Sample Code

Creating a collaborative session

Enable nearby devices to share an AR experience by using a peer-to-peer multiuser strategy.

Download

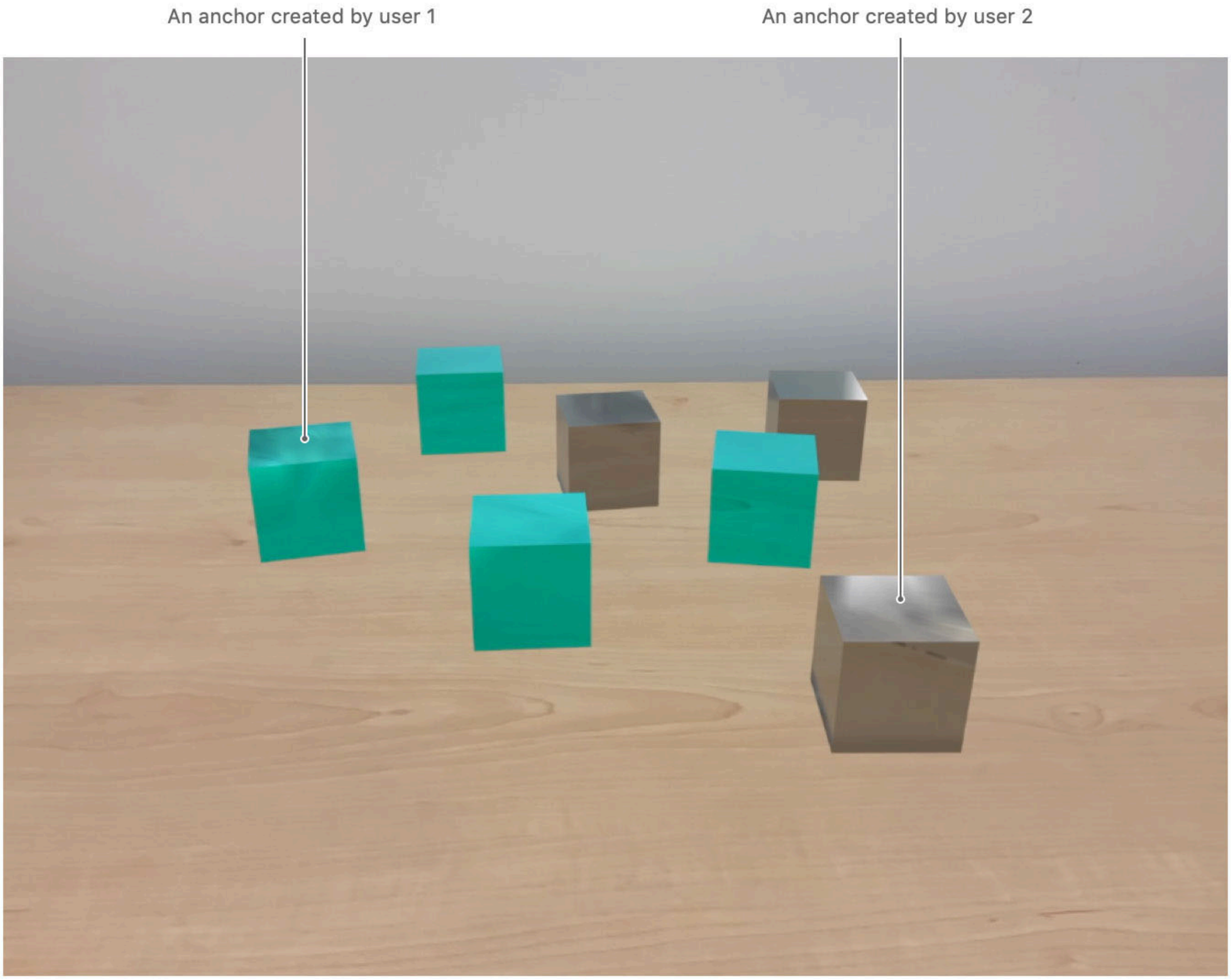
iOS 13.0+ | iPadOS 13.0+ | Xcode 16.0+

Overview

As an AR app runs, ARKit gathers information about a user's physical environment by processing the camera feed from the user's device. To effect a multiuser AR experience in which users learn more about the environment by sharing the information from their device's camera feed with other users, you enable *collaboration*.

Throughout a collaborative session, ARKit periodically provides data for you to share with peer users, and you choose a network protocol to send that data. In addition to information about the layout of the physical environment––the *world data*––collaboration data includes an anchor for each participant. These anchors indicate each peer's approximate location, which you can use, for example, to place virtual content that represents the peer user. ARKit also provides you with any anchors that the peer users create themselves.

This sample app allows multiple users to view a common horizontal surface and place blocks on top of the surface, with a unique color for each user.



Although this sample draws its graphics using RealityKit, it doesn't use RealityKit's mechanism for over-the-network entity synchronization. Instead, it uses RealityKit as a renderer only as necessary to demonstrate ARKit's collaborative session.

To create a shared AR experience using a host-guest approach, see Creating a multiuser AR experience.

Enable Collaboration

Collaborative sessions are available when your session uses `ARWorldTrackingConfiguration`. To enable collaboration, set `isCollaborationEnabled` to `true`.

```
configuration = ARWorldTrackingConfiguration()

// Enable a collaborative session.
configuration?.isCollaborationEnabled = true
```

```
// Enable realistic reflections.
configuration?.environmentTexturing = .automatic

// Begin the session.
arView.session.run(configuration!)
```

Gather collaboration data

When collaboration is enabled, ARKit periodically invokes `session(_:didOutputCollaborationData:)`, which provides collaboration data that you can share with nearby users. You are responsible for sending collaboration data over the network, including choosing the network framework and implementing the code. The data you send is a serialized version of the `ARSession.CollaborationData` object provided by your session. Before you send collaboration data over the network, first serialize it using [`NSKeyedArchiver`.

```
func session(_ session: ARSession, didOutputCollaborationData data: ARSession.Collab
    guard let multipeerSession = multipeerSession else { return }
    if !multipeerSession.connectedPeers.isEmpty {
        guard let encodedData = try? NSKeyedArchiver.archivedData(withRootObject: da
        else { fatalError("Unexpectedly failed to encode collaboration data.") }
        // Use reliable mode if the data is critical, and unreliable mode if the dat
        let dataIsCritical = data.priority == .critical
        multipeerSession.sendToAllPeers(encodedData, reliably: dataIsCritical)
    } else {
        print("Deferred sending collaboration to later because there are no peers.")
    }
}
```

It's safe to ignore the collaboration data if no peers have joined the session. In that case, ARKit outputs the collaboration data later to try again. The alternative approach of enabling collaboration only after peers have joined is not supported, because doing so restarts the session.

Send collaboration data to others

You choose the network protocol with which to share collaboration data. This sample app sends collaboration data using Multipeer Connectivity.

```
func sendToPeers(_ data: Data, reliably: Bool, peers: [MCPeerID]) {
    guard !peers.isEmpty else { return }
    do {
        try session.send(data, toPeers: peers, with: reliably ? .reliable : .unrelia
    } catch {
        print("error sending data to peers \(peers): \(error.localizedDescription)")
    }
}
```

Update your session with collaboration data

When you receive collaboration data from peer users, you instantiate an `ARSession.CollaborationData` object with it, and pass the object to your session via `update(with:)`.

```
func receivedData(_ data: Data, from peer: MCPeerID) {
    if let collaborationData = try? NSKeyedUnarchiver.unarchivedObject(ofClass: ARSe
        arView.session.update(with: collaborationData)
        return
    }
    // ...
```

Facilitate world map merging

For ARKit to know where two users are with respect to each other, it has to recognize overlap across their respective world maps. When ARKit succeeds in fitting the two world maps together, it can begin sharing those users' respective locations and any anchors they created with each other.

To aid ARKit with world map merging, a user must point their device near an area that another user has viewed. The sample app accomplishes this by asking the users to hold their devices side by side.

```
messageLabel.displayMessage("""
    A peer wants to join the experience.
    Hold the phones next to each other.
    """, duration: 6.0)
```

Identify when ARKit merges world data

The first time ARKit successfully merges world data from another user, it calls your app's `session(_:didAdd:)`, passing in an `ARSessionDelegate` that identifies the other user. This action notifies you of the merging event.

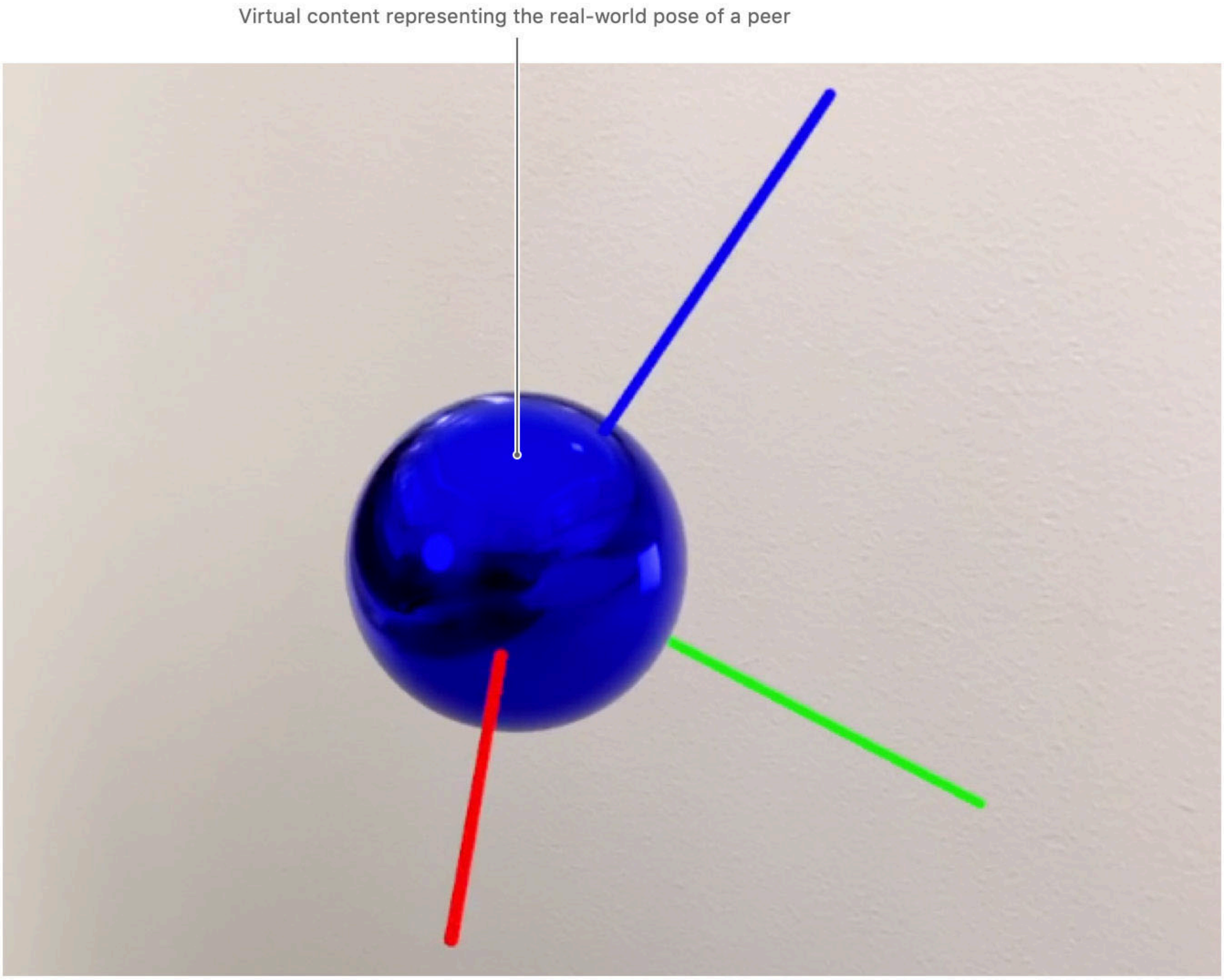
```
func session(_ session: ARSession, didAdd anchors: [ARAnchor]) {
    for anchor in anchors {
        if let participantAnchor = anchor as? ARParticipantAnchor {
            messageLabel.displayMessage("Established joint experience with a peer.")
            // ...
```

Visualize users by displaying virtual content

When ARKit successfully merges two users' world data, you can then initiate actions to begin the multiuser experience. The sample adds virtual content in the real-world location of newly joined peer users to visualize them in AR.

```
let anchorEntity = AnchorEntity(anchor: participantAnchor)

let coordinateSystem = MeshResource.generateCoordinateSystemAxes()
anchorEntity.addChild(coordinateSystem)

let color = participantAnchor.sessionIdentifier?.toRandomColor() ?? .white
let coloredSphere = ModelEntity(mesh: MeshResource.generateSphere(radius: 0.03),
                                materials: [SimpleMaterial(color: color, isMetallic
anchorEntity.addChild(coloredSphere)

arView.scene.addAnchor(anchorEntity)
```

The multicolored coordinate system shown in the following illustration represents the real-world pose of a peer user. ARKit periodically refreshes participant anchors to reflect any updates in the real-world location and orientation of the user it tracks. This process is a part of the collaboration data your app shares and uses to update its session.



Check an anchor’s owner

When ARKit merges two users’ world data, it collects all the anchors created by both users and calls `session(_:didAdd:)` to notify each user of the collection. To check which user created an anchor, you compare the anchor’s `sessionIdentifier` with the active session’s `identifier`. If the anchor’s session ID is different from the active session’s ID, the other user created the anchor.

Color Virtual Content Based on the User

To distinguish virtual content by user, you choose a different color for each user. The sample app uses the `toRandomColor` function to assign user colors.

```
let color = anchor.sessionIdentifier?.toRandomColor() ?? .white
```

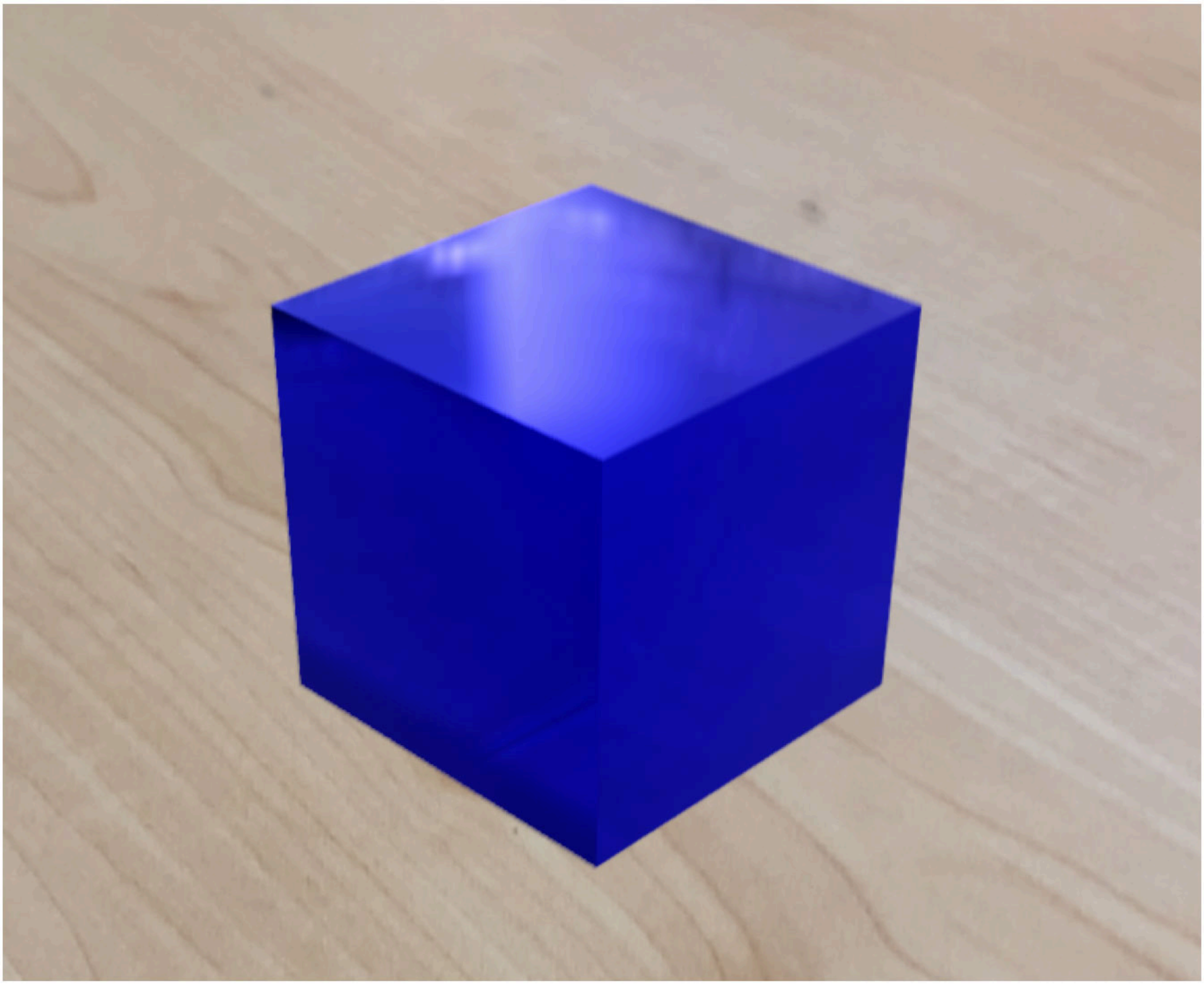
The random color function works by applying a modulo operation to the anchor’s session ID, and interpreting the result as an index into a color array.

```
func toRandomColor() -> UIColor {
    var firstFourUUIDBytesAsUInt32: UInt32 = 0
    let data = withUnsafePointer(to: self) {
        return Data(bytes: $0, count: MemoryLayout.size(ofValue: self))
    }
    _ = withUnsafeMutableBytes(of: &firstFourUUIDBytesAsUInt32, { data.copyBytes(to:

    let colors: [UIColor] = [.red, .green, .blue, .yellow, .magenta, .cyan, .purple,
    .orange, .brown, .lightGray, .gray, .darkGray, .black, .white]

    let randomNumber = Int(firstFourUUIDBytesAsUInt32) % colors.count
    return colors[randomNumber]
}
```

Place virtual content

When ARKit notifies you of a new nonparticipant anchor in `session(_:didAdd:)`, place a block geometry tinted with the color calculated in the previous section.

```
let coloredCube = ModelEntity(mesh: MeshResource.generateBox(size: boxLength),
                              materials: [SimpleMaterial(color: color, isMetallic: t
// Offset the cube by half its length to align its bottom with the real-world surfac
coloredCube.position = [0, boxLength / 2, 0]

// Attach the cube to the ARAnchor via an AnchorEntity.
//   World origin -> ARAnchor -> AnchorEntity -> ModelEntity
let anchorEntity = AnchorEntity(anchor: anchor)
anchorEntity.addChild(coloredCube)
arView.scene.addAnchor(anchorEntity)
```



The sample app uses common ARKit techniques to place virtual objects. For more information about mapping screen touches to real-world locations, see Detecting Images in an AR Experience.

See Also

Shared Experiences

{} Streaming an AR experience

Control an AR experience remotely by transferring sensor and user input over the network.

{} Creating a multiuser AR experience

Enable nearby devices to share an AR experience by using a host-guest multiuser strategy.

`class ARParticipantAnchor`

An anchor for another user in multiuser augmented reality experiences.

`class CollaborationData`

An object that holds information that a user has collected about the physical environment.