Sample Code

Building local experiences with room tracking

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

Download

visionOS 26.0+ | Xcode 26.0+

Overview

This sample allows your app to keep track of rooms as discrete, identifiable places, and enables you to provide a customized virtual experience inside a specific room, and to get notified when someone enters or leaves the room. These customizations can be as simple as knowing when to stop room-specific animations, or to support the creation of location-specific virtual content such as in-game treasures, effects, or even portals to virtual worlds that contain other content.

This sample demonstrates how to use room tracking by enabling a person to place spheres in a space and continuously query the framework as to whether those spheres are in the same room as the person. As someone moves into, through, and out of the room, ARKit delivers `RoomAnchor` updates that represent the latest knowledge of the current room. This structure provides a `contains(_:)` query method that you use to determine if the spheres are in the current room, and highlight them accordingly.

The app has an *occlusion mode*, in which the room geometry the framework renders is a transparent occluder that hides virtual objects outside the room. It also has a *wall selection mode*, in which someone may select a specific wall for the purpose of replacing it with a video or virtual portal.

> **Note**
>
> This app requires Xcode 16 and visionOS 2 or later, and an Apple Vision Pro. ARKit room tracking isn't supported in Simulator.

Ensure all data providers are in an authorized state

Your app must request permission to use certain visionOS capabilities before being able to access data associated with them. For example, attempting to access the `RoomTrackingProvider` displays a permission sheet asking the user to authorize your app's access. If the user has previously denied this request, the app displays an error message in the scene. For information about using a `RoomTrackingProvider`, see Setting up access to ARKit data. For information about best practices for privacy, see Adopting best practices for privacy and user preferences.

> **Note**
>
> To use the room tracking capabilities in visionOS in your app, you need to provide the `NSWorldSensingUsageDescription` key in your app's `Info.plist` along with a description of why your app uses this feature. This sample already provides this key and description.

```
func areAllDataProvidersAuthorized() async -> Bool {
    // It's sufficient to check that the authorization status isn't `denied`.
    // If it's `notdetermined`, ARKit presents a permission pop-up menu that appears
    // as the session runs.
    let authorization = await ARKitSession().queryAuthorization(for: [.worldSensing]
    return authorization[.worldSensing] != .denied
}
```

Configure room tracking

Set up room tracking by first configuring an `ARKitSession` instance, then add a `WorldTrackingProvider` and a `RoomTrackingProvider` to the session as shown in the following example:

```
private let session = ARKitSession()
private let worldTracking = WorldTrackingProvider()
private let roomTracking = RoomTrackingProvider()
```

In addition to instantiating the world and room tracking providers in the `AppState`, you need to create storage for the in-room anchors the app tracks:

```
/// A dictionary that contains `RoomAnchor` structures.
private var roomAnchors = [UUID: RoomAnchor]()
/// A dictionary that contains `WorldAnchor` structures.
private var worldAnchors = [UUID: WorldAnchor]()
/// A dictionary that contains `ModelEntity` structures for spheres.
private var sphereEntities = [UUID: ModelEntity]()
/// A dictionary that contains `ModelEntity` structures for room anchors.
private var roomEntities = [UUID: ModelEntity]()
```

You also need to create the materials the framework uses to render the in-room anchors:

```
// Material for spheres in the current room.
private let inRoomSphereMaterial = SimpleMaterial(color: .green, roughness: 0.2, isM
// Material for spheres not in the current room.
private let outOfRoomSphereMaterial = SimpleMaterial(color: .red, roughness: 0.2, is
// Material the app applies to room entities to show occlusion effects.
private let occlusionMaterial = OcclusionMaterial()
// Material for current room walls.
private var wallMaterial = UnlitMaterial(color: .blue)
```

Allow a person to place room tracking anchors

Placing a `roomAnchor` object in the room consists of two processes. The first phase allows the person to review the anchor, which the sample renders as a sphere in front of the device from the person's perspective:

```
private func createPreviewSphere() -> ModelEntity {
    let sphereMesh = MeshResource.generateSphere(radius: 0.1)
    let sphereMaterial = SimpleMaterial(color: .gray.withAlphaComponent(0.5), roughn
    let sphere = ModelEntity(mesh: sphereMesh, materials: [sphereMaterial])

    // Enables gestures on the preview sphere.
    // Looking at the preview and using a pinch gesture causes a world anchored sphe
    sphere.generateCollisionShapes(recursive: false, static: true)
    // Ensures the preview only accepts indirect input (for tap gestures).
    sphere.components.set(InputTargetComponent(allowedInputTypes: [.indirect]))
```

```
    // The preview sphere only becomes visible once someone clicks the Add a Sphere
    sphere.isEnabled = false

    return sphere
}
```

The second phase allows a person to place the sphere (a `WorldAnchor`) in their surroundings with a tap gesture. Gestures such as this are SwiftUI view modifiers you apply to the room's `View`, as shown below:

```
.gesture(SpatialTapGesture().targetedToAnyEntity().onEnded { event in
    if event.entity == previewSphere {
        Task {
            // To place a sphere you need to:
            // 1. Create a world anchor with the translation of that offset transfor
            // 2. Create the sphere's geometry in `processWorldTrackingUpdates()` af
            await appState.addWorldAnchor(at: event.entity.transformMatrix(relativeT
            appState.showPreviewSphere = false
        }
    }
})
```

As a person places spheres in the room, they appear in green to indicate they're anchors in the current room. If a person leaves the room, all of the room anchors in the previous room dim and become red to indicate a person has left the room. If there are anchors in the room a person enters into, they change color to indicate the person is currently in the room.

This changing state and the property of a room being *current* is what allows an app to make decisions about what actions, animations, or other processes make sense in a specific location.

Check the current room and respond to updates

As a person moves from room to room, ARKit's room tracking process checks to see which room is current and reports back changes to the app through the `RoomTrackingProvider` property `anchorUpdates`, which is an asynchronous sequence of all anchor updates. As these updates come in, a `Task` view modifier in the app's `WorldAndRoomView` calls a method that looks for anchors to update, as demonstrated here:

```
func processRoomTrackingUpdates() async {
    for await update in roomTracking.anchorUpdates {
        let roomAnchor = update.anchor
```

```
        switch update.event {
        case .removed:
            if roomAnchor.isCurrentRoom {
                colliderWallsRoot.children.removeAll()
                if let currentRenderedWall {
                    renderWallRoot.removeChild(currentRenderedWall)
                }
            }
            roomAnchors.removeValue(forKey: roomAnchor.id)
            roomEntities[roomAnchor.id]?.removeFromParent()
            roomEntities.removeValue(forKey: roomAnchor.id)
            updateSphereState()
        case .added, .updated:
            roomAnchors[roomAnchor.id] = roomAnchor
            guard let roomMeshResource = roomAnchor.geometry.asMeshResource() else {
            if update.event == .added {
                let roomEntity = ModelEntity(mesh: roomMeshResource, materials: [occ
                roomEntity.transform = Transform(matrix: roomAnchor.originFromAnchor
                roomEntities[roomAnchor.id] = roomEntity
                roomEntity.isEnabled = roomAnchor.isCurrentRoom
                roomRoot.addChild(roomEntity)

            } else if update.event == .updated {
                guard let roomEntity = roomEntities[roomAnchor.id] else { continue }
                roomEntity.model?.mesh = roomMeshResource
                roomEntity.transform = Transform(matrix: roomAnchor.originFromAnchor
                roomEntity.isEnabled = roomAnchor.isCurrentRoom
            }

            updateSphereState()

            if roomAnchor.isCurrentRoom {
                currentRoomID = roomAnchor.id
                if renderWallRoot.isEnabled {
                    await updateCurrentRoomWalls(for: roomAnchor)
                }
            }
        }
    }
}
```

Find and select walls

Room tracking also enables someone to find and select walls in the current room. You can use this as an additional interaction surface, such as creating a “portal” to another virtual space. The process of selecting a wall in a room is split into two modes: an *unlocked mode* where actively looking at a specific wall causes ARKit to highlight it in blue, and *locked mode* where a person has selected a wall and it receives continuous updates from the `RoomTrackingProvider`. The *unlocked mode* requires performing a ray cast query in the direction of the a person’s head, which returns the first wall that it hits, as shown here:

```
/// Updates the wall in front of the person when a wall isn't in a selected state.
func updateFacingWall() {
    guard renderWallRoot.isEnabled && !isWallSelectionLocked else {
        return
    }
    // Update within 10 m.
    let distance: Float = 10

    let deviceAnchor = worldTracking.queryDeviceAnchor(atTimestamp: CACurrentMediaTi
    guard let deviceAnchor, deviceAnchor.isTracked == true else {
        return
    }
    let deviceInOriginCoordinates = deviceAnchor.originFromAnchorTransform

    let lookAtPointInDeviceCoordinate = SIMD4<Float>(0, 0, -distance, 1)
    let lookAtPointInOriginCoordinates = deviceInOriginCoordinates * lookAtPointInDe

    guard let scene = colliderWallsRoot.scene else {
        logger.error("Failed to find the scene of `colliderWallsRoot`.")
        return
    }

    let hitWall = scene.raycast(from: deviceInOriginCoordinates.columns.3.xyz, to: 1

    guard !hitWall.isEmpty else {
        return
    }
    // Render the first hit wall.
    renderWallRoot.children.removeAll()

    let hitEntity = hitWall[0].entity
    currentRenderedWall = hitEntity
    renderWallRoot.addChild(hitEntity)
}
```

```
/// Updates walls under the collider walls root.
///
/// If someone has chosen and locked a wall, this method updates and renders that wa
/// If someone hasn't locked a wall, the method updates and renders the wall in fron
/// them  in `WorldAndRoomView` at a rate of 10 Hz.
private func updateCurrentRoomWalls(for roomAnchor: RoomAnchor) async {
    let newColliderWalls = Entity()
    let wallGeometries = roomAnchor.geometries(of: .wall)
    for wallGeometry in wallGeometries {
        guard let wallMeshResource = wallGeometry.asMeshResource() else {
            continue
        }

        let wallEntity = ModelEntity(mesh: wallMeshResource, materials: [wallMateria
        wallEntity.transform = Transform(matrix: roomAnchor.originFromAnchorTransfor

        guard let shape = try? await ShapeResource.generateStaticMesh(from: wallMesh
            logger.error("Failed to create ShapeResource from wall geometries.")
            continue
        }

        wallEntity.collision = CollisionComponent(shapes: [shape], isStatic: true)
        newColliderWalls.addChild(wallEntity)
    }
    // Clear old walls.
    colliderWallsRoot.children.removeAll()
    colliderWallsRoot.addChild(newColliderWalls)

    if isWallSelectionLocked {
        let wallCandidateEntities = Array(newColliderWalls.children)
        updateLockedWall(wallCandidateEntities: wallCandidateEntities)
    }
}
```

Keep focus on the current room

Room tracking operates only in the current room a person is in. If someone leaves one room and enters another, the previous room is no longer valid, and the framework only updates mesh-room associations and plane-room associations for the current room. Only use the current room anchor and discard any noncurrent rooms.

Be aware of limitations

Clutter in a room, large furniture elements, and very large spaces may interfere with ARKit's ability to accurately detect walls and fully detect the dimensions of a room. In the case of very large indoor spaces, or in rooms with low-light conditions, the framework may only provide a floor mesh. Additionally, visionOS doesn't support using room tracking outdoors or when Apple Vision Pro is in Travel Mode. In these cases, there's no current room. For more information on implementing immersive experiences, see Human Interface Guidelines > Immersive experiences.

Warning

Be mindful of how much content you include in immersive scenes that use the `mixed` style. Content that fills a significant portion of the screen, even if that content is partially transparent, can prevent the person from seeing potential hazards in their surroundings. If you want to immerse the person in your content, configure your space with the `full` style. For more information, see Creating fully immersive experiences in your app.

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.

{} Exploring object tracking with ARKit

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

{} Object tracking with Reality Composer Pro experiences

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

{} Placing entities using head and device transform

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