smooth tracking during load.
- Added experimental D3D 12 rendering support.
- Added events for focus change.
- Added events for audio device changes requiring sound restart.
- Added ControllerTracked state.
- Reduced head tracking latency on Gear VR by updating on render thread
- Improved performance by reducing lock contention.
- Exposed power management (CPU and GPU levels) on Android.
- Exposed queue-ahead on Android to trade latency for CPU-GPU parallelism.

API Changes

- `OVRTracker.GetPose()` no longer takes a prediction time. It now takes an optional index specifying the tracker whose pose you want.
- `OVRTracker.frustum` has been replaced by `OVRTracker.GetFrustum()`, which takes an optional index specifying the tracker whose frustum you want.
- `OVRManager.isUserPresent` is true when the proximity sensor detects the user.
- `OVRInput.GetControllerLocal[Angular]Velocity/Acceleration` exposes the linear and angular velocity and rotation of each Touch controller.
- `OVRDisplay.velocity` exposes the head's linear velocity.
- `OVRDisplay.angularAcceleration` exposes the head's angular acceleration.
- Removed OVRGamepadController.cs and OVRInputControl.cs scripts, which have been replaced by the new OVRInput.cs script. Refer to *OVRInput* for more information.
- Added public member `Tracking Origin Type` to OVR Manager.
- Added "floor level" reference frame for apps that need accurate floor height.
- Removed `OVRVolumeControl` in favor of Universal Menu's built-in volume meter.
- `OVRManager.queueAhead` now controls Gear VR latency mode, allowing you to trade latency for CPU-GPU parallelism. Queue-ahead is now automatically managed on Rift.
- Events `OVRmanager.VrFocusLost` and `VrFocusAcquired` occur when the app loses and regains VR focus (visibility on the HMD).
- Events `OVRManager.AudioOutChanged` and `AudioInChanged` occur when audio devices change and make audio playback impossible without a restart.
- `OVRManager.cpuLevel` controls CPU power-saving vs performance trade-off on Gear VR.
- `OVRManager.gpuLevel` controls GPU power-saving vs performance trade-off on Gear VR.
- Added ability to hold a named mutex throughout runtime.

Bug Fixes

- Removed redundant axial deadzone handling from Xbox gamepad input.
- Fixed `OVRManager.monoscopic` to display left eye buffer to both eyes and use center eye pose.
- Application lifetime management now works, even without the Utilities.
- Fixed crash when running VR apps with Rift disconnected.
- `OVRManager.isUserPresent` now correctly reports proximity sensor output.

- `OVRManager.isHMDPresent` now correctly reports Gear VR docking state.
- We now prevent AFR SLI on NVIDIA hardware to avoid black screens/flicker.
- Fixed drift between TimeWarp start matrix and view matrix.
- Fixed crash when main monitor is on one adapter and Rift is on another.
- Fixed crash on Mac due to OVRPlugin being uninitialized before first access.
- Increased dead zone on OVRInput stick input to prevent drift.
- Fixed handedness issue with angular head velocity on Rift.
- Fixed handedness issue with rotation of OVROverlay quads.
- Fixed crash in OVROverlay when using D3D 12 and compressed textures.
- Fixed crashes due to thread synchronization checks on Gear VR.
- Fixed loss of input handling while paused.
- Fixed artifact in which bars appeared around mirror image when using occlusion mesh.
- Fixed Android logcat spam about OVR_TW_SetDebugMode.
- Gear VR logs now report VRAPI loader version, not SystemActivites version.
- Statically linking MSVC runtime to avoid missing DLL dependencies.
- Fixed black screen when HMD was reconnected: notifying Unity of display lost.

Known Issues

- Volume control will be missing on mobile applications until the release of Mobile SDK 1.0.2. To restore OVRVolumeControl, please use an older copy of the Utilities.

Utilities 1.3.2 and OVRPlugin

Some versions of Unity require special handling to enable built-in VR support. This will not be required in future versions of Unity.

You must download and install OVRPlugin from our website if you are using the following Unity versions:

- Unity v 5.3.3p3 through 5.3.4p4
- Unity v 5.4.0b11 through 5.4.0b15

Unity v 5.3.3p3 through 5.3.4p4

To use Unity 5.3.3p3 through 5.3.4p4 with the Oculus Rift or Samsung Gear VR, you must download and install OVRPlugin for Unity 1.3.2, available *here*. Later versions of Unity 5.3.x may be used without special handling.

After you have downloaded and installed Unity, take these steps to install OVRPlugin:

1. Close the Unity Editor if it is currently running.
2. Navigate to C:\Program Files\Unity\Editor\Data\VR\oculus
3. Delete all contents of the directory.
4. Extract the OVRPlugin zip, open the folder 5.3\oculus, and copy all of its contents into C:\Program Files \Unity\Editor\Data\VR\oculus.

Note: Do not install OVRPlugin version 1.3.2 with any version of Unity 5.3 prior to 5.3.3p3 or it will not work properly.

Unity v 5.4.0b11 through 5.4.0b15

To use Unity 5.4.0b11 through 5.4.0b15 with the Oculus Rift or Samsung Gear VR, you must download and install OVRPlugin for Unity 1.3.2, available *here*. Later versions of Unity 5.4.x may be used without special handling.

After you have downloaded and installed Unity, take these steps to install OVRPlugin:

1. Close the Unity Editor if it is currently running.
2. Navigate to C:\Program Files\Unity\Editor\Data\VR\Unity
3. Delete all contents of the directory.
4. Extract the OVRPlugin zip, open the folder 5.4\Unity, and copy all of its contents into C:\Program Files\Unity\Editor\Data\VR\Unity.

Note: Do not install OVRPlugin version 1.3.2 with any version of Unity 5.4 prior to 5.4.0b11, or it will not work properly.

Utilities 1.3.0 and OVRPlugin

Oculus Utilities for Unity 5 version 1.3.0 is for use with **Unity 5.3.4p1 only**. Please be sure to update to this Unity version, available here: *http://unity3d.com/unity/qa/patch-releases/5.3.4p1*

To use Unity 5.3.4p1 with the Oculus Rift or Samsung Gear VR, you **must** download and install our OVRPlugin for Unity 1.3.0, available from our Downloads Page.

After you have downloaded and installed Unity 5.3.4p1, take these steps to install OVRPlugin:

1. Close the Unity Editor if it is currently running.
2. Navigate to the following directory: C:\Program Files\Unity\Editor\Data\VR\oculus
3. Delete all contents of the directory.
4. Extract the OVRPlugin zip and copy all files into the directory C:\Program Files\Unity\Editor\Data\VR\oculus.

Note: Do not install OVRPlugin version 1.3.0 with any version of Unity other than 5.3.4p1, or it will not work properly.

0.1 Beta Utilities for Unity Release Notes

This document provides an overview of new features, improvements, and fixes included in the latest version of the Oculus Utilities for Unity.

Utilities for Unity 0.1.3.0 Beta

Overview of Major Changes

Note: For detailed information about Unity version compatibility, please see *Compatibility and Requirements*.

This document provides an overview of new features, improvements, and fixes included in the latest version of the Utilities for Unity 5.x. For information on first-party changes to Unity VR support for Oculus, see the Unity Release Notes for the appropriate version.

Utilities for Unity 0.1.3 extends OVRInput support to mobile. OVRInputControl and OVRGamepadController are now deprecated and will be removed in a future release.

Mobile input bindings are now automatically added to InputManager.asset if they do not already exist - it is no longer required to replace InputManager.asset with Oculus' version. However, this asset is still provided for now to maintain legacy support for the deprecated OVRGamepadController and OVRInputControl scripts.

New Features

- Default mobile input bindings are now programmatically generated when projects are imported if they do not already exist.
- Replacing InputManager.asset is no longer required to enable gamepad support on mobile.

API Changes

- Added mobile support OVRInput.
- Deprecated OVRInputControl and OVRGamepadController.

Bug Fixes

- Fixed mobile gamepad thumbstick deadzone/drift handling and axis scaling.
- Fixed mobile gamepad support when multiple gamepads are paired.
- Fixed mobile gamepad bindings for triggers, D-pad, thumbstick presses, etc.

Utilities for Unity 0.1.2.0 Beta

Overview of Major Changes

This version adds an alpha preview release of OVRInput, which provides a unified input API for accessing Oculus Touch and Microsoft Xbox controllers.

New Features

- Redesigned input API for Oculus Touch controllers and Xbox gamepads.
- Added h264 hardware-decoder plugin for Gear VR.
- Added "face-locked" layer support to OVROverlay when parented to the camera.
- Reduced latency in the pose used by the main thread for raycasting, etc.
- Updated to PC SDK 0.7 and Mobile SDK 0.6.2.0.
- Enabled VRSettings.renderScale on Gear VR.
- Several minor performance optimizations.
- SDKExamples
    - Restored MoviePlayerSample

API Changes

- The Utilities package now requires Unity 5.1 or higher.
- Added OVRInput API alpha. Refer to documentation for usage.
- Exposed LeftHand/RightHand anchors for tracked controllers in OVRCameraRig.

Bug Fixes

- Restored ability to toggle settings such as monoscopic rendering and position tracking.
- HSWDismissed event is now correctly raised when the HSW is dismissed.
- Fixed handedness of reported velocity and acceleration values.
- OVRPlayerController now moves at a consistent speed regardless of scale.

Known Issues

- Tearing in OS X: Editor preview and standalone players do not vsync properly, resulting in a vertical tear and/or judder on DK2.
- When switching between a mobile application and System Activities screen, the back button becomes stuck in the "down" state. For more information and workarounds, please see Troubleshooting and Known Issues.

Utilities for Unity 0.1.0.0 Beta

Overview

This is the initial release of Oculus Utilities for Unity, for use with Unity versions 5.1.2 and later. The Utilities extend Unity's built-in virtual reality support with the following features:

- Standard camera rig with head-tracked "anchors".
- Detailed tracking information such as head acceleration, velocity, and latency and the IR tracker pose.
- Control over the tracking method (positional, head model only, rotation-only).
- Information about the current user's IPD, eye relief, and eye height.
- Control over graphics performance/quality features such as monoscopic rendering, queue-ahead (PC-only), and chromatic aberration.
- Experimental overlay support for high-quality text and video on a world-space quad.
- First-person shooter-style locomotion.
- Screen dimming on scene transitions.
- Notification when out of positional tracking range.
- Cross-platform gamepad support.
- Debug overlay with real-time performance and latency graphs.
- (Android only) Access to the Oculus platform UI.
- (Android only) Performance and power management.

The Oculus Utilities for Unity expose largely the same API as the Oculus Unity Integration, but they offer all the benefits of Unity's built-in VR support:

- Improved rendering efficiency with less redundant work performed for each eye.
- Seamless integration with the Unity Editor, including in-Editor preview and direct mode support.
- Improved stability and tighter integration with features like anti-aliasing and shadows.
- Non-distorted monoscopic preview on the main monitor.
- Oculus SDK 0.6.0.1 support (PC and mobile).

API Changes in 0.1.0 Beta

The following Unity API changes were made:

- Unity 5.1.1p3 or higher is now required.
- Some script functionality was replaced by UnityEngine.VR.
- Added Crescent Bay support.
- Removed Linux support, for now.
- Removed OvrCapi, the low-level C# wrapper for LibOVR.
- Removed Gear VR MediaSurface functionality. Use Mobile Movie Texture or similar.
- The integration now uses a single stereo camera instead of two separate cameras for the left and right eyes.
- Direct mode now works in the Editor.
- To use VR, you must enable Player Settings -> Virtual Reality Supported.

- DirectToRift.exe no longer applies.
- Editor preview is now monoscopic and does not include lens correction.
- The experimental overlay quad script OVROverlay also works on the PC.

Known Issues

- Pitch, roll, and translation are off for the tracking reference frame in Unity 5.1.1, especially in apps with multiple scenes.
- Mac OS X tearing. VSync is currently broken on the Mac, but works when you build for Gear VR.
- Performance loss. CPU utilization may be slightly higher than in previous versions of Unity.
- OVRPlayerController might end up in an unexpected rotation after OVRDisplay.RecenterPose() is called. To fix it, call RecenterPose() again.

0.6 PC Unity Integration Release Notes

This document provides an overview of new features, improvements, and fixes included in the Oculus Unity Integration that shipped with 0.6 of the Oculus PC SDK.

PC Unity Integration 0.6.0.0

New Features

- Disabled eye texture anti-aliasing when using deferred rendering. This fixes the blackscreen issue.
- Eliminated the need for the DirectToRift.exe in Unity 4.6.3p2 and later.
- Removed the hard dependency from the Oculus runtime. Apps now render in mono without tracking when VR isn't present.

0.5 Mobile Unity Integration Release Notes

This document provides an overview of new features, improvements, and fixes included in the Oculus Unity Integration that shipped with version 0.5 of the Oculus Mobile SDK.

Mobile Unity Integration 0.5.1

Overview of Major Changes

The most significant change in 0.5.1 is to System Activities event handling in Unity. The 0.5.0 code for handling System Activities events in Unity was doing heap allocations each frame. Though this was not a leak, it would cause garbage collection to trigger much more frequently. Even in simple applications, garbage collection routinely takes 1 to 2 milliseconds. In applications that were already close to dropping below 60 Hz, the increased garbage collection frequency could cause notable performance drops. The event handling now uses a single buffer allocated at start up.

As with Mobile SDK v 0.5.0, Unity developers using this SDK version must install the Oculus Runtime for Windows or OS X. This requirement will be addressed in a future release of the SDK.

Bug Fixes

- Rework System Activities Event handling to prevent any per-frame allocations that could trigger Garbage Collector.

Known Issues

- For use with the Mobile SDK, we recommend Unity versions 4.6.3. The Mobile SDK is compatible with Unity 5.0.1p2, which addresses a problem with OpenGL ES 3.0, but there is still a known Android ION memory leak. Please check back for updates.

Mobile Unity Integration 0.5.0

Overview of Major Changes

The Mobile Unity Integration is now synced with the Oculus PC SDK 0.5.0.1 Beta. Please ensure you have installed the corresponding 0.5.0.1 Oculus runtime; it can be found at the following location: *https://developer.oculus.com/downloads/*

VrPlatform entitlement checking is now disabled by default in Unity; handling for native development is unchanged. If your application requires this feature, please refer to the Mobile SDK Documentation for information on how to enable entitlement checking.

New Features

w

- Synced with the Oculus PC SDK 0.5.0.1 Beta.
- VrPlatform entitlement checking is now disabled by default.

Bug Fixes

- Health and Safety Warning no longer displays in editor Play Mode if a DK2 is not attached.

Known Issues

- For use with the Mobile SDK, we recommend Unity versions 4.6.3, which includes Android 5.0 - Lollipop support as well as important Android bug fixes. While the Mobile SDK is compatible with Unity 5.0.0p2 and higher, several issues are still known to exist, including an Android ION memory leak and compatibility issues with OpenGL ES 3.0. Please check back for updates.

Mobile Unity Integration 0.5.0

Overview of Major Changes

The Mobile Unity Integration is now synced with the Oculus PC SDK 0.5.0.1 Beta. Please ensure you have installed the corresponding 0.5.0.1 Oculus runtime; it can be found at the following location: *https://developer.oculus.com/downloads/*

VrPlatform entitlement checking is now disabled by default in Unity; handling for native development is unchanged. If your application requires this feature, please refer to the Mobile SDK Documentation for information on how to enable entitlement checking.

New Features

- Synced with the Oculus PC SDK 0.5.0.1 Beta.
- VrPlatform entitlement checking is now disabled by default.

Bug Fixes

- Health and Safety Warning no longer displays in editor Play Mode if a DK2 is not attached.

Known Issues

- For use with the Mobile SDK, we recommend Unity versions 4.6.3, which includes Android 5.0 - Lollipop support as well as important Android bug fixes. While the Mobile SDK is compatible with Unity 5.0.0p2 and higher, several issues are still known to exist, including an Android ION memory leak and compatibility issues with OpenGL ES 3.0. Please check back for updates.

0.4 Mobile Unity Integration Release Notes

This document provides an overview of new features, improvements, and fixes included in the Oculus Unity Integration that shipped with version 0.4 of the Oculus Mobile SDK.

Mobile Unity Integration 0.4.3

New Features

- New Mobile Unity Integration Based on Oculus PC SDK 0.4.4

Mobile Unity Integration 0.4.2

Overview of Major Changes

If you are developing with Unity, we recommend updating to Unity 4.6.1, which contains Android 5.0 – Lollipop support.

We would like to highlight the inclusion of the new Mobile Unity Integration with full DK2 support based on the Oculus PC SDK 0.4.4. As this is a significant API refactor, please refer to the Unity Development Guide: Migrating From Earlier Versions section for information on how to upgrade projects built with previous versions of the Mobile Unity Integration.

API Changes

- Fix for camera height discrepancies between the Editor and Gear VR device.
- Moonlight Debug Util class names now prefixed with OVR to prevent namespace pollution.
- Provide callback for configuring VR Mode Parms on OVRCameraController; see OVRModeParms.cs for an example.

Mobile Unity Integration 0.4.1

Overview of Major Changes

Added support for Android 5.0 (Lollipop) and Unity Free.

New Features

- Added Unity Free support for Gear VR developers.

Mobile Unity Integration 0.4.0

Overview of Major Changes

First public release of the Oculus Mobile SDK.

Bug Fixes

- Unity vignette rendering updated to match native (slightly increases effective FOV).
- Unity volume pop-up distance to match native.

Migrating From Earlier Versions

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

API Changes

The following are changes to Unity components:

Table 3: Unity Components

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

The following are changes to helper classes:

Table 4: Helper Classes

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

The following are changes to events:

Table 5: 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.

Behavior Changes

- OVRCameraRig's position is always the initial center eye position.
- Eye anchor Transforms are tracked in OVRCameraRig's local space.
- OVRPlayerController's position is always at the user's feet.
- IPD and FOV are fully determined by profile (PC only).
- Layered rendering: multiple OVRCameraRigs are fully supported (not advised for mobile).
- OVRCameraRig.*EyeAnchor Transforms give the relevant poses.

Upgrade Procedure

To upgrade, follow these steps:

1. Ensure you didn't modify the structure of the OVRCameraController prefab. If your eye cameras are on Game Objects named "CameraLeft" and "CameraRight" which are children of the OVRCameraController Game Object (the default), then the prefab should cleanly upgrade to OVRCameraRig and continue to work properly with the new integration.
2. Write down or take a screenshot of 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
    - OVR Internal folder (if applicable)
    - Any file in the Plugins folder with "Oculus" or "OVR" in the name
    - Android-specific assets in the Plugins/Android folder, including: vrlib.jar, libOculusPlugin.so, res/raw and res/values folders
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 described above. Note that the substitution of prefabs does not take place until after all script compile errors have been fixed.
9. Re-apply your previous settings to OVRCameraRig, OVRPlayerController, and OVRManager. Note that the runtime camera positions have been adjusted to better match the camera positions set in the Unity editor. If this is undesired, you can get back to the previous positions by adding a small offset to your camera:
    a. **Adjust the camera's y-position.**
        a. If you previously used an OVRCameraController without an OVRPlayerController, add 0.15 to the camera y-position.
        b. If you previously used an OVRPlayerController with *Use Player Eye Height* checked on its OVRCameraContoller, then you have two options. You may either (1) rely on the new default player eye-height (which has changed from 1.85 to 1.675); or (2) uncheck *Use Profile Data* on the converted OVRPlayerController and then manually set the height of the OVRCameraRig to 1.85 by setting its y-position. Note that if you decide to go with (1), then this height should be expected to change when profile customization is added with a later release.
        c. If you previously used an OVRPlayerController with *Use Player Eye Height* unchecked on its OVRCameraContoller, then be sure uncheck *Use Profile Data* on your converted OVRPlayerController. Then, add 0.15 to the y-position of the converted OVRCameraController.
    b. **Adjust the camera's x/z-position.** If you previously used an OVRCameraController without an OVRPlayerController, add 0.09 to the camera z-position relative to its y rotation (i.e. +0.09 to z if it has 0 y-rotation, -0.09 to z if it has 180 y-rotation, +0.09 to x if it has 90 y-rotation, -0.09 to x if it has 270 y-rotation). If you previously used an OVRPlayerController, no action is needed.
10. Re-start Unity

Common Script Conversions

```
OVRCameraController -> OVRCameraRig
   cameraController.GetCameraPosition() -> cameraRig.rightEyeAnchor.position
   cameraController.GetCameraOrientation() -> cameraRig.rightEyeAnchor.rotation
   cameraController.NearClipPlane -> cameraRig.rightEyeCamera.nearClipPlane
   cameraController.FarClipPlane -> cameraRig.rightEyeCamera.farClipPlane
   cameraController.GetCamera() -> cameraRig.rightEyeCamera

   ---------------------------------------------------------------------
   if ( cameraController.GetCameraForward( ref cameraForward ) &&
   cameraController.GetCameraPosition( ref cameraPosition ) )
   {
```

```
...

to

if (OVRManager.display.isPresent)
{

     // get the camera forward vector and position
     Vector3 cameraPosition = cameraController.centerEyeAnchor.position;
     Vector3 cameraForward = cameraController.centerEyeAnchor.forward;
...
-------------------------------------------------------------------------

OVRDevice.ResetOrientation();
to
OVRManager.display.RecenterPose();

-------------------------------------------------------------------------

cameraController.ReturnToLauncher();

to

OVRManager.instance.ReturnToLauncher();

-------------------------------------------------------------------------

OVRDevice.GetBatteryTemperature();
OVRDevice.GetBatteryLevel();

to

OVRManager.batteryTemperature
OVRManager.batteryLevel

-------------------------------------------------------------------------

OrientationOffset

Set rotation on the TrackingSpace game object instead.

-------------------------------------------------------------------------

FollowOrientation

-------------------------------------------------------------------------

FollowOrientation is no longer necessary since OVRCameraRig applies tracking
in local space. You are free to script the rig's pose or make it a child of
another Game Object.
```

Oculus Changes in Unity

This document provides a non-exhaustive list of changes to Oculus support in Unity. For more information on first-party changes to Unity VR support for Oculus, see the Unity Release Notes for the appropriate version.

Oculus Changes in Unity v 5.3.5p2 and 5.4.0b21

Overview of Major Changes

Built-In Audio Spatialization

Unity Beta versions v 5.4.0b16 and later feature a basic version of our Oculus Native Spatializer Plugin for Unity 5 (ONSP). It makes it trivially easy to add basic spatialization (HRTF transforms) to audio point sources in your Unity project. For more information, see *First-Party Audio Spatialization (Beta)* in our Oculus Native Spatializer for Unity guide.

New Features

- If you assign Unity's VR splash image, it will display as a world-locked quad using Asynchronous TimeWarp, which allows full position-tracked rendering while Unity prioritizes loading. To improve the user's experience, the splash screen appears within 0.5s of app launch.

Bug Fixes

- Fixed a number of OpenGL ES errors, such as EGL_BAD_ALLOC due to implicit management of Unity's WindowSurface.
- Fixed extra 180 degree yaw rotation on all OVROverlay quads.
- Fixed a few cases where D3D texture references were not released.
- Fixed start-up crash in Unity 4 in some applications.

Unity Sample Framework Release Notes

This section describes changes for each version release of the Unity Sample Framework Unity Project.

1.12 Unity Sample Framework

1.12.0

This version of the Oculus Sample Framework for Unity 5 pulls in the most recent Avatar SDK and Utilities for Unity versions. Importing the Avatar SDK and Utilities for Unity 5 separately is not required. It is compatible with Unity 5.4 and up - please check *Compatibility and Version Requirements* on page 5 for up-to-date Unity version recommendations.

For complete instructions on downloading and using the Sample Framework, see *Unity Sample Framework* on page 72 in our developer documentation.

New Features

- Added OverlayUIDemo, which demonstrates creating a UI with a VR Compositor Layer to improve image quality and anti-aliasing. Includes a quad overlay for Rift, and a quad and a cylinder overlay for mobile.

Known Issues

- In Oculus Rift builds, Oculus Remote support is currently limited to opening and closing the menu system.
- Sample Framework Android builds use a custom manifest and are not visible from Applications, and cannot be launched from Oculus Home or the Android Application Launcher. To launch:

  1. Install the APK to your phone.
  2. Open *Settings* > *Applications* > *Application Manager* > *Gear VR Service*.
  3. Select *Storage*.
  4. Select *Manage Storage*.

5. Toggle *Add icon to app list to On*.
6. Close *Settings*.
7. Open *Apps*.
8. Select *Gear VR Service*.
9. Select *Oculus Sample Framework* to launch.

1.11 Unity Sample Framework

1.11.0

The Oculus Unity Sample Framework assists developers in implementing Unity applications by providing sample scenes and guidelines for common VR-specific features such as hand presence, crosshairs, driving, and first-person movement.

This download includes a Unity Package of the Sample Framework. VR applications of the Sample Framework are also available for the Oculus Rift from our *Downloads page*, and for the Samsung Gear VR from the Gallery section of the Oculus Store.

This version of the Oculus Sample Framework for Unity 5 pulls in the most recent Avatar SDK and Utilities for Unity versions. Importing the Avatar SDK and Utilities for Unity 5 separately is not required. It is compatible with Unity 5.4 and up - please check *Compatibility and Version Requirements* on page 5 for up-to-date Unity version recommendations.

For complete instructions on downloading and using the Sample Framework, see *Unity Sample Framework* on page 72 in our developer documentation.

New Features

- Added Hands category with three new scenes.
  - AvatarWithGrab uses the Unity Avatar SDK and the Utilities for Unity scripts OVRGrabber and OVRGrabbable to illustrate hands presence with Touch. Pick up and throw blocks from a table using the Touch grip buttons.
  - CustomControllers is a simple sample displaying tracked and animated Touch models in a scene.
  - CustomHands uses low-resolution custom hand models and the Utilities for Unity scripts OVRGrabber and OVRGrabbable to illustrate hands presence with Touch. Pick up and throw blocks from a table using the Touch grip buttons. May be used as a reference for implementing your own hand models.
- Added Touch support.
- The Sample Framework now pulls in the Avatar SDK, which is also available separately from our *Downloads page*.
- The project download now ships as a unitypackage rather than a zipped Unity project.

Known Issues

- In Oculus Rift builds, Oculus Remote support is currently limited to opening and closing the menu system.
- Sample Framework Android builds use a custom manifest and are not visible from Applications, and cannot be launched from Oculus Home or the Android Application Launcher. To launch:
  1. Install the APK to your phone.
  2. Open *Settings* > *Applications* > *Application Manager* > *Gear VR Service*.
  3. Select *Storage*.
  4. Select *Manage Storage*.
  5. Toggle *Add icon to app list to On*.
  6. Close *Settings*.

7. Open *Apps*.
8. Select *Gear VR Service*.
9. Select *Oculus Sample Framework* to launch.

1.5 Unity Sample Framework

1.5.1

The Oculus Unity Sample Framework assists developers in implementing Unity applications by providing sample scenes and guidelines for common VR-specific features such as crosshairs, driving, and first-person movement.

This download includes a Unity Project of the Sample Framework. VR applications of the Sample Framework are also available for the Oculus Rift from our Downloads page, and for the Samsung Gear VR from the Gallery section of the Oculus Store.

This version of the Oculus Sample Framework for Unity 5 pulls in Utilities for Unity version 1.9. Importing Utilities for Unity 5 separately is not required. It is compatible with Unity 5.3 and up - please check *Compatibility and Version Requirements* on page 5 for up-to-date Unity version recommendations.

For complete instructions on downloading and using the Sample Framework, see *Unity Sample Framework* on page 72 in our developer documentation.

New Features

- Added two Mono Optimization sample scenes in which near content is rendered stereoscopically and distant content is rendered monoscopically. Depending on the scene, this approach may produce significant rendering performance improvements. For a detailed explanation, see Hybrid Mono Rendering in UE4 and Unity in our Developer Blog.

Known Issues

- In Oculus Rift builds, Oculus Remote support is currently limited to opening and closing the menu system.
- Sample Framework Android builds use a custom manifest and are not visible from Applications, and cannot be launched from Oculus Home or the Android Application Launcher. To launch:

  1. Install the APK to your phone.
  2. Open *Settings* > *Applications* > *Application Manager* > *Gear VR Service*.
  3. Select *Storage*.
  4. Select *Manage Storage*.
  5. Toggle *Add icon to app list to On*.
  6. Close *Settings*.
  7. Open *Apps*.
  8. Select *Gear VR Service*.
  9. Select *Oculus Sample Framework* to launch.

1.5.0

The Oculus Unity Sample Framework assists developers in implementing Unity applications by providing sample scenes and guidelines for common VR-specific features such as crosshairs, driving, and first-person movement.

This download includes a Unity Project of the Sample Framework. VR applications of the Sample Framework are also available for the Oculus Rift from our Downloads page, and for the Samsung Gear VR from the Gallery Apps section of the Oculus Store.

This version of the Oculus Sample Framework for Unity 5 pulls in Utilities for Unity version 1.5. Importing Utilities for Unity 5 separately is no longer required. It is compatible with Unity 5.3 and up - please check *Compatibility and Version Requirements* on page 5 for up-to-date Unity version recommendations.

For complete instructions on downloading and using the Sample Framework, see *Unity Sample Framework* on page 72 in our developer documentation.

New Features

- Added Movie Player, Per-Eye Cameras, Multiple Cameras, and Render Frame Rate scenes.
- Reorganized in-VR scenes list structure to de-clutter the scenes menu.
- Now includes Utilities for Unity 5 v 1.5; separately importing the Utilities unitypackage is no longer required.

Known Issues

- In Oculus Rift builds, Oculus Remote support is currently limited to opening and closing the menu system.

1.3 Unity Sample Framework

This version of the Oculus Sample Framework for Unity 5 is for use with our Utilities for Unity version 1.3 and compatible Unity versions, and supports the Oculus Rift CV1 runtime. Please check *Compatibility and Version Requirements* on page 5 for up-to-date Unity version recommendations.

The Oculus Unity Sample Framework assists developers in implementing Unity applications by providing sample scenes and guidelines for common VR-specific features such as crosshairs, driving, and first-person movement. The Sample Framework can guide developers in producing reliable, comfortable applications and avoiding common mistakes.

The Oculus Unity Sample Framework consists of a Unity project as well as application binaries for playing the sample scenes in VR. Sample scenes are navigated and controlled in-app with a simple UI, which also provides explanatory notes.

This download includes a Unity Project of the Sample Framework. VR applications of the Sample Framework are also available for the Oculus Rift/DK2 from our Downloads page, and for the Samsung Gear VR from the Gallery Apps section of the Oculus Store.

For complete instructions on downloading and using the Sample Framework, see *Unity Sample Framework* on page 72 in our developer documentation.

Migrating to Utilities from the Integration Package

Moving to Unity 5.1 is a substantial upgrade for VR development, and we recommend carefully choosing your time frame for making the update. You may encounter problems related to VR performance or otherwise.

Please let us know about any issues you encounter in the *Oculus Unity Forum*, and keep your eye out for updates.

Delete Previously-Imported Assets

If you have previously imported a Unity integration package, delete all Oculus Integration content before importing the new Unity package. For detailed instructions, see *Importing the Oculus Utilities Package*.

Upgrade Procedure

1. Replace any usage of OVRManager.instance.virtualTextureScale or OVRManager.instance.nativeTextureScale with UnityEngine.VR.VRSettings.renderScale. The value of

renderScale is equal to nativeTextureScale * virtualTextureScale. If you set renderScale to a value that is less than or equal to any value it has had since the application started, then virtual texture scaling will be used. If you increase it higher, to a new maximum value, then the native scale is increased and the virtual scale is set to 1.

2. Replace any usage of OVRManager.instance.eyeTextureAntiAliasing with UnityEngine.QualitySettings.antiAliasing. Instead of multisampling the back-buffer when VR is enabled, Unity multisamples the eye buffers.
3. Remove any usage of OVRManager.instance.timeWarp and OVRManager.instance.freezeTimeWarp. TimeWarp is always on and cannot be frozen.
4. Do not assume there are Cameras on OVRCameraRig.leftEyeAnchor or rightEyeAnchor. Instead of calling GetComponent<Camera>(), use Camera.main or, for backward compatibility, use OVRCameraRig.leftEyeCamera or rightEyeCamera.
5. Move any scripts, image effects, tags, or references from the Cameras on OVRCameraRig.leftEyeAnchor and rightEyeAnchor to the one on centerEyeAnchor.
6. Remove any usage of OvrCapi.cs. The CAPI C# binding is no longer available. If you need to access CAPI, use UnityEngine.VR.VRDevice.GetNativePtr() to get an ovrHmd pointer and then pass it to a native plugin that uses the Oculus SDK corresponding to your Unity version. For more on which Unity versions correspond to which SDKs, see "Integration Versions" in *Compatibility and Requirements*.

Importing Utilities Package into Legacy Projects

The legacy Oculus Unity Integration used a separate Camera component on each eye anchor. Oculus Utilities for Unity uses single camera on the center eye anchor. If the Oculus Utilities package is imported into an old project built with the legacy Unity integration, editor scripts will patch up old OVRCameraRig Game Objects and enable PlayerSettings.virtualRealitySupported, so the project should work without further action.

Note: Don't forget to move any scripts, image effects, tags, or references from the left and right eye anchors to the center eye anchor as noted above.

Unity 4.x Legacy Integration Developer Guide

The Unity Legacy Integration Developer Guide is designed to get you started with creating great VR experiences in Unity 4.

Introduction

This guide covers developing Unity 4 games and applications for the Oculus Rift and Samsung Gear VR.

The legacy integration provides VR support for development with **Unity 4.x** Professional and Free editions. It also includes Unity prefabs, C# scripts, sample scenes, and more to assist with development.

Projects using **Unity 5.1 and later** should use the *Oculus Utilities for Unity* package instead of this integration. Developers beginning new projects should use Unity 5.1+.

For information on recommended and supported Unity versions, see *Compatibility and Requirements*.

This guide covers:

- Getting started
- Downloading and installing the Oculus Unity Integration
- Package contents
- Input Mapping

- How to use the provided samples, assets, and sample applications
- Configuring Unity VR projects for build to various targets
- Getting Started Frequently Asked Questions
- Debugging and Performance Analysis
- Getting Started Tutorial (build a simple VR game)

Most information in this guide applies to the use of the Utilities package for either Rift or Mobile development. Any exceptions are clearly indicated where they occur.

For a complete API reference, see our *Unity Developer Reference*.

When developing for both Rift and mobile platforms, keep in mind that the requirements for PC and mobile VR applications differ substantially. If you would like to generate builds for both PC and mobile from a single project, it is important to follow the more stringent mobile development best practices.

Compatibility and Requirements

System and Hardware Requirements

To verify that you are using supported hardware, please review the relevant PC or mobile setup documentation:

- PC SDK: *Getting Started with the SDK*
- Mobile SDK: *System and Hardware Requirements*

Unity Version Compatibility

Unity 5.1 or later provide built-in support for Rift and Gear VR development. The optional Oculus Utilities for Unity package offers additional developer resources.

All developers should use Unity 5. Legacy support is available for Unity 4 - see our *Unity 4.x Legacy Integration Developer Guide* on page 106 for more information.

Unity Version	Recommended Release
Unity 5.4	Unity 5.4.5f1 and later
Unity 5.5	Unity 5.5.2p3 and later
Unity 5.6	Unity 5.6.0f3 and later
Unity 4	Unity 4.7.0f1 (legacy support only)

For complete details Oculus SDK or Integration version compatibility with Unity, see *Unity-SDK Version Compatibility*.

OS Compatibility

- **Windows**: Windows 7, 8, 10
- **Mac**: OS X Yosemite, El Capitan, Sierra

OS X development requires the Oculus Rift Runtime for OS X, available from our *Downloads page*. Note that runtime support for OS X is legacy only. It does not support consumer versions of Rift.

Unity Personal and Professional Licenses

The Unity Personal and Professional licenses both provide built-in Rift support. Mobile developers using the Unity Free license receive basic Android support automatically. Mobile developers using a Professional license require an Android Pro Unity license.

For more information, see *License Comparisons* in Unity's documentation.

Controllers

You may wish to have a controller for development or to use with the supplied demo applications. Available controllers include the Oculus Touch or Xbox 360 controller for Rift, and the Gear VR Controller for mobile development.

Downloading the Legacy Integration

All Oculus Unity development materials are available for download at our Developer site: *https://developer.oculus.com/downloads/*

Legacy Integration Contents

- Oculus Integration package
- Project Settings assets

Included with Mobile SDK

- SDKExamples (source)
- BlockSplosion sample application (source and pre-built apk, for Mobile SDK)
- Shadowgun sample application (pre-built-apk, for Mobile SDK)

SDK Examples is a resource for mobile developers that includes Unity example scenes illustrating the use of common resources such as menus, crosshairs, and more. See *Oculus Mobile SDKExamples* for more information.

Also Available

Additional development resources are available separately from our *Downloads page*.

- Legacy Oculus Spatializer for Unity (included with Oculus Audio SDK Plugins; the Native OSP is for use with Unity 5)
- OVRMonitor (includes the VrCapture library and Oculus Remote Monitor client for mobile performance analysis and debugging; see our *OVRMonitor documentation* for more information).

Preparing for Development: Rift

We recommend Rift developers begin by reviewing the *PC Developer Guide*. Mobile developers should review *Mobile VR Application Development*.

Recommended Configuration

- On Windows, enable Direct3D 11 - it exposes the most advanced VR rendering capabilities. Direct3D 9 and Windows OpenGL are not supported. D3D 12 is currently available as an experimental feature.
- 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).

Preparing for Development: Mobile

This guide describes setup requirements for Oculus mobile VR development with Unity.

Android SDK Setup and Oculus Mobile SDK

The Android SDK is required for mobile development with Unity. However, most Unity developers do not need to install Android Studio or NDK. Unity developers should follow the instructions in our *Device Setup guide*, and install the Java Development Kit (JDK) and Android SDK before beginning development. See Unity's *Getting Started with Android Development* for more information.

OVRMonitor is available to Unity 4 developers as a separate download. However, Unity 4 developers must download the Oculus Mobile SDK for the SDK Examples.

When developing for mobile, please be sure to fully review all of the relevant performance and design documentation, especially the *Unity Best Practices: Mobile*. Mobile apps are subject to more stringent limitations and requirements and computational limitations which should be taken into consideration from the ground up.

Application Signing

Mobile applications require two different signatures at different stages of development. Be sure to read the *Application Signing* section of the Mobile SDK documentation for more information.

Application Entitlement Checking

Entitlement checking, used to protect apps from unauthorized distribution, is disabled by default in Unity. For more information and instructions, see *Getting Started with the SDK* in our Platform guide.

Getting Started

This section describes how to begin working in Unity.

Importing the Unity Integration

The Oculus Integration asset package is the heart of the supplied development resources. Importing it will install the minimum set of required files necessary for VR integration into Unity. We have also included various assets, scripts, and sample scenes to assist with development.

Create or Open Unity Project

If you are already working in a Unity project, save your work before beginning.

Otherwise, create a new project into which you will import the Oculus assets:

1. From the Unity menu, select *File* > *New Project*.
2. Click the *Browse* button and select the folder where the Unity project will be located.
3. Make sure that the *Setup defaults for:* field is set to *3D*.
4. You do not need to import any standard or pro Unity asset packages, as the Oculus Unity integration is fully self-contained.
5. Click the *Create* button. Unity will reopen with the new project loaded.

Delete Previously-Imported Assets

If you have previously imported a Unity Integration package, delete all Oculus Integration content before importing the new Unity package. Be sure to close the Unity Editor, then navigate to your Unity project folder and delete the following:

Folder	Content to Delete
Assets/Plugins	Oculus.* OVR.*
Assets/Plugins/ Android/	*Oculus* AndroidManifest.xml *vrapi* *vrlib* *vrplatlib*
Assets/Plugins/x86/	Oculus.* OVR.*
Assets/Plugins/ x86_64/	Oculus.* OVR.*

Import Integration Package

To import the integration into Unity, select *Assets* > *Custom Package...* and select the Unity Integration .unityPackage to import the assets into your new project. Alternately, you can simply find the .unityPackage file in your file system and double-click to launch.

When the *Importing package* dialog box opens, leave all of the boxes checked and select *Import*. The import process may take a few minutes to complete.

Copy Project Settings (Mobile)

The mobile Unity Integration includes a *Project Settings* folder which provides default settings for a VR mobile application. You may manually copy these files to your [Project]/Assets/ProjectSettings folder. Be sure to keep a copy in your Mobile SDK folder to copy later for use with additional projects.

To Import BlockSplosion Sample Application (Mobile)

Note: The Room sample scene included in the integration package may be used for Rift development.

To import the SDKExamples into Unity, select *Assets* > *Custom Package...* and select BlockSplosion.unityPackage to import the assets into your new project. Alternately, you can simply find the .unityPackage file in your file system and double-click to launch.

When the *Importing package* dialog box opens, leave all of the boxes checked and select *Import*. The import process may take a few minutes to complete.

Each sample application project includes a *ProjectSettings* folder, which provides default settings for the VR mobile application. Copy these files to your [Project]/Assets/ProjectSettings folder.

Adding VR to an Existing Unity Project

Note: This process is covered in greater detail in our Tutorial *Build a Simple VR Unity Game*.

The Unity Integration package may be used to integrate Oculus VR into an existing project. This may be useful as a way of getting oriented to VR development, but dropping a VR camera into a Unity game that wasn't designed with VR best practices in mind is unlikely to produce a great experience.

1. Import package.
2. Instantiate OVRCameraRig if you already have locomotion figured out or instantiate OVRPlayerController to walk around.
3. Copy any scripts from the non-VR camera to the OVRCameraRig. Any image effect should go to both the Left/RightEyeAnchor GameObjects. These are children of a TrackingSpace GameObject, which is itself a child of OVRCameraRig. The TrackingSpace GameObject allows clients to change the frame of reference used by tracking, e.g., for use with a game avatar.
4. Disable your old non-VR camera.
5. Build your project and run normally.

Note: This is one simple method for adding VR to an existing application, but is by no means the only way. For example, you may not always wish to use OVRPlayerController.

A Detailed Look at the Unity Integration

This section examines the Unity integration, including the directory structure of the integration, the Unity prefabs are described, and several key C# scripts.

Contents

OVR

The contents of the OVR folder in OculusUnityIntegration.unitypackage are uniquely named and should be safe to import into an existing project.

The OVR directory contains the following subdirectories:

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 main GUI display.
Moonlight	Contains classes designed for mobile Gear VR development. Holds sub-folders with mobile equivalents of all top-level folders (Editor, Materials, Prefabs, et cetera).
Prefabs	Contains the main Unity prefabs that are used to provide the VR support for 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 VR framework and Unity components together. Many of these scripts work together within the various Prefabs.
Shaders	Contains various Cg shaders required by some of the OVR components.
Textures	Contains image assets that are required by some of the script components.

Note: We strongly recommend that developers not directly modify the included OVR scripts.

Plugins

The Plugins folder contains vrapi.so and the OculusPlugin.dll, which enables the VR framework 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; and Android/libOculusPlugin.so, vrlib.jar, and AndroidManifest.xml for Android.

Prefabs

The current integration for adding VR support into Unity applications is based on two prefabs that may be added into a scene:

- OVRCameraRig
- OVRPlayerController

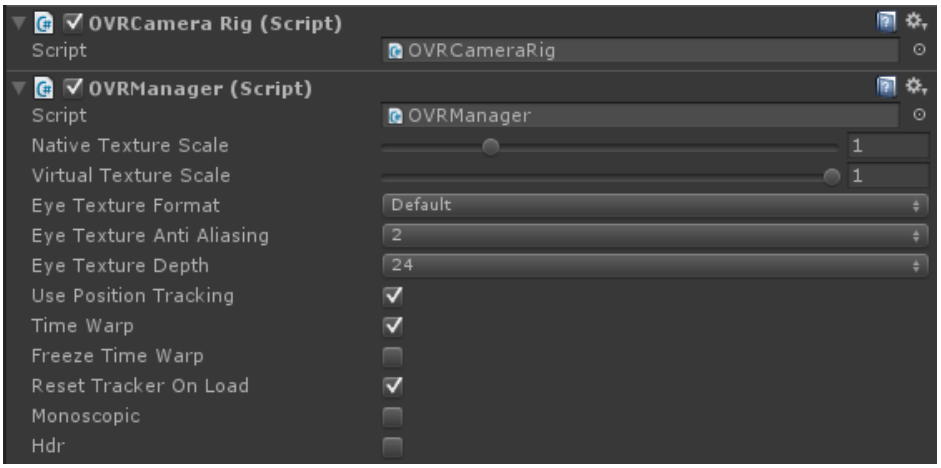
To use, simply drag and drop one of the prefabs into your scene.

OVRCameraRig

OVRCameraRig replaces the regular 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 Gear VR and Rift.

Note: Make sure to turn off any other Camera in the scene to ensure that OVRCameraRig is the only one being used.

Figure 9: 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

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. It 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.

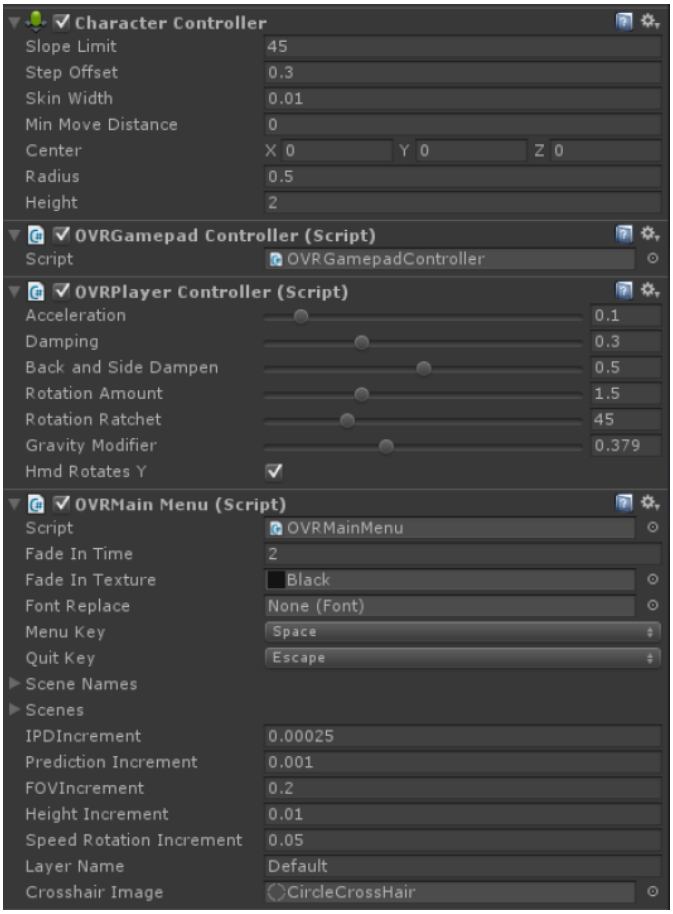
Note: Make sure that collision detection is active in the environment.

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

- OVRPlayerController.cs

- OVRGamepadController.cs

Figure 10: Prefabs: OVRPlayerController, expanded in the inspector



Unity Components

This section gives a general overview of the Components provided by the legacy integration.

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 halfway 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 comfortable VR support to a scene.

Important: All camera control should be done through this component. You should understand this script when implementing your own camera control mechanism.

Mobile and PC Public Members

Updated Anchors	Allows clients to filter the poses set by tracking. Used to modify or ignore positional tracking.

GameObject Structure

TrackingSpace	A GameObject that defines the reference frame used by tracking. You can move this relative to the OVRCameraRig for use cases in which the rig needs to respond to tracker input. For example, OVRPlayerController changes the position and rotation of TrackingSpace to make the character controller follow the yaw of the current head pose.

OVRManager

OVRManager is the main interface to the VR hardware. It is a singleton that exposes the Oculus SDK to Unity, and includes helper functions that use the stored Oculus 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 values in the Unity inspector.

OVRManager.cs contains the following public members:

Table 6: Mobile and PC Public Members

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.
Eye Texture Antialiasing	Sets the level of antialiasing for the eye RenderTextures.

Table 7: PC-Only Public Members

Native Texture Scale	Each camera in the camera controller creates a RenderTexture that is the ideal size for obtaining the sharpest pixel density (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	Disables the IR tracker and causes head position to be inferred from the current rotation using the head model. To fully ignore tracking or otherwise modify tracking behavior, see OVRCameraRig.UpdatedAnchors above
Mirror to Display	When enabled, the undistorted rendered output appears on your desktop in addition to the Rift. If disabled, you may add your own scripts next to OVRCameraRig and set that GameObject's Camera component to render whatever you like. Disabling may slightly improve performance.

Time Warp (desktop only)	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 (desktop only)	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.

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 sensor.

OVRCommon

OVRCommon	OVRCommon is a collection of reusable static functions.

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 VR framework. 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 GameObject contains the matrix which motor control bases it direction on. This GameObject 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 Mac OS. Please use the conventional Unity input methods for gamepad input.
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.
OVRTrackerBounds	Warns players when the HMD moves outside the trackable range.

GameObject Structure

ForwardDirection	An empty GameObject attached to the OVRPlayerController prefab containing the matrix upon which motor control bases its direction. This GameObject should also house the body geometry which will be seen by the player. See TrackingSpace in "OVRCameraRig" for more information

For more information on OVRInput, see *OVRInput* on page 118.

Oculus Mobile SDKExamples

The Mobile SDK includes sample applications to illustrate implementation of common functionality such as a first-person scene and crosshairs.

To import SDKExamples into Unity, begin by creating a new, empty project. Then select *Assets >Import Package > Custom Package...* and select SDKExamples.unityPackage to import the assets into your project. Alternately, you can locate the SDKExamples.unityPackage and double-click to launch, which will have the same effect.

Once imported, replace your Unity project's ProjectSettings folder with the ProjectSettings folder included with SDKExamples.

Note: If you don't replace the ProjectSettings folder, imported scenes will show console errors.

You will find the following sample scenes located in Assets/Scenes:

Cubes	A 3D array of cubes and an OVRCameraRig.
Multicamera	An example of switching cameras in one scene.
Room	A cubic room formed from six cubes enclosing an OVRPlayerController.

The example scripts are located in Assets/OVR/Moonlight/:

OVRChromaticAberration.cs	Drop-in component for toggling chromatic aberration correction on and off for Android.
OVRCrosshair.cs	A component that adds a stereoscopic crosshair to a scene.
OVRDebugGraph.cs	Drop-in component for toggling the TimeWarp debug graph, which is no longer available.
OVRDebugHeadController.cs	A simple behavior that can be attached to the parent of the CameraRig to provide movement via the gamepad, useful for testing applications in Unity without an HMD.
OVRInputControl.cs	Cross-platform wrapper for Unity Input.
OVRModeParms.cs	Example code for de-clocking your application to reduce power and thermal load as well as how to query the current power level state.
OVRMonoscopic.cs	Drop-in component for toggling Monoscopic rendering on and off for Android.
OVROverlay.cs	Add to an object with a Quad mesh filter to have the quad rendered as a TimeWarp overlay instead by drawing it into the eye buffer.
OVRPlatformMenu.cs	Helper component for detecting Back Key long-press to bring-up the Universal Menu and Back Key short-press to bring up the Confirm-Quit to Home Menu. Additionally implements a Wait Timer for displaying Long Press Time. For more information on interface guidelines and requirements, please review *Interface Guidelines* and *Universal Menu* in the *Mobile SDK documentation*.
OVRResetOrientation.cs	Drop-in component for resetting the camera orientation.
OVRTimeWarpUtils.cs	Demonstrates the interface calls for setting important configs, including min vsyncs, enable tw debug graph, and enable cac.
OVRTouchpad.cs	Interface class to a touchpad.
OVRVolumeControl.cs	An on-screen display that shows the current system audio volume.
OVRWaitCursor.cs	Helper component for auto-rotating a wait cursor.

See our *Oculus Utilities for Unity Reference Manual* for a more detailed look at these and other C# scripts. Undocumented scripts may be considered internal, and should generally never be modified.

OVRInput

OVRInput exposes a unified input API for multiple controller types. It may be used to query virtual or raw controller state, such as buttons, thumbsticks, triggers, and capacitive touch data. It currently supports the Oculus Touch, Microsoft Xbox controllers, and the Oculus remote on desktop platforms. For mobile development, it supports the Gear VR Controller as well as the touchpad and back button on the Gear VR headset. Gear VR gamepads must be Android compatible and support Bluetooth 3.0.

For keyboard and mouse control, we recommend using the UnityEngine.Input scripting API (see Unity's *Input scripting reference* for more information).

Mobile input bindings are automatically added to InputManager.asset if they do not already exist.

For more information, see *OVRInput* in the *Unity Scripting Reference* on page 79. For more information on Unity's input system and Input Manager, documented here: *http://docs.unity3d.com/Manual/Input.html* and *http://docs.unity3d.com/ScriptReference/Input.html*.

`SetControllerVibration()` support for Oculus Touch is now deprecated; please use *OVRHaptics for Oculus Touch* on page 48 instead.

Oculus Touch Tracking

OVRInput provides Touch position and orientation data through `GetLocalControllerPosition()` and `GetLocalControllerRotation()`, which return a Vector3 and Quaternion, respectively.

Controller poses are returned by the constellation tracking system and are predicted simultaneously with the headset. These poses are reported in the same coordinate frame as the headset, relative to the initial center eye pose, and may be used for rendering hands or objects in the 3D world. They are also reset by `OVRManager.display.RecenterPose()`, similar to the head and eye poses.

Gear VR Controller

Gear VR Controller provides orientation data through `GetLocalControllerRotation()`, which returns a quaternion.

Gear VR positions the controller relative to the user by using a body model to estimate the controller's position. Whether the controller is visualized on the left or right side of the body is determined by left-handedness versus right-handedness, which is specified by users during controller pairing.

To query handedness of a paired controller, use `IsControllerConnected()` or `GetActiveController()` to query for `RTrackedRemote` or `LTrackedRemote`.

For example:

```
// returns true if right-handed controller connected
OVRInput.IsControllerConnected(OVRInput.Controller.RTrackedRemote);
```

Use `OVRInput.Get()` to query controller touchpad input. You may query the input position with Axis2D:

```
OVRInput.Get(OVRInput.Axis2D.PrimaryTouchpad, OVRInput.Controller.RTrackedRemote);
```

A touchpad touch occurs when the user's finger makes contact with the touchpad without actively clicking it. Touches may be queried with `OVRInput.Get(OVRInput.Touch.PrimaryTouchpad)`. Touchpad clicks are alias to virtual button One clicks, and may be queried with `OVRInput.Get(OVRInput.Button.PrimaryTouchpad)`.

The volume and home buttons are reserved.

OVRInput Usage

The primary usage of OVRInput is to access controller input state through `Get()`, `GetDown()`, and `GetUp()`.

- `Get()` queries the current state of a control.
- `GetDown()` queries if a control was pressed this frame.
- `GetUp()` queries if a control was released this frame.

Control Input Enumerations

There are multiple variations of `Get()` that provide access to different sets of controls. These sets of controls are exposed through enumerations defined by OVRInput as follows:

Control	Enumerates
OVRInput.Button	Traditional buttons found on gamepads, Touch controllers, the Gear VR Controller touchpad and back button, and the Gear VR headset touchpad and back button.
OVRInput.Touch	Capacitive-sensitive control surfaces found on the Oculus Touch and Gear VR Controller.
OVRInput.NearTouch	Proximity-sensitive control surfaces found on the Oculus Touch controllers.
OVRInput.Axis1D	One-dimensional controls such as triggers that report a floating point state.
OVRInput.Axis2D	Two-dimensional controls including thumbsticks and the Gear VR Controller touchpad. Report a Vector2 state.

A secondary set of enumerations mirror the first, defined as follows:

OVRInput.RawButton
OVRInput.RawTouch
OVRInput.RawNearTouch
OVRInput.RawAxis1D
OVRInput.RawAxis2D

The first set of enumerations provides a virtualized input mapping that is intended to assist developers with creating control schemes that work across different types of controllers. The second set of enumerations provides raw unmodified access to the underlying state of the controllers. We recommend using the first set of enumerations, since the virtual mapping provides useful functionality, as demonstrated below.

Button, Touch, and NearTouch

In addition to traditional gamepad buttons, the Oculus Touch controllers feature capacitive-sensitive control surfaces which detect when the user's fingers or thumbs make physical contact (a "touch"), as well as when they are in close proximity (a "near touch"). This allows for detecting several distinct states of a user's interaction with a specific control surface. For example, if a user's index finger is fully removed from a control surface, the NearTouch for that control will report false. As the user's finger approaches the control and gets within close proximity to it, the NearTouch will report true prior to the user making physical contact. When the user makes physical contact, the Touch for that control will report true. When the user pushes the index trigger down, the Button for that control will report true. These distinct states can be used to accurately detect the user's interaction with the controller and enable a variety of control schemes.

The Gear VR Controller touchpad may be queried for both touch status and click status, where "touch" refers to the user's finger making contact with the touchpad without actively clicking it.

Example Usage

```
// returns true if the primary button (typically "A") is currently pressed.
OVRInput.Get(OVRInput.Button.One);

// returns true if the primary button (typically "A") was pressed this frame.
OVRInput.GetDown(OVRInput.Button.One);

// returns true if the "X" button was released this frame.
OVRInput.GetUp(OVRInput.RawButton.X);

// returns a Vector2 of the primary (typically the Left) thumbstick's current state.
// (X/Y range of -1.0f to 1.0f)
OVRInput.Get(OVRInput.Axis2D.PrimaryThumbstick);

// returns true if the primary thumbstick is currently pressed (clicked as a button)
OVRInput.Get(OVRInput.Button.PrimaryThumbstick);

// returns true if the primary thumbstick has been moved upwards more than halfway.
// (Up/Down/Left/Right - Interpret the thumbstick as a D-pad).
OVRInput.Get(OVRInput.Button.PrimaryThumbstickUp);

// returns a float of the secondary (typically the Right) index finger trigger's current state.
// (range of 0.0f to 1.0f)
OVRInput.Get(OVRInput.Axis1D.SecondaryIndexTrigger);

// returns a float of the left index finger trigger's current state.
// (range of 0.0f to 1.0f)
OVRInput.Get(OVRInput.RawAxis1D.LIndexTrigger);

// returns true if the left index finger trigger has been pressed more than halfway.
// (Interpret the trigger as a button).
OVRInput.Get(OVRInput.RawButton.LIndexTrigger);

// returns true if the secondary gamepad button, typically "B", is currently touched by the user.
OVRInput.Get(OVRInput.Touch.Two);

// returns true after a Gear VR touchpad tap
OVRInput.GetDown(OVRInput.Button.One);

// returns true on the frame when a user's finger pulled off Gear VR touchpad controller on a swipe
 down
OVRInput.GetDown(OVRInput.Button.DpadDown);

// returns true the frame AFTER user's finger pulled off Gear VR touchpad controller on a swipe right
OVRInput.GetUp(OVRInput.RawButton.DpadRight);

// returns true if the Gear VR back button is pressed
OVRInput.Get(OVRInput.Button.Two);

// Returns true if the the Gear VR Controller trigger is pressed down
OVRInput.Get(OVRInput.Button.PrimaryIndexTrigger);

// Queries active Gear VR Controller touchpad click position
// (normalized to a -1.0, 1.0 range, where -1.0, -1.0 is the lower-left corner)
OVRInput.Get(OVRInput.Axis2D.PrimaryTouchpad, OVRInput.Controller.RTrackedRemote);

// If no controller is specified, queries the touchpad position of the active Gear VR Controller
OVRInput.Get(OVRInput.Axis2D.PrimaryTouchpad);

// returns true if the Gear VR Controller back button is pressed
OVRInput.Get(OVRInput.Button.Back);

// returns true on the frame when a user's finger pulled off Gear VR Controller back button
OVRInput.GetDown(OVRInput.Button.Back);
```

In addition to specifying a control, `Get()` also takes an optional controller parameter. The list of supported controllers is defined by the OVRInput.Controller enumeration (for details, refer to *OVRInput* in the *Unity Scripting Reference* on page 79.

Specifying a controller can be used if a particular control scheme is intended only for a certain controller type. If no controller parameter is provided to `Get()`, the default is to use the `Active` controller, which corresponds to the controller that most recently reported user input. For example, a user may use a pair of Oculus Touch controllers, set them down, and pick up an Xbox controller, in which case the Active controller will switch to the Xbox controller once the user provides input with it. The current Active controller can be queried with `OVRInput.GetActiveController()` and a bitmask of all the connected Controllers can be queried with `OVRInput.GetConnectedControllers()`.

Example Usage:

```
// returns true if the Xbox controller's D-pad is pressed up.
OVRInput.Get(OVRInput.Button.DpadUp, OVRInput.Controller.Gamepad);

// returns a float of the Hand Trigger's current state on the Left Oculus Touch controller.
OVRInput.Get(OVRInput.Axis1D.PrimaryHandTrigger, OVRInput.Controller.Touch);

// returns a float of the Hand Trigger's current state on the Right Oculus Touch controller.
OVRInput.Get(OVRInput.Axis1D.SecondaryHandTrigger, OVRInput.Controller.Touch);
```

Querying the controller type can also be useful for distinguishing between equivalent buttons on different controllers. For example, if you want code to execute on input from a gamepad or Touch controller, but not on a Gear VR Touchpad, you could implement it as follows:

```
if (OVRInput.GetActiveController() != OVRInput.Controller.Touchpad) { /* do input handling */ }
```

Note that the Oculus Touch controllers may be specified either as the combined pair (with `OVRInput.Controller.Touch`), or individually (with `OVRInput.Controller.LTouch` and `RTouch`). This is significant because specifying LTouch or RTouch uses a different set of virtual input mappings that allow more convenient development of hand-agnostic input code. See the virtual mapping diagrams in *Touch Input Mapping* for an illustration.

Example Usage:

```
// returns a float of the Hand Trigger's current state on the Left Oculus Touch controller.
OVRInput.Get(OVRInput.Axis1D.PrimaryHandTrigger, OVRInput.Controller.LTouch);

// returns a float of the Hand Trigger's current state on the Right Oculus Touch controller.
OVRInput.Get(OVRInput.Axis1D.PrimaryHandTrigger, OVRInput.Controller.RTouch);
```

This can be taken a step further to allow the same code to be used for either hand by specifying the controller in a variable that is set externally, such as on a public variable in the Unity Editor.

Example Usage:

```
// public variable that can be set to LTouch or RTouch in the Unity Inspector
public Controller controller;
…
// returns a float of the Hand Trigger's current state on the Oculus Touch controller
// specified by the controller variable.
OVRInput.Get(OVRInput.Axis1D.PrimaryHandTrigger, controller);

// returns true if the primary button ("A" or "X") is pressed on the Oculus Touch controller
// specified by the controller variable.
OVRInput.Get(OVRInput.Button.One, controller);
```

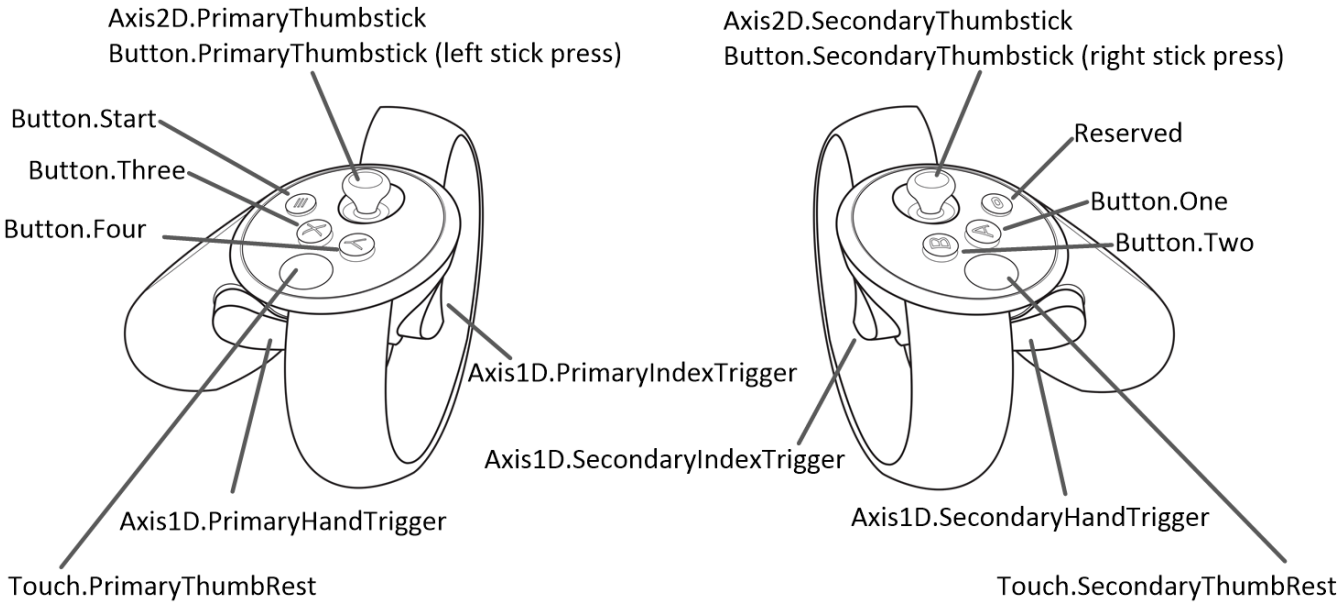
This is convenient since it avoids the common pattern of if/else checks for Left/Right hand input mappings.

Touch Input Mapping

The following diagrams illustrate common input mappings for Oculus Touch controllers. For more information on additional mappings that are available, refer to *OVRInput* in the *Unity Scripting Reference* on page 79.

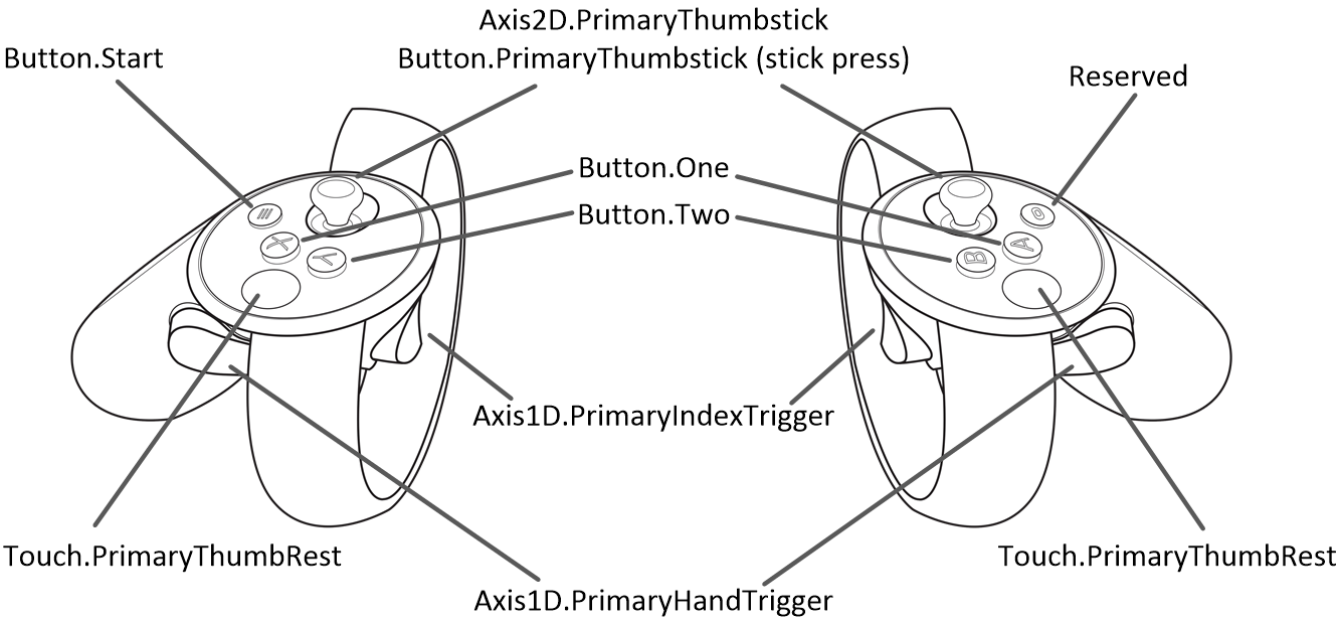
Virtual Mapping (Accessed as a Combined Controller)

When accessing the Touch controllers as a combined pair with OVRInput.Controller.Touch, the virtual mapping closely matches the layout of a typical gamepad split across the Left and Right hands.



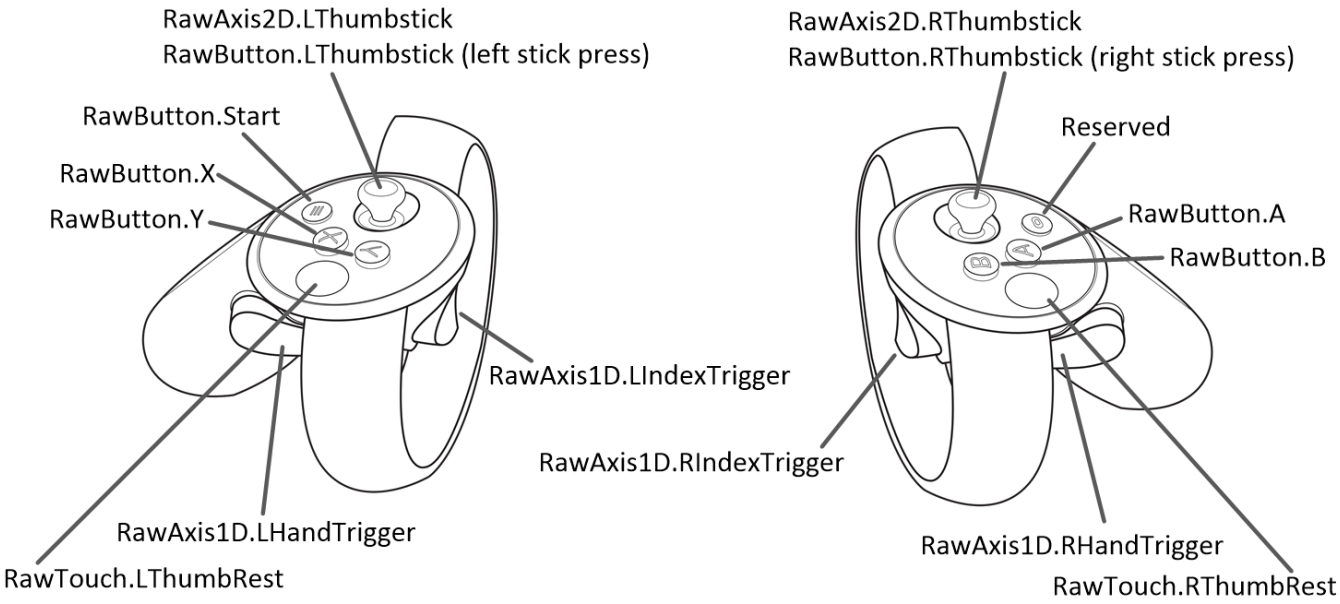
Virtual Mapping (Accessed as Individual Controllers)

When accessing the Left or Right Touch controllers individually with OVRInput.Controller.LTouch or OVRInput.Controller.RTouch, the virtual mapping changes to allow for hand-agnostic input bindings. For example, the same script can dynamically query the Left or Right Touch controller depending on which hand it is attached to, and Button.One will be mapped appropriately to either the A or X button.



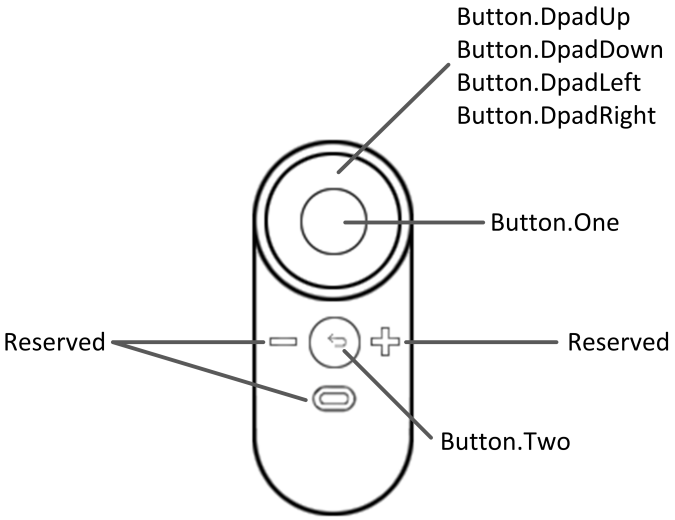
Raw Mapping

The raw mapping directly exposes the Touch controllers. The layout of the Touch controllers closely matches the layout of a typical gamepad split across the Left and Right hands.

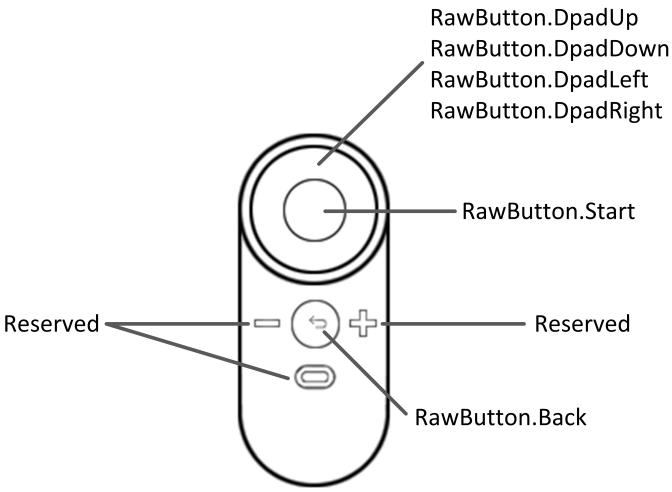


Rift Remote Input Mapping

Virtual Mapping



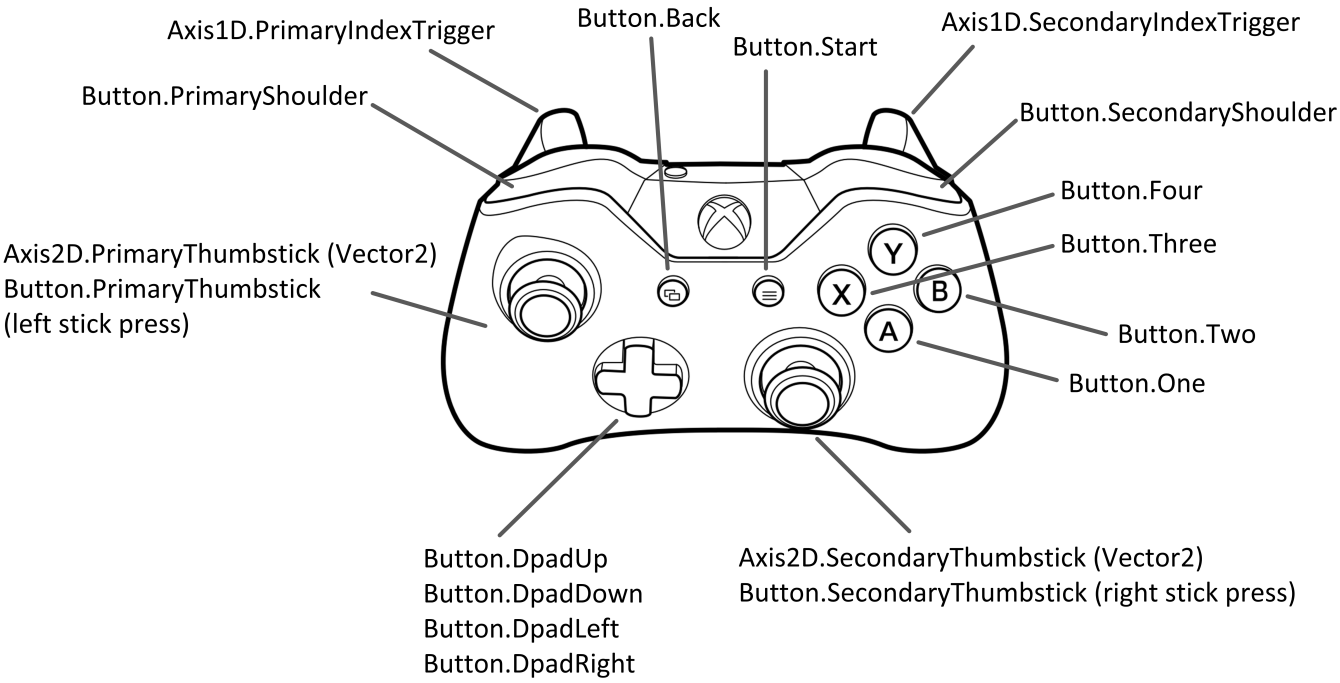
Raw Mapping



Xbox Input Handling

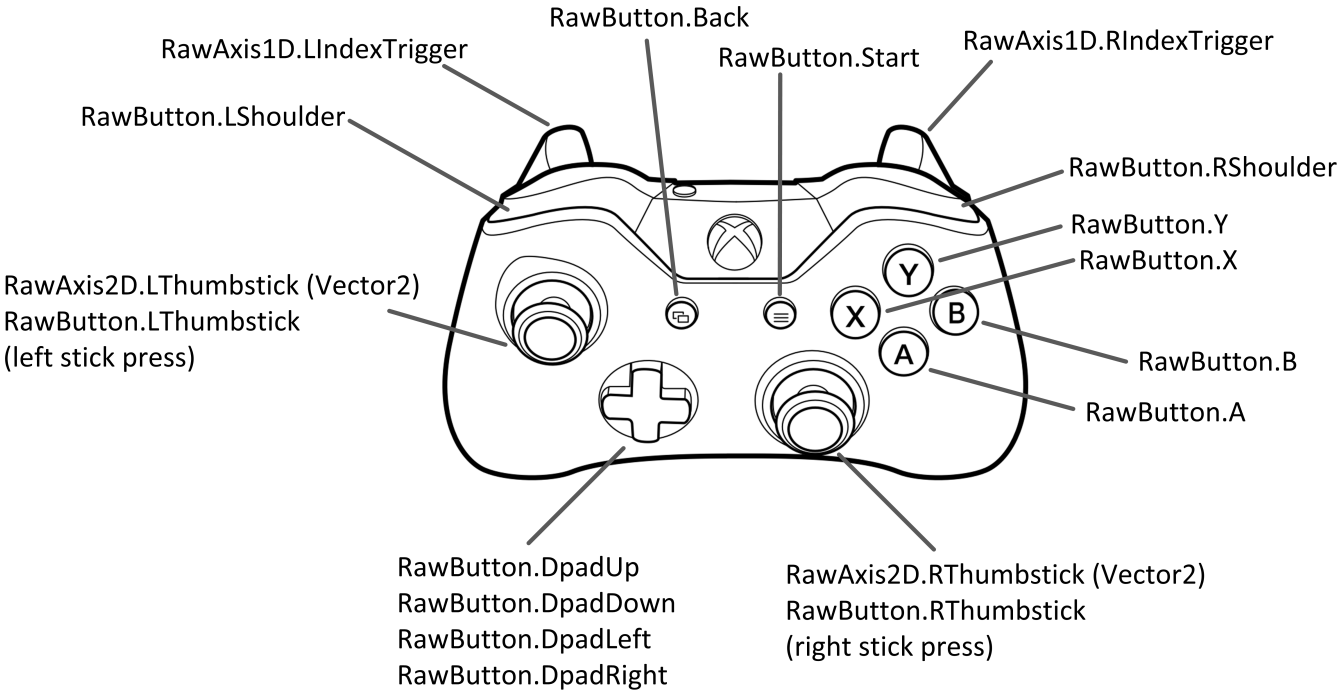
Virtual Mapping

This diagram shows a common implementation of Xbox controller input bindings using OVRInput.Controller.Gamepad.



Raw Mapping

The raw mapping directly exposes the Xbox controller.



Configuring for Build

This section describes building your project to PC and mobile targets.

PC Build Target: Microsoft Windows and Mac OS X

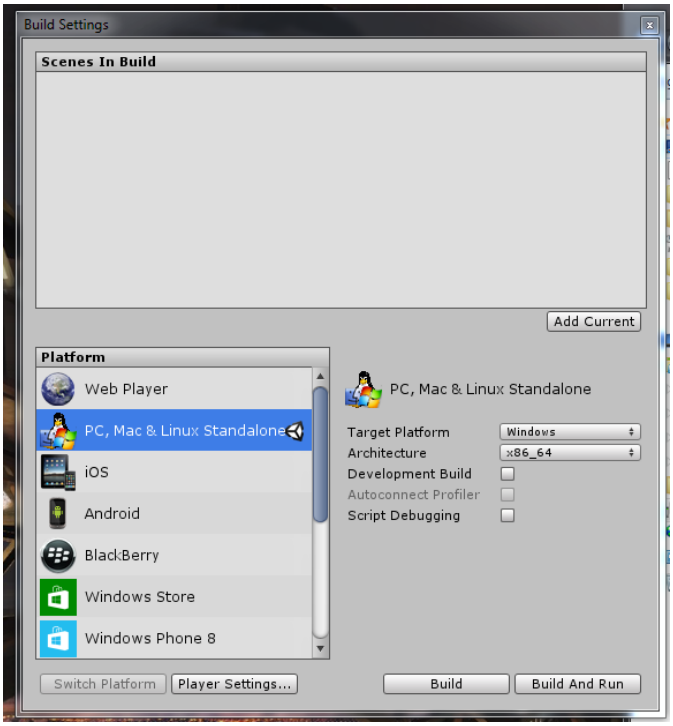
This section describes targeting Unity project builds to Microsoft Windows and Mac OS X.

Build Settings and Player Settings
To build the demo as a standalone full screen application, 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:

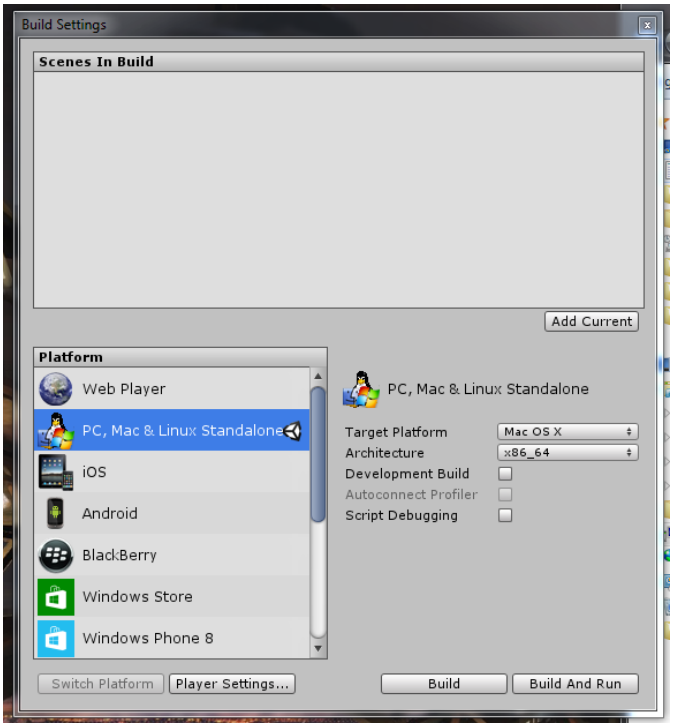
For Windows, set Target Platform to Windows and set Architecture to either x86 or x86 64.

Figure 11: Build Settings: PC



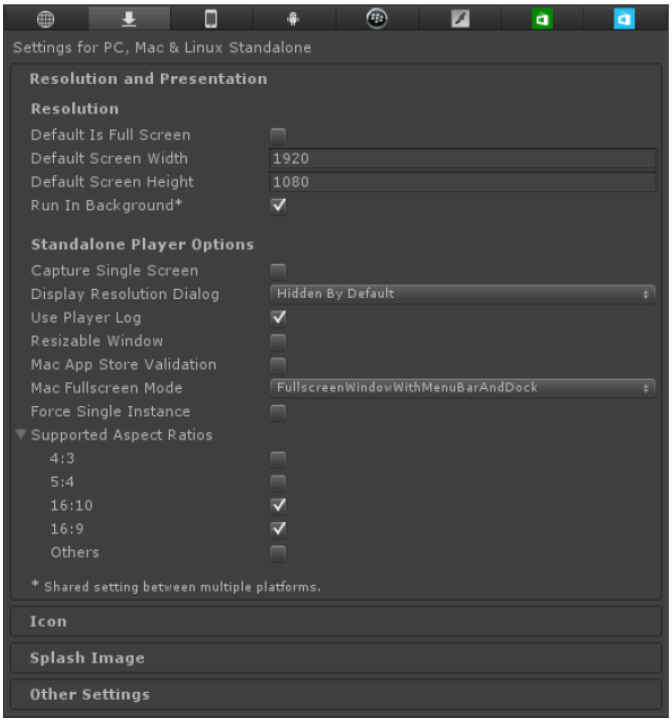
For Mac, set Target Platform to Mac OS X.

Figure 12: Build Settings: Mac



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

Figure 13: Player Settings



In the *Build Settings* pop-up, select *Build*. 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 full screen mode as a standalone application.

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 14: Quality settings for Oculus demo

The most important value to modify is *Anti aliasing* - it must be increased to compensate for the stereo rendering, which reduces the effective horizontal resolution by 50%. An anti-aliasing value of 2X is ideal - 4x may be used if you have performance to spare, and 8x usually isn't worth it.

Note: A quality setting called *Fastest* has been added to address a potential performance issue with Unity 4.5 and OS X 10.9. 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.

Running the Build

Now that the project is properly configured for VR, it's time to install and run the application.

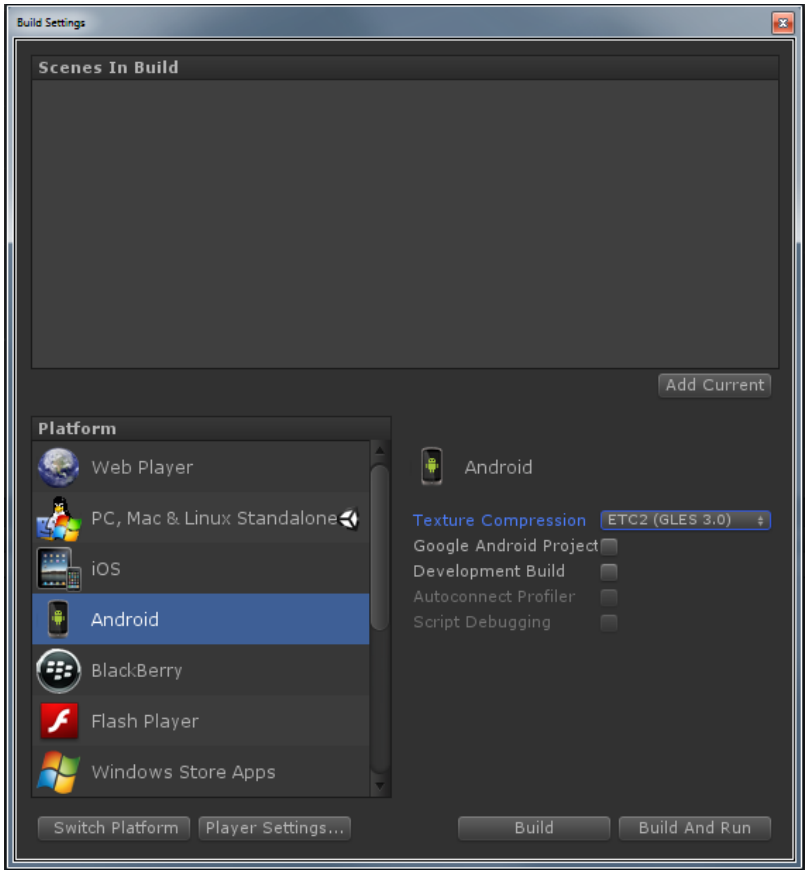
PC builds create a single executable file that may be used in either Direct Display or Extended Display modes.

Mobile Build Target: Android

This section describes targeting Unity project builds to Android.

Build Settings

1. From the *File* menu, select *Build Settings…*. From the *Build Settings…* menu, select *Android* as the platform. Set *Texture Compression* to *ETC2 (GLES 3.0)*.



2. Add any scenes you wish to include in your build to *Scenes In Build*..

Player Settings

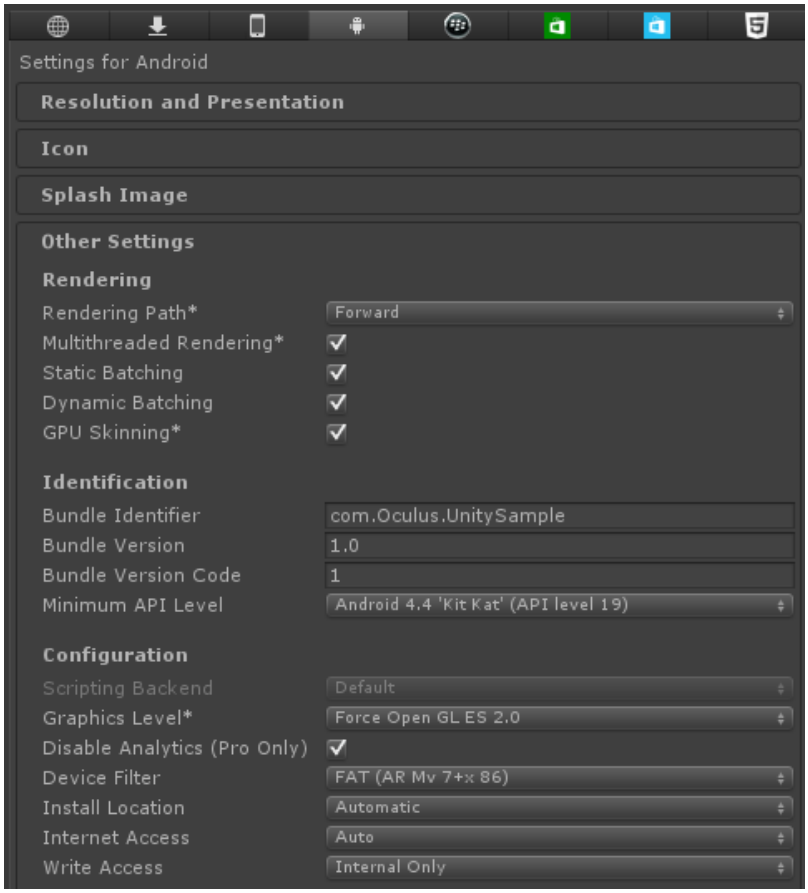
1. Click the *Player Settings…* button and select the Android tab. Set *Default Orientation* to *Landscape Left* in *Settings for Android* (may be collapsed).

   Note: The *Use 24-bit Depth Buffer* option appears to be ignored for Android. A 24-bit window depth buffer always appears to be created.

2. As a minor optimization, 16 bit buffers, color and/or depth may be used. Most VR scenes should be built to work with 16 bit depth buffer resolution and 2x MSAA. If your world is mostly pre-lit to compressed textures, there will be little difference between 16 and 32 bit color buffers.
3. Select the *Splash Image* section. For *Mobile Splash* image, choose a solid black texture.

   Note: Custom Splash Screen support is not available with Unity Free. A head-tracked Unity logo screen is provided instead.

4. While still in *Player Settings*, select *Other Settings* and verify that *Rendering Path** is set to *Forward*, *Multithreaded Rendering** is selected, and *Install Location* is set to *Force Internal*, as shown below:



5. Set the *Stripping Level* to the maximum level your app allows. It will reduce the size of the installed .apk file.

Note: This feature is not available for Unity Free.

Checking *Optimize Mesh Data* may improve rendering performance if there are unused components in your mesh data.

Quality Settings

1. Go to the *Edit* menu and choose *Project Settings*, then *Quality*. In the Inspector, set *Vsync Count* to *Don't Sync*. The TimeWarp rendering performed by the Oculus Mobile SDK already synchronizes with the display refresh.

Note: *Anti Aliasing* should **not** be enabled for the main framebuffer.

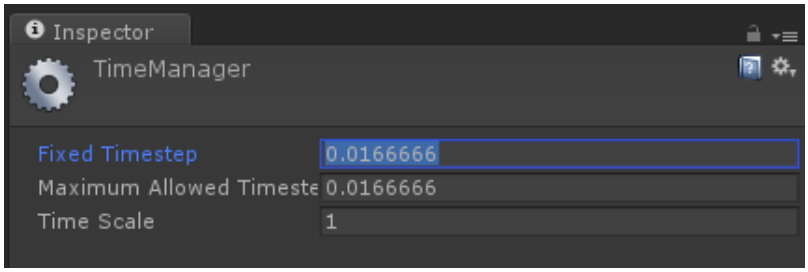
2. *Anti Aliasing* should be set to *Disabled*. You may change the camera render texture antiAliasing by modifying the `Eye Texture Antialiasing` parameter on OVRManager. The current default is 2x MSAA. Be mindful of the performance implications. 2x MSAA runs at full speed on chip, but may still increase the number of tiles for mobile GPUs which use variable bin sizes, so there is some performance cost. 4x MSAA runs at half speed, and is generally not fast enough unless the scene is very undemanding.
3. *Pixel Light Count* is another attribute which may significantly impact rendering performance. A model is re-rendered for each pixel light that affects it. For best performance, set *Pixel Light Count* to zero. In this case, vertex lighting will be used for all models depending on the shader(s) used.

Time Settings

Note: The following Time Settings advice is for applications which hold a solid 60 FPS, updating all game and/or application logic with each frame. The following Time Settings recommendations may be detrimental for apps that don't hold 60FPS.

Go to the *Edit -> Project Settings -> Time* and change both *Fixed Timestep* and *Maximum Allowed Timestep* to "0.0166666" (i.e., 60 frames per second).

Fixed Timestep is the frame-rate-independent interval at which the physics simulation calculations are performed. In script, it is the interval at which `FixedUpdate()` is called. The *Maximum Allowed Timestep* sets an upper bound on how long physics calculations may run.



Android Manifest File

Note: These manifest requirements are intended for development and differ from our submission requirements. Before submitting your application, please be sure to follow the manifest requirements described by our *Application Submission Guidelines*.

Open the AndroidManifest.xml file located under Assets/Plugins/Android/. You will need to configure your manifest with the necessary VR settings, as shown in the following manifest segment:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="<packagename>"
android:versionCode="1" android:versionName="1.0" android:installLocation="internalOnly">
<application android:theme="@android:style/Theme.Black.NoTitleBar.Fullscreen" >
<meta-data android:name="com.samsung.android.vr.application.mode" android:value="vr_only"/>
<activity  android:screenOrientation="landscape"
android:launchMode="singleTask"
android:configChanges="screenSize|orientation|keyboardHidden|keyboard">
</activity>
</application>
<uses-sdk android:minSdkVersion="19" android:targetSdkVersion="19" />
<uses-feature android:glEsVersion="0x00030000" />
```

- Replace `<packagename>` in the first line with your actual package name, such as "com.oculus.cinema".
- Unity will overwrite the required setting `android:installLocation="internalOnly"` if the Player Setting *Install Location* is not set to *Force Internal*.
- The Android theme should be set to the solid black theme for comfort during application transitioning: `Theme.Black.NoTitleBar.Fullscreen`
- The `vr_only` meta data tag should be added for VR mode detection.
- The required screen orientation is landscape: `android:screenOrientation="landscape"`
- We recommended setting your `configChanges` as follows: `android:configChanges="screenSize|orientation|keyboardHidden|keyboard"`
- The `minSdkVersion` and `targetSdkVersion` are set to the API level supported by the device. For the current set of devices, the API level is 19.
- **Do not** add the noHistory attribute to your manifest.

Running the Build

Now that the project is properly configured for VR, it's time to install and run the application on the Android device.

Applications written for development are not launched through the Oculus Home menu system. Instead, build the application directly to your phone, and you will be prompted to insert your phone into the Gear VR headset to launch the application automatically.

To run the application in the future, remove your phone from the Gear VR headset, launch the application directly from the phone desktop or Apps folder, and insert the device into the Gear VR when prompted to do so.

1. Copy an Oculus Signature File specific to your mobile device to the folder Project/Assets/Plugins/Android/assets/ or the application will not run. See the *Application Signing* section of the Mobile SDK documentation for more information.

2. Be sure the project settings from the steps above are saved with *File > Save Project*.
3. If you are not already connected to your phone via USB, connect now. Unlock the phone lock screen.
4. From the *File* menu, select *Build Settings…*. While in the *Build Settings* menu, add the Main.scene to *Scenes in Build*. Next, verify that Android is selected as your *Target Platform* and select *Build and Run*. If asked, specify a name and location for the .apk.

   The .apk will be installed and launched on your Android device.

Sample Unity Application Demos

This section describes the sample Unity applications provided by Oculus as a reference for development.

Running Pre-Built demos: PC

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 using Direct Display mode, you will be able to see the stereo image on your 2D display as well.

Running Pre-Built demos: Mobile

To run the pre-built demos, you must first install the demo packages (.apk) and sample media to your Android device.

Connect to the device via USB and open a command prompt. Run the installToPhone.bat script included with the SDK. This script will copy and install both the Unity and Native sample applications as well as any sample media to your Android device. You should now see application icons for the newly-installed apps on the Android Home screen.

For more information about these sample apps please review the Initial SDK Setup section in *Device and Environment Setup Guide*.

To test a sample application, perform the following steps:

- From the Android Home screen, press the icon of the VR app you wish to run.
- A toast notification will appear with a dialog like the following: “Insert Device: To open this application, insert your device into your Gear VR”
- Insert your device into the supported Gear VR hardware.

The app should now launch.

Pre-Built Demo Controls

BlockSplosion (mobile only)

In BlockSplosion, the camera position does not change, but the user's head orientation will be tracked, allowing them to aim before launching a block.

- 1-dot Button or Samsung gamepad tap launches a block in the facing direction.

- 2-dot Button resets the current level.
- Left Shoulder Button (L) skips to the next level.

Tuscany (PC only)

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.

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.

Shadowgun

In Shadowgun, locomotion allows the camera position to change.

- Left Analog Stick will move the player forward, back, left, and right.
- Right Analog Stick will rotate the player view left, right, up, and down. However, you will likely want to rotate your view just by looking with the VR headset.

Troubleshooting and Known Issues

This section outlines some currently known issues with the Oculus Unity Integration and the Oculus Utilities for Unity.

Rift

The app does not launch as a VR app.

Verify that you have installed the Oculus App and completed setup as described in *Preparing for Rift Development* on page 6.

Verify that you have selected *Virtual Reality Supported* in *Player Settings*.

are using a compatible runtime - see *Compatibility and Requirements* for more details.

Verify that the HMD is plugged in and working normally.

Verify that you have not selected D3D 9 or Windows GL as the renderer (Legacy Integration only).

Mobile

The app does not launch as a VR app.

Verify that you selected *Virtual Reality Supported* in *Player Settings* before building your APK.

Applications fail to launch on Gear VR with error message "thread priority security exception make sure the apk is signed".

You must sign your application with an Oculus Signature File (osig). See "Sign your App with an Oculus Signature File" in *Preparing for Mobile Development* on page 6 for instructions.

General Issues

Unity 5 hangs while importing assets from SDKExamples.

Unity 5 is known to import ETC2-compressed assets very slowly.

Receiving OVRPlugin console errors after importing a new version of Utilities.

Be sure to delete any previously-imported Utilities packages from your Unity project before importing a new version. If you are receiving errors and have not done so, delete the relevant folders in your project and re-import Utilities. For more information, please see *Importing the Oculus Utilities Package* on page 8.

Contact Information

Questions?

Visit our developer support forums at *https://developer.oculus.com*.

Our Support Center can be accessed at *https://support.oculus.com*.

Debugging and Performance Analysis in Unity

In this guide, we'll review baseline targets, recommendations, tools, resources, and common workflows for performance analysis and bug squashing for Unity VR applications.

General Tips

VR application debugging is a matter of getting insight into how the application is structured and executed, gathering data to evaluate actual performance, evaluating it against expectation, then methodically isolating and eliminating problems.

When analyzing or debugging, it is crucial to proceed in a controlled way so that you know specifically what change results in a different outcome. Focus on bottlenecks first. Only compare apples to apples, and change one thing at a time (e.g., resolution, hardware, quality, configuration).

Always be sure to profile, as systems are full of surprises. We recommend starting with simple code, and optimizing as you go - don't try to optimize too early.

We recommend creating a 2D, non-VR version of your camera rig so you can swap between VR and non-VR perspectives. This allows you to spot check your scenes, and it may be useful if you want to do profiling with third-party tools (e.g., Adreno Profiler).

It can be useful to disable *Multithreaded Rendering* in *Player Settings* during performance debugging. This will slow down the renderer, but also give you a clearer view of where your frame time is going. Be sure to turn it back on when you're done!

Performance Targets

Before debugging performance problems, establish clear targets to use as a baseline for calibrating your performance.

These targets can give you a sense of where to aim, and what to look at if you're not making frame rate or are having performance problems.

Below you will find some general guidelines for establishing your baselines, given as approximate ranges unless otherwise noted.

Mobile

- 60 FPS (required by Oculus)
- 50-100 draw calls per frame
- 50,000-100,000 triangles or vertices per frame

PC

- 90 FPS (required by Oculus)
- 500-1,000 draw calls per frame
- 1-2 million triangles or vertices per frame

For more information, see:

- *PC SDK Developer Guide*
- *Mobile VR Application Development*

Unity Profiling Tools

This section details tools provided by Unity to help you diagnose application problems and bottlenecks.

Unity Profiler

Unity comes with a built-in profiler (see Unity's *Profiler manual*). The Unity Profiler provides per-frame performance metrics, which can be used to help identify bottlenecks.

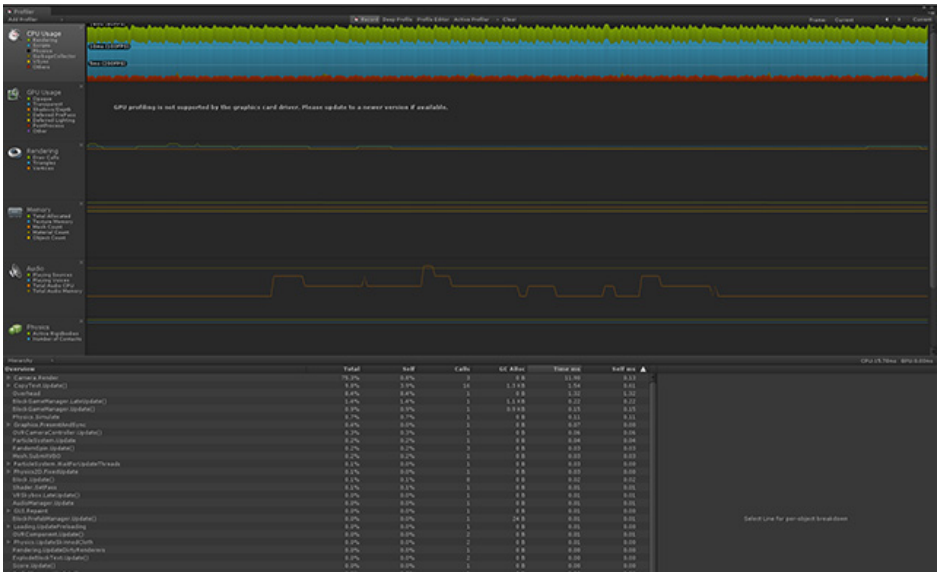
Unity Pro comes with a built-in profiler. The profiler provides per-frame performance metrics, which can be used to help identify bottlenecks.

PC Setup

To use Unity Profiler with a Rift application, select *Development Build* and *Autoconnect Profiler* in *Build Settings* and build your application. When you launch your application, the Profiler will automatically open.

Mobile Setup

You may profile your application as it is running on your Android device using adb or Wi-Fi. For steps on how to set up remote profiling for your device, please refer to the Android section of the following Unity documentation: *https://docs.unity3d.com/Documentation/Manual/Profiler.html*.



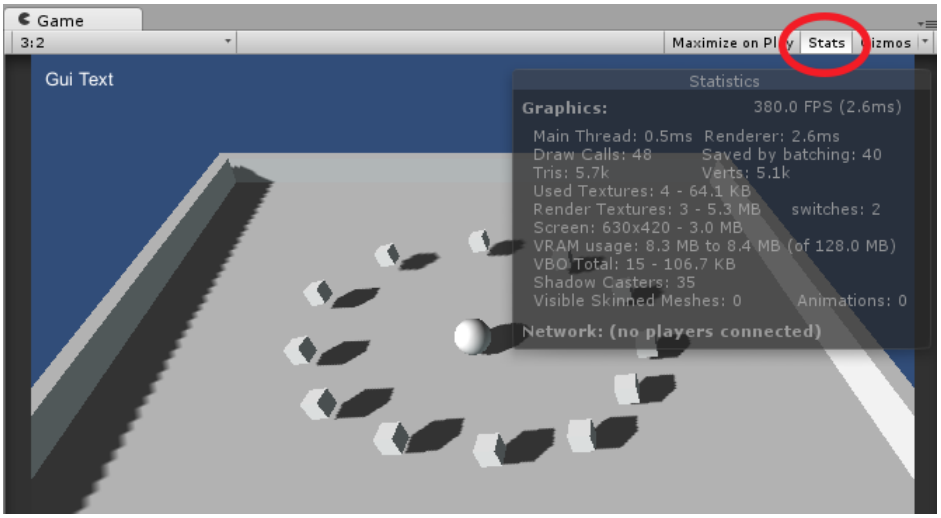
The Unity Profiler displays CPU utilization for the following categories: Rendering, Scripts, Physics, GarbageCollector, and Vsync. It also provides detailed information regarding Rendering Statistics, Memory Usage (including a breakdown of per-object type memory usage), Audio and Physics Simulation statistics.

GPU Usage data for Android is not available at this time.

The Unity profiler only displays performance metrics for your application. If your app isn't performing as expected, you may need to gather information on what the entire system is doing.

Show Rendering Statistics

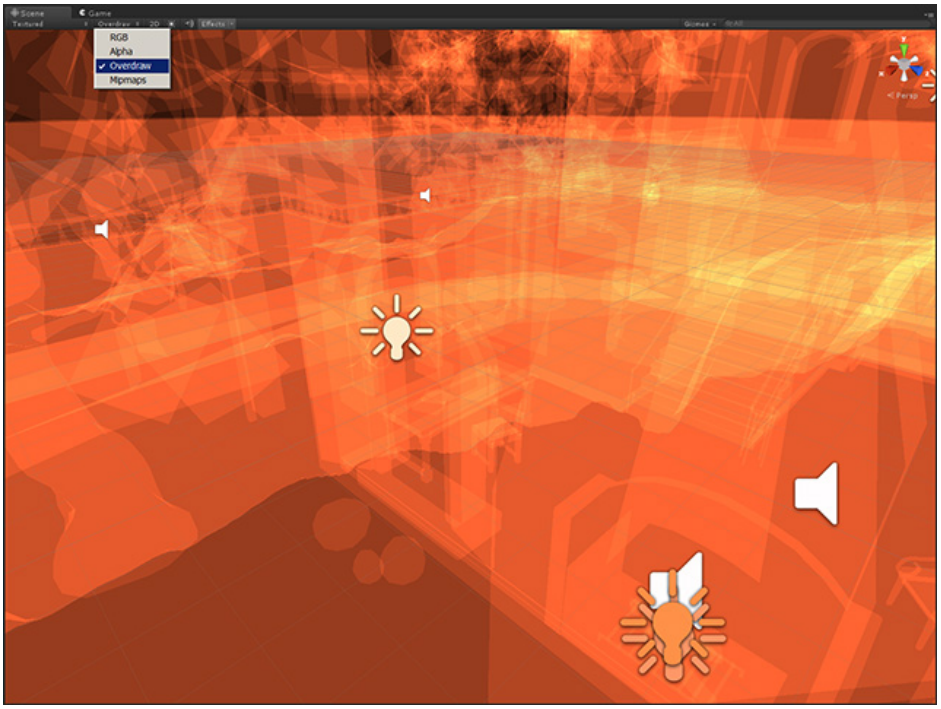
Unity provides an option to display real-time rendering statistics, such as FPS, Draw Calls, Tri and Vert Counts, VRAM usage. While in the Game View, pressing the Stats button above the Game View will display an overlay showing realtime render statistics.Viewing stats in the Editor can help analyze and improve batching for your scene by indicating how many draw calls are being issued and how many are being saved by batching (the OverDraw render mode is helpful for this as well).



Show GPU Overdraw

Unity provides a specific render mode for viewing overdraw in a scene. From the Scene View Control Bar, select OverDraw in the drop-down Render Mode selection box.

In this mode, translucent colors will accumulate providing an overdraw "heat map" where more saturated colors represent areas with the most overdraw.



Unity Built-in Profiler

Unity Built-in Profiler (not to be confused with Unity Profiler) provides frame rate statistics through logcat, including the number of draw calls, min/max frametime, number of tris and verts, et cetera.

To use this profiler, connect to your device over Wi-Fi using ADB over TCPIP as described in the *Wireless usage* section of Android's adb documentation. Then run `adb logcat` while the device is docked in the headset.

See *Unity's Measuring Performance with the Built-in Profiler* for more information. For more on using adb and logcat, see *Android Debugging* in the Mobile SDK documentation.

Oculus Performance Heads-Up Display (HUD) for the Rift

Oculus Performance Head-Up Display (HUD)

The Oculus Performance Head-Up Display (HUD) is an important, easy-to-use tool for viewing timings for render, latency, and performance headroom in real-time as you run an application in the Oculus Rift. The HUD is easily accessible through the Oculus Debug Tool provided with the PC SDK. For more details, see the *Performance Head-Up Display* and *Oculus Debug Tool* sections of the Oculus Rift Developers Guide.

Compositor Mirror

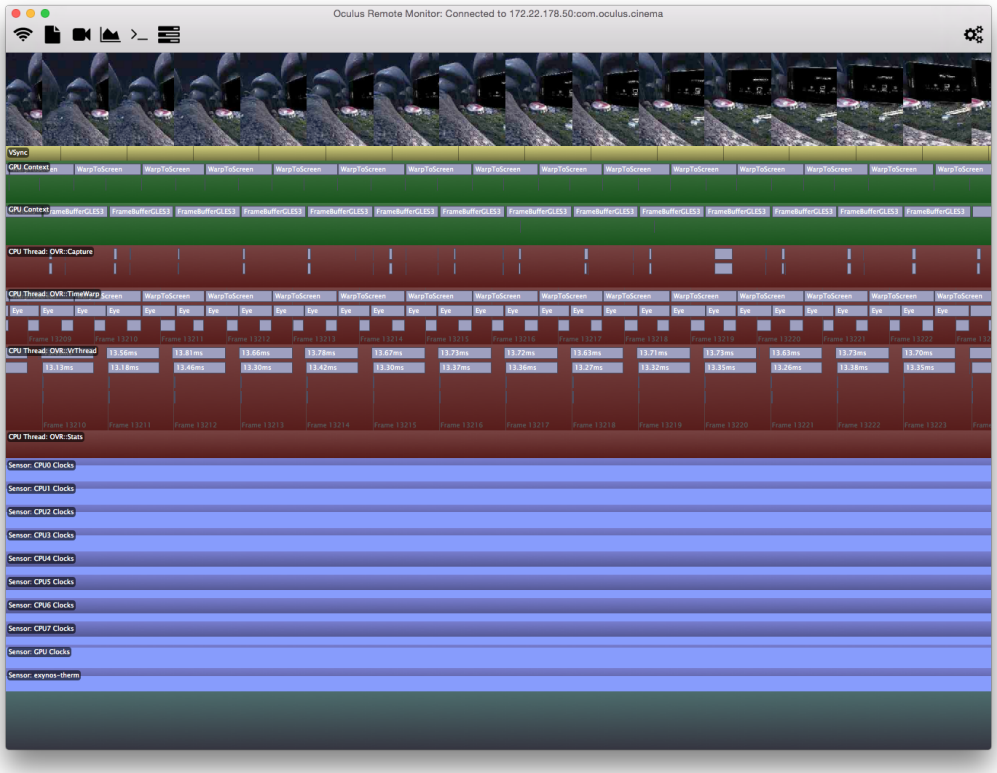
The compositor mirror is an experimental tool for viewing exactly what appears in the headset, with Asynchronous TimeWarp and distortion applied.

The compositor mirror is useful for development and troubleshooting without having to wear the headset. Everything that appears in the headset will appear, including Oculus Home, Guardian boundaries, in-game notifications, and transition fades. The compositor mirror is compatible with any game or experience, regardless of whether it was developed using the native PC SDK or a game engine.

For more details, see the *Compositor Mirror* section of the PC SDK Guide.

Oculus Remote Monitor for Gear VR

Oculus Remote Monitor is a client for Windows and Mac OS X that connects to VR applications running on remote devices to capture, store, and display the streamed-in data. It provides visibility into Android VR and GLES activity, and includes low-res rendered image snapshots for a visual reference to its timeline-based display. Remote Monitor is available for download from our Downloads page.



The Remote Monitor client uses VrCapture, a low-overhead remote monitoring library. VrCapture is designed to help debug behavior and performance issues in mobile VR applications. VrCapture is included automatically in any project built with Unity 5, or compiled with the Legacy Integration.

For more information on setup, configuration, and usage, please see *VrCapture and Oculus Remote Monitor*.

Additional Third-Party Tools

ETW + GPUView

Event Tracing for Windows (ETW) is a trace utility provided by Windows for performance analysis. *GPUView* view provides a window into both GPU and CPU performance with DirectX applications. It is precise, has low overhead, and covers the whole Windows system. Custom event manifests.

ETW profiles the whole system, not just the GPU. For a sample debug workflow using ETW to investigate queuing and system-level contention, see Example Workflow: PC below.

Windows 10 replaces ETW with *Tracelogging*.

Systrace

Reports complete Android system utilization. Available here: *http://developer.android.com/tools/help/systrace.html*

NVIDIA NSight

NSight is a CPU/GPU debug tool for NVIDIA users, available in a *Visual Studio version* and an *Eclipse version*.

Mac OpenGL Monitor

An OpenGL debugging and optimizing tool for OS X. Available here: *https://developer.apple.com/library/mac/technotes/tn2178/_index.html#//apple_ref/doc/uid/DTS40007990*

APITrace

https://apitrace.github.io/

Analyzing Slowdown

In this guide, we take a look at three of the areas commonly involved with slow application performance: pixel fill, draw call overhead, and slow script execution.

Pixel Fill

Pixel fill is a function of overdraw and of fragment shader complexity. Unity shaders are often implemented as multiple passes (draw diffuse part, draw specular part, and so forth). This can cause the same pixel to be touched multiple times. Transparency does this as well. Your goal is to touch almost all pixels on the screen only one time per frame.

Unity's Frame Debugger (described in *Unity Profiling Tools* on page 63) is very useful for getting a sense of how your scene is drawn. Watch out for large sections of the screen that are drawn and then covered, or for objects that are drawn multiple times (e.g., because they are touched by multiple lights).

Z-testing is faster than drawing a pixel. Unity does culling and opaque sorting via bounding box. Therefore, large background objects (like your Skybox or ground plane) may end up being drawn first (because the bounding box is large) and filling a lot of pixels that will not be visible. If you see this happen, you can move those objects to the end of the queue manually. See *Material.renderQueue* in Unity's Scripting API Reference for more information.

Frame Debugger will clearly show you shadows, offscreen render targets, et cetera.

Draw Calls

Modern PC hardware can push a lot of draw calls at 90 fps, but the overhead of each call is still high enough that you should try to reduce them. On mobile, draw call optimization is your primary scene optimization.

Draw call optimization is usually about batching multiple meshes together into a single VBO with the same material. This is key in Unity because the state change related to selecting a new VBO is relatively slow. If you select a single VBO and then draw different meshes out of it with multiple draw calls, only the first draw call is slow.

Unity batches well when given properly formatted source data. Generally:

- Batching is only possible for objects that share the same material pointer.
- Batching doesn't work on objects that have multiple materials.
- Implicit state changes (e.g. lightmap index) can cause batching to end early.

Here is a quick checklist for maximizing batching:

- Use as few textures in the scene as possible. Fewer textures require fewer unique materials, so they are easier to batch. Use texture atlases.
- Bake lightmaps at the largest atlas size possible. Fewer lightmaps require fewer material state changes. Gear VR can push 4096 lightmaps without too much trouble, but watch your memory footprint.

- Be careful not to accidentally instance materials. Note that accessing `Renderer.material` automatically creates an instance (!) and opts that object of batching. Use `Renderer.sharedMaterial` instead whenever possible.
- Watch out for multi-pass shaders. Add `noforwardadd` to your shaders whenever you can to prevent more than one directional from applying. Multiple directionals generally break batching.
- Mark all mesh that never moves as `Static` in the editor. Note that this will cause the mesh to be combined into a mega mesh at build time, which can increase load time and app size on disk, though usually not in a material way. You can also create a static batch at runtime (e.g., after generating a procedural level out of static parts) using StaticBatchingUtility.
- Watch your static and dynamic batch count vs the total draw call count using the Profiler, internal profiler log, or stats gizmo.

Script Performance

Unity's C# implementation is fast, and slowdown from script is usually the result of a mistake and/or an inadvertent block on slow external operations such as memory allocation. The Unity Profiler can help you find and fix these scripts.

Try to avoid foreach, lamda, and LINQ structures as these allocate memory needlessly at runtime. Use a for loop instead. Also, be wary of loops that concatenate strings.

Game Object creation and destruction takes time. If you have a lot of objects to create and destroy (say, several hundred in a frame), we recommend pooling them.

Don't move colliders unless they have a rigidbody on them. Creating a rigidbody and setting `isKinematic` will stop physics from doing anything but will make that collider cheap to move. This is because Unity maintains two collider structures, a static tree and a dynamic tree, and the static tree has to be completely rebuilt every time any static object moves.

Note that coroutines execute in the main thread, and you can have multiple instances of the same coroutine running on the same script.

We recommend targeting around 1-2 ms maximum for all Mono execution time.

PC Debug Workflow

In this guide, we'll use the example of a hypothetical stuttering app scene and walk through basic steps debugging steps.

Where to Start

Begin by running the scene with the *Oculus Performance HUD*.

If the scene drops more than one frame every five seconds, check the render time. If it's more than 8 ms, have a look at GPU utilization. Otherwise, look at optimizing CPU utilization. If observed latency is greater than 30 ms, have a look at queuing.

CPU Profiling (Unity Profiler)

Look for the tallest bars in the CPU Usage graph in the Unity Profiler. Sort hierarchy by Total CPU time, and expand to see which objects and calls take the most time.

If you find garbage collection spikes, don't allocate memory each frame.

GPU Profiling (Unity Profiler)

Are your rendering stats too high? (For reference baselines, see *Performance Targets*.

Check for hogs in your hierarchy or timeline view, such as any single object that takes 8 ms to render. The GPU may also wait for long stalls on CPU. Other potential problem areas are mesh rendering, shadows, vsync, and subsystems.

Best Practices: Mobile

This section provides simple guidelines to help your Unity app perform well with Samsung Gear VR.

Good performance is critical for all VR applications, but the limitations inherent to mobile development warrant special consideration.

We recommend that you also review *Design Guidelines* and *Mobile VR Design and Performance Guidelines* in the *Mobile SDK documentation*.

Design Considerations

Startup Sequence

For good VR experiences, all graphics should be rendered such that the user is always viewing a proper three-dimensional stereoscopic image. Additionally, head-tracking must be maintained at all times.

An example of how to do this during application startup is demonstrated in the SDKExamples Startup_Sample scene:

- Solid black splash image is shown for the minimum time possible.
- A small test scene with 3D logo and 3D rotating widget or progress meter is immediately loaded.
- While the small startup scene is active, the main scene is loaded in the background.
- Once the main scene is fully loaded, the start scene transitions to the main scene using a fade.

Universal Menu Handling

Applications must handle the Back Key long-press action which launches the Universal Menu as well as the Back Key short-press action which launches the "Confirm-Quit to Home" Menu and exits the current application, returning to the Oculus Home application.

An example of demonstrating this functionality is in the SDKExamples GlobalMenu_Sample scene. For more information about application menu options and access, see *Universal Menu* in the Mobile SDK documentation.

Best Practices

- Be **Batch Friendly**. Share materials and use a texture atlas when possible.
- Prefer **lightmapped, static geometry**.
- **Prefer lightprobes instead of dynamic lighting** for characters and moving objects.
- **Bake as much detail into the textures as possible.** E.g., specular reflections, ambient occlusion.
- **Only render one view per eye.** No shadow buffers, reflections, multi-camera setups, et cetera.
- **Keep the number of rendering passes to a minimum.** No dynamic lighting, no post effects, don't resolve buffers, don't use grabpass in a shader, et cetera.
- **Avoid alpha tested / pixel discard transparency.** Alpha-testing incurs a high performance overhead. Replace with alpha-blended if possible.
- Keep **alpha blended transparency** to a minimum.
- **Use Texture Compression.** Favor ETC2.
- **MSAA may be enabled on the Eye Render Textures.**

General CPU Optimizations

To create a VR application or game that performs well, careful consideration must be given to how features are implemented. It must run at 60 FPS. Avoid any hitching or laggy performance during any point that the player is in your game.

Recommendations

- Be mindful of the total number of GameObjects and components your scenes use.
- Model your game data and objects efficiently. You will generally have plenty of memory.
- Minimize the number of objects that actually perform calculations in `Update()` or `FixedUpdate()`.
- Reduce or eliminate physics simulations when they are not actually needed.
- Use object pools to respawn frequently-used effects or objects instead of allocating new ones at runtime.
- Use pooled AudioSources versus PlayOneShot sounds, as the latter allocate a GameObject and destroy it when the sound is done playing.
- Avoid expensive mathematical operations whenever possible.
- Cache frequently-used components and transforms to avoid lookups each frame.
- Use the Unity Profiler to:
    - Identify expensive code and optimize as needed.
    - Identify and eliminate Garbage Collection allocations that occur each frame.
    - Identify and eliminate any spikes in performance during normal play.
- Do not use Unity's `OnGUI()` calls.
- Do not enable gyro or the accelerometer. In current versions of Unity, these features trigger calls to expensive display calls.
- All best practices for mobile app and game development generally apply.

Rendering Optimization

Be conservative on performance from the start.

- Keep draw calls down.
- Be mindful of texture usage and bandwidth.
- Keep geometric complexity to a minimum.
- Be mindful of fillrate.

Reducing Draw Calls

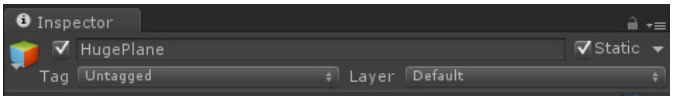
Keep the total number of draw calls to a minimum. A conservative target would be less than **100 draw calls per frame**.

Unity provides several built-in features to help reduce draw calls such as batching and culling.

Draw Call Batching

Unity attempts to combine objects at runtime and draw them in a single draw call. This helps reduce overhead on the CPU. There are two types of draw call batching: Static and Dynamic.

Static batching is used for objects that will not move, rotate or scale, and must be set explicitly per object. To mark an object static, select the Static checkbox in the object Inspector.



Note: Static batching is not available for Unity Free.

Dynamic batching is used for moving objects and is applied automatically when objects meet certain criteria, such as sharing the same material, not using real-time shadows, or not using multipass shaders. More information on dynamic batching criteria may be found here: *https://docs.unity3d.com/Documentation/Manual/DrawCallBatching.html*

Culling

Unity offers the ability to set manual per-layer culling distances on the camera via Per-Layer Cull Distance. This may be useful for culling small objects that do not contribute to the scene when viewed from a given distance. More information about how to set up culling distances may be found here: *https://docs.unity3d.com/Documentation/ScriptReference/Camera-layerCullDistances.html*.

Unity also has an integrated Occlusion Culling system. The advice to early VR titles is to favor modest "scenes" instead of "open worlds," and Occlusion Culling may be overkill for modest scenes. More information about the Occlusion Culling system can be found here: *http://blogs.unity3d.com/2013/12/02/occlusion-culling-in-unity-integration-4-3-the-basics/*.

Reducing Memory Bandwidth

- **Texture Compression**: Texture compression offers a significant performance benefit. Favor ETC2 compressed texture formats.
- **Texture Mipmaps**: Always use mipmaps for in-game textures. Fortunately, Unity automatically generates mipmaps for textures on import. To see the available mipmapping options, switch *Texture Type* to *Advanced* in the texture inspector.
- **Texture Filtering**: Trilinear filtering is often a good idea for VR. It does have a performance cost, but it is worth it. Anisotropic filtering may be used as well, but keep it to a single anisotropic texture lookup per fragment.
- **Texture Sizes**: Favor texture detail over geometric detail, e.g., use high-resolution textures over more triangles. We have a lot of texture memory, and it is pretty much free from a performance standpoint. That said, textures from the Asset Store often come at resolutions which are wasteful for mobile. You can often reduce the size of these textures with no appreciable difference.
- **Framebuffer Format**: Most scenes should be built to work with a 16 bit depth buffer resolution. Additionally, if your world is mostly pre-lit to compressed textures, a 16 bit color buffer may be used.
- **Screen Resolution**: Setting Screen.Resolution to a lower resolution may provide a sizeable speedup for most Unity apps.

Reduce Geometric Complexity

Keep geometric complexity to a minimum. 50,000 static triangles per-eye per-view is a conservative target.

Verify model vert counts are mobile-friendly. Typically, assets from the Asset Store are high-fidelity and will need tuning for mobile.

Unity Pro provides a built-in **Level of Detail** System (not available in Unity Free), allowing lower-resolution meshes to be displayed when an object is viewed from a certain distance. For more information on how to set up a LODGroup for a model, see the following: *https://docs.unity3d.com/Documentation/Manual/LevelOfDetail.html*

Verify your vertex shaders are mobile friendly. And, when using built-in shaders, favor the Mobile or Unlit version of the shader.

Bake as much detail into the textures as possible to reduce the computation per vertex, for example, baked bumpmapping as demonstrated in the Shadowgun project: *https://docs.unity3d.com/430/Documentation/Manual/iphone-PracticalRenderingOptimizations.html*

Be mindful of GameObject counts when constructing your scenes. The more GameObjects and Renderers in the scene, the more memory consumed and the longer it will take Unity to cull and render your scene.

Reduce Pixel Complexity and Overdraw

Pixel Complexity: Reduce per-pixel calculations by baking as much detail into the textures as possible. For example, bake specular highlights into the texture to avoid having to compute the highlight in the fragment shader.

Verify your fragment shaders are mobile friendly. And, when using built-in shaders, favor the Mobile or Unlit version of the shader.

Overdraw: Objects in the Unity opaque queue are rendered in front to back order using depth-testing to minimize overdraw. However, objects in the transparent queue are rendered in a back to front order without depth testing and are subject to overdraw.

Avoid overlapping alpha-blended geometry (e.g., dense particle effects) and full-screen post processing effects.

Tutorial: Build a Simple VR Unity Game

This section describes the steps necessary to build, load, and run a simple Unity 3D application on the Oculus Rift or Samsung Gear VR.

It is intended to serve as a basic introduction for developers who are new to VR development and to Unity. Once the necessary tools are set up, this process should take a few hours to complete. By the end, you will have a working mobile application that you can play and demonstrate on your Oculus Rift or Gear VR device, to the amazement of your friends and loved ones.

We will build and modify the Unity game Roll-a-ball to add VR capability. The game is controllable by keyboard or by the Samsung EI-GP20 gamepad.

Requirements

- Oculus Rift or Gear VR with compatible Samsung phone
- Samsung EI-GP20 gamepad (required for Mobile; optional for Desktop)
- PC running Windows 7, 8 or 10, or a Mac running OS X 10
- Unity 4 (for version compatibility, see *Compatibility and Requirements*)

You will also need to refer to the relevant Oculus SDK documentation, available for download here: *https://developer.oculus.com/documentation/*

Installation and Preparation

1. Install the appropriate Oculus SDK and prepare for development.

   Desktop: Download and install the Oculus PC SDK and Unity Integration from *Oculus PC SDK Downloads*. Prepare for development as described in the *Oculus Rift Getting Started Guide*. By the time you have completed this process, you should be able to run the Demo Scene as described in that guide.

Mobile: Download and install the Oculus Mobile SDK from *Oculus Mobile SDK Downloads*. Prepare for development as described by the *Device and Environment Setup Guide*. By the time you have completed this process, you should be able to communicate with your Samsung phone via USB. To verify this, retrieve the device ID from your phone by connecting via USB and sending the command `adb devices` from a command prompt. If you are communicating successfully, the phone will return its device ID. You may wish to make a note of it - you will need it later to request a Oculus Signature File (see step four in *Modify Roll-a-ball for VR* for more information).

2. Install Unity.

   Check which version of the Unity editor you should download an install in our *Compatibility and Version Requirements* on page 5, then download the appropriate version here: *http://docs.unity3d.com/Manual/index.html*. Unity provides extensive documentation to introduce users to the environment. You may wish to begin by reviewing their documentation to gain a basic familiarity with core concepts such as the Editor, GameObjects, prefabs, projects, and scenes.

3. Build the Unity Roll-a-ball application.

   Unity provides a number of video tutorials that walk you through the process of creating a simple game. The first in the series provides instructions for creating the Roll-a-ball application, in which you use the keyboard or gamepad to control a ball that rolls around a game board and picks up floating token counters:*http://unity3d.com/learn/tutorials/projects/roll-a-ball*

   The development process is covered in eight short video tutorials which run from around five to fifteen minutes in length. Allow for a few hours to complete the procedure.

   The final video in the series, "107. Publishing the game," describes building the Roll-a-ball game for play in a web browser. You may skip this lesson if you wish for the purposes of this exercise, as we will follow a different procedure for building a playable application (PC/Mac) or APK (Android).

   Note: We refer to the assets, folders, and so forth by the names used in the Unity tutorial, so it is helpful to follow the names they use in their example.

4. Duplicate your Roll-a-ball project (optional).

   Once you have completed building Roll-a-ball, you may wish to create a duplicate Roll-a-ball project specifically for VR development. It can be useful to retain a backup of the original unmodified Roll-a-ball project in case you make mistakes or wish to work with it later without the VR assets.

   To duplicate the Roll-a-ball project, simply navigate in your OS to the Unity project folder containing your Roll-a-ball project, copy the folder and all of its contents, and rename it. For this tutorial, we will use the project folder name *Roll-a-ball-VR*.

5. Launch the new project and prepare the game scene.

   1. Launch Unity and select *File* > *Open Project...* and select the project folder location for Roll-a-ball-VR in order to launch the project.
   2. In your *Project* tab, open *Assets* > *_Scenes* and select "MiniGame."
   3. Press F2 and rename the scene "VRMiniGame."
   4. Open the scene "VRMiniGame."

Modify Roll-a-ball for VR

1. Import the Oculus Unity Integration Package.

   In Unity, select *Assets* > *Import Package* > *Custom Package...*. Navigate to the folder where you have installed the Oculus SDK.

   PC SDK: open the *OculusUnityIntegration* folder, select OculusUnityIntegration.unitypackage, and click *Open*.

Mobile SDK: open the *UnityIntegration* folder in *VrUnity*, select UnityIntegration.unityPackage, and click *Open*.

This will open Unity's *Import Package* dialog box, which lists the assets included in the package. Leave all boxes checked and select *Import*.

For more information on the contents of the integration package, see *A Detailed Look at the Unity Integration*.

2. Replace Main Camera with OVRCameraRig.

We will replace the Roll-a-ball Main Camera with OVRCameraRig, an Oculus prefab VR camera included with our Unity Integration. OVRCameraRig renders two stereoscopic images of the game scene with the appropriate distortion.

Main Camera tracks the player ball, but we will modify our camera to overlook the game board from a fixed position, so the player may look around at whatever angle they choose.

Rather than deleting the Main Camera, simply deselect it to make it inactive in the scene. Select *Main Camera* in your Hierarchy view and uncheck the check box at the top left of the Inspector view.

Note: Only one camera may be active in a Unity scene at any given time.

In the Project view, open the *OVR* folder and select the *Prefabs* folder. Select *OVRCameraRig* and drag it into the Hierarchy view to instantiate it.

3. Elevate OVRCameraRig above the game board.

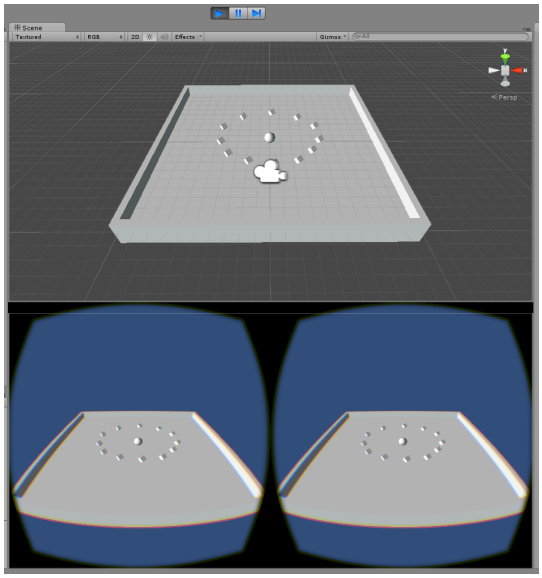
Select *OVRCameraRig* in the Hierarchy view and set the *Position* fields of the OVRCameraRig Transform to the following values: X = 0; Y = 10; Z = -15.

4. Rotate OVRCameraRig forward for a better view.

Set the *Rotation* field of the *OVRCameraRig Transform* to the following value: X = 35; Y = 0; Z = 0.

Enter Play mode by pressing the play button, and note that the Game view now shows the image rendered in two distorted and stereoscopic views as illustrated below. If you are using the PC SDK, you will see the *Health and Safety Warning* appear over the game; press any key to continue past it.

Figure 15: Roll-a-ball VR in Unity Scene and Game Views



5. Save your scene and project before building your application.
6. Sign your application (Mobile Only).

   To access your Samsung phone's VR capabilities, you will need to sign your application with an Oculus Signature File (osig). If you recorded your device ID earlier, you may use it now to request your osig file. Note that you need only one osig per mobile device.

   You may obtain an osig from our self-service portal here: *https://dashboard.oculus.com/tools/osig-generator/*. Once you have received an osig, copy it to your Unity project folder in /Roll-a-ball-VR/Assets/Plugins/Android/assets/.

   More information may be found on application signing in *Application Signing* in our Mobile guide.

Build and Play

Build and Launch

If you are developing for desktop, you will build an executable file that you may launch with your PC or Mac. If you are developing for mobile, you will build an APK and load it onto your phone, and then insert the phone into the Gear VR to launch your game. Follow the build instructions as described by the section *Configuring for Build* section.

Play

Go ahead and try it out! You may use your keyboard or a paired Samsung gamepad to control your ball and collect the game pickup objects.

Note: Because the GUIText display we built in the Roll-a-ball tutorial will not work with OVRCameraRig without substantial modification, you will not see the score counter or "You win!" message when all the pieces have been collected.

Launching the Game on Gear VR

If you select *Apps* from the Samsung home screen, you will see Roll-a-ball-VR listed with the other apps. You may launch the application directly, and when instructed, put the phone in the Gear VR. It will not be visible from Oculus Home.

Getting Started FAQ

If you're new to Oculus development with Unity, this FAQ can answer the most common beginner questions.

Question: What's the best way to get started if you're a beginner?

Answer: Browse through this FAQ, and check out our *Getting Started Guide* on page 5. Read through Unity's excellent *documentation* and try out some of their introductory *tutorials* to get acquainted with Unity development.

When you're ready to get into the trenches, find out what the version of Unity 5 we recommend at our *Compatibility and Requirements* page, then download and install it. Next, build your own simple VR game by following the instructions in our *Tutorial: Build Your First VR App* on page 18.

You can also browse around on our *Unity Developer Forum*.

Question: What are the system requirements for Unity development for Oculus? What operating systems are supported for Unity development?

Answer: For the most up-to-date information, see *Unity Compatibility and Requirements*. We currently support Windows and OS X for development. The Oculus Rift requires Windows 7, 8 or 10.

Question: What version of Unity should I use?

Answer: Our latest version recommendations may be found in our *Unity Compatibility and Requirements* document. Be sure to check back regularly, as we update it frequently as new SDKs and Unity versions are released. You can find an archive of information in our *Unity-SDK Version Compatibility* list.

Question: What other tools and resources do you provide to Unity developers?

Answer: To find the latest tools we provide, check out our *Other Oculus Resources for Unity Developers* on page 24.

Question: What do I need to run Rift applications that I build with Unity?

Answer: You will need a compatible Windows PC, a Rift, and the Oculus software. For more details, see *Preparing for Rift Development* on page 6

Question: I want to focus on mobile development for the Samsung Gear VR. What do I need to do to get started? Do I need to download the Oculus Mobile SDK?

Answer: The Android SDK is required for mobile development with Unity. However, most Unity developers do not need to download the Oculus Mobile SDK, or to install Android Studio or NDK. For more details, see *Preparing for Mobile Development* on page 6.

Question: Can I develop a single application for both Samsung Gear VR and the Oculus Rift?

Answer: Yes, but when developing for both Rift and mobile platforms, keep in mind that the requirements for PC and mobile VR applications differ substantially. If you would like to generate builds for both PC and mobile from a single project, it is important to follow the more stringent mobile development best practices, as well as meeting the required 90 fps required by the Rift.

Question: What is the difference between the Oculus Unity 4 Legacy Integration and the Oculus Utilities for Unity 5? How do the differences between Unity 4 and 5 affect development?

Answer: All developers should use Unity 5 and the optional Oculus Utilities package. The Unity 4 Integration is maintained for legacy purposes only.

In Unity 4, you must import our Legacy Integration for Unity 4 unitypackage and add the supplied VR camera and player controller to your application to add Rift or Gear VR support.

Question: Where can I ask questions or get help?

Answer: Visit our developer support forums at *https://developer.oculus.com*. Our Support Center can be accessed at *https://support.oculus.com*.

Unity-SDK Version Compatibility

This reference describes the relationship between Unity versions, Oculus PC and Mobile SDKs, and Oculus Unity Integration and Utilities packages.

Utilities for Unity 5.x Versions

We are currently supporting the 5.4, 5.5, and 5.6 branches of the Unity editor, which are all under active development. Our Utilities for Unity 5 package no longer supports Unity versions prior to 5.3.4p5.

Release Date	Utilities	PC SDK	Mobile SDK	Unity
3/30/2017	1.13.0	1.12.0	1.0.4.5	5.6.0f3 or later
3/30/2017	1.13.0	1.12.0	1.0.4.5	5.5.2p3 or later
3/30/2017	1.13.0	1.12.0	1.0.4.5	5.4.5f1 or later
3/10/2017	1.12.0	1.12.0	1.0.4.5	5.5.2p3 or later
3/10/2017	1.12.0	1.12.0	1.0.4.5	5.4.5f1 or later
3/10/2017	1.12.0	1.12.0	1.0.4.5	5.3.8f1 or later (deprecated)
2/1/2017	1.11.0	1.11.0	1.0.4.5	5.5.1p2 or later
2/8/2017	1.11.0	1.11.0	1.0.4.5	5.4.4p3 or later
2/1/2017	1.11.0	1.11.0	1.0.4.5	5.3.7p4 or later (deprecated)
11/28/2016	1.10.0	1.10.0	1.0.4	5.5.0p1 or later
11/28/2016	1.10.0	1.10.0	1.0.4	5.4.3p3 or later
11/28/2016	1.10.0	1.10.0	1.0.4	5.3.7p2 or later
11/1/2016	1.9.0	1.9.0	1.0.4	5.4.2p2 or later
11/1/2016	1.9.0	1.9.0	1.0.4	5.3.6p8 or later
9/15/2016	1.8.0	1.8.0	1.0.3	5.4.1p1 or later
9/15/2016	1.8.0	1.8.0	1.0.3	5.3.6p5 or later
8/18/2016	1.7.0	1.7.0	1.0.3	5.4.0p3 or later
8/18/2016	1.7.0	1.7.0	1.0.3	5.3.6p3 or later
7/28/2016	1.6.0	1.5.0	1.0.3	5.4.0b16 or later
7/28/2016	1.6.0	1.6.0	1.0.3	5.3.6p1 or later
6/30/2016	1.5.0	1.5.0	1.0.3	5.4.0b16 and later

Release Date	Utilities	PC SDK	Mobile SDK	Unity
6/30/2016	1.5.0	1.5.0	1.0.3	5.3.4p5 and later
4/20/2016	1.3.2	1.3.2	1.0.0.1	5.4.0b11 and later
4/20/2016	1.3.2	1.3.2	1.0.0.1	5.3.3p3 and later
3/28/2016	1.3.0	1.3.0	1.0.0.1	5.3.4p1
10/30/2015	0.1.3.0 Beta	0.8.0.0	0.6.2.0	5.2.2p2
10/1/2015	0.1.2.0 Beta	0.7.0.0	0.6.2.0	5.2.1p2
7/6/2015	0.1.0.0 Beta	0.6.0.1	0.5.0	5.1.1

Unity 4.x Integration Versions

Release Date	Integration	PC SDK	Mobile SDK	Unity
3/28/2016	1.3.0	1.3.0	1.0.0.1	4.7.0f1
10/30/2015	0.8.0.0	0.8.0.0	1.0.0	4.6.9p1
9/08/2015	0.6.2.0	0.7.0.0	0.6.2.0	4.6.7+
8/14/2015	0.6.1.0	0.6.0.1	0.6.1.0	4.6.7+
8/7/2015	0.6.0.2	0.6.0.1	0.6.0.1	4.6.7
6/25/2015	D 0.6.0.1	0.5.0.1	0.6.0.1	4.6
6/12/2015	M 0.6.0.1	0.5.0.1	0.6.0.1	4.6
5/15/2015	D 0.6.0.0	0.5.0.1	0.5.0	4.6
3/31/2015	M 0.5.0	0.5.0.1	0.5.0	4.6
3/26/2015	D 0.5.0.1	0.5.0.1	0.5.0	4.6
3/20/2015	M 0.4.3.1	0.4.4	0.4.3.1	4.5
2/27/2015	M 0.4.3	0.4.4	0.4.3	4.5
1/23/2015	M 0.4.2	0.3.2	0.4.2	4.5
1/7/2015	M 0.4.1	0.3.2	0.4.1	4.4
12/4/2014	D 0.4.4	0.4.4	0.4.0	4.6
11/12/2014	M 0.4.0	0.3.2	0.4.0	4.4
11/10/2014	D 0.4.3.1	0.4.3.1	N/A	4.5
10/24/2014	D 0.4.3	0.4.3	N/A	4.5
9/4/2014	D 0.4.2	0.4.2	N/A	4.5
8/11/2014	D 0.4.1	0.4.1	N/A	4.4
7/24/2014	D 0.4.0	0.4.0	N/A	4.4
5/22/2014	D 0.3.2	0.3.2	N/A	4.3
4/14/2014	D 0.3.1	0.3.1	N/A	4.3
10/10/2013	D 0.2.5	0.2.5	N/A	4.3

Release Notes

This section describes changes for each version release.

1.3 Unity 4 Legacy Integration Release Notes

Unity 4.x Legacy Integration 1.3.0

This document provides an overview of new features, improvements, and fixes included in the latest version of the Unity 4.x Legacy Integration. For information on first-party changes to Unity VR support for Oculus, see the Unity Release Notes for the appropriate version. You will find an updated scripting reference for the included C# scripts in our *Unity Developer Reference*.

Note: For detailed information about Unity version compatibility, please see *Compatibility and Requirements*.

This public release incorporates all changes from private releases 0.9.0 through 1.3.0. It adds support for PC SDK 1.3, including support for Rift consumer version hardware.

Users may now close applications when switching to Home. When they do this, Unity receives a call from the runtime to close the application. `OnApplicationQuit()` callback behaviors still work as before. Keep this in mind if you have previously written a custom shutdown procedure.

VR Focus

When VR Focus is lost in Unity 4 applications, all cameras are disabled and `Time.timeScale` is set to `0`. Applications should check `Time.timeScale` and pause gameplay and audio if it is 0.

Note that Rift and Samsung Gear VR applications will pause rendering when not in focus in VR. Recentering requests initiated from the Universal Menu are now implemented and will be handled automatically by the Unity Integration.

New Features

- Added support for PC SDK 1.3, including support for Rift consumer version hardware.
- Added support for Asynchronous TimeWarp and Phase Sync.
- Added Rift Remote controller support.
- Added application lifecycle management, including VR-initiated backgrounding and exit.
- Exposed proximity sensor.
- Added support for multiple trackers.
- Exposed velocity and acceleration for all tracked nodes.
- Added VR Focus support. When VR Focus is lost, `Time.timeScale` is set to `0`.
- Added support for recentering from within the Home Universal Menu.
- Audio output now automatically uses the Rift headphones, if enabled in the Oculus app.
- Rift's inter-axial distance slider now affects the distance between Unity's eye poses.
- Exposed controller visibility so apps can respond to occlusion, et cetera.

API Changes

- `OVRManager.isUserPresent` is true when the proximity sensor detects the user.
- `OVRInput.GetControllerLocal[Angular]Velocity/Accelerations` exposes the linear and angular velocity and rotation of each Touch controller.
- Added `OVRInput.GetControllerPosition|OrientationTracked(..)` for controller visibility.

Bug Fixes

- Removed redundant axial deadzone handling from Xbox gamepad input.
- Increased deadzone on OVRInput stick input to prevent drift.
- Fixed issue where frustum/occlusion culling was less effective on Gear VR.
- Fixed several errors and race conditions due to hazards in start-up ordering.

Known Issues

- The Rift's microphone is not the default when calling `Microphone.Start(null,..)`. Please find the entry in `Microphone.devices` that contains the word `Rift` and use it.
- When you stop playing in the Unity editor, the last frame from the previous play remains frozen on the Rift.
- If an application's simulation time is paused, OVRInput will not update.
- Volume control will be missing on mobile applications until the release of Mobile SDK 1.0.2. OVRVolumeControl has been removed, but it may still be accessed by using previous versions of the Oculus Unity 4 Integration.
- If a VR app is running while a Rift is unplugged from a PC and plugged back in without restarting the application, then the Rift display will appear black. To fix this, add the following lines to OVRPlugin.cs and disable your OVRCameraRig and OVRManager if `OVRPlugin.shouldRecreateDistortionWindow` is `true`. You will have to quit and re-start the app before it can use VR again.

```
private enum Status
{
Debug = 0,
…
ShouldRecenter,
+ ShouldRecreateDistortionWindow,
}
…
public static bool shouldRecenter { get { return GetStatus(Status.ShouldRecenter); } }

+ public static bool shouldRecreateDistortionWindow { get { return
 GetStatus(Status.ShouldRecreateDistortionWindow); }
```

0.8 Unity Legacy Integration Release Notes

This document provides an overview of new features, improvements, and fixes included in the Oculus Unity Integration version 0.8.

Unity 4.x Legacy Integration 0.8.0.0

Overview of Major Changes

This document provides an overview of new features, improvements, and fixes included in the latest version of the Unity 4.x Legacy Integration. For information on first-party changes to Unity VR support for Oculus, see the Unity Release Notes for the appropriate version.

Unity 4.x Legacy Integration 0.8.0.0 extends OVRInput support to mobile. OVRInputControl and OVRGamepadController are now deprecated and will be removed in a future release.

Mobile input bindings are now automatically added to InputManager.asset if they do not already exist - it is no longer required to replace InputManager.asset with Oculus' version. However, this asset is still provided for now to maintain legacy support for the deprecated OVRGamepadController and OVRInputControl scripts.

Mobile Unity Development

Mobile SDK 1.0.0 includes major changes to VrApi - please see the *Mobile SDK 1.0.0 Release Notes* for more information.

New Features

- Default mobile input bindings are now programmatically generated when projects are imported if they do not already exist.
- Replacing InputManager.asset is no longer required to enable gamepad support on mobile.

API Changes

- Added mobile support OVRInput.
- Deprecated OVRInputControl and OVRGamepadController.

Bug Fixes

- Fixed mobile gamepad thumbstick deadzone/drift handling and axis scaling.
- Fixed mobile gamepad support when multiple gamepads are paired.
- Fixed mobile gamepad bindings for triggers, D-pad, thumbstick presses, etc.

Known Issues

- Some PC configurations may encounter instability when targeting the Windows platform with this release of the Unity 4.x Legacy integration. Please report any issues on our *Developer Forums*.

0.6 Unity Legacy Integration Release Notes

This document provides an overview of new features, improvements, and fixes included in the Oculus Unity Integration version 0.6.

Unity 4.x Legacy Integration 0.6.2.0

Overview of Major Changes

0.6.2.0 pulls in the 0.7.0 Desktop plugins.

The source for the Unity SDKExample MediaSurface Plugin is now provided. The Media Surface Plugin provides a native library which is used with the Android MediaPlayer for hardware decoding of a single video to a texture. Note that the SDKExample MediaSurface Plugin is not intended to be production quality, and is provided for reference purposes. We have made the source available in case you would like to modify it for your use. The source and build files are located at the following SDK path: VrAppSupport/MediaSurfacePlugin.

This version adds an alpha release of OVRInput, which provides a unified input API for accessing Oculus Touch and Microsoft Xbox controllers.

New Features

- Added alpha OVRInput script to expose API for Oculus Touch and XInput-based controllers.
- Now dynamically loads OpenGL ES symbols instead of hard-linking to libGLESv3.so. (Mobile)

API Changes

- Now requires the new Android Plugin Java library OculusUtilities.jar (found in Assets/Plugins/Android).

Bug Fixes

- Fixed Editor error message in Unity Free due to unsigned OVRPlugin.dll.
- Fixed thread affinity failing to be set for the TimeWarp and DeviceManager threads. (Mobile)

- Fixed device reset when repeatedly plugging and unplugging the Travel Adapter on the Galaxy SAMSUNG S6. (Mobile)
- Fixed app crash occurring when using Oculus Runtime for OS X earlier than 0.5.0.1.

Unity 4.x Legacy Integration 0.6.1.0

Overview of Major Changes

Legacy Integration 0.6.1.0 expands upon the changes in Unity 4.x PC Integration 0.6.0.2. It is fully compatible with both our Mobile and PC SDKs.

For more information on important changes and updates relevant to this version, see the Release Notes for Integration 0.6.0.2 below.

PC Developers

This release of the Unity Legacy Integration is compatible with the Oculus Runtime 0.6 and 0.7. **All Unity projects built with Unity 4 must update to this release**, or they will not work on PCs using Runtime 0.7.

OS X Developers

Mac developers using Legacy Integration 0.6.1.0 must update to the Oculus Runtime for OS X 0.5.0.1. Before updating your runtime, **be sure** to run uninstall.app (found in /Applications/Oculus/) to remove your previous installation.

Mobile Developers

The MediaSurface functionality has been split from the main Unity Integration and provided as a separate plugin, libOculusMediaSurface.so included with the Unity integration package.

New Features

- SDKExamples
  - MoviePlayer_Sample now looks for media files in the folder Streaming Assets instead of following a hard-coded path on the internal SD card (default mp4 provided).
  - MoviePlayer_Sample defaults to using Unity MovieTexture functionality on the PC (media file expected as Ogg Theora).

API Changes

- No longer required to issue ResetVrModeParms to dynamically adjust CPU and GPU clock levels.

Bug Fixes

- Added back one frame of latency to Unity apps to avoid object motion judder by giving Unity apps one frame of additional GPU time (0.5.1 performance parity).

Unity 4.x PC Integration 0.6.0.2

Overview of Major Changes

Note: This release is intended for PC development only. Mobile developers should wait for Legacy Integration version 0.6.1.0, which will ship very soon.

This release of the Unity Legacy Integration is compatible with the Oculus Runtime 0.6 and 0.7. **All Unity projects built with Unity 4 must update to this release**, or they will not work on PCs using Runtime 0.7.

In most cases, migrating to this version of the legacy integration will be easy - simply open your project in Unity, import the custom Integration unityPackage, and rebuild.

If you have previously used our C API wrappers in OvrCapi.cs, note that they have been removed and you will need to write your own native plugins or replacement wrappers to access functionality from LibOVRRT. This change is also required to use CAPI with our Unity 5 Utilities, so writing your own plugins or wrappers now will help prepare for your migration to Unity 5.

A single executable file is now produced for PC builds, which will work in both Extended and Direct Display modes.

This version removes support for D3D 9 and Windows GL.

Mac developers using Legacy Integration 0.6.0.2 must update to the Oculus Runtime for OS X 0.5.0.1. Before updating your runtime, **be sure** to run uninstall.app (found in /Applications/Oculus/) to remove your previous installation.

New Features

- Compatible with Oculus Runtime versions 0.6 and 0.7.
- PC builds now produce a single executable file for use in both Extended and Direct Display modes.
- Oculus runtime log messages now visible in the Unity console.
- Unity Editor Game View displays undistorted monoscopic preview.
- No longer necessary to rotate Mac monitor for DK2 in System Preferences->Displays.

API Changes

- Removed all C API wrappers from OvrCapi.cs.
- Removed D3D9, Windows GL, and Linux support.

Bug Fixes

- Fixed C# script problem that caused D3D11 Exclusive Mode errors.
- Fixed several Editor Game View issues:
    - Now works for all aspect ratio settings.
    - Fixed bug that blacked out Game Views when more than one were created.
    - Fixed Game View positioning bug.

Known Issues

- Mac tearing: Editor Game View and standalone players do not vsync properly, resulting in a vertical tear and/or judder on DK2. Android players are unaffected, even if built on a Mac.
- No VR support for in-editor preview on Windows until Unity 4.6.7p4.
- Slight increase in latency for trivial scenes (i.e., scenes that take < 3 ms to render).

Mobile Unity Integration 0.6.0.1

New Features

- Allow Unity MediaSurface dimensions to be modified via plugin interface.

Bug Fixes

- Fixed not being able to enable chromatic aberration correction in the Unity plugin.

- Fixed adjusting clock levels from Unity during load.

Mobile Unity Integration 0.6.0

Overview of Major Changes

- Oculus Runtime is no longer required for mobile development.
- Synced with the Oculus PC SDK 0.6.0.0 beta.
- Allows clients to re-map plugin event IDs.

For both the PC and Mobile SDKs we recommend the following Unity versions or higher: Unity Pro 4.6.3, Unity Professional 5.0.2.p2, Unity Free 4.6, or Unity Personal 5.0.2.p2. For mobile development, compatibility issues are known to exist with Unity 5 and OpenGL ES 3.0 – please check back for updates. Earlier versions of Unity 5 should not be used with the Mobile SDK.

New Features

- VrAPI: improved frame prediction, in particular for Unity.

Bug Fixes

- Fixed prediction glitch every 64 frames.
- Use correct prediction for OVR_GetCameraPositionOrientation.
- Fixed location of the PlatformMenu Gaze Cursor Timer

PC Unity Integration 0.6.0.0

New Features

- Disabled eye texture anti-aliasing when using deferred rendering. This fixes the black screen issue.
- Eliminated the need for the DirectToRift.exe in Unity 4.6.3p2 and later.
- Removed the hard dependency from the Oculus runtime. Apps now render in mono without tracking when VR isn't present.

0.5 Mobile Unity Integration Release Notes

This document provides an overview of new features, improvements, and fixes included in the Oculus Unity Integration that shipped with version 0.5 of the Oculus Mobile SDK.

Mobile Unity Integration 0.5.1

Overview of Major Changes

The most significant change in 0.5.1 is to System Activities event handling in Unity. The 0.5.0 code for handling System Activities events in Unity was doing heap allocations each frame. Though this was not a leak, it would cause garbage collection to trigger much more frequently. Even in simple applications, garbage collection routinely takes 1 to 2 milliseconds. In applications that were already close to dropping below 60 Hz, the increased garbage collection frequency could cause notable performance drops. The event handling now uses a single buffer allocated at start up.

As with Mobile SDK v 0.5.0, Unity developers using this SDK version must install the Oculus Runtime for Windows or OS X. This requirement will be addressed in a future release of the SDK.

Bug Fixes

- Rework System Activities Event handling to prevent any per-frame allocations that could trigger Garbage Collector.

Known Issues

- For use with the Mobile SDK, we recommend Unity versions 4.6.3. The Mobile SDK is compatible with Unity 5.0.1p2, which addresses a problem with OpenGL ES 3.0, but there is still a known Android ION memory leak. Please check back for updates.

Mobile Unity Integration 0.5.0

Overview of Major Changes

The Mobile Unity Integration is now synced with the Oculus PC SDK 0.5.0.1 Beta. Please ensure you have installed the corresponding 0.5.0.1 Oculus runtime; it can be found at the following location: *https://developer.oculus.com/downloads/*

VrPlatform entitlement checking is now disabled by default in Unity; handling for native development is unchanged. If your application requires this feature, please refer to the Mobile SDK Documentation for information on how to enable entitlement checking.

New Features

w

- Synced with the Oculus PC SDK 0.5.0.1 Beta.
- VrPlatform entitlement checking is now disabled by default.

Bug Fixes

- Health and Safety Warning no longer displays in editor Play Mode if a DK2 is not attached.

Known Issues

- For use with the Mobile SDK, we recommend Unity versions 4.6.3, which includes Android 5.0 - Lollipop support as well as important Android bug fixes. While the Mobile SDK is compatible with Unity 5.0.0p2 and higher, several issues are still known to exist, including an Android ION memory leak and compatibility issues with OpenGL ES 3.0. Please check back for updates.

Mobile Unity Integration 0.5.0

Overview of Major Changes

The Mobile Unity Integration is now synced with the Oculus PC SDK 0.5.0.1 Beta. Please ensure you have installed the corresponding 0.5.0.1 Oculus runtime; it can be found at the following location: *https://developer.oculus.com/downloads/*

VrPlatform entitlement checking is now disabled by default in Unity; handling for native development is unchanged. If your application requires this feature, please refer to the Mobile SDK Documentation for information on how to enable entitlement checking.

New Features

- Synced with the Oculus PC SDK 0.5.0.1 Beta.
- VrPlatform entitlement checking is now disabled by default.

Bug Fixes

- Health and Safety Warning no longer displays in editor Play Mode if a DK2 is not attached.

Known Issues

- For use with the Mobile SDK, we recommend Unity versions 4.6.3, which includes Android 5.0 - Lollipop support as well as important Android bug fixes. While the Mobile SDK is compatible with Unity 5.0.0p2 and higher, several issues are still known to exist, including an Android ION memory leak and compatibility issues with OpenGL ES 3.0. Please check back for updates.

0.4 Mobile Unity Integration Release Notes

This document provides an overview of new features, improvements, and fixes included in the Oculus Unity Integration that shipped with version 0.4 of the Oculus Mobile SDK.

Mobile Unity Integration 0.4.3

New Features

- New Mobile Unity Integration Based on Oculus PC SDK 0.4.4

Mobile Unity Integration 0.4.2

Overview of Major Changes

If you are developing with Unity, we recommend updating to Unity 4.6.1, which contains Android 5.0 – Lollipop support.

We would like to highlight the inclusion of the new Mobile Unity Integration with full DK2 support based on the Oculus PC SDK 0.4.4. As this is a significant API refactor, please refer to the Unity Development Guide: Migrating From Earlier Versions section for information on how to upgrade projects built with previous versions of the Mobile Unity Integration.

API Changes

- Fix for camera height discrepancies between the Editor and Gear VR device.
- Moonlight Debug Util class names now prefixed with OVR to prevent namespace pollution.
- Provide callback for configuring VR Mode Parms on OVRCameraController; see OVRModeParms.cs for an example.

Mobile Unity Integration 0.4.1

Overview of Major Changes

Added support for Android 5.0 (Lollipop) and Unity Free.

New Features

- Added Unity Free support for Gear VR developers.

Mobile Unity Integration 0.4.0

Overview of Major Changes

First public release of the Oculus Mobile SDK.

Bug Fixes

- Unity vignette rendering updated to match native (slightly increases effective FOV).
- Unity volume pop-up distance to match native.

Migrating From Earlier Versions

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

API Changes

The following are changes to Unity components:

Table 8: Unity Components

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

The following are changes to helper classes:

Table 9: Helper Classes

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

The following are changes to events:

Table 10: 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.

Behavior Changes

- OVRCameraRig's position is always the initial center eye position.
- Eye anchor Transforms are tracked in OVRCameraRig's local space.
- OVRPlayerController's position is always at the user's feet.
- IPD and FOV are fully determined by profile (PC only).
- Layered rendering: multiple OVRCameraRigs are fully supported (not advised for mobile).
- OVRCameraRig.*EyeAnchor Transforms give the relevant poses.

Upgrade Procedure

To upgrade, follow these steps:

1. Ensure you didn't modify the structure of the OVRCameraController prefab. If your eye cameras are on Game Objects named "CameraLeft" and "CameraRight" which are children of the OVRCameraController Game Object (the default), then the prefab should cleanly upgrade to OVRCameraRig and continue to work properly with the new integration.
2. Write down or take a screenshot of 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
    - OVR Internal folder (if applicable)
    - Any file in the Plugins folder with "Oculus" or "OVR" in the name
    - Android-specific assets in the Plugins/Android folder, including: vrlib.jar, libOculusPlugin.so, res/raw and res/values folders
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 described above. Note that the substitution of prefabs does not take place until after all script compile errors have been fixed.
9. Re-apply your previous settings to OVRCameraRig, OVRPlayerController, and OVRManager. Note that the runtime camera positions have been adjusted to better match the camera positions set in the Unity editor. If this is undesired, you can get back to the previous positions by adding a small offset to your camera:
    a. **Adjust the camera's y-position.**
        a. If you previously used an OVRCameraController without an OVRPlayerController, add 0.15 to the camera y-position.
        b. If you previously used an OVRPlayerController with *Use Player Eye Height* checked on its OVRCameraContoller, then you have two options. You may either (1) rely on the new default player eye-height (which has changed from 1.85 to 1.675); or (2) uncheck *Use Profile Data* on the converted OVRPlayerController and then manually set the height of the OVRCameraRig to 1.85 by setting its y-position. Note that if you decide to go with (1), then this height should be expected to change when profile customization is added with a later release.
        c. If you previously used an OVRPlayerController with *Use Player Eye Height* unchecked on its OVRCameraContoller, then be sure uncheck *Use Profile Data* on your converted OVRPlayerController. Then, add 0.15 to the y-position of the converted OVRCameraController.
    b. **Adjust the camera's x/z-position.** If you previously used an OVRCameraController without an OVRPlayerController, add 0.09 to the camera z-position relative to its y rotation (i.e. +0.09 to z if it has 0 y-rotation, -0.09 to z if it has 180 y-rotation, +0.09 to x if it has 90 y-rotation, -0.09 to x if it has 270 y-rotation). If you previously used an OVRPlayerController, no action is needed.
10. Re-start Unity

Common Script Conversions

```
OVRCameraController -> OVRCameraRig
   cameraController.GetCameraPosition() -> cameraRig.rightEyeAnchor.position
   cameraController.GetCameraOrientation() -> cameraRig.rightEyeAnchor.rotation
   cameraController.NearClipPlane -> cameraRig.rightEyeCamera.nearClipPlane
   cameraController.FarClipPlane -> cameraRig.rightEyeCamera.farClipPlane
   cameraController.GetCamera() -> cameraRig.rightEyeCamera

   ---------------------------------------------------------------------
   if ( cameraController.GetCameraForward( ref cameraForward ) &&
   cameraController.GetCameraPosition( ref cameraPosition ) )
   {
```

```
...

to

if (OVRManager.display.isPresent)
{

     // get the camera forward vector and position
     Vector3 cameraPosition = cameraController.centerEyeAnchor.position;
     Vector3 cameraForward = cameraController.centerEyeAnchor.forward;
...
-------------------------------------------------------------------------

OVRDevice.ResetOrientation();
to
OVRManager.display.RecenterPose();

-------------------------------------------------------------------------

cameraController.ReturnToLauncher();

to

OVRManager.instance.ReturnToLauncher();

-------------------------------------------------------------------------

OVRDevice.GetBatteryTemperature();
OVRDevice.GetBatteryLevel();

to

OVRManager.batteryTemperature
OVRManager.batteryLevel

-------------------------------------------------------------------------

OrientationOffset

Set rotation on the TrackingSpace game object instead.

-------------------------------------------------------------------------

FollowOrientation

-------------------------------------------------------------------------

FollowOrientation is no longer necessary since OVRCameraRig applies tracking
in local space. You are free to script the rig's pose or make it a child of
another Game Object.
```