Oculus Mobile Unity Integration Guide

v. 0.4

1. Introduction

This document provides a basic guide to the Unity mobile SDK, which includes several sample Unity applications, SDK Examples, and an integration package for creating your own Unity virtual reality application.

Before using this SDK and the documentation, if you have not already done so, review the *Device and Environment Setup Guide* to ensure that you are using a supported device, and to ensure that your Android Development Environment and mobile device are configured and set up to build and run Android applications.

1.1 Requirements

Mobile SDK is compatible with **Unity Pro 4.5** (supports multi-threaded rendering for Android). We recommend Unity Pro 4.5.5.p1, which includes fixes for mt-rendering and random crashes with dynamic batching enabled.

For additional system and hardware requirements, see the *Device and Environment Setup Guide*.

2. First Steps

2.1 Package contents

Included with the mobile SDK, you will find a set of pre-built Unity Demos, as well as an SDK Examples project with multiple scenes demonstrating useful mobile VR scenarios and functionality. More information about both the pre-built demos and SDK Examples project can be found in the sections which follow.

2.2 Running the Pre-built Unity Demos

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.

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:

1. You should see new app icons on the Android Home screens.
2. A toast will appear with a dialog like the following: “Insert Device: To open this application, insert your device into your Gear VR”
3. Insert your device into the supported Gear VR hardware.
4. The app should now launch.

2.3 Control Layout

2.3.1 Application Specific Controls

BlockSplosion

In BlockSplosion, the camera position does not change, but the Gear VR will track the user’s head orientation, allowing them to aim before launching a block.

- 1-dot Button or Gear VR touchpad tap launches a block in the facing direction.
- 2-dot Button resets the current level.
- Left Shoulder Button (L) skips to the next level.

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 Gear VR.

2.3.2 Supplied Gamepad Controller Button Mappings

Button / Axis Name	**Input Manager Mapping / Axis Value**	**Sensitivity**
Button A / 1	joystick button 0	1000
Button B / 2	joystick button 1	1000
Button X / 3	joystick button 2	1000
Button Y / 4	joystick button 3	1000
Left Shoulder Button	joystick button 4	1000
Right Shoulder Button	joystick button 5	1000
Left Trigger Button	n/a	1000
Right Trigger Button	n/a	1000
Left Analog X Axis	X axis	1
Left Analog Y Axis	Y axis	1
Right Analog X Axis	3rd axis (Joysticks and Scroll wheel)	1
Right Analog Y Axis	4th axis (Joysticks)	1
Dpad X Axis	5th axis (Joysticks)	1000
Dpad Y Axis	6th axis (Joysticks)	1000
Play Button	n/a	1000
Select Button	joystick button 11	1000
Start Button	joystick button 10	1000

2.3.3 Supplied Gear VR Touchpad Controller Mappings

Button / Axis Name	**Input Manager Mapping / Axis Value**	**Sensitivity**
Mouse 0 / Button 1	mouse 0 *and* joystick button 0	1000
Mouse 1 / Button 2	mouse 1 *and* joystick button 1	1000
Mouse X	Mouse Movement X axis	0.1
Mouse Y	Mouse Movement Y axis	0.1

These controller and touchpad input mappings can be set up in Unity under *Project Settings -> Input Manager*. The touchpad and the Back Button are mapped as *Mouse 0* and *Mouse 1,* respectively.

Note that an InputManager.asset file with default input settings suitable for use with the Moonlight Input Control script is included with the Oculus Unity Integration Package.

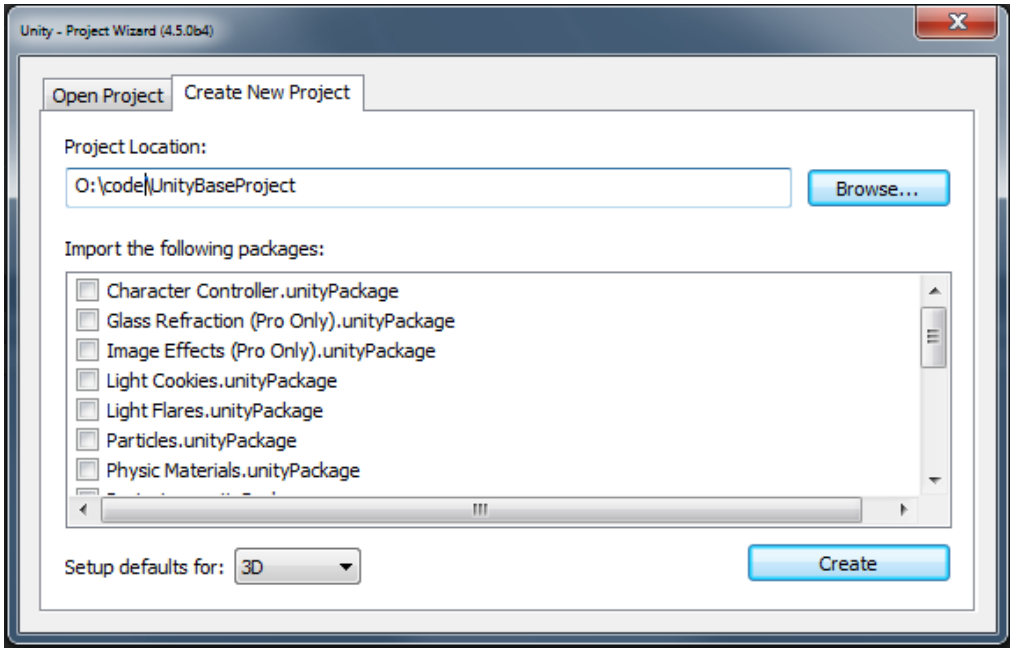
3. Using the Oculus Mobile Unity Integration

The next section will walk you through setting up and building BlockSplosion for Android.

3.1 Installation

If you have Unity open, save your work before proceeding.

Create a new, empty project that will be used to import the Oculus assets into. From the Unity menu, select *File -> New Project*. Click the *Browse* button and select the folder where the Unity project will be located.



[Unity Pro 4.5]

Make sure that the *Setup defaults for:* field is set to *3D*.

You do not need to import any standard or pro Unity asset packages; the Oculus Unity Integration is fully self contained.

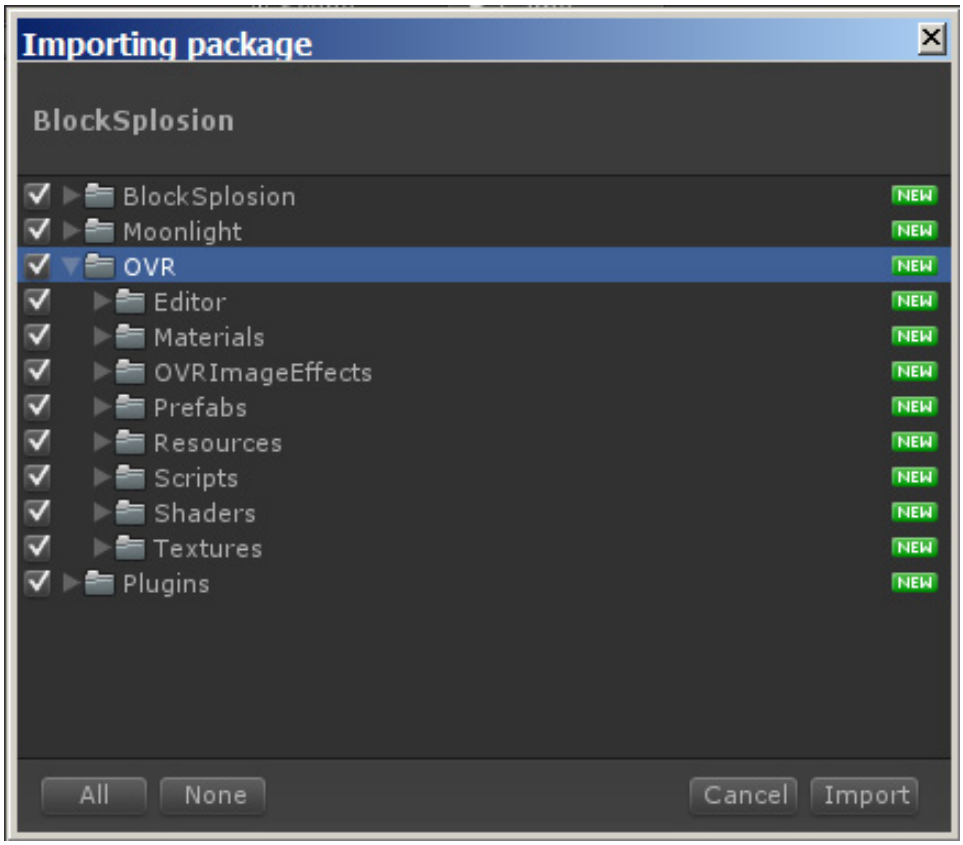
Click the *Create* button. Unity will re-open with the new project loaded.

The Oculus Unity Integration package must be imported into the new project. Go to the *Assets* menu, choose *Import Package*, then *Custom Package*. In the file dialog, browse to the

location of Blocksplosion.unitypackage (e.g., C:\Dev\Oculus\Mobile\VrUnity\Blocksplosion). Select the file and click the *Open* button. The *Importing package* dialog will appear. The import process may take a few minutes to complete.

Alternately, you can navigate to the location of Blocksplosion.unitypackage and double-click the file.

Click *Import* and Unity will decompress the package contents into the project folder.



[Unity Pro 4.5]

3.2 Working with the Unity Integration

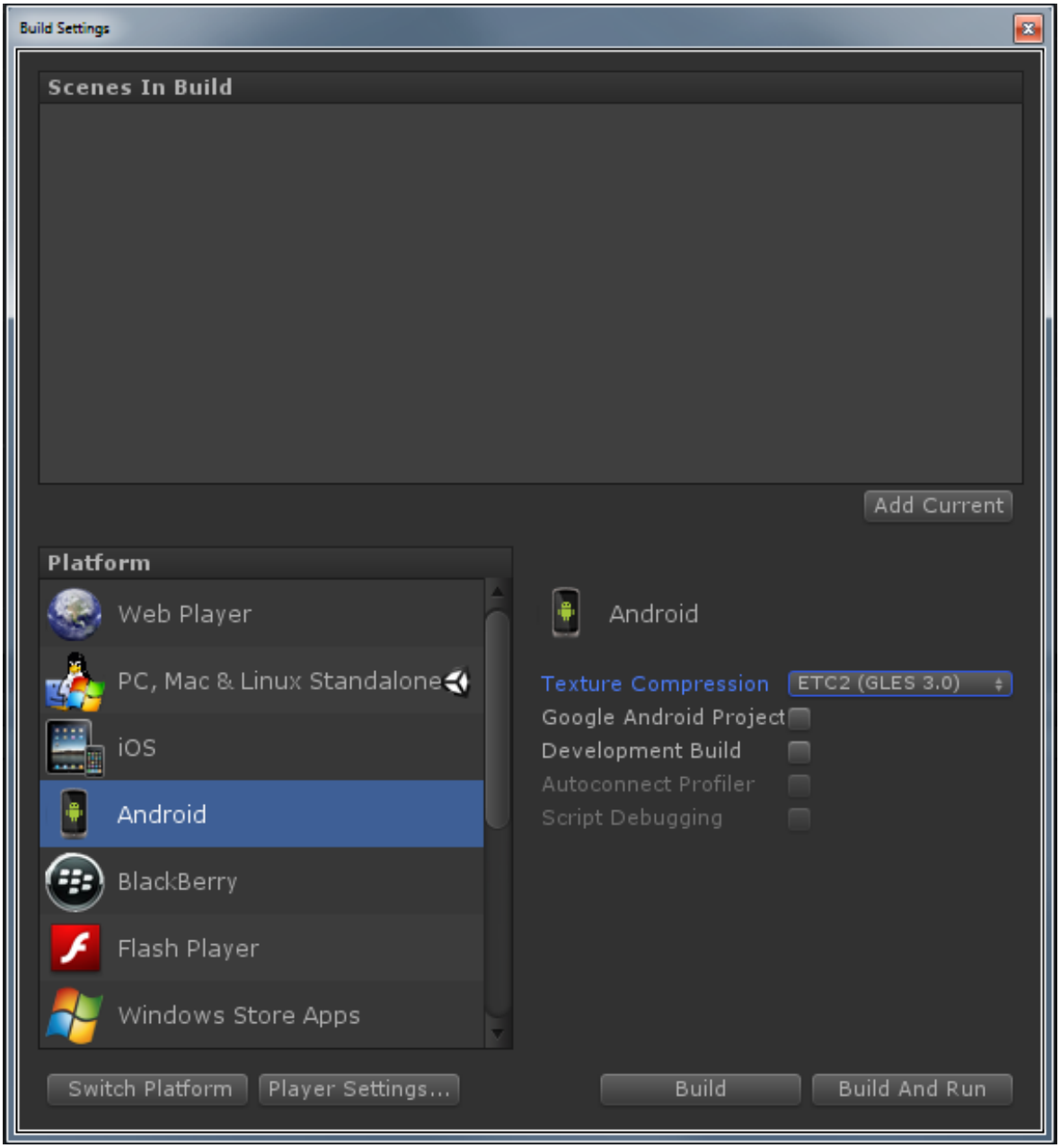
Click on the project browser tab and navigate into Blocksplosion/Scenes. Double-click on the Main.scene to load it as the current scene. If you are prompted to save the current scene, select *Don't save*.

The BlockSplosion sample includes a *Project Settings* folder which provides default settings for a VR mobile application. You may manually copy this to your [Project]/Assets/ProjectSettings folder.

We will walk through setting these options, as well as a few additional options, below.

3.2.1 Configuring Build Settings

From the *File* menu, select *Build Settings*…. From the *Build Settings* menu, select *Android* as the platform.

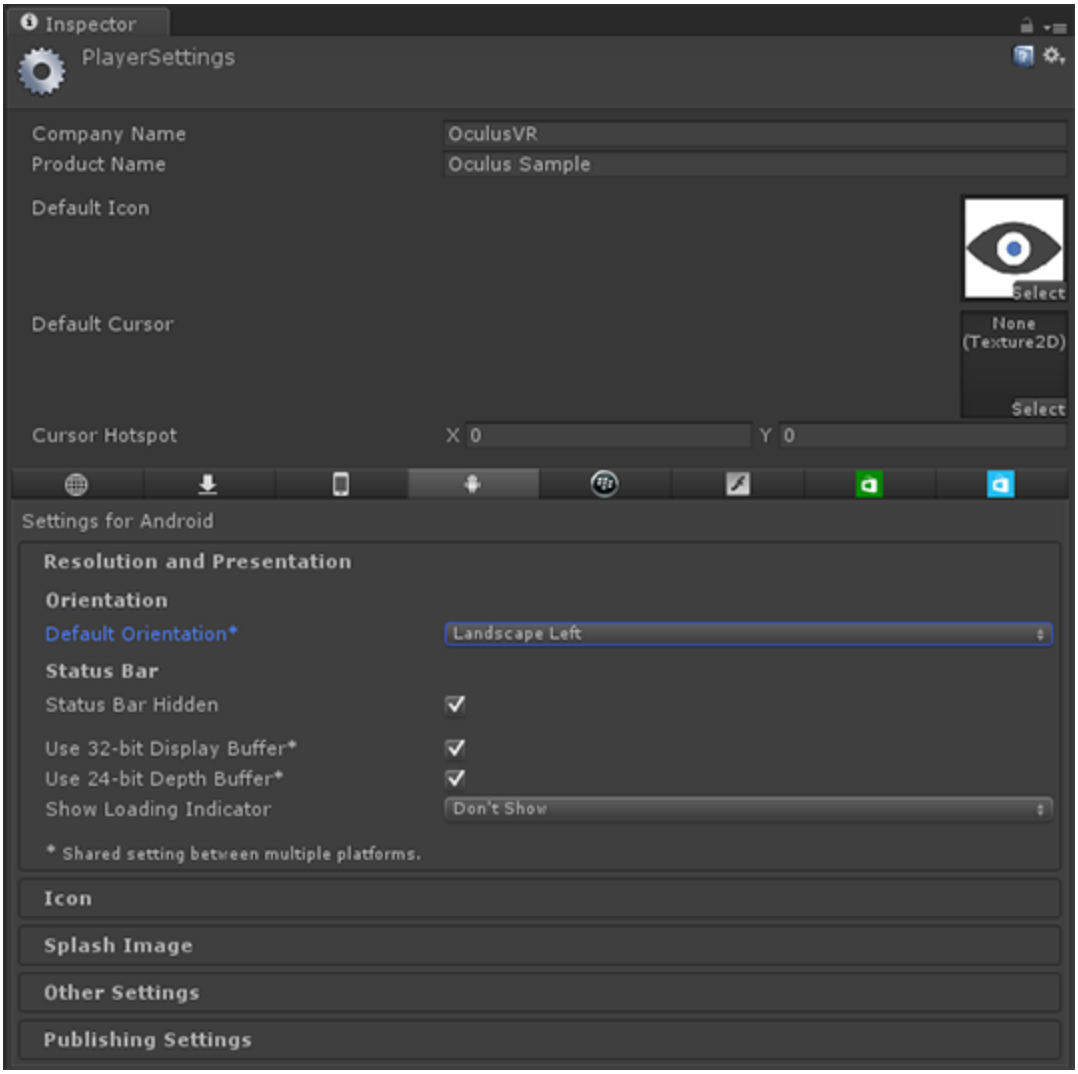


[Unity Pro 4.5]

Set *Texture Compression* to *ETC2 (GLES 3.0)*.

3.2.2 Configuring Player Settings

Click the *Player Settings…* button and select the *Android* tab. Set *Default Orientation* to *Landscape Left*.



[Unity Pro 4.5]

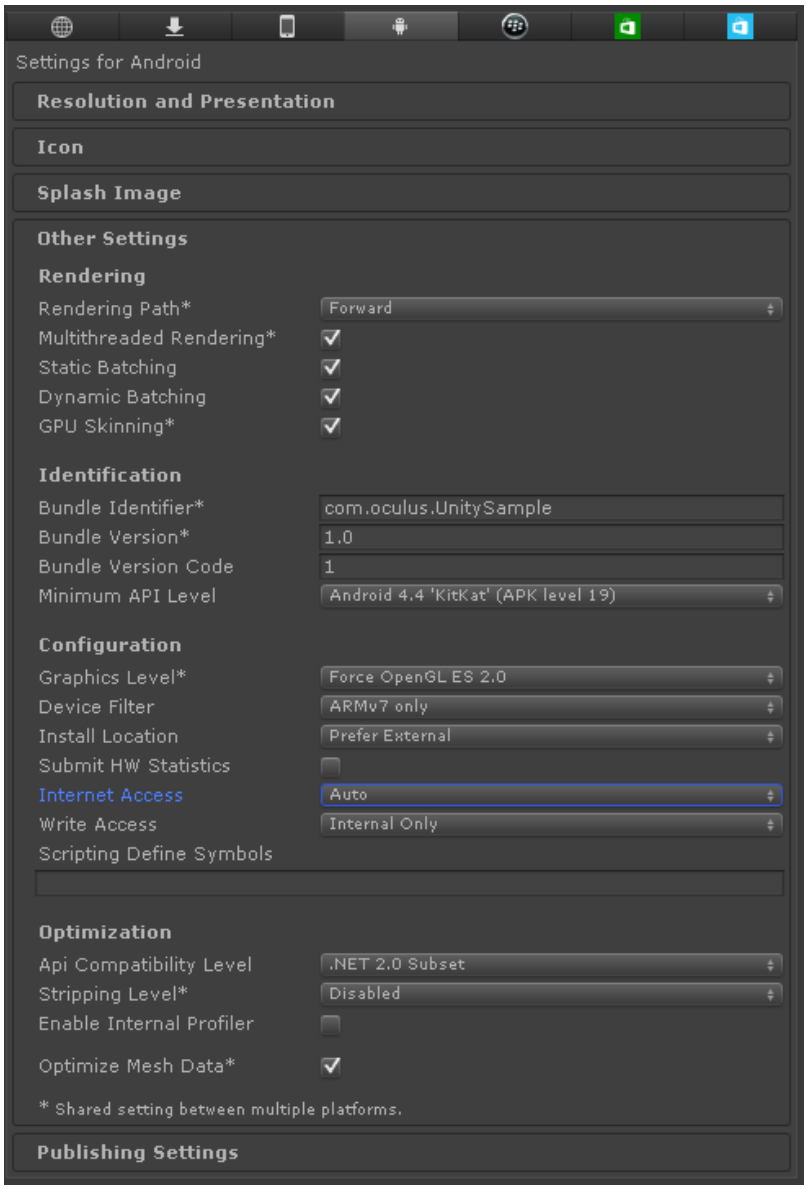
Note: The Use 24-bit Depth Buffer *option appears to be ignored for Android. A 24-bit window depth buffer always appears to be created.*

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.

Note: You may change the camera render texture depth buffer precision from 24 to 16 bits by modifying the depth precision parameter in the camera render texture creation section in OVRCameraController.cs*.*

Select the *Splash Image* section. For *Mobile Splash image*, choose a solid black texture.

While still in *Player Settings*, select *Other Settings* and make sure both *Forward Rendering* and *Multithreaded Rendering** are selected as shown below:



[Unity Pro 4.5]

Set the *Stripping Level* to the maximum level your app allows. It will reduce the size of the installed .apk file. Checking *Optimize Mesh Data* may improve rendering performance if there are unused components in your mesh data.

3.2.3 Configuring Quality Settings

Go to the *Edit* menu and choose *Project Settings*, then *Quality Settings*. 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.

[Unity Pro 4.5]

Note: Antialiasing should **not** be enabled for the main framebuffer.

You may change the camera render texture antiAliasing by modifying the parameter in the camera render texture creation section in OVRCameraController.cs. 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.

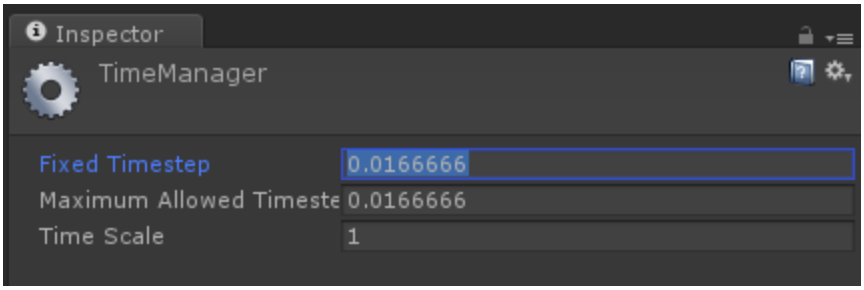
Pixel Light Count is another attribute which may significantly impact rendering performance. A model is re-rendered for each pixel light that affects it. For best performance, set *Pixel Light Count* to zero. In this case, vertex lighting will be used for all models depending on the shader(s) used.

3.2.4 Configuring Time Settings

Note: The following Time Settings advice is for applications which hold a solid 60FPS, 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 `OnFixedUpdate()` is called. The *Maximum Allowed Timestep* sets an upper bound on how long physics calculations may run.



[Unity Pro 4.5]

3.2.5 Configuring the Android Manifest File

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:

```
<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:configChanges="screenSize|orientation|keyboardHidden|keyboard">
  </activity>
</application>
<activity android:name="com.oculusvr.vrlib.PlatformActivity"
android:theme="@android:style/Theme.Black.NoTitleBar.Fullscreen" android:launchMode="singleTask"
android:screenOrientation="landscape"
android:configChanges="screenSize|orientation|keyboardHidden|keyboard">
</activity>
<uses-sdk android:minSdkVersion="19" android:targetSdkVersion="19" />
<uses-feature android:glEsVersion="0x00030000" />
<uses-permission android:name="android.permission.CAMERA" />
```

- 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.
- `PlatformActivity` represents the Universal Menu and is activated when the user long-presses the HMT button. The Universal Menu is implemented in VrLib and simply requires the activity to be included in your manifest.
- CAMERA permission is needed for the pass-through camera in the Universal Menu.
- **Do not** add the `noHistory` attribute to your manifest.

Note that submission requirements will have a few adjustments to these settings. Please refer to the submission guidelines available in our Developer Center: https://developer.oculus.com

3.2.6 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.

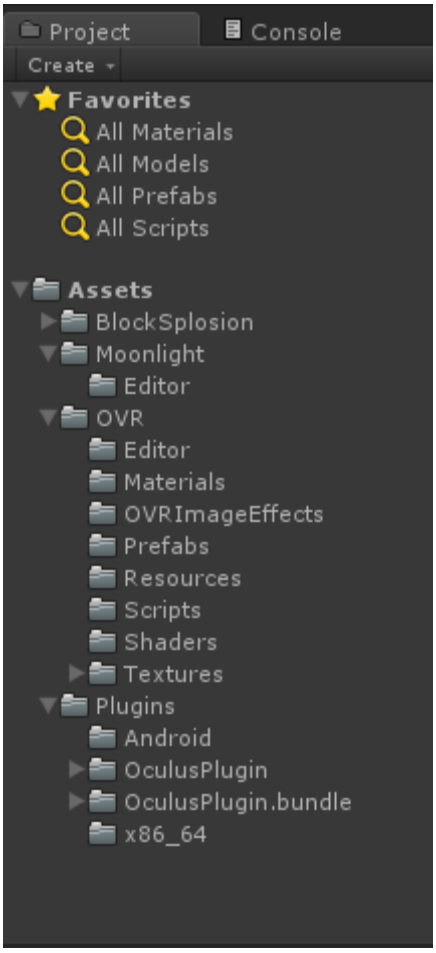
First, make sure the project settings from the steps above are saved with *File -> Save Project*.

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.

4. A Detailed Look at the Unity Integration

In general, the mobile Unity SDK projects are structured as shown below.



[Unity Pro 4.5]

The *OVR* folder contains the bulk of the Oculus SDK Unity integration code and assets in the following directories:

- **Editor**: Contains scripts that add functionality to the Unity Editor, and enhance several C# component scripts.
- **Materials**: Contains materials that are used for graphical components within the integration, such as the Magnetometer compass and the main GUI display.
- **OVRImageEffects**: Contains scripts and lens correction shaders for the PC and Mac OS X. For mobile, the lens correction is handled by TimeWarp.
- **Prefabs**: Contains the main prefabs used to provide the framework for VR into a Unity scene: OVRCameraController and OVRPlayerController.

- **Resources**: Contains prefabs and other objects that are required and instantiated by some OVR scripts, such as the magnetometer compass geometry, and the volume popup.
- **Scripts**: Contains the C# files that are used to tie the VR interface and Unity components together. Many of these scripts work together within the Prefabs. The script code interfaces to the Oculus Unity plugin.
- **Shaders**: Contains the OVRGUIShader for GUI rendering.
- **Textures**: Contains image assets required by some of the script components.

The **Moonlight** folder contains some utility classes that exist in the mobile version of the Oculus Unity SDK. In future releases of the SDK these files may be integrated into the **OVR** folder.

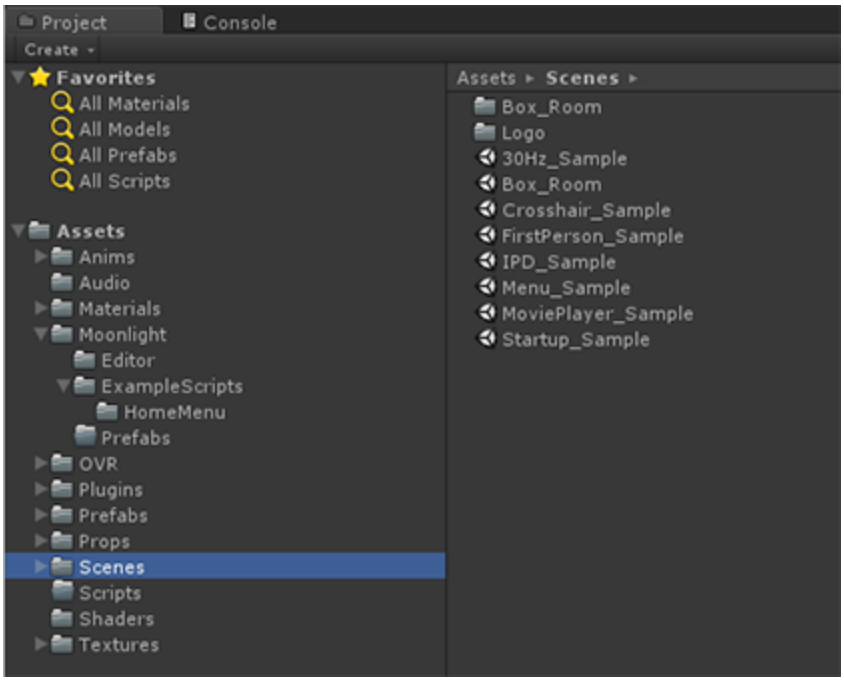
The **Plugins** folder contains the DLLs that interface to Oculus LibOVR and native Android code. The native code talks to the hardware to collect and manage sensor data from the Gear VR and also implements the Timewarp functionality. If the Plugins folder is not present directly under the Assets folder, Unity will not find the Oculus plugin and nothing will render.

The **BlockSplosion** folder contains all non-OVR scripts and assets specific to this particular project.

5. Oculus Mobile Unity Examples

Included with the Mobile SDK is an SDK Examples project which demonstrates useful scenarios and functionality.

Open the SDK Examples project in Unity by selecting *File -> Open Project*. Click *Open Other…* and choose the *SDKExamples* project folder located in your SDK install location. Click *Open*.



[Unity Pro 4.5]

You will find the following sample scenes located in Assets/Scenes:

- **30Hz_Sample** - An example of how to set the TimeWarp vsync rate to support 30Hz apps, as well as how to enable Chromatic Aberration Correction and Monoscopic Rendering for Android. For more information on 30Hz TimeWarp and Chromatic Aberration Correction for Android, please review the *TimeWarp* Technical Note.
- **Box_Room** - A simple box room for testing.
- **Crosshair_Sample** - An example of how to use a 3D cursor in the world with three different modes.
- **FirstPerson_Sample** - An example of how to attach avatar geometry to the OVRPlayerController.
- **GlobalMenu_Sample** - An example demonstrating Back Key long-press action and the Universal Menu. Additionally demonstrates a gaze cursor with trail. For more

information on Interface Guidelines and requirements, please review the following documents: *Interface Guidelines* and *Universal Menu*.

- **Menu_Sample** - An example demonstrating a simple in-game menu activated by Back Key short-press action. The menu also uses the Battery Level API for displaying the current battery level and temperature.
- **MoviePlayer_Sample** - An example demonstrating basic in-game video using Android MediaPlayer.
- **SaveState_Sample** - An example demonstrating saving the state of the game on pause and loading it on resume. Click on the objects in the scene to change their color. When you run the scene again, the objects should be in the color you had selected before exiting.
- **Startup_Sample** - An example of a quick, comfortable VR app loading experience utilizing a black splash screen, VR enabled logo scene, and an async main level load. For more information on interface guidelines, please review the following documents: *Interface Guidelines* and *Universal Menu*.

The example scripts are located in Assets/Moonlight/:

- **Crosshair3D.cs** - Detailed code for how to create judder-free crosshairs tied to the camera view.
- **StartupSample.cs** - Example code for loading a minimal-scene on startup while loading the main scene in the background.
- **TimeWarp30HzSample.cs** - Example code for setting up TimeWarp to support 30Hz apps as well as toggling Chromatic Aberration Correction and Monoscopic Rendering on and off.
- **HomeMenu.cs** - Example code for an animated menu.
- **HomeButton.cs** - Example code which provides button commands (used in conjunction with HomeMenu.cs).
- **HomeBattery.cs** - Example code for using the SDK Battery Level API and interactively modifying a visual indicator.
- **MoviePlayerSample.cs** - Example code and documentation for how to play an in-game video on a textured quad using Android MediaPlayer.
- **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 here: *TimeWarp*.
- **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.
- **OVRScreenFade.cs** - Drop-in component for issuing screen fades on level loads.
- **OVRWaitCursor.cs** - Helper component for auto-rotating a wait cursor.

- **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 the following documents: *Interface Guidelines* and *Universal Menu*.

6. DK2 Compatibility

While this revision of the mobile Unity SDK fully supports the DK1, the DK2 does have some support.

In order to use the DK2 with this SDK, you will need to do the following:

- Open the Oculus Config Utility
- Go to *Tools* -> *Service* and select *Pause*.

The Service needs to be paused in order for head-tracking to work, otherwise the Service will grab the sensor full-time.

7. Known Issues

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

NGUI with Multi-threaded Rendering

When multithreaded rendering is enabled, NGUI geometry appears stretched/distorted and may flicker.

This issue has been fixed in Unity 4.5.3.f3.

Random Crashes with Dynamic Batching Enabled

This issue has been fixed in Unity 4.5.5.p1.

DK1 HMDs do not display correctly or have double vision

This has been known to occur at certain resolutions when mirroring a display and while the current platform is set to *Android*.

Switch the platform back to **PC, Mac and Linux Standalone** if you are testing with a DK1 HMD.

Mac OS X Game View Render

The rendered game view in the editor currently draws over the top of the Scene and Game tabs.

The issue has been reported to Unity.

Flipped Rendering in Editor with UseCameraTexture set to false

The rendered game view in the editor renders flipped when *Use Camera Texture* on the camera controller is set to *false* and *Android* is selected as the target platform. This only occurs when using the PC editor.

OnGui Calls Fail to Render

The drawing of the screen edge vignettes and flushing of commands with the camera occurs in
OnPostRender. If any other post rendering happens after this, at a minimum performance will be badly affected due to a double buffer flush/resolve, but generally is not expected to work at all.

It may be possible to interject the post effect rendering using the OnCustomPostRenderEventHandler in OVRCamera.cs. OVRScreenFade uses this approach.

There is no workaround at this time.

8. Troubleshooting

Game scene is missing or just renders the background color when pressing Play in Unity.

Check the [Project Path]/Assets/Plugins/ folder and make sure that the Oculus plugins have not moved. See the Unity console for additional information.

After importing the latest Oculus Unity Integration package, your game is generating exceptions.

It is possible there were changes made to the OVR framework that require you to update your camera prefabs. The easiest way to do this is to compare the changes between the Camera Controller preb you were using (*OVRCameraController* or *OVRPlayerController*) and the new one and compare changes.

After importing the latest Oculus Unity Integration package your existing OVRPlayerController transform is changed.

It is possible there were changes made to the OVR framework that may cause the *OVRPlayerController* transform to be swapped with the child *OVRCameraController* transform. Swapping the values back should fix the issue.

After importing the latest Oculus Unity Integration package the rendering is corrupted.

We require the orientation to be landscape. Check that your *defaultOrientation* in *Player Settings* is set to *Landscape Left*.

After importing the latest Oculus Unity Integration package the app does not launch as a VR app.

Ensure you have administrator rights to the system you are installing the integration to.

Issues with updating to the latest Oculus Unity Integration with Team Licensing and Perforce Integration enabled.

If you have Team Licensing and Perforce Integration enabled, you may need to check out the OVR and Plugins folders manually before importing the updated unity package.

Building Application for Android may fail with Zipalign Error.

If you have build failures with the following error about zipalign:

```
Error building Player: Win32Exception:
ApplicationName='D:/Android/sdk\tools\zipalign.exe', CommandLine='4
"C:\Users\Username\Documents\New Unity Project
1\Temp/StagingArea/Package_unaligned.apk" "C:\Users\Username\Documents\New Unity
Project 1\Temp/StagingArea/Package.apk"', CurrentDirectory='Temp/StagingArea'
```

This can be fixed by copying the zipalign.exe from the API level you're building for into the sdk\tools directory. To find this, look in the build-tools directory in your SDK installation, in the folder for the API level you're building for. For example, if you're targeting API level 19, the directory is sdk\build-tools\19.1.0. Copy zipalign.exe into sdk\tools and try building your project again.

Last Update: Nov 10, 2014