

Hi 🙌 ~ I'm Vina Melody

大家早上好！

我是 Vina， 来自新加坡。

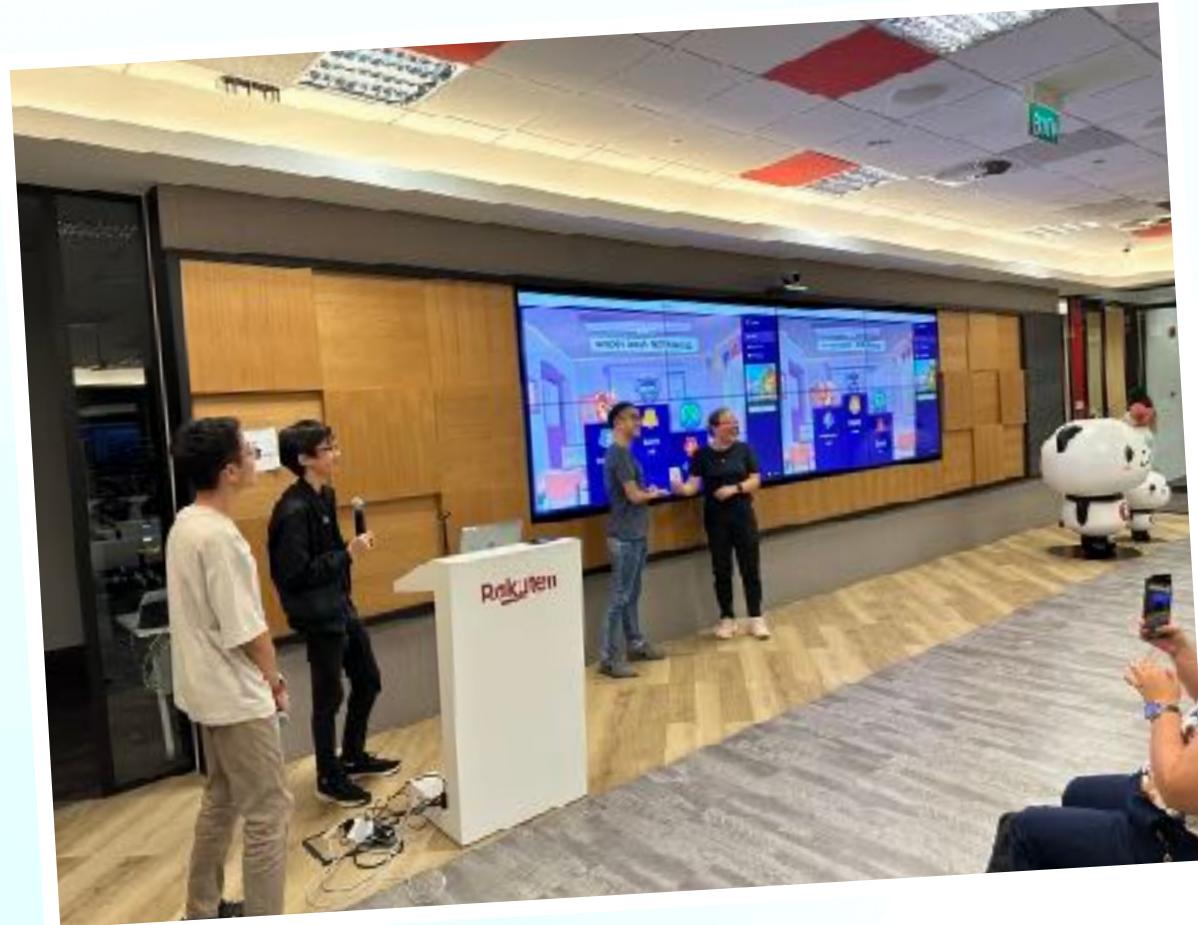
在我开始之前我想要感谢并祝贺 Let's VisionOS 团队 — Siqi, Frank, Clare, Pei Hao, Xiao Shen, SwiftGG XReality.Zone 谢谢你们举办了这次精彩的会议。

我不会说中文， 所以我就用英文继续吧^^



Hi 🙌 ~ I'm Vina Melody

- 📱 iOS Software Engineer
- iOS Conf SG
- iOS Dev Scout Meetup
- Visit us!



Practical Techniques to Glow Up Your Immersive Content

From Zero to Immersive Hero



Vina Melody, 31 March 2024

What are we building ?



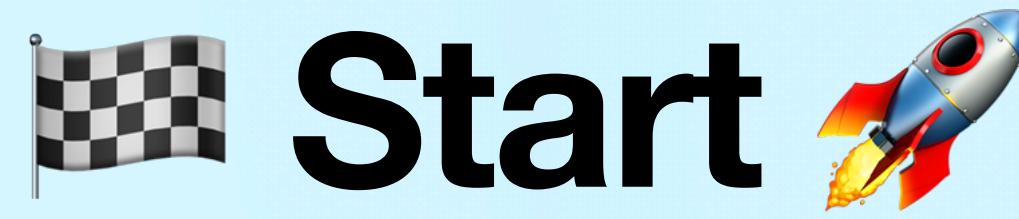
- Ways to place a virtual object (Entity)
- Interactions
- Audio
- Animations
- Attachments
- 警告：有很多代码



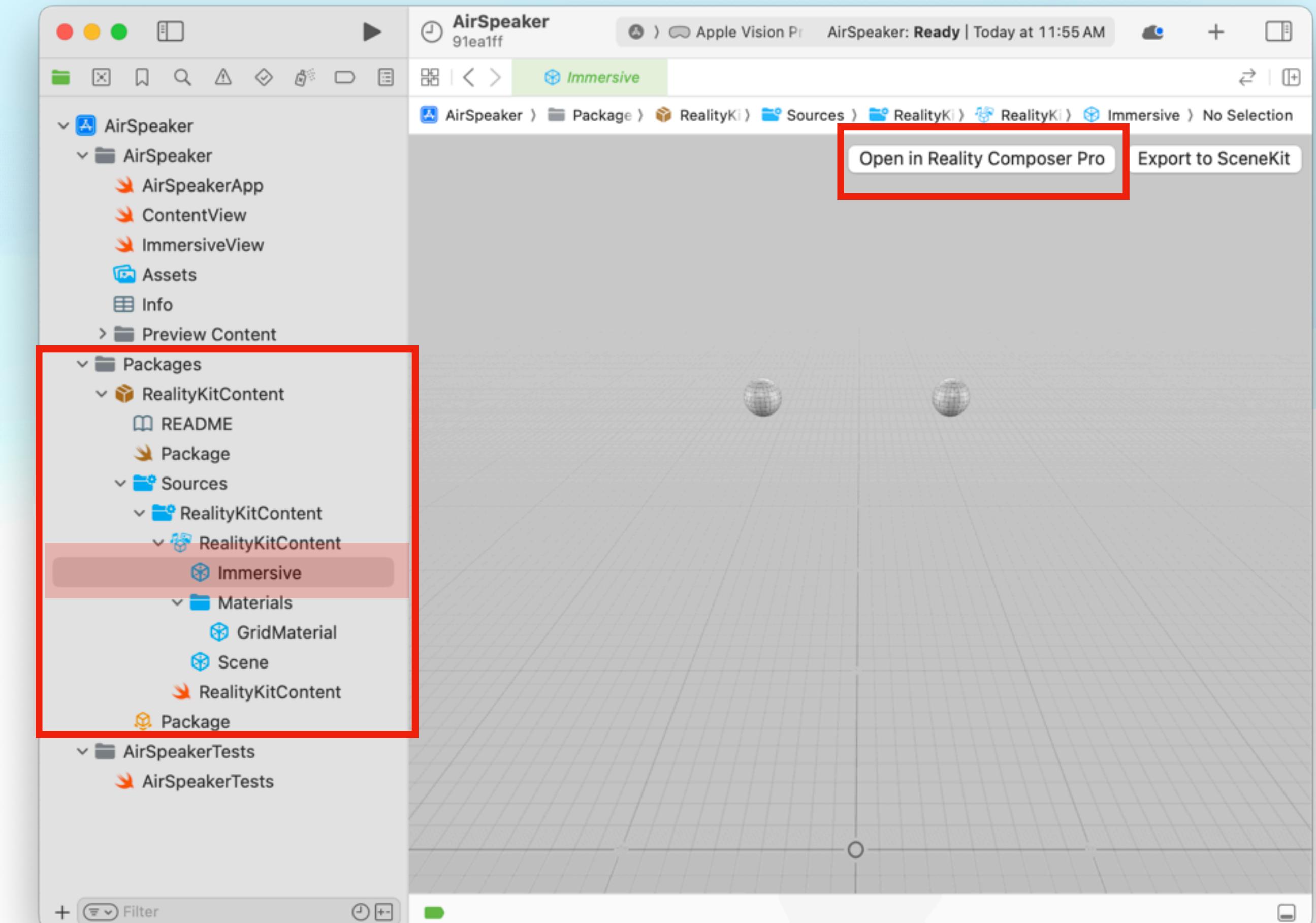
What are we building ?



Build Small Things!



Reality Composer Pro





Clean up

- Trash RealityKitContent package
- Trash ContentView.swift

```
// AirSpeakerApp.swift

@main
struct AirSpeakerApp: App {
    var body: some Scene {
        WindowGroup {
            ContentView()      ✘ Cannot find 'ContentView' in scope
        }.windowStyle(.volumetric)

        ImmersiveSpace(id: "ImmersiveSpace") {
            ImmersiveView()
        }
    }
}
```



Clean up

```
@main
struct AirSpeakerApp: App {

    var body: some Scene {
        WindowGroup {
            EmptyView()
        }
        .windowStyle(.volumetric)

        ImmersiveSpace(id: "ImmersiveSpace") {
            ImmersiveView()
        }
    }
}
```





Clean up

```
@main
struct AirSpeakerApp: App {

    var body: some Scene {
        WindowGroup {
            EmptyView()
                .persistentSystemOverlays(.hidden)
        }
        .windowStyle(.volumetric)

        ImmersiveSpace(id: "ImmersiveSpace") {
            ImmersiveView()
        }
    }
}
```



Clean up

```
@main
struct AirSpeakerApp: App {

    @Environment(\.openImmersiveSpace) var openImmersiveSpace

    var body: some Scene {
        WindowGroup {
            EmptyView()
                .persistentSystemOverlays(.hidden)
                .task {
                    await openImmersiveSpace(id: "ImmersiveSpace")
                }
            }
            .windowStyle(.volumetric)

            ImmersiveSpace(id: "ImmersiveSpace") {
                ImmersiveView()
            }
        }
    }
}
```

Load a ModelEntity

```
import RealityKitContent

struct ImmersiveView: View {

    var body: some View {

        RealityView { content in

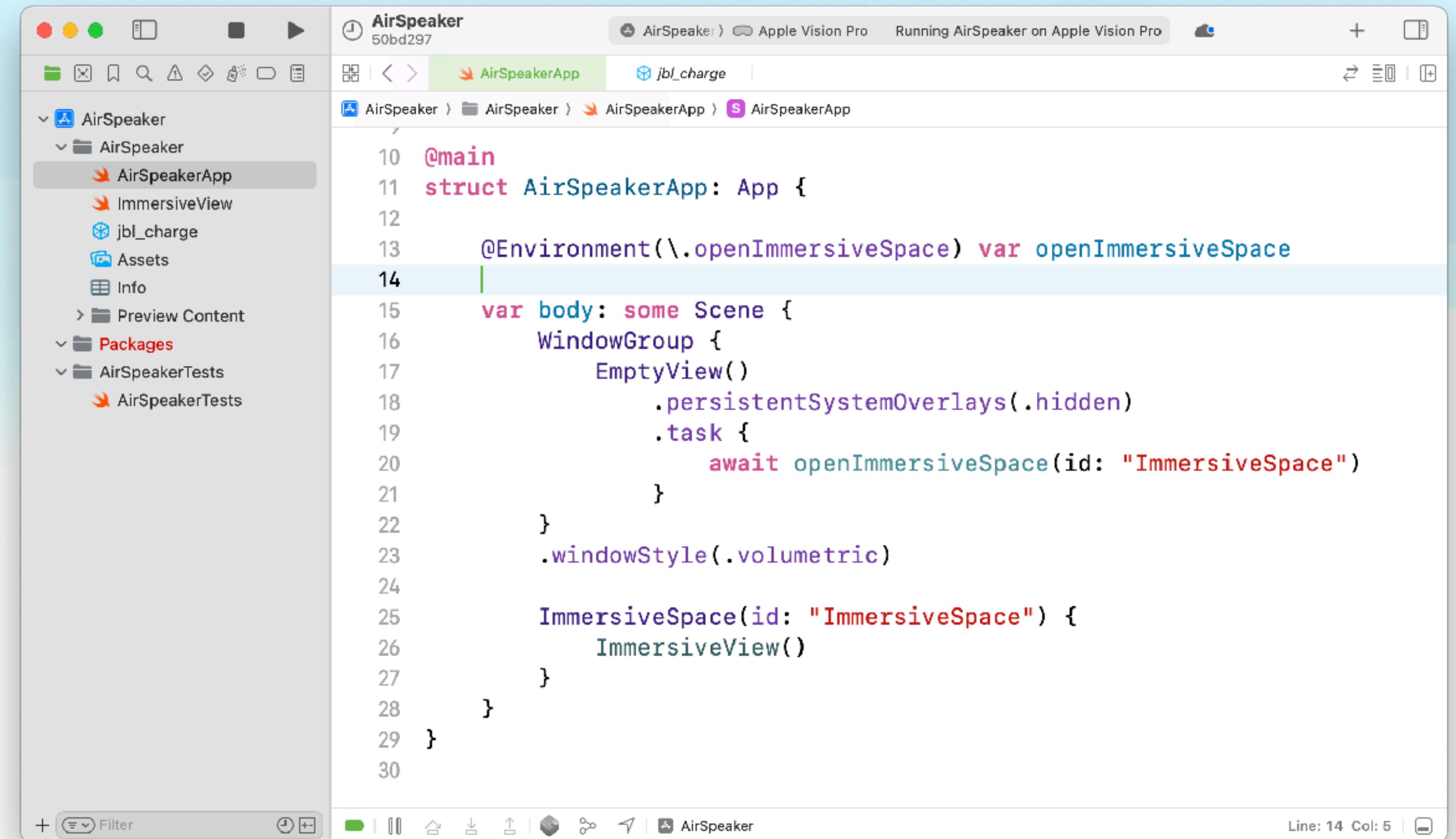
            // Add the initial RealityKit content
            if let scene = try? await Entity(named: "Immersive", in: realityKitContentBundle) {
                content.add(scene)
            }
        }
    }
}
```

Load a ModelEntity

```
struct ImmersiveView: View {  
    @State var speaker: ModelEntity? = nil  
  
    var body: some View {  
        RealityView { content in  
  
            do {  
                speaker = try await ModelEntity(named: "jbl_charge")  
  
                if let speaker {  
                    content.add(speaker)  
                }  
            }  
            catch {  
                print("Error loading the model")  
            }  
        }  
    }  
}
```

🏁 Checkpoint 1

- Preparation
- Load ModelEntity



The screenshot shows the Xcode interface with the project "AirSpeaker" open. The left sidebar displays the project structure:

- AirSpeaker
- AirSpeaker
- AirSpeakerApp (selected)
- ImmersiveView
- jbl_charge
- Assets
- Info
- Preview Content
- Packages
- AirSpeakerTests
- AirSpeakerTests

The main editor area shows the following SwiftUI code:

```
10 @main
11 struct AirSpeakerApp: App {
12
13   @Environment(\.openImmersiveSpace) var openImmersiveSpace
14
15   var body: some Scene {
16     WindowGroup {
17       EmptyView()
18       .persistentSystemOverlays(.hidden)
19       .task {
20         await openImmersiveSpace(id: "ImmersiveSpace")
21       }
22     }
23     .windowStyle(.volumetric)
24
25     ImmersiveSpace(id: "ImmersiveSpace") {
26       ImmersiveView()
27     }
28   }
29 }
30
```

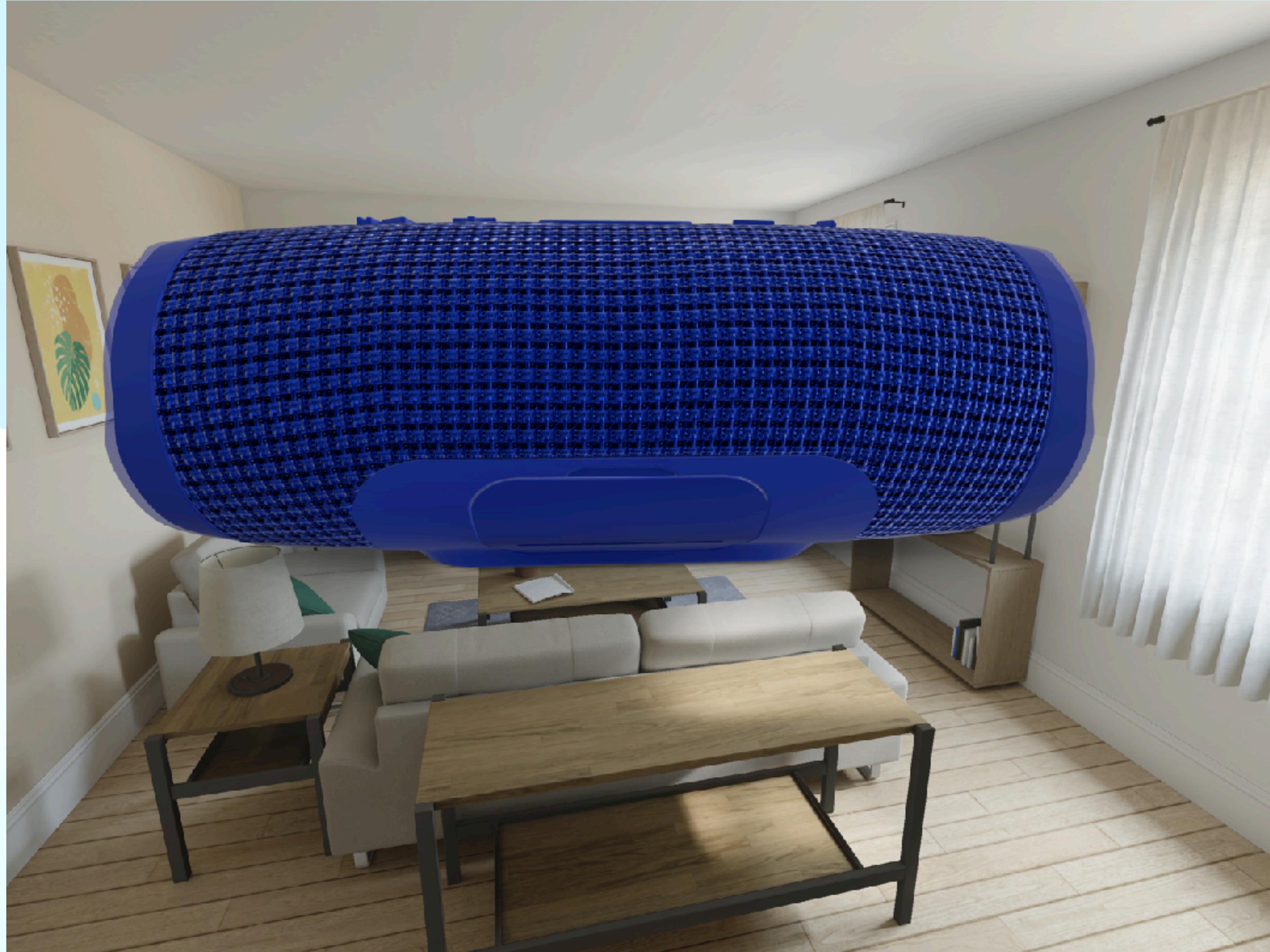
The code is focused on setting up an ImmersiveView within a WindowGroup, utilizing the `@Environment` modifier to access the `openImmersiveSpace` environment variable.

The 3D Model



<https://sketchfab.com/3d-models/jbl-charge-3-blue-animated-b4967a9241704431b718314ede21cad3>

Load a ModelEntity

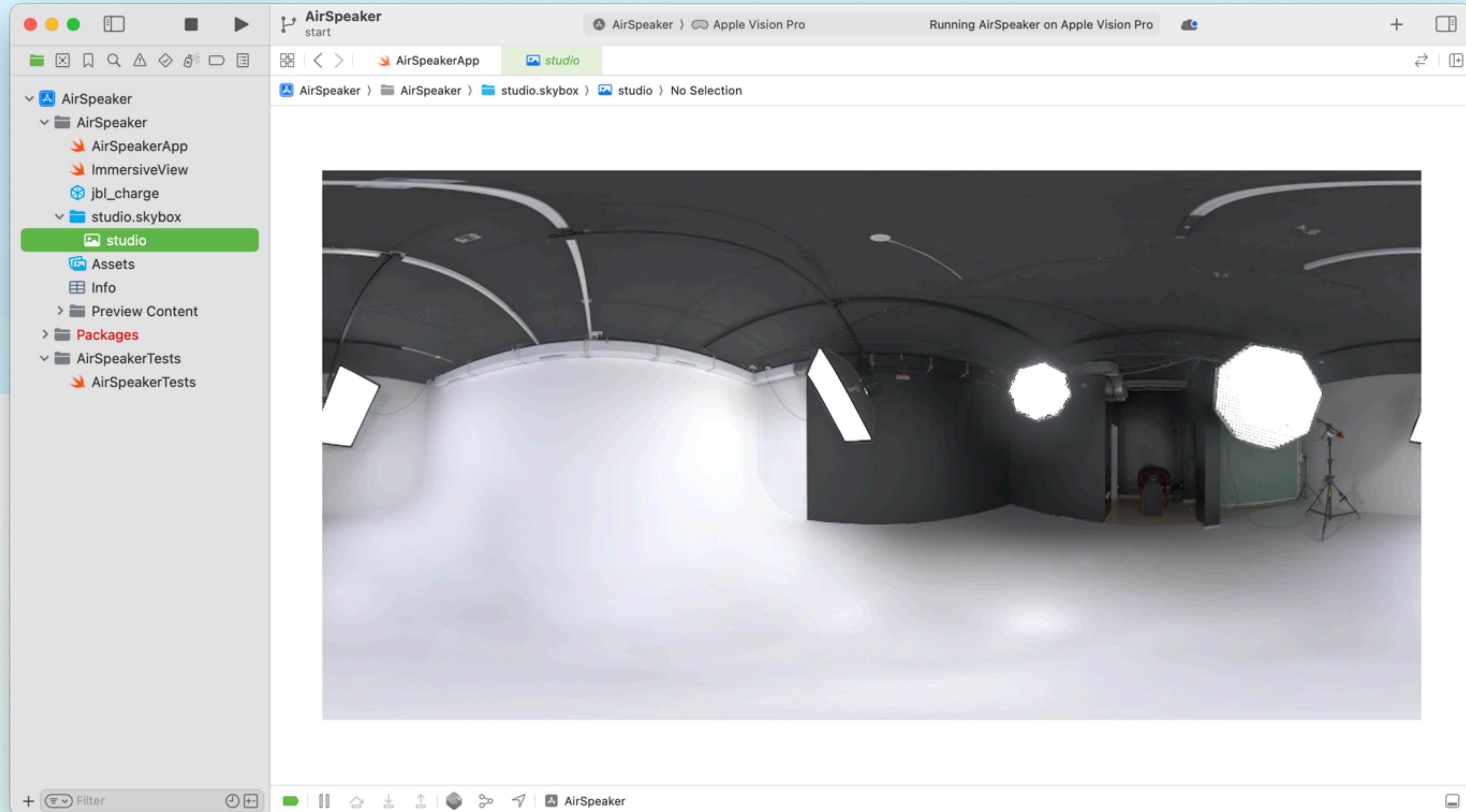


ModelEntity

```
struct ImmersiveView: View {  
  
    @State var speaker: ModelEntity? = nil  
  
    var body: some View {  
        RealityView { content in  
            do {  
                speaker = try await ModelEntity(named: "jbl_charge")  
  
                if let speaker {  
                    speaker.scale = [0.00075, 0.00075, 0.00075]  
                    speaker.position.z = -1  
  
                    content.add(speaker)  
                }  
            }  
            catch {  
                print("Error creating Entity")  
            }  
        }  
    }  
}
```



ImageBasedLightComponent



ImageBasedLightComponent

```
var body: some View {
    RealityView { content in
        do {
            speaker = try await ModelEntity(named: "jbl_charge")
            let environment = try await EnvironmentResource(named: "studio")
            if let speaker {
                speaker.scale = [0.00075, 0.00075, 0.00075]
                speaker.position.z = -1
                speaker.components.set(ImageBasedLightComponent(source: .single(environment)))
                speaker.components.set(ImageBasedLightReceiverComponent(imageBasedLight: speaker))
            }
            content.add(speaker)
        }
    } catch {
        print("Error loading the model")
    }
}
```

ImageBasedLightComponent



Before



After

Shadow

```
var body: some View {
    RealityView { content in
        do {
            speaker = try await ModelEntity(named: "jbl_charge")
            let environment = try await EnvironmentResource(named: "studio")

            if let speaker {
                speaker.scale = [0.00075, 0.00075, 0.00075]
                speaker.position.z = -1

                speaker.components.set(ImageBasedLightComponent(source: .single(environment)))
                speaker.components.set(ImageBasedLightReceiverComponent(imageBasedLight: speaker))

                speaker.components.set(GroundingShadowComponent(castsShadow: true))
            }

            content.add(speaker)
        }
    }
    catch {
        print("Error loading the model")
    }
}
```

Shadow



Handle drag gestures

```
var body: some View {
    RealityView { content in
        do {
            ...
            if let speaker = ...
            {
                let speakerBounds = speaker.model!.mesh.bounds.extents
                speaker.components.set(CollisionComponent(shapes: [.generateBox(size: speakerBounds)]))
                speaker.components.set(InputTargetComponent())
                speaker.components.set(GroundingShadowComponent(castsShadow: true))
                content.add(speaker)
            }
        } catch {...}
    }
    .gesture(dragGesture)
}

private var dragGesture: some Gesture {...}
```

Handle drag gestures

```
var dragGesture: some Gesture {
    DragGesture()
        .targetedToAnyEntity()
        .onChanged { value in
            let entity = value.entity

            if !isDragging {
                isDragging = true
                dragStartPosition = entity.position(relativeTo: nil)
            }

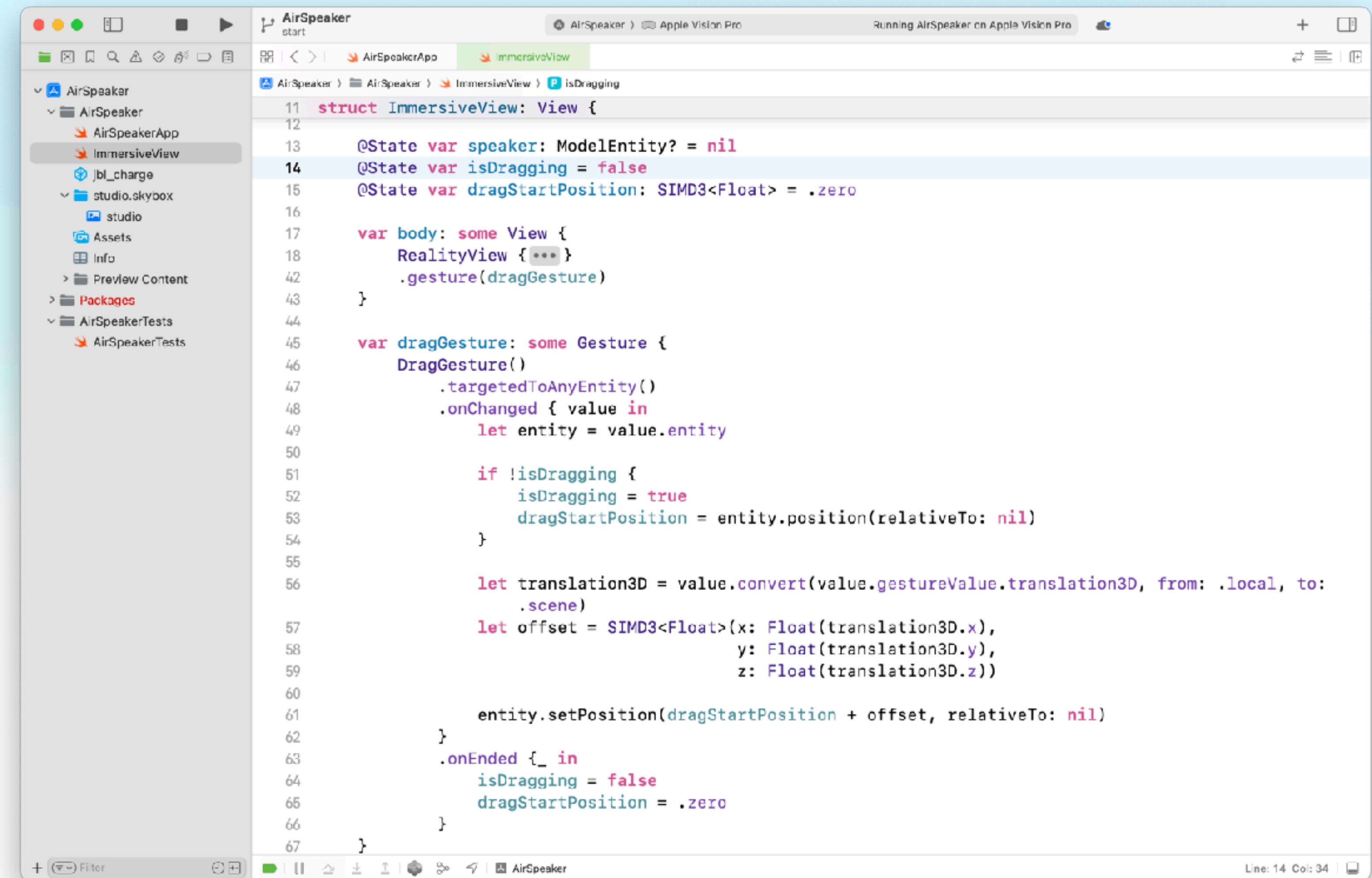
            let translation3D = value.convert(value.gestureValue.translation3D,
                                              from: .local, to: .scene)
            let offset = SIMD3<Float>(x: Float(translation3D.x),
                                         y: Float(translation3D.y),
                                         z: Float(translation3D.z))

            entity.setPosition(dragStartPosition + offset, relativeTo: nil)
        }

        .onEnded { _ in
            isDragging = false
            dragStartPosition = .zero
        }
}
```

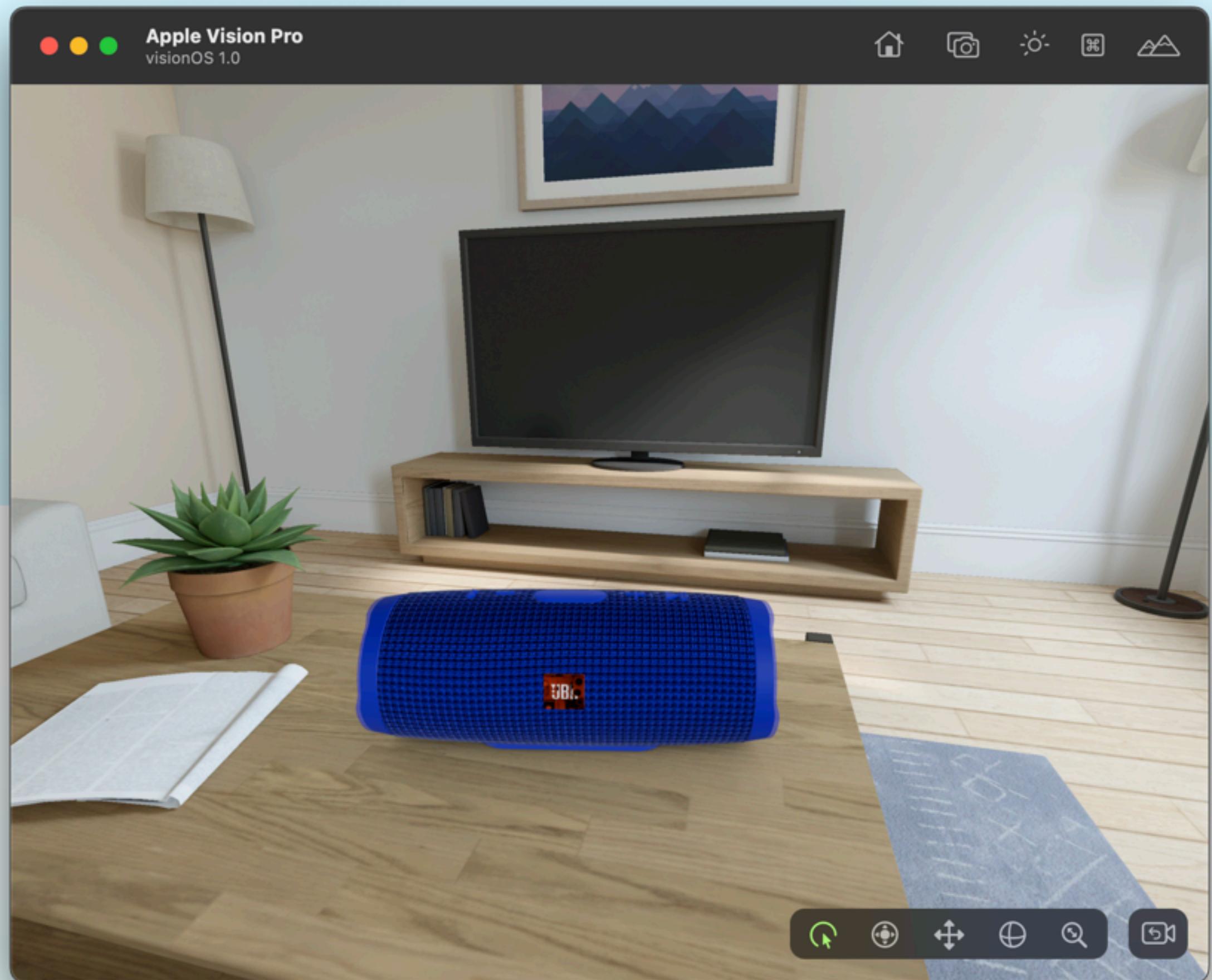
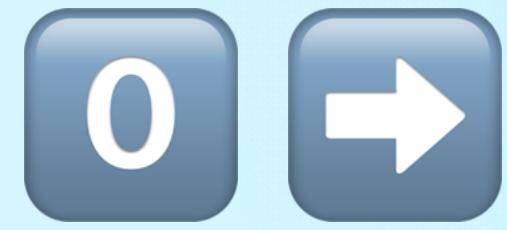
🏁 Checkpoint

- Scaling
- ImageBasedLight
- Shadow
- Drag gestures



The screenshot shows the Xcode interface with the project navigation bar at the top. The main area displays the code for the `ImmersiveView` struct. The code handles drag gestures to move a 3D entity in a scene. It includes state variables for the speaker, dragging status, and drag start position, along with logic to update the entity's position based on the drag gesture.

```
11 struct ImmersiveView: View {
12
13     @State var speaker: ModelEntity? = nil
14     @State var isDragging = false
15     @State var dragStartPosition: SIMD3<Float> = .zero
16
17     var body: some View {
18         RealityView { ... }
19             .gesture(dragGesture)
20     }
21
22     var dragGesture: some Gesture {
23         DragGesture()
24             .targetedToAnyEntity()
25             .onChanged { value in
26                 let entity = value.entity
27
28                 if !isDragging {
29                     isDragging = true
30                     dragStartPosition = entity.position(relativeTo: nil)
31                 }
32
33                 let translation3D = value.convert(value.gestureValue.translation3D, from: .local, to:
34                     .scene)
35                 let offset = SIMD3<Float>(x: Float(translation3D.x),
36                                         y: Float(translation3D.y),
37                                         z: Float(translation3D.z))
38
39                 entity.setPosition(dragStartPosition + offset, relativeTo: nil)
40             }
41             .onEnded {_ in
42                 isDragging = false
43                 dragStartPosition = .zero
44             }
45     }
46 }
```



Anchoring



```
RealityView { content in
    do {
        speaker = try await ModelEntity(named: "jbl_charge")
        let environment = try await EnvironmentResource(named: "studio")

        if let speaker {
            speaker.scale = [0.00075, 0.00075, 0.00075]
            speaker.position.z = -1

            speaker.components.set(ImageBasedLightComponent(source: .single(environment)))
            speaker.components.set(ImageBasedLightReceiverComponent(imageBasedLight: speaker))

            speaker.components.set(GroundingShadowComponent(castsShadow: true))
            content.add(speaker)
        }
    }
    catch {...}
}
.gesture(dragGesture)
```

Anchoring



```
RealityView { content in
    do {
        speaker = try await ModelEntity(named: "jbl_charge")
        let environment = try await EnvironmentResource(named: "studio")

        if let speaker {
            let table = {
                let anchor = AnchorEntity(.plane(.horizontal,
                                                classification: .table,
                                                minimumBounds: [0.5, 0.5]))
                anchor.addChild(speaker, preservingWorldTransform: true)
                return anchor
            }()
        }

        speaker.scale = [0.00075, 0.00075, 0.00075]
        speaker.position.z = -1

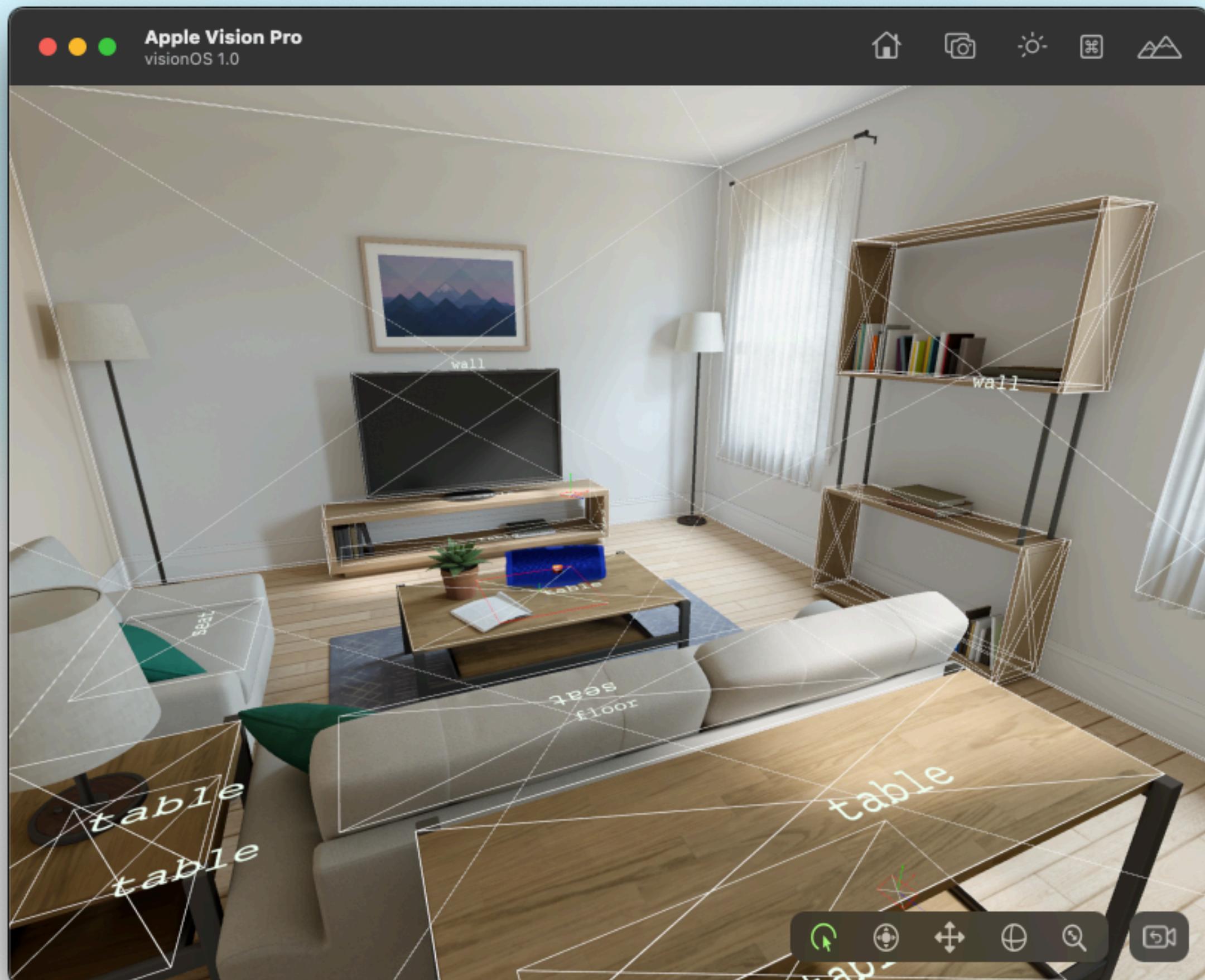
        ...
        content.add(table)
    }
} catch {...}
}
.gesture(dragGesture)
```

Anchoring



The screenshot shows the Xcode interface with the following details:

- Project Navigator:** Shows the project structure for "AirSpeaker".
- File Navigator:** Shows the file structure for "ImmersiveView".
- Editor:** Displays the Swift code for the `ImmersiveView` struct. The code initializes a speaker by trying to await a ModelEntity named "jbl_charge" and an EnvironmentResource named "studio". It then creates an anchor entity at a table, adds the speaker to it, and returns the anchor. The speaker's scale is set to [0.00075, 0.00075, 0.00075]. The code also configures components like ImageBasedLightComponent and CollisionComponent for the speaker model.
- Visualizations:** A sidebar with a red border highlights the "Anchoring" visualization, which shows the speaker being anchored to a table. Other options include "Axes", "Bounds", "Collision Shapes & Axes", "Occlusion Mesh", and "Surfaces".
- Output Window:** Shows three lines of log output from the Apple Vision Pro device:
 - [0x1019b2e00] Options: 1x1 [FFFFFFFF,FFFFFF] 0082D060
 - [0x1019b2e00] Decoding completed without errors
 - [0x1019b2e00] Releasing session



Animation

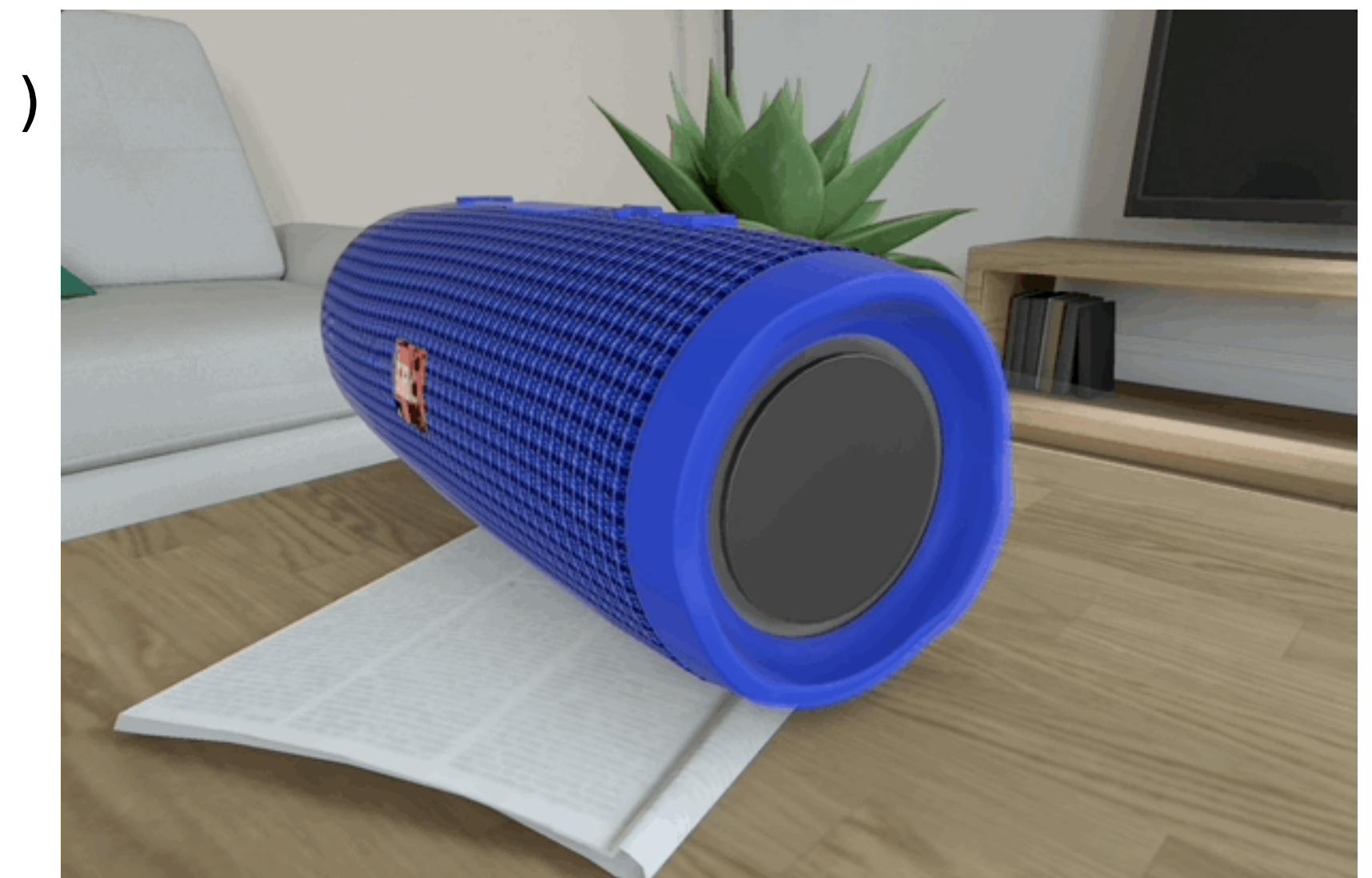
```
struct ImmersiveView: View {  
  
    var body: some View {  
        RealityView { content in  
            do {  
  
                if let speaker {  
                    speaker.components.set(GroundingShadowComponent(castsShadow: true))  
                    content.add(table)  
                }  
            }  
            catch {...}  
        }  
        .gesture(dragGesture)  
    }  
}
```

Animation

```
struct ImmersiveView: View {  
    @State var animation: AnimationResource? = nil  
  
    var body: some View {  
        RealityView { content in  
            do {  
  
                if let speaker {  
                    speaker.components.set(GroundingShadowComponent(castsShadow: true))  
                    content.add(table)  
                }  
            }  
            catch {...}  
        }  
        .gesture(dragGesture)  
    }  
}
```

Animation

```
struct ImmersiveView: View {  
  
    @State var animation: AnimationResource? = nil  
  
    var body: some View {  
        RealityView { content in  
            do {  
  
                if let speaker {  
                    speaker.components.set(GroundingShadowComponent(castsShadow: true))  
  
                    animation = speaker.availableAnimations[0]  
                    if let animation {  
                        speaker.playAnimation(animation.repeat())  
                    }  
  
                    content.add(table)  
                }  
            }  
            catch {...}  
        }  
        .gesture(dragGesture)  
    }  
}
```



Audio

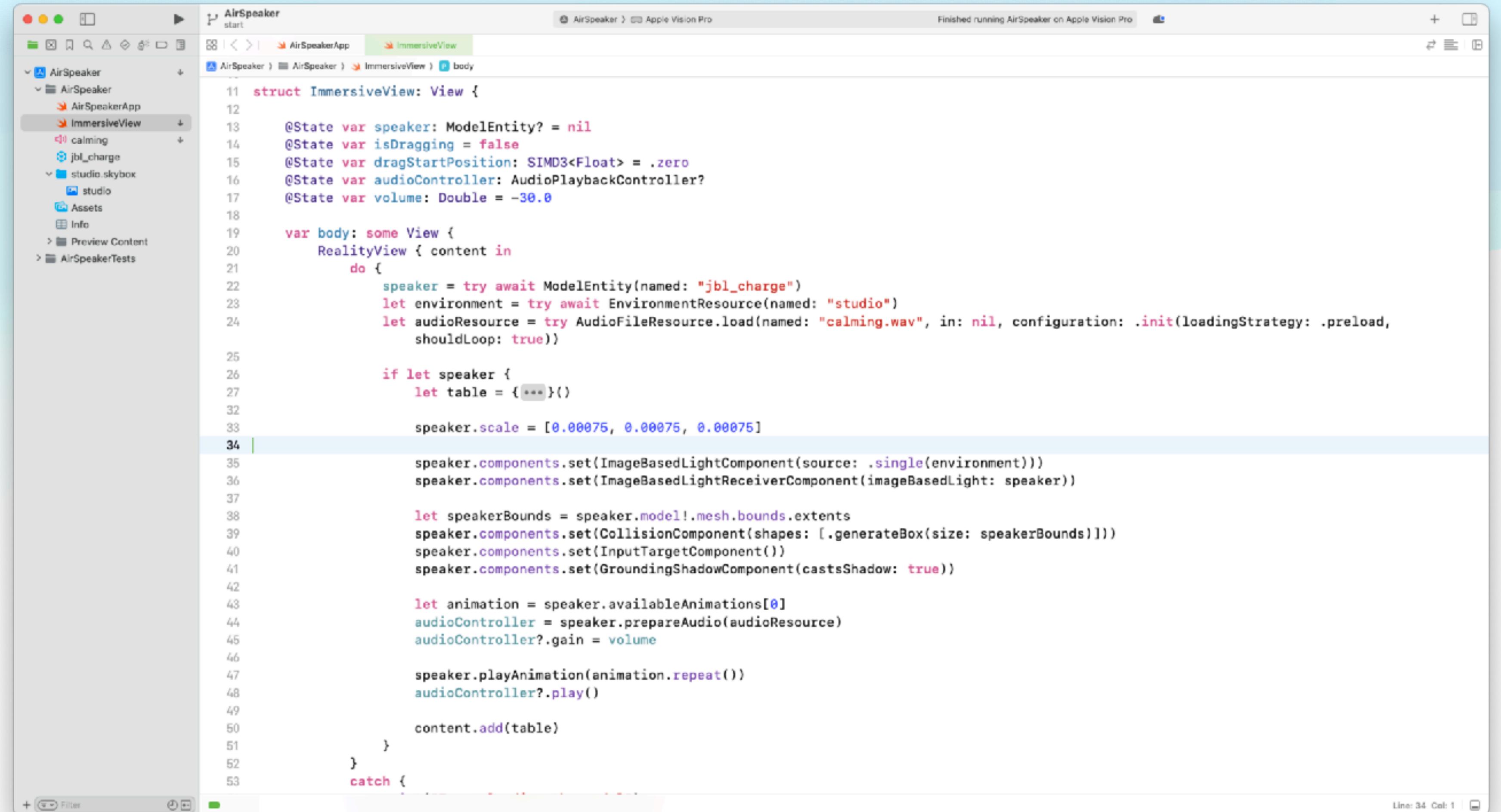
```
struct ImmersiveView: View {  
  
    @State var audioController: AudioPlaybackController?  
  
    var body: some View {  
        RealityView { content, attachments in  
            do {  
  
                let audioResource = try AudioFileResource.load(named: "calming.wav", in: nil,  
configuration: .init/loadingStrategy: .preload, shouldLoop: true))  
  
                if let speaker {  
  
                    animation = speaker.availableAnimations[0]  
                    audioController = speaker.prepareAudio(audioResource)  
  
                    if let animation {  
                        speaker.playAnimation(animation.repeat())  
                    }  
                    audioController?.play()  
                    content.add(table)  
                }  
            }  
        }  
    }  
}
```

Audio Volume

```
struct ImmersiveView: View {  
  
    @State var audioController: AudioPlaybackController?  
    @State var volume: Double = -30.0  
  
    var body: some View {  
        RealityView { content, attachments in  
            do {  
  
                let audioResource = try AudioFileResource.load(named: "calming.wav", in: nil,  
configuration: .init/loadingStrategy: .preload, shouldLoop: true))  
  
                if let speaker {  
  
                    animation = speaker.availableAnimations[0]  
                    audioController = speaker.prepareAudio(audioResource)  
  
                    audioController?.gain = volume  
                }  
            }  
        }  
    }  
}
```

🏁 Checkpoint 2

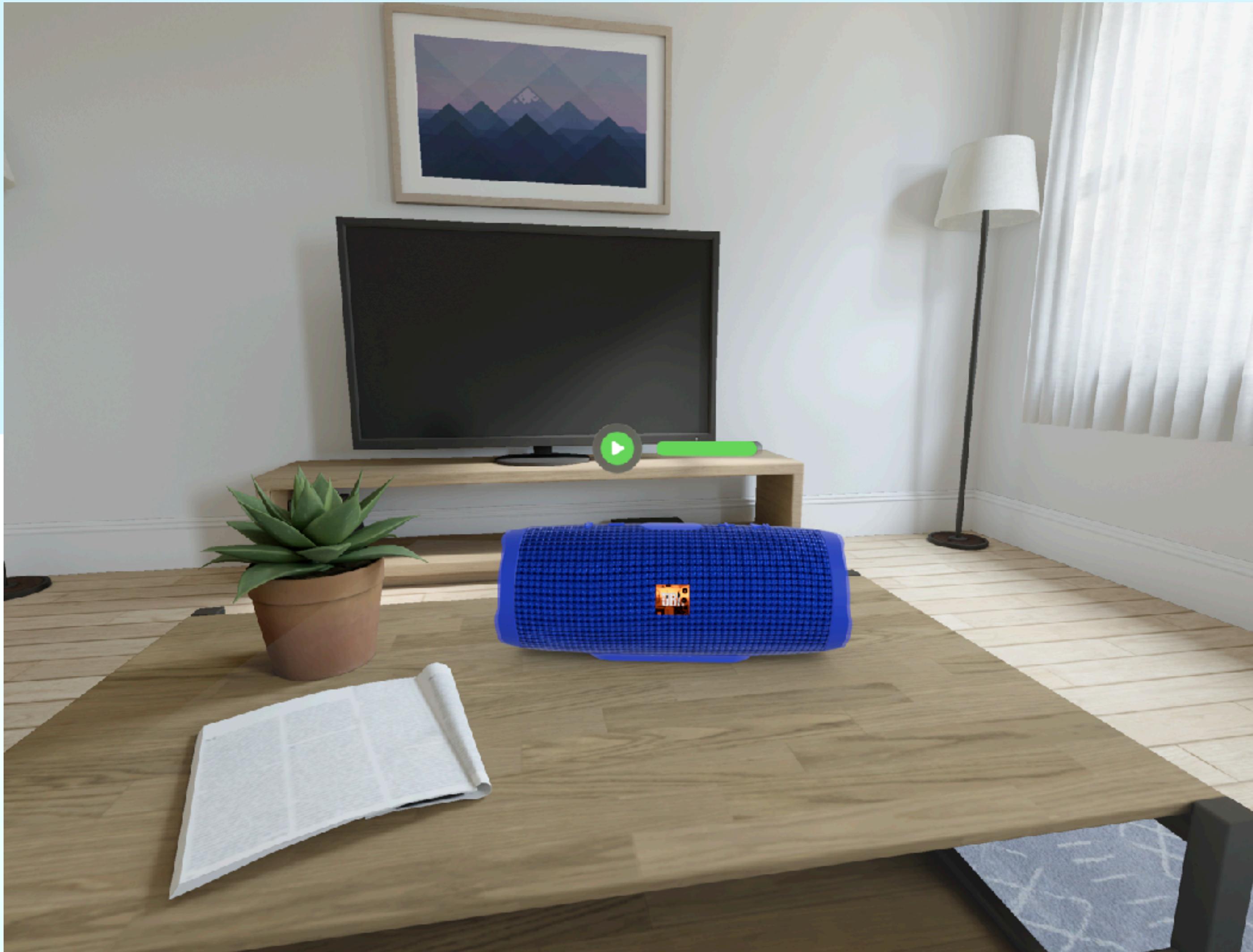
- AnchorEntity
- Animation
- Audio
- Audio volume



The screenshot shows the Xcode interface with the AirSpeaker project open. The left sidebar displays the project structure, including files like AirSpeaker, AirSpeakerApp, ImmersiveView, and various assets. The main editor window shows the ImmersiveView.swift file, which contains Swift code for initializing a speaker, setting up components, and playing audio. The code includes imports for RealityKit and SceneKit, and uses try await to load resources.

```
11 struct ImmersiveView: View {
12
13     @State var speaker: ModelEntity? = nil
14     @State var isDragging = false
15     @State var dragStartPosition: SIMD3<Float> = .zero
16     @State var audioController: AudioPlaybackController?
17     @State var volume: Double = -30.0
18
19     var body: some View {
20         RealityView { content in
21             do {
22                 speaker = try await ModelEntity(named: "jbl_charge")
23                 let environment = try await EnvironmentResource(named: "studio")
24                 let audioResource = try AudioFileResource.load(named: "calming.wav", in: nil, configuration: .init(loadingStrategy: .preload,
25                                                               shouldLoop: true))
26
27                 if let speaker {
28                     let table = { ... }()
29
30                     speaker.scale = [0.00075, 0.00075, 0.00075]
31
32                     speaker.components.set(ImageBasedLightComponent(source: .single(environment)))
33                     speaker.components.set(ImageBasedLightReceiverComponent(imageBasedLight: speaker))
34
35                     let speakerBounds = speaker.model!.mesh.bounds.extents
36                     speaker.components.set(CollisionComponent(shapes: [.generateBox(size: speakerBounds)]))
37                     speaker.components.set(InputTargetComponent())
38                     speaker.components.set(GroundingShadowComponent(castsShadow: true))
39
40                     let animation = speaker.availableAnimations[0]
41                     audioController = speaker.prepareAudio(audioResource)
42                     audioController?.gain = volume
43
44                     speaker.playAnimation(animation.repeat())
45                     audioController?.play()
46
47                     content.add(table)
48
49                 }
50             }
51         }
52     }
53 }
```

Attachment



RealityView

Creating a reality view

```
init(make: (inout RealityViewContent) async -> Void, update: ((inout RealityViewContent) -> Void)?)
```

Creates a RealityView.

Available when Content conforms to View.

```
init<A>(make: (inout RealityViewContent, RealityViewAttachments) async -> Void, update: ((inout RealityViewContent, RealityViewAttachments) -> Void)?, attachments: () -> A)
```

Creates a RealityView.

Available when Content conforms to View.

```
init<P>(make: (inout RealityViewContent) async -> Void, update: ((inout RealityViewContent) -> Void)?, placeholder: () -> P)
```

Creates a RealityView.

Available when Content conforms to View.

```
init<A, P>(make: (inout RealityViewContent, RealityViewAttachments) async -> Void, update: ((inout RealityViewContent, RealityViewAttachments) -> Void)?, placeholder: () -> P, attachments: () -> A)
```

Creates a RealityView.

Available when Content conforms to View.

```
var body: some View
```

The content and behavior of the view.

Attachment

```
struct ImmersiveView: View {  
  
    @State var speaker: ModelEntity? = nil  
    @State var isDragging = false  
    @State var dragStartPosition: SIMD3<Float> = .zero  
    @State var audioController: AudioPlaybackController?  
    @State var volume: Double = -30.0  
  
    var body: some View {  
        RealityView { content in  
            do {...}  
            catch {...}  
        }  
        .gesture(dragGesture)  
    }  
  
    private var dragGesture: some Gesture {...}  
}
```

Attachment

```
struct ImmersiveView: View {  
  
    @State var speaker: ModelEntity? = nil  
    @State var isDragging = false  
    @State var dragStartPosition: SIMD3<Float> = .zero  
    @State var animation: AnimationResource? = nil  
    @State var audioController: AudioPlaybackController?  
    @State var volume: Double = -30.0  
    @State var isPlaying: Bool = false  
  
    var body: some View {  
        RealityView { content, attachments in  
            do {...}  
            catch {...}  
        } update: { content, attachments in  
            // update methods  
        } attachments: {  
            // add attachments here  
        }  
        .gesture(dragGesture)  
    }  
  
    private var dragGesture: some Gesture {...}  
}
```

Attachment

```
attachments: {  
}
```

Attachment

```
attachments: {  
    Attachment(id: "speaker_controls") {  
        Toggle(isOn: $isPlaying) {  
            Label("Play", systemImage:.isPlaying ? "pause.fill" : "play.fill")  
                .font(.largeTitle)  
                .padding()  
        }  
        .toggleStyle(.button)  
        .buttonStyle(.bordered)  
        .tint(.green)  
        .labelStyle(.iconOnly)  
        .padding()  
        .glassBackgroundEffect(in: Circle())  
    }  
}
```

Attachment

```
attachments: {  
    Attachment(id: "speaker_controls") {  
        Toggle(isOn: $isPlaying) {  
            Label("Play", systemImage:.isPlaying ? "pause.fill" : "play.fill")  
                .font(.largeTitle)  
                .padding()  
        }  
        .toggleStyle(.button)  
        .buttonStyle(.bordered)  
        .tint(.green)  
        .labelStyle(.iconOnly)  
        .padding()  
        .glassBackgroundEffect(in: Circle())  
  
        Slider(value: $volume, in: (-60.0)...(0.0))  
            .tint(.green)  
            .frame(maxWidth: 200)  
            .onChange(of: volume) { _, newValue in  
                let vol = Audio.Decibel(floatLiteral: newValue)  
                if let audioController, audioController.isPlaying {  
                    audioController.fade(to: vol, duration: 0)  
                }  
            }  
    }  
}
```

Attachment

```
attachments: {
    Attachment(id: "speaker_controls") {
        HStack {
            Toggle(isOn: $isPlaying) {
                Label("Play", systemImage: isPlaying ? "pause.fill" : "play.fill")
                    .font(.largeTitle)
                    .padding()
            }
            .toggleStyle(.button)
            .buttonStyle(.bordered)
            .tint(.green)
            .labelStyle(.iconOnly)
            .padding()
            .glassBackgroundEffect(in: Circle())
        }

        Slider(value: $volume, in: (-60.0)...(0.0))
            .tint(.green)
            .frame(maxWidth: 200)
            .onChange(of: volume) { _, newValue in
                let vol = Audio.Decibel(floatLiteral: newValue)
                if let audioController, audioController.isPlaying {
                    audioController.fade(to: vol, duration: 0)
                }
            }
        }
    }
}
```

Attachment

```
struct ImmersiveView: View {  
  
    var body: some View {  
        RealityView { content, attachments in  
            do {  
                if let speaker {  
                    speaker.components.set(GroundingShadowComponent(castsShadow: true))  
  
                    content.add(table)  
                }  
            }  
            catch {...}  
        } update: { content, attachments in  
            // update methods  
        } attachments: {  
            Attachments(id: "speaker_controls") {...}  
        }  
    }  
}
```

Attachment

```
struct ImmersiveView: View {  
  
    var body: some View {  
        RealityView { content, attachments in  
            do {  
                if let speaker {  
                    speaker.components.set(GroundingShadowComponent(castsShadow: true))  
  
                    if let attachment = attachments.entity(for: "speaker_controls") {  
                        attachment.position = [0, 0.3, 0]  
                        speaker.addChild(attachment, preservingWorldTransform: true)  
                    }  
                }  
                content.add(table)  
            }  
        }  
        catch {...}  
    } update: { content, attachments in  
        // update methods  
    } attachments: {  
        Attachments(id: "speaker_controls") {...}  
    }  
}
```

Attachment

```
private func animate() {  
    guard let speaker, let animation, let audioController else { return }  
  
    if isPlaying {  
        speaker.playAnimation(animation.repeat())  
        audioController.play()  
    } else {  
        speaker.stopAllAnimations()  
        audioController.stop()  
    }  
}
```

Attachment

```
if let speaker {  
  
    animation = speaker.availableAnimations[0]  
    audioController = speaker.prepareAudio(audioResource)  
  
    if let animation {  
        speaker.playAnimation(animation.repeat())  
    }  
    audioController?.play()  
  
    content.add(table)  
}
```

Attachment

```
if let speaker {  
    animation = speaker.availableAnimations[0]  
    audioController = speaker.prepareAudio(audioResource)  
    animate()  
    content.add(table)  
}
```

Attachment

```
struct ImmersiveView: View {  
  
    @State var speaker: ModelEntity? = nil  
    @State var isDragging = false  
    @State var dragStartPosition: SIMD3<Float> = .zero  
    @State var audioController: AudioPlaybackController?  
    @State var volume: Double = -30.0  
    @State var isPlaying: Bool = false  
  
    var body: some View {  
        RealityView { content, attachments in  
            do {...}  
            catch {...}  
        } update: { content, attachments in  
            // update methods  
        } attachments: {  
            Attachment(id: "speaker_controls") {...}  
        }  
        .gesture(dragGesture)  
    }  
  
    private var dragGesture: some Gesture {...}  
}
```

Attachment

```
struct ImmersiveView: View {  
  
    @State var speaker: ModelEntity? = nil  
    @State var isDragging = false  
    @State var dragStartPosition: SIMD3<Float> = .zero  
    @State var audioController: AudioPlaybackController?  
    @State var volume: Double = -30.0  
    @State var isPlaying: Bool = false  
  
    var body: some View {  
        RealityView { content, attachments in  
            do {...}  
            catch {...}  
        } update: { content, attachments in  
            animate()  
        } attachments: {  
            Attachment(id: "speaker_controls") {...}  
        }  
        .gesture(dragGesture)  
    }  
  
    private var dragGesture: some Gesture {...}  
}
```



Final Checkpoint !!

- Attachment

The screenshot shows the Xcode interface with the following details:

- Title Bar:** AirSpeaker start
- Project Navigator:** Shows the project structure with files like AirSpeaker, AirSpeakerApp, ImmersiveView (selected), calming, jbl_charge, studio.skybox, Assets, Info, Preview Content, and AirSpeakerTests.
- Search Bar:** AirSpeakerApp ImmersiveView
- Editor Area:** Displays the ImmersiveView.swift code. The code defines a struct ImmersiveView that contains logic for speaker controls, animations, and volume management using SwiftUI's ViewBuilder and State.
- Top Status Bar:** Clean Succeeded | Today at 1:21 AM
- Bottom Status Bar:** Line: 55 Col: 39

```
11 struct ImmersiveView: View {
12     var body: some View {
13         if let attachment = attachments.entity(for: "speaker_controls") {
14             attachment.position = [0, 0.3, 0]
15             speaker.addChild(attachment, preservingWorldTransform: true)
16         }
17
18         animation = speaker.availableAnimations[0]
19         audioController = speaker.prepareAudio(audioResource)
20         audioController?.gain = volume
21
22         animate()
23
24         content.add(table)
25     }
26
27     catch {
28         print("Error loading the model")
29     }
30
31     update: { content, attachments in
32         animate()
33     }
34     attachments: {
35         Attachment(id: "speaker_controls") {
36             HStack {
37                 Toggle(isOn: $isPlaying) {
38                     Label("Play", systemImage:.isPlaying ? "pause.fill" : "play.fill")
39                         .font(.largeTitle)
40                         .padding()
41                 }
42                 .toggleStyle(.button)
43                 .buttonStyle(.bordered)
44                 .tint(.green)
45                 .labelStyle(.iconOnly)
46                 .padding()
47                 .glassBackgroundEffect(in: Circle())
48
49                 Slider(value: $volume, in: (-60.0)...(0.0))
50                     .tint(.green)
51                     .frame(maxWidth: 200)
52                     .onChange(of: volume) { _, newValue in
53                         let vol = Audio.Decibel(floatLiteral: newValue)
54                         if let audioController, audioController.isPlaying {
55                             audioController.fade(to: vol, duration: 0)
56                         }
57                     }
58
59             }
60         }
61     }
62
63     content: {
64         HStack {
65             Text("Volume: ")
66             Text(String(format: "%.1f", volume))
67         }
68     }
69
70     animation: Animation.easeIn
71
72     volume: Double
73     isPlaying: Bool
74     speaker: Speaker
75     audioController: AudioController?
76     attachments: [Attachment]
77     table: Table
78     content: Content
79     animation: Animation
80     volume: Double
81     isPlaying: Bool
82     speaker: Speaker
83     audioController: AudioController?
84 }
```

RealityView / ModelEntity / Environment Resource / AudioPlaybackController /
AudioFileResource / AnchorEntity / ImageBasedLightComponent /
CollisionComponent / InputTargetComponent / AnimationResource/
ImageBasedLightReceiverComponent / GroundingShadowComponent /
Attachment / gain / prepareAudio / SIMD3<Float> / persistentSystemOverlays /
glassBackgroundEffect / availableAnimations / DragGesture / play /
translation3D / targetedToAnyEntity / isPlaying / position

You've made it

100
==

非常感谢