Unreal

Version 1.8.0

Copyrights and Trademarks

Contents

Unreal Introduction

Unreal provides built-in support for Oculus Rift and Gear VR development on Windows.

Unreal apps run on the Oculus platform automatically apply stereoscopic rendering to the main camera as well as positional and orientation tracking for the Rift, or orientation tracking for Gear VR.

For Epic's documentation about Oculus development, go to *https://docs.unrealengine.com/latest/INT/Platforms/VR/*.

Oculus SDK

Epic provides three versions of the Unreal Engine which include Oculus support:

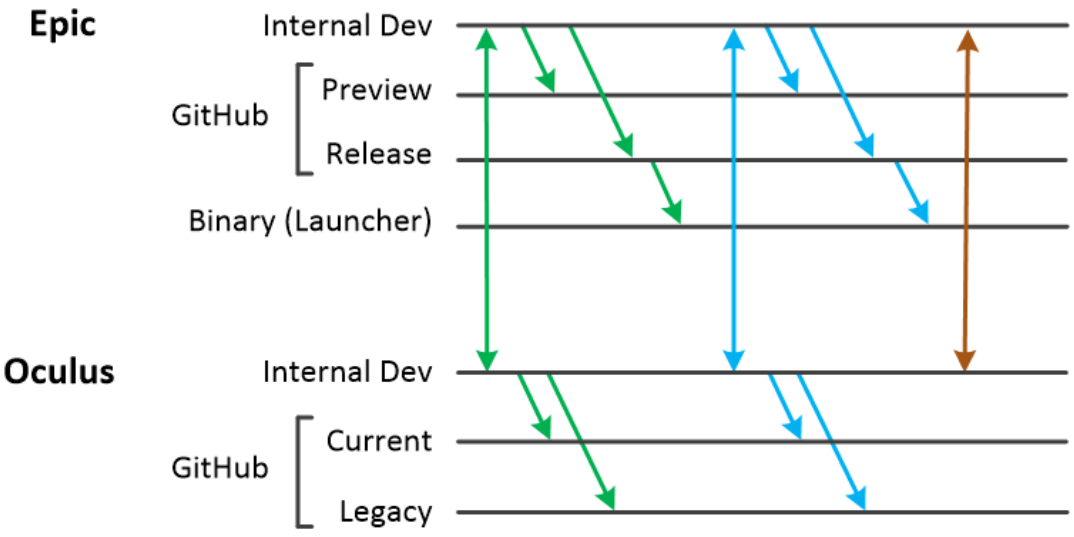
- The standard binary of the engine, available through the Launcher
- Source code for the main release version, hosted on GitHub
- Source code for a preview version, hosted on GitHub

In addition, Oculus maintains its own GitHub repository, which is generally one version ahead of Epic's main public release. We offer:

- Source code for the current branch with the latest Oculus integration features; less stable (corresponding to Epic's Preview branch)
- Source code for the legacy branch (corresponding to Epic's Release branch), which receives backported features from our current branch.

Note: Our features ship to the GitHub versions we maintain in our own repository, unless otherwise noted.

This development sequence is illustrated in the following chart:



Note that our features ship first to the GitHub versions we maintain in our own repository.

Which Version Should I Use?

For beginning developers, we recommend the binary version of the Unreal engine available through the Launcher - it is the most stable version, and does not require pulling from GitHub or compiling the engine source code. It is typically a few months behind the latest Oculus SDK features.

For most professional developers, we recommend the engine versions hosted on Epic's private GitHub repository here: *https://github.com/EpicGames/UnrealEngine*. They are available to developers who are subscribed to the private EpicGames/UnrealEngine repository. For more information on accessing this repository, see *https://www.unrealengine.com/ue4-on-github*.

For professional developers who would like access to the very latest SDK features, we recommend the engine versions hosted on Oculus's private GitHub repository here: *https://github.com/Oculus-VR/UnrealEngine.git*. Note that API changes may occur when these branches get merged back into Epic's version of the engine.

For instructions on building the Unreal Engine from source, see Epic's *Building Unreal Engine from Source* guide.

Unreal/Oculus SDK Version Compatibility

Table 1: Epic GitHub Repository

Branch	Oculus PC SDK	Oculus Mobile SDK	Tag
4.10	0.8.0	0.6.2	4.10.4-release
4.11	1.3.0	1.0.1	4.11.2-release
4.12	1.3.2	1.0.1	4.12.5-release
4.13	1.6.0	1.0.3	4.13.1-release

Table 2: Oculus Github Repository

Branch	Oculus PC SDK	Oculus Mobile SDK	Tag
4.10	1.4.0	1.0.2	oculus-4.10.4-release-1.4.0
4.11	1.6.0	1.0.3	oculus-4.11.2-release-1.6.0
4.12	1.8.0	1.0.3	oculus-4.12.5-release-1.8.0
4.13	1.8.0	1.0.3	oculus-4.13.1-release-1.8.0

Getting Started

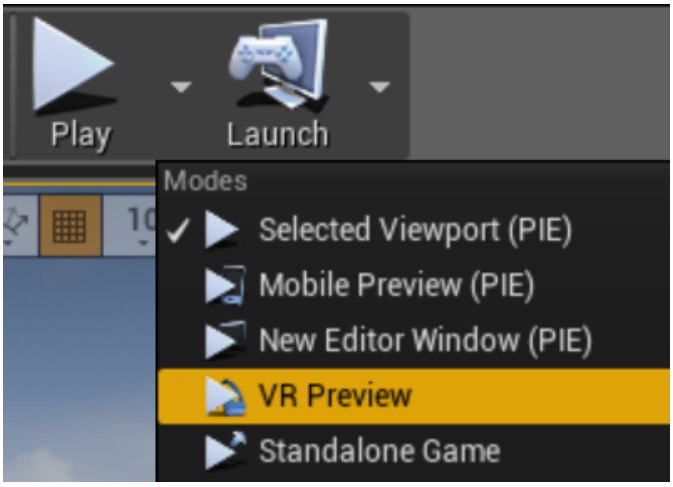
Overview

The easiest way to get started with Oculus development in Unreal is to experiment with the HMD support provided by default as a Player Start. If Unreal detects an Oculus Rift runtime (e.g., the Oculus app is installed), it will automatically provide orientation and positional tracking and stereoscopic rendering for any game.

Standalone PC executables will automatically display in Oculus Rift when run in full-screen mode if a Rift is detected on your system. You may build a project normally and preview it in the Engine by selecting the VR Preview mode in the Play button pulldown options.

Note: In order to play an in-development application, you will need to enable running applications from unknown sources in the Oculus app settings, available through the gear icon in the upper-right. Select *Settings* > *General* and toggle *Unknown Sources* on to allow.

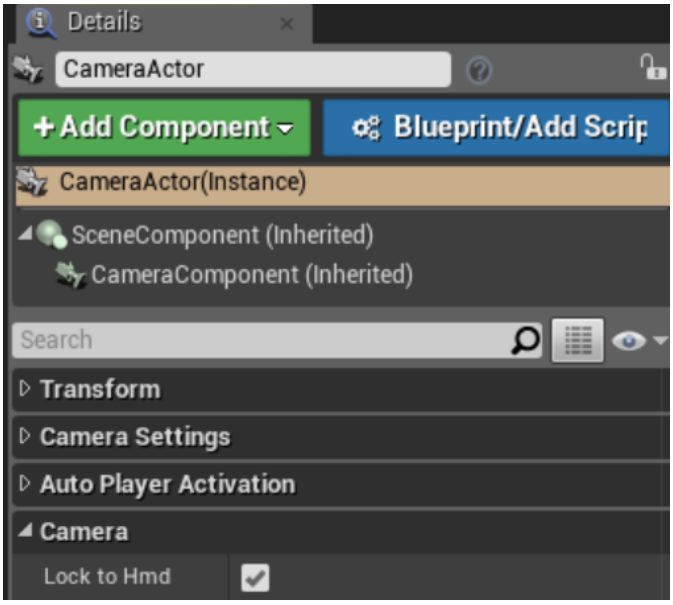
To preview a level in the Rift while editing with the engine, select the dropdown arrow near the Play button and select VR Preview.



To play a scene in the Rift with a standalone PC game, simply maximize the game window and it will be mirrored to the Rift display.

Camera VR

This is a good way to get a quick sense of VR development with minimal overhead, but for actual development, we generally recommend adding a Camera actor to your map and enabling Oculus support by selecting the camera in Viewport and checking the *Lock to Hmd* checkbox in the *Details* tab (available in UE4 11).



Placing a camera in the scene allows you to control the orientation of the camera view when the game loads, so that you can control the exact perspective that will be visible to the user. This is not possible with the Play Start described above.

An additional benefit to using the Camera actor is that you can attach meshes and they will update their position following the HMD view with very little latency. This is generally the best way to add head-locked elements such as cockpit details. Note that we generally discourage head-locked UI elements, but it can be an attractive feature when used carefully.

Platform Features

Unreal applications may use our Platform SDK to add features related to security, community (e.g., matchmaking), revenue (e.g., in-app purchases), and engagement (e.g., leaderboards). For more information, see our *Platform SDK documentation*.

Entitlement checking may be used to protect apps from unauthorized distribution. Most Unreal developers should use the *Verify Entitlements* Blueprint under Oculus Platform. For more information and instructions, see *Entitlements: App Authorizations* in our Platform guide. Applications should always have entitlement checks enabled for submissions to the Oculus store.

Unreal Mobile Development

Mobile Development Setup

Begin preparing for Gear VR development by following the instructions in our Mobile *Device Setup* guide, and install the Java Development Kit (JDK), Native Development Kit, and Android SDK before beginning development. Most Unreal developers do not need to install Android Studio.

Then do any remaining necessary setup for Android development for Unreal, following Epic's *Android Quick Start*.

Be sure to review all of the relevant performance and design documentation for Oculus mobile development. Mobile apps are subject to 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.

Note that your Gear VR application will not run without an Oculus Signature File (osig).

To add your osig for Unreal development:

1. Follow the instructions in *Application Signing* to download your osig.
2. Navigate to the following directory: \Engine\Build\Android\Java\.
3. Create the new directory *assets* inside the directory. Be sure the name is uncapitalized.
4. Copy your osig to this directory.

Workflow

We recommend setting your phone to *Developer Mode*, so you can run VR apps on your phone without inserting it into the Gear VR headset. This allows you to quickly review your progress on the device without much overhead.

You may also launch the engine with a configuration option to use the GearVR plugin and Mobile renderer on the PC. This allows you to preview mobile development from the desktop using an Oculus Rift. To do so, disable OculusRift and SteamVR plugins for your project, and add `-OpenGL -FeatureLevelES2` to your command line.

To run your app during development, we generally recommend connecting your phone via USB, selecting the *Launch* pulldown menu, and then selecting your phone under the listed devices. Particularly if you are modifying maps and shaders but not changing the code, this is often much faster than building an APK.

It is possible to preview a Gear VR application in the Oculus Rift during development. However, it is not generally useful to do so, because Rift applications are subject to substantially different performance requirements (see "Performance Targets" in *Debugging and Performance Analysis in Unreal*). You may find it easiest to use the 2D preview in Unreal, and then either build an APK or use *Launch on Device* when you need to view the app in VR.

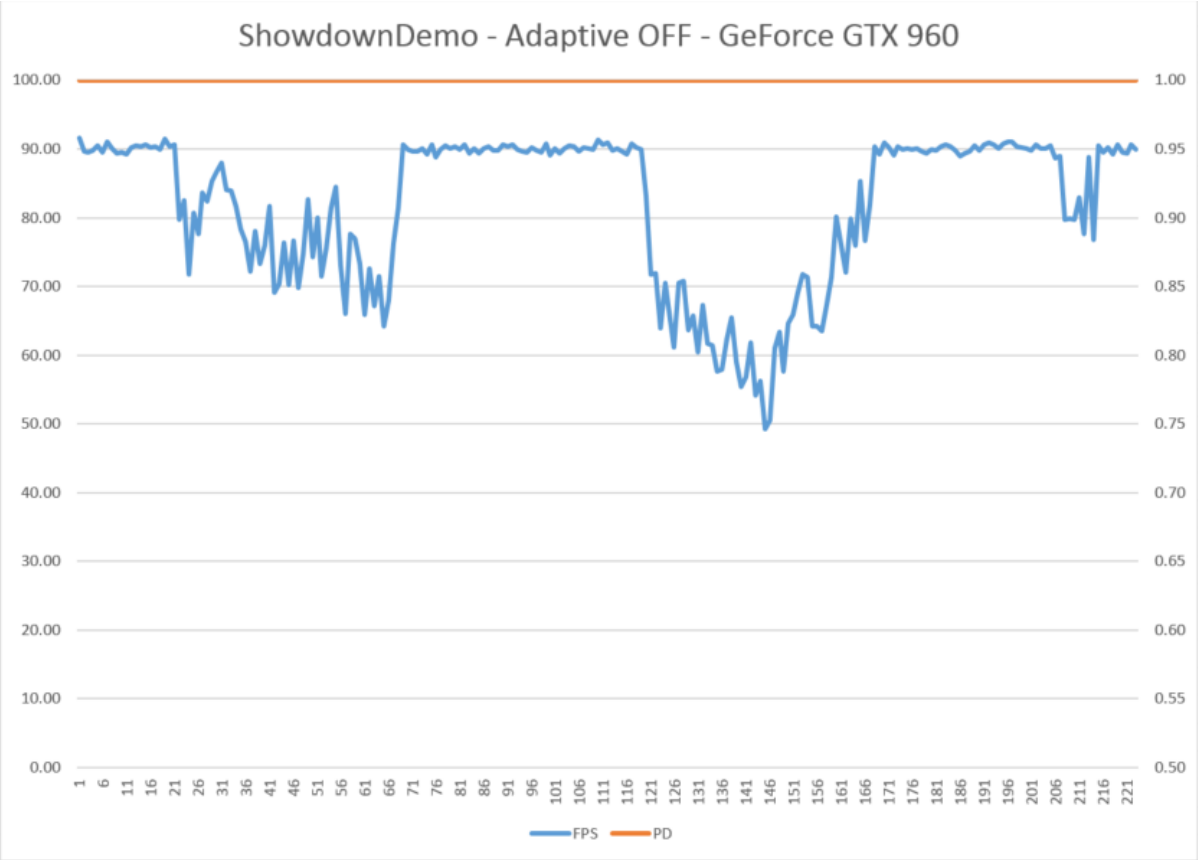
Note: Mobile Content Scale Factor is not currently supported for Oculus development.

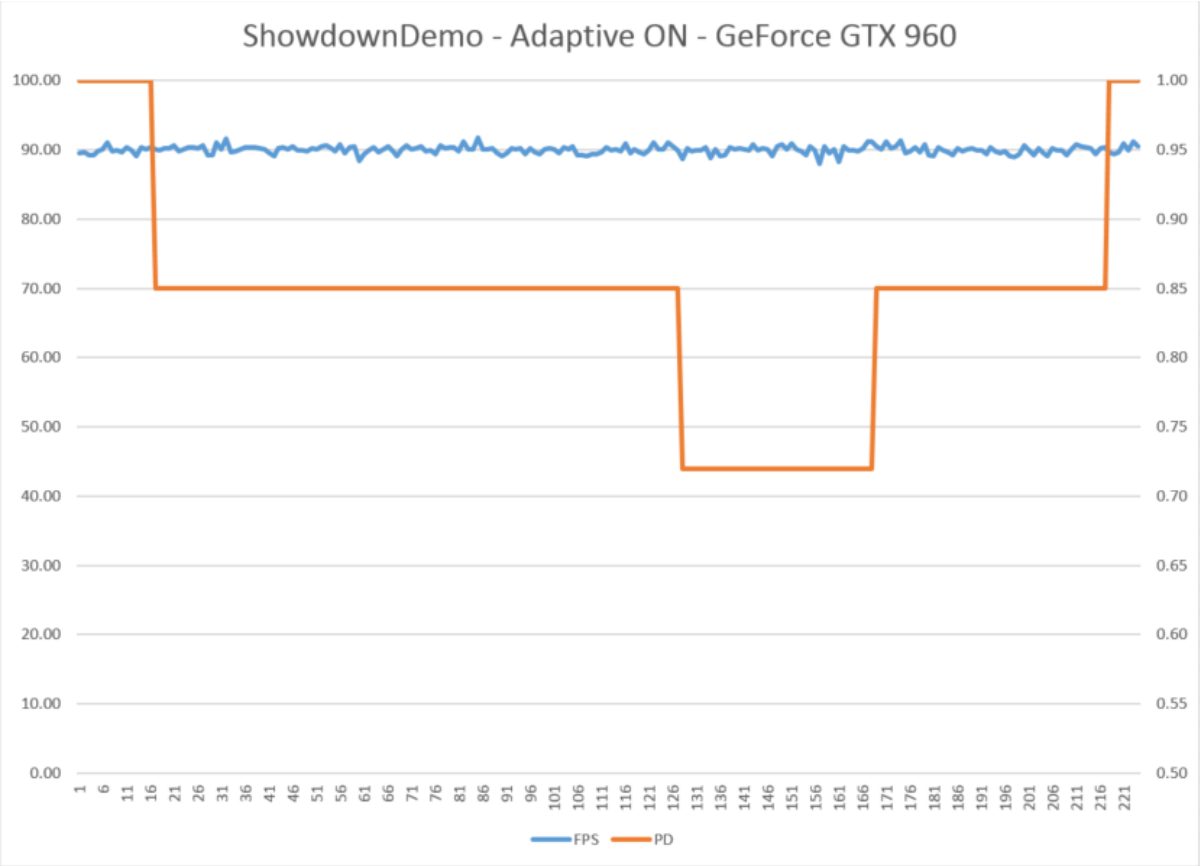
Adaptive Viewport Scaling

Adaptive Viewport Scaling allows applications to scale down the application viewport as GPU resources exceed 85% utilization, and to scale up as they become more available. For CPU-bound applications, this feature has the potential to improve visual quality.

If you do not want some Actors within your level (e.g., text displays) to be scaled, they should be drawn using a separate layer, which does not get scaled by pixel density.

The following charts illustrate pixel density (gold) and frames per second (blue) on a demo application with Adaptive Viewport Scaling off and on, respectively.





To enable Adaptive Viewport Scaling, use the console command `hmd pdadaptive on`. You may specify a minimum and maximum scaling factor (default 1 = normal density). See *Unreal Console Commands* on page 18 for more information.

If you do not want some Actors within your level (e.g., text displays) to be scaled, they should be drawn using a separate layer, which does not get scaled by pixel density.

Note: To minimize the perceived artifacts from changing resolution, there is a two-second minimum delay between every resolution change.

Haptics for Rift Controllers

You may use the standard *Play Haptic Effect* Blueprint to send a specified haptic curve to the Oculus Touch or Xbox controller. For more information, see Unreal's *Play Haptic Effect guide*.

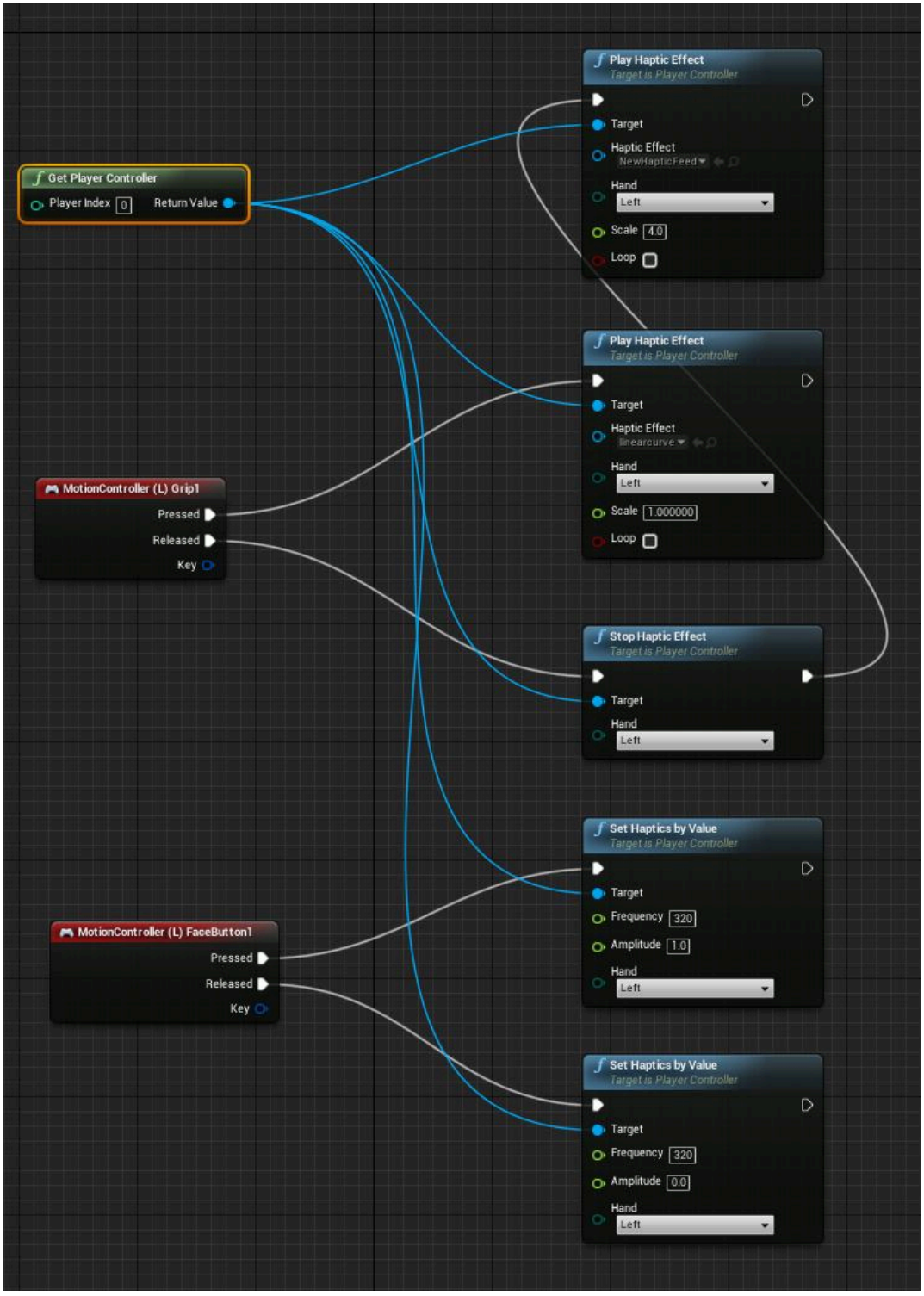
Haptics in Unreal Engine 4.13

PlayHapticEffects may be configured to play haptic waves based on three types of input. Right-click Content Browser to bring up the context menu, then select Miscellaneous. and select one of the following three options:

- Haptic Feedback Buffer: Plays a buffer of bytes 0-255,
- Haptic Feedback Curve: Draw the haptic linear curve you wish to play using the Haptic Curve Editor, or
- Haptic Feedback Soundwave: Select a mono audio file to be converted into a haptic effect of corresponding amplitude.

The following Blueprint illustrates a simple haptics sequence on the Oculus Touch controller using *Play Haptic Effect*. This example sends vibrations using *Play Haptic Effect* when the left controller grip button is pressed. When the button is released, *Stop Haptic Effect* sends a stop command to the Touch controller.

When the left controller X button is pressed, a constant vibration is sent by *Set Haptics by Value* until the button is released. Note that *Set Haptics by Value* calls are limited to 30 Hz; additional calls will be disregarded.

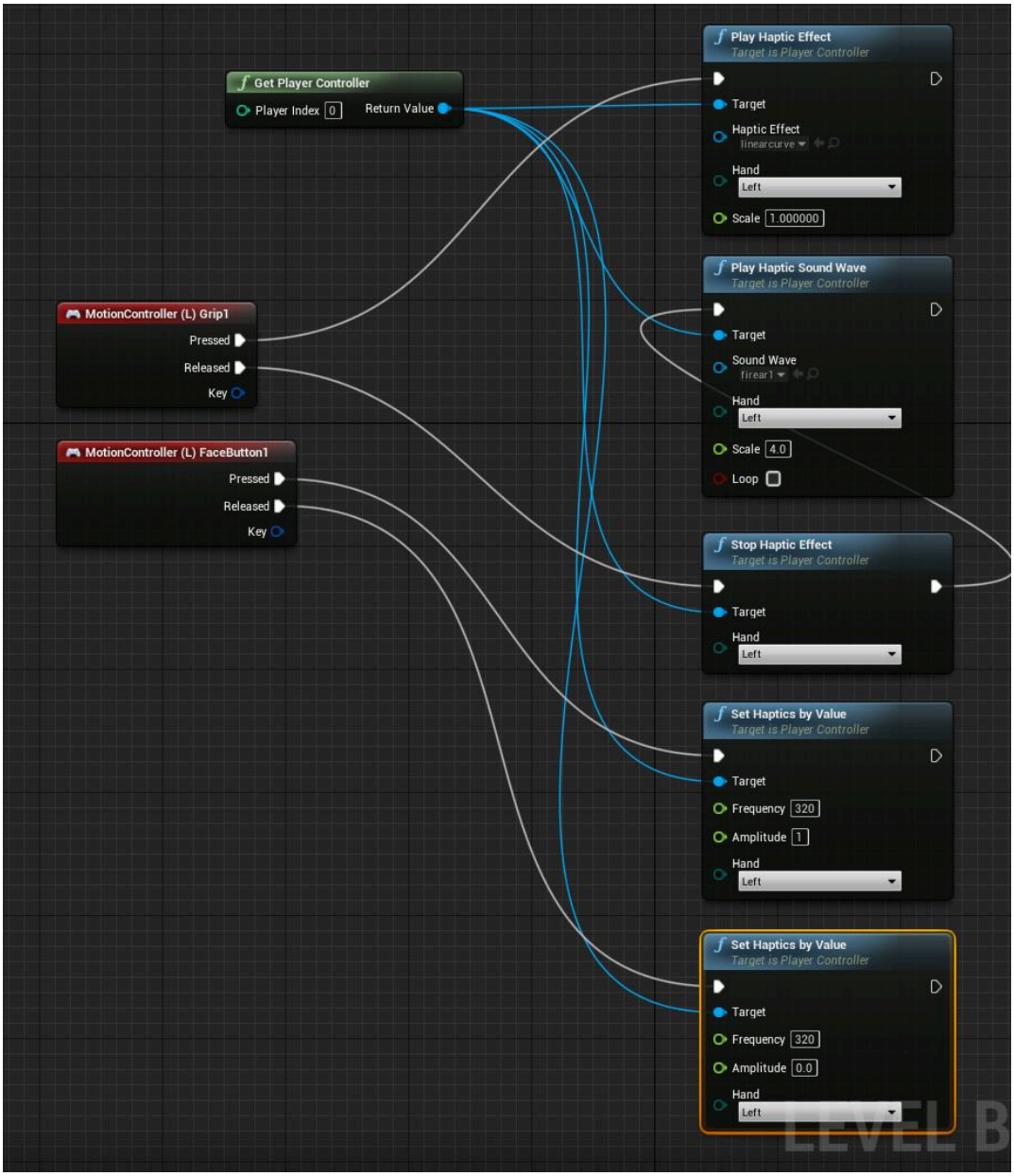


Haptics in Unreal Engine 4.12

In addition to Play Haptic Effects, Unreal 4.12 adds Play Haptic Soundwave.

The following Blueprint illustrates a simple haptics sequence on the Oculus Touch controller using *Play Haptic Effects* and *Play Haptic Soundwave*. This example sends vibrations using *Play Haptic Effect* when the left controller grip button is pressed. When the button is released, *Play Haptic Soundwave* sends a second vibration to the controller.

When the left controller X button is pressed, a constant vibration is sent by *Set Haptics by Value* until the button is released. Note that *Set Haptics by Value* calls are limited to 30 Hz; additional calls will be disregarded.



`APlayerController::PlayHapticSoundWave` takes a mono soundwave as an argument. It downsamples the wave into a series of bytes that serially describe the amplitude of the wave (uint8 values 0-255). Each byte is then multiplied by the factor specified in Scale (max = 255), and haptic vibrations are sent to the targeted Oculus Touch controller. Each controller must be targeted individually. Call *Stop Haptic Effect* to stop haptic playback.

Guardian System Boundary Component

OculusRiftBoundaryComponent exposes an API for interacting with the Oculus Guardian System.

Note: The Guardian System is not yet supported by public versions of the Oculus runtime.

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.

Additional handling may be implemented by applications using the class `UOculusRiftBoundaryComponent`. Possible use cases include pausing the game if the user leaves the Play Area, placing geometry in the world based on boundary points to create a "natural" integrated barrier with in-scene objects, disabling UI when the boundary is being rendered to avoid visual discomfort, et cetera.

All `UOculusRiftBoundaryComponent` public methods are available as Blueprints.

Please see OculusRiftBoundaryComponent.h for additional details.

Basic Use

Boundary types are `Boundary_Outer` and `Boundary_PlayArea`.

Device types are `HMD`, `LTouch`, `RTouch`, `Touch` (i.e., both controllers), and `All`.

Applications may query the interaction between devices and the Outer Boundary or Play Area by using UOculusRiftBoundaryComponent::GetTriggeredPlayAreaInfo() or UOculusRiftBoundaryComponent::GetTriggeredOuterBoundaryInfo().

Applications may also query arbitrary points relative to the Play Area or Outer Boundary using UOculusRiftBoundaryComponent::CheckIfPointWithinOuterBounds() or UOculusRiftBoundaryComponent::CheckIfPointWithinPlayArea(). This may be useful for determining the location of particular Actors in a scene relative to boundaries so, for example, they are spawned within reach, et cetera.

Results are returned as a struct called FBoundaryTestResult, which includes the following members:

Member	Type	Description
IsTriggering	bool	Returns true if the device or point triggers the queried boundary type.
DeviceType	ETrackedDeviceType	Device type triggering boundary.
ClosestDistance	float	Distance between the device or point and the closest point of the test area.
ClosestPoint	FVector	Describes the location in tracking space of the closest boundary point to the queried device or point.

Member	Type	Description
ClosestPointNormal	FVector	Describes the normal of the closest boundary point to the queried device or point.

All dimensions, points, and vectors are returned in Unreal world coordinate space.

Applications may request that boundaries be displayed or hidden using `RequestOuterBoundaryVisible()`. 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. 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 `UOculusRiftBoundaryComponent::IsOuterBoundaryDisplayed()` and `UOculusRiftBoundaryComponent::IsOuterBoundaryTriggered()`.

You may bind delegates using the object `OnOuterBoundaryTriggered`.

Additional Features

You may set the boundary color of the automated Guardian System visualization (alpha is unaffected) using `UOculusRiftBoundaryComponent::SetOuterBoundaryColor()`. Use `UOculusRiftBoundaryComponent::ResetOuterBoundaryColor()` to reset to default settings.

`UOculusRiftBoundaryComponent::GetOuterBoundaryPoints()` and `UOculusRiftBoundaryComponent::GetPlayAreaPoints()` return an array of up to 256 3D 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 `UOculusRiftBoundaryComponent::GetOuterBoundaryDimensions()` or

`UOculusRiftBoundaryComponent::GetPlayAreaDimensions()`, which return a vectors containing the width, height, and depth in tracking space units, with height always returning 0.

VR Compositor Layers

In some versions of Unreal, you may add transparent or opaque quadrilateral, cubemap, or cylindrical overlays to your level as compositor layers.

TimeWarp compositor layers are rendered at the same frame rate as the compositor, rather than rendering at the application frame rate. They are less prone to judder, and are raytraced through the lenses, which improves the clarity of textures displayed on them. This is useful for displaying easy-to-read text.

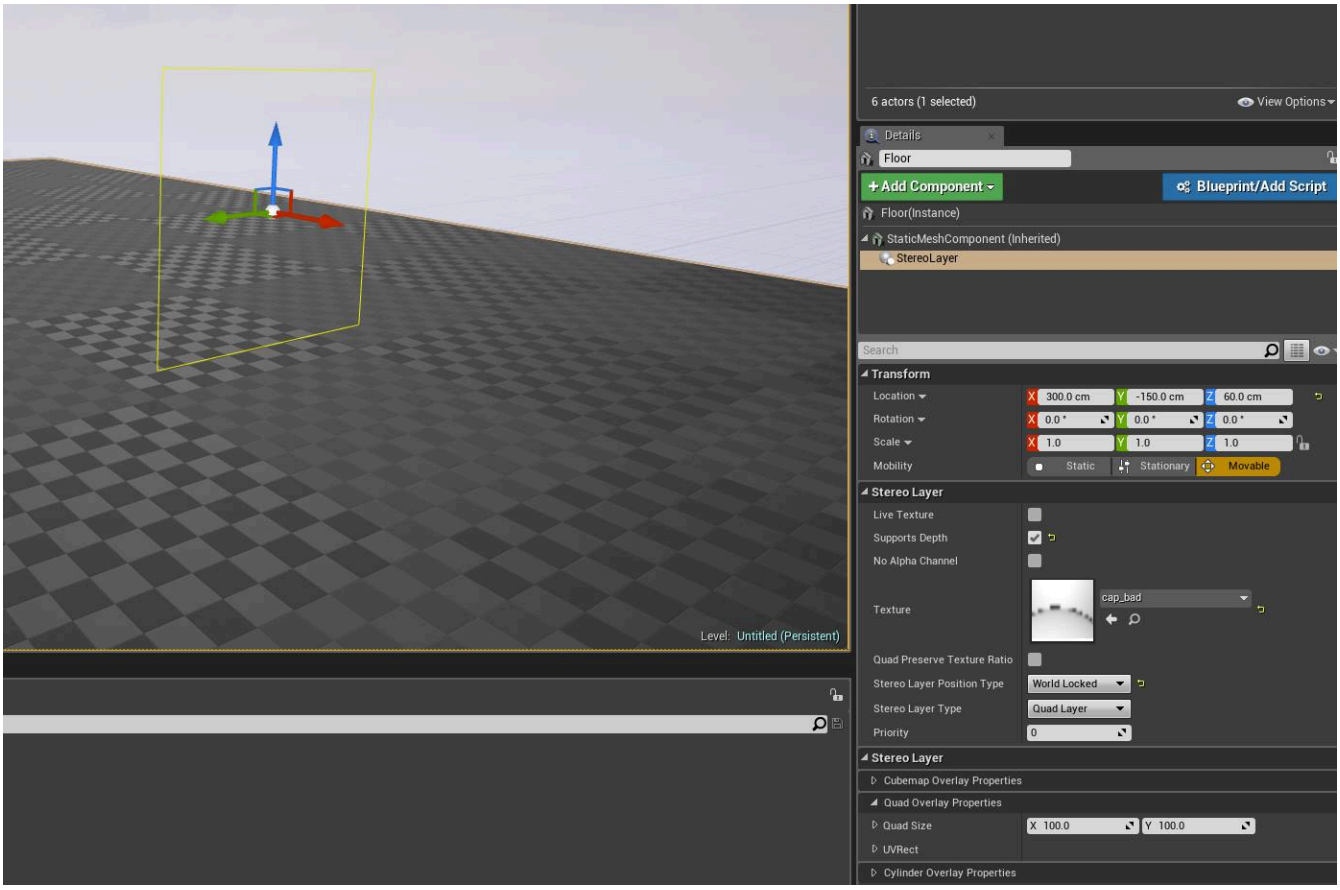
Gaze cursors and UIs are good candidates for rendering as quadrilateral compositor layers. Cylinders may be useful for smooth-curve UI interfaces. Cubemaps may be used for startup scenes or skyboxes.

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. This can significantly improve application startup times.

Layer Type	Supported Version	Supported Platform
Quadrilateral	4.12, 4.13	Rift, mobile
Cylinder	4.12, 4.13	mobile
Cubemap	4.13	mobile

By default, VR compositor layers are always displayed on top of all other objects in the scene. You may set compositor layers to respect depth positioning by enabling *Supports Depth*. If you are using multiple layers, use the *Priority* setting to control the depth order in which the layers will be displayed, with lower values indicating greater priority (e.g., 0 is before 1).

Note that enabling *Supports Depth* may affect performance, so use it with caution and be sure to assess its impact.



To create an overlay:

1. Create an Actor and add it to the level. If you don't want the Actor to be visible in the scene, use an Empty Actor.
2. Select the Actor, select *Add Component*, and choose *Stereo Layer*.
3. Under the Stereo Layer options, set *Stereo Layer Type* to *Quad Layer* or *Cylinder Layer*.
4. Set *Stereo Layer Type* to *Face Locked*, *Torso Locked*, or *World Locked*.
5. Set the overlay dimensions in world units in *Quad Stereo Layer Properties* or *Cylinder Stereo Layer Properties*.
6. Select *Supports Depth* in *Stereo Layer* to set your compositor layer to not always appear on top of other scene geometry. Note that this may affect performance.
7. Configure Texture and additional attributes as appropriate.

The Actor you slave the component to will be fixed at the center of the quad or the cylinder.

You may add up to three VR compositor layers to Gear VR applications, and up to fifteen to Rift applications.

Unreal Console Commands

This document reviews useful console commands available for Unreal development.

The console may be summoned by pressing the ~ (tilde) or tab keys while your game is running. For more information, see "Useful VR Console Commands" in Unreal's *VR Cheat Sheet*.

Table 3: Configuration Commands

Command	Description
stereo onloff	Enables/Disables stereo mode.
stereo e=0.064	Eye distance (m).
stereo w2m=100	Overrides default worldunits-to-meters scale.
stereo ncp=10 fcp=10000	Overrides near clipping and/or far clipping planes for stereo rendering (in cm).
stereo cs=1 ps=1	Overrides camera and position scale.
stereo show	Shows current ipd and head model offset.
stereo reset	Resets stereo settings.
hmd enableldisable	Enables/Disables HMD.
hmd pd [0..3.0]	Sets pixel density in the center (default is 1.0).
hmd pdadaptive onloff	Enables/Disables adaptive pixel density (see *Adaptive Viewport Scaling* on page 9 for mor details).
hmd pdmax [0.5..2.0]	Sets maximum adaptive pixel density (ignored if hmd pdadaptive is off).
hmd pdmin [0.5..2.0]	Sets minimum adaptive pixel density (ignored if hmd pdadaptive is off).
hmd sp [30..300]	Overrides screenpercentage for stereo mode. (Deprecated, use 'hmd pd xxx' instead).
hmd hqdistortion onloff	Enables/Disables high-quality distortion.
hmd mirror onloff	Enables/Disables mirroring to the desktop window.
hmd mirror mode [0..4]	Sets mirror mode: 0=Distorted, 1=Undistorted, 2=SingleEye, 3=SingleEye Letterboxed, 4=SingleEye Cropped
hmdpos onloffltoggle	Enables/Disables/Toggles positional tracking.
hmdpos enforce onloff	Enables/Disables head tracking even if not in stereo (for testing purposes).

Command	Description
hmdpos reset {yaw}	Resets position and rotation, applies yaw (in degrees) if provided.
hmdpos resetrot {yaw}	Resets rotation only, applies yaw (in degrees) if provided.
hmdpos resetpos	Resets position only.
hmdpos show	Outputs status of positional tracking to log.
hmdpos floorleye	Selects tracking origin.

Table 4: Misc Commands

Command	Description
hmd stats	Shows HMD-related stats.
hmd grid	Toggles lens-centered grid.
hmd updateongt onloff	Enables/Disables updating HMD pose on game thread. On by default.
hmd updateonrt onloff	Enables/Disables updating HMD pose on render thread, for lower latency. On by default.
hmd cubemap [gearvr] [xoff=N] [yoff=N] [zoff=N] [yaw=N]	Generates a cube map image of your application. May be used for VR app previews in the Store. Cube map PNGs will be saved in the directory GameDir/Saved/Cubemaps. gearvr: If specified, cube map size will be 6x1024x1024, otherwise it will be 6*2048x2048. xoff, yoff, zoff: Offset from the current player's location. yaw: override yaw rotation (degrees).
hmd setint PerfHUDMode [0..4]	Selects performance HUD mode, set to 0 to disable.
hmd setint DebugHudStereoMode [0..3]	Selects debug HUD stereo mode, set to 0 to disable.
hmdversion	Prints Oculus SDK version used and Oculus Plugin info.

Debugging and Performance Analysis in Unreal

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.

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 Store)
- 50-100 draw calls per frame
- 50,000-100,000 triangles or vertices per frame

PC

- 90 FPS (required by Oculus Store)
- 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*

Rift Performance HUD

The Oculus Performance Heads-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. You may activate it in the Viewport by pressing the ~ key.

For more details, see the *Performance Heads-Up Display* and *Oculus Debug Tool* sections of the Oculus Rift Developers Guide.

Mobile Debug Console

If your phone is set to Developer Mode, you may bring up a debug console for VR apps by pressing the screen with four fingers simultaneously while the application is running.

Enter `stat unit` in the console for information about your application frame rate and GPU performance.

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/

Other Resources

For detailed information about Oculus development, go to:

- Unreal: Virtual Reality Development: *https://docs.unrealengine.com/latest/INT/Platforms/VRZ/*
- Unreal: Oculus Rift wiki: *https://wiki.unrealengine.com/Oculus_Rift*
- Oculus Forums/Unreal: *https://forums.oculus.com/developer/categories/unreal*
- Unreal Forums/VR Development: *https://forums.unrealengine.com/forumdisplay.php?27-VR-Development*

Contact

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

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

Release Notes

This section describes changes for each version release.

Oculus Unreal Engine 4 Integration 1.8 Release Notes

1.8.0

These release notes describe changes to the Unreal Engine versions 4.12 and 4.13 available from the Oculus GitHub repository.

The Oculus UE4 integration ships with various versions of Unreal, and there is no external SDK per se. Clicking on the "Download" link above will redirect you to our *Unreal Introduction* on page 4, which walks through various options for working with the Oculus Unreal integration.

Unreal provides built-in support for Oculus Rift and Gear VR development on Windows, and Unreal apps run on the Oculus platform automatically apply stereoscopic rendering to the main camera as well as positional and orientation tracking for the Rift, or orientation tracking for Gear VR.

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 *Guardian System Boundary Component* on page 14 for more information.
- Added depth ordering support to VR compositor layers.
- Added cubemap support for VR compositor overlays to 4.13 (mobile only).
- Added Online Subsystem support for Regular Leaderboards and Cloudsaves.
- Added a Blueprint for retrieving Oculus ID/Username.
- Added Blueprints for `UOculusRiftBoundaryComponent` public methods.

API Changes

- Added OculusRiftBoundaryComponent API for the Oculus Guardian System.

Bug Fixes

- Fixed black screen issue affecting Samsung Note4 and certain Adreno-based devices.

Known Issues

- UE4.12 and 4.13 in Oculus GitHub repository: A significant drop in frame rate occurs when UE4 is not in focus in VR preview mode. To avoid this issue, uncheck the Use Less CPU when in *Background* in *Edit* > *Editor Preferences* > *General* (left sidebar) > *Miscellaneous* (left sidebar) > *Performance*.
- Exclusive Mode issues: Multiple initializations of the DXGISwapChain may cause flickering as the screen switches modes and a black screen when rendering to the Rift with a different GPU from the one the game is using to render the eye buffers.
- *Stereo Layer Position Type*: *World Locked* is not currently supported for cylinder TimeWarp overlays.

Oculus Unreal Engine 4 Integration 1.7 Release Notes

1.7.0

Unreal provides built-in support for Oculus Rift and Gear VR development on Windows.

Unreal apps run on the Oculus platform automatically apply stereoscopic rendering to the main camera as well as positional and orientation tracking for the Rift, or orientation tracking for Gear VR.

For more information on downloading and using the Oculus Unreal integration, please see *Unreal Introduction* on page 4.

Online Subsystems

Several features of the Oculus Platform SDK may be accessed through the Unreal Online Subsystems interface:

- Achievements
- Rooms
- Matchmaking
- Identity
- Entitlements
- P2P networking
- Friends leaderboards

Friends leaderboards is not currently available through Online Subsystems in Epic's 4.13 Preview build.

Not all functionality of these features is available through the Online Subsystems interface. Developers who need unsupported features should use the Oculus Platform Plugin available with our Platform SDK.

Online Subsystems access is available in the following ways:

- Oculus Platform SDK v. 1.6 and 1.7 include a standalone OSS plugin (Rift only)
- Epic GitHub: 4.13.0-preview-2 includes OSS support (Rift and Gear VR)
- Oculus GitHub: oculus-4.12.5-release-1.7.0 and oculus-4.13.0-preview-2-1.7.0 includes OSS support (Rift and Gear VR)

For more information on Unreal Subsystems, see Epic's Online Subsystems Overview.

New Features

- Added adaptive viewport scaling, which automatically changes pixel density based on resource availability, with configurable min/max. See *Unreal Console Commands* on page 18 for details. (Rift only)
- Added Oculus Platform SDK support to mobile VR applications to 4.12 and 4.13 in the Oculus repository.
- Added Blueprint for Oculus Platform entitlement checks (available in SI 1.6).
- A subset of Oculus Platform features are now available through the Unreal Online Subsystems interface.
- Added cylinder TimeWarp overlay support (Gear VR only) for UI components, etc. For more information, see *VR Compositor Layers* on page 16.
- Added ability develop using GearVR plugin and Mobile renderer on the PC using the Oculus Rift. To use, disable OculusRift and SteamVR plugins for your project, and add `-OpenGL -FeatureLevelES2` to your command line.

API Changes

- Added Cylinder layer type to FHMDLayerDesc in HeadMountedDisplayCommon.h.
- In IInputInterface.h, FHapticFeedbackBuffer uint8* RawData was changed to TArray<uint8> RawData, and uint8 *CurrentPtr; was changed to uint32 CurrentPtr.
- IInputInterface::SetHapticFeedbackBuffer was removed and HapticBuffer was added to FHapticFeedbackValues::FHapticFeedbackBuffer* HapticBuffer. You should now use IInputInterface::SetHapticFeedbackValues, whether you are using a buffer or not.
- APlayerController::SetHatpicsByValue calls to Oculus Touch are now limited to 30 per second due to performance issues. Additional calls with a value other than zero (i.e., stop commands) are discarded upon receipt.

Known Issues

- UE4.12 and 4.13 in Oculus GitHub repository: A significant drop in frame rate occurs when UE4 is not in focus in VR preview mode. To avoid this issue, uncheck the Use Less CPU when in *Background* in *Edit* > *Editor Preferences* > *General* (left sidebar) > *Miscellaneous* (left sidebar) > *Performance*.
- *Stereo Layer Position Type*: *World Locked* is not currently supported for cylinder TimeWarp overlays.

Oculus Unreal Engine 4 Integration 1.6 Release Notes

1.6.0

Unreal provides built-in support for Oculus Rift and Gear VR development on Windows.

Unreal apps run on the Oculus platform automatically apply stereoscopic rendering to the main camera as well as positional and orientation tracking for the Rift, or orientation tracking for Gear VR.

For more information on downloading and using the Oculus Unreal integration, please see *Unreal Introduction* on page 4.

The Wwise redirection patch is no longer required in Unreal Engine 4.11 or 4.12. Requires the WwiseUE4Plugin from AudioKinetic's 'master' branch on GitHub.

4.12

New Features

- Added adaptive viewport scaling, which automatically changes pixel density based on resource availability, with configurable min/max. See *Unreal Console Commands* on page 18 for details. (Rift only) UPDATE: This feature has been delayed until a future release.
- Added Blueprints plugin that supports calls for Gear VR functionality such as GPU/GPU throttling, querying headphone connection, querying device temperature, and more.
- Added Blueprint for Oculus Platform entitlement checks.

API Changes

- Oculus Touch: Added PlayHapticsSoundwave to Blueprints, which permits playback of soundwaves (stored in UE4 in USoundWave assets) directly to the Oculus Touch controller.
- Added SetHapticFeedbackBuffer to the C++ Haptics interface, to submit byte arrays to be played at 320 Hz on the Oculus Touch controller.

Bug Fixes

- Fixed bug where unfocused Wwise apps continued playing audio. Requires the WwiseUE4Plugin from AudioKinetic's 'master' branch on Github.

4.11

New Features

- Added adaptive viewport scaling, which automatically changes pixel density based on resource availability, with configurable min/max. See *Unreal Console Commands* on page 18 for details. (Rift only) UPDATE: This feature has been delayed until a future release.

API Changes

- Oculus Touch: Added PlayHapticsSoundwave to Blueprints, which permits playback of soundwaves (stored in UE4 in USoundWave assets) directly to the Touch controller.
- Added SetHapticFeedbackBuffer to the C++ Haptics interface, to submit byte arrays to be played at 320 Hz on the Touch controller.

Bug Fixes

- Fixed bug where unfocused Wwise apps continued playing audio. Requires the WwiseUE4Plugin from AudioKinetic's 'master' branch on GitHub.

Oculus Unreal Engine 4 Integration 1.5 Release Notes

1.5.0

Unreal provides built-in support for Oculus Rift and Gear VR development on Windows.

Unreal apps run on the Oculus platform automatically apply stereoscopic rendering to the main camera as well as positional and orientation tracking for the Rift, or orientation tracking for Gear VR.

For more information on downloading and using the Oculus Unreal integration, please see *Unreal Introduction* on page 4.

4.12

New Features

- Added async splash (Gear VR).
- Refactored layers management (Gear VR).
- Enabled default loading icon mode (Gear VR).
- Added ability to capture cube maps for Oculus Store preview (see *Unreal Console Commands* on page 18 for usage instructions).
- Added call ovr_IdentifyClient to identify client to service.
- Added OnlineSubsystemOculus to interface Oculus platform.

API Changes

- Added GetNumOfTrackingSensors / GetTrackingSensorProperies to IHeadMountedDisplay.

Bug Fixes

- Fixed issue with BeginPlay.

- Fixed rounding error that could cause incorrect viewport sizes.
- Now use ovr_GetSessionStatus() instead of SubmitFrame result to determine VR focus.
- Fixed crash creating HMD when it is attached to D3D11 adapter with no other monitors.
- Fixed issue where IHeadMountedDisplayModule::IsHMDConnected would be called more than necessary.

4.11

New Features

- Added async splash (Gear VR).
- Refactored layers management (Gear VR).
- Enabled default loading icon mode (Gear VR).
- Added ability to capture cube maps for Oculus Store preview (see *Unreal Console Commands* on page 18 for usage instructions).
- Added ovr_IdentifyClient call to identify client to service.

API Changes

- Added GetNumOfTrackingSensors / GetTrackingSensorProperies to IHeadMountedDisplay.

Bug Fixes

- Now uses ovr_GetSessionStatus() instead of SubmitFrame result to determine VR focus.
- Fixed issue where IHeadMountedDisplayModule::IsHMDConnected would be called more than necessary.

Oculus Unreal Engine 4 Integration 1.4 Release Notes

1.3.2

Unreal provides built-in support for Oculus Rift and Gear VR development on Windows.

Unreal apps run on the Oculus platform automatically apply stereoscopic rendering to the main camera as well as positional and orientation tracking for the Rift, or orientation tracking for Gear VR.

For more information on downloading and using the Oculus Unreal integration, please see *Unreal Introduction* on page 4.

New Features

- Updated integration up to 4.11.2.
- Added Oculus Rift video layer support to 4.11 integration.
- Upgraded Oculus Mobile SDK to VrApi 1.0.2; Volume Control layer is now rendered correctly.

Bug Fixes

- Fixed potential Gear VR crash when EnterVRMode is called before RenderThread is started.
- Switched LOD calculation from view-vector dot product to pure distance for more consistent result (otherwise, LOD could be different for each eye).
- Fixed incorrect logic for ATW splash rotation (PC).

Known Issues

- Mirror window may appear stretched with hmd mirror mode 2 until you switch focus to another application and back.

Oculus Unreal Engine 4 Integration 1.3 Release Notes

1.3.2

Unreal provides built-in support for Oculus Rift and Gear VR development on Windows.

Unreal apps run on the Oculus platform automatically apply stereoscopic rendering to the main camera as well as positional and orientation tracking for the Rift, or orientation tracking for Gear VR.

For more information on downloading and using the Oculus Unreal integration, please see *Unreal Introduction* on page 4.

New Features

- Updated integration up to 4.11.1 and 4.10.4.
- Added quad layer support for Gear VR.
- Added Gear VR video layer support to 4.11 integration.
- Separated Alt-Enter from switching between stereo/mono; currently, Alt-Enter switches between fullscreen/windowed mode for mirror window.
- Added proper support for fullscreen mirror window.

Bug Fixes

- Fixed base orientation issue with quad layers rendering.
- Fixed Steam VR compatibility issue.
- Fixed Oculus Touch haptic logic: vibration won't be send to Oculus Touch unless the controllers are currently active.