Developing for visionOS

Augmenos

Intro

Miguel Garcia Gonzalez

Augmenos

Supply chain professional by day, spatial computing engineer by night.

- Native visionOS developer specializing in SwiftUI, ARKit, RealityKit, and Reality Composer Pro.
- Featured apps on the App Store for visionOS and Apple Newsroom.
- Currently at Spotify, graduate of MIT School of Engineering.
- Portfolio: augmenos.com↗
- GitHub: github.com/augmenos↗

Museas↗

Astronoma↗

Astronoma App Preview

Agenda

1. Overview of visionOS and Apple Vision Pro.
2. Setting up development environment, tips, and tricks.
3. Live code demo: building a native visionOS app.
4. Deeper dive of hand tracking APIs with sample project.
5. Final comments on the future of visionOS.

Overview

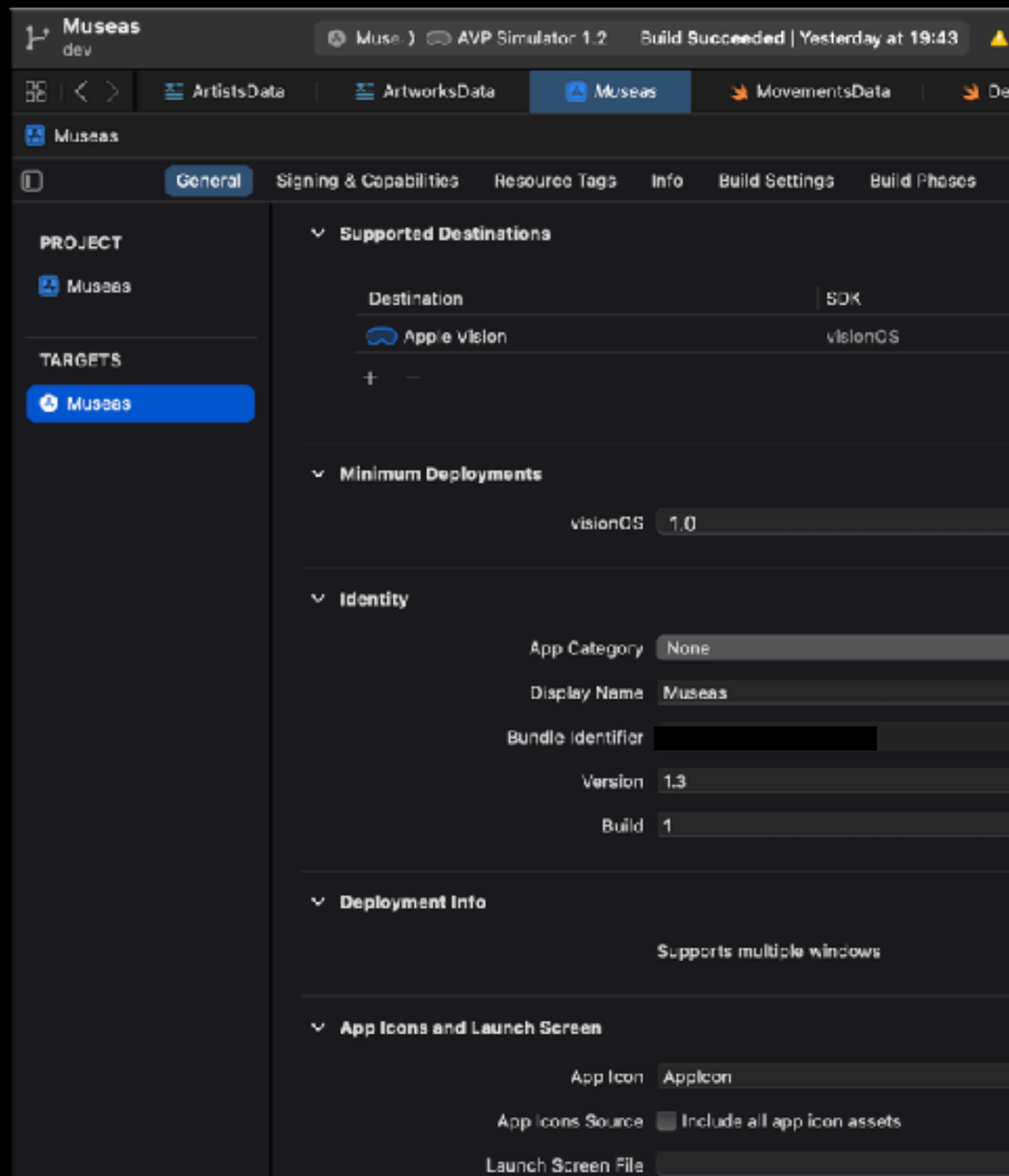
visionOS and Apple Vision Pro

- visionOS 1.0 launched 12 months ago, AVP launched 6 months ago. We are in the very early stages.
- Uses native SKDs and APIs already in use across iOS: SwiftUI, RealityKit, ARKit.
- Development on Xcode, Apple silicon (M-series chip) is required. Unity also supported.
- Incredible hardware and software. Intuitive, impressive, but sometimes impractical.

Environment

Setup

- Enroll into Apple Developer Program ($99/Year).
- Latest Xcode on Apple silicon (M-series chip). MacBook Pro (minimum 18GB memory, 1TB storage).
- Install visionOS simulator (Version 1.2). Version 2.0 also available in beta.
- Install other critical programs: Reality Converter and Reality Composer Pro.
- The simulator works, but it's not ideal and can be buggy. AVP strongly recommended.



World
main

World 〉 AVP Simulator 1.2 World: Ready | Today at 15:06

WorldApp

World 〉 World 〉 WorldApp 〉 No Selection

- World
  - README
  - World
    - WorldApp
    - Modules
      - Modules
      - TableOfContents
      - ModuleCard
      - ModuleDetail
    - Globe
    - Orbit
    - Solar System
    - RealityViews
    - Entities
    - Systems
    - Modifiers
    - Model
    - Settings
    - Resources
    - Info
    - Preview Content
    - Localizable
    - InfoPlist
  - Packages
    - WorldAssets
      - README
      - Package
      - Sources
        - WorldAssets
          - WorldAssets
          - SunPositionSystem
          - WorldAssets
      - Package
  - Products
  - Configuration
    - SampleCode

```swift
import SwiftUI
import WorldAssets

/// The main entry point of the Hello World experience.
@main
struct WorldApp: App {
    // The view model.
    @State private var model = ViewModel()

    // The immersion styles for different modules.
    @State private var orbitImmersionStyle: ImmersionStyle = .mixed
    @State private var solarImmersionStyle: ImmersionStyle = .full

    var body: some Scene {
        // The main window that presents the app's modules.
        WindowGroup(String(localized: "Hello World",
                           comment: "The name of the app. This is the typical title for many example apps in
                               programming tutorials."),
                    id: "modules") {
            Modules()
                .environment(model)
        }
        .windowStyle(.plain)

        // A volume that displays a globe.
        WindowGroup(id: Module.globe.name) {
            Globe()
                .environment(model)
        }
        .windowStyle(.volumetric)
        .defaultSize(width: 0.6, height: 0.6, depth: 0.6, in: .meters)

        // An immersive space that places the Earth with some of its satellites
        // in your surroundings.
        ImmersiveSpace(id: Module.orbit.name) {
            Orbit()
                .environment(model)
        }
        .immersionStyle(selection: $orbitImmersionStyle, in: .mixed)

        // An immersive Space that shows the Earth, Moon, and Sun as seen from
        // Earth orbit.
        ImmersiveSpace(id: Module.solar.name) {
            SolarSystem()
                .environment(model)
        }
```

Line: 1 Col: 1

WorldAssets

Earth

- Root
  - Earth
    - Sphere_001
    - Earth
      - vector_from_float
      - pan_uvs

Filter

Earth

Layer Data

Default Prim	Root
Meters Per Unit	1
Up Axis	Y

Project Browser | Shader Graph | Audio Mixer | Statistics

Collections

- All Files

Project

- WorldAssets.rkassets
  - Moon
    - 0
  - Pole
    - 0
  - Satellite
    - 0
  - Sun
    - 0
  - Telescope
    - 0

Filter

Earth.usda | Globe.usda | Globe.usdz | M_earth_baseColor.png | M_earth_clouds_normal.png

M_earth_emissive.jpg | M_earth_normal.png | M_earth_roughness.jpg | Moon | Moon.usda

Select a file to view its properties.

A Day in the Life
Planet Earth
You can't feel it, but Earth is constantly in motion. All planets spin on an invisible axis. Ours makes one full turn every 24 hours, bringing days and nights to our home.
When your part of the world faces the Sun, it's daytime; when it rotates away, we move into night. When you see a sunrise or sunset, you're witnessing the Earth in motion.
Want to explore Earth's rotation and axial tilt? Check out our interactive 3D globe and be hands-on with Earth's movements.
View Globe

Development

Frameworks

SwiftUI

- Best framework for visionOS apps.
- Supports 3D capabilities: depth, gestures, effects.
- Integrates with RealityKit for volumetric interfaces.
- Seamlessly works with UIKit.

ARKit

- Apple’s 3D rendering engine.
- Adapts to physical lighting, casts shadows.
- Supports MaterialX for advanced material authoring.

RealityKit

- Understands surroundings for interactive experiences.
- Powers core features in Shared Space.
- Offers APIs: Plane Estimation, Scene Reconstruction, World Tracking.

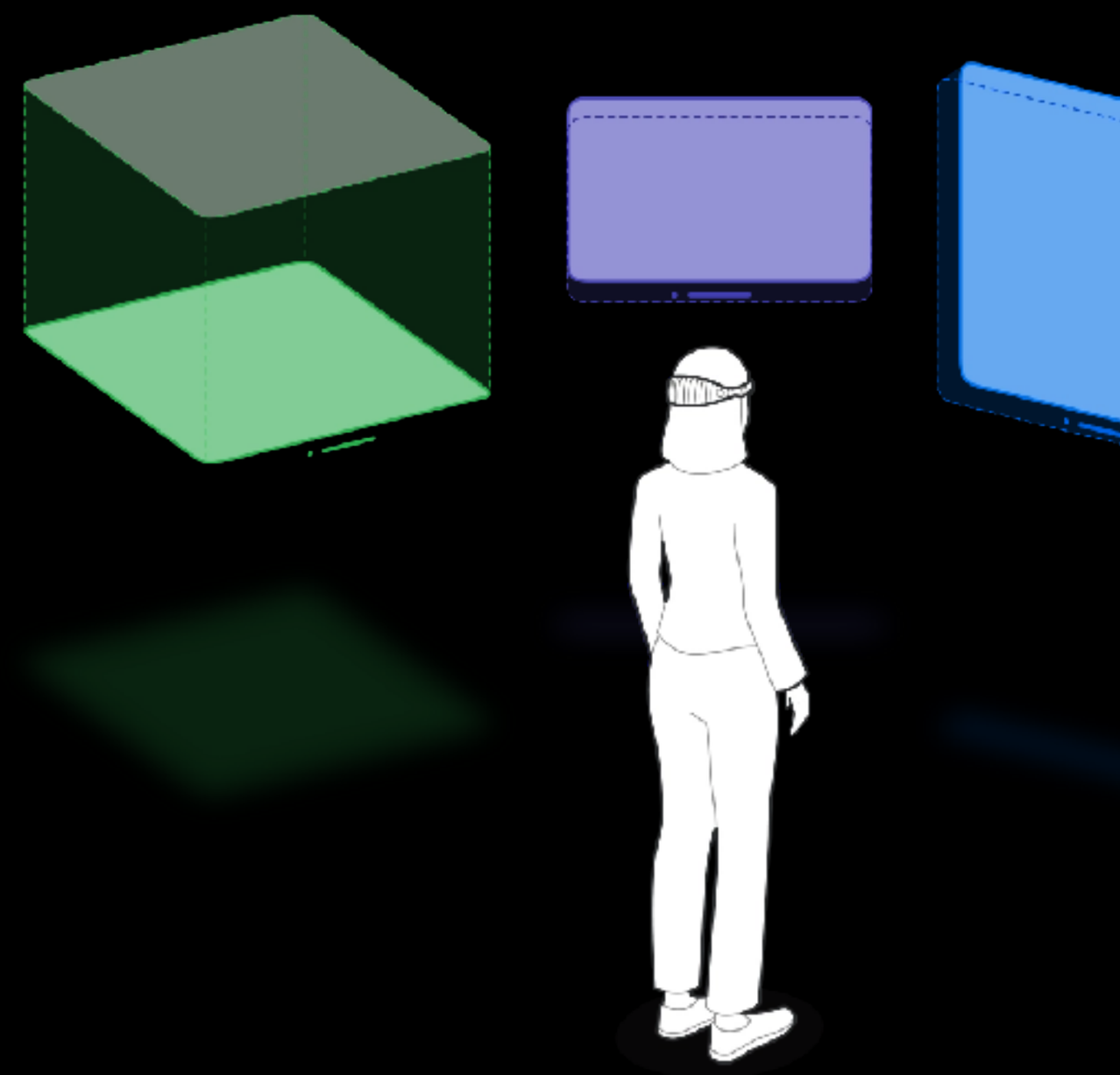
Development

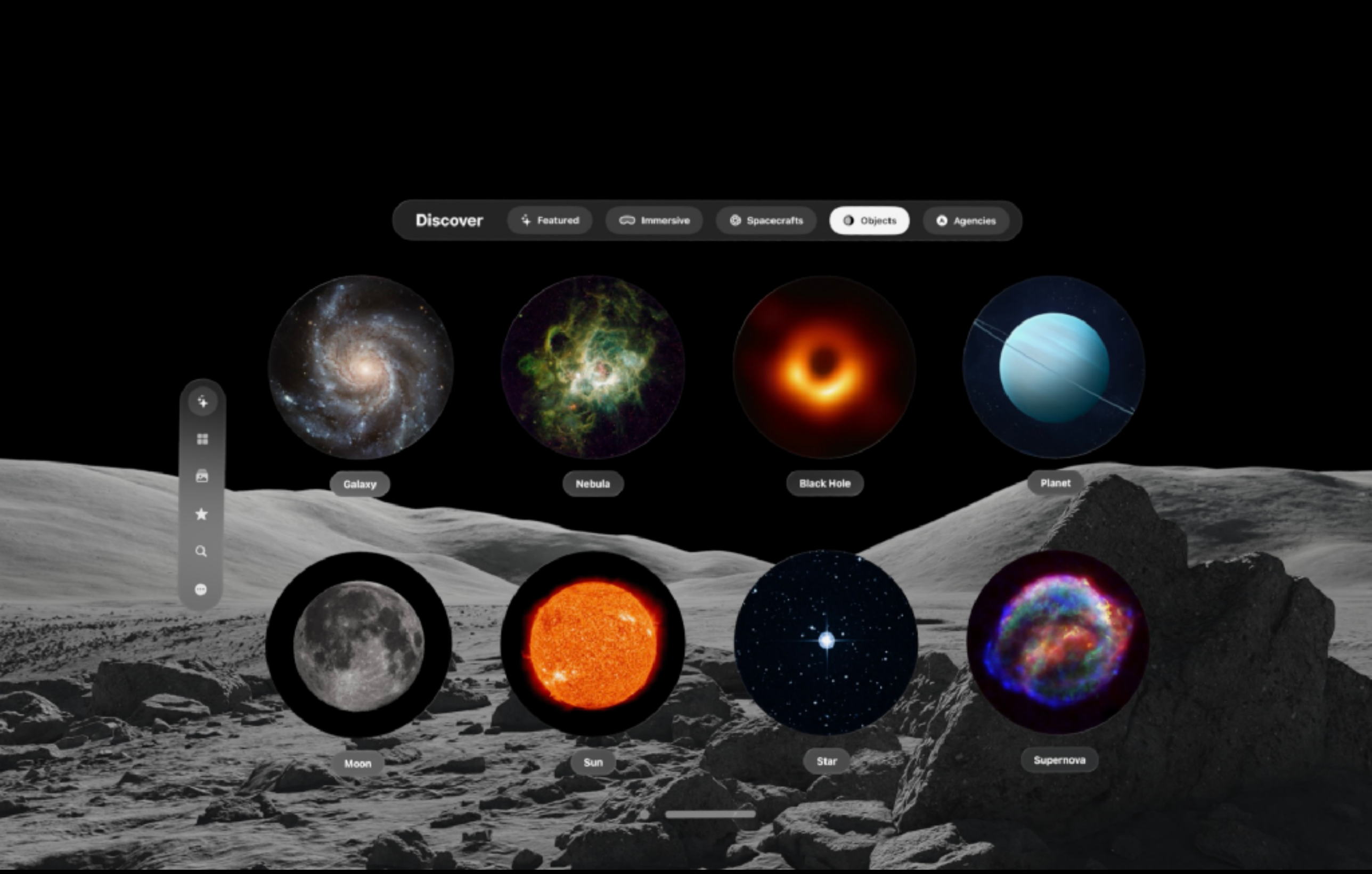
Frameworks

Windows
Traditional 2D windows but with added depth effects and 3D overlays.

Volumes
Full 3D view, contained in a cube, for displaying models. Viewable from any angle.

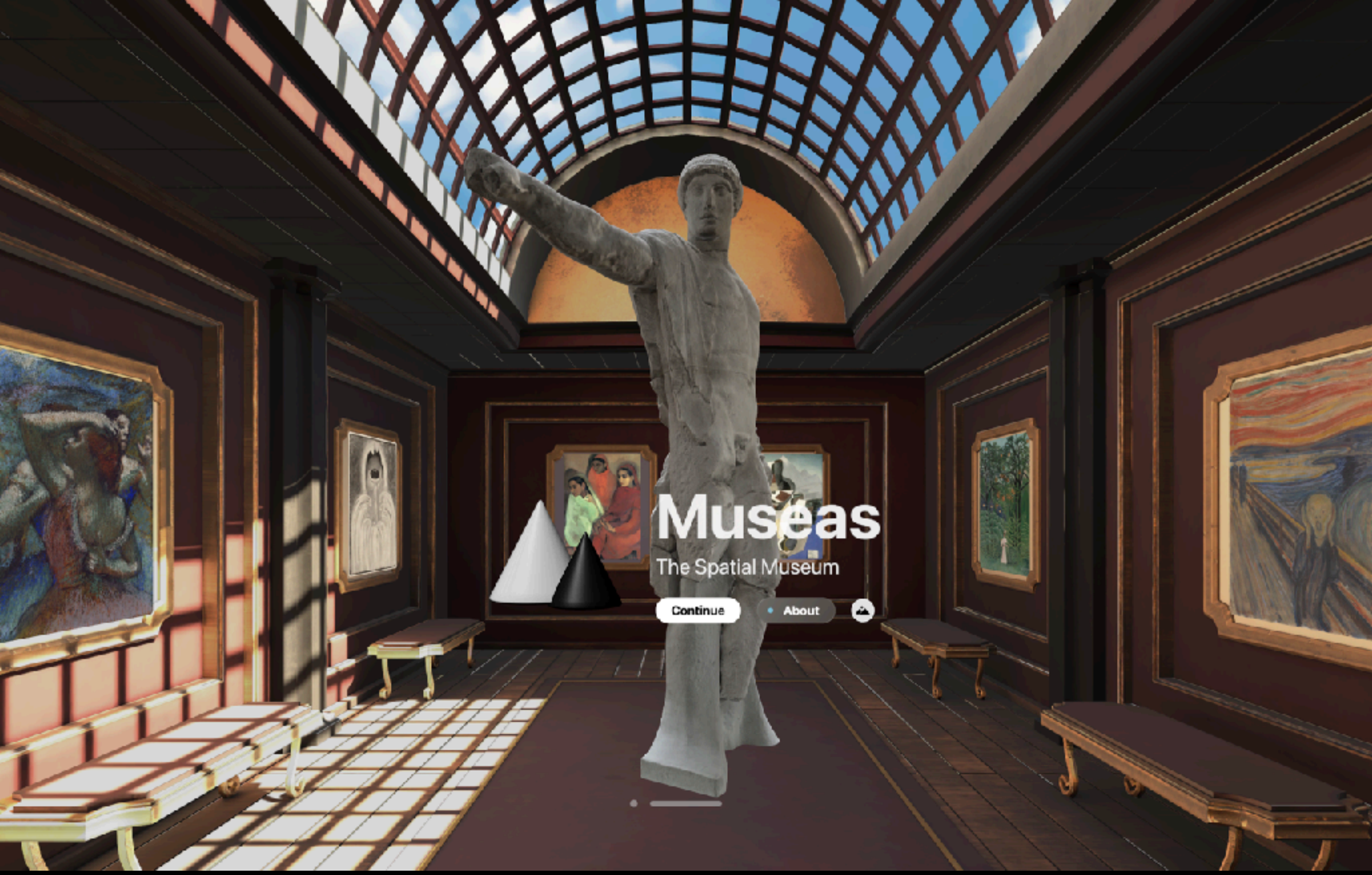
Spaces
In essence, a virtual reality view. Full space where only that app's content will appear.





Window Astronoma App

Volume Mexico Vision App

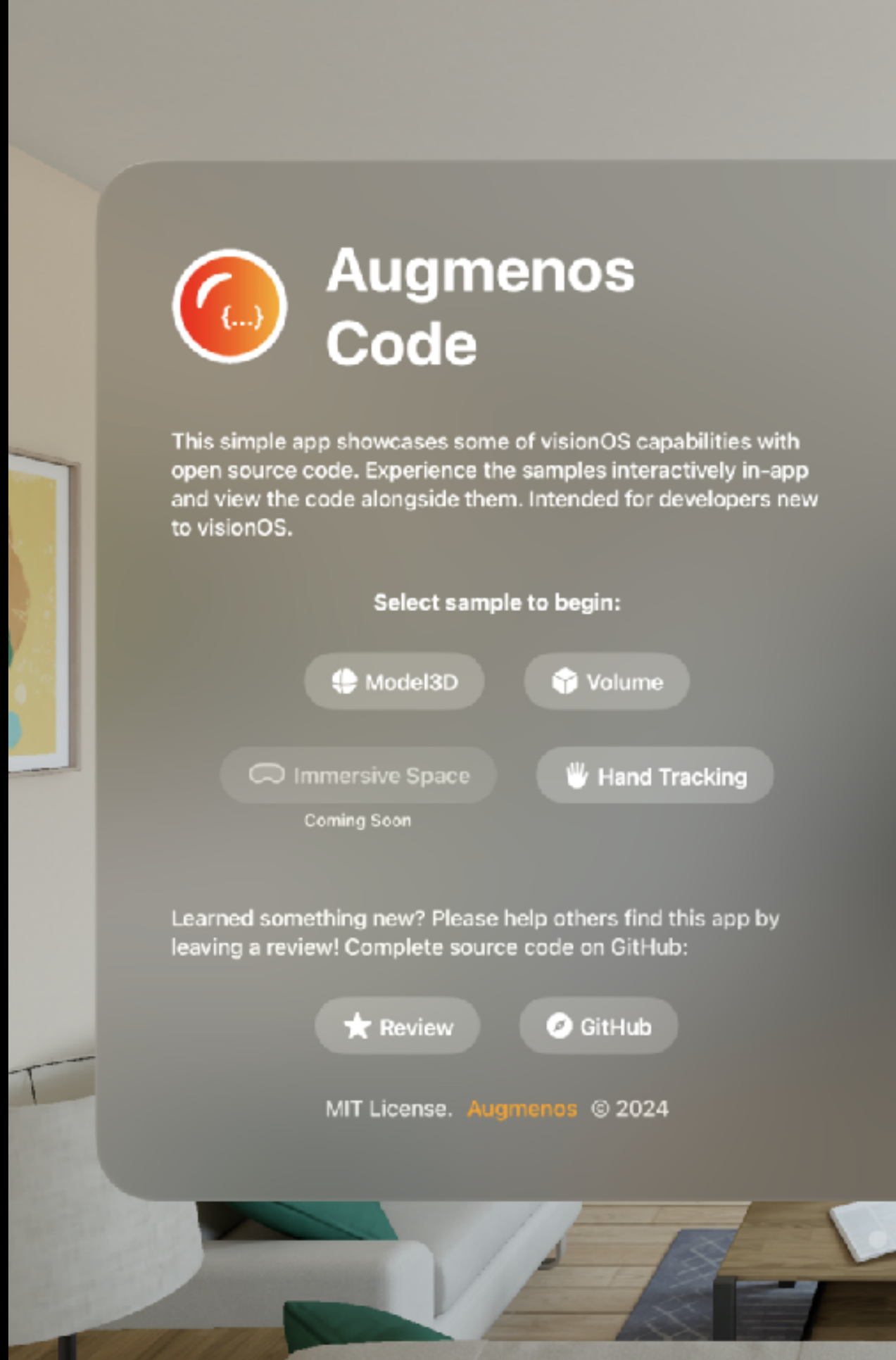


Space Museas App

Development

Let’s Code 👨‍💻

- Project: “Augmenos Code” (live on the App Store↗, open source on GitHub↗).
- Uses windows, volumes, and immersive spaces to learn about visionOS APIs.
- Focus of this presentation: loading and presenting 3D model into a window from Reality Composer Pro scene using SwiftUI and Model3D.



Model3D

A view that asynchronously loads and displays a 3D model.

Model3D is a SwiftUI view to quickly render a 3D model. ModelEntity is a more robust representation of a physical object that RealityKit renders and optionally simulates for advanced functionality.

Here we load and display scene "Heart" from RealityKitContent:

Learn more about Model3D:

Apple Documentation

WWDC Video

```
import SwiftUI
import RealityKit
import RealityKitContent

struct Model3DView: View {
    @Environment(.openURL) var openURL

    var body: some View {

        HStack(spacing: 50) {
            // Left Column
            VStack(alignment: .leading) {
                Text("Model3D")
                    .font(.extraLargeTitle)
                    .padding(.bottom, 30)

                Text("A view that asynchronously
loads and displays a 3D model.")
                    .padding(.bottom, 20)

                HStack(alignment: .top, spacing: 30)
{
                    Image(systemName:
"exclamationmark.bubble.fill")
                        .font(.largeTitle)
```

Model3D

A view that asynchronously loads and displays a 3D model.

Model3D is a SwiftUI view to quickly render a 3D model. ModelEntity is a more robost representation of a physical object that RealityKit renders and optionally simulates for advanced functionality.

Here we load and display scene "Heart" from RealityKitContent:

Learn more about Model3D:

Apple Documentation

WWDC Video

```
import SwiftUI
import RealityKit
import RealityKitConten

struct Model3DView: Vie
    @Environment(.openU

    var body: some View

        HStack(spacing:
            // Left Col
            VStack(alig
                Text("M
                    .fo
                    .pa

                Text("A
loads and displays a 3D
                    .pa

                HStack(
{
                    Ima
"exclamationmark.bubble
```

augmenos / AugmenosCode

Type / to search

Code | Issues | Pull requests | Discussions | Actions | Projects | Security | Insights | Settings

AugmenosCode

Public

Edit Pins | Watch 1 | Fork 0 | Starred 1

main | 2 Branches | 1 Tags | Go to file | t | + | Code

miggiegg Update README.md 35cac3c · 1 hour ago 14 Commits

AugmenosCode.xcodeproj	Upload project	last week
AugmenosCode	Upload project	last week
Packages/RealityKitContent	Upload project	last week
images	Add files via upload	1 hour ago
LICENSE	Initial commit	last week
README.md	Update README.md	1 hour ago

README | MIT license

Augmenos Code

About

An educational app that showcases some of visionOS capabilities with open source code. Intended for developers learning visionOS. Download on the App Store or Xcode project.

apps.apple.com/app/augmenos-code/i...

arkit | swiftui | realitykit | visionos | vision-pro | reality-composer-pro

Readme
MIT license
Activity
Custom properties
1 star
1 watching
0 forks
Report repository

Releases 1

Augmenos Code v1.0.0 Latest
last week

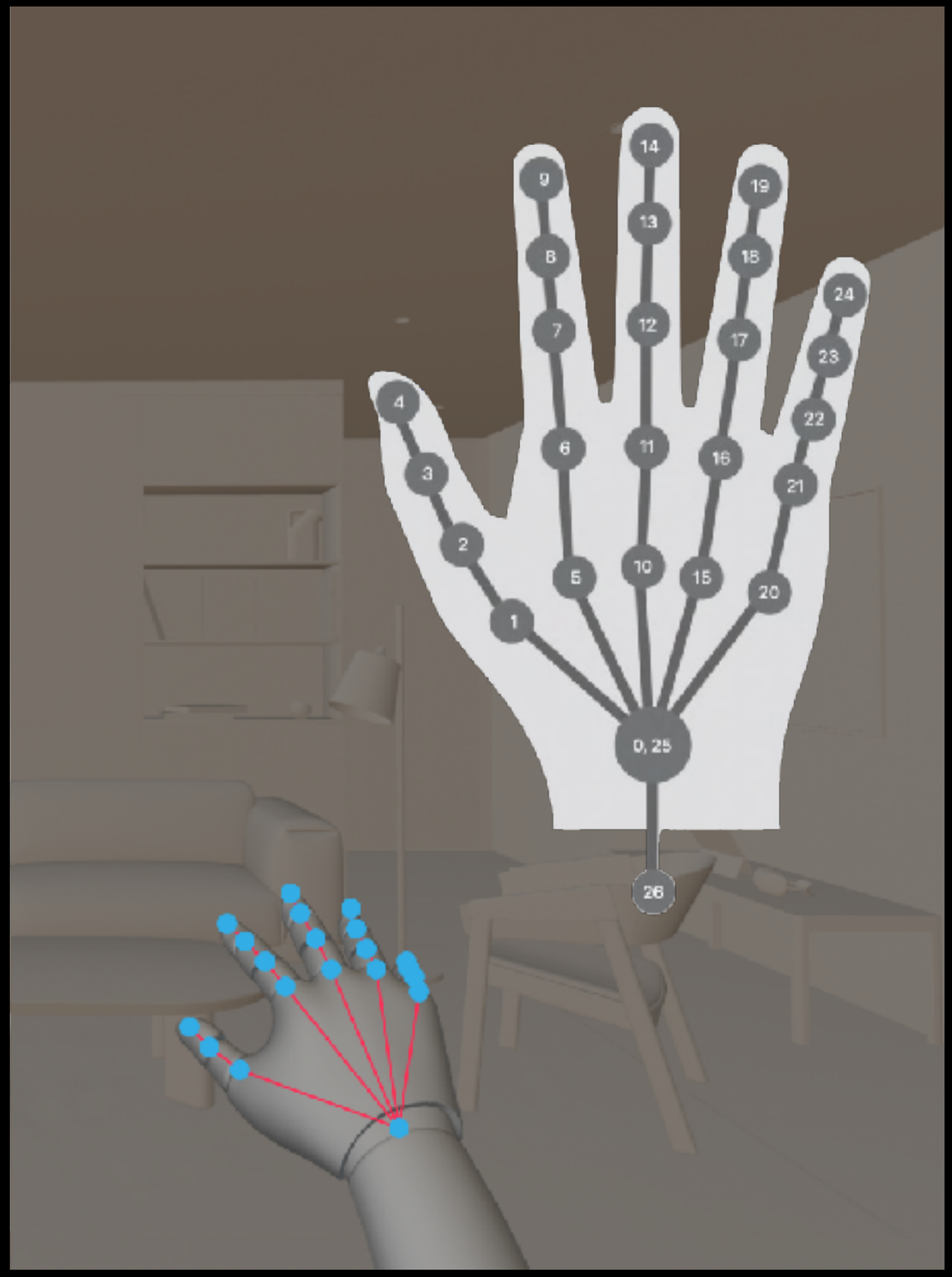
Packages

Demo

Tracking

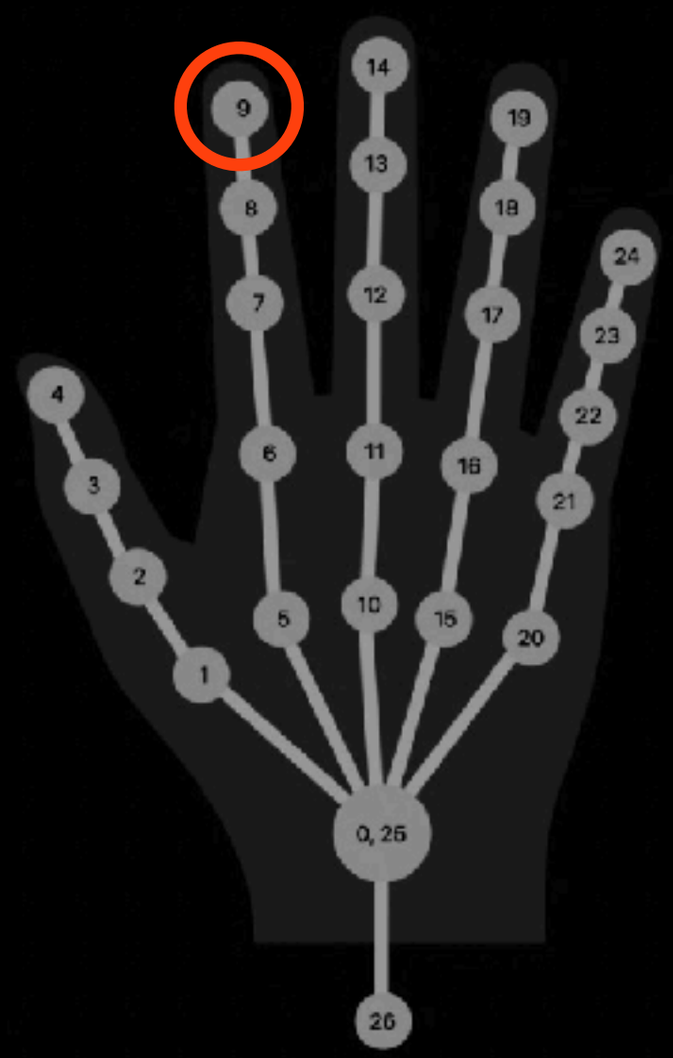
Objects and Hands

- Hand gestures already built-into visionOS, but not tracking.
- HandAnchor allows us to track a user's hand position, rotation and size (transform) in the scene as well as each joint within the hand.
- **Important**: Requires user permission via ARKitSession and Full Space (also physical AVP, cannot test in simulator).
- More advanced object tracking API introduced at WWDC24 for visionOS 2.0. Currently in beta.



Mapping

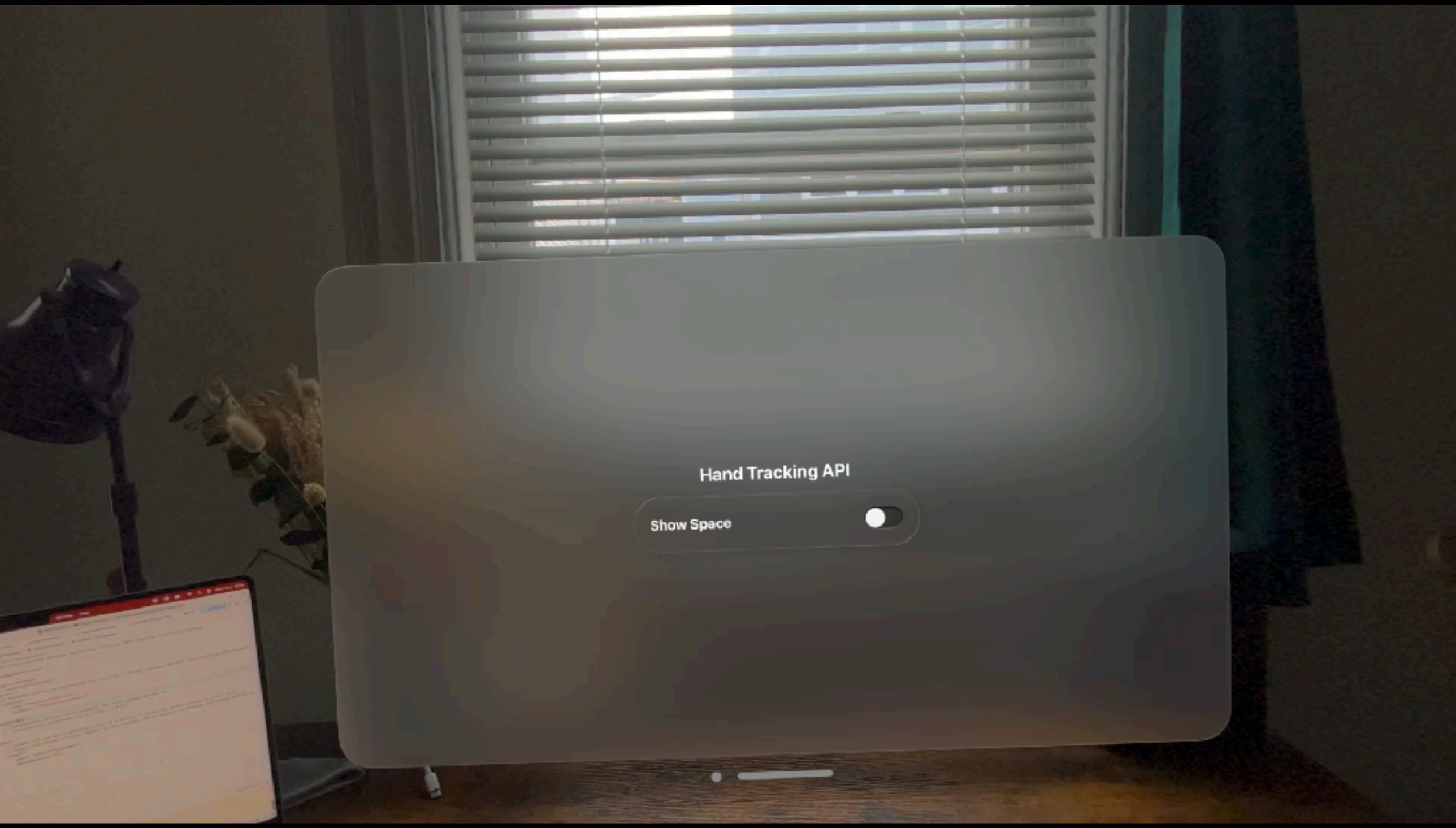
Hands



0	.handWrist
1	.handThumbKnuckle
2	.handThumbIntermediateBase
3	.handThumbIntermediateTip
4	.handThumbTip
5	.handIndexFingerMetacarpal
6	.handIndexFingerKnuckle
7	.handIndexFingerIntermediateBase
8	.handIndexFingerIntermediateTip
9	.handIndexFingerTip
10	.handMiddleFingerMetacarpal
11	.handMiddleFingerKnuckle
12	.handMiddleFingerIntermediateBase
13	.handMiddleFingerIntermediateTip

14	.handMiddleFingerTip
15	.handRingFingerMetacarpal
16	.handRingFingerKnuckle
17	.handRingFingerIntermediateBase
18	.handRingFingerIntermediateTip
19	.handRingFingerTip
20	.handLittleFingerMetacarpal
21	.handLittleFingerKnuckle
22	.handLittleFingerIntermediateBase
23	.handLittleFingerIntermediateTip
24	.handLittleFingerTip
25	.handForearmWrist
26	.handForearmArm

Source: Apple ↗



HandTrackingApp Video Preview

HandTrackingProvider

A source of live data about the position of a person's hands and hand joints.

HandTrackingProvider requires both a Full Space and user permission with NSHandsTrackingUsageDescription in Info.plist.

Warning: HandTrackingProvider will not work on the simulator. The app will crash!

Here we have a toggle that will open the mixed full space to use HandTrackingProvider using .onChange(of: showImmersiveSpace):

Mixed Immersive Space

Learn more about HandTrackingProvider:

Apple Documentation

WWDC Video

```
@MainActor class ImmersiveViewModel: ObservableObject {

    private let session = ARKitSession()
    private let handTracking = HandTrackingProvider()
    private let sceneReconstruction =
SceneReconstructionProvider()

    private var contentEntity = Entity()

    private var meshEntities = UUID: ModelEntity

    private let fingerEntities: [HandAnchor.Chirality:
ModelEntity] = [
        .left: .createFingertip(),
        .right: .createFingertip()
    ]

    func setupContentEntity() -> Entity {
        for entity in fingerEntities.values {
            contentEntity.addChild(entity)
        }

        return contentEntity
    }

    func runSession() async {
```

```swift
import SwiftUI
import RealityKit
import RealityKitContent

struct ImmersiveView: View {
    // Create and manage the view model as a state object
    @StateObject var model = ImmersiveViewModel()

    var body: some View {
        // RealityView is used to integrate RealityKit content into SwiftUI
        RealityView { content in
            // Add the main content entity to the RealityView
            content.add(model.setupContentEntity())
        }
        // Start the AR session when the view appears
        .task {
            await model.runSession()
        }
        // Process hand tracking updates continuously
        .task {
            await model.processHandUpdates()
        }
        // Process scene reconstruction updates continuously
        .task {
            await model.processReconstructionUpdates()
        }
        // Add a spatial tap gesture recognizer
        .gesture(SpatialTapGesture().targetedToAnyEntity().onEnded({ value in
            // Convert the tap location from global to scene coordinates
            let location3D = value.convert(value.location3D, from: .global, to: .scene)
            // Add a cube at the tapped location
            model.addCube(tapLocation: location3D)
        }))
    }
}
```

```swift
import Foundation
import RealityKit
import ARKit

@MainActor class ImmersiveViewModel: ObservableObject {

    // Set up AR session and providers
    private let session = ARKitSession()
    private let handTracking = HandTrackingProvider()
    private let sceneReconstruction = SceneReconstructionProvider()

    // Main entity to hold all content
    private var contentEntity = Entity()

    // Dictionary to store mesh entities
    private var meshEntities = [UUID: ModelEntity]()

    // Create fingertip entities for both hands
    private let fingerEntities: [HandAnchor.Chirality: ModelEntity] = [
        .left: .createFingertip(),
        .right: .createFingertip()
    ]


    // Set up the main content entity
    func setupContentEntity() -> Entity {
        // Add fingertip entities to the main content entity
        for entity in fingerEntities.values {
            contentEntity.addChild(entity)
        }

        return contentEntity
    }

    // Start the AR session
    func runSession() async {
        do {
            try await session.run([sceneReconstruction, handTracking])
        } catch {
            print("Failed to start session: \(error)")
        }
    }
```

```
// Process hand tracking updates
func processHandUpdates() async {
    for await update in handTracking.anchorUpdates {
        let handAnchor = update.anchor
        guard handAnchor.isTracked else { continue }

        // Get the index fingertip joint
        let fingertip = handAnchor.handSkeleton?.joint(.indexFingerTip)
        guard ((fingertip?.isTracked) != nil) else { continue }

        // Calculate the transform for the fingertip
        let originFromWrist = handAnchor.originFromAnchorTransform
        guard let wristFromIndex = fingertip?.anchorFromJointTransform else { continue }
        let originFromIndex = originFromWrist * wristFromIndex

        // Update the position of the fingertip entity
        fingerEntities[handAnchor.chirality]?.setTransformMatrix(originFromIndex, relativeTo: nil)
    }
}
```

```
class ImmersiveViewModel
    // Process scene reconstruction updates
    func processReconstructionUpdates() async {
        for await update in sceneReconstruction.anchorUpdates {
            let meshAnchor = update.anchor

            // Generate a mesh from the anchor
            guard let shape = try? await ShapeResource.generateStaticMesh(from: meshAnchor) else { continue }

            switch update.event {
            case .added:
                // Create and set up a new entity for the mesh
                let entity = ModelEntity()
                entity.transform = Transform(matrix: meshAnchor.originFromAnchorTransform)
                entity.collision = CollisionComponent(shapes: [shape], isStatic: true)
                entity.physicsBody = PhysicsBodyComponent()
                entity.components.set(InputTargetComponent())

                // Store and add the entity
                meshEntities[meshAnchor.id] = entity
                contentEntity.addChild(entity)
            case .updated:
                // Update an existing mesh entity
                guard let entity = meshEntities[meshAnchor.id] else { fatalError("Fatal Error") }
                entity.transform = Transform(matrix: meshAnchor.originFromAnchorTransform)
                entity.collision?.shapes = [shape]
            case .removed:
                // Remove a mesh entity
                meshEntities[meshAnchor.id]?.removeFromParent()
                meshEntities.removeValue(forKey: meshAnchor.id)
            }
        }
    }
```

```swift
class ImmersiveViewModel
    // Add a cube at the specified location
    func addCube(tapLocation: SIMD3<Float>) {
        let placementLocation = tapLocation + SIMD3<Float>(0, 0.2, 0)

        // Create a cube entity
        let entity = ModelEntity(
            mesh: .generateBox(size: 0.1, cornerRadius: 0.0),
            materials: [SimpleMaterial(color: .cyan, isMetallic: false)],
            collisionShape: .generateBox(size: SIMD3<Float>(repeating: 0.1)),
            mass: 1.0)

        // Set the cube's position and properties
        entity.setPosition(placementLocation, relativeTo: nil)
        entity.components.set(InputTargetComponent(allowedInputTypes: .all))

        // Add physics properties to the cube
        let material = PhysicsMaterialResource.generate(friction: 0.8, restitution: 0.0)
        entity.components.set(PhysicsBodyComponent(shapes: entity.collision!.shapes,
                                                   mass: 1.0,
                                                   material: material,
                                                   mode: .dynamic))

        // Add the cube to the scene
        contentEntity.addChild(entity)
    }
}
```

HandTrackingApp Screenshot

Conclusion

and Final Thoughts

- Critical to learn SwiftUI, RealityKit, and ARKit to get started.
- Apple Vision Pro not required at first. Start with iOS and iPhone/iPad.
- Simulator is not recommended. Actual hardware strongly advised.
- visionOS dev experience has been excellent. Great documentation and projects, helpful community.
- It is the future, no doubt in my mind.

Augmenos

Copyright © 2024 Augmenos

Apple Vision Pro, visionOS, ARKit, and RealityKit are registered trademarks of Apple Inc.