Article

Bringing your ARKit app to visionOS

Update an iPadOS or iOS app that uses ARKit, and provide an equivalent experience in visionOS.

Overview

If you use ARKit to create an augmented reality experience on iPhone or iPad, you need to rethink your use of that technology when bringing your app to visionOS. ARKit plays a crucial role in delivering your content to the display in iPadOS and iOS. In visionOS, you use ARKit only to acquire data about the person's surroundings, and you do so using a different set of APIs.

In visionOS, you don't need a special view to display an augmented reality interface. Build windows with your app's content using SwiftUI or UIKit. When you display those windows, visionOS places them in the person's surroundings for you. If you want to control the placement of any 2D or 3D content in the person's surroundings, build your content using SwiftUI and RealityKit.

When migrating your app to visionOS, reuse as much of your app's existing content as you can. visionOS supports most of the same technologies as iOS, so you can reuse project assets, 3D models, and most custom views. Don't reuse your app's ARKit code or any code that relies on technologies visionOS doesn't support.

For general guidance on how to port apps to visionOS, see Bringing your existing apps to visionOS.

Adopt technologies available in both iOS and visionOS

To create a single app that runs in both iOS and visionOS, use technologies that are available on both platforms. While ARKit in iOS lets you create your interface using several different technologies, the preferred technologies in visionOS are SwiftUI and RealityKit. If you're not currently using RealityKit for 3D content, consider switching to it before you start adding visionOS support. If you retain code that uses older technologies in your iOS app, you might need to re-create much of that code using RealityKit when migrating to visionOS.

If you use Metal to draw your app's content, you can bring your code to visionOS to create content for 2D views or to create fully immersive experiences. You can't use Metal to create 3D content that integrates with the person's surroundings. This restriction prevents apps from sampling pixels of the person's surroundings, which might contain sensitive information. For information on how to create a fully immersive experience with Metal, see Drawing fully immersive content using Metal.

> **Note**
>
> In visionOS 2.0 and later, you can create a mixed immersive experience with Metal as well. Watch Render Metal with passthrough in visionOS for more information and sample code.

Convert 3D assets to the USDZ format

The recommended format for 3D assets in iOS and visionOS is USDZ. This format offers a compact single file for everything, including your models, textures, behaviors, physics, anchoring, and more. If you have assets that don't use this format, use the Reality Converter tool that comes with Xcode to convert them for your project.

When building 3D scenes for visionOS, use Reality Composer Pro to create your scenes that incorporate your USDZ assets. With Reality Composer Pro, you can import your USD files and edit them in place, nondestructively. If your iOS app applies custom materials to your assets, convert those materials to shader graphs in the app.

Although you can bring models and materials to your project using USDZ files, you can't bring custom shaders you wrote using Metal. Replace any custom shader code with MaterialX shaders. Many digital content creation tools support the MaterialX standard, and let you create dynamic shaders and save them with your USDZ files. Reality Composer Pro and RealityKit support MaterialX shaders, and incorporate them with your other USDZ asset content. See the MaterialX site for more information.

Update your interface to support visionOS

In visionOS, you manage your app's content, and the system handles the integration of that content with the person's surroundings. This approach differs from iOS, where you use a special ARKit view to blend your content and the live camera content. Bringing your interface to visionOS therefore means you need to remove this special ARKit view and focus only on your content.

If you can display your app's content using SwiftUI or UIKit views, build a window with those views and present it from your visionOS app. If you use other technologies to incorporate 2D or 3D content into the person's surroundings, make the following substitutions in the visionOS version of your app.

If you create your AR experience using:	Update to:
RealityKit and `ARView`	RealityKit and `RealityView`
SceneKit and `ARSCNView`	RealityKit and `RealityView`
SpriteKit and `ARSKView`	RealityKit or SwiftUI

A `RealityView` is a SwiftUI view that manages the content and animations you create using RealityKit and Reality Composer Pro. You can add a `RealityView` to any of your app's windows to display 2D or 3D content. You can also add the view to an `ImmersiveSpace` scene, which you use to integrate your RealityKit content into the person's surroundings.

> **Note**
>
> You can load iOS storyboards into a visionOS app, but you can't customize your interface for visionOS or include 3D content. If you want to share interface files between iOS and visionOS, adopt SwiftUI views or create your interface programmatically.

For more information about how to use `RealityView` and respond to interactions with your content, see Adding 3D content to your app.

Replace your ARKit code

ARKit provides different APIs for iOS and visionOS, and the way you use ARKit services on the platforms is also different. In iOS, you must use ARKit to put your content onscreen, and you can also use it to manage interactions between your content and a person's surroundings. In visionOS, the system puts your content onscreen, so you only use ARKit to manage interactions with the surroundings. Because of this more limited usage, some apps don't need ARKit at all in visionOS.

The only time you use ARKit in visionOS is when you need one of the following services:

- Plane detection
- Image tracking
- Scene reconstruction
- Hand tracking
- World tracking and device-pose prediction

Use plane detection, image tracking, and scene reconstruction to facilitate interactions between your app's virtual content and real-world items. For example, use plane detection to detect a

tabletop on which to place your content. Use world tracking to record anchors that you want to persist between launches of your app. Use hand tracking if your app requires custom hand-based input.

> **Important**
>
> In visionOS 2.0 and later, use `SpatialTrackingSession` for available AR data instead.

To start ARKit services in your app, create an `ARKitSession` object and run it with the data providers for each service. Unlike ARKit in iOS, services in visionOS are independent of one another, and you can start and stop each one at any time. The following example shows how to detect horizontal and vertical planes. Data providers deliver new information using an asynchronous sequence.

```
let session = ARKitSession()
let planeData = PlaneDetectionProvider(alignments: [.horizontal, .vertical])

Task {
    try await session.run([planeData])

    for await update in planeData.anchorUpdates {
        switch update.event {
        case .added, .updated:
            // Update plane representation.
            print("Updated planes.")
        case .removed:
            // Indicate plane removal.
            print("Removed plane.")
        }
    }
}
```

If you use the world-tracking data provider in visionOS, ARKit automatically persists the anchors you add to your app's content. You don't need to persist these anchors yourself.

For more information about how to use ARKit, see ARKit.

Isolate ARKit features not available in visionOS

If your app uses ARKit features that aren't present in visionOS, isolate that code to the iOS version of your app. The following features are available in iOS, but don't have an equivalent in visionOS:

- Face tracking
- Body tracking
- Geotracking and placing anchors using a latitude and longitude
- Object detection
- App Clip Code detection
- Video frame post-processing

Although whole body tracking isn't available in visionOS, you can track the hands of the person wearing the device. Hand gestures are an important way of interacting with content in visionOS. SwiftUI handles common types of interactions like taps and drags, but you can use custom hand tracking for more complex gestures your app supports.

If you use ARKit raycasting in iOS to detect interactions with objects in the person's surroundings, you might not need that code in visionOS. SwiftUI and RealityKit handle both direct and indirect interactions with your app's content in 3D space, eliminating the need for raycasting in many situations. In other situations, you can use the features of ARKit and RealityKit to manage interactions with your content. For example, you might use ARKit hand tracking to determine where someone is pointing in the scene, and use scene reconstruction to build a mesh you can integrate into your RealityKit content.

See Also

iOS migration and compatibility

Determining whether to bring your app to visionOS

Decide whether to bring your existing iPadOS or iOS app to visionOS.

Bringing your existing apps to visionOS

Build a version of your iPadOS or iOS app using the visionOS SDK, and update your code for platform differences.

Making your existing app compatible with visionOS

Modify your iPadOS or iOS app to run successfully in visionOS as a compatible app.