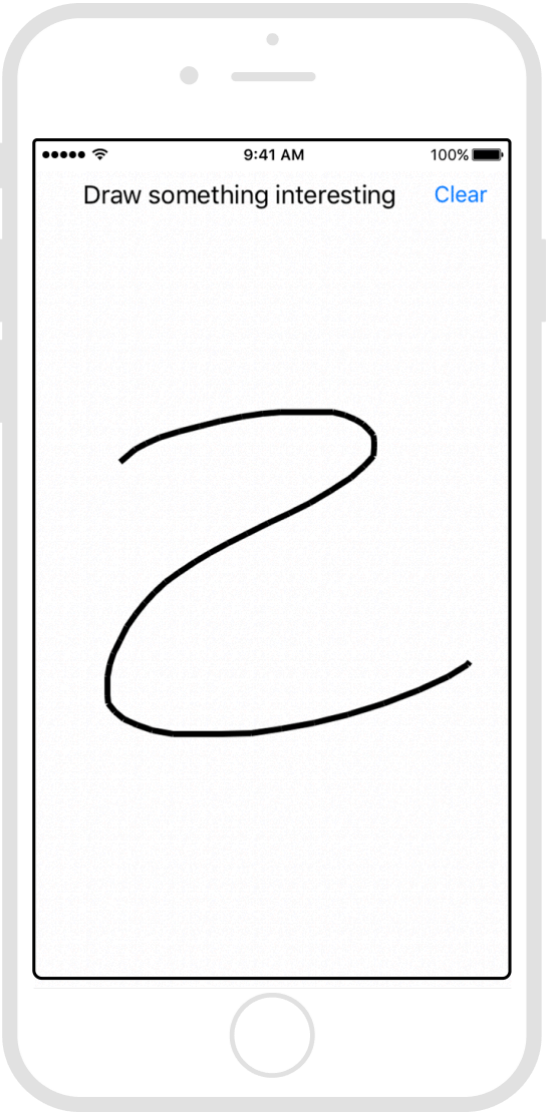
Article

Implementing coalesced touch support in an app

Learn how to create a simple app that handles coalesced touches.

Overview

The following image shows a simple drawing app that captures touches and renders the resulting path onscreen. The app tracks all touches reported by UIKit, including coalesced touches. The app builds the path by drawing line segments from one touch point to the next.



Provide storage for the touches

The main view of the app uses incoming touch events to build a set of `Stroke` objects. The following image shows the definition of the `Stroke` class and the associated `StrokeSample` class, which store information about each touch event.

```
class Stroke {
    var samples = [StrokeSample]()
    func add(sample: StrokeSample) {
        samples.append(sample)
    }
}

struct StrokeSample {
    let location: CGPoint
    let coalescedSample: Bool
    init(point: CGPoint, coalesced : Bool = false) {
        location = point
```

```
            coalescedSample = coalesced
        }
    }
```

The main view maintains a collection of `Stroke` objects that have been created using the `StrokeCollection` class, the implementation of which is shown in the following code. The `strokes` property of this class stores the completed strokes and the `activeStroke` property contains a stroke object that's currently being modified. Calling the `acceptActiveStroke` method moves the active stroke to the set of completed strokes.

```
class StrokeCollection {
    var strokes = [Stroke]()
    var activeStroke: Stroke? = nil

    func acceptActiveStroke() {
        if let stroke = activeStroke {
            strokes.append(stroke)
            activeStroke = nil
        }
    }
}
```

Retrieve the coalesced touches

The following code shows the portion of the main drawing view that creates new `Stroke` objects. The view doesn't support multitouch, so only the first touch event needs to be tracked. The `touchesBegan(_:with:)` method creates a new stroke object and marks it as the active stroke. New touch data is added to the active stroke until the `touchesEnded(_:with:)` method is called, at which point the stroke is accepted into the stroke collection. If the touch sequence is interrupted for any reason, the `touchesCancelled(_:with:)` method abandons the currently active stroke.

```
class DrawingView : UIView {
   var strokeCollection: StrokeCollection? {
      didSet {
         // If the strokes change, redraw the view's content.
         if oldValue !== strokeCollection {
            setNeedsDisplay()
         }
      }
```

```
    }

    // Initialization methods...

    // Touch Handling methods
    override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
        // Create a new stroke and make it the active stroke.
        let newStroke = Stroke()
        strokeCollection?.activeStroke = newStroke

        // The view does not support multitouch, so get the samples
        //  for only the first touch in the event.
        if let coalesced = event?.coalescedTouches(for: touches.first!) {
            addSamples(for: coalesced)
        }
    }

    override func touchesMoved(_ touches: Set<UITouch>, with event: UIEvent?) {
        if let coalesced = event?.coalescedTouches(for: touches.first!) {
            addSamples(for: coalesced)
        }
    }

    override func touchesEnded(_ touches: Set<UITouch>, with event: UIEvent?) {
        // Accept the current stroke and add it to the stroke collection.
        if let coalesced = event?.coalescedTouches(for: touches.first!) {
            addSamples(for: coalesced)
        }
        // Accept the active stroke.
        strokeCollection?.acceptActiveStroke()
    }

    override func touchesCancelled(_ touches: Set<UITouch>, with event: UIEvent?) {
        // Clear the last stroke.
        strokeCollection?.activeStroke = nil
    }

    // More methods...
```

The touch input methods of `DrawingView` use the `addSamples` method (shown in the following code) to incorporate new touches into the active stroke. This method creates a new `Stroke Sample` for each touch point and adds that sample to the active stroke. The example flags

coalesced touches internally, but the touches are no different from the regular touches reported by the system.

```
func addSamples(for touches: [UITouch]) {
   if let stroke = strokeCollection?.activeStroke {
      // Add all of the touches to the active stroke.
      for touch in touches {
         if touch == touches.last {
            let sample = StrokeSample(point: touch.preciseLocation(in: self))
            stroke.add(sample: sample)
         } else {
            // If the touch is not the last one in the array,
            //  it was a coalesced touch.
            let sample = StrokeSample(point: touch.preciseLocation(in: self),
                                      coalesced: true)
            stroke.add(sample: sample)
         }
      }
      // Update the view.
      self.setNeedsDisplay()
   }
}
```

> **Note**
>
> When capturing drawing input from Apple Pencil, you can use the `preciseLocation(in:)` method instead of the `location(in:)` method to get more precise touch information. Use the `preciseLocation(in:)` method only for capturing drawing-related input. For general interactions with your interface, continue to get the touch location using the `location(in:)` method.

The remaining methods of the `DrawingView` class take the touch samples and turn them into the rendered output. The app's Clear button releases the view's current `StrokeCollection` object and creates a new one.