Creating Immersive Apps with Core Motion

Session 704

John Blackwell, Core Motion Engineer
Ahmad Bleik, Core Motion Engineer
Overview
Authorization
Historical Accelerometer
DeviceMotion
Badger with Attitude
Overview

Authorization

Historical Accelerometer

DeviceMotion

Badger with Attitude
Core Motion
At a glance

- Accelerometer
- Gyroscope
- Magnetometer
- Altimeter

CoreMotion framework

Application
Motion Interfaces
Motion Interfaces

CMMotionManager
Motion Interfaces

CMMotionManager

CMAltimeter
Motion Interfaces

CMMotionManager
CMAltimeter
CMPedometer
Motion Interfaces

CMMotionManager
CMAltimeter
CMPedometer
CMMotionActivityManager
Motion Interfaces

CMMotionManager
CMAltimeter
CMPedometer
CMMotionActivityManager
CMSensorRecorder
Overview
Authorization
Historical Accelerometer
DeviceMotion
Badger with Attitude
Sensitive Interfaces

CMAltimeter
CMPedometer
CMMotionActivityManager
CMSensorRecorder
Authorization

Sensitive API causes prompt
Authorization

Sensitive API causes prompt
Appears only once
“MyGreatApp” Would Like to Access Your Motion & Fitness Activity

Your authorization description here!

Don’t Allow  OK
// Authorization Check
let pedometer = CMPedometer()
let now = Date()

pedometer.queryPedometerData(from: now, to: now) { (data, error) in
    if let code = error?._code {
        if code == CMErrorMotionActivityNotAuthorized.rawValue {
            // Ask the user for authorization!
        }
    }
}
Authorization Check

```swift
let pedometer = CMPedometer()
let now = Date()

pedometer.queryPedometerData(from: now, to: now) { (data, error) in
    if let code = error?._code {
        if code == CMErrorMotionActivityNot Authorized.rawValue {
            // Ask the user for authorization!
        }
    }
}
```
let pedometer = CMPedometer()
let now = Date()

pedometer.queryPedometerData(from: now, to: now) { (data, error) in
    if let code = error?.code {
        if code == CMErrorMotionActivityNotAuthorized.rawValue {
            // Ask the user for authorization!
        }
    }
}

// Authorization Check
// Authorization Check
let pedometer = CMPedometer()
let now = Date()

pedometer.queryPedometerData(from: now, to: now) { (data, error) in
    if let code = error?._code {
        if code == CMErrorMotionActivityNotAuthorized.rawValue {
            // Ask the user for authorization!
        }
    }
}
Authorization Status

// CMAltimeter, CMPedometer, CMMotionActivityManager, CMSensorRecorder

@available(iOS 11.0, *)
@available(watchOS 4.0, *)
open class func authorizationStatus() -> CMAuthorizationStatus
Authorization Status

```swift
@available(iOS 11.0, *)
@available(watchOS 4.0, *)
public enum CMAuthorizationStatus : Int {
    case notDetermined
    case restricted
    case denied
    case authorized
}
```
if CMPedometer.isStepCountingAvailable() {

    switch CMPedometer.authorizationStatus() {
        case .notDetermined: // Handle state before user prompt
            break
        case .restricted: // Handle system-wide restriction
            break
        case .denied: // Handle user denied state
            break
        case .authorized: // Ready to go!
            break
    }
}
// Authorization Check

// Best Practice: Check availability first!
if CMPedometer.isStepCountingAvailable() {
    switch CMPedometer.authorizationStatus() {
    case .notDetermined: // Handle state before user prompt
        break
    case .restricted: // Handle system-wide restriction
        break
    case .denied: // Handle user denied state
        break
    case .authorized: // Ready to go!
        break
    }
}
// Authorization Check

// Best Practice: Check availability first!
if CMPedometer.isStepCountingAvailable() {

    switch CMPedometer.authorizationStatus() {
        case .notDetermined: // Handle state before user prompt
            break
        case .restricted: // Handle system-wide restriction
            break
        case .denied: // Handle user denied state
            break
        case .authorized: // Ready to go!
            break
    }
}
Overview
Authorization
Historical Accelerometer
DeviceMotion
Badger with Attitude
Historical Accelerometer
CMSensorRecorder

Records 50Hz accelerometer in the background
Historical Accelerometer
CMSensorRecorder

Records 50Hz accelerometer in the background
Request up to 36 hours
Historical Accelerometer

CMSensorRecorder

Records 50Hz accelerometer in the background

Request up to 36 hours

Stored for up to three days
## Historical Accelerometer

### Availability

<table>
<thead>
<tr>
<th>Device</th>
<th>Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apple Watch</td>
<td>✓</td>
</tr>
<tr>
<td>Apple Watch Series 1</td>
<td>✓</td>
</tr>
<tr>
<td>Apple Watch Series 2</td>
<td>✓</td>
</tr>
</tbody>
</table>
## Historical Accelerometer

### Availability

<table>
<thead>
<tr>
<th>Device</th>
<th>Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apple Watch</td>
<td>✔️</td>
</tr>
<tr>
<td>Apple Watch Series 1</td>
<td>✔️</td>
</tr>
<tr>
<td>Apple Watch Series 2</td>
<td>✔️</td>
</tr>
<tr>
<td>iPhone 7 and 7 Plus (on iOS 11)</td>
<td>✔️</td>
</tr>
</tbody>
</table>
App Inspiration
Automotive Performance Tracker
App Inspiration
Automotive Performance Tracker

Use Motion Activity for automotive periods
App Inspiration
Automotive Performance Tracker

Use Motion Activity for automotive periods

Automotive detection improved in iOS 11
App Inspiration
Automotive Performance Tracker

Use Motion Activity for automotive periods

Automotive detection improved in iOS 11

Use Sensor Recorder for performance data
App Inspiration
Automotive Performance Tracker

Use Motion Activity for automotive periods

Automotive detection improved in iOS 11

Use Sensor Recorder for performance data

Low-power, all-day experience
Historical Accelerometer

Best practices

Choose the minimum duration
Historical Accelerometer

Best practices

Choose the minimum duration
Decimate if possible
Overview
Authorization
Historical Accelerometer
DeviceMotion
Badger with Attitude
Sensors
What goes into DeviceMotion?
Sensors
What goes into DeviceMotion?

Accelerometer
• Acceleration from user and gravity
Sensors
What goes into DeviceMotion?

Accelerometer
• Acceleration from user and gravity

Gyroscope
• Rotation rate
Sensors
What goes into DeviceMotion?

Accelerometer
- Acceleration from user and gravity

Gyroscope
- Rotation rate

Magnetometer
- Local fields and Earth’s field
Sensors

Challenges

Accelerometer

- Distinguishing user vs. gravity
Sensors

Challenges

Accelerometer
- Distinguishing user vs. gravity

Gyroscope
- Bias over time
Sensors

Challenges

Accelerometer
• Distinguishing user vs. gravity

Gyroscope
• Bias over time

Magnetometer
• Distinguishing local vs. Earth
DeviceMotion
DeviceMotion

3D attitude during motion
DeviceMotion

3D attitude during motion

Fuses accelerometer, gyroscope, and magnetometer
DeviceMotion

3D attitude during motion

Fuses accelerometer, gyroscope, and magnetometer

Allows you to focus on the app
DeviceMotion

More references

<table>
<thead>
<tr>
<th>Title</th>
<th>WWDC Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>What’s New in Core Motion</td>
<td>WWDC 2011</td>
</tr>
<tr>
<td>Understanding Core Motion</td>
<td>WWDC 2012</td>
</tr>
<tr>
<td>Health And Fitness With Core Motion</td>
<td>WWDC 2016</td>
</tr>
</tbody>
</table>
# Reference Frames

<table>
<thead>
<tr>
<th></th>
<th>Accelerometer and Gyroscope</th>
<th>Magnetometer</th>
</tr>
</thead>
<tbody>
<tr>
<td>xArbitraryZVertical</td>
<td></td>
<td></td>
</tr>
<tr>
<td>xArbitraryCorrectedZVertical</td>
<td></td>
<td></td>
</tr>
<tr>
<td>xMagneticNorthZVertical</td>
<td></td>
<td></td>
</tr>
<tr>
<td>xTrueNorthZVertical</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Reference Frames

<table>
<thead>
<tr>
<th></th>
<th>Accelerometer and Gyroscope</th>
<th>Magnetometer</th>
</tr>
</thead>
<tbody>
<tr>
<td>xArbitraryZVertical</td>
<td><img src="green_check.png" alt="Green Check" /></td>
<td><img src="red_x.png" alt="Red X" /></td>
</tr>
<tr>
<td>xArbitraryCorrectedZVertical</td>
<td></td>
<td></td>
</tr>
<tr>
<td>xMagneticNorthZVertical</td>
<td></td>
<td></td>
</tr>
<tr>
<td>xTrueNorthZVertical</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Reference Frames

<table>
<thead>
<tr>
<th>Reference Frame</th>
<th>Accelerometer and Gyroscope</th>
<th>Magnetometer</th>
</tr>
</thead>
<tbody>
<tr>
<td>xArbitraryZVertical</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>xArbitraryCorrectedZVertical</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>xMagneticNorthZVertical</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>xTrueNorthZVertical</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Game Control
Accelerometer

Tilt left and right to steer
Game Control
Accelerometer

Tilt left and right to steer
Game Control
Accelerometer

Tilt left and right to steer

Estimate gravity from accelerometer
Game Control
Accelerometer

Tilt left and right to steer
Estimate gravity from accelerometer
Determine tilt from gravity
Acceleration Ambiguity

Gestures can be ambiguous
Acceleration Ambiguity

Gestures can be ambiguous
Acceleration Ambiguity

Gestures can be ambiguous
Acceleration Ambiguity

Gestures can be ambiguous

Could isolate gravity by averaging
Acceleration Ambiguity

Gestures can be ambiguous
Could isolate gravity by averaging
Filtering affects responsiveness
Acceleration Ambiguity

- Gestures can be ambiguous
- Could isolate gravity by averaging
- Filtering affects responsiveness
- DeviceMotion means less filtering
xArbitraryZVertical

Default reference frame
xArbitraryZVertical

Default reference frame

Great for tip and tilt
xArbitraryZVertical

Default reference frame

Great for tip and tilt

Accelerometer and gyroscope fused
xArbitraryZVertical

Default reference frame

Great for tip and tilt

Accelerometer and gyroscope fused

Gravity for tilt
xArbitraryZVertical

Default reference frame
Great for tip and tilt
Accelerometer and gyroscope fused
Gravity for tilt
Demo a bit later!
xArbitraryZVertical

Great for gestures
Check out SwingWatch

Health and Fitness with Core Motion

WWDC 2016
Game Control

Attitude for aiming
Game Control

Attitude for aiming

Attitude provides rotation from reference frame
Game Control

Attitude for aiming

Attitude provides rotation from reference frame

Avoid taking integral of raw gyroscope
Uses magnetometer to improve horizontal attitude
xArbitraryCorrectedZVertical

Uses magnetometer to improve horizontal attitude

Reliable attitude
xArbitraryCorrectedZVertical

Uses magnetometer to improve horizontal attitude

Reliable attitude

Provides fixed center reference
Magnetometer

Provides world reference
Magnetometer

Provides world reference

Raw magnetometer susceptible to disturbances
Magnetometer

Provides world reference

Raw magnetometer susceptible to disturbances
  • Within the device
Magnetometer

Provides world reference

Raw magnetometer susceptible to disturbances

• Within the device
• Outside the device
xMagneticNorthZVertical and xTrueNorthZVertical

Orients device to the world
xMagneticNorthZVertical and xTrueNorthZVertical

Orients device to the world

Handles:

• Device level effects
xMagneticNorthZVertical and xTrueNorthZVertical

Orients device to the world

Handles:
• Device level effects
• Challenging magnetometer situations
xMagneticNorthZVertical and xTrueNorthZVertical

Orients device to the world

Handles:
• Device level effects
• Challenging magnetometer situations

Pick frame based on your app needs
App Inspiration
xMagneticNorth and xTrueNorth
App Inspiration

xMagneticNorth and xTrueNorth

Star gazing apps
App Inspiration
xMagneticNorth and xTrueNorth

Star gazing apps

Augmented Reality apps
App Inspiration

xMagneticNorth and xTrueNorth

Star gazing apps

Augmented Reality apps

Check out ARKit
Heading

Direction with respect to north

Heading: 0°
Heading

Direction with respect to north

Heading: 36°
Heading

Direction with respect to north

Heading: 72°
Heading

Direction with respect to north

Could use CoreLocation
Direction with respect to north
Could use CoreLocation
CoreLocation’s heading can fuse course
Heading

Direction with respect to north
Could use CoreLocation
CoreLocation’s heading can fuse course
Could calculate from attitude
Heading

Direction with respect to north
Could use CoreLocation
CoreLocation’s heading can fuse course
Could calculate from attitude
DeviceMotion now provides heading
Fuses accelerometer, gyroscope, and magnetometer
Heading

Fuses accelerometer, gyroscope, and magnetometer

iOS only
Valid for XMagneticNorth, XTrueNorth

```swift
// CMDeviceMotion

@available(iOS 11.0, *)
open var heading: Double { get }
```
Heading

Valid for XMagneticNorth, XTrueNorth

0-359 degrees from X axis (North)

// CMDeviceMotion
@available(iOS 11.0, *)
open var heading: Double { get }
DeviceMotion
Best practices

Check for availability
DeviceMotion
Best practices

Check for availability

@available(iOS 5.0, *)
open class func availableAttitudeReferenceFrames() -> CMAttitudeReferenceFrame
DeviceMotion
Best practices

Check for availability

Reference frame choice is key
DeviceMotion
Best practices

Check for availability

Reference frame choice is key
• Attitude definition
DeviceMotion
Best practices

Check for availability

Reference frame choice is key

• Attitude definition
• Sensors used
Overview
Authorization
Historical Accelerometer
DeviceMotion
Badger with Attitude
Badger with Attitude

Ahmad Bleik, Core Motion Engineer
Badger
Badger
Badger with Attitude

Badger controls

Getting started with DeviceMotion

Gesture detection
Badger Controls
Swipe gestures
Badger Controls
Swipe gestures
Badger Controls
Swipe gestures
Badger Controls
Motion gestures
Badger Controls
Motion gestures
Badger Controls
Motion gestures
Badger Controls
Motion gestures
Badger Controls
Motion gestures
Badger with Attitude
Badger with Attitude

DeviceMotion
Badger with Attitude

DeviceMotion
• Sensor fusion
Badger with Attitude

DeviceMotion
  • Sensor fusion

Query mechanisms
Badger with Attitude

DeviceMotion
- Sensor fusion

Query mechanisms
- Push
Badger with Attitude

DeviceMotion
• Sensor fusion

Query mechanisms
• Push
• Pull
Detecting discrete gestures over time
DeviceMotion

Push

Detecting discrete gestures over time

Get data at a fixed interval
DeviceMotion
Push

Detecting discrete gestures over time
Get data at a fixed interval

// Push
func startDeviceMotionUpdates(using referenceFrame: CMAAttitudeReferenceFrame,
to queue: OperationQueue,
withHandler handler: CoreMotion.CMDeviceMotionHandler)
DeviceMotion

Pull
DeviceMotion

Pull

Current device state
DeviceMotion
Pull

Current device state
Responsive
DeviceMotion

Pull

Current device state
Responsive

// Pull
func startDeviceMotionUpdates(using referenceFrame: CMAttitudeReferenceFrame)
// Using DeviceMotion

import CoreMotion

let motionManager = CMMotionManager()

// Before starting game logic

guard motionManager.isDeviceMotionAvailable else {
    print("Device Motion is not available.")
    return
}

let myFrame = CMAttitudeReferenceFrame.xArbitraryZVertical

guard CMMotionManager.availableAttitudeReferenceFrames().contains(myFrame) else {
    print("The reference frame XArbitraryZVertical is not available.")
    return
}
// Using DeviceMotion

import CoreMotion

let motionManager = CMMotionManager()

// Before starting game logic

guard motionManager.isDeviceMotionAvailable else {
    print("Device Motion is not available."
    return
}

let myFrame = CMAttitudeReferenceFrame.xArbitraryZVertical

guard CMMotionManager.availableAttitudeReferenceFrames().contains(myFrame) else {
    print("The reference frame XArbitraryZVertical is not available."
    return
}
// Using DeviceMotion

import CoreMotion

let motionManager = CMMotionManager()

// Before starting game logic

guard motionManager.isDeviceMotionAvailable else {
    print("Device Motion is not available.")
    return
}

let myFrame = CMAttitudeReferenceFrame.xArbitraryZVertical

guard CMMotionManager.availableAttitudeReferenceFrames().contains(myFrame) else {
    print("The reference frame XArbitraryZVertical is not available.")
    return
}
// Using DeviceMotion

import CoreMotion
let motionManager = CMMotionManager()

// Before starting game logic
guard motionManager.isDeviceMotionAvailable else {
    print("Device Motion is not available.")
    return
}

let myFrame = CMAttitudeReferenceFrame.xArbitraryZVertical
guard CMMotionManager.availableAttitudeReferenceFrames().contains(myFrame) else {
    print("The reference frame XArbitraryZVertical is not available.")
    return
}
Gesture Detection

Jump
Gesture Detection

Jump

+x
+y
Gesture Detection

Jump

[x]

[+y]
Gesture Detection

Jump
Gesture Detection

Jump

Rotating rate property
Gesture Detection

Jump

Rotating rate property

Detect a pulse
Gesture Detection
Jump

Rotating rate property
Detect a pulse
Use the push mechanism
// Starting DeviceMotion updates using push

motionManager.deviceMotionUpdateInterval = 1 / 50

motionManager.startDeviceMotionUpdates(using: CMAAttitudeReferenceFrame.xArbitraryZVertical, to: queue, withHandler: motionHandler)
// Starting DeviceMotion updates using push

```swift
motionManager.deviceMotionUpdateInterval = 1 / 50

motionManager.startDeviceMotionUpdates(using: CMAAttitudeReferenceFrame.xArbitraryZVertical,
                                   to: queue,
                                   withHandler: motionHandler)
```
// Starting DeviceMotion updates using push

motionManager.deviceMotionUpdateInterval = 1 / 50

motionManager.startDeviceMotionUpdates(using: CMAAttitudeReferenceFrame.xArbitraryZVertical, to: queue, withHandler: motionHandler)
// Rotation rate

// motionHandler
{ (deviceMotion: CMDeviceMotion?, error: Error?) in
    if let error = error {
        print("Encountered error: \(error!)")
        return
    }

    let rotationRate = deviceMotion.rotationRate
    var rateAlongHorizontal = rotationRate.y
    // ...
    rateAlongHorizontalBuffer.addSample(rateAlongHorizontal)
}
// Rotation rate

// motionHandler
{ (deviceMotion: CMDeviceMotion?, error: Error?) in

    if let error = error {
        print("Encountered error: \(error!)")
        return
    }

    let rotationRate = deviceMotion.rotationRate
    var rateAlongHorizontal = rotationRate.y
    // ...
    rateAlongHorizontalBuffer.addSample(rateAlongHorizontal)
}

// Rotation rate

// motionHandler
{(deviceMotion: CMDeviceMotion?, error: Error?) in
    if let error = error {
        print("Encountered error: \(error!)")
        return
    }

    let rotationRate = deviceMotion.rotationRate
    var rateAlongHorizontal = rotationRate.y
    // ...
    rateAlongHorizontalBuffer.addSample(rateAlongHorizontal)
}
// Rotation rate

// motionHandler
{(deviceMotion: CMDeviceMotion?, error: Error?) in
    if let error = error {
        print("Encountered error: \(error!)")
        return
    }

    let rotationRate = deviceMotion.rotationRate
    var rateAlongHorizontal = rotationRate.y
    // ...
    rateAlongHorizontalBuffer.addSample(rateAlongHorizontal)
}
// Rotation rate

// motionHandler
{(deviceMotion: CMDeviceMotion?, error: Error?) in
    if let error = error {
        print("Encountered error: \
(error!)")
        return
    }

    let rotationRate = deviceMotion.rotationRate
    var rateAlongHorizontal = rotationRate.y
    // ...
    rateAlongHorizontalBuffer.addSample(rateAlongHorizontal)
}
// Rotation rate

// motionHandler
{(deviceMotion: CMDeviceMotion?, error: Error?) in
  if let error = error {
    print("Encountered error: \(error!)")
    return
  }

  let rotationRate = deviceMotion.rotationRate
  var rateAlongHorizontal = rotationRate.y
  // ...
  rateAlongHorizontalBuffer.addSample(rateAlongHorizontal)
}
// Check the buffer

func renderer(_ renderer: SCNSceneRenderer, updateAtTime time: TimeInterval) {
    // ...
    let didJump = rateAlongHorizontalBuffer.mean() > jumpThreshold
    // ...
}
// Check the buffer

```swift
func renderer(_ renderer: SCNSceneRenderer, updateAtTime time: TimeInterval) {
    // ...
    let didJump = rateAlongHorizontalBuffer.mean() > jumpThreshold
    // ...
}
```
// Check the buffer

func renderer(_ renderer: SCNSceneRenderer, updateAtTime time: TimeInterval) {
    // ...
    let didJump = rateAlongHorizontalBuffer.mean() > jumpThreshold
    // ...
}
Gesture Detection

Squat
Gesture Detection
Squat

Gravity
Gesture Detection

Squat
Gesture Detection

Squat
Gesture Detection
Squat

User acceleration property
Gesture Detection
Squat

User acceleration property

Regardless of attitude
Gesture Detection

Squat

User acceleration property

Regardless of attitude

Use the push mechanism
// User acceleration

// motionHandler
{(deviceMotion: CMDeviceMotion?, error: Error?) in

  // Rotation rate
  // ...

  let gravity = deviceMotion.gravity
  let userAcceleration = deviceMotion.userAcceleration

  let userAccelerationAlongGravity = userAcceleration.x * gravity.x +
         userAcceleration.y * gravity.y +
         userAcceleration.z * gravity.z

  // ...
  accelerationAlongGravityBuffer.addSample(userAccelerationAlongGravity)
}
// User acceleration

// motionHandler
{(deviceMotion: CMDeviceMotion?, error: Error?) in

  // Rotation rate
  // ...

  let gravity = deviceMotion.gravity
  let userAcceleration = deviceMotion.userAcceleration

  let userAccelerationAlongGravity = userAcceleration.x * gravity.x +
                                     userAcceleration.y * gravity.y +
                                     userAcceleration.z * gravity.z

  // ...
  accelerationAlongGravityBuffer.addSample(userAccelerationAlongGravity)
}
// User acceleration

// motionHandler
{ (deviceMotion: CMDeviceMotion?, error: Error?) in

    // Rotation rate
    // ...

    let gravity = deviceMotion.gravity

    // ...
    let userAcceleration = deviceMotion.userAcceleration

    let userAccelerationAlongGravity = userAcceleration.x * gravity.x +
                                        userAcceleration.y * gravity.y +
                                        userAcceleration.z * gravity.z

    // ...
    accelerationAlongGravityBuffer.addSample(userAccelerationAlongGravity)
}
// User acceleration

// motionHandler
{(deviceMotion: CMDeviceMotion?, error: Error?) in

// Rotation rate
// ...

let gravity = deviceMotion.gravity
let userAcceleration = deviceMotion.userAcceleration

let userAccelerationAlongGravity = userAcceleration.x * gravity.x +
                             userAcceleration.y * gravity.y +
                             userAcceleration.z * gravity.z

// ...
accelerationAlongGravityBuffer.addSample(userAccelerationAlongGravity)
}
// User acceleration

// motionHandler
{
  (deviceMotion: CMDeviceMotion?, error: Error?) in

  // Rotation rate
  // ...

  let gravity = deviceMotion.gravity
  let userAcceleration = deviceMotion.userAcceleration

  let userAccelerationAlongGravity = userAcceleration.x * gravity.x +
  userAcceleration.y * gravity.y +
  userAcceleration.z * gravity.z

  // ...

  accelerationAlongGravityBuffer.addSample(userAccelerationAlongGravity)
}
// Check the buffer

func renderer(_ renderer: SCNSceneRenderer, updateAtTime time: TimeInterval) {
    // ...
    let didSquat = accelerationAlongGravityBuffer.mean() > squatThreshold
    // ...
}
// Check the buffer

func renderer(_ renderer: SCNSceneRenderer, updateAtTime time: TimeInterval) {
    // ...
    let didSquat = accelerationAlongGravityBuffer.mean() > squatThreshold
    // ...
}
Gesture Detection

Tilt
Gesture Detection

Tilt

Gravity

+X

+Y
Gesture Detection

Tilt
Gesture Detection

Tilt

Current state of the device
Gesture Detection

Tilt

Current state of the device

Responsive
Gesture Detection

Tilt

Current state of the device

Responsive

Use the pull mechanism
// Pull DeviceMotion samples

func renderer(_ renderer: SCNSceneRenderer, updateAtTime time: TimeInterval) {
    // ...
    let deviceMotion = motionManager.deviceMotion
    let gravity = deviceMotion.gravity

    // Component of gravity in the x-z body frame
    let xzComponent = sqrt(pow(gravity.x, 2) + pow(gravity.z, 2))

    let tilt = atan2(gravity.y, xzComponent)
}
func renderer(_ renderer: SCNSceneRenderer, updateAtTime time: TimeInterval) {
    // ...
    let deviceMotion = motionManager.deviceMotion
    let gravity = deviceMotion.gravity
    // Component of gravity in the x-z body frame
    let xzComponent = sqrt(pow(gravity.x, 2) + pow(gravity.z, 2))
    let tilt = atan2(gravity.y, xzComponent)
}
// Pull DeviceMotion samples

func renderer(_ renderer: SCNSceneRenderer, updateAtTime time: TimeInterval) {
    // ...
    let deviceMotion = motionManager.deviceMotion
    let gravity = deviceMotion.gravity

    // Component of gravity in the x-z body frame
    let xzComponent = sqrt(pow(gravity.x, 2) + pow(gravity.z, 2))

    let tilt = atan2(gravity.y, xzComponent)
}
// Pull DeviceMotion samples

func renderer(_ renderer: SCNSceneRenderer, updateAtTime time: TimeInterval) {
    // ...
    let deviceMotion = motionManager.deviceMotion
    let gravity = deviceMotion.gravity

    // Component of gravity in the x-z body frame
    let xzComponent = sqrt(pow(gravity.x, 2) + pow(gravity.z, 2))

    let tilt = atan2(gravity.y, xzComponent)
}
// Pull DeviceMotion samples

func renderer(_ renderer: SCNSceneRenderer, updateAtTime time: TimeInterval) {
    // ...
    let deviceMotion = motionManager.deviceMotion
    let gravity = deviceMotion.gravity

    // Component of gravity in the x–z body frame
    let xzComponent = sqrt(pow(gravity.x, 2) + pow(gravity.z, 2))

    let tilt = atan2(gravity.y, xzComponent)
}
// Stop Device Motion updates

if motionManager.isDeviceMotionActive {
    motionManager.stopDeviceMotionUpdates()
}
Summary

Authorization
Summary

Authorization

DeviceMotion
Summary

Authorization

DeviceMotion

• Sensor fusion
Authorization

DeviceMotion

• Sensor fusion

• Smooth and consistent experience
Summary

Authorization

DeviceMotion

• Sensor fusion
• Smooth and consistent experience
• Performance enhancements
Authorization

DeviceMotion
  • Sensor fusion
  • Smooth and consistent experience
  • Performance enhancements

Query mechanism
Summary

Authorization

DeviceMotion
- Sensor fusion
- Smooth and consistent experience
- Performance enhancements

Query mechanism
- Push: Gesture over time
Summary

Authorization

DeviceMotion
• Sensor fusion
• Smooth and consistent experience
• Performance enhancements

Query mechanism
• Push: Gesture over time
• Pull: Responsive
## Related Sessions

<table>
<thead>
<tr>
<th>Session</th>
<th>Venue</th>
<th>Date</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introducing ARKit: Augmented Reality for iOS</td>
<td>Hall 3</td>
<td>Tue</td>
<td>5:10PM-6:10PM</td>
</tr>
<tr>
<td>What's New in Location Technologies</td>
<td>Grand Ballroom B</td>
<td>Thu</td>
<td>3:10PM-3:50PM</td>
</tr>
</tbody>
</table>
## Related Sessions

<table>
<thead>
<tr>
<th>Session</th>
<th>Venue</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introducing ARKit: Augmented Reality for iOS</td>
<td>Hall 3</td>
<td>Tue 5:10PM-6:10PM</td>
</tr>
<tr>
<td>What's New in Location Technologies</td>
<td>Grand Ballroom B</td>
<td>Thu 3:10PM-3:50PM</td>
</tr>
<tr>
<td>Advances in SceneKit Rendering</td>
<td></td>
<td>WWDC 2016</td>
</tr>
<tr>
<td>Lab</td>
<td>Location</td>
<td>Time</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>Core Motion Lab</td>
<td>Technology Lab D</td>
<td>Thu 4:10PM-6:00PM</td>
</tr>
<tr>
<td>Location and Mapping Technologies Lab</td>
<td>Technology Lab B</td>
<td>Wed 11:00AM-1:00PM</td>
</tr>
<tr>
<td>Location and Mapping Technologies Lab</td>
<td>Technology Lab K</td>
<td>Fri 10:00AM-12:00PM</td>
</tr>
<tr>
<td>ARKit Lab</td>
<td>Technology Lab A</td>
<td>Wed 1:00PM-3:00PM</td>
</tr>
<tr>
<td>ARKit Lab</td>
<td>Technology Lab A</td>
<td>Thu 12:00PM-2:00PM</td>
</tr>
</tbody>
</table>
More Information
