Sample Code

Accessing the main camera

Add camera-based features to enterprise apps.

Download

visionOS 26.0+ | Xcode 26.0+

Overview

This sample code project demonstrates how to use ARKit to access and display the main camera feed in your visionOS app. You can use this functionality to implement computer vision-powered experiences or livestreaming in your enterprise app. For instance, support technicians can livestream their surroundings to remote experts for improved guidance.

Configure the sample code project

Replace `Enterprise.license` with your license file. The sample app requires a valid license file to display the main camera.

Request the entitlement

Main camera access is a part of enterprise APIs for visionOS, a collection of APIs that unlock capabilities for enterprise customers. To use main camera access, you need to apply for the `Main camera access` entitlement. For more information, including how to apply for this entitlement, see Building spatial experiences for business apps with enterprise APIs for visionOS.

Add usage descriptions for ARKit data access

To help protect people's privacy, visionOS limits app access to cameras and other sensors in Apple Vision Pro. You need to add an `NSMainCameraUsageDescription` to your app's information property list file to provide a usage description that explains how your app uses the data those sensors provide. People see this description when your app prompts for access to camera data.

> **Note**
>
> Prior to visionOS 26, `CameraFrameProvider` is only available in an immersive space. See Setting up access to ARKit data to learn more about opening an immersive space and requesting authorization for ARKit data access. To learn more about best practices for privacy, see Adopting best practices for privacy and user preferences.

Access and display main camera frames

The code example below accesses and displays the main camera feed, which is a stereo camera that consists of left and right cameras. A person can configure the app to display the left, right, or combined stereo frames, with or without rectification, at one of two resolutions.

The `MainCameraView` renders two instances of `CameraFrameView`, one to display a preview of the left camera feed and another to display a preview of the right camera feed. The left and right previews are instances of `AVSampleBufferDisplayLayer`, and `CameraSessionManager` manages their updates. `CameraSessionManager` accesses camera frames using ARKit and renders them to their respective display layers. The `MainCameraView` uses a `task(priority:_:)` modifier to run `CameraSessionManager`.

```
struct MainCameraView: View {
    @State private var sessionManager = CameraSessionManager()

    var body: some View {
        Group {
            if CameraSessionManager.isSupported == false {
                // ...
            } else {
                HStack {
                    CameraFrameView(preview: sessionManager.leftPreview)
                    CameraFrameView(preview: sessionManager.rightPreview)
                }
            }
        }
        .task {
            await sessionManager.run()
        }
```

```
    }
}
```

To access the main camera, `CameraSessionManager` starts an `ARKitSession` with a `CameraFrameProvider`, and then requests `CameraFrameProvider.CameraFrameUpdates` in the format that the person using the app specifies.

```
final class CameraSessionManager {

    // ...

    /// Begin reading and rendering the main camera's frames.
    func run() async {
        await withTaskGroup(of: Void.self) { group in
            group.addTask {
                await self.runCameraFrameProvider()
            }

            // ...
        }
    }

    private func runCameraFrameProvider() async {
        let arkitSession = ARKitSession()

        // ...

        let cameraFrameProvider = CameraFrameProvider()
        try? await arkitSession.run([cameraFrameProvider])

        // ...

        // See the next section for the implementation of `observeCameraFrameUpdates
        await observeCameraFrameUpdates()
    }
}
```

ARKit delivers a stream of `CameraFrame` instances, and each frame includes a `CameraFrame.Sample`. As each `CameraFrame` arrives, `CameraSessionManager` updates its left and right `CameraFeed` instances using the respective `CameraFrame.Sample`.

```
final class CameraSessionManager {
    //...

    private var leftCameraFeed = CameraFeed()
    private var rightCameraFeed = CameraFeed()

    // ...

    /// The layer displaying the left camera preview.
    var leftPreview: AVSampleBufferDisplayLayer {
        leftCameraFeed.preview
    }

    /// The layer displaying the right camera preview.
    var rightPreview: AVSampleBufferDisplayLayer {
        rightCameraFeed.preview
    }

    //...

    private func observeCameraFrameUpdates() async {
        guard let cameraFrameProvider else { return }

        // Find the `CameraVideoFormat` that corresponds to the `CameraConfiguration
        let formats = CameraVideoFormat
            .supportedVideoFormats(for: .main, cameraPositions: configuration.camera
            .filter({ $0.cameraRectification == configuration.cameraRectification })

        // Find the resolution format.
        let desiredFormat = isHighResolution ?
        formats.max { $0.frameSize.height < $1.frameSize.height }
        : formats.min { $0.frameSize.height < $1.frameSize.height }

        // Request an asynchronous sequence of camera frames.
        guard let desiredFormat,
              let cameraFrameUpdates = cameraFrameProvider.cameraFrameUpdates(for: c
            return
        }

        for await cameraFrame in cameraFrameUpdates {
            if let leftSample = cameraFrame.sample(for: .left) {
                try? await leftCameraFeed.update(using: leftSample)
            }
```

```
            if let rightSample = cameraFrame.sample(for: .right) {
                try? await rightCameraFeed.update(using: rightSample)
            }
        }
    }
}
```

The `CameraFeed` class updates its `preview`, an instance of `AVSampleBufferDisplayLayer`, by rendering the `CameraFrame.Sample` to the display layer. This is the same instance of `AVSampleBufferDisplayLayer` that `MainCameraView` displays.

```
final class CameraFeed {
    /// A preview layer that presents the captured video frames.
    let preview = AVSampleBufferDisplayLayer()

    /// Renders the `pixelBuffer` in the `Sample` to the preview layer.
    /// - Parameters:
    ///     - using: The `sample` to render to the preview layer.
    func update(using sample: CameraFrame.Sample?) async throws {
        guard let sample else {
            await preview.sampleBufferRenderer.flush(removingDisplayedImage: true)
            return
        }

        let presentationTimeStamp = CMTime(seconds: sample.parameters.captureTimesta
        let timingInfo = CMSampleTimingInfo(duration: .invalid,
                                            presentationTimeStamp: presentationTimeS
                                            decodeTimeStamp: .invalid)

        try? sample.buffer.withUnsafeBuffer { pixelBuffer in
            let sampleBuffer = try CMSampleBuffer(imageBuffer: pixelBuffer,
                                                  formatDescription: CMVideoFormatDe
                                                  sampleTiming: timingInfo)
            if preview.sampleBufferRenderer.isReadyForMoreMediaData {
                preview.sampleBufferRenderer.enqueue(sampleBuffer)
            }
        }
    }
}
```

See Also

Enterprise APIs for visionOS

- Building spatial experiences for business apps with enterprise APIs for visionOS

  Grant enhanced sensor access and increased platform control to your visionOS app by using entitlements.

- Displaying video from connected devices

  Show video from devices connected with the Developer Strap in your visionOS app.

- Locating and decoding barcodes in 3D space

  Create engaging, hands-free experiences based on barcodes in a person’s surroundings.