Leveraging Touch Input on iOS

...And getting the most out of Apple Pencil

Session 220

Dominik Wagner UIKit Engineer
New and Recent Hardware
3D Touch
iPhone 6s and iPhone 6s Plus

A Peek at 3D Touch
Presidio
Thursday 4:00PM
Faster Touch Scanning
iPad Air 2 and iPad Pro
Apple Pencil
iPad Pro

Precise location
240 Hz scan rate
Tilt, orientation, and force
Palm rejection
Siri Remote
Apple TV

UIFocusEngine
Game Controller
Indirect touches
Siri Remote
Apple TV

UIFocusEngine
Game Controller
Indirect touches

Apple TV Tech Talks
Controlling Game Input for Apple TV  Mission  Wednesday 5:00PM
Building a Drawing App
Building a Drawing App

Agenda

New API
Step-by-step
Sample code available
Say Hello to SpeedSketch

One sheet of paper you can draw on
Full support for Apple Pencil and 3D Touch
Building a Drawing App
Model and capture
Building a Drawing App
Model and capture

Series of strokes
UITouch in event callbacks
Copy the relevant data
Building a Drawing App

Model and capture

```swift
struct StrokeSample {
    let location: CGPoint
}
```
Building a Drawing App

Model and capture

```swift
class Stroke {
    var samples: [StrokeSample] = []

    func add(sample: StrokeSample)
}
```
Building a Drawing App

Model and capture

class Stroke {
    var samples: [StrokeSample] = []
    var state: StrokeState = .active

    func add(sample: StrokeSample)
}

enum StrokeState {
    case active
    case done
    case cancelled
}
Building a Drawing App

Model and capture

class StrokeCollection {
    var strokes: [Stroke] = []

    func add(doneStroke: stroke)
}
class StrokeCollection {
    var strokes: [Stroke] = []
    var activeStroke: Stroke?
    func add(doneStroke: stroke)
}
Building a Drawing App

Model and capture

Where to capture?

• UIGestureRecognizer
• UIView
• Up the responder chain
Building a Drawing App

Capture

Custom UIGestureRecognizer
Targeting the main view controller
View controller facilitates update

Stroke Gesture Recognizer

Canvas View Controller

Stroke View
import UIKit.UIGestureRecognizerSubclass

class StrokeGestureRecognizer: UIGestureRecognizer {
}

}
import UIKit.UIGestureRecognizerSubclass

class StrokeGestureRecognizer: UIGestureRecognizer {

    var stroke = Stroke()

}
import UIKit.UIGestureRecognizerSubclass

class StrokeGestureRecognizer: UIGestureRecognizer {

    var stroke = Stroke()

    override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {

    }

}
import UIKit.UIGestureRecognizerSubclass

class StrokeGestureRecognizer: UIGestureRecognizer {
    var stroke = Stroke()

    func appendTouches(_ touches: Set<UITouch>, event: UIEvent?) -> Bool {
    }

    override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
    }
}

}
import UIKit.UIGestureRecognizerSubclass

class StrokeGestureRecognizer: UIGestureRecognizer {
    var stroke = Stroke()

    func appendTouches(_ touches: Set<UITouch>, event: UIEvent?) -> Bool {
        // Append touches to the stroke
    }

    override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
        if appendTouches(touches, event: event) {
            state = .began
        }
    }
}
import UIKit.UIGestureRecognizerSubclass

class StrokeGestureRecognizer: UIGestureRecognizer {

    var stroke = Stroke()

    func appendTouches(_ touches: Set<UITouch>, event: UIEvent?) -> Bool {
        // Append touches to Stroke
        return true
    }

    override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
        if appendTouches(touches, event: event) {
            state = .began
        }
    }

    override func touchesMoved(_ touches: Set<UITouch>, with event: UIEvent?) {
        if appendTouches(touches, event: event) {
            state = .changed
        }
    }
}

import UIKit.UIGestureRecognizerSubclass

class StrokeGestureRecognizer: UIGestureRecognizer {

    var stroke = Stroke()

    func appendTouches(_ touches: Set<UITouch>, event: UIEvent?) -> Bool ...

    override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
        if appendTouches(touches, event: event) {
            state = .began
        }
    }

    override func touchesMoved(_ touches: Set<UITouch>, with event: UIEvent?) {
        if appendTouches(touches, event: event) {
            state = .changed
        }
    }

    override func touchesEnded(_ touches: Set<UITouch>, with event: UIEvent?) ...
    override func touchesCancelled(_ touches: Set<UITouch>, with event: UIEvent?) ...
}
import UIKit.UIGestureRecognizerSubclass

class StrokeGestureRecognizer: UIGestureRecognizer {

    var stroke = Stroke()

    func appendTouches(_ touches: Set<UITouch>, event: UIEvent?) -> Bool …

    override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
        if appendTouches(touches, event:event) {
            state = .began
        }
    }

    override func touchesMoved(_ touches: Set<UITouch>, with event: UIEvent?) {
        if appendTouches(touches, event:event) {
            state = .changed
        }
    }

    override func touchesEnded(_ touches: Set<UITouch>, with event: UIEvent?) …
    override func touchesCancelled(_ touches: Set<UITouch>, with event: UIEvent?) …

    override func reset() {
        stroke = Stroke()
        super.reset()
    }
}
import UIKit.UIGestureRecognizerSubclass

class StrokeGestureRecognizer: UIGestureRecognizer {

    var stroke = Stroke()

    func appendTouches(_ touches: Set<UITouch>, event: UIEvent?) -> Bool {
    }

    override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
        if appendTouches(touches, event: event) {
            state = .began
        }
    }

    override func touchesMoved(_ touches: Set<UITouch>, with event: UIEvent?) {
        if appendTouches(touches, event: event) {
            state = .changed
        }
    }

    override func touchesEnded(_ touches: Set<UITouch>, with event: UIEvent?) {
    }

    override func touchesCancelled(_ touches: Set<UITouch>, with event: UIEvent?) {
    }

    override func reset() {
        stroke = Stroke()
        super.reset()
    }
}
class CanvasViewController: UIViewController {

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

}
class CanvasViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        let strokeRecognizer = StrokeGestureRecognizer(
            target: self,
            action: #selector(strokeUpdated(_:))
        )

        view.addGestureRecognizer(strokeRecognizer)
    }
}
class CanvasViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        let strokeRecognizer = StrokeGestureRecognizer(
            target: self,
            action: #selector(strokeUpdated(_:))
        )

        view.addGestureRecognizer(strokeRecognizer)
    }

    func strokeUpdated(_ strokeGesture: StrokeGestureRecognizer) {
        view.strokeToDraw = strokeGesture.stroke
    }
}
```swift
class CanvasViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        let strokeRecognizer = StrokeGestureRecognizer(
            target: self,
            action: #selector(strokeUpdated(_:))
        )

        view.addGestureRecognizer(strokeRecognizer)
    }

    func strokeUpdated(_ strokeGesture: StrokeGestureRecognizer) {
        view.strokeToDraw = strokeGesture.stroke
    }
}
```
Let’s have a look
Pencil tip
Analysis of First Attempt

Missed events

- Drawing engine
- Did not use the new iOS 9.0 API
Anatomy of a Stroke
Anatomy of a Stroke
Anatomy of a Stroke
Anatomy of a Stroke

Began

Ended
Anatomy of a Stroke
Anatomy of a Stroke
Anatomy of a Stroke

Began

Coalesced

Moved

Moved

Moved

Moved

Moved

Moved

Moved

Moved

Moved

Moved

Moved

Moved

Moved

Moved

Moved

Moved

Moved

Ended
Anatomy of a Stroke
class UIEvent {
    public func coalescedTouches(for touch:UITouch) -> [UITouch]?
}
class StrokeGestureRecognizer: UIGestureRecognizer {
    override func touchesMoved(_ touches: Set<UITouch>, with event: UIEvent?) {
        if let touch = touches.first {
            appendTouch(touch)
        }
    }
}
class StrokeGestureRecognizer: UIGestureRecognizer {
    override func touchesMoved(_ touches: Set<UITouch>, with event: UIEvent?) {
        if let touch = touches.first {
            for coalescedTouch in event.coalescedTouches(for: touch) {
                appendTouch(touch)
            }
        }
    }
}
Pencil

Coalesced touches (gray)

Live touch (black)

Too many coalesced touches

Excessive gap
Analysis of Second Attempt

Speed is still lacking

UIKit helped us by coalescing
Do Not Draw on Every Touch Event

Display refresh rate is 60 Hz
Incoming event frequency can reach 240 Hz and more
Do not try to draw faster than screen can refresh
When to Render?

**UIView**

- Use `setNeedsDisplay()`
- Implement `draw(_ rect: CGRect)`

**GLKView, MTLView**

- Set the `enableSetNeedsDisplay` property to true
- If required, draw at a steady rate using a CADisplayLink object
class StrokeCGView: UIView {

    var strokeToDraw: Stroke? {
        didSet {
            drawImageAndUpdate()
        }
    }
}

}
class StrokeCGView: UIView {

    var strokeToDraw: Stroke? {
        didSet {
            setNeedsDisplay()
        }
    }
}
Even Better

Mark only the changed areas

- `setNeedsDisplayIn(rect: changedRect)`

Activate `drawsAsynchronously` on the layer

```swift
class StrokeCGView: UIView {

    override init(frame: CGRect) {
        super.init(frame: frame)
        layer.drawsAsynchronously = true
    }
}
```
Steady amount of coalesced touches

Still some lag
Improve Perceived Latency

Use predicted touches

class UIEvent {
    public func predictedTouches(for touch:UITouch) -> [UITouch]?
}

Predicted Touches

Add predicted touches to your data structure temporarily. Choose their appearance, depending on your app:

- To appear like actual touches
- To appear as tentative
class StrokeGestureRecognizer: UIGestureRecognizer {

    override func touchesMoved(_ touches: Set<UITouch>, with event: UIEvent?) {
        if let touch = touches.first {

            for coalescedTouch in event.coalescedTouches(for: touch) {
                appendTouch(touch)
            }
        }
    }
}
class StrokeGestureRecognizer: UIGestureRecognizer {
    override func touchesMoved(_ touches: Set<UITouch>, with event: UIEvent?) {
        if let touch = touches.first {

            for coalescedTouch in event.coalescedTouches(for:touch) {
                appendTouch(touch)
            }

            for predictedTouch in event.predictedTouches(for:touch) {
                appendTouchTemporarily(touch)
            }
        }
    }
}
Predicted Touches

```swift
class StrokeGestureRecognizer: UIGestureRecognizer {
    override func touchesMoved(_ touches: Set<UITouch>, with event: UIEvent?) {
        if let touch = touches.first {
            clearTemporaryTouches()

            for coalescedTouch in event.coalescedTouches(for: touch) {
                appendTouch(touch)
            }

            for predictedTouch in event.predictedTouches(for: touch) {
                appendTouchTemporarily(touch)
            }
        }
    }
}
```
Predicted touches
What Have We Seen so Far

Collect input data using a UIGestureRecognizer
Access coalesced touches
Make rendering fast and efficient
Use predicted touches
Apple Pencil
Apple Pencil

Touch types

```swift
public var type: UITouchType { get }
```
Apple Pencil
Touch types

```swift
public var type: UITouchType { get }

enum UITouchType : Int {
  case direct
  case indirect
  case stylus
}
```
Apple Pencil
Higher precision

```swift
func preciseLocation(in view: UIView?) -> CGPoint
func precisePreviousLocation(in view: UIView?) -> CGPoint
```
# Apple Pencil and 3D Touch

## Force

<table>
<thead>
<tr>
<th>Device</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>iPhone 6s (Plus)</td>
<td>0.0 - maximumPossibleForce</td>
</tr>
<tr>
<td>Apple Pencil on iPad Pro</td>
<td>0.0 - maximumPossibleForce</td>
</tr>
<tr>
<td>Touch on iPad Pro and all previous devices</td>
<td>0.0</td>
</tr>
</tbody>
</table>

```swift
public var force: CGFloat { get }
public var maximumPossibleForce: CGFloat { get }
```
Apple Pencil and 3D Touch

Force

```swift
func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent)
func touchesMoved(_ touches: Set<UITouch>, with event: UIEvent)
func touchesEnded(_ touches: Set<UITouch>, with event: UIEvent)
func touchesCancelled(_ touches: Set<UITouch>, with event: UIEvent)
```
Apple Pencil and 3D Touch

Tap recognition

```swift
func touchesMoved(_ touches: Set<UITouch>, with event: UIEvent) {
    cancelTap()
}
```

Use UITapGestureRecognizer
Force
Add to the model

```swift
struct StrokeSample {
    let location: CGPoint
}
```
Force
Add to the model

```swift
struct StrokeSample {
    let location: CGPoint
    var force: CGFloat?
}
```
Width based on force
Apple Pencil
Tilt
Apple Pencil

Tilt

Altitude
Apple Pencil
Tilt

```swift
var altitudeAngle: CGFloat { get }
```
Apple Pencil
Orientation
Apple Pencil
Orientation
Apple Pencil

Orientation
Apple Pencil

Orientation
Apple Pencil
Orientation

Azimuth
func azimuthAngle(in view: UIView?) -> CGFloat
func azimuthUnitVector(in view: UIView?) -> CGVector
Apple Pencil

Force

Along axis
Apple Pencil

Force

Perpendicular to device surface
```swift
var perpendicularForce: CGFloat {
    get {
        return force / sin(altitudeAngle)
    }
}
```
```
var perpendicularForce: CGFloat {
    get {
        return min(force / sin(altitudeAngle), maximumPossibleForce)
    }
}
```
Apple Pencil

Force
Apple Pencil
Estimated properties

```swift
var estimatedProperties: UITouchProperties { get }
```
var estimatedProperties: UITouchProperties { get }

struct UITouchProperties : OptionSet {

}

Apple Pencil
Estimated properties
Apple Pencil
Estimated properties

```swift
var estimatedProperties: UITouchProperties { get }

struct UITouchProperties : OptionSet {
    static var force: UITouchProperties { get }
}
```
Apple Pencil
Estimated properties

```swift
var estimatedProperties: UITouchProperties { get }

struct UITouchProperties : OptionSet {
    static var force: UITouchProperties { get }
    static var azimuth: UITouchProperties { get }
    static var altitude: UITouchProperties { get }
}
```
Estimated properties

```
var estimatedProperties: UITouchProperties { get }

struct UITouchProperties : OptionSet {
    static var force: UITouchProperties { get }
    static var azimuth: UITouchProperties { get }
    static var altitude: UITouchProperties { get }
    static var location: UITouchProperties { get }
}
```
var estimatedProperties: UITouchProperties { get }

struct UITouchProperties : OptionSet {
    static var force: UITouchProperties { get }
    static var azimuth: UITouchProperties { get }
    static var altitude: UITouchProperties { get }
    static var location: UITouchProperties { get }
}

var estimatedPropertiesExpectingUpdates: UITouchProperties { get }
Apple Pencil
Estimated properties with updates

```swift
func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent)
func touchesMoved(_ touches: Set<UITouch>, with event: UIEvent)
func touchesEnded(_ touches: Set<UITouch>, with event: UIEvent)
func touchesCancelled(_ touches: Set<UITouch>, with event: UIEvent)
```
Apple Pencil

Estimated properties with updates

def touchesBegan(_ touches: Set<UITouch>, with event: UIEvent)
def touchesMoved(_ touches: Set<UITouch>, with event: UIEvent)
def touchesEnded(_ touches: Set<UITouch>, with event: UIEvent)
def touchesCancelled(_ touches: Set<UITouch>, with event: UIEvent)
def touchesEstimatedPropertiesUpdated(_ touches: Set<UITouch>)
Apple Pencil

Estimated properties with updates

Check `estimatedPropertiesExpectingUpdates`

Use `estimationUpdateIndex` as key to index your sample

Look up the `estimationUpdateIndex` in `touchesEstimatedPropertiesUpdated(_:)`

Some updates will arrive after `touchesEnded:`
override func touchesEstimatedPropertiesUpdated(_ touches: Set<UITouch>) {
    for touch in touches {
    }
}
override func touchesEstimatedPropertiesUpdated(_ touches: Set<UITouch>) {
    for touch in touches {
        let estimationIndex = touch.estimationUpdateIndex!
    }
}
override func touchesEstimatedPropertiesUpdated(_ touches: Set<UITouch>) {
    for touch in touches {
        let estimationIndex = touch.estimationUpdateIndex!
        let (sample, sampleIndex) = samplesExpectingUpdates[estimationIndex]
    }
}
override func touchesEstimatedPropertiesUpdated(_ touches: Set<UITouch>) {
    for touch in touches {
        let estimationIndex = touch.estimationUpdateIndex!
        let (sample, sampleIndex) = samplesExpectingUpdates[estimationIndex]
        let updatedSample = updatedSample(with: touch)
    }
}
override func touchesEstimatedPropertiesUpdated(_ touches: Set<UITouch>) {
    for touch in touches {
        let estimationIndex = touch.estimationUpdateIndex!
        let (sample, sampleIndex) = samplesExpectingUpdates[estimationIndex]
        let updatedSample = updatedSample(with: touch)
        stroke.update(sample: updatedSample, at: sampleIndex)
    }
}
override func touchesEstimatedPropertiesUpdated(_ touches: Set<UITouch>) {
    for touch in touches {
        let estimationIndex = touch.estimatedUpdateIndex!
        let (sample, sampleIndex) = samplesExpectingUpdates[estimationIndex]
        let updatedSample = updatedSample(with: touch)
        stroke.update(sample: updatedSample, at: sampleIndex)
        if touch.estimatedPropertiesExpectingUpdates == [] {
            samplesExpectingUpdates.removeValue(forKey: sampleIndex)
        }
    }
}
Azimuth
Finishing Touches
Canvas
Finishing Touches
Canvas
Finishing Touches
Canvas
Adjusting gestures

Scroll view UIPanGestureRecognizer vs. StrokeGestureRecognizer

Disable scrolling with Apple Pencil

```swift
class UIGestureRecognizer {
    public var allowedTouchTypes: [NSNumber]
}

let pan = scrollView.panGestureRecognizer
pan.allowedTouchTypes = [UITouchType.direct.rawValue as NSNumber]
strokeRecognizer.allowedTouchTypes = [UITouchType.stylus.rawValue as NSNumber]
```
class UIGestureRecognizer {
    public var requiresExclusiveTouchType: Bool
}

Finishing Touches
Adjusting gestures
Summary

New properties of UITouch
Coalesced and predicted touches
Property estimation
Adjusting gestures
Sample
More Information

<table>
<thead>
<tr>
<th>Session</th>
<th>Location</th>
<th>Date-Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controlling Game Input for Apple TV</td>
<td>Mission</td>
<td>Wednesday 5:00PM</td>
</tr>
<tr>
<td>A Peek at 3D Touch</td>
<td>Presidio</td>
<td>Thursday 4:00PM</td>
</tr>
<tr>
<td>Advanced Touch Input on iOS</td>
<td></td>
<td>WWDC 2015</td>
</tr>
<tr>
<td>Labs</td>
<td>Frameworks</td>
<td>Date/Time</td>
</tr>
<tr>
<td>-----------------------</td>
<td>--------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>UIKit and UIKit Animations Lab</td>
<td>Lab C</td>
<td>Thursday 1:00PM</td>
</tr>
<tr>
<td>Cocoa Touch and 3D Touch Lab</td>
<td>Lab C</td>
<td>Friday 10:30AM</td>
</tr>
</tbody>
</table>