Sample Code

Saving and loading world data

Serialize a world-tracking session to resume it later on.

Download

iOS 12.0+ | iPadOS 12.0+ | Xcode 16.0+

Overview

This sample app demonstrates a simple AR experience for iOS 12 devices. Before exploring the code, try building and running the app to familiarize yourself with the user experience it demonstrates:

1. Run the app. You can look around and tap to place a virtual 3D object on real-world surfaces. (Tap again to relocate the object.)
2. After you've explored the environment, the Save Experience button becomes available. Tap it to save ARKit's world-mapping data to local storage.
3. Tap the Load Experience button. (You can do this immediately, or after quitting and relaunching the app, even if the app has been terminated in the background.)
4. While ARKit attempts to resume an AR session from the saved world-mapping data, the app displays a snapshot of the camera view from the time that data was saved. For best results, move the device so that the camera view matches the screenshot.

Follow the steps below to see how this app uses the `ARWorldMap` class to save and restore ARKit's spatial mapping state.

Getting started

Requires Xcode 10.0, iOS 12.0 and an iOS device with A9 or later processor.

Run the AR session and place AR content

This app extends the basic workflow for building an ARKit app. (For details, see Tracking and visualizing planes.) It defines an `ARWorldTrackingConfiguration` with plane detection enabled, then runs that configuration in the `ARSession` attached to the `ARSCNView` that displays the AR experience.

When `UITapGestureRecognizer` detects a tap on the screen, the `handleSceneTap` method uses ARKit hit-testing to find a 3D point on a real-world surface, then places an `ARAnchor` marking that position. When ARKit calls the delegate method `renderer(_:didAdd:for:)`, the app loads a 3D model for `ARSCNView` to display at the anchor's position.

Capture and save the AR world map

An `ARWorldMap` object contains a snapshot of all the spatial mapping information that ARKit uses to locate the user's device in real-world space. To save a map that can reliably be used for restoring your AR session later, you'll first need to find a good time to capture the map.

ARKit provides a `worldMappingStatus` value that indicates whether it's currently a good time to capture a world map (or if it's better to wait until ARKit has mapped more of the local environment). This app uses that value to provide visual feedback and choose when to make the Save Experience button available:

```
// Enable Save button only when the mapping status is good and an object has been pl
switch frame.worldMappingStatus {
case .extending, .mapped:
    saveExperienceButton.isEnabled =
        virtualObjectAnchor != nil && frame.anchors.contains(virtualObjectAnchor!)
default:
    saveExperienceButton.isEnabled = false
}
statusLabel.text = """
Mapping: \(frame.worldMappingStatus.description)
Tracking: \(frame.camera.trackingState.description)
"""
```

When the user taps the Save Experience button, the app calls `getCurrentWorldMap(completionHandler:)` to capture the map from the running ARSession, then serializes it to a `Data` object with `NSKeyedArchiver` and writes it to local storage:

```
sceneView.session.getCurrentWorldMap { worldMap, error in
    guard let map = worldMap
```

```
        else { self.showAlert(title: "Can't get current world map", message: error!.

    // Add a snapshot image indicating where the map was captured.
    guard let snapshotAnchor = SnapshotAnchor(capturing: self.sceneView)
        else { fatalError("Can't take snapshot") }
    map.anchors.append(snapshotAnchor)

    do {
        let data = try NSKeyedArchiver.archivedData(withRootObject: map, requiringSe
        try data.write(to: self.mapSaveURL, options: [.atomic])
        DispatchQueue.main.async {
            self.loadExperienceButton.isHidden = false
            self.loadExperienceButton.isEnabled = true
        }
    } catch {
        fatalError("Can't save map: \(error.localizedDescription)")
    }
}
```

To help a user resume the AR experience from this map later, the app also captures a snapshot of the camera view with the example `SnapshotAnchor` class and stores it in the world map.

Load and relocalize to a saved map

When the app launches, it checks local storage for a world map file it may have saved in an earlier session:

```
do {
    guard let worldMap = try NSKeyedUnarchiver.unarchivedObject(ofClass: ARWorldMap.
        else { fatalError("No ARWorldMap in archive.") }
    return worldMap
} catch {
    fatalError("Can't unarchive ARWorldMap from file data: \(error)")
}
```

If that file exists and can be deserialized as an `ARWorldMap` object, the app makes its Load Experience button available. When you tap the button, the app tells ARKit to attempt resuming the session captured in that world map, by creating and running an `ARWorldTrackingConfiguration` using that map as the `initialWorldMap`:

```
let configuration = self.defaultConfiguration // this app's standard world tracking
configuration.initialWorldMap = worldMap
sceneView.session.run(configuration, options: [.resetTracking, .removeExistingAnchor
```

ARKit then attempts to *relocalize* to the new world map—that is, to reconcile the received spatial-mapping information with what it senses of the local environment. This process is more likely to succeed if the user moves to areas of the local environment that they visited during the previous session. To help the user successfully resume the saved experience, this app uses the example `SnapshotAnchor` class to save a camera image in the world map, then displays that image while ARKit is relocalizing.

Restore AR content after relocalization

Saving a world map also archives all anchors currently associated with the AR session. After you successfully run a session from a saved world map, the session contains all anchors previously saved in the map. You can use saved anchors to restore virtual content from a previous session.

In this app, after relocalizing to a previously saved world map, the virtual object placed in the previous session automatically appears at its saved position. The same `ARSCNView` delegate method `renderer(_:didAdd:for:)` fires both when you directly add an anchor to the session and when the session restores anchors from a world map. To determine which saved anchor represents the virtual object, this app uses the `ARAnchor` `name` property.

```
func renderer(_ renderer: SCNSceneRenderer, didAdd node: SCNNode, for anchor: ARAnch
    guard anchor.name == virtualObjectAnchorName
        else { return }

    // save the reference to the virtual object anchor when the anchor is added from
    if virtualObjectAnchor == nil {
        virtualObjectAnchor = anchor
    }
    node.addChildNode(virtualObject)
}
```

In your own AR experience, you can choose among various techniques for restoring virtual content associated with saved anchors. For example:

- An app for visualizing furniture from a fixed catalog might save an identifier for each placed object in the corresponding anchor's `name`, then use that identifier to determine which 3D model to display when resuming a session from a saved map.

- A game that places virtual characters to play in the user’s environment might create various custom `ARAnchor` subclasses to store gameplay data specific to each character, so that resuming a session from a saved map also restores the state of the game. (See `ARAnchor` Subclassing Notes and `ARAnchorCopying`.)

See Also

World Data

`class ARWorldMap`

The state in a world-tracking AR session during which a device maps the user’s position in physical space and proximity to anchor objects.