Sample Code

Leveraging touch input for drawing apps

Capture touches as a series of strokes and render them efficiently on a drawing canvas.

Download

iOS 10.0+ | iPadOS 10.0+ | Xcode 11.3+

Overview

This sample project, Speed Sketch, shows how to render a series of strokes a user makes by moving their finger or Apple Pencil across the screen. The project also demonstrates how to:

- Distinguish between finger and Apple Pencil touches.
- Improve drawing performance.
- Enhance the app for Apple Pencil.

Distinguish between finger and Apple Pencil touches

With Speed Sketch, users can draw by moving their finger or Apple Pencil across the screen, and the system reports the touches to the app. Speed Sketch captures these touches using the custom gesture recognizer, `StrokeGestureRecognizer`.

The stroke gesture recognizer receives each touch event, and appends data from the touches to an array of data samples.

```
override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
    if trackedTouch == nil {
        trackedTouch = touches.first
        initialTimestamp = trackedTouch?.timestamp
        collectForce = trackedTouch!.type == .pencil || view?.traitCollection.forceT
        if !isForPencil {
            // Give other gestures, such as pan and pinch, a chance by
            // slightly delaying the `.begin.
            fingerStartTimer = Timer.scheduledTimer(
                withTimeInterval: cancellationTimeInterval,
                repeats: false,
                block: { [weak self] (timer) in
                    guard let strongSelf = self else { return }
                    if strongSelf.state == .possible {
                        strongSelf.state = .began
                    }
                }
            )
        }
    }
    if append(touches: touches, event: event) {
        if isForPencil {
            state = .began
        }
    }
}

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

override func touchesEnded(_ touches: Set<UITouch>, with event: UIEvent?) {
    if append(touches: touches, event: event) {
        stroke.state = .done
        state = .ended
    }
}

override func touchesCancelled(_ touches: Set<UITouch>, with event: UIEvent?) {
    if append(touches: touches, event: event) {
        stroke.state = .cancelled
        state = .failed
    }
```

```
}
```

The touchesBegan(_:with:) method sets a timer to give other gesture recognizers, such as pan and pinch, time to handle touches that the user makes with their finger. This behavior is unique to drawing apps such as Speed Sketch that start capturing and drawing a stroke the moment the user touches the screen.

In order to track finger and Apple Pencil touches separately, Speed Sketch uses two instances of `StrokeGestureRecognizer`: one for finger touches and the other to track touches coming from Apple Pencil.

```
self.fingerStrokeRecognizer = setupStrokeGestureRecognizer(isForPencil: false)
self.pencilStrokeRecognizer = setupStrokeGestureRecognizer(isForPencil: true)
```

To simplify the code, the app uses a helper method to create the gesture recognizers.

```
func setupStrokeGestureRecognizer(isForPencil: Bool) -> StrokeGestureRecognizer {
    let recognizer = StrokeGestureRecognizer(target: self, action: #selector(strokeU
    recognizer.delegate = self
    recognizer.cancelsTouchesInView = false
    scrollView.addGestureRecognizer(recognizer)
    recognizer.coordinateSpaceView = cgView
    recognizer.isForPencil = isForPencil
    return recognizer
}
```

The gesture recognizer distinguishes between the touch types by setting the allowedTouchTypes property to UITouch.TouchType.pencil when tracking Apple Pencil touches, and to UITouch.TouchType.direct when tracking touches from a finger.

```
var isForPencil: Bool = false {
    didSet {
        if isForPencil {
            allowedTouchTypes = [UITouch.TouchType.pencil.rawValue as NSNumber]
        } else {
            allowedTouchTypes = [UITouch.TouchType.direct.rawValue as NSNumber]
        }
    }
}
```

Each time the user draws a stroke on the screen, the stroke gesture recognizer calls the `strokeUpdate(_:)` method. This method checks to see if the recognizer is for Apple Pencil, and if so, puts the app into pencil mode. While the app is in pencil mode, the user can use one finger to pan the drawing canvas. When the app isn't in pencil mode—that is, when the user is drawing with their finger—the user uses two fingers to pan the drawing canvas.

```
var pencilMode = false {
    didSet {
        if pencilMode {
            scrollView.panGestureRecognizer.minimumNumberOfTouches = 1
            pencilButton.isHidden = false
            if let view = fingerStrokeRecognizer.view {
                view.removeGestureRecognizer(fingerStrokeRecognizer)
            }
        } else {
            scrollView.panGestureRecognizer.minimumNumberOfTouches = 2
            pencilButton.isHidden = true
            if fingerStrokeRecognizer.view == nil {
                scrollView.addGestureRecognizer(fingerStrokeRecognizer)
            }
        }
    }
}
```

Improve drawing performance

The stroke gesture recognizer receives touch events from the system. For most apps, the time span between each touch event is enough to handle the gesture. However, users of drawing apps expect greater precision, which requires the app to gather all the touches reported since the last delivered touch event.

To gather the additional touches, Speed Sketch calls the `coalescedTouches(for:)` method. This method returns an array of `UITouch` objects representing the touches received by the system but not delivered in the last event. The additional touches allow Speed Sketch to provide a smoother rendering of the stroke by storing the coalesced touches as part of the stroke's data sample.

```
if collectsCoalescedTouches {
    if let event = event {
        let coalescedTouches = event.coalescedTouches(for: touchToAppend)!
        let lastIndex = coalescedTouches.count - 1
        for index in 0..<lastIndex {
            saveStrokeSample(stroke: stroke, touch: coalescedTouches[index], view: v
        }
        saveStrokeSample(stroke: stroke, touch: coalescedTouches[lastIndex], view: v
    }
} else {
    saveStrokeSample(stroke: stroke, touch: touchToAppend, view: view, coalesced: fa
}
```

In addition to coalesced touches, Speed Sketch uses touch data predicted by the system as a way to enhance the perceived accuracy of the app's drawing capabilities. Speed Sketch retrieves the predicted touches by calling the `predictedTouches(for:)` method for the event, then saves the touches as part of the data sample for the stroke.

```
if usesPredictedSamples && stroke.state == .active {
    if let predictedTouches = event?.predictedTouches(for: touchToAppend) {
        for touch in predictedTouches {
            saveStrokeSample(stroke: stroke, touch: touch, view: view, coalesced: fa
        }
    }
}
```

The app replaces the predicted touches with actual touches as they become available.

```
override func touchesEstimatedPropertiesUpdated(_ touches: Set<UITouch>) {
    for touch in touches {
        guard let index = touch.estimationUpdateIndex else {
            continue
        }
        if let (stroke, sampleIndex) = outstandingUpdateIndexes[Int(index.intValue)]
            var sample = stroke.samples[sampleIndex]
            let expectedUpdates = sample.estimatedPropertiesExpectingUpdates
            if expectedUpdates.contains(.force) {
                sample.force = touch.force
                if !touch.estimatedProperties.contains(.force) {
                    // Only remove the estimate flag if the new value isn't estimate
                    sample.estimatedProperties.remove(.force)
                }
            }
            sample.estimatedPropertiesExpectingUpdates = touch.estimatedPropertiesEx
            if touch.estimatedPropertiesExpectingUpdates == [] {
                outstandingUpdateIndexes.removeValue(forKey: sampleIndex)
            }
            stroke.update(sample: sample, at: sampleIndex)
        }
    }
}
```

Enhance for Apple Pencil

Apple Pencil can sense tilt (altitude), force (pressure), and orientation (azimuth), which drawing apps can use to affect the appearance of strokes. For instance, Speed Sketch uses azimuth to enhance the strokes when using the app's Calligraphy drawing tool.

```
func drawCalligraphy(in context: CGContext,
                     toSample: StrokeSample,
                     fromSample: StrokeSample,
                     forceAccessBlock: (_ sample: StrokeSample) -> CGFloat) {

    var fromAzimuthUnitVector = Stroke.calligraphyFallbackAzimuthUnitVector
    var toAzimuthUnitVector = Stroke.calligraphyFallbackAzimuthUnitVector

    if fromSample.azimuth != nil {

        if lockedAzimuthUnitVector == nil {
            lockedAzimuthUnitVector = fromSample.azimuthUnitVector
        }
        fromAzimuthUnitVector = fromSample.azimuthUnitVector
        toAzimuthUnitVector = toSample.azimuthUnitVector
        if fromSample.altitude! > azimuthLockAltitudeThreshold {
            fromAzimuthUnitVector = lockedAzimuthUnitVector!
        }
        if toSample.altitude! > azimuthLockAltitudeThreshold {
            toAzimuthUnitVector = lockedAzimuthUnitVector!
        } else {
            lockedAzimuthUnitVector = toAzimuthUnitVector
        }

    }
    // Rotate 90 degrees
    let calligraphyTransform = CGAffineTransform(rotationAngle: CGFloat.pi / 2.0)
    fromAzimuthUnitVector = fromAzimuthUnitVector.applying(calligraphyTransform)
    toAzimuthUnitVector = toAzimuthUnitVector.applying(calligraphyTransform)
    let fromUnitVector = fromAzimuthUnitVector * forceAccessBlock(fromSample)
    let toUnitVector = toAzimuthUnitVector * forceAccessBlock(toSample)

    context.beginPath()
    context.addLines(between: [
        fromSample.location + fromUnitVector,
        toSample.location + toUnitVector,
        toSample.location - toUnitVector,
        fromSample.location - fromUnitVector
        ])
    context.closePath()

    context.drawPath(using: .fillStroke)

}
```

Speed Sketch also displays the altitude and azimuth data as part of the stroke when the user selects the Debug drawing tool.

```
func drawDebugMarkings(in context: CGContext, fromSample: StrokeSample) {

    let isEstimated = fromSample.estimatedProperties.contains(.azimuth)
    guard displayOptions == .debug,
        fromSample.predicted == false,
        fromSample.azimuth != nil,
        (!fromSample.coalesced || isEstimated) else {
            return
    }

    let length = CGFloat(20.0)
    let azimuthUnitVector = fromSample.azimuthUnitVector
    let azimuthTarget = fromSample.location + azimuthUnitVector * length
    let altitudeStart = azimuthTarget + (azimuthUnitVector * (length / -2.0))
    let transformToApply = CGAffineTransform(rotationAngle: fromSample.altitude!)
    let altitudeTarget = altitudeStart + (azimuthUnitVector * (length / 2.0)).applyi

    // Draw altitude as black line coming from the center of the azimuth.
    altitudeSettings(in: context)
    context.beginPath()
    context.move(to: altitudeStart)
    context.addLine(to: altitudeTarget)
    context.strokePath()
    // Draw azimuth as blue (or orange if estimated) line.
    azimuthSettings(in: context)
    if isEstimated {
        context.setStrokeColor(UIColor.orange.cgColor)
    }
    context.beginPath()
    context.move(to: fromSample.location)
    context.addLine(to: azimuthTarget)
    context.strokePath()

}
```

Starting with the second generation Apple Pencil, users can double-tap their finger on Apple Pencil to request an action from the app (see Apple Pencil interactions for more information). When possible, apps should honor the system settings for double-tap actions on Apple Pencil, which include:

- Switching to the eraser or the last used tool
- Displaying a color pallet
- Ignoring double tap

Speed Sketch doesn't have an eraser tool or color pallet, but it does have different drawing tools: Calligraphy, Ink, and Debug. When the user's preferred double-tap action is `UIPencilPreferredAction.switchPrevious` and the user double-taps their finger on Apple Pencil, Speed Sketch switches to the tool last used by the user. The sample app ignores the other preferred actions.

```
func pencilInteractionDidTap(_ interaction: UIPencilInteraction) {
    if UIPencilInteraction.preferredTapAction == .switchPrevious {
        leftRingControl.switchToPreviousTool()
    }
}
```

In order for Speed Sketch to receive the double-tap event, it adds a `UIPencilInteraction` object to the canvas view.

```
let pencilInteraction = UIPencilInteraction()
pencilInteraction.delegate = self
view.addInteraction(pencilInteraction)
```

For additional information on supporting touch input in drawing apps and Apple Pencil, watch the WWDC 2016 session video Leveraging Touch Input on iOS and the Tech Talks session video Designing for iPad Pro and Apple Pencil.

See Also

Touches

Handling touches in your view

Use touch events directly on a view subclass if touch handling is intricately linked to the view’s content.

Handling input from Apple Pencil

Learn how to detect and respond to touches from Apple Pencil.

Tracking the force of 3D Touch events

Manipulate your content based on the force of touches.

{} Illustrating the force, altitude, and azimuth properties of touch input

Capture Apple Pencil and touch input in views.

`class UITouch`

An object representing the location, size, movement, and force of a touch occurring on the screen.