→ E-BOOK

Build XR with Unity

Meta XR



1st Edition

© 2026 Unity Technologies

Unity®

Content

Introduction **5**

Unity and Meta. 6

Who’s this guide for? 6

Meta VR Glasses 6

What are Meta VR Glasses? 7

Capabilities to go deeper 8

VR or MR application 8

Controls 8

Use cases. 9

Unity’s XR Interaction Toolkit **10**

SDKs for XR development 10

OpenXR 12

Editor version and modules. 12

Getting started in VR with Unity’s VR template 13

Running the template on a device. 15

Meta Horizon Link for Windows 19

Key XRI components **20**

1. XR Origin: Move in the virtual world 21

Camera 21

XR Input Modality Manager / Controllers / Hand 21

Locomotion with XR Body Transformer 24

2. Teleportation area / anchor. 25

3. XR Grab Interactable 26

4. XR Knob 27

5. Spatial UI 27

Event System / XR UI Input Module Component 28

Tracked Device Graphic Raycaster. 28

6. Video playback 28

Develop for Meta VR Glasses 29

Meta Quest Build Profile 31

OpenXR support. 32

Requirements 33

Support eye and hand tracking for
Look and Pinch input 33

Unity's gaze and hands extensions 33

Spatial audio. 35

Building a Hello World VR game for hand tracking 36

Step 1: Configure the Editor for VR development. 37

Step 2: Initialize a new scene and add
the XR Origin prefab 40

Step 3: Set up Environment Geometry
and basic interactivity 41

Step 4: Configure Socket Interactors 43

Step 5: Implement teleportation and movement 45

Step 6: Test locally with the XR Interaction Simulator . . . 46

Step 7: Build and deploy to VR hardware 46

Building with Unity CLI and AI 48

How it works 49

Unlocking agentic workflows 49

Unity official plugins 50

Optimize graphics for Meta VR Glasses 51

Configure foveated rendering 52

Gaze-based foveated rendering 53

Enable subsampled layout. 54

Quad views. 54

Shader optimizations for Meta XR 55

Configure Application SpaceWarp 55

Enable dynamic resolution 55

Enable Adaptive Performance. 56

Meta SDKs 57

Meta utilities 60

Meta XR Simulator 60

Meta XR Operator 62

How it works 63

What can your agent do? 63

Getting started in Unity 65

Meta Quest Developer Hub 65

More resources 67

Step-by-step video tutorials 67

Unity CLI resources 68

Meta XR Operator resources. 68

More samples. 68

Unity Learn. 72



Introduction

Embark on journeys through virtual realms, teleport between dimensions, or merge digital marvels with the real world; the possibilities of immersive technologies are endless.

While extended reality (XR) hardware is not new, recent innovations are making it significantly more practical and accessible for daily use. Passthrough capabilities allow users to remain aware of their physical surroundings while wearing a virtual reality (VR) device. Meanwhile, intuitive hand-tracking input simplifies navigation, making user interactions feel natural and seamless.

The form factor is also evolving to reach a broader audience: lighter devices, higher-resolution displays, and more comfortable designs.

The debut of Meta VR Glasses introduces a fresh wave of users searching for titles made for this new device class.

This guide will help you understand the key steps to testing and building those user experiences in Unity. Previous experience with Unity is recommended.

© 2026 Unity Technologies

Unity and Meta



Unity is the world's leading platform for XR development. By pairing Meta's leadership in hardware and operating systems with Unity's role as the hub for interactive content creation, you have the tools to reach the largest possible audience for your XR application.

In this guide, you'll learn about the tools, methodologies, and techniques essential for crafting immersive and interactive realities. From Unity's ready-made components, to testing, deploying, and choosing a distribution platform, you'll get the tips and guidance you need to bring your XR applications to life.

Who's this guide for?

This guide is for Unity XR developers who want to create a new XR title or port their existing application to Meta VR devices like Meta VR Glasses or the Meta Quest family of devices.

Meta VR Glasses

Meta VR Glasses are Meta's most advanced VR device yet, designed for entertainment, productivity, and gaming, allowing you to reach a broader audience than ever before.

© 2026 Unity Technologies



What are Meta VR Glasses?

At about 100 grams, Meta VR Glasses are Meta's lightest and most comfortable VR device, enabling longer wear time. The battery and computer have moved to a tethered puck for a two-part system. They feature a 5K Infinite Display with the user's eyes and hands as the primary input. And since they sit within the same Meta VR ecosystem you already know (same OS, same store, same OpenXR stack), your existing Quest apps can run on them.

If you're building for Meta VR Glasses, we recommend reviewing your UI and interaction design to take full advantage of the eye and hand inputs, while maintaining controller support for any titles that rely on it.

Eye and hand tracking is the primary input method for Meta VR Glasses, and applications designed for controllerless users should support this. Meta Quest Touch Plus Controllers are also available to purchase and use with Meta VR Glasses. You can use your existing Touch Plus Controllers during development.

© 2026 Unity Technologies

Capabilities to go deeper

Developers can unlock meaningful new experiences with four capabilities on Meta VR Glasses:

— **Eye Tracking**: Meta VR Glasses support foveated rendering and gaze input. Eye tracking data can inform both visual quality and interaction design, enabling more responsive and immersive experiences.

Note: Eye tracking data is private by default. Your app must request eye tracking permission, and users can choose whether to grant it. Make sure to plan a fallback (such as a hand raycasting) for users that choose not to grant eye tracking permission.

— **Quad Views Rendering**: Render each eye's periphery at lower resolution than the center, cutting GPU cost while maintaining perceived visual quality , giving developers more headroom for scene complexity.

— **Spatial Audio**: Unity supports 7.1.4 surround sound with Enhanced Audio Foundation, enabling Spatial Audio experiences for VR with seven surround channels, four height channels and a low-frequency effects channel. Select 7.1.4 in Output Channel Layout.

— **5K Infinite Display**: The 5K Infinite Display uses micro-OLED panels with a per-eye resolution of 2412 × 2288 pixels at 37 PPD, compared with 25 PPD on Quest 3 with a higher contrast. Consider supporting high resolutions when possible to take full advantage of the Meta VR Glasses' displays.

VR or MR application

Virtual reality (VR) provides fully immersive experiences, while mixed reality (MR) enhances reality.

Meta VR Glasses' passthrough capability displays a live view of the real world to the user, through forward-facing cameras, allowing them to see their physical surroundings while wearing the device.

Passthrough is required for MR applications where digital elements are overlaid on the real world. It also enhances user safety and convenience by allowing them to interact with their environment without removing their Meta VR Glasses. This feature bridges the gap between complete virtual immersion and the user's actual physical space.

Controls

Most VR systems enable tracked six degrees of freedom (6-DOF) controllers, with hand tracking and gesture recognition becoming increasingly common on VR devices. However, there are also situations where VR users will need to switch back to game controllers. Unity lets you enable and manage switching between these input methods, which is covered later in this document.

© 2026 Unity Technologies

Use cases

VR suits fully immersive story-driven experiences, while MR can provide environment-based activities.

Practical considerations will influence the decision to choose either VR or MR for your application. MR might be ideal for training simulations or educational experiences where real-world integration helps, while immersive VR is best suited for fully immersive storytelling or entertainment.

In this e-book, we'll introduce how to get started with fully immersive applications, which is the primary medium for game content. However, development with mixed reality or augmented reality (AR) is also possible with Unity's XR Interaction Toolkit (XRI) and AR Foundation (which add new components to the project that allow passthrough and provide features for MR applications).



© 2026 Unity Technologies



Unity's XR Interaction Toolkit

The XR Interaction Toolkit (XRI) package enables you to develop XR interactions efficiently by providing a standardized approach to interactivity like grabbing, touching, locomoting, and manipulating objects.

This toolkit provides a framework that makes 3D and UI interactions available from Unity input events. The core of the XR Interaction Toolkit system is a set of base **Interactor** and **Interactable** components, and an **Interaction Manager** that ties these two types of components together. The XR Interaction Toolkit also contains components that you can use for locomotion and drawing visuals, such as ray lines, reticles, and grab affordances that show users where they can interact with objects and what they can interact with.

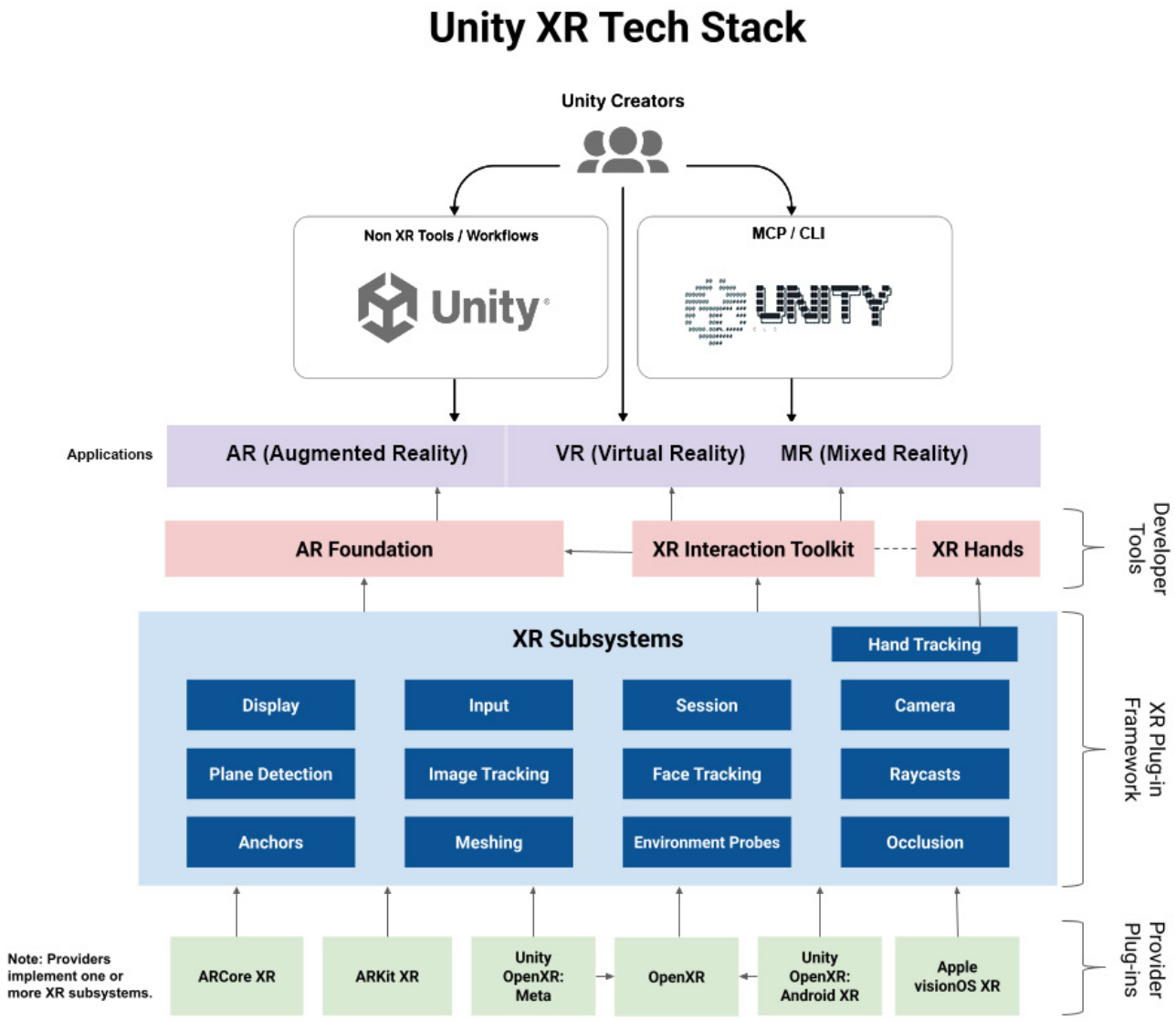
Leveraging Unity's cross-platform capabilities, the XR Interaction Toolkit enables developers to focus on designing XR experiences without the complexity of managing disparate input and interaction models across different platforms and devices.

SDKs for XR development

Unity supports XR development through its plug-in framework and a set of feature tools and packages. The XR Plug-in Management package, available from the Unity **Project Settings** window, provides developers with a choice of available plug-ins for their target XR platforms.

© 2026 Unity Technologies

Below is a diagram that illustrates the current XR plug-in framework structure and how it integrates with platform provider implementations:



OpenXR compatible platforms can also have their non-exclusive device features accessed via OpenXR APIs and extensions, for example, Meta devices.

Each XR platform vendor will typically have its own APIs to use all the device-specific features of the hardware. For example, Meta devices utilize the Meta XR All-in-One SDK, available on the Unity Asset Store.

Check your device's developer guides on how to access the required SDK for your hardware.

© 2026 Unity Technologies

OpenXR

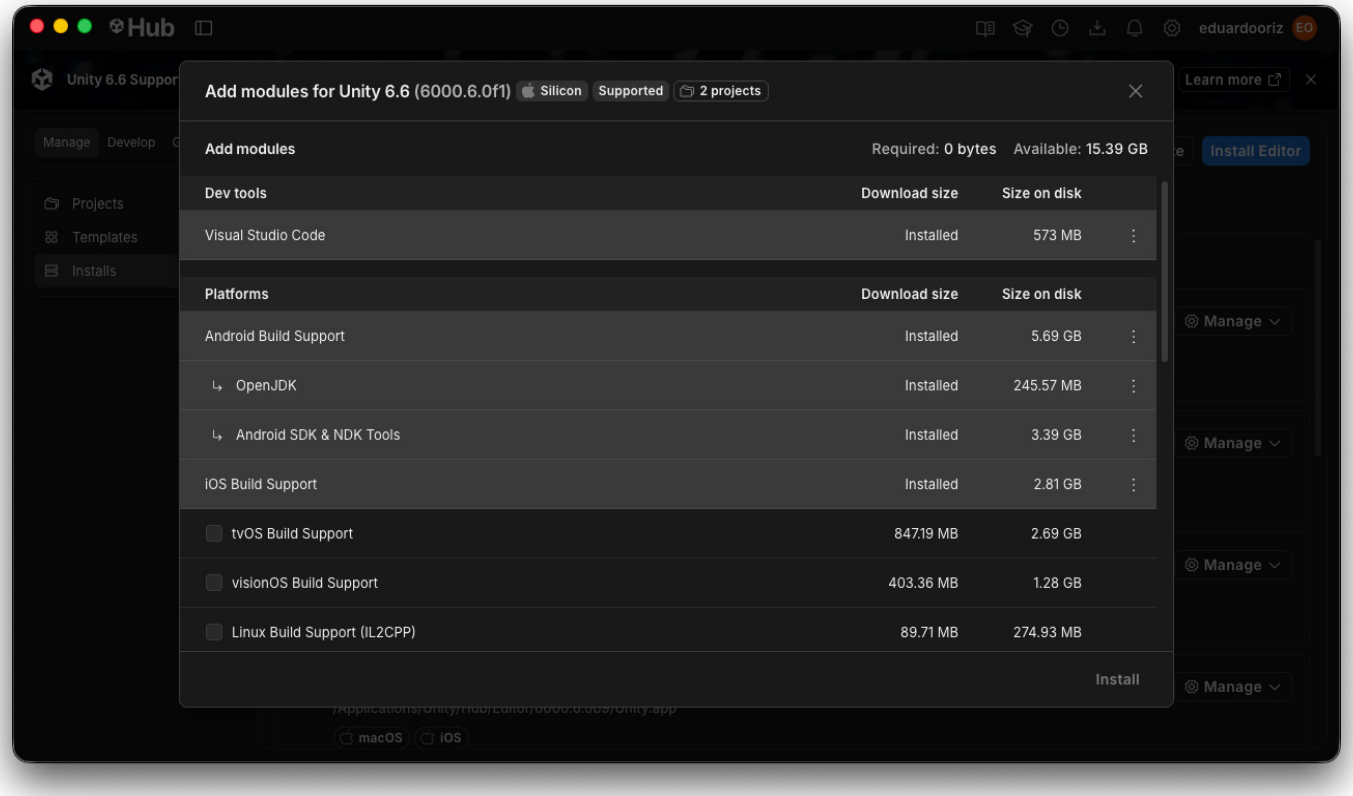
OpenXR is a royalty-free framework developed by the Khronos Group, which consists of hardware and software vendors from across the XR industry. The specification is designed to ease the creation of cross-platform XR experiences. Unity's OpenXR plug-in should work with any device that supports conformant OpenXR runtimes.

OpenXR provides a standardized interface for XR hardware, allowing developers to create applications that are compatible with a wide range of XR devices without needing to write device-specific code. This makes it easier to develop and deploy XR applications across various platforms, ensuring broader compatibility and reducing development time and effort.

Editor version and modules

This guide uses Unity 6.6, and it's recommended that you use the same version, or a newer one. We'll be using the package XRI version 3.6, but similar workflows are expected in newer 3.x versions.

When you install the Editor, you'll be prompted to add modules to the installation. To support mobile VR platforms like Meta VR Glasses or Meta Quest 3, you'll need the **Android Build Support**, **OpenJDK**, **Android SDK**, and **NDK** modules.



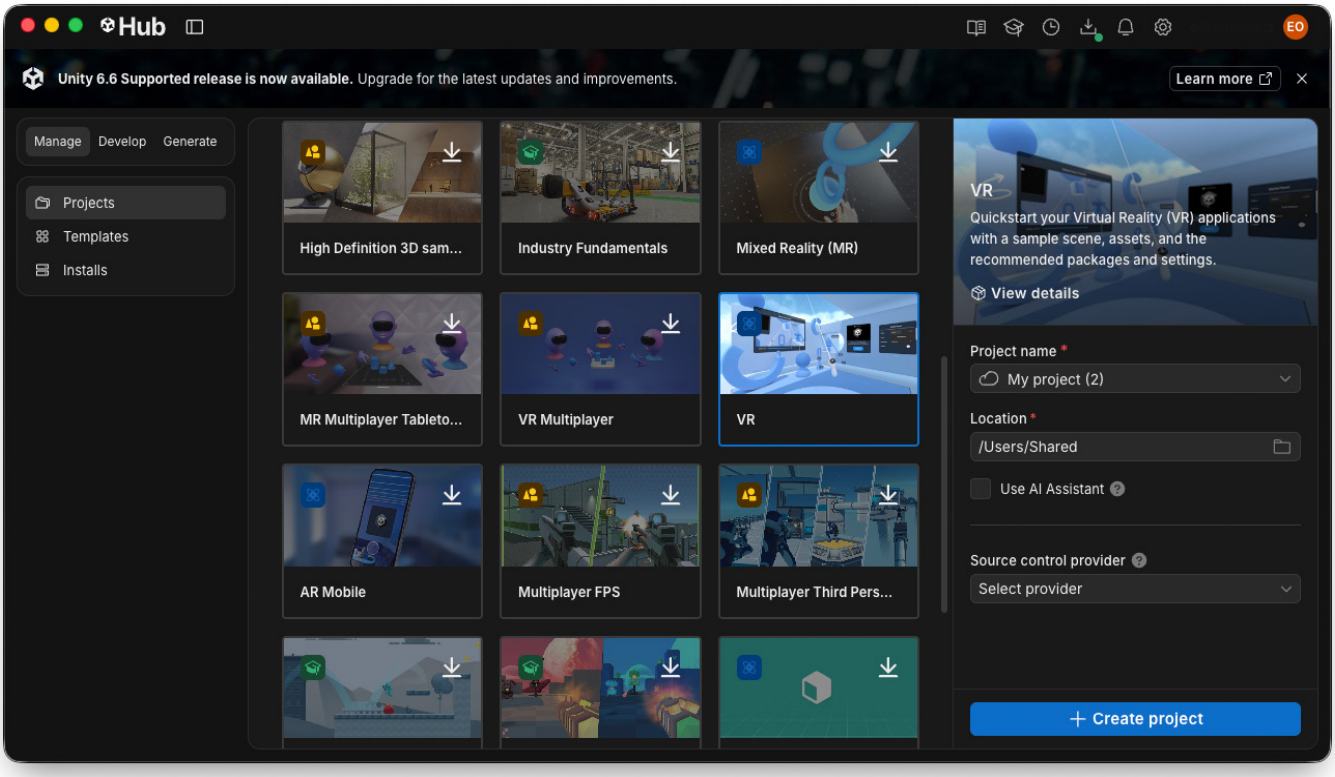
© 2026 Unity Technologies

Getting started in VR with Unity's VR template

The **VR** template is available on the Unity Hub and contains a scene that gives you prefabs you can use to build your application. The template provides sample scenes, assets, and recommended packages and settings. To get the template follow these instructions:

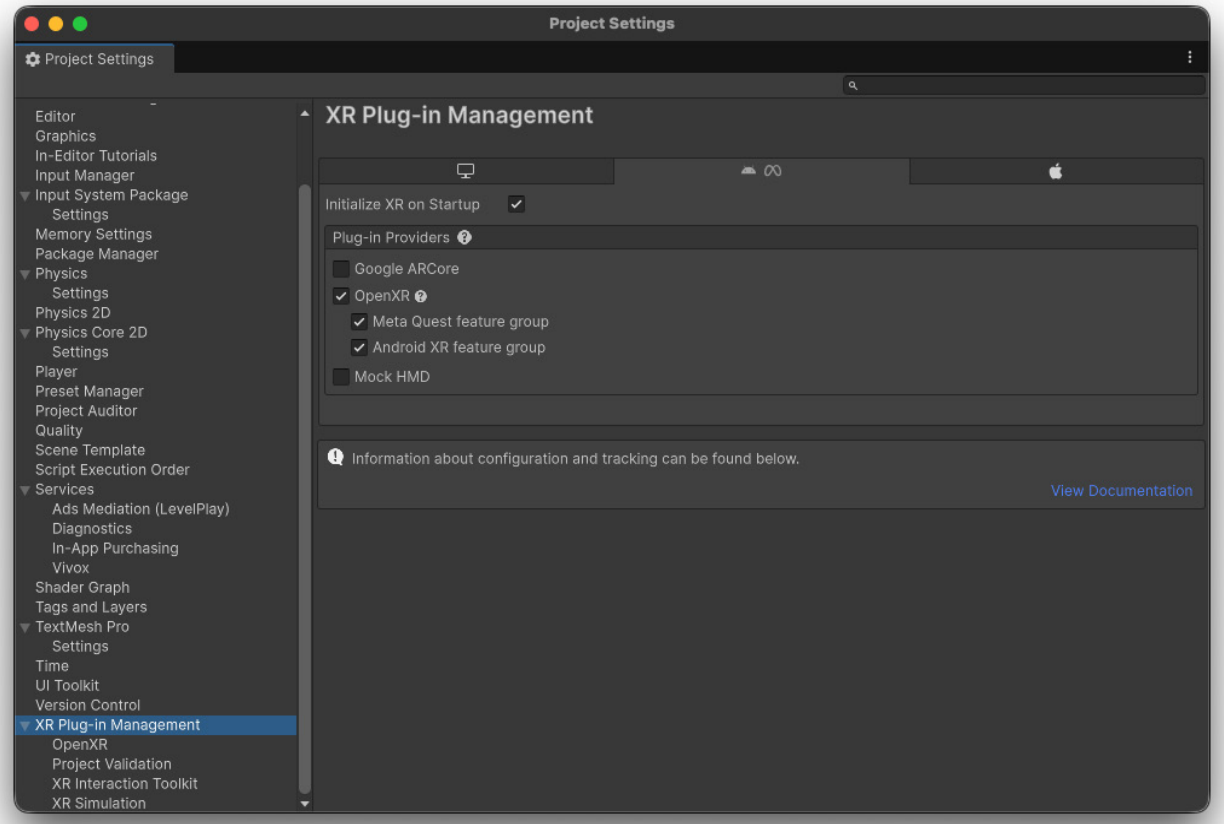
1. Open the Unity Hub and select the **New project** button.
2. Change the **Editor version** to the latest official Unity version available.
3. In the templates list, locate the **VR** template and select the **Download template** button.
4. Choose a name for your project and select its location in the **Project** settings.
5. Click the **Create project** button to launch the Unity Editor.

Unity will then open, and you'll be presented with the welcome screen.

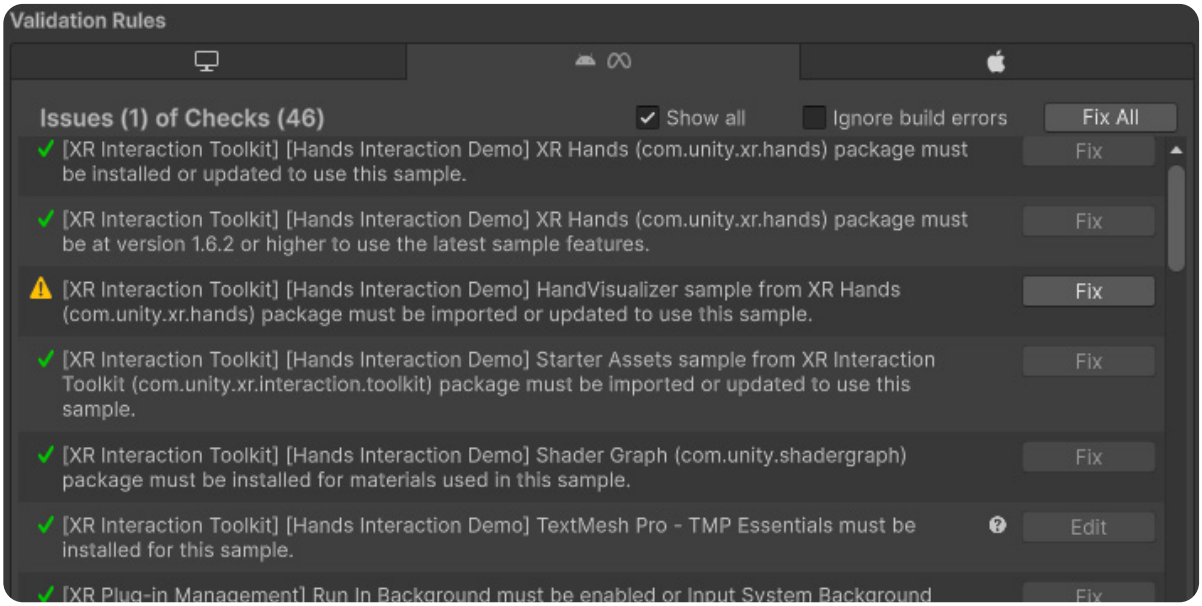


The **VR** template's documentation contains a guide on how to navigate the sample scene. One of the first steps you should take is to ensure the **VR** template runs on a VR device so you can experience it. In the **Project Settings** window, check that **OpenXR** and both **Meta** and **Android XR** feature groups are enabled. This will make the sample run on Meta devices, but will also ensure that portability to other platforms is possible.

© 2026 Unity Technologies



In the left panel of the **Project Settings** window, under the **XR Plug-in Management** section, you'll find the **Project Validation** tool, which helps you fix/add any missing components needed to run your XR project. For this sample, you'll want the **HandsVisualizer** demo from the **XR Hands** package to be updated so you can test hand controls.



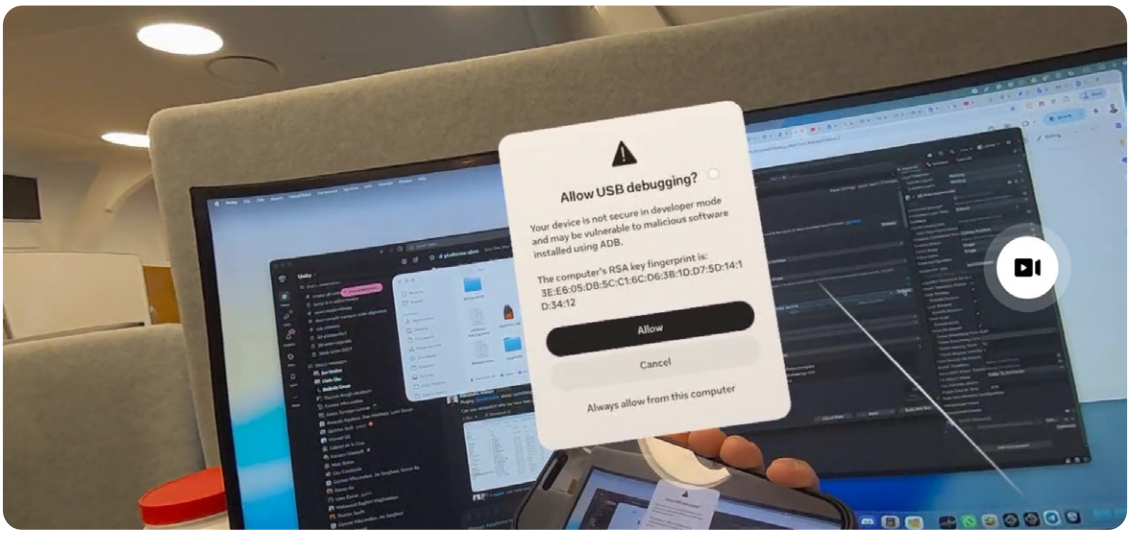
Tip: AI agents can help you fix errors in your project using the new Unity CLI and the Unity Pipeline package. Meta XR Operator lets them interact with the running app and check the result. Those tools are covered later in this guide.

© 2026 Unity Technologies

Running the template on a device

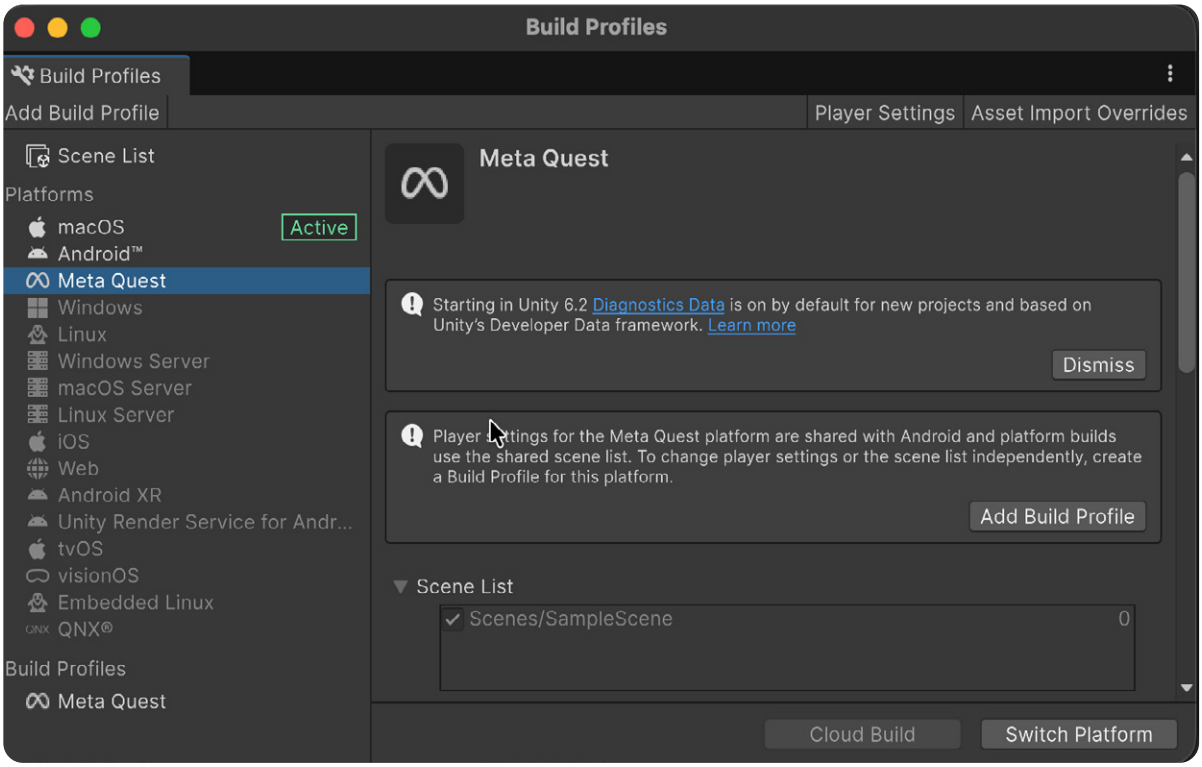
To test the **VR** template on a device, follow these instructions:

1. Enable Developer mode on the VR device.
2. Connect the VR device to your computer via the data cable and accept the pop-up dialog about USB debugging.



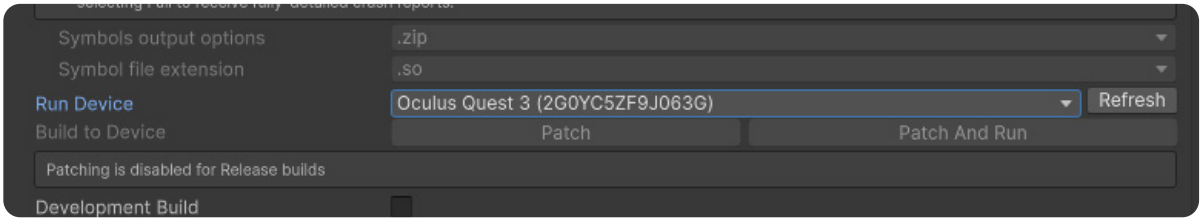
3. In the Unity Editor, from the main menu select **File > Build Profiles**, then select **Switch Platform** to **Meta Quest**.

If you didn't have the option available, check that you aren't missing the Android Build Support modules for your Unity Editor version.



© 2026 Unity Technologies

If everything is connected properly, you will see your Meta device under **Run Device**. Your Meta device will then be available in the list of **Build Profiles** to deploy to.



4. Select **Build and Run**.

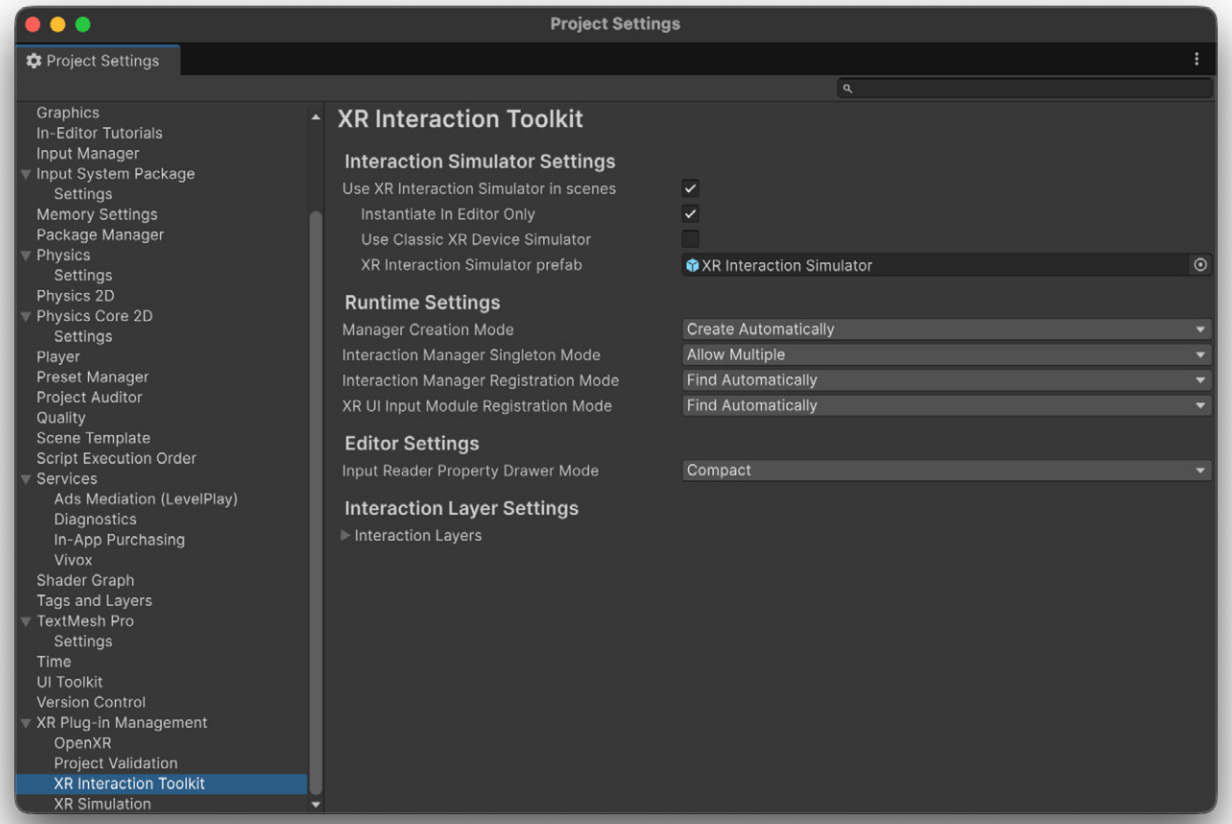
The first time you do this it might require several minutes to complete the compilation, after some time, the APK will be copied over to the device and you'll be able to interact with the world and see what functionality the elements and components in the scene bring.



Testing with the XR Interaction Simulator

You can use the XR Interaction Simulator, part of the **XR Interaction Toolkit (XRI)** package, to test XR interactions in the Editor without needing a physical headset.

© 2026 Unity Technologies



This will save you time during development by allowing you to interact and move in your application directly from the Unity Editor.

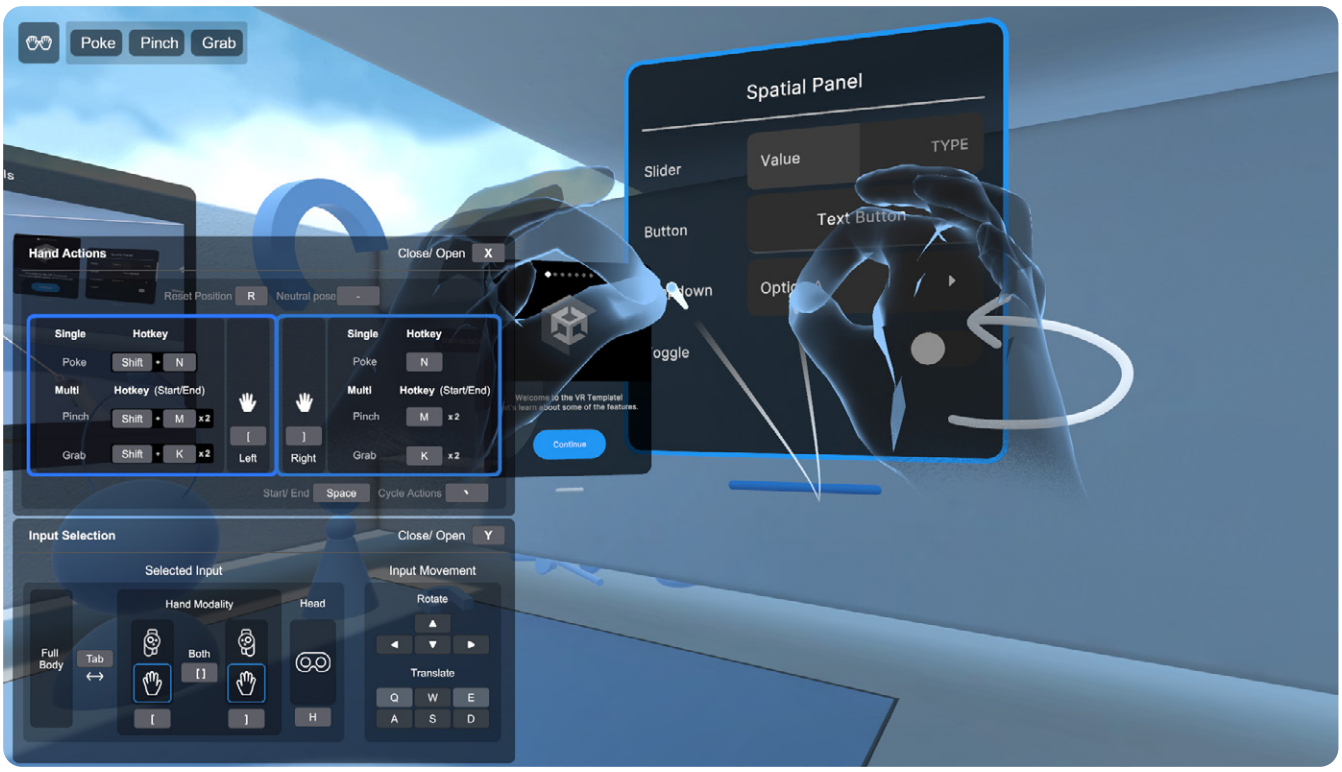
You can enable the **Use XR Interaction Simulator in scenes** from the XR Interaction Toolkit **Project Settings**. This option will instantiate the **XR Interactor Simulator** prefab before you enter Play mode.

Note: The simulator requires the **XR Origin** prefab to be present in the scene, which is already added in the sample scene.

© 2026 Unity Technologies



You can swap between simulated hands and simulated physical controllers by pressing the shortcuts shown in the UI panels, like users of real devices. In the lower-left corner, you can see the shortcuts and options to simulate controls or hand gestures.

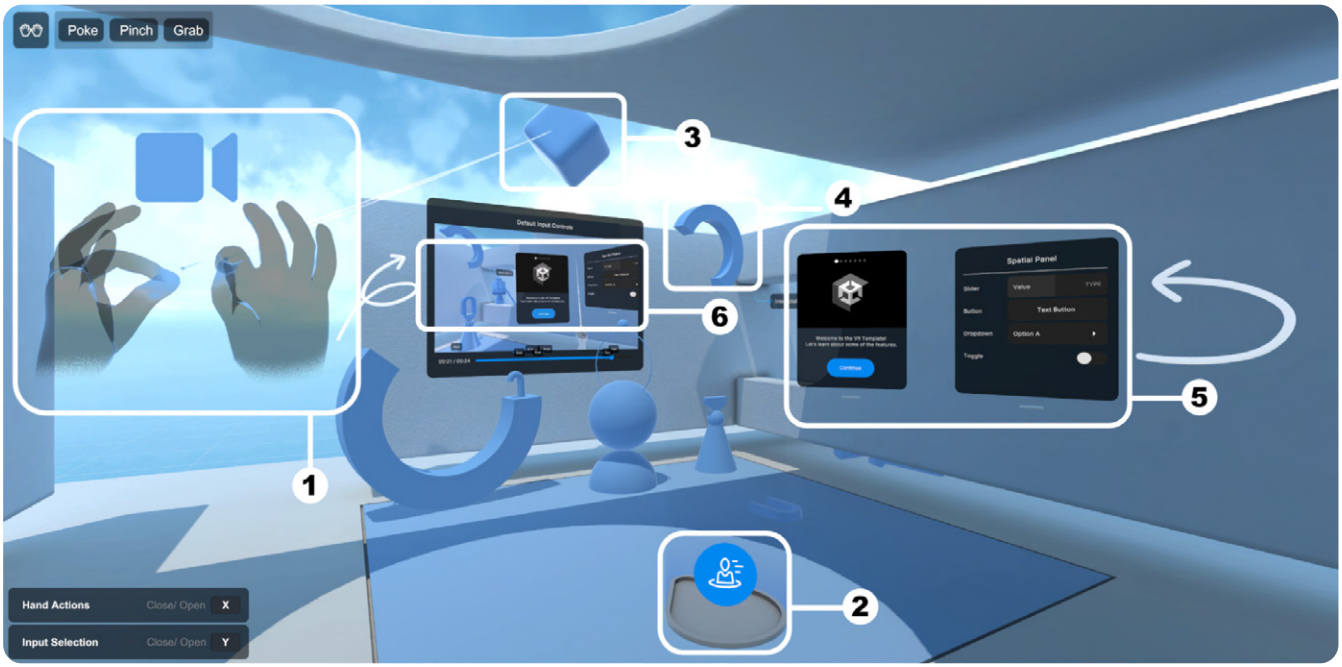


Meta Horizon Link for Windows

Another alternative for quick iteration for Windows users is using the Meta Horizon Link application. Meta Horizon Link reduces your iteration time by directly launching the app you're developing from the Editor in your device. No need to build the app on PC and deploy it to a Meta VR device every time you need to test.

© 2026 Unity Technologies

Key XRI components



The prefabs in the XRI package samples and VR template might be all you need to easily block out a level and functionality for your project. You can swap the visuals later, configure exposed values, or add functionality on top of it. For now, let's check out these key components.

© 2026 Unity Technologies

1. XR Origin: Move in the virtual world

The XROrigin component is essential and serves as the center of tracking space in an XR scene. It's typically attached to the XR Origin GameObject and stores a reference to the GameObject that will be manipulated via locomotion, which is typically the same XR Origin GameObject itself. It includes some other key components relative to the player character's movement, interaction, and mixed-reality features.

Camera

The **Camera** prefab handles camera positioning in relation to XR devices with support for different tracking modes such as **floor** or **device** level.

The **Camera** prefab has the **Tracked Pose Driver** component attached to it to synchronize its position and rotation with input data from a tracked XR device. In other words, when the user looks around, the **Camera** GameObject also looks around, showing the virtual environment. The **Camera** prefab includes the Tunneling Vignette Controller component to help make locomotion more comfortable.

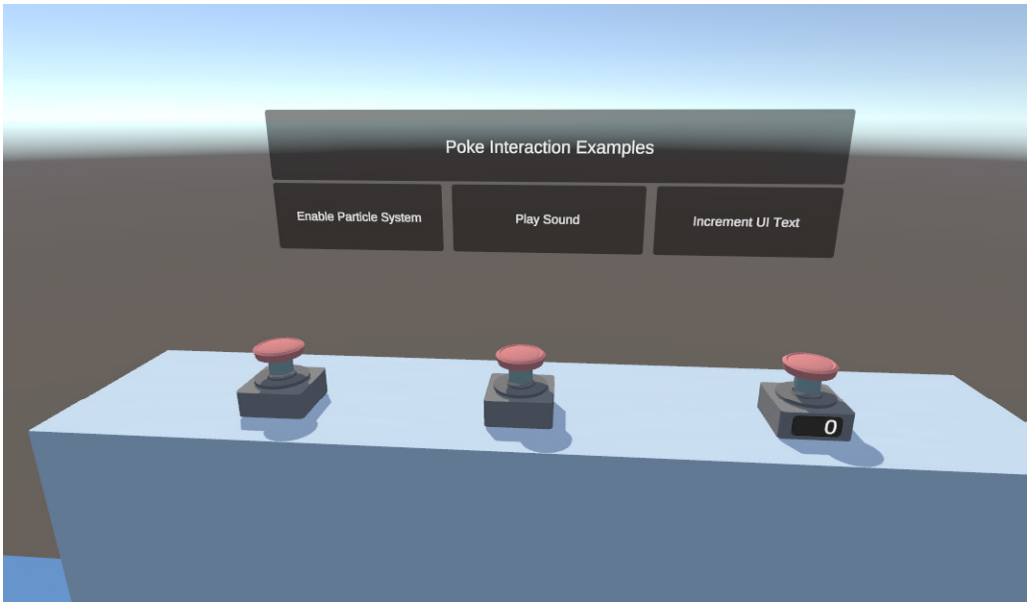
XR Input Modality Manager / Controllers / Hand

The XR Input Modality Manager component manages the dynamic switching between hand tracking and controllers during runtime based on their tracking status. It toggles between different sets of GameObjects representing hands and controllers, ensuring that the appropriate input method is active. If hand tracking begins, it switches to the hand group of interactors; if motion controllers are used and tracked, the **XR Input Modality Manager** component activates the motion controller group. This component is beneficial even in projects that don't use hand tracking, as it prevents showing controllers in their default position when not tracked.

The **XR Input Modality Manager** component includes references to the **left/right hands** and **left/right controllers** prefabs. Below are some key components that make the controls interact with the virtual world:

- **Tracked Pose Drivers**: These drivers are added at the root of the GameObject, to position and rotate the virtual control following your real controllers or hands.
- **Interactor components**: These components are required to interact with interactable GameObjects. There are different types of interactions depending on how you wish to manipulate the virtual world:
    - **Poke**: Anything can be done upon the press, or poke, of a button. This component enables your control to do so.

© 2026 Unity Technologies



Poke Interactables can be found in the Starter Assets sample of the XRI package.

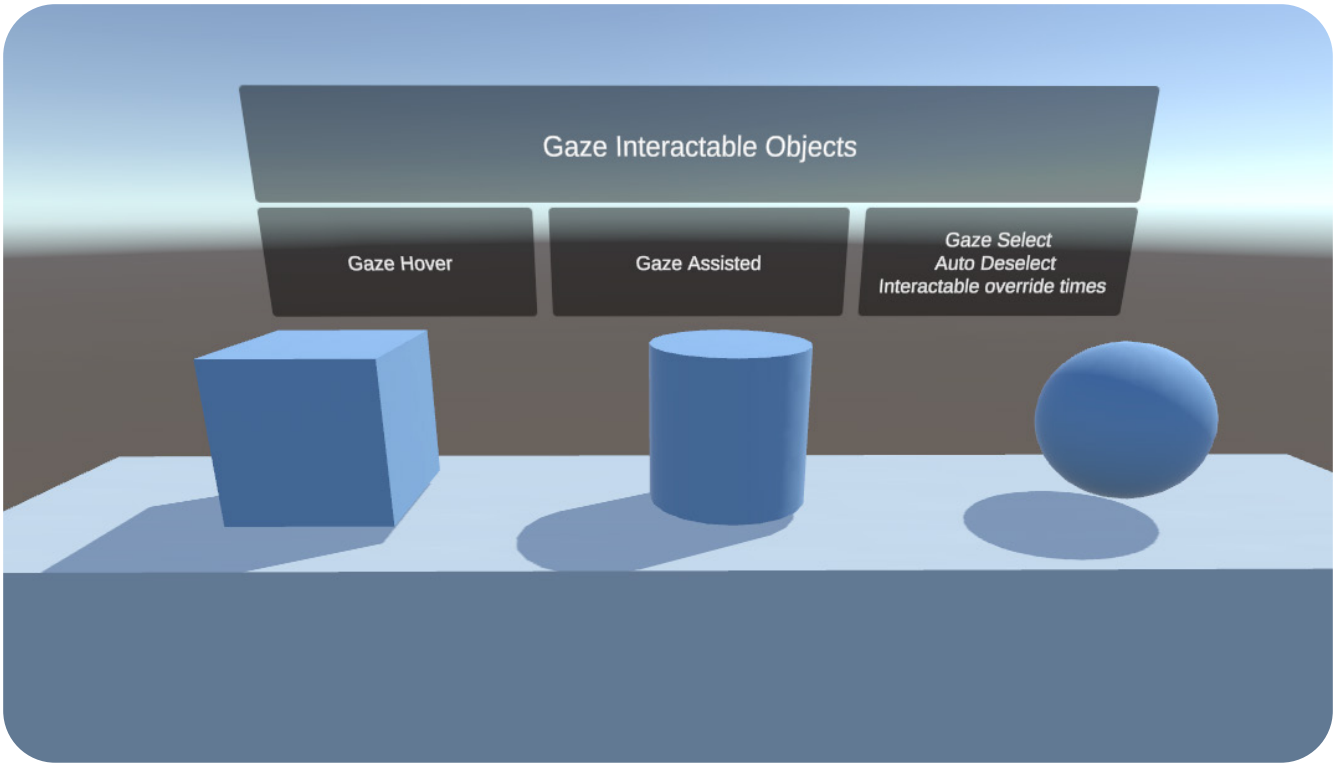
— **Near-Far**: This component interacts with both close and distant interactables when the user either touches or points at them. The sample prefab is configured with additional components that will draw line visuals to help guide distance-based interactions.

— **XR Ray**: The **XR Ray** component lets you interact with interactable GameObjects at a distance. It uses ray casts to detect eligible targets. You can also add additional components to it, for visual or audio representation, and to fine tune the behavior. In the sample, this component has been renamed **Teleport**, since that's what it's used for.

— **Gaze**: The XR Gaze Interactor component doesn't follow hand positions, but instead follows head or eye movement. It allows interaction in VR with objects through gaze. This is useful to reveal or highlight elements when the user is looking around in the virtual space. It can also be used in combination with other interactors to assist in object selection.

Note: The Gaze Interactor child prefab needs to be enabled and the **Eye Gaze Interaction Profile** needs to be added in the **OpenXR** settings inside **Project Settings**.

© 2026 Unity Technologies



Examples of Gaze Interactables can be found in the Starter Assets sample of the XRI package.

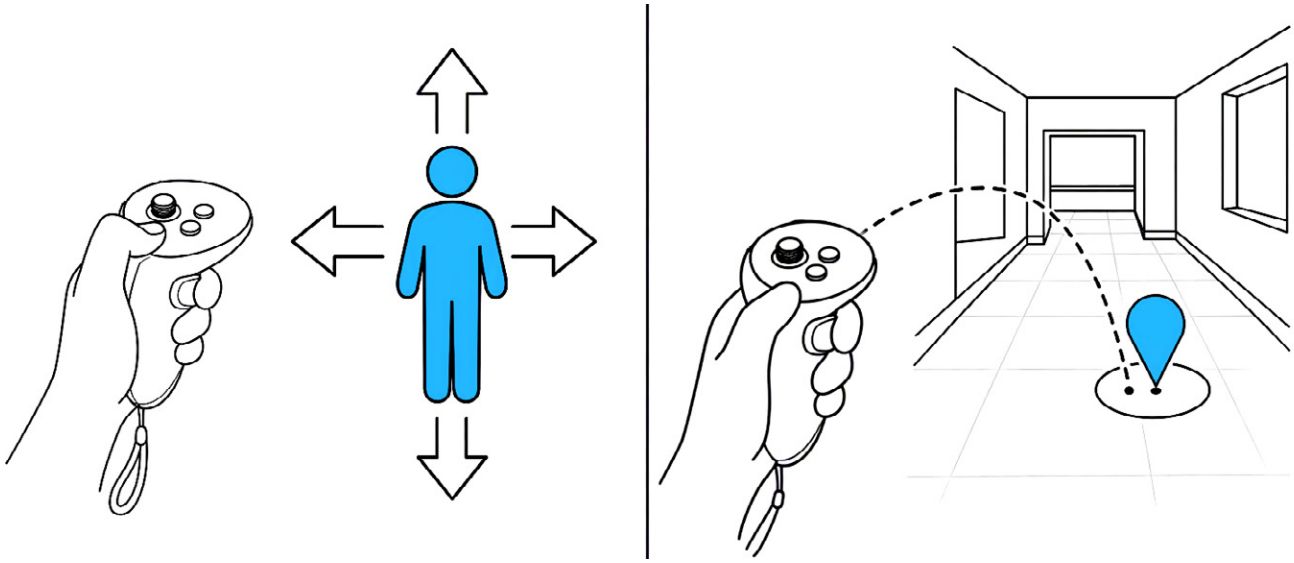
    - **Socket**: Present in other demo scenes of the XRI package, the **Socket Interactor** component is a perfect interactor for puzzle mechanics, an inventory system, key holes, or switches. A **socket** is defined as the target for a specific interactable, such as a keyhole for a key or a battery socket for a battery. You can read more about them in the documentation.

- **Visual representation**: The visual representation of the control contains the 3D model that will follow the **Tracked Pose Driver** and controllers to animate them.

    - **Controller**: Controller uses a component that reads input values and drives the thumbstick, trigger, and grip transforms.
    - **Hand**: With Hand tracking, things differ a little under the hood: poking is no longer the press of a button, but recognizing a hand gesture. Doing a gesture also requires animating the mesh skeleton. The Hand Visualizer component will automatically animate a hand mesh for each hand using hand tracking data from the device. Unity has also included stabilization components to smooth out any inconsistencies with tracking data. To fully support Meta VR Glasses, you'll need to ensure eye and hand input is implemented. Note: Using hands on some platforms might require user permissions, but the prefab in the template and samples will already handle that for you.

© 2026 Unity Technologies

Locomotion with XR Body Transformer



Most games offer several options for player's movement such as free move locomotion or teleporting. To support hands-first devices like Meta VR Glasses, you should focus on teleporting first.

The **XR Origin** component represents the virtual space's reference point, typically where the user is located. The **Locomotion** prefab with the **XR Locomotion Mediator** component manages and controls access to the **XR Origin** component. **XR Body Transformer** component coordinates with the XR Locomotion Mediator and applies queued transformations to the **XR Origin** component in priority order, preventing conflicts and ensuring a smooth and predictable movement experience in XR applications.

The main methods for manipulating the user's position are turning, moving, teleportation and climbing.

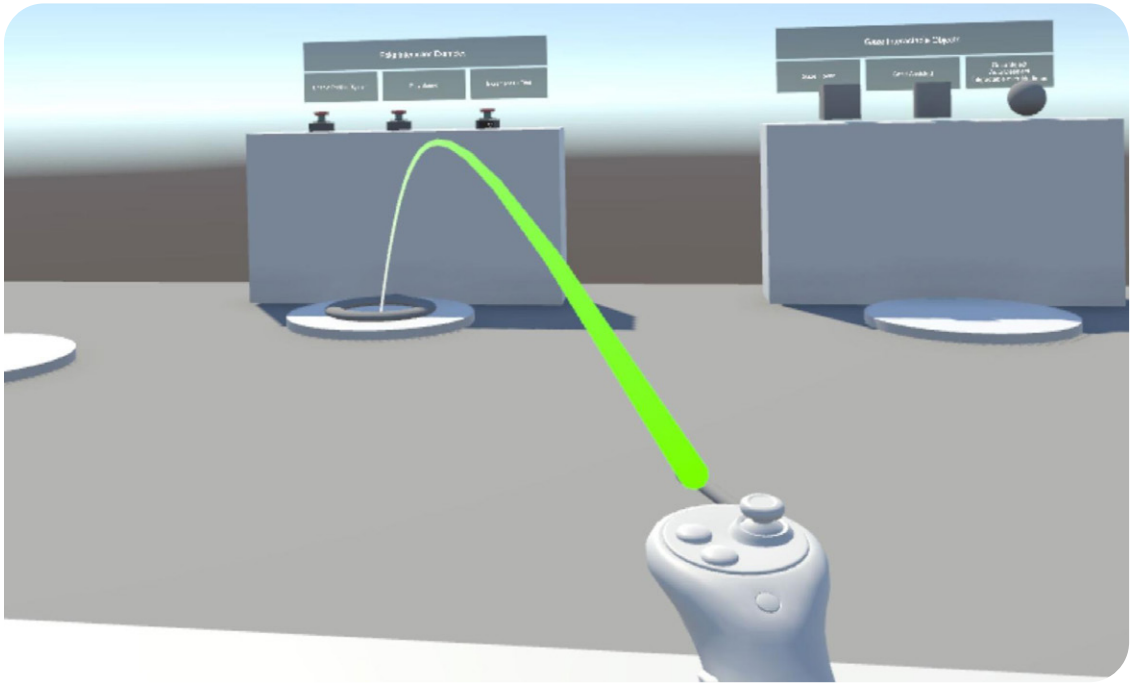
— **Turn**: The XR Interaction Toolkit provides two components for player rotation: Snap Turn and Continuous Turn. **Snap Turn** allows users to rotate in fixed degree increments, customizable via the **Snap Turn Provider** component. **Continuous Turn** offers smooth rotation at a variable speed set by the **Continuous Turn Provider** component. Both components integrate with Unity's input system for ease of use. The XR Interaction Toolkit comes with preconfigured input actions, simplifying development and ensuring these turn components are ready to use with minimal setup.

— **Move**: The **DynamicMoveProvider** is a variant of continuous movement control, designed to adjust the frame of reference for movement direction based on user preferences. It allows for forward direction controls, either head-relative or hand-relative, depending on the user's choice for each hand, using either the forward direction of the head or the forward direction of the hand or controller. This provides a customizable and intuitive movement experience in XR environments.

— **Teleportation**: **Teleportation** in the XR Interaction Toolkit allows users to move instantly to a new location within the virtual environment. This is achieved by using the Teleport/XR Ray Interactor described in the previous section and aiming at a target location with a special types of Interactables, called a **Teleportation Anchor** or a surface with a **Teleport Area** component, and initiating a teleport action configured on the XR Ray Interactor component. The teleportation system

© 2026 Unity Technologies

includes a visual indicator, such as an arc or line, showing the path and destination of the teleport. It's a key feature for navigating larger virtual spaces comfortably, especially in scenarios where continuous movement might cause discomfort or isn't feasible due to space constraints. The **Teleportation Provider** component handles teleportation in XR environments. It's responsible for moving the **XR Origin** component to a specified location at the user's request.



Teleporting using anchor points

While the number of components that make an immersive XR experience possible can seem overwhelming, but you don't need to build your character controller from scratch. Instead, add the **Complete XR Origin Set Up Hands Variant** prefab to your scene and you'll have everything set up and ready to go. You can always fine tune and enable/disable components to make it work for your project.

2. Teleportation area / anchor

Teleportation area and **Teleportation anchor** are components in the **Teleport Area Setup** prefab. They differ only in where you land. A **Teleportation Area** covers a wide surface, and you land exactly where you aim within it. A **Teleportation Anchor** marks one fixed spot a ray must hit for users to move to.

The **TeleportationArea** component creates a free-form surface: you land wherever the teleport ray hits a GameObject with its collider component. It's best suited to floors, terrain, and walkable platforms, because a single component can cover a large region while still letting the user choose the exact spot they want to land. The user picks that spot by aiming the ray from their controller or hand, typically shown as a curved beam or line with a reticle marking the landing point.

The **TeleportationAnchor** component creates a fixed predetermined destination. Hitting anywhere on its collider sends you to a specific **anchorTransform**, where the user faces

© 2026 Unity Technologies

a determined direction, regardless of where the ray landed. The **TeleportationAnchor** component is best used for spots that must be exact for users to have the right experience: a chair, a cockpit, a station in front of a control panel, a marked pad, etc.

The practical use cases are as follows:

	Teleportation Area	Teleportation Anchor
Landing point	Ray hit point	The anchor transform
Landing rotation	From **matchOrientation** only	Inherits the anchor's orientation
Implementation	One component per floor	One component per spot
Landing precision	Player chooses	Designer chooses

3. XR Grab Interactable

The **XR Grab Interactable** component enables basic grab functionality in XR environments. It allows a GameObject with this component attached to it to be grabbed or selected by an interactor (like a VR hand controller) and to follow its movements. A new **Grab Interactable** GameObject created from the top XR menu will include three core components: a Grab Interactable component, an XR General Grab Transformer component, and a Rigidbody component.

One of the features of the **XR Grab Interactable** component are its **Tracking** or **Movement Types**:

— **Kinematic**: GameObjects move directly with the interactor, bypassing physics forces and collisions, offering precise control over their position and rotation. This is ideal for predictable and scripted interactions.

— **Instantaneous**: The GameObject's position and rotation are updated immediately to match the interactor's (such as hands or controllers), without physics calculations, ensuring responsive, direct control.

— **Velocity Tracking**: Movement is driven by dynamically updating the GameObject's velocity based on the interactor's motion, allowing for realistic physics interactions and natural-feeling behaviors, like inertia and momentum.

The **XR General Grab Transformer** component supports moving and rotating with one or two interactors (single or multiple). The **Rigidbody** component is what makes GameObjects react to the physics world using the collider shape that better conforms to the shape of the GameObject.

© 2026 Unity Technologies

4. XR Knob

The **XR Knob** component extends the **XR Base Interactable** class. Grabbing it doesn't move the GameObject into your hand: it converts your hand's motion into rotation, which is useful for things like steering wheels, valves, and other elements that spin on an axis. Try it out with some of the GameObjects in the scene.

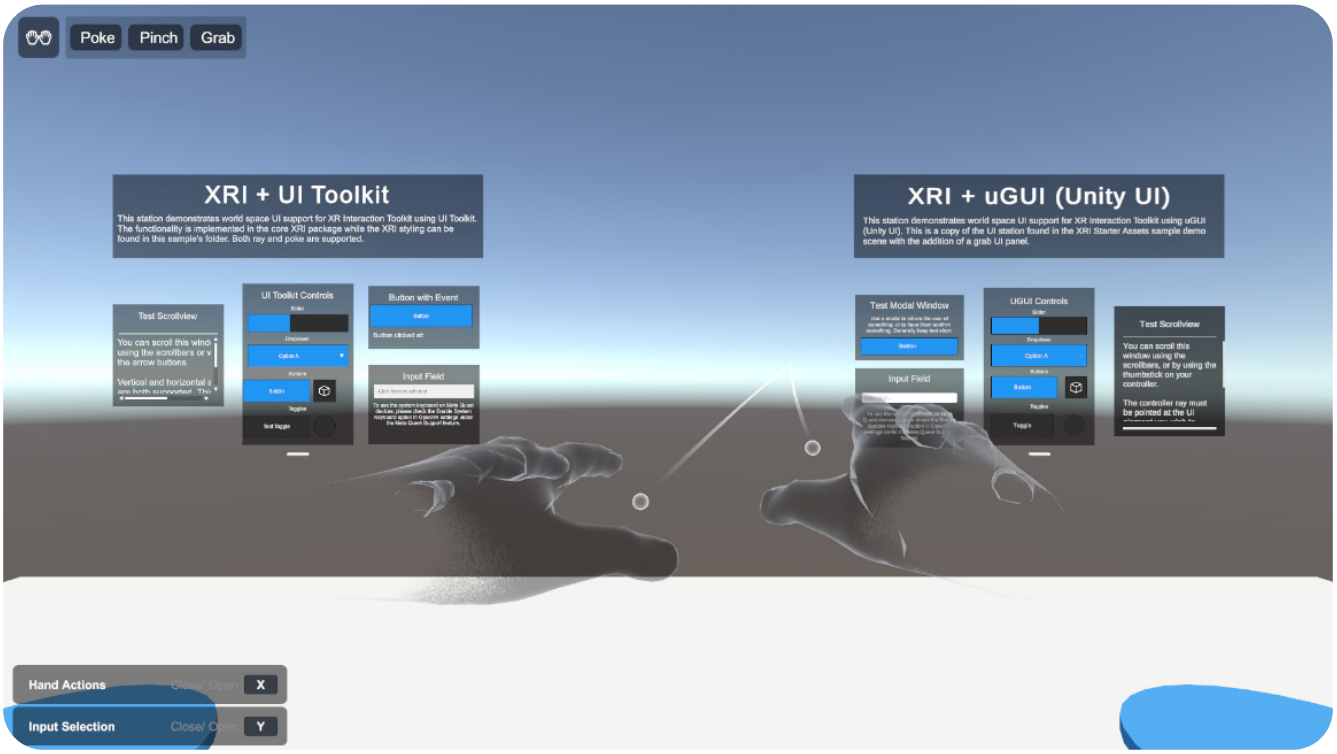
5. Spatial UI

When users visit your XR worlds, they'll need to interact with a user interface at some stage. The World Space UI sample, which ships as part of the XR Interaction Toolkit, demonstrates how UI components react to input from the user's controllers. Both of Unity's UI systems are supported: uGUI and UI Toolkit.

uGUI is Unity's UI GameObject-based UI system. You can build user interfaces in the scene from GameObjects with components (Canvas, RectTransform, Image, Button), and laid out and styled in the Inspector window with anchors.

UI Toolkit is a newer system. It's modeled on web development: UXML files define the structure, USS stylesheets (similar to CSS) handle the styling, and a flexbox engine does the layout. The elements are lightweight objects, not GameObjects, which is why it usually performs better on large or complex UIs.

The demo scene has three examples of UI interactions. A core element of creating the UI functionality is the **XR UI Input Module** component located on the **Event System** GameObject.



UI elements in the demo scene in the World Space UI sample show both common UI implemented with both systems.

© 2026 Unity Technologies

Event System / XR UI Input Module Component

The **XR UI Input Module** is the component that the XRI package requires to properly interface with the Event System. It manages interactions between Unity UI (uGUI) in the XR environment and works in concert with the **Near-Far, Poke and XR Ray Interactors**, to ensure XRI interactions with the UI are processed correctly. It supports a wide range of input methods, including 3D tracked devices as well as non-XR sources like mouse, touch, and gamepad controls if configured to do so.

Tracked Device Graphic Raycaster

On each GameObject with a **Canvas** component (the area where the UI will be drawn), a **Tracked Device Graphic Raycaster** component is required. This is a custom implementation of the **GraphicRaycaster** for the Unity XR Interaction Toolkit. The **Tracked Device Graphic Raycaster** component lets you use 3D tracked devices to highlight and select UI elements on a canvas.

6. Video playback

To make your project more immersive and interactive, you can play videos in the 3D world space with the **Video Player** component. **The Video Player** component, and all the other additional components required to make video playback work, are already attached to the **Tutorial Player** prefab. The **Video Player** component includes playback controls that you can disable if you don't need them.

All these prefabs enable designers to quickly build a new game or VR project with the most common interactions already enabled.

© 2026 Unity Technologies



Develop for Meta VR Glasses

Meta VR Glasses are built for everyday VR, with the user's eyes and hands as the primary interaction models. If you already have your project built for Meta Quest 2 or Meta Quest 3, it will already work for Meta VR Glasses. Key considerations to keep in mind when porting your project to Meta VR Glasses are the different screen specs, field of view (FOV), and hand tracking input.





	Meta VR Glasses	Meta Quest 3
Display resolution	2412 × 2288 pixels per eye	2064 × 2208 pixels per eye
FOV (degrees)	70° horizontal × 66° vertical	110° horizontal × 96° vertical
Pixels per degree	37	25
Refresh rate	72 Hz, 90 Hz, 120 Hz	72 Hz, 90 Hz, 120 Hz

© 2026 Unity Technologies

Memory	12 GB	8 GB
Weight	About 100 g	515 g
Processor	Snapdragon Reality Elite	Snapdragon XR2 Gen 2
Battery life	Up to 3 hours of media playback	Up to 2.2 hours of usage on average
Primary input	Eye and Hand tracking; Touch Plus controllers available for purchase separately	Touch Plus controllers
Eye tracking	Supported	Not supported
Audio	Spatial audio, Dolby Atmos	Spatial audio

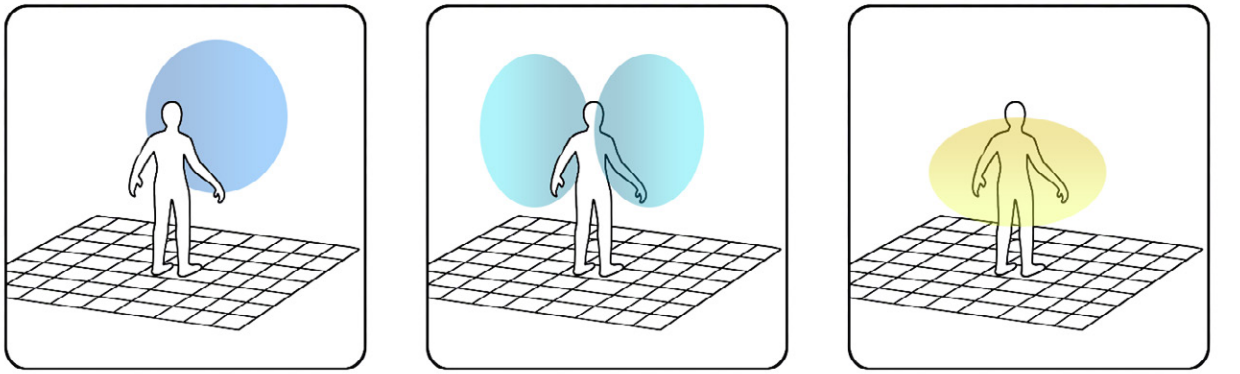
The two major differences between Meta Quest devices and Meta VR Glasses in display and hand tracking are as follows:

— Meta VR Glasses provide higher resolution per degree. If your existing Quest game has GUI placed near the edges of the screen, you may need to adjust its location toward the center.

— Meta VR Glasses use eyes and hands as primary inputs. Although they are compatible with Touch Plus controllers, you'll reach the largest audience possible by supporting eyes and hands. Meta VR Glasses are equipped with six tracking cameras facing different directions to track hands across a wide range of positions..

Note: If Meta VR Glasses are used in very low-light conditions, a prompt will appear informing users the cameras might not work as expected under those conditions.

To help you decide control support for your application the table below shows the use cases and target audience for both types of controls.



© 2026 Unity Technologies

	Hand support	Controllers and hand support	Controllers support
Intended audience	Non-gamers Casual gamers	Casual gamers Core gamers	Core gamers
Controller priorities	Social presence Comfort	Let users decide	Precise input Competitive gameplay
Unique interactions	Hand gestures Physics-based hands Surface interactions	No interactions to hands or controllers	Sliding locomotion Guns Out-of-range motions

If your app requires controllers, only Meta VR Glasses users with paired controllers can discover, buy, and use it, so a hands-first mode widens your reach on the Meta Horizon Store.

Meta Quest Build Profile

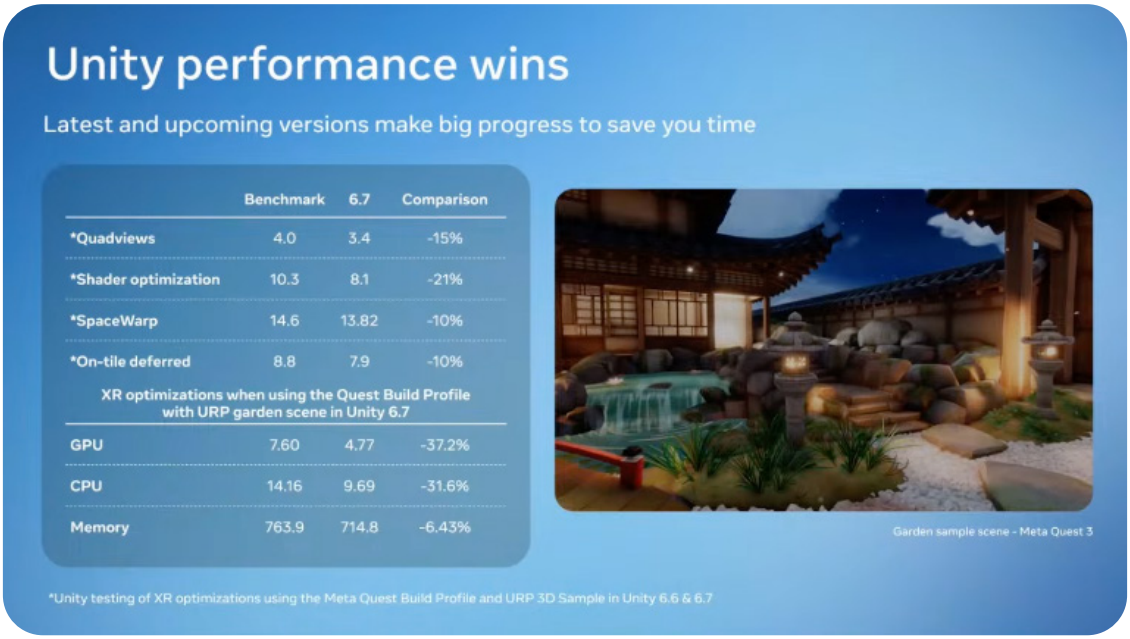
When you build your application for Meta VR Glasses, you should select the **Meta Quest Build Profile**, which supports the following graphical optimizations for Meta VR Glasses:

- Quad Views rendering:
  - Most effective when the application is GPU-bound (foveated rendering best when CPU-bound)
  - Better post-processing support than foveation
  - Fixed center inset
- Deferred Rendering improvements over the default deferred rendering render path
- On-tile Depth
- Application SpaceWarp optimizations
- Shader optimizations

For Meta VR Glasses builds, the **Meta Quest Built Profile** also allows for other performance improvements, which we'll look at in a later chapter.

The following slide from the Developing for Meta VR Glasses livestream shows some of the performance gains of configurations included in the **Meta Quest Built Profile**.

© 2026 Unity Technologies



OpenXR support

Because Meta VR Glasses support OpenXR, you can leverage Unity's cross-platform **XR packages** to build for Meta VR Glasses or easily port current OpenXR projects to them. Unity 6.6 and newer versions provide access to familiar tools such as the XR Interaction Toolkit (XRI), XR Hands, and the XR Interaction Simulator to design, build, and deploy your experiences.



Two Point Hospital Mixed Reality Edition: Look and pinch input

Unity provides additional features that you can use in your project to take advantage of Meta VR Glasses' hardware and hands-based input. To learn more about how to design an application for Meta VR Glasses, check out Meta's app design documentation.

© 2026 Unity Technologies

The sections that follow outline how to configure a new or existing Unity XR project for Meta VR Glasses. For more information about Meta VR Glasses, refer to Meta's Get started with Meta VR Glasses documentation.

Requirements

To build a project for Meta VR Glasses, your project must meet the following requirements:

- For this guide, use Unity 6.6 or later.
- Support eye and hand input to reach users without controllers.
- Target the **IL2CPP** backend.
    - You can set the scripting backend with the Scripting Backend Player setting. If you use the Meta Quest build profile, Unity sets the **Scripting Backend** to **IL2CPP** by default.
- Ship as a 64-bit binary.
    - You can set the size of the binary with the Target Architecture player setting. If you use the Meta Quest build profile, Unity sets the **Target Architecture** to **ARM64** by default.

Note: You can use a 32-bit binary for faster iteration in the Unity Editor, but the build you ship must target ARM64 with the IL2CPP backend.

Support eye and hand tracking for Look and Pinch input

The user's eyes and hands are the primary input for Meta VR Glasses. Your project should aim to support eye and hand tracking to target Meta VR Glasses because controllers aren't included.

Meta VR Glasses use **Look** and **Pinch** interactions to enable natural interactions with virtual content. With **Look** and **Pinch**, a user can look at an element to target it, then pinch their fingers to select it. For more information, check out Meta's Get started with hands and eyes documentation.

Unity's gaze and hands extensions

To enable interactions based on where the user is looking as well as hand recognition on Meta devices, several extensions are added to the XR Plugin management when installing the **XR hands** and **Unity OpenXR**: **Meta** packages.

Unity's **XR Hands** package allows you to access hand tracking data. Unity provides additional Meta-specific extensions that you can use to develop gaze and hands-based experiences for Meta VR Glasses.

© 2026 Unity Technologies

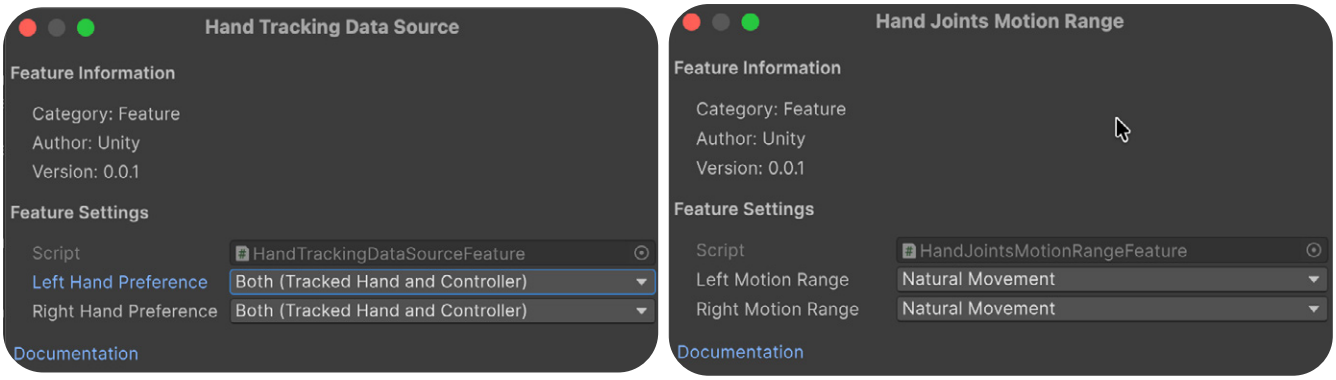
The following table outlines the Meta-specific extensions Unity provides for gaze and hand interaction:

Extension	Package	Description
Meta Hand Tracking Wide Motion Mode OpenXR feature	XR Hands	Use inference algorithms to estimate hand poses, even when hands are outside the normal camera tracking volume.
Meta Hand Tracking Frequency Hint OpenXR feature	XR Hands	Request higher frequency hand tracking updates for improved responsiveness.
Meta Social eye gaze	Unity OpenXR: Meta	Provides access to eye-tracking data, including the position and orientation of the user's eyes. Useful for driving avatar animations. For gaze-based interactions in Unity OpenXR, add the Gaze Interaction Profile and the Gaze Interactor prefab from XRI.

The following table lists other features that you can enable from the **OpenXR** menu that will enhance your hand tracking implementation.

Hand Tracking Data Source	XR Hands	Determines which input sources process hand data, whether tracking actual hand movements or simulating hands based on controller position (ideal when targeting hardware lacking native hand tracking). The default setting, "Both," allows the system to select the optimal data source dynamically.
Hand Joints Motion Range	XR Hands	Configures virtual hands to use their natural range of motion or appear as though they are holding physical controllers.
Extrapolated Hand Poses	XR Hands	Retrieves hand and joint data directly from the runtime to support custom extrapolation. Uninterpolated hand data can be selected when higher physics accuracy is required.

© 2026 Unity Technologies



Some of the new features available in the XR Plug-in Management settings.

Spatial audio

To enhance immersion, you can also use an audio spatializer plug-in, which uses physical characteristics of a scene, such as the distance and angle between an **AudioListener** component and an **AudioSource** component to modify the properties of sound transmitted to the user.

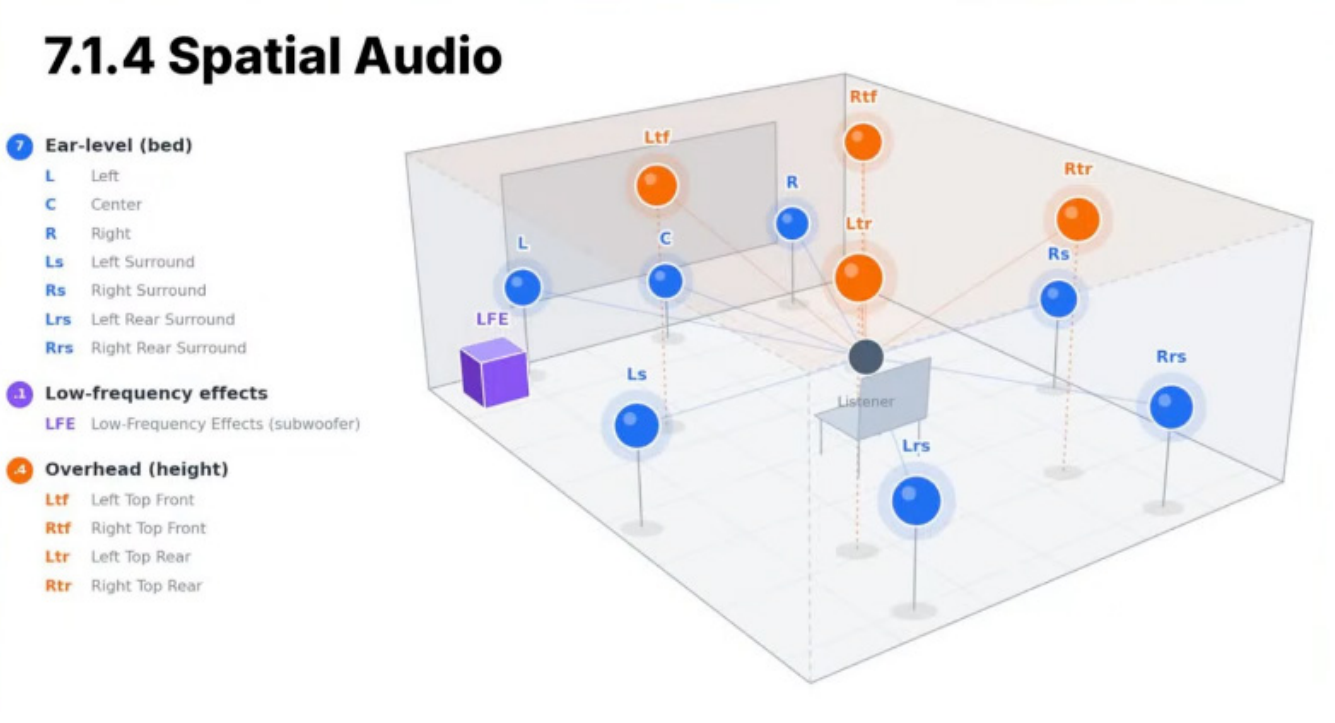


Diagram of cinematic 7.1.4 surround sound layout virtualized with spatial audio.

© 2026 Unity Technologies



Building a Hello World VR game for hand tracking

Building virtual reality experiences no longer requires thousands of lines of custom C# code. With Unity's **XR Interaction Toolkit (XRI)**, you can set up complete VR environments, construct interactive object mechanics, and deploy your build to modern VR devices using modular prefabs and no-code components.

In this section, you'll build a new test game where you'll put different colored balls onto their matching socket. To do this, you'll learn about and use modular prefabs, with your hands as the main way to control the game.

© 2026 Unity Technologies

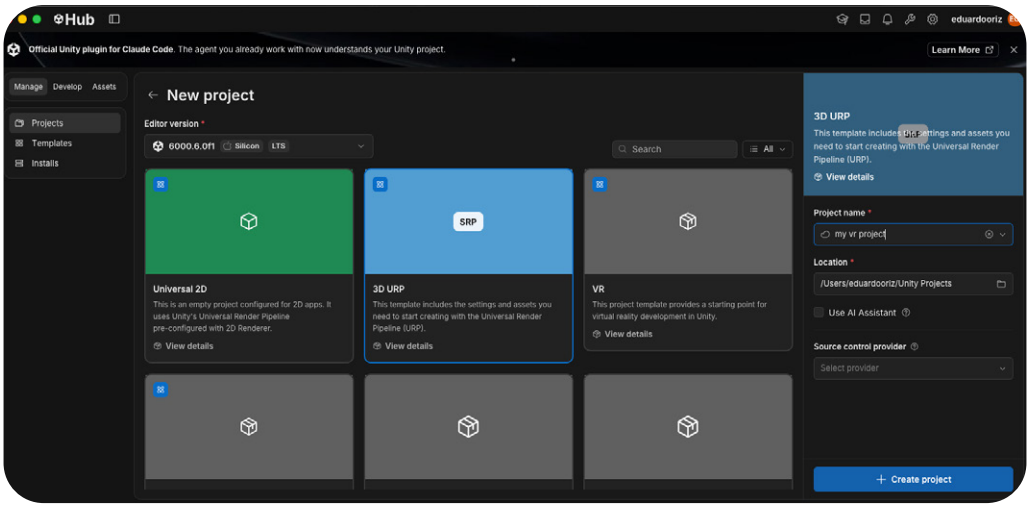
Step 1: Configure the Editor for VR development

There are two starting points you can choose from for this VR game: build XR functionality from an existing project (like a 3D URP project) or directly from the VR project template. The following is a reference table with the requirements needed in both cases to develop the project correctly:

Project Template	**Existing 3D (non-VR) URP project**	**New VR project template**
Packages required		
XR Plugin management	They need to be installed from Package Manager	Already included
XR Interaction Toolkit		
XR Hands		
Open XR Android XR		
Open XR Meta		
Configuration in Project Settings		
Enable Open XR	Has to be enabled from XR Plug-in Management	Already enabled
Enable Meta Quest feature group	Has to be enabled from XR Plug-in Management	Already enabled
Enable Hand Tracking Subsystem	Has to be enabled from Open XR settings	Already enabled
Enable Meta Hand Tracking Aim	Has to be enabled from Open XR settings	Already enabled
Enable Interaction Profiles	You have to add Hands, Meta Quest interaction profiles	You have to add Hand Interaction Profile
Samples that we'll use		
Starter Assets	Install it from XRI samples in Package Manager	Install it from XRI samples in Package Manager
Hands Interaction Demo	Install it from XRI samples in Package Manager	Install it from XRI samples in Package Manager

© 2026 Unity Technologies

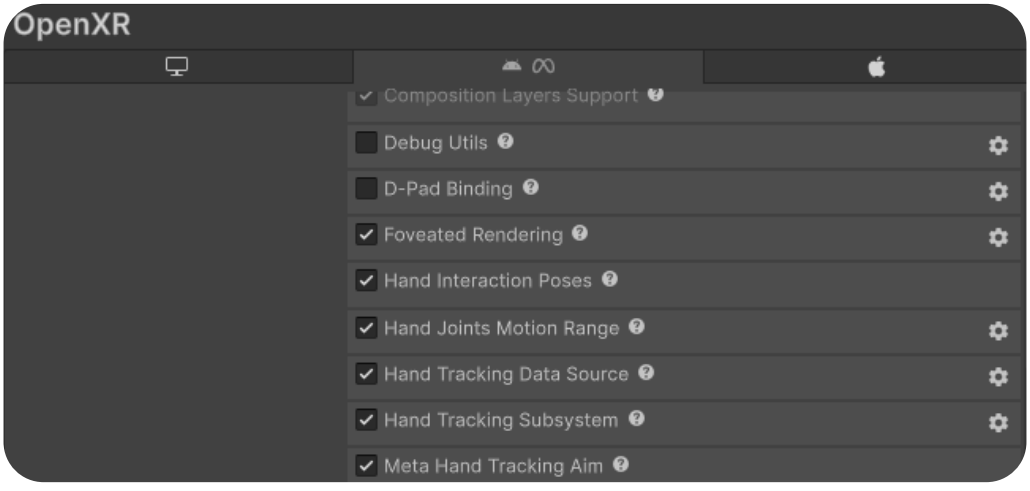
Project Validation		
Usage	Run it before testing, likely more warning to check	Run it before testing
Testing		
On device	Add a Build Profile for Meta	Add a Build Profile for Meta
XR Interactor Simulator	Enable Use XR Interaction Simulator in scenes inside XR Plug-in Management	Enable Use XR Interaction Simulator in scenes inside XR Plug-in Management



You can either start from a new **VR** template project, which comes with all the settings preconfigured, or you can bring the XR packages and settings to your existing 3D project.

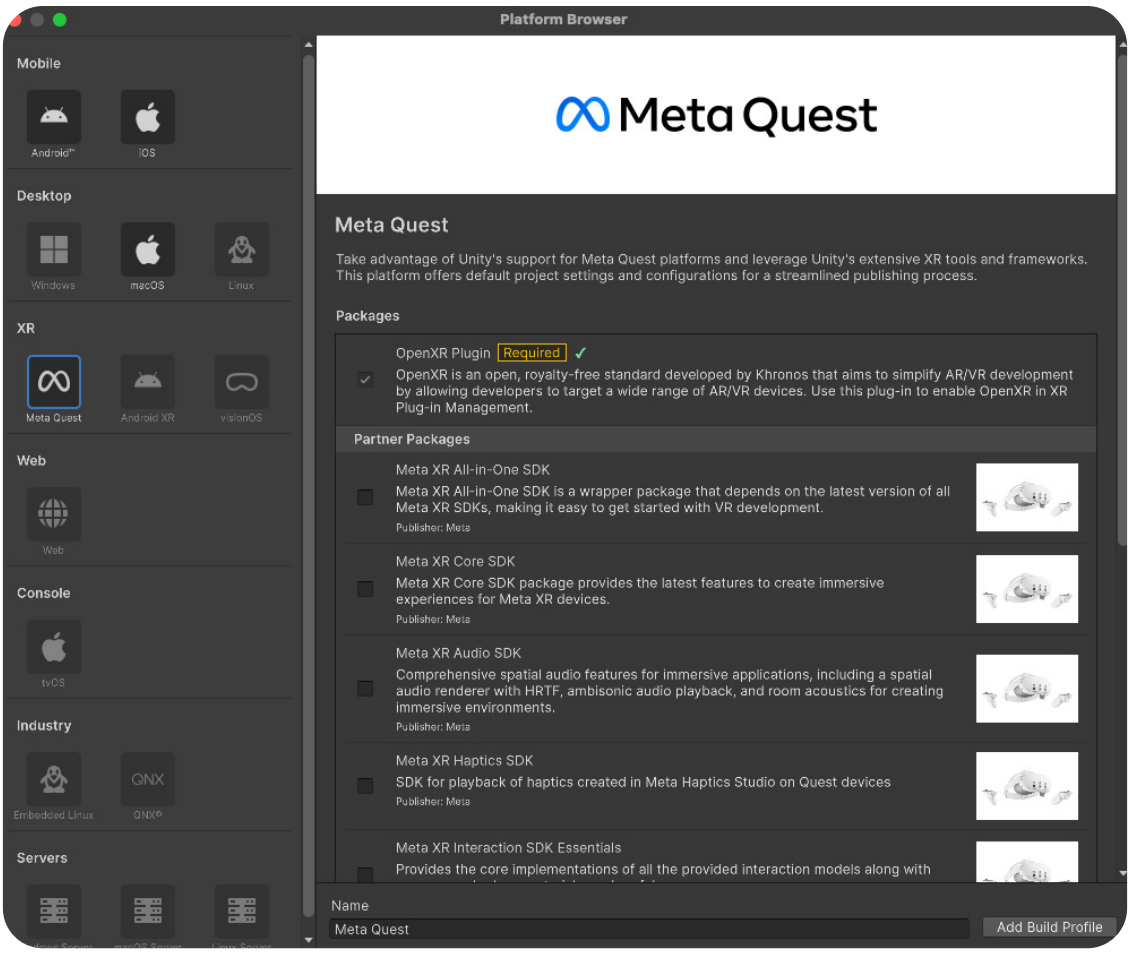
Important: Resolve any new Project Validation issues that might come up after changing settings.

Make sure **Hand tracking** capabilities are enabled from the Open XR menu if you're adding VR capabilities to your existing 3D project.



© 2026 Unity Technologies

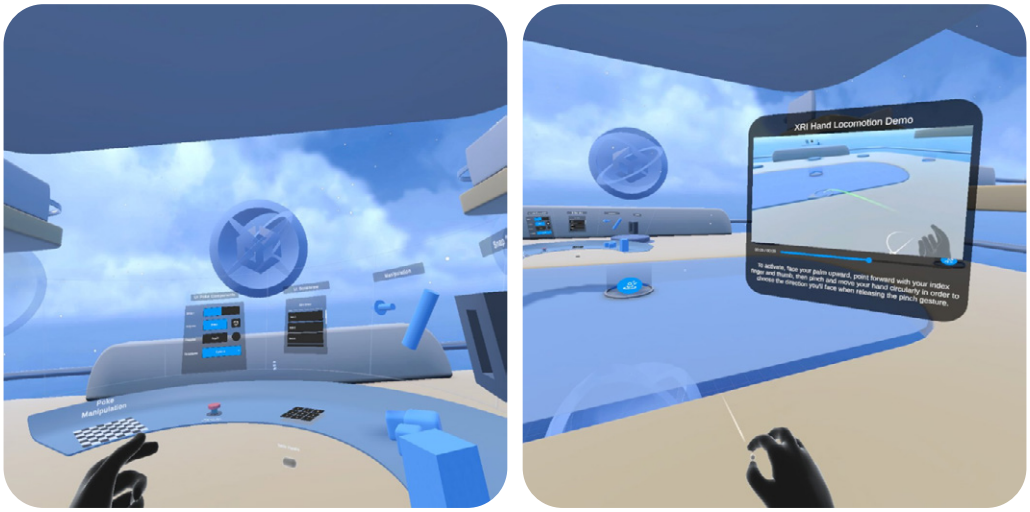
In both cases, you'll need to make sure to be using the correct Build Profile for your Meta device.



In the **XR Plugin Management** window (**Edit** > **Project Setting** > **XR Plugin Management**), enable **Use XR Interaction Simulator in scenes**, so you can continually test without having to take the VR device on and off.

Alongside the **Starter Assets** sample, you'll need to import the **Hands Interaction Demo** sample from the **Package Manager**, under the **All Samples** section. The demo scene in the **Hands Interaction Demo** sample contains elements similar to the **VR** template, but is mostly focused around hand gestures. In the **Hands Interaction** sample, you can test interactions and locomotion with hand controls, which are the primary input for Meta VR Glasses.

© 2026 Unity Technologies



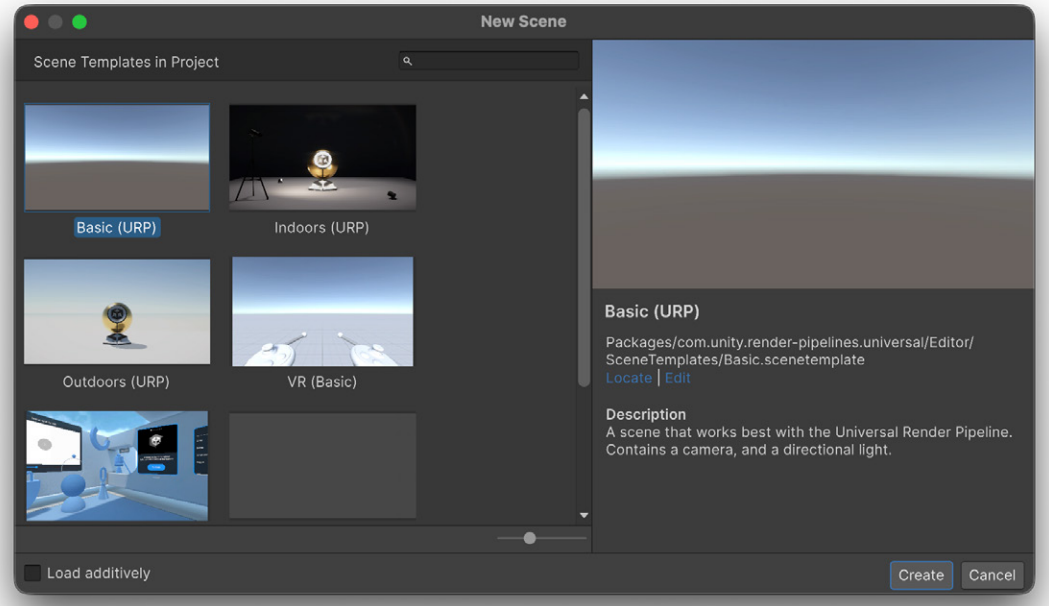
Hand Interaction Demo scene

Step 2: Initialize a new scene and add the XR Origin prefab

Once the project settings are configured, you can create the foundation of your scene:

1. Create a new 3D scene (**File** > **New Scene**) and select **Basic (URP)**.
2. Remove the default **Main Camera** from the **Hierarchy** window.
3. Add the **XR Origin Hands (XR Rig)** prefab from the **Hands Interaction Demo/Prefabs** folder (**Assets** > **Samples** > **XR Interaction Toolkit** > **3.5.1** > **Hands Interaction Demo** > **Prefabs** > **XR Origin Hands (XR Rig)**)

The **XR Origin Hands (XR Rig)** prefab handles tracking the user's XR device position, height, and hand controllers in 3D space.

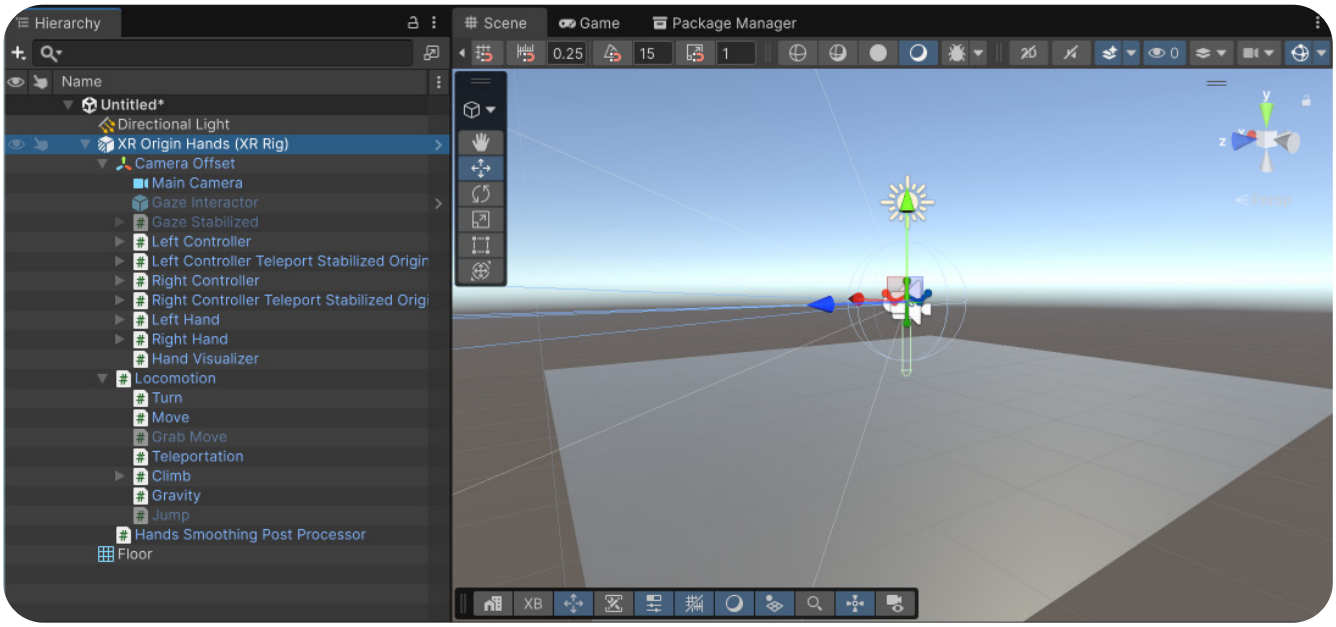


© 2026 Unity Technologies

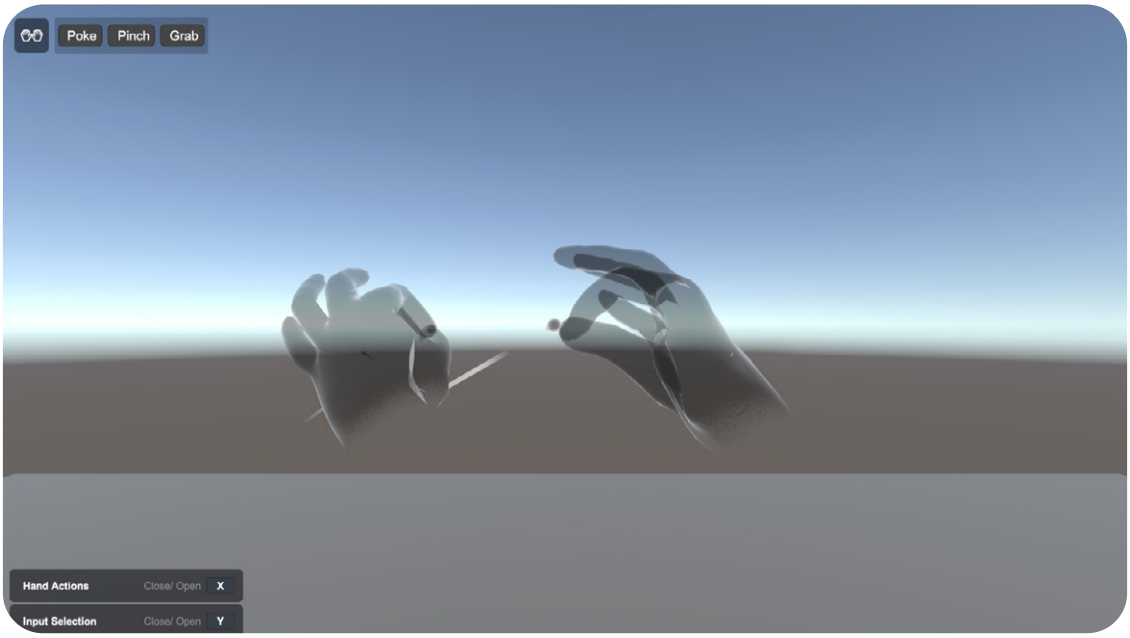
Step 3: Set up Environment Geometry and basic interactivity

Next, you'll build out your physical space and add grab mechanics:

1. Create a **Plane** GameObject (**GameObject** > **3D Object** > **Plane**) and name it "Floor".



The XR origin prefab contains all you need for hand / controller interaction and locomotion



2. Add a **Cube** GameObject (**GameObject** > **3D Object** > **Cube**), rename it "Table", and adjust its **Transform** component's values so it can be used as a table.
3. Create an empty GameObject and name it "GrabbableObject Blue".

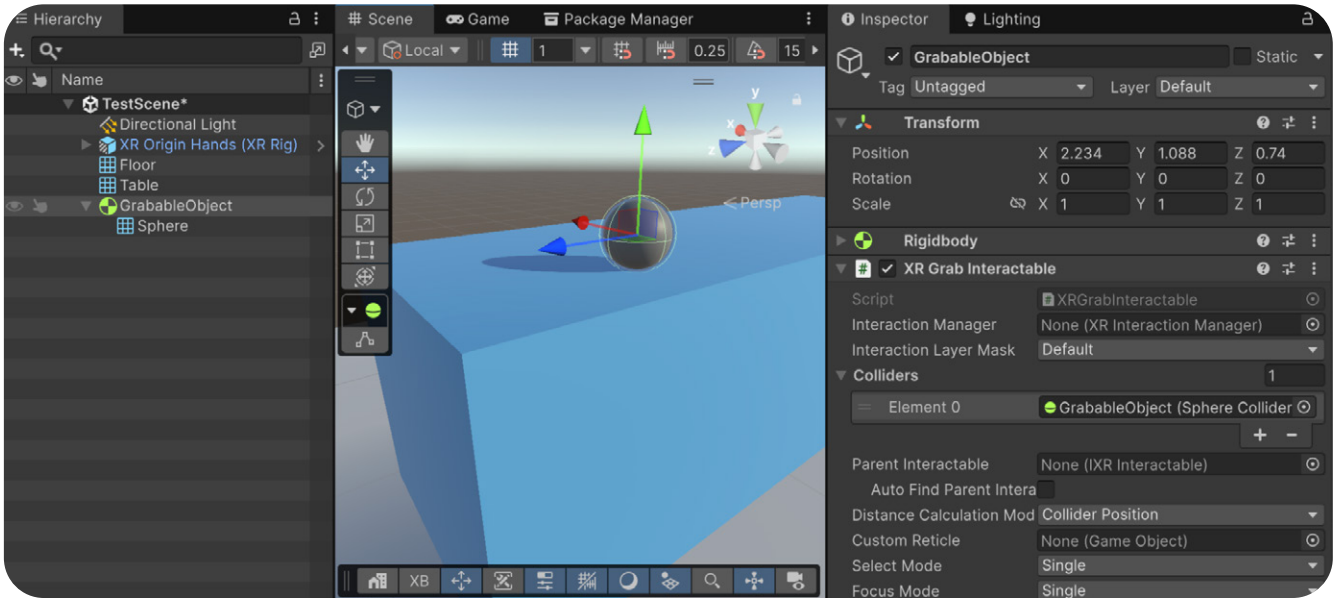
© 2026 Unity Technologies

This will be the GameObject that users pick up in your game.

4. Create a **Sphere** GameObject, rename it "Sphere Blue", add a blue material to it, make it a child GameObject of the **GrabbableObject Blue** GameObject, and give it a scale of **0.2** to make it more appropriate for the player's hands.

We suggest nesting the **Sphere Blue** GameObject inside the **GrabbableObject Blue** GameObject so scripts and components stay on the parent **GrabbableObject Blue** GameObject. When your art asset is finalized, you can then just swap out the **Sphere Blue** GameObject.

5. Place the **GrabbableObject** GameObject on top of the **Table** GameObject.
6. Remove the **Sphere Collider** component from the **Sphere Blue**GameObject, as you only want the visual (**Mesh Renderer** component) there.
7. Add the **XR Grab Interactable** and **Sphere Collider** component to the **GrabbableObject** GameObject, and adjust the **Radius** property to match the **Scale** property of the **Sphere Blue** GameObject.
8. In the **GrabbableObject** GameObject's **XR Grab Interactable** component, use the foldout (triangle) to expand the **Colliders** section, and add the **GrabbableObject** GameObject's **Sphere Collider** component.



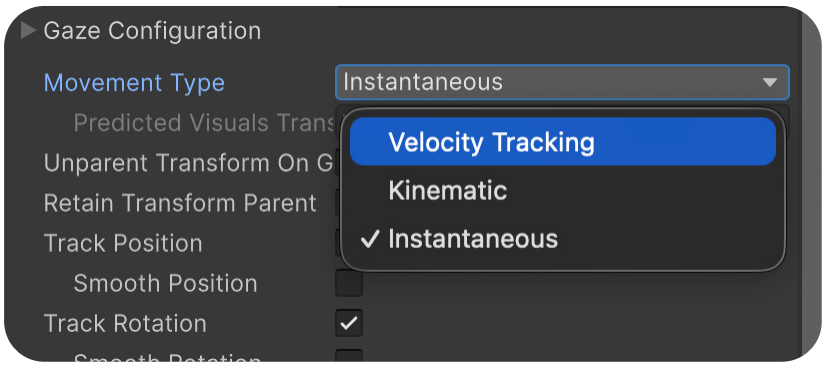
This collider is used for selecting and interacting with the interactable. If left empty, the XR Grab Interactable looks for colliders in child GameObjects.

The default **Movement Type** of grabable is set to **Instantaneous** by default, which means that the **GrabbableObject** GameObject will immediately attach to your controls, but it will move through other GameObjects.

9. Open the **Movement Type** property dropdown and select **Velocity Tracking**.

© 2026 Unity Technologies

Velocity tracking makes the **GrabbableObject** GameObject follow your controls' velocity, but still collide with other GameObjects in the way, like the **Table** GameObject. **Kinematic** makes the GameObject follow the physics rules without being affected by external forces like gravity or other GameObjects blocking the path.



There are many more features in the **XR Grab Interactable** component. One particular feature that you might want to set up is an **Attach Transform**, which can be an empty GameObject inside the **GrabbableObject** that will be the anchor position and rotation of your GameObject. This feature is particularly useful for GameObjects such as weapons, keys, or any shape with a handle.

Step 4: Configure Socket Interactors

Now you'll add the functionality that allows GameObjects to snap into specific slots or containers (for example, putting a key into a lock or placing a tool on a belt) you'll use a **Socket interactor**:

1. Create a **Cube** GameObject, name it "Socket Blue", add a blue material to it, and make it a similar size to the **Sphere Blue** GameObject.

This will be a small platform the sphere will sit on.

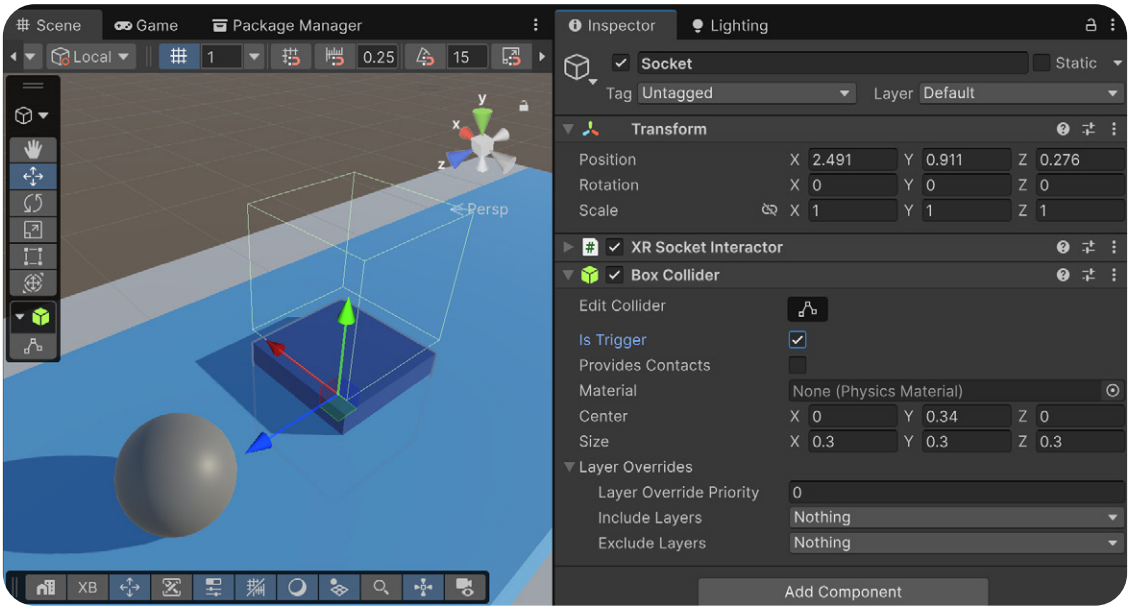
2. Add the **XR Socket Interactor** component to the **Socket Blue** GameObject.

This will cause the **Socket Blue** GameObject to snap the **Sphere Blue** GameObject into place and stay attached to it.

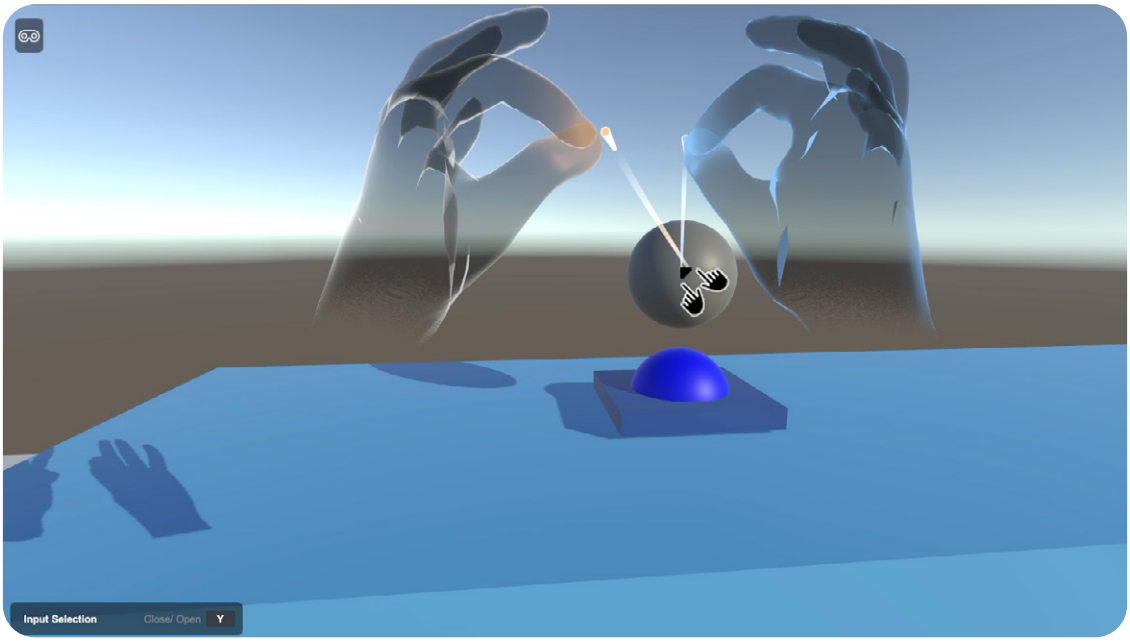
3. Add a **Box Collider** component to the **Socket Blue** GameObject, enable the **IsTrigger** property, and offset the **Y** position so the collider sits slightly above the **Socket Blue** GameObject, but this time we'll displace it.

As soon as the **Sphere Blue** GameObject comes into contact with the **Box Collider** component, it will get into position.

© 2026 Unity Technologies



4. Create an empty GameObject where you want the snapping spot to be and add a **3D Trigger Collider** component.



Upon contact with the collider we can previsualize where the interactable object will snap into position.

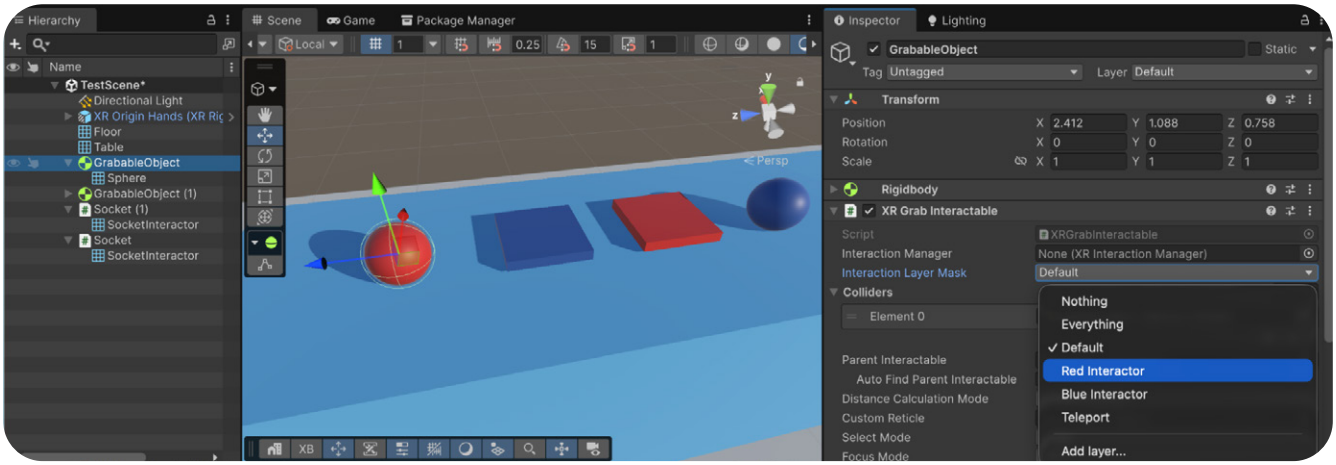
5. Adjust the **Attach Transform** point so the **Sphere Blue** GameObject snaps into the exact desired orientation when placed on the **Socket Blue** GameObject.

You can repeat the same process as before: create an empty GameObject, place it inside the **Socket Blue** GameObject, and use its transform for the **Attach Transform** position.

In a real game, you'll want to configure the socket filters so it only accepts specific **XR Grab Interactables** (for example, ensuring only a specific key triggers the socket). To do this, you can make use of **Interaction layers**.

© 2026 Unity Technologies

6. Create duplicates of the **Sphere Blue** GameObject,**Socket Blue** GameObject, and **GrabbableObject Blue**, rename them “Sphere Red”, “Socket Red”, and “GrabbableObject Red”, respectively, and add a red material to the **Sphere Red** and **Socket Red** GameObjects.
7. Select **Add layer** inside the **GrabbableObject** GameObject. Create a new layer for the blue elements and one for the red elements and make sure the **Socket Blue**, **Socket Red**, **Sphere Blue**, **Sphere Red**, **GrabbableObject Blue**, **and GrabbableObject Red** GameObjectsuse the corresponding interaction layer.
8. You can also filter by **Interactor**, for example, to allow interacting with **Gaze Interactor** but not with **Near-Far interactors**.



To create the red elements, duplicate the blue Socket and GrabbableObject, change the color and create a new Interaction Layer for them

Step 5: Implement teleportation and movement

To give users a way to move around the environment without motion sickness, there are two components you can use, depending on the use case:

— **Teleportation Area**: Add an **XR Teleportation Area** component to your ground/floor mesh. This allows users to point their controller beam anywhere on that surface to move.

— **Teleportation Anchor**: For designated landing zones (for example, standing at a workbench), create a GameObject and add an **XR Teleportation Anchor** component to it to force the user to face a specific direction upon arrival. This is what you’ll use in your test game.

To implement teleportation and movement in the scene, follow these instructions:

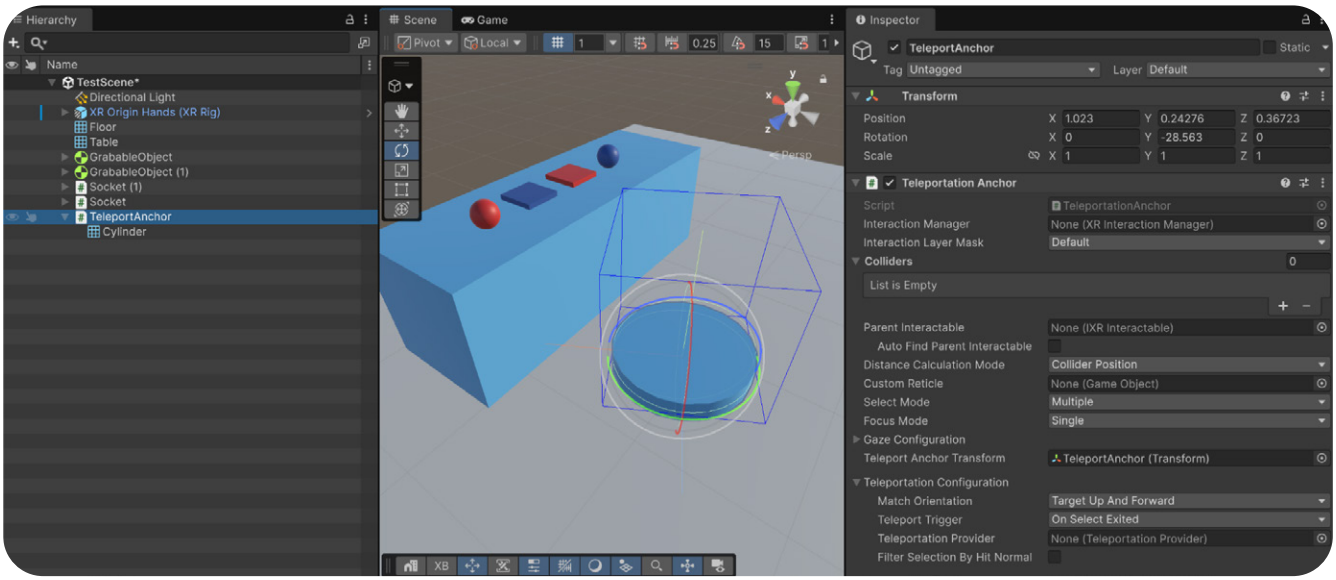
1. Create an empty GameObject, name it **TeleportAnchor**, and assign the **Teleportation Anchor** component to it.

This will display an area representing the teleportation spot.

© 2026 Unity Technologies

2. Create a **Cylinder** GameObject and add it as a child GameObject of the **TeleportAnchor** GameObject.
3. Change the **Match Orientation** property to **Target Up** and **Forward**.

This allows users to face the same angle as the TeleportationAnchor GameObject. It also includes configurable events, and can use Interaction Layers.



4. Ensure the **XR Origin** controllers have an **XR Ray Interactor** configured for line-of-sight teleportation visuals.

Step 6: Test locally with the XR Interaction Simulator

You don't need to put on a physical XR device every time you tweak a setting; to test locally with the XR Interaction Simulator, follow these instructions:

1. Enable the **XR Interaction Simulator** (**Project Settings** > **XR Interaction Toolkit** > **Enable XR Interaction Simulator**).
2. Enter Play mode.
3. Use mouse and keyboard controls to simulate head movements, controller positions, and hand interactions (grabbing, pointing, teleporting) directly within the Game view.

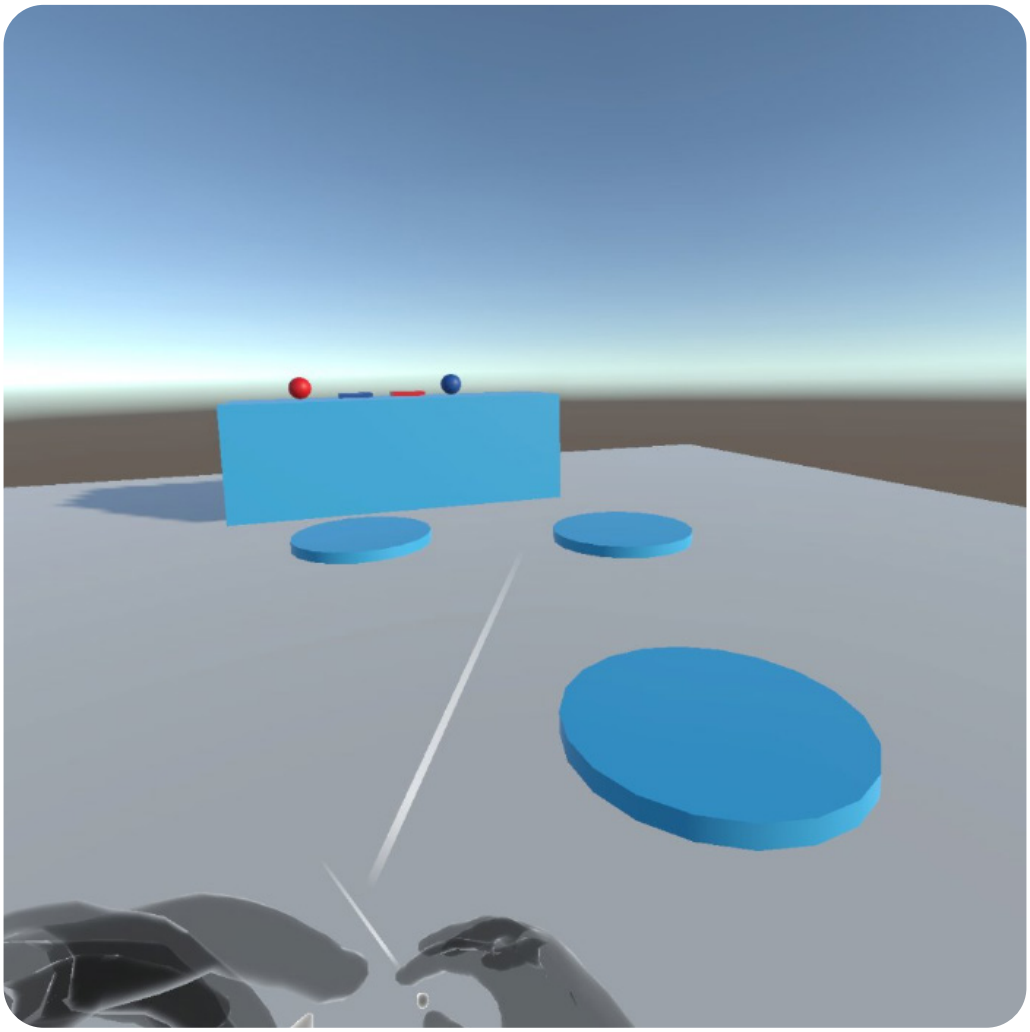
Step 7: Build and deploy to VR hardware

Once the scene functions as intended in the simulator, follow these instructions to build and deploy the scene to your VR hardware:

1. Connect your standalone VR device (For example, Meta VR Glasses or Quest 3) via USB-C.

© 2026 Unity Technologies

2. Open **Build Settings** (**File** > **Build Profiles**) and switch to your target platform (Meta Quest).
3. Add the scene you created to the **Scene List** and unselect all other ones.
4. Select **Build and Run** to package the application and deploy it directly onto the hardware device.



You should now have a fully functioning XR game complete with hands as controllers, teleportation anchors that allow the user to move around the scene, grabable objects, and sockets that snaps the corresponding grabable object into place!

© 2026 Unity Technologies

Building with Unity CLI and AI

For nearly two decades, developers have relied on the graphical interface of the Unity Editor to visualize and manage projects. However, clicking through panels and menus can become a bottleneck when handling repetitive tasks, running automations, or integrating AI-driven workflows. At the same time, having AI agents navigate graphical user interfaces by interpreting screenshots requires heavy processing, consumes tens of thousands of tokens, and slows down execution. The new Unity CLI and Unity Pipeline package solve both of these issues by giving both human developers and AI agents a fast, lightweight command-line interface directly into the Unity Editor. You can also get Meta VR specific knowledge by adding in the Meta VR CLI, which covers everything from VR performance requirements, spatial UI, Quest debugging, and more.



© 2026 Unity Technologies

How it works

The setup consists of two key components working in tandem:

1. **Unity CLI (Standalone Binary)**: Manages your local environment: installing Editor versions and platform modules, handling licensing, and creating or opening projects.
2. **Unity Pipeline Package**: Installed directly into your project, it transforms the active Unity Editor into a local background server.

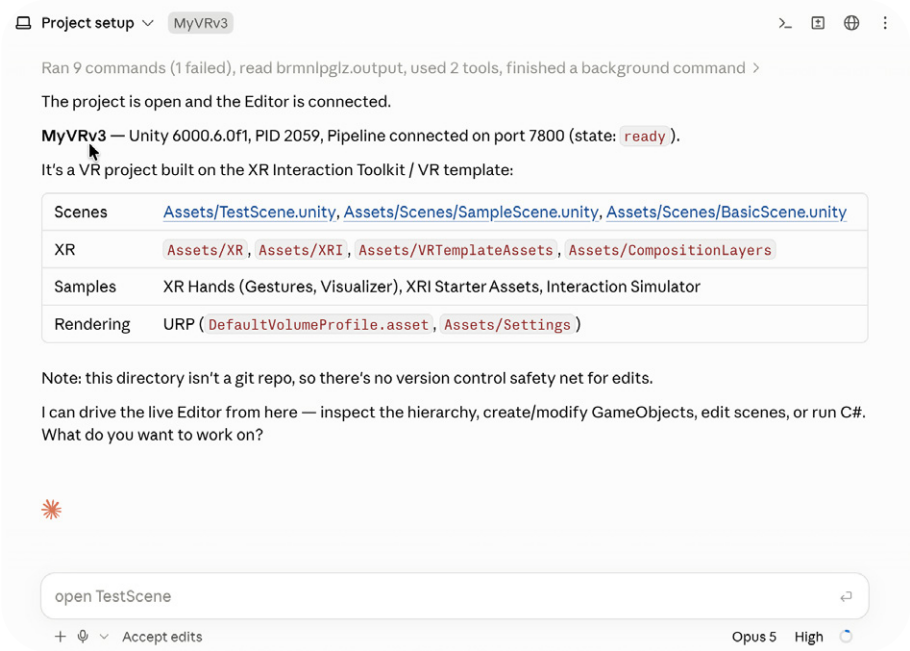
Because the Editor remains running in the background, project scenes and asset databases stay warm in memory. This eliminates cold-start engine boots for every command.

Getting started requires just a few commands:

1. **Install Unity CLI**: Use the platform installer script (or install via Homebrew on macOS/Linux) Check out the documentation for a more in depth installation guide.
2. **Install the Pipeline**: Navigate to your **Project** folder in the terminal and run “unity pipeline install” or ask your AI agent to install it for you. You can also add --force to force the latest version. For more information, refer to the commands indicated in the Unity Pipeline package documentation.

Unlocking agentic workflows

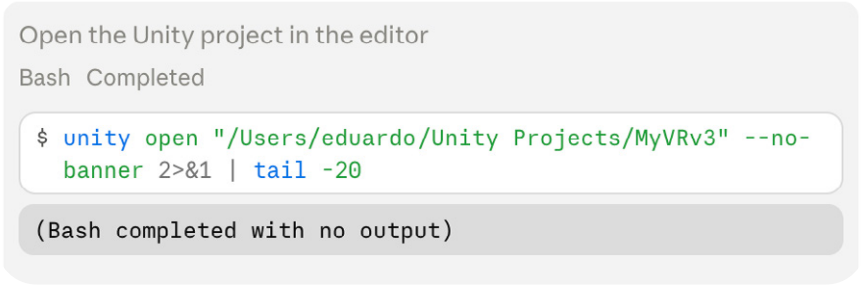
The biggest advantage of the Unity CLI is native AI agent compatibility. Whether you use Claude Code, Codex, Cursor, Copilot, or Gemini CLI, any agent capable of running shell commands can control your Unity Editor. The agent simply sends CLI commands and receives structured JSON or plain text responses, enabling single-prompt workflows like generating scripts, configuring components, template-based project setup and launching Play mode instantly for automated testing and iteration.



Ready to work on our VR scene via Claude and Unity CLI

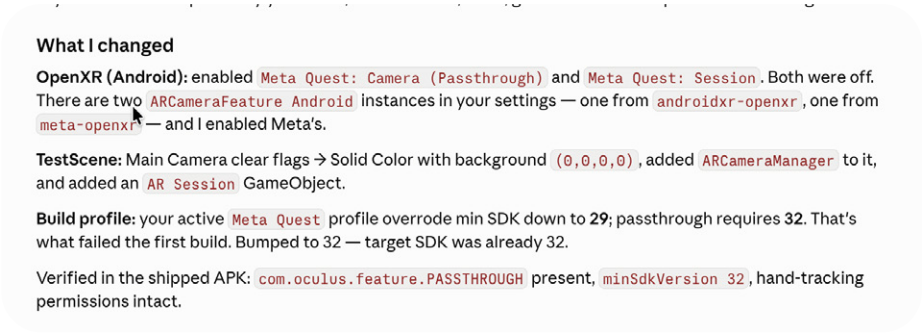
© 2026 Unity Technologies

You can ask your AI agent to control the Editor, build your application, or help fix errors using **Unity CLI. Meta VR CLI** adds Meta-specific development knowledge and tools, while **Meta XR Operator** lets the agent interact with the running XR app and inspect the result. Together, they support a build-test-verify workflow rather than stopping at a successful compilation. Meta XR Operator setup is covered in the **Meta utilities** section.



For some of the tasks Claude is running CLI commands

With project and Editor access, you can let the agent take control of required packages or SDKs, for example, using the previous example, you can ask to enable Camera Passthrough for the environment and make it an MR experience. If there are missing packages or settings, the agent can often identify and set them up, though it's worth reviewing the changes.



Unity official plugins

AI agents will make things work, but perhaps not in the way that Unity workflows are meant to be used. The official plugins for Claude and Codex teach the agents how to do a certain job the way it was actually meant to be done: where a Unity project keeps its assets, how an atlas is supposed to be built, what a URP renderer feature has to satisfy before it will run at all. At launch, the plugin installs 29 skills (more will be added over time) with a single command, with no configuration, no per-project setup, and nothing to copy between machines when you move to a new one. None of that is new work for a Unity developer, because every one of those jobs is something you already do in a normal week. What changes is that the agent now approaches them holding Unity's official answer instead of assembling one from third-party sources. You can now spend more time on the result and less time worrying about correcting the approach.

Currently, there are official plugins available for both Claude and Codex, with more coming soon.

Unity's official plugin for Claude
Unity's official plugin for Codex

© 2026 Unity Technologies

Optimize graphics for Meta VR Glasses



© 2026 Unity Technologies

The Meta VR Glasses feature a larger on-chip GPU memory compared to the Meta Quest 3. This enhancement allows the tiled renderer to divide each frame into fewer tiles, thereby improving GPU performance. To fully utilize this capability, several new features are available to help you create highly optimized applications.

- Gaze-based foveated rendering, with optional dynamic foveation – Renders full resolution at the point where the user is looking and lower resolution elsewhere, with dynamic foveation adjusting the foveation level as GPU load changes.
- Subsampled layout (OpenXR) – Optimizes how eye textures are sampled to improve foveated rendering performance.
- Quad Views (OpenXR) – A form of foveated rendering that draws the central focus area at full resolution and the periphery at lower resolution, reducing render time while maintaining perceived visual quality.
- Shader optimizations – Automatic and configurable shader optimizations that reduce GPU frame time, shader instruction count, and memory bandwidth, and improve GPU occupancy.
- Application SpaceWarp (OpenXR) – Lets the application render fewer frames while the runtime generates intermediate frames from motion and depth data to maintain a high frame rate.
- Automatic viewport dynamic resolution – Keeps your project's render resolution within a specified range to maintain a stable frame rate and improve graphical performance.
- Adaptive Performance – Reports CPU and GPU utilization from the OS so your app can detect thermal or workload pressure and adjust quality settings, resolution, or effects to hold a stable frame rate.

The following sections outline the ways you can use these features in your project's graphics to get the most from Meta VR Glasses' higher-resolution display.

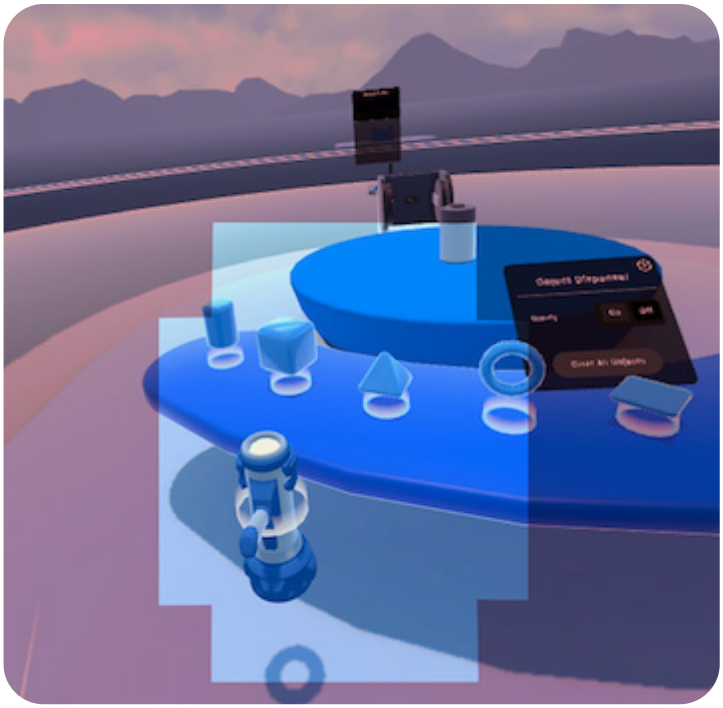
Configure foveated rendering

Foveated rendering can provide performance gains in your Meta VR Glasses project because it only renders the area the user is looking at in full resolution, reducing the GPU work needed for each frame.

If you're porting an existing Quest 2 or Quest 3 project to the Meta VR Glasses, we recommend reevaluating the type of foveated rendering your project uses. Because of differences in hardware, fixed foveated rendering (FFR) returns a smaller saving on Meta VR Glasses compared to Quest 3.

© 2026 Unity Technologies

On projects that target Meta VR Glasses, you can either use gaze-based foveated rendering or Quad Views. To learn more about the differences, and when to use each optimization, refer to Compare Quad Views and foveated rendering (OpenXR package documentation).



Example of foveated rendering. The tinted area shows the regions of the screen that are rendered at lower resolution to use fewer GPU resources

Tip: Always benchmark your app to confirm which technique performs best for your project.

Gaze-based foveated rendering

For projects that target Meta VR Glasses, gaze-based foveated rendering results in better performance than FFR because it provides full resolution at the gaze point, and lower resolution elsewhere.

You can also enable dynamic foveation to adjust the foveation level as GPU load changes, staying within the maximum level you set while the high-resolution area follows the user's gaze when eye tracking is enabled. This setting requires OpenXR 1.19 preview and a nonzero foveation level.

To learn how to configure gaze-based and dynamic foveation in Unity, check out the Enable and control foveated rendering documentation.

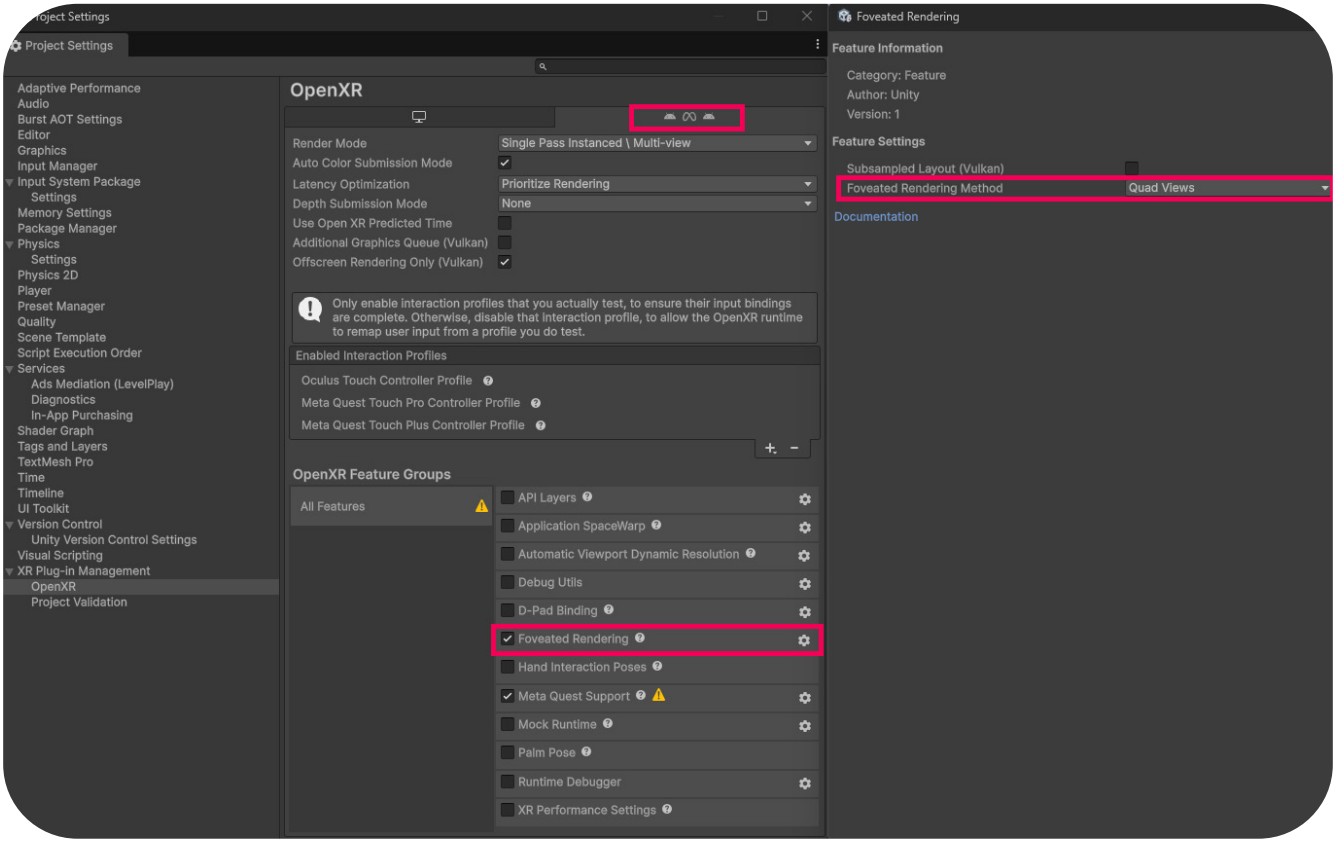
© 2026 Unity Technologies

Enable subsampled layout

You can improve foveated rendering performance with Subsampled layout (OpenXR, Vulkan only), which optimizes eye texture sampling. This means the GPU reads fewer samples from the low-resolution peripheral regions of the eye texture, which reduces memory bandwidth and improves frame time. With SpaceWarp, benchmark total app and compositor cost.

Quad views

Quad views (OpenXR) is a type of foveated rendering that renders peripheral areas at lower resolution and the central focus area at full resolution. Enabling Quad Views reduces render time while maintaining perceived visual quality.



In XR-Plugin management. Under OpenXR Feature Groups, select All Features and enable the Foveated Rendering feature. Click the gear to open the Foveated Rendering feature sub-options. Select Quad Views under Foveated Rendering Method and use Single Pass Instanced/Multi-view.

Note: Quad Views can result in an increased CPU cost. Profile your application to understand whether the GPU gains of Quad Views offset the CPU overhead and result in a net performance gain.

© 2026 Unity Technologies

Shader optimizations for Meta XR

Unity automatically enables shader optimizations for Meta Quest build targets to reduce GPU frame time, shader instruction count, and memory bandwidth, and improve GPU occupancy.

Unity also provides additional shader optimizations that you can configure in your project.

For more information, check out the Configure shader optimizations for Meta Quest documentation.

Configure Application SpaceWarp

Application SpaceWarp is an optimization for OpenXR that helps applications maintain a high frame rate. It allows the application to render fewer frames while the runtime generates intermediate frames using motion and depth data.

For more information, check out the Application SpaceWarp in OpenXR documentation.

Enable dynamic resolution

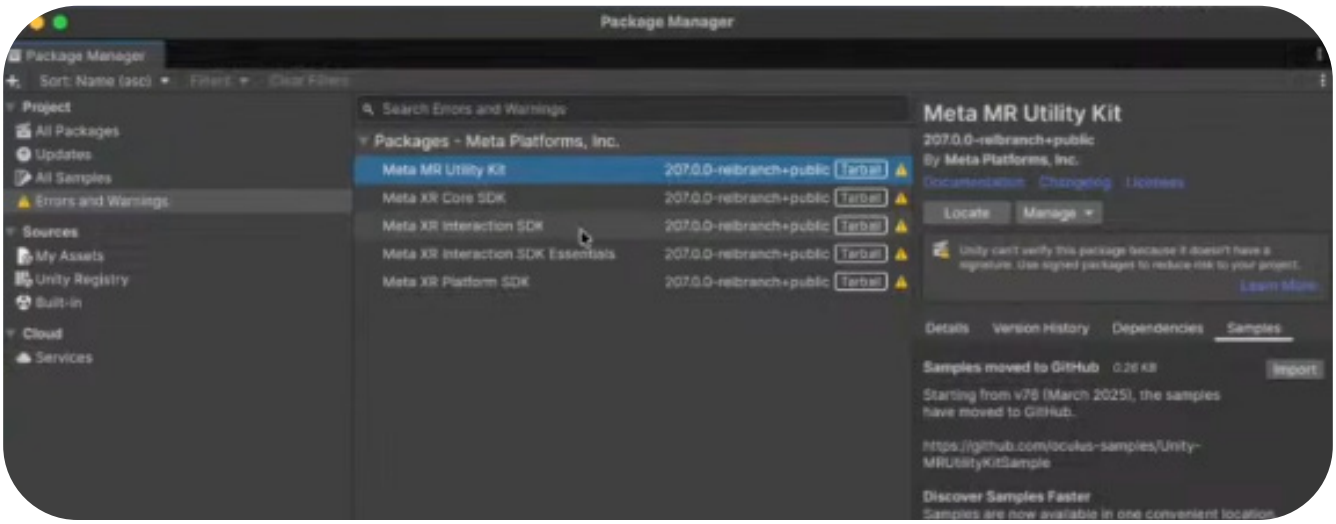
Automatic viewport dynamic resolution controls the resolution of your XR project, and keeps it within a specified range; it requires Vulkan, a supported OpenXR provider and URP without compatibility mode. Enabling automatic viewport dynamic resolution can maintain a stable frame rate and improve graphical performance. For more information, check out the Automatic viewport dynamic resolution documentation.

Use dynamic resolution and Adaptive Performance to maintain the target frame rate under GPU load and increase image quality when performance headroom is available.

© 2026 Unity Technologies

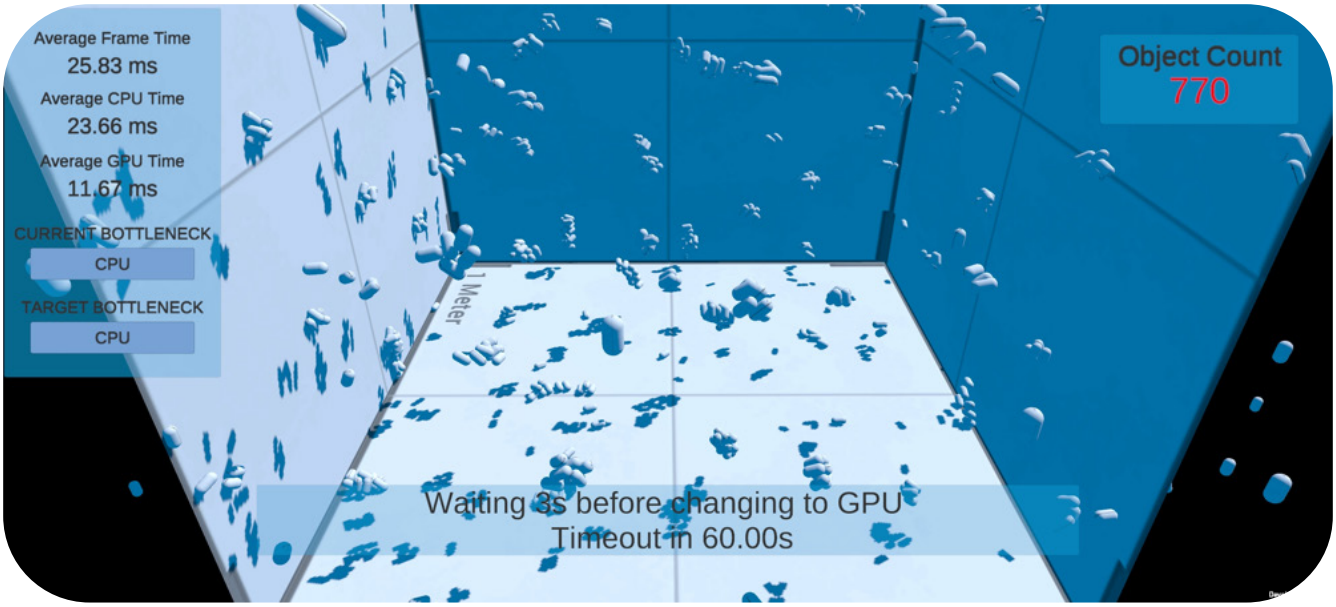
Enable Adaptive Performance

The **Adaptive Performance basic provider** reports CPU and GPU utilization from Meta VR Glasses to the Adaptive Performance system.



To use Adaptive Performance in your project, enable Adaptive Performance from Project Settings, and select Basic Provider from the Providers list.

Your app can read this data to detect thermal or workload pressure on the device and adjust quality settings, resolution, or effects to maintain a stable frame rate.



Adaptive Performance samples in the documentation

For more information, refer to OpenXR Meta documentation.

© 2026 Unity Technologies



Meta SDKs

Designed as a comprehensive toolkit for spatial computing, the Meta XR All-in-One SDK consolidates Meta's primary developer tools to streamline the creation of VR and MR applications for Meta VR devices.

Meta Quest

All-in-One

Meta XR All-in-One SDK is also available on the Unity Asset Store.

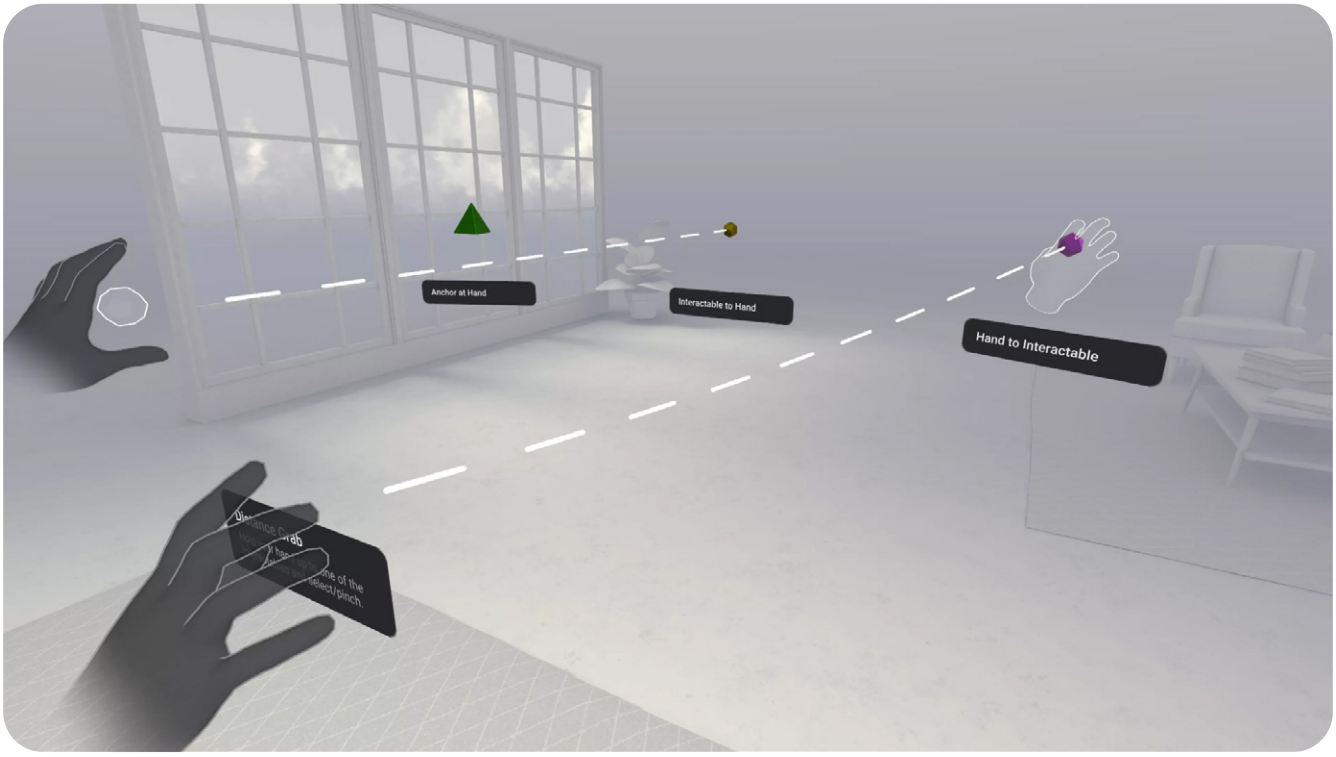
For developers choosing between Meta workflows and Unity's XRI ecosystem, the primary difference lies in framework integration. Through Unity OpenXR, Unity's XRI package is designed to work with AR Foundation (with support for Meta devices via OpenXR: Meta) while Meta's SDK doesn't require AR Foundation as it handles Meta device's tracking, passthrough, controls on its own. Consequently, transitioning between Unity's package and Meta's SDK requires adapting to different API structures. However, both frameworks build upon identical OpenXR APIs under the hood.

If you're developing a cross-platform project you will likely want to choose Unity's XRI, if your project is solely targeting Meta VR devices, going with Meta's SDK and its building blocks can be a suitable option.

© 2026 Unity Technologies

As long as both SDKs support the same underlying features, developers can access those features through either path, though behavior and performance can differ. In the following list you can find some of the packages included in the Meta XR All-in-one SDK:

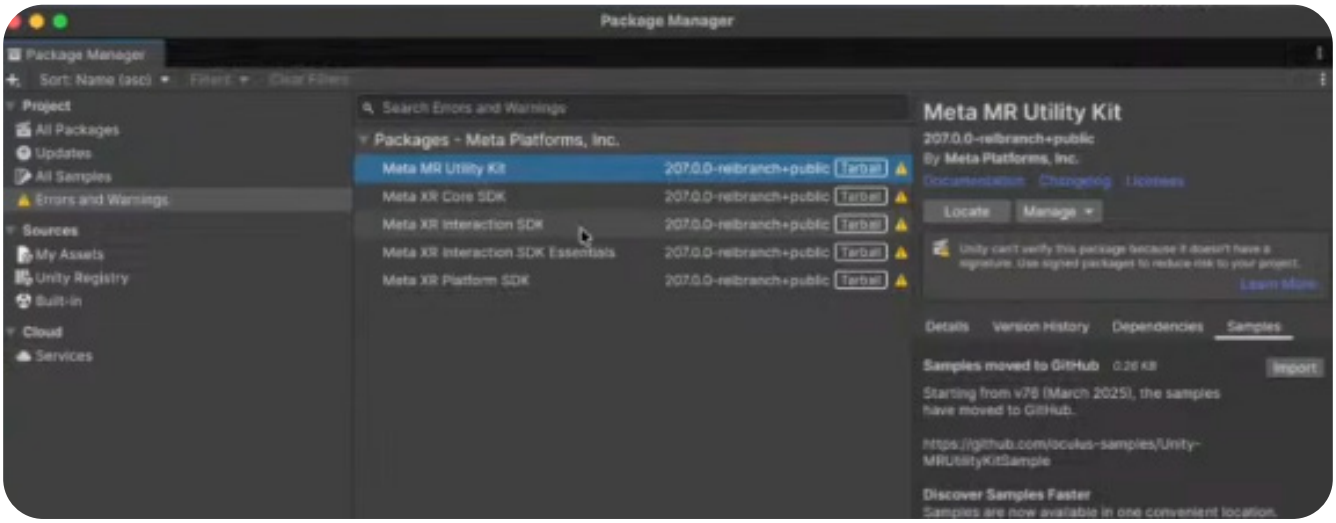
- **Meta XR Core SDK**: The foundational SDK required by the Interaction SDK OVR integration for Quest and Meta VR Glasses development. Handles rendering, camera rigs, tracking, and lower-level mixed reality features like Passthrough and Spatial Anchors. It also includes **Meta XR Operator** as an optional, experimental component for agent-driven runtime testing.
- **Meta XR Interaction SDK & Interaction SDK Essentials**: Modular tools and components for adding interactions like ray casting, poke, locomotion, grabbing, hand tracking, pose detection, and controller inputs.
- **Meta Mixed Reality Utility Kit (MRUK)**: High-level utilities built on top of the Core SDK's Scene API to simplify building spatially-aware apps (for example, scene manipulation, content spawning, and debugging).
- **Meta XR Platform SDK**: Integrates Meta Horizon social and platform services, such as user identity, matchmaking, in-app purchases, achievements, cloud storage, and multiplayer features.
- **Meta XR Audio SDK**: Provides spatial audio features, including HRTF-based 3D sound, ambisonics, and room acoustics simulation.
- **Meta XR Haptics SDK**: Allows integration of customized controller haptic feedback created with Meta Haptics Studio.
- **Meta XR Simulator**: A developer tool for testing and debugging Quest app features directly inside the Unity Editor without putting on a VR device.



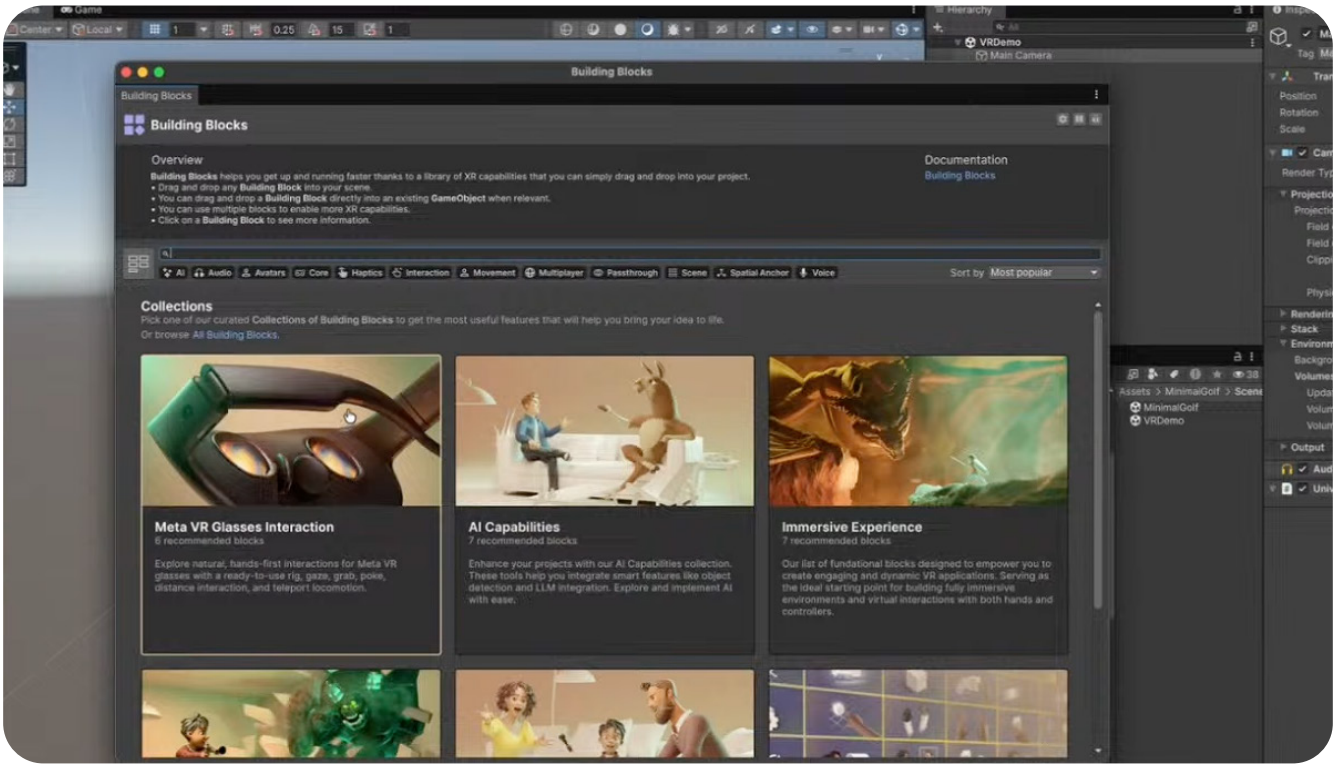
Example of Meta XR Interaction SDK

© 2026 Unity Technologies

The **Meta Interaction SDK** offers comprehensive support for hand gestures; its OVR integration relies on the Meta XR Core SDK. The Meta Interaction SDK features the Meta XR Project Setup Tool, which helps streamline project configuration by guiding developers through key recommendations, similar to configuring XR Plug-in Management, saving significant setup time.



Meta's SDK includes new building blocks for Meta VR Glasses, accessible through the Meta XR SDK menu once you install the Meta All-in-One SD. Meta building blocks are the equivalent to Unity's prefabs that come with XRI, which essentially provide all the pieces needed for you to build your VR application.



© 2026 Unity Technologies

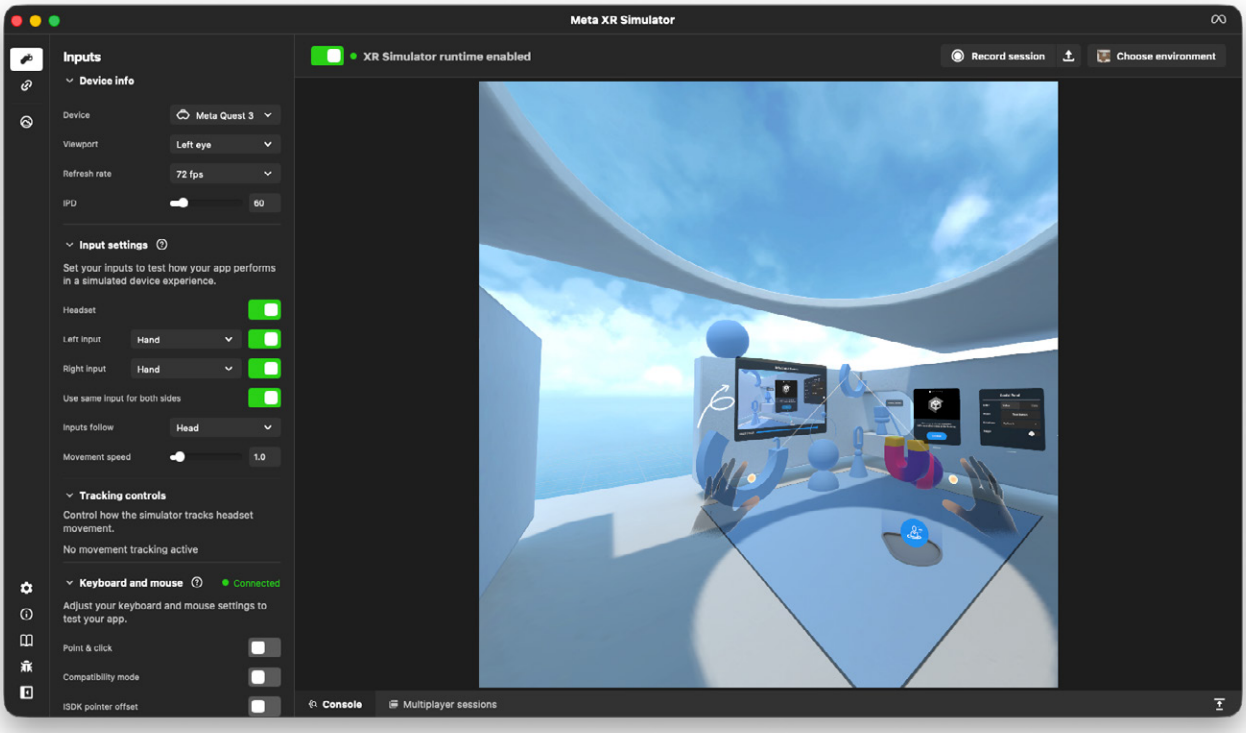
Meta utilities

Meta's developer tools help you simulate XR features, test the running experience, and configure your devices. **Meta XR Simulator** provides the desktop runtime, **Meta XR Operator** connects that runtime to an AI agent for testing and verification, and **Meta Quest Developer Hub** supports device management and deployment.

Meta XR Simulator

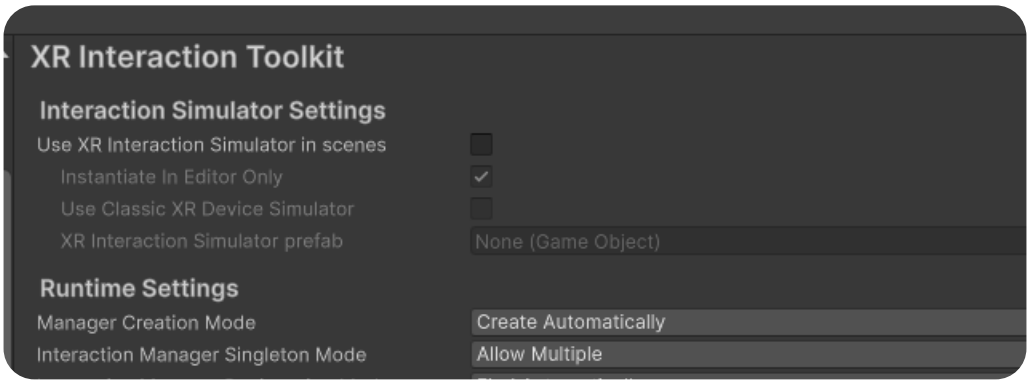
The Meta XR Simulator is a lightweight, desktop standalone app that's developer-focused and designed to accelerate building and testing VR and MR applications without needing to put on a VR device. By running directly on your development machine, the Meta XR Simulator allows you to simulate Meta VR devices at the API level; speeding up iteration times for everything from spatial anchoring and Passthrough to multiplayer session testing and input simulation. It will also allow you to switch from controllers to hand tracking simulation.

© 2026 Unity Technologies



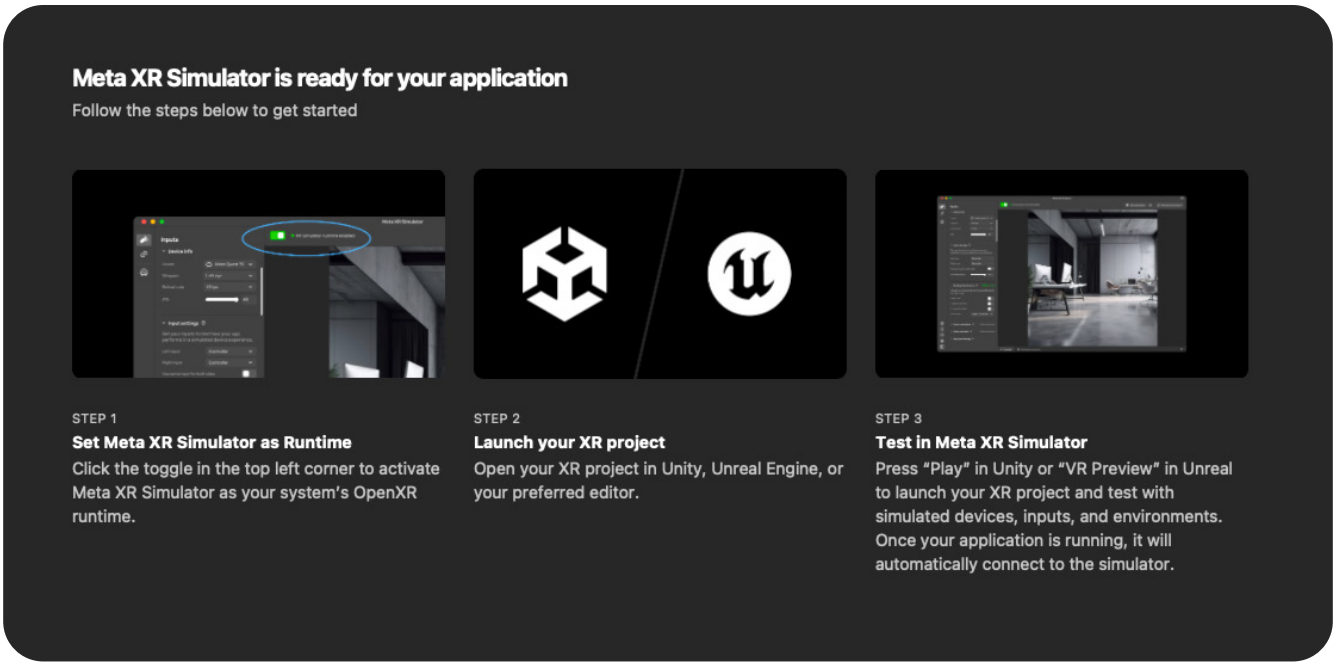
You can pair Meta XR Simulator with **Meta XR Operator** to let an AI agent drive simulated inputs and inspect the application. The next section explains how to connect them. For manual iteration, you can continue using the Simulator's keyboard and mouse controls.

Important: To avoid clashes, disable Unity's XR Interaction Simulator while you use Meta's XR Simulator.

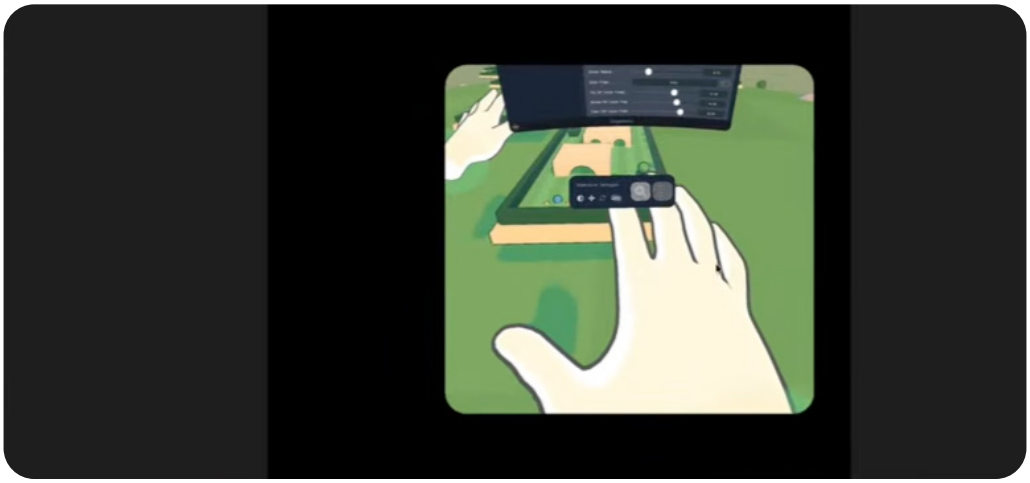


© 2026 Unity Technologies

You need to install the Meta XR Simulator app and enable XR Simulator, in the upper-left corner.



Note: Make sure the **Game** view is displayed in the Unity Editor while you use the Meta XR Simulator.



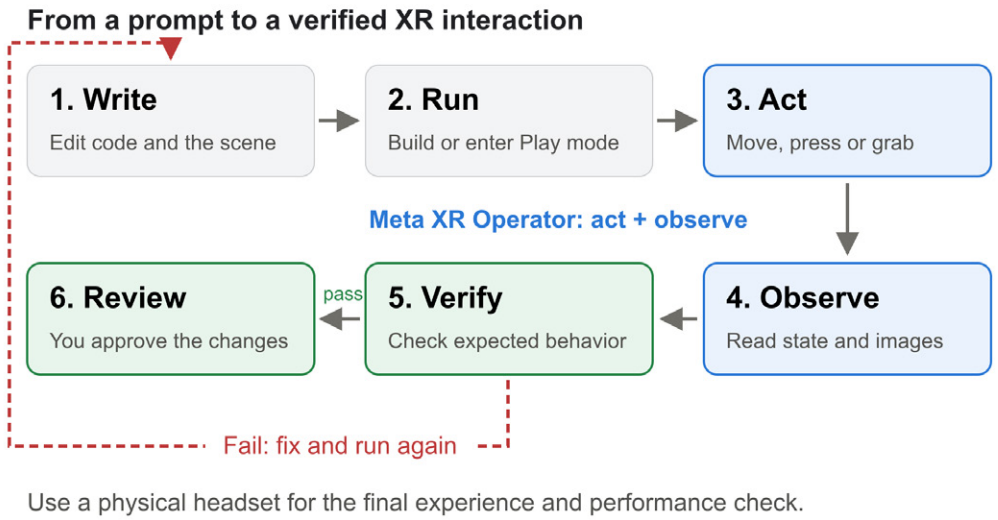
Meta XR simulator allows you to test gaze features, and FOV simulation

Meta XR Operator

The **Meta XR Operator** gives AI coding agents the ability to see, navigate, and interact with your running XR application. It is an optional, experimental component of the **Meta XR Core** SDK that works with OpenXR-based Unity apps without requiring changes to your application's code. This closes the gap between generating code and checking that the resulting experience works.

© 2026 Unity Technologies

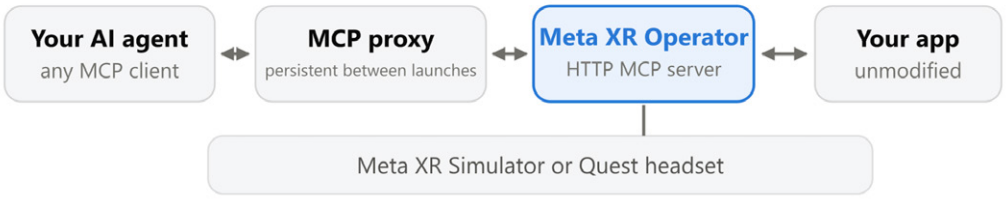
While Unity CLI and the Unity Pipeline package help an agent edit and launch your project, Meta XR Operator lets it act inside the running application and inspect the result. For example, an agent can open a menu, select a button, capture the rendered view, and use that feedback to diagnose a problem before trying a fix.



Meta XR Operator adds interaction and observation to the AI-assisted development loop. You review the resulting changes.

How it works

Meta XR Operator is an **OpenXR API layer** between your app and the runtime. It exposes live state and input controls through a local server using the **Model Context Protocol (MCP)**. An MCP proxy connects your compatible AI agent to that server. You can use this workflow with **Meta XR Simulator**, the Unity Editor with **Quest Link** on Windows, or development builds on a **Meta Quest** headset.



The MCP proxy connects your agent to Meta XR Operator's local server in the running app.

What can your agent do?

Meta XR Operator provides tools for understanding and interacting with your application:

— **Inspect the scene**: Read the Unity scene hierarchy and UI canvases, query XR session state, and inspect spatial entities such as walls, floors, and furniture.

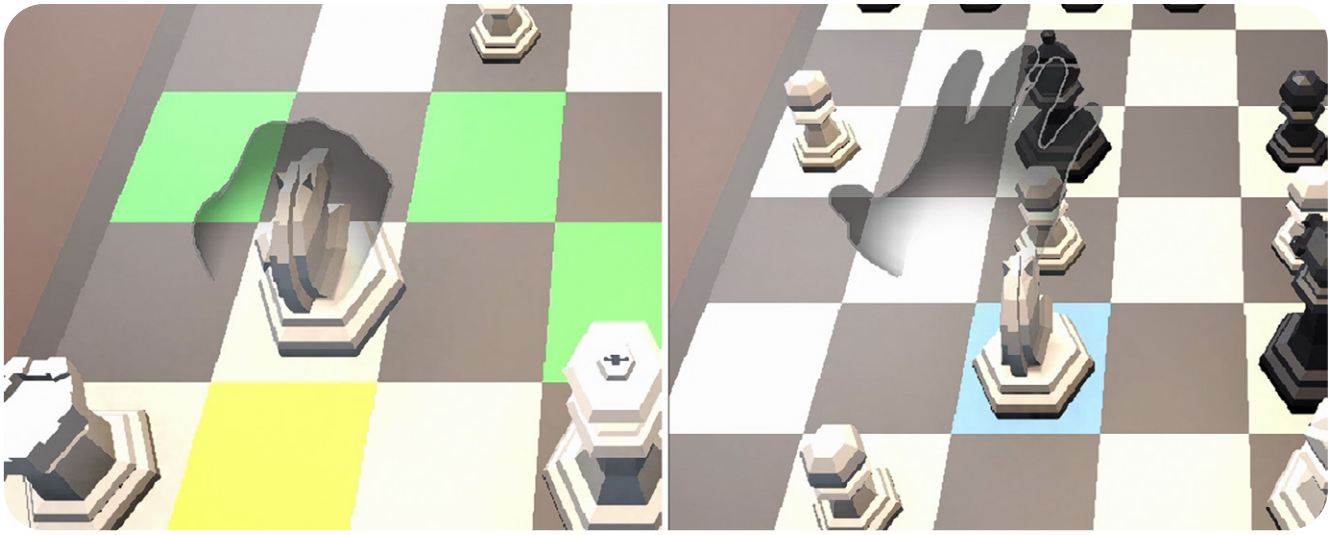
© 2026 Unity Technologies

— **Control interactions**: Move the head, controllers, and hands; send button, trigger, and thumbstick input; and perform gestures such as pinch, poke, and grab, including gaze-and-pinch interactions for Meta VR Glasses.

— **Check the result**: Capture composited images and read live state to reproduce an issue or verify that an interaction behaves as expected.



In the Shooting Gallery demo, an agent aims and fires through simulated controller input, then observes the application's response.



The VR chess demo shows a simulated hand grabbing and releasing a piece, exercising the app's hand interaction path.

© 2026 Unity Technologies

Getting started in Unity

To try Meta XR Operator in your project, follow these instructions:

1. Install **Meta XR Core SDK v207 or later** and **OpenXR Plugin 1.17.0 or later**. Enable OpenXR for your desktop platform in **Project Settings > XR Plug-in Management**.
2. Open **Window > Meta > Meta XR SDK** and select **AI Tools**. Choose your agent and follow the XR Operator card to install the tools, run the generated connection command, and select **Activate XR Operator**.
3. Install the project-level skills from the same panel. On Windows, review the Meta XR Operator rules in the **Project Setup Tool**; on macOS, the panel enables the layer when you enter Play mode.
4. Enable **Meta XR Simulator**, disable Unity's **XR Interaction Simulator**, and enter Play mode. Launch your agent from the project folder and ask: "*What is the current XR session state?*"

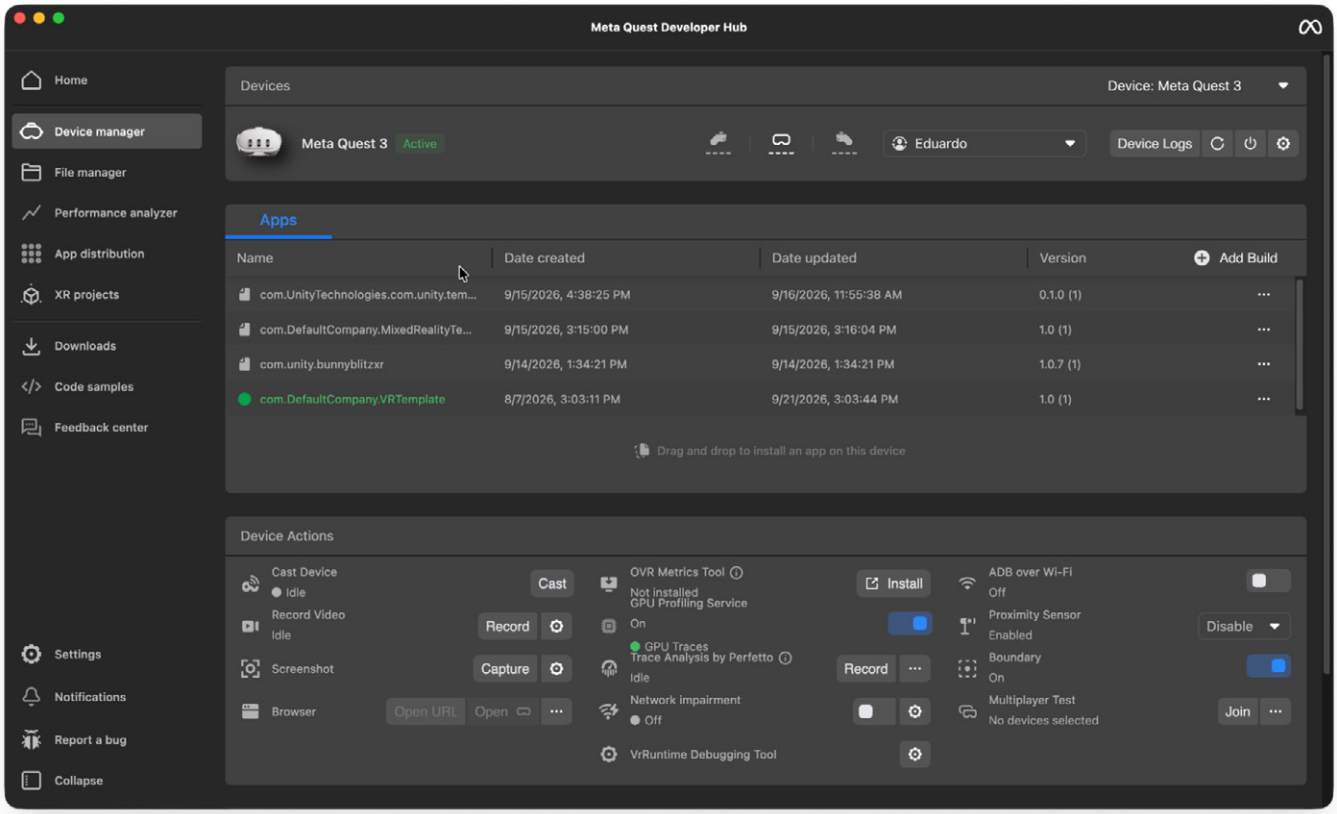
Once the connection works, try a scoped prompt such as "Open the main menu, select Play, and capture a screenshot to check that the game has started." For the full setup and headset-specific instructions, refer to the Getting started with Meta XR Operator documentation.

Important: Meta XR Operator is experimental; its tools and behavior may change. Use it for development and testing, not as a dependency in your production app. Simulator checks do not replace testing comfort, tracking, and performance on physical hardware. Live app tools are available only while the app is running.

Meta Quest Developer Hub

The Meta Quest Developer Hub (MQDH) is a companion desktop application for Windows and macOS designed to streamline iteration, device management, and debugging for Quest developers.

© 2026 Unity Technologies



It serves as a central control panel for your VR/MR development workflow, providing tools to accomplish the following:

Manage and deploy: Configure Meta VR Glasses and Quest headsets, toggle Developer Mode, manage Wi-Fi settings, and deploy APK builds directly to connected devices over USB or Wi-Fi.

Debug and profile: Stream live system logs via Logcat, analyze real-time performance metrics (FPS, GPU/CPU utilization), explore on-device files, and trigger custom command-line actions.

Streamline testing: Cast your VR Glasses and Quest headset's display straight to your desktop, capture high-quality gameplay video/screenshots, disable the proximity sensor or boundary system for uninterrupted testing, and simulate inputs via keyboard and mouse.

Distribute and manage SDKs: Download and update the latest Meta tools/SDKs, upload app builds to the Meta VR Developer Dashboard, and manage multi-developer team accounts.

© 2026 Unity Technologies



More resources

As hardware and Unity developer tools rapidly evolve, new video tutorials and resources are frequently released. To stay updated on these valuable guides, we encourage you to subscribe to Unity's official YouTube channel and social media handles. Below are helpful resources to explore.

Some of them might use older Unity and XRI versions for the key components, but the way everything interacts remains similar, so don't hesitate to have a look at the following content:

Step-by-step video tutorials

- Developing for Meta VR Glasses, Unity livestream
- Building in VR with XRI
- Make a VR Multiplayer game – part 1
- Make a VR Multiplayer game – part 2
- Make a VR Multiplayer game – part 3
- MR Multiplayer Sample Overview
- Create a MR Multiplayer Visualizer from scratch

© 2026 Unity Technologies

Unity CLI resources

— Unity Command-Line Interface

— Getting started with Unity CLI

Meta XR Operator resources

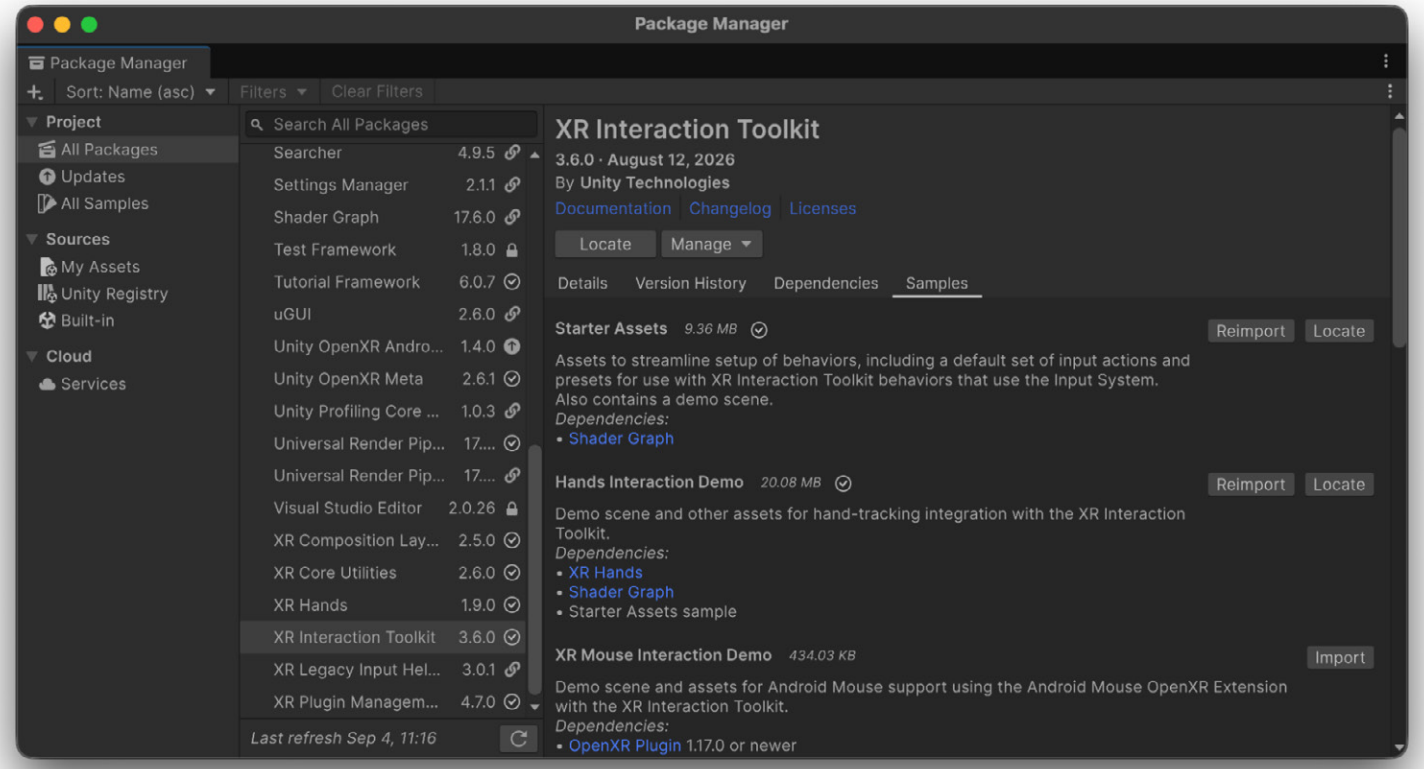
— Getting started with Meta XR Operator

— Using Meta XR Operator with Meta XR Simulator

— Meta XR Operator: Close the build-test-verify loop for VR

More samples

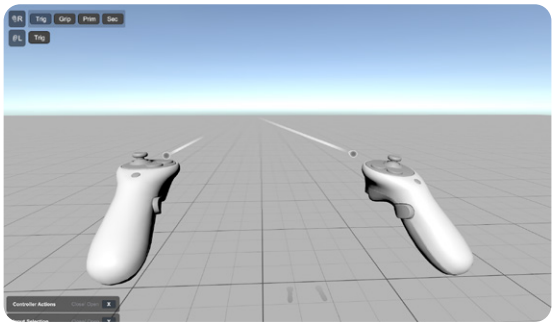
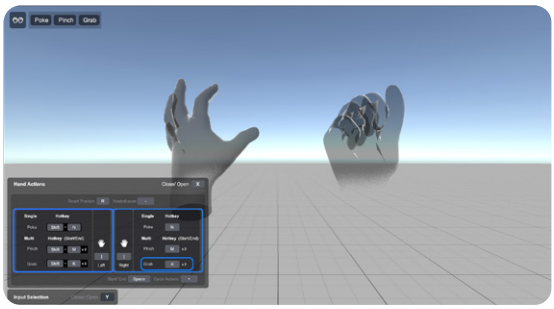
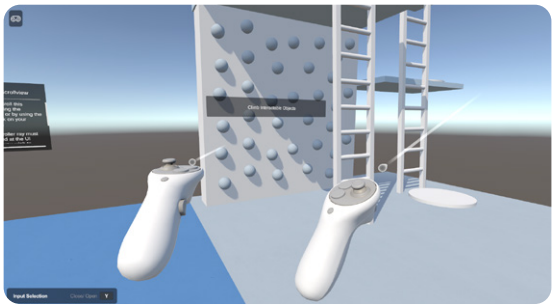
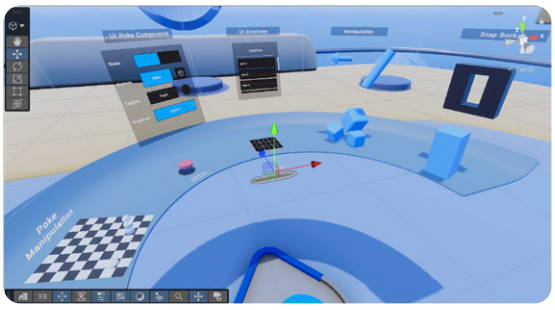
There are other scene examples that you can use to explore one feature area at a time or try a sandbox scene to test different components working together.



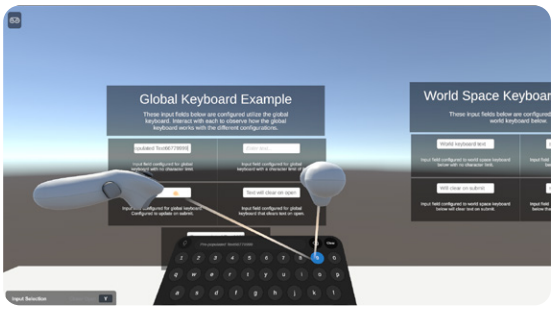
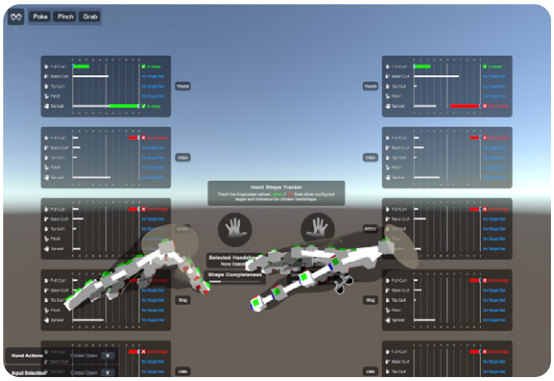
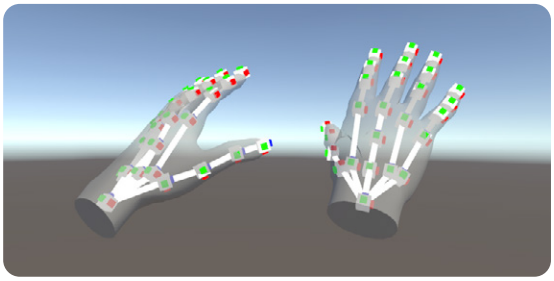
Packages include additional demo scenes under Samples

In the table below, you'll find some other demos of what's available through the Unity Hub, as well as the **VR** template and **XRI** samples available from the Package Manager that can help you get started with your Meta VR Glasses.

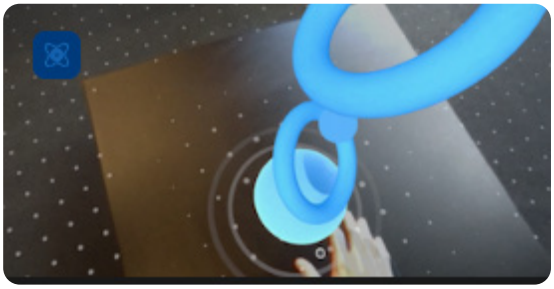
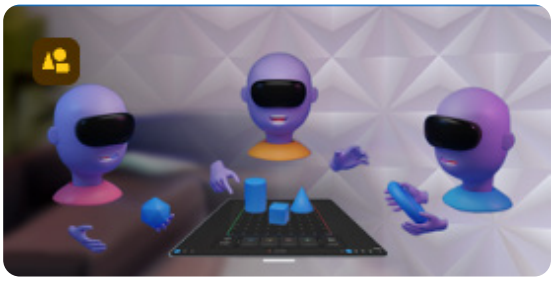
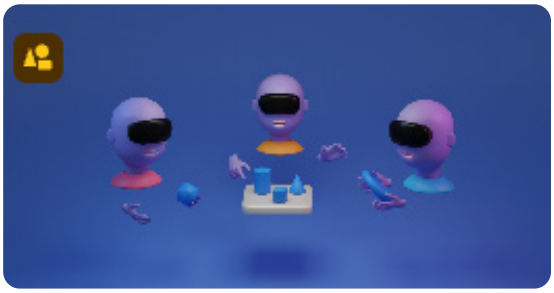
© 2026 Unity Technologies

Scene name	How to get it	Description
  BasicScene	Unity Hub/VR template	An empty scene with just the **XR Origin** (**XR Rig**) prefab, character controller, and input controls. Ideal to start building from, or add additively to your 3D world to start navigating in VR quickly. To add hand controls, just add **Left** and **Right Hand Tracking** prefabs as child GameObjects of the **Camera Offset** GameObject inside the **XR Origin** (**XR Rig**) prefab and add a reference to them in the XR Input Modality manager
 DemoScene	XRI package sample	Starter assets in the form of prefabs with pre-setup interactions to help you fast track the creation of elements for your project. It contains a demo scene.
  HandsDemoScene	XRI package sample	A scene set up with controller and hand controls to test teleporting, UI, and 3D object interactions

© 2026 Unity Technologies

 KeyboardDemo	XRI package sample	A scene with examples of virtual keyboards. You can test input in the UI and see how a world space keyboard and main player keyboard are set up.
 HandGestures	XRI Hands package sample	Sample showing how to detect and debug hand gestures.
 HandCapture	XRI Hands package sample	Sample for recording real-time hand data to create custom gestures.
 HandVisualizer	XRI Hands package sample	Basic sample showing how to visualize hands.

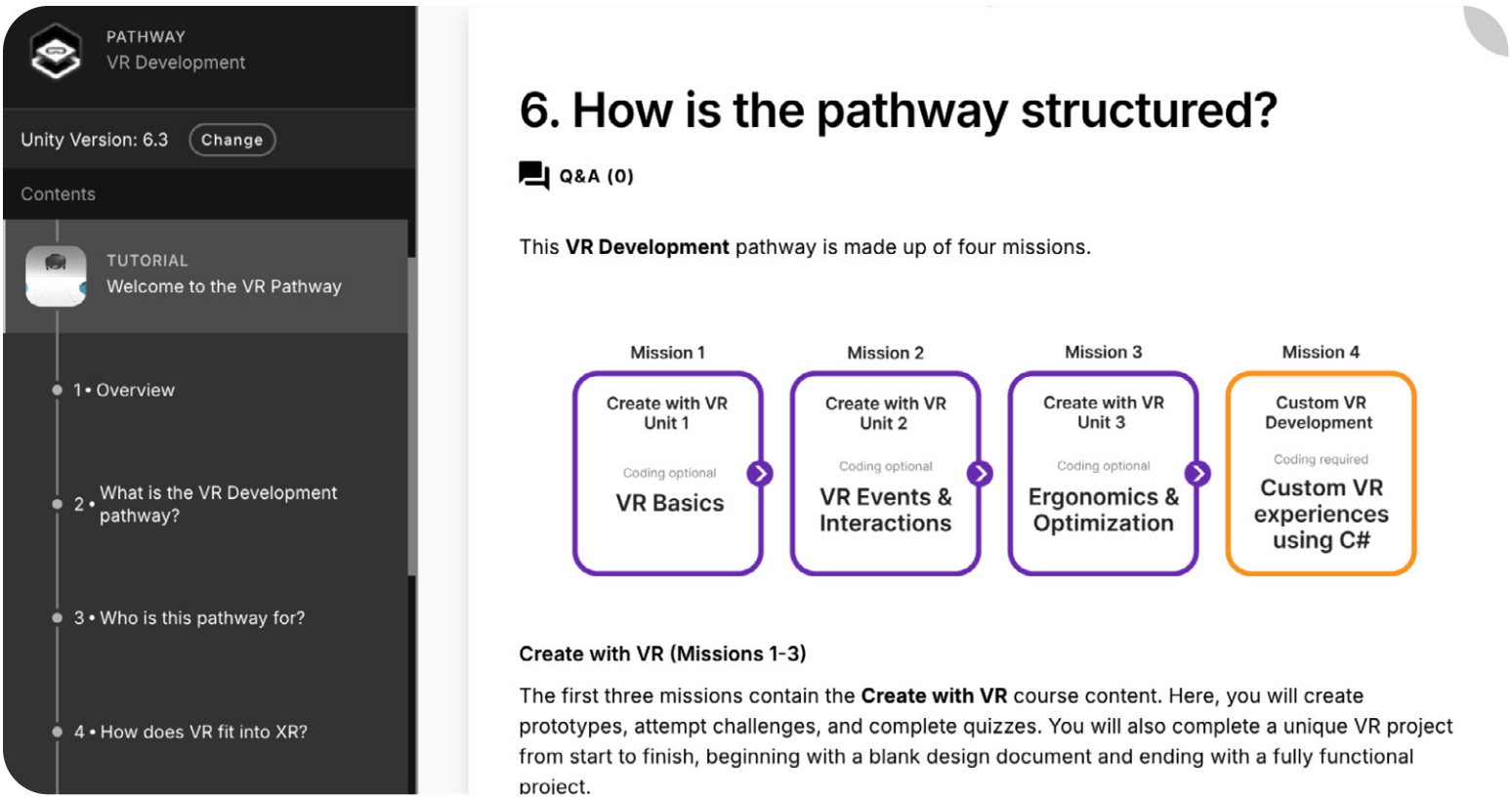
© 2026 Unity Technologies

 SampleCompositionLayersScene	XR Composition layers package	Sample showing how to use Composition Layers. They allow developers to define areas in a scene or view that are composed separately from the normal 3D rendering.
 Mixed Reality template	Unity Hub/MR template	Unity's **Mixed Reality (MR)** template provides a starting point for mixed reality development in Unity. The template configures project settings, pre-installs packages, and includes various pre-configured example assets to demonstrate how to set up a project that is ready for MR.
 Mixed Reality multiplayer tabletop template	Unity Hub/ MR Multiplayer Tabletop template	The MR Multiplayer Tabletop Template combines XRI, AR Foundation, Networking for GameObjects, and Unity gaming Services to create a starting point for Tabletop MR Multiplayer apps.
 VR Multiplayer template	Unity Hub/ VR Multiplayer template	The VR Multiplayer Template combines XRI, Networking for GameObjects, and Unity gaming Services to create a starting point for VR Multiplayer apps.

© 2026 Unity Technologies

Unity Learn

Unity offers guided and structured learning experiences on our learning platform, Unity Learn. On Unity Learn, you'll find the **VR Development Pathway**, which is ideal for beginners who want to learn the foundations of VR development in Unity.



The Learn structured courses of the VR pathway

© 2026 Unity Technologies


Unity®