Building Apps with RealityKit

Ross Dexter, RealityKit Engineer
Courtland Idstrom, RealityKit Engineer
Expands on the intro session
Applied usage of RealityKit
RealityKit Recap

AR First
Realistic rendering and simulation
Designed for Swift
Building Memory Cards
Building Memory Cards

Prototype
Building Memory Cards

Prototype

Adding polish
Building Memory Cards

Prototype

Adding polish

Tracking game state
Building Memory Cards

Prototype

Adding polish

Tracking game state

Multiplayer
Prototype
RealityKit Elements

ARView

Scene
RealityKit Elements

ARView

Scene

Anchor
RealityKit Elements

- ARView
  - Scene
    - Anchor
      - Entity
      - Entity
      - Entity
      - Entity
      - ...
RealityKit Elements

- ARView
  - Scene
    - Anchor
      - Entity
      - Entity
      - Entity
      - Entity
      - Entity
      - Entity
RealityKit Elements

- ARView
  - Scene
    - Anchor
      - Entity
        - Entity
        - Entity
        - Entity
        - Entity
      - Entity
        - Entity
        - Entity
        - Entity
        - Entity
      ...
RealityKit Elements

ARView

Scene

Anchor

Entity
Entity
Entity
Entity
Entity
Entity
Entity
Entity
Anchoring

Integrated with ARKit
Anchoring

Integrated with ARKit

Create AnchorEntity
Anchoring

Integrated with ARKit

Create AnchorEntity

Declare anchoring type
Anchoring

Integrated with ARKit

Create AnchorEntity

Declare anchoring type

Add to Scene
Anchoring

Integrated with ARKit

Create AnchorEntity

Declare anchoring type

Add to Scene

Automatically tracks target
Anchoring Types

- Plane
- Image
- Object
- Face
- Body
- Camera
- ARRaycastResult
- World
- ARAnchor
Anchoring

Single anchor for Memory Cards
Anchoring

Single anchor for Memory Cards

Horizontal surface, 20 cm$^2$
Anchoring

Single anchor for Memory Cards

Horizontal surface, $20 \text{ cm}^2$

Places center of game board in the world
import UIKit
import RealityKit

class ViewController: UIViewController {
    @IBOutlet var arView: ARView!

    override func viewDidLoad() {
        super.viewDidLoad()
    }
}
import UIKit
import RealityKit

class ViewController: UIViewController {
    @IBOutlet var arView: ARView!

    override func viewDidLoad() {
        super.viewDidLoad()
    }
}

override func viewDidLoad() {
    super.viewDidLoad()
}
// Memory Cards Prototype

import UIKit
import RealityKit

class ViewController: UIViewController {

    @IBOutlet var arView: ARView!

    override func viewDidLoad() {
        super.viewDidLoad()
    }

    override func viewDidLoad() {
        super.viewDidLoad()
    }
}
// Memory Cards Prototype

import UIKit
import RealityKit

class ViewController: UIViewController {

    @IBOutlet var arView: ARView!

    override func viewDidLoad() {
        super.viewDidLoad()
        // Create an anchor for a horizontal plane with a minimum area of 20 cm²
        let anchor = AnchorEntity(plane: .horizontal, minimumBounds: [0.2, 0.2])
        arView.scene.addAnchor(anchor)

        // Attach content to anchor here
    }
}
// Memory Cards Prototype

import UIKit
import RealityKit

class ViewController: UIViewController {

    @IBOutlet var arView: ARView!

    override func viewDidLoad() {
        super.viewDidLoad()

        // Create an anchor for a horizontal plane with a minimum area of 20 cm²
        let anchor = AnchorEntity(plane: .horizontal, minimumBounds: [0.2, 0.2])
        arView.scene.addAnchor(anchor)

        // Attach content to anchor here
    }
}
Load Model Assets

Supports usdz and Reality File
Load Model Assets

Supports usdz and Reality File

Synchronous and asynchronous loading
Load Model Assets

Supports usdz and Reality File
Synchronous and asynchronous loading
Automatically imports
Load Model Assets

Supports usdz and Reality File

Synchronous and asynchronous loading

Automatically imports
- Entity hierarchy
Load Model Assets

Supports usdz and Reality File

Synchronous and asynchronous loading

Automatically imports

• Entity hierarchy
• Meshes
Load Model Assets

Supports usdz and Reality File

Synchronous and asynchronous loading

Automatically imports
• Entity hierarchy
• Meshes
• Materials
Load Model Assets

Supports usdz and Reality File

Synchronous and asynchronous loading

Automatically imports

- Entity hierarchy
- Meshes
- Materials
- Animations
// Load Model Assets

var cardTemplates: [Entity] = []

// Load the model asset for each card
for index in 1...8 {
    let assetName = "memory_card_(index)"
    let cardTemplate = try! Entity.loadModel(named: assetName)
    cardTemplates.append(cardTemplate)
}
// Load Model Assets

var cardTemplates: [Entity] = []

// Load the model asset for each card
for index in 1...8 {
    let assetName = "memory_card_\((index)"
    let cardTemplate = try! Entity.loadModel(named: assetName)
    cardTemplates.append(cardTemplate)
}
Create Cards

Game has sixteen cards
Create Cards

Game has sixteen cards
• Eight card types
Create Cards

Game has sixteen cards

- Eight card types
- Two instances of each type
Create Cards

Game has sixteen cards
• Eight card types
• Two instances of each type

Could call Entity.loadModel() again
Create Cards

Game has sixteen cards
• Eight card types
• Two instances of each type

Could call Entity.loadModel() again

Cloning makes things easier
Cloning Entities

Entity.clone()
Cloning Entities

Entity.clone()

- Creates identical copy
Cloning Entities

Entity.clone()

• Creates identical copy
• References same assets
Cloning Entities

Entity.clone()

• Creates identical copy
• References same assets
• Can clone recursively
Cloning Entities

Entity.clone()

• Creates identical copy
• References same assets
• Can clone recursively
• Clone is a copy, not an instance
// Create Cards

var cards: [Entity] = []

for cardTemplate in cardTemplates {
    // Clone each card template twice
    for _ in 1...2 {
        cards.append(cardTemplate.clone(recursive: true))
    }
}
// Create Cards

var cards: [Entity] = []

for cardTemplate in cardTemplates {
    // Clone each card template twice
    for _ in 1...2 {
        cards.append(cardTemplate.clone(recursive: true))
    }
}
Build the Board

Anchor is the center of play area
Build the Board

Anchor is the center of play area

Cards arranged in 4-by-4 grid
Build the Board

Anchor is the center of play area
Cards arranged in 4-by-4 grid
Add cards to anchor to display in AR
// Build the Board

// Shuffle the cards so they are randomly ordered
cards.shuffle()

// Position the shuffled cards in a 4-by-4 grid
for (index, card) in cards.enumerated() {
    let x = Float(index % 4) - 1.5
    let z = Float(index / 4) - 1.5

    // Set the position of the card
    card.position = [x * 0.1, 0, z * 0.1]

    // Add the card to the anchor
    anchor.addChild(card)
}
// Build the Board

// Shuffle the cards so they are randomly ordered
cards.shuffle()

// Position the shuffled cards in a 4-by-4 grid
for (index, card) in cards.enumerated() {
    let x = Float(index % 4) - 1.5
    let z = Float(index / 4) - 1.5
    // Set the position of the card
    card.position = [x * 0.1, 0, z * 0.1]
    // Add the card to the anchor
    anchor.addChild(card)
}

// Add the card to the anchor
anchor.addChild(card)
// Build the Board

// Shuffle the cards so they are randomly ordered
cards.shuffle()

// Position the shuffled cards in a 4-by-4 grid
for (index, card) in cards.enumerated() {
    let x = Float(index % 4) - 1.5
    let z = Float(index / 4) - 1.5

    // Set the position of the card
    card.position = [x * 0.1, 0, z * 0.1]

    // Add the card to the anchor
    anchor.addChild(card)
}

Adding Interaction

Flip cards when we tap on them
Adding Interaction

Flip cards when we tap on them
Adding Interaction

Flip cards when we tap on them
Need to know what we’re tapping on
Adding Interaction

Flip cards when we tap on them

Need to know what we’re tapping on

RealityKit’s solution — hit testing
Hit Testing

Hit testing turns screen point into ray
Hit Testing

Hit testing turns screen point into ray
Hit Testing

Hit testing turns screen point into ray
Ray is cast into scene
Hit Testing

Hit testing turns screen point into ray
Ray is cast into scene
Returns intersected entities
Hit Testing

Hit testing turns screen point into ray
Ray is cast into scene
Returns intersected entities
ARView provides hit testing methods
- entity(at point:)
- entities(at point:)

Diagram: Camera pointing towards a 3D object
// Hit Testing

@IBAction func onTap(_ sender: UITapGestureRecognizer) {
    let tapLocation = sender.location(in: arView)

    // Get the entity at the location we've tapped, if one exists
    if let card = arView.entity(at: tapLocation) {
        // For testing purposes, print the name of the tapped entity
        print(card.name)

        // Add interaction code here
    }
}
// Hit Testing

@IBAction func onTap(_ sender: UITapGestureRecognizer) {
    let tapLocation = sender.location(in: arView)

    // Get the entity at the location we've tapped, if one exists
    if let card = arView.entity(at: tapLocation) {
        // For testing purposes, print the name of the tapped entity
        print(card.name)

        // Add interaction code here
    }
}
Collision Shapes

To be hit-testable, entities need collision
Collision Shapes

To be hit-testable, entities need collision

Collision shapes are simple geometry
Collision Shapes

To be hit-testable, entities need collision

Collision shapes are simple geometry
• Easy to define
Collision Shapes

To be hit-testable, entities need collision

Collision shapes are simple geometry
- Easy to define
- Efficient for intersection and collision
Collision Shapes

To be hit-testable, entities need collision shapes.

Collision shapes are simple geometry:
- Easy to define
- Efficient for intersection and collision
- Required for hit testing
// Adding Collision Shapes

var cardTemplates: [ModelEntity] = []

// Load the model asset for each card
for index in 1...8 {
    let assetName = "memory_card_(index)"
    let cardTemplate = try! Entity.loadModel(named: assetName)
    cardTemplates.append(cardTemplate)
}
// Adding Collision Shapes

var cardTemplates: [ModelEntity] = []

// Load the model asset for each card
for index in 1...8 {
    let assetName = "memory_card_(index)"
    let cardTemplate = try! Entity.loadModel(named: assetName)

    // Generate collision shapes for the card so we can interact with it
    cardTemplate.generateCollisionShapes(recursive: true)

    // Give the card a name so we'll know what we're interacting with
    cardTemplate.name = assetName

    cardTemplates.append(cardTemplate)
}
// Adding Collision Shapes

var cardTemplates: [ModelEntity] = []

// Load the model asset for each card
for index in 1...8 {
    let assetName = "memory_card_(index)"
    let cardTemplate = try! Entity.loadModel(named: assetName)

    // Generate collision shapes for the card so we can interact with it
    cardTemplate.generateCollisionShapes(recursive: true)

    // Give the card a name so we'll know what we're interacting with
    cardTemplate.name = assetName

    cardTemplates.append(cardTemplate)
}
Animation

Built-in animation support
Animation

Built-in animation support

Transform animation
Animation

Built-in animation support

Transform animation
• Position
Animation

Built-in animation support

Transform animation

• Position
• Rotation
Animation

Built-in animation support

Transform animation

• Position
• Rotation
• Scale
Animation

Built-in animation support

Transform animation
• Position
• Rotation
• Scale

Asset animation
Animation

Built-in animation support

Transform animation
• Position
• Rotation
• Scale

Asset animation
Animation

Built-in animation support

Transform animation
• Position
• Rotation
• Scale

Asset animation

Completion handler
Transform Animation

Timing functions
Transform Animation

Timing functions

- Linear
Transform Animation

Timing functions

• Linear
• Ease in
Transform Animation

Timing functions

• Linear
• Ease in
Transform Animation

Timing functions
- Linear
- Ease in
- Ease out
Transform Animation

Timing functions

• Linear
• Ease in
• Ease out
Transform Animation

Timing functions
- Linear
- Ease in
- Ease out
- Ease in and out
Transform Animation

Timing functions
- Linear
- Ease in
- Ease out
- Ease in and out
Transform Animation

Timing functions
• Linear
• Ease in
• Ease out
• Ease in and out
• Cubic bezier for more customization
// Adding Transform Animation, Flip Face-Up

// Copy card's current transform
var flipUpTransform = card.transform

// Set the card to rotate to π radians (180 degrees)
flipUpTransform.rotation = simd_quatf(angle: .pi, axis: [1, 0, 0])

// Move the card to the new transform over 0.25 seconds
let flipUpController = card.move(to: flipUpTransform,
                                  relativeTo: card.parent,
                                  duration: 0.25,
                                  timingFunction: .easeInOut)
flipUpController.completionHandler {
    // Card is done flipping face-up
}
// Adding Transform Animation, Flip Face-Up

// Copy card's current transform
var flipUpTransform = card.transform

// Set the card to rotate to \( \pi \) radians (180 degrees)
flipUpTransform.rotation = simd_quatf(angle: .pi, axis: [1, 0, 0])

// Move the card to the new transform over 0.25 seconds
let flipUpController = card.move(to: flipUpTransform, relativeTo: card.parent, duration: 0.25, timingFunction: .easeInOut)

flipUpController.completionHandler {
    // Card is done flipping face-up
}
// Adding Transform Animation, Flip Face-Up

// Copy card's current transform
var flipUpTransform = card.transform

// Set the card to rotate to π radians (180 degrees)
flipUpTransform.rotation = simd_quatf(angle: .pi, axis: [1, 0, 0])

// Move the card to the new transform over 0.25 seconds
let flipUpController = card.move(to: flipUpTransform,
                                 relativeTo: card.parent,
                                 duration: 0.25,
                                 timingFunction: .easeInOut)

flipUpController.completionHandler {
    // Card is done flipping face-up
}

// Adding Transform Animation, Flip Face-Up

// Copy card's current transform
var flipUpTransform = card.transform

// Set the card to rotate to π radians (180 degrees)
flipUpTransform.rotation = simd_quatf(angle: .pi, axis: [1, 0, 0])

// Move the card to the new transform over 0.25 seconds
let flipUpController = card.move(to: flipUpTransform,
    relativeTo: card.parent,
    duration: 0.25,
    timingFunction: .easeInOut)

flipUpController.completionHandler {
    // Card is done flipping face-up
}

// Adding Transform Animation, Flip Face-Up

// Copy card's current transform
var flipUpTransform = card.transform

// Set the card to rotate to π radians (180 degrees)
flipUpTransform.rotation = simd_quatf(angle: .pi, axis: [1, 0, 0])

// Move the card to the new transform over 0.25 seconds
let flipUpController = card.move(to: flipUpTransform,
    relativeTo: card.parent,
    duration: 0.25,
    timingFunction: .easeInOut)

flipUpController.completionHandler {
    // Card is done flipping face-up
}
// Adding Transform Animation, Flip Face-Down

// Copy card's current transform
var flipDownTransform = card.transform

// Set the card to rotate back to 0 degrees
flipDownTransform.rotation = simd_quatf(angle: 0, axis: [1, 0, 0])

// Move the card to the new transform over 0.25 seconds
let flipDownController = card.move(to: flipDownTransform,
    relativeTo: card.parent,
    duration: 0.25,
    timingFunction: .easeInOut)

flipDownController.completionHandler {
    // Card is done flipping back to face-down
}
Adding Polish
Advanced Assets
Advanced Assets
Advanced Assets

Can still use Entity.loadModel()
Advanced Assets

Can still use `Entity.loadModel()`

Larger assets take longer to load
Advanced Assets

Can still use Entity.loadModel()

Larger assets take longer to load

App will be blocked during loading
Advanced Assets

Can still use Entity.loadModel()

Larger assets take longer to load

App will be blocked during loading

With lots of assets, load times add up
Asynchronous Loading

Use Entity.loadModelAsync() to unblock app
Asynchronous Loading

Use Entity.loadModelAsync() to unblock app

Assets loaded in background
Asynchronous Loading

Use `Entity.loadModelAsync()` to unblock app

Assets loaded in background

Allows app to continue uninterrupted
Asynchronous Loading

Use Entity.loadModelAsync() to unblock app
Assets loaded in background
Allows app to continue uninterrupted
Receive callback when loading is finished
Asynchronous Loading

Use Entity.loadModelAsync() to unblock app

Assets loaded in background

Allows app to continue uninterrupted

Receive callback when loading is finished

Combine load requests and wait for all
// Asynchronous Loading

// Load a model asynchronously
_ = Entity.loadModelAsync(named: "vintage_car_green")
    .sink { model in
        // Model has been loaded
    }
// Asynchronous Loading

// Load a model asynchronously
_ = Entity.loadModelAsync(named: "vintage_car_green")
    .sink { model in
        // Model has been loaded
    }
// Asynchronous Loading

// Load a model asynchronously
_ = Entity.loadModelAsync(named: "vintage_car_green")
    .sink { model in
        // Model has been loaded
    }
// Asynchronous Loading

// Load two models asynchronously
_ = Entity.loadModelAsync(named: "vintage_car_green")
    .append(Entity.loadModelAsync(named: "vintage_car_yellow"))
    .collect()
    .sink { models in
        // Both models have been loaded
    }
// Asynchronous Loading

// Load two models asynchronously
_ = Entity.loadModelAsync(named: "vintage_car_green")
.append(Entity.loadModelAsync(named: "vintage_car_yellow"))
.collect()
.sink { models in
    // Both models have been loaded
}
// Asynchronous Loading

// Load two models asynchronously
_ = Entity.loadModelAsync(named: "vintage_car_green")
 .append(Entity.loadModelAsync(named: "vintage_car_yellow"))
 .collect()
 .sink { models in
     // Both models have been loaded
 }
// Asynchronous Loading

// Load two models asynchronously
_variant = Entity.loadModelAsync(named: "vintage_car_green")
 .append(Entity.loadModelAsync(named: "vintage_car_yellow"))
 .collect()

.sink { models in
    // Both models have been loaded
}

// Asynchronous Loading

// Load all eight models asynchronously
_ = Entity.loadModelAsync(named: "vintage_car_green")
    .append(Entity.loadModelAsync(named: "vintage_car_yellow"))
    .append(Entity.loadModelAsync(named: "vintage_robot_blue"))
    .append(Entity.loadModelAsync(named: "vintage_robot_red"))
    .append(Entity.loadModelAsync(named: "vintage_drummer_red"))
    .append(Entity.loadModelAsync(named: "vintage_drummer_green"))
    .append(Entity.loadModelAsync(named: "vintage_plane_green"))
    .append(Entity.loadModelAsync(named: "vintage_plane_yellow"))
    .collect()
    .sink { models in
        // All models have been loaded
    }

// Asynchronous Loading

// Load all eight models asynchronously
_ = Entity.loadModelAsync(named: "vintage_car_green")
 .append(Entity.loadModelAsync(named: "vintage_car_yellow"))
 .append(Entity.loadModelAsync(named: "vintage_robot_blue"))
 .append(Entity.loadModelAsync(named: "vintage_robot_red"))
 .append(Entity.loadModelAsync(named: "vintage_drummer_red"))
 .append(Entity.loadModelAsync(named: "vintage_drummer_green"))
 .append(Entity.loadModelAsync(named: "vintage_plane_green"))
 .append(Entity.loadModelAsync(named: "vintage_plane_yellow"))
 .collect()
 .sink { models in
    // All models have been loaded
}
// Asynchronous Loading

// Load all eight models asynchronously
_ = Entity.loadModelAsync(named: "vintage_car_green")
    .append(Entity.loadModelAsync(named: "vintage_car_yellow"))
    .append(Entity.loadModelAsync(named: "vintage_robot_blue"))
    .append(Entity.loadModelAsync(named: "vintage_robot_red"))
    .append(Entity.loadModelAsync(named: "vintage_drummer_red"))
    .append(Entity.loadModelAsync(named: "vintage_drummer_green"))
    .append(Entity.loadModelAsync(named: "vintage_plane_green"))
    .append(Entity.loadModelAsync(named: "vintage_plane_yellow"))
    .collect()

    .sink { models in
            // All models have been loaded
        }
Synchronous vs. Asynchronous
Synchronous vs. Asynchronous
Occlusion

Reveals video passthrough

Simulates real world objects
Occlusion

Reveals video passthrough

Simulates real world objects
// Adding Occlusion Plane

// Create plane mesh, 0.5 meters wide & 0.5 meters deep
let planeMesh = MeshResource.generatePlane(width: 0.5, depth: 0.5)

// Create occlusion material
let material = OcclusionMaterial()

// Create ModelEntity using mesh and materials
let occlusionPlane = ModelEntity(mesh: planeMesh, materials: [material])

// Position plane below game board
occlusionPlane.position.y = -0.001

// Add to anchor
anchor.addChild(occlusionPlane)
// Adding Occlusion Plane

// Create plane mesh, 0.5 meters wide & 0.5 meters deep
let planeMesh = MeshResource.generatePlane(width: 0.5, depth: 0.5)

// Create occlusion material
let material = OcclusionMaterial()

// Create ModelEntity using mesh and materials
let occlusionPlane = ModelEntity(mesh: planeMesh, materials: [material])

// Position plane below game board
occlusionPlane.position.y = -0.001

// Add to anchor
anchor.addChild(occlusionPlane)
// Adding Occlusion Plane

// Create plane mesh, 0.5 meters wide & 0.5 meters deep
let planeMesh = MeshResource.generatePlane(width: 0.5, depth: 0.5)

// Create occlusion material
let material = OcclusionMaterial()

// Create ModelEntity using mesh and materials
let occlusionPlane = ModelEntity(mesh: planeMesh, materials: [material])

// Position plane below game board
occlusionPlane.position.y = -0.001

// Add to anchor
anchor.addChild(occlusionPlane)
// Adding Occlusion Plane

// Create plane mesh, 0.5 meters wide & 0.5 meters deep
let planeMesh = MeshResource.generatePlane(width: 0.5, depth: 0.5)

// Create occlusion material
let material = OcclusionMaterial()

// Create ModelEntity using mesh and materials
let occlusionPlane = ModelEntity(mesh: planeMesh, materials: [material])

// Position plane below game board
occlusionPlane.position.y = -0.001

// Add to anchor
anchor.addChild(occlusionPlane)
// Adding Occlusion Plane

// Create plane mesh, 0.5 meters wide & 0.5 meters deep
let planeMesh = MeshResource.generatePlane(width: 0.5, depth: 0.5)

// Create occlusion material
let material = OcclusionMaterial()

// Create ModelEntity using mesh and materials
let occlusionPlane = ModelEntity(mesh: planeMesh, materials: [material])

// Position plane below game board
occlusionPlane.position.y = -0.001

// Add to anchor
anchor.addChild(occlusionPlane)
Adding Occlusion Box

Create box mesh, 0.5 meters on all sides

```swift
let boxSize: Float = 0.5
let boxMesh = MeshResource.generateBox(size: boxSize)
```

Create Occlusion Material

```swift
let material = OcclusionMaterial()
```

Create ModelEntity using mesh and materials

```swift
let occlusionBox = ModelEntity(mesh: boxMesh, materials: [material])
```

Position box with top slightly below game board

```swift
occlusionBox.position.y = -boxSize / 2 - 0.001
```

Add to anchor

```swift
anchor.addChild(occlusionBox)
```
// Adding Occlusion Box

// Create box mesh, 0.5 meters on all sides
let boxSize: Float = 0.5
let boxMesh = MeshResource.generateBox(size: boxSize)

// Create Occlusion Material
let material = OcclusionMaterial()

// Create ModelEntity using mesh and materials
let occlusionBox = ModelEntity(mesh: boxMesh, materials: [material])

// Position box with top slightly below game board
occlusionBox.position.y = -boxSize / 2 - 0.001

// Add to anchor
anchor.addChild(occlusionBox)
// Adding Occlusion Box

// Create box mesh, 0.5 meters on all sides
let boxSize: Float = 0.5
let boxMesh = MeshResource.generateBox(size: boxSize)

// Create Occlusion Material
let material = OcclusionMaterial()

// Create ModelEntity using mesh and materials
let occlusionBox = ModelEntity(mesh: boxMesh, materials: [material])

// Position box with top slightly below game board
occlusionBox.position.y = -boxSize / 2 - 0.001

// Add to anchor
anchor.addChild(occlusionBox)
// Adding Occlusion Box

// Create box mesh, 0.5 meters on all sides
let boxSize: Float = 0.5
let boxMesh = MeshResource.generateBox(size: boxSize)

// Create Occlusion Material
let material = OcclusionMaterial()

// Create ModelEntity using mesh and materials
let occlusionBox = ModelEntity(mesh: boxMesh, materials: [material])

// Position box with top slightly below game board
occlusionBox.position.y = -boxSize / 2 - 0.001

// Add to anchor
anchor.addChild(occlusionBox)
// Adding Occlusion Box

// Create box mesh, 0.5 meters on all sides
let boxSize: Float = 0.5
let boxMesh = MeshResource.generateBox(size: boxSize)

// Create Occlusion Material
let material = OcclusionMaterial()

// Create ModelEntity using mesh and materials
let occlusionBox = ModelEntity(mesh: boxMesh, materials: [material])

// Position box with top slightly below game board
occlusionBox.position.y = -boxSize / 2 - 0.001

// Add to anchor
anchor.addChild(occlusionBox)
Tracking Game State
Entity and Component

Composition over inheritance
Promotes reuse
Flexible and scalable
Entity and Component

Composition over inheritance
Promotes reuse
Flexible and scalable
Entity and Component

Model
Entity
Entity and Component
Entity and Component

Model

Collision
Entity and Component

Model
Collision
Physics
Entity and Component

Model
Collision
Physics
Entity and Component

Remove Physics
Entity and Component

Remove Physics

Card properties
• Hidden/Revealed
Entity and Component

Remove Physics

Card properties
- Hidden/Revealed
- Kind
Entity and Component

Remove Physics

Card properties
- Hidden/Revealed
- Kind
Custom Component

Struct containing your properties
Custom Component

Struct containing your properties

Conforms to Component protocol
Custom Component

- Struct containing your properties
- Conforms to Component protocol
- Attach data to Entity
Custom Component

Struct containing your properties
Conforms to Component protocol
Attach data to Entity
Optionally conforms to Codable
// Declare Card Component
struct CardComponent: Component, Codable {
    var revealed = false
    var kind = ""
}

// Load a model
let entity = try! Entity.loadModel(named: "memory_card_1")

// Remove Physics Body Component
entity.physicsBody = nil

// Add Card Component
entity.components[CardComponent.self] = CardComponent()

// Modify kind property
entity.components[CardComponent.self]?.kind = "memory_card_1"
// Declare Card Component
struct CardComponent: Component, Codable {
    var revealed = false
    var kind = ""
}

// Load a model
let entity = try! Entity.loadModel(named: "memory_card_1")

// Remove Physics Body Component
entity.physicsBody = nil

// Add Card Component
entity.components[CardComponent.self] = CardComponent()

// Modify kind property
entity.components[CardComponent.self]?.kind = "memory_card_1"
// Declare Card Component
struct CardComponent: Component, Codable {
    var revealed = false
    var kind = ""
}

// Load a model
let entity = try! Entity.loadModel(named: "memory_card_1")

// Remove Physics Body Component
entity.physicsBody = nil

// Add Card Component
entity.components[CardComponent.self] = CardComponent()

// Modify kind property
entity.components?[CardComponent.self]!.kind = "memory_card_1"
// Declare Card Component
struct CardComponent: Component, Codable {
    var revealed = false
    var kind = ""
}

// Load a model
let entity = try! Entity.loadModel(named: "memory_card_1")

// Remove Physics Body Component
entity.physicsBody = nil

// Add Card Component
entity.components[CardComponent.self] = CardComponent()

// Modify kind property
entity.components[CardComponent.self]?.kind = "memory_card_1"
// Declare Card Component

struct CardComponent: Component, Codable {
    var revealed = false
    var kind = ""
}

// Load a model
let entity = try! Entity.loadModel(named: "memory_card_1")

// Remove Physics Body Component
entity.physicsBody = nil

// Add Card Component
entity.components[CardComponent.self] = CardComponent()

// Modify kind property
entity.components[CardComponent.self]!.kind = "memory_card_1"
// Declare Card Component
struct CardComponent: Component, Codable {
    var revealed = false
    var kind = ""
}

// Load a model
let entity = try! Entity.loadModel(named: "memory_card_1")

// Remove Physics Body Component
entity.physicsBody = nil

// Add Card Component
entity.components[CardComponent.self] = CardComponent()

// Modify kind property
entity.components[CardComponent.self]?.kind = "memory_card_1"
Custom Entity

Common configurations
Custom Entity

Common configurations

Compile-time type checking
Custom Entity

Common configurations
Compile-time type checking
Encapsulate functionality
Custom Entity

Common configurations
Compile-time type checking
Encapsulate functionality
Updates to multiple components
Creating a Custom Entity

Create class derived from Entity
Creating a Custom Entity

Create class derived from Entity
Creating a Custom Entity

Create class derived from Entity

Include RealityKit components
Creating a Custom Entity

Create class derived from Entity

Include RealityKit components
Creating a Custom Entity

Create class derived from Entity

Include RealityKit components

Add property for custom component
Creating a Custom Entity

Create class derived from Entity

Include RealityKit components

Add property for custom component

Extend with methods

Card

Entity

Model

Collision

Card

func reveal()

func hide()
Create Entity Class

// Declare custom entity
class CardEntity: Entity, HasModel, HasCollision {

    // Card property for convenient access to card state
    public var card: CardComponent {
        get { return components[CardComponent.self] ?? CardComponent() }
        set { components[CardComponent.self] = newValue }
    }
}

Create Entity Class

```swift
// Declare custom entity
class CardEntity: Entity, HasModel, HasCollision {

    // Card property for convenient access to card state
    public var card: CardComponent {
        get { return components[CardComponent.self] ?? CardComponent() }
        set { components[CardComponent.self] = newValue }
    }
}
```
Create Entity Class

```swift
// Declare custom entity
class CardEntity: Entity, HasModel, HasCollision {

    // Card property for convenient access to card state
    public var card: CardComponent {
        get { return components[CardComponent.self] ?? CardComponent() }
        set { components[CardComponent.self] = newValue }
    }
}
```
## Create Entity Class

```swift
// Declare custom entity
class CardEntity: Entity, HasModel, HasCollision {

  // Card property for convenient access to card state
  public var card: CardComponent {
    get { return components[CardComponent.self] ?? CardComponent() }
    set { components[CardComponent.self] = newValue }
  }
}
```
// Declare custom entity
class CardEntity: Entity, HasModel, HasCollision {

    // Card property for convenient access to card state
    public var card: CardComponent {
        get { return components[CardComponent.self] ?? CardComponent() }
        set { components[CardComponent.self] = newValue }
    }
}
// Extend CardEntity with additional methods
extension CardEntity {

    // Animate, change state
    func reveal() {

        // Update revealed property
        card.revealed = true

        // Flip card over to reveal contents
        var transform = self.transform
        transform.rotation = simd_quatf(angle: .pi, axis: [1, 0, 0])
        move(to: transform, relativeTo: parent, duration: 0.25, timingFunction: .easeInOut)
    }
}
// Extend CardEntity with additional methods
extension CardEntity {

    // Animate, change state
    func reveal() {

        // Update revealed property
        card.revealed = true

        // Flip card over to reveal contents
        var transform = self.transform
        transform.rotation = simd_quatf(angle: .pi, axis: [1, 0, 0])
        move(to: transform, relativeTo: parent, duration: 0.25, timingFunction: .easeInOut)
    }
}
// Extend CardEntity with additional methods
extension CardEntity {

    // Animate, change state
    func reveal() {

        // Update revealed property
        card.revealed = true

        // Flip card over to reveal contents
        var transform = self.transform
        transform.rotation = simd_quatf(angle: .pi, axis: [1, 0, 0])
        move(to: transform, relativeTo: parent, duration: 0.25, timingFunction: .easeInOut)
    }
}
// Extend CardEntity with additional methods
extension CardEntity {

    // Animate, change state
    func reveal() {

        // Update revealed property
        card.revealed = true

        // Flip card over to reveal contents
        var transform = self.transform
        transform.rotation = simd_quatf(angle: .pi, axis: [1, 0, 0])
        move(to: transform, relativeTo: parent, duration: 0.25, timingFunction: .easeInOut)
    }
}
/ Extend CardEntity with additional methods

extension CardEntity {

    // Animate, change state
    func reveal() {

        // Update revealed property
        card.revealed = true

        // Flip card over to reveal contents
        var transform = self.transform
        transform.rotation = simd_quatf(angle: .pi, axis: [1, 0, 0])
        move(to: transform, relativeTo: parent, duration: 0.25, timingFunction: .easeInOut)
    }
}
// Tap handler using CardEntity

@IBAction func onTap(_ sender: UITapGestureRecognizer) {

    // Entity under cursor, if it’s a CardEntity
    if let cardEntity = arView.entity(at: sender.location(in: arView)) as? CardEntity {

        // Check card’s revealed state
        if cardEntity.card.revealed {

            // Hide card
            cardEntity.hide()
        } else {

            // Reveal card
            cardEntity.reveal()
        }
    }
}
// Tap handler using CardEntity

@IBAction func onTap(_ sender: UITapGestureRecognizer) {

    // Entity under cursor, if it’s a CardEntity
    if let cardEntity = arView.entity(at: sender.location(in: arView)) as? CardEntity {

        // Check card’s revealed state
        if cardEntity.card.revealed {

            // Hide card
            cardEntity.hide()
        } else {

            // Reveal card
            cardEntity.reveal()
        }
    }
}
// Tap handler using CardEntity

@IBAction func onTap(_ sender: UITapGestureRecognizer) {

    // Entity under cursor, if it’s a CardEntity
    if let cardEntity = arView.entity(at: sender.location(in: arView)) as? CardEntity {

        // Check card’s revealed state
        if cardEntity.card.revealed {

            // Hide card
            cardEntity.hide()
        } else {

            // Reveal card
            cardEntity.reveal()
        }
    }
}
// Tap handler using CardEntity
@IBAction func onTap(_ sender: UITapGestureRecognizer) {

    // Entity under cursor, if it’s a CardEntity
    if let cardEntity = arView.entity(at: sender.location(in: arView)) as? CardEntity {
        // Check card’s revealed state
        if cardEntity.card.revealed {
            // Hide card
            cardEntity.hide()
        } else {
            // Reveal card
            cardEntity.reveal()
        }
    }
}
// Tap handler using CardEntity
@IBAction func onTap(_ sender: UITapGestureRecognizer) {

    // Entity under cursor, if it’s a CardEntity
    if let cardEntity = arView.entity(at: sender.location(in: arView)) as? CardEntity {

        // Check card’s revealed state
        if cardEntity.card.revealed {

            // Hide card
            cardEntity.hide()
        } else {

            // Reveal card
            cardEntity.reveal()
        }
    }
}
// Tap handler using CardEntity

@IBAction func onTap(_ sender: UITapGestureRecognizer) {

    // Entity under cursor, if it's a CardEntity
    if let cardEntity = arView.entity(at: sender.location(in: arView)) as? CardEntity {

        // Check card’s revealed state
        if cardEntity.card.revealed {

            // Hide card
            cardEntity.hide()
        } else {

            // Reveal card
            cardEntity.reveal()
        }
    }
}
“Just add multiplayer...”
Augmented Reality Mapping
Augmented Reality Mapping
Multiplayer Overview

Automatic scene synchronization
Multiplayer Overview

Automatic scene synchronization

Built on MultipeerConnectivity
Multiplayer Overview

Automatic scene synchronization
Built on MultipeerConnectivity
Easy to use ownership model
Multiplayer Overview

Automatic scene synchronization
Built on MultipeerConnectivity
Easy to use ownership model
Optimized for AR
Adopting Multiplayer

Designate host, client
Adopting Multiplayer

Designate host, client
Adopting Multiplayer

Designate host, client

Establish connection
Adopting Multiplayer

Designate host, client

Establish connection

Enable Collaborative Session
Adopting Multiplayer

Designate host, client
Establish connection
Enable Collaborative Session
Place synchronized Anchor
Adopting Multiplayer

Designate host, client
Establish connection
Enable Collaborative Session
Place synchronized Anchor
Manage ownership
// MultipeerConnectivity Session Setup
import MultipeerConnectivity

// Create Multipeer Session
let myPeerID = MCPeerID(displayName: "Memory Game")
let mcSession = MCSession(peer: myPeerID, securityIdentity: nil,
                            encryptionPreference: .required)

// Advertise or Browse, depending on role
if role == .host {
    // Host Creates MCNearbyServiceAdvertiser and Starts Advertising
} else {
    // Client Creates MCNearbyServiceBrowser and Starts Browsing
}

// Use Multipeer session to Synchronize RealityKit scene
arView.scene.synchronizationService = try? MultipeerConnectivityService(session: mcSession)
// MultipeerConnectivity Session Setup
import MultipeerConnectivity

// Create Multipeer Session
let myPeerID = MCPeerID(displayName: "Memory Game")
let mcSession = MCSession(peer: myPeerID, securityIdentity: nil,
                          encryptionPreference: .required)

// Advertise or Browse, depending on role
if role == .host {
    // Host Creates MCNearbyServiceAdvertiser and Starts Advertising
} else {
    // Client Creates MCNearbyServiceBrowser and Starts Browsing
}

// Use Multipeer session to Synchronize RealityKit scene
arView.scene.synchronizationService = try? MultipeerConnectivityService(session: mcSession)
// MultipeerConnectivity Session Setup
import MultipeerConnectivity

// Create Multipeer Session
let myPeerID = MCPeerID(displayName: "Memory Game")
let mcSession = MCSession(peer: myPeerID, securityIdentity: nil,
                           encryptionPreference: .required)

// Advertise or Browse, depending on role
if role == .host {
    // Host Creates MCNearbyServiceAdvertiser and Starts Advertising
} else {
    // Client Creates MCNearbyServiceBrowser and Starts Browsing
}

// Use Multipeer session to Synchronize RealityKit scene
arView.scene.synchronizationService = try? MultipeerConnectivityService(session: mcSession)
/** MultipeerConnectivity Session Setup**

```swift
import MultipeerConnectivity

// Create Multipeer Session
let myPeerID = MCPeerID(displayName: "Memory Game")
let mcSession = MCSession(peer: myPeerID, securityIdentity: nil,
                            encryptionPreference: .required)

// Advertise or Browse, depending on role
if role == .host {
    // Host Creates MCNearbyServiceAdvertiser and Starts Advertising
}
else {
    // Client Creates MCNearbyServiceBrowser and Starts Browsing
}

// Use Multipeer session to Synchronize RealityKit scene
arView.scene.synchronizationService = try? MultipeerConnectivityService(session: mcSession)
```
// MultipeerConnectivity Session Setup
import MultipeerConnectivity

// Create Multipeer Session
let myPeerID = MCPeerID(displayName: "Memory Game")
let mcSession = MCSession(peer: myPeerID, securityIdentity: nil,
                           encryptionPreference: .required)

// Advertise or Browse, depending on role
if role == .host {
    // Host Creates MCNearbyServiceAdvertiser and Starts Advertising
} else {
    // Client Creates MCNearbyServiceBrowser and Starts Browsing
}

// Use Multipeer session to Synchronize RealityKit scene
arView.scene.synchronizationService = try? MultipeerConnectivityService(session: mcSession)
AR Collaborative Session

Introduced in ARKit 3

Enable on AR Session
// Create a new tracking configuration
let config = ARWorldTrackingConfiguration()

// Enable collaboration
config.isCollaborationEnabled = true

// Instruct ARKit to use the configuration
arView.session.run(config, options: [])
// Create a new tracking configuration
let config = ARWorldTrackingConfiguration()

// Enable collaboration
config.isCollaborationEnabled = true

// Instruct ARKit to use the configuration
arView.session.run(config, options: [])
// Create a new tracking configuration
let config = ARWorldTrackingConfiguration()

// Enable collaboration
config.isCollaborationEnabled = true

// Instruct ARKit to use the configuration
arView.session.run(config, options: [])
// Create a new tracking configuration
let config = ARWorldTrackingConfiguration()

// Enable collaboration
config.isCollaborationEnabled = true

// Instruct ARKit to use the configuration
arView.session.run(config, options: []
}
Synchronized Anchor

Allow player to choose

Same world location
Synchronized Anchor

Allow player to choose
Same world location
// Host - Tap to place board

@IBAction func onTap(_ sender: UITapGestureRecognizer) {
    // Find position under cursor
    guard let result = arView.raycast(sender.location(in: arView),
                                       allowing: .existingPlaneGeometry, alignment: .horizontal).first else {
        return
    }
}
// Host - Tap to place board
@IBAction func onTap(_ sender: UITapGestureRecognizer) {
    // Find position under cursor
    guard let result = arView.raycast(sender.location(in: arView),
        allowing: .existingPlaneGeometry, alignment: .horizontal).first else {
        return
    }
}
// Host - Tap to place board
@IBAction func onTap(_ sender: UITapGestureRecognizer) {
    // Find position under cursor
    guard let result = arView.raycast(sender.location(in: arView),
                                       allowing: .existingPlaneGeometry, alignment: .horizontal).first else {
        return
    }

    // Create ARKit ARAnchor and add to ARSession
    let arAnchor = ARAnchor(name: "Memory Game Board", transform: result.worldTransform)
    arView.session.add(anchor: arAnchor)

    // Create a RealityKit AnchorEntity and add to the scene
    let anchorEntity = AnchorEntity(anchoring: arAnchor)
    arView.scene.addAnchor(anchoring: anchorEntity)

    // Add the game board to the scene here
}
// Host - Tap to place board
@IBAction func onTap(_ sender: UITapGestureRecognizer) {
    // Find position under cursor
    guard let result = arView.raycast(sender.location(in: arView),
      allowing: .existingPlaneGeometry, alignment: .horizontal).first else {
        return
    }

    // Create ARKit ARAnchor and add to ARSession
    let arAnchor = ARAnchor(name: "Memory Game Board", transform: result.worldTransform)
    arView.session.add(anchor: arAnchor)

    // Create a RealityKit AnchorEntity and add to the scene
    let anchorEntity = AnchorEntity(ANCHOR: arAnchor)
    arView.scene.addAnchor(ANCHOR: anchorEntity)

    // Add the game board to the scene here
}
Host - Tap to place board

@IBAction func onTap(_ sender: UITapGestureRecognizer) {
    // Find position under cursor
    guard let result = arView.raycast(sender.location(in: arView),
            allowing: .existingPlaneGeometry, alignment: .horizontal).first else {
        return
    }

    // Create ARKit ARAnchor and add to ARSession
    let arAnchor = ARAnchor(name: "Memory Game Board", transform: result.worldTransform)
    arView.session.add(anchor: arAnchor)

    // Create a RealityKit AnchorEntity and add to the scene
    let anchorEntity = AnchorEntity(anchor: arAnchor)
    arView.scene.addAnchor(anchorEntity)

    // Add the game board to the scene here
}
// Host - Tap to place board

@IBAction func onTap(_ sender: UITapGestureRecognizer) {
    // Find position under cursor
    guard let result = arView.raycast(sender.location(in: arView),
                                       allowing: .existingPlaneGeometry, alignment: .horizontal).first else {
        return
    }

    // Create ARKit ARAnchor and add to ARSession
    let arAnchor = ARAnchor(name: "Memory Game Board", transform: result.worldTransform)
    arView.session.add(anchor: arAnchor)

    // Create a RealityKit AnchorEntity and add to the scene
    let anchorEntity = AnchorEntity(anchor: arAnchor)
    arView.scene.addAnchor(anchorEntity)

    // Add the game board to the scene here
}

Entity Ownership

Right to modify an Entity

Host

Client
Entity Ownership

Right to modify an Entity

One owner at a time
Entity Ownership

Right to modify an Entity
One owner at a time
Defaults to Entity creator
Entity Ownership

Right to modify an Entity
One owner at a time
Defaults to Entity creator
Transferrable
Entity Ownership

Right to modify an Entity
One owner at a time
Defaults to Entity creator
Transferrable
Configurable
Entity Ownership

Host

Client
Entity Ownership

Host

Card

Create

Card

Client

Card

Create

Card
Entity Ownership

Host

Card

Card

Client

Card

Update

Card
Entity Ownership

Host

Client

Card

Card

Update
Ownership Transfer
Ownership Transfer

Host

Request

Client

Card

Card

Card
Ownership Transfer

Request → Accept

Host

Client

Card

Card

Card
// Client asks for ownership to reveal card

// Request ownership

```swift
card.requestOwnership { result in

    // Test if ownership was granted
    if result == .granted {

        // Reveal card
        card.reveal()

    } else {

        // Allow player to select a different card

    }
```

```
// Client asks for ownership to reveal card

// Request ownership
card.requestOwnership { result in

  // Test if ownership was granted
  if result == .granted {

    // Reveal card
    card.reveal()

  } else {
    // Allow player to select a different card

  }

}
// Client asks for ownership to reveal card

// Request ownership
card.requestOwnership { result in

  // Test if ownership was granted
  if result == .granted {

    // Reveal card
    card.reveal()

  } else {
    // Allow player to select a different card
  }
}
// Client asks for ownership to reveal card

// Request ownership
card.requestOwnership { result in

    // Test if ownership was granted
    if result == .granted {

        // Reveal card
        card.reveal()

    } else {

        // Allow player to select a different card
    }
}
// Client asks for ownership to reveal card

// Request ownership
card.requestOwnership { result in

    // Test if ownership was granted
    if result == .granted {

        // Reveal card
        card.reveal()

    } else {

        // Allow player to select a different card

    }

}
// Reveal card and decline transfer requests
extension CardEntity {

    // Animate, change state
    func reveal() {

        // Update revealed property
        card.revealed = true

        // Don’t automatically accept ownership requests
        synchronization?.ownershipTransferMode = .manual

        // Flip card over to reveal contents
    }
}
// Reveal card and decline transfer requests
extension CardEntity {

// Animate, change state
func reveal() {

    // Update revealed property
card.revealed = true

    // Don’t automatically accept ownership requests
    synchronization?.ownershipTransferMode = .manual

    // Flip card over to reveal contents
}
}
// Reveal card and decline transfer requests
extension CardEntity {

    // Animate, change state
    func reveal() {

        // Update revealed property
        card.revealed = true

        // Don’t automatically accept ownership requests
        synchronization?.ownershipTransferMode = .manual

        // Flip card over to reveal contents
    }
}
extension CardEntity {

    // Animate, change state
    func reveal() {
        // Update revealed property
        card.revealed = true

        // Don’t automatically accept ownership requests
        synchronization?.ownershipTransferMode = .manual

        // Flip card over to reveal contents
    }
}
// Hide card and accept transfer requests
extension CardEntity {

    // Animate, change state
    func hide() {

        // Update revealed property
        card.revealed = false

        // Automatically accept transfer requests
        synchronization?.ownershipTransferMode = .autoAccept

        // Flip card over to hide contents
    }
}
// Hide card and accept transfer requests
extension CardEntity {

  // Animate, change state
  func hide() {

    // Update revealed property
    card.revealed = false

    // Automatically accept transfer requests
    synchronization?.ownershipTransferMode = .autoAccept

    // Flip card over to hide contents
  }
}
// Hide card and accept transfer requests
extension CardEntity {

// Animate, change state
func hide() {

    // Update revealed property
card.revealed = false

    // Automatically accept transfer requests
    synchronization?.ownershipTransferMode = .autoAccept

    // Flip card over to hide contents
}
}
Selection Indication
Local-Only Entities

Owner

Peer
Local-Only Entities

State only visible to local client

Owner

Peer
Local-Only Entities

State only visible to local client

Remove synchronization
Local-Only Entities

State only visible to local client
Remove synchronization
Children are also unshared
// Add local only selection indicator when revealing
extension CardEntity {
    func reveal() {
        card.revealed = true
        synchronization?.ownershipTransferMode = .manual
    }
}
// Add local only selection indicator when revealing
extension CardEntity {
    func reveal() {
        card.revealed = true
        synchronization?.ownershipTransferMode = .manual

        // Create local-only selection indicator
        let selection = SelectionEntity()
        selection.position.y = 0.1

        // Remove synchronization component
        selection.synchronization = nil

        // Add as child
        addChild(selection)
    }
}
// Add local only selection indicator when revealing
extension CardEntity {
    func reveal() {
        card.revealed = true
        synchronization?.ownershipTransferMode = .manual

        // Create local-only selection indicator
        let selection = SelectionEntity()
        selection.position.y = 0.1

        // Remove synchronization component
        selection.synchronization = nil

        // Add as child
        addChild(selection)
    }
}
// Add local only selection indicator when revealing
extension CardEntity {
    func reveal() {
        card.revealed = true
        synchronization?.ownershipTransferMode = .manual

        // Create local-only selection indicator
        let selection = SelectionEntity()
        selection.position.y = 0.1

        // Remove synchronization component
        selection.synchronization = nil

        // Add as child
        addChild(selection)
    }
}
/ Add local only selection indicator when revealing

extension CardEntity {
    func reveal() {
        card.revealed = true
        synchronization?.ownershipTransferMode = .manual

        // Create local-only selection indicator
        let selection = SelectionEntity()
        selection.position.y = 0.1

        // Remove synchronization component
        selection.synchronization = nil

        // Add as child
        addChild(selection)
    }
}

// Remove selection indicator on hide
extension CardEntity {
    func hide() {
        card.revealed = false
        synchronization?.ownershipTransferMode = .autoAccept
    }
}
/ Remove selection indicator on hide
definition CardEntity { 
    func hide() {
        card.revealed = false
        synchronization?.ownershipTransferMode = .autoAccept

        // Iterate children looking for Selection Entity
        for child in children where child is SelectionEntity {
            // Remove child and exit loop
            child.removeFromParent()
            break
        }
    }
}
// Remove selection indicator on hide
extension CardEntity {
  func hide() {
    card.revealed = false
    synchronization?.ownershipTransferMode = .autoAccept

    // Iterate children looking for Selection Entity
    for child in children where child is SelectionEntity {
      // Remove child and exit loop
      child.removeFromParent()
      break
    }
  }
}
// Remove selection indicator on hide
extension CardEntity {
    func hide() {
        card.revealed = false
        synchronization?.ownershipTransferMode = .autoAccept

        // Iterate children looking for Selection Entity
        for child in children where child is SelectionEntity {
            // Remove child and exit loop
            child.removeFromParent()
            break
        }
    }
}
Host

Client
Summary

Anchoring
Summary

Anchoring

Asset loading
Summary

Anchoring
Asset loading
Interaction
Summary

Anchoring
Asset loading
Interaction
Custom components and entities
Summary

Anchoring
Asset loading
Interaction
Custom components and entities
Multiplayer
# More Information


<table>
<thead>
<tr>
<th>Session Title</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introducing RealityKit and Reality Composer</td>
<td>Tuesday, 4:00</td>
</tr>
<tr>
<td>Introducing ARKit 3</td>
<td>Tuesday, 5:00</td>
</tr>
</tbody>
</table>
## More Information

developer.apple.com/wwdc19/605

<table>
<thead>
<tr>
<th>Event</th>
<th>Date/Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>RealityKit and Reality Composer Lab</td>
<td>Wednesday, 12:00</td>
</tr>
<tr>
<td>RealityKit and Reality Composer Lab</td>
<td>Thursday, 3:00</td>
</tr>
</tbody>
</table>