Introducing ARKit

Augmented Reality for iOS

Session 602

Mike Buerli, ARKit Engineer
Stefan Misslinger, ARKit Engineer

Augmented Reality


Reset demo
SOFAS
ARMCHAIRS
TABLES

Augmented Reality

Triangulation

Computer Vision

Scene Understanding

Surface Estimation

SLAM

Feature Detection

Sensor Fusion

Camera Intrinsics

Camera Calibration

Light Estimation

Bundle Adjustment

Visual-inertial Navigation

Feature Matching

Nonlinear Optimization

Optimal Correction

NEW

ARKit

NEW

Mobile AR platform

High-level API

iOS (A9 and up)

Tracking

World tracking

Visual inertial odometry

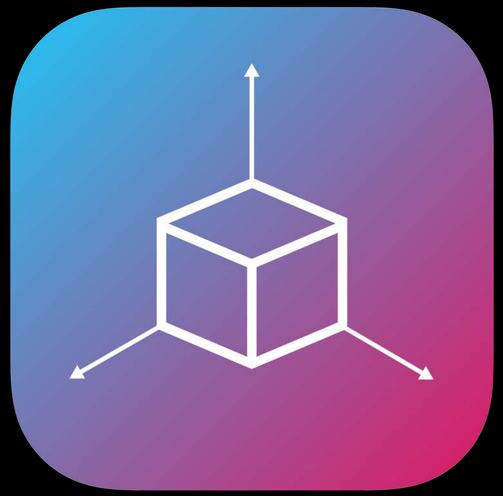
No external setup

Scene Understanding

Plane detection

Hit-testing

Light estimation



Rendering

Easy integration

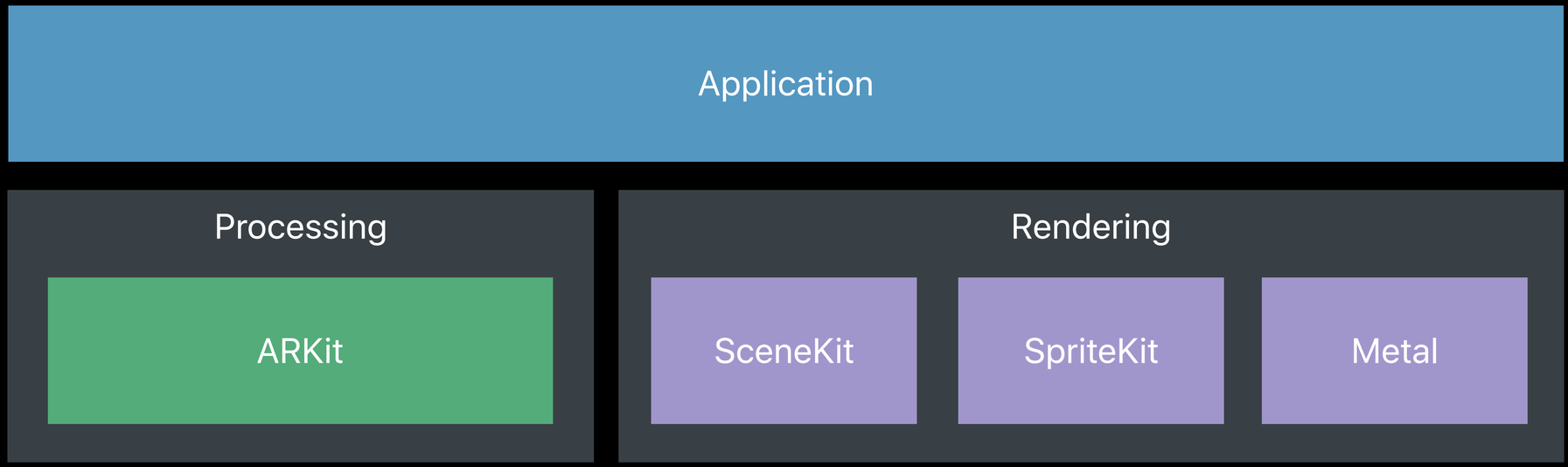
AR views

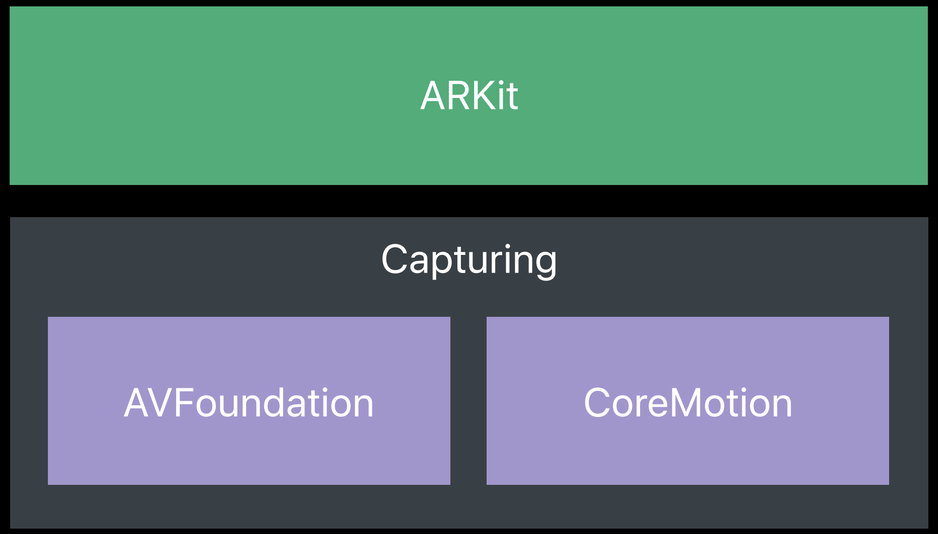
Custom rendering

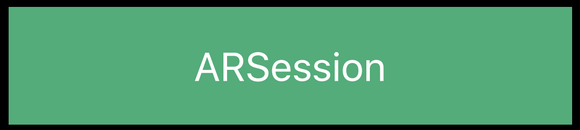

unity


UNREAL
ENGINE

Getting Started

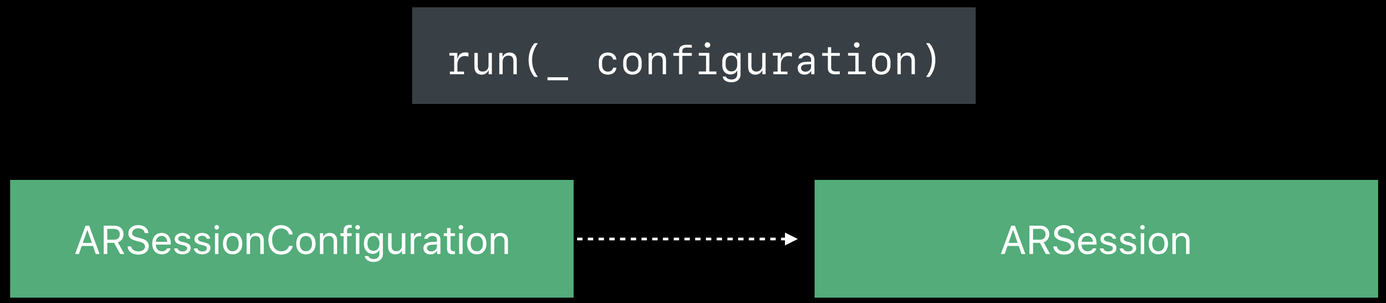

Application
Processing
ARKit
Rendering
SceneKit
SpriteKit
Metal

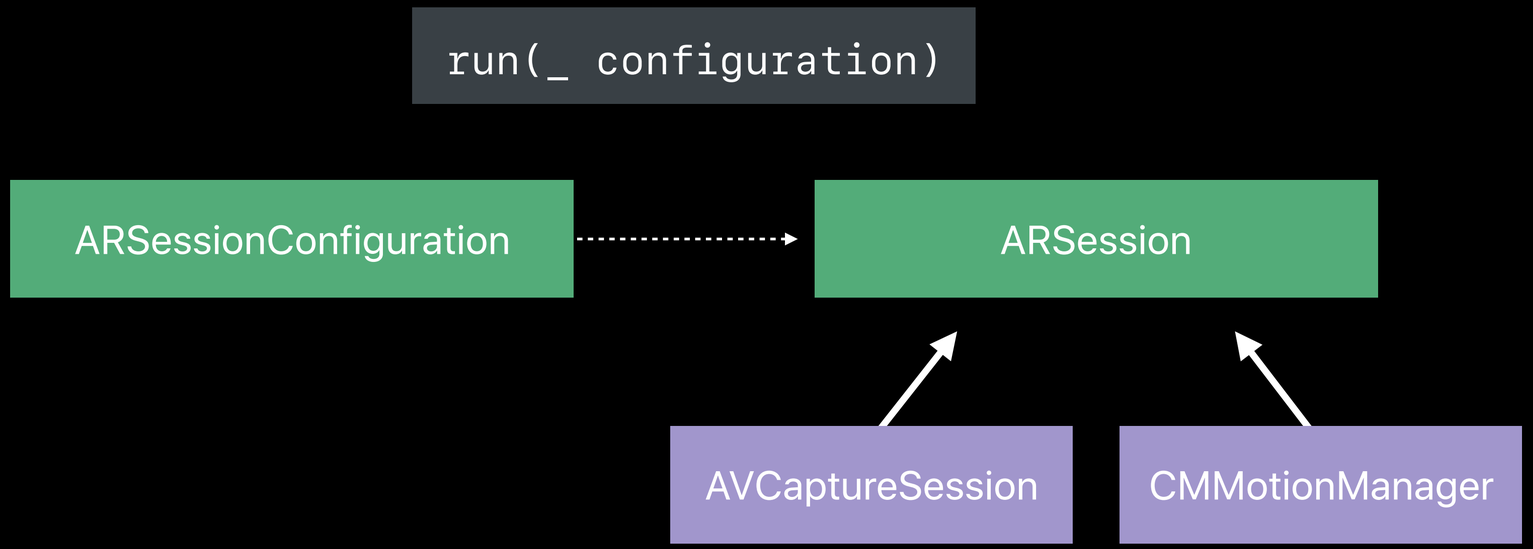

ARKit
Capturing
AVFoundation
CoreMotion

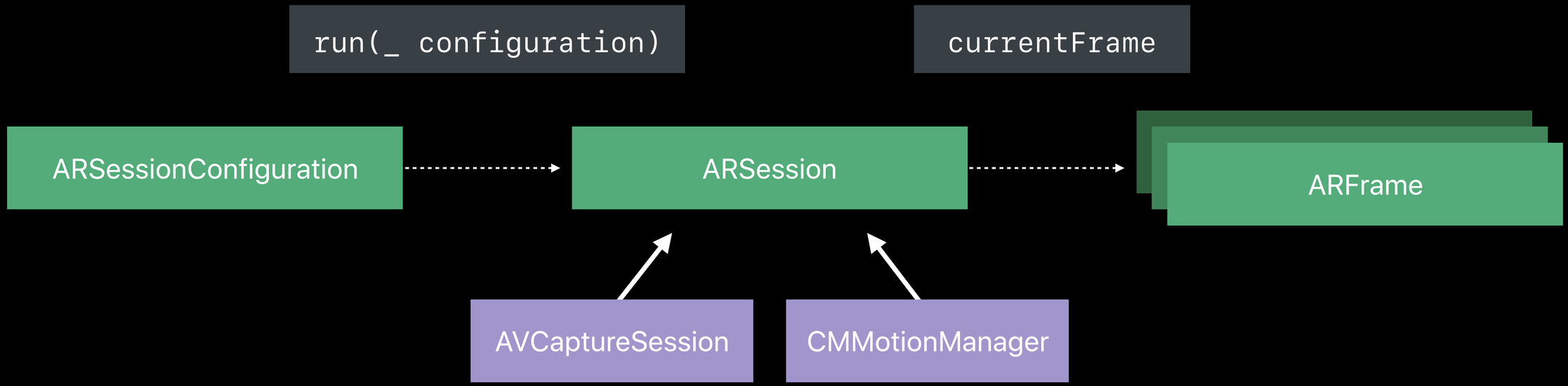

ARSession

ARSessionConfiguration

ARSession


run(_ configuration)
ARSessionConfiguration
ARSession


run(_ configuration)
ARSessionConfiguration
ARSession
AVCaptureSession
CMMotionManager


run(_ configuration)
currentFrame
ARSessionConfiguration
ARSession
ARFrame
AVCaptureSession
CMMotionManager

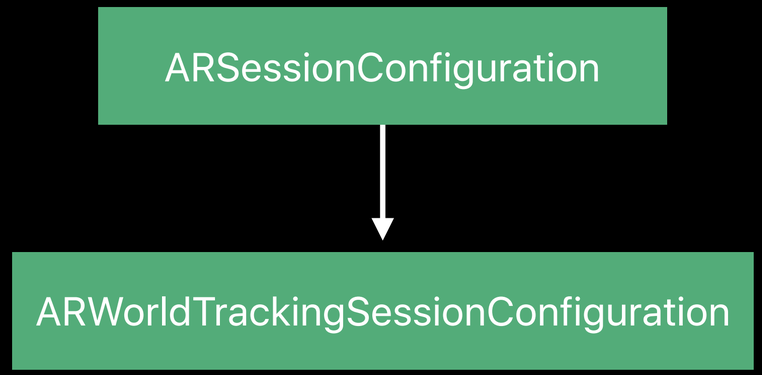
ARSessionConfiguration

ARSessionConfiguration

Configuration Classes

ARSessionConfiguration

Configuration Classes



ARSessionConfiguration

Configuration Classes

Enable/Disable Features

ARSessionConfiguration

Configuration Classes

Enable/Disable Features

Availability

ARSessionConfiguration

Configuration Classes

Enable/Disable Features

Availability

```
if ARWorldTrackingSessionConfiguration.isSupported {
    configuration = ARWorldTrackingSessionConfiguration()
}
else {
    configuration = ARSessionConfiguration()
}
```

ARSession

ARSession

Manage AR processing

ARSession

Manage AR processing

```
// Run your session
session.run(configuration)
```

ARSession

Manage AR processing

```
// Run your session
session.run(configuration)

// Pause your session
session.pause()
```

ARSession

Manage AR processing

```
// Run your session
session.run(configuration)

// Pause your session
session.pause()

// Resume your session
session.run(session.configuration)
```

ARSession

Manage AR processing

```
// Run your session
session.run(configuration)

// Pause your session
session.pause()

// Resume your session
session.run(session.configuration)

// Change your configuration
session.run(otherConfiguration)
```

ARSession

Manage AR processing

Reset tracking

ARSession

Manage AR processing

Reset tracking

```
// Reset tracking
session.run(configuration, options: .resetTracking)
```

ARSession

Manage AR processing

Reset tracking

Session updates

ARSession

Manage AR processing

Reset tracking

Session updates

```
// Access the latest frame
func session(_: ARSession, didUpdate: ARFrame)
```

ARSession

Manage AR processing

Reset tracking

Session updates

```
// Access the latest frame
func session(_: ARSession, didUpdate: ARFrame)

// Handle session errors
func session(_: ARSession, didFailWithError: Error)
```

ARSession

Manage AR processing

Reset tracking

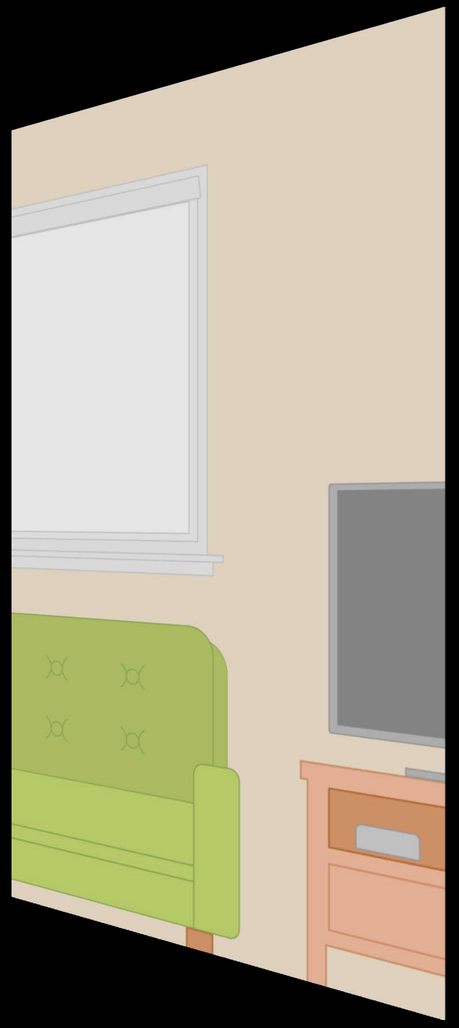
Session updates

Current frame

ARFrame

ARFrame

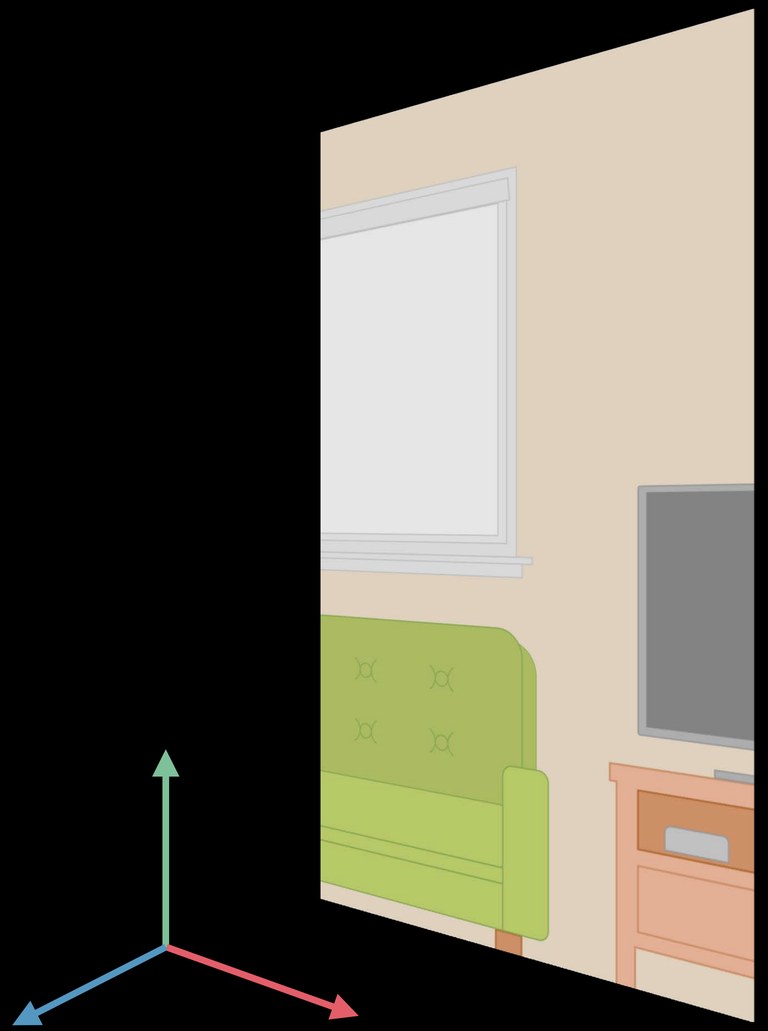
Captured image



ARFrame

Captured image

Tracking information

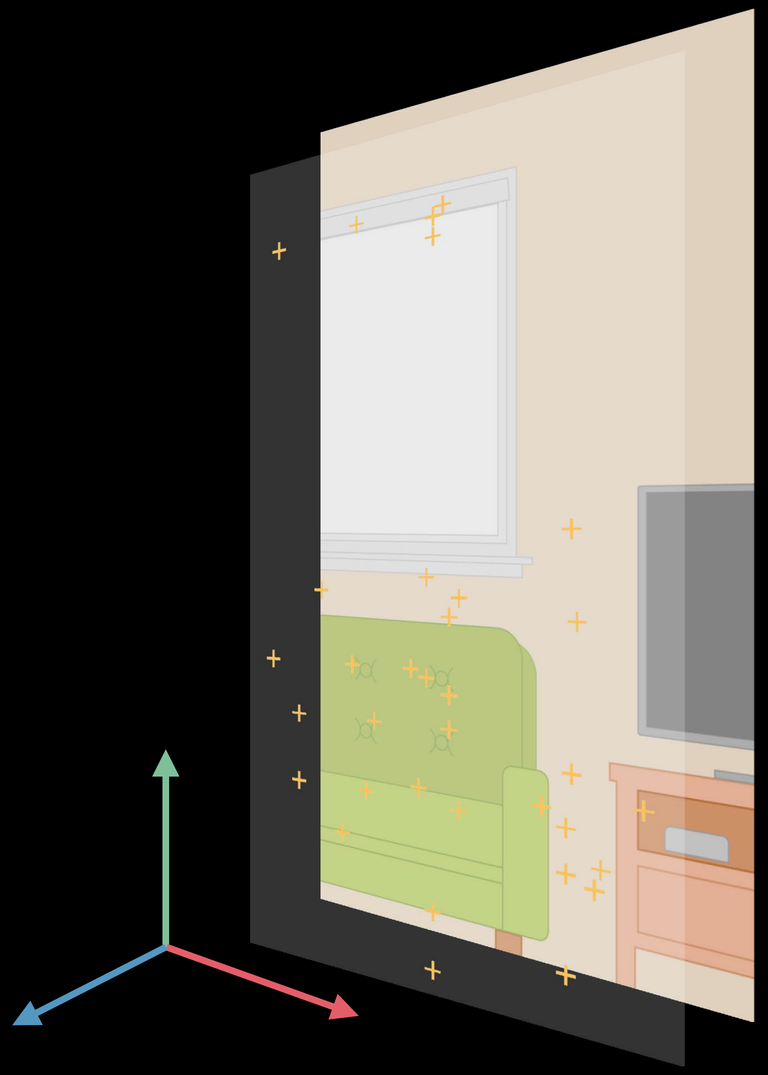


ARFrame

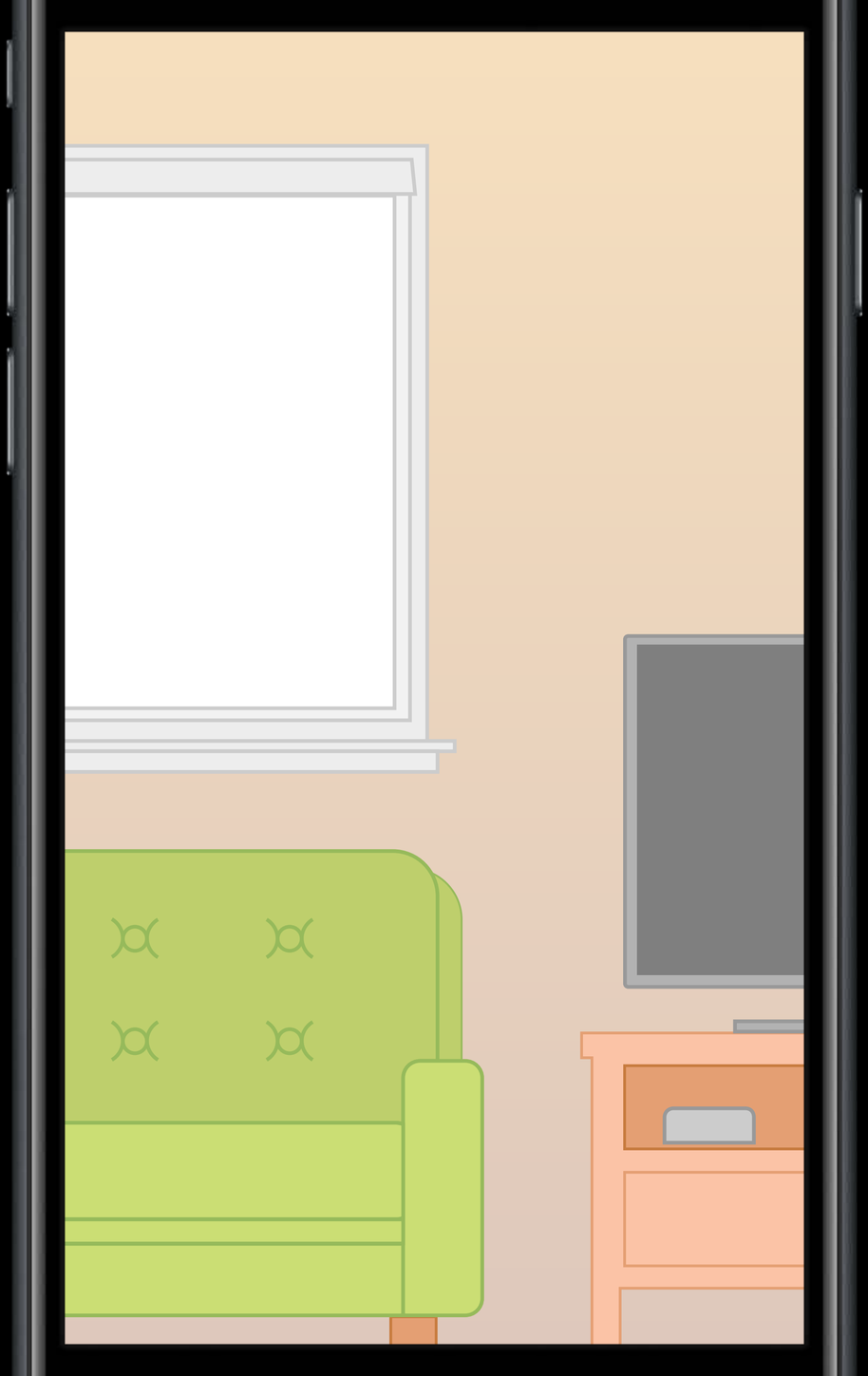
Captured image

Tracking information

Scene information



ARAnchor



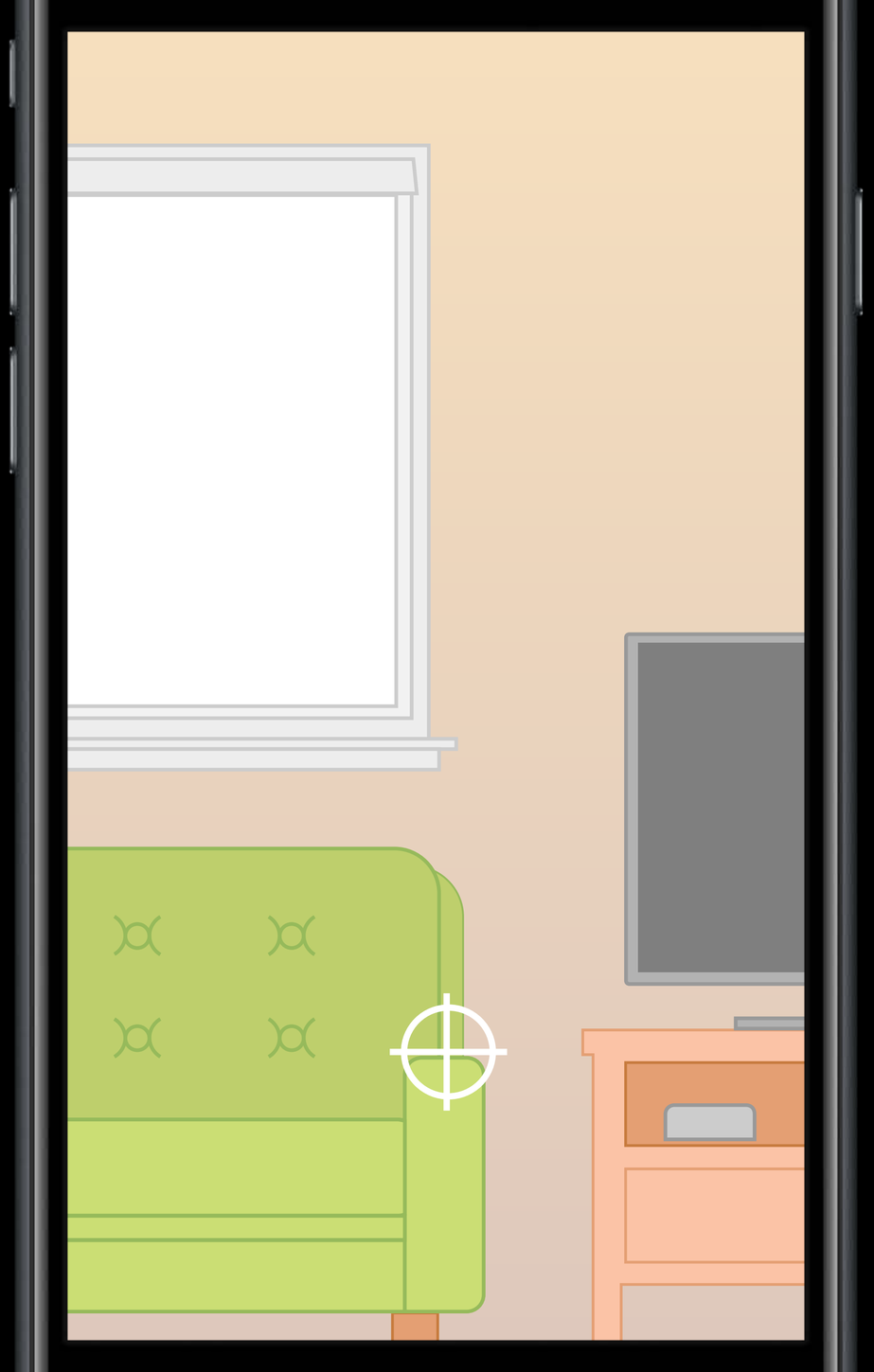
ARAnchor

Real-world position and orientation

ARAnchor

Real-world position and orientation

ARSession add/remove

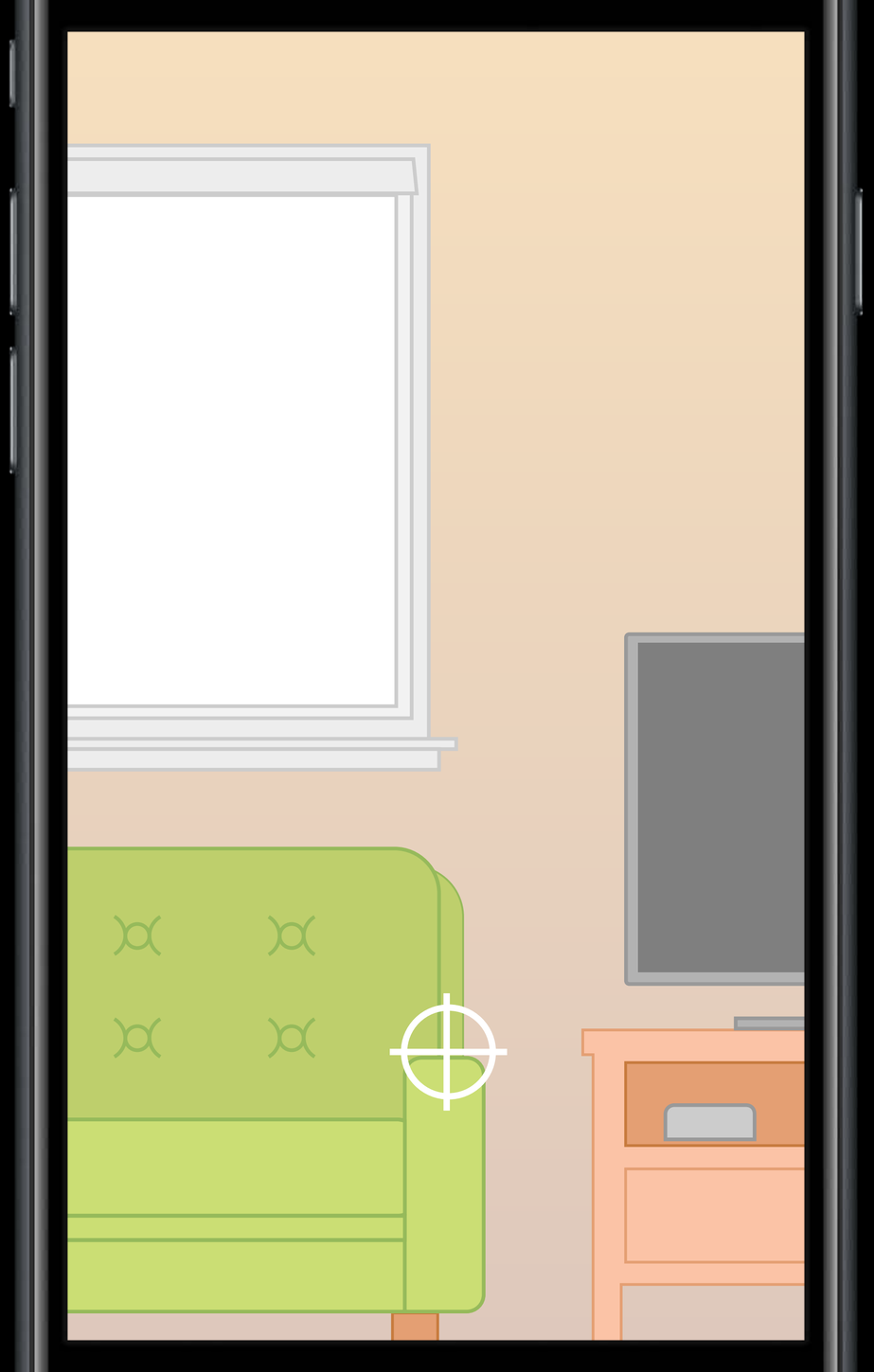


ARAnchor

Real-world position and orientation

ARSession add/remove

ARFrame anchors



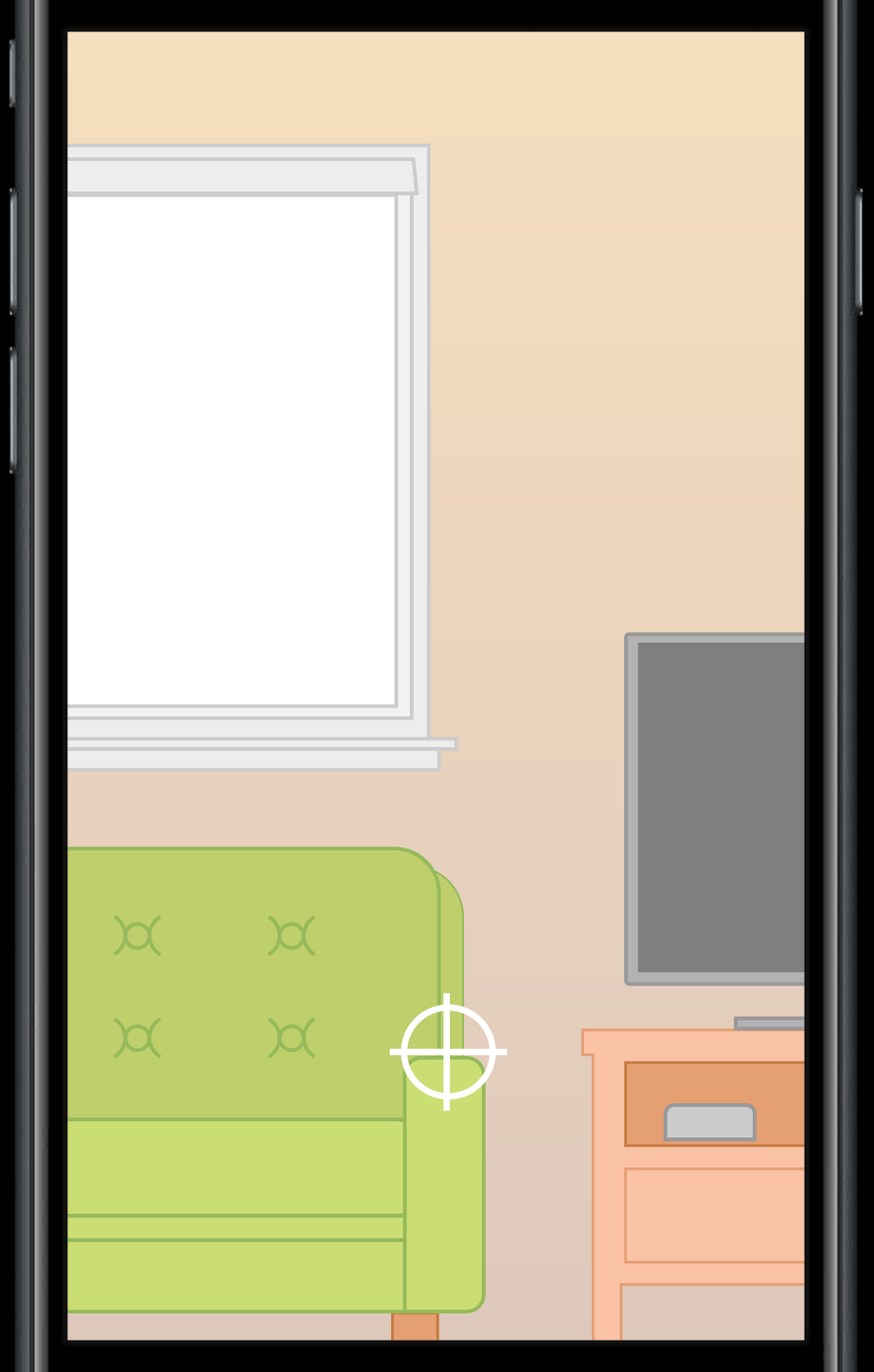
ARAnchor

Real-world position and orientation

ARSession add/remove

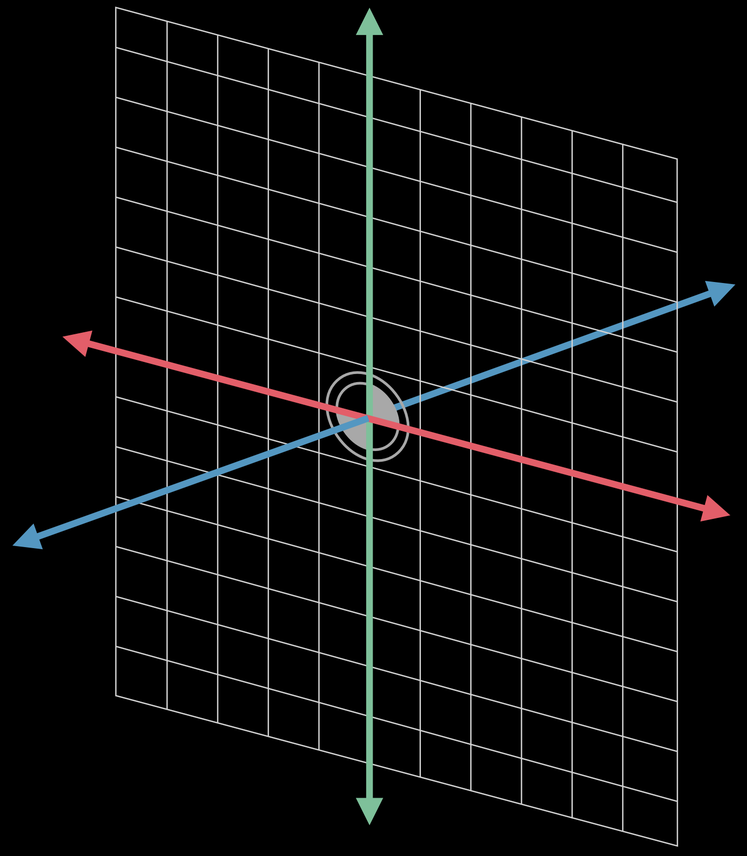
ARFrame anchors

ARSessionDelegate add/update/remove



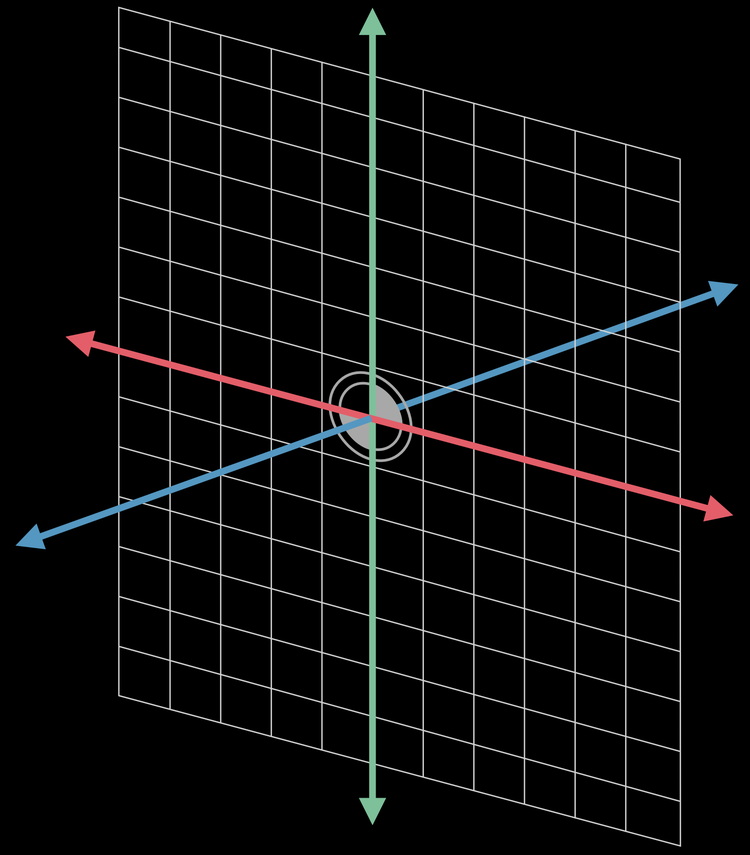
Tracking

World Tracking



World Tracking

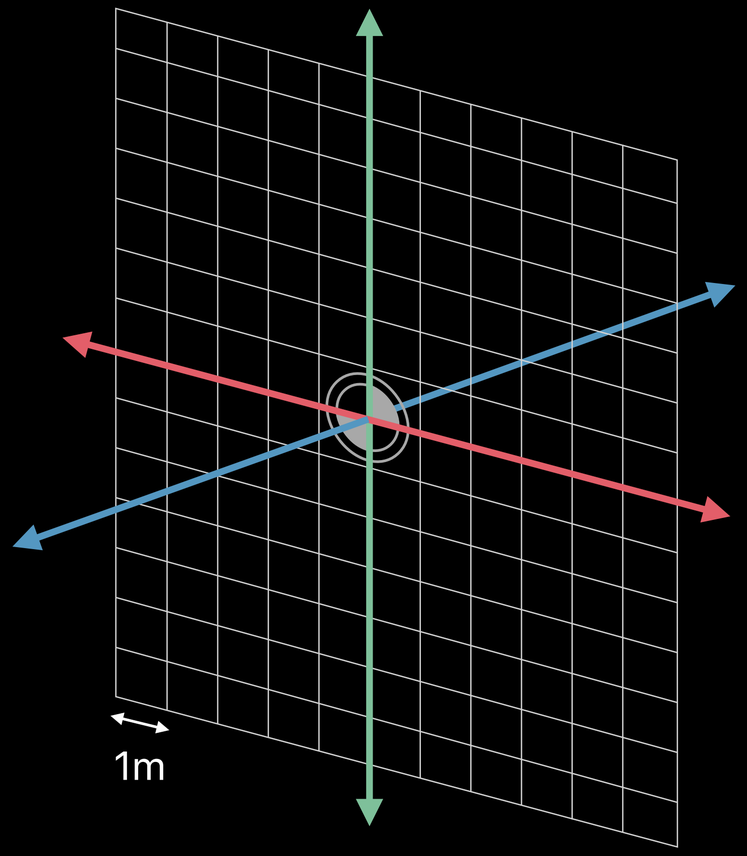
Position and orientation



World Tracking

Position and orientation

Physical distances

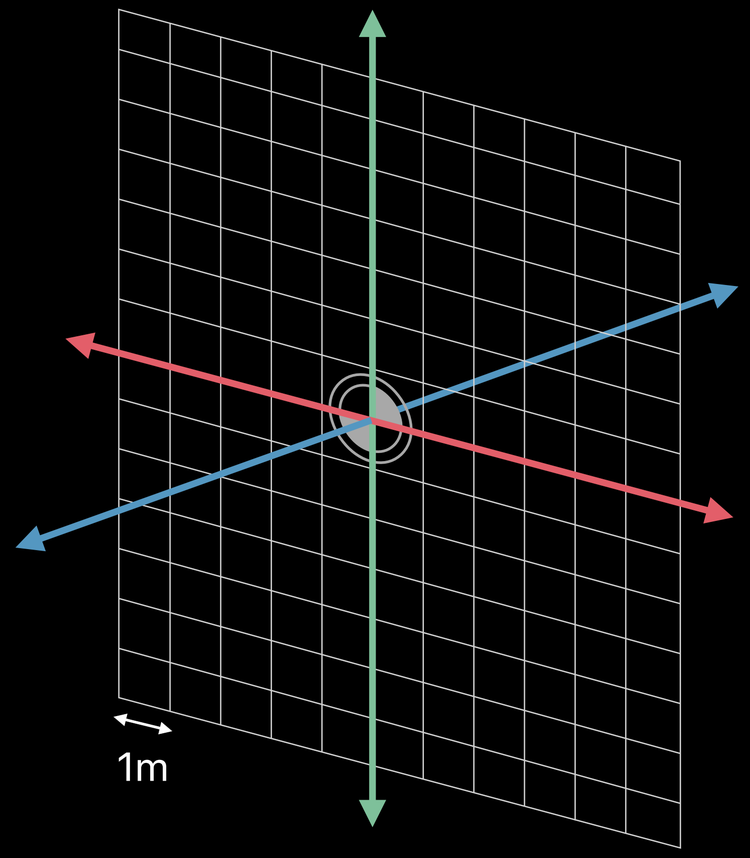


World Tracking

Position and orientation

Physical distances

Relative to starting position



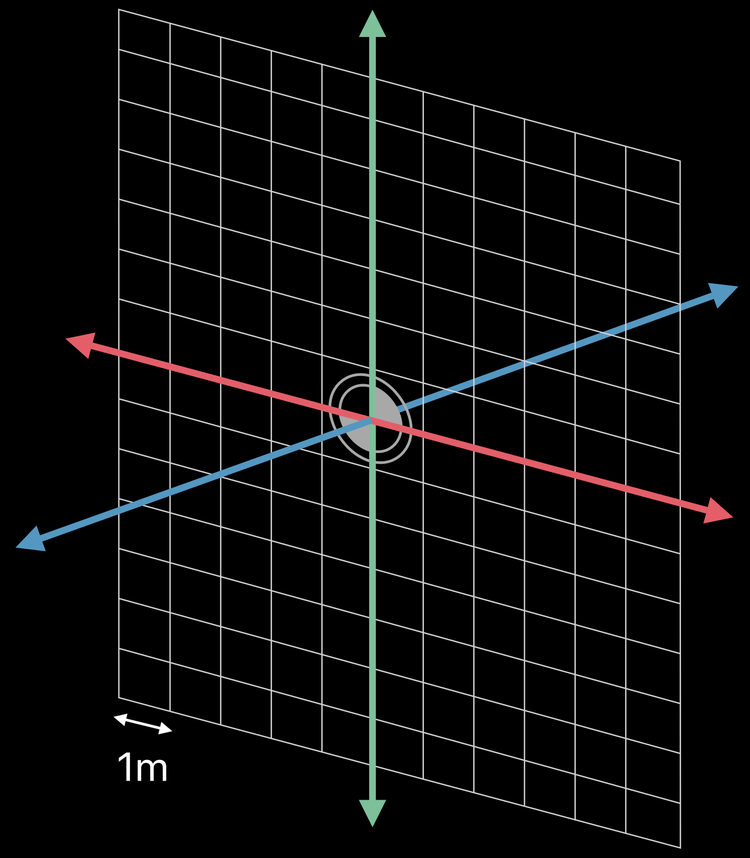
World Tracking

Position and orientation

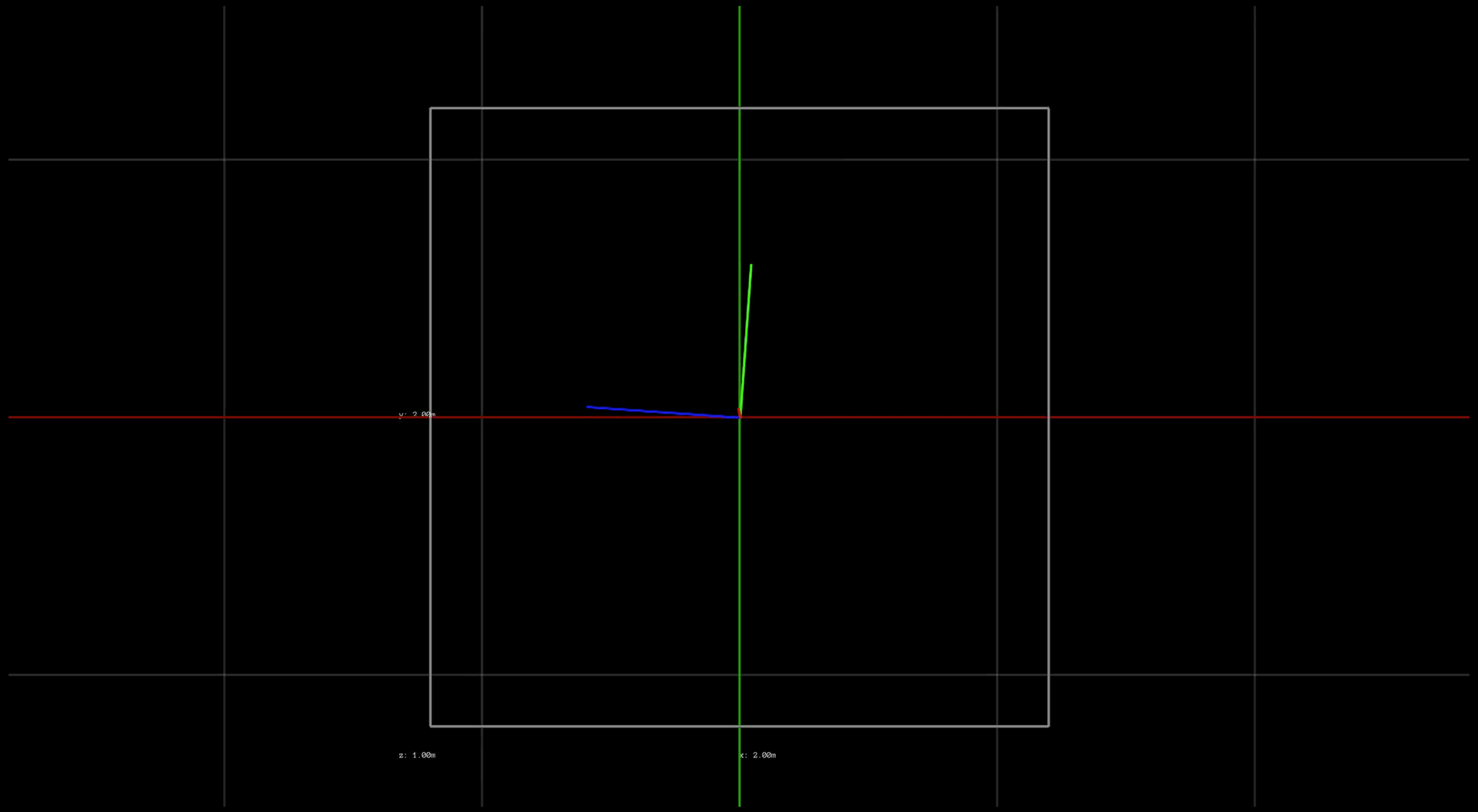
Physical distances

Relative to starting position

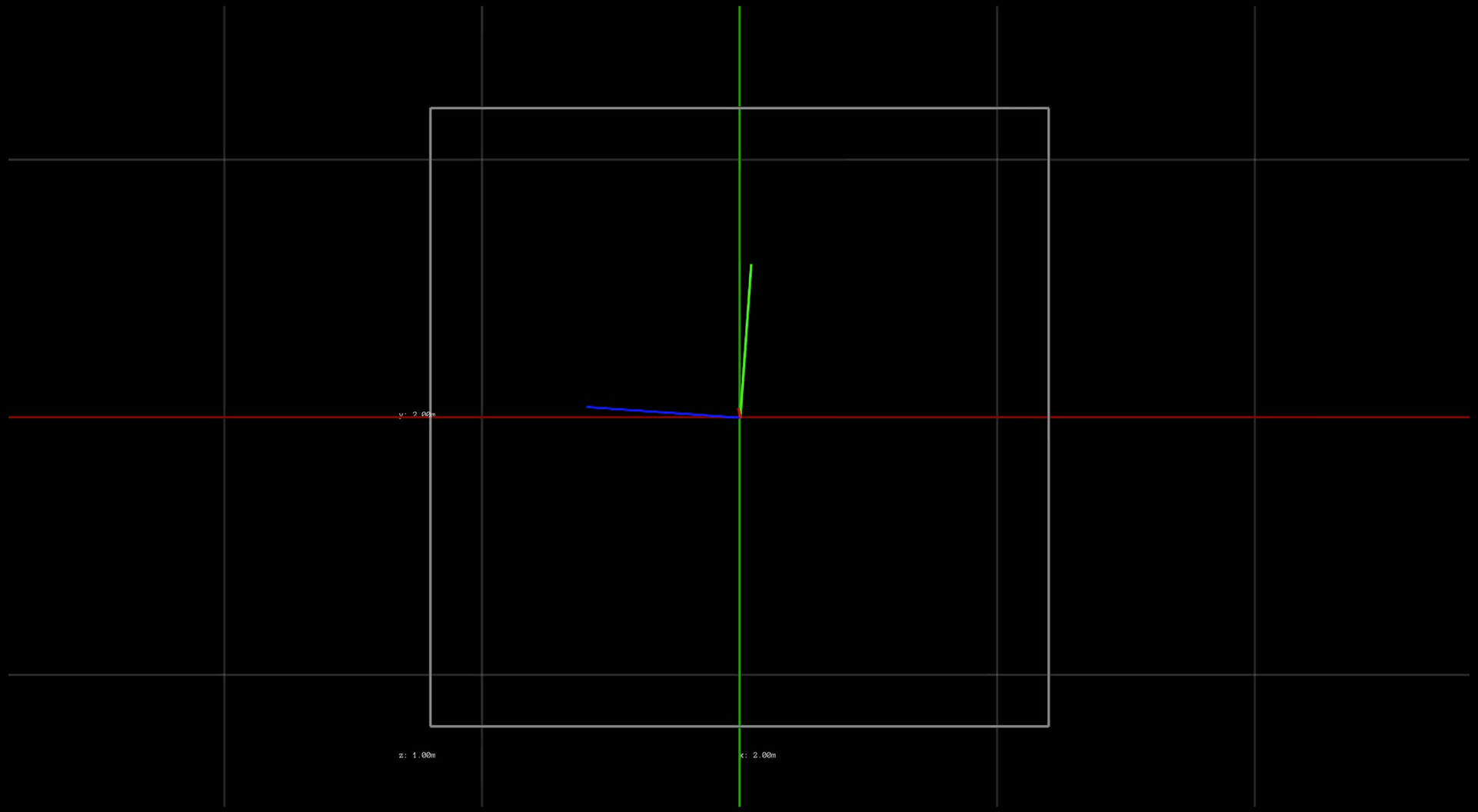
3D-feature points



World Tracking



World Tracking



```
// Create a session
let mySession = ARSession()

// Set ourselves as the session delegate
mySession.delegate = self

// Create a world tracking configuration
let configuration = ARWorldTrackingSessionConfiguration()

// Run the session
mySession.run(configuration)
```

```
// Create a session
let mySession = ARSession()

// Set ourselves as the session delegate
mySession.delegate = self

// Create a world tracking configuration
let configuration = ARWorldTrackingSessionConfiguration()

// Run the session
mySession.run(configuration)
```

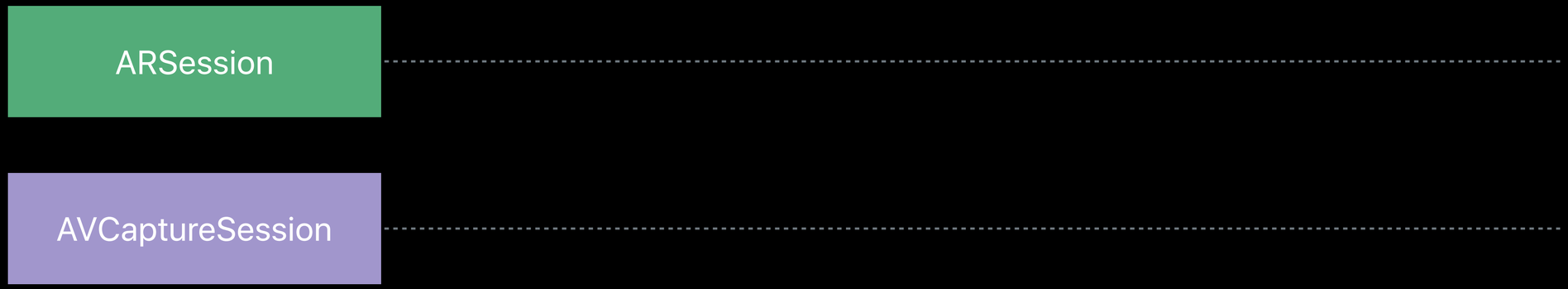
```
// Create a session
let mySession = ARSession()

// Set ourselves as the session delegate
mySession.delegate = self

// Create a world tracking configuration
let configuration = ARWorldTrackingSessionConfiguration()

// Run the session
mySession.run(configuration)
```

```
// Create a session
let mySession = ARSession()

// Set ourselves as the session delegate
mySession.delegate = self

// Create a world tracking configuration
let configuration = ARWorldTrackingSessionConfiguration()

// Run the session
mySession.run(configuration)
```

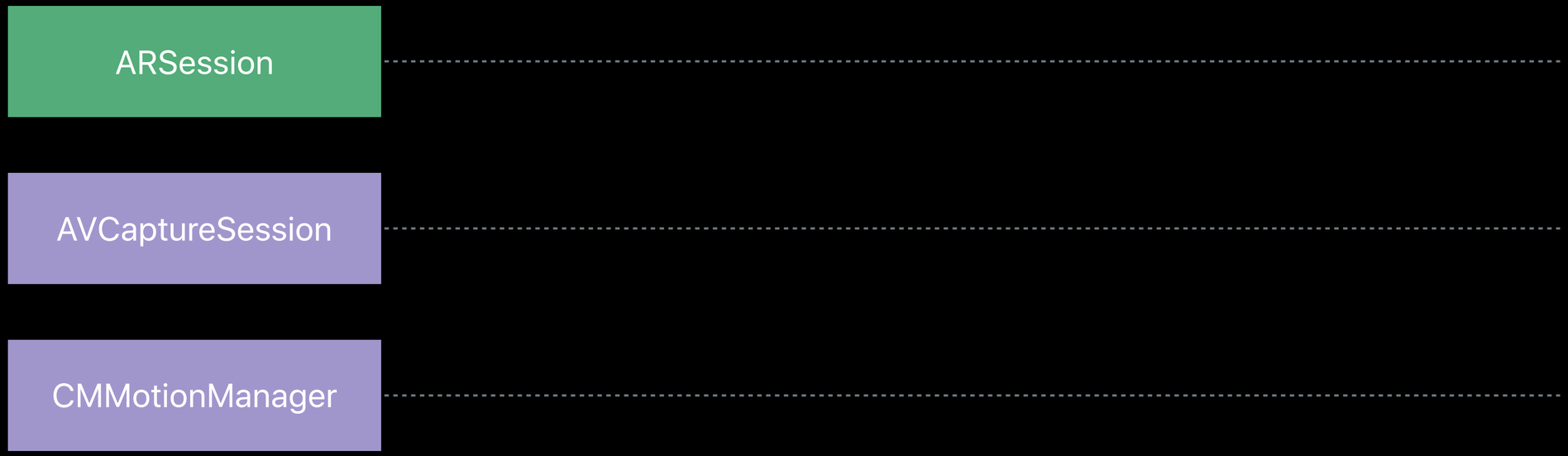
```
// Create a session
let mySession = ARSession()

// Set ourselves as the session delegate
mySession.delegate = self

// Create a world tracking configuration
let configuration = ARWorldTrackingSessionConfiguration()

// Run the session
mySession.run(configuration)
```

ARSession

ARSession

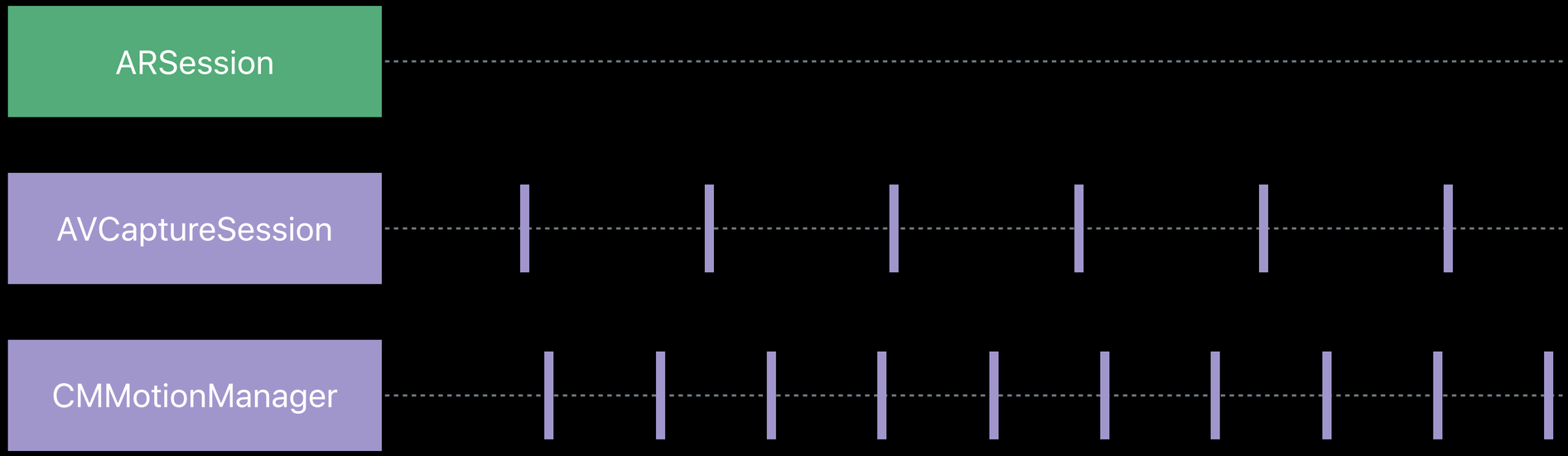


ARSession

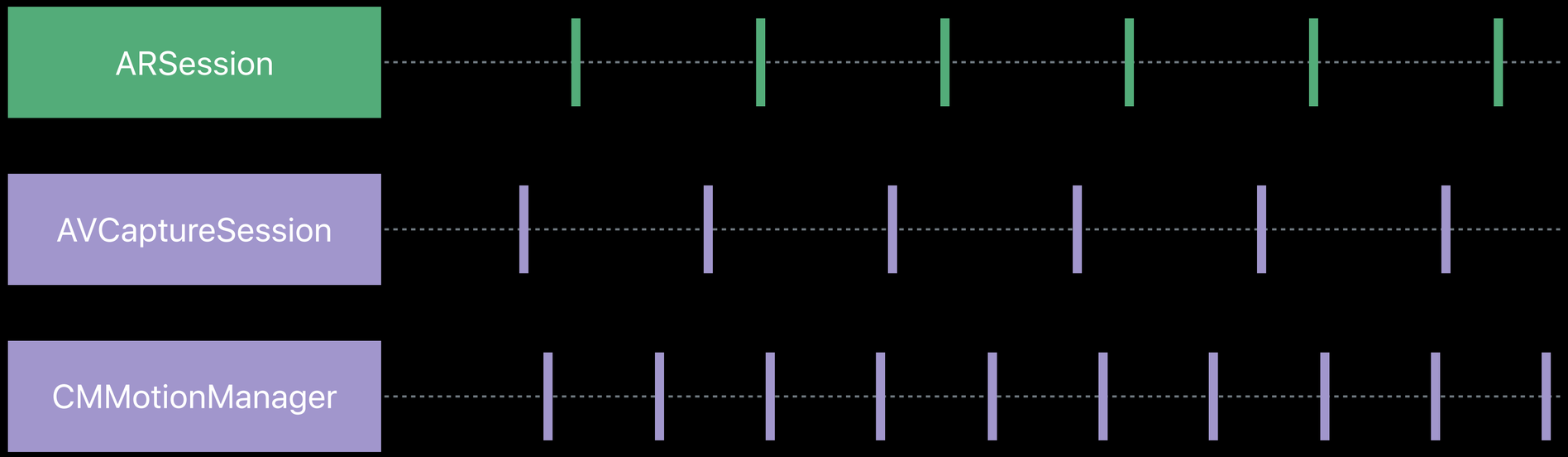


ARSession

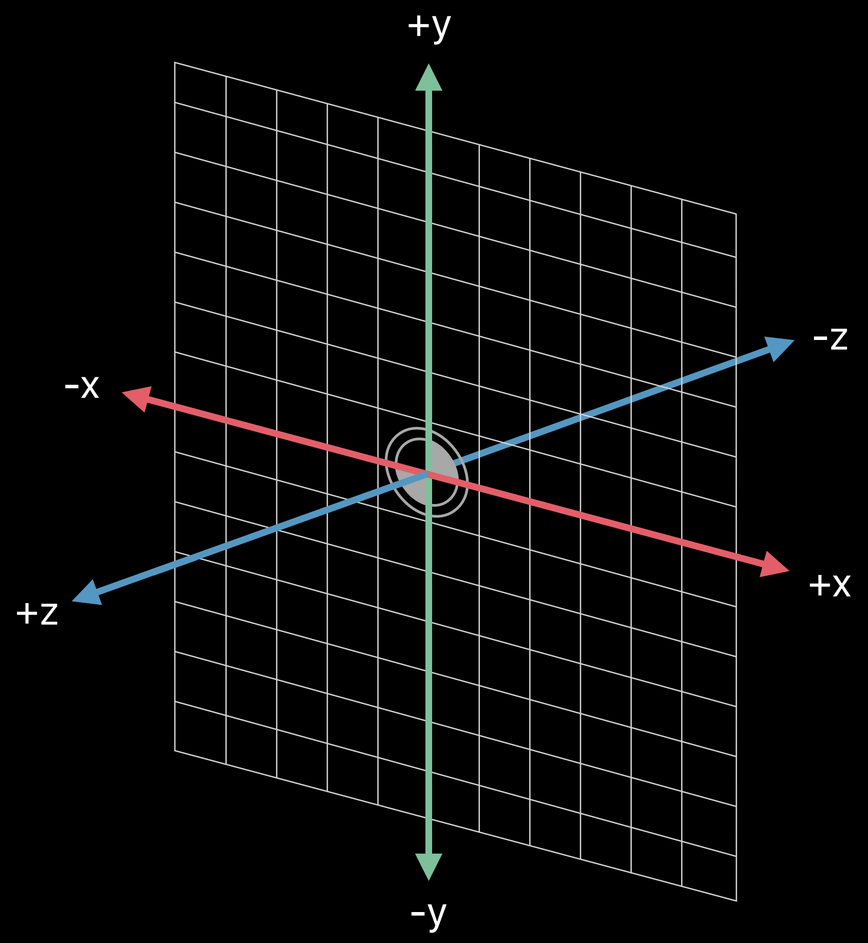
ARSession



ARSession

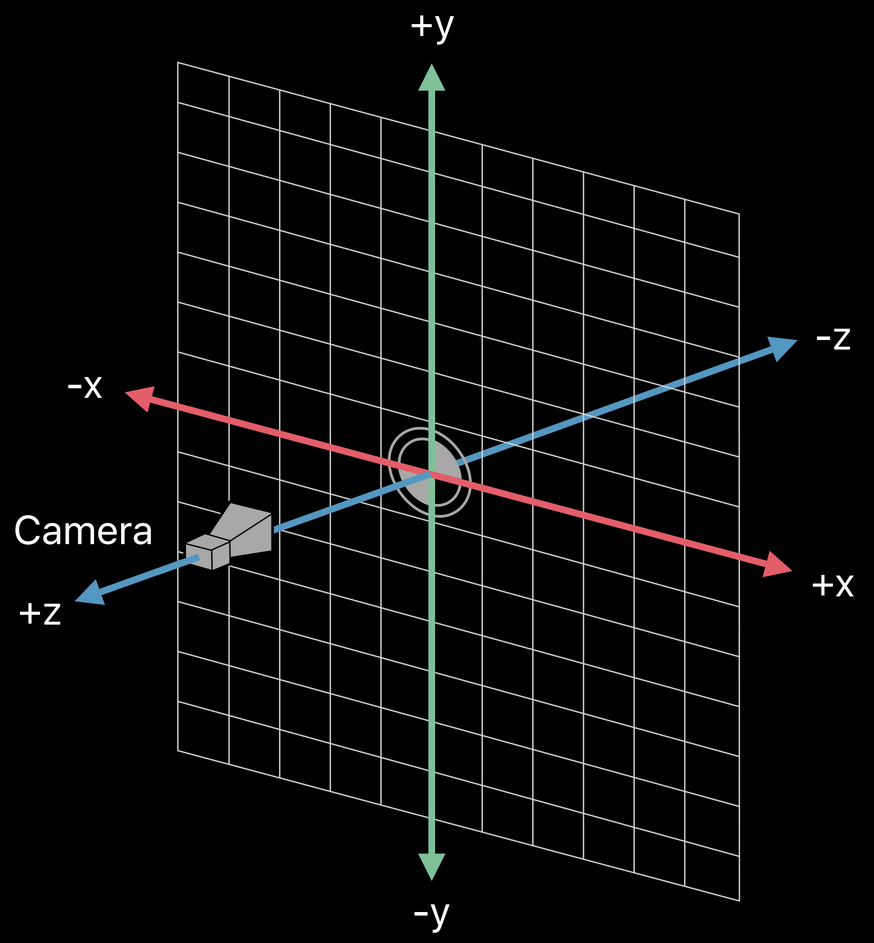


ARCamera



ARCamera

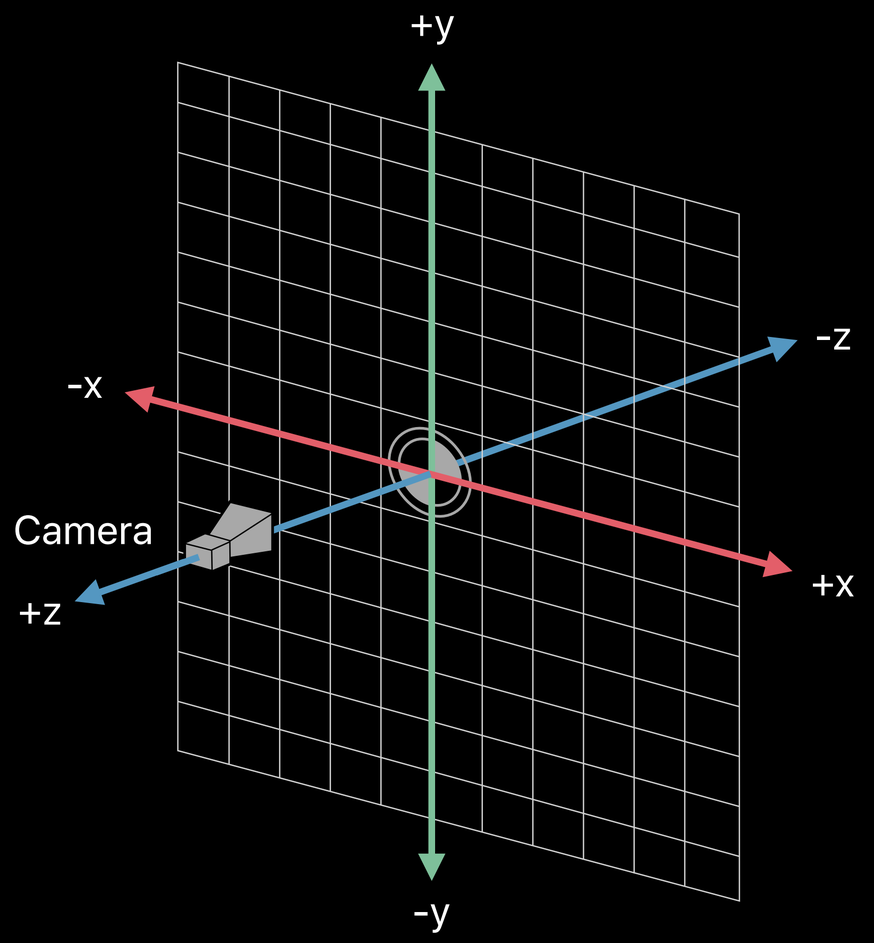
Transform



ARCamera

Transform

Tracking state

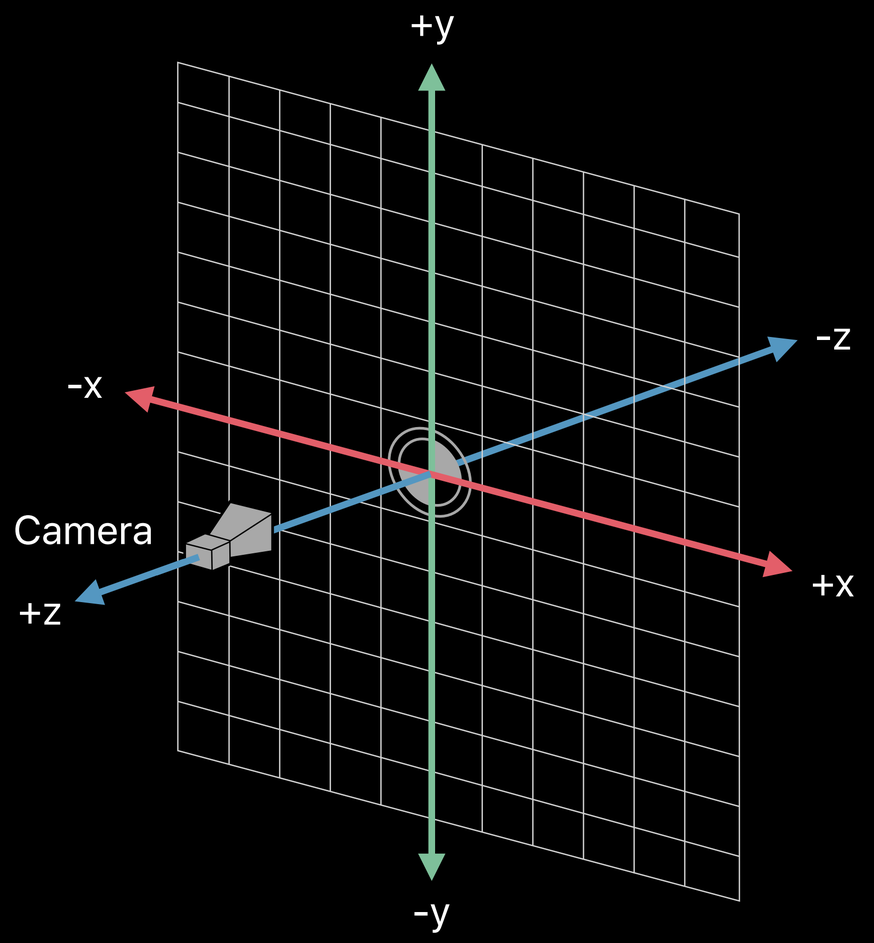


ARCamera

Transform

Tracking state

Camera intrinsics



Demo

Creating Your First ARKit Application

Tracking Quality

Tracking Quality

Uninterrupted sensor data

Tracking Quality

Uninterrupted sensor data

Textured environments

Tracking Quality

Uninterrupted sensor data

Textured environments

Static scenes

Tracking State

Not Available

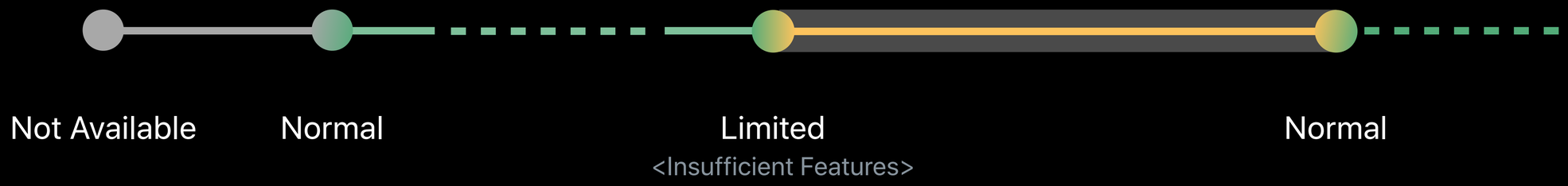
Normal

Limited

<Insufficient Features>

Normal

Tracking State




```
func session(_ session: ARSession, cameraDidChangeTrackingState camera: ARCamera) {
    if case .limited(let reason) = camera.trackingState {
        // Notify user of limited tracking state
        ...
    }
}
```

Session Interruptions

Camera input unavailable

Tracking is stopped

Session Interruptions

Camera input unavailable

Tracking is stopped

```
func sessionWasInterrupted(_ session: ARSession) {
    showOverlay()
}

func sessionInterruptionEnded(_ session: ARSession) {
    hideOverlay()
    // Optionally restart experience
    ...
}
```

Scene Understanding

Stefan Misslinger, ARKit Engineer

Scene Understanding

Scene Understanding

Scene Understanding

Plane detection

Scene Understanding

Plane detection

Hit-testing

Scene Understanding

Plane detection

Hit-testing

Light estimation

Plane Detection

Plane Detection

Horizontal with respect to gravity

Plane Detection

Horizontal with respect to gravity

Runs over multiple frames

Plane Detection

Horizontal with respect to gravity

Runs over multiple frames

Aligned extent for surface

Plane Detection

Horizontal with respect to gravity

Runs over multiple frames

Aligned extent for surface

Plane merging

```
// Enable plane detection on a session


// Create a new world tracking configuration
let configuration = ARWorldTrackingSessionConfiguration()

// Enable plane detection
configuration.planeDetection = .horizontal

// Change configuration on currently running session
mySession.run(configuration)
```

```
// Enable plane detection on a session


// Create a new world tracking configuration
let configuration = ARWorldTrackingSessionConfiguration()

// Enable plane detection
configuration.planeDetection = .horizontal

// Change configuration on currently running session
mySession.run(configuration)
```

```
// Enable plane detection on a session



// Create a new world tracking configuration
let configuration = ARWorldTrackingSessionConfiguration()

// Enable plane detection
configuration.planeDetection = .horizontal

// Change configuration on currently running session
mySession.run(configuration)
```

```
// Enable plane detection on a session


// Create a new world tracking configuration
let configuration = ARWorldTrackingSessionConfiguration()

// Enable plane detection
configuration.planeDetection = .horizontal

// Change configuration on currently running session
mySession.run(configuration)
```

```
// Enable plane detection on a session


// Create a new world tracking configuration
let configuration = ARWorldTrackingSessionConfiguration()

// Enable plane detection
configuration.planeDetection = .horizontal

// Change configuration on currently running session
mySession.run(configuration)
```

ARPlaneAnchor

ARPlaneAnchor

Transform

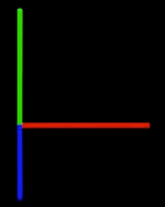
ARPlaneAnchor

Transform

```
// Called when a new plane was detected
func session(_ session: ARSession, didAdd anchors: [ARAnchor]) {
    addPlaneGeometry(forAnchors: anchors)
}
```

ARPlaneAnchor

Transform



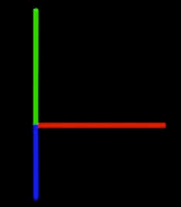
```
// Called when a new plane was detected
func session(_ session: ARSession, didAdd anchors: [ARAnchor]) {
    addPlaneGeometry(forAnchors: anchors)
}
```

ARPlaneAnchor

Transform

Extent

Center



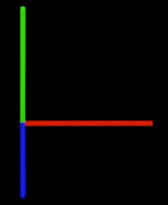
```
// Called when a new plane was detected
func session(_ session: ARSession, didAdd anchors: [ARAnchor]) {
    addPlaneGeometry(forAnchors: anchors)
}
```

ARPlaneAnchor

Transform

Extent

Center

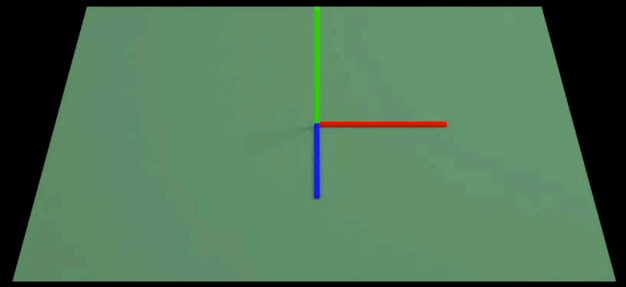


ARPlaneAnchor

Transform

Extent

Center



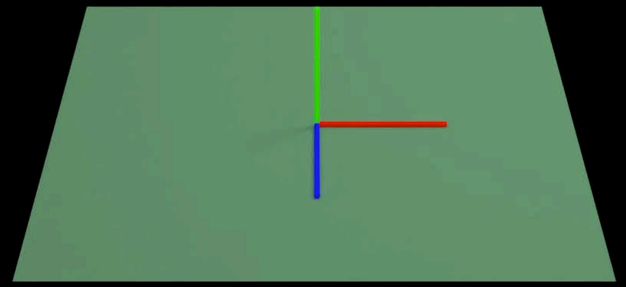
```
// Called when a plane's transform or extent is updated
func session(_ session: ARSession, didUpdate anchors: [ARAnchor]) {
    updatePlaneGeometry(forAnchors: anchors)
}
```

ARPlaneAnchor

Transform

Extent

Center



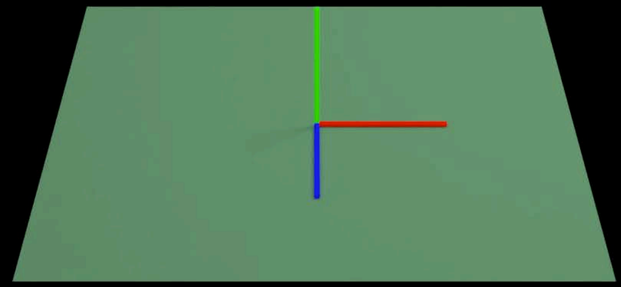
```
// Called when a plane's transform or extent is updated
func session(_ session: ARSession, didUpdate anchors: [ARAnchor]) {
    updatePlaneGeometry(forAnchors: anchors)
}
```

ARPlaneAnchor

Transform

Extent

Center

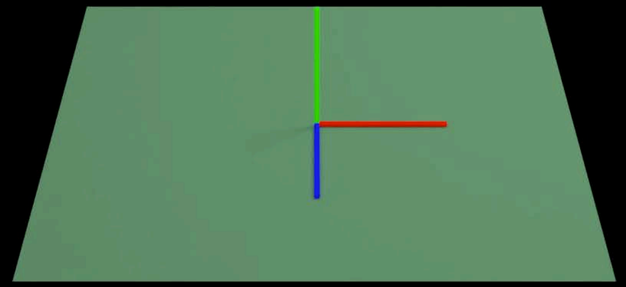


ARPlaneAnchor

Transform

Extent

Center



```
// Called when a plane was removed as a result of a merge
func session(_ session: ARSession, didRemove anchors: [ARAnchor]) {
    removePlaneGeometry(forAnchors: anchors)
}
```

ARPlaneAnchor

Transform

Extent

Center

```
// Called when a plane was removed as a result of a merge
func session(_ session: ARSession, didRemove anchors: [ARAnchor]) {
    removePlaneGeometry(forAnchors: anchors)
}
```

Hit-Testing

Hit-Testing

Intersect ray with real world

Hit-Testing

Intersect ray with real world

Uses scene information

Hit-Testing

Intersect ray with real world

Uses scene information

Results sorted by distance

Hit-Testing

Intersect ray with real world

Uses scene information

Results sorted by distance

Hit-test types

Hit-Test Types

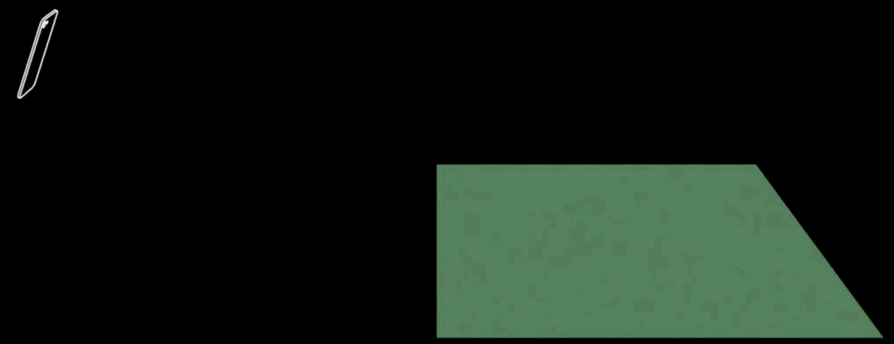
Hit-Test Types

Existing plane using extent

Hit-Test Types

Existing plane using extent

Existing plane



Hit-Test Types

Existing plane using extent

Existing plane

Estimated plane

Hit-Test Types

Existing plane using extent

Existing plane

Estimated plane

Feature point

```
// Adding an ARAnchor based on hit-test
let point = CGPoint(x: 0.5, y: 0.5)  // Image center

// Perform hit-test on frame
let results = frame.hitTest(point, types: [.existingPlane, .estimatedHorizontalPlane])

// Use the first result
if let closestResult = results.first {

   // Create an Anchor for it
   let anchor = ARAnchor(transform: closestResult.worldTransform)

   // Add it to the session
   session.add(anchor: anchor)

}
```

```
// Adding an ARAnchor based on hit-test
let point = CGPoint(x: 0.5, y: 0.5)  // Image center

// Perform hit-test on frame
let results = frame.hitTest(point, types: [.existingPlane, .estimatedHorizontalPlane])

// Use the first result
if let closestResult = results.first {

    // Create an Anchor for it
    let anchor = ARAnchor(transform: closestResult.worldTransform)

    // Add it to the session
    session.add(anchor: anchor)

}
```

```swift
// Adding an ARAnchor based on hit-test
let point = CGPoint(x: 0.5, y: 0.5)  // Image center

// Perform hit-test on frame
let results = frame.hitTest(point, types: [.existingPlane, .estimatedHorizontalPlane])

// Use the first result
if let closestResult = results.first {

    // Create an Anchor for it
    let anchor = ARAnchor(transform: closestResult.worldTransform)

    // Add it to the session
    session.add(anchor: anchor)

}
```

```swift
// Adding an ARAnchor based on hit-test
let point = CGPoint(x: 0.5, y: 0.5)  // Image center

// Perform hit-test on frame
let results = frame.hitTest(point, types: [.existingPlane, .estimatedHorizontalPlane])

// Use the first result
if let closestResult = results.first {

    // Create an Anchor for it
    let anchor = ARAnchor(transform: closestResult.worldTransform)

    // Add it to the session
    session.add(anchor: anchor)

}
```

```swift
// Adding an ARAnchor based on hit-test
let point = CGPoint(x: 0.5, y: 0.5)  // Image center

// Perform hit-test on frame
let results = frame.hitTest(point, types: [.existingPlane, .estimatedHorizontalPlane])

// Use the first result
if let closestResult = results.first {

    // Create an Anchor for it
    let anchor = ARAnchor(transform: closestResult.worldTransform)

    // Add it to the session
    session.add(anchor: anchor)

}
```

```swift
// Adding an ARAnchor based on hit-test
let point = CGPoint(x: 0.5, y: 0.5)  // Image center

// Perform hit-test on frame
let results = frame.hitTest(point, types: [.existingPlane, .estimatedHorizontalPlane])

// Use the first result
if let closestResult = results.first {

    // Create an Anchor for it
    let anchor = ARAnchor(transform: closestResult.worldTransform)

    // Add it to the session
    session.add(anchor: anchor)

}
```

Light Estimation

Light Estimation

Ambient intensity based on captured image

Light Estimation

Ambient intensity based on captured image

Defaults to 1000 lumen

Light Estimation

Ambient intensity based on captured image

Defaults to 1000 lumen

Enabled by default

Light Estimation

Ambient intensity based on captured image

Defaults to 1000 lumen

Enabled by default

```
configuration.isLightEstimationEnabled = true
```

Light Estimation

Ambient intensity based on captured image

Defaults to 1000 lumen

Enabled by default

```
configuration.isLightEstimationEnabled = true
```

```
// Get ambient intensity value
let intensity = frame.lightEstimate?.ambientIntensity
```

Demo

Scene understanding in ARKit

Rendering

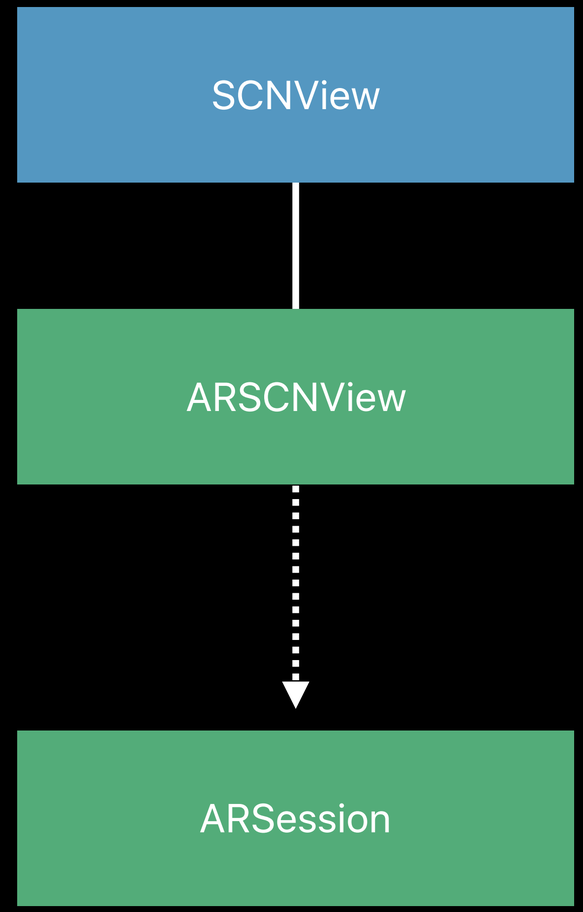
SceneKit

SpriteKit

Metal

SceneKit

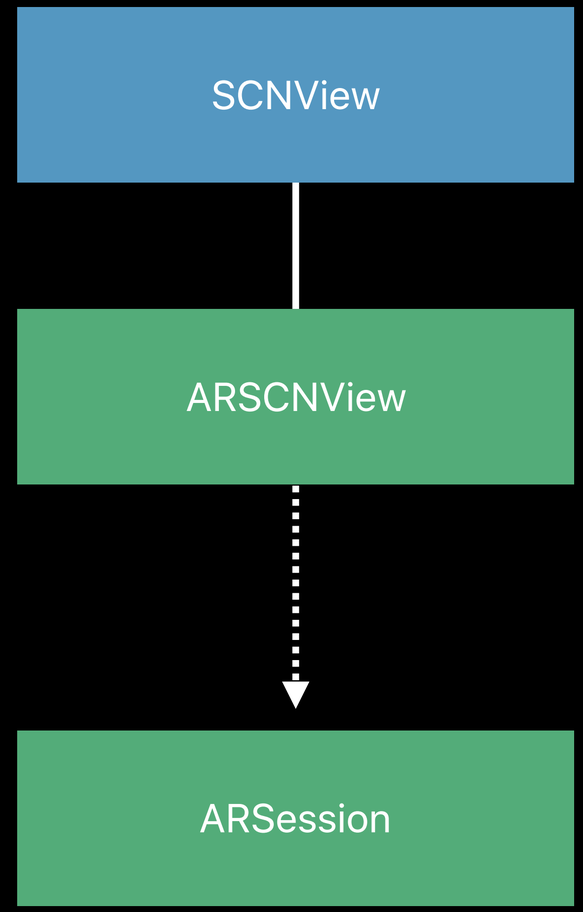
ARSCNView



SceneKit

ARSCNView

Draws captured image

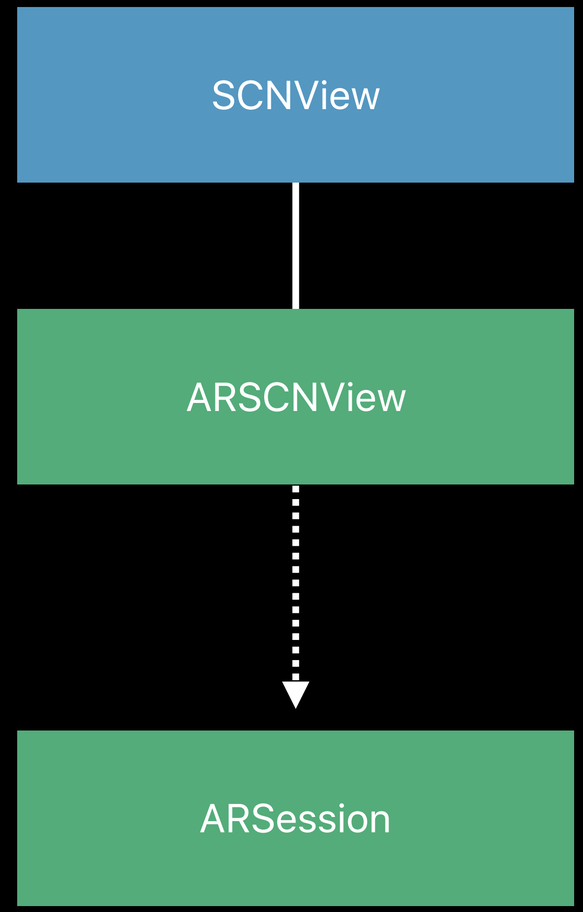


SceneKit

ARSCNView

Draws captured image

Updates a SCNCamera



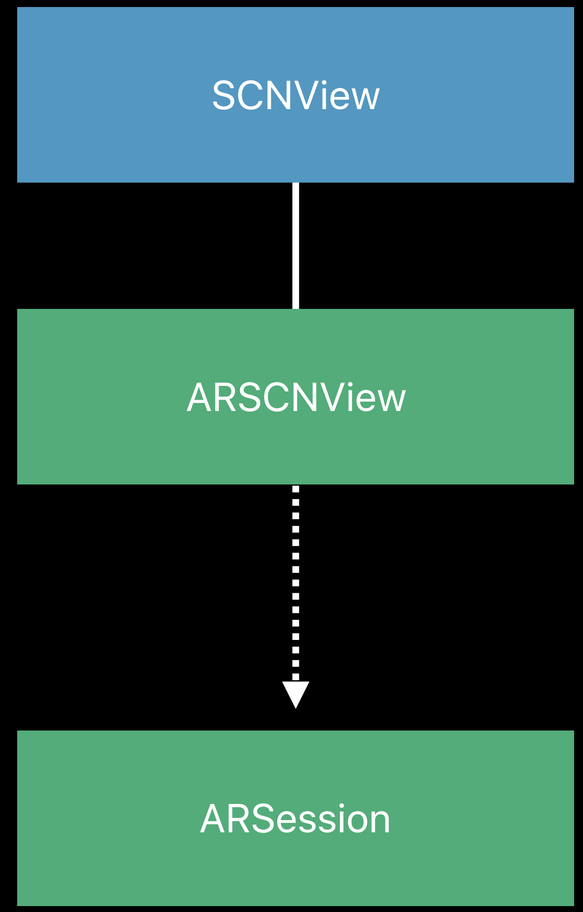
SceneKit

ARSCNView

Draws captured image

Updates a SCNCamera

Updates scene lighting



SceneKit

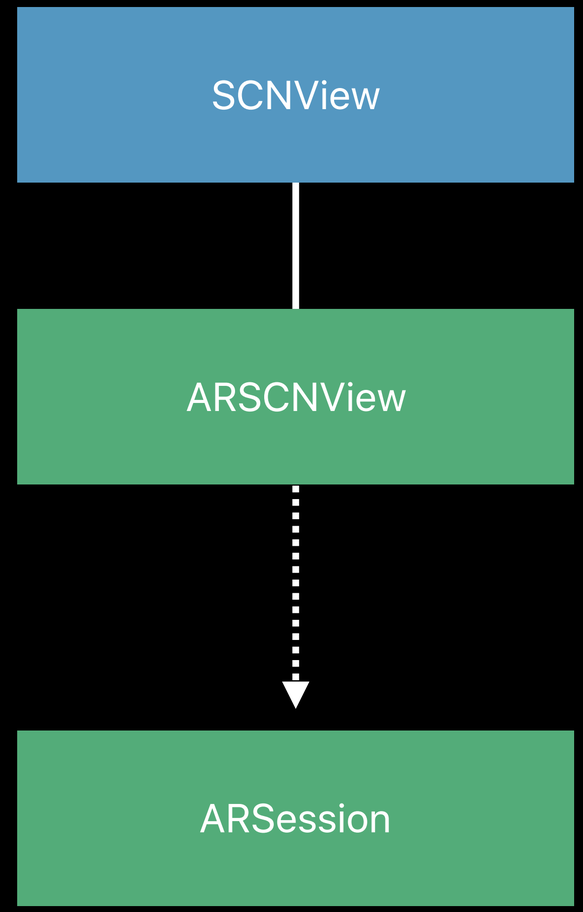
ARSCNView

Draws captured image

Updates a SCNCamera

Updates scene lighting

Maps SCNNodes to ARAnchors



SceneKit

ARSCNViewDelegate

ARSCNView

ARSCNViewDelegate

SceneKit

ARSCNViewDelegate

ARSCNView

```
func session(_: ARSession, didAdd: [ARAnchor])
```

ARSCNViewDelegate

SceneKit

ARSCNViewDelegate

ARSCNView

```
func session(_: ARSession, didAdd: [ARAnchor])
```


ARSCNViewDelegate

```
func renderer(_: SCNSceneRenderer,
        nodeFor: ARAnchor) -> SCNNode?
```

SceneKit

ARSCNViewDelegate

ARSCNView

```
func session(_: ARSession, didAdd: [ARAnchor])
```

ARSCNViewDelegate

```
func renderer(_: SCNSceneRenderer,
        nodeFor: ARAnchor) -> SCNNode?
```

```
func renderer(_: SCNSceneRenderer,
         didAdd: SCNNode,
            for: ARAnchor)
```

SceneKit

ARSCNViewDelegate

ARSCNView

ARSCNViewDelegate

SceneKit

ARSCNViewDelegate

ARSCNView

```
func session(_: ARSession, didUpdate: [ARAnchor])
```

ARSCNViewDelegate

SceneKit

ARSCNViewDelegate

ARSCNView

```
func session(_: ARSession, didUpdate: [ARAnchor])
```


ARSCNViewDelegate

```
func renderer(_: SCNSceneRenderer,
     willUpdate: SCNNode,
            for: ARAnchor)
```

SceneKit

ARSCNViewDelegate

ARSCNView

```
func session(_: ARSession, didUpdate: [ARAnchor])
```

ARSCNViewDelegate

```
func renderer(_: SCNSceneRenderer,
     willUpdate: SCNNode,
            for: ARAnchor)
```

```
func renderer(_: SCNSceneRenderer,
      didUpdate: SCNNode,
            for: ARAnchor)
```

SceneKit

ARSCNViewDelegate

ARSCNView

ARSCNViewDelegate

SceneKit

ARSCNViewDelegate

ARSCNView

```
func session(_: ARSession, didRemove: [ARAnchor])
```

ARSCNViewDelegate

SceneKit

ARSCNViewDelegate

ARSCNView

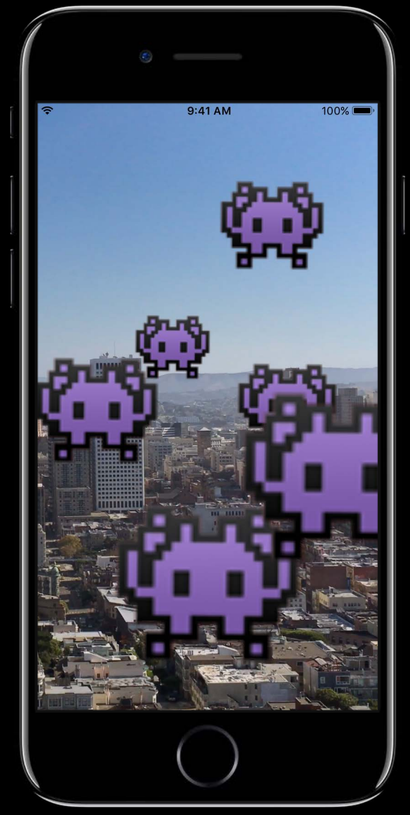
```
func session(_: ARSession, didRemove: [ARAnchor])
```

ARSCNViewDelegate

```
func renderer(_: SCNSceneRenderer,
      didRemove: SCNNode,
            for: ARAnchor)
```

SpriteKit

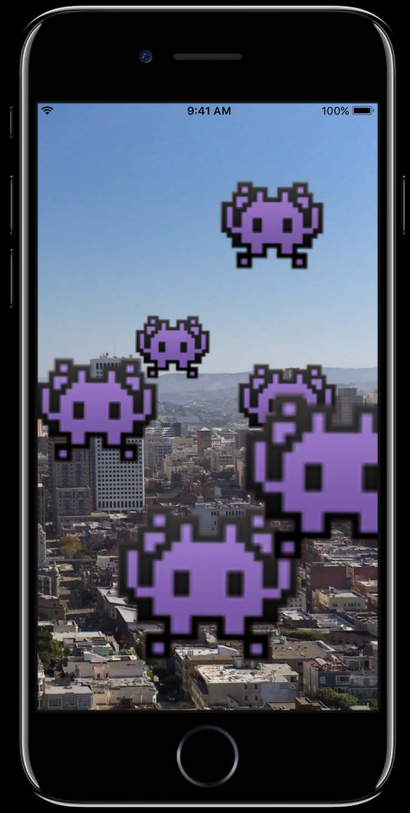
ARSKView



SpriteKit

ARSKView

Contains ARSession



SpriteKit

ARSKView

Contains ARSession

Draws captured image

SpriteKit

ARSKView

Contains ARSession

Draws captured image

Maps SKNodes to ARAnchors

SpriteKit

ARSKView

Contains ARSession

Draws captured image

Maps SKNodes to ARAnchors

Sprites are billboarded to physical locations

SpriteKit

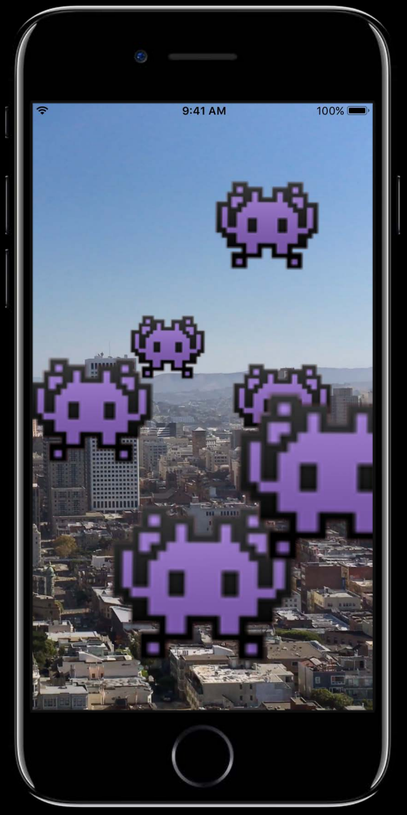
ARSKView

Contains ARSession

Draws captured image

Maps SKNodes to ARAnchors

Sprites are billboarded to physical locations



Custom Rendering

Processing

Custom Rendering

Processing

Draw camera background

Custom Rendering

Processing

Draw camera background

Update virtual camera

Custom Rendering

Processing

Draw camera background

Update virtual camera

Update lighting

Custom Rendering

Processing

Draw camera background

Update virtual camera

Update lighting

Update transforms for geometry

Custom Rendering

Accessing ARFrame

Polling

```
if let frame = mySession.currentFrame {
    if( frame.timestamp > _lastTimestamp ) {
         updateRenderer(frame)   // Update renderer with frame
        _lastTimestamp = frame.timestamp
    }
}
```

Custom Rendering

Accessing ARFrame

Polling

```
if let frame = mySession.currentFrame {
    if( frame.timestamp > _lastTimestamp ) {
         updateRenderer(frame)   // Update renderer with frame
        _lastTimestamp = frame.timestamp
    }
}
```

Delegate

```
func session(_ session: ARSession, didUpdate frame: ARFrame) {
    // Update renderer with frame
    updateRenderer(frame)
}
```

```
func updateRenderer(_ frame: ARFrame) {

    // Draw background camera image
    drawCameraImage(withPixelBuffer: frame.capturedImage)

    // Update virtual camera
    let viewMatrix = simd_inverse(frame.camera.transform)
    let projectionMatrix = frame.camera.projectionMatrix
    updateCamera(viewMatrix, projectionMatrix)

    // Update lighting
    updateLighting(frame.lightEstimate?.ambientIntensity)

    // Update geometry based on anchors
    drawGeometry(forAnchors: frame.anchors)
}
```

```
func updateRenderer(_ frame: ARFrame) {

    // Draw background camera image
    drawCameraImage(withPixelBuffer: frame.capturedImage)

    // Update virtual camera
    let viewMatrix = simd_inverse(frame.camera.transform)
    let projectionMatrix = frame.camera.projectionMatrix
    updateCamera(viewMatrix, projectionMatrix)

    // Update lighting
    updateLighting(frame.lightEstimate?.ambientIntensity)

    // Update geometry based on anchors
    drawGeometry(forAnchors: frame.anchors)
}
```

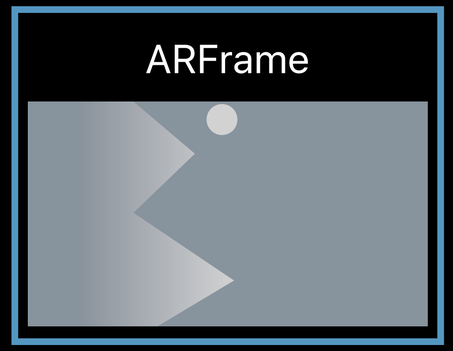
```
func updateRenderer(_ frame: ARFrame) {

    // Draw background camera image
    drawCameraImage(withPixelBuffer: frame.capturedImage)

    // Update virtual camera
    let viewMatrix = simd_inverse(frame.camera.transform)
    let projectionMatrix = frame.camera.projectionMatrix
    updateCamera(viewMatrix, projectionMatrix)

    // Update lighting
    updateLighting(frame.lightEstimate?.ambientIntensity)

    // Update geometry based on anchors
    drawGeometry(forAnchors: frame.anchors)
}
```

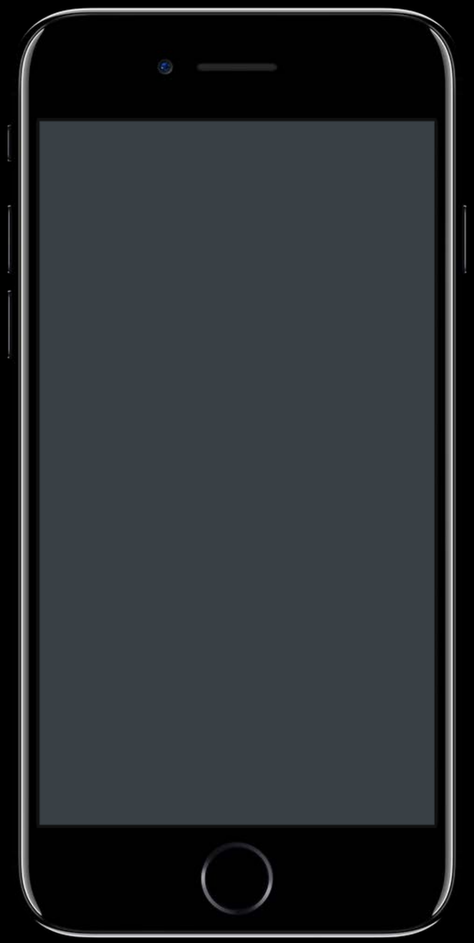
```
func updateRenderer(_ frame: ARFrame) {

    // Draw background camera image
    drawCameraImage(withPixelBuffer: frame.capturedImage)

    // Update virtual camera
    let viewMatrix = simd_inverse(frame.camera.transform)
    let projectionMatrix = frame.camera.projectionMatrix
    updateCamera(viewMatrix, projectionMatrix)

    // Update lighting
    updateLighting(frame.lightEstimate?.ambientIntensity)

    // Update geometry based on anchors
    drawGeometry(forAnchors: frame.anchors)
}
```

```
func updateRenderer(_ frame: ARFrame) {

    // Draw background camera image
    drawCameraImage(withPixelBuffer: frame.capturedImage)

    // Update virtual camera
    let viewMatrix = simd_inverse(frame.camera.transform)
    let projectionMatrix = frame.camera.projectionMatrix
    updateCamera(viewMatrix, projectionMatrix)

    // Update lighting
    updateLighting(frame.lightEstimate?.ambientIntensity)

    // Update geometry based on anchors
    drawGeometry(forAnchors: frame.anchors)
}
```

Custom Rendering

Drawing to a viewport





Custom Rendering

Drawing to a viewport

ARFrame

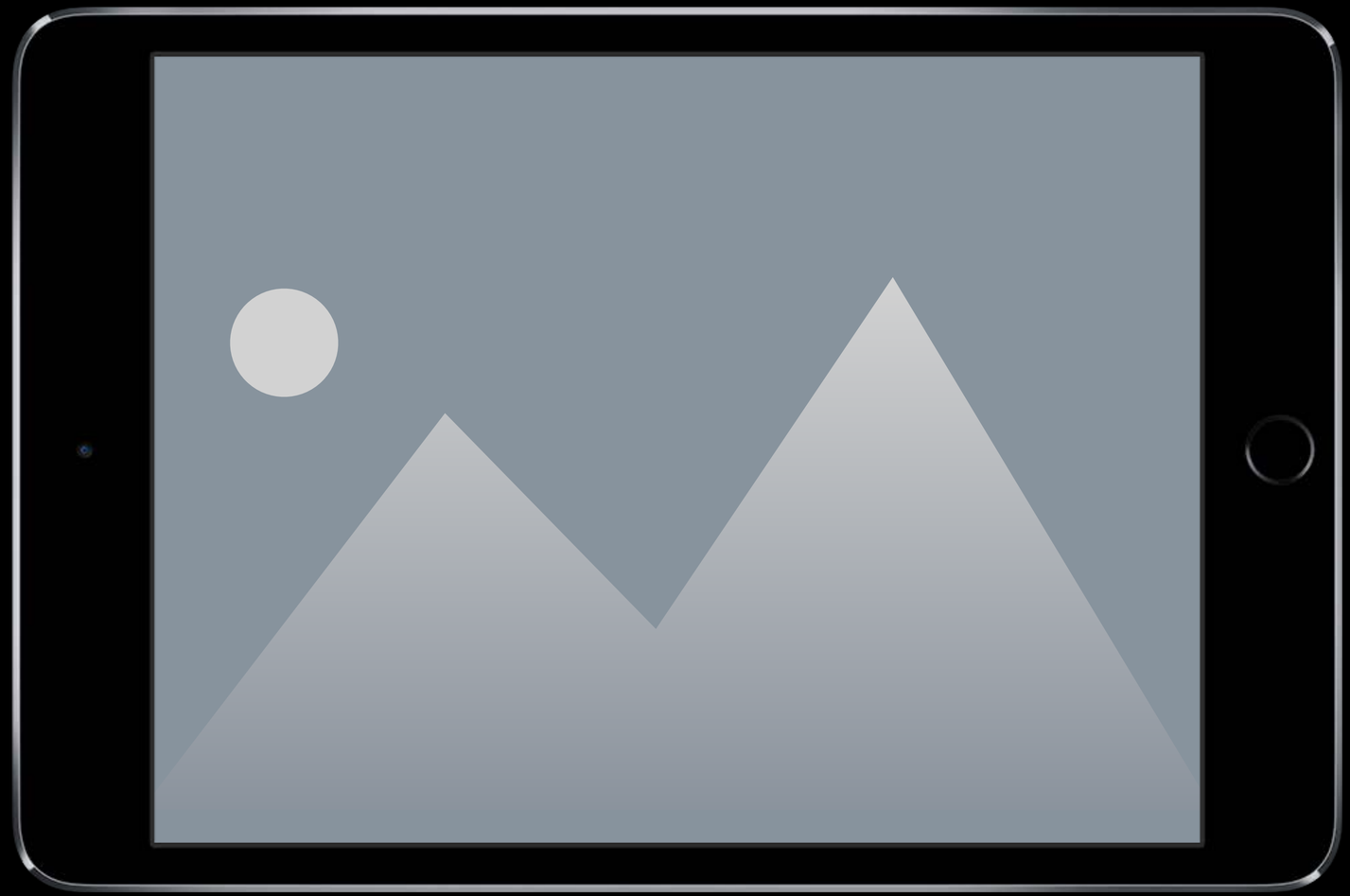
Custom Rendering

Drawing to a viewport

Custom Rendering

Drawing to a viewport

ARFrame



Custom Rendering

Drawing to a viewport

Custom Rendering

Drawing to a viewport

Helper methods

Custom Rendering

Drawing to a viewport

Helper methods

```
// Get the frame's display transform for the given viewport size and orientation
let transform = frame.displayTransform(withViewportSize: viewportSize, orientation: .portrait)
```

Custom Rendering

Drawing to a viewport

Helper methods

```
// Get the frame's display transform for the given viewport size and orientation
let transform = frame.displayTransform(withViewportSize: viewportSize, orientation: .portrait)
```

```
// Get the camera's projection matrix for the given viewport size and orientation
let projectionMatrix = camera.projectionMatrix(withViewportSize: viewportSize,
                                               orientation: .portrait,
                                                     zNear: 0.001,
                                                      zFar: 1000)
```

Summary

High-level API

Tracking

Scene Understanding

Rendering

More Information

https://developer.apple.com/wwdc17/602

Related Sessions

Introducing Metal	Executive Ballroom	Tuesday 1:50PM
SceneKit: What's New	Grand Ballroom A	Wednesday 11:00AM
Going Beyond 2D with SpriteKit	Executive Ballroom	Friday 10:00AM

Labs

ARKit Lab	Technology Lab A	Wed 1:00PM–3:00PM
AR/VR Get Together	Technology Lab A	Wed 6:30-7:45PM
ARKit Lab	Technology Lab A	Thu 12:00PM–3:00PM


WWDC17