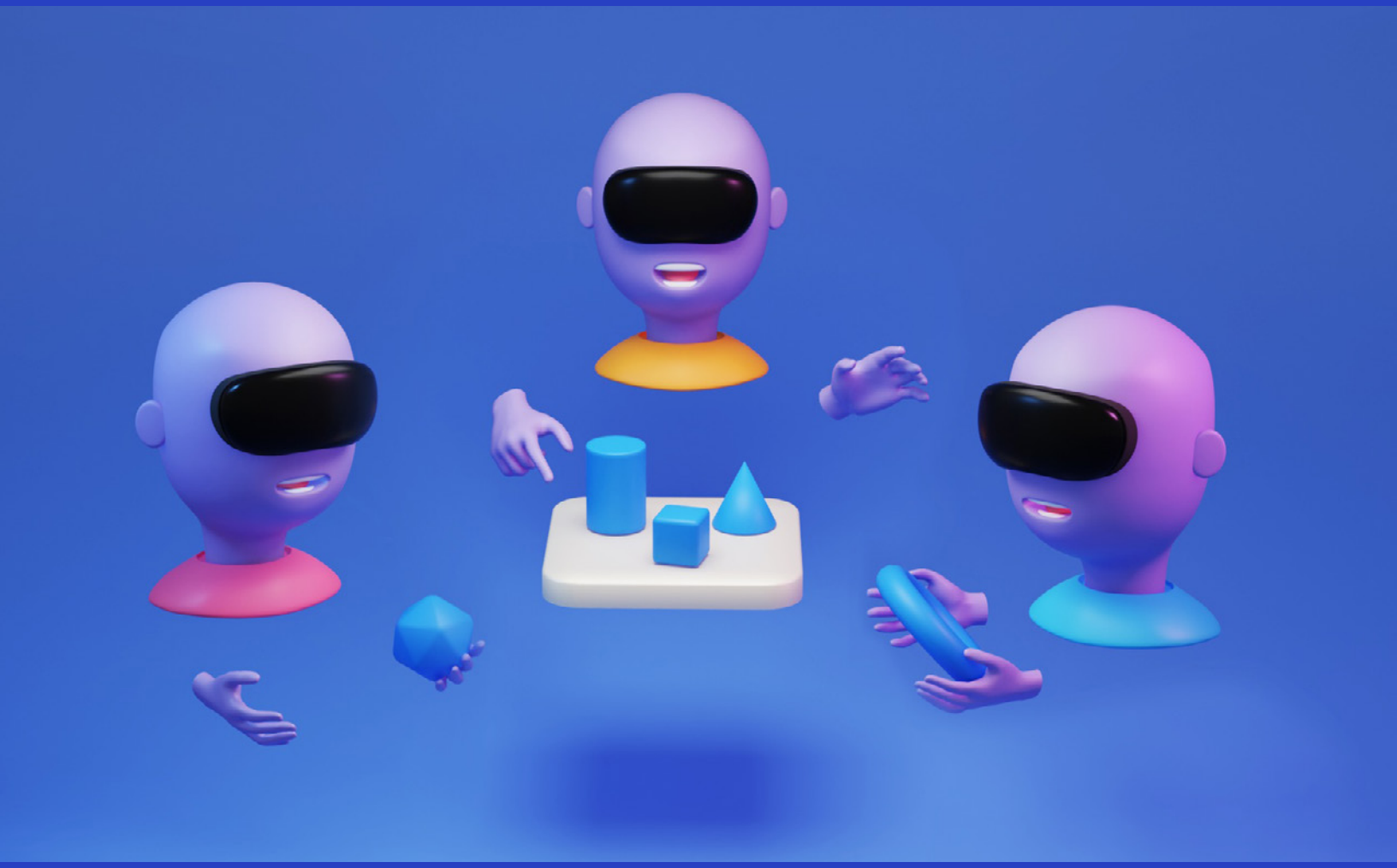
Unity®

UNITY 2022 LTS EDITION → E-BOOK



Create virtual and mixed reality experiences in Unity

Contents

Introduction

Embark on journeys through immersive virtual realms, teleport between dimensions, or merge digital marvels with the real world – the possibilities of virtual reality (VR) and mixed reality (MR) invite creators to bring their imagination to life.

Our comprehensive guide will help both aspiring creators and seasoned developers to delve into, and understand, the intricacies of building VR and MR experiences (or collectively referred to as “XR”) using Unity.

Unity is one of the leading platforms for creating XR experiences, as these results from 2024 show:

— More than 50% of “Week One Must Play” Apple Arcade Games for Vision Pro were made with Unity.[1]

— Two-thirds of the most popular Quest experiences were made with Unity as of February 2024.[2]

— At least 70% of top-selling Quest games are made with Unity as of February 2024.[3]

— More than 60% of the top-grossing VR Steam Experiences in 2023 were made with Unity.[4]

1 Source: Apple Inc.
2 Source: SteamDB
3 Source: SteamDB
4 Source: SteamDB

You'll learn about the tools, methodologies, and techniques essential for crafting immersive and interactive realities. From constructing environments to implementing intuitive interactions, you'll get the tips and guidance you need to bring your VR and MR applications to life.

Step into a world where reality meets imagination, where the boundaries of the tangible blur with the ethereal. Let Unity be your guide as you embark on the journey of crafting immersive realities that captivate and inspire.

Main author

Daniel Stringer is a seasoned 3D artist, Unity developer, and content creator, covering two decades of expertise spanning diverse industries. He has two decades of experience with creating games, training and simulations, architectural visualization, film, and VFX. Daniel's repertoire includes the development of award-winning VR applications, notably contributing to the healthcare sector in the UK and commercial radio industries.

You can find more of his content on YouTube and Udemy, specializing primarily in Unity-related subjects with a focus on VR.

Unity contributors

The following colleagues from Unity's XR product, product marketing, and content marketing teams contributed to this guide:

Adam Axler, Dan Martinez, Eduardo Oriz, Isaac Seah, Kevin Semple, Leah Martin, Matt Fuad, Miruna Dumitrascu, Sam Zhang, Shanti Zachariah, and Tricia Becker.

Population ONE by BigBox VR (left), *Job Simulator* by Owlchemy Labs (right)

Industry Terms

In the context of developing with Unity, XR is an umbrella term that includes the following types of applications:

— **Mixed reality (MR)**: MR typically refers to headset systems that use video see-through displays, where the real world is captured by cameras and digitally combined with virtual content before being presented to the user's eyes.

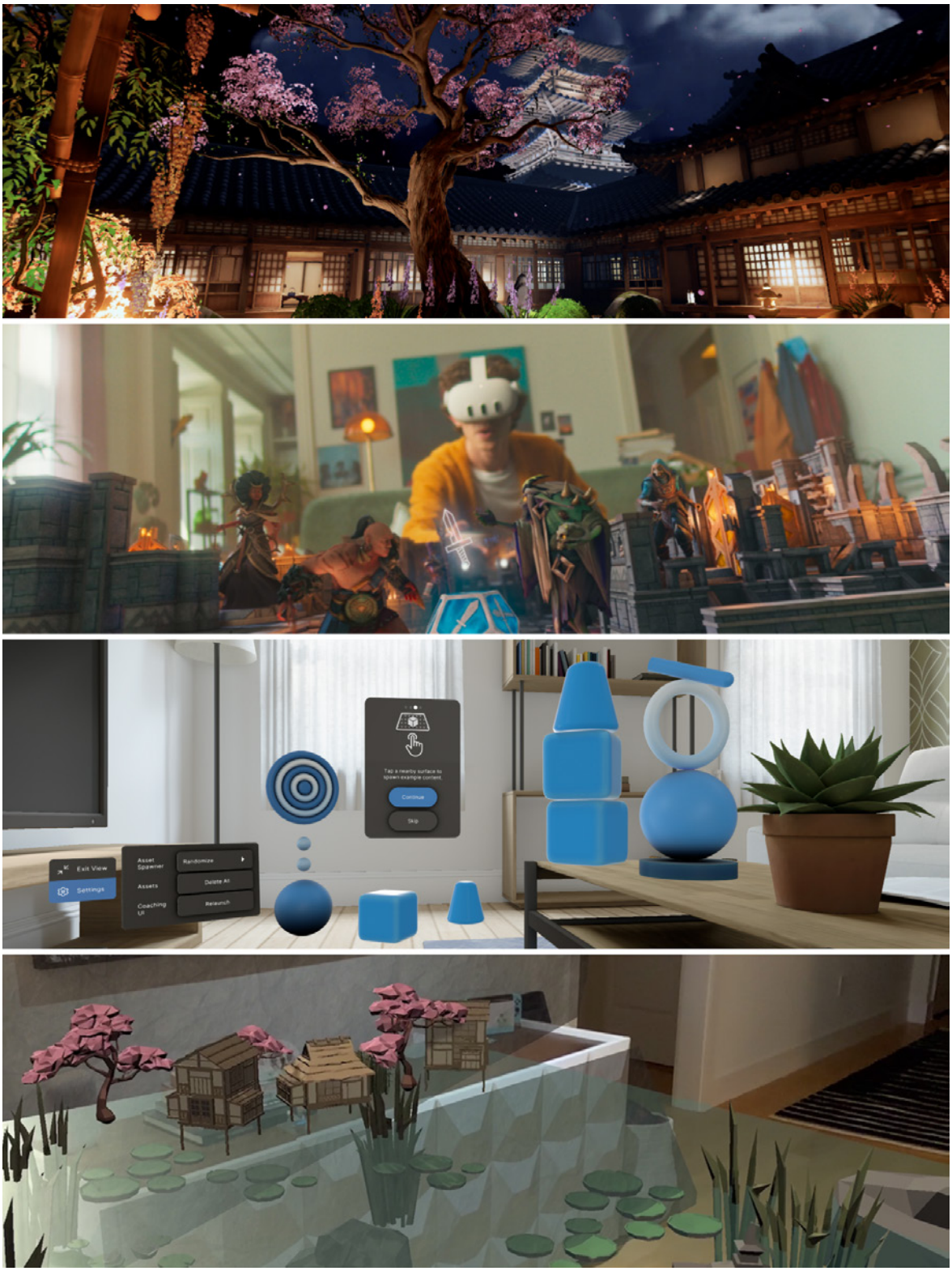
— **Virtual reality (VR)**: VR offers a fully immersive digital experience, transporting users into entirely simulated environments, often through headsets or specialized devices.

— **Augmented reality (AR)**: AR refers to systems that use optical see-through displays, such as transparent glasses or visors, where the user can directly see the real world, and virtual content is overlaid on top of it.

— **Spatial computing:** This refers to processes and technologies where digital information is seamlessly integrated into the physical world, creating an immersive experience that blends the virtual and real environments.

The span between VR to AR is a continuum, with MR falling somewhere in the middle, depending on the capabilities of a particular device and the intent of your application.

These terms represent a spectrum of immersive technologies, each offering unique experiences and opportunities for creative exploration in game development and beyond.



From top to bottom: The Unity URP 3D sample running in VR; Demeo Battles, by Resolution Games, being played on a Meta Quest 3 headset; interactive samples for Unity's visionOS Spatial support, and an AR application adding elements to a phone's camera feed

Choose your reality: VR, AR, or MR

This section explores the factors you can consider to help make a decision about what type of immersive game you want to make.

The many acronyms and choices in XR game development

Immersive experience goals

The desired level of immersion influences the choice. VR provides full immersion, while AR and MR offer contextual enhancements to reality.

What you decide here will also determine your target hardware, whether that's VR headsets, AR-compatible smartphones and glasses, or MR devices.

A note about MR: An important feature of XR headsets that makes MR possible is **Passthrough.** This refers to the capability of the headset to display a live feed of the real world to the user, usually through front-mounted cameras, allowing them to see their physical surroundings while wearing the headset.

Passthrough is required for MR apps 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 the headset. This feature bridges the gap between complete virtual immersion and the user's actual physical space.

User interaction and engagement

What do you want users to interact with in your application? This is a pivotal question. AR and MR combine physical and virtual environments, while VR isolates users within an exclusively digital realm.

Narrative and gameplay elements

The narrative and gameplay mechanics play a crucial role. VR might suit a fully immersive story-driven experience, while AR/MR can enhance location-based gameplay.

Use cases

Practical applications will influence the decision. AR/MR might be ideal for training simulations or educational experiences where real-world integration is beneficial.

Development resources and constraints

Developer expertise, resources, and technical constraints also impact the choice, as each reality type demands distinct development approaches.

I Expect You To Die 2: The Spy and the Liar, the sequel of the award-winning VR game from Schell Games

Motion Considerations

Another important consideration is motion, for the reasons listed here:

— **User comfort:** Excessive or unrealistic motion can lead to discomfort or motion sickness. Ensure your app provides smooth movement to avoid disorientation and nausea.

— **Interaction design:** Users should move and interact within the virtual space intuitively. Design movement mechanics based on the type of experience (e.g., gaming, training, exploration).

— **Performance constraints:** High-speed or complex motion might require more processing power, affecting app performance. Ensure the app runs smoothly on intended devices by testing and optimizing performance throughout the development cycle (see the section on testing and iteration later on in the guide).

— **Physical space limitations:** Consider the physical space users have. Will they be using the app at home or at their work location for training? Design your apps so they can be used in different locations safely.

— **User accessibility:** Not all users can move freely. Provide alternative interaction methods for those with mobility restrictions to make your app accessible to more users.

Understand and address these aspects to ensure you're developing an accessible, comfortable, and engaging AR/VR experience.

Controls, motion, and physical space context will all be specific to your game or industrial XR application.

Worldbuilding in XR

Worldbuilding is not just about creating a visually stunning environment. It's about crafting an immersive world that tells a story, conveys emotion, and captivates users. This section looks at the building blocks for worldbuilding in XR.

Narrative development

Every virtual world begins with a strong concept that guides its development. This can be a unique setting, a compelling story, or an intriguing mystery that players can unravel. The concept should serve as the backbone for all elements of the world, from visual design to interactive elements.

Environmental storytelling

Design with purpose

Every element in your virtual environment should serve a purpose, whether it's to advance the story, enhance the atmosphere, or provide clues and challenges. Avoid clutter or elements that don't contribute to the overall experience.

Visual and audio cues

Use visual and audio cues to guide users through the story and setting. Lighting, soundscapes, and architectural design can direct attention, set the mood, and hint at the underlying story.

Interactive elements

Incorporate interactive elements that reveal more about the world and its story. These can be ancient artifacts that unveil part of the history, journals that provide first-person accounts, or environmental puzzles that unlock secret areas or lore.

Demeo from Resolution Games, a MR game available for all the major platforms

User engagement strategies

Immersion through detail

Pay attention to small details in the environment. Textures, sound effects, and interactive objects should match your artistic intent. Create a cohesive, consistent experience to immerse and engage your players.

Encourage exploration

Reward users for exploring your world. Hidden areas, secret lore, and easter eggs can motivate users to look beyond the main path, enriching their experience and engagement with the world.

Adaptive difficulty

Implement adaptive difficulty mechanisms that adjust challenges based on the user's performance. This keeps the experience engaging for users of all skill levels, ensuring that it's challenging but not frustrating.

Social interaction

If applicable, integrate social interaction into your world. Multiplayer elements, where users can explore and solve puzzles together, can add a layer of engagement and make the experience more dynamic.

Job Simulator by Owlchemy Labs, includes richly detailed environments that provide increasingly challenging, but fun, tasks.

Iterative design

Use feedback to iteratively refine your world. Adjust the narrative flow, redesign certain areas for better navigation, or add more interactive elements to enhance engagement.

Incorporating these strategies into your worldbuilding can elevate the quality and impact of your VR and MR experiences. By focusing on narrative development, environmental storytelling, and user engagement, you create not just a game or an application but a compelling world that users will want to return to time and again.

Learn more about game design and level design in Unity by downloading these e-books:

Download

Download

New to Unity?

Unity Learn provides a wealth of free resources for learning how to navigate in and use the Editor. It's recommended that new users work through the three Pathways courses: Unity Essentials, Junior Programmer and Creative Core. Once you've completed the foundational Pathways courses, try the VR Development pathway.

Finally, you can view the deep dive sessions and keynote from Unite and GDC to stay on top of the latest developments in the XR space and Unity. Start with these links:

- Unity at GDC 2024
- Unite 2023
- XRI Session | XR Developer Night 2024
- XR Roadmap 2024 | XR Developer Night 2024
- Hand Input & OpenXR Development | XR Developer Night 2024
- XR Composition Layers | XR Developer Night 2024
- Mixed Reality Authoring & Simulation | XR Developer Night 2024
- AR Foundation | XR Developer Night 2024

Asset creation for XR

There are important considerations when creating or sourcing your art assets for XR games. From small props to characters, follow these tips to save yourself time later in the development cycle.

Concept and design

Start with a clear concept of what you want to create. Sketch your ideas and consider how your assets will be viewed in a 3D 360 degree space. These concepts will help shape your ideas, from how the environment might look, to how puzzles might work. Sketches can help convey your idea to others early on so you and your collaborators can make more informed decisions and plan effectively.

Concepts can be created as simple hand drawings, particularly at an early stage, to quickly plan out interactions or visuals. In larger studios you might have a concept artist that can bring ideas to life; for individuals or small teams you might also try AI to help portray your idea.

Modeling

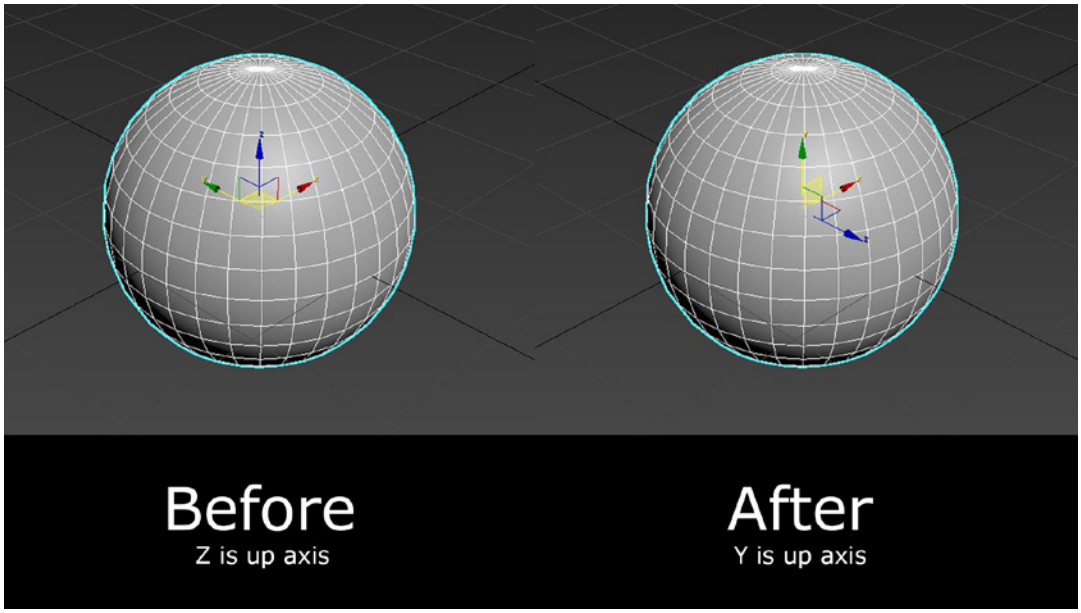
Use 3D modeling software (like Blender or Autodesk Maya or 3ds Max) to build your model. Keep poly count (number of polygons) in mind for performance, especially for mobile AR. You can create high-poly models initially, and then optimize them later.

Pivot point

The pivot point acts as the model's center of rotation and scaling. Properly positioned, it can streamline animation processes, making movements and transformations more natural and intuitive. For instance, a pivot point at the base of a figure allows for more realistic and stable rotations, akin to a person turning on their feet. This positioning is crucial for accurate interaction with the environment in both animations and game development.

When you work in a 3D scene in Unity, the y-axis represents vertical movement or elevation. This orientation is a standard convention in many 3D graphics environments, where the x-axis represents horizontal movement (left and right), the y-axis vertical movement (up and down), and the z-axis depth (forward and backward). Understanding this axis system is crucial for correctly positioning, rotating, and scaling objects within Unity.

When working in your 3D modeling software you can set the pivot point of the model, ensuring that the orientation matches that of Unity. Below is an example of how to set the pivot point in 3ds Max, which uses the z-axis as the up axis.



Adjusting the up axis

You can also correct the orientation of a model in Unity with these two steps:

1. Create an empty GameObject in the Hierarchy.

2. Nest your model within this GameObject, ensuring its position is set to 0,0,0 across all axes to achieve perfect centering.

This method results in your model being anchored to a parent GameObject that maintains a neutral rotation of 0 across all axes. It also guarantees the model is oriented correctly with the z-axis facing forward and maintains a uniform scale of 1 on all axes. Additionally, you can also try the Bake Axis Conversion in the Importer.

Texturing

Apply textures to give your model a realistic or stylized look. Consider using UV mapping for detailed textures. Depending on what your aesthetic is for your game you might require PBR Textures. Unity's default Universal Render Pipeline (URP) shader is physically based, so it simulates the physical properties of materials in a realistic way. It uses algorithms that accurately depict how light interacts with surfaces, considering factors like texture, reflectivity, and roughness. Physically based (PBR) shaders in Unity typically involve two main components: Metallic-roughness workflow and Specular-glossiness workflow, each handling different aspects of material properties. The result is a more lifelike and dynamic visual appearance, enhancing the realism of 3D scenes and objects in Unity-based games or simulations.

Trim sheets

A trim sheet is a texturing tool used in 3D modeling and game development. It consists of a single texture containing a variety of trim patterns and details, like moldings, edges, and borders. Artists can apply these trims to different parts of a 3D model to add intricate details without needing multiple unique textures. This method is efficient for optimizing game performance, as it reduces the number of individual textures required, while still allowing for visually complex and varied designs on the models.

Trim sheet used for various meshes across the environment

Rigging and animation

If your model requires movement and the mesh needs to deform, such as for a character, then you can rig it with a skeleton structure and animate it.

First-person VR apps require you to set up avatar hands. You can use the Runtime IK feature in the Animation Rigging package to solve the bones of the arm and align the hands to the position of the tracked controller with a TwoBoneIK constraint.

The elbows are an interesting challenge because their positions are not commonly tracked. If the elbow position can be inferred procedurally then the TwoBoneIK constraint can align the elbow bone using the Hint transform as input.

Arashi: Castles of Sin - Final Cut by Endeavor One, has VR hands with arms using the TwoBoneIK system.

Model optimization

You should optimize your models for performance in Unity by reducing the polygon count and file size, and ensuring textures are efficiently sized. Mesh detail in the model can be captured by creating normal maps.

A normal map 3D is a texture that encodes the direction of normals (perpendicular vectors) for each pixel of a surface. This allows for detailed surface textures, like bumps or indentations, without increasing the polygon count of the model. It creates the illusion of depth and detail on a flat surface by altering the way light interacts with it, enhancing its realism.

The best ways to generate a normal map for 3D modeling include:

— **From high-poly models:** Create a detailed high-poly model and a low-poly version. Then bake the surface details of the high-poly model onto the low-poly model as a normal map using software like Blender or Substance Painter.

— **Using 2D image editors:** Tools like Adobe Photoshop or GIMP with plugins can convert grayscale height maps into normal maps. This method is useful for surface textures like fabric or stone.

— **Dedicated normal map generators:** Software like CrazyBump or NormalMap Online can generate normal maps from 2D images.

— **Photogrammetry:** Create normal maps by converting real-world objects into digital models using photogrammetry techniques.

SUPERHOT VR by SUPERHOT Team goes for a low-poly aesthetic that matches the theme and enables the performance required for the gameplay.

Exporting

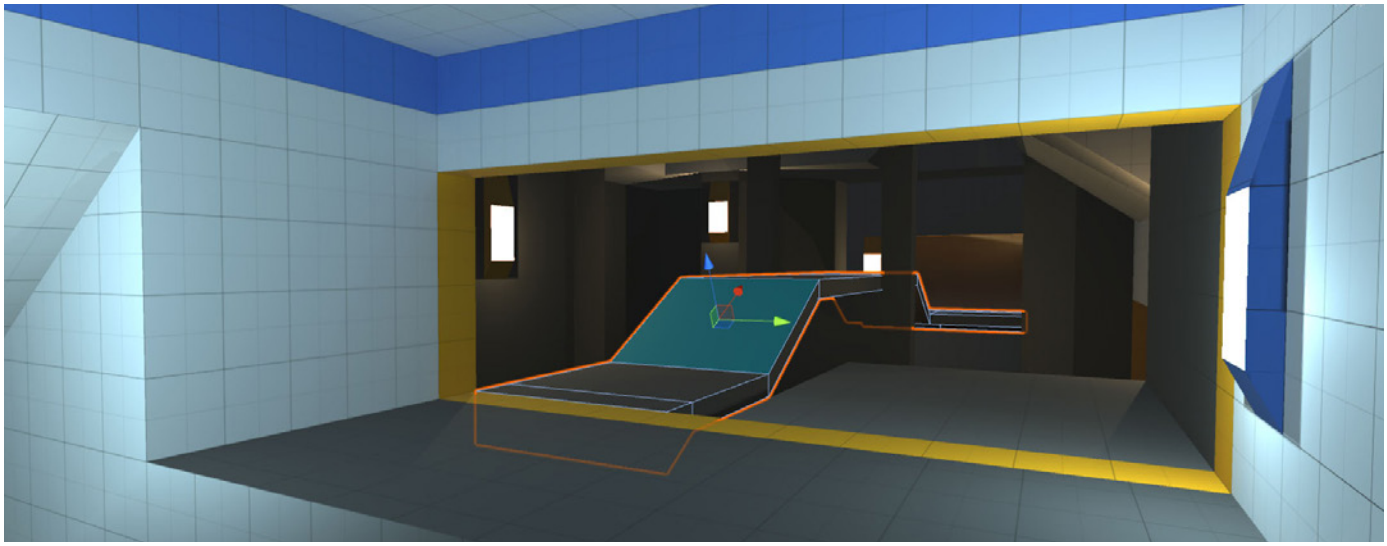
Export the model in a format compatible with Unity (FBX is recommended).

The export scale of 3D models is important when working with AR and VR in Unity. The scale determines how the model fits within the virtual world and interacts with real-world dimensions, especially in AR where correlation to the physical world is crucial. A mismatch

in scale can lead to unrealistic or impractical interactions. It's essential to ensure the model's scale is consistent with Unity's units and the intended real-world or virtual environment scale.

In Unity, the measurement scale is intuitively designed, where 1 unit corresponds precisely to 1 meter. To gauge the scale of your imported assets, simply introduce a cube GameObject into your scene. This cube, with dimensions of 1m x 1m x 1m, serves as an immediate and accurate reference for assessing the scale of any asset.

When creating your environments and props the key is to balance aesthetics and performance, ensuring a smooth and engaging AR/VR experience.



You can prototype levels with ProBuilder and then export them with FBX Exporter to 3D modeling software to tweak them.

XR in Unity: Key systems and toolsets

Rendering pipelines

Unity's rendering pipelines offer rich rendering capabilities for various platform and performance needs.

The garden environment from the Unity URP 3D Sample on the left and a HDRP volumetric environment on the right

The garden environment from the Unity URP 3D Sample on the left and a HDRP volumetric environment on the right

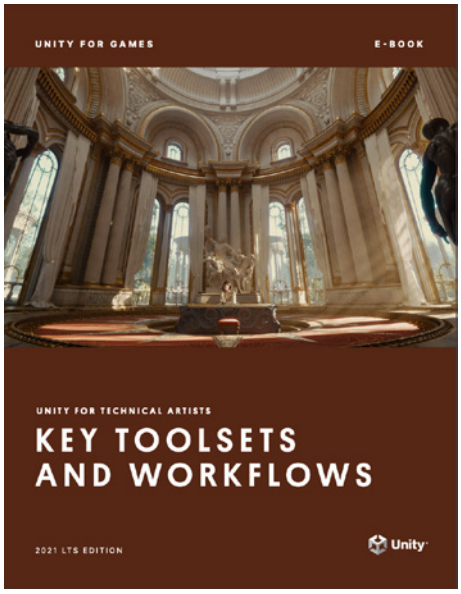
The **Universal Render Pipeline** **(URP)** is the recommended pipeline for developing XR applications, URP is lightweight and optimized for a broad range of devices, from mobile to high-end. URP offers modern rendering techniques, optimized shaders, and customizable post-processing effects.

The **High-Definition Render Pipeline** **(HDRP)** is for high-end PC and console platforms, delivering top-tier visual quality with physically based rendering, advanced lighting, materials, and intricate post-processing. Use HDRP for experiences where your target hardware is guaranteed to be able to deliver a very high refresh rate when pushing graphical boundaries.

The **Scriptable Render Pipeline** **(SRP)** is the foundational framework for URP, HDRP, and your own custom render pipelines tailored to specific project needs and platform specifications.

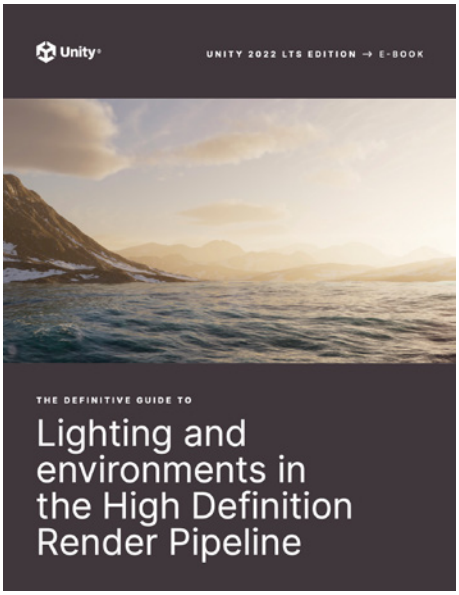
The **Built-In Render Pipeline** is Unity's legacy pipeline. We recommend that you use URP for general multiplatform game development from Unity 2022 LTS and newer.

Choosing the right pipeline hinges on project requirements, platform targets, and the desired balance between performance and visual quality. Each pipeline caters to different needs, offering varying levels of performance optimizations and visual fidelity options.



Download

Download



Download

AR Foundation

AR Foundation is a Unity package that offers a powerful and unified framework for crafting immersive AR experiences. It enables the implementation of essential AR features, including plane detection, world tracking, face detection, environmental understanding, and occlusion.

AR Foundation contains interfaces for AR features, but doesn't implement any features itself. To use AR Foundation on a target platform, you'll also need a separate provider plug-in package for that platform. Unity officially supports the following provider plug-ins:

— Google ARCore XR Plug-in on Android

— Apple ARKit XR Plug-in on iOS

— Apple visionOS XR Plug-in on visionOS

— OpenXR Plug-in on HoloLens 2

— Unity OpenXR: Meta on Meta Quest

See this page for the list of features and platforms supported with AR Foundation.

AR Foundation provides an API that streamlines the development process across multiple platforms. By abstracting the intricacies of platform-specific implementations, it helps you to focus on the creative work of AR experiences that can reach a diverse range of players across devices.

The AR sample is available via the AR Foundation documentation.

XR Interaction Toolkit

The XR Interaction Toolkit (XRI) package enables you to develop XR interactions efficiently by providing a standardized approach to interactions like grabbing, touching, locomotion, and object manipulation.

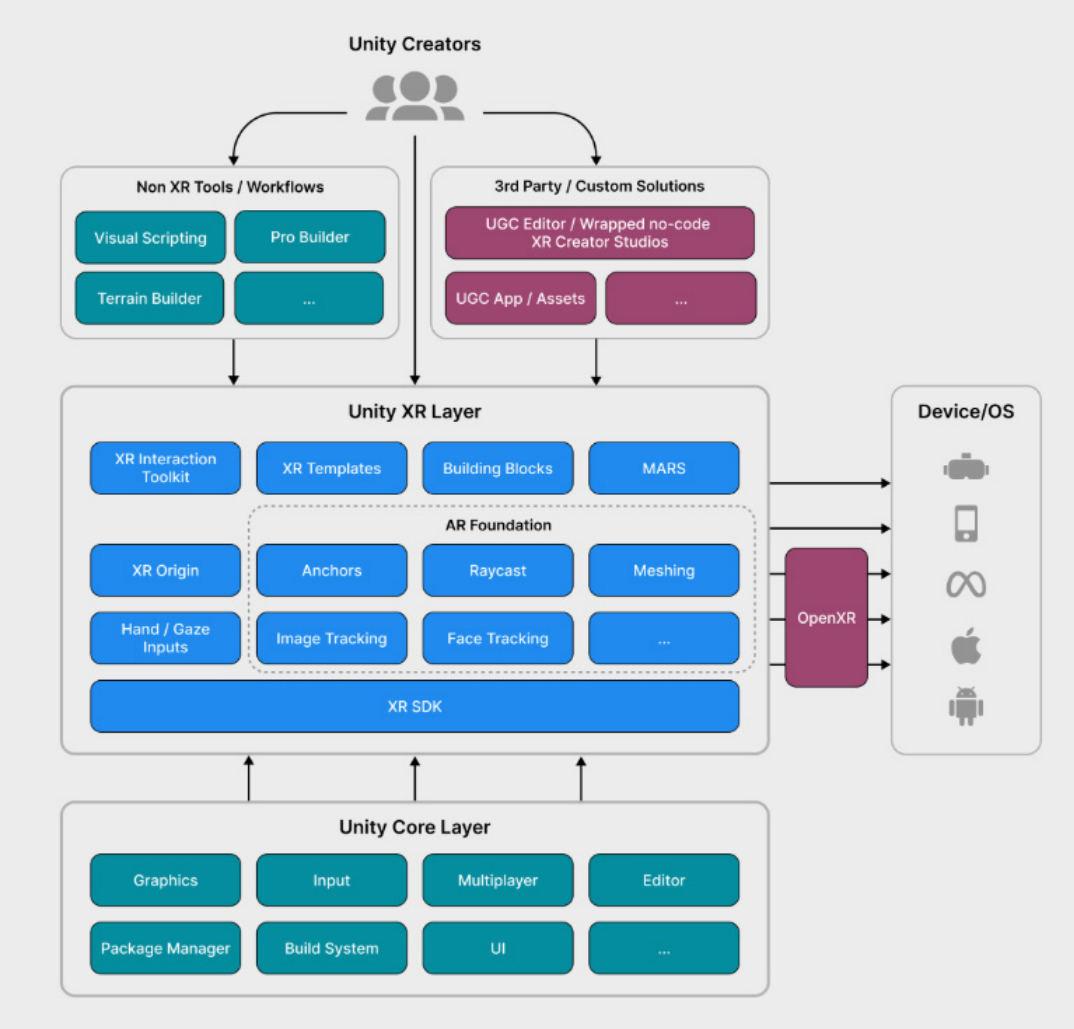
This toolkit provides a framework that makes 3D and UI interactions available from Unity input events. The core of its system is a set of base **Interactor** and **Interactable** components, and an **Interaction Manager** that ties these two types of components together. It also contains components that you can use for locomotion and drawing visuals.

Leveraging Unity's cross-platform capabilities, the XR Interaction Toolkit enables developers to focus on designing XR experiences without the complexity of managing disparate 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.

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



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

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

OpenXR

OpenXR is a royalty-free framework developed by Khronos, designed for creating 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.

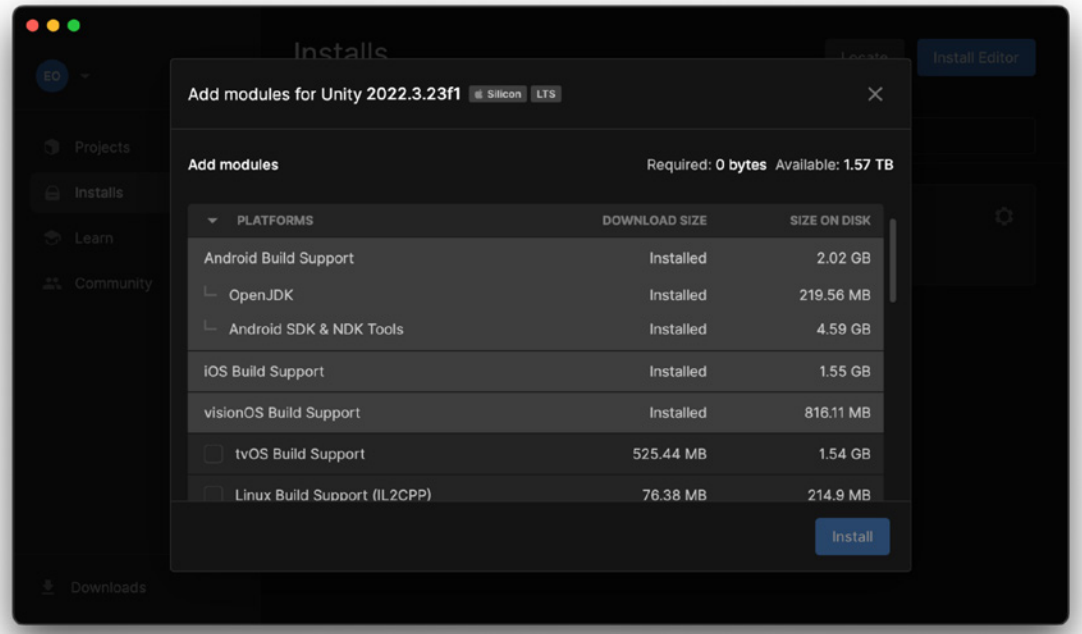
Editor version and modules

This guide uses the Unity 2022 LTS Editor and it's recommended that you use the same version, or a newer one.

Device	VR	MR/ Spatial	AR	Unity module	XR plug-in providers
Meta Quest 3	√	√		Android Build Support	ARCore, Oculus, OpenXR
Meta Quest 2	√	√		Android Build Support	ARCore, Oculus, OpenXR
Apple Vision Pro	√			visionOS Support, iOS Build Support	ARKit
HTC Vive Focus 3	√			Android Build Support	Vive wave XR, OpenXR
HTC Vive Pro 2	√			PC, Mac & Linux Standalone	Vive wave XR, OpenXR
PlayStation®VR2	√			Requires Sony developer registration	
Android mobile devices			√	Android Build Support	ARCore
Apple iPhone and iPad (A9 processor or later)			√	iOS Build Support	ARKit
Microsoft HoloLens			√	Universal Windows Platform (UWP)	Windows Mixed Reality XR Plugin
Magic Leap 2			√	Android Build Support	Magic Leap XR Plugin
Meta Quest 3	√	√		Android Build Support	ARCore, Oculus, OpenXR

For each target platform Unity supports, the corresponding platform build modules and XR plugins you need to download

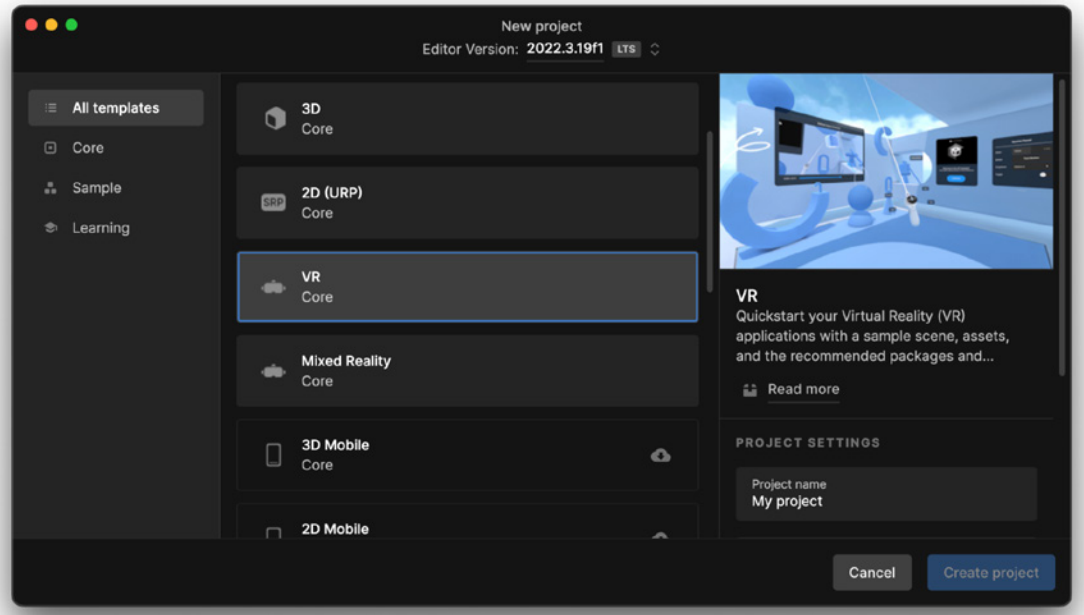
When you install the Editor you'll be prompted to add modules to the installation. To support mobile VR platforms like Meta Quest 2 you'll need (depending on your target mobile devices) the **Android Build Support**, **OpenJDK, Android SDK and NDK**, and/or the **iOS Build Support** and **visionOS Build Support** modules.



The VR core samples

Use the VR template, available in the Unity Hub, to start a new VR project. The template provides sample scenes, assets, and recommended packages and settings. To get the template:

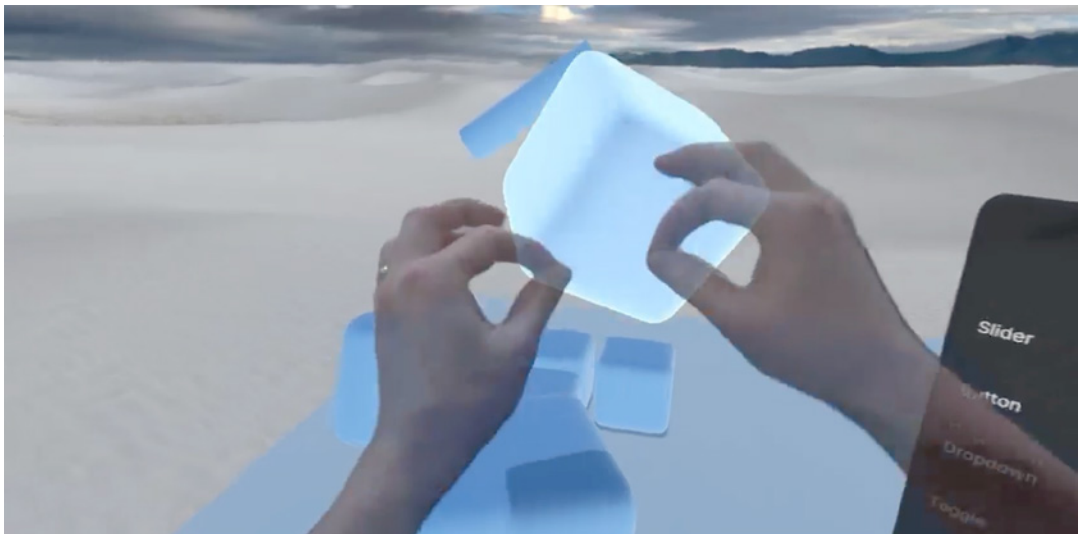
1. Open your Unity Hub, and click on the blue **New Project** button.
2. Change the Editor version to the latest **Unity 2022 LTS version** available**.**
3. In the templates list locate the **VR - Core** template. Click the **Download template** button
4. Choose a name for your project and select its location in the Project settings. Unity will now open, and you will be presented with the welcome screen.



The VR Core template in the Unity Hub

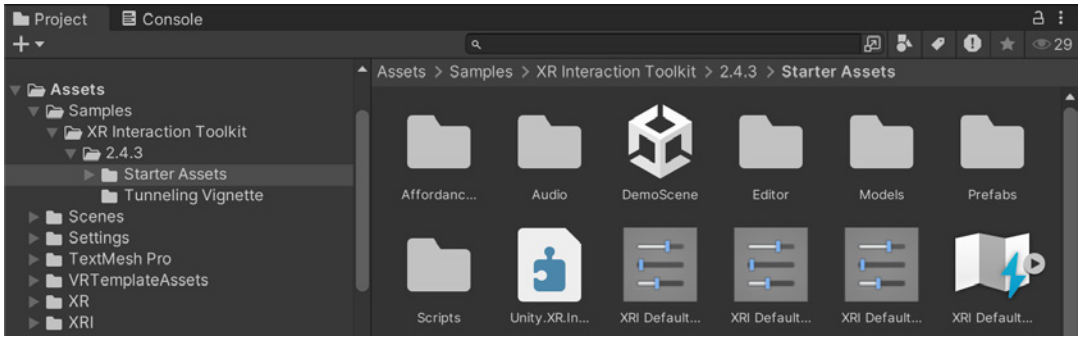
Updating to version 3.0

Note that by default the 2.5.x version of XRI might be the one in use when you start a new project based on this template from the Unity Hub (as of the release of this book). If it's already version 3.0 you can skip this section. We recommend updating to version 3.0 from the Package Manager. This update contains a series of important improvements and features, such as compatibility with visionOS. It shows best practices for handling shared space input in visionOS, which is integrated with XRI's new Near-Far and Poke interactors. You can read more about these updates in this page of the documentation.



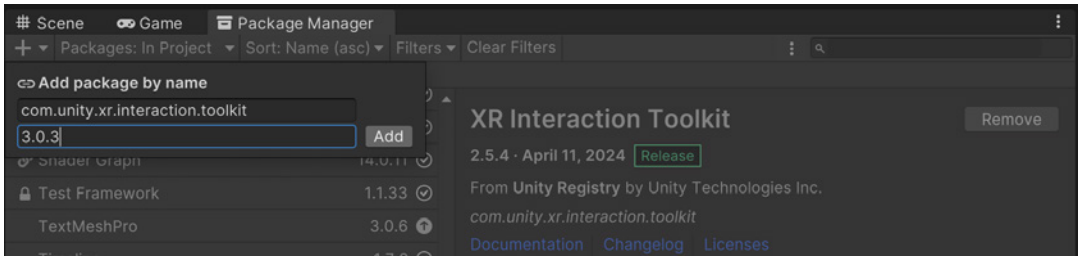
visionOS gesture examples included in the samples of XRI 3.0

Before you install version 3.0, remove the Starter Asset folder belonging to version 2.5.x to avoid conflicts, as explained in this documentation page.



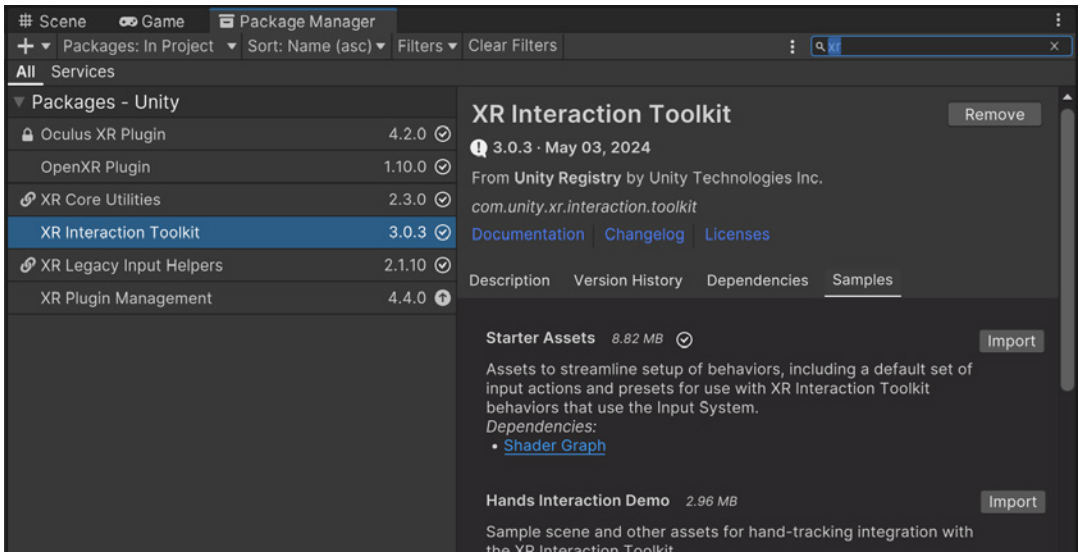
Before updating to version 3.0 of XRI, delete the Starter Assets folder

Now you can manually install XRI 3.0 if an upgrade option was not available, adding the package by name and indicating its version should work, as explained in the documentation.



Manually updating the XRI package and the sample

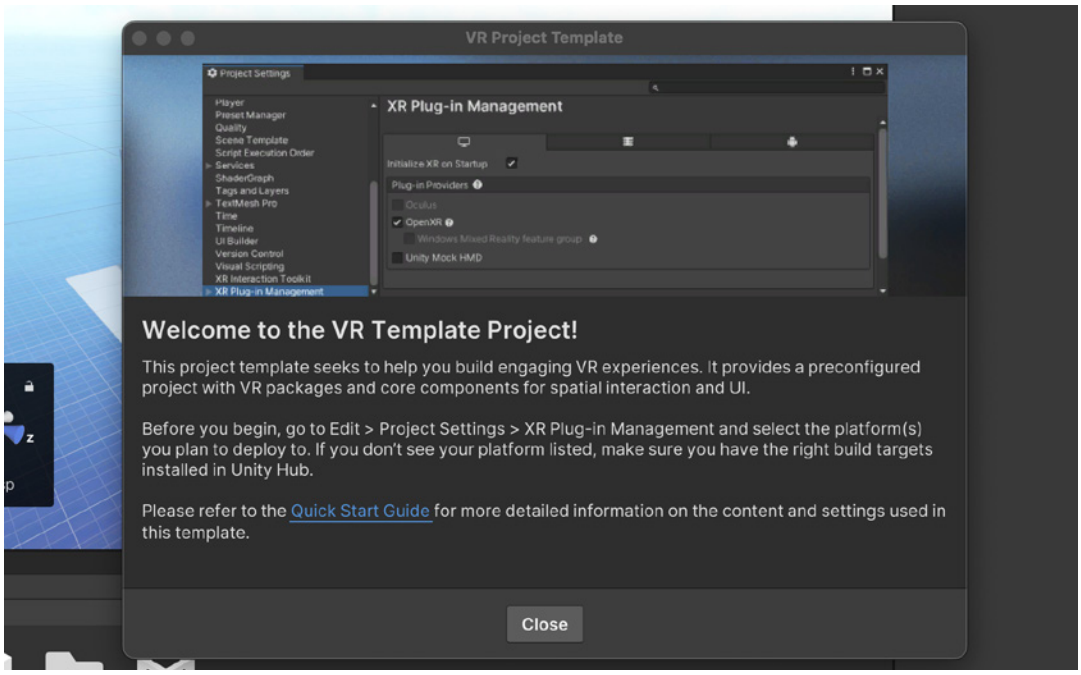
With the 3.0 version of XRI installed, a new version of the Starter Assets sample will be available via the Package Manager.



Importing version 3.0 of the Starter Assets

If you get a prompt asking to update scripts using an API that has changes, click on Yes.

The project is ready to be explored and you'll be presented with the **Welcome to the VR Template Project** window; if you don't see it, open it from the **Toolbar > Tutorials > Welcome** dialog.

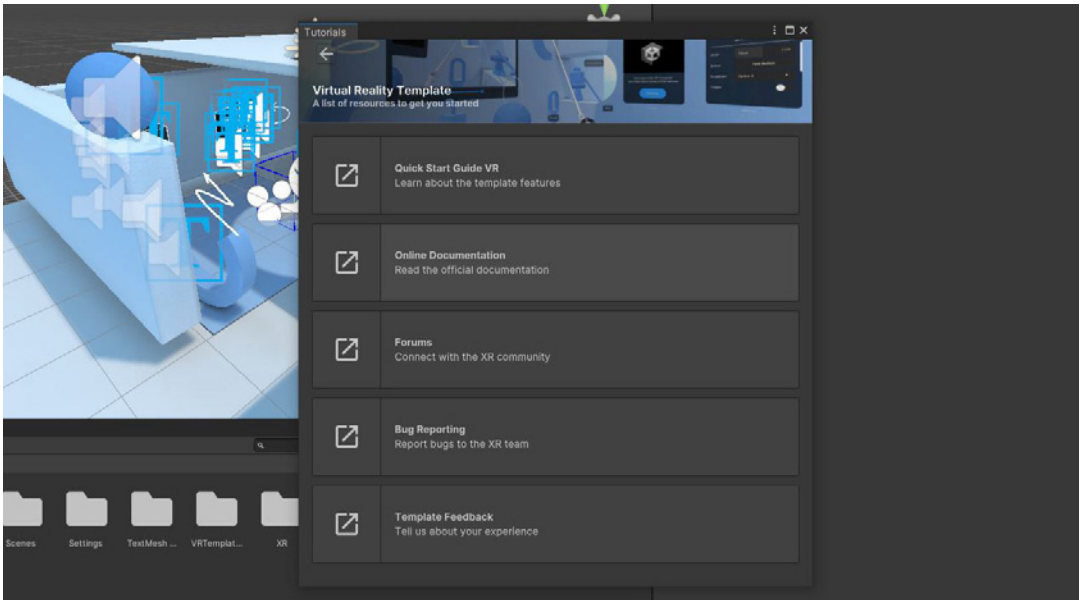


The welcome screen in the VR Template Project

The welcome window will prompt you to visit the **XR Plug-in Management** settings and select the platform you wish to deploy to. The XR Plug-in Management settings are available within the Project Settings. If you don't see your platform listed you can revisit the Hub and install additional modules. You can read more about the XR Plug-in Management settings in the upcoming section.

Tutorial window

A tutorial window will also appear providing more information on the sample project.



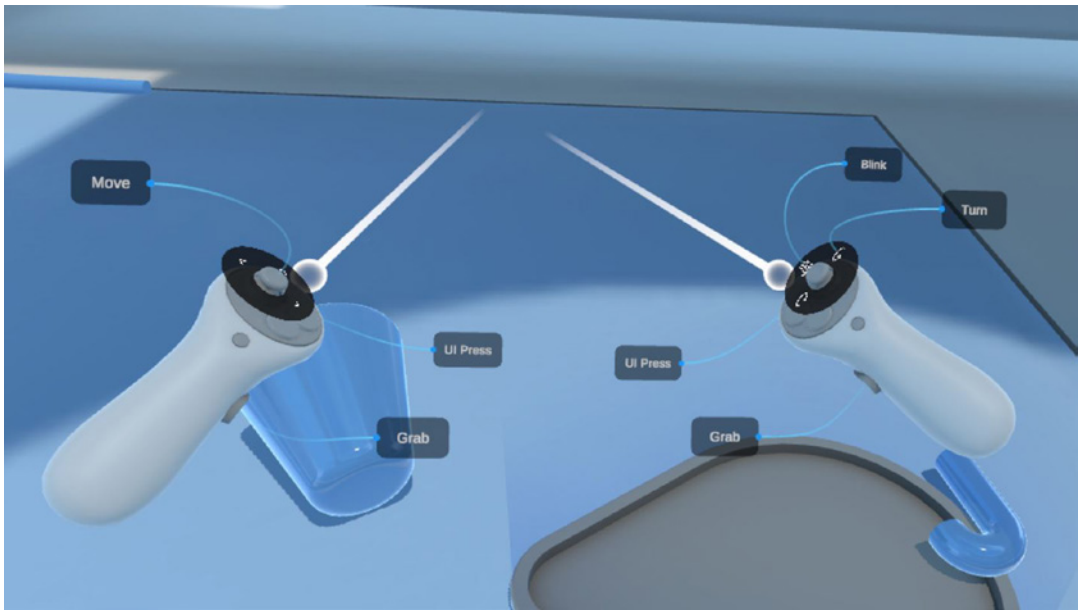
The Tutorial window

The sample scene

The sample scene included with the VR template is set up with essential elements required for interactions and navigation.

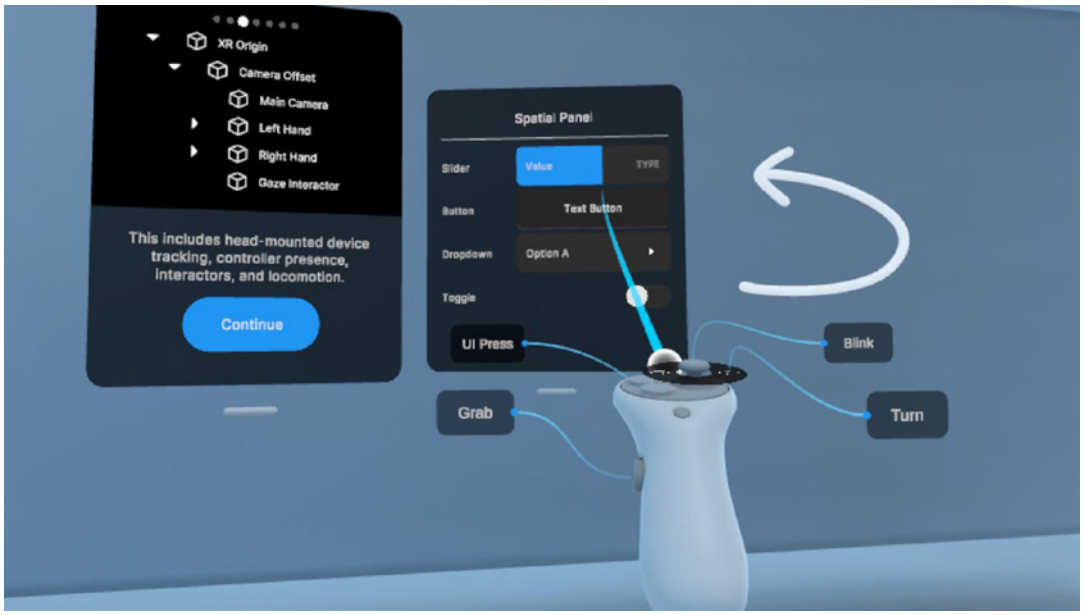
Testing the VR sample scene

Enter Play mode in the Editor to test out the features of the scene. You'll notice that the controllers include UI tooltips to help you navigate and manipulate items in the scene. If you don't have a headset the XR Device Simulator section later on in this e-book demonstrates how to use the simulator to test without one.



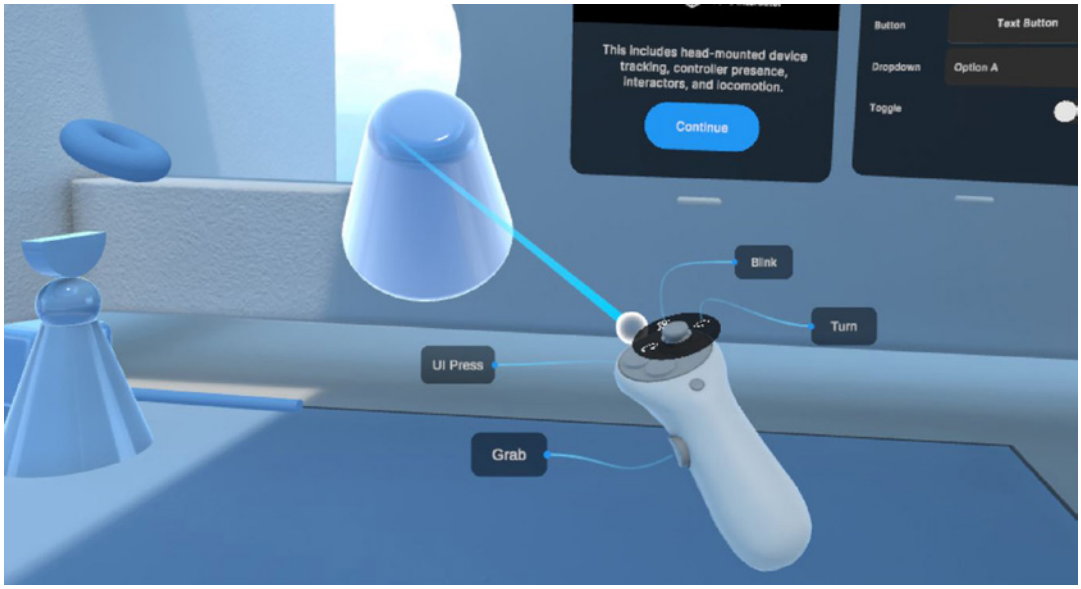
Tracked Controllers with tooltips

Use the VR controllers to interact with various UI elements.



Interacting with UI in VR

Pick up and handle the various objects in the scene.

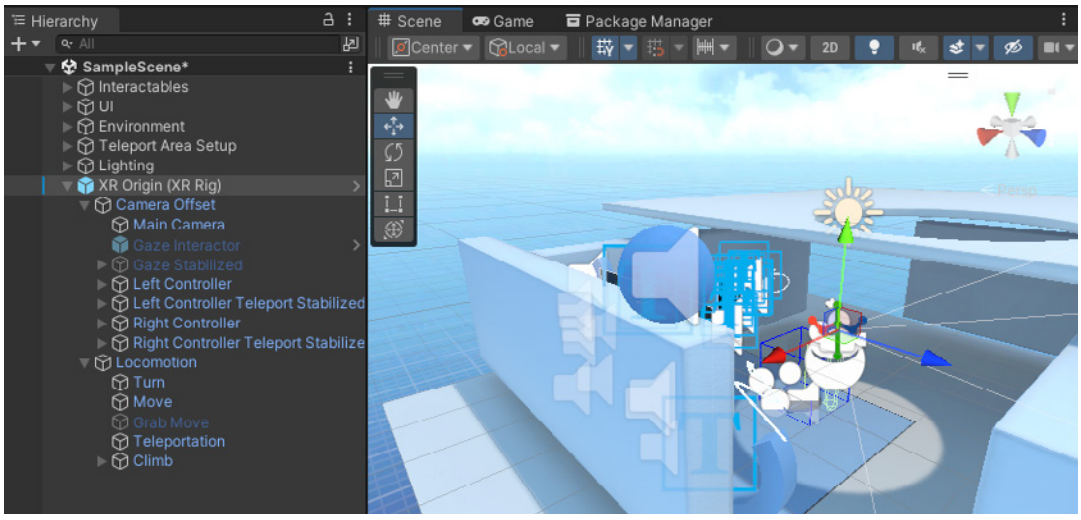


Grabbing objects using distance grab

The scene includes plenty of functionality to help kickstart your game development journey.

Sample scene setup

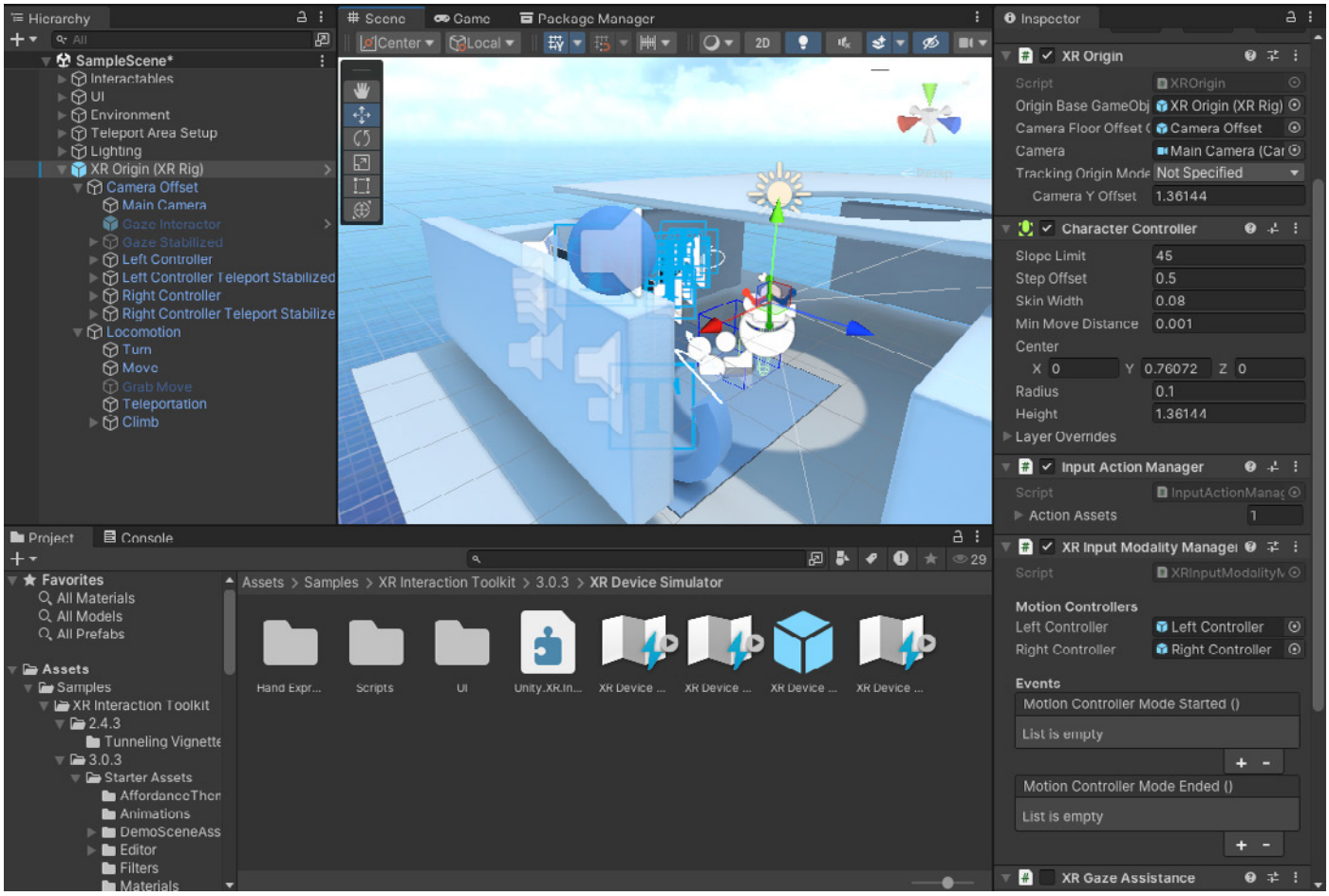
Let's look at some of the core components in the sample scene.



The prefab variant Complete XR Origin Setup Variant in the sample scene Hierarchy in XRI 2.5.x

The Hierarchy includes a prefab variant called **Complete XR Origin Set up Variant** in version 2.5.x that contains all the required components for you to start navigating and interacting with the VR world. After updating to XRI 3.0.3 you should remove that prefab and add the **XR Origin (XR rig)** prefab from the Starter Assets in 3.0.3.

It includes the setup for everything required for fully functional user interaction including grabbing, engaging with UI components, and navigation using locomotion techniques like teleporting, snap turning, and moving around using the thumbsticks on the controllers.



You'll need to manually add the prefab variant XR Origin (XR Rig) to the sample scene Hierarchy in XRI 3.0.3

Additionally, the prefab contains all the core elements like the following:

— **Input Action Manager:** This class is responsible for managing the enabling and disabling of input actions within XR environments.

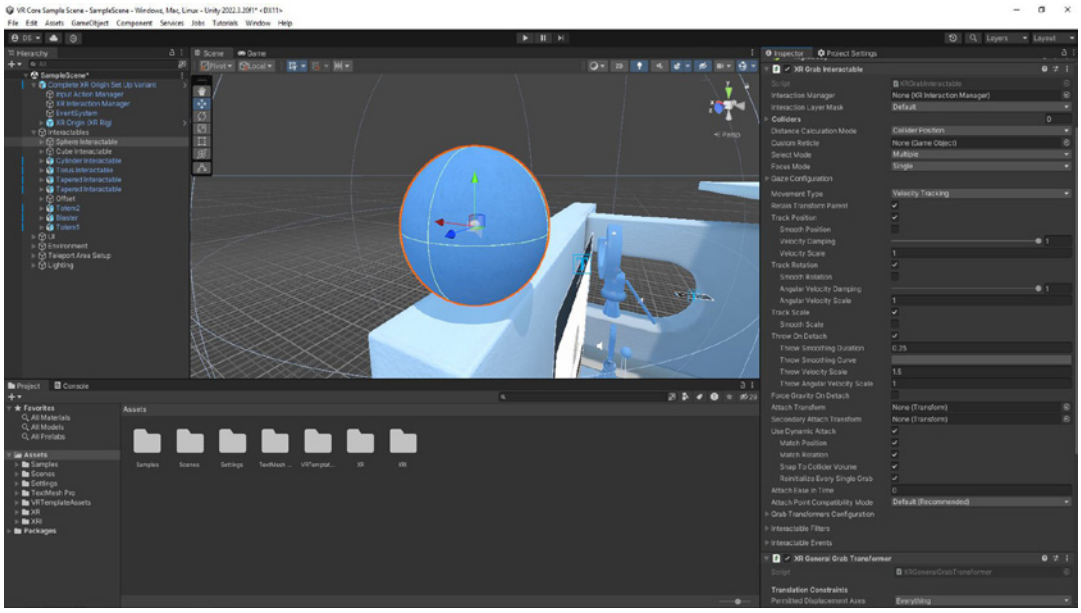
— **XR Interaction Manager:** This is responsible for managing interactions between XR controllers, interactable objects, and the environment.

— **XR Origin:** This is responsible for managing the spatial origin for XR sessions; this plays a key role in translating physical movements and orientations into the virtual world. It contains the camera, left and right controller tracking, and the locomotion setup.

An upcoming section provides more details on these components.

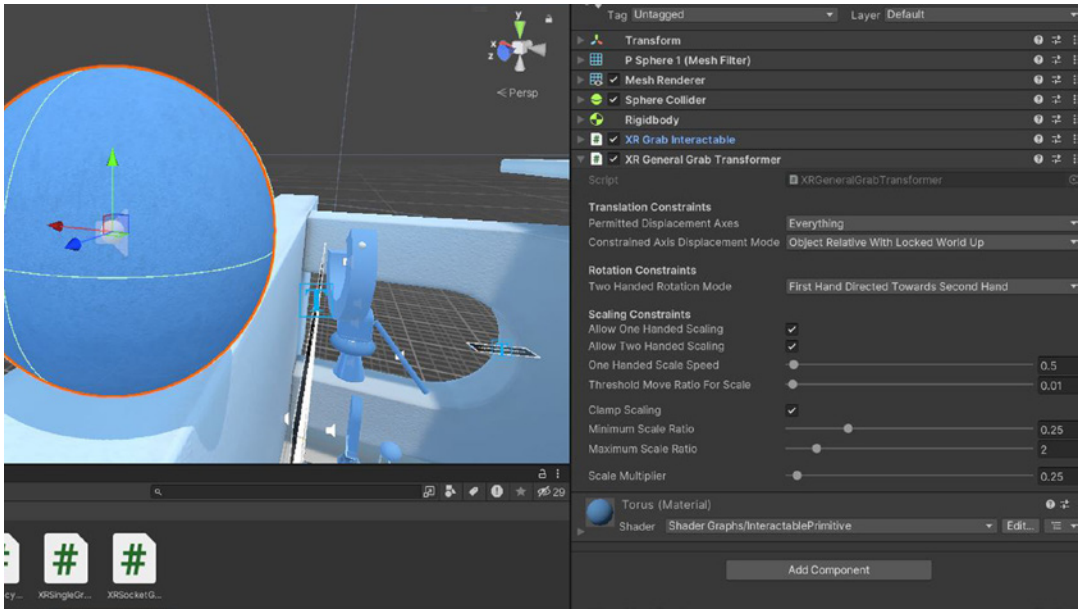
The Hierarchy also contains a parent GameObject called **Interactables** that contains all the GameObjects that can be engaged with in various ways.

For example, there are a couple of regular grabbable objects such as the sphere, made possible with the XR Grab Interactable, a component of the XR Interaction Toolkit.



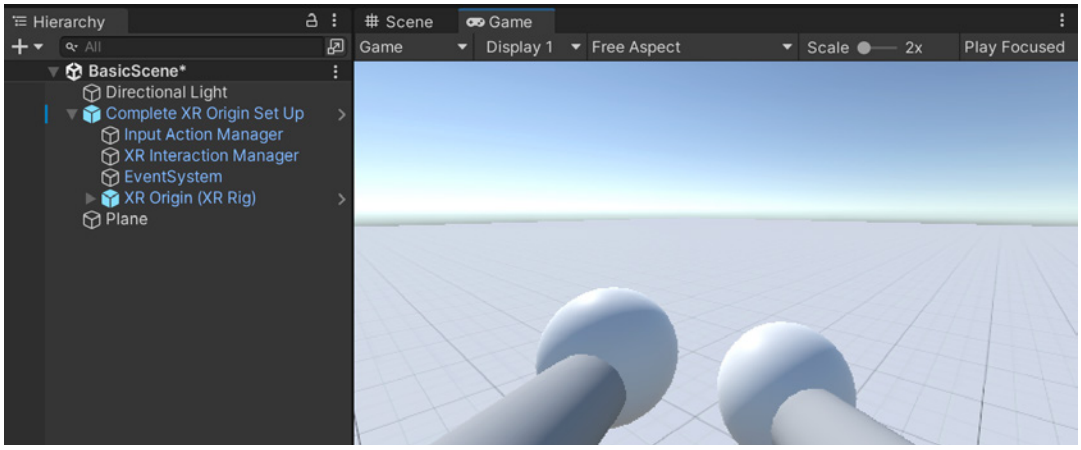
A grabbable object with the XR Grab Interactable component

The interactables can also be scaled up and down when grabbed with two hands in VR; this is made possible with the XR General Grab Transformer script.



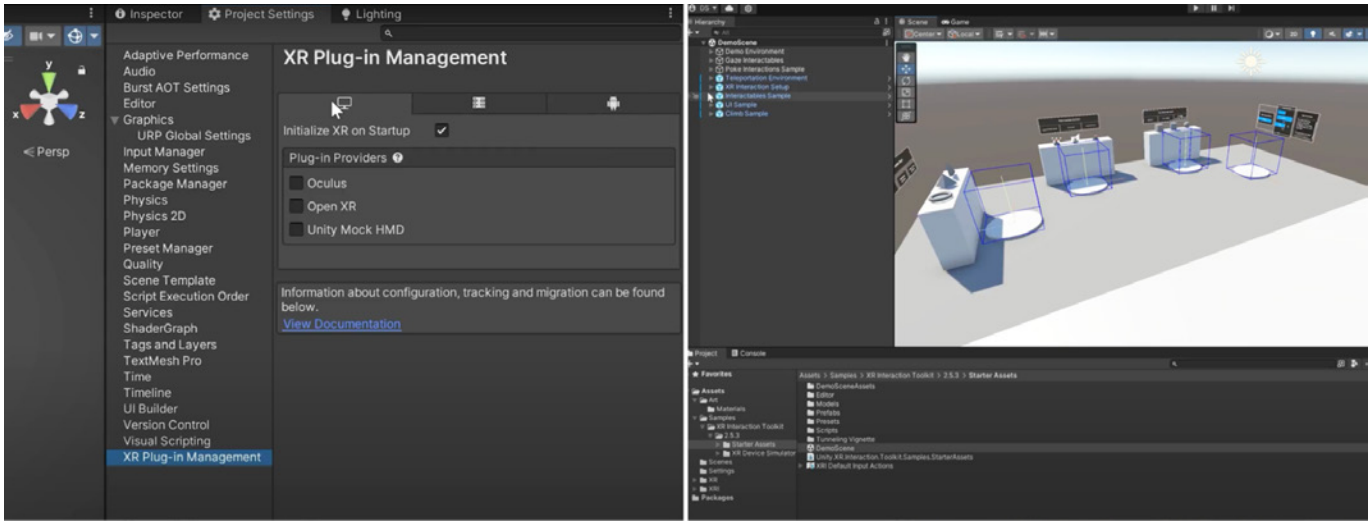
The XR General Grab Transformer allows for object manipulation.

The project contains another scene called BasicScene which contains the setup prefab ready for you to add your environment and interactions.



The BasicScene, ready to add your environment with the controls already setup; remember to delete the Complete XR Origin Set Up and add XR Origin (XR Rig) if you are on XRI 3.0.3

You can watch the following tutorial to follow along with setting up the sample.



Watch the tutorial

Project templates

More examples of how to use various features are available from Github for you to download and explore:

— XR Interaction Toolkit Examples

— Mixed Reality (MR) Example for Meta-OpenXR

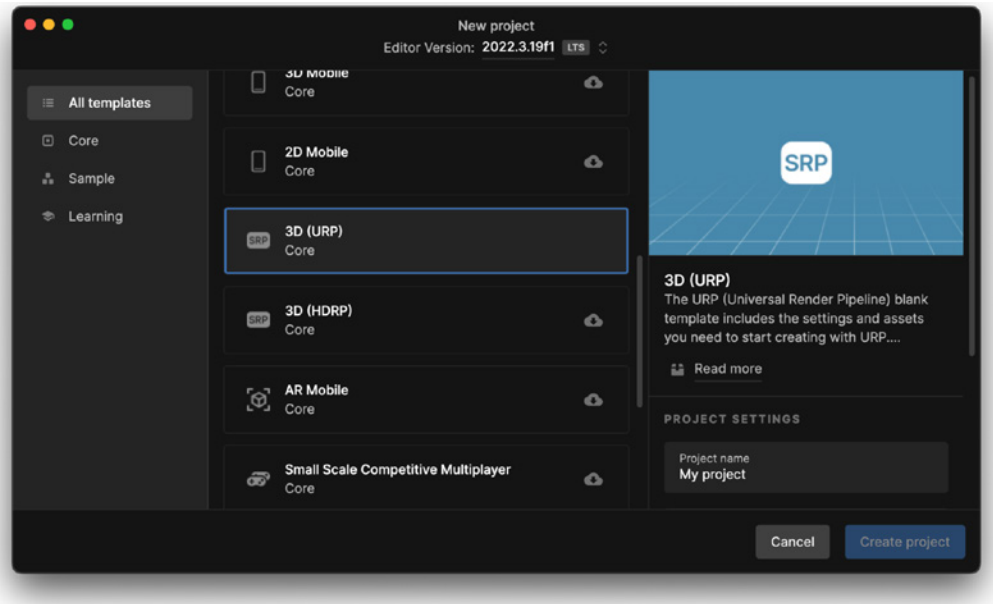
— XR Simulation Environments

Create a new VR Project using URP

If you follow the steps for setting up the VR Core template then you should be ready to move on to starting a new empty VR project and looking in detail at the main components.

Start with the 3D URP template

To start a new project, open the Unity Hub, select **New Project**, and choose the latest version of **Unity 2022 LTS.** From the template list scroll down to **3D (URP)** and download it.



Choose the 3D URP template in the Unity Hub

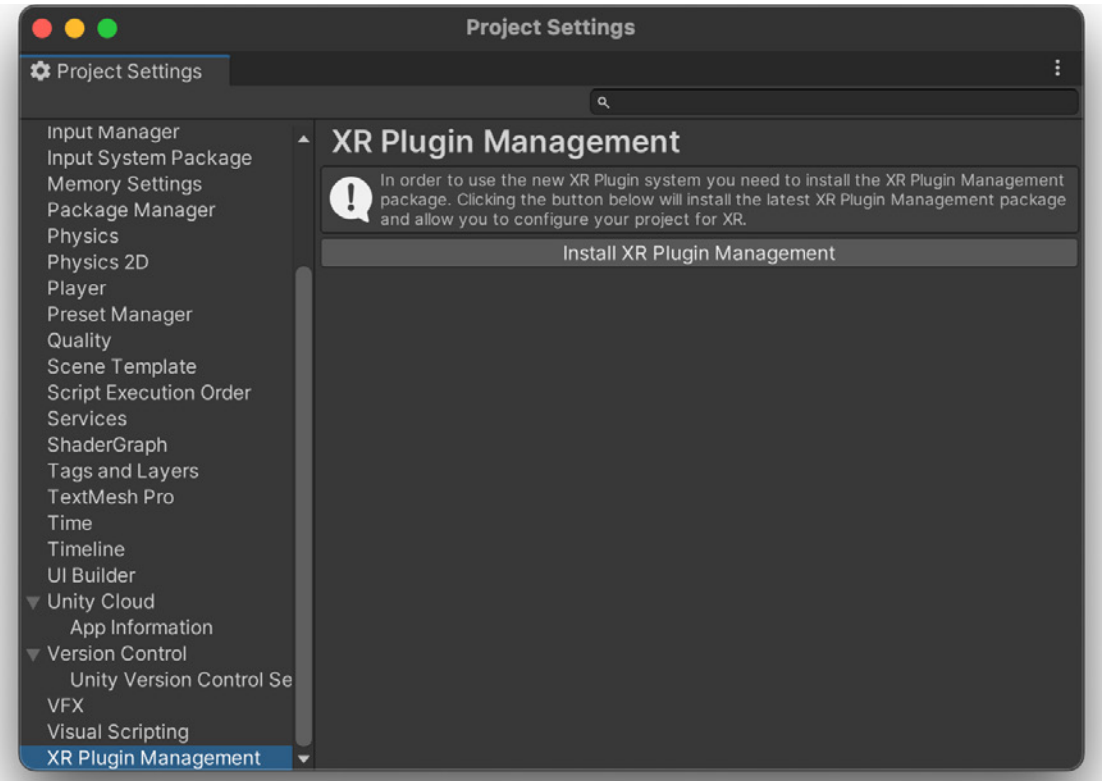
Fill out the project settings and click Create Project.

In the Editor an Inspector window opens with links to URP documentation, forums and bug reporting.

XR Plug-in Management

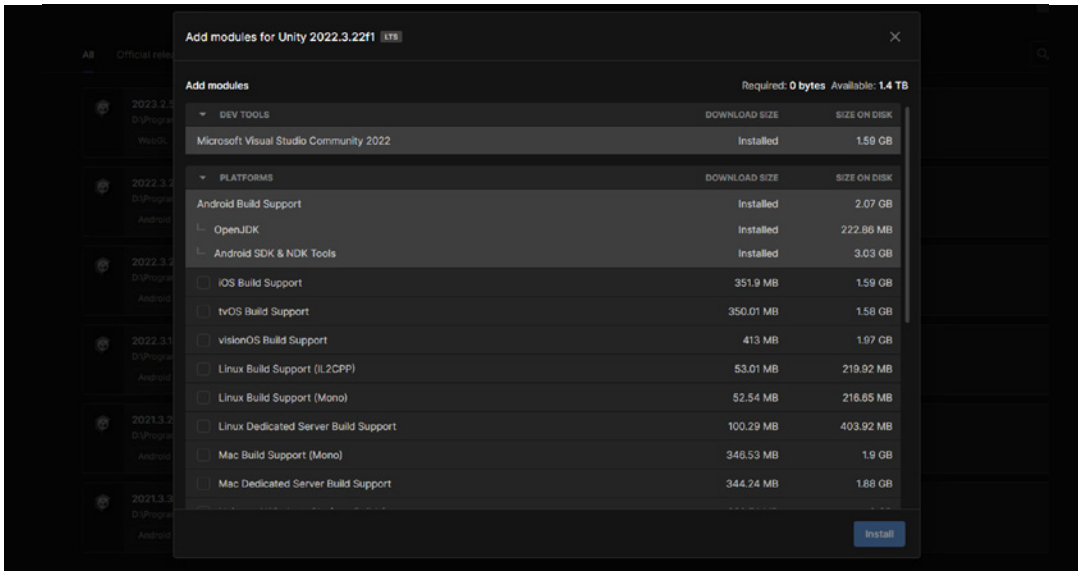
The XR Plug-in Management package, which is available via the Project Settings window, manages the integration and configuration of XR plug-ins. This includes settings for VR, AR, and MR devices. It allows you to select and configure the appropriate XR SDKs and plug-ins that your project will use to support various hardware. Essentially, it serves as a centralized hub for setting up and customizing how your Unity project interacts with different XR technologies.

Navigate to the Project Settings window and select the **XR Plug-in Management** package from the list and install it by clicking on the bar.



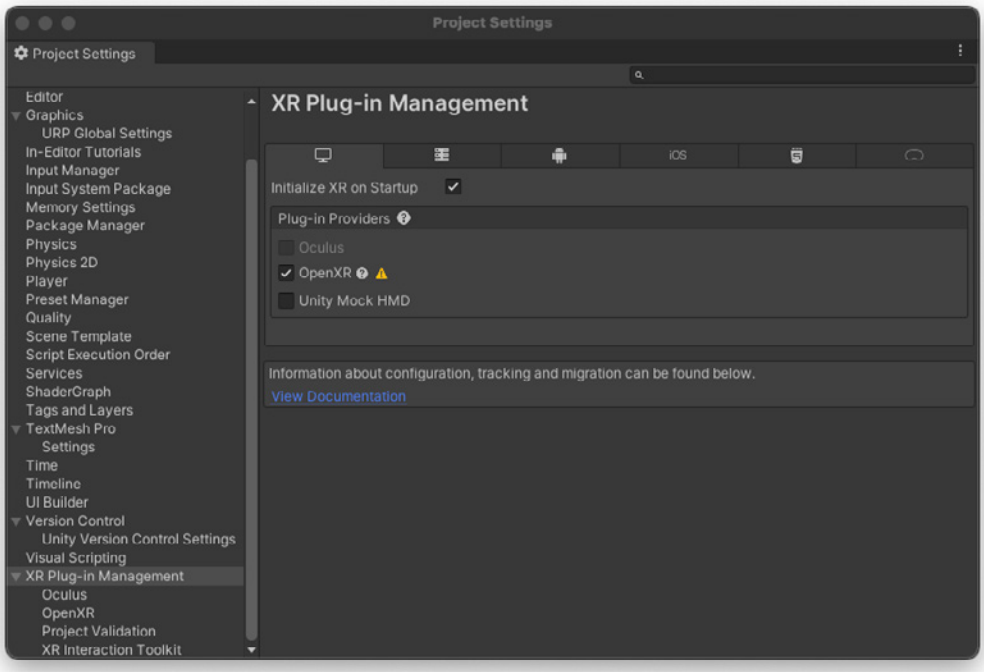
Installing the XR Plug-in Management package

Now you can select your build platform and plug-in provider. Ensure that your required build module is installed. If you don't see your platform listed, go to the Unity Hub and locate your installs. From here you can add any modules by clicking the cog icon and then **Add Modules**.



Adding a build module through the Hub

Go back to the Editor and your platform(s) should be listed. Select a **Plug-in Provider** for each platform. For this example we'll select **OpenXR.**



Selecting OpenXR as the build target

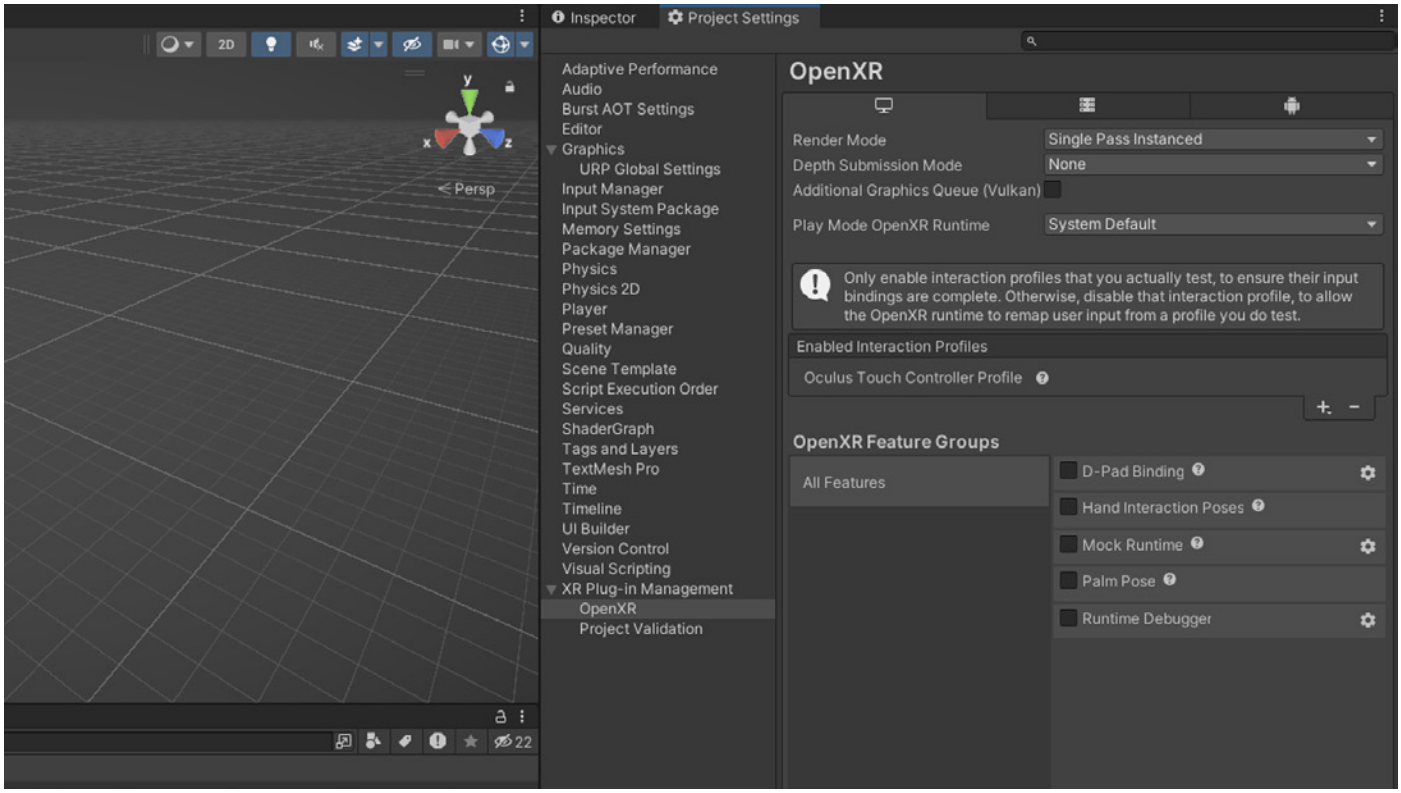
Note: To enable this input system you might be prompted to restart the Editor; when the Editor restarts you will be able to check the OpenXR option if it's not already selected.

Add an interaction profile

Use an interaction profile in the XR Plug-in Management package to set up presets so your game understands how to interact with your target device. For example, if your game supports Meta Quest then you would choose the Oculus Touch controller profile. You can assign multiple profiles if you're targeting multiple devices.

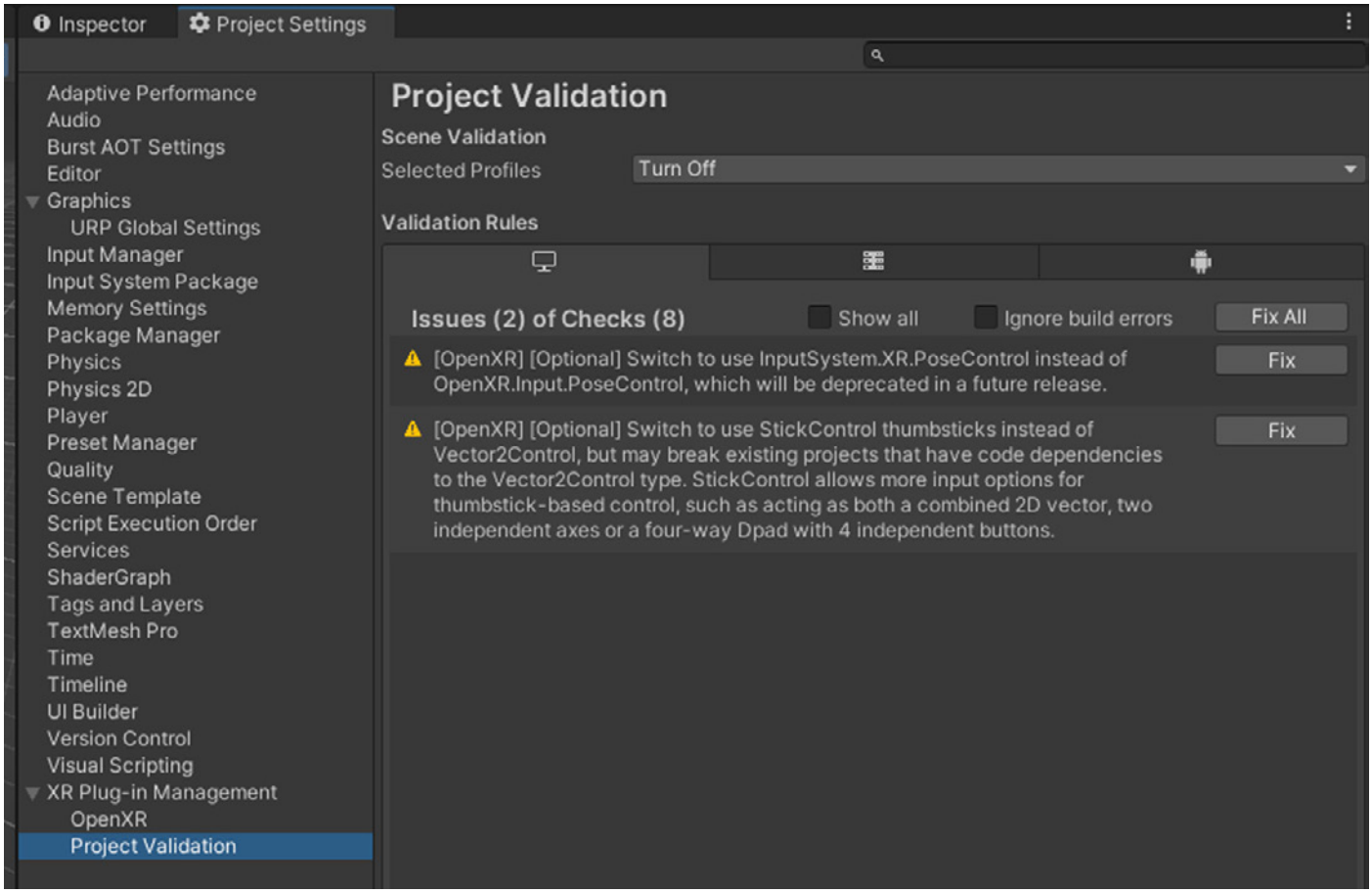
Once the XR Plug-in Management package is installed and set to OpenXR you can allocate an interaction profile to OpenXR so that when you run your application the controllers are tracked. This depends on your target hardware.

Under the section called **Enabled Interaction Profiles** click on the **+** icon to select your controller type from the list.



Selecting an interaction profile

Check the Project Validation tab to ensure there are no warnings. This can be done quickly by selecting **Project Validation** in the left-side column. Any issues will be listed here; click on **Fix All** to resolve them.



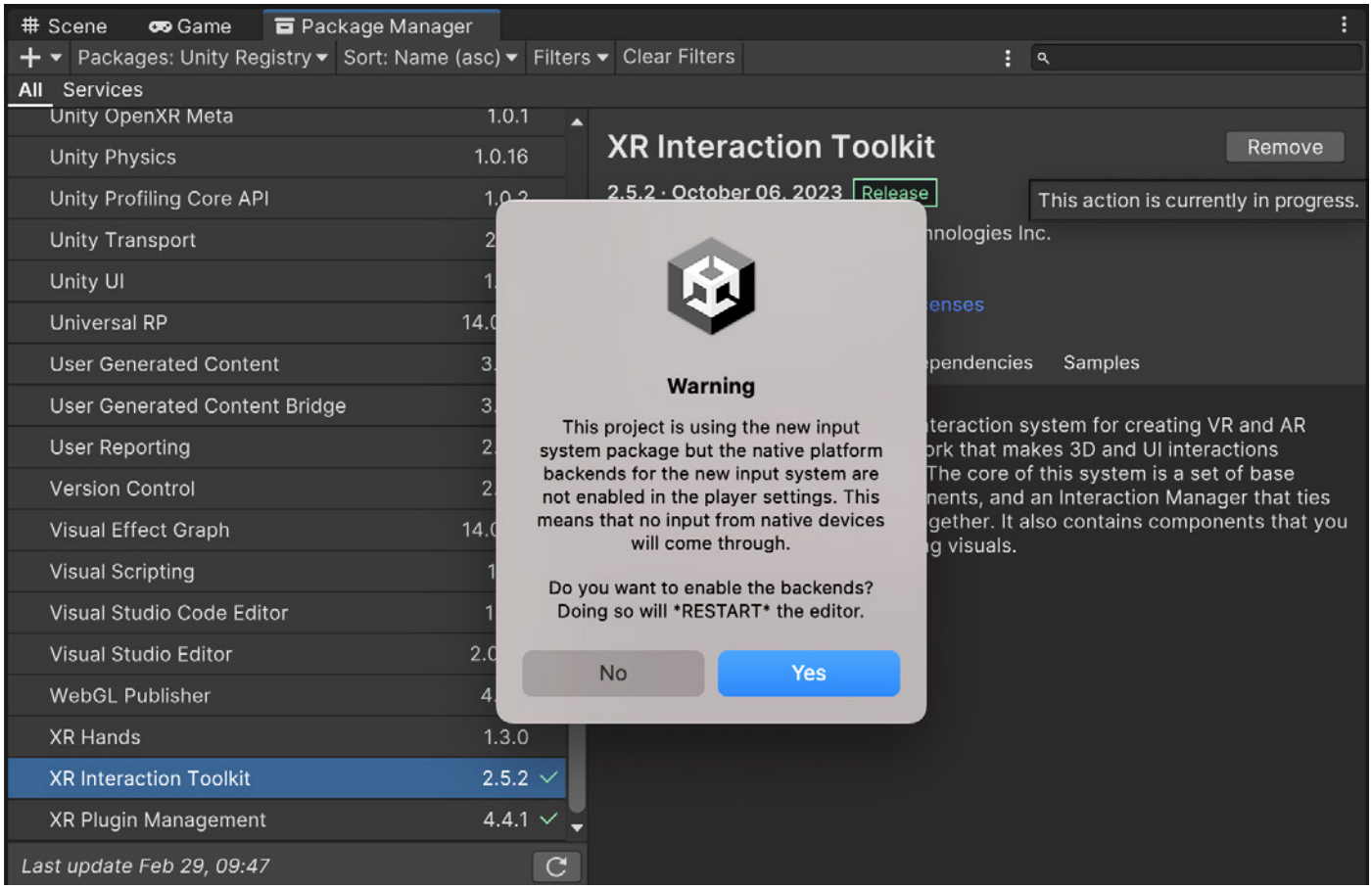
Project Validation

The next step is to install the XR Interaction Toolkit (XRI) package.

Implement XRI Toolkit

You'll find an introduction to the XRI Toolkit earlier in this guide; now let's look at how to implement it.

Via **Window** > **Package Manager**, choose the XR Interaction Toolkit from the Unity Registry dropdown, and click Install. Click on **Yes** if prompted about enabling the Input System backend and the Editor will restart.

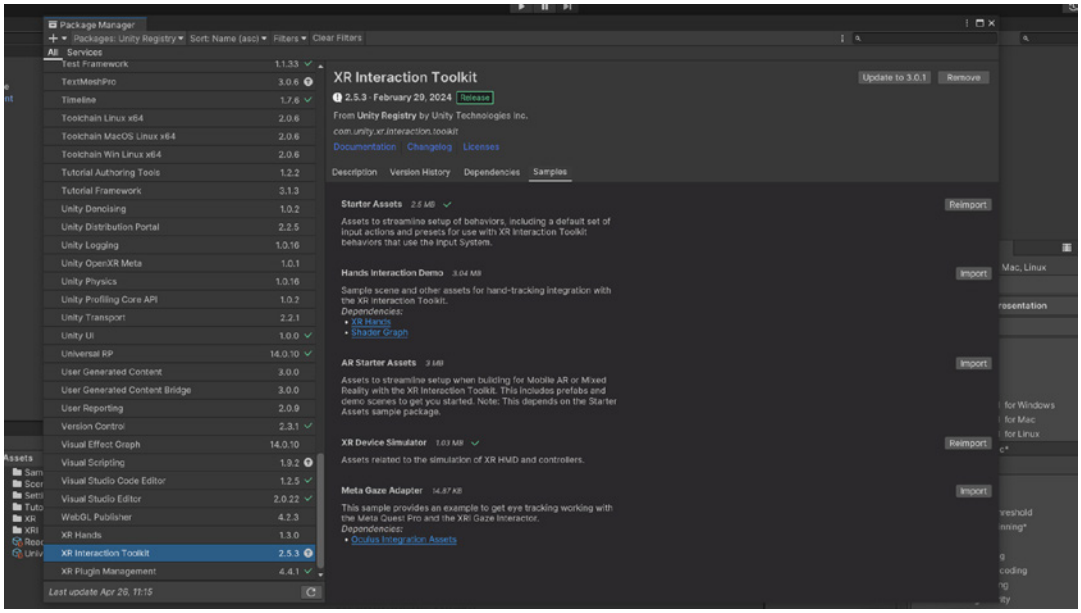


Installing the XRI Toolkit package

Using the Starter Assets to explore XRI

The steps in the following section are based on using the Starter Assets included in the package. This can be an efficient way to streamline the setup of behaviors, input actions, and a demo scene that showcase features of the XR Interaction Toolkit.

If you're interested in this approach then click on **Samples** and import the Starter Assets sample into the project.

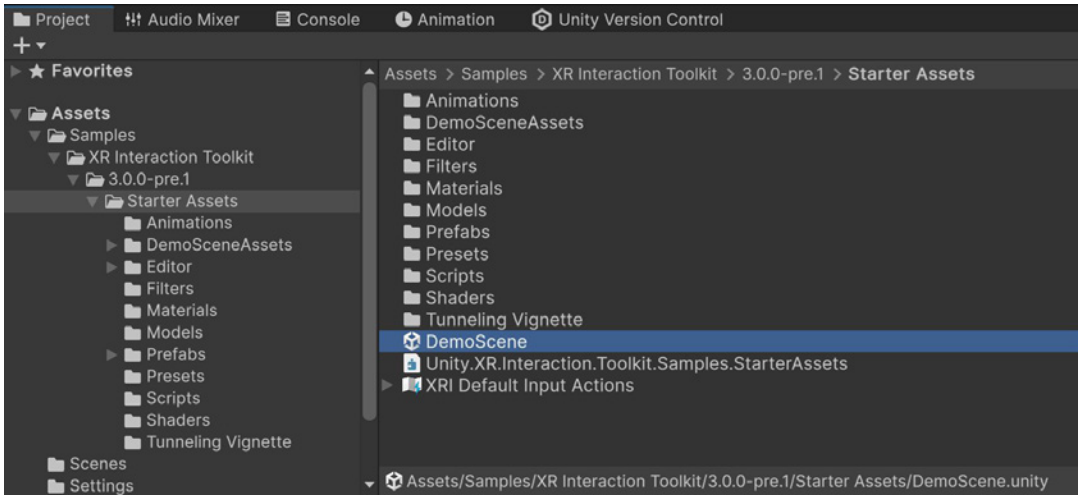


XR Interaction Toolkit Starter Assets

The demo scene

In this section we'll examine the role of each of the prefabs and components included in the XRI Toolkit demo scene. If you can't find them, make sure to have the sample installed as explained in the previous section.

The Starter Assets sample includes a demo scene via **samples folder** > **XR Interaction Toolkit** > **(toolkit version)** > **Starter Assets > DemoScene**.



This demo scene includes everything you need to start creating your own VR games:

— Navigation

— Grab interaction along with various movement types

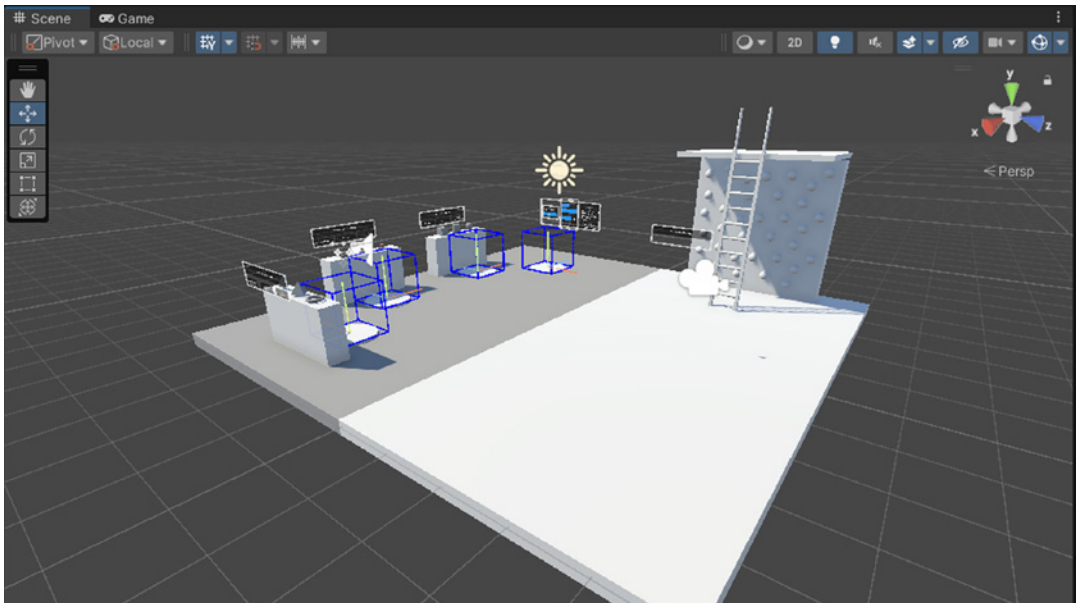
— Poke interaction

— Gaze interaction

— Climb interaction

— UI interaction

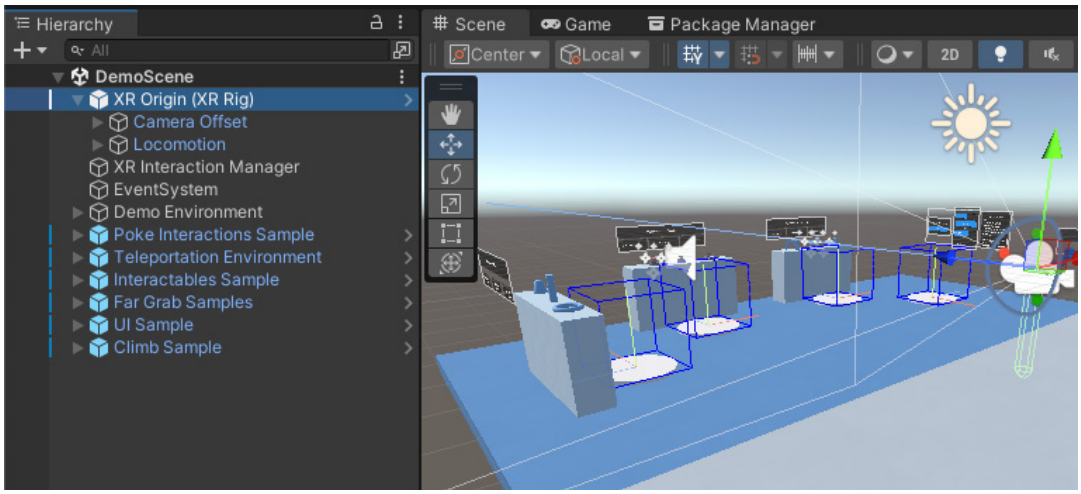
— Climbing



The XR Interaction Toolkit demo scene

XR interaction setup in the Starter Assets demo scene

The demo scene in XRI 3.0 contains the **XR Origin (XR Rig) prefab** which includes the **Input Action Manager** component and **XR Interaction Manager** GameObject.



The XR Origin (XR Rig) prefab

Input Action Manager

The **Input Action Manager** is responsible for managing the enabling and disabling of input actions within XR environments. It works with a list of InputActionAsset objects, allowing for centralized control over input actions across different assets. It ensures that actions are responsive when needed and dormant when not.This is particularly useful in VR and AR scenarios where dynamic input handling is crucial for user interaction and experience.

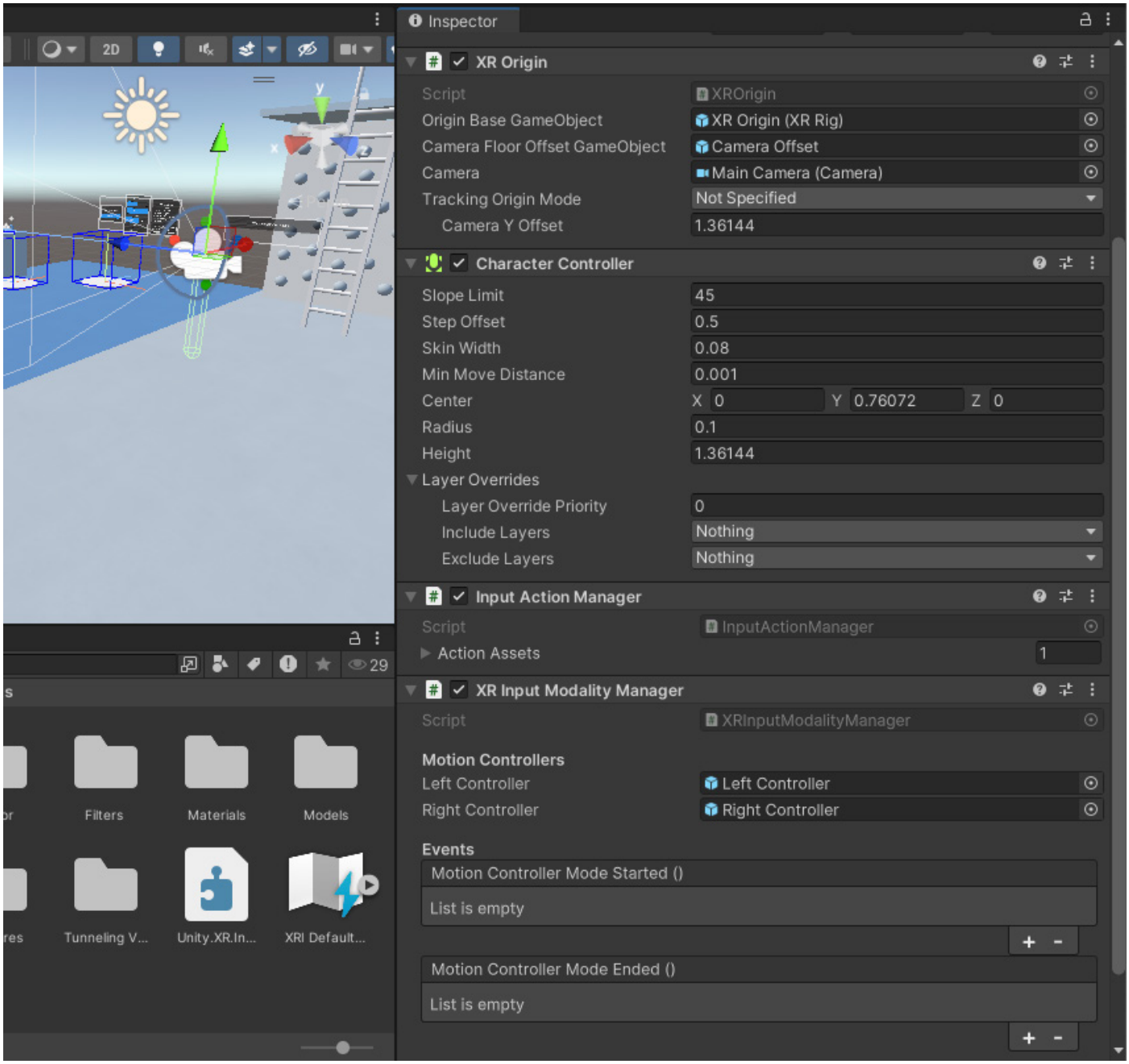
XR Interaction Manager

This manager acts as the central hub for managing interactions between Interactors (e.g., VR hand controllers, pointers) and interactables (objects in the VR/AR space that can be interacted with, like buttons or props the player can grab) It ensures that when interactors and interactables are enabled, they register themselves with an Interaction Manager so they can communicate and function properly in the scene. See the XR Interaction Toolkit Glossary for more information.

XR Origin

The XR Origin prefab is responsible for managing the spatial origin for XR sessions. It handles camera positioning in relation to XR devices with support for different tracking modes such as “floor” or “device” level. It plays a key role in translating physical movements and orientations into the virtual world. It’s essential for setting up the XR environment in Unity, ensuring accurate and responsive interactions within the virtual space.

The XR Origin prefab contains a **Character Controller** and the **XR Input Modality Manager.**



XR Origin components

The Character Controller Driver

The **Character Controller Driver** is designed to adjust a virtual character's height and position on the real world movements of the player's VR headset. It listens to when the player starts and stops moving in the VR space and updates the Character Controller component that handles movement and collision.

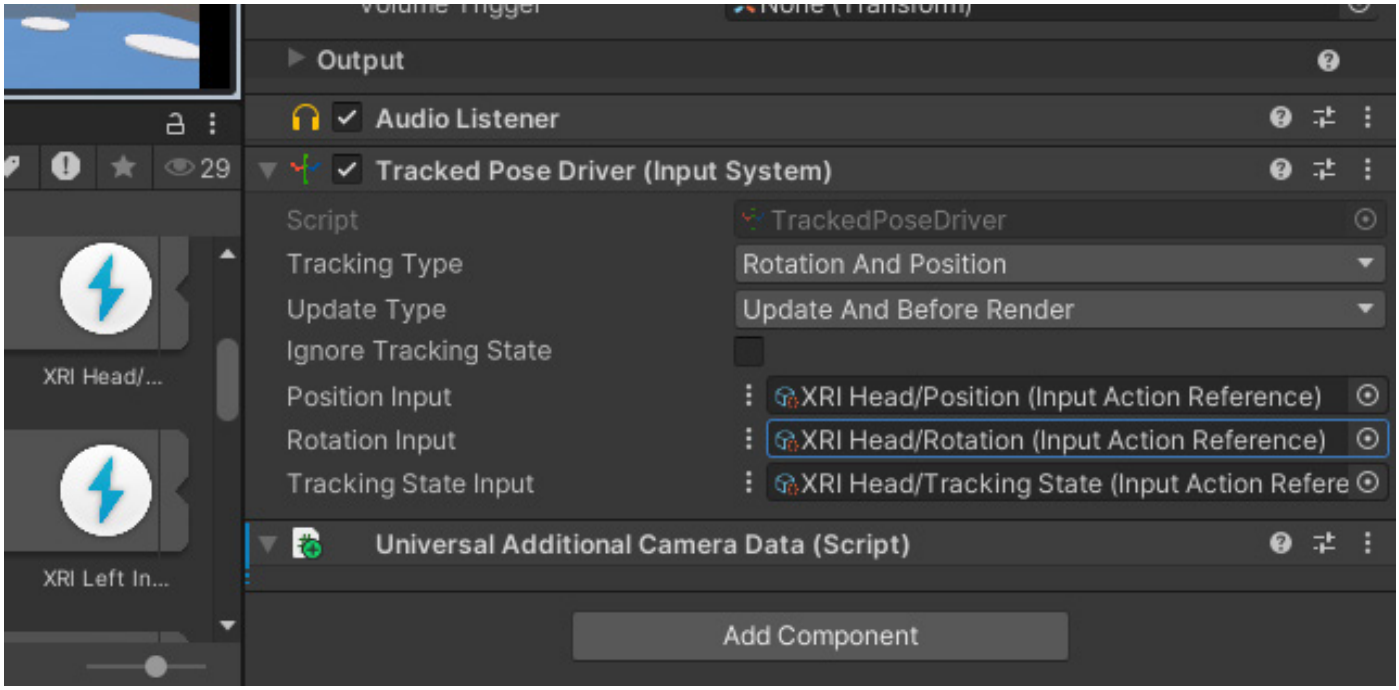
XR Input Modality Manager

The **XR Input Modality Manager** manages the dynamic switching between hand tracking and motion 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, it activates the motion controller group. This component is beneficial even in projects not using hand tracking, as it prevents showing controllers in their default position when not tracked.

The XR Origin prefab has two children GameObjects: The **Camera Offset** that houses the main camera and the functionality for tracking the controllers, and the **Locomotion System**.

TrackedPoseDriver (Input System)

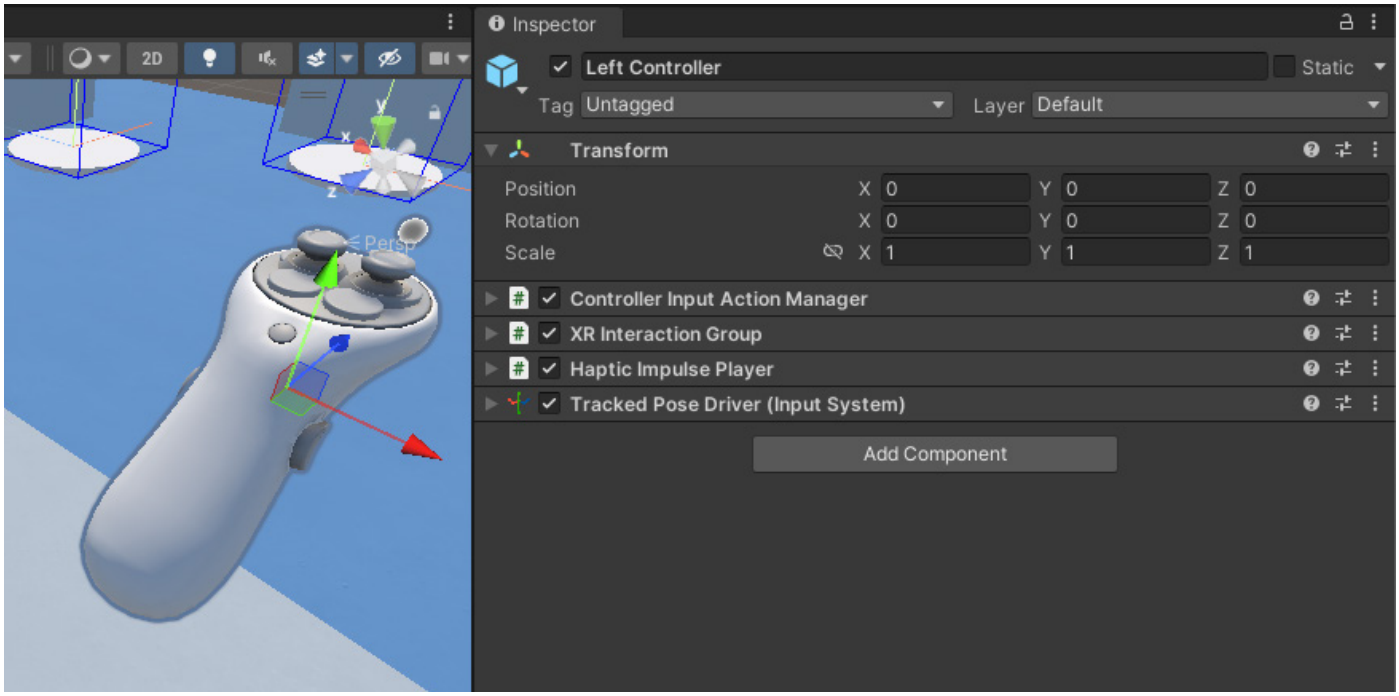
The main camera has a **TrackedPoseDriver (Input System)** component. This plays a crucial role in mapping real-world physical movements to virtual ones like tracking of the user's head in real-time so the motions can be translated into VR.



The Tracked Pose Driver (Input System) component on the main camera

Inputs are mapped to the input actions supplied by the XRI Toolkit so that they're ready to use.

The setup for the Left and Right Controllers is also contained with the Camera Offset. Both of these objects have the same components, the **Controller Input Action Manager**, and an **XR Interaction Group component**.



Components for the Controllers

Controller Input Action Manager

The Controller Input Action Manager is a component on both the left and right controllers. It helps manage how the VR controller interacts with objects and responds to user inputs in the VR application. It can switch between different ways of moving and interacting like pointing at distant objects with a ray or teleporting around.

It ensures that only the appropriate actions (like moving or teleporting) are possible at any given time, based on what the player is doing.

XR Interaction Group component

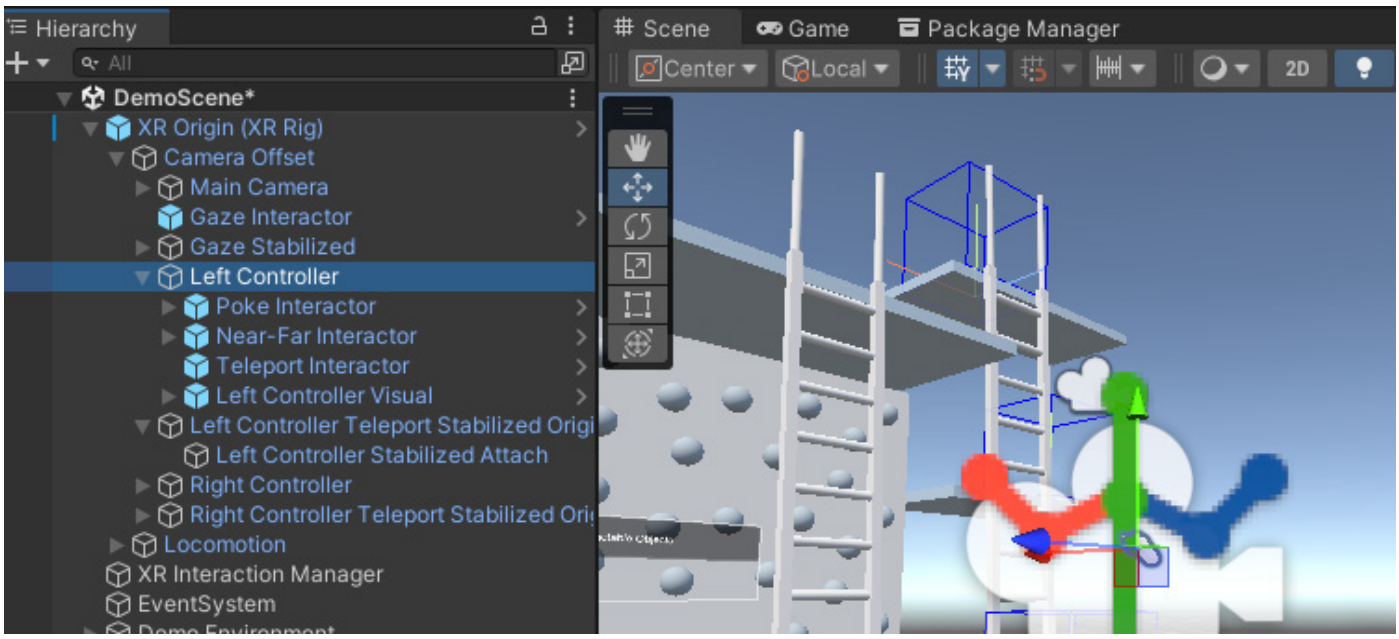
The XR Interaction Group Component serves as a system to manage a group of interactive elements, ensuring that within the group only one interactive element or interactor can be active at a time.

XR Transform Stabilizer

The last set of objects within the XR Origin prefab are the **Left/Right Controller Stabilized Origin** GameObjects. Both have the **XRTransformStabilizer** script attached, which helps make the XR environment feel more precise and less prone to the effects of small, unintended movements. Assigning the transforms of the Left and Right Controllers as the target object will smooth out the movement, stabilizing any jittery or erratic movement in the tracking data.

Interaction setup

The systems that control teleporting, grabbing, poking, and interacting are all contained within the Left and Right Controllers.



Interaction setups contained within the Left and Right Controller

Each of these interactors have their corresponding component attached; the **Poke Interactor** GameObject, for example, which allows the player to pick up objects, has a component called **XR Poke Interactor**.

XR Direct Interactor

The XR Direct Interactor component enables direct interaction with interactables that can be touched, moved, or grabbed by the player, like buttons, levers, and many other objects and props.

The Poke, Ray and Teleport Interactors all allow for specific interactions within the VR world; we'll take a look at the various interactables later in the e-book.

Locomotion System

This manages and controls access to the XR Origin prefab, which represents the virtual space's reference point, typically where the player or user is located. It ensures that only one Locomotion Provider can move the XR Origin at any given time, preventing conflicts and ensuring a smooth and predictable movement experience in XR applications.

The methods for manipulating the player's position are turn, move, teleportation and climbing.

Turn

The XRI Toolkit provides two components for player rotation: **Snap Turn** and **Continuous Turn**. Snap Turn allows players 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 toolkit comes with preconfigured input actions, simplifying development and ensuring these turn components are ready to use with minimal setup.

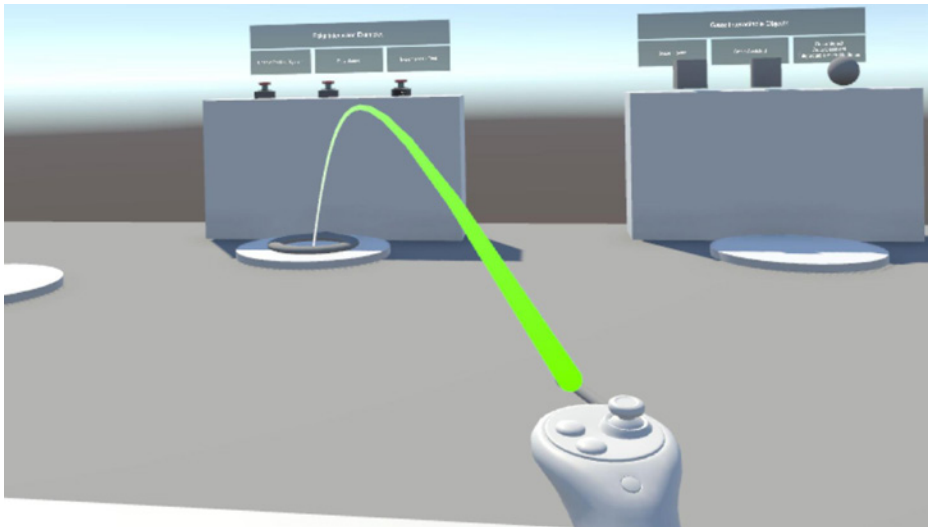
Move

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, such as head-relative or hand-relative, depending on the user's choice for each hand. This provides a customizable and intuitive movement experience in XR environments.

Teleportation

Teleportation in the XRI Toolkit allows players to move instantly to a new location within the virtual environment. This is achieved by aiming at a target location, called a **Teleportation Anchor**, and initiating a teleport action. The teleportation system 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 to a specified location at the user's request. Key features include a delay time setting for teleportation, which can be used for comfort settings like a tunneling vignette effect. It also supports different types of orientation matching for the destination, like world space up, target up, and target up and forward. The provider can queue teleport requests and manage the application of these requests, including orientation and positioning transformations, to achieve the desired teleportation effect.



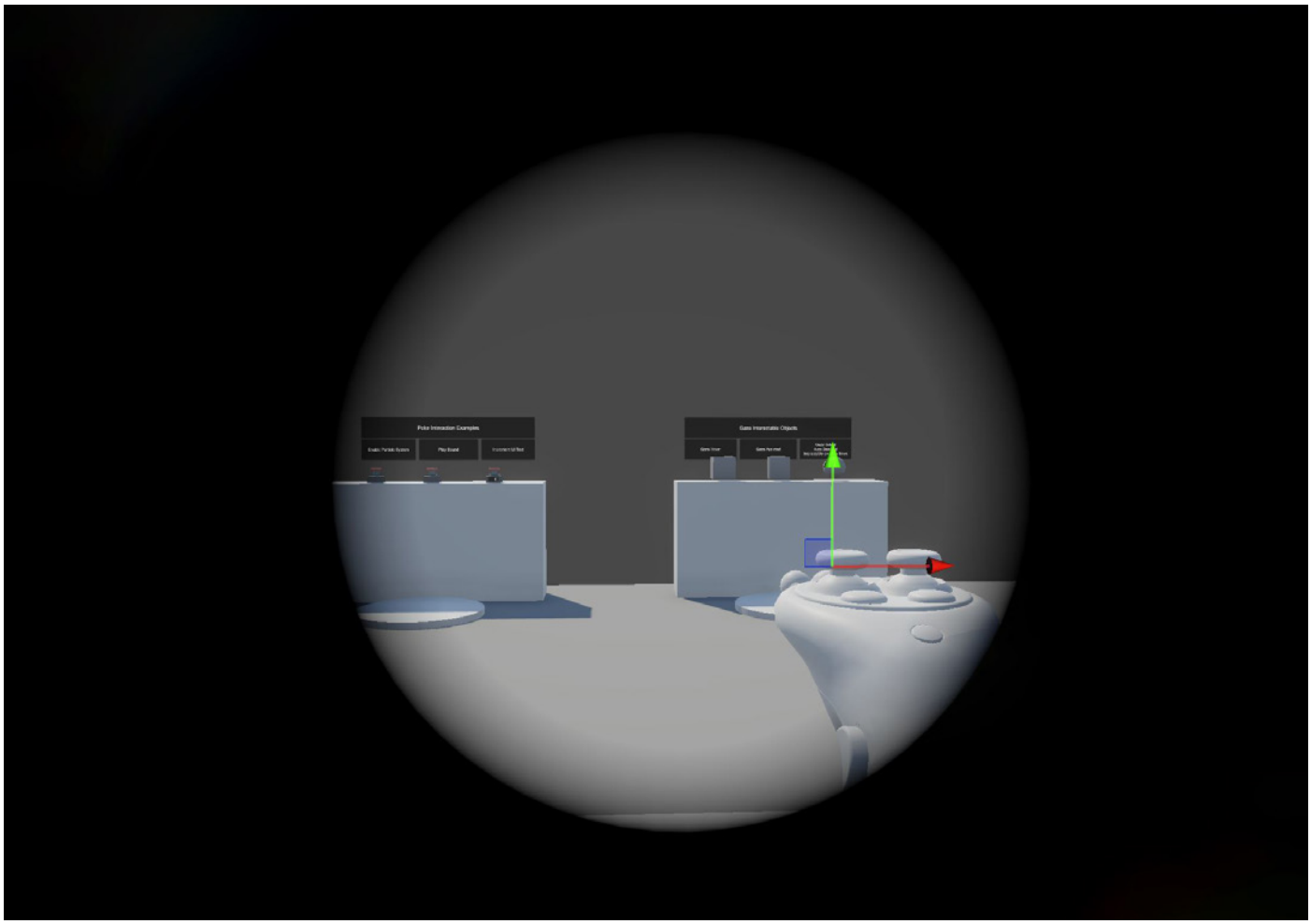
Teleporting using anchor points

Climb

The **ClimbProvider** component facilitates climbing mechanics in XR environments. It enables users to interact with ClimbInteractable objects, allowing them to move the XR Origin in response to the user's actions. The climbing motion is counterbalanced by the movement of the user's interactor, and can be constrained along each axis of the interactable.

Tunneling Vignette

The sample scene also includes a **TunnellingVignette** prefab, which is a child of the Main Camera. If you navigate in the scene you'll notice a black border fade in and out as you move. This effect is called VR Vignetting or dynamic field-of-view (FOV) reduction and can reduce motion sickness. It works by reducing peripheral vision, focusing the user's attention to the center of the screen. By reducing peripheral motion the disparity between visual and vestibular inputs decreases, reducing the likelihood of motion sickness.

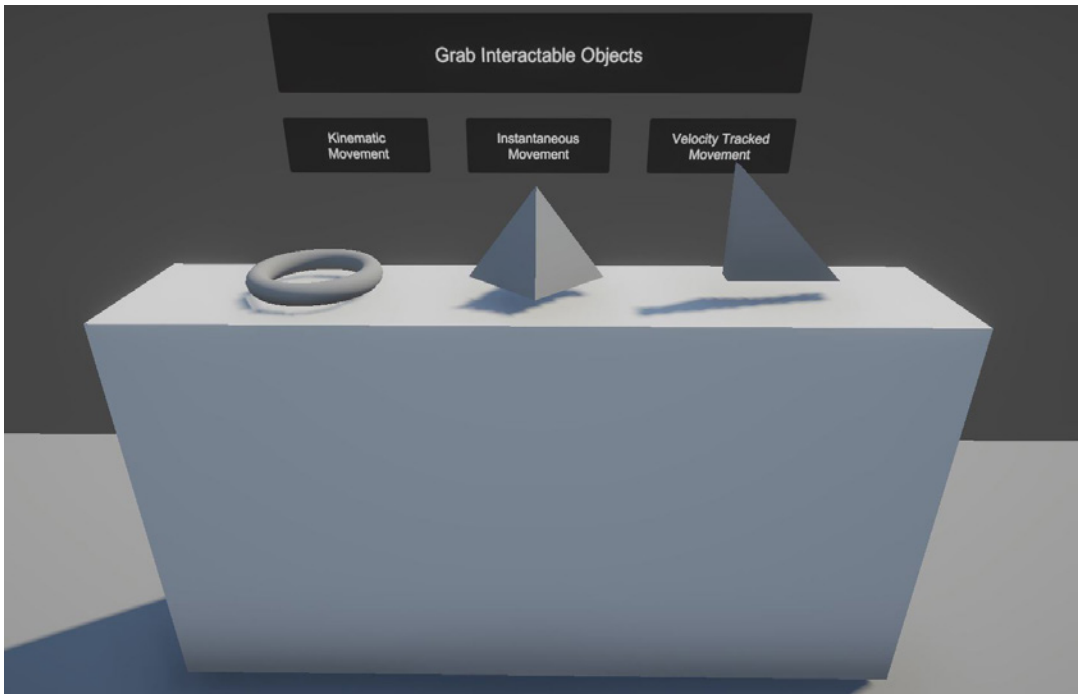


The TunnellingVignette effect in action in the sample scene

Interacting with the VR world

An essential part of creating a VR game is giving the player the ability to interact with the items in the game.

The demo scene showcases three different types of grab interactable that offer different methods of picking up objects. An interactable object requires the **XRGrabInteractable** component.



The three grab interactables in the sample scene.

XRGrabInteractable

The XRGrabInteractable component enables basic grab functionality in VR or AR environments. It allows an object with this component to be grabbed or selected by an interactor (like a VR hand controller) and to follow its movements.

One of the features of the XRGrabInteractable component are its tracking, or **Movement Types**:

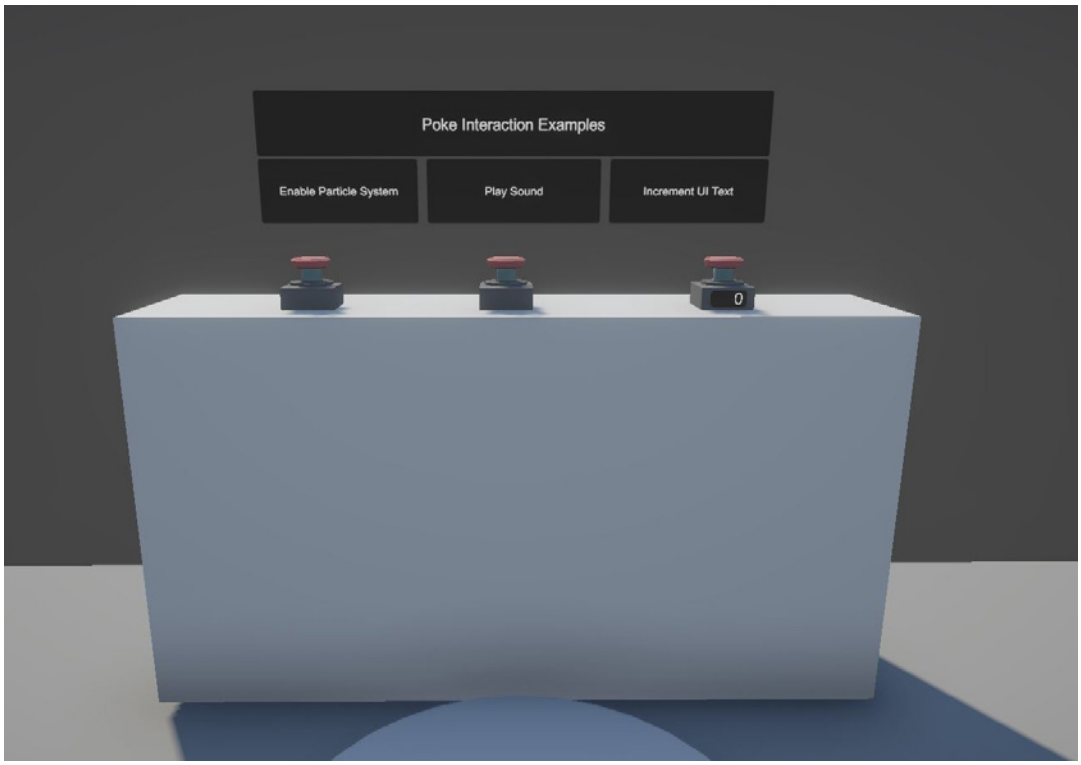
— **Kinematic:** Objects move directly with the interactor, bypassing physics forces and collisions, offering precise control over their position and rotation. This is ideal for predictable and scripted interactions.

— **Instantaneous:** The object's position and rotation are updated immediately to match the interactor's, without physics calculations, ensuring responsive, direct control.

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

Poke interactables

The demo scene also includes a designated area for testing out three different interactions using the poke functionality: Enabling a particle system, playing a sound, and incrementing UI text.



Sample scene with poke interactable types

XR Simple Interactor

Ultimately, anything can be done upon the press of the button, or poke. The root GameObject of the poke interactable contains the **XR Simple Interactable** component. This extends the **XRBaseInteractable** class, inheriting its properties and methods. The **XRSimpleInteractor** class focuses on providing events for various interaction states. It's these events that allow developers to customize the object's response when the poke is detected.

XR Poke Filter

The XR Poke Filter is a component that adds poke interaction to objects in VR and AR environments. It functions by associating itself with an XRBaseInteractable; in the demo scene this is the XRSimpleInteractor. Additionally, it provides functionality to determine if and how an object should respond to being poked based on these custom settings. The class includes various lifecycle methods for initialization, setup, and cleanup, as well as functions for better integration and debugging within the Editor. The XR Poke Filter plays a crucial role in enhancing the interactivity and realism of objects by facilitating nuanced poke interactions.

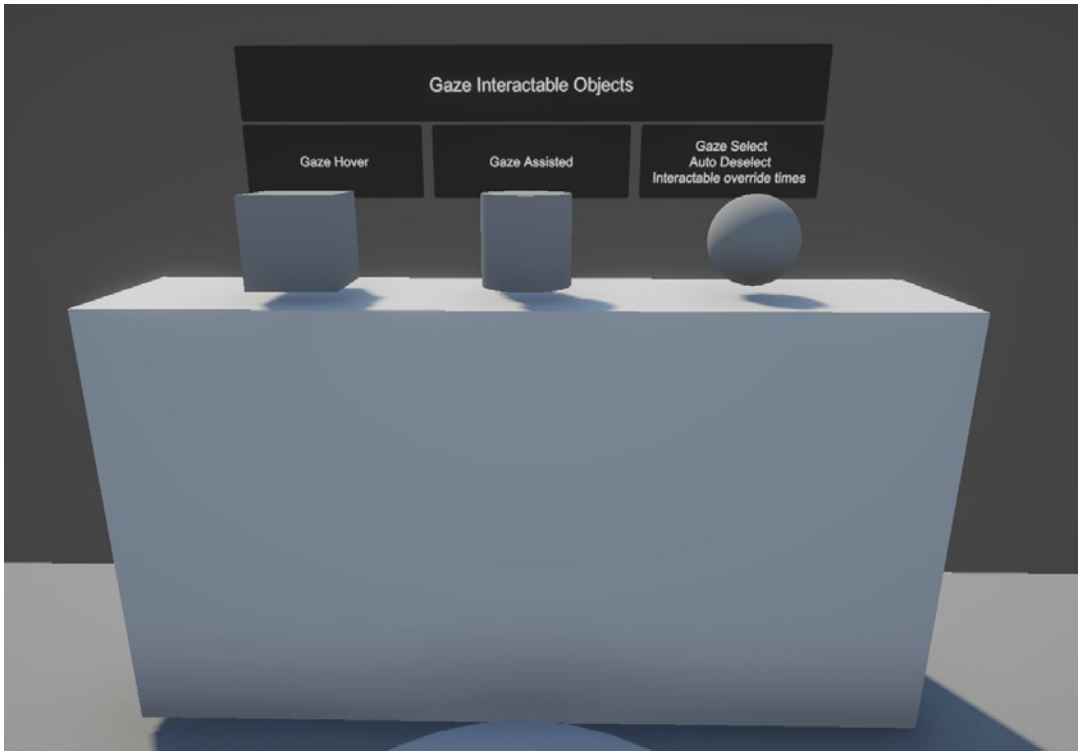
XR Poke Follow Affordance

The XRPokeFollowAffordance class in the XRI Toolkit is used to animate UI elements, like buttons, in VR and AR settings when they are poked or interacted with. Key features include smooth movement, the ability to return to the original position post-interaction, and constraints on movement distance for a realistic interaction experience.

Gaze interactables

A gaze interactable in the context of the XRI Toolkit refers to a type of interaction method where the user can interact with objects in a virtual environment simply by looking at them. This is particularly useful in VR and AR applications where traditional input methods like hand controllers or touch might not be as effective or available, and great for accessibility purposes as it can allow users with limited mobility to engage with VR/AR content.

The sample scene showcases three examples, seen in the image below, of how the gaze functionality can be used in your games.



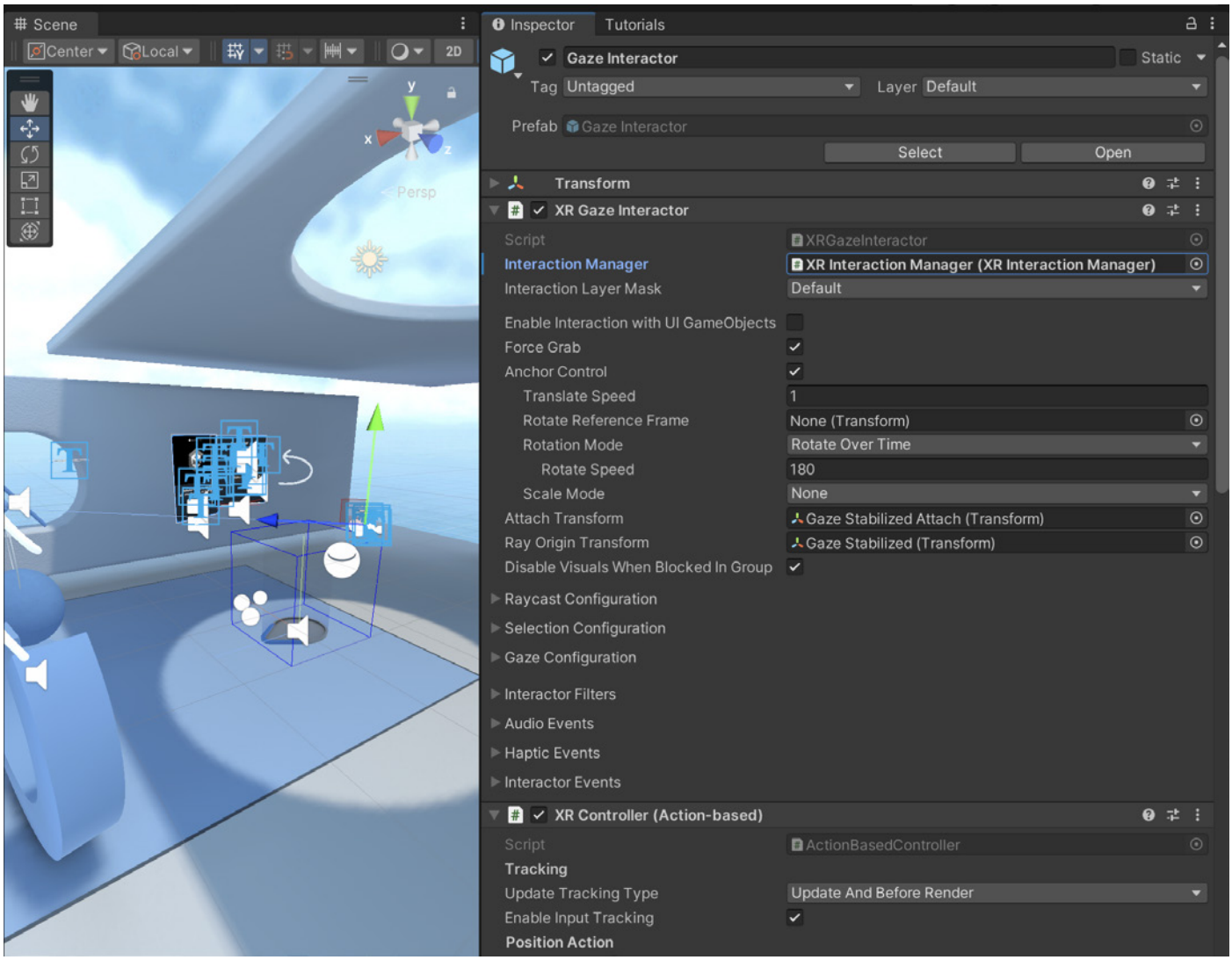
Gaze interactable setup in the demo scene

Clicking on one of these interactables in the scene shows that the functionality is driven by the XRSimpleInteractor component. The events on this component have been used to create the functionality that occurs upon the user's gaze.

The gaze functionality is driven mainly by the three components attached to the Gaze Interactor object within the XR Origin: The XR Gaze Interactor, Gaze Input Manager & the Tracked Pose Driver (Input System)

XR Gaze Interactor

The XR Gaze Interactor extends the XRRayInteractor script to allow interaction in VR with objects through gaze. The script features customizable gaze assistance, allowing for either a fixed size or a collider-based dynamic sizing of the interaction area. It also includes properties for adjusting gaze interaction sensitivity and precision. Automatic creation and scaling of a “snap volume” for gaze assistance are supported, enhancing interaction accuracy. Additionally, the script manages the processing of interactable targets and includes methods to check interaction capabilities with various objects. It also supports overriding default behaviors for gaze interaction, like **Hover to Select** and **Auto Deselect** times, offering a flexible and comprehensive solution for gaze-based VR interactions.



The Gaze Interactor component in the VR Core sample

Gaze Input Manager

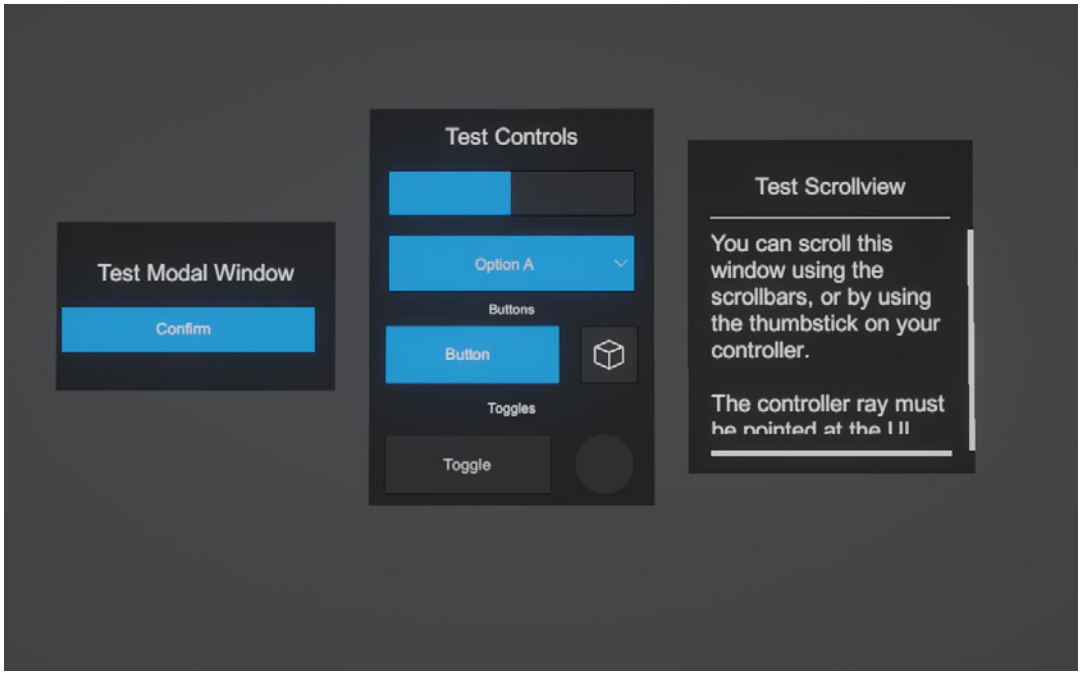
The GazeInputManager script manages input fallback for the XRGazeInteractor when eye tracking is not available. It checks for eye tracking support at startup and logs if an eye tracking device is found. If no such device is detected, it subscribes to device connection events to enable eye tracking if a device is connected later. The script also features a toggle to enable fallback to head tracking in the absence of eye tracking. This ensures continuous functionality by switching to head tracking if eye tracking is unavailable, thereby maintaining interaction capabilities. Overall, GazeInputManager enhances user experience by adapting to the available input methods for gaze interaction.

The XRGazeInteractor and XRSimpleInteractor work together to offer a flexible and robust interaction system in VR, catering to a wide range of scenarios from distant gaze-based selection to close-up manual manipulation.

Interacting with UI Elements

When players visit your XR worlds they will need to interact with a user interface at some stage. The XR Interaction Toolkit example scene demonstrates how UI components react to input from the player's VR controllers.

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.



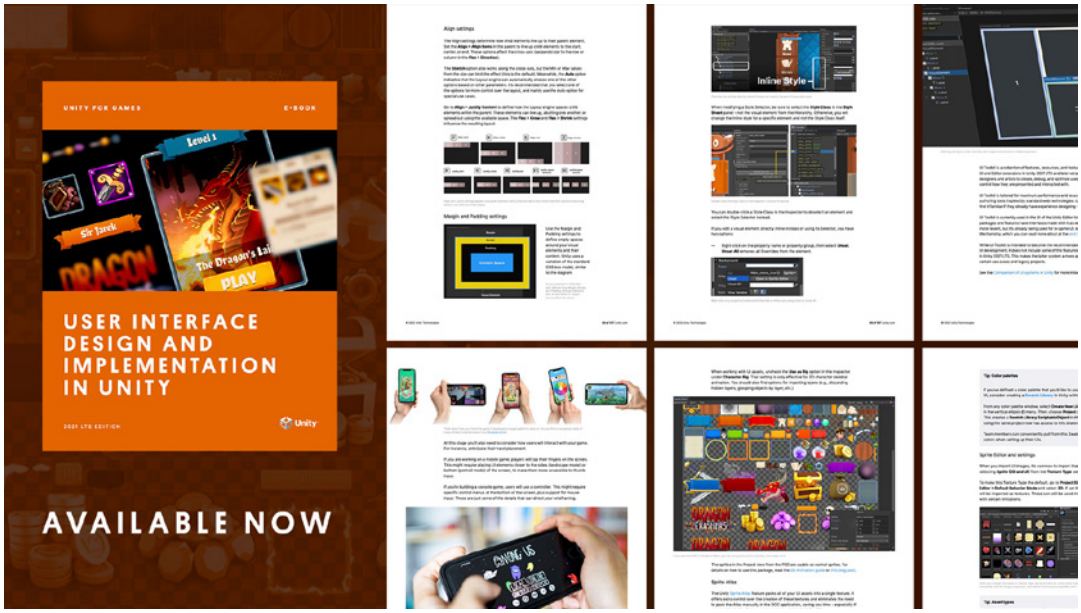
UI elements in the sample scene

XR UI Input Module Component

The **XR UI Input Module** is the component that XRI requires to properly interface with the Event System. It manages interactions between XR environment users and UI elements. It works in concert with the XR Ray Interactor to ensure XRI interactions with the UI are processed correctly. It supports a wide range of input methods, including 3D tracked devices or other non-XR sources like mouse, touch, and gamepad controls if configured to do so. The script defines interfaces for XR interactors to interact with UI elements, handling events like hover and clicks, and is compatible with both Unity's legacy Input Manager and the newer Input System. It acts as a bridge, translating user actions in an XR environment into UI interactions.

Tracked Device Graphic Raycaster

On each of the Canvas objects a **Tracked Device Graphic Raycaster** component is present. This is a custom implementation of the GraphicRaycaster for the Unity XRI Toolkit, specifically designed to handle raycasting in XR environments. It extends the functionality of the standard GraphicRaycaster to work with tracked devices in XR, allowing it to detect interactions with UI elements on a Canvas.



Learn more about UI in Unity with this free e-book and the accompanying sample project.

Download

The XRI Toolkit's example scene offers a valuable starting point for developers venturing into the realm of XR game development. The demo is essentially a sandbox of possibilities, showcasing a wide array of interactive elements, controls, and mechanics that are fundamental to XR experiences.

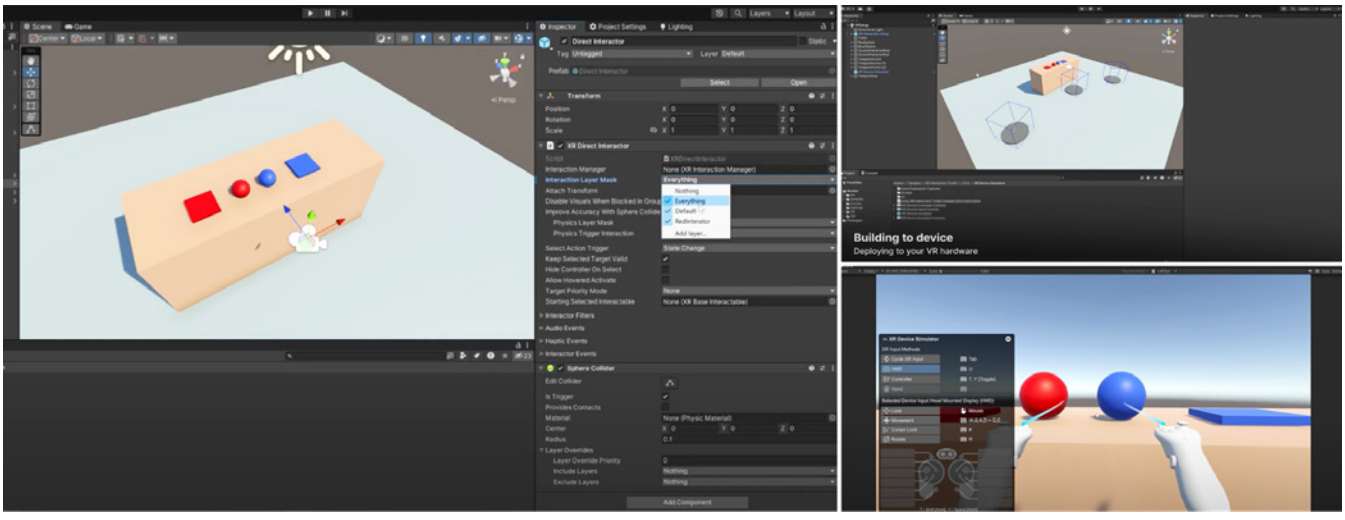
Explore this scene to gain insights into the implementation of various XR features, such as object manipulation, teleportation, UI interaction in 3D space, and controller input management.

The beauty of this scene lies in its modularity and the ease with which you can modify, combine, or expand upon its elements. Whether you're aiming to create an immersive VR game, an educational AR experience, or anything in between, this example scene can be a springboard that helps you realize your vision.

VR development with an example project

This section presents some of the main steps in developing a VR project in Unity. It covers the decision-making driving each development phase, as well as practical steps and workflows in Unity.

If you want to also follow the process of starting a new project step by step you can watch this video.



See this tutorial that accompanies the XR e-book. The video provides a tour through the XR Interaction Toolkit 2.5.x for Unity 2022 LTS.

Watch the tutorial

The big idea

An idea for a game can come to you as quickly as a flash of lightning, or over time, via a more deliberate process. Either way, it's a good idea to anchor your concept in a genre or theme that you're genuinely passionate about.

Early prototyping in Unity with a headset

Keep your approach broad and open-ended in the initial ideation phase. At this point you don't need to immerse yourself in the minutiae; allow that first spark of an idea to breathe and take shape organically. Think of it as sketching the outline of a grand painting – the details are not your focus yet. Instead, concentrate on the overall theme, the emotions you want to evoke, and the core experience you wish to offer your players. This is the time to let your imagination roam freely, unencumbered by the constraints of practicality or technicalities. Every groundbreaking game starts as a mere flicker in the mind's eye – and this is your moment to kindle that flicker into a flame.

The example idea that we'll develop in this section is for a horror escape room game that requires the player to solve clues in order to escape. That's the high-level view. Now let's dig a little deeper.

Plan ahead

With the initial idea ready, the next stage involves transforming this broad concept into a structured plan. This critical phase lays the groundwork for the development process. Here are its key steps:

1. **Define the core gameplay mechanics:** Start by detailing the fundamental mechanics of the game. How will the player interact with the VR environment? What are the main actions the player will perform? In our example horror game, this might include exploring eerie environments, solving puzzles, and evading or confronting threats. You also want to start thinking about how these mechanics will be implemented in Unity. For example, the XRI Toolkit provides functionality that can help shape puzzles, such as the XR Socket Interactor.
2. **Create a basic storyline or theme:** Even if your game is not narrative-heavy, having a basic storyline or theme can guide the development. For a horror game, this could involve deciding on the setting (like a haunted house, abandoned asylum or creepy crypt), the background story, and any key characters or enemies.
3. **Sketch out level designs and environments**: Begin with visualizing the game's environments. Sketch out rough layouts of levels or key areas. Consider how the player will navigate these spaces and how the layout can contribute to the game's atmosphere.
4. **Identify the unique selling point (USP):** What will make your game different from others in its genre? Will it be a unique gameplay mechanic, compelling story, innovative use of VR technology, or a particular style of art and sound design? The USP(s) are key to attracting and retaining players.
5. **Develop a prototype:** Start with a simple prototype to test your core gameplay mechanics. This doesn't need to be complex; it's about getting a feel for how the game plays and making adjustments early on. Prototyping in VR is crucial as it gives you a sense of scale, distance, and interaction that's hard to gauge on a regular screen.
6. **Gather feedback early:** Show your prototype to friends, fellow developers, or potential players. Early feedback can be invaluable in identifying strengths and weaknesses in your game concept and mechanics.
7. **Plan the development process**: After these previous steps, you should be ready to outline the major milestones for the project. This can include detailed game design, art and sound production, programming, testing, and polishing. If you're a solo developer or small team, try to set realistic timelines and priorities.
8. **Consider the technical requirements:** VR/AR development has specific technical considerations, like ensuring a stable frame rate to prevent motion sickness, optimizing for different VR hardware, and implementing intuitive VR controls.
9. **Document your design**: Create a detailed game design document (GDD). This should include everything from game mechanics, story, level design, art and sound direction, user interface design, and technical requirements.
10. **Budget and resources planning:** Assess the resources you have and the ones you'll need, including software, hardware, and additional work from freelancers.

Work through these steps to help you turn your initial idea into an executable plan to develop your final game. You can pick up many more practical tips and ideas about how to plan for and develop your game in these two e-books from Unity:

Download

Download

Let's return to the horror game example. After working through the above steps, we now know that we're aiming to create a 5-10 minute VR experience that puts the player in a crypt. The player must solve three puzzles to get keys that will open a coffin in the center of the room. They can escape the crypt via the coffin. To add some fear and stress the ceiling will be slowly lowered, with protruding spikes. So the player will have to act fast.

Fire up the engine

Unity version and URP

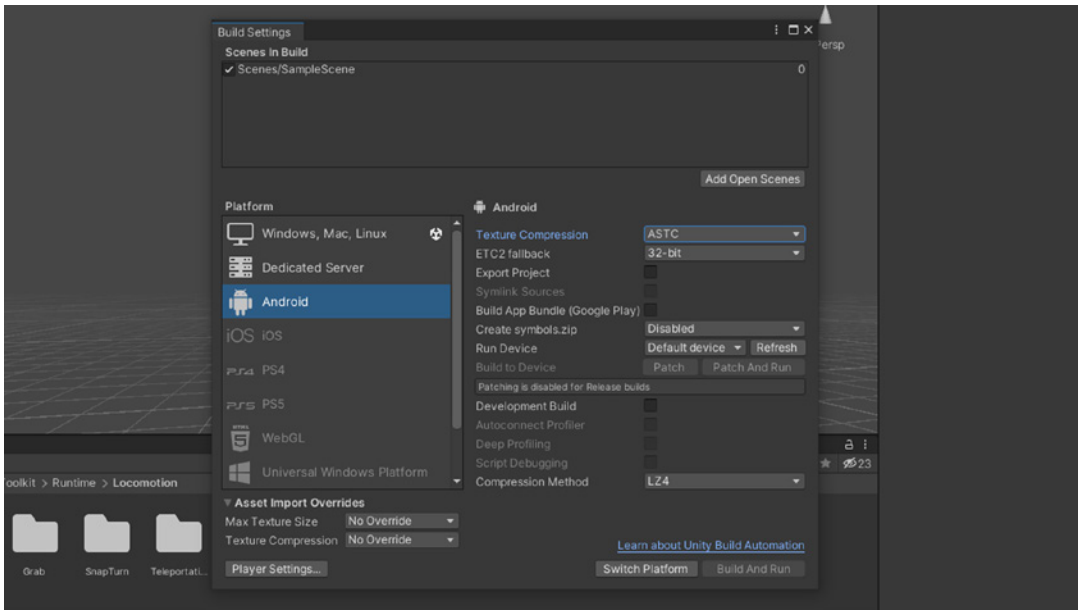
It's time to start creating your game in Unity. In this example we use Unity 2022 LTS and URP as it's the most performant render pipeline for XR applications. The target hardware is a Meta Quest 2 headset.

Install the XR Plug-in Management package and select OpenXR for all platforms.

Next, assign an interaction profile; we'll add the **Meta Quest Touch Pro Controller Profile** in this example since it uses the Meta Quest 2 headset.

Build Settings

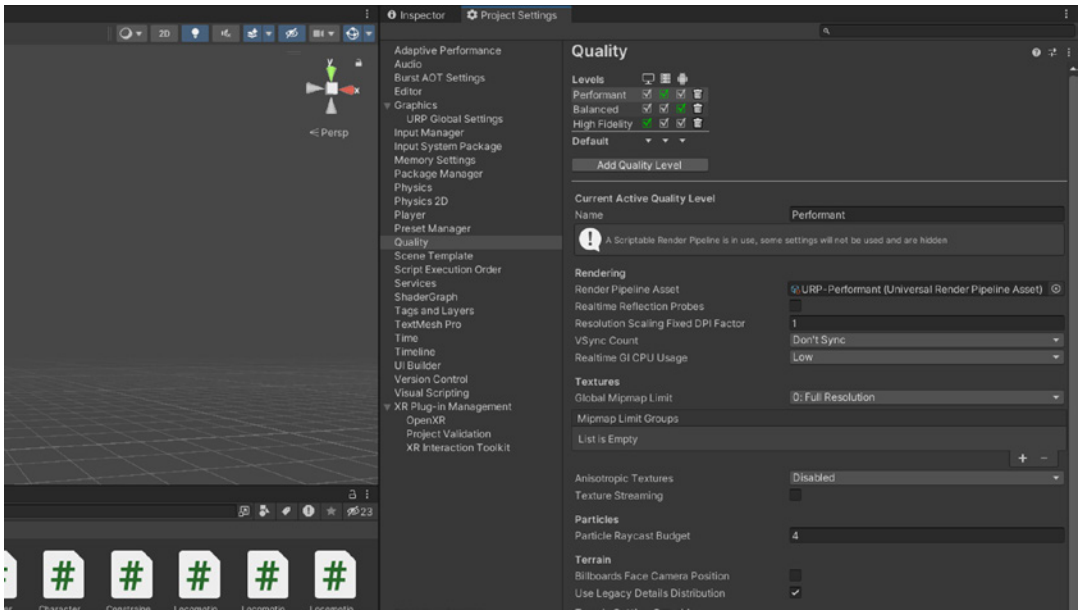
In the Build Settings, we set the target to Android, and change the texture compression to ASTC (Adaptive Scalable Texture Compression), which is an efficient method well-suited for VR applications. This will help to balance the high-quality texture demands of VR with the performance constraints of the devices.



Build Settings for the example horror VR project

Quality Settings

The last setup step is to adjust the Quality Settings via the Project Settings window. Choose **Performant** as the default.



Choosing Performant as the default in Quality Settings

The Performant URP Asset is set up to optimize rendering performance. It achieves this by reducing draw calls, optimizing shader execution, and offering better control over lighting and shadow rendering. You can create visually rich applications while maintaining high performance on target platforms, a critical factor in maintaining immersion and preventing motion sickness in VR. As URP is a scriptable render pipeline, you can also tailor the rendering process to suit the specific needs of your project.

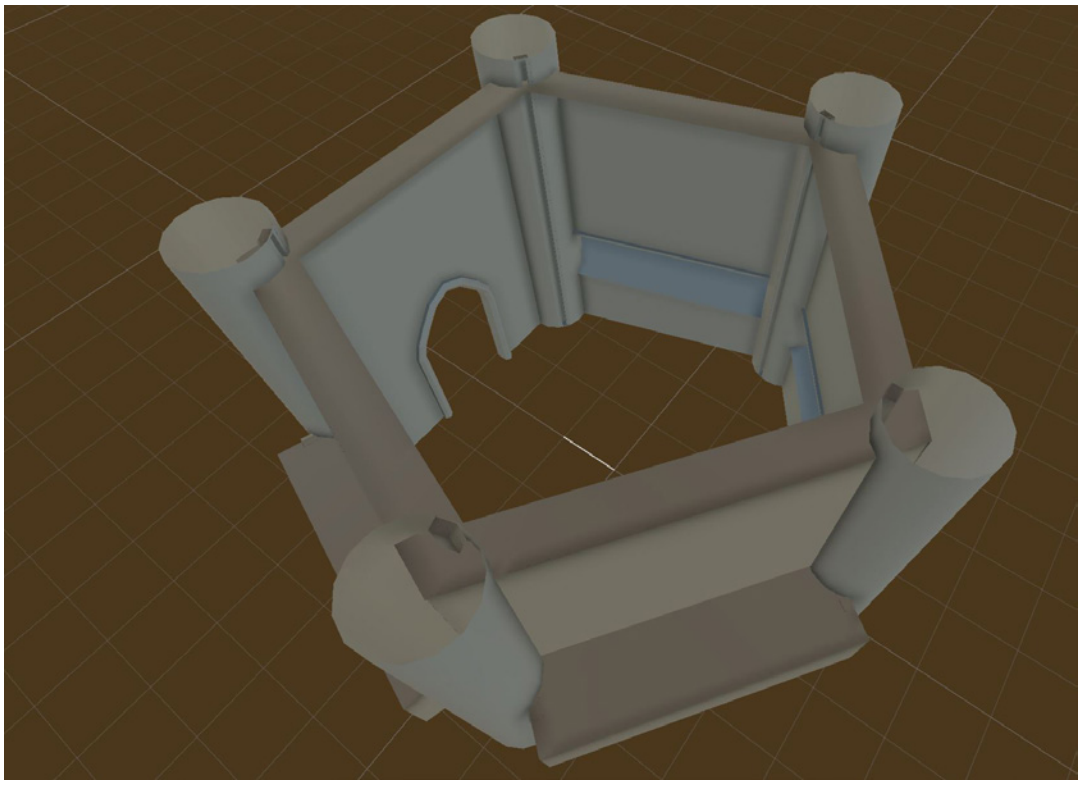
Overall, an important goal to aim for is to create a great-looking game that runs well on the lowest-quality device you're targeting – this provides a strong foundation to build upon later.

Construct the world

Prototype and modeling

In this phase your art pipeline should kick in with the prototyping of your 3D environment. The example horror game is fairly simple because it will take part in a single environment, the crypt.

We'll start by creating a 3D blockout to check the scale of the space and see where to place the puzzles. These are simple shapes that form the volume and allow us to start thinking about the mechanics of the descending ceiling.



Block out for the environment

A pentagon shape is chosen for the environment, with a small tunnel leading away from the crypt that will be the escape route for the player.

Efficient prototyping is possible in Unity with the ProBuilder package. You can create within the environment you are going to be playing in, giving you maximum creative flexibility and efficient iteration cycles. This is useful for VR applications as it's important to keep control of scale within the scene. Being able to mock up an idea using ProBuilder and then pop on the headset to test it out makes development faster.

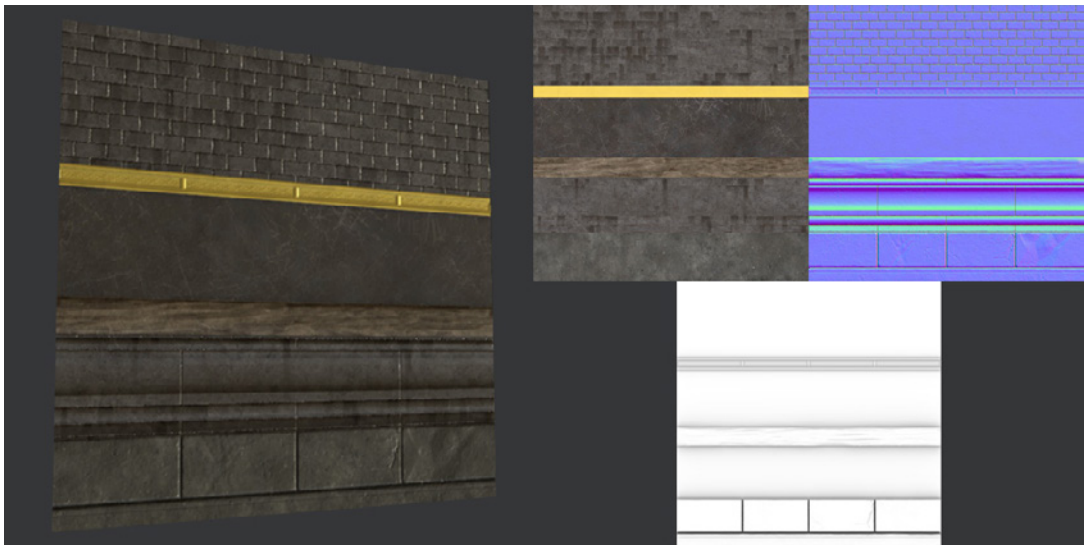
The level design e-book linked to earlier has an extensive section on using ProBuilder tools, and the following video tutorials also provide helpful tips:

Watch the tutorial

Watch the tutorial

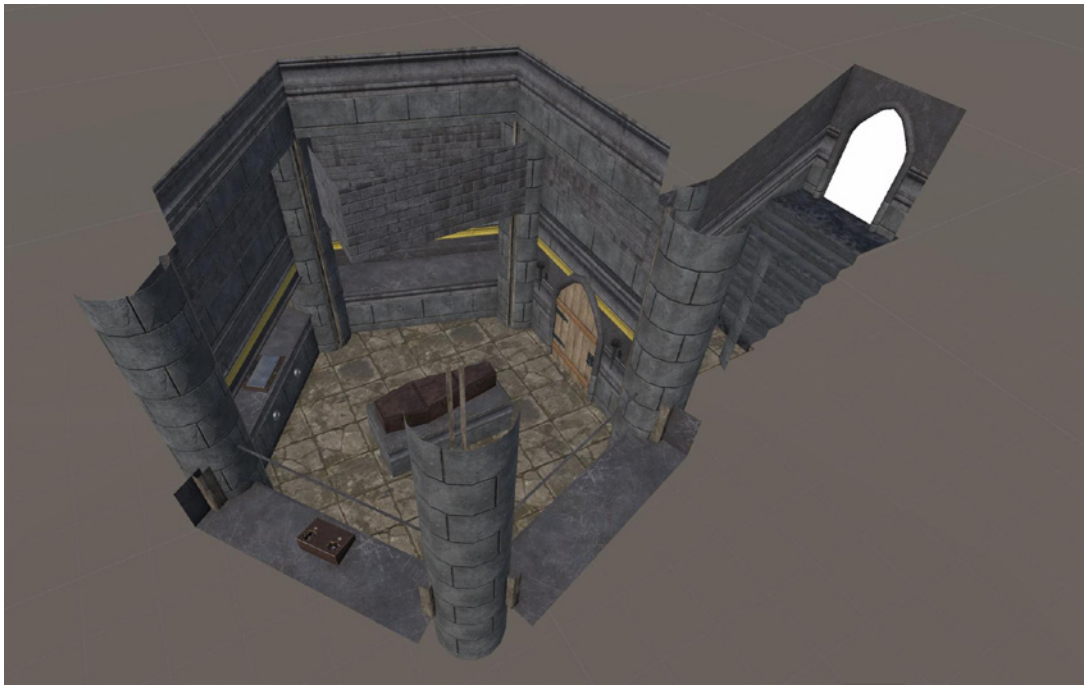
Efficient texturing

From the blockout we can now start to model and add textures. In the example horror game there is a lot of repetition, which helps us to create a small kit of models consisting of a pillar, wall, and doorway. These can be duplicated to form the environment; we can also attach them together to create one overall model so that a trim sheet can be used to texture the entire structure, as seen in the image below:



Trim sheet for the environment

Areas of the trim sheet are applied to areas of the model with the resulting textured crypt seen in the following image:



The textured 3D environment

If you look carefully you can see how areas of the structure are allocated to the trim sheet.

In the example project, we continue to make all the props, the puzzles, and the exit, bring them all into Unity, and assign shaders (in this case it's the default URP Lit shader). Now let's look at how to light this environment

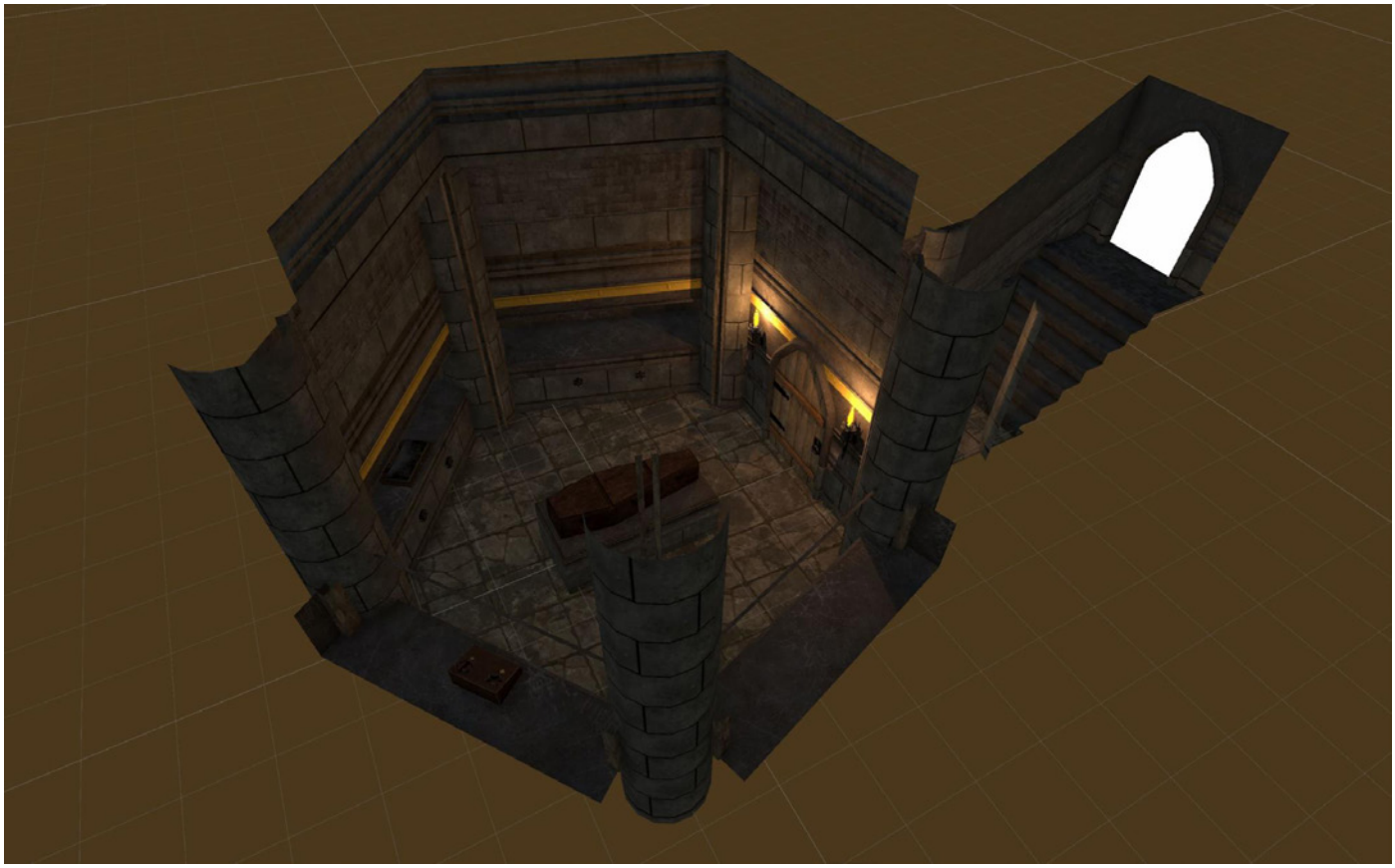
Let there be light

How to choose a lighting method?

In the crypt, the light sources are from a skylight and two torches on either side of the door. The torch lights are static, making this a good scene for baked lighting (where lighting information is precalculated and baked into texture maps).

However, items that move, such as the door and the coffin lids, will not cast a shadow in baked lighting; we can try mixed lighting, where baked lighting is calculated for static objects and dynamic objects receive real-time lighting. Ultimately though, we'll opt for baked lighting because players probably won't notice that the coffin lids lack a shadow when they open. It's these types of trade-offs that experienced developers will be familiar with from their real-world projects.

Below is an image of the scene with the baked lighting.



The result of baking the lighting

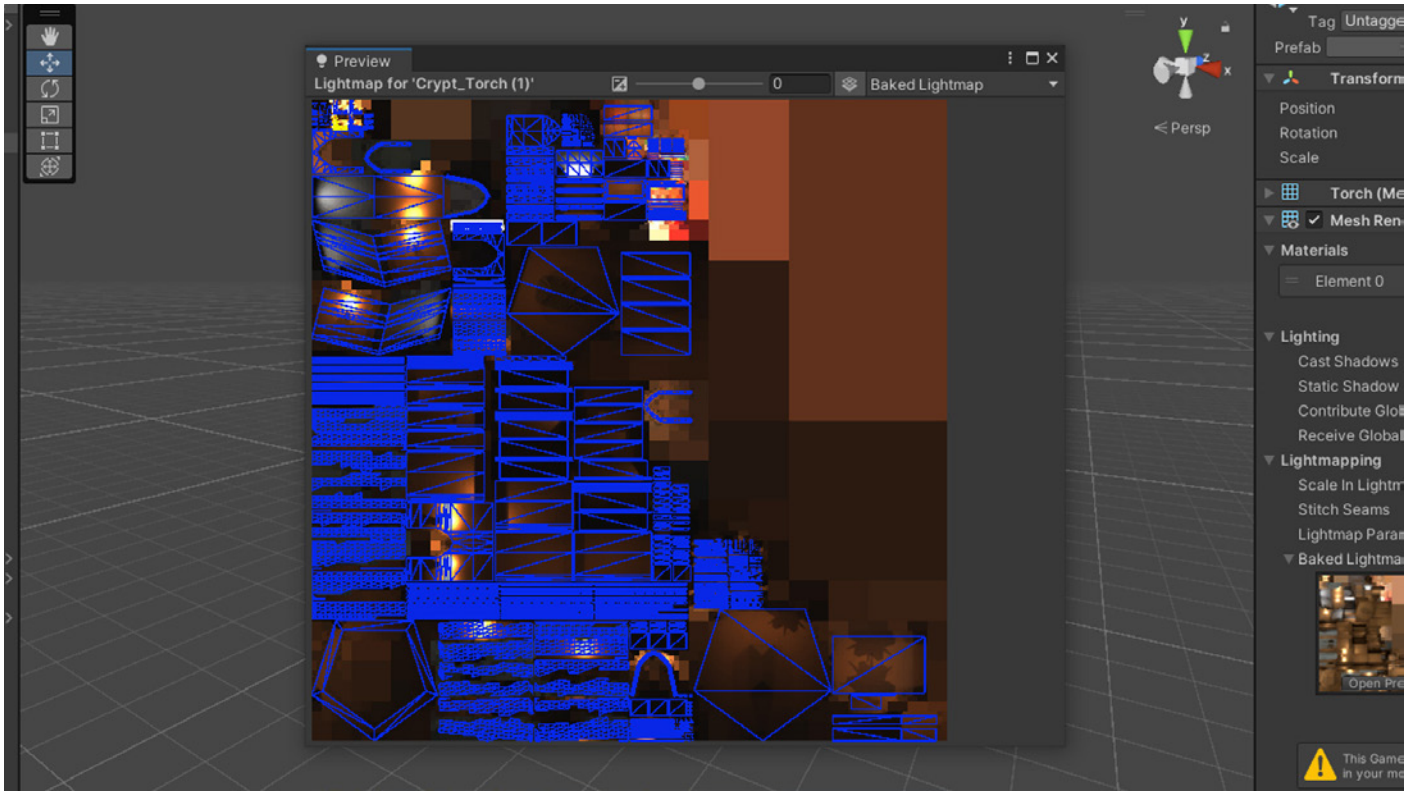
The scene is lit by one 512×512 lightmap. Getting the right settings for your scene is a balance between performance and quality.

Optimizing the lighting

Our aim with a small scene like the crypt is one lightmap. The original size of the lightmap was 1024, which looked great. To test, we reduce it to 512, which results in three lightmaps, adding to the draw calls. A draw call is a command to the GPU to render a batch of geometry with a certain set of settings, like materials and textures. Each draw call incurs a certain amount of processing overhead, so reducing the number of draw calls can improve performance, especially on lower-end hardware.

We can get everything onto one 512 lightmap by reducing the texel density. This refers to the ratio of pixels in a lightmap to the actual physical space they represent in the 3D world. It's a measure of how many lightmap pixels (texels) are used to represent a unit area of a surface. Higher texel density means more texels are used per unit area, resulting in greater detail and higher resolution in the lighting and shadow effects on that surface. Conversely, lower texel density uses fewer texels per unit area, leading to less detailed lighting and shadows.

Let's change the texel density value from 40 to 20 and bake again. This time we get the single 512×512 lightmap.



Baked lightmap data for the crypt environment

The quality of the lightmap is good; when we test in the headset there is minimal impact to the overall visual quality. We'll leave the lighting as is for now and can increase quality later on if the game is super performant.

Read this blog for troubleshooting lighting challenges in a Unity project.

Now let's look at how the player is going to navigate this space.

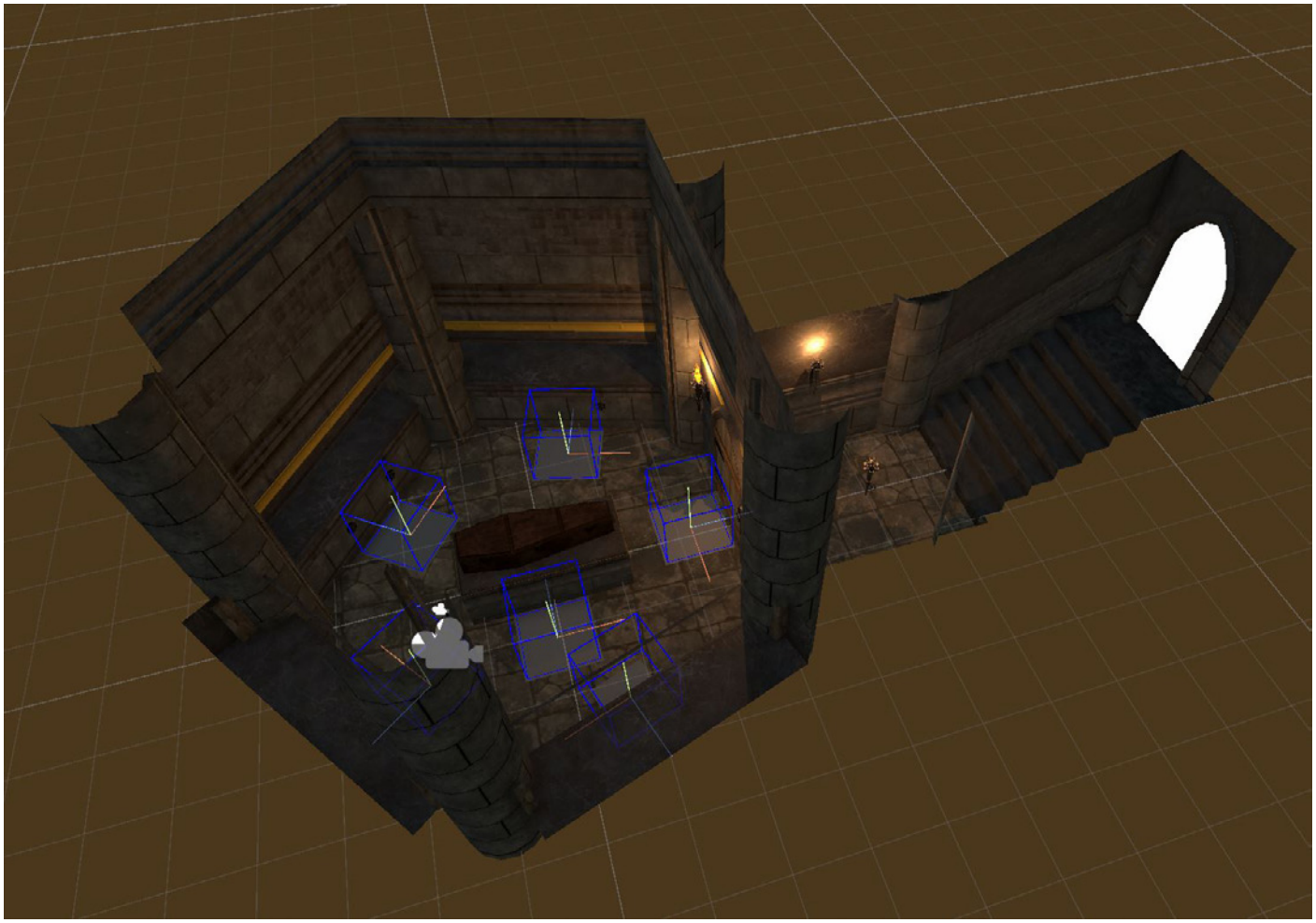
Navigate the virtual world

With the scene lit we can now add the functionality so the player can move around it. We'll add the same prefab that's used in the demo scene for the XR Rig. Then we add an empty GameObject and an XR interaction Manager component to it.

Teleportation Type

Should players freely teleport around or use the teleport anchors? In the crypt environment, the teleport anchors will allow us to identify the important areas of the scene, acting as a type of hint to the player – if they are able to teleport to a location then it must be of some importance. This is a good solution due to the timed nature of the game.

Let’s place the teleport anchors at the puzzle locations, the coffin, and the door.



The location of the teleport anchors

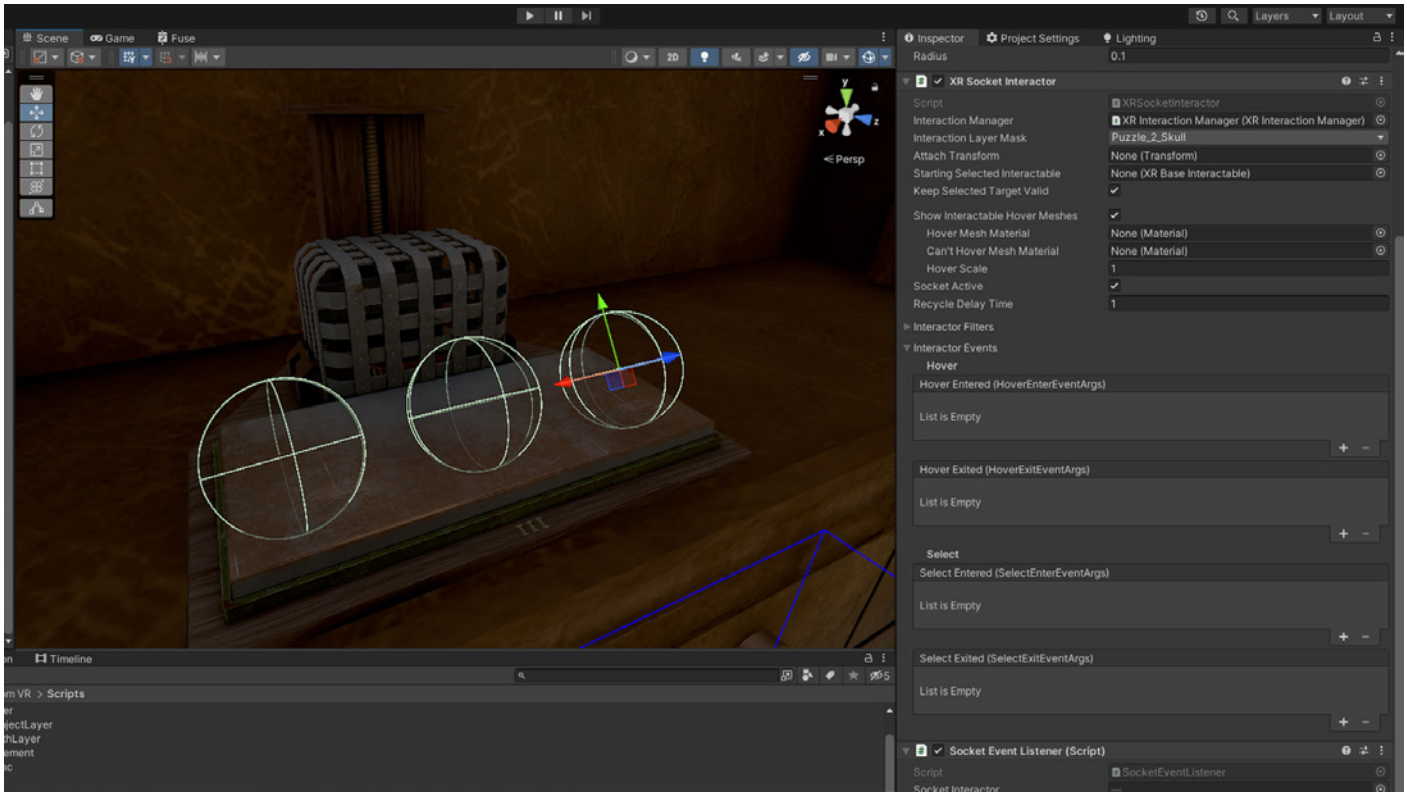
The teleport anchors let you customize the direction the player faces when they teleport. In our example game, the player should be facing the object that’s important at their specific location, like a puzzle, when they teleport. To achieve this the teleportation configuration should be set to **Target Up And Forward** so the player will match the orientation of the teleportation anchor.

The style of the components can be easily customized. For example, to change the ray style of the teleporter, you can select the **Teleport Interactor** in the XR Origin’s Right Controller. The XR Ray Interactor’s **Raycast Configuration** allows you to change the line type, and the **XR Interactor Line Visual** will give you control of the color.

Create puzzles using XR Interaction Toolkit

In the scene below from the horror game example, players are presented with two engaging puzzles, each offering a key as a reward upon completion. These keys are essential for unlocking the coffin situated at the center of the room. Both puzzles share a similar interactive design, challenging players to explore the room and discover specific items that can be used to solve the puzzles.

The first puzzle involves finding a concealed item in the room. Once located, this item enables the player to unlock a drawer, revealing one of the keys. All these interactive elements are seamlessly integrated and managed using the XR Interaction Toolkit, ensuring a smooth and immersive experience for the player.



One of the puzzles from the horror game example that utilizes the Socket Interactor

The functionality we can use for this puzzle is a combination of the XR Grab Interactor and the XR Socket Interactor. For example the puzzle has three socket interactors that require an object to be placed inside.

The XR Socket Interactor, which is looked at in more detail below, has an Interaction Layer Mask that allows you to specify a layer that the socket interactor will interact with. When a valid object is placed into the socket a custom script listens for events coming from the XR Socket Interactor.

```
using UnityEngine;
using UnityEngine.XR.Interaction.Toolkit;

Unity Script | 0 references
public class SocketEventListener : MonoBehaviour
{
    [SerializeField] private XRSocketInteractor socketInteractor;
    public Puzzle linkedPuzzle;

    Unity Message | 0 references
    private void OnEnable()
    {
        socketInteractor.selectEntered.AddListener(SocketActivated);
        socketInteractor.hoverEntered.AddListener(SocketHovered);
        socketInteractor.hoverExited.AddListener(SocketHoverExit);
    }

    Unity Message | 0 references
    private void OnDisable()
    {
        socketInteractor.selectEntered.RemoveListener(SocketActivated);
        socketInteractor.hoverEntered.RemoveListener(SocketHovered);
        socketInteractor.hoverExited.RemoveListener(SocketHoverExit);
    }

    2 references
    private void SocketActivated(SelectEnterEventArgs arg0)
    {
        linkedPuzzle.PuzzlePieceFound();
        arg0.interactableObject.transform.GetComponent<Collider>().enabled = false;
    }

    2 references
    private void SocketHovered(HoverEnterEventArgs arg0) => arg0.interactableObject.transform.GetComponent<ResetGrabbedItem>().enabled = false;

    2 references
    private void SocketHoverExit(HoverExitEventArgs arg0) => arg0.interactableObject.transform.GetComponent<ResetGrabbedItem>().enabled = true;
}
```

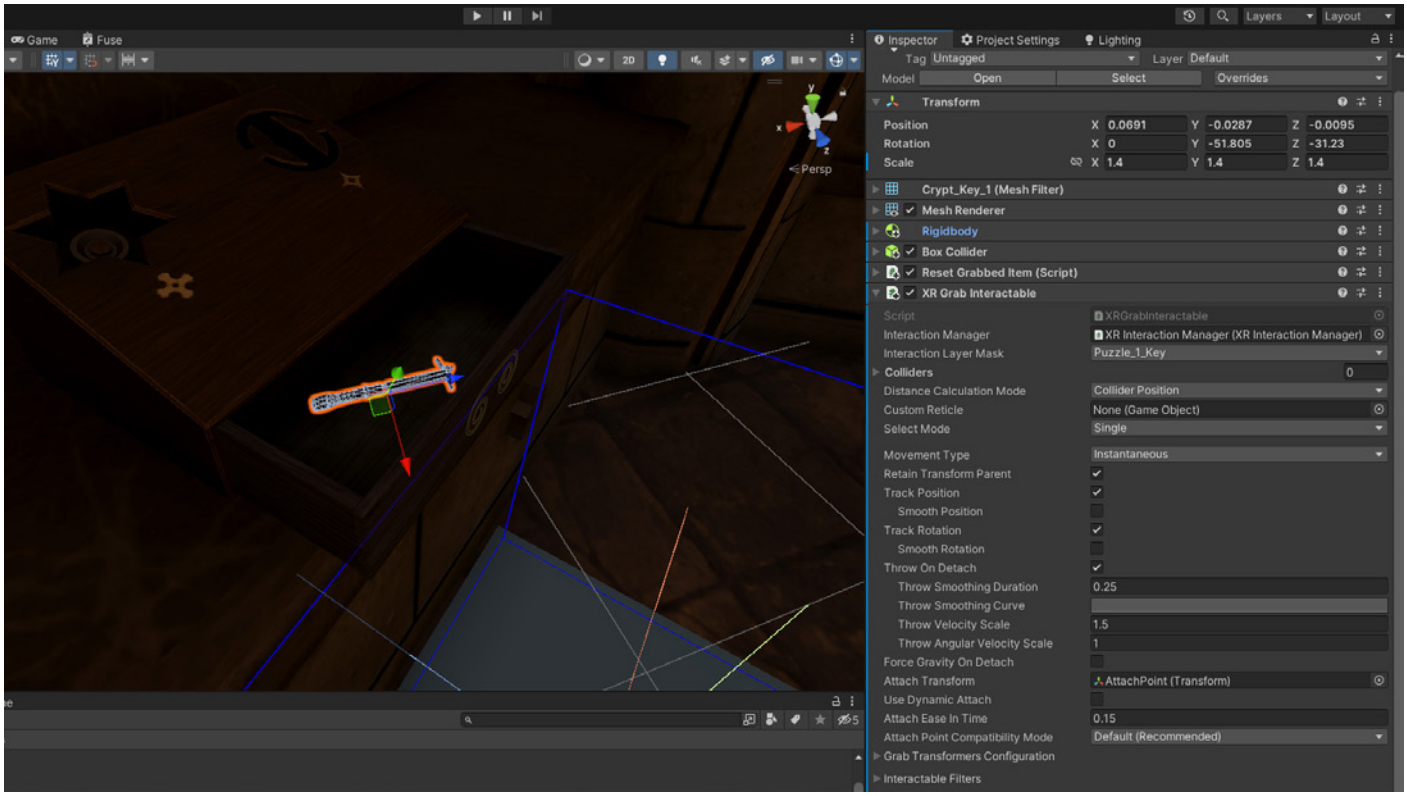
Custom script that listens to the Socket Interactor

By leveraging the core components of the XR Interaction Toolkit, together with a few simple custom classes to control gameplay, you can create intricate immersive interactions.

Let’s look more closely at some of the XR Interaction Toolkit functionality used in our horror game example.

XR Grab Interactor

The Grab Interactor was introduced in the Interacting With The VR World section of this e-book. Each GameObject for this puzzle that will be picked up by the user will be assigned the XR Grab Interactor component.



A grabbable object with the XR Grab Interactable

XR Socket Interactor

The **XRSocketInteractor** class is a sophisticated tool designed for handling socket-based interactions in XR environments. It offers a variety of features, including the ability to define how interactable objects scale when attached to a socket, with modes like No Scaling, Fixed Scaling, or Dynamic Scaling.

The class enhances user experience by displaying hover meshes to indicate potential interactions and provides an option for snapping objects to the socket. It includes settings for hover mesh materials, both for available and unavailable interactions, and allows for the activation and deactivation of socket interactions. The script manages collisions and target processing, ensuring proper handling of valid interaction targets and collision events. Advanced customization options are available for socket transformations and interactable processing, including adjustments for socket snapping radius and interactable transformations. Additionally, the XRSocketInteractor handles various Unity lifecycle events, ensuring robust and efficient management of interactables within XR environments.

We can leverage this component by using them to place the grabbed objects into the puzzle.

Create a Socket Interactor

The first puzzle requires two objects to be placed into specific areas. We create an empty GameObject and add to it the XR Socket Interactor component. A Sphere Collider is added and marked as a trigger. We need this Socket Interactor to only accept one specific object; in order to achieve that functionality we need to specify a layer using the **Interaction Layer Mask** dropdown. Creating a new layer here and assigning it will ensure the Socket Interactor is now limited to this specific layer. We create a layer called **PuzzlePeice1** using the dropdown and assign it.

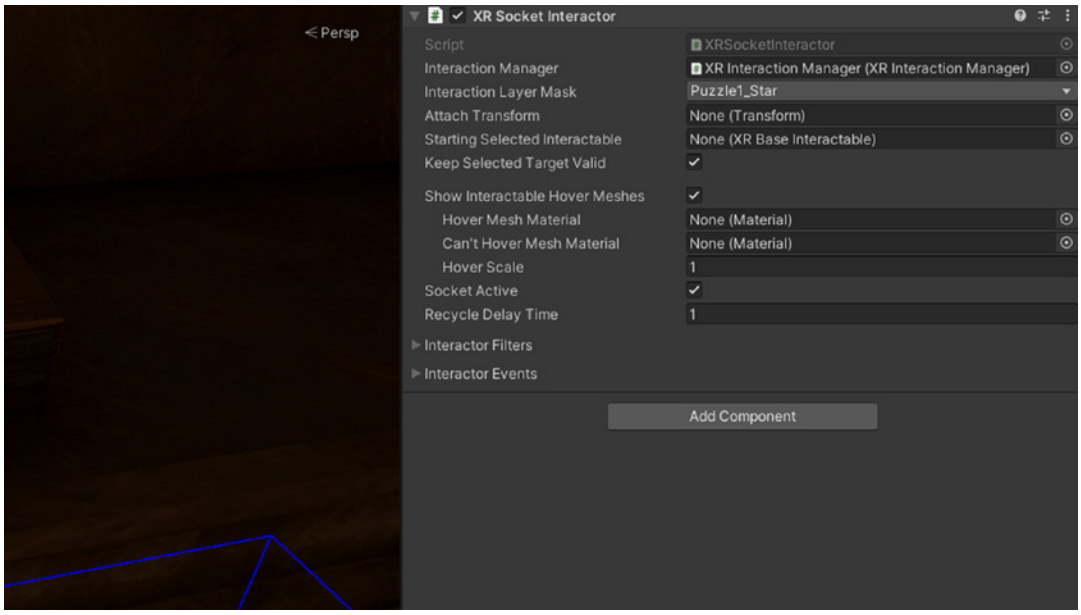
A Socket Interactor showing the correct object highlight mesh.

The next step is to locate the puzzle piece that fits in this Socket Interactor and change the XR Grab Interactable's Interaction Layer Mask to match the Socket Interactor. Now only this specific grabbable object will work with the socket interactor.

Below is the puzzle with the Socket Interactor in place.

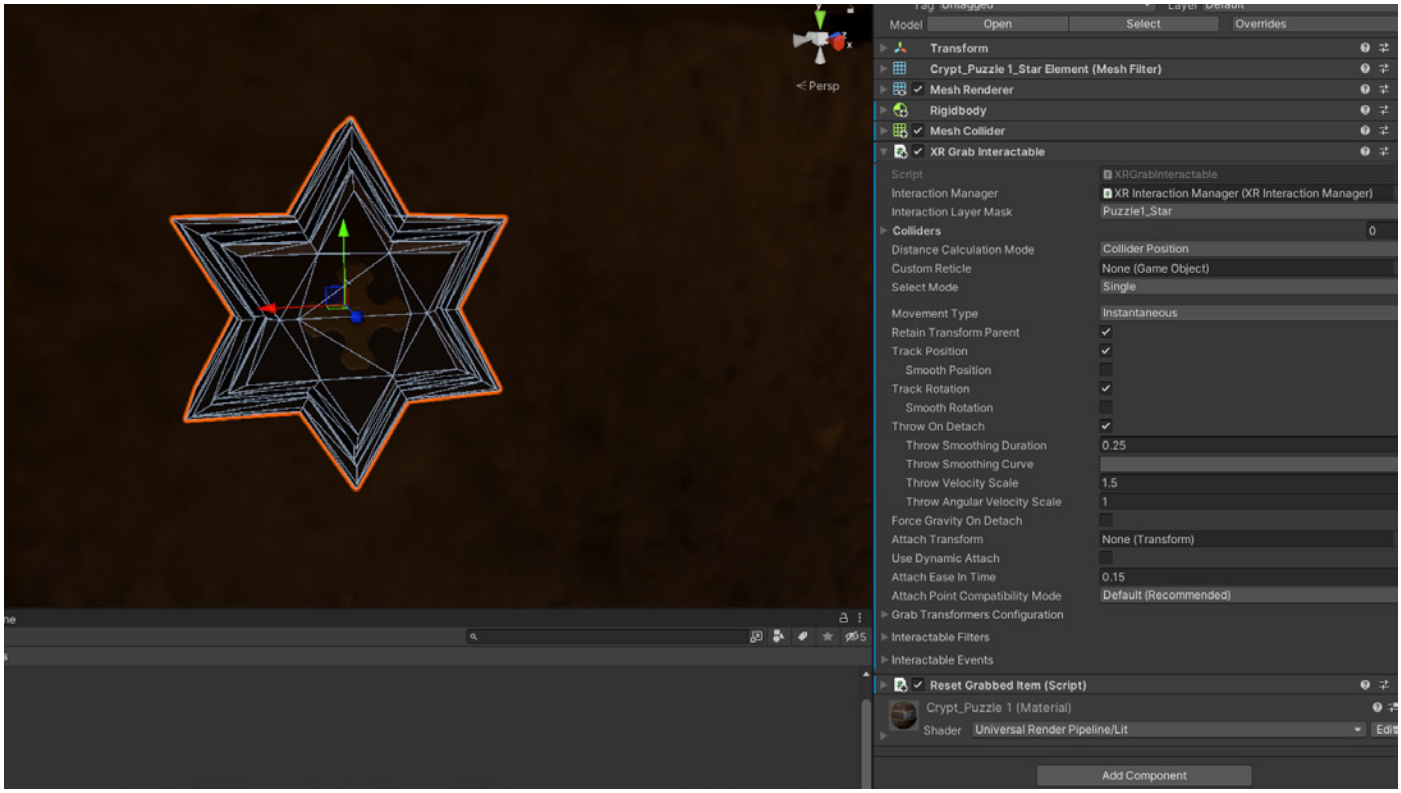
Socket Interactor in place

The Socket Interactor is assigned a layer to accept:



Setting the Layer Mask

The XR Grab Interactor GameObject that is part of the puzzle also has its Interaction Mask set to the same layer to match the Socket Interactor.

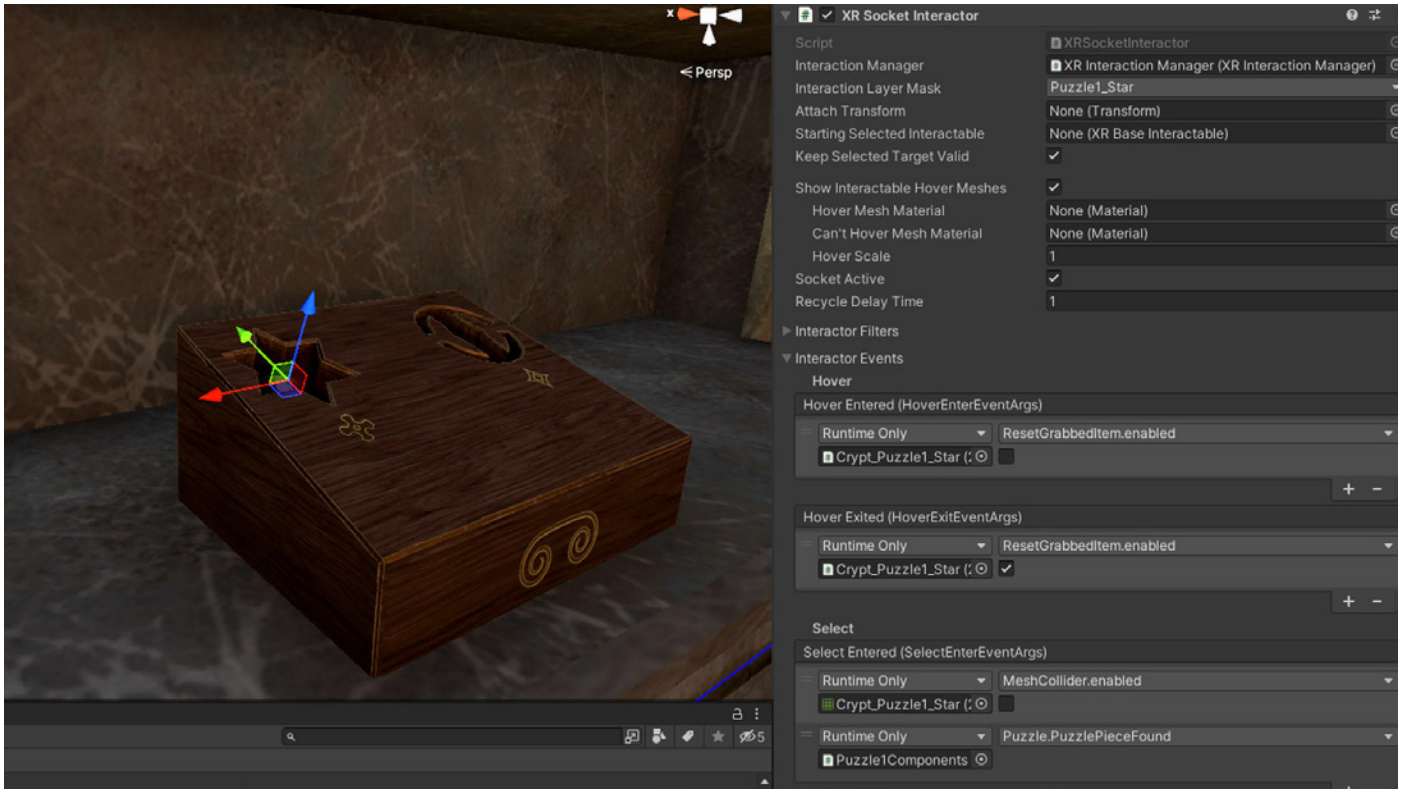


The Grabbable object has its interaction layer set to the same as the Socket Interactor.

We can further customize the XR Socket Interactor to display a mesh when we hover into the trigger, along with colors that display a valid or invalid object.

Using the Socket Interactors in this way allows you to quickly build a puzzle system for a game, and leveraging the events of the Socket Interactor allows you to customize its functionality.

For the horror game example, we can write a simple C# script that is a component of the puzzle. This script knows how many steps are required to complete the puzzle, as an int. When the player places the correct object in the socket the **SelectEntered** event gets called by the Socket Interactor. Using this event we can tell our puzzle script that a piece has been found.



Events on the Socket Interactor being used to drive interactions

A **puzzlePieceCounter** is incremented and then checked to see if we have reached the required amount. When the required pieces have been found an event is called and the puzzle is over. The following code snippet is what controls this simple interaction.

```
using UnityEngine;
using UnityEngine.Events;

Unity Script | 1 reference
public class Puzzle : MonoBehaviour
{
    //how many peices does this puzzle require
    [SerializeField] private int requiredAmountOfPuzzlePieces = 0;

    //keep track of found peices
    private int puzzlePieceCounter = 0;

    //Event for puzzle completion
    public UnityEvent OnPuzzleCompleted;

    //Track if the puzzle is completed
    private bool isPuzzleCompleted = false;


    //accessed by the socket interactors - 'SelectEntered' event
    1 reference
    public void PuzzlePieceFound()
    {
        puzzlePieceCounter++;

        CheckPuzzleCompletion();
    }

    //check to see if the user completed the puzzle!
    1 reference
    private void CheckPuzzleCompletion()
    {
        if(isPuzzleCompleted)
            return;

        if(puzzlePieceCounter == requiredAmountOfPuzzlePieces)
        {
            OnPuzzleCompleted?.Invoke();
            isPuzzleCompleted = true;
            Debug.Log("Puzzle is completed");
        }
    }
}
```

Code for controlling the puzzle

By combining the functionality of the XRI Toolkit with a couple of scripts to keep track of the completed interactions, we can efficiently create a puzzle for our game. The XRI Toolkit, the Grab and Socket Interactors, and a simple layer of logic to connect results in an immersive interaction for the player. They need to navigate in the room to find the pieces they need for the puzzle, get them into position, and then grab the key as a reward – all while the spiked ceiling is coming down.

The rest of the game can use the components available in the XRI Toolkit to help speed up development. Once all the puzzles are in it's time to look at the Unity Profiler and see how the game is performing.

Testing and iteration

Testing is a critical step in the game development process for many reasons:

- Bug identification and fixing
- Gameplay balance
- User experience and usability
- Performance optimization
- Compatibility testing
- Feedback
- Compliance
- Market preparation

Unity offers a suite of profiling and testing tools including the Unity Profiler, Memory Profiler and Profiler Analyzer. This Unity Discussions post provides a helpful overview of advanced learning resources for these profiling tools, including our dedicated e-book, *Ultimate guide to profiling Unity games*.

In addition to the profiling tools there is the Unity Test Framework for creating, managing, and running automated tests on your game. You can find many advanced testing, debugging, QA, and profiling knowledge in the Unity best practices hub, including these articles:

— Tools for profiling and debugging

— Speed up and improve QA testing with Unity's debug class

— How to debug game code with Roslyn Analyzers

— Testing and quality assurance tips for Unity projects

And see the following video tutorials for profiling tips:

Watch it

Watch it

Watch it

The Unity Asset Store also provides third-party testing tools that can be integrated into your workflow. Incorporating some of these practices and utilizing Unitys tools can significantly enhance the quality and success of your game.

XR Device Simulator

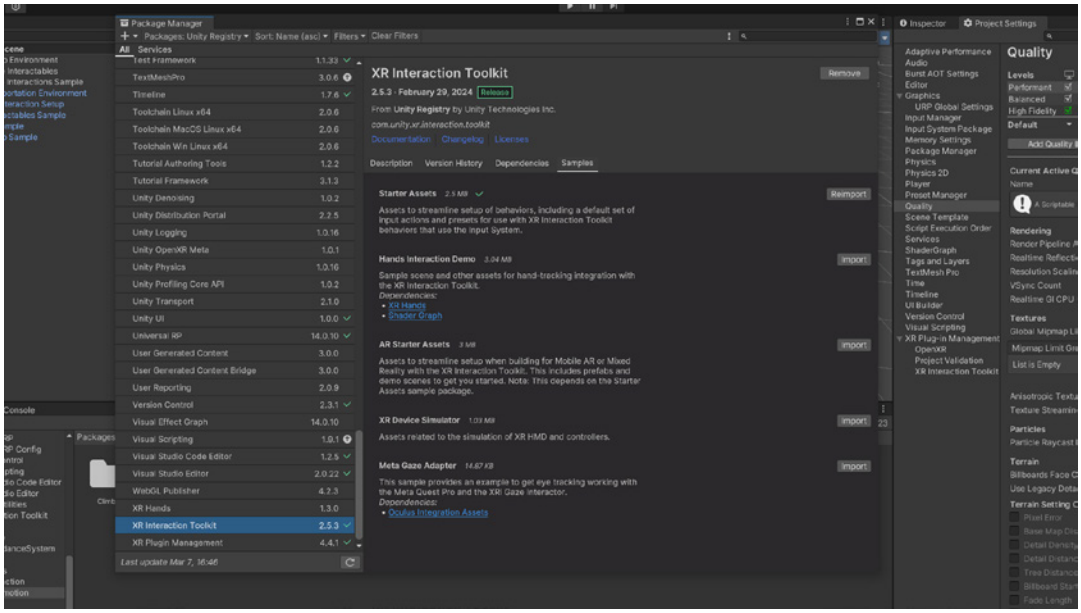
The XR Device Simulator is a runtime utility that's included as part of the Samples add-on in the XR Interaction Toolkit package. This utility lets you simulate user inputs from plain key presses (be it from a keyboard and mouse combo or a controller) to drive the XR headset and controller devices in the scene.

XR Device Simulator can save you time because it lets you efficiently test and iterate on your project directly in the Unity Editor, without the need for constantly putting on a physical device. It allows for the simulation of various environmental conditions and user scenarios so you can develop an application that handles a wide range of real-world conditions and user inputs. It can provide early insights into potential performance issues, potentially reducing the need for extensive optimizations and hardware testing late in the development cycle.

Let's take a look at how we can bring the simulator into a sample scene we used earlier.

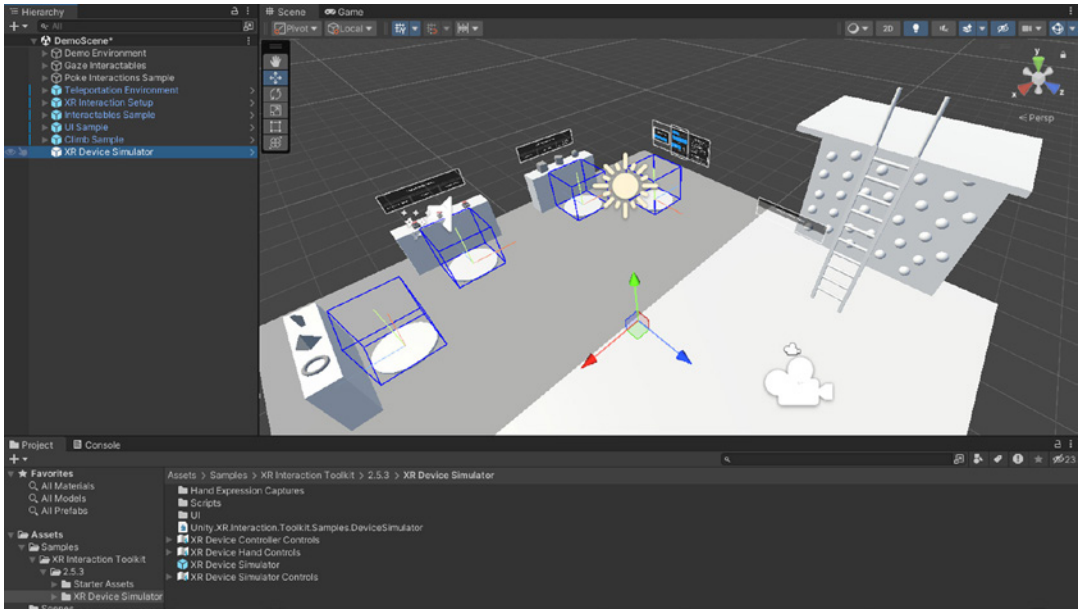
Installing the XD Device Simulator

The XD Device Simulator requires importing, just like the starter assets. Navigate to it via **Package Manager > XR Interaction Toolkit > Samples**. Select the simulator in the list and click on Import.



The Package Manager with the XR Device Simulator ready for importing

The XR Device Simulator is easy to test in your scene. In the demo scene you can locate the **XR Device Simulator** prefab from the Project window and drop it into the scene, as shown here:



Locating the XR Device Simulator prefab and dropping it into the demo scene

Going into Play mode in the Editor will launch the simulator, letting you move around using your mouse and keyboard. A set of onscreen instructions guides you on what features are linked to which keyboard key or mouse button.

The XR Device Simulator running in Play mode

You can now navigate the scene and interact with the elements in your game as you would if you were wearing a headset.

The Unity Profiler

The Unity Profiler provides you with detailed information about the various aspects of your game's runtime behavior, such as CPU usage, memory allocation, rendering, and network operations. By presenting real-time data in an easy-to-read format, the Profiler enables developers to identify and analyze performance bottlenecks and inefficiencies.

The Profiler is particularly valuable in a development environment where resource management is critical for maintaining smooth gameplay, especially for complex scenes or on lower-end hardware. It provides a comprehensive view of a game's performance metrics, helping you to make informed decisions about where to optimize.

When using the Profiler to optimize XR games and applications, it's important to know what specific aspects to focus on. Here's a breakdown of key areas to monitor:

CPU usage:

— **Main thread:** Check if the main thread is overloaded. High usage can lead to frame rate drops.

— **Scripts:** Look at the script execution time. Poorly optimized scripts can be costly.

— **Physics:** In XR, physics calculations can be intensive; monitor their impact on CPU.

GPU usage:

— **Rendering:** XR demands high and stable frame rates. Monitor render times to ensure the GPU isn't a bottleneck.

— **Draw calls:** Numerous draw calls can slow down the GPU. Aim to reduce them for better performance.

Memory usage:

— **Total allocated memory:** Keep an eye on how much memory your application uses.

— **Garbage collection:** Frequent garbage collection can cause performance spikes so look for ways to reduce allocations.

Frame rate:

— **Stability:** XR applications should maintain a consistent frame rate. Look for drops or inconsistencies.

— **Target frame rate:** Ensure your application meets the target frame rate for the intended XR platform.

Network performance (for multiplayer XR):

— **Latency:** High latency can disrupt the user experience in multiplayer XR.

— **Data transfer:** Monitor the amount of data being transferred and optimize to reduce lag.

Audio profiling:

— **Audio source management:** Multiple audio sources can be demanding. Ensure efficient use.

— **DSP load:** Check Digital Signal Processing load, especially if you're using advanced audio effects.

Rendering pipeline:

— **Batches and SetPass calls:** Optimize for fewer batches and SetPass calls to reduce GPU load.

— **VR-specific metrics**: If available, look at VR-specific metrics like latency and inter-pupillary distance adjustments.

UI profiling:

— **Canvas rendering:** Complex UIs can be taxing. Optimize canvas elements and their updates.

Custom profiling markers:

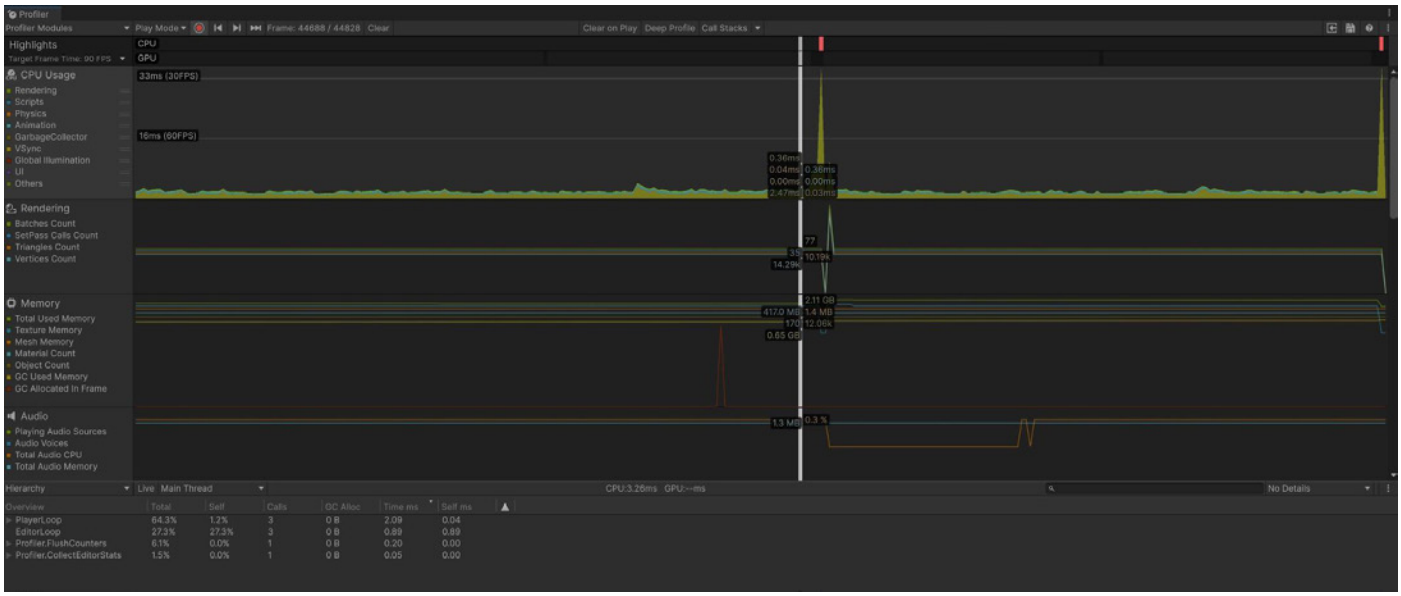
— **User-defined markers:** Use custom markers in your code to monitor specific methods or processes that are crucial for your application.

Thermal throttling (especially for mobile XR):

— **Device temperature:** High-performance demands can lead to overheating in mobile devices, causing thermal throttling.

By carefully analyzing these areas, you can identify performance bottlenecks and optimize your XR game or application accordingly. Remember, the goal is to maintain a balance between visual fidelity, responsiveness, and smooth performance.

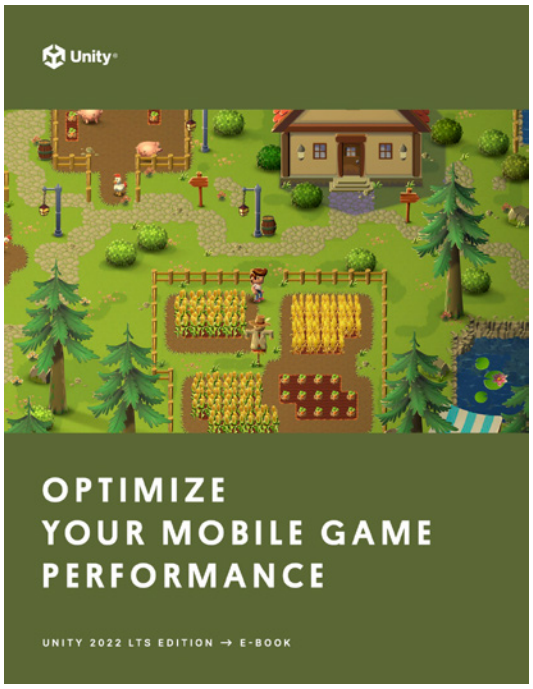
You can open the Profiler window via **Window** > **Analysis** > **Profiler**, or press **Ctrl+7**. When you press Play the Profiler starts and the view will be populated with data captured from your project.



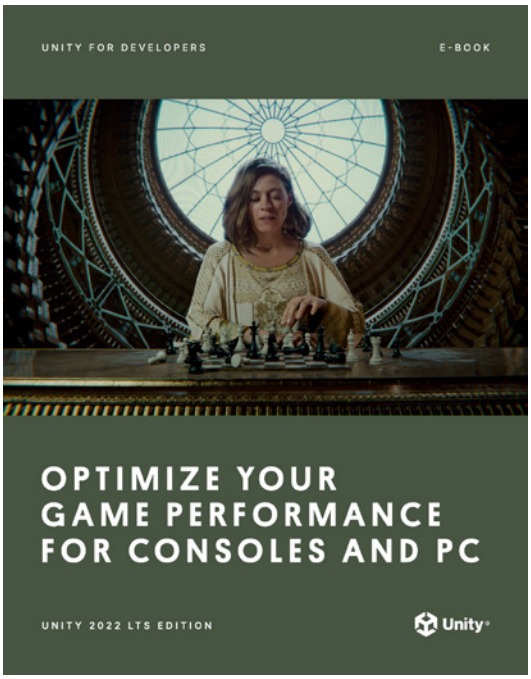
Profiling a project in the Profiler

You can pause at any time to identify areas within your game that are causing the frame rate to drop. Utilizing the left hand menu you can select the specific layer such as **Rendering** or **CPU Usage** to get more information. Each item is color coded to help you pinpoint quickly where the game is experiencing performance issues.

In addition to the aforementioned profiling e-book, you'll find many optimization tips in our two advanced optimization guides:



Download



Download

Build and deployment

You've spent time constructing your world and building your interactions, testing, and optimizing. Now it's time to unveil your game to the world. You can now build your game straight to your device or you might like to submit it to a store. Each store has its own set of guidelines and requirements for your app, so be sure to check for documentation on submitting your app.

Building VR apps

Mixed reality and spatial computing applications

Mixed reality (MR) applications integrate the real world and a virtual one in a hybrid landscape where the blending of these elements should feel seamless to the user.

MR development combines aspects of VR and AR worldbuilding, including the following.

User interaction and interface design

There are common principles of UX and UI design from VR and AR that you can apply to MR apps. You'll need to design intuitive interfaces and interaction models that feel natural across different devices. This includes leveraging hand tracking, gaze controls, and voice commands, ensuring that users can interact with the virtual world in a way that feels instinctive.

Spatial awareness and physics

Understanding and integrating the physics of the virtual world is vital across VR, AR, and XR. This includes creating realistic interactions between virtual objects and, in the case of AR, blending these interactions with the real world. Developers must account for spatial awareness, ensuring that virtual objects behave in ways that are consistent with user expectations regarding gravity, collision, and other physical properties.

Cross-platform development strategies

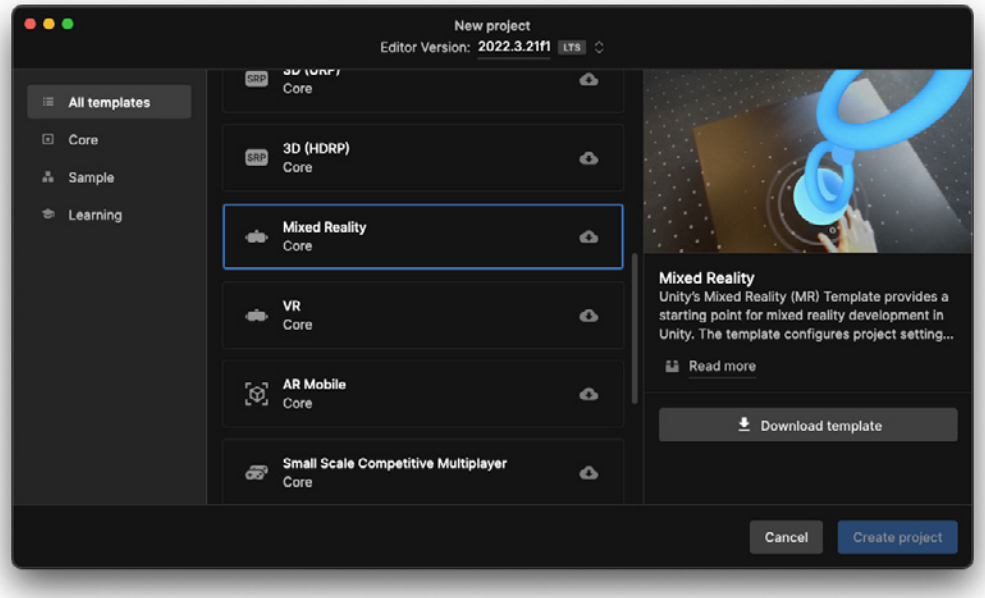
With the diverse range of devices available for VR and AR, from standalone headsets to mobile-based systems, developing with cross-platform compatibility in mind is crucial. Unity provides tools and workflows that allow developers to create applications that can be deployed across multiple platforms with minimal adjustments. This includes adapting user interfaces, interaction models, and performance optimizations to suit the capabilities and limitations of different devices.

Environmental design and immersion

Environmental design in MR combines immersion techniques from both VR and AR. This involves not just the visual aspect but also incorporating spatial audio, haptic feedback, and other sensory inputs to create a convincing and engaging world. The goal is to make users feel present within the virtual environment, whether it's a completely virtual world in VR or a digital overlay in the physical world in AR.

MR template

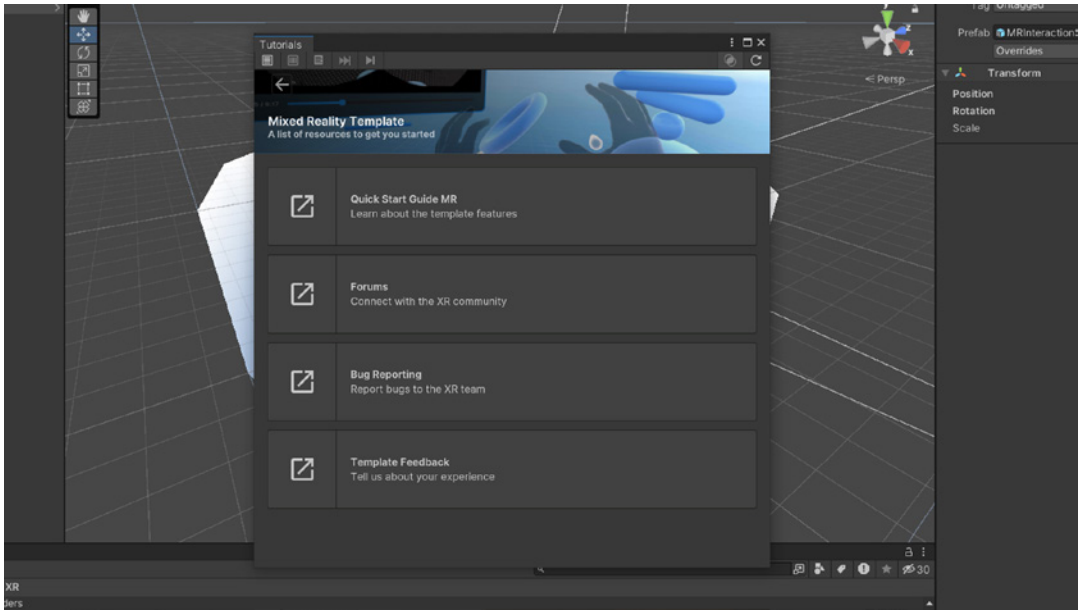
Unity's MR template is designed to serve as a starting point for mixed reality development in Unity.



The Mixed Reality template in the Unity Hub

Download the template and create a new project. It's recommended that you're using at least Unity 2022 LTS or newer.

The project will open with a welcome screen. There is also a window that provides links to tutorials, the dedicated forum, bug reporting, and more.



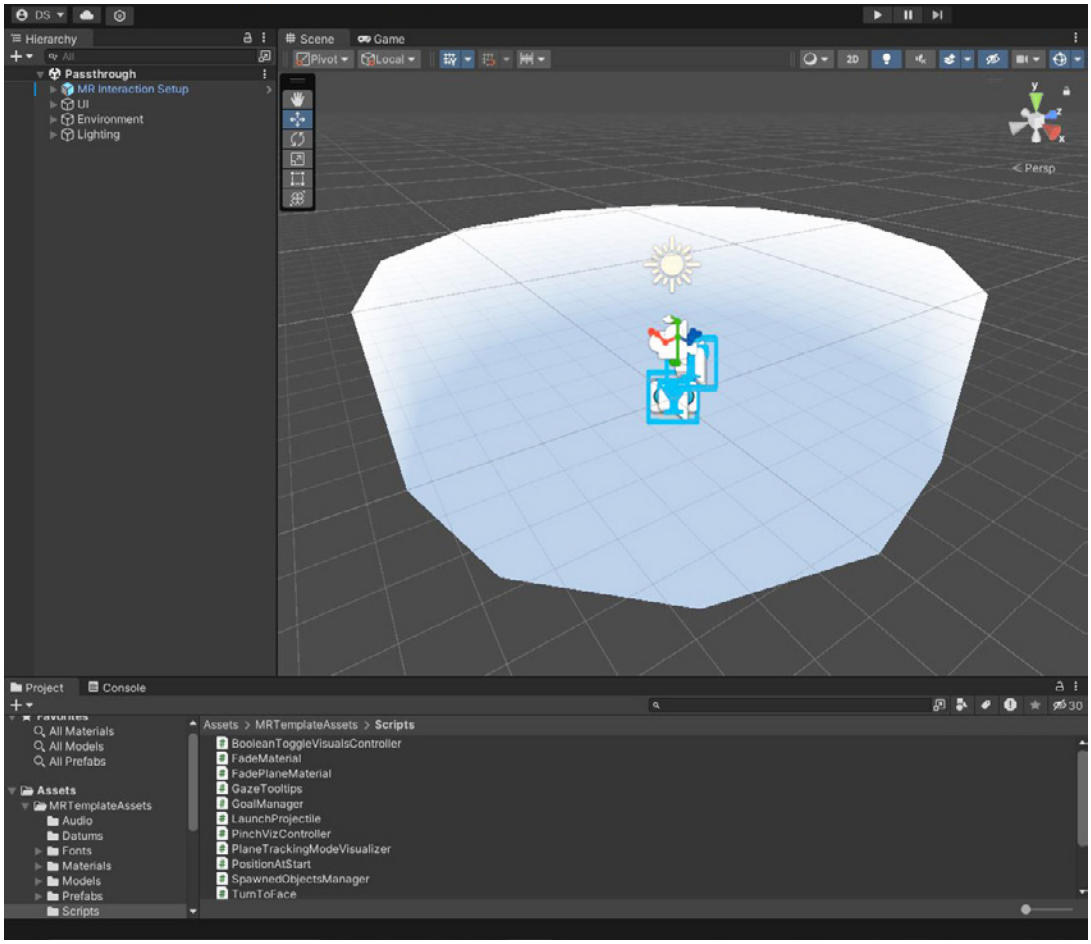
Tutorial content window

You'll also find links to the documentation for the MR template in the Quick Start list.

The MR Template utilizes the OpenXR, AR Foundation and XR Interaction Toolkit toolsets.

MR Interaction Setup prefab

The **MR Interaction Setup** prefab is the primary GameObject for configuring the XR camera and the point of origin in the XR experience. It allows for different input systems like Controllers and Hands and includes various interactors like poke, direct, and ray.



The MR Interaction Setup prefab in the Hierarchy of the MR template

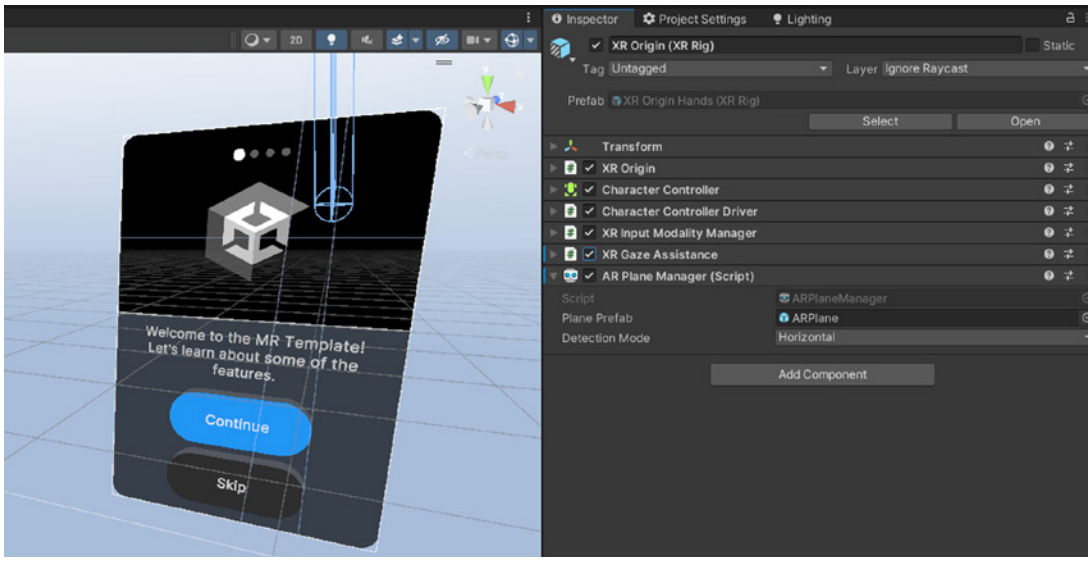
The MR Interaction Setup prefab has various components that should be familiar to you if you've read earlier chapters in this guide: The Input Action Manager, XR Interaction Manager and XR Origin (XR Rig). Let's look at some of the components unique to the MR template.

AR Session

The **AR Session** component manages the life cycle and settings for an AR session. There's just one session active at any time.

AR Plane Manager

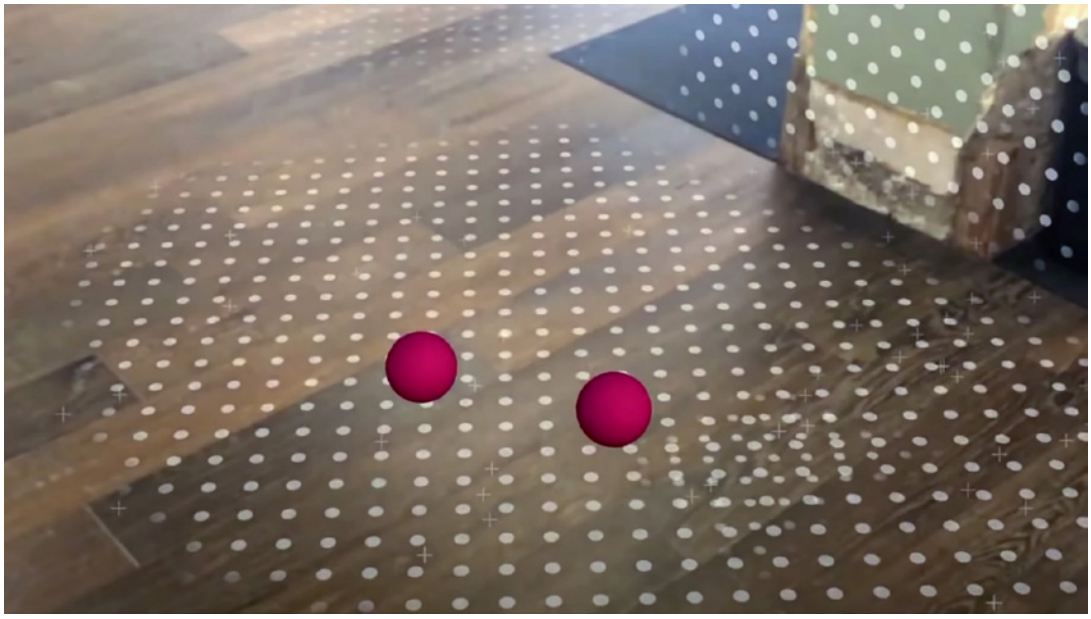
The **AR Plane Manager** is responsible for generating GameObjects for each plane detected in the player's surroundings. It utilizes the device cameras and AR foundation to manage detected planes. This component is situated on the same GameObject as the XR Origin component.



The Plane Manager component on the XR Origin.

A plane is a flat surface represented by a pose, dimensions, and boundary points. Examples of features that can be detected as planes are horizontal tables, floors, walls, and countertops. A Detection mode is available that can be set to horizontal, vertical or both. Additionally, the manager provides the option to visualize the detected planes using custom prefabs.

AR Plane on a vertical surface



Planes detected on the floor and walls

Passthrough

Passthrough technology allows VR headsets to show both the real world and virtual objects simultaneously, using the cameras on the device to capture what's around you. In the MR template, passthrough works by combining software and hardware to mix digital content with the real world instantly. This feature is mainly used in AR experiences, letting people see and interact with their real environment while they're in a virtual scene. In Unity passthrough relies on several important parts:

— **XR Plugin Framework:** This system is essential for enabling passthrough, as it provides the necessary interfaces and controls for accessing camera feeds and other hardware resources.

— **Hardware support:** Your target device must have the necessary hardware to support passthrough, such as cameras capable of capturing live video from the user's environment. The quality and capabilities of this hardware can significantly impact the effectiveness and realism of the passthrough experience.

— **Camera feed access:** The core of passthrough functionality is the ability to access the live feed from the device's cameras. This feed is then rendered in real-time within the user's field of view, creating a window to the real world within the virtual environment.

— **Software components:** Specific components within the Unity project, such as the AR Foundation package for ARKit and ARCore devices or the Oculus Integration package for Oculus devices, provide the tools and APIs necessary for implementing passthrough. These components manage the complexities of rendering the camera feed, handling user input, and integrating digital content with the physical environment.

Testing the MR template

The project settings within the MR template are preconfigured, with the XR Plug-in Management package set to use OpenXR with the Meta Quest feature group. Two interaction profiles have been added, the **Meta Quest Touch Pro Controller Profile**, and the **Oculus Touch Controller Profile.**

We'll test the template on the Meta Quest. Meta Quest is an Android-based platform that supports the Android build target for standalone VR/MR apps, and the Windows build target for PCVR apps via Meta Quest Link. For this template the mixed reality features are currently limited to standalone/Android support so the build target is set to Android. If you get error messages, ensure that Android is the chosen platform for the project as the default Windows, Mac, Linux platform might throw errors.

To enable the plane detection to function, the headset will need to be configured to the space the user is playing in. On the Meta Quest this involves setting up the physical space within the settings. This will tell the headset where the walls, floor, ceiling and other items are.

Upon building and running the project you will be presented with a UI that takes you on a small guided tour of features of the sample project. You have the ability to tap on horizontal surfaces and place 3D objects, move them around and through a UI activated by your hands remove them, along with some other functions.

In Unity's Mixed Reality project template, a lot of the difficult setup work for passthrough technology is simplified. This means developers get a package with settings, scripts, and resources already adjusted for creating MR projects. It's designed to help you get started more easily with applications that blend the digital and real world. Despite this, depending on what your project needs and which devices you're targeting, you might still have to tweak some settings, add extra parts, or write your own code to get everything working just right.

Spatial computing with Apple Vision Pro

A Unity sample running as a windowed app in visionOS

Apple Vision Pro is Apple's first spatial computer. It integrates the physical and digital worlds, allowing users to interact with both simultaneously in applications. The device leverages advanced sensors, display technologies, and software algorithms to create highly realistic VR and AR experiences.

By recognizing and responding to the user's environment and movements, spatial computing on Apple Vision Pro enables a wide range of applications, from gaming and entertainment to education and professional collaboration.

WHAT THE GOLF? by Triband, running as an immersive experience in Apple Vision Pro

Get started with visionOS In Unity

This section looks at the different types of Apple Vision Pro experiences you can create with Unity:

- **Immersive (Mixed Reality):** Experiences that blend digital content with the real world and can run alongside other apps in the Shared Space, made possible by Unity's newly developed PolySpatial technology
- **Fully Immersive (Virtual Reality):** Ports of existing virtual reality games or new, fully immersive experiences that replace a player's surroundings with another environment
- **Windowed:** Content that runs in a 2D window and can be resized and repositioned by the user in the Shared Space

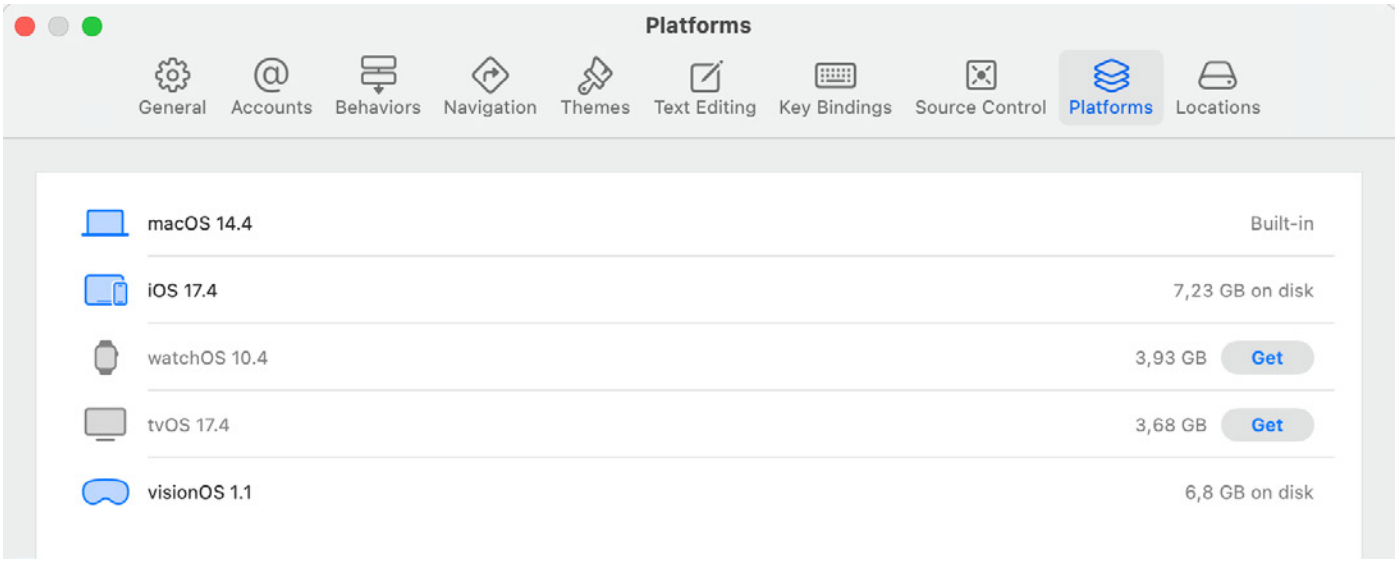
You'll need the following hardware to create visionOS apps:

— A Mac with Apple silicon is required for visionOS development, and Xcode 15.2 or newer with visionOS simulator

— visionOS Simulator requires an Apple Silicon (M1 or newer) Mac

— visionOS Simulator needs to be installed on Xcode Settings / Platforms

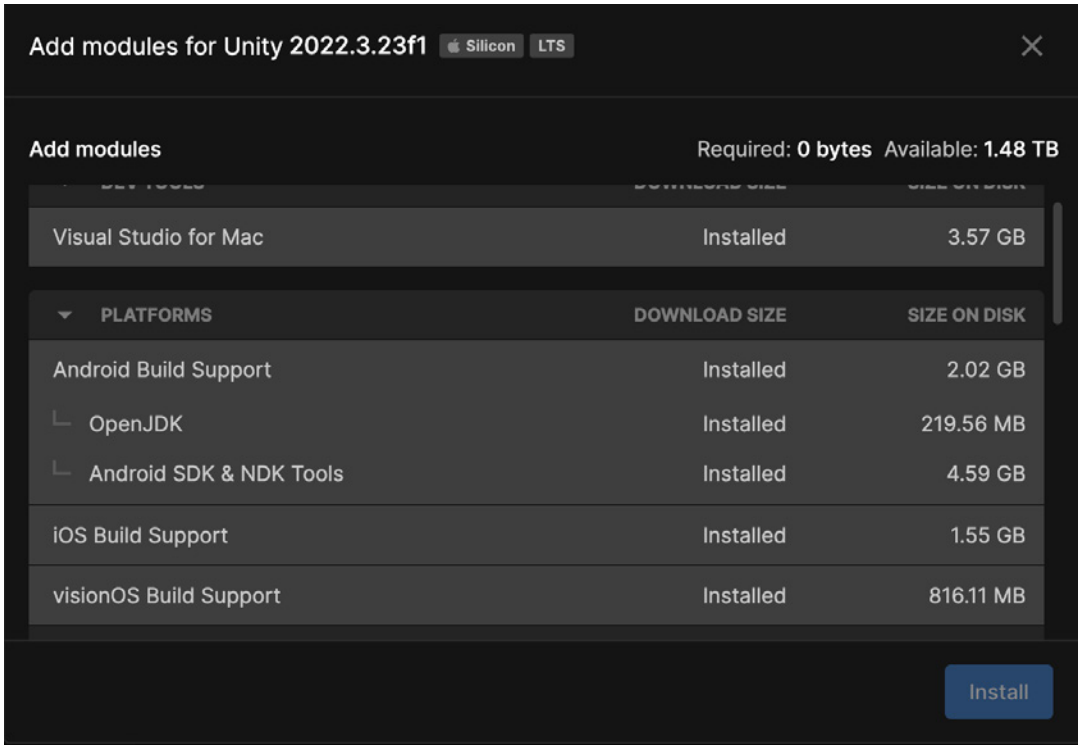
— If you don't have visionOS hardware, use the simulator built into XCode



The available simulators in Xcode

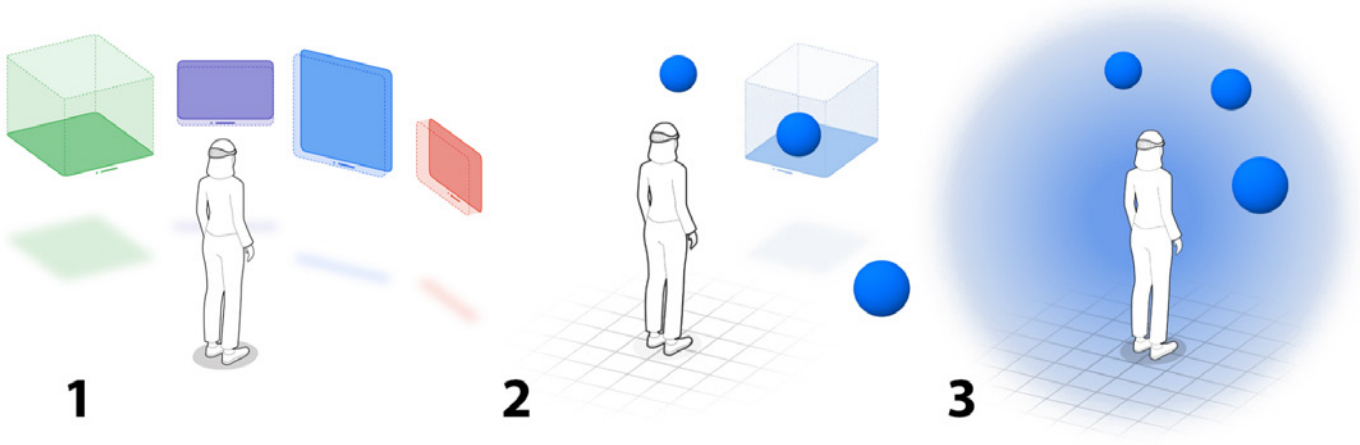
To get started with developing visionOS apps download and install the latest Unity 2022 LTS version from the Unity Hub.

From the modules list, select and install the **visionOS Build Support** and **iOS Build Support** modules.



An illustration of how apps are shown and accessed in Vision Pro; images courtesy of AppleAbout visionOS and Unity

About visionOS and Unity



An illustration of how apps are shown and accessed in Vision Pro; images courtesy of Apple

Apps in visionOS are launched from their icon in the Home View, similar to the grid of icons on an iPhone or iPad. The image above illustrates how apps are displayed in Vision Pro:

1. Apps that run content on a window or volume launch in the Shared Space **(1)**, where they exist side by side, and like multiple apps on a Mac desktop, you can move them and rearrange them.

2. When an app uses a volume **(2)** only the content inside the Volume Camera will be visible; the camera's mode is set to **Bounded**, which will show the content of the app in the Shared Space.

3. Apps can also take the whole space available in Immersive Space **(3)** when the camera's mode is set to **Unbounded**; this hides the Shared Space apps and focuses only on the open app.

In both Bounded and Unbounded modes the camera passthrough and physical space will blend the real world in front of you with the virtual 3D objects.

Shared Space with apps running in a Volume Camera with Bounded mode applied

The two additional modes where the camera feed and physical space are not used for the content of an application are **Fully Immersive VR**, which puts the player or user into a fully virtual world, and **Windowed Apps**, which you can use for 2D or 3D experiences that are rendered in a flat floating window in front of you and can coexist in the Shared Space. iPhone and iPad apps run as Windowed Apps on visionOS.

The visionOS Home View from where you can launch all the apps available on your device.

Here's a summary of the main features and requirements of these different XR experiences for Apple Vision Pro:

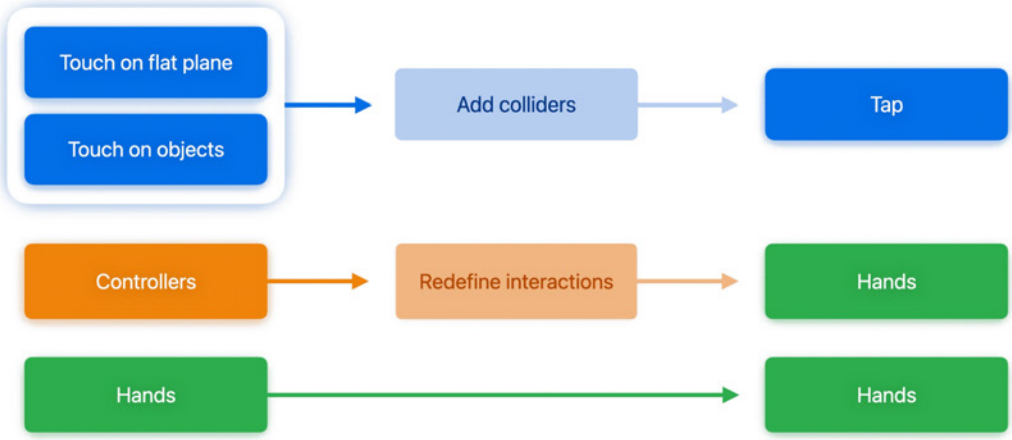
App type	Shared Space	AR	Rendering technology	Play to device	Requires Vision Pro plugin
Fully Immersive VR	No	No	Metal	No	Yes
MR in Bounded mode	Yes	Yes	RealityKit	Yes	Yes
MR in Unbounded mode	No	Yes	RealityKit	Yes	Yes
Windowed App	Yes	No	Metal	No	No

The Vision Pro plugin will automatically install visionOS packages if it's necessary.

Let's look at how interactivity works in Unity projects for the Apple Vision Pro.

Interaction

In visionOS apps, users use their eyes and hands to look at and interact with content. They select items by tapping their fingers together. Full hand tracking as well as head pose data is also available to the app, with the user's permission. Custom gestures make the interactions feel realistic, with the tap gesture being the common way of interactivity.



Converting existing applications interactions to visionOS: Source: Apple Inc.

You need to attach input colliders to your objects so they can receive these events. You can look and tap to select an object from a distance, or you can reach out and directly touch an object with a finger.

In Unity, taps are available as WorldTouch events. They are similar to 2D tap events, but have a full 3D position.

If you are already working with touch, such as on an iPhone, you can add appropriate input colliders and continue to use tap as your primary input mechanism.

If you are using VR controllers, you will have to redefine your interactions in terms of either tap- or hand-based input, depending on how complex they are.

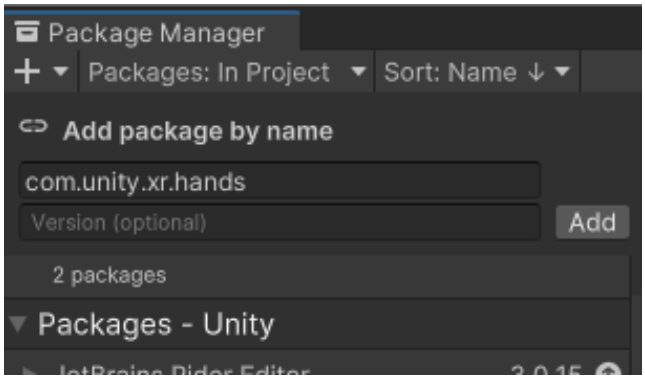
3D Touch and TouchSpace

In both bounded and unbounded volumes, a 3D touch input is provided when the user looks at an object with an input collider and performs the "pinch" gesture. The *SpatialPointerDevice* Input device provides that information to the developer. If the user holds the pinch gesture, a drag is initiated and the application is provided with "move" updates relative to the original start point.

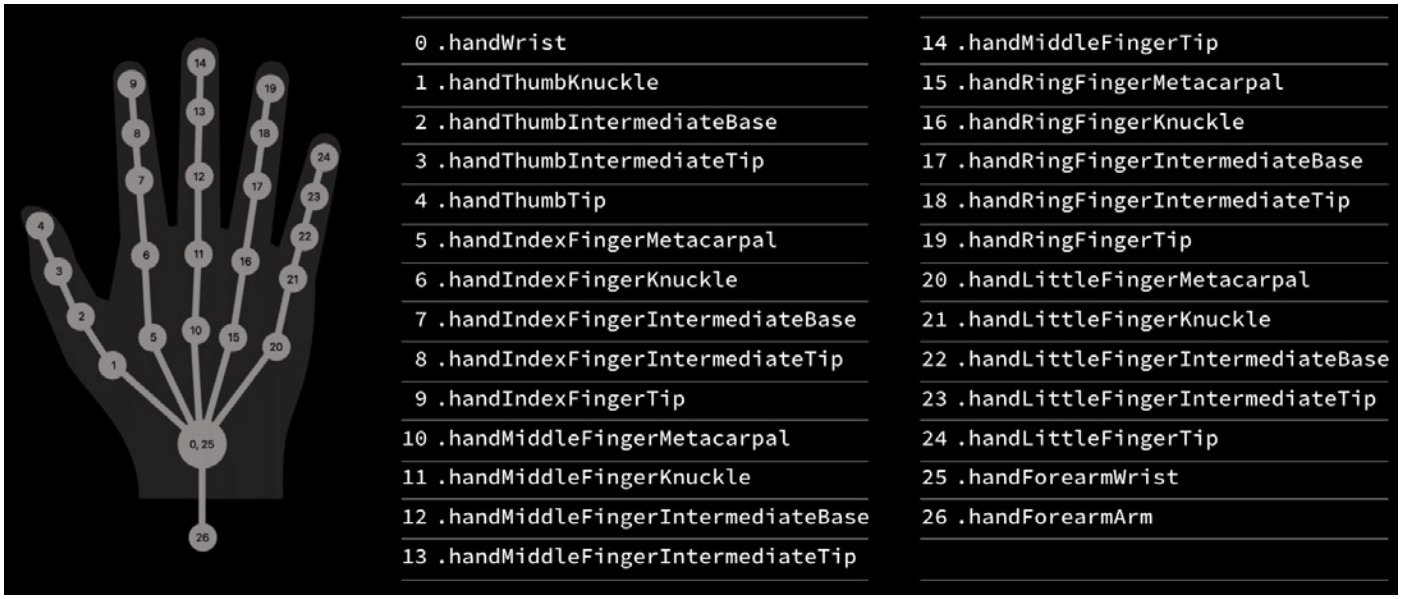
This input device has a VR counterpart called *VisionOSSpatialPointerDevice*. The primary difference between the two is that the interaction doesn't require colliders. Thus, VisionOSSpatialPointerDevice is missing input controls related to the interaction (targetId, interactionPosition, etc.).

Skeletal Hand Tracking

Skeletal hand tracking is provided by the Hand Subsystem in the XR Hands Package. Using a Hand Visualizer component in the scene, users can show a skinned mesh or per-joint geometry for the player's hands, as well as physics objects for hand-based physics interactions. You can also access raw data with the XR Hands package that's available in the Package Manager.



The XR Hands package tracks articulations and allows you to define your own gestures or replace the hands visuals for custom meshes.



Mapping of the hand data to a hand representation

You can see an overview of the available input methods in the following table; note that hand tracking and AR data require permissions by the user.

App type	Look and tap (Input System touchspace events)	Head pose (Input System head pose events)	Hands (XR Interaction Toolkit or XR Hands package)	AR data (AR Foundation package)	Bluetooth devices (Input System)
Fully Immersive VR	Yes	Yes	Yes	No	Yes
MR in Bounded mode	Yes	No	Yes	No	Yes
MR in Unbounded mode	Yes	Yes	Yes	Yes	Yes
Windowed App	Yes	No	No	No	Yes

Fully Immersive VR

Job Simulator, by Owlchemy Labs, is a VR app for visionOS. Read more about their game in this Unity blog post.

Creating or porting VR apps to visionOS is possible using the Unity workflows that you are familiar with, such as VR development, as detailed earlier in this e-book.

The build process for visionOS is similar to deploying on iOS. Unity generates an Xcode project and from Xcode, you can run the project on a simulator or a device and publish.

In the context of Apple Vision Pro development a fully immersive VR app uses Unity's graphics technology and the Metal framework to render your project.

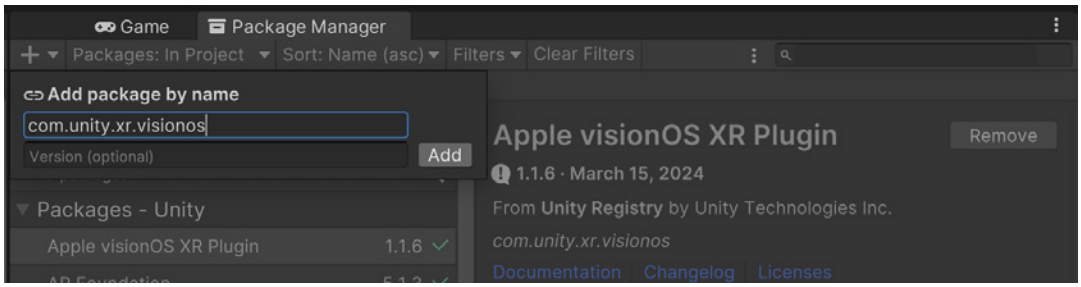
If your project already runs on other VR platforms it can likely run on visionOS as well.

It's recommended to use URP because it supports foveated rendering. This feature concentrates more pixels of resolution in the area of vision where your gaze is most likely to be focused, reducing the resolution outside this area, thereby improving performance and the user experience. Foveated rendering is enabled by default. Without it, Apple Vision Pro doesn't allow you to render at the full native resolution of the device. Developers porting existing apps might need to disable it as a fallback but it's not recommended.

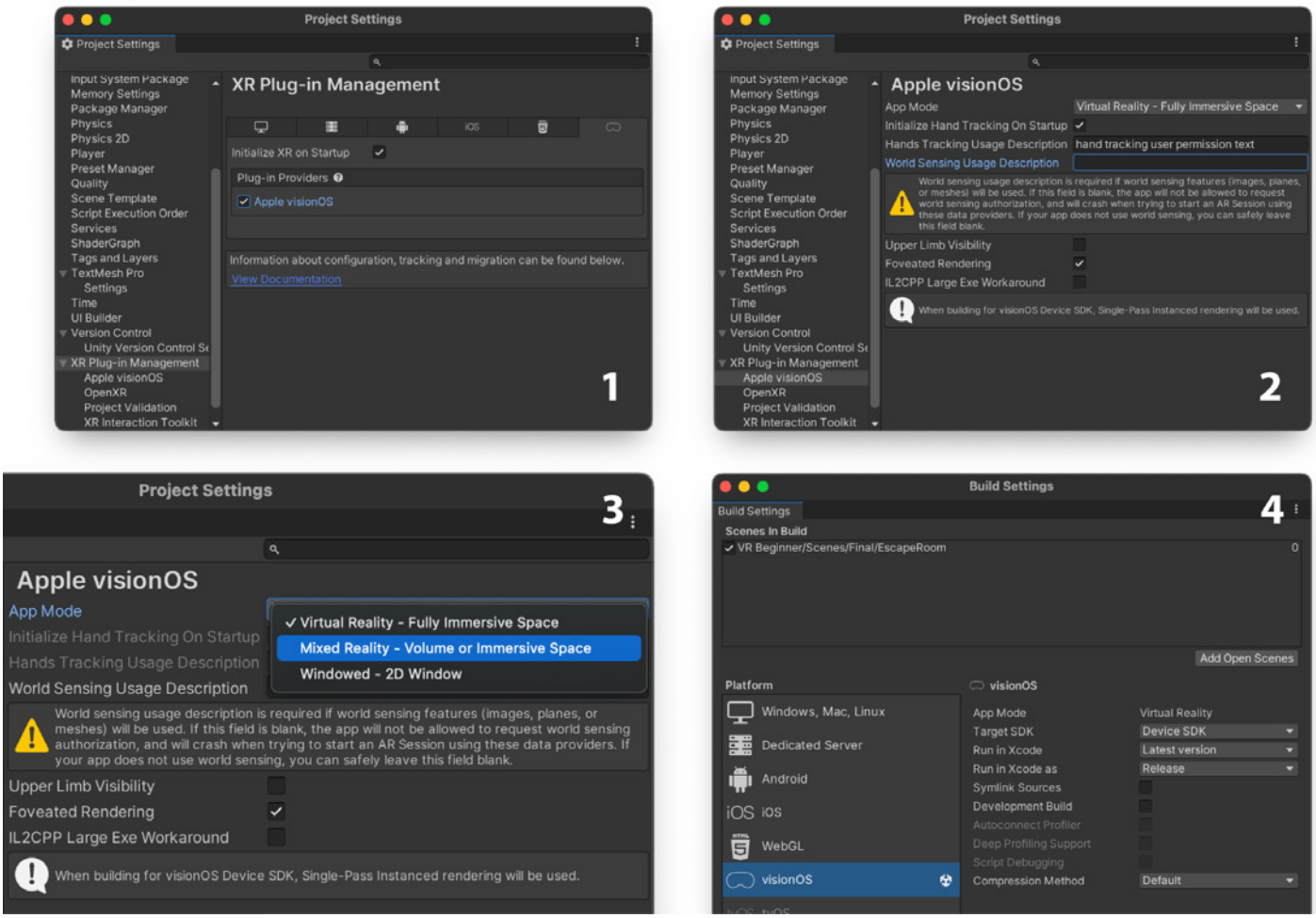
You can optimize your project using single-pass instance rendering, which now supports the Metal graphics API. In single-pass rendering, Unity submits one draw call for both eyes, reducing the CPU overhead from rendering for each eye separately.

In visionOS the depth buffer is used for reprojection. If there's depth information missing the system will render an error color to indicate the issue. All Unity shaders should work out of the box.

To run your app in Fully Immersive mode you'll need to install the Apple visionOS XR Plugin; if it doesn't appear in your Unity version you can add it from the XR Plug-in Management package by selecting the **com.unity.xr.visionos package** from the Package Manager dropdown.



It's good practice to also check the Project Validation section, in the XR Plug-in Management package, for potential settings that need changing to run on visionOS.



Installing the visionOS Plugin allows you to define what type of app you're building: VR, MR, or Windowed

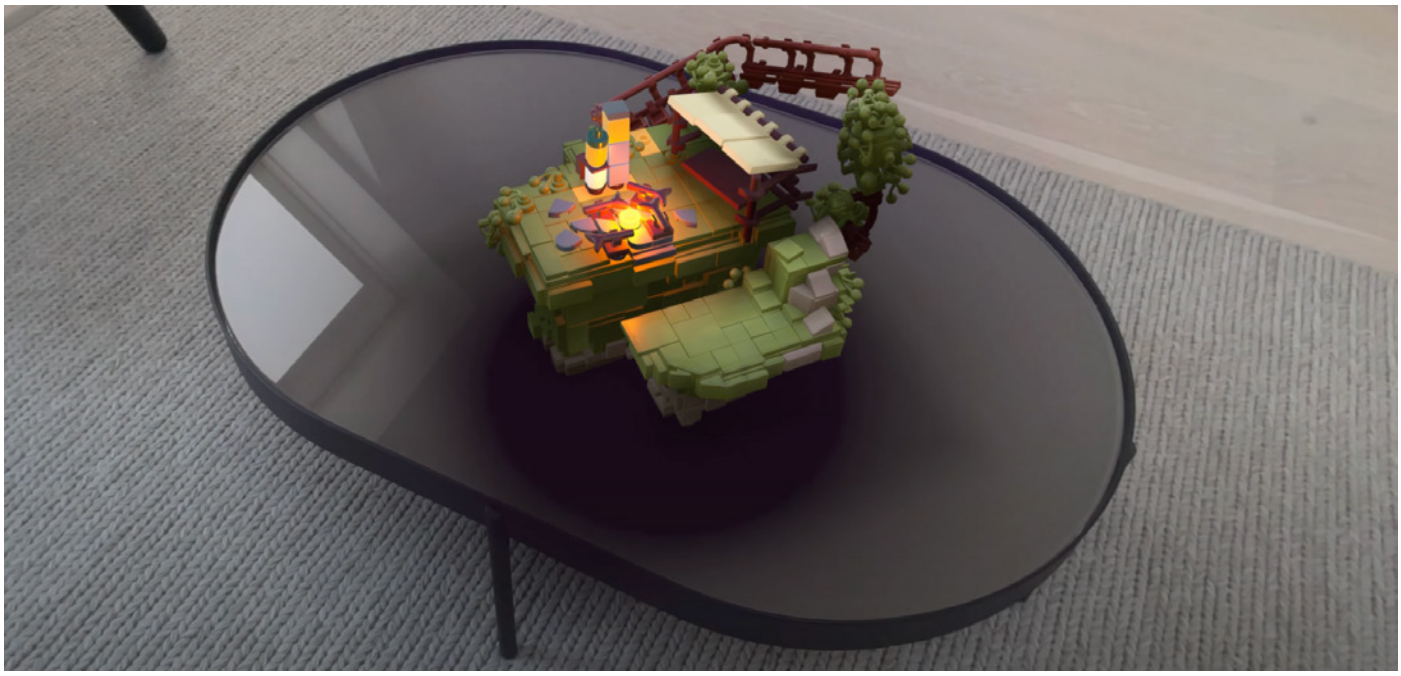
Once you've enabled the visionOS Plugin in XR Plug-in Management, click on **Apple visionOS** in the left-hand menu, select your app type and from there, the type should be reflected in the Build Settings window.

From the Build Settings remember to select the **Target SDK**>**Simulator SDK** if you want to run the project in the Xcode simulator, or **Device SDK** to run it on a Vision Pro device.

A VR app running on the visionOS Simulator

Read the documentation for more information about developer VR apps for Vision Pro. Unity also provides in-depth e-books for learning about how to use Unity graphics systems and tools.

MR apps



Read about how a team of six from Light Brick Studio ported LEGO® Builder's Journey to Apple Vision Pro in just three months.

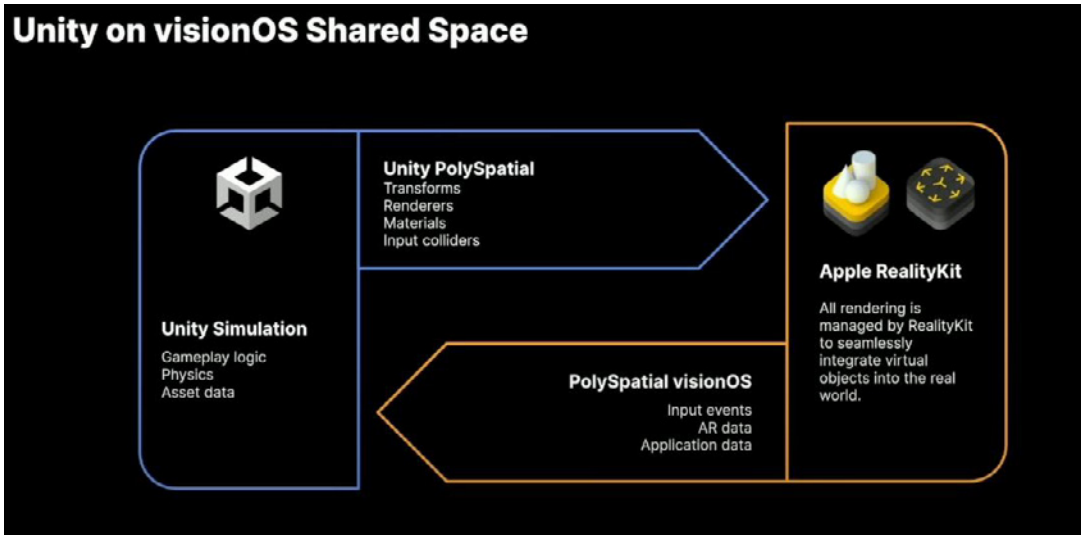
You can create MR apps that run in Apple Vision Pro Shared Space with Unity's PolySpatial technology. These apps are simulated with Unity, but rendered with RealityKit, the system renderer of visionOS. Most of the Unity VR development paradigms apply here, with the addition of AR components.

To start developing MR apps, go to the Apple visionOS project settings window and change the app mode to **Mixed Reality - Volume or Immersive Space.** A pop-up prompt will ask you to install PolySpatial. Another option is to manually add the package by name: com.unity.polyspatial.visionos

Both the visionOS Plugin and Polyspatial packages are required for creating MR apps

Unity PolySpatial

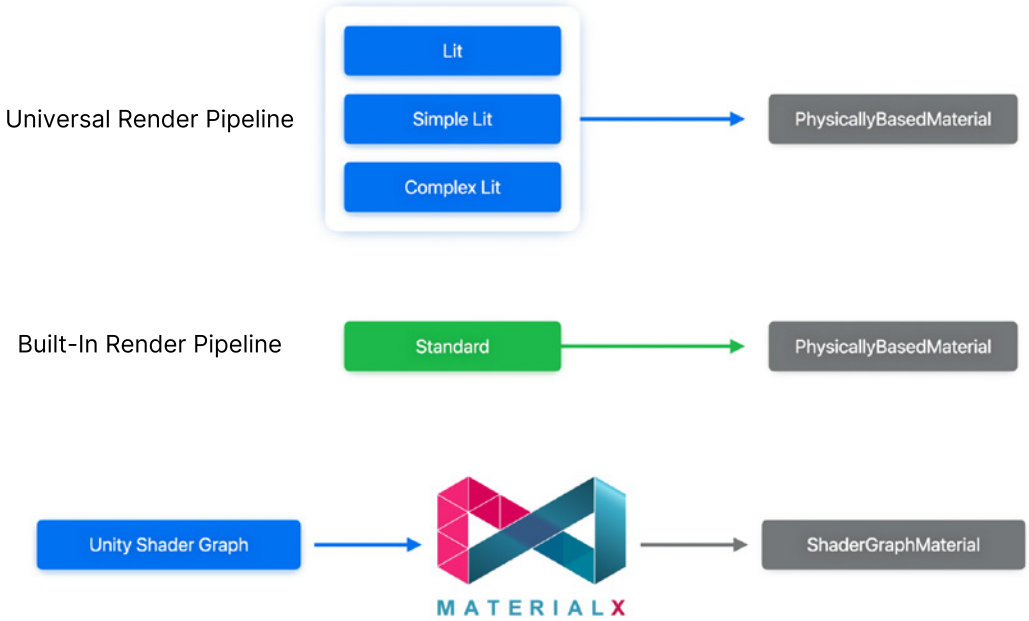
Unity PolySpatial, as part of Unity's visionOS packages, is the underlying technology that enables you to create spatial experiences for the visionOS platform that can coexist alongside other apps within the Shared Space.



Overview of Unity's simulations and graphics converted into Apple's RealityKit via Unity's PolySpatial technology

Graphics and simulation

PolySpatial takes care of the translation needed for graphics to be rendered with RealityKit and ensures that many of Unity's features and workflows will work in MR.



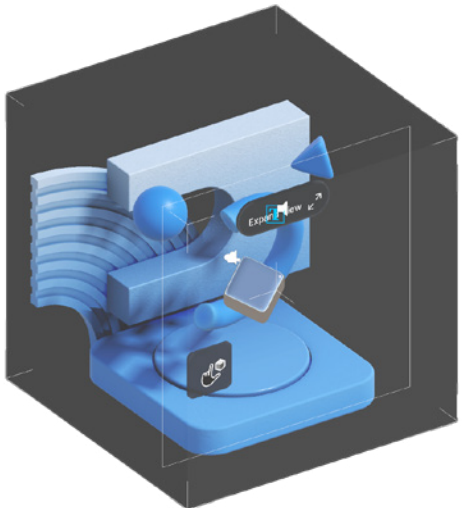
The conversion flow that happens under the hood from Unity materials to RealityKit materials.

Let's take a look at the process more closely:

- **Materials**: The URP Lit, Simple Lit, and Complex Lit shaders and custom materials made in Shader Graph will work in RealityKit. MaterialX is used as the exchange format for complex materials and for converting custom shaders. Most of the Shader Graph nodes are supported, but you can find the support details in the documentation. There's also support for effects materials, unlit, and occlusion materials.
- **Mesh renderers:** The Mesh Renderer and Skinned Mesh Renderer are supported in URP or the Built-In Render Pipeline.
- **Particle effects:** Simple effects created with the Built-In Particle System are translated to RealityKit. Complex effects are baked into meshes and sprite effects are converted into meshes too. VFX Graph is not supported in MR but you can render the effects into a Render texture to use in your project.
- **Sprites**: These are supported.
- **Simulation**: World simulation features will continue to work, like Monobehaviours, ScriptableObjects and other standard tools:
  - Physics
  - Animation and Timeline
  - Pathfinding
  - MonoBehaviours
  - Other non-rendering features

Volume Cameras

Volume cameras are a new concept in PolySpatial. Your app can switch between the two types at any time. The Volume Camera component can cull a layer mask, define its dimensions – the bigger the size the more scene view that will be displayed – and it has Unity events to trigger functions based on changes to the app window.. The camera mode is set up with a **Volume Camera Configuration** asset.



Setting up a bounded camera in Unity

Bounded camera

In this mode Unity can define the area of the space that is used in the devices based on real world units to display the content inside the Volume Camera.

Find samples of bounded cameras in the documentation

Unbounded camera

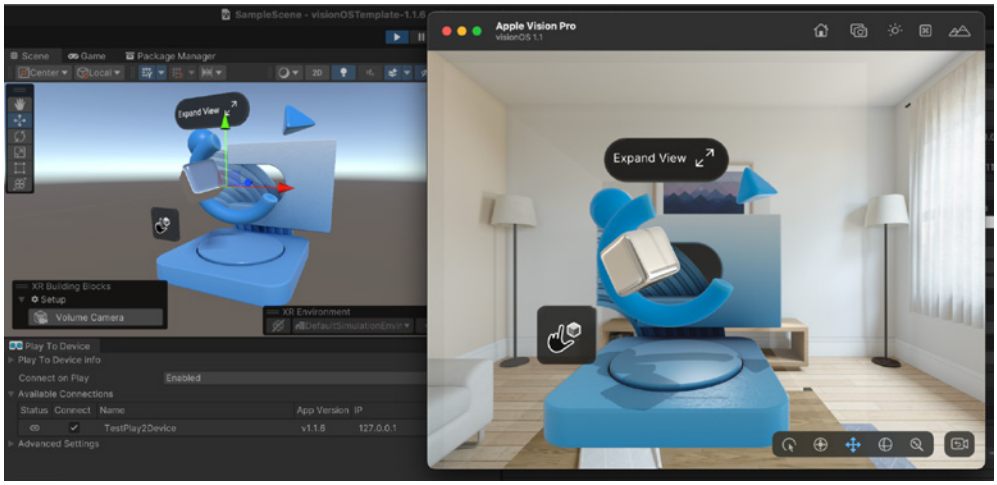
In this mode the MR experience can transition to a full space. Here an app has the freedom to utilize the entire visual field of the user. Your unbounded content is the only content you will see in the space. You can't set up the dimensions of the Volume Camera or the Output dimensions; Unity units are mapped to real-world units (1 Unity unity = 1 meter) There can only be one unbounded camera at a time.

An unbounded MR app that fully utilizes the space; read about other samples to test in the documentation.

Play To Device

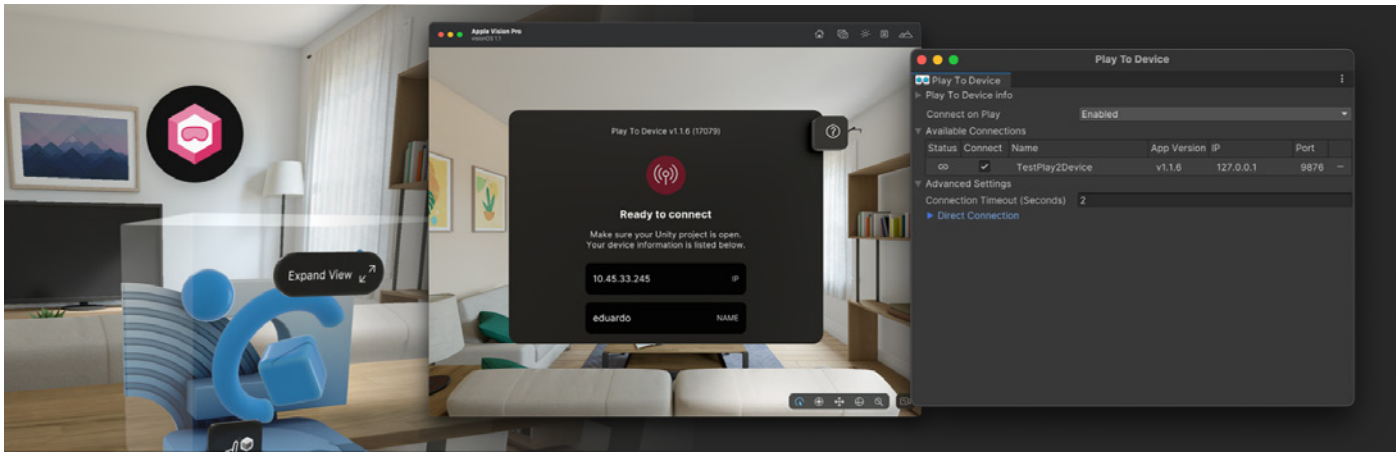
This feature allows devs to preview their content in Play mode in a visionOS device and simulator without the need to create an Xcode build. It's available for apps rendered using RealityKit, which are the mixed reality apps.

With Play to Device you can efficiently check placement of elements, change material textures and Shader Graphs, test interaction, adjust values, and more. Any interactions you perform on the device or simulator will be synchronized back to the Editor. You can learn more about this workflow by reading this Discussions post or the documentation.



Play To Device running side by side with Unity's Play mode

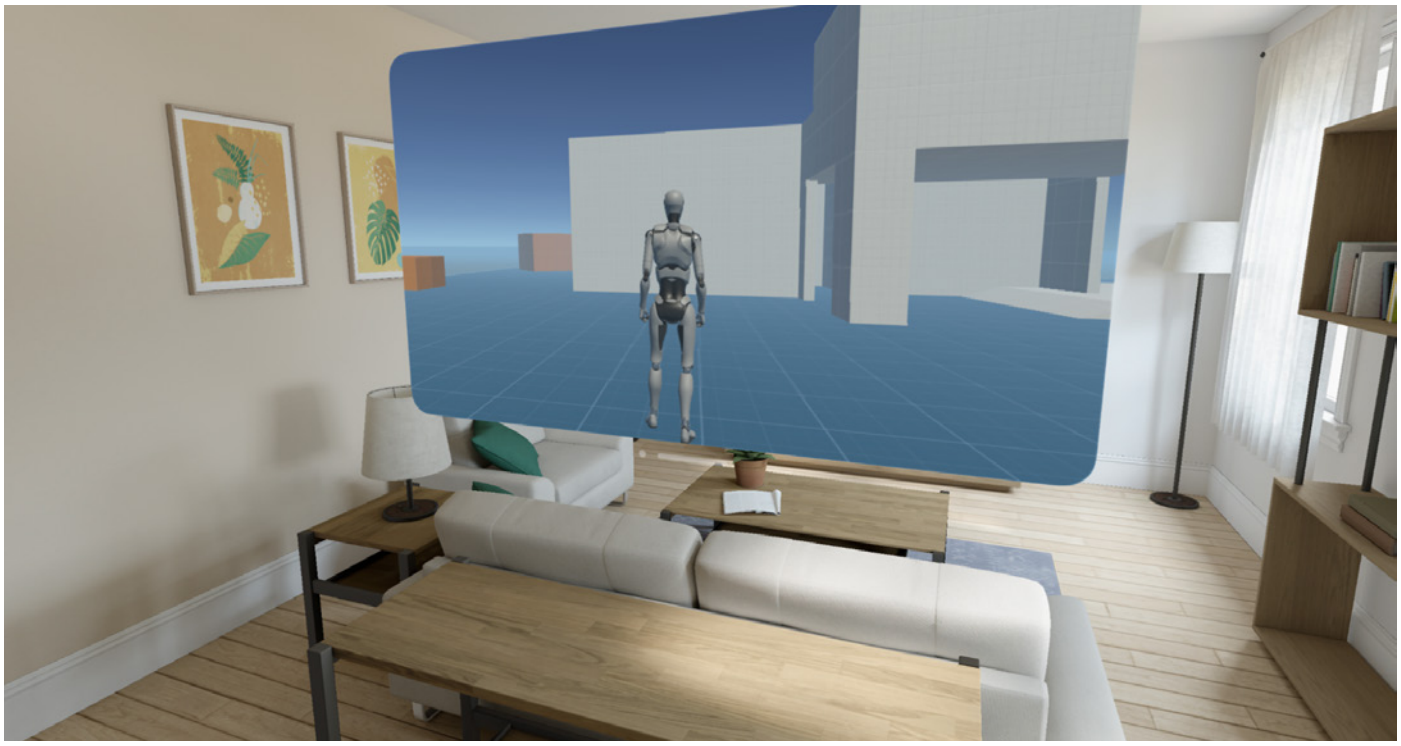
This feature is delivered through the Play to Device Host application, which can be installed on the visionOS Simulator or directly on Apple Vision Pro (via Testflight).



Install Play to Device in the Simulator by drag and drop and open it, the feature in Unity should detect the host application and it will connect to it when entering play mode (you can disable this on Connect on Play)

Windowed apps

In this mode, content is displayed within a 2D window, allowing users to resize and reposition it within the Shared Space. Like the fully immersive VR apps, Windowed apps utilize Unity rendering based on Metal, so you can expect games made for traditional 2D displays to run in Windowed mode on visionOS.



Windowed Unity application running in visionOS

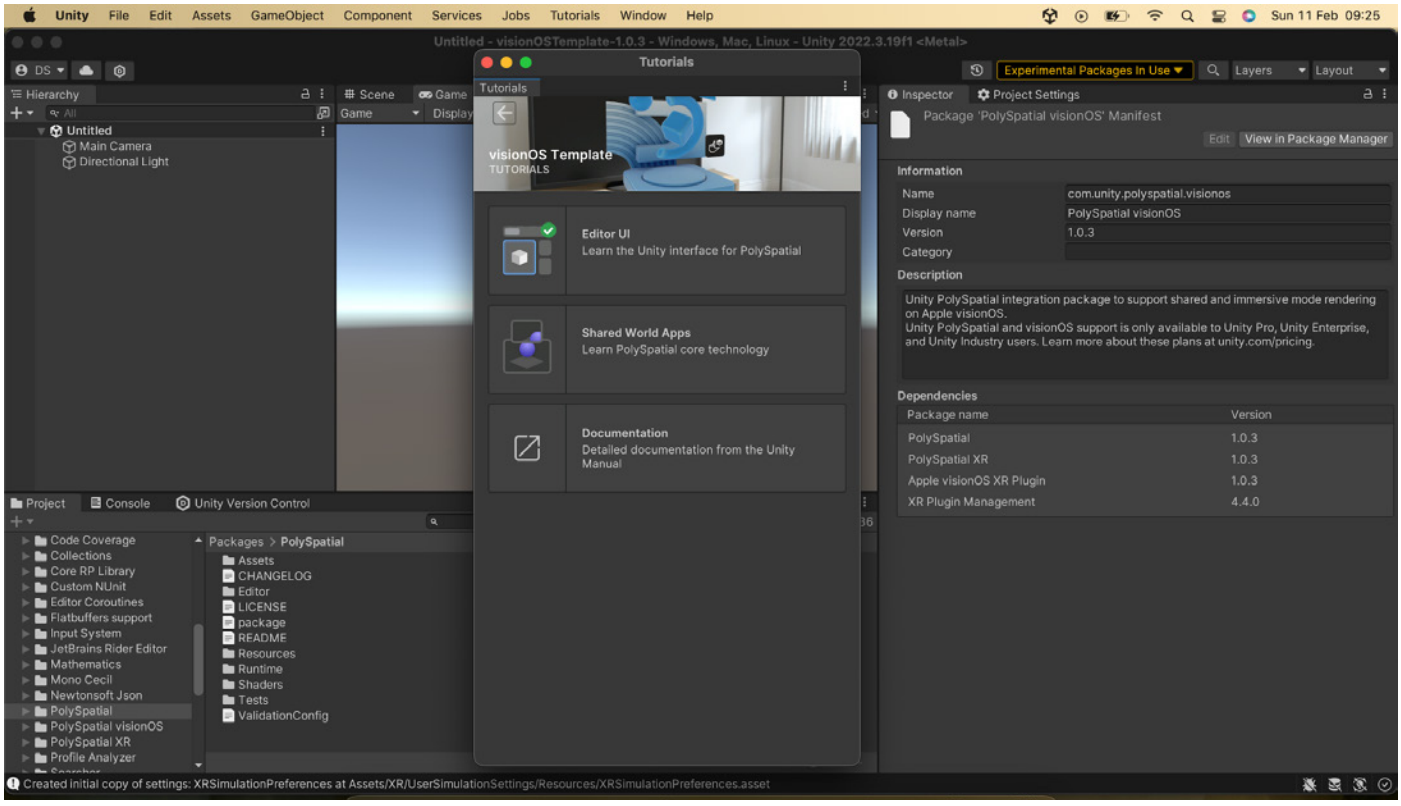
visionOS project template

A good way to start with a new project is to download the visionOS project template. You can download the template from 'Starting a new visionOS project from the Immersive App Template' where a link to the template is provided.

The visionOS template contains examples of bounded and unbounded MR experiences, responsive layout and XR input and interactions. This sample uses tapping, hand tracking, and plane detection in an unbounded volume scene.

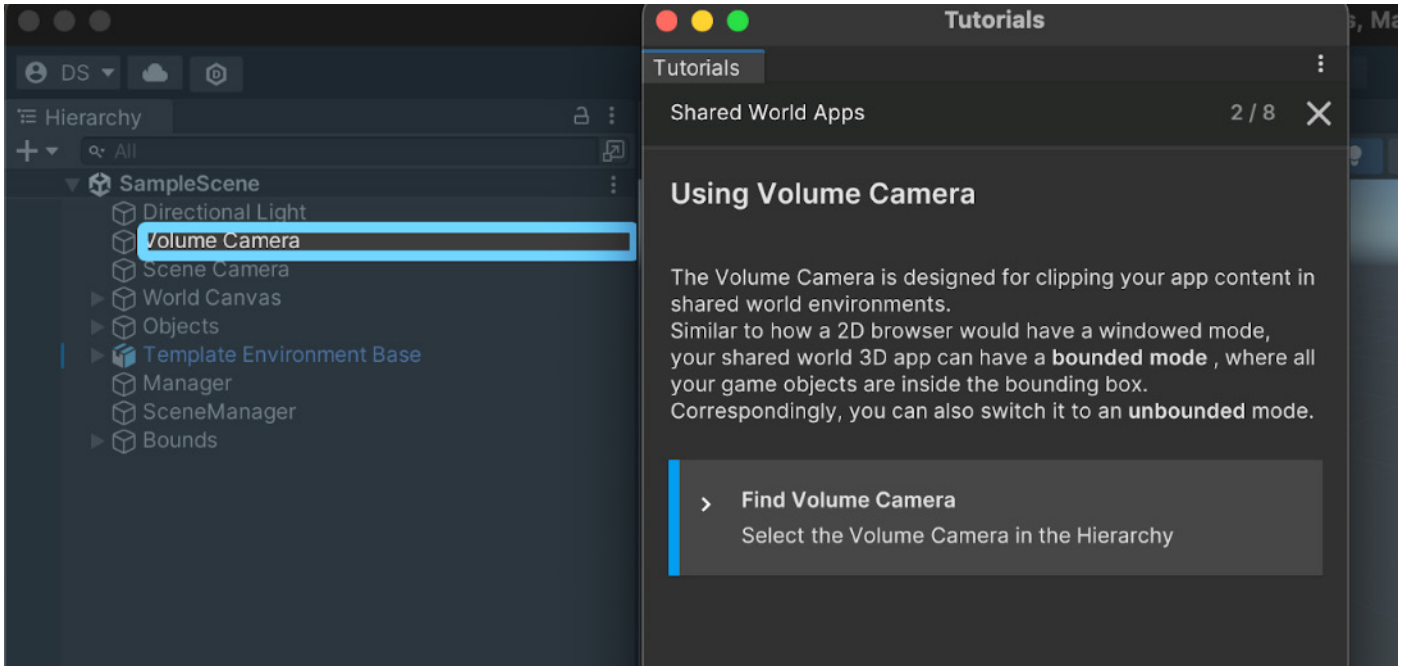
The template configures project settings, preinstalls the correct XR packages, and includes various preconfigured assets to demonstrate how to set up a project that is ready to deploy to visionOS.

You can open the template from the Unity Hub by selecting the **Add project from disk** option. Once opened you will be able to use the in-Editor tutorial; if you don't see it go to **Tutorials > Show Tutorials**.



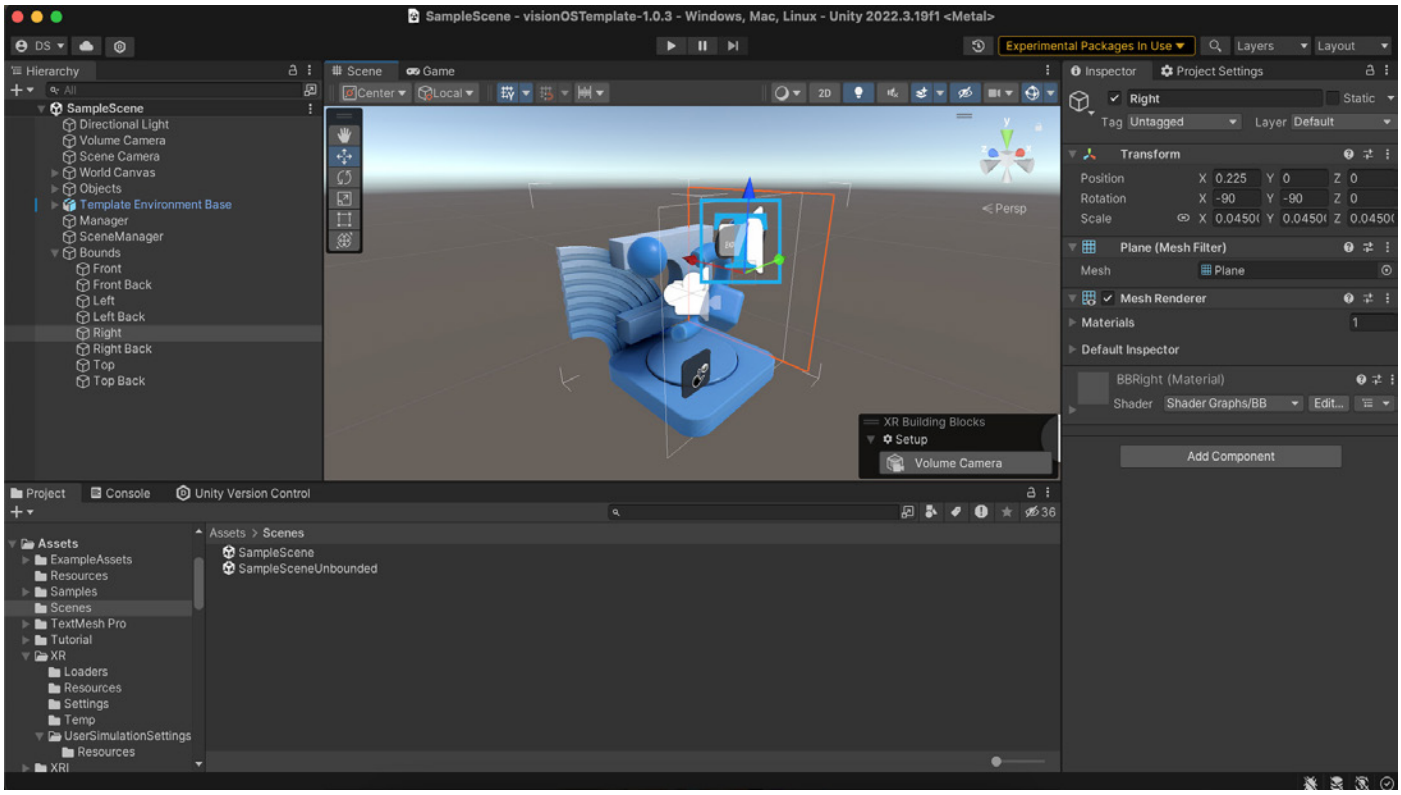
Tutorial Window giving access to various PolySpatial resources

The tutorials will take you on a guided tour of the various features of the template.



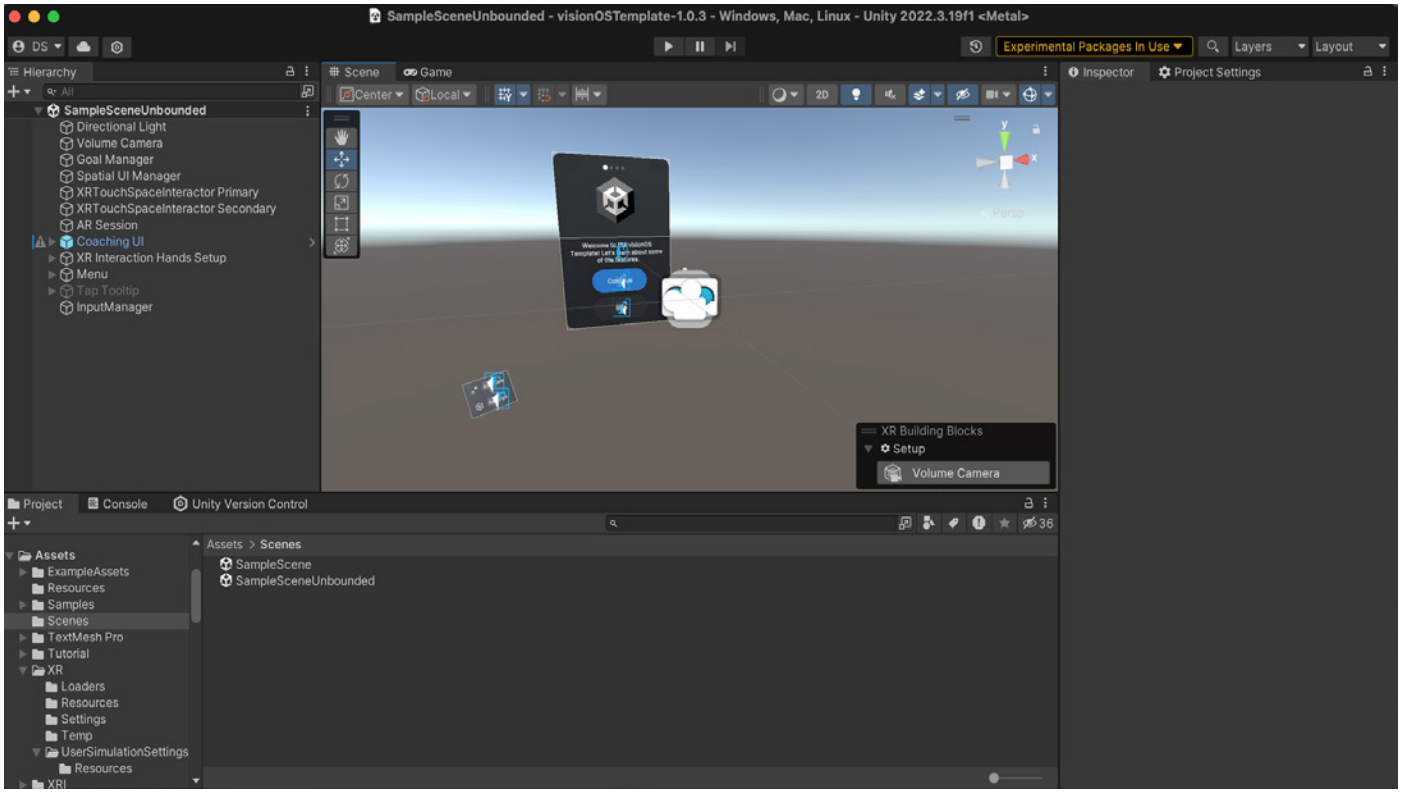
Guided tour of the project and its components

There are two sample scenes included with the project, the SampleScene, and the SampleSceneUnbounded, each showcasing the setup and capabilities of bounded and unbounded mode.



The SampleScene demonstrating the bounded MR app approach.

You can even experience these apps within the Editor by using the visionOS simulator and the Play To Device application as described previously.



The SampleSceneUnbounded, showcasing the setup of an unbounded MR app

More resources

You can connect with the Unity community to get insights and tips on XR development in Unity's Discussion space for visionOS development. For information on upcoming features, see our roadmap.

Professional training services

For those aiming at creating professional experiences in visionOS, Unity provides 1:1 live and hands-on training. This personalized training provides your team with a Unity expert who will cover:

— Your objectives and unique projects with visionOS.

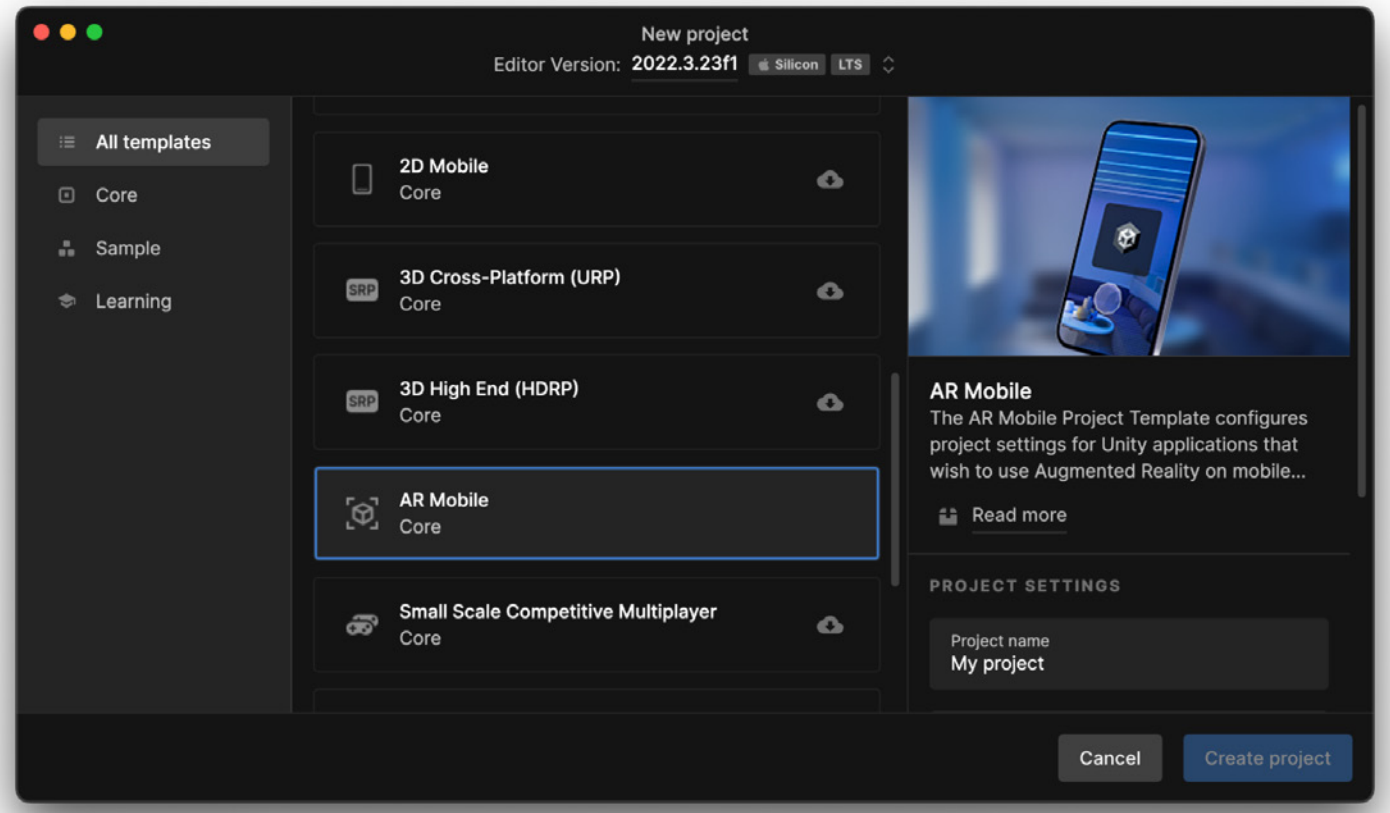
— The core concepts of developing for visionOS, and understanding Apple Vision Pro and its capabilities for your project.

— The development environment, configuration settings, and building your first application.

What about AR in Unity?

A screenshot from *Peridot,* the latest project from the popular AR developer Niantic Labs

Developing augmented reality (AR) applications is an exciting process that leverages Unity's powerful frameworks like AR Foundation which comprises AR development platforms like ARKit (for iOS) and ARCore (for Android). With a wide range of support for various platforms and extensive community resources.



A good starting point can be the AR Mobile Core template, which you can find more about here.

Glossary

A

AR Foundation: A Unity framework that allows developers to build augmented reality (AR) experiences across multiple platforms using a single API.

ASTC (adaptive scalable texture compression): A texture compression technique that helps manage the balance between quality and performance, particularly useful for mobile and VR applications.

B

Baked Lighting: Pre-calculated lighting information stored in texture maps, used to enhance performance in graphical applications by eliminating the need for real-time lighting calculations.

C

CPU usage: Measures how much processor resources an application is using. High CPU usage can impact the performance and responsiveness of the application.

Cross-platform development: The practice of developing an application that can run on multiple hardware platforms with little or no modification.

D

Draw calls: Commands sent to the GPU to render graphics; reducing the number of draw calls can significantly improve graphical performance.

DSP Load (digital signal processing load): The amount of processing power used to handle audio signals. High DSP loads can impact performance.

F

Frame rate: The number of frames displayed per second. A stable and high frame rate is crucial for smooth gameplay and VR experiences.

G

Garbage collection: The process of reclaiming memory allocated to objects that are no longer needed by the application. Frequent garbage collection can cause performance spikes.

GPU usage: Measures how much of the graphics processor's resources are being used. It's a key factor in rendering performance and visual quality.

H

HDRP (High Definition Render Pipeline): A Unity rendering pipeline designed for high-fidelity graphics on high-end hardware.

I

Input system: Unity's system for managing input from various devices, such as keyboards, mice, and game controllers, in a unified way.

Interaction Layer Mask: Used in Unity's XR Interaction Toolkit to specify which objects an interactor can interact with based on their layer.

L

LOD (Level of Detail): A technique used to improve performance by adjusting the complexity of 3D models based on their distance from the camera.

M

Memory usage: The amount of RAM an application is using. Optimizing memory usage is crucial for performance, especially in resource-constrained environments like VR.

N

Normal map: A texture type that simulates high-detail surfaces without increasing the polygon count, adding depth and detail to surfaces.

O

OpenXR: A royalty-free standard that aims to standardize the development of VR and AR applications across multiple platforms.

Optimization: The process of making an application run more efficiently by reducing resource consumption and improving performance.

P

Passthrough: A feature in VR devices that allows users to see the real world around them through the device's cameras.

Performance optimization: Techniques used to enhance the speed and efficiency of an application, crucial for maintaining high frame rates and smooth operation in VR/AR.

PolySpatial: A technology or concept related to spatial computing, emphasizing the integration and interaction of digital objects and environments within physical space in a seamless and intuitive manner. In the context of Unity, PolySpatial refers to tools, frameworks, or practices designed to facilitate the creation of such spatially integrated experiences, leveraging the platform's capabilities for real-time 3D content creation and interaction.

Q

Quality Settings: Unity settings that allow developers to adjust the graphical quality of their applications to optimize performance or visual fidelity.

R

Rendering pipeline: The process by which graphics are processed and displayed on the screen. Unity offers several rendering pipelines, including URP and HDRP.

Rigging: The process of creating the bone structure of a 3D model to enable it to move in a realistic manner.

S

SDK (Software Development Kit): A set of software tools and libraries designed to help developers create applications for a specific platform or device.

SRP (Scriptable Render Pipeline): A feature in Unity that allows developers to customize the rendering process to meet the specific needs of their project.

T

Texel density: Refers to the density of texture pixels (texels) applied to a model, affecting the visual quality and performance of the textures.

Trim sheets: Texturing technique using a single texture that includes various details, which can be applied to different parts of a 3D model to add complexity without multiple textures.

U

URP (Universal Render Pipeline): A Unity rendering pipeline optimized for performance across a wide range of devices, from mobile to high-end PCs.

User interaction: The ways in which a user interacts with a game or application, including through controllers, gestures, voice commands, and more.

V

VR (Virtual Reality): A technology that allows users to interact with a computer-generated environment, often using headsets for an immersive experience.

Velocity tracking movement: A feature in Unity's XR Interaction Toolkit that calculates an object's movement based on the velocity of the interactor, allowing for realistic physics interactions.

X

XR (Extended Reality): A term encompassing all forms of computer-altered reality, including augmented reality (AR), virtual reality (VR), and mixed reality (MR).

XR Interaction Toolkit: A Unity package that provides a set of tools and components for building interactive XR applications.


Unity®