Sample Code

Playing spatial audio

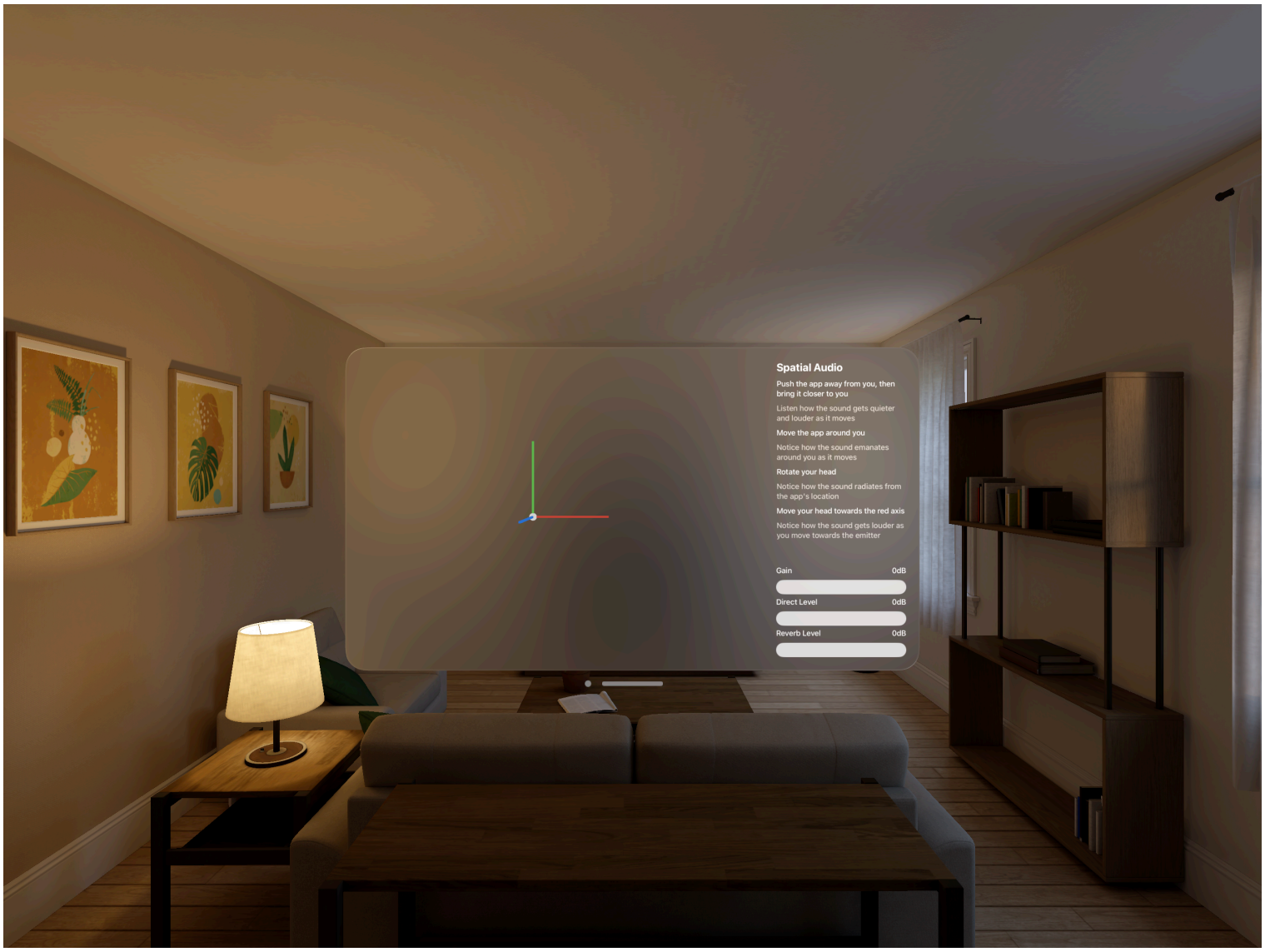
Create and adjust spatial audio in visionOS with RealityKit.

Download

visionOS 2.0+ | Xcode 16.0+

Overview

This sample demonstrates how to load and play a spatial audio file in a visionOS app with the `SpatialAudioComponent`. As the following image shows, you can use this component to configure how an entity emits sounds into a person's environment:



Create the axis visualizer

Using existing RealityKit shapes, the sample creates an axis visualizer in the app's main view to represent the x, y, and z axes of the audio source:

```
import RealityKit

struct AxisVisualizer {
    static func make() -> Entity {
        /// The entity that contains four different meshes.
        let entity = Entity()

        /// The width, length, and radius values that each mesh uses.
        let width: Float = 0.0025
        let length: Float = 0.1
        let radius: Float = 0.005

        // ...
```

```
    }
}
```

The structure uses a `make()` method to create the entity that contains the axis mesh.

To create the representation of the x-axis, the app creates a box mesh with `generateBox(size:cornerRadius:)` and a red `UnlitMaterial`, then combines the two with `ModelEntity`:

```
static func make() -> Entity {
    // ...

    /// The box for the x-axis.
    let xAxisMesh = MeshResource.generateBox(size: [length, width, width])

    /// The unlit red material.
    let xAxisMaterial = UnlitMaterial(color: .systemRed)

    /// The entity with the box and material that represents the x-axis.
    let xAxisEntity = ModelEntity(mesh: xAxisMesh, materials: [xAxisMaterial])

    // Set the position of the x-axis entity in 3D space.
    xAxisEntity.position = [0.5 * length, 0, 0]

    // Add the x-axis to the parent entity.
    entity.addChild(xAxisEntity)

    // ...
}
```

The app follows similar steps to create the representation of the y and z axes, by adjusting the color of the material, the position of the entity, and the corresponding vector of three scalar values representing the width, height, and depth of the box.

To create an origin point and complete the visualizer, the app creates a white sphere at the default position, using `generateSphere(radius:)`:

```
static func make() -> Entity {
    // ...

    /// The sphere for the origin point.
    let originMesh = MeshResource.generateSphere(radius: radius)
```

```
    /// The unlit white material.
    let originMaterial = UnlitMaterial(color: .white)

    /// The entity with the sphere and white material that represents the origin poi
    let originEntity = ModelEntity(mesh: originMesh, materials: [originMaterial])

    // Add the origin entity to the main entity.
    entity.addChild(originEntity)

    return entity
}
```

Create a decibel slider

To adjust the decibels for the gain, the direct level, and the reverb level of the audio source, the app creates the `DecibelSlider` view that the app adds to the main body view. The view contains a `name` property, which represents the name of the property that it controls, and a `value` variable, which stores the decibel value:

```
import SwiftUI

/// A view that formats as a slider to adjust decibel values.
struct DecibelSlider: View {
    /// The name of the value that changes.
    let name: String

    /// The binding to a numerical double that stores the decibel value.
    let value: Binding<Double>

    var body: some View {
        VStack {
            HStack {
                Text(name)
                Spacer()
                Text(value.wrappedValue.formatted(.number.precision(.fractionLength(
                    .monospacedDigit()
            }
            /// The slider with a range of -60 to 0.
            Slider(value: value, in: -60 ... 0)
        }
    }
}
```

Using a `Slider` within the view, a person can control the `value` property from the bounded linear range of values between -60 and 0.

Set up the main window

To create the main view, the app combines the following subviews:

- `audioSource`
- `description`
- `config`

```
import SwiftUI
import RealityKit

struct SpatialAudioView: View {
    /// The new entity to contain the audio sample.
    let entity = Entity()

    /// The gain value of the audio source.
    @State private var gain: Audio.Decibel = .zero

    /// The direct signal that emits from the audio source.
    @State private var directLevel: Audio.Decibel = .zero

    /// The reverb of the audio source.
    @State private var reverbLevel: Audio.Decibel = .zero

    var body: some View {
        HStack {
            audioSource

            VStack {
                description
                Spacer()
                configuration
            }
            .padding(30)
            .frame(width: 350)
        }
    }
```

```
    // ...
}
```

The view creates `entity` to hold the audio sample. The `gain`, `directLevel`, and `reverbLevel` properties represent the default values for the audio source.

Play the spatial audio

When the necessary properties are in place, the `audioSource` view loads the audio file and configures it for continuous playback. Then the app sets up `entity` with a `SpatialAudioComponent` to play the audio in the reality view:

```
var audioSource: some View {
    RealityView { content in
        // Add the entity to the `RealityView`.
        content.add(entity)

        /// The name of the audio source.
        let audioName: String = "FunkySynth.m4a"

        /// The configuration to loop the audio file continuously.
        let configuration = AudioFileResource.Configuration(shouldLoop: true)

        // Load the audio source and set its configuration.
        guard let audio = try? AudioFileResource.load(
            named: audioName,
            configuration: configuration
        ) else {
            print("Failed to load audio file.")
            return
        }

        /// The focus for the directivity of the spatial audio.
        let focus: Double = 0.5

        // Add a spatial component to the entity that emits in the forward direction
        entity.spatialAudio = SpatialAudioComponent(directivity: .beam(focus: focus)

        // Set the entity to play audio.
        entity.playAudio(audio)
    }
```

```
    // ...
}
```

After adding the entity to the reality view, the app attaches the `.onAppear` and `.onChange` modifiers to spawn the axis visualizer. Then the app enables the view to modify the entity's `gain`, `directLevel`, and `reverbLevel` by adjusting the corresponding representation values:

```
var audioSource: some View {
    RealityView { content in

        // ...

    }
    // Create a 3D axis representation and add it as a child.
    .onAppear { entity.addChild(AxisVisualizer.make()) }

    // Enable the view to change the gain parameter.
    .onChange(of: gain) { entity.spatialAudio?.gain = gain }

    // Enable the view to change the direct level parameter.
    .onChange(of: directLevel) { entity.spatialAudio?.directLevel = directLevel }

    // Enable the view to change the reverb parameter.
    .onChange(of: reverbLevel) { entity.spatialAudio?.reverbLevel = reverbLevel }
}
```

Showcase text descriptions

The sample creates the `description` view to display the collection of texts and guide people through the audio experience in the app:

```
var description: some View {
    VStack(alignment: .leading, spacing: 12) {
        Text("Spatial Audio")
            .font(.title)

        Text("Push the app away from you, then bring it closer to you")
        Text("Notice how the sound gets quieter and louder as it moves")
            .foregroundStyle(.secondary)

        Text("Move the app around you")
```

```
        Text("Notice how the sound emanates around you as it moves")
            .foregroundStyle(.secondary)

        Text("Rotate your head")
        Text("Notice how the sound radiates from the app's location")
            .foregroundStyle(.secondary)

        Text("Move your head towards the red axis")
        Text("Notice how the sound gets louder as you move towards the emitter")
            .foregroundStyle(.secondary)
    }
}
```

Control the spatial audio

To adjust the `gain`, `directLevel`, and `reverbLevel` properties, the sample implements the `configuration` view, adding the `decibelSlider` that allows people to drag to adjust the values:

```
var configuration: some View {
    VStack {
        /// The slider to control the gain value.
        DecibelSlider(name: "Gain", value: $gain)

        /// The slider to control the direct level value.
        DecibelSlider(name: "Direct Level", value: $directLevel)

        /// The slider to control the reverb level.
        DecibelSlider(name: "Reverb Level", value: $reverbLevel)
    }
}
```

Related to

`struct` SpatialAudioComponent

A component that configures how sounds emit from an entity into a person's environment.