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

0️⃣➡️🦸🏻‍♀️🦸🏽‍♀️🦸🏻‍♀️🦸

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

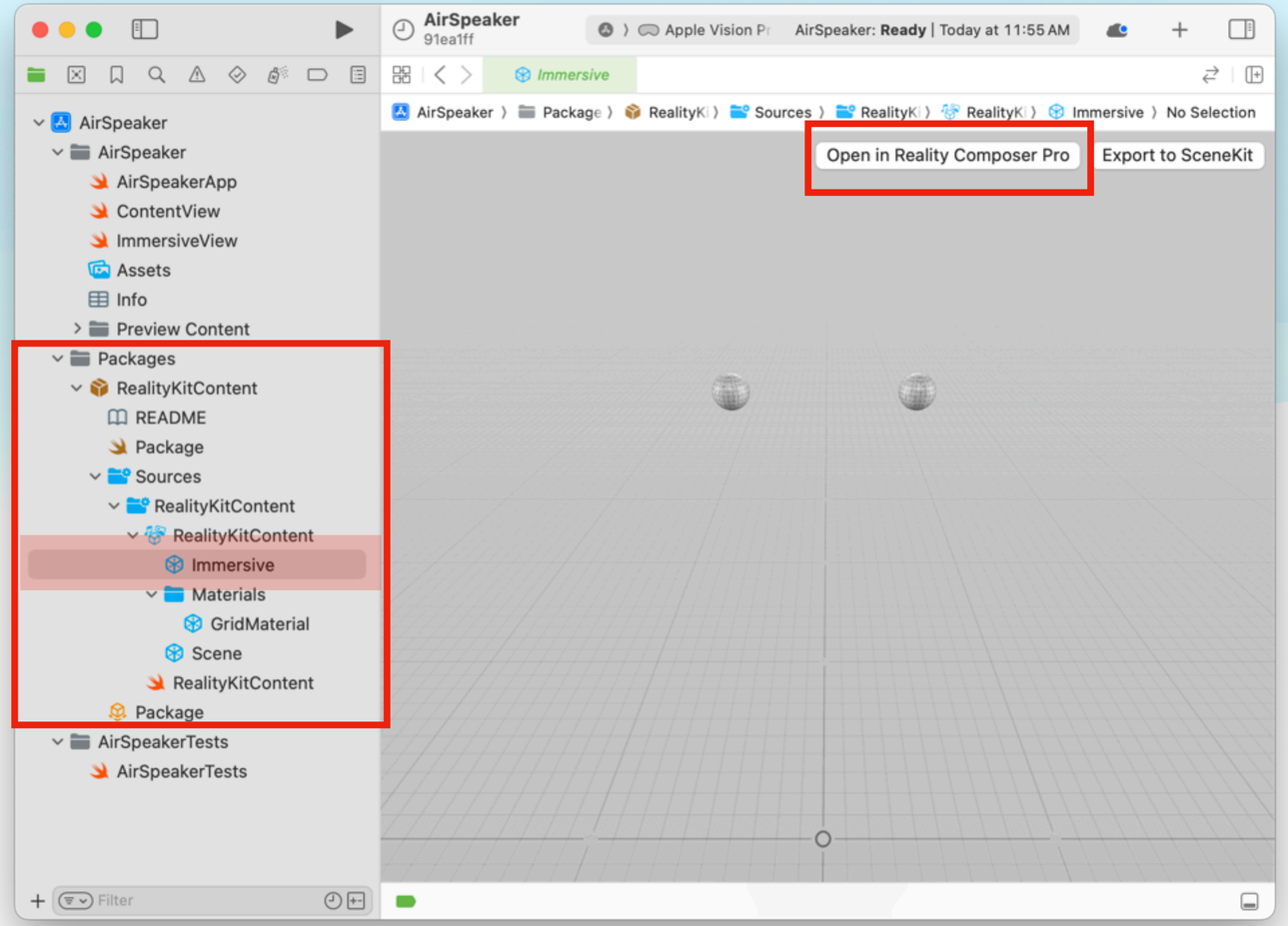
What are we building ?

🎶🎵🔈🔉🔊🎵🎶

Build Small Things!

🏁 Start 🚀

Reality Composer Pro



🕵🏻‍♀️ Clean up

🗑 RealityKitContent package
🗑 ContentView.swift

```
//  AirSpeakerApp.swift

@main
struct AirSpeakerApp: App {
    var body: some Scene {
        WindowGroup {
            ContentView()
        }.windowStyle(.volumetric)

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

Cannot find 'ContentView' in scope

👩🏻‍🔍 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

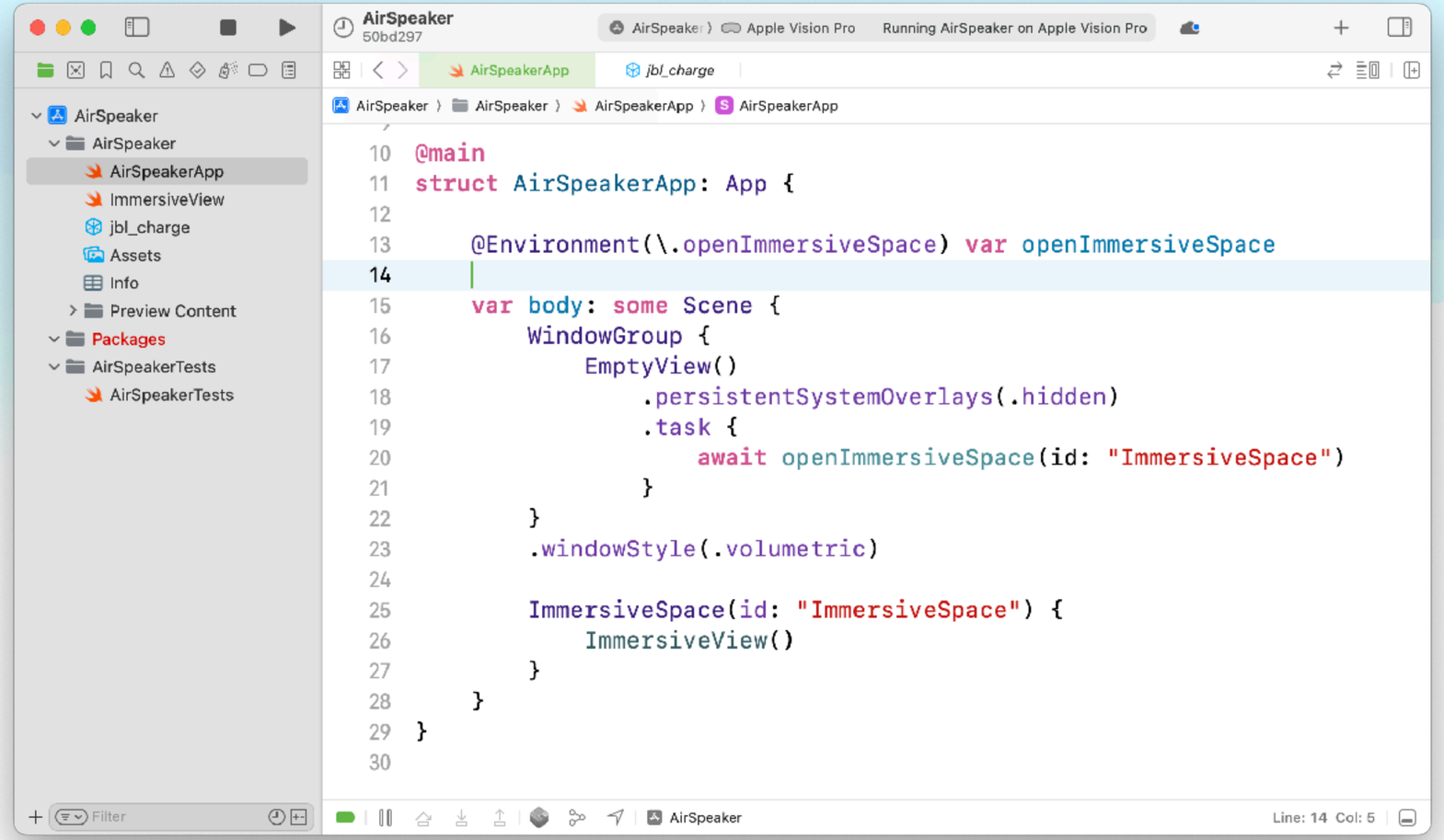
```
import RealityKitContent

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

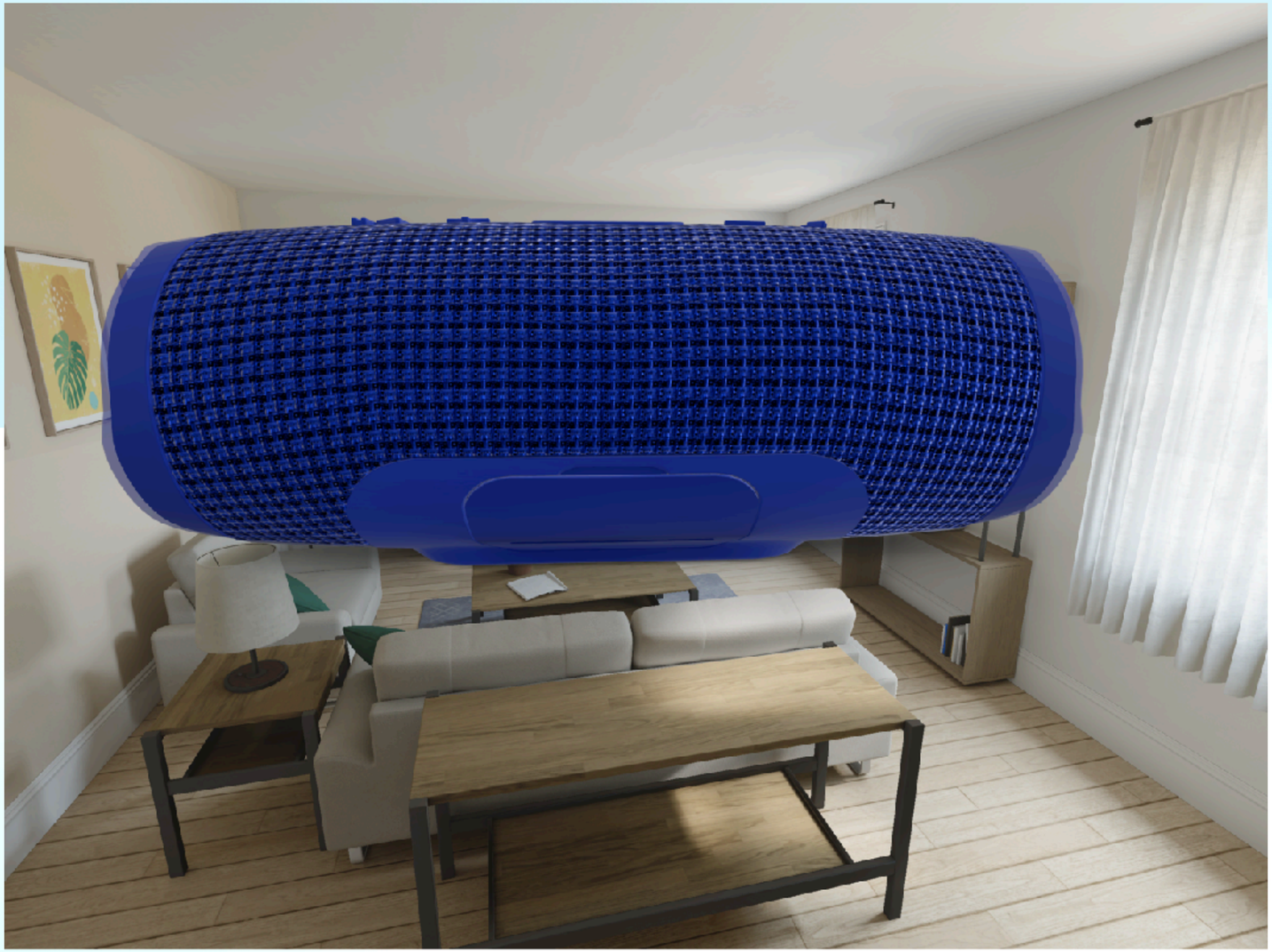
- Preparation
- Load ModelEntity



The 3D Model

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

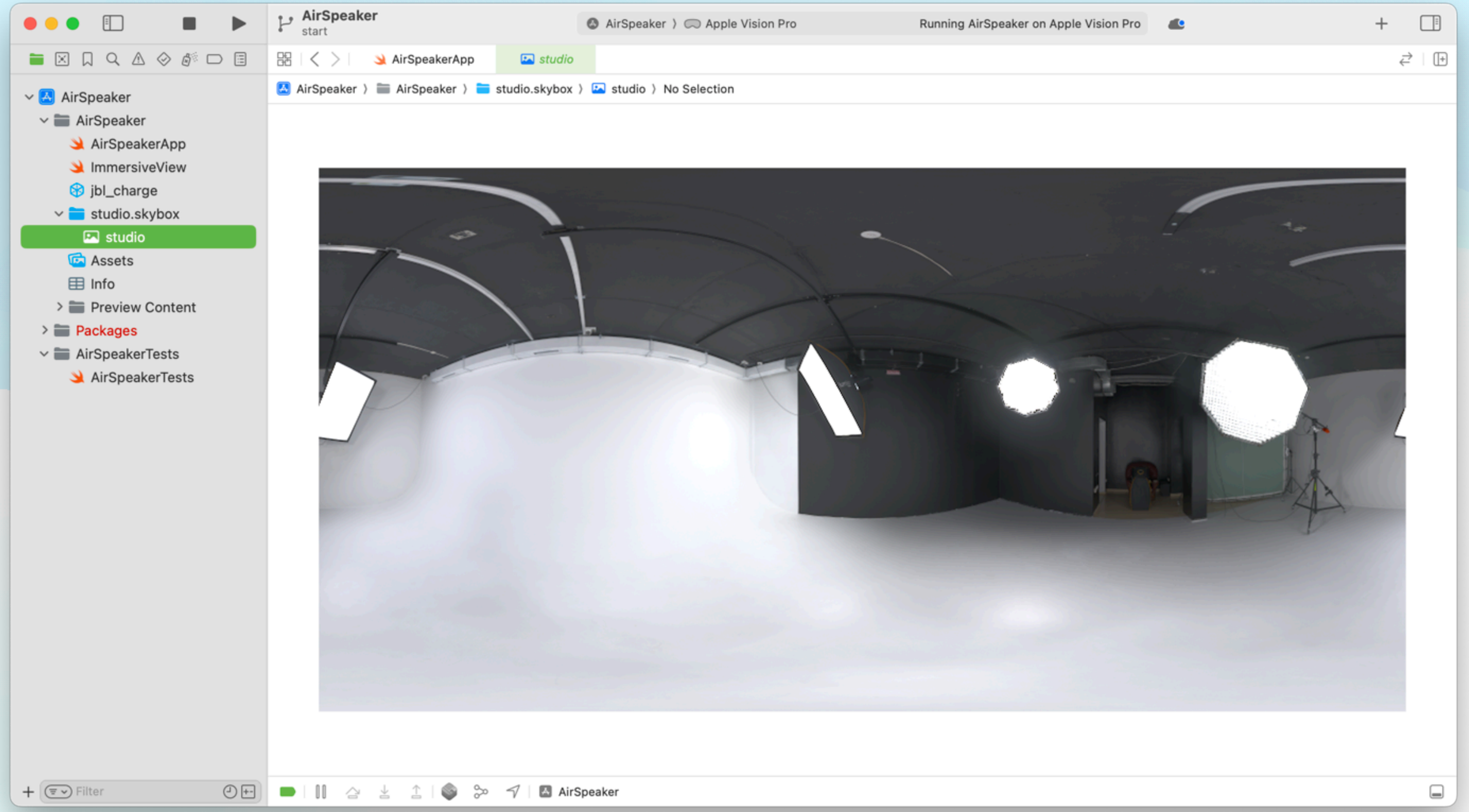
Load a ModelEntity



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



https://polyhaven.com/a/studio_small_08

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 {...}
```

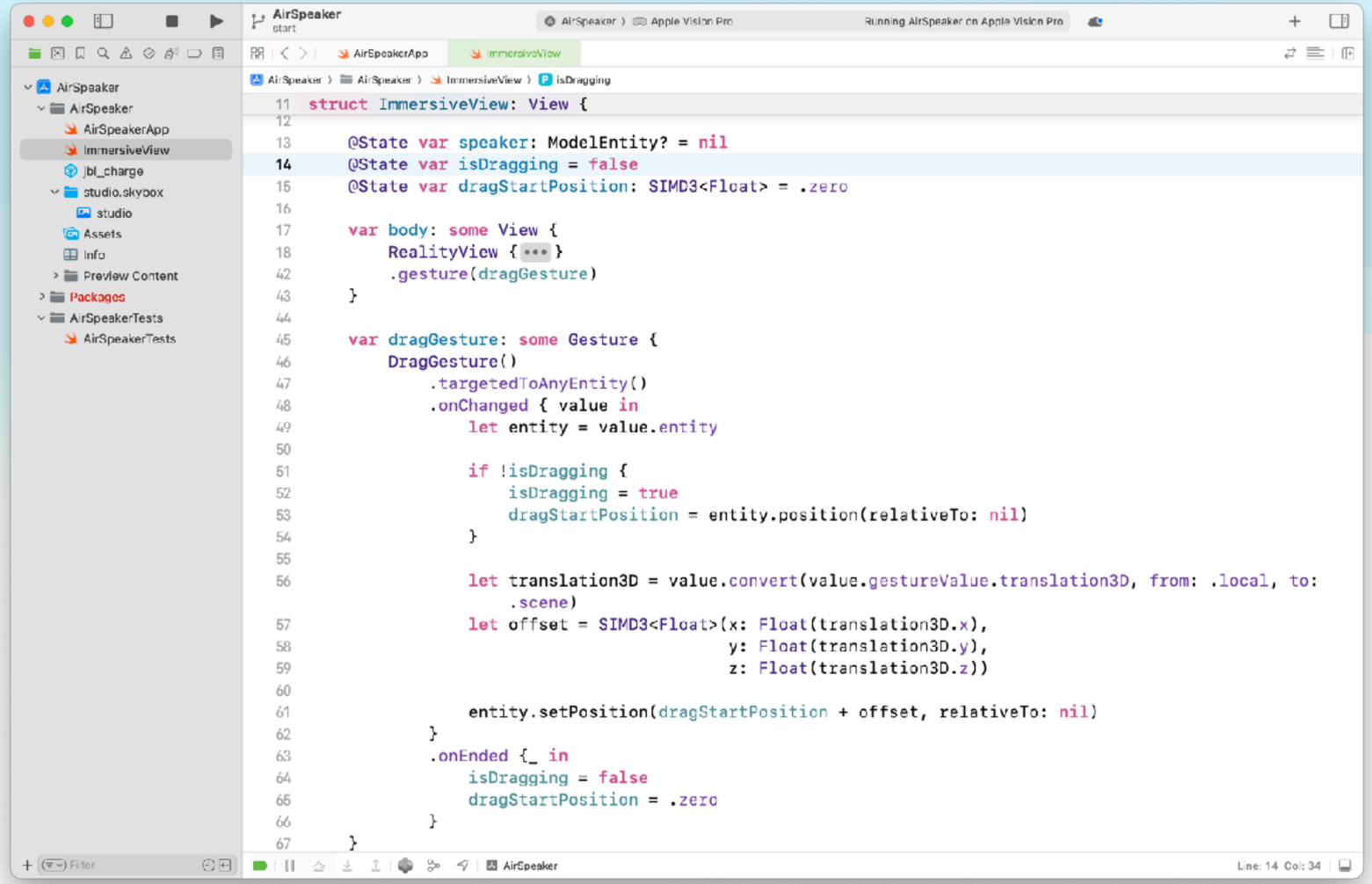
Handle drag gestures

```
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
            }
    }
```

https://developer.apple.com/documentation/realitykit/transforming-realitykit-entities-with-gestures

🏁 Checkpoint

- Scaling
- ImageBasedLight
- Shadow
- Drag gestures




```
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 { ••• }
42          .gesture(dragGesture)
43      }
44
45      var dragGesture: some Gesture {
46          DragGesture()
47              .targetedToAnyEntity()
48              .onChanged { value in
49                  let entity = value.entity
50
51                  if !isDragging {
52                      isDragging = true
53                      dragStartPosition = entity.position(relativeTo: nil)
54                  }
55
56                  let translation3D = value.convert(value.gestureValue.translation3D, from: .local, to:
                        .scene)
57                  let offset = SIMD3<Float>(x: Float(translation3D.x),
58                                            y: Float(translation3D.y),
59                                            z: Float(translation3D.z))
60
61                  entity.setPosition(dragStartPosition + offset, relativeTo: nil)
62              }
63              .onEnded {_ in
64                  isDragging = false
65                  dragStartPosition = .zero
66              }
67      }
```

0️⃣ ➡️ 🦸🏽‍♀️🦸🏽‍♀️🦸🏻‍♀️🦸

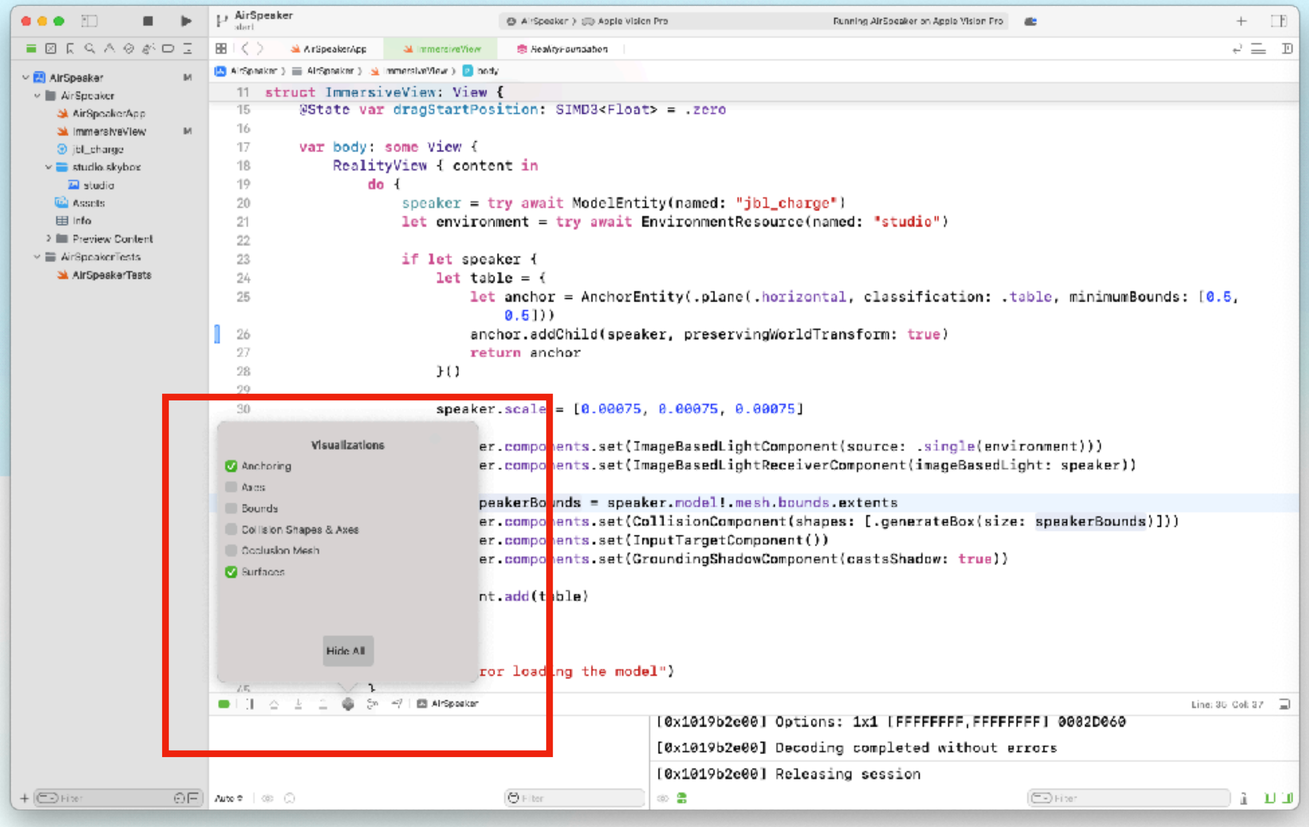
Anchoring ⚓ 🚢

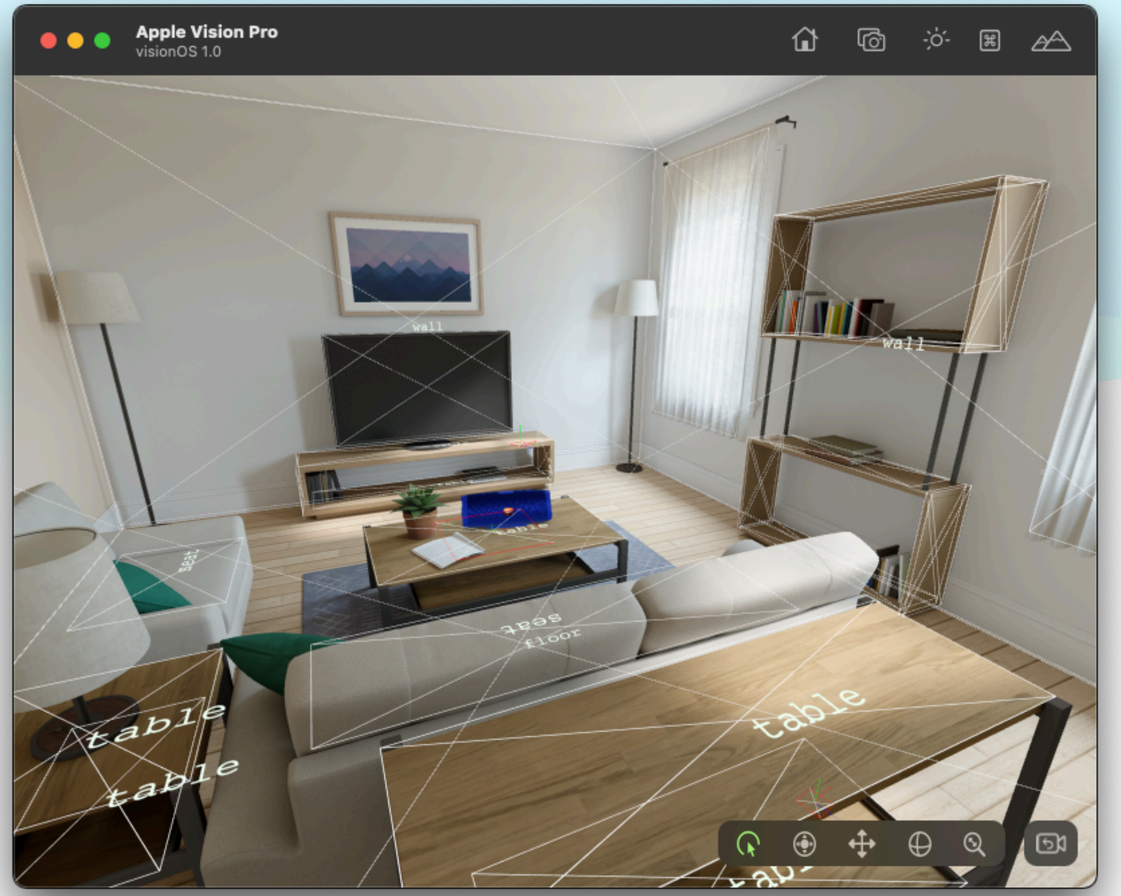
```
RealityView { content in
    do {
        speaker = try await ModelEntity(named: "jbl_charge")
        let environment = try await EnvironmentResource(named: "studio")

        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 ⚓ 🚢


struct ImmersiveView: View {
@State var dragStartPosition: SIMD3<Float> = .zero
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 {
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]
Visualizations
Anchoring
Axes
Bounds
Collision Shapes & Axes
Occlusion Mesh
Surfaces
Hide All
[0x1019b2e00] Options: 1x1 [FFFFFFFF,FFFFFFFF] 0002D060
[0x1019b2e00] Decoding completed without errors
[0x1019b2e00] Releasing session


Apple Vision Pro
visionOS 1.0
wall
wall
table
table
table

Animation

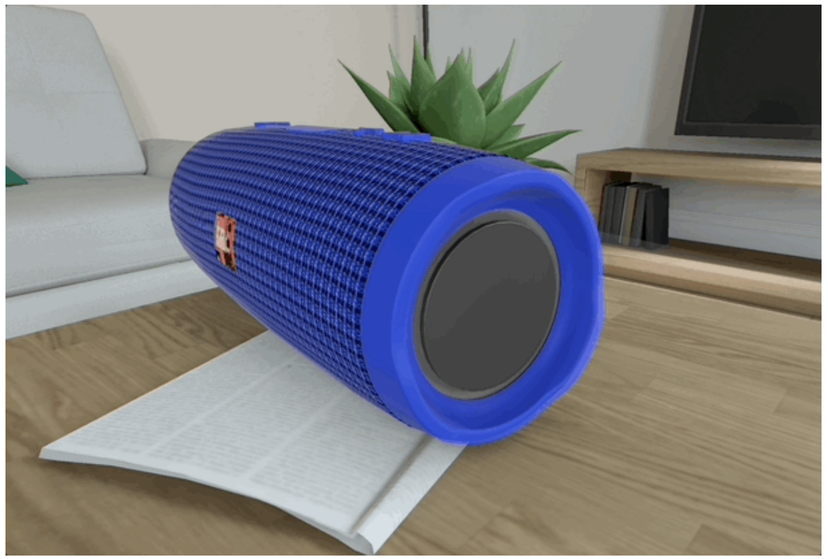
```
struct ImmersiveView: View {

    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

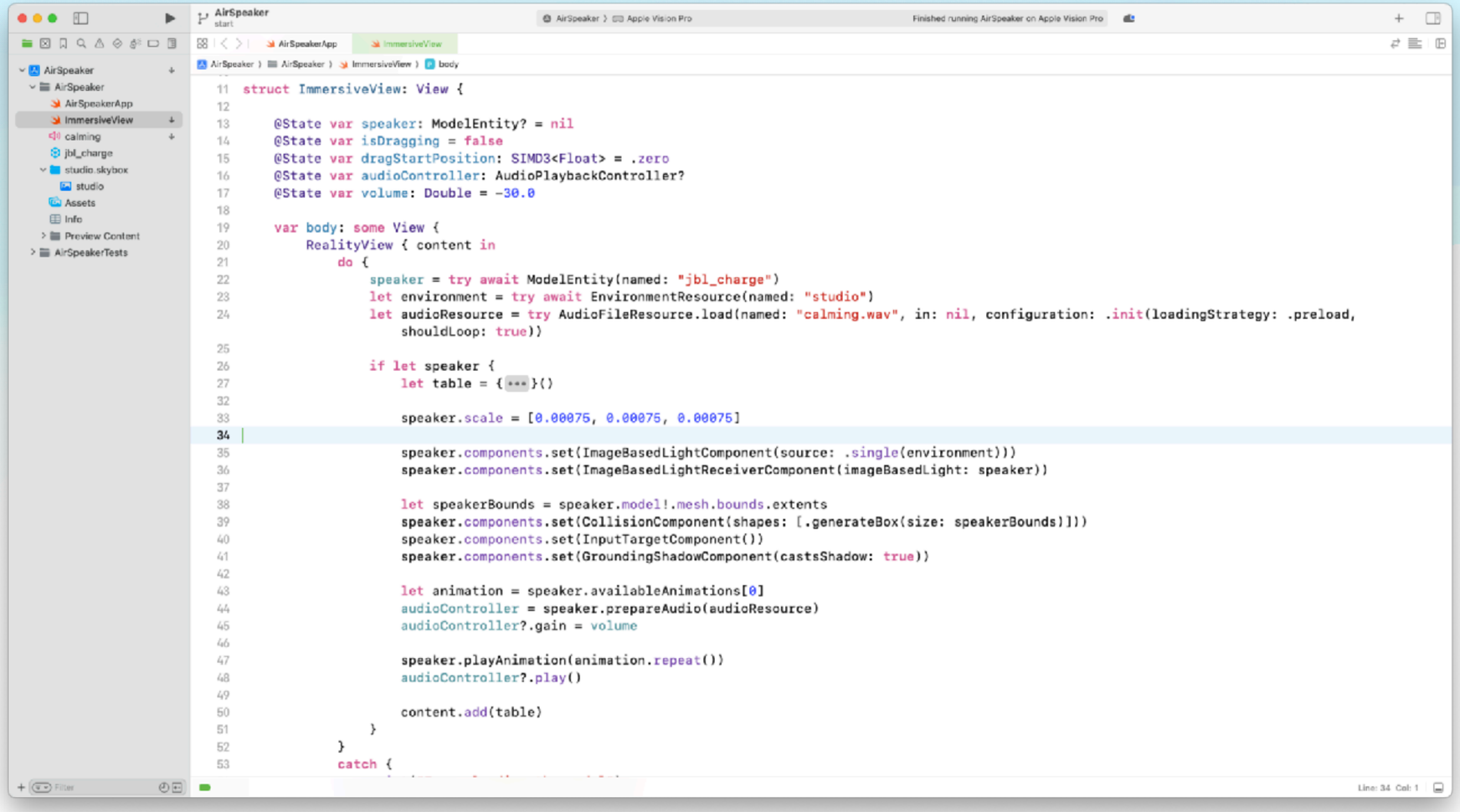
```
struct ImmersiveView: View {

    @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

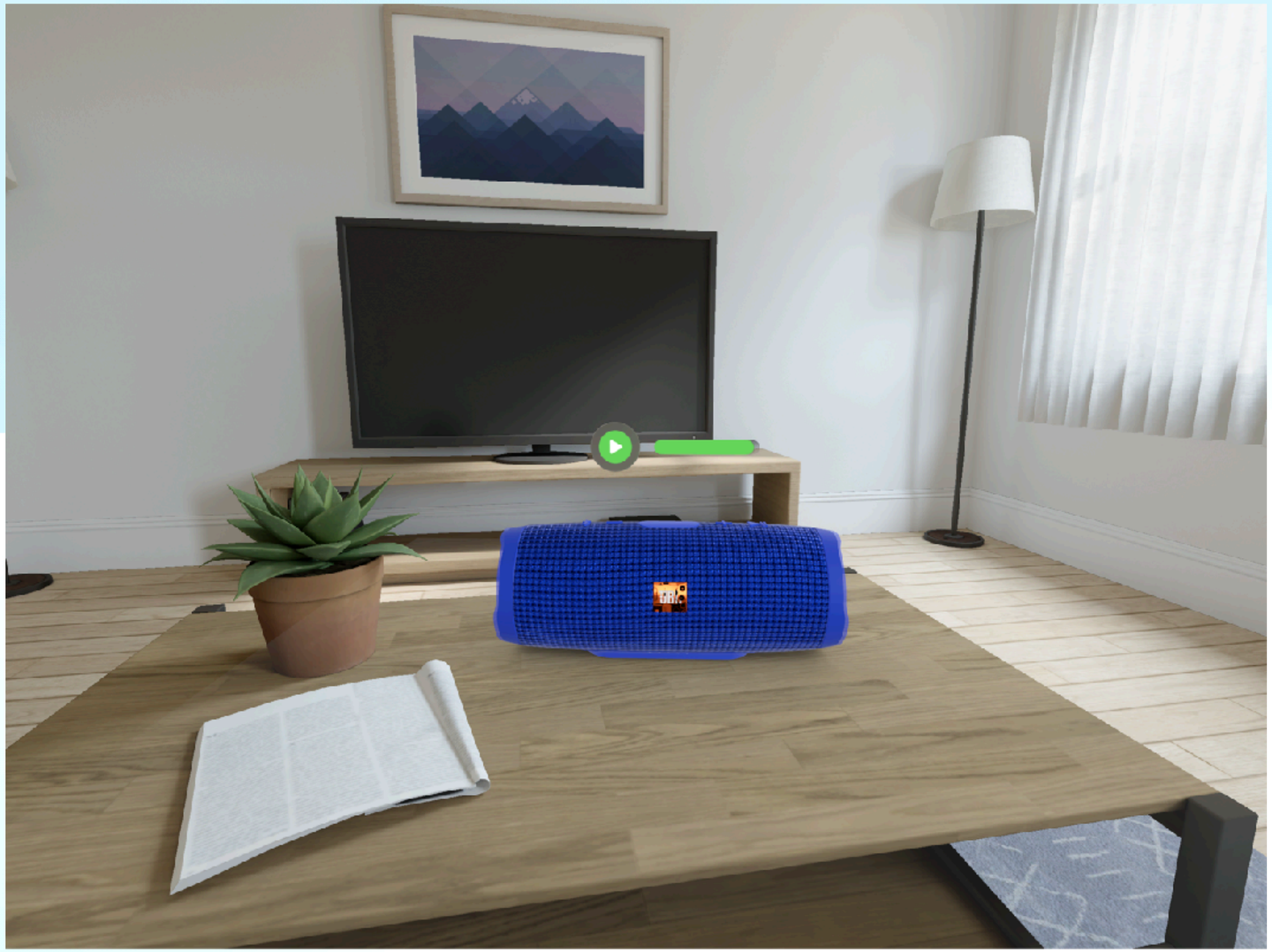
```
struct ImmersiveView: View {

    @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

                        content.add(table)
                    }
                }
```

🏁 Checkpoint 2

- AnchorEntity
- Animation
- Audio
- Audio volume



Attachment



RealityView

Creating a reality view

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

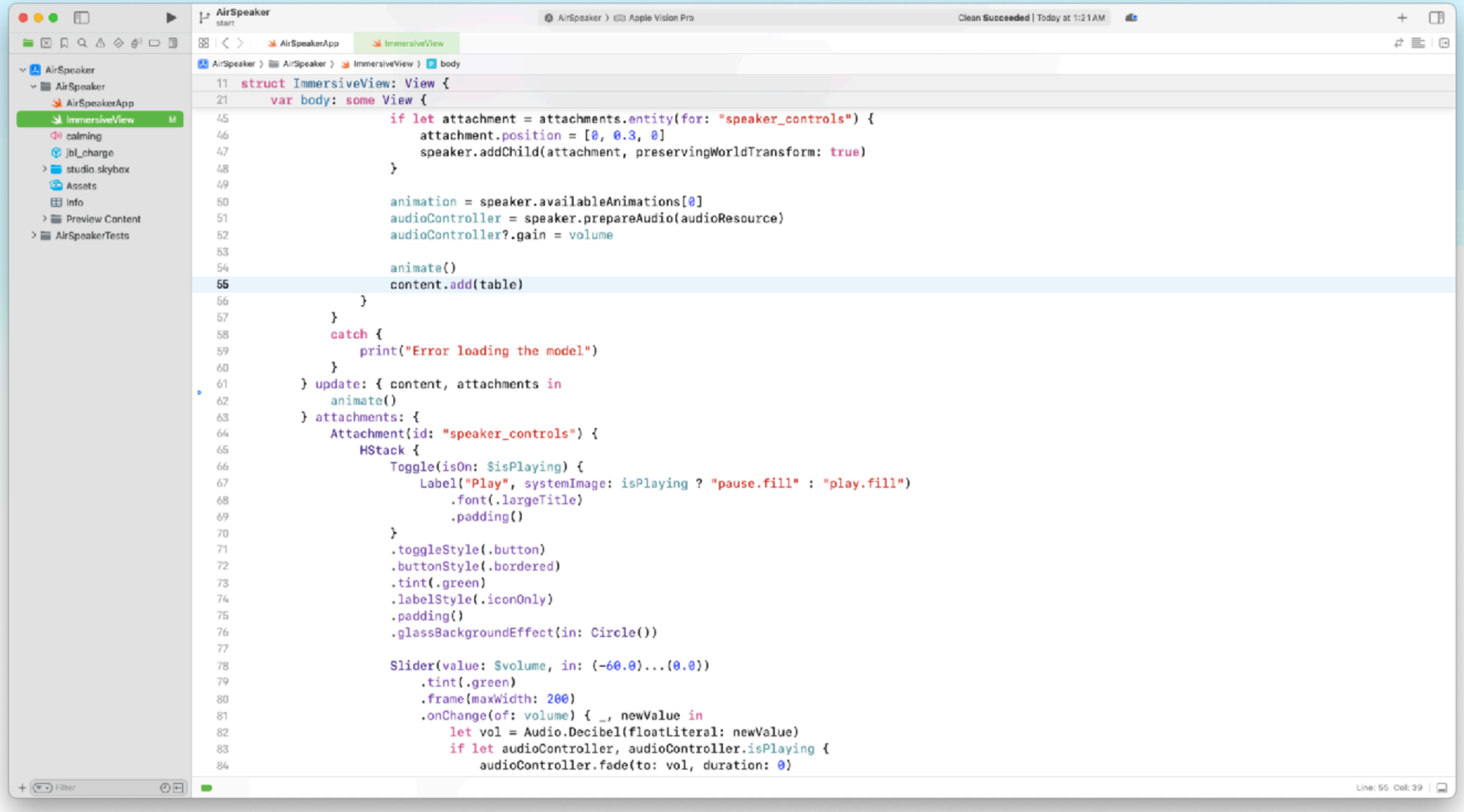
```
struct ImmersiveView: View {

    @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



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

非常感谢