Sample Code

Placing objects and handling 3D interaction

Place virtual content at tracked, real-world locations, and enable the user to interact with virtual content by using gestures.

Download

iOS 13.0+ | iPadOS 13.0+ | Xcode 16.0+

Overview

The key facet of an AR experience is the ability to intermix virtual and real-world objects. A flat surface is the optimum location for setting a virtual object. To assist ARKit with finding surfaces, you tell the user to move their device in ways that help ARKit prepare the experience. ARKit provides a view that tailors its instructions to the user, guiding them to the surface that your app needs.

To enable the user to put a virtual item on the real-world surface when they tap the screen, ARKit incorporates ray casting, which provides a 3D location in physical space that corresponds to the screen's touch location. When the user rotates or otherwise moves the virtual items they place, you respond to the respective touch gestures and correlate that input to the virtual content's look in the physical environment.

> **Note**
>
> ARKit requires a device with an A9 or later processor. ARKit is not available in iOS Simulator.

Set a goal to coach the user's movement

To enable your app to detect real-world surfaces, you use a world tracking configuration. For ARKit to establish tracking, the user must physically move their device to allow ARKit to get a sense of perspective. To communicate this need to the user, you use a view provided by ARKit that presents the user with instructional diagrams and verbal guidance, called `ARCoachingOverlayView` For example, when you start the app, the first thing the user sees is a message and animation from the coaching overlay telling them to move their device left and right, repeatedly, in order to get started.

To enable the user to place virtual content on a horizontal surface, you set the coaching overlay goal accordingly.

```
func setGoal() {
    coachingOverlay.goal = .horizontalPlane
}
```

The coaching overlay then tailors its instructions according to the goal you choose. After ARKit gets a sense of perspective, the coaching overlay instructs the user to find a surface.

Respond to coaching events

To make sure the coaching overlay provides guidance to the user whenever ARKit determines it's necessary, you set `activatesAutomatically` to `true`.

```
func setActivatesAutomatically() {
    coachingOverlay.activatesAutomatically = true
}
```

The coaching overlay activates automatically when the app starts, or when tracking degrades past a certain threshold. In those situations, ARKit notifies your delegate by calling `coachingOverlayViewWillActivate(_:)`. In response to this event, hide your app's UI to enable the user to focus on the instructions that the coaching overlay provides.

```
func coachingOverlayViewWillActivate(_ coachingOverlayView: ARCoachingOverlayView) {
    upperControlsView.isHidden = true
}
```

When the coaching overlay determines that the goal has been met, it disappears from the user's view. ARKit notifies your delegate that the coaching process has ended, which is when you show your app's main user interface.

```
func coachingOverlayViewDidDeactivate(_ coachingOverlayView: ARCoachingOverlayView)
    upperControlsView.isHidden = false
}
```

Place virtual content

To give the user an idea of where they can place virtual content, annotate the environment to give them a preview. The sample app draws a square that gives the user visual confirmation of the shape and alignment of the surfaces that ARKit is aware of.

To figure out where to put the square in the real world, you use an `ARRaycastQuery` to ask ARKit where any surfaces exist in the real world. First, you create a ray-cast query that defines the 2D point on the screen you're interested in. Because the focus square is aligned with the center of the screen, you create a query for the screen center.

```
func getRaycastQuery(for alignment: ARRaycastQuery.TargetAlignment = .any) -> ARRayc
    return raycastQuery(from: screenCenter, allowing: .estimatedPlane, alignment: al
}
```

Then, you execute the ray-cast query by asking the session to cast it.

```
func castRay(for query: ARRaycastQuery) -> [ARRaycastResult] {
    return session.raycast(query)
}
```

ARKit returns a position in the `results` parameter that includes the depth of where that point lies on a surface in the real world. To give the user a preview of where on the real-world surface a user can place their virtual content, update the focus square's position using the ray-cast result's `worldTransform`:

```
func setPosition(with raycastResult: ARRaycastResult, _ camera: ARCamera?) {
    let position = raycastResult.worldTransform.translation
    recentFocusSquarePositions.append(position)
    updateTransform(for: raycastResult, camera: camera)
}
```

The ray-cast result also indicates how the surface is angled with respect to gravity. To preview the angle at which the user's virtual content can be placed on the surface, update the focus square's `simdWorldTransform` with the result's orientation.

```
func updateOrientation(basedOn raycastResult: ARRaycastResult) {
    self.simdOrientation = raycastResult.worldTransform.orientation
}
```

If your app offers different types of virtual content, give the user an interface to choose from. The sample app exposes a selection menu when the user taps the plus button. When the user chooses an item from the list, you instantiate the corresponding 3D model and anchor it in the world at the focus square's current position.

```
func placeVirtualObject(_ virtualObject: VirtualObject) {
    guard focusSquare.state != .initializing, let query = virtualObject.raycastQuery
        self.statusViewController.showMessage("CANNOT PLACE OBJECT\nTry moving left
        if let controller = self.objectsViewController {
            self.virtualObjectSelectionViewController(controller, didDeselectObject:
        }
        return
    }

    let trackedRaycast = createTrackedRaycastAndSet3DPosition(of: virtualObject, fro
                                                              withInitialResult: vir

    virtualObject.raycast = trackedRaycast
    virtualObjectInteraction.selectedObject = virtualObject
    virtualObject.isHidden = false
}
```

Refine the position of virtual content over time

As the session runs, ARKit analyzes each camera image and learns more about the layout of the physical environment. When ARKit updates its estimated size and position of real-world surfaces, you may need to update the position of your app's virtual content to match. To help make it easy, ARKit notifies you when it corrects its understanding of the scene by way of an `ARTracked Raycast`.

```
func createTrackedRaycastAndSet3DPosition(of virtualObject: VirtualObject, from quer
                                          withInitialResult initialResult: ARRaycast
    if let initialResult = initialResult {
        self.setTransform(of: virtualObject, with: initialResult)
    }
```

```
    return session.trackedRaycast(query) { (results) in
        self.setVirtualObject3DPosition(results, with: virtualObject)
    }
}
```

ARKit successively repeats the query you provide to a tracked ray cast, and it calls the closure you provide only when the results differ from prior results. The code you provide in the closure is your response to ARKit's updated scene understanding. In this case, you check your ray-cast intersections against the updated planes and apply those positions to your app's virtual content.

```
private func setVirtualObject3DPosition(_ results: [ARRaycastResult], with virtualOb

    guard let result = results.first else {
        fatalError("Unexpected case: the update handler is always supposed to returr
    }

    self.setTransform(of: virtualObject, with: result)

    // If the virtual object is not yet in the scene, add it.
    if virtualObject.parent == nil {
        self.sceneView.scene.rootNode.addChildNode(virtualObject)
        virtualObject.shouldUpdateAnchor = true
    }

    if virtualObject.shouldUpdateAnchor {
        virtualObject.shouldUpdateAnchor = false
        self.updateQueue.async {
            self.sceneView.addOrUpdateAnchor(for: virtualObject)
        }
    }
}
```

Manage tracked ray casts

Because ARKit continues to call them, tracked ray casts can increasingly consume resources as the user places more virtual content. Stop the tracked ray cast when you no longer need refined positions over time, such as when a virtual balloon takes flight, or when you remove a virtual object from your scene.

```
func removeVirtualObject(at index: Int) {
    guard loadedObjects.indices.contains(index) else { return }
```

```
    // Stop the object's tracked ray cast.
    loadedObjects[index].stopTrackedRaycast()

    // Remove the visual node from the scene graph.
    loadedObjects[index].removeFromParentNode()
    // Recoup resources allocated by the object.
    loadedObjects[index].unload()
    loadedObjects.remove(at: index)
}
```

To stop a tracked ray cast, you call its `stopTracking()` function:

```
func stopTrackedRaycast() {
    raycast?.stopTracking()
    raycast = nil
}
```

Enable user interaction with virtual content

To allow users to move virtual content in the world after they've placed it, implement a pan gesture recognizer.

```
func createPanGestureRecognizer(_ sceneView: VirtualObjectARView) {
    let panGesture = ThresholdPanGesture(target: self, action: #selector(didPan(_:)))
    panGesture.delegate = self
    sceneView.addGestureRecognizer(panGesture)
}
```

When the user pans an object, you request its position along the object's path across the plane. Because the object's position is transitory, use a `raycast(_:)` instead of using a tracked ray cast. In this case, a one-time hit test is appropriate because you don't need refined position results over time for these requests.

```
func translate(_ object: VirtualObject, basedOn screenPos: CGPoint) {
    object.stopTrackedRaycast()

    // Update the object by using a one-time position request.
    if let query = sceneView.raycastQuery(from: screenPos, allowing: .estimatedPlane
        viewController.createRaycastAndUpdate3DPosition(of: object, from: query)
```

```
    }
}
```

Ray casting gives you orientation information about the surface at a given screen point. While dragging, you avoid quick changes in orientation by subtracting the gesture's rotation from the current object rotation.

```
@objc
func didRotate(_ gesture: UIRotationGestureRecognizer) {
    guard gesture.state == .changed else { return }

    trackedObject?.objectRotation -= Float(gesture.rotation)

    gesture.rotation = 0
}
```

Handle interruption in tracking

In cases where tracking conditions are poor, ARKit invokes your delegate's `sessionWasInterrupted(_:)`. In these circumstances, the positions of your app's virtual content may be inaccurate with respect to the camera feed, so hide your virtual content.

```
func hideVirtualContent() {
    virtualObjectLoader.loadedObjects.forEach { $0.isHidden = true }
}
```

Restore your app's virtual content when tracking conditions improve. To notify you of improved conditions, ARKit calls your delegate's `session(_:cameraDidChangeTrackingState:)`function, passing in a camera `trackingState` equal to `ARTrackingStateNormal`.

```
func session(_ session: ARSession, cameraDidChangeTrackingState camera: ARCamera) {
    statusViewController.showTrackingQualityInfo(for: camera.trackingState, autoHide
    switch camera.trackingState {
    case .notAvailable, .limited:
        statusViewController.escalateFeedback(for: camera.trackingState, inSeconds:
    case .normal:
        statusViewController.cancelScheduledMessage(for: .trackingStateEscalation)
        showVirtualContent()
    }
}
```

Restore an interrupted AR experience

When a session is interrupted, ARKit asks if you want to try to restore the AR experience. You do that by opting in to *relocalization*, by overriding `sessionShouldAttemptRelocalization(_:)` and returning `true`.

```
func sessionShouldAttemptRelocalization(_ session: ARSession) -> Bool {
    return true
}
```

During relocalization, the coaching overlay displays tailored instructions to the user. To allow the user to focus on the coaching process, hide your app's UI when coaching is enabled.

```
func coachingOverlayViewWillActivate(_ coachingOverlayView: ARCoachingOverlayView) {
    upperControlsView.isHidden = true
}
```

When ARKit succeeds in restoring the experience, show your app's UI again so everything appears the way it was before the interruption. When the coaching overlay disappears from the user's view, ARKit invokes your `coachingOverlayViewDidDeactivate(_:)` callback, which is where you restore your app's UI.

```
func coachingOverlayViewDidDeactivate(_ coachingOverlayView: ARCoachingOverlayView)
    upperControlsView.isHidden = false
}
```

Enable the user to start over rather than restore

If the user decides to give up on restoring the session, you restart the experience in your delegate's `coachingOverlayViewDidRequestSessionReset(_:)` function. ARKit invokes this callback when the user taps the coaching overlay's Start Over button.

```
func coachingOverlayViewDidRequestSessionReset(_ coachingOverlayView: ARCoachingOver
    restartExperience()
}
```

See Also

Raycasting

`class ARRaycastQuery`

A mathematical ray you use to find 3D positions on real-world surfaces.

`class ARTrackedRaycast`

A raycast query that ARKit repeats in succession to give you refined results over time.

`class ARRaycastResult`

Information about a real-world surface found by examining a point on the screen.