Sample Code

Exploring object tracking with ARKit

Find and track real-world objects in visionOS using reference objects trained with Create ML.

Download

visionOS 2.0+ | Xcode 16.0+

Overview

The sample app demonstrates how to use a reference object to discover and track a specific object in a person's surroundings in visionOS. This capability allows you to create engaging experiences based on objects in a person's surroundings and lets you attach digital content to these objects. For example, you can build an app that uses reference objects to describe the specific assembly of a machine a person is testing or repairing. Using this reference model, when ARKit recognizes that object, you can attach digital content to it, such as a diagram of the device, more information about its function, and so on.

This sample includes a reference object that ARKit uses to recognize a Magic Keyboard in someone's surroundings.

> Note
>
> This sample code project is associated with WWDC24 Session 10101 — Explore object tracking for visionOS.

Configure the sample code project

Note

This app requires Xcode 16 and visionOS 2 or later, and an Apple Vision Pro. Object tracking isn't supported in the visionOS simulator.

1. In the project's settings, select Signing and Capabilities.
2. Select your team name from the drop-down menu.
3. Pair Xcode with your device wirelessly or using the developer strap.
4. Click Run or press Command-R to launch the app.

Import the reference object to track a specific object

Object tracking demonstrates two methods for importing a reference object into the app. The first loads reference objects directly from the app's bundle, as shown here:

```
func loadBuiltInReferenceObjects() async {
    // Only allow one loading operation at any given time.
    guard !didStartLoading else { return }
    didStartLoading.toggle()

    print("Looking for reference objects in the main bundle ...")

    // Get a list of all reference object files in the app's main bundle and attempt
    var referenceObjectFiles: [String] = []
    if let resourcesPath = Bundle.main.resourcePath {
        try? referenceObjectFiles = FileManager.default.contentsOfDirectory(atPath:
    }

    fileCount = referenceObjectFiles.count
    updateProgress()

    await withTaskGroup(of: Void.self) { group in
        for file in referenceObjectFiles {
            let objectURL = Bundle.main.bundleURL.appending(path: file)
            group.addTask {
                await self.loadReferenceObject(objectURL)
                await self.finishedOneFile()
            }
        }
    }
}
```

In the second method, a person can provide a URL for a reference object file through a file importer dialog:

```
.fileImporter(isPresented: $fileImporterIsOpen, allowedContentTypes: [referenceObjec
    switch results {
    case .success(let fileURLs):
        Task {
            // Try to load each selected file as a reference object.
            for fileURL in fileURLs {
                guard fileURL.startAccessingSecurityScopedResource() else {
                    print("Failed to get sandboxed access to the file \(fileURL)")
                    return
                }
                await appState.referenceObjectLoader.addReferenceObject(fileURL)
                fileURL.stopAccessingSecurityScopedResource()
            }
        }
    case .failure(let error):
        print("Failed to open file with error: \(error)")
    }
}
```

Run object tracking on a session

To start receiving events, create an `ObjectTrackingProvider` that you initialize with a reference object, and then start an `ARKitSession` with the `ObjectTrackingProvider` you created, as shown below:

```
func startTracking() async -> ObjectTrackingProvider? {
    let referenceObjects = referenceObjectLoader.enabledReferenceObjects

    guard !referenceObjects.isEmpty else {
        fatalError("No reference objects to start tracking")
    }

    // Run a new provider every time when entering the immersive space.
    let objectTracking = ObjectTrackingProvider(referenceObjects: referenceObjects)
    do {
        try await arkitSession.run([objectTracking])
    } catch {
        print("Error: \(error)" )
        return nil
    }
    self.objectTracking = objectTracking
    return objectTracking
}
```

Handle adding, updating, removing, and visualizing objects

ARKit delivers an asynchronous stream of updates as it detects changes in the scene. Your app needs to process these as they arrive and update the scene in response. The example below demonstrates handling these events inside the app's `RealityView`:

```
Task {
    let objectTracking = await appState.startTracking()
    guard let objectTracking else {
        return
    }

    // Wait for object anchor updates and maintain a dictionary of visualizations
    // that are attached to those anchors.
    for await anchorUpdate in objectTracking.anchorUpdates {
        let anchor = anchorUpdate.anchor
        let id = anchor.id

        switch anchorUpdate.event {
        case .added:
            // Create a new visualization for the reference object that ARKit just c
            // The app displays the USDZ file that the reference object was trained 
            // a wireframe on top of the real-world object, if the .referenceobject 
            // that USDZ file. If the original USDZ isn't available, the app display
            let model = appState.referenceObjectLoader.usdzsPerReferenceObjectID[anc
            let visualization = ObjectAnchorVisualization(for: anchor, withModel: mo
            self.objectVisualizations[id] = visualization
            root.addChild(visualization.entity)
        case .updated:
            objectVisualizations[id]?.update(with: anchor)
        case .removed:
            objectVisualizations[id]?.entity.removeFromParent()
            objectVisualizations.removeValue(forKey: id)
        }
    }
}
```

When the app adds objects to the scene, it attaches virtual content to the reference object using the `ObjectAnchorVisualization` entity to render a wireframe that shows the reference object's outline.

Create your own reference objects

Creating your own reference objects requires an iPhone, iPad, or other device that you can use to create high-fidelity scans of the physical object you want to model, and a Mac with an M2 chip or later to process the images and create a reference object using Create ML.

See Also

ARKit

{} Happy Beam

Leverage a Full Space to create a fun game using ARKit.

Setting up access to ARKit data

Check whether your app can use ARKit and respect people's privacy.

{} Incorporating real-world surroundings in an immersive experience

Create an immersive experience by making your app's content respond to the local shape of the world.

{} Placing content on detected planes

Detect horizontal surfaces like tables and floors, as well as vertical planes like walls and doors.

{} Tracking specific points in world space

Retrieve the position and orientation of anchors your app stores in ARKit.

Tracking preregistered images in 3D space

Place content based on the current position of a known image in a person's surroundings.

{} Object tracking with Reality Composer Pro experiences

Use object tracking in visionOS to attach digital content to real objects to create engaging experiences.

{} Building local experiences with room tracking

Use room tracking in visionOS to provide custom interactions with physical spaces.

{} Placing entities using head and device transform

Query and react to changes in the position and rotation of Apple Vision Pro.