Sample Code

Happy Beam

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

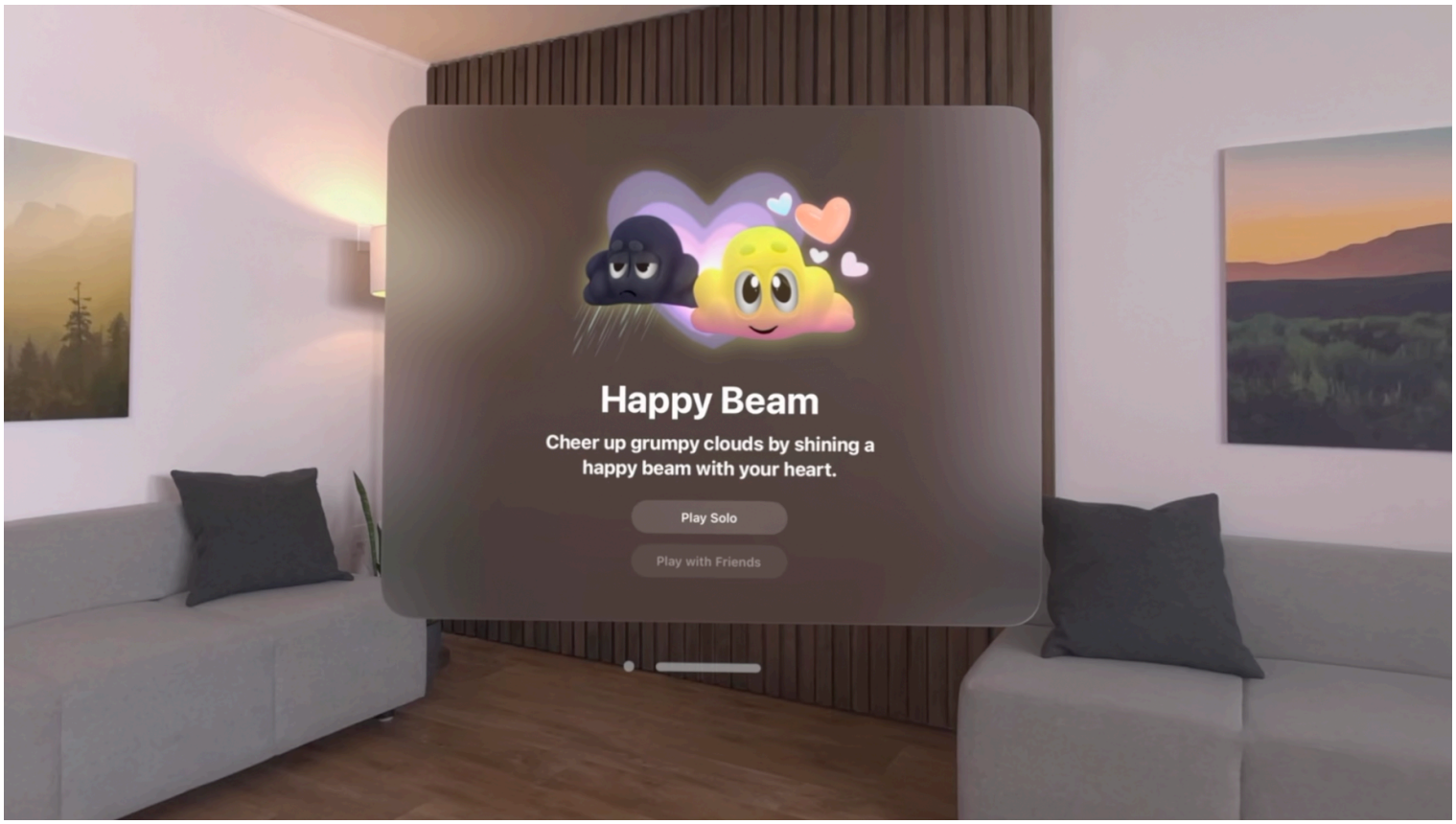
Download

visionOS 2.0+ | Xcode 16.0+

Overview

In visionOS, you can create fun, dynamic games and apps using several different frameworks to create new kinds of spatial experiences: RealityKit, ARKit, SwiftUI, and Group Activities. This sample introduces Happy Beam, a game where you and your friends can hop on a FaceTime call and play together.

You'll learn the mechanics of the game where grumpy clouds float around in the space, and people play by making a heart shape with their hands to project a beam. People aim the beam at the clouds to cheer them up, and a score counter keeps track of how well each player does cheering up the clouds.



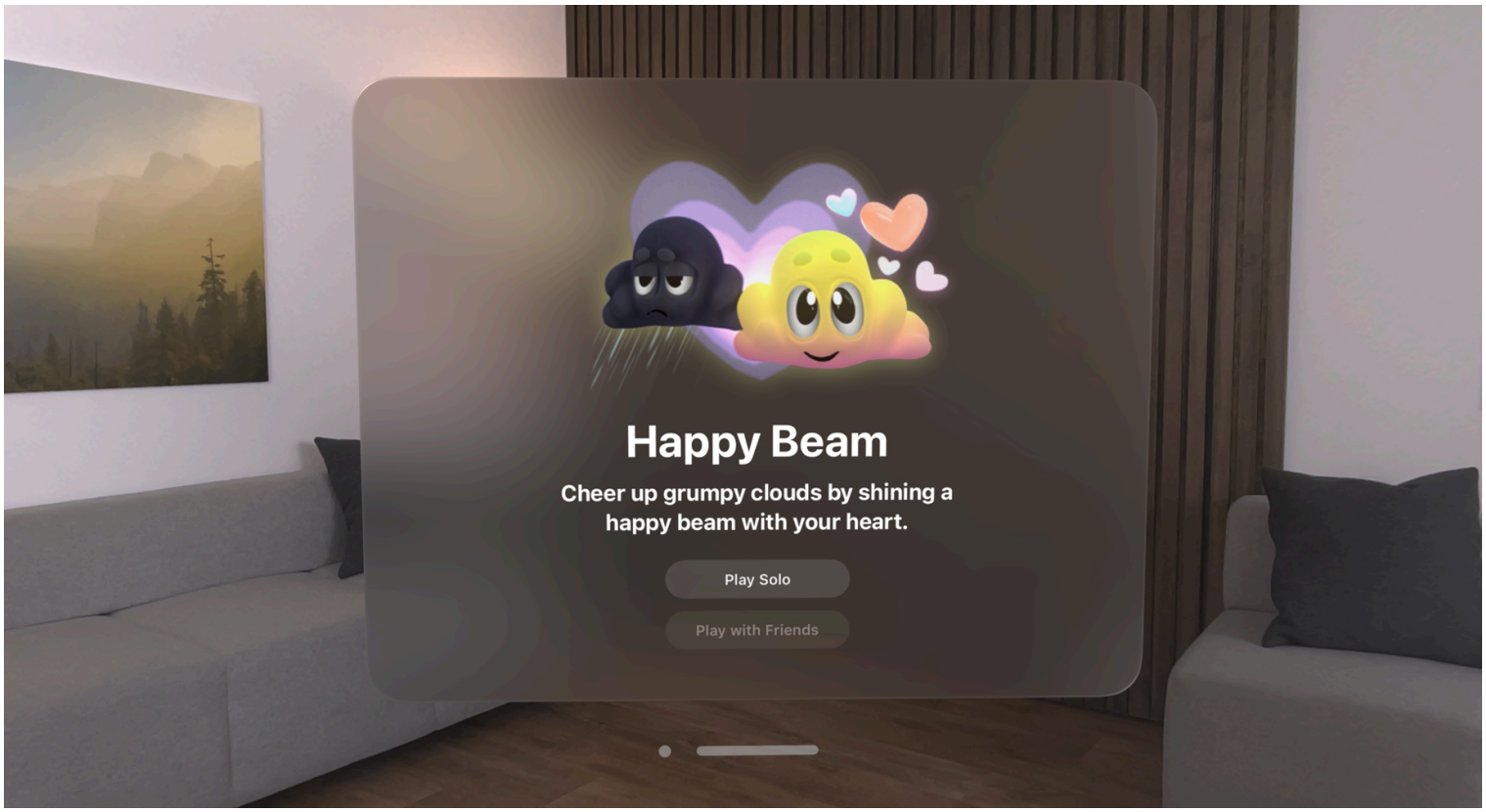
Play

Design the game interface in SwiftUI

Most apps in visionOS launch as a window that opens different scene types depending on the needs of the app.

Here you see how Happy Beam presents a fun interface to people by using several SwiftUI views that display a welcome screen, a coaching screen that gives instructions, a scoreboard, and a game-ending screen.

Welcome window | Instructions | Scoreboard | Ending window



The following shows you the primary view in the app that displays each phase of gameplay:

```
struct HappyBeam: View {
    @Environment(\.openImmersiveSpace) private var openImmersiveSpace
    @Environment(GameModel.self) var gameModel

    @State private var session: GroupSession<HeartProjection>? = nil
    @State private var timer = Timer.publish(every: 1, on: .main, in: .common).autoc
    @State private var subscriptions = Set<AnyCancellable>()

    var body: some View {
        let gameState = GameScreen.from(state: gameModel)
        VStack {
            Spacer()
            Group {
                switch gameState {
                case .start:
                    Start()
                case .soloPlay:
                    SoloPlay()
                case .lobby:
                    Lobby()
                case .soloScore:
                    SoloScore()
                case .multiPlay:
```

```
                    MultiPlay()
                case .multiScore:
                    MultiScore()
                }
            }
            .glassBackgroundEffect(
                in: RoundedRectangle(
                    cornerRadius: 32,
                    style: .continuous
                )
            )
        }
    }
}
```

When 3D content starts to appear, the game opens an immersive space to present content outside of the main window and in a person's surroundings.

```
@main
struct HappyBeamApp: App {
    @State private var gameModel = GameModel()
    @State private var immersionState: ImmersionStyle = .mixed

    var body: some SwiftUI.Scene {
        WindowGroup("HappyBeam", id: "happyBeamApp") {
            HappyBeam()
                .environmentObject(gameModel)
        }
        .windowStyle(.plain)

        ImmersiveSpace(id: "happyBeam") {
            HappyBeamSpace(gestureModel: HeartGestureModelContainer.heartGestureMode
                .environmentObject(gameModel)
        }
        .immersionStyle(selection: $immersionState, in: .mixed)
    }
}
```

The `HappyBeam` container view declares a dependency on `openImmersiveSpace`:

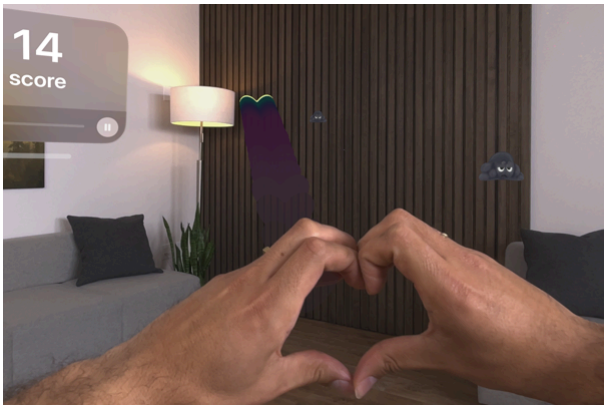
```
@Environment(\.openImmersiveSpace) private var openImmersiveSpace
```

It later uses that dependency to open the space from the app's declaration when it's time to start showing 3D content:

```
if gameModel.countDown == 0 {
    Task {
        await openImmersiveSpace(id: "happyBeam")
    }
}
```

Detect a heart gesture with ARKit

The Happy Beam app recognizes the central *heart-shaped hands* gesture using ARKit's support for 3D hand tracking in visionOS. Using hand tracking requires a running session and authorization from the wearer. It uses the `NSHandsTrackingUsageDescription` user info key to explain to players why the app requests permission for hand tracking.




```
Task {
    do {
        try await session.run([handTrackingProvider])
    } catch {
        print("ARKitSession error:", error)
    }
}
```

Hand-tracking data isn't available when your app is only displaying a window or volume. Instead, it's available when you present an immersive space, as in the previous example.

You can detect gestures using ARKit data with a level of accuracy that depends on your use case and intended experience. For example, Happy Beam could require strict positioning of finger joints to closely resemble a heart shape. Instead, however, it prompts people to make a heart shape and uses a heuristic to indicate when the gesture is close enough.

The following checks whether a person's thumbs and index fingers are almost touching:

```
// Get the position of all joints in world coordinates.
let originFromLeftHandThumbKnuckleTransform = matrix_multiply(
```

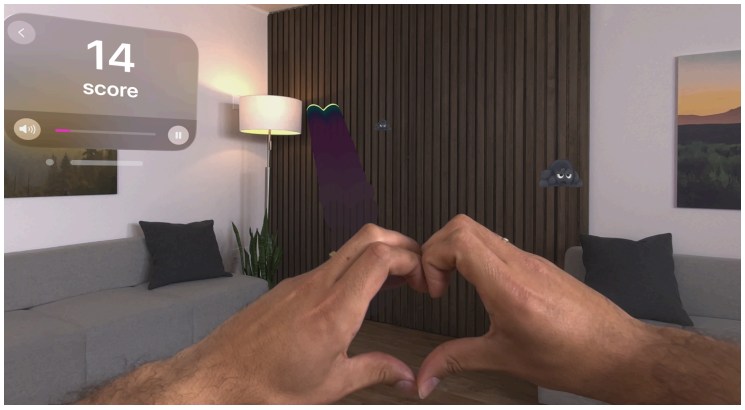
```
    leftHandAnchor.originFromAnchorTransform, leftHandThumbKnuckle.anchorFromJointTr
).columns.3.xyz
let originFromLeftHandThumbTipTransform = matrix_multiply(
    leftHandAnchor.originFromAnchorTransform, leftHandThumbTipPosition.anchorFromJoi
).columns.3.xyz
let originFromLeftHandIndexFingerTipTransform = matrix_multiply(
    leftHandAnchor.originFromAnchorTransform, leftHandIndexFingerTip.anchorFromJoint
).columns.3.xyz
let originFromRightHandThumbKnuckleTransform = matrix_multiply(
    rightHandAnchor.originFromAnchorTransform, rightHandThumbKnuckle.anchorFromJoint
).columns.3.xyz
let originFromRightHandThumbTipTransform = matrix_multiply(
    rightHandAnchor.originFromAnchorTransform, rightHandThumbTipPosition.anchorFromJ
).columns.3.xyz
let originFromRightHandIndexFingerTipTransform = matrix_multiply(
    rightHandAnchor.originFromAnchorTransform, rightHandIndexFingerTip.anchorFromJoi
).columns.3.xyz

let indexFingersDistance = distance(originFromLeftHandIndexFingerTipTransform, origi
let thumbsDistance = distance(originFromLeftHandThumbTipTransform, originFromRightHa

// Heart gesture detection is true when the distance between the index finger tips c
// and the distance between the thumb tip centers is each less than four centimeters
let isHeartShapeGesture = indexFingersDistance < 0.04 && thumbsDistance < 0.04
if !isHeartShapeGesture {
    return nil
}

// Compute a position in the middle of the heart gesture.
let halfway = (originFromRightHandIndexFingerTipTransform - originFromLeftHandThumbT
let heartMidpoint = originFromRightHandIndexFingerTipTransform - halfway

// Compute the vector from left thumb knuckle to right thumb knuckle and normalize (
let xAxis = normalize(originFromRightHandThumbKnuckleTransform - originFromLeftHandT

// Compute the vector from right thumb tip to right index finger tip and normalize (
let yAxis = normalize(originFromRightHandIndexFingerTipTransform - originFromRightHa

let zAxis = normalize(cross(xAxis, yAxis))

// Create the final transform for the heart gesture from the three axes and midpoint
let heartMidpointWorldTransform = simd_matrix(
    SIMD4(xAxis.x, xAxis.y, xAxis.z, 0),
```

```
    SIMD4(yAxis.x, yAxis.y, yAxis.z, 0),
    SIMD4(zAxis.x, zAxis.y, zAxis.z, 0),
    SIMD4(heartMidpoint.x, heartMidpoint.y, heartMidpoint.z, 1)
)
```

Support several kinds of input

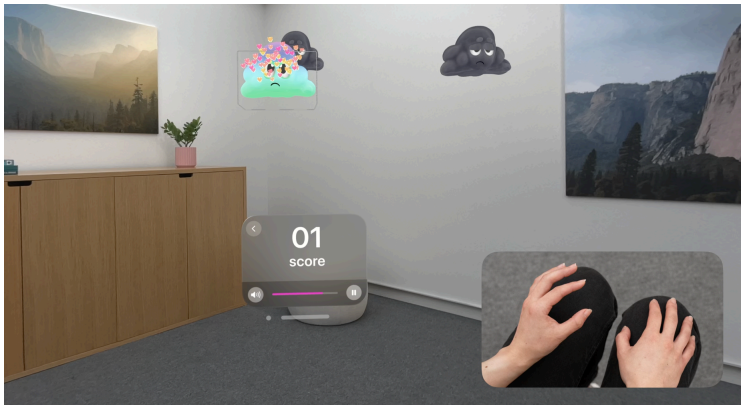
To support accessibility features and general user preferences, include multiple kinds of input in an app that uses hand tracking as one form of input.

Happy Beam supports several kinds of input:



Interactive hands input from ARKit with the custom heart gesture.

Drag gesture input to rotate the stationary beam on its platform.



Accessibility components from RealityKit to support custom actions for cheering up the clouds.

Game Controller support to make control over the beam more interactive from Switch Control.

Display 3D content with RealityKit

The 3D content in the app comes in the form of assets that you can export from Reality Composer Pro. You place each asset in the `RealityView` that represents your immersive space.

The following shows how Happy Beam generates clouds when the game starts, as well as materials for the floor-based beam projector. Because the game uses collision detection to keep

score — the beam cheers up grumpy clouds when they collide — you make collision shapes for each model that might be involved.

```
@MainActor
func placeCloud(start: Point3D, end: Point3D, speed: Double) async throws -> Entity
    let cloud = await loadFromRealityComposerPro(
        named: BundleAssets.cloudEntity,
        fromSceneNamed: BundleAssets.cloudScene
    )!
        .clone(recursive: true)

    cloud.generateCollisionShapes(recursive: true)
    cloud.components[PhysicsBodyComponent.self] = PhysicsBodyComponent()

    var accessibilityComponent = AccessibilityComponent()
    accessibilityComponent.label = "Cloud"
    accessibilityComponent.value = "Grumpy"
    accessibilityComponent.isAccessibilityElement = true
    accessibilityComponent.traits = [.button, .playsSound]
    accessibilityComponent.systemActions = [.activate]
    cloud.components[AccessibilityComponent.self] = accessibilityComponent

    let animation = cloudMovementAnimations[cloudPathsIndex]

    cloud.playAnimation(animation, transitionDuration: 1.0, startsPaused: false)
    cloudAnimate(cloud, kind: .sadBlink, shouldRepeat: false)
    spaceOrigin.addChild(cloud)

    return cloud
}
```

Add SharePlay support for multiplayer gaming experiences

You use the Group Activities framework in visionOS to support SharePlay during a FaceTime call. Happy Beam uses Group Activities to sync the score, active players list, and the position of each player's projected beam.

> **Note**
>
> Developers using the Apple Vision Pro developer kit can test spatial SharePlay experiences on-device by installing the Persona Preview Profile.

Use a reliable channel to send information that's important to be correct, even if it can be slightly delayed as a result. The following shows how Happy Beam updates the game model's score state in response to a score message:

```
sessionInfo.reliableMessenger = GroupSessionMessenger(session: newSession, deliveryM

Task {
    for await (message, sender) in sessionInfo!.reliableMessenger!.messages(of: Scor
        gameModel.clouds[message.cloudID].isHappy = true
        gameModel
            .players
            .filter { $0.name == sender.source.id.asPlayerName }
            .first!
            .score += 1
    }
}
```

Use an unreliable messenger for sending data with low-latency requirements. Because the delivery mode is unreliable, some messages might not make it. Happy Beam uses the unreliable mode to send live updates to the position of the beam when each participant in the call chooses the Spatial option in FaceTime.

```
sessionInfo.messenger = GroupSessionMessenger(session: newSession, deliveryMode: .ur
```

The following shows how Happy Beam serializes beam data for each message:

```
// Send each player's beam data during FaceTime calls where players have selected th
func sendBeamPositionUpdate(_ pose: Pose3D) {
    if let sessionInfo = sessionInfo, let session = sessionInfo.session, let messeng
        let everyoneElse = session.activeParticipants.subtracting([session.localPart

        if isShowingBeam, gameModel.isSpatial {
            messenger.send(BeamMessage(pose: pose), to: .only(everyoneElse)) { erro
                if let error = error { print("Message failure:", error) }
            }
```

```
            }
        }
    }
```

See Also

Related samples

{} 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.

{} Hello World

Use windows, volumes, and immersive spaces to teach people about the Earth.

{} Destination Video

Leverage SwiftUI to build an immersive media experience in a multiplatform app.

{} Diorama

Design scenes for your visionOS app using Reality Composer Pro.

Related articles

Setting up access to ARKit data

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

{} Placing content on detected planes

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

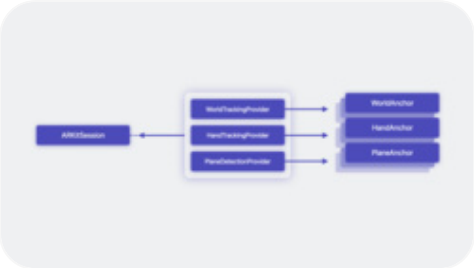
{} Tracking specific points in world space

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

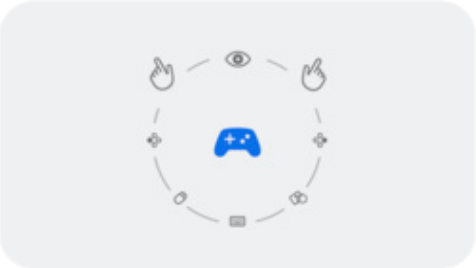
Tracking preregistered images in 3D space

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

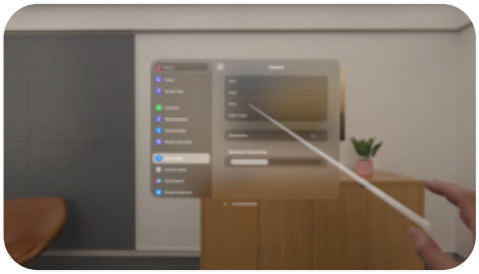
Related videos



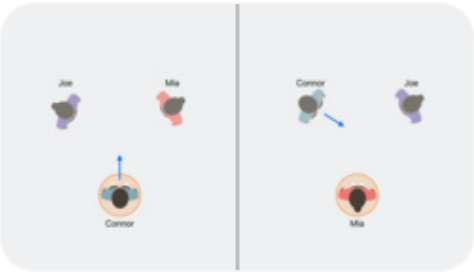
Meet ARKit for spatial computing



Build great games for spatial computing



Create accessible spatial experiences



Build spatial SharePlay experiences