#WWDC19

Introducing ARKit 3

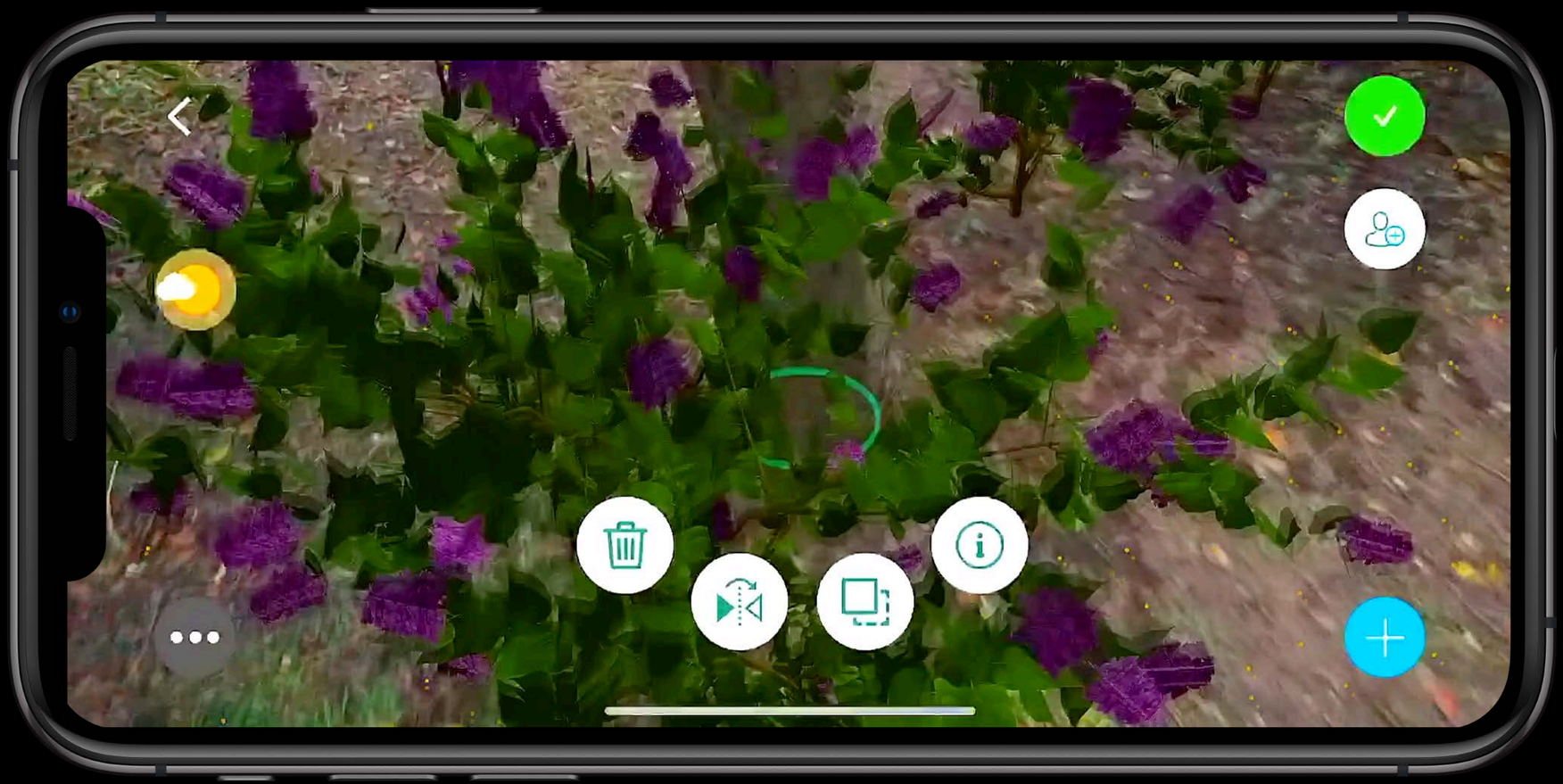
Andreas Moeller, ARKit Engineer
Thomas Berton, ARKit Engineer

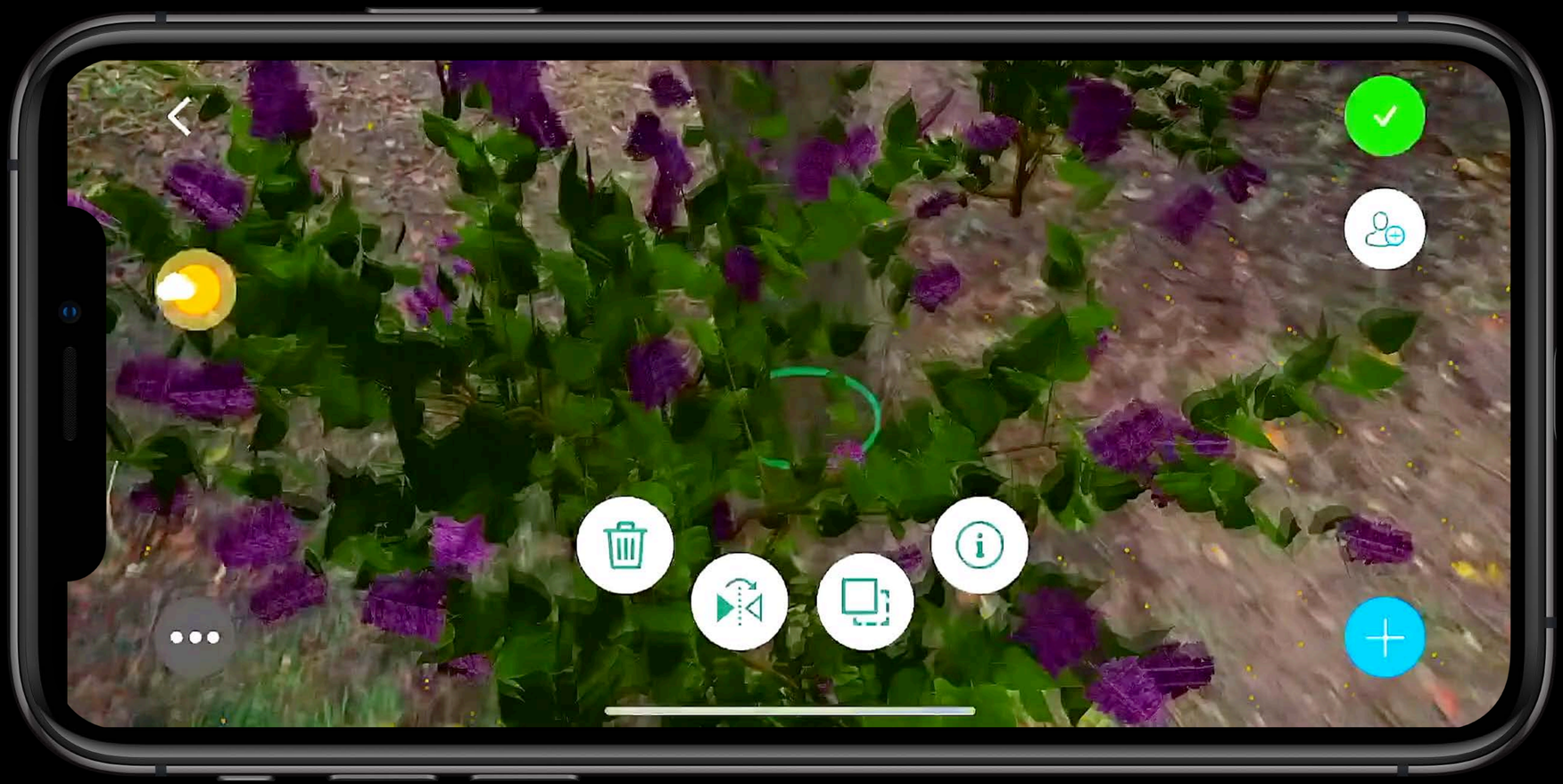
© 2019 Apple Inc. All rights reserved. Redistribution or public display not permitted without written permission from Apple.

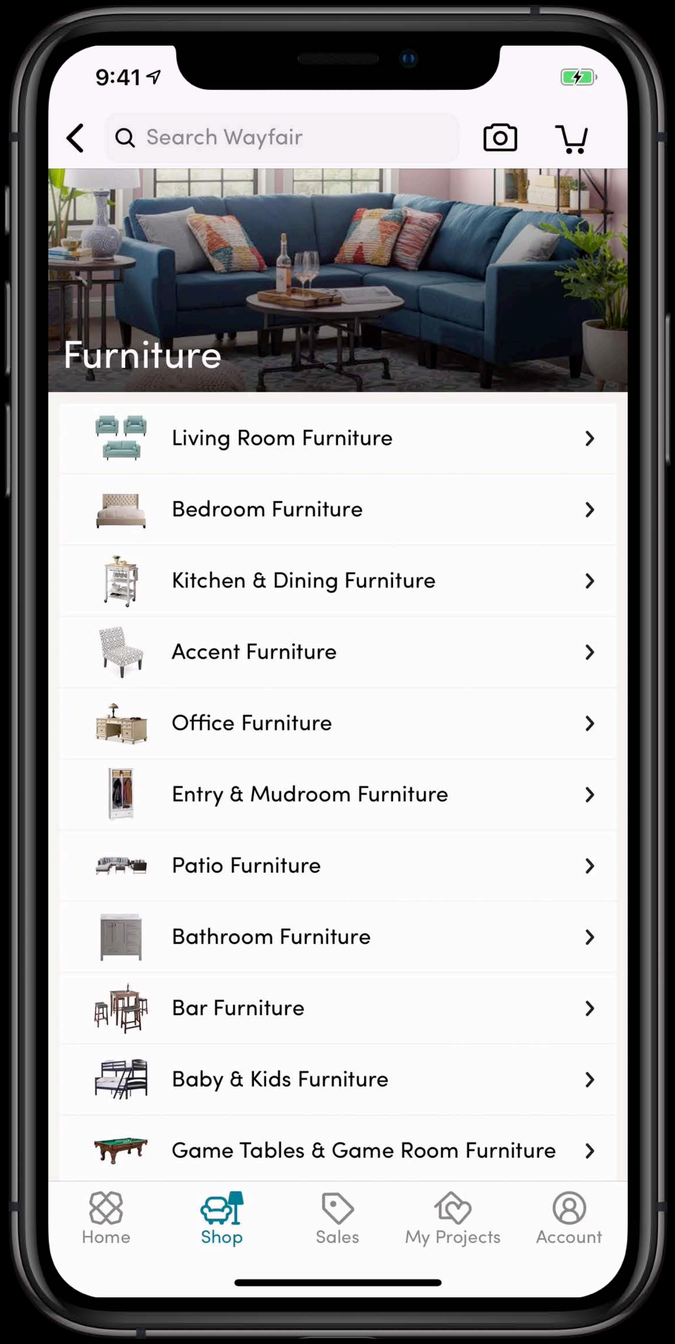
Introducing ARKit 3

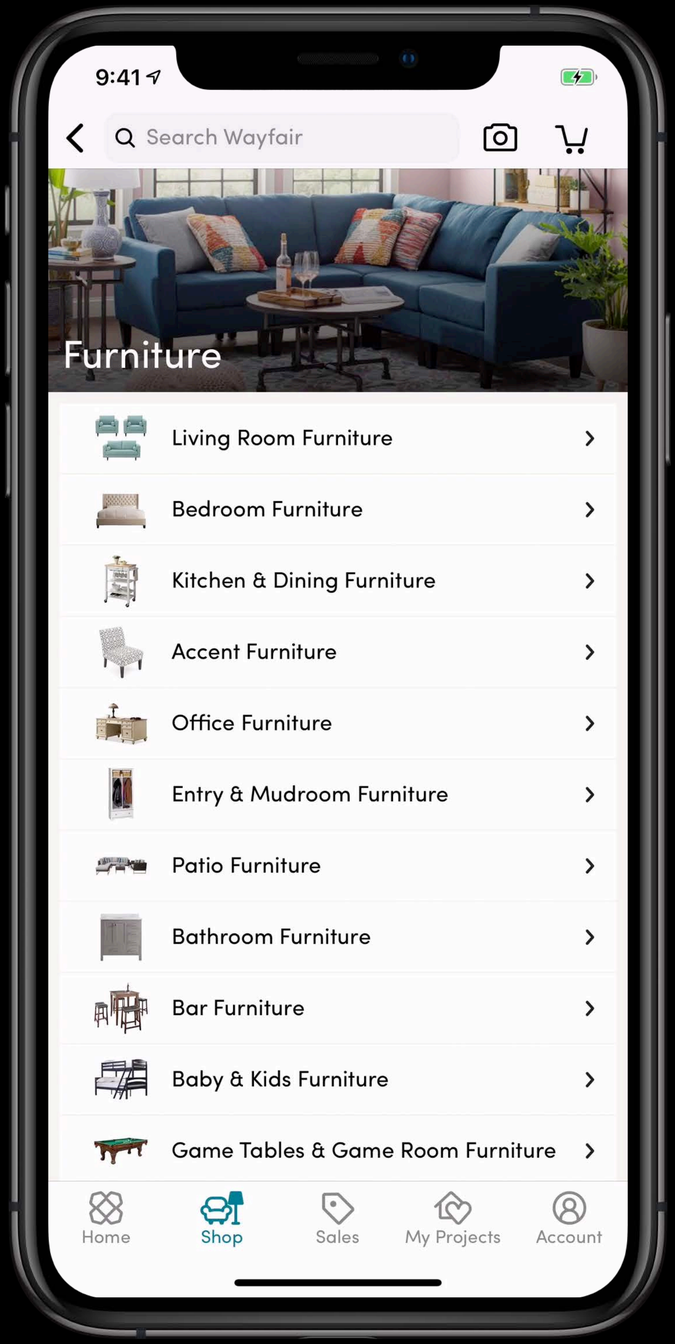

TNT
TNT

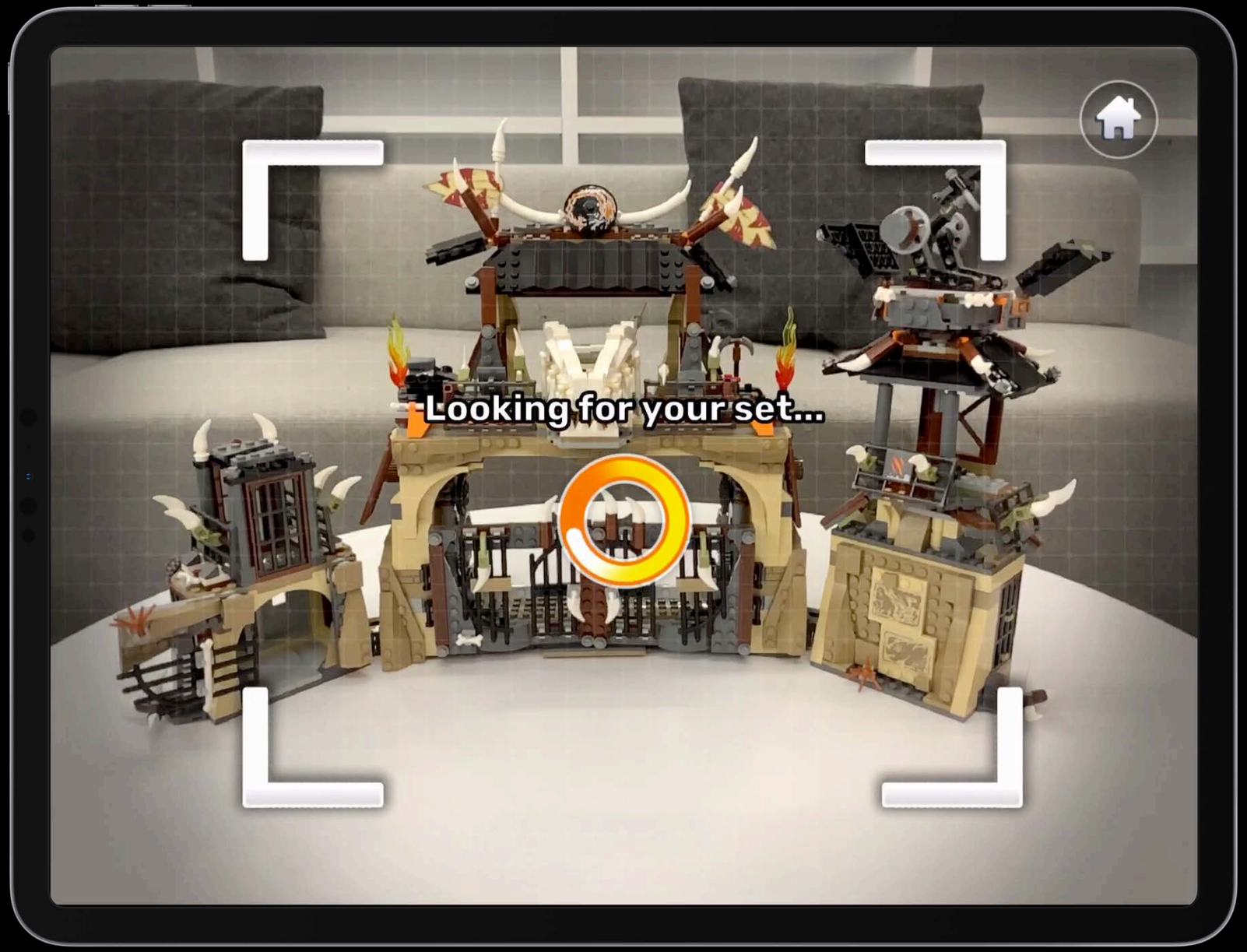

TNT
TNT

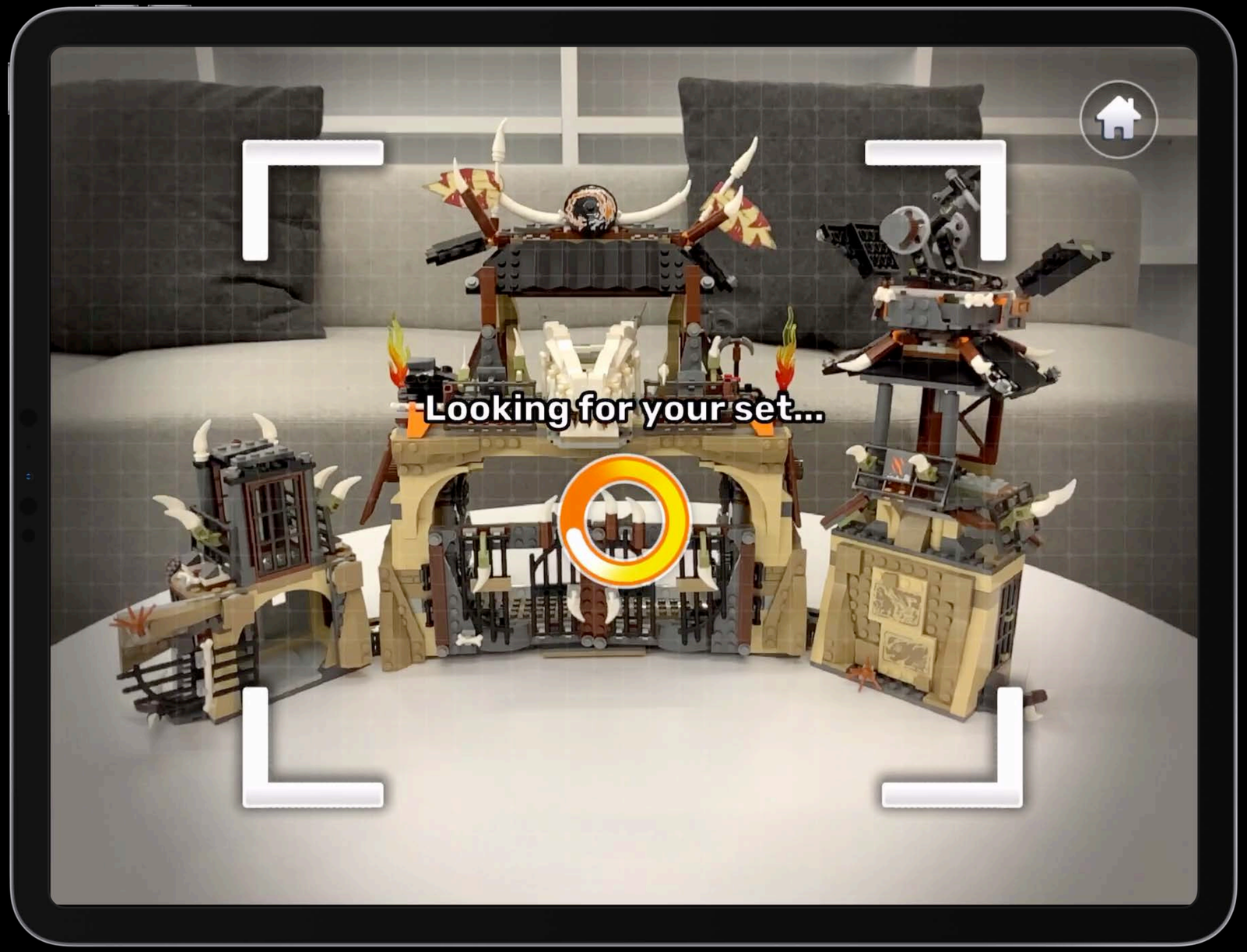




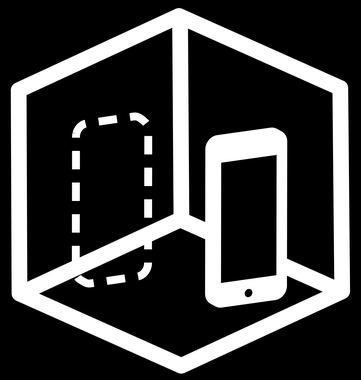

9:41
Search Wayfair
Furniture
Living Room Furniture
Bedroom Furniture
Kitchen & Dining Furniture
Accent Furniture
Office Furniture
Entry & Mudroom Furniture
Patio Furniture
Bathroom Furniture
Bar Furniture
Baby & Kids Furniture
Game Tables & Game Room Furniture
Home
Shop
Sales
My Projects
Account


9:41
Search Wayfair
Furniture
Living Room Furniture
Bedroom Furniture
Kitchen & Dining Furniture
Accent Furniture
Office Furniture
Entry & Mudroom Furniture
Patio Furniture
Bathroom Furniture
Bar Furniture
Baby & Kids Furniture
Game Tables & Game Room Furniture
Home
Shop
Sales
My Projects
Account


Looking for your set...

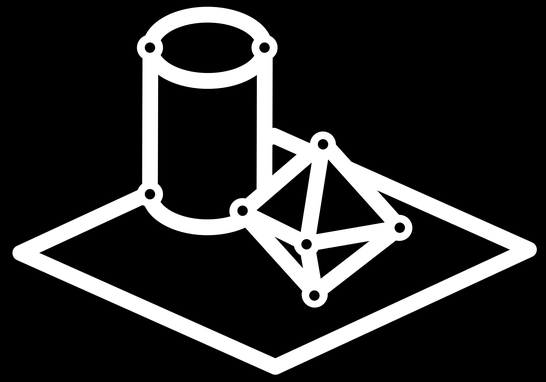

Looking for your set...

ARKit Recap



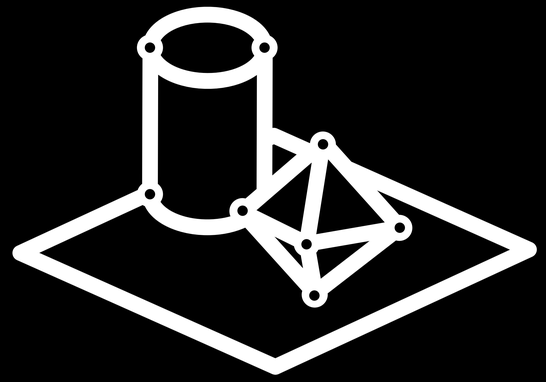
Tracking

Tracking

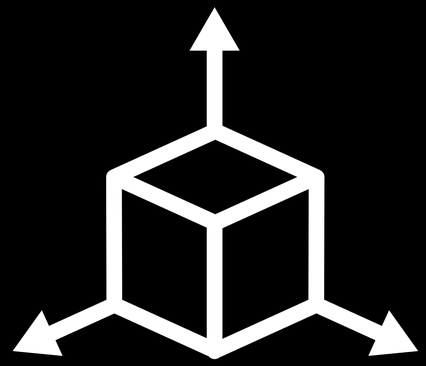


Scene Understanding

Tracking



Scene Understanding



Rendering

Rendering with ARKit

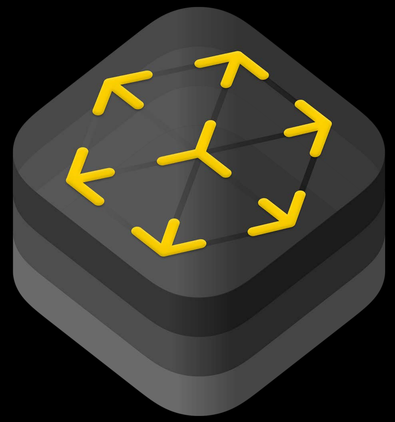
SceneKit

SpriteKit

Metal

Rendering with ARKit

RealityKit



Introducing ARKit 3

Visual Coherence

HDR Environment Textures

Faster Reference Image Loading

Motion Capture

Positional Tracking

Face Tracking Enhancements

Detect up to 100 Images

Simultaneous Front and Back Camera

Auto-detect Image Size

AR Coaching UI

Record and Replay of Sequences

People Occlusion

Introducing ARKit 3

More Robust 3D Object Detection

RealityKit Integration

Raycasting

ML Based Plane Detection

Multiple-face Tracking

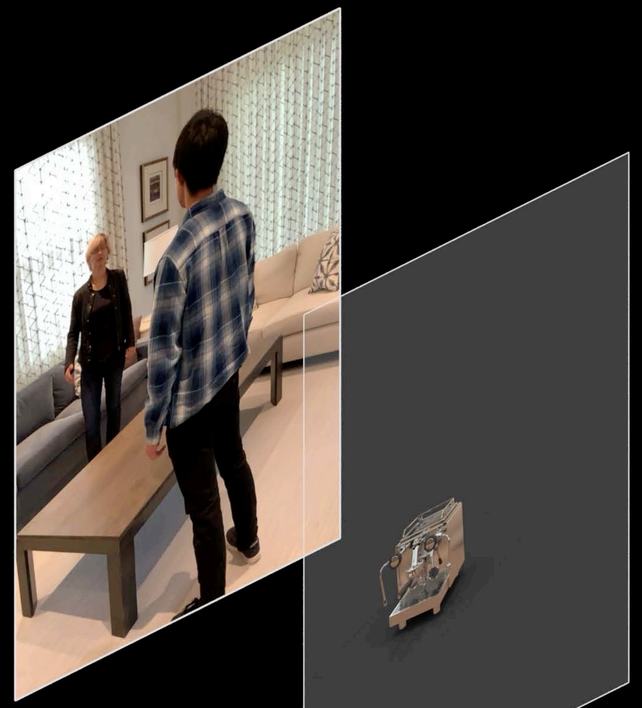
Collaborative Session

New Plane Classes

AR QuickLook Additions

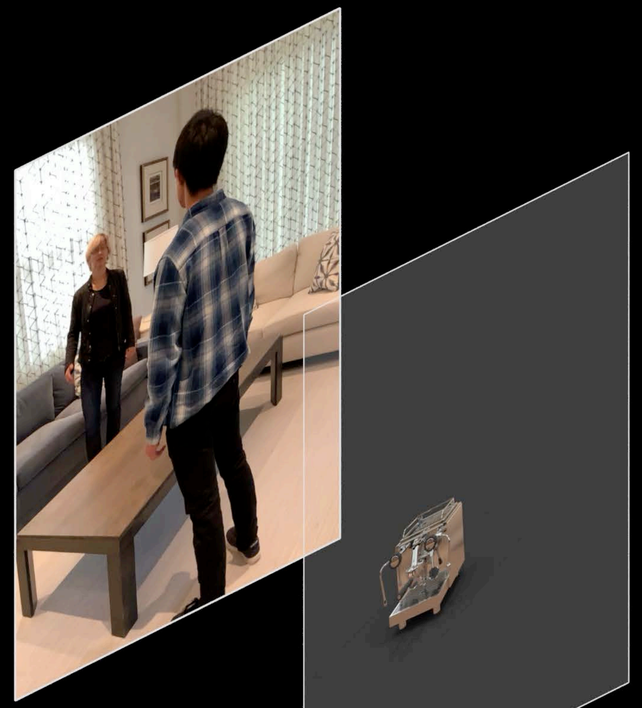
People Occlusion


Camera
Rendering

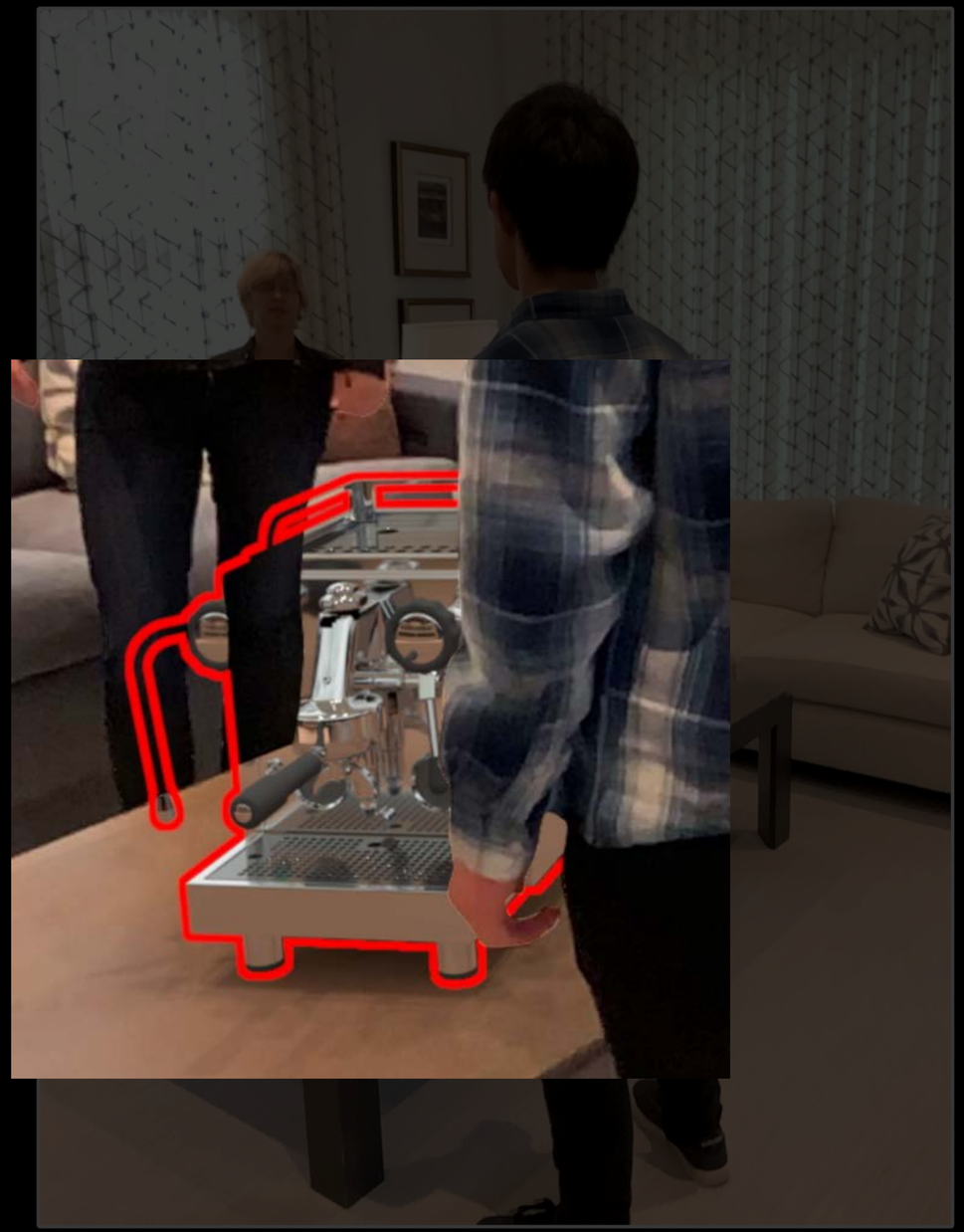


Camera

Rendering



Segmentation Camera Rendering People




Segmentation
Camera
Rendering
People

Segmentation + Depth

Camera

People

Rendering

People

People Occlusion

People Occlusion

Enables virtual content to be rendered behind people

People Occlusion

Enables virtual content to be rendered behind people

Works for multiple people in the scene

People Occlusion

Enables virtual content to be rendered behind people

Works for multiple people in the scene

Works for fully and partially visible people

People Occlusion

Enables virtual content to be rendered behind people

Works for multiple people in the scene

Works for fully and partially visible people

Integrated with ARView and ARSCNView

People Occlusion

Enables virtual content to be rendered behind people

Works for multiple people in the scene

Works for fully and partially visible people

Integrated with ARView and ARSCNView

Depth estimation

People Occlusion

Enables virtual content to be rendered behind people

Works for multiple people in the scene

Works for fully and partially visible people

Integrated with ARView and ARSCNView

Depth estimation

Available on A12 and later

People Occlusion

Frame semantics

```
class ARConfiguration : NSObject {

    var frameSemantics: ARConfiguration.FrameSemantics { get set }

    class func supportsFrameSemantics(ARConfiguration.FrameSemantics) -> Bool

}
```

People Occlusion

Frame semantics

```
class ARConfiguration : NSObject {

    var frameSemantics: ARConfiguration.FrameSemantics { get set }

    class func supportsFrameSemantics(ARConfiguration.FrameSemantics) -> Bool

}
```

People Occlusion

Frame semantics

```
class ARConfiguration : NSObject {

    var frameSemantics: ARConfiguration.FrameSemantics { get set }

    class func supportsFrameSemantics(ARConfiguration.FrameSemantics) -> Bool

}
```

People Occlusion

Frame semantics options

Person Segmentation

Person Segmentation with Depth

```
let configuration = ARWorldTrackingConfiguration()
configuration.frameSemantics = .personSegmentation
session.run(configuration)
```

People Occlusion

Frame semantics options

Person Segmentation

Person Segmentation with Depth

```
let configuration = ARWorldTrackingConfiguration()
configuration.frameSemantics = .personSegmentationWithDepth
session.run(configuration)
```

People Occlusion

Additional available data

Segmentation buffer

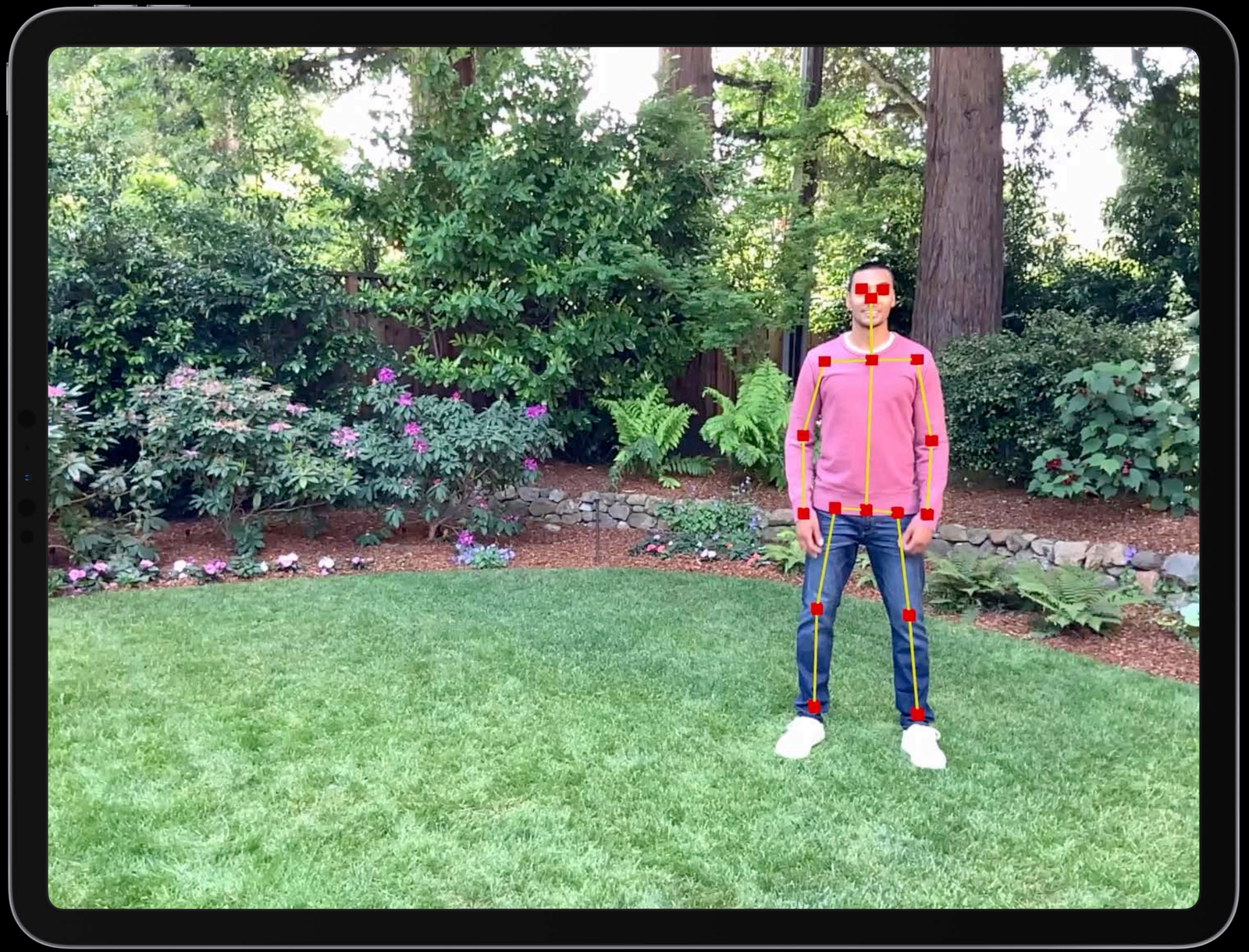
Estimated depth data buffer

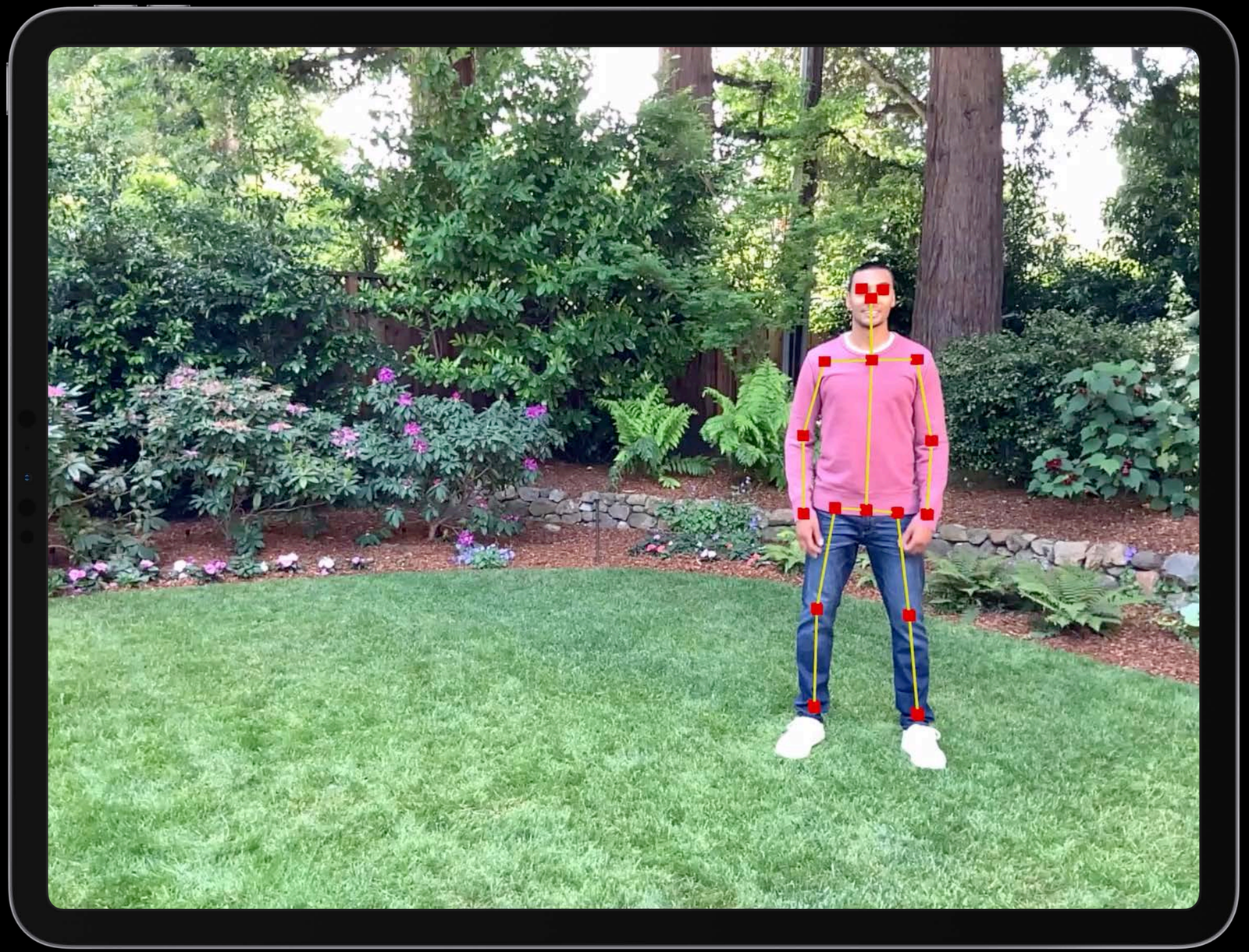
```
open class ARFrame : NSObject, NSCopying {

    open var segmentationBuffer: CVPixelBuffer? { get }

    open var estimatedDepthData: CVPixelBuffer? { get }

}
```

Demo

People occlusion

Motion Capture





Motion Capture

Motion Capture

Tracks human body in 2D and 3D

Motion Capture

Tracks human body in 2D and 3D

Provides skeleton representation

Motion Capture

Tracks human body in 2D and 3D

Provides skeleton representation

Enables driving a virtual character

Motion Capture

Tracks human body in 2D and 3D

Provides skeleton representation

Enables driving a virtual character

Available on A12 and later

2D Body Detection

Setup

Frame semantics option

Supported on world, image, and orientation tracking configurations

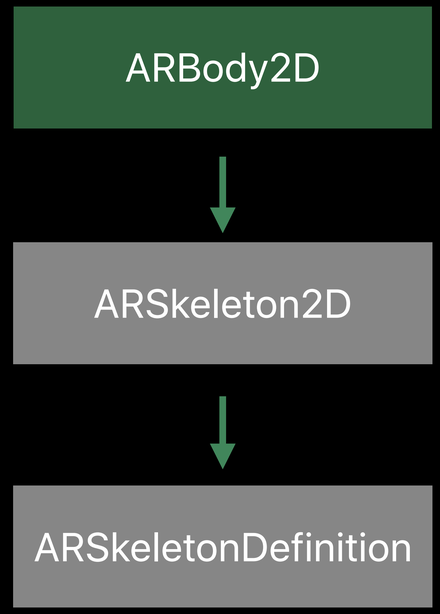
```
let configuration = ARWorldTrackingConfiguration()
configuration.frameSemantics = .bodyDetection
session.run(configuration)
```

2D Body Detection

Result data

ARBody2D provided in ARFrame

Contains 2D skeleton

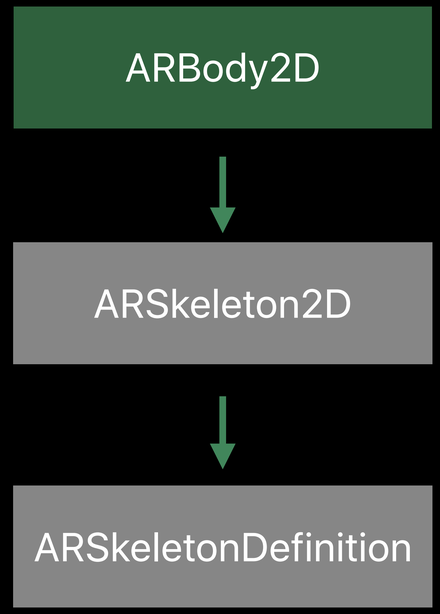


2D Body Detection

Result data

ARBody2D provided in ARFrame

Contains 2D skeleton



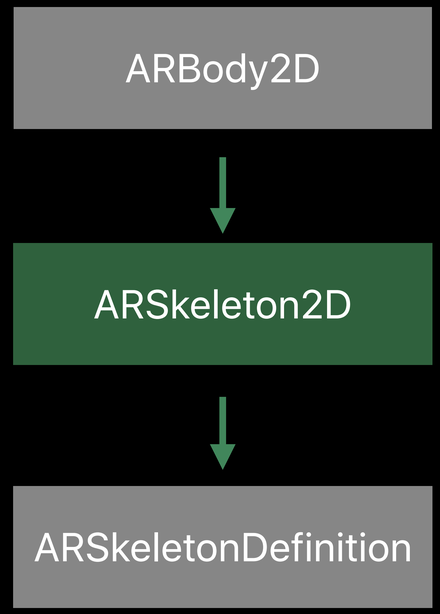

```
open class ARFrame : NSObject, NSCopying {
    open let detectedBody: ARBody2D?
}
```

ARSkeleton2D

Joint landmarks in normalized image space

Flat hierarchy for efficient processing

Skeleton definition

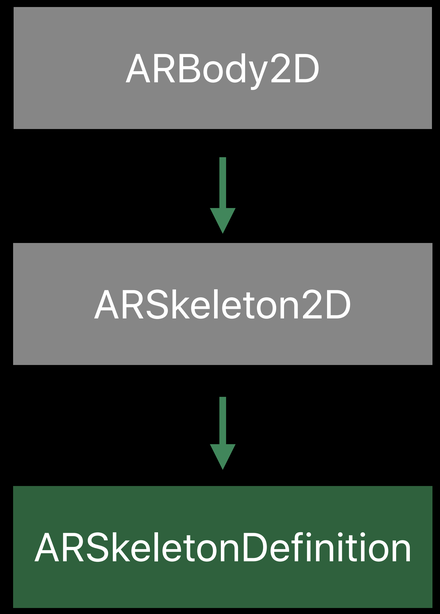


ARSkeletonDefinition

Information how to interpret skeleton data

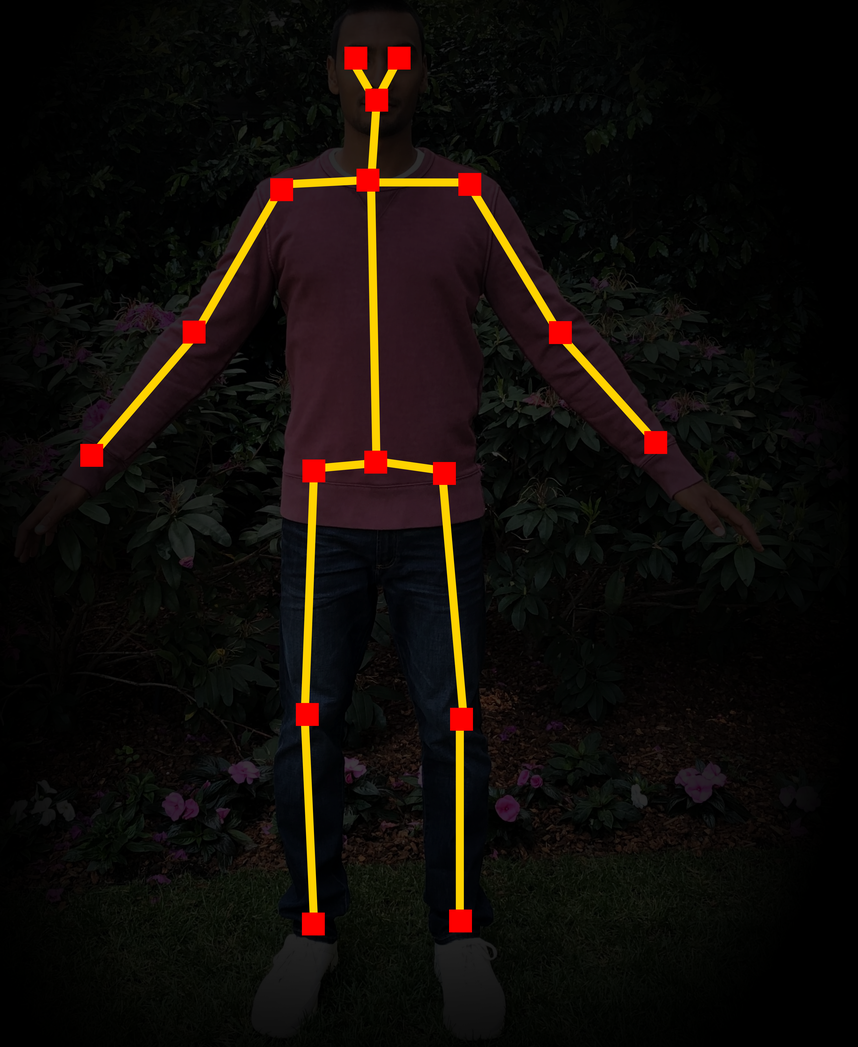
Joint hierarchy

Named joints

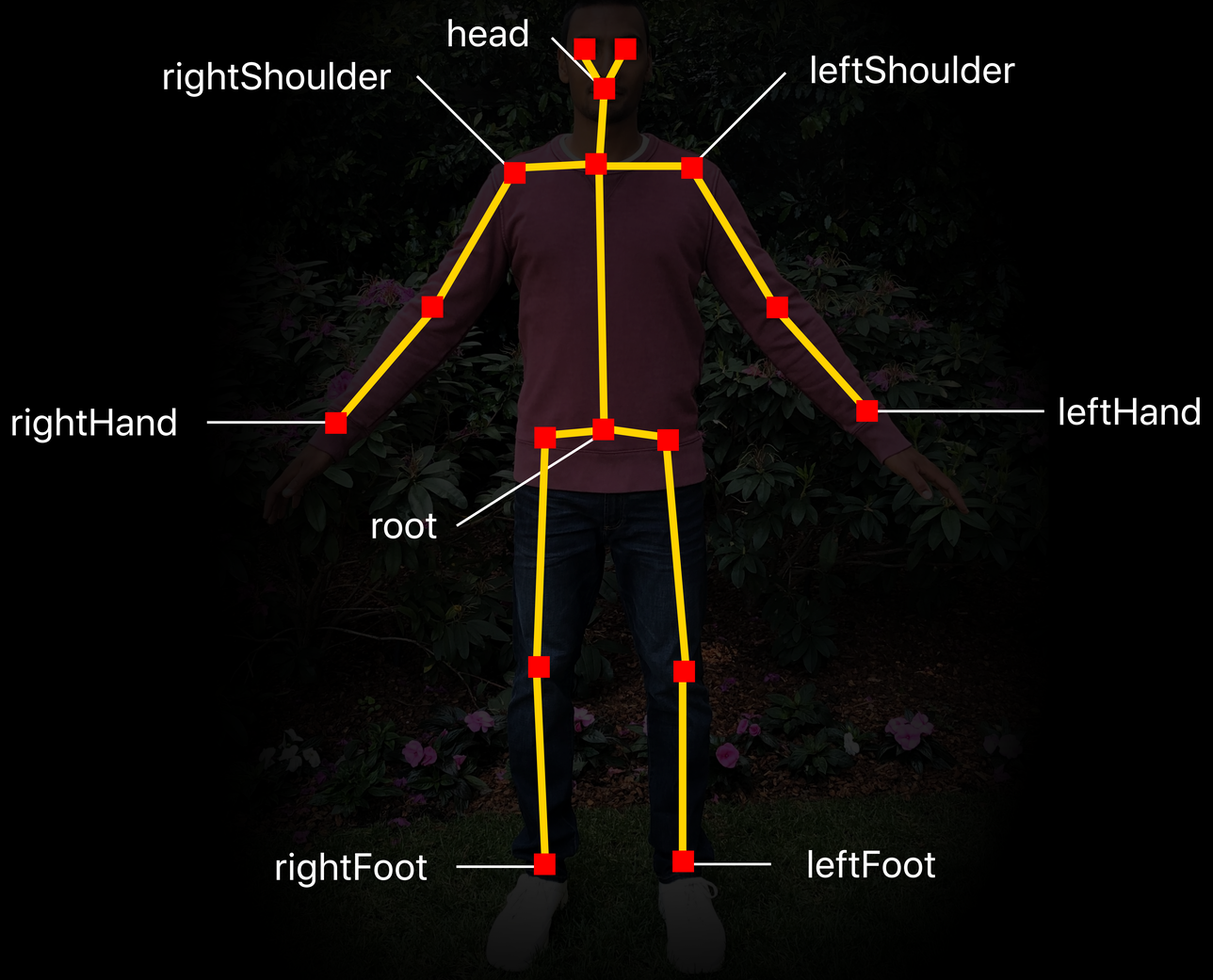


2D Skeleton Representation

2D Skeleton Representation



2D Skeleton Representation



3D Motion Capture

Concept

Tracks a human body pose in 3D space

Provides a 3D skeleton representation

Provides scale estimation

Anchored in world coordinates

ARBodyTrackingConfiguration

NEW

3D body tracking

2D body detection frame semantics enabled by default

Selected world tracking features available

ARBodyTrackingConfiguration

NEW

3D body tracking

2D body detection frame semantics enabled by default

Selected world tracking features available

```
if ARBodyTrackingConfiguration.isSupported {
   let configuration = ARBodyTrackingConfiguration()
   session.run(configuration)
}
```

ARBodyAnchor

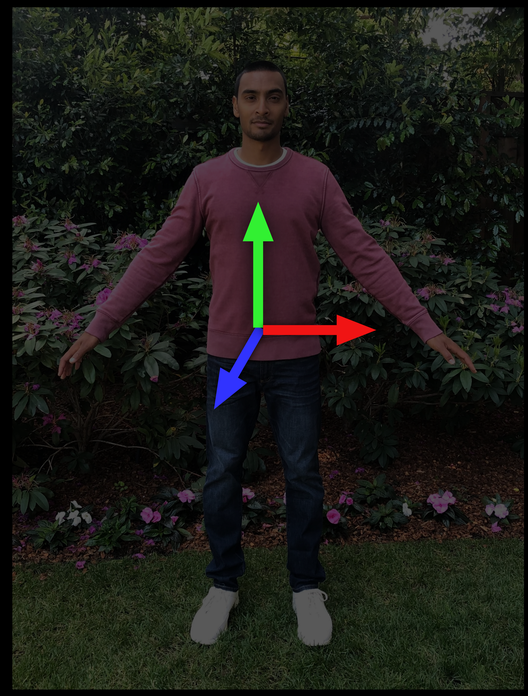
NEW

```
open class ARBodyAnchor : ARAnchor {

    open var transform: simd_float4x4 { get }

    open var estimatedScaleFactor: Float { get }

    open var skeleton: ARSkeleton3D { get }

}
```

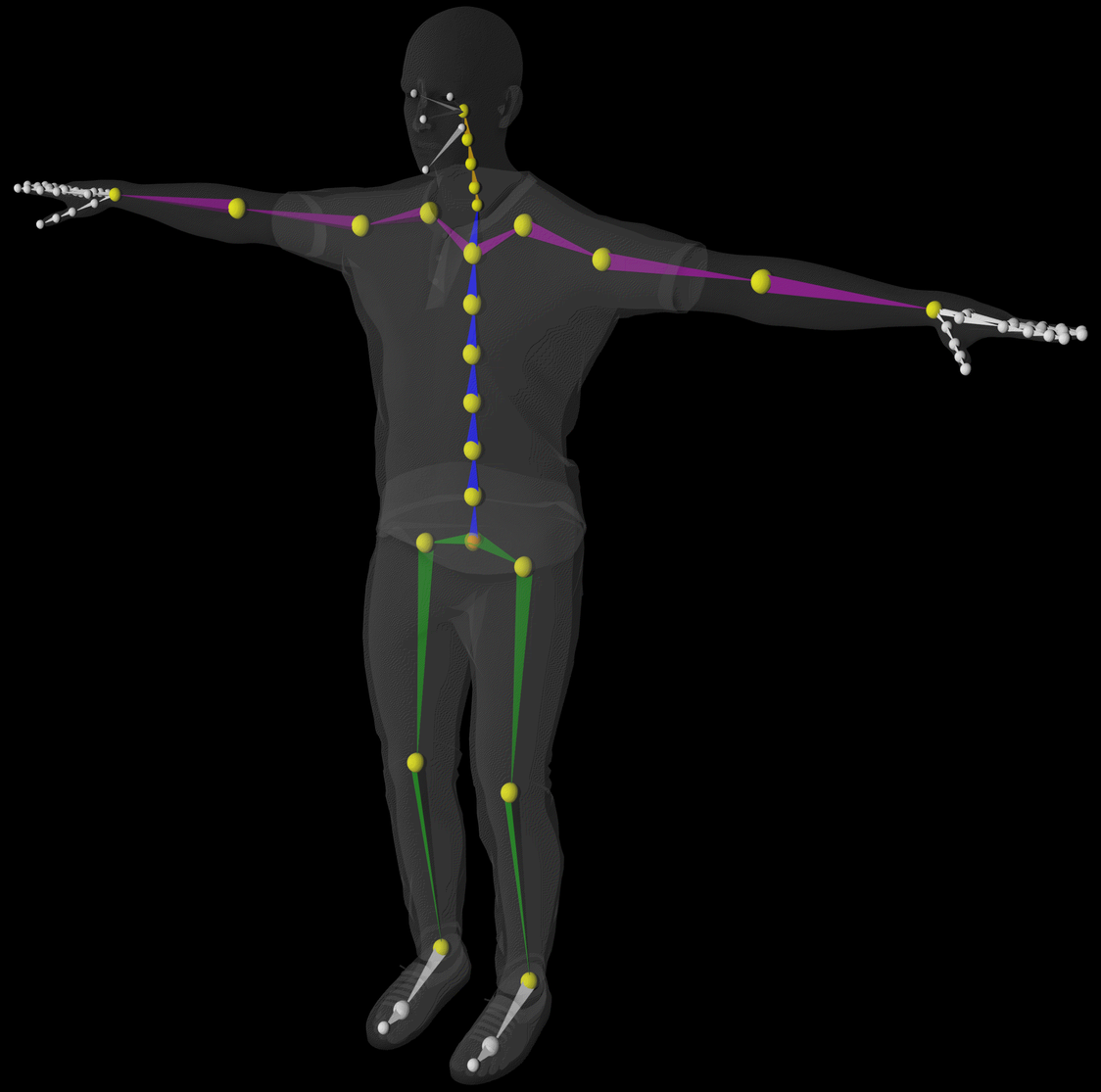
ARBodyAnchor

NEW

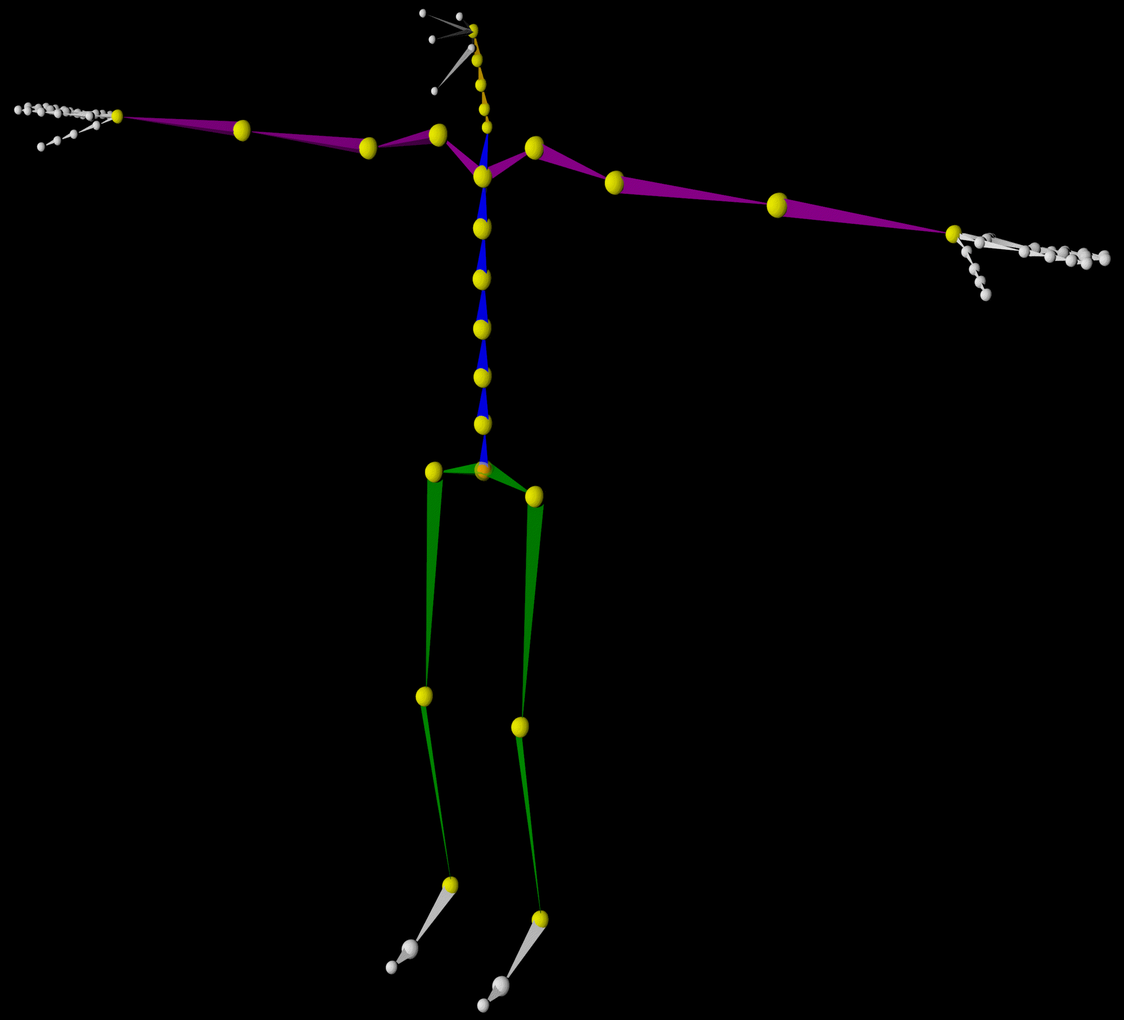
```
open class ARBodyAnchor : ARAnchor {

    open var transform: simd_float4x4 { get }

    open var estimatedScaleFactor: Float { get }

    open var skeleton: ARSkeleton3D { get }

}
```



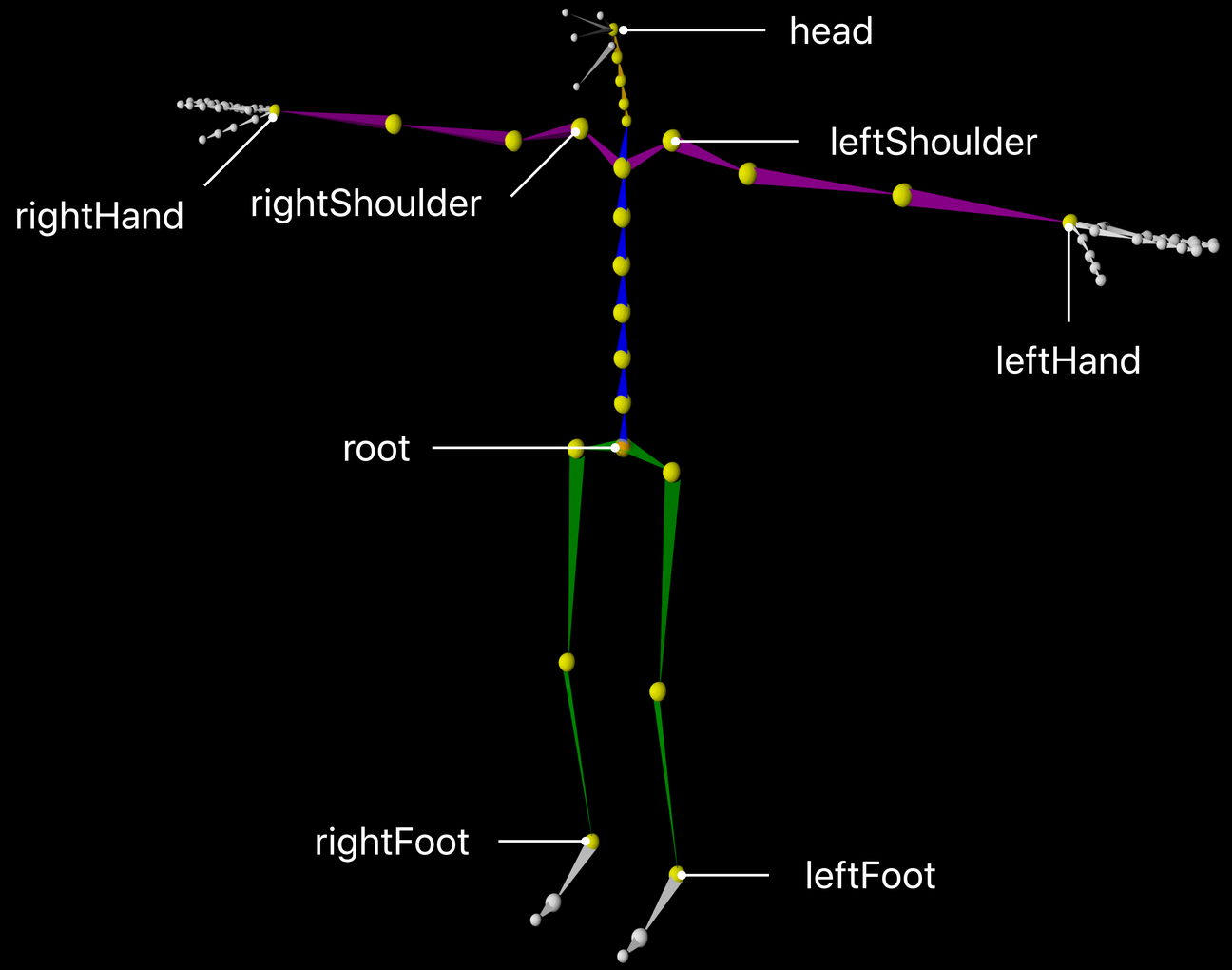
3D Skeleton Representation



3D Skeleton Representation



3D Skeleton Representation



Animating 3D Characters

Animating 3D Characters

Drive a model based on 3D skeleton pose

Animating 3D Characters

Drive a model based on 3D skeleton pose

Requires rigged mesh

Animating 3D Characters

Drive a model based on 3D skeleton pose

Requires rigged mesh

Built into RealityKit

Animating 3D Characters

Drive a model based on 3D skeleton pose

Requires rigged mesh

Built into RealityKit

BodyTrackedEntity

```
// Animating a 3D character with RealityKit

// Add body anchor
let bodyAnchor = AnchorEntity(.body)
arView.scene.addAnchor(bodyAnchor)

// Load rigged mesh
Entity.loadBodyTrackedAsync(named: "robot").sink(receiveValue: { (character) in

   // Assign body anchor
   bodyAnchor.addChild(character)
})
```

```
// Animating a 3D character with RealityKit

// Add body anchor
let bodyAnchor = AnchorEntity(.body)
arView.scene.addAnchor(bodyAnchor)

// Load rigged mesh
Entity.loadBodyTrackedAsync(named: "robot").sink(receiveValue: { (character) in

    // Assign body anchor
    bodyAnchor.addChild(character)
})
```

```
// Animating a 3D character with RealityKit

// Add body anchor
let bodyAnchor = AnchorEntity(.body)
arView.scene.addAnchor(bodyAnchor)

// Load rigged mesh
Entity.loadBodyTrackedAsync(named: "robot").sink(receiveValue: { (character) in

    // Assign body anchor
    bodyAnchor.addChild(character)
})
```

```
// Animating a 3D character with RealityKit

// Add body anchor
let bodyAnchor = AnchorEntity(.body)
arView.scene.addAnchor(bodyAnchor)

// Load rigged mesh
Entity.loadBodyTrackedAsync(named: "robot").sink(receiveValue: { (character) in

    // Assign body anchor
    bodyAnchor.addChild(character)
})
```

Simultaneous Front and Back Camera

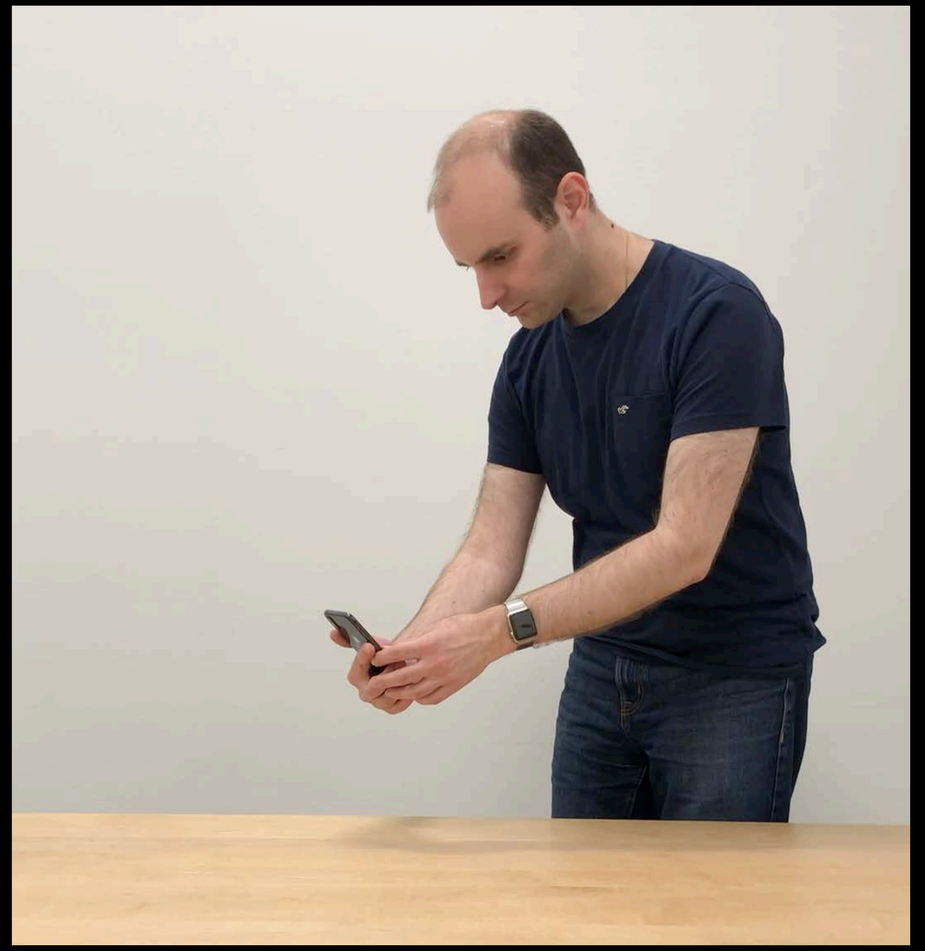
Simultaneous Front and Back Camera

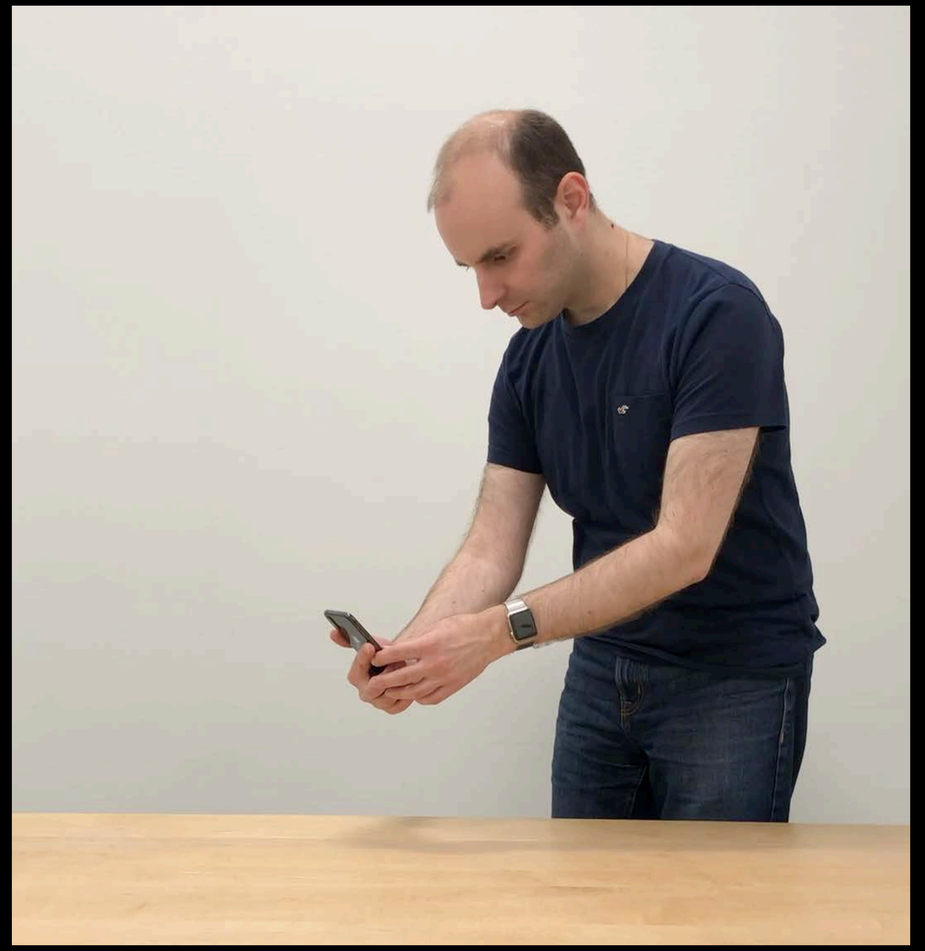
AR experiences using front and back camera

Enables World Tracking with face data

Enables Face Tracking with device orientation and position

Supported on A12 and later





```
// Enable face tracking in world tracking configuration

let configuration = ARWorldTrackingConfiguration()
if configuration.supportsUserFaceTracking {
    configuration.userFaceTrackingEnabled = true
}
session.run(configuration)
```

```
// Enable face tracking in world tracking configuration

let configuration = ARWorldTrackingConfiguration()
if configuration.supportsUserFaceTracking {
    configuration.userFaceTrackingEnabled = true
}
session.run(configuration)
```

```
// Enable face tracking in world tracking configuration

let configuration = ARWorldTrackingConfiguration()
if configuration.supportsUserFaceTracking {
    configuration.userFaceTrackingEnabled = true
}
session.run(configuration)
```

```
// Enable face tracking in world tracking configuration

let configuration = ARWorldTrackingConfiguration()
if configuration.supportsUserFaceTracking {
    configuration.userFaceTrackingEnabled = true
}
session.run(configuration)


// Receive face data

func session(_ session: ARSession, didAdd anchors: [ARAnchor]) {
    for anchor in anchors where anchor is ARFaceAnchor {
        …
    }
}
```

```
// Enable face tracking in world tracking configuration

let configuration = ARWorldTrackingConfiguration()
if configuration.supportsUserFaceTracking {
    configuration.userFaceTrackingEnabled = true
}
session.run(configuration)



// Receive face data

func session(_ session: ARSession, didAdd anchors: [ARAnchor]) {
    for anchor in anchors where anchor is ARFaceAnchor {
        …
    }
}
```

```
// Enable world tracking in face tracking configuration

let configuration = ARFaceTrackingConfiguration()
if configuration.supportsWorldTracking {
   configuration.worldTrackingEnabled = true
}
session.run(configuration)
```

```
// Enable world tracking in face tracking configuration

let configuration = ARFaceTrackingConfiguration()
if configuration.supportsWorldTracking {
   configuration.worldTrackingEnabled = true
}
session.run(configuration)
```

```
// Enable world tracking in face tracking configuration

let configuration = ARFaceTrackingConfiguration()
if configuration.supportsWorldTracking {
    configuration.worldTrackingEnabled = true
}
session.run(configuration)
```

```swift
// Enable world tracking in face tracking configuration

let configuration = ARFaceTrackingConfiguration()
if configuration.supportsWorldTracking {
   configuration.worldTrackingEnabled = true
}
session.run(configuration)


// Access world position and orientation

func session(_ session: ARSession, didUpdate frame: ARFrame) {
   let transform = frame.camera.transform
   …
}
```

```swift
// Enable world tracking in face tracking configuration

let configuration = ARFaceTrackingConfiguration()
if configuration.supportsWorldTracking {
    configuration.worldTrackingEnabled = true
}
session.run(configuration)



// Access world position and orientation

func session(_ session: ARSession, didUpdate frame: ARFrame) {
    let transform = frame.camera.transform
    …
}
```

Collaborative Session

Thomas Berton, ARKit Engineer

Saving and Loading Maps

Recap

Enables multi-user experiences

Ability to save and load ARWorldMap on multiple devices

One-time map sharing between devices

Collaborative Session

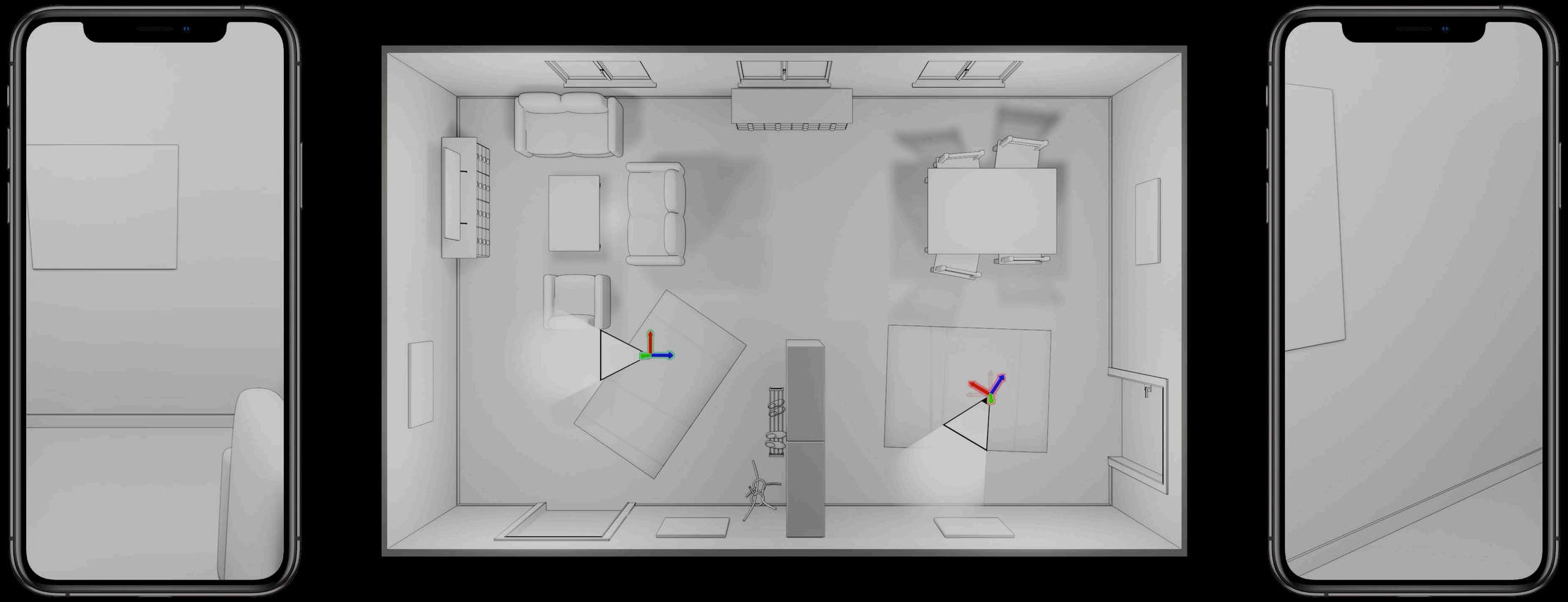
NEW

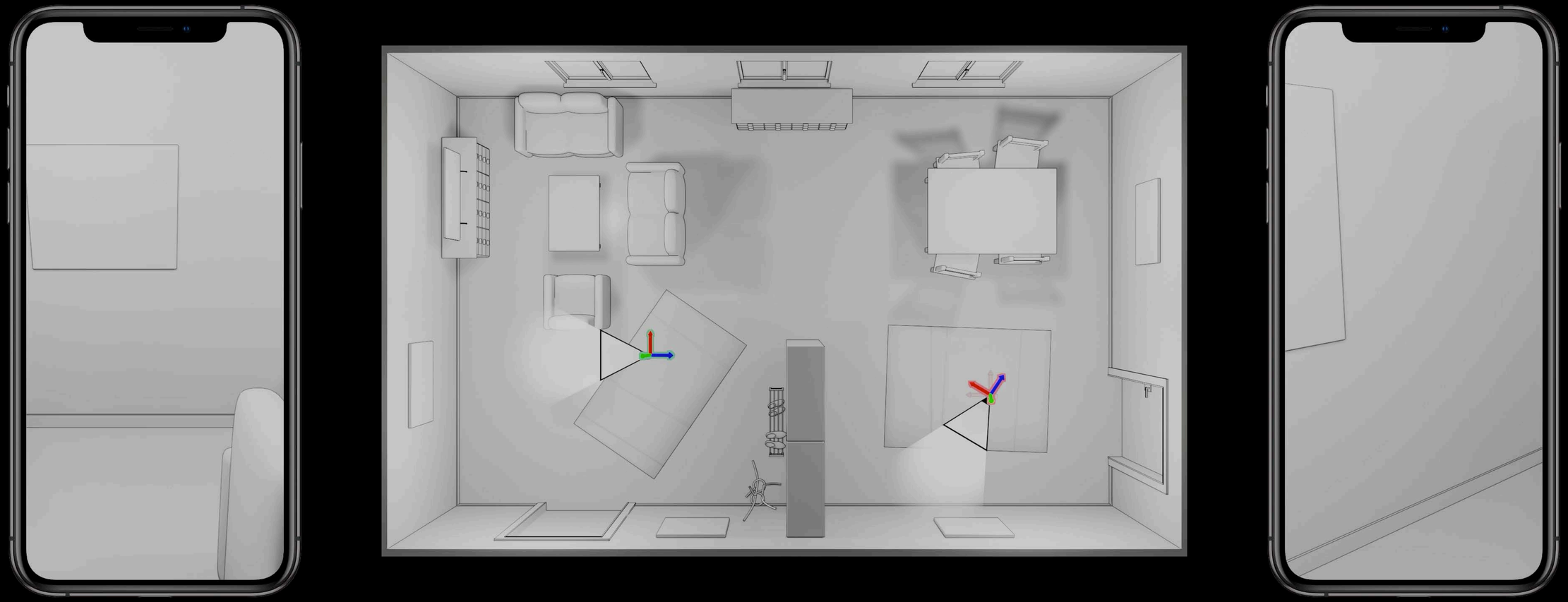
Continuously shares mapping information between multiple devices

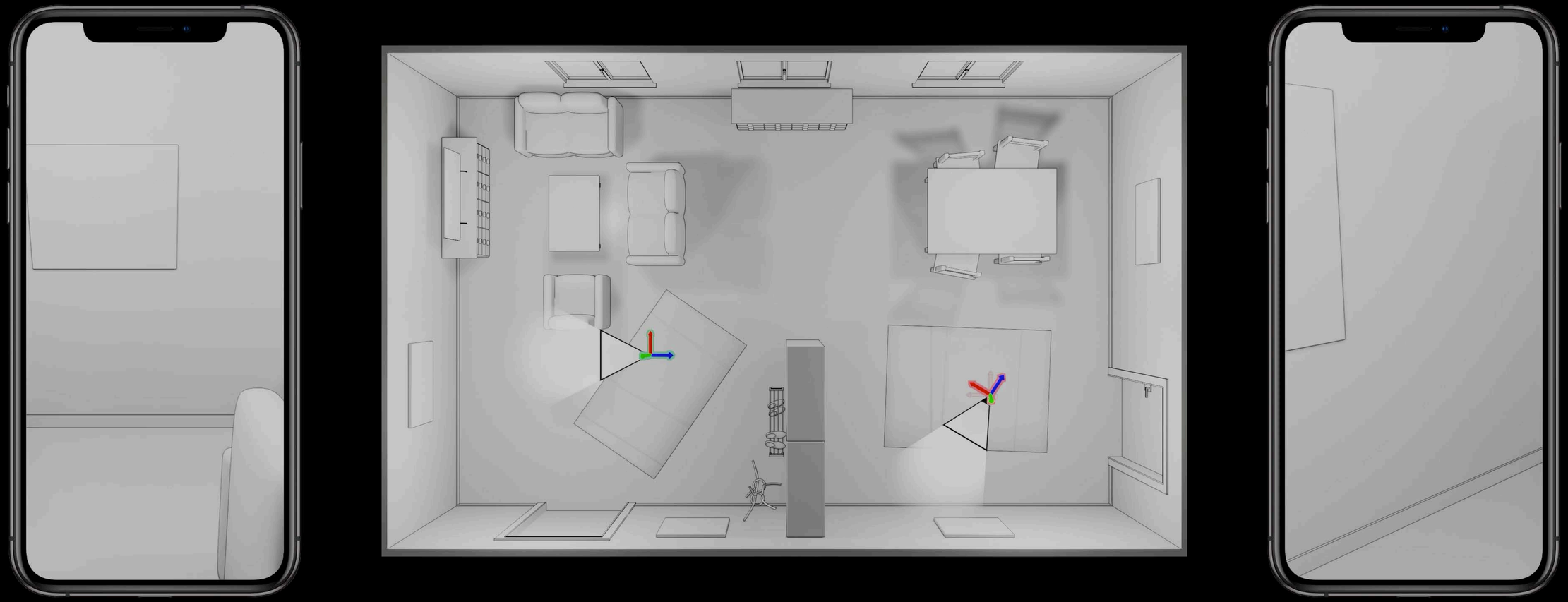
Ad-hoc multi-user experiences

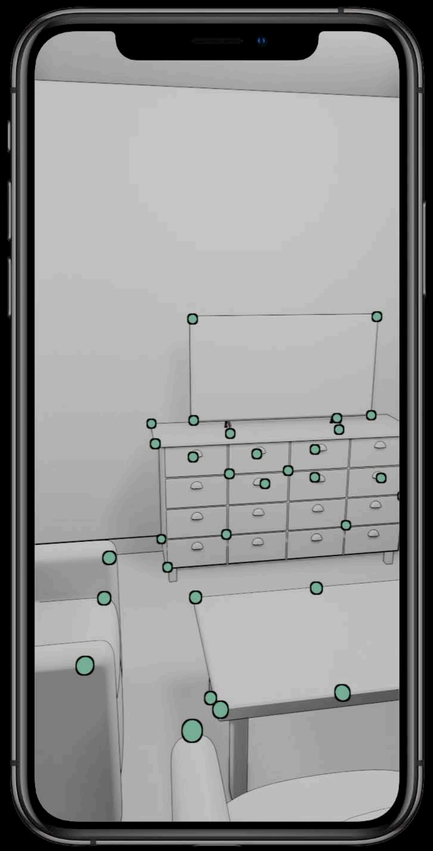
ARAnchors shared and identifiable on all devices

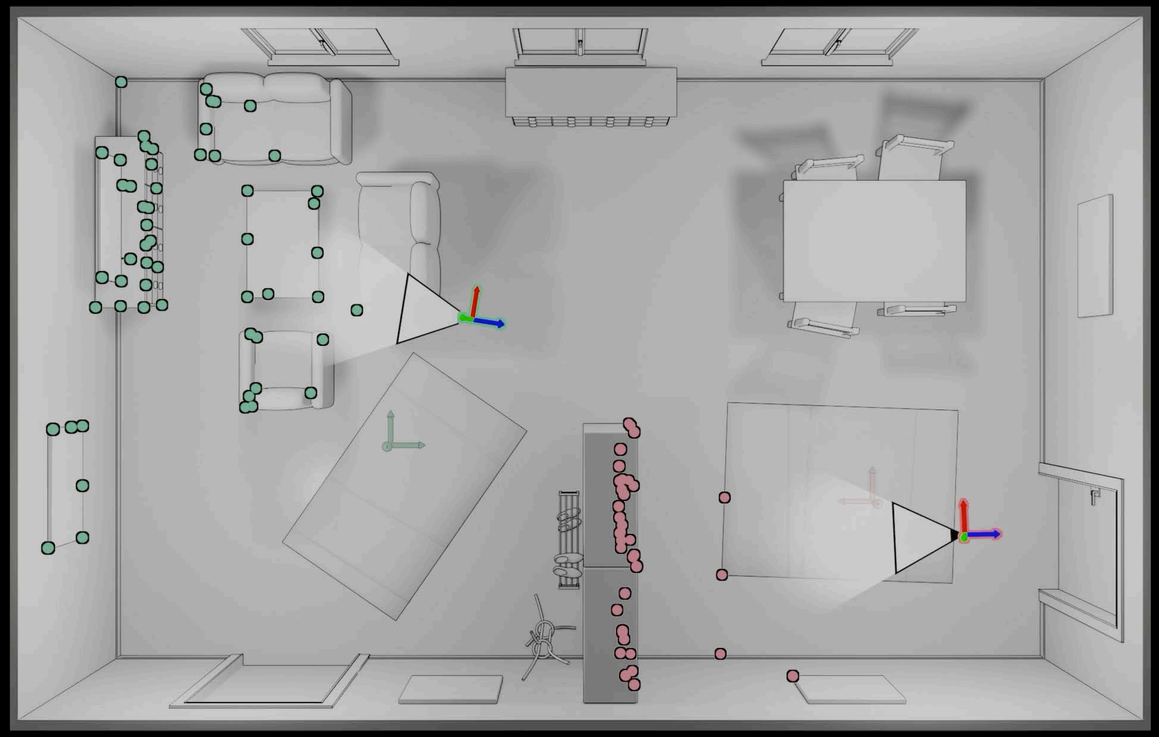
Individual coordinate systems

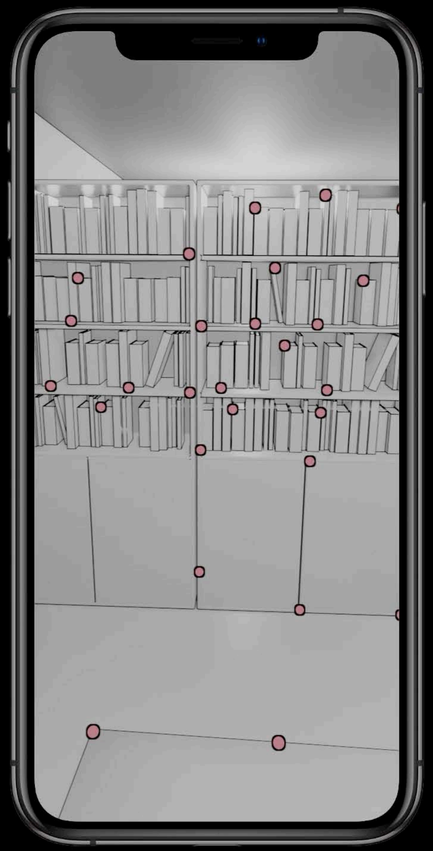


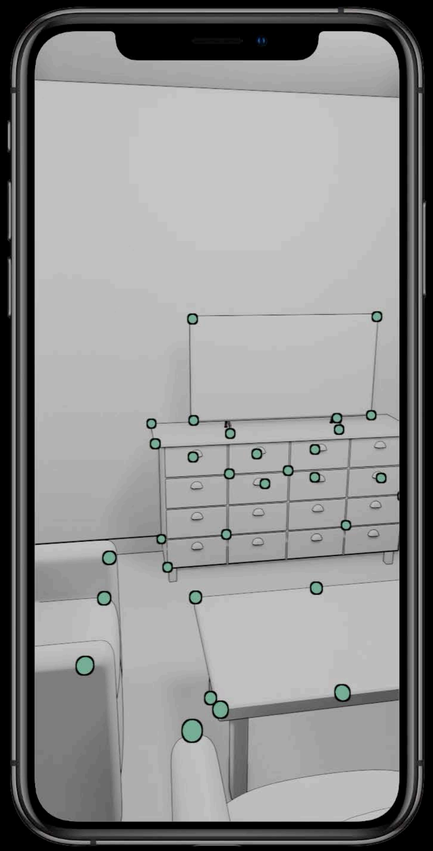


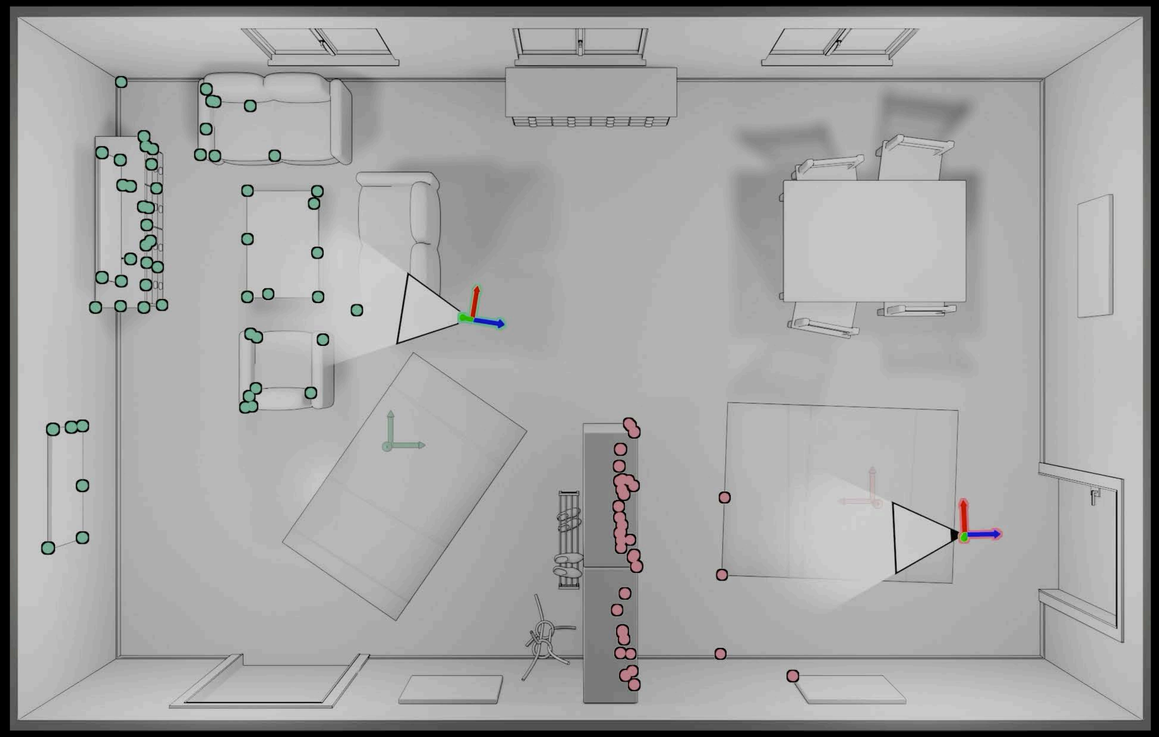


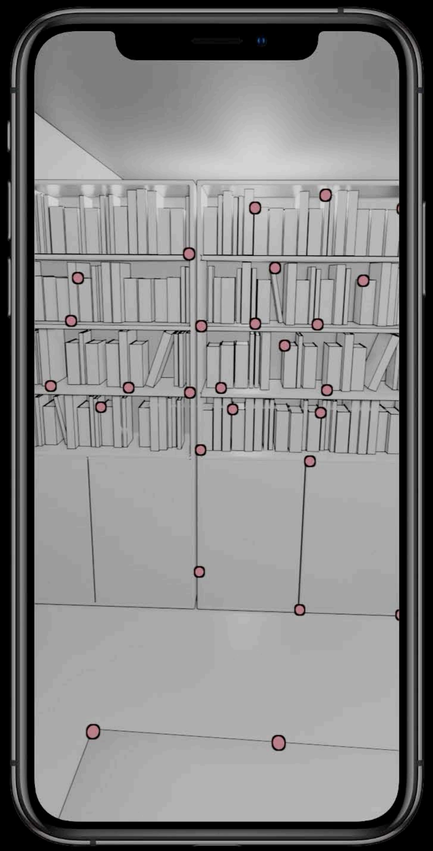


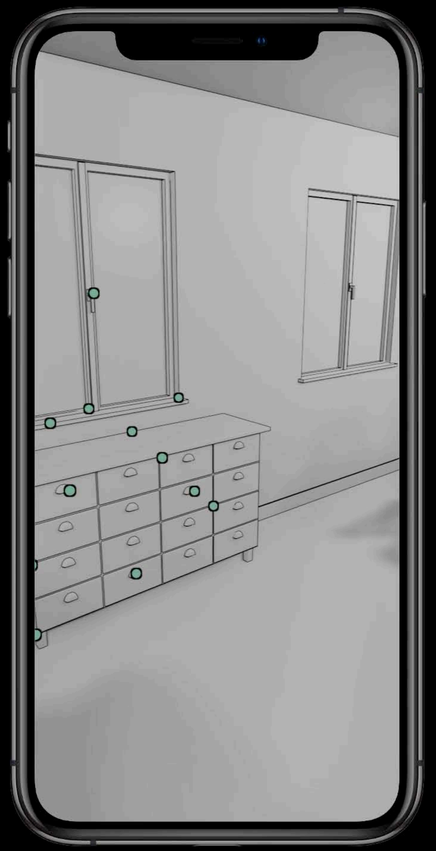


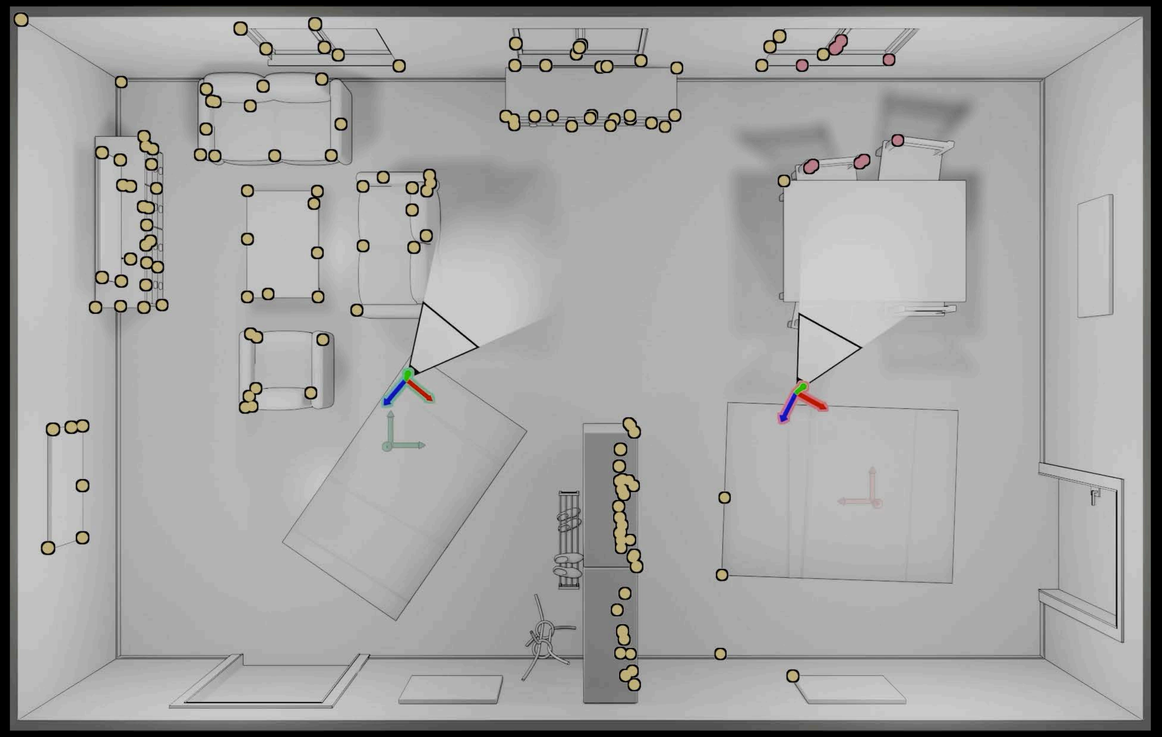


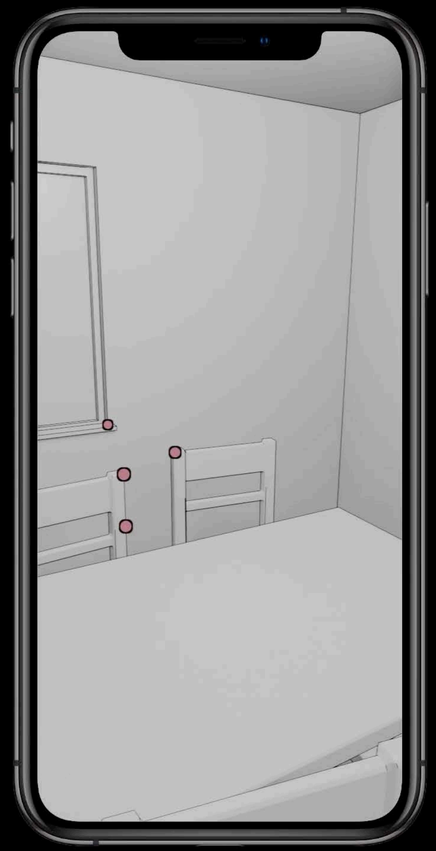


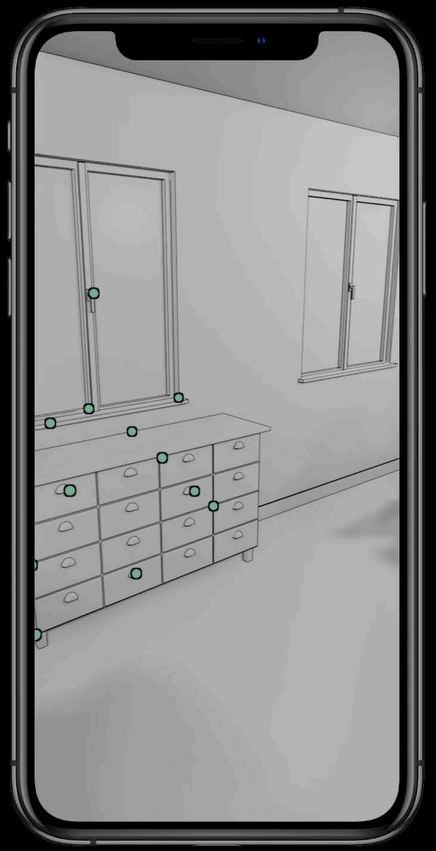


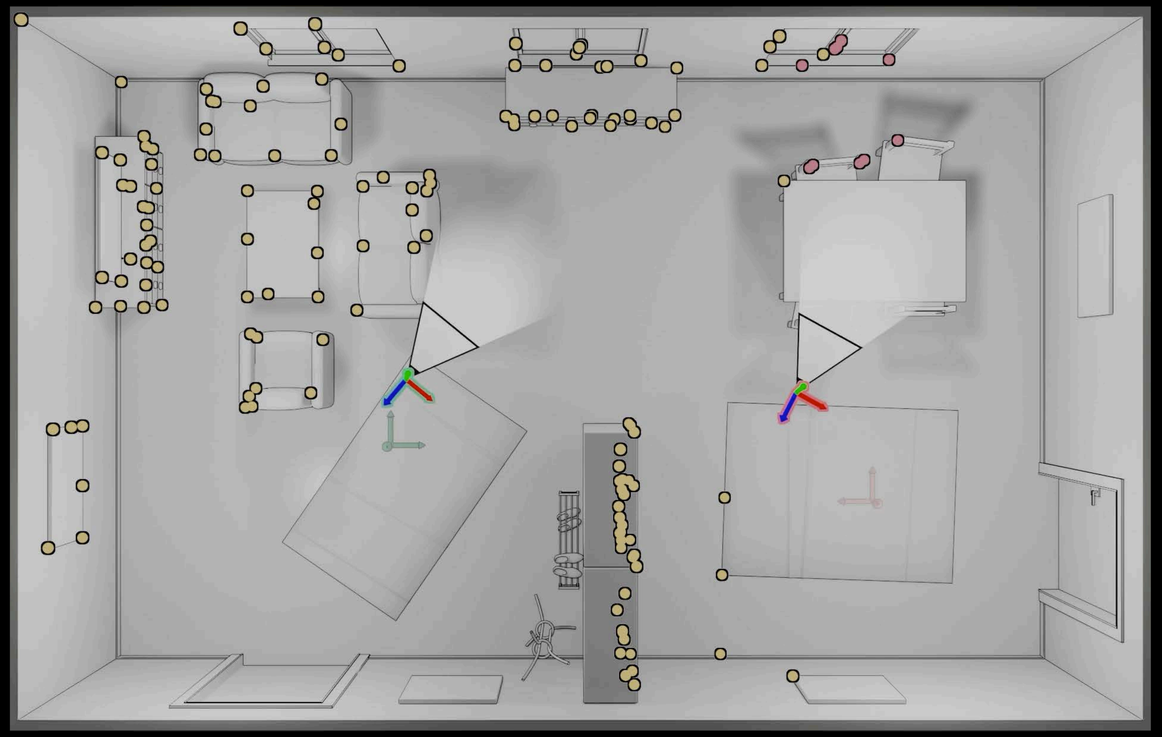


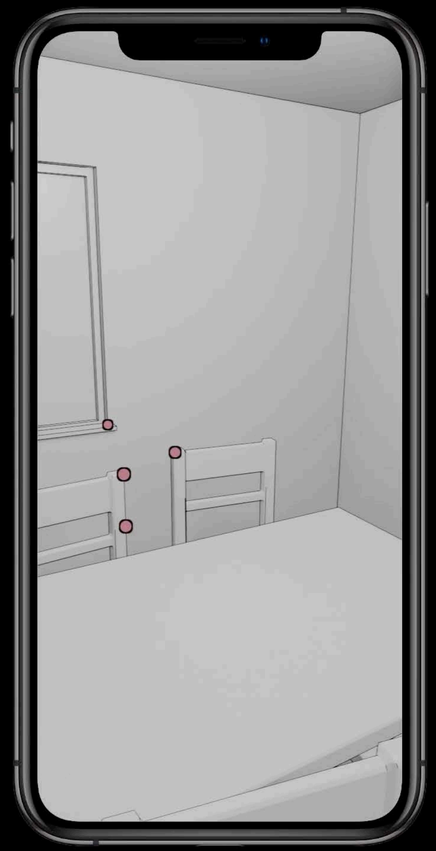


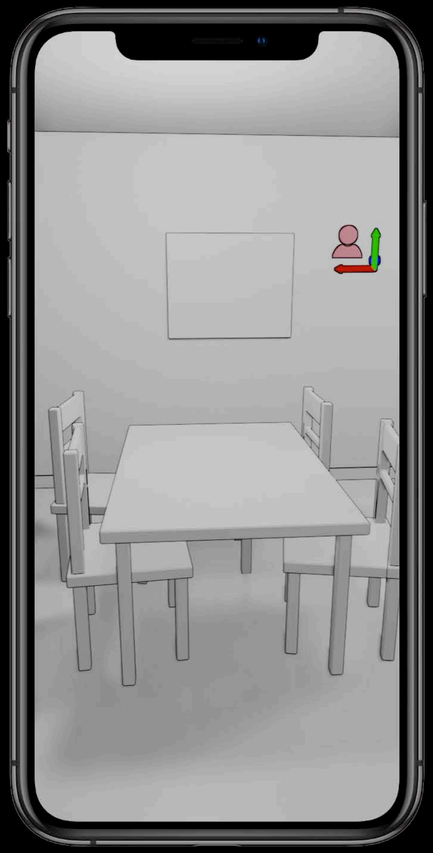


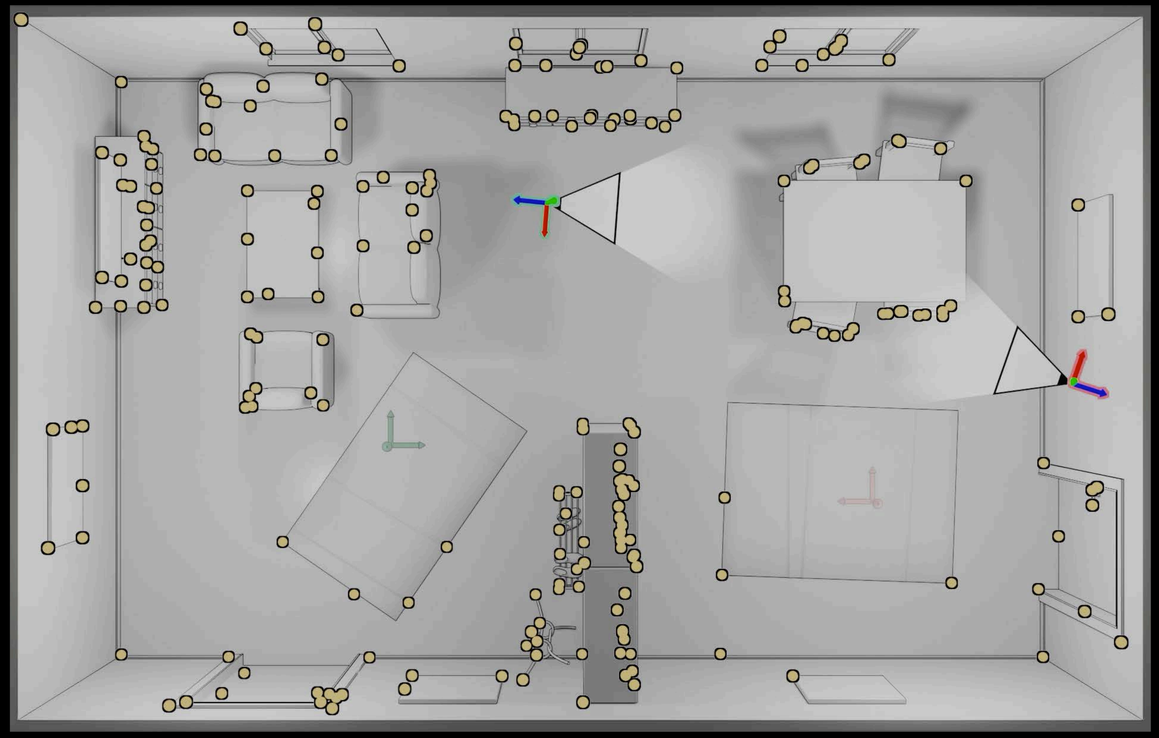


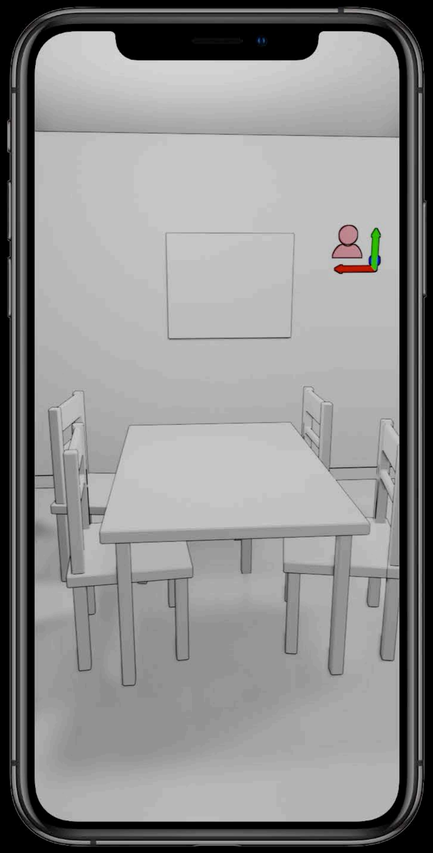


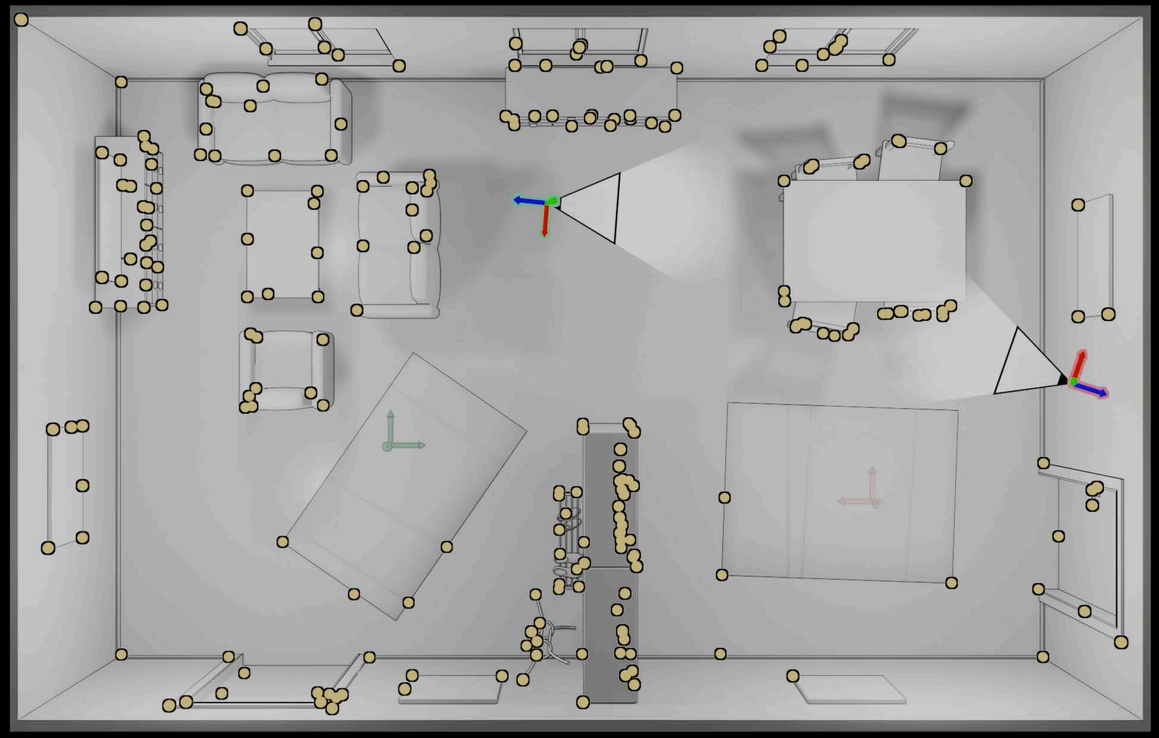












```
// Enable a collaborative session with RealityKit

// Set up networking
setupMultipeerConnectivity()

// Initialize synchronization service
arView.scene.synchronizationService =
    try? MultipeerConnectivityService(session: mcSession)


// Create configuration and enable the collaboration mode
let configuration = ARWorldTrackingConfiguration()
configuration.isCollaborationEnabled = true
arView.session.run(configuration)
```

```
// Enable a collaborative session with RealityKit

// Set up networking
setupMultipeerConnectivity()

// Initialize synchronization service
arView.scene.synchronizationService =
    try? MultipeerConnectivityService(session: mcSession)



// Create configuration and enable the collaboration mode
let configuration = ARWorldTrackingConfiguration()
configuration.isCollaborationEnabled = true
arView.session.run(configuration)
```

```
// Enable a collaborative session with RealityKit

// Set up networking
setupMultipeerConnectivity()

// Initialize synchronization service
arView.scene.synchronizationService =
    try? MultipeerConnectivityService(session: mcSession)


// Create configuration and enable the collaboration mode
let configuration = ARWorldTrackingConfiguration()
configuration.isCollaborationEnabled = true
arView.session.run(configuration)
```

```
// Enable a collaborative session with RealityKit

// Set up networking
setupMultipeerConnectivity()

// Initialize synchronization service
arView.scene.synchronizationService =
    try? MultipeerConnectivityService(session: mcSession)
```

```
// Create configuration and enable the collaboration mode
let configuration = ARWorldTrackingConfiguration()
configuration.isCollaborationEnabled = true
arView.session.run(configuration)
```

Collaborative Session

User A

ARWorldTrackingConfiguration

User B

ARWorldTrackingConfiguration

Collaborative Session

User A

ARWorldTrackingConfiguration

```
.isCollaborationEnabled = true
```

User B

ARWorldTrackingConfiguration

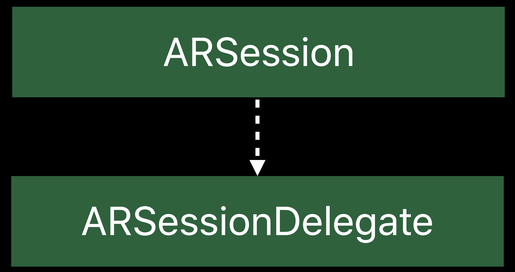
```
.isCollaborationEnabled = true
```

Collaborative Session

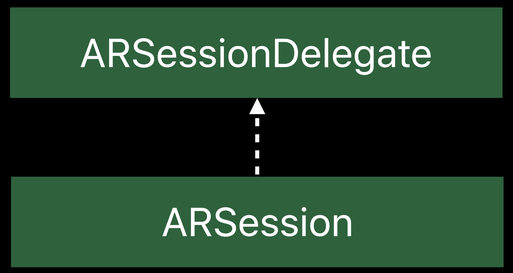
Collaborative Session

Collaborative Session

User A

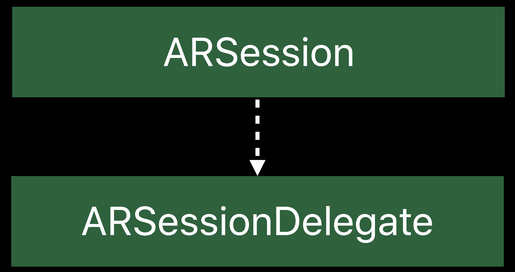


User B

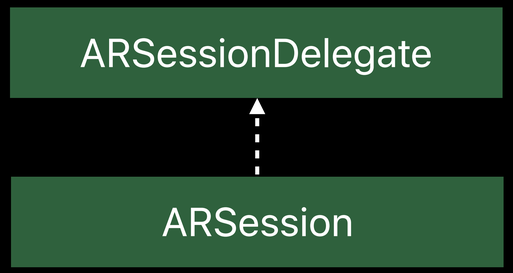


Collaborative Session

User A

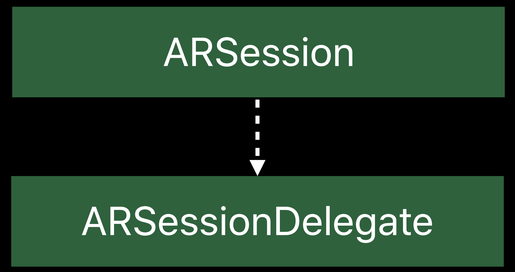


User B

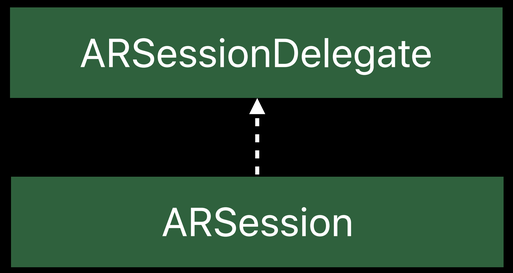


Collaborative Session

User A

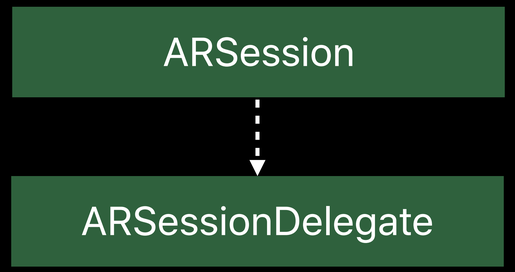


User B

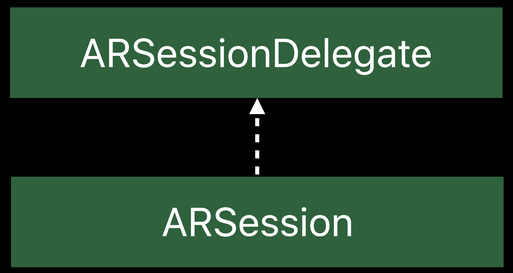


Collaborative Session

User A

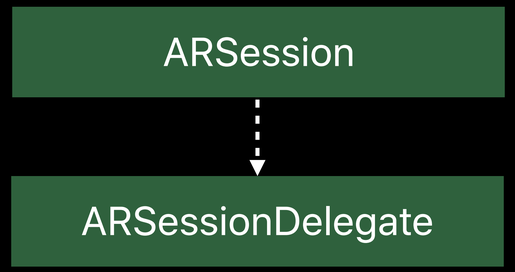


User B

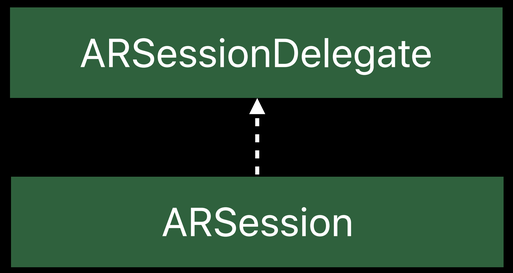


Collaborative Session

User A

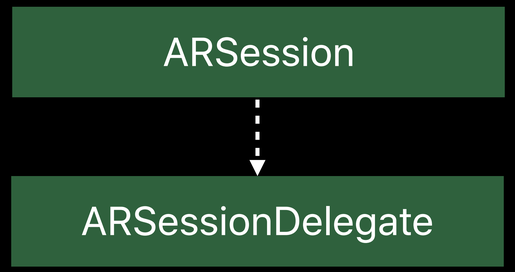


User B

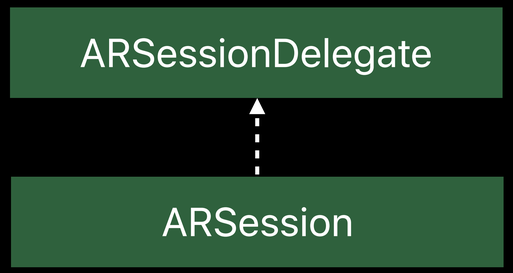


Collaborative Session

User A



User B



```
// Enable a collaborative session with ARKit

// Set up networking
setupMultipeerConnectivity()

// Create configuration and enable the collaboration mode
let configuration = ARWorldTrackingConfiguration()
configuration.isCollaborationEnabled = true
session.run(configuration)
```

```
// Enable a collaborative session with ARKit

// Set up networking
setupMultipeerConnectivity()

// Create configuration and enable the collaboration mode
let configuration = ARWorldTrackingConfiguration()
configuration.isCollaborationEnabled = true
session.run(configuration)
```

```
// Enable a collaborative session with ARKit

// Set up networking
setupMultipeerConnectivity()

// Create configuration and enable the collaboration mode
let configuration = ARWorldTrackingConfiguration()
configuration.isCollaborationEnabled = true
session.run(configuration)
```

```swift
// Session callback when some collaboration data is available
override func session(_ session: ARSession, didOutputCollaborationData data:
ARSession.CollaborationData) {
    // Send collaboration data to other participants
    mcSession.send(data, toPeers: participantIds, with: .reliable)
}


// Multipeer Connectivity session delegate callback upon receiving data from peers
func session(_ session: MCSession, didReceive data: Data, fromPeer peerID: MCPeerID) {
    // Update the session with collaboration data received from another participant
    let collaborationData = ARSession.CollaborationData(data)
    session.update(from: collaborationData)
}
```

```swift
// Session callback when some collaboration data is available
override func session(_ session: ARSession, didOutputCollaborationData data:
ARSession.CollaborationData) {
    // Send collaboration data to other participants
    mcSession.send(data, toPeers: participantIds, with: .reliable)
}
```

```swift
// Multipeer Connectivity session delegate callback upon receiving data from peers
func session(_ session: MCSession, didReceive data: Data, fromPeer peerID: MCPeerID) {
    // Update the session with collaboration data received from another participant
    let collaborationData = ARSession.CollaborationData(data)
    session.update(from: collaborationData)
}
```

```swift
// Session callback when some collaboration data is available
override func session(_ session: ARSession, didOutputCollaborationData data:
ARSession.CollaborationData) {
    // Send collaboration data to other participants
    mcSession.send(data, toPeers: participantIds, with: .reliable)
}
```

```swift
// Multipeer Connectivity session delegate callback upon receiving data from peers
func session(_ session: MCSession, didReceive data: Data, fromPeer peerID: MCPeerID) {
    // Update the session with collaboration data received from another participant
    let collaborationData = ARSession.CollaborationData(data)
    session.update(from: collaborationData)
}
```

Collaborative Session Data

Automatic exchange of user created ARAnchor

Session identifier on each anchor

ARParticipantAnchor representing participant position

AR Coaching UI

Coaching in AR Apps

On-boarding

Throughout the experience

Human interface guideline recommendations

AR Coaching View

NEW

Built-in UI overlay for AR applications

Guides users to good tracking experience

Consistent design throughout applications

Automatic activation and deactivation

Adjustable coaching goals

AR Coaching View

Overlays

Move iPhone to start

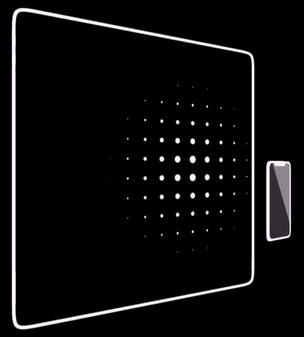
AR Coaching View

Overlays

Move iPhone to start

AR Coaching View

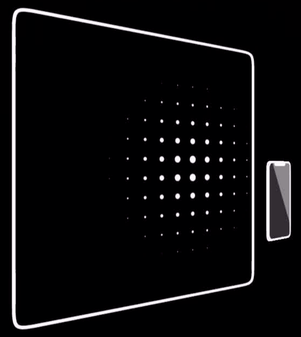
Overlays



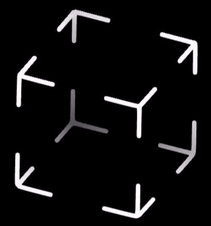
Move iPhone to start

AR Coaching View

Overlays



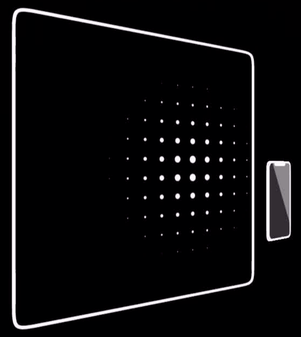
Move iPhone to start



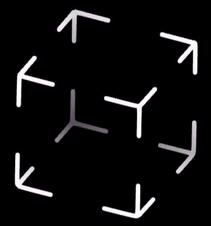
Continue to move iPhone

AR Coaching View

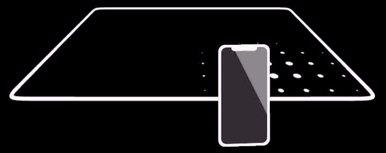
Overlays



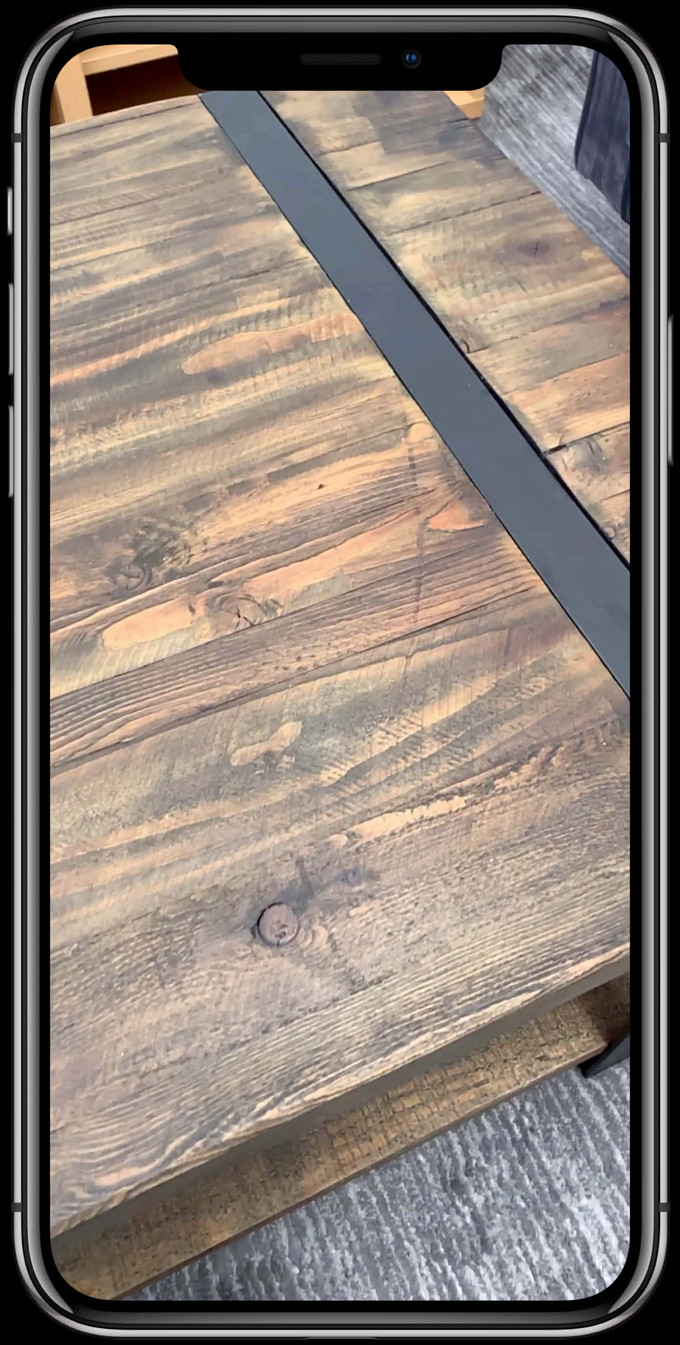
Move iPhone to start

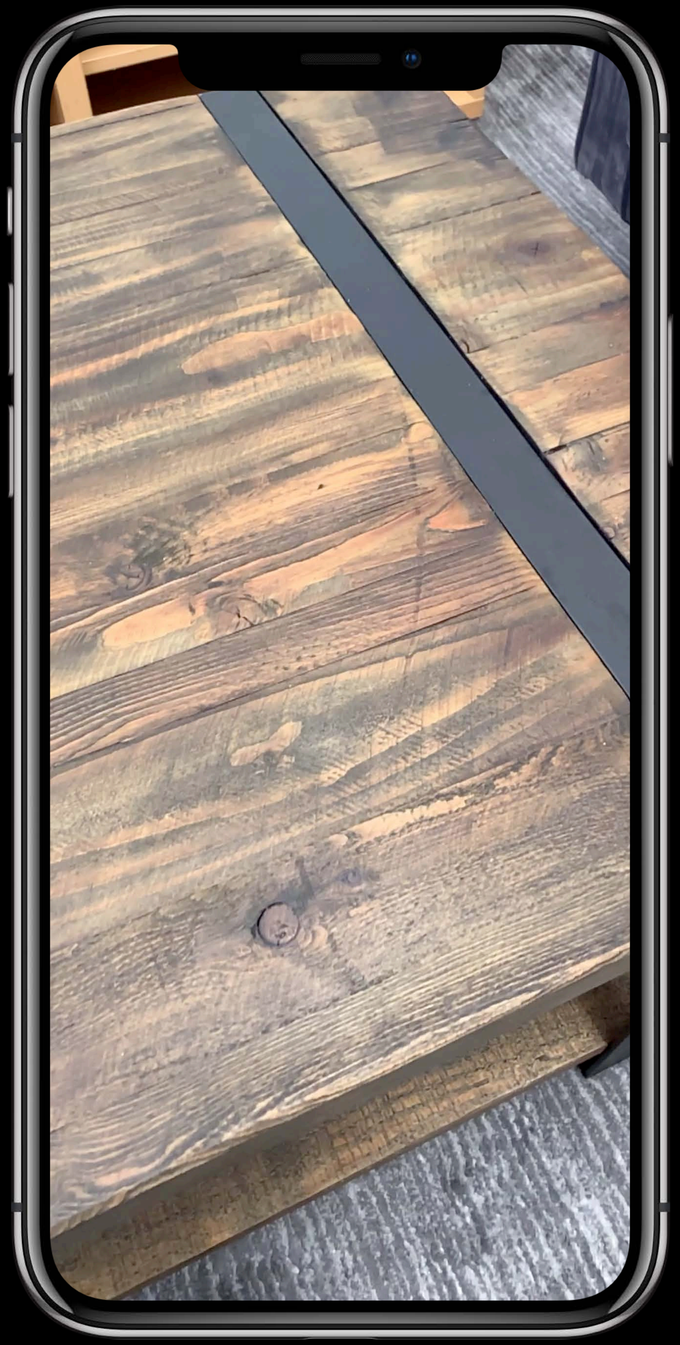


Continue to move iPhone



Return to the previous area to resume





AR Coaching View

Setup

Add as a child of any UI view

Connect to the ARSession

Optionally set a delegate

Specify coaching goals in source code

AR Coaching View

Setup

Add as a child of any UI view

Connect to the ARSession

Optionally set a delegate

Specify coaching goals in source code

```
coachingOverlay.goal = .horizontalPlane
```

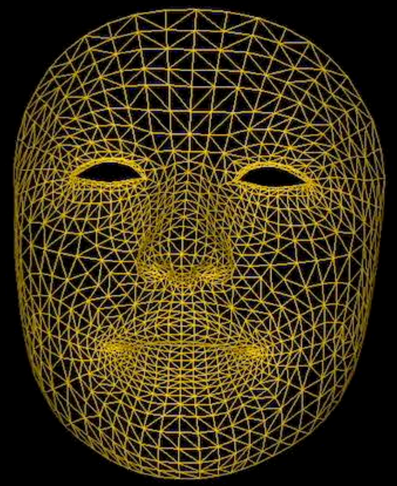
AR Coaching View

ARCoachingViewDelegate

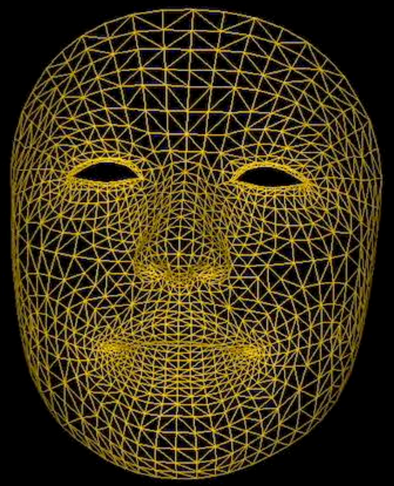
React to activation and deactivation

React to relocalization abort request

```
protocol ARCoachingOverlayViewDelegate {
    func coachingOverlayViewWillActivate(ARCoachingOverlayView)
    func coachingOverlayViewDidDeactivate(ARCoachingOverlayView)
    func coachingOverlayViewDidRequestSessionReset(ARCoachingOverlayView)
}
```



Face Tracking



Multiple-Face Tracking

NEW

Up to 3 faces tracked simultaneously

Persistent face anchor IDs within ARSession

```
open class ARFaceTrackingConfiguration : ARConfiguration {

    open class var supportedNumberOfTrackedFaces: Int { get }

    open var maximumNumberOfTrackedFaces: Int

}
```

ARPositionalTrackingConfiguration

NEW

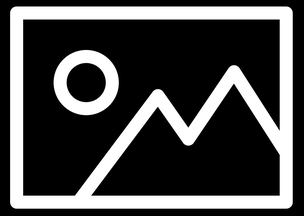
Intended for tracking only use cases

Low power consumption

Uses lower-resolution camera image

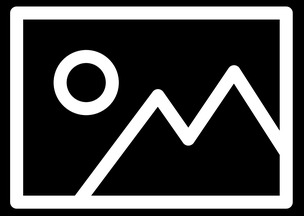
Scene Understanding Improvements

Scene Understanding Improvements

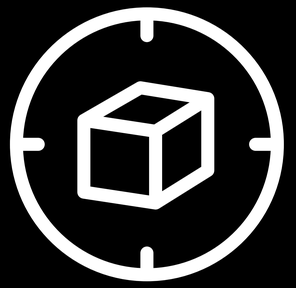


Up to 100 images
Automatic scale estimation
Quality feedback at runtime

Scene Understanding Improvements

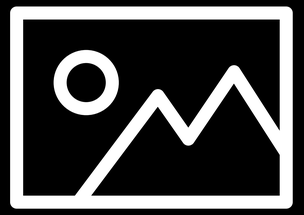


Up to 100 images
Automatic scale estimation
Quality feedback at runtime

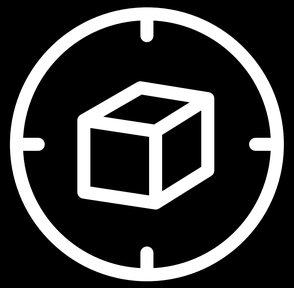


ML-enhanced object detection
Faster recognition
More robust

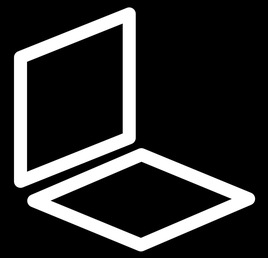
Scene Understanding Improvements



Up to 100 images
Automatic scale estimation
Quality feedback at runtime



ML-enhanced object detection
Faster recognition
More robust



ML-supported plane estimation
More accurate boundaries
Faster extension

Plane Classification

```
class ARPlaneAnchor : ARAnchor {
  var classification: ARPlaneAnchor.Classification
}
```

```
enum ARPlaneAnchor.Classification {
   case wall
   case floor
   case ceiling
   case table
   case seat
}
```

Plane Classification

NEW

```
class ARPlaneAnchor : ARAnchor {
  var classification: ARPlaneAnchor.Classification
}
```

```
enum ARPlaneAnchor.Classification {
   case wall
   case floor
   case ceiling
   case table
   case seat
   case door
   case window
}
```

Plane Classification

NEW

```
class ARPlaneAnchor : ARAnchor {
  var classification: ARPlaneAnchor.Classification
}
```

```
enum ARPlaneAnchor.Classification {
   case wall
   case floor
   case ceiling
   case table
   case seat
   case door
   case window
}
```

Raycasting

Ideal for object placement

Supports any surface alignment

Raycasts tracked over time

Benefits from plane estimation updates

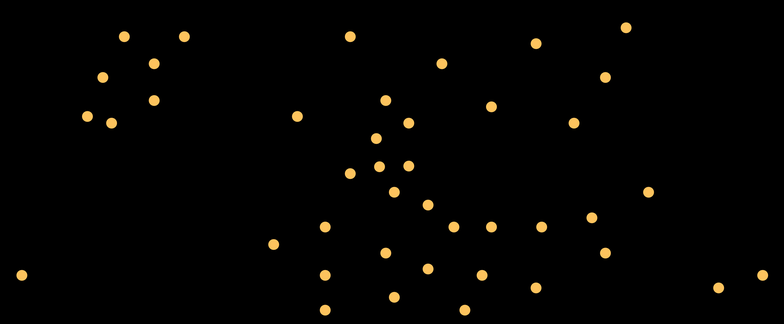
Raycasting

Ideal for object placement

Supports any surface alignment

Raycasts tracked over time

Benefits from plane estimation updates



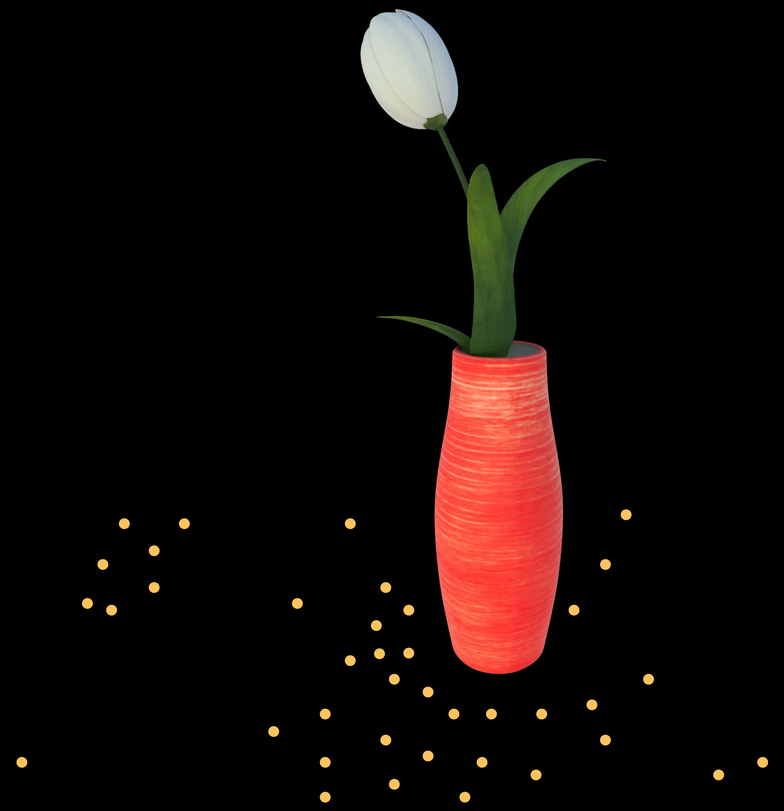
Raycasting

Ideal for object placement

Supports any surface alignment

Raycasts tracked over time

Benefits from plane estimation updates



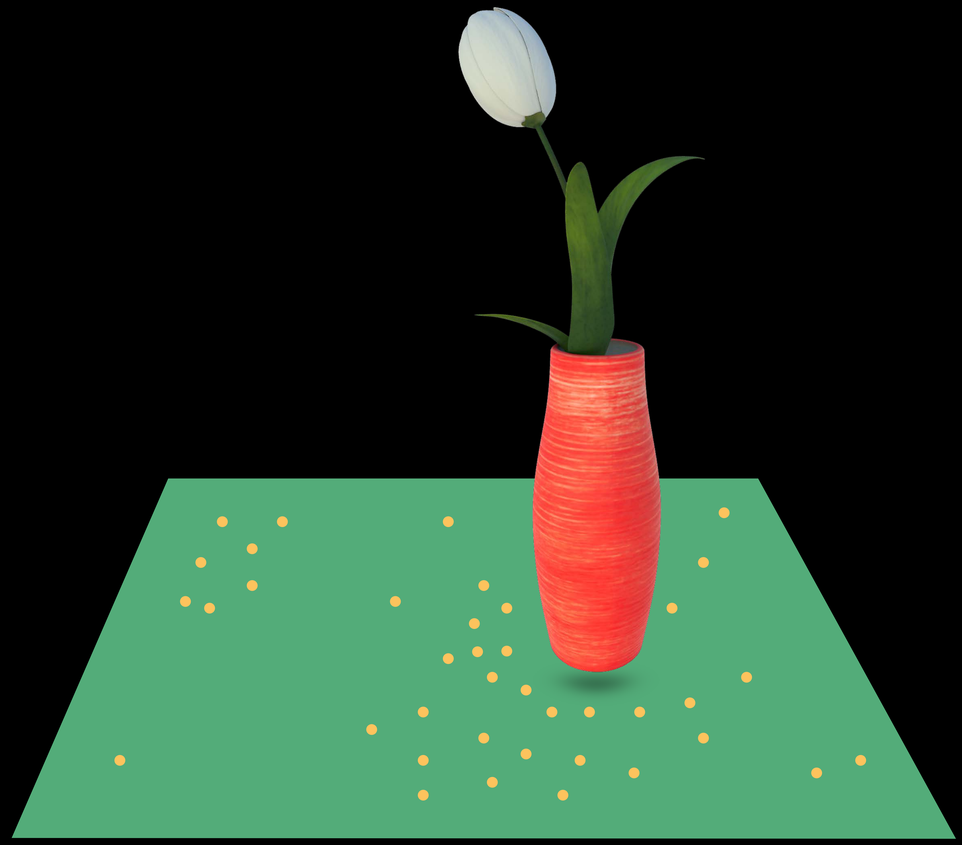
Raycasting

Ideal for object placement

Supports any surface alignment

Raycasts tracked over time

Benefits from plane estimation updates



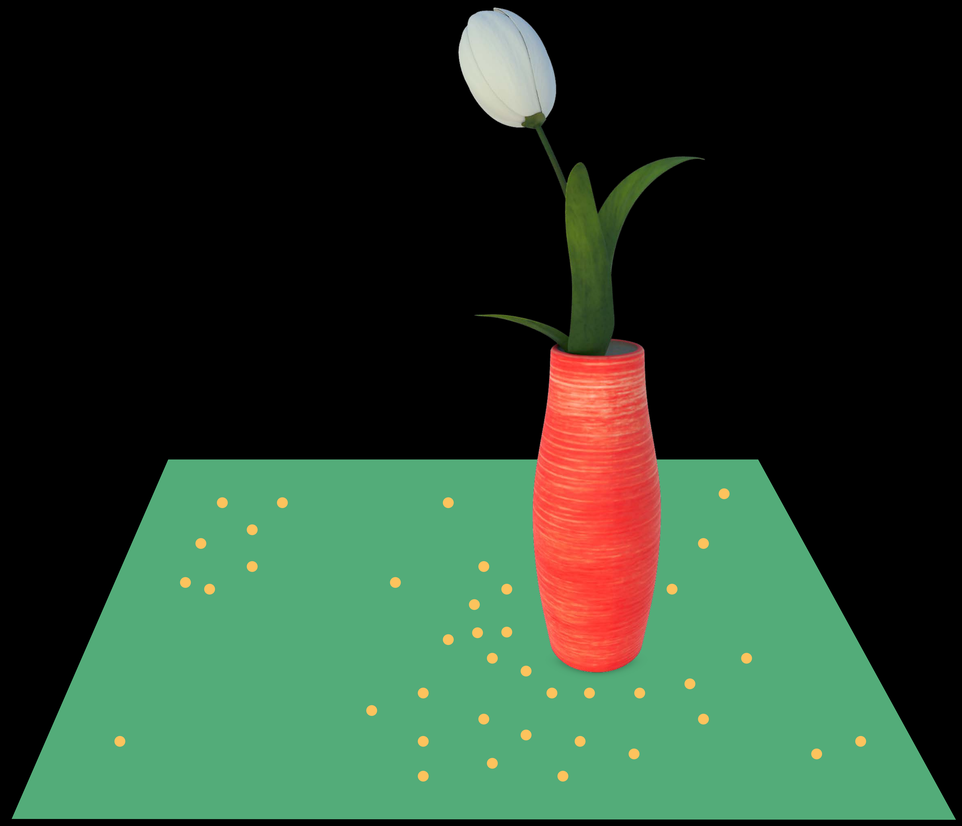
Raycasting

Ideal for object placement

Supports any surface alignment

Raycasts tracked over time

Benefits from plane estimation updates



```
// Create a raycast query
let query = arView.raycastQuery(from: screenCenter,
                                allowing: .estimatedPlane,
                                alignment: .any)


// Perform the raycast
let raycast = session.trackedRaycast(query) { results in
    // Refine object position with raycast update
    if let result = results.first {
        object.transform = result.transform
    }
}

// Stop tracking the raycast
raycast.stop()
```

```
// Create a raycast query
let query = arView.raycastQuery(from: screenCenter,
                                allowing: .estimatedPlane,
                                alignment: .any)


// Perform the raycast
let raycast = session.trackedRaycast(query) { results in
    // Refine object position with raycast update
    if let result = results.first {
        object.transform = result.transform
    }
}

// Stop tracking the raycast
raycast.stop()
```

```
// Create a raycast query
let query = arView.raycastQuery(from: screenCenter,
                                allowing: .estimatedPlane,
                                alignment: .any)

// Perform the raycast
let raycast = session.trackedRaycast(query) { results in
    // Refine object position with raycast update
    if let result = results.first {
        object.transform = result.transform
    }
}

// Stop tracking the raycast
raycast.stop()
```

```
// Create a raycast query
let query = arView.raycastQuery(from: screenCenter,
                                allowing: .estimatedPlane,
                                alignment: .any)


// Perform the raycast
let raycast = session.trackedRaycast(query) { results in
    // Refine object position with raycast update
    if let result = results.first {
        object.transform = result.transform
    }
}
```

```
// Stop tracking the raycast
raycast.stop()
```

Visual Coherence Enhancements

Activated and deactivated automatically based on device capabilities

Can be explicitly disabled in render options

```
class ARView {
    var renderOptions: ARView.RenderOptions { get set }
}
```

Depth of Field

Motion Blur

Visual Coherence Enhancements

NEW

HDR Environment textures

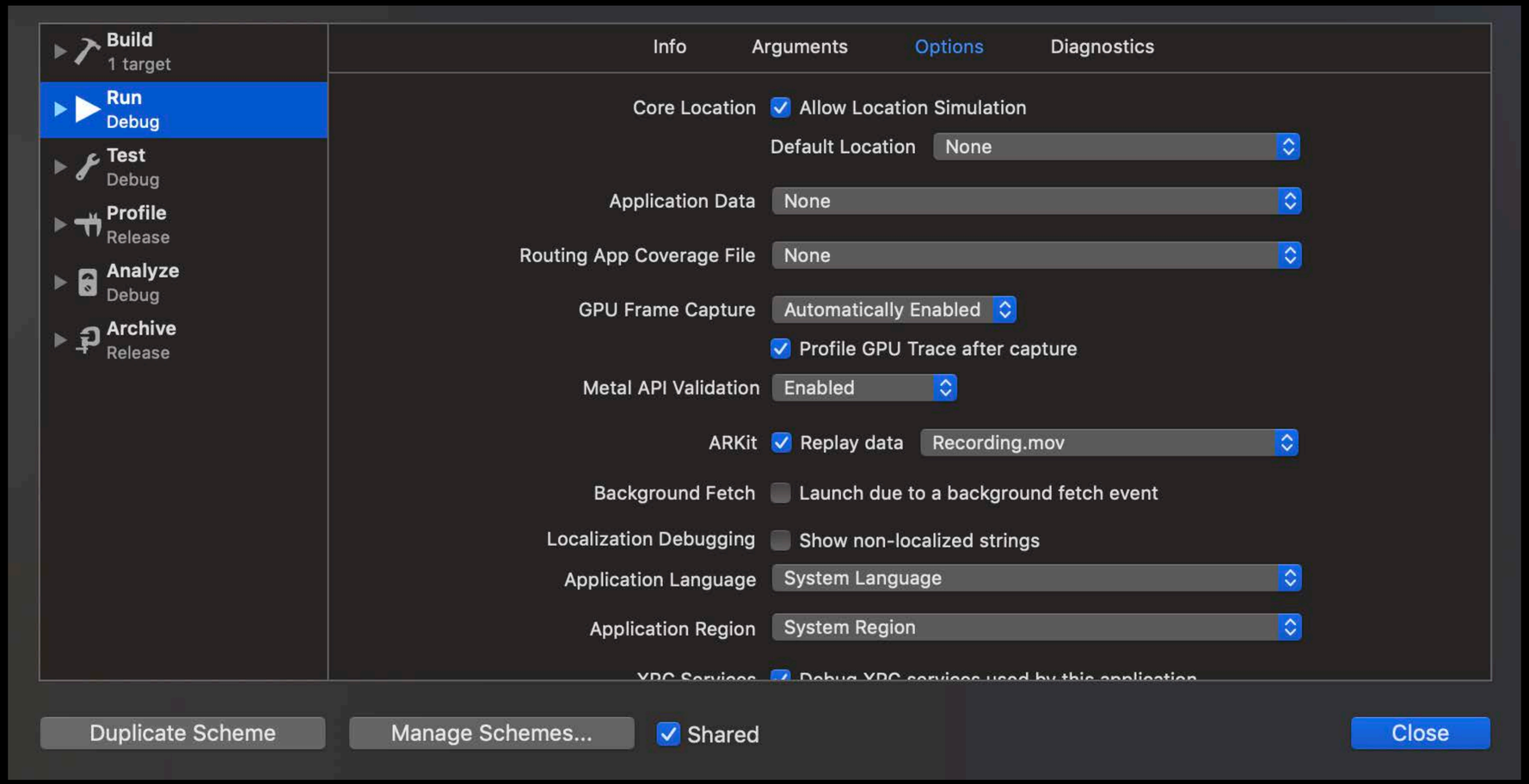
Camera grain

```
class ARWorldTrackingConfiguration {
    var wantsHDREnvironmentTextures: Bool { get set }
}

class ARFrame {
    var cameraGrainIntensity: Float { get }
    var cameraGrainTexture: MTLTexture? { get }
}
```

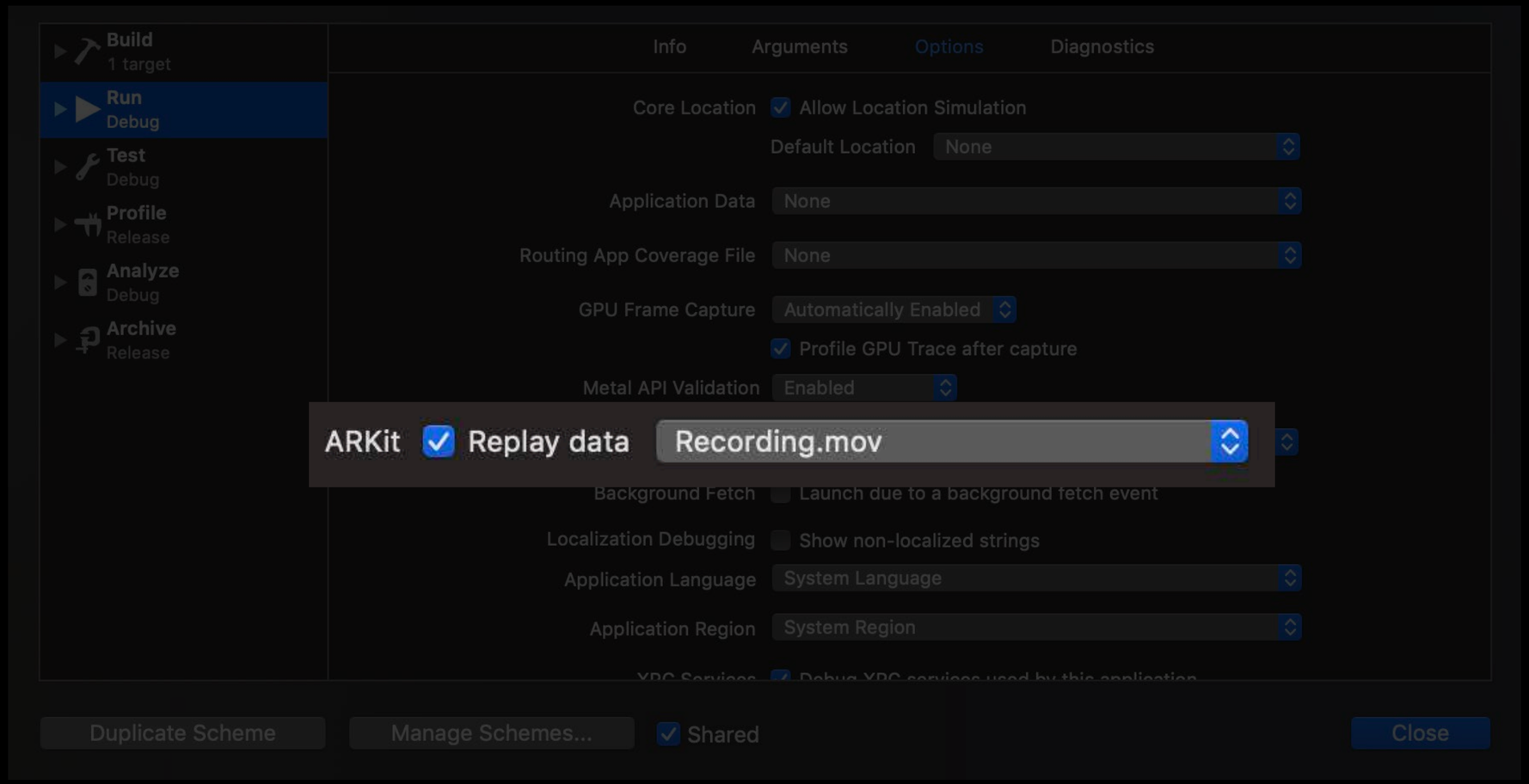
Record and Replay

Record and Replay

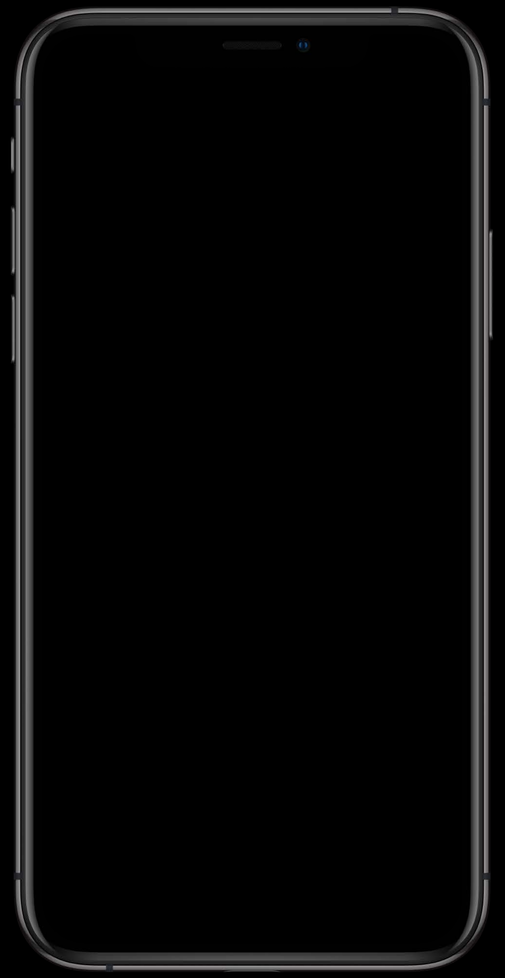
Record and Replay



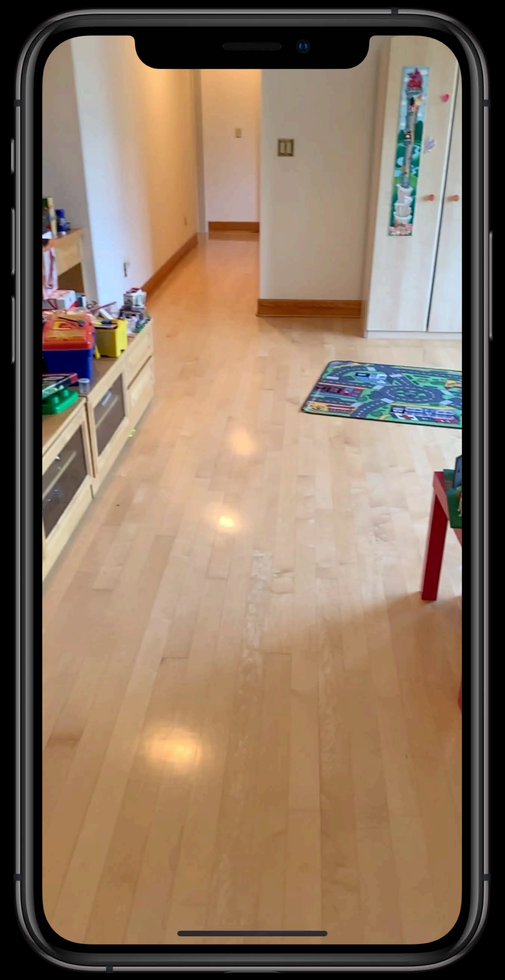
Record and Replay



Record and Replay



Record and Replay



Introducing ARKit 3

Visual Coherence

HDR Environment Textures

Faster Reference Image Loading

Motion Capture

Positional Tracking

Face Tracking Enhancements

Detect up to 100 Images

Simultaneous Front and Back Camera

Auto-detect Image Size

AR Coaching UI

Record and Replay of Sequences

People Occlusion

Introducing ARKit 3

More Robust 3D Object Detection

RealityKit Integration

Raycasting

ML Based Plane Detection

Multiple-face Tracking

Collaborative Session

New Plane Classes

AR QuickLook Additions

More Information

developer.apple.com/wwdc19/604

ARKit Lab	Wednesday, 12:00
Bringing People into AR	Wednesday, 4:00
Building Collaborative AR Experiences	Thursday, 2:00


WWDC19