Sample Code

Placing content on detected planes

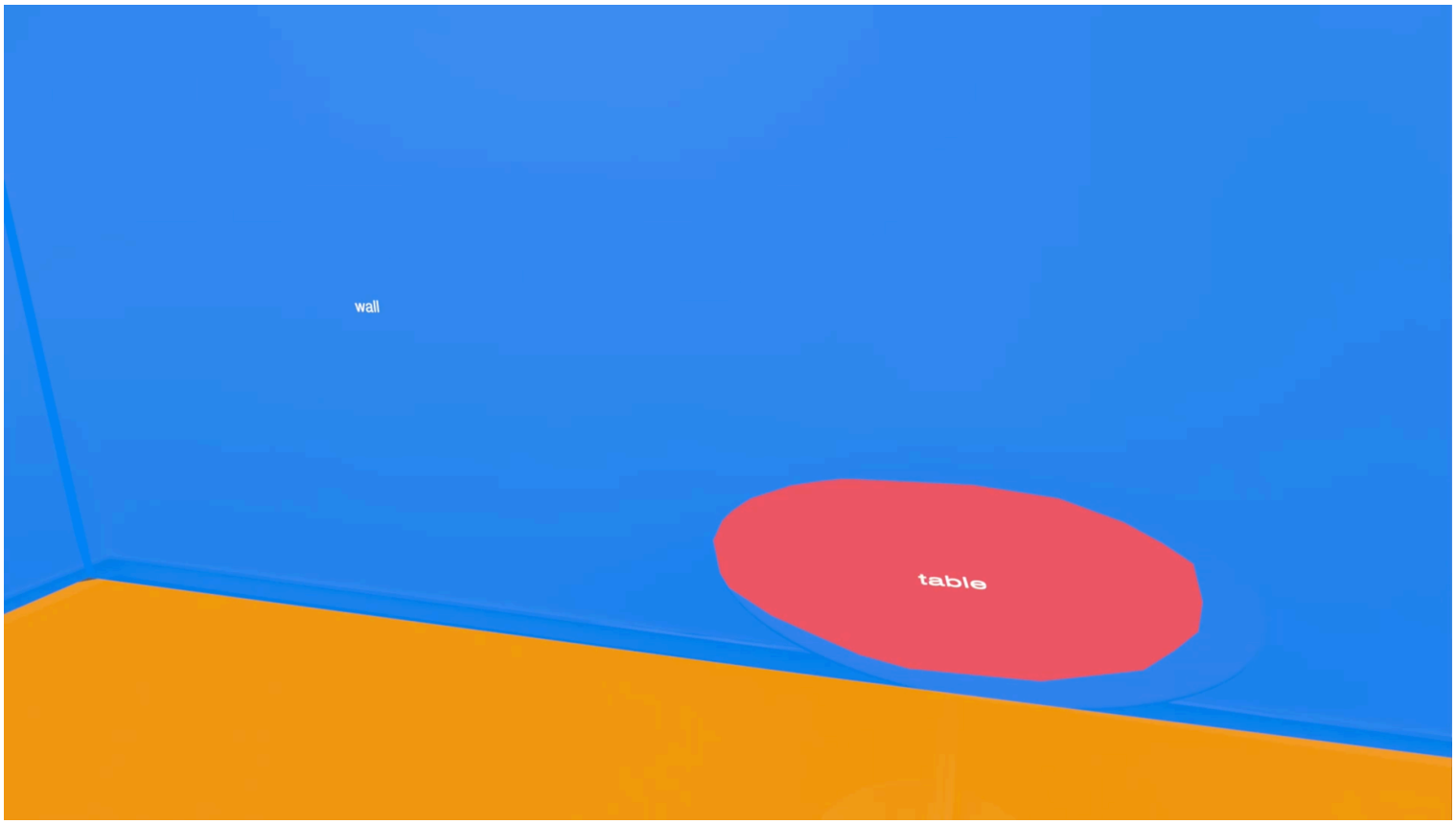
Detect horizontal surfaces like tables and floors, as well as vertical planes like walls and doors.

Download

visionOS 26.0+ | Xcode 26.0+

Overview

Flat surfaces are an ideal place to position content in an app that uses a Full Space in visionOS. They provide a place for virtual 3D content to live alongside a person's surroundings. Use plane detection in ARKit to detect these kinds of surfaces and filter the available planes based on criteria your app might need, such as the size of the plane, its proximity to someone, or a required plane orientation.



Play ⊙

Use RealityKit anchor entities for basic plane anchoring

If you don't need a specific plane in your app and you're rendering your app's 3D content in RealityKit, you can use an `AnchorEntity` instead. This approach lets you attach 3D content to a plane without prompting the person for world-sensing permission and without any particular knowledge of where that plane is relative to the person.

The following shows an anchor that you can use to attach entities to a table:

```
AnchorEntity(.plane(.horizontal, classification: .table, minimumBounds: [0.5, 0.5]))
```

Anchor entities don't let you choose a specific plane in a person's surroundings, but rather let you ask for a plane with certain characteristics. When you need more specific plane selection or real-time information about the plane's position and orientation in the world, use `ARKitSession` and `PlaneDetectionProvider`.

Configure an ARKit session for plane detection

Plane-detection information comes from an `ARKitSession` that's configured to use a `PlaneDetectionProvider`. You can choose to detect horizontal planes, vertical planes, or both. Each plane that ARKit detects comes with a classification, like `PlaneAnchor.Classification.table` or `PlaneAnchor.Classification.floor`. You can use these classifications to

further refine which kinds of planes your app uses to present content. Plane detection requires `ARKitSession.AuthorizationType.worldSensing` authorization.

The following starts a session that detects both horizontal and vertical planes, but filters out planes classified as windows:

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

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

    for await update in planeData.anchorUpdates {
        if update.anchor.classification == .window {
            // Skip planes that are windows.
            continue
        }
        switch update.event {
        case .added, .updated:
            await updatePlane(update.anchor)
        case .removed:
            await removePlane(update.anchor)
        }

    }
}
```

Create and update entities associated with each plane

If you're displaying content that needs to appear attached to a particular plane, update your content whenever you receive new information from ARKit. When a plane is no longer available in the person's surroundings, ARKit sends a removal event. Respond to these events by removing content associated with the plane.

The following shows plane updates that place a text entity on each plane in a person's surroundings; the text entity displays the kind of plane ARKit detected:

```
@MainActor var planeAnchors: [UUID: PlaneAnchor] = [:]
@MainActor var entityMap: [UUID: Entity] = [:]

@MainActor
func updatePlane(_ anchor: PlaneAnchor) {
```

```
    if planeAnchors[anchor.id] == nil {
        // Add a new entity to represent this plane.
        let entity = ModelEntity(mesh: .generateText(anchor.classification.descripti
        entityMap[anchor.id] = entity
        rootEntity.addChild(entity)
    }

    entityMap[anchor.id]?.transform = Transform(matrix: anchor.originFromAnchorTrans
}

@MainActor
func removePlane(_ anchor: PlaneAnchor) {
    entityMap[anchor.id]?.removeFromParent()
    entityMap.removeValue(forKey: anchor.id)
    planeAnchors.removeValue(forKey: anchor.id)
}
```

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.

{} 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.

{} 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.