Lab 2b: Setting up Your project for VR

Technical Requirements

To implement the projects and exercises in this lab, you will need the following:

- A PC or Mac capable of running Unity 2021.3.18 LTS or later, along with an internet connection to download files.
- A VR headset supported by the Unity XR platform.

To get started with your Unity project ready for VR, you should identify the VR platform(s) and device(s) you initially plan to target.

Choosing your target VR Platform and Toolkits

You need to manage the VR plugins you will need and use additional toolkit(s) for developing the project. Generally, your Unity VR project will need to include the following:

- Select a **Target Platform** for your builds
- Install an **XR Plugin** that drives the VR devices.
- Within your scene, you will include a VR-enabled camera rig for tracking head and hand locations in 3D space.

Device	Dev Platform	Target Platform	VR Runtime	Official XR Plugin	Optional Device Toolkit
Oculus Quest	Windows, MacOS X	Android	Oculus Quest	Oculus	OVR

Enabling Virtual Reality for Your Platform

Create a project using 3D template. We will now enable the project and scene so that it runs in virtual reality. These first couple of tasks are similar, regardless of which device you are targeting:

- Setting the **target platform** for your project
- builds Installing the **XR plugin** for our device
- Installing the XR Interaction (XRI) Toolkit **(package)**
- Creating a VR enabled **XR Rig** camera rig

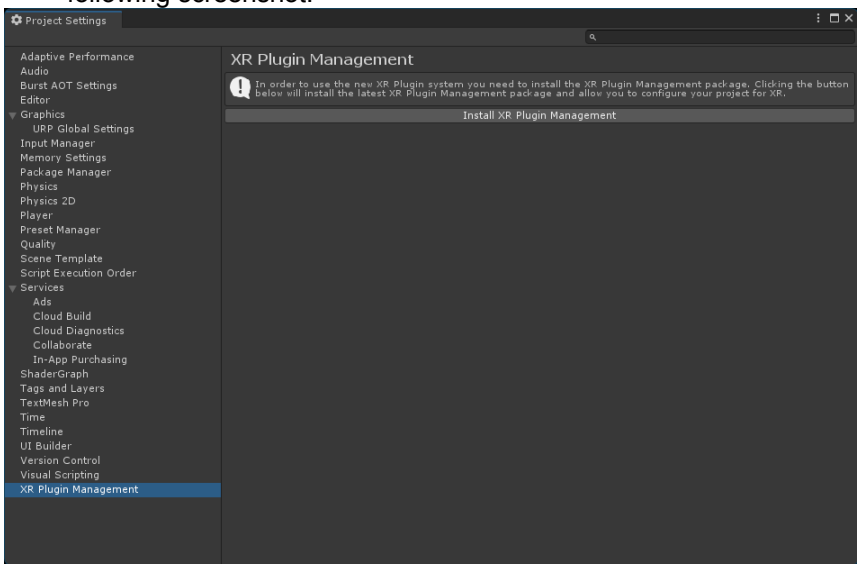
Setting the target platform

1. Since you are building for Oculus Quest, choose **Android**.
2. Then, click **Switch Platform**.

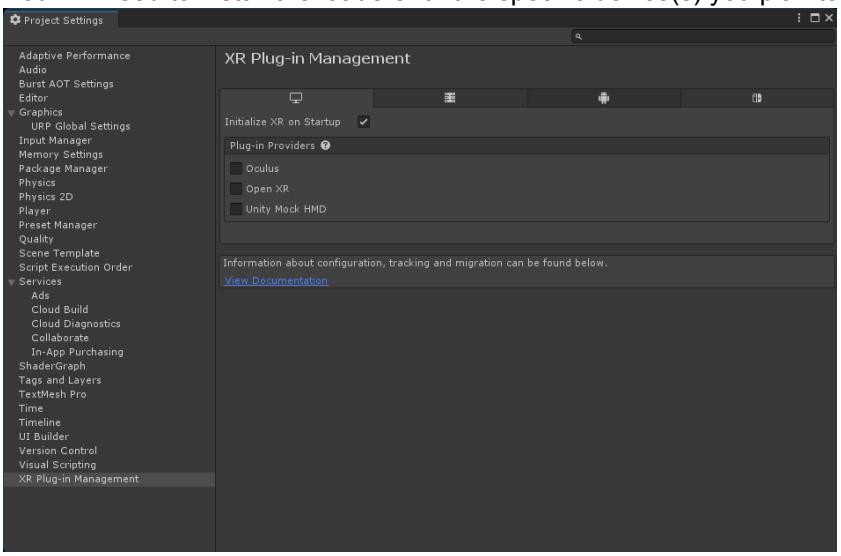
Installing XR Plugin Management

When VR is enabled in your Unity project, it renders stereoscopic camera views and runs on a VR headset.

1. Open the **Project Settings** window (**Edit | Project Settings**).
2. Select **XR Plugin Management** from the left tab menu.
3. If necessary, click **Install XR Plugin Management**, as shown in the following screenshot:

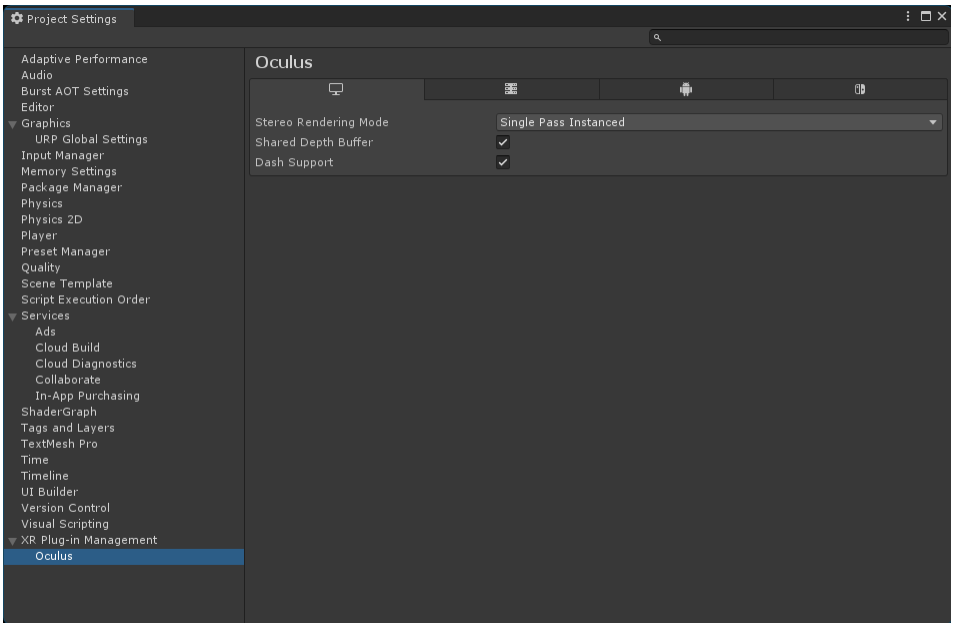


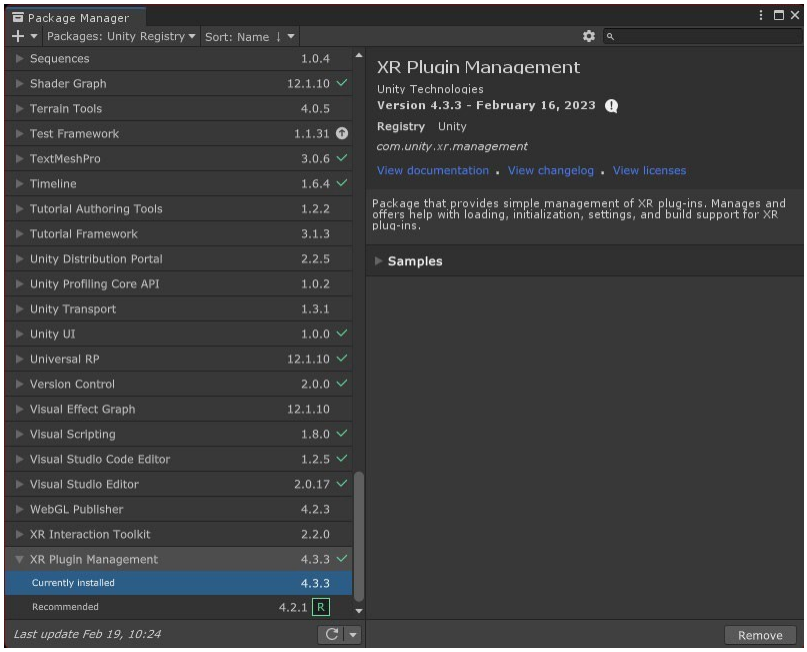
You will need to install the loaders for the specific device(s) you plan to target with this project.



Once one or more of the desired plugins have been installed, the plugin will be added to the list in the left-hand menu, under **XR Plug-in Management**.

In the following screenshot, for example, the Oculus plugin is installed and you can see that the Oculus settings are now available:



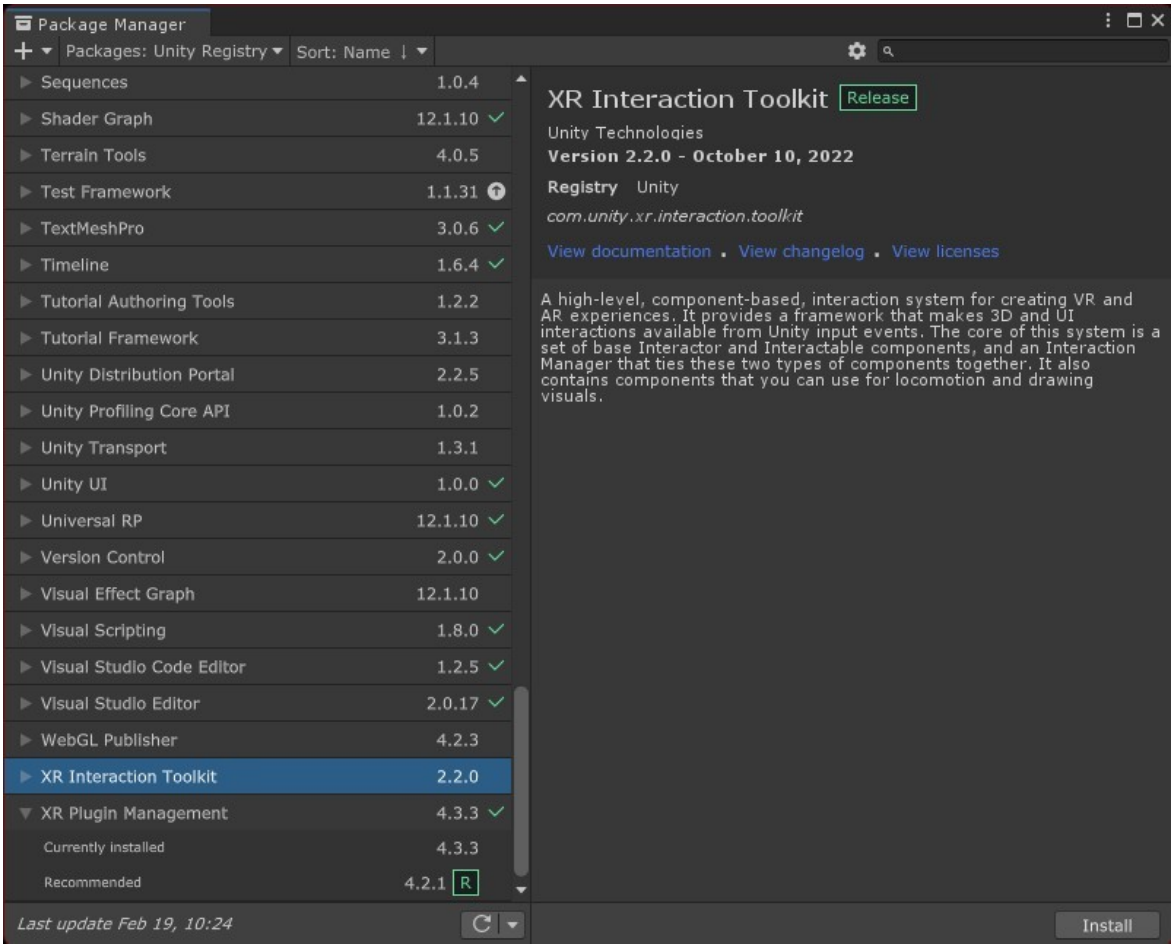


Installing the XR Interaction Toolkit

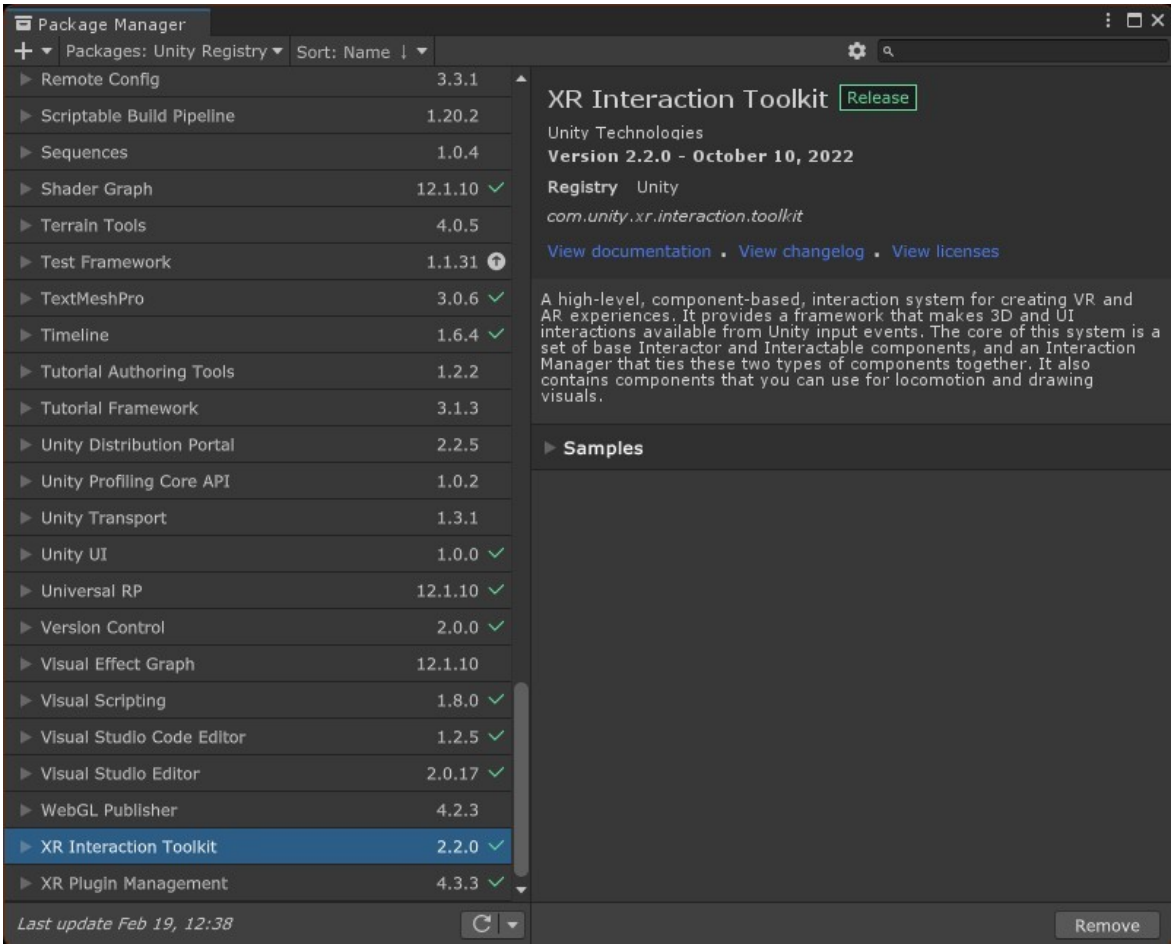
We are going to be using Unity's new **XRI Toolkit** in the projects throughout this module.

The XR Interaction Toolkit can be installed using **Package Manager**. Follow these steps to install it in your project:

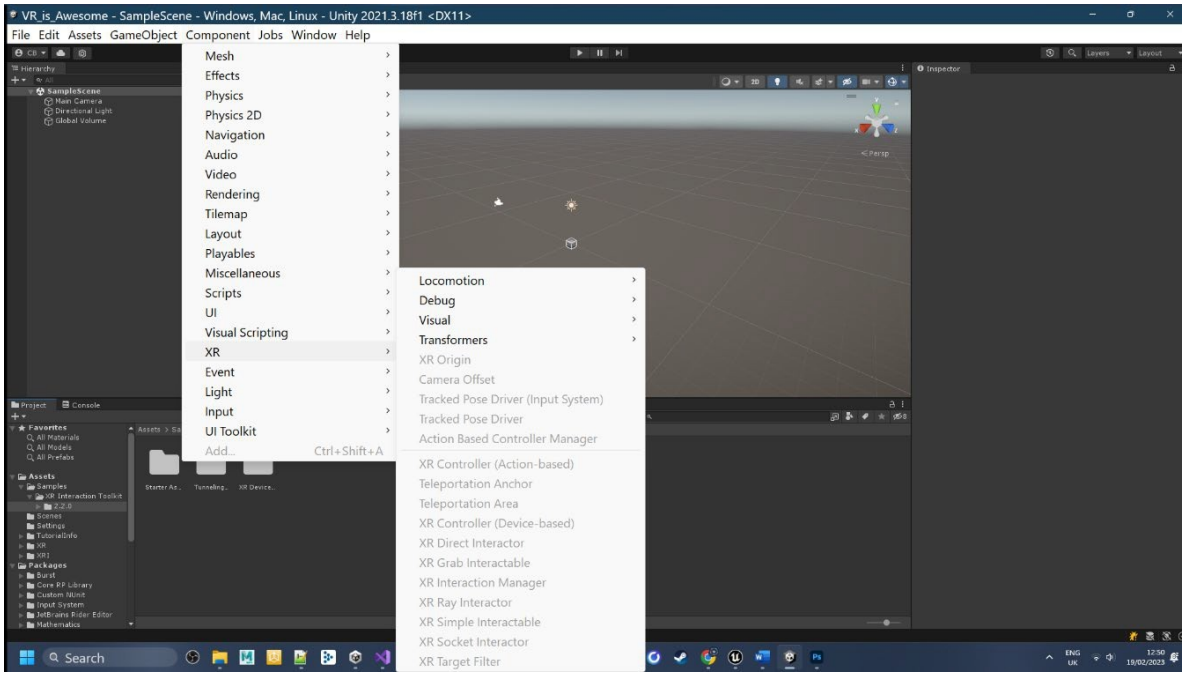
1. Open **Package Manager** (**Window | Package Manager**).
2. Filter the list to **All Packages** (Use the drop-down list at the top-left of the window). At the time of writing, XRI is still in preview, so you may also need to select **Show Preview Packages** from the **Advanced** dropdown menu.
3. Type xr interaction in the search area and, with the package selected (XR Interaction Toolkit), press Install.



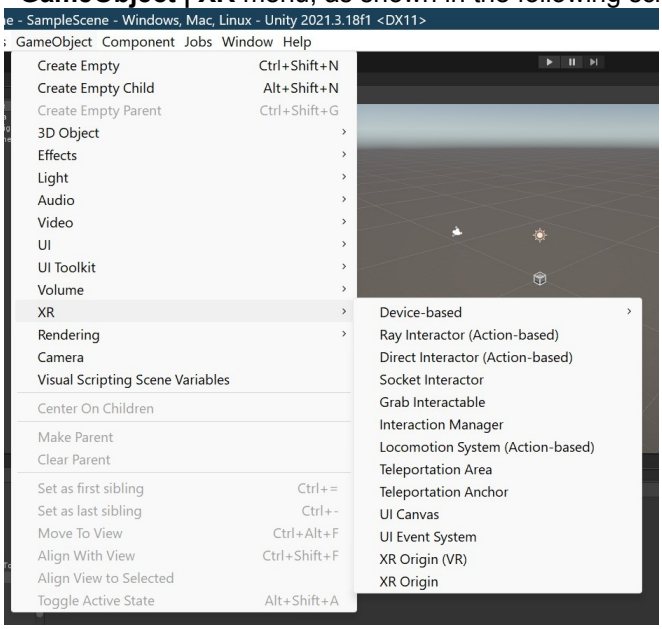
The XR Interaction Toolkit package is shown in the following screenshot:



With XRI installed, you now have access to its components through the main menu bar, that is, **Component | XR**.



There are some pre-made game objects you can readily add to your scenes through the main **GameObject | XR** menu, as shown in the following screenshot:



Migrating a Simple Scene

To migrate a simple Scene, follow the steps below.

1. Create an instance of the **XR Rig Prefab**.

Migrating a Camera that was previously at the root of the Scene hierarchy allows you to swap the existing Camera to an XR Rig by instancing the XR Rig Prefab that this package contains.

You can find this Prefab in the Packages/XR Legacy Input Helpers/Prefabs folder.

2. Drag the Prefab into your Scene hierarchy to create a new instance of it.

3. Move the XR Rig to match the Camera's location.

4. Now that you have an instance of the XR Rig Prefab, you need to configure the XR Rig GameObject so the resulting Camera positions will be correct when your application starts.

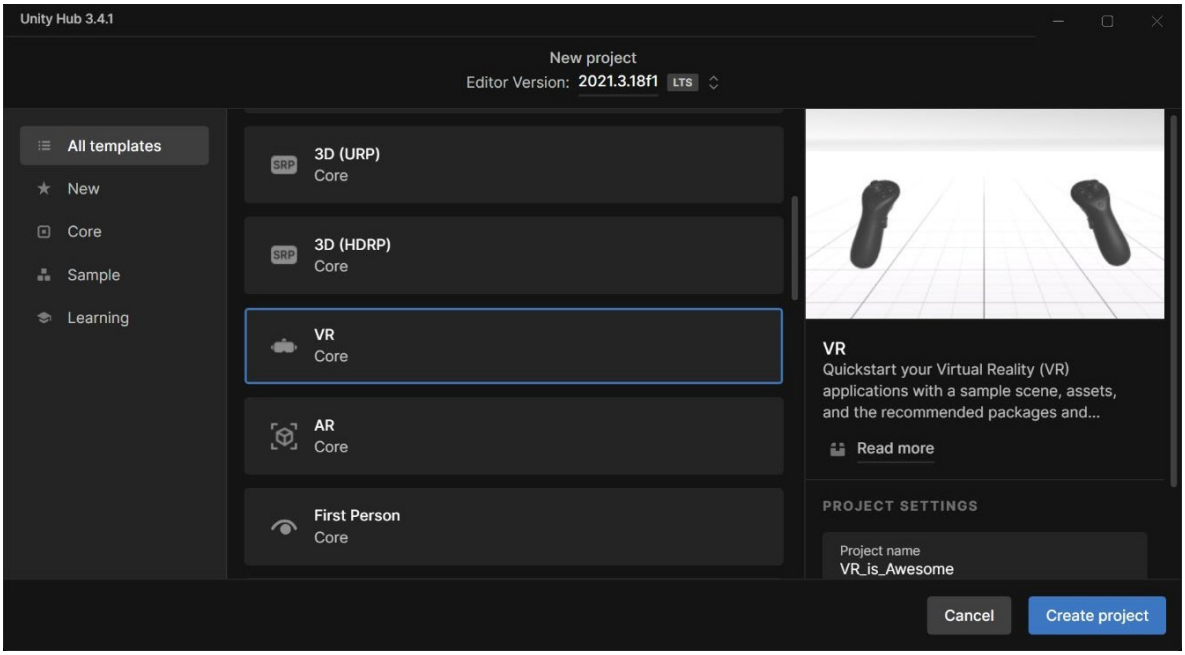
5. Change the position and rotation of the XR Rig GameObject so it matches the position of the current Main Camera. If you're applying a scale transformation to the Main Camera, make sure that you also apply this scale to the XR Rig GameObject. If you scale the XR Rig, it's highly recommended to use a uniform scale across all three axes.

6. Replicate the position change, if necessary. If the Main Camera in your non-migrated Scene is above the floor (its Y component is > 0.0f), you need to replicate this position change. You can do this in a few different ways, depending on what the original position the change was intended to replicate.

7. Add an interaction manager Game object.
8. Remove the Main Camera

This adds two new objects to the root of your scene hierarchy: **XR Interaction Manager** and **XR Rig**. It also removes the default Main Camera from the Hierarchy, as XR Rig has its own Main Camera child object.

If this is not working, the easiest thing is to do is start a NEW Unity Project Using the VR template which has everything you need for a basic set-up and migrate in the previous created scene if you have any.



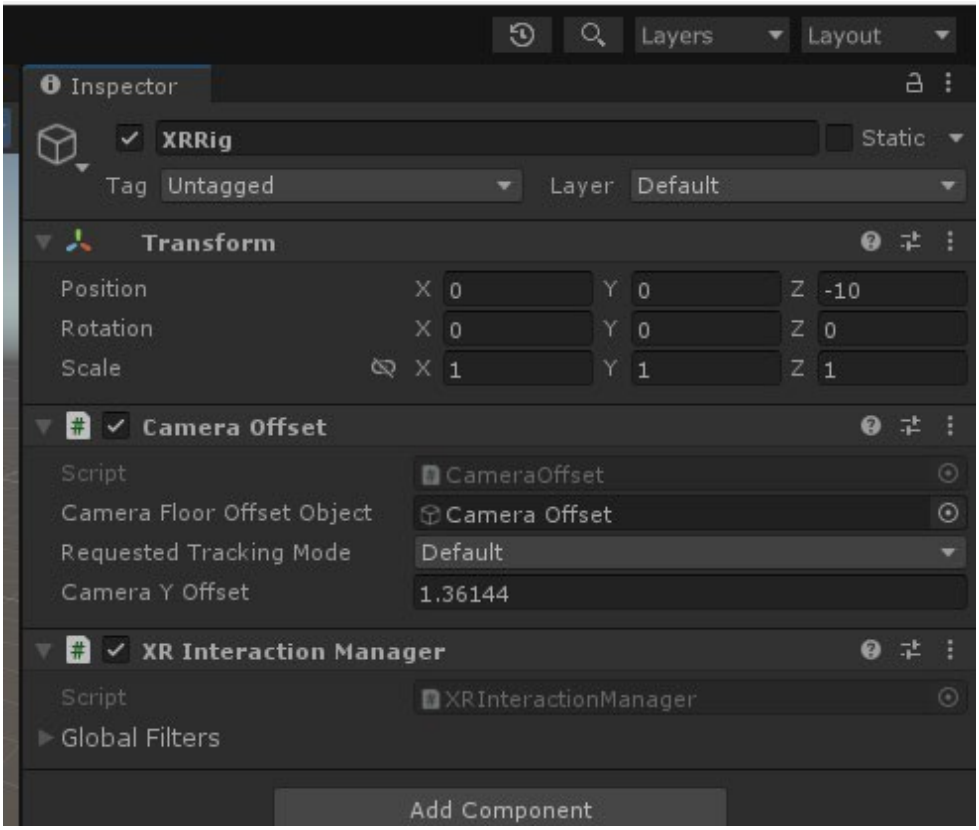
Exploring the XR Rig objects and Components

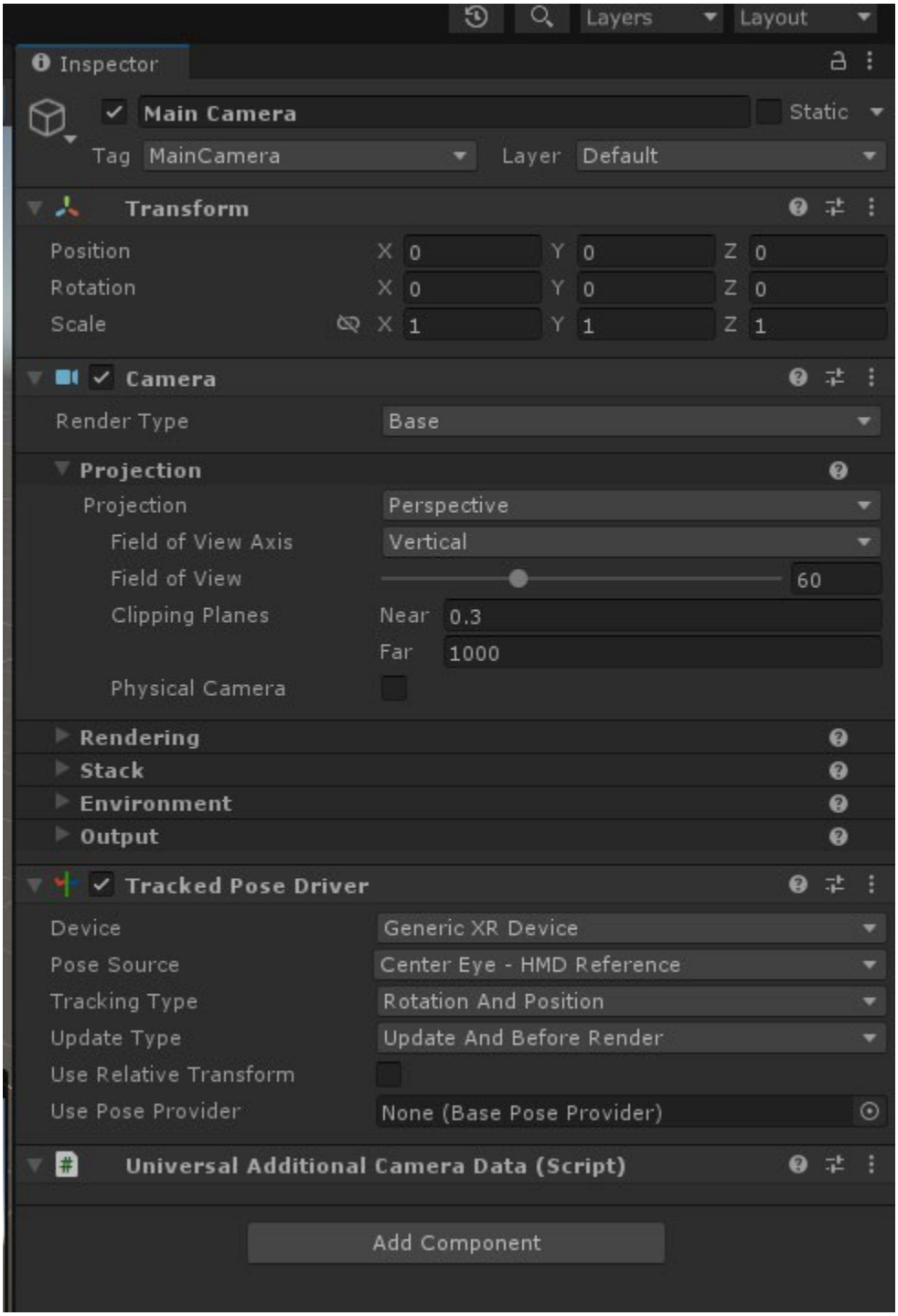
Select the **XR Interaction Manager** object.

In the **Inspector** window, you can see that it has a corresponding **XR Interaction Manager** component. XRI requires you have one **XR Interaction Manager** component in the scene so that you can manage communications between the *interactors* and *interactables* in the scene

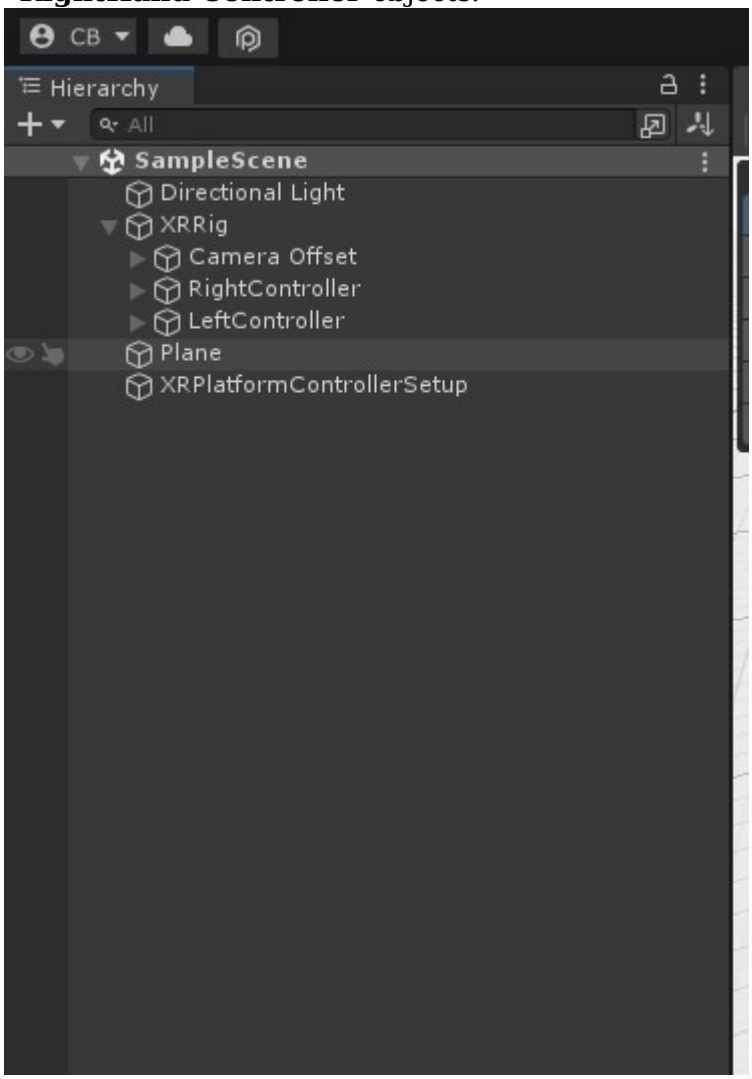
The scene now also includes an **XR Rig** object, which has a child hierarchy of GameObjects. As shown in the following screenshot, it has an **XR Rig** component,

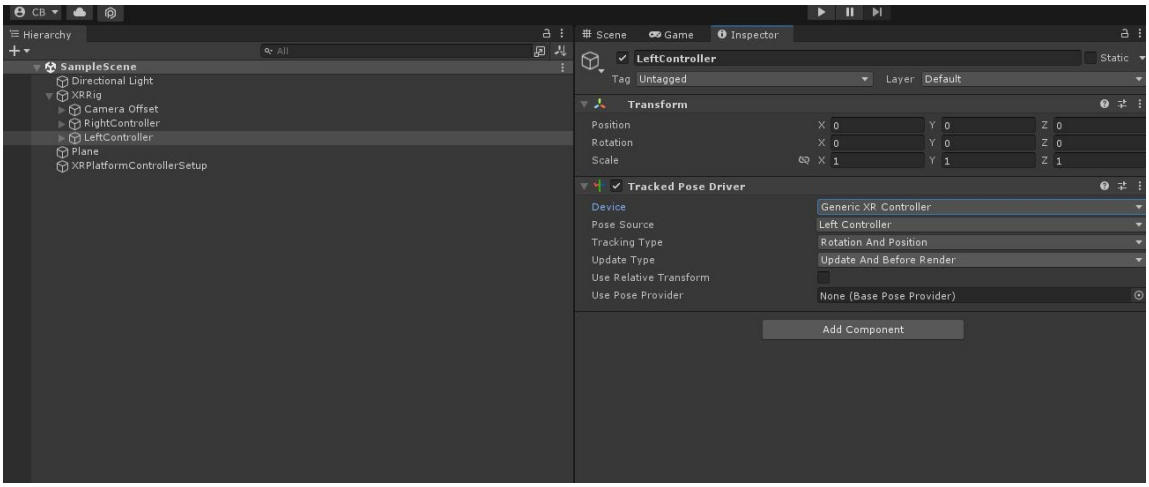
This defines the camera's (player's head) default height above the floor and is needed when we're not using **Floor** as the tracking origin:


Layers
Layout
Inspector
XRRig
Static
Tag
Untagged
Layer
Default
Transform
Position
X 0
Y 0
Z -10
Rotation
X 0
Y 0
Z 0
Scale
X 1
Y 1
Z 1
Camera Offset
Script
CameraOffset
Camera Floor Offset Object
Camera Offset
Requested Tracking Mode
Default
Camera Y Offset
1.36144
XR Interaction Manager
Script
XRInteractionManager
Global Filters
Add Component

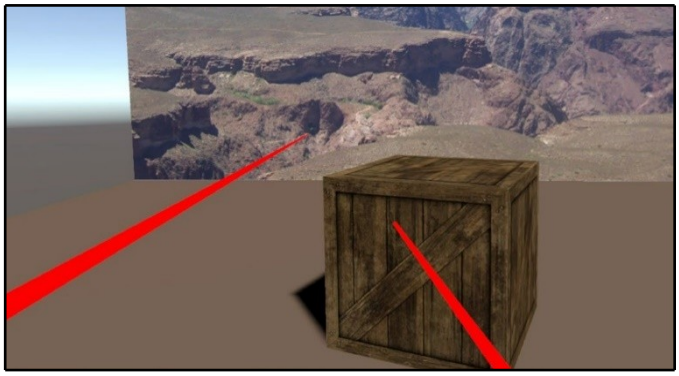

Layers
Layout
Inspector
Main Camera
Static
Tag MainCamera
Layer Default
Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Camera
Render Type Base
Projection
Projection Perspective
Field of View Axis Vertical
Field of View 60
Clipping Planes Near 0.3
Far 1000
Physical Camera
Rendering
Stack
Environment
Output
Tracked Pose Driver
Device Generic XR Device
Pose Source Center Eye - HMD Reference
Tracking Type Rotation And Position
Update Type Update And Before Render
Use Relative Transform
Use Pose Provider None (Base Pose Provider)
Universal Additional Camera Data (Script)
Add Component

The children of **Camera Offset** are the **Main Camera**, **LeftHand Controller**, and
RightHand Controller objects.




Hierarchy
SampleScene
Directional Light
XRRig
Camera Offset
RightController
LeftController
Plane
XRPlatformControllerSetup
Scene
Game
Inspector
LeftController
Static
Tag Untagged
Layer Default
Transform
Position
Rotation
Scale
Tracked Pose Driver
Device
Generic XR Controller
Pose Source
Left Controller
Tracking Type
Rotation And Position
Update Type
Update And Before Render
Use Relative Transform
Use Pose Provider
None (Base Pose Provider)
Add Component

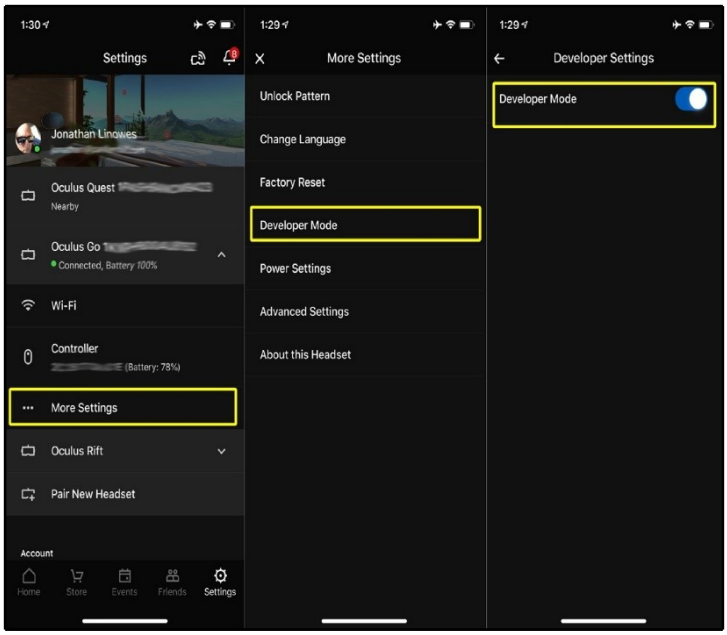
The following screenshot shows the built app running in VR with the default laser raycast hands:



Building for Oculus Quest

Ensure your device is in **Developer Mode**, as follows:

1. Open the **Oculus** companion app on the phone connected to the device.
2. Go to the **Settings** menu (lower-right corner).
3. Select the device (for example, Oculus Quest).
4. Select **More Settings | Developer Mode** and switch it on, as shown in the following screenshot:



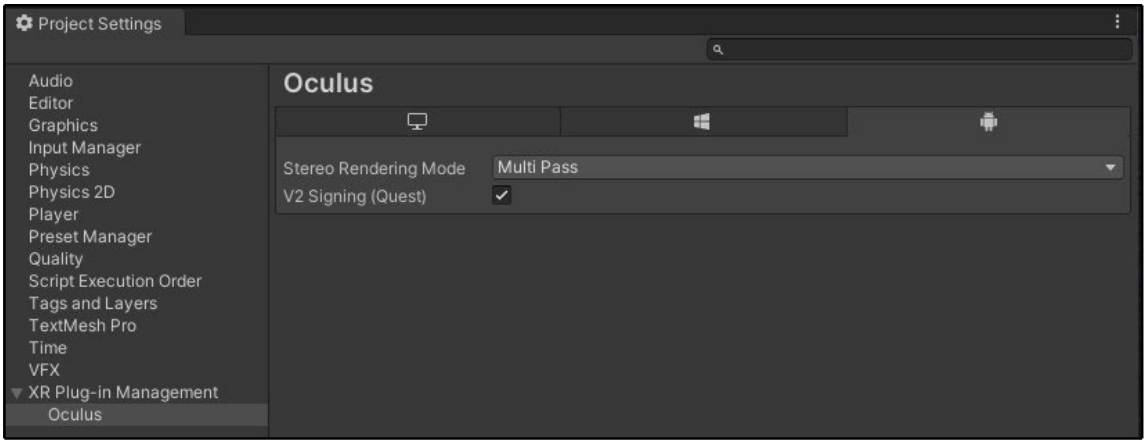
Setting up for Oculus mobile VR

To target Android-based Oculus mobile VR, configure your Unity **Build Settings** so that they target the **Android** platform via **File | Build Settings... | Android | Switch**

Platform. Next, if it's not already installed, add the **Oculus XR Plugin**, as follows:

1. Open **XR Plug-in Management** in **Project Settings** using **Edit | Project Settings| XR Plugin Management.**
2. Click the **Android** tab.
3. In the **Plug-in Providers** list, check the **Oculus** checkbox. The Oculus SDK will be installed.
4. You'll now see an **Oculus** option in the left tab menu. Select it.
5. Press **Create** to create a serialized instance of the settings data in order to modify the default Oculus settings.
6. Check the **V2 Signing** checkbox, especially if you're targeting Quest.

The Oculus settings for Android are shown in the following screenshot:



To preview the scene in your device, you could use **OVR Scene Quick Preview**, which is included with the Oculus Integration Toolkit.

Other Android and optimization settings

To build for an Android device, you also need to configure other **Player Settings**. Some settings are required for it to run, while others are recommended for it to run better:

1. At the top, set **Company Name** and **Product Name**. The product defaults to the project name that was used when it was created via Unity Hub.
2. In **Other Settings | Package Name**, set a unique app ID using a Java package name format. It defaults as a composite of the company and product names; for example, mine is com.UVRP3.VR_is_awesome.
3. The minimum API level will vary, depending on your target device. Oculus Quest requires **Minimum API Level: Android 6.0 Marshmellow (API Level 23)**. Oculus Go requires a minimum of **Android 5.0 Lollipop (API Level 21)**.

Some of these current recommended settings include the following:

- **Build Settings | Texture Compress: ASTC**. **Player | Other Settings | Color Space: Linear**.
- **Auto Graphics:** Unchecked. **OpenGLES3** should be the first in the list.

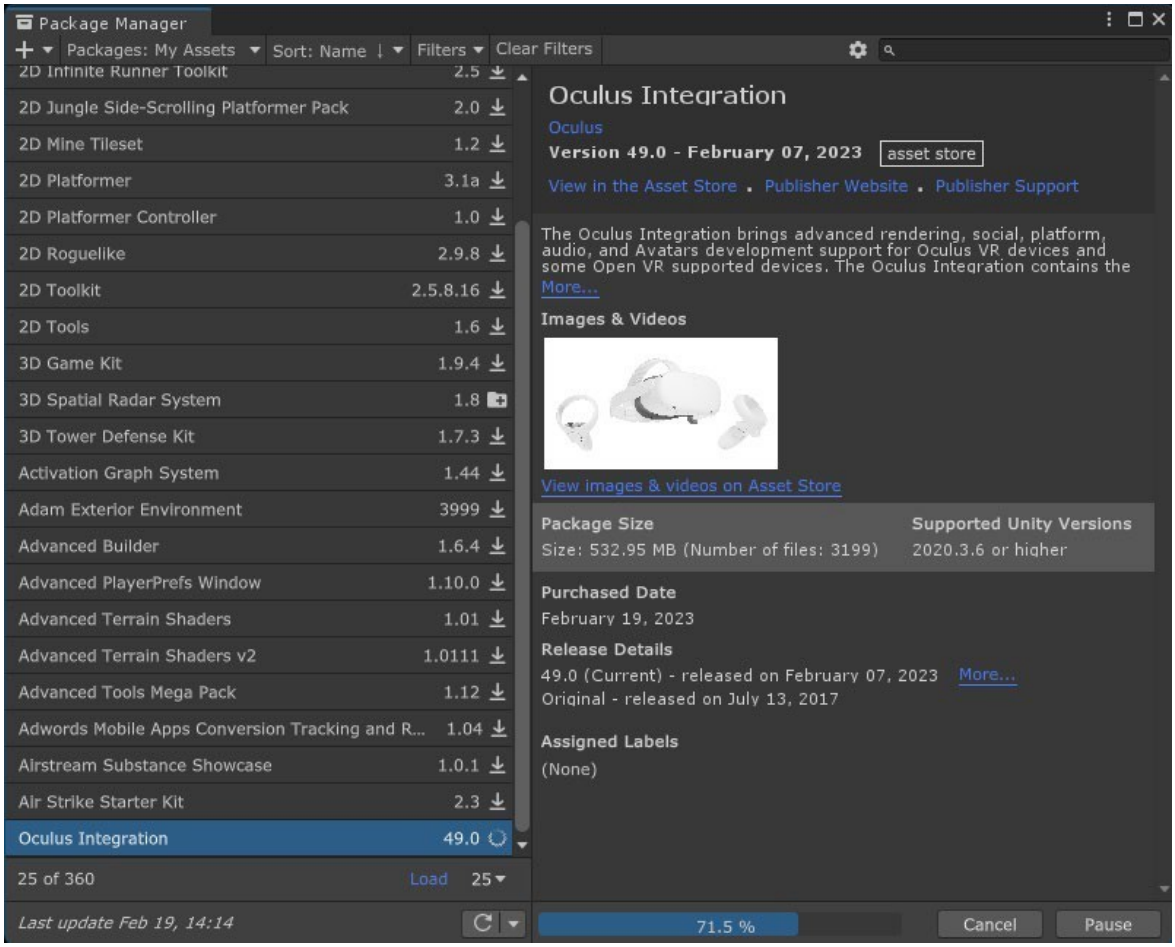
- **Vulcan** should be the second, if at all, but not on Go.
- **Itithreaded Rendering**: Checked.
- Also, review the recommended **Graphics pipeline** and **Quality** settings.

Click **Build And Run**.

The SDK we've installed is sufficient for the projects in this book. However, Oculus provides a rich collection of optional tools in the **Oculus Integration toolkit**.

Installing the Oculus Integration toolkit

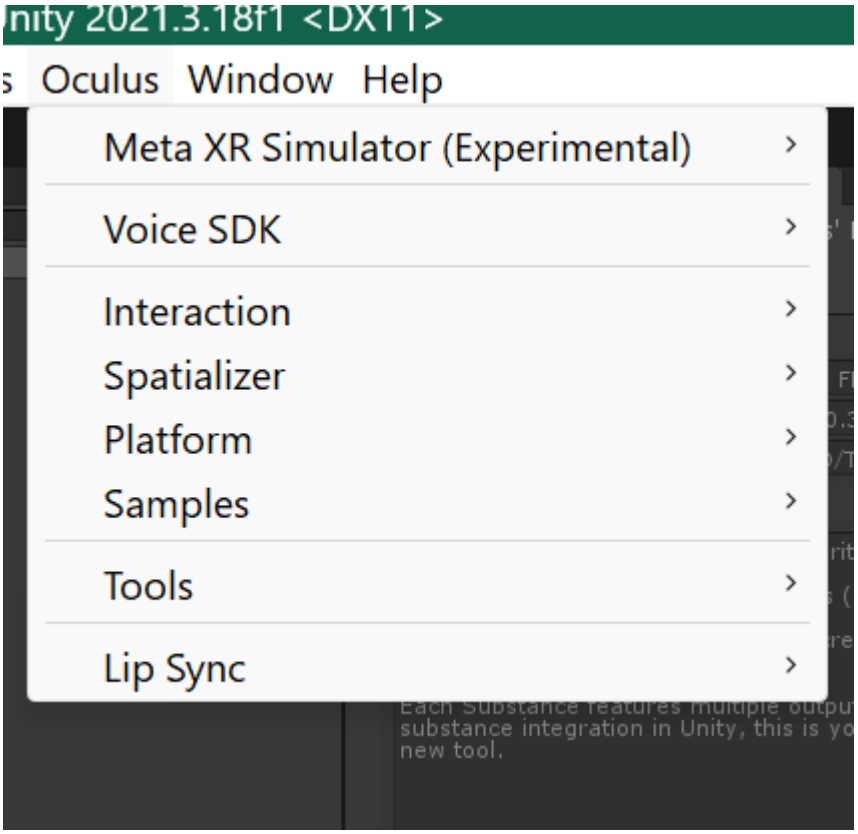
To install the Oculus Integration toolkit from the Asset Store, follow these steps:



In your web browser, go to https://assetstore.unity.com/packages/tools/ integration/oculus-integration-82022.

1. Click **Open In Unity** and **Install** it in your project.
2. If prompted, also allow updates to the **OVRPlugin for the Unity Editor** or other packages.
3. Since we're using the Universal Render Pipeline, you should convert the imported materials using **Edit | Render Pipeline | Universal Render Pipeline | Upgrade**

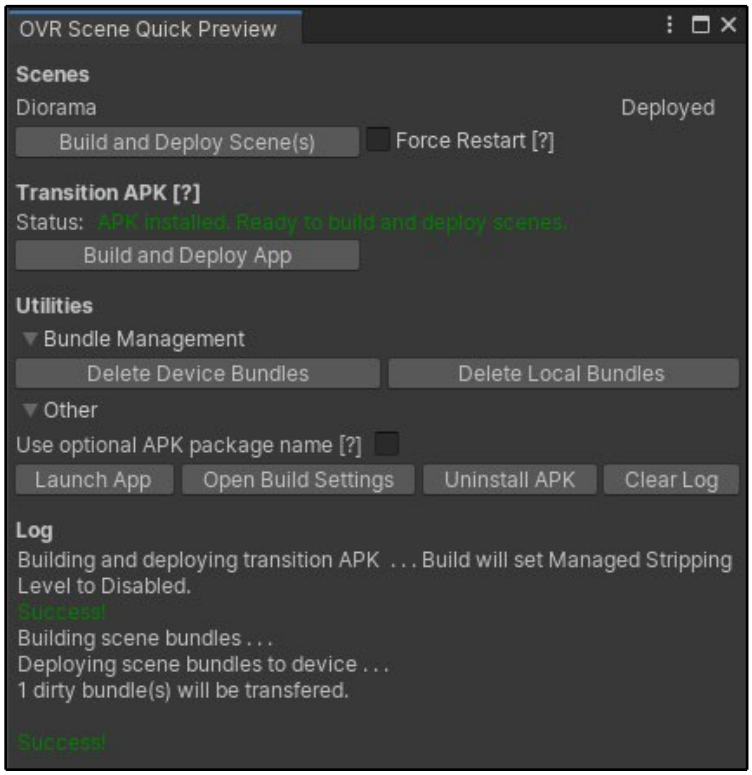
Project Materials.



Press the **OVR Scene Quick Preview** button to play the scene in your device.

You may be prompted to install the OVR Transition APK. The Transition APK helps Unity launch your working scene in the device.

After doing this, press **Build And Deploy Scene** to quick- build the scene for preview in the HMD. The dialog box is shown in the following screenshot:



To build and run this, press the **OVR Build APK And Run** option.

In your own projects, you may choose to replace our **XR Rig** camera rig with the OVRCameraRig prefab provided in the Oculus Integration package, which can found in the Assets/Oculus/VR/Prefabs/ folder and is used in their various example scenes.

Using adb

When you build a project from Unity, it creates an .apk file and installs it on the mobile device connected to your development machine. If you already have an .apk file, you can install it manually using the Android adb command-line tool.

The **Android Debug Bridge (adb)** is an important command-line tool that you should get to know so that you can communicate with an Android device for side-loading apps and debugging. It's included with the Android SDK. The full documentation can be found at https://developer.android.com/studio/command-line/adb. If you do not have the adb command on your system, you can install it as follows:

1. Download **Android Platform Tools** from https://developer.android.com/ studio/releases/platform-tools.
2. Unzip the folder.
3. Navigate to the folder containing adb.exe.
4. Open the Terminal/Command Prompt at this folder. For example, in Windows Explorer, you can type cmd into the address bar.

I like to move the unzipped platform-tools/ folder into my /Program Files/ folder and add it to the Windows environment PATH.

Now, when you type adb devices at the Command Prompt, you should see the connected Android device. To install the APK file built from Unity, use the adb install [path to apk file] command. Other useful adb commands include the following:

- adb help: Lists the available adb commands
- adb devices: Lists the currently connected devices
- adb install [path to apk file]: Sideload installs an APK package on your device
- adb install -r [path to apk file]: Installs and replaces a package
- adb shell pm list packages -3: Lists the packages already installed on the device
- adb uninstall [packagename]: Removes a package from the device; for example, com.UVRP3.VR_is_Awesome

If you have trouble running adb install, you may need to add the **Oculus ADB Drivers**. Follow these steps to do so:

1. Download the zip file containing the driver from https://developer.oculus. com/downloads/package/oculus-adb-drivers/.
2. Unzip the file.
3. Right-click on the .inf file and select **Install**.

Adb can also be used to access the Android device logs, which are very useful when you're debugging:

- adb logcat -v time -s Unity >logs.txt: Streams the logs, filtered for Unity messages, and pipes them to the local logs.txt file. Run this command and then start your app.
- adb logcat -d -v time -s Unity >logs.txt: Dumps the most recent Unity log messages into logs.txt and exits.
- adb logcat -c: Clears the logs on the device.