Sample Code

Locating and decoding barcodes in 3D space

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

Download

visionOS 2.0+ | Xcode 16.0+

Overview

This sample code project demonstrates how to use barcode detection in ARKit to detect, decode, and place content near barcodes in your visionOS app. ARKit enables your visionOS app to detect barcodes (including QR codes), locate their positions in a person's surroundings, and decode their contents. For example, a warehouse worker can use your app to retrieve an item by looking at a package's barcode to confirm its contents.

Configure your sample code project

Replace `Enterprise.license` with your license file. The sample app requires a valid license file to detect barcodes.

Request the entitlement

Barcode detection is a part of enterprise APIs for visionOS, a collection of APIs that unlock capabilities for enterprise customers. To use barcode detection, you need to apply for the `Spatial barcode and QR code scanning` 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 `NSWorldSensingUsageDescription` to your app's information property list 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 world-sensing data.

> **Note**
>
> In visionOS, ARKit 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.

Detect, decode, and highlight barcodes

Your visionOS app can detect barcodes in a person's surroundings and highlight the barcode the person is looking for. The following code example detects and highlights every Code 128 or QR Code symbology in a person's surroundings. The code example includes three steps: detecting the barcode, printing its decoded content, and creating the highlight animation.

To start detecting barcodes, create a `BarcodeDetectionProvider` to get the positions of barcodes. Next, specify the `BarcodeAnchor.Symbology` to indicate the types of barcodes you want ARKit to detect in the person's surroundings. Then, start an `ARKitSession` with the `BarcodeDetectionProvider`.

```
import SwiftUI
import RealityKit
import ARKit
import Combine

struct ImmersiveView: View {
    @State private var arkitSession = ARKitSession()
    @State private var root = Entity()
    @State private var fadeCompleteSubscriptions: Set<AnyCancellable> = []

    var body: some View {
        RealityView { content in
            content.add(root)
        }
        .task {
```

```
            // Check whether there's support for barcode detection; otherwise, handl
            guard BarcodeDetectionProvider.isSupported else { return }

            // Specify the symbologies you want to detect.
            let barcodeDetection = BarcodeDetectionProvider(symbologies: [.code128,

            do {
                try await arkitSession.run([barcodeDetection])

                for await update in barcodeDetection.anchorUpdates where update.ever
                    let anchor = update.anchor

                    // Play an animation to indicate when the system detects a barc
                    playAnimation(for: anchor)

                    // Use the anchor's decoded contents and symbology to take actio
                    print(
                        """

                        Payload: \(anchor.payloadString ?? "")
                        Symbology: \(anchor.symbology)
                        """)
                }
            } catch {
                // Handle the error.
                print(error)
            }
        }
    }
}
```

ARKit delivers an asynchronous stream of updates as it detects changes in the scene. Each update includes a `BarcodeAnchor` containing the barcode's payload, extent, and transform. To implement the highlight animation, create a plane that you size and position to match that of the barcode, then fade it in and out.

> **Note**
>
> You define a barcode's extents in the x-z plane. They have a width (x-axis), depth (z-axis), and zero height (y-axis).

```
// Define this function in `ImmersiveView`.
func playAnimation(for anchor: BarcodeAnchor) {
    guard let scene = root.scene else { return }

    // Create a plane and size it to match the barcode.
    let extent = anchor.extent
    let entity = ModelEntity(mesh: .generatePlane(width: extent.x, depth: extent.z),
    entity.components.set(OpacityComponent(opacity: 0))

    // Position the plane over the barcode.
    entity.transform = Transform(matrix: anchor.originFromAnchorTransform)
    root.addChild(entity)

    // Fade the plane in and out.
    do {
        let duration = 0.5
        let fadeIn = try AnimationResource.generate(with: FromToByAnimation<Float>(
            from: 0,
            to: 1.0,
            duration: duration,
            isAdditive: true,
            bindTarget: .opacity)
        )
        let fadeOut = try AnimationResource.generate(with: FromToByAnimation<Float>(
            from: 1.0,
            to: 0,
            duration: duration,
            isAdditive: true,
            bindTarget: .opacity))

        let fadeAnimation = try AnimationResource.sequence(with: [fadeIn, fadeOut])

        _ = scene.subscribe(to: AnimationEvents.PlaybackCompleted.self, on: entity,
            // Remove the plane after the animation completes.
            entity.removeFromParent()
        }).store(in: &fadeCompleteSubscriptions)

        entity.playAnimation(fadeAnimation)
    } catch {
        // Handle the error.
    }
}
```

> **Note**
>
> Because `BarcodeDetectionProvider` has a low refresh rate, use its transform to initialize the position of content relative to a stationary barcode.

Determine the ideal barcode width

The sample code project can't read barcodes that are too small to appear clearly in a person's field of view. Larger barcodes generally improve readability, providing that they remain within the field of view. The minimum barcode size depends on its `BarcodeAnchor.Symbology`. Refer to the table below to determine the minimum width required for a barcode to be readable under nominal lighting conditions (100 lux) at an arm's length distance (~40 cm).

Symbology	Minimum width (in cm)
Aztec Code	2.0
Codabar	5.5
Code 39	6.5
Code 39 Checksum	6.5
Code 39 Full ASCII	3.0
Code 39 Full ASCII Checksum	4.5
Code 93	5.0
Code 93i	5.0
Code 128	2.5
Data Matrix	1.0
EAN-8	3.0
EAN-13	4.0
GS1 DataBar	3.0
GS1 DataBar Expanded	6.5

Symbology	Minimum width (in cm)
GS1 DataBar Limited	3.0
ITF	3.5
ITF-14	5.0
ITF Checksum	3.5
MicroPDF417	6.5
Micro QR Code	2.0
MSI Plessey	4.5
PDF417	6.0
QR Code	1.5
UPC-E	2.5

See Also

Enterprise APIs for visionOS

{} Accessing the main camera

Add camera-based features to enterprise apps.

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.