Leveraging Touch Input on iOS

. . . And getting the most out of Apple Pencil

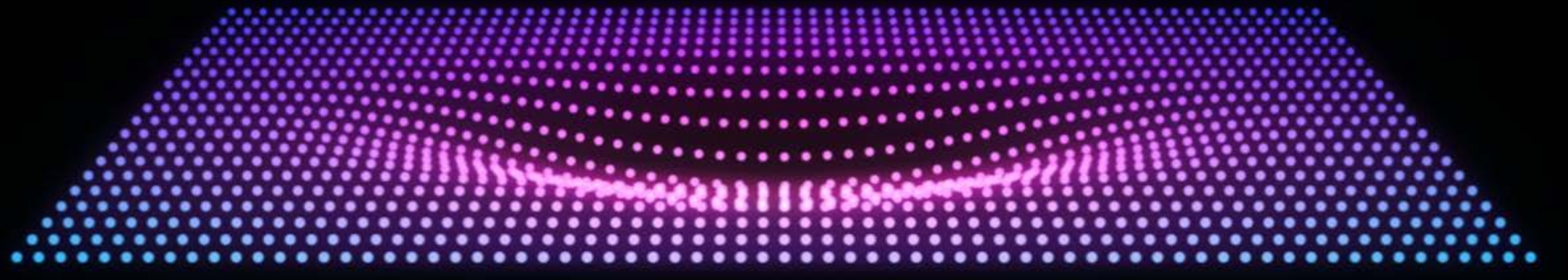
Session 220

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

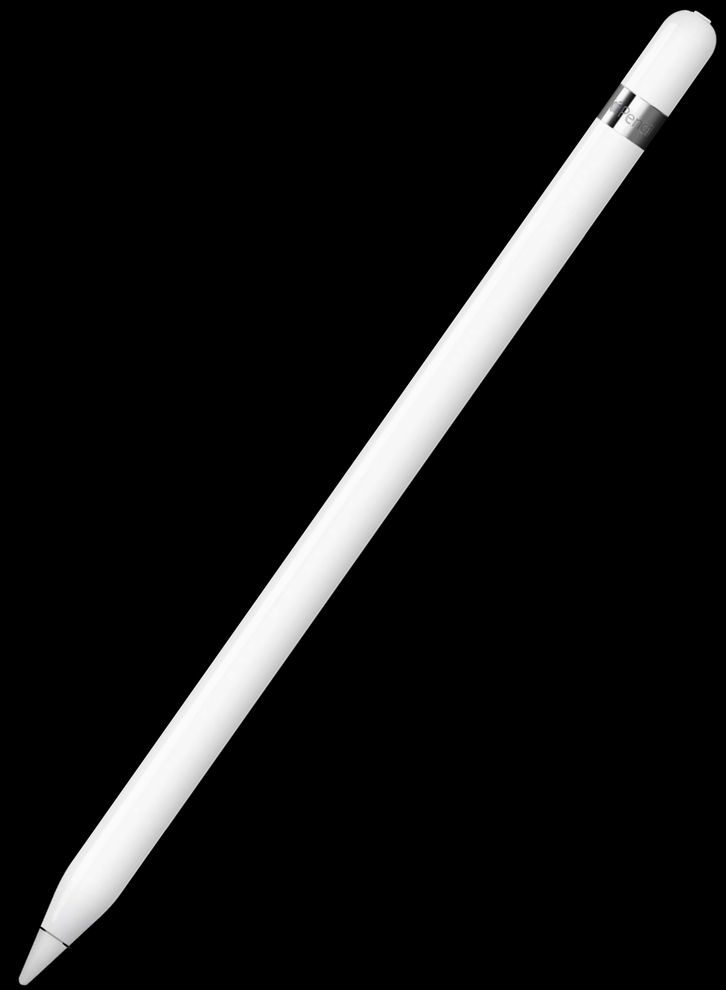
iPad Pro

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

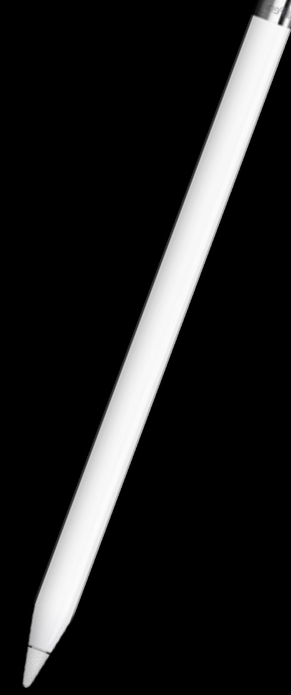
Say Hello to SpeedSketch

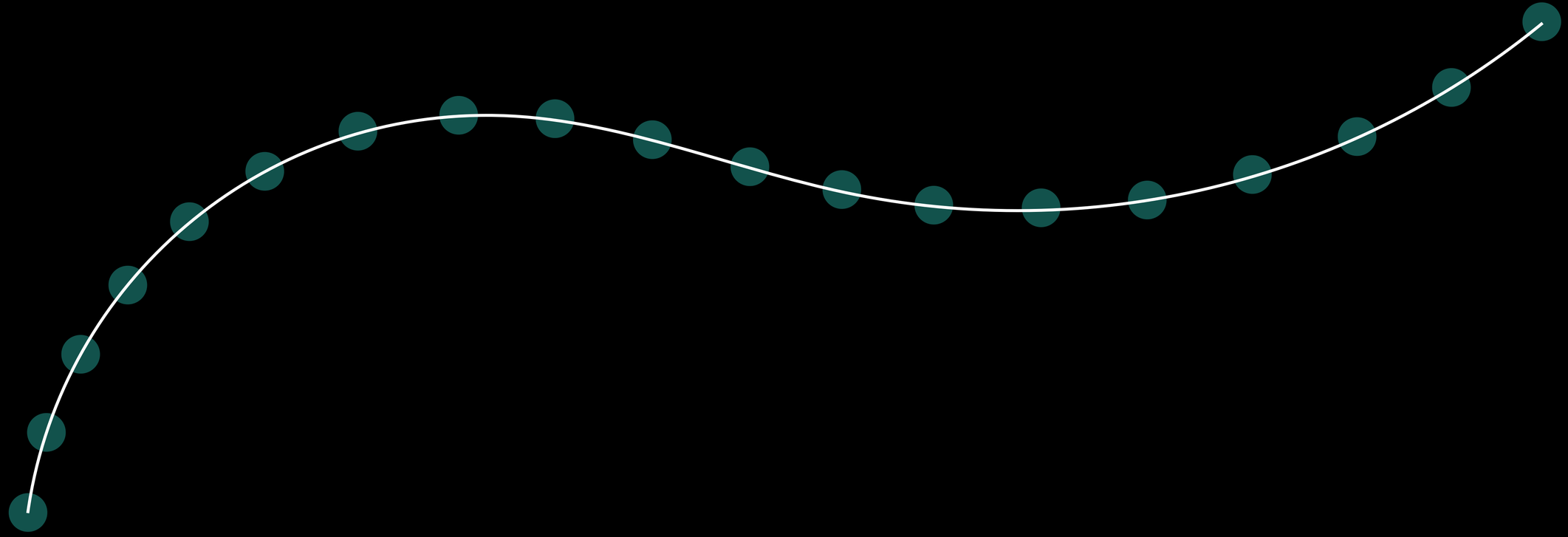
One sheet of paper you can draw on

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

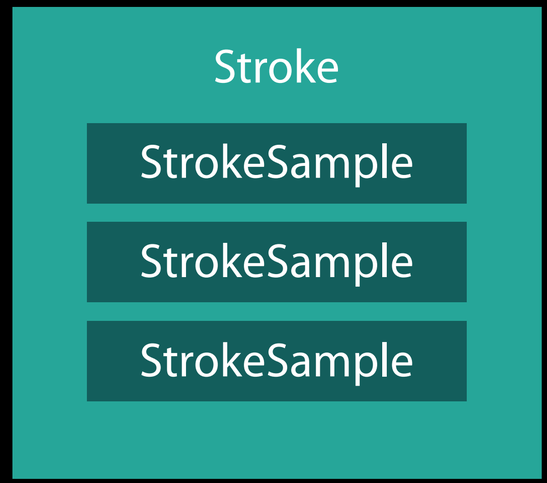
```
struct StrokeSample {
    let location: CGPoint
}
```

StrokeSample

Building a Drawing App

Model and capture

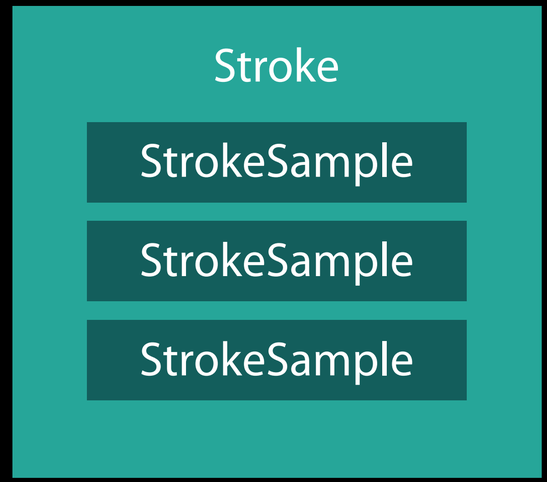
```
class Stroke {
    var samples: [StrokeSample] = []



    func add(sample: StrokeSample)
}
```



Building a Drawing App

Model and capture

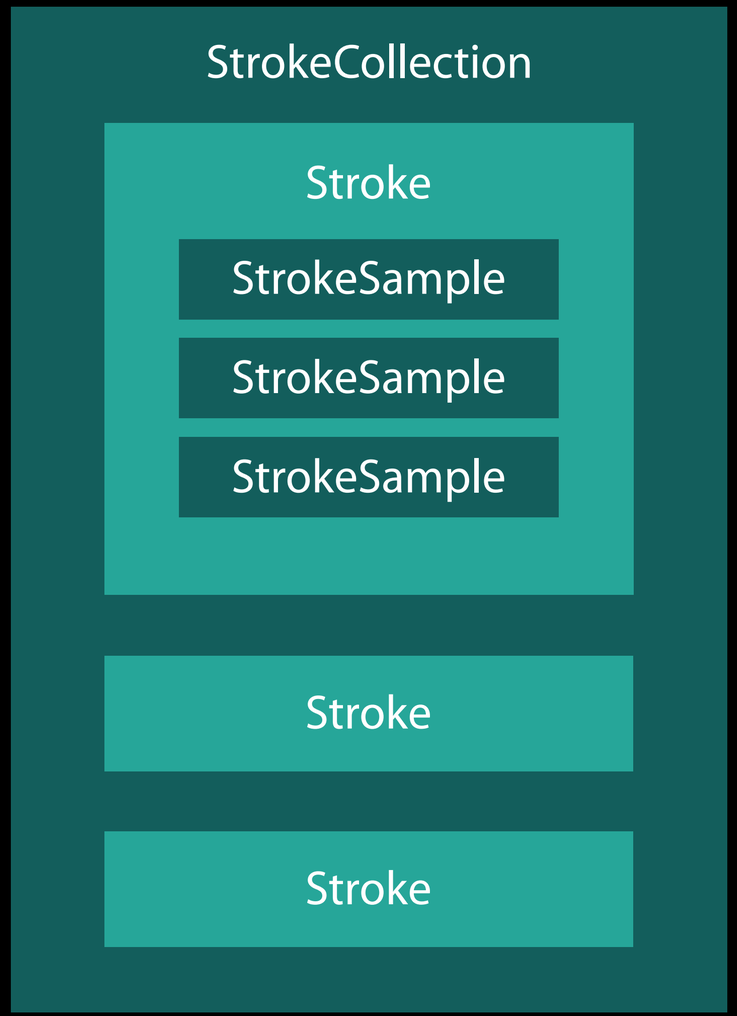
```
class Stroke {
    var samples: [StrokeSample] = []
    var state: StrokeState = .active

    func add(sample: StrokeSample)
}

enum StrokeState {
    case active
    case done
    case cancelled
}
```



Building a Drawing App

Model and capture

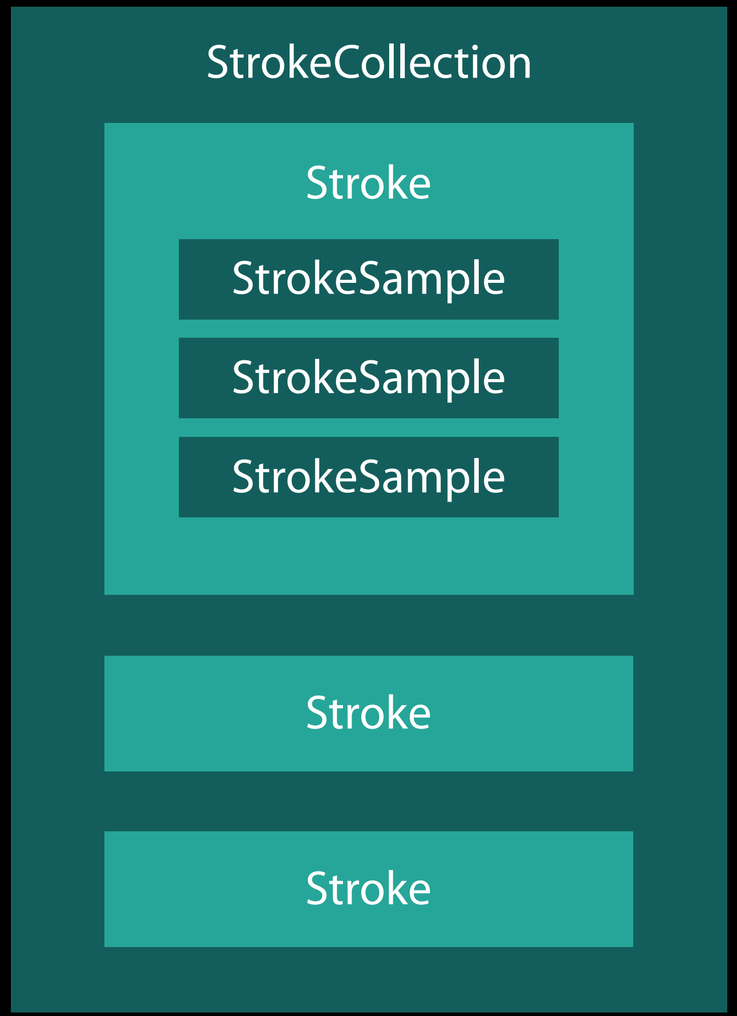
```
class StrokeCollection {
    var strokes: [Stroke] = []

    func add(doneStroke: stroke)
}
```



Building a Drawing App

Model and capture

```
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

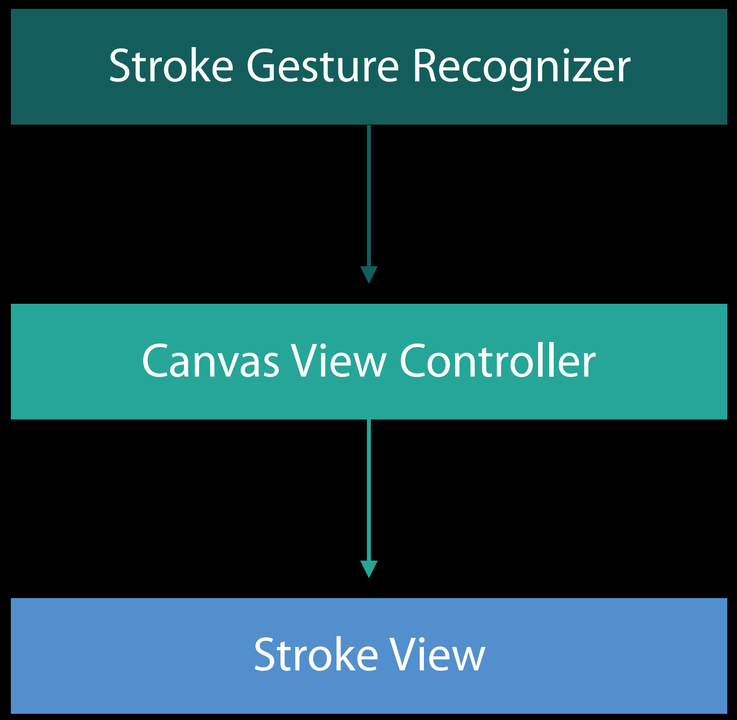
Building a Drawing App

Capture

Custom UIGestureRecognizer

Targeting the main view controller

View controller facilitates update



```
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?) {


    }

}
```

```swift
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?) {



    }

}
```

```swift
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
        }
    }

}
```

```swift
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
        }
    }









}
```

```swift
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?) …




}
```

```swift
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()
    }
}
```

```swift
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()

    }

}
```

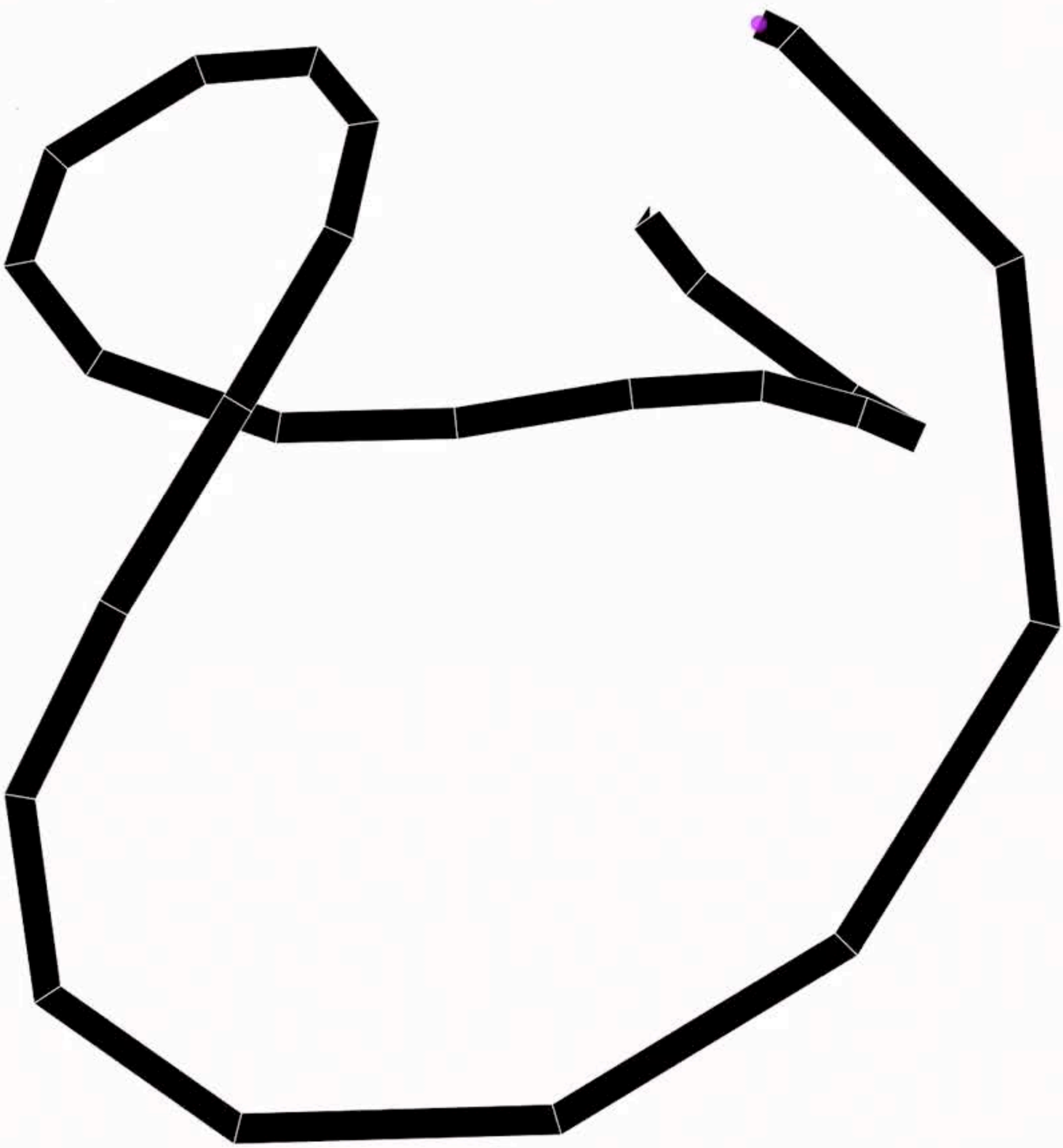
```
class CanvasViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

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

        view.addGestureRecognizer(strokeRecognizer)
    }



}
```

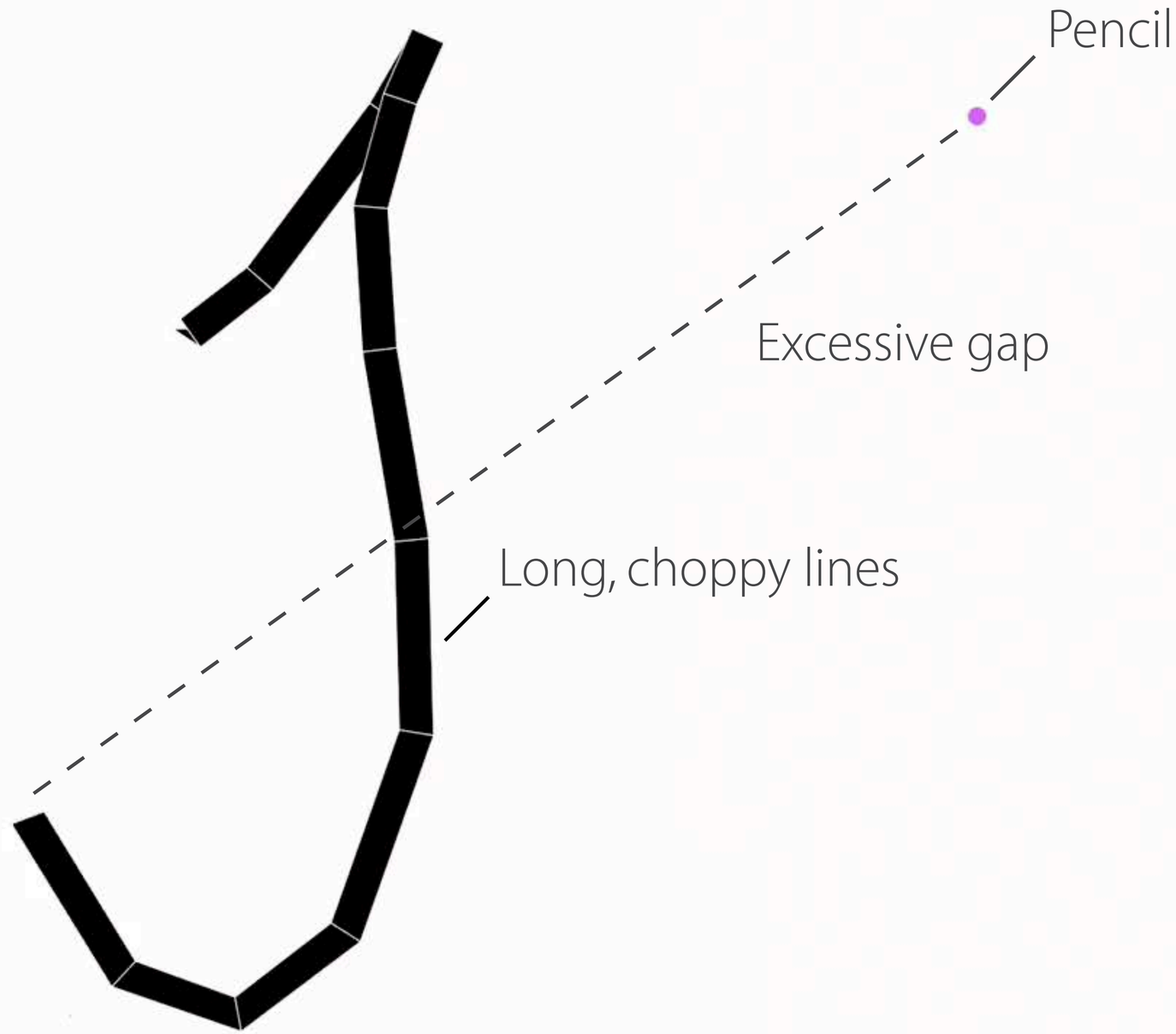
```
class CanvasViewController: UIViewController {

    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



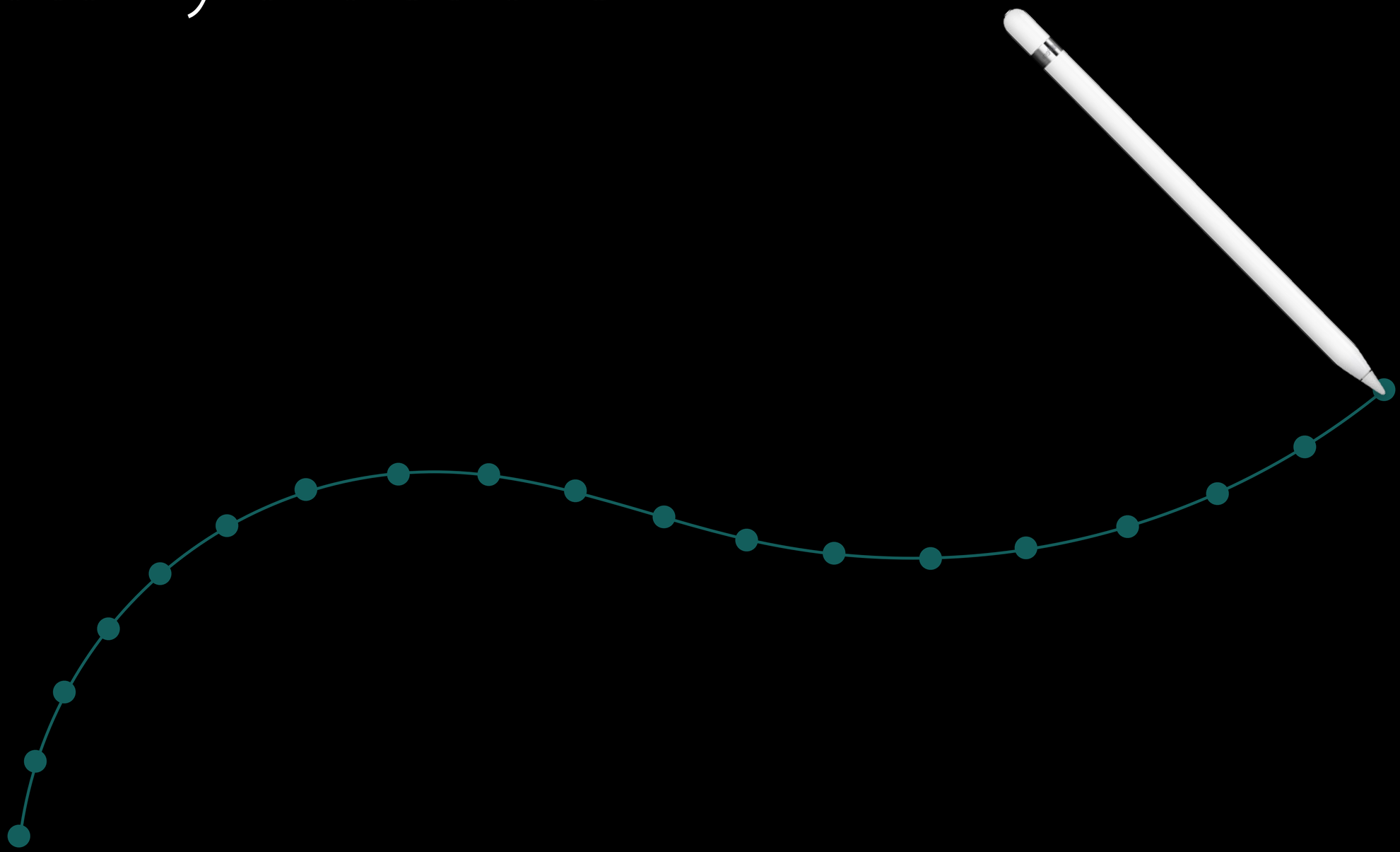

Pencil
Excessive gap
Long, choppy lines

Analysis of First Attempt

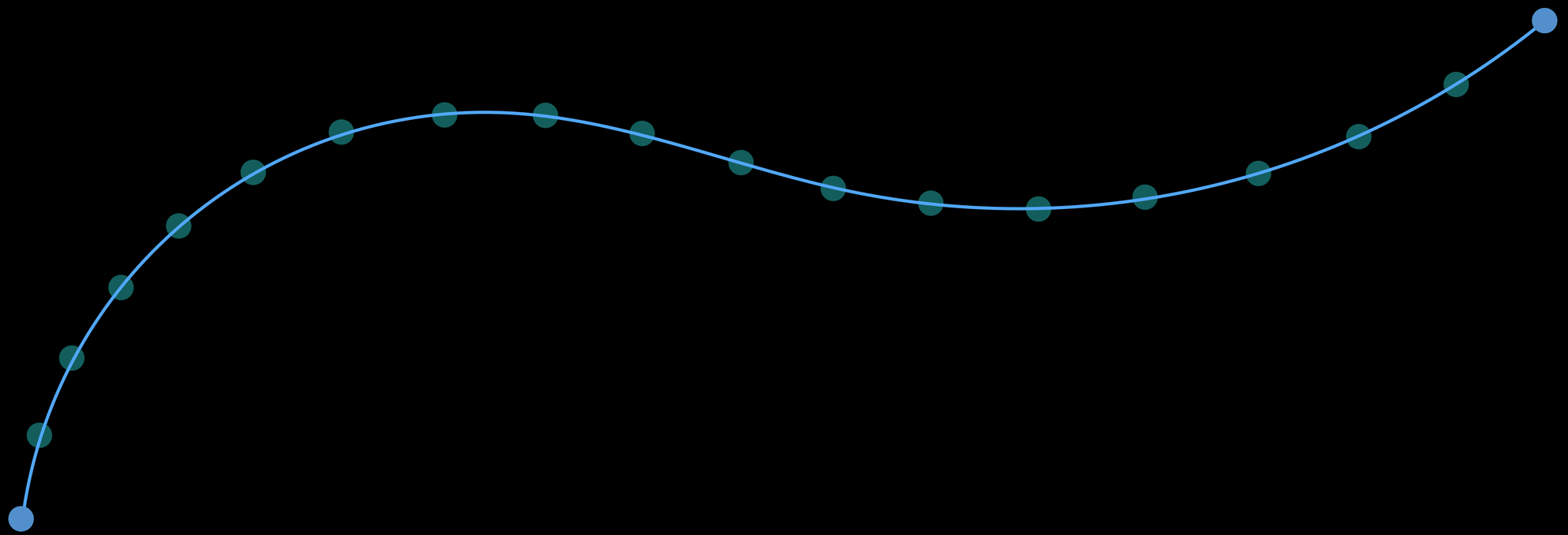
Missed events

- Drawing engine
- Did not use the new iOS 9.0 API

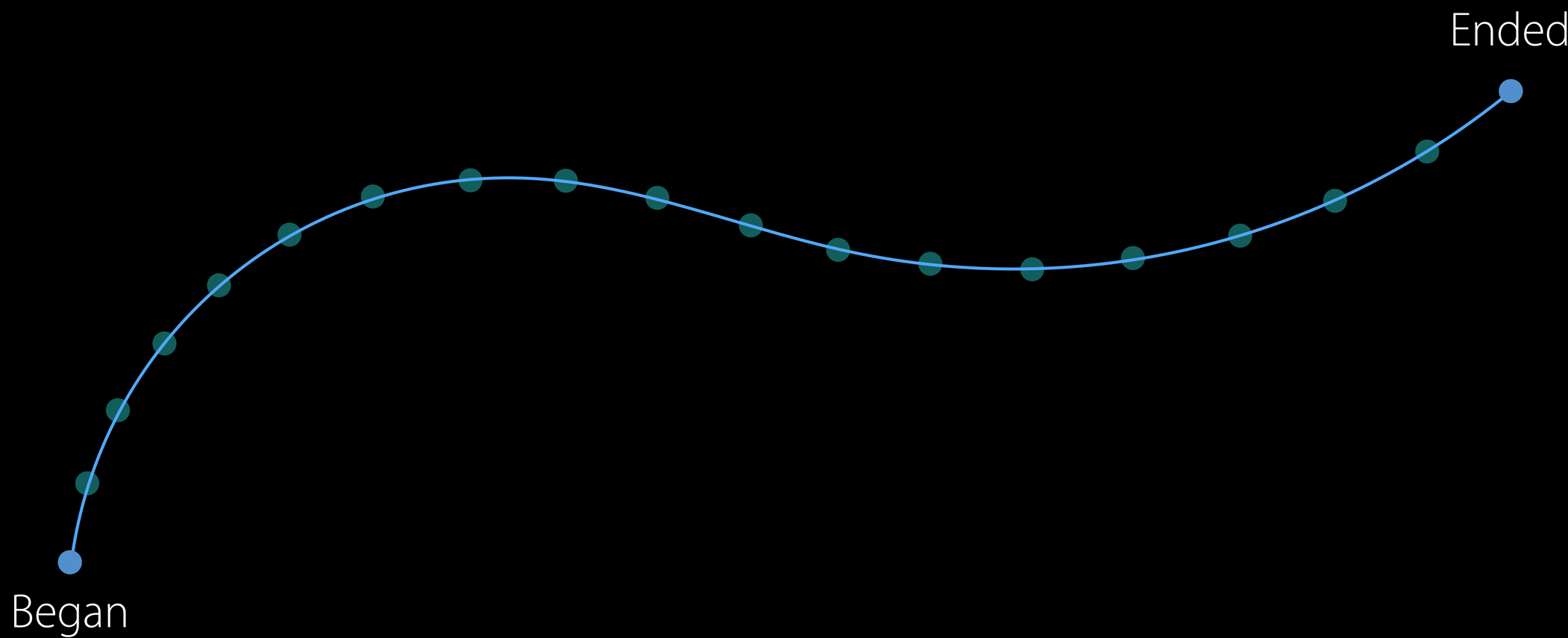
Anatomy of a Stroke



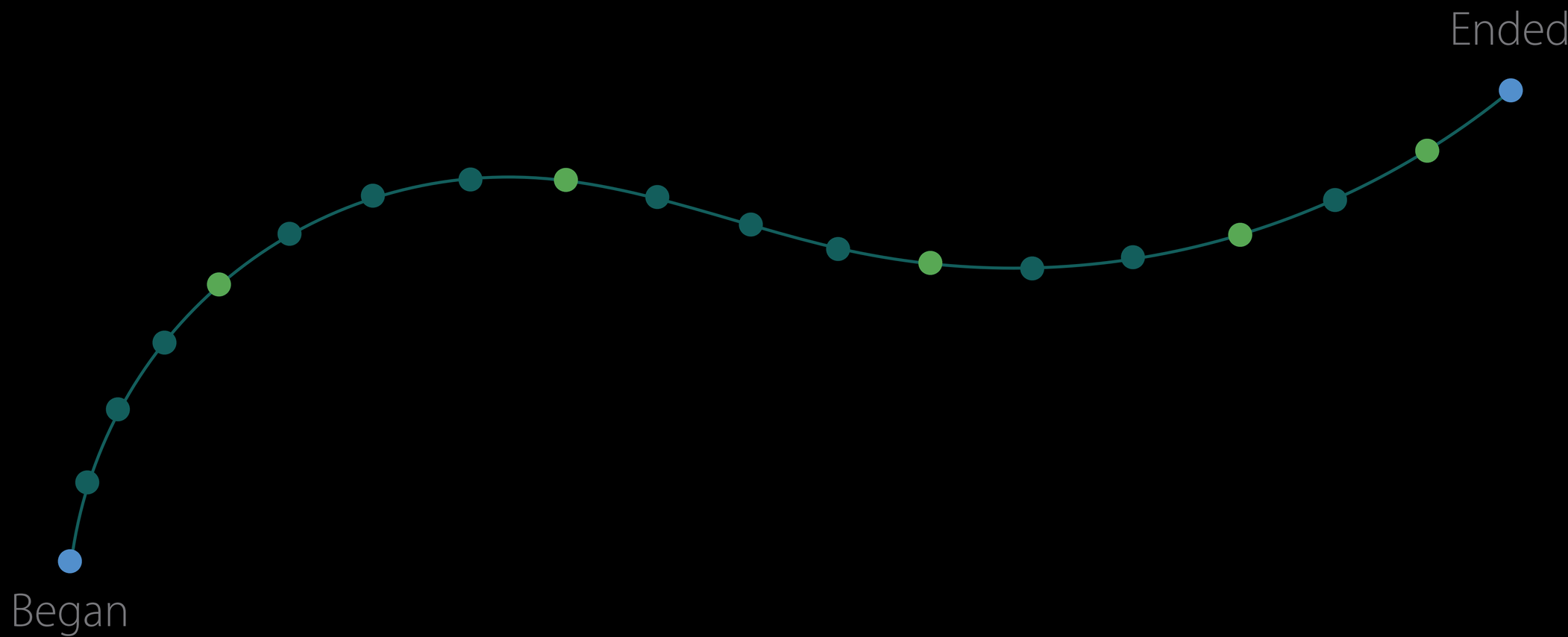
Anatomy of a Stroke



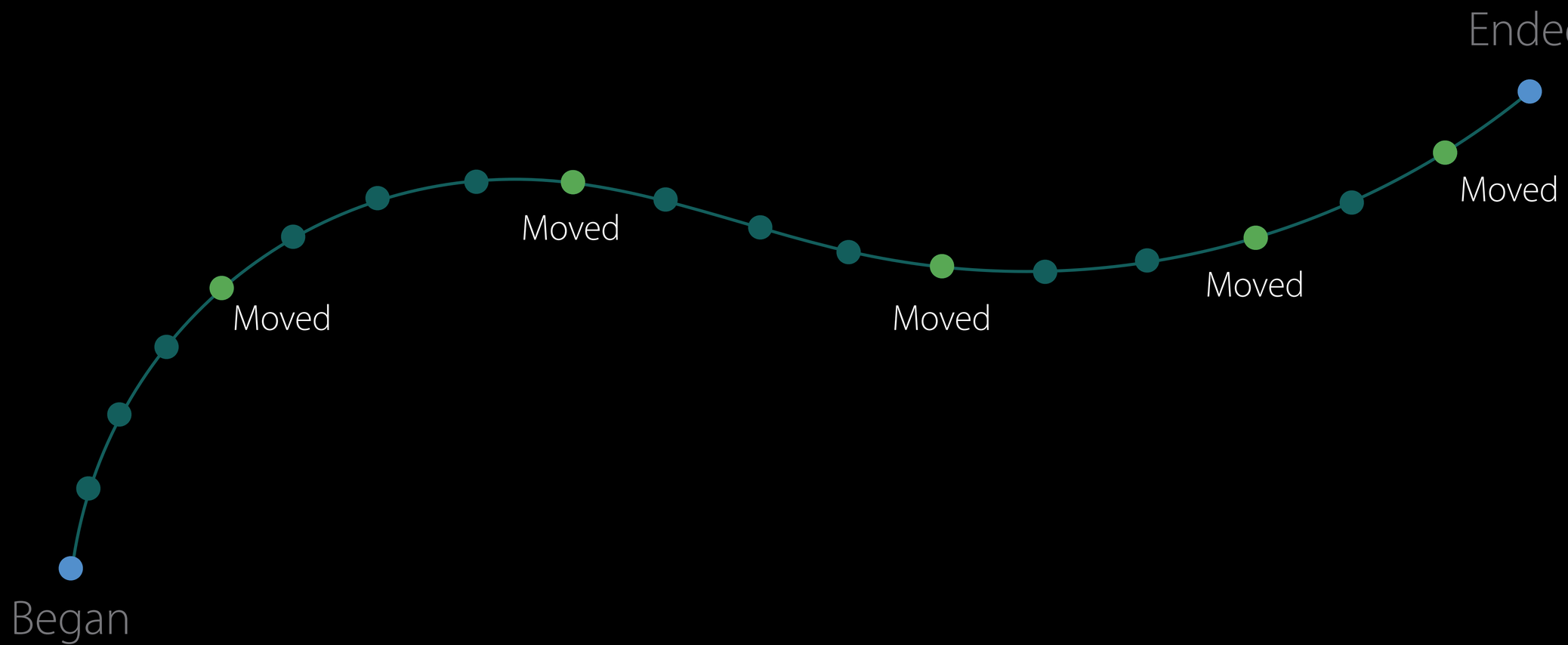
Anatomy of a Stroke



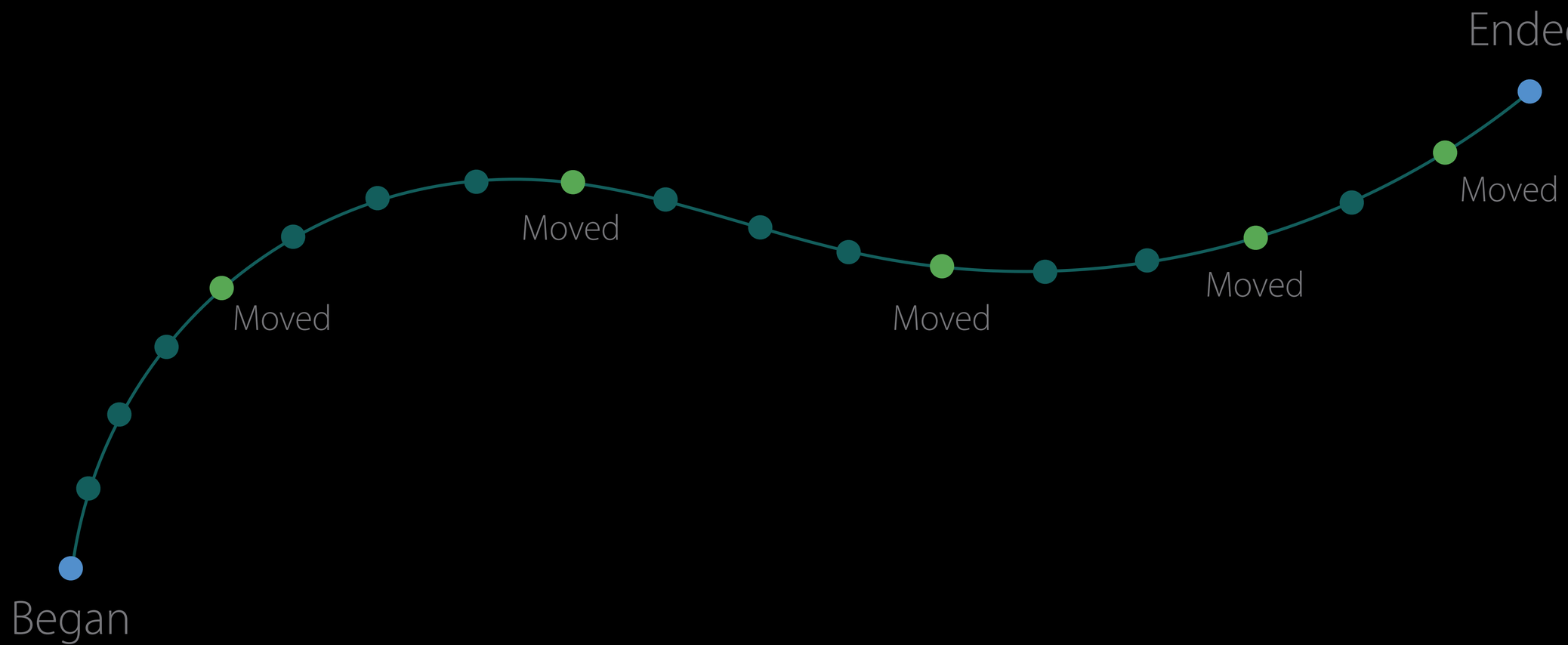
Anatomy of a Stroke



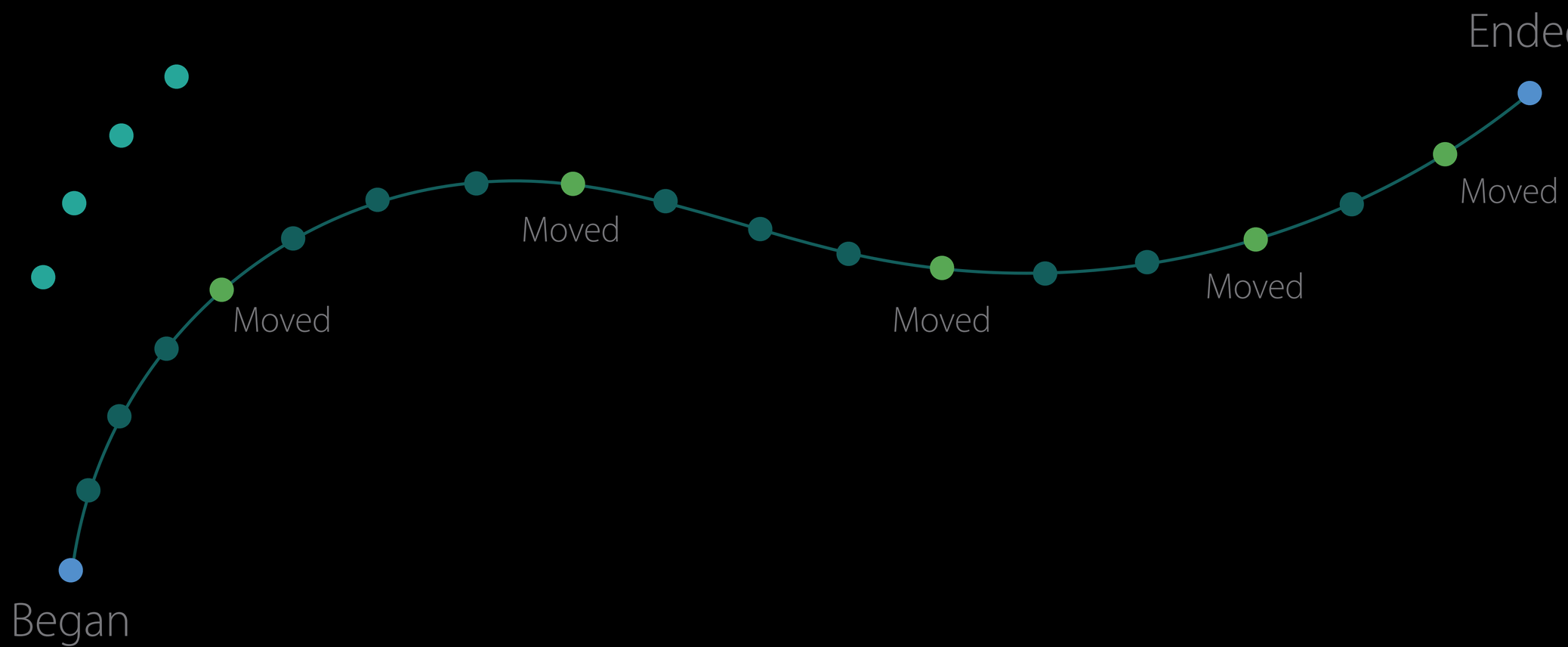
Anatomy of a Stroke



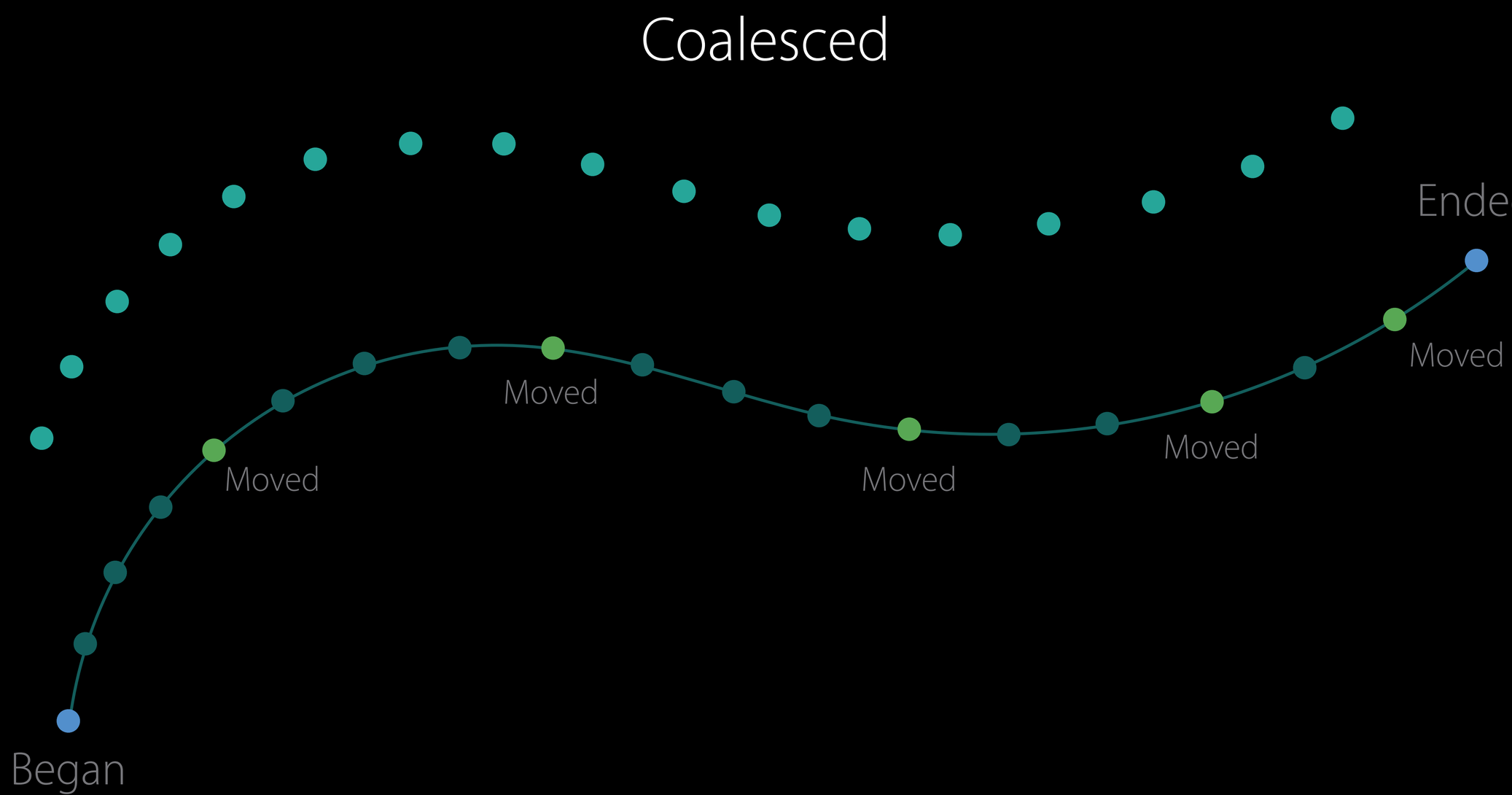
Anatomy of a Stroke



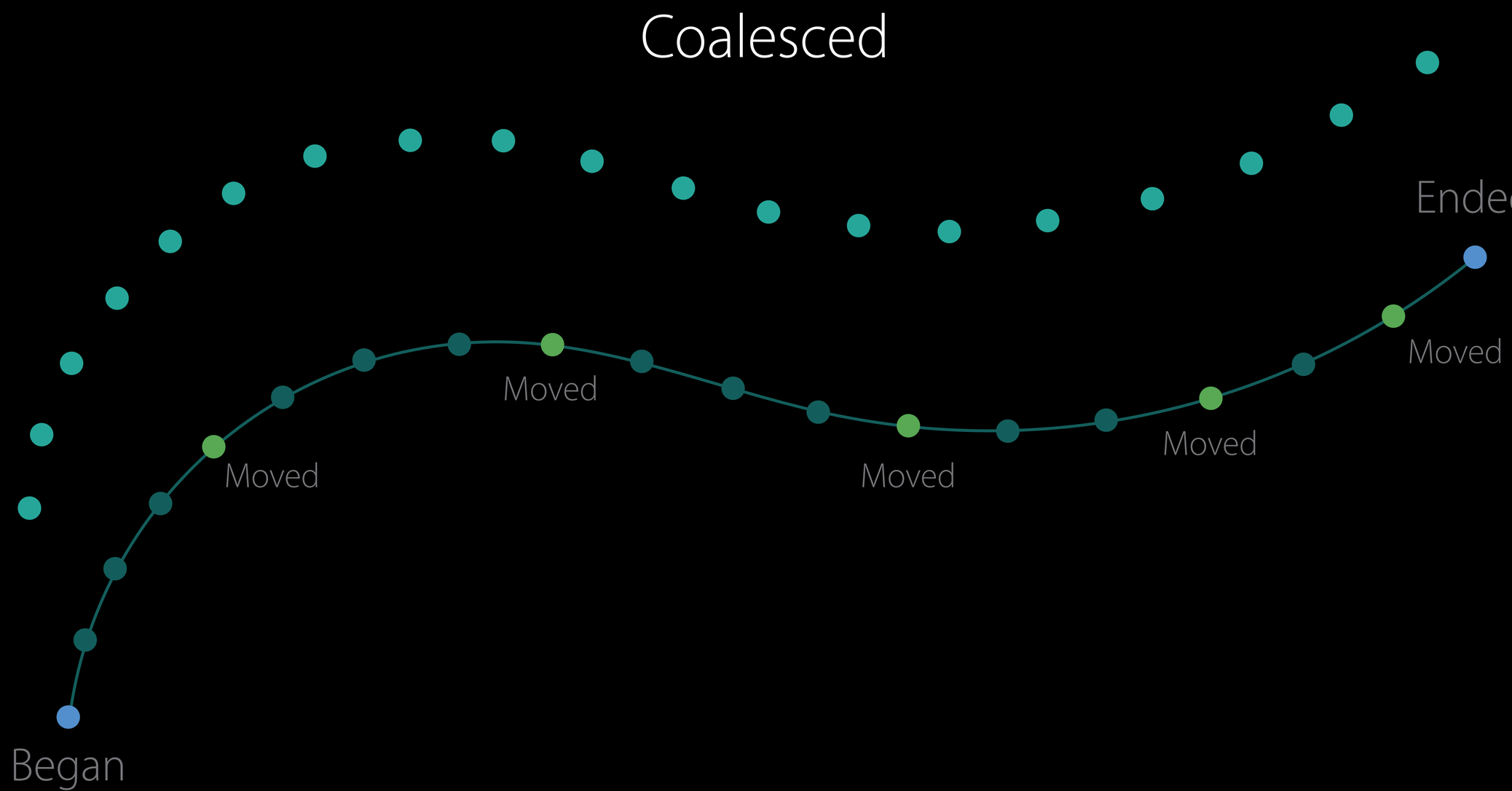
Anatomy of a Stroke



Anatomy of a Stroke



Anatomy of a Stroke



Coalesced Touches

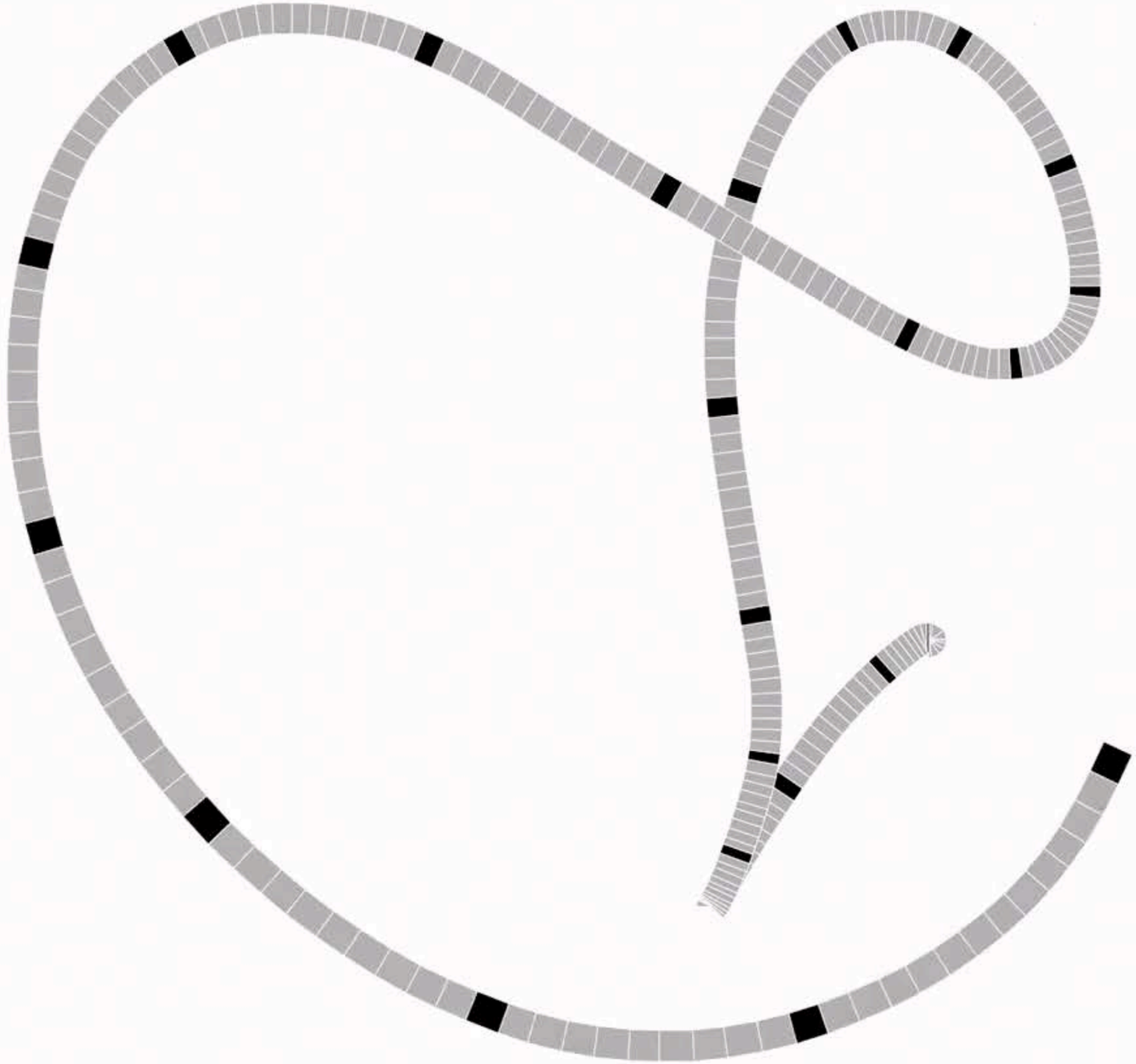
```
class UIEvent {
   public func coalescedTouches(for touch: UITouch) -> [UITouch]?
}
```

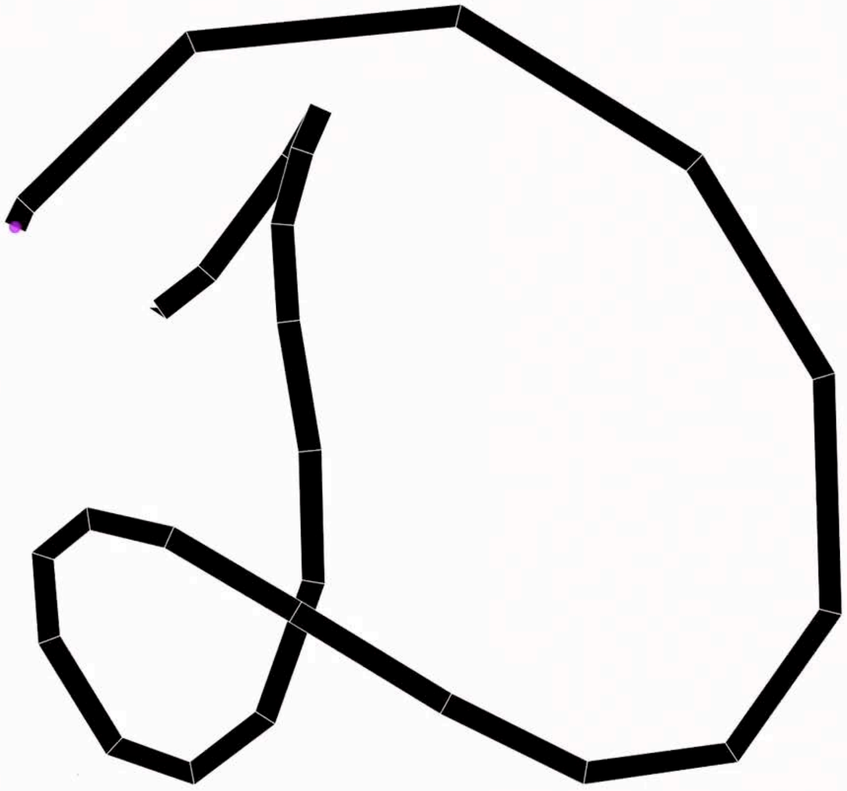
Coalesced Touches

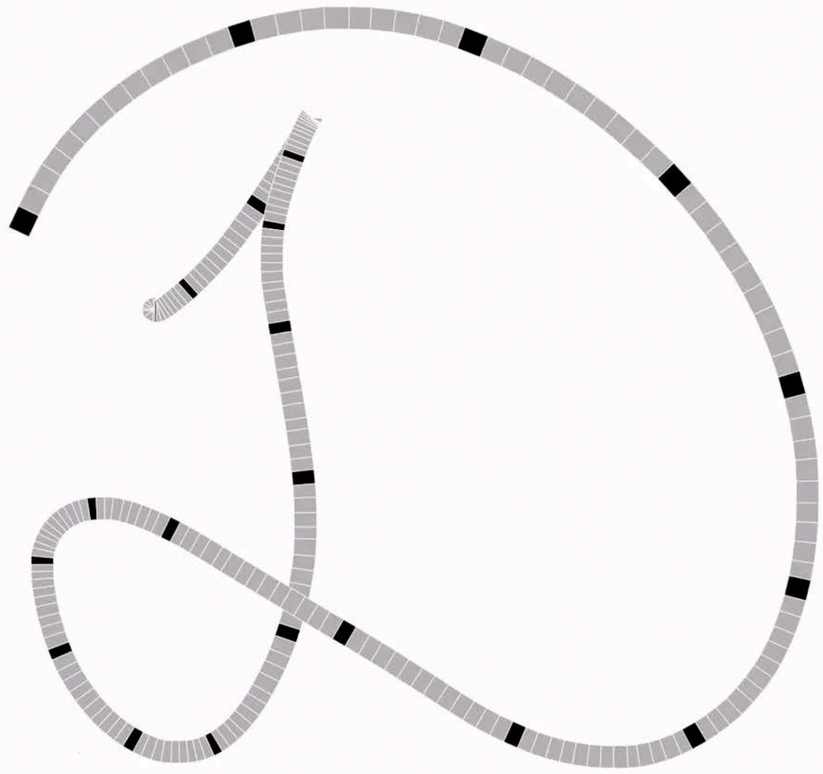
```
class StrokeGestureRecognizer: UIGestureRecognizer {
   override func touchesMoved(_ touches: Set<UITouch>, with event: UIEvent?) {
        if let touch = touches.first {

            appendTouch(touch)

        }
    }
}
```

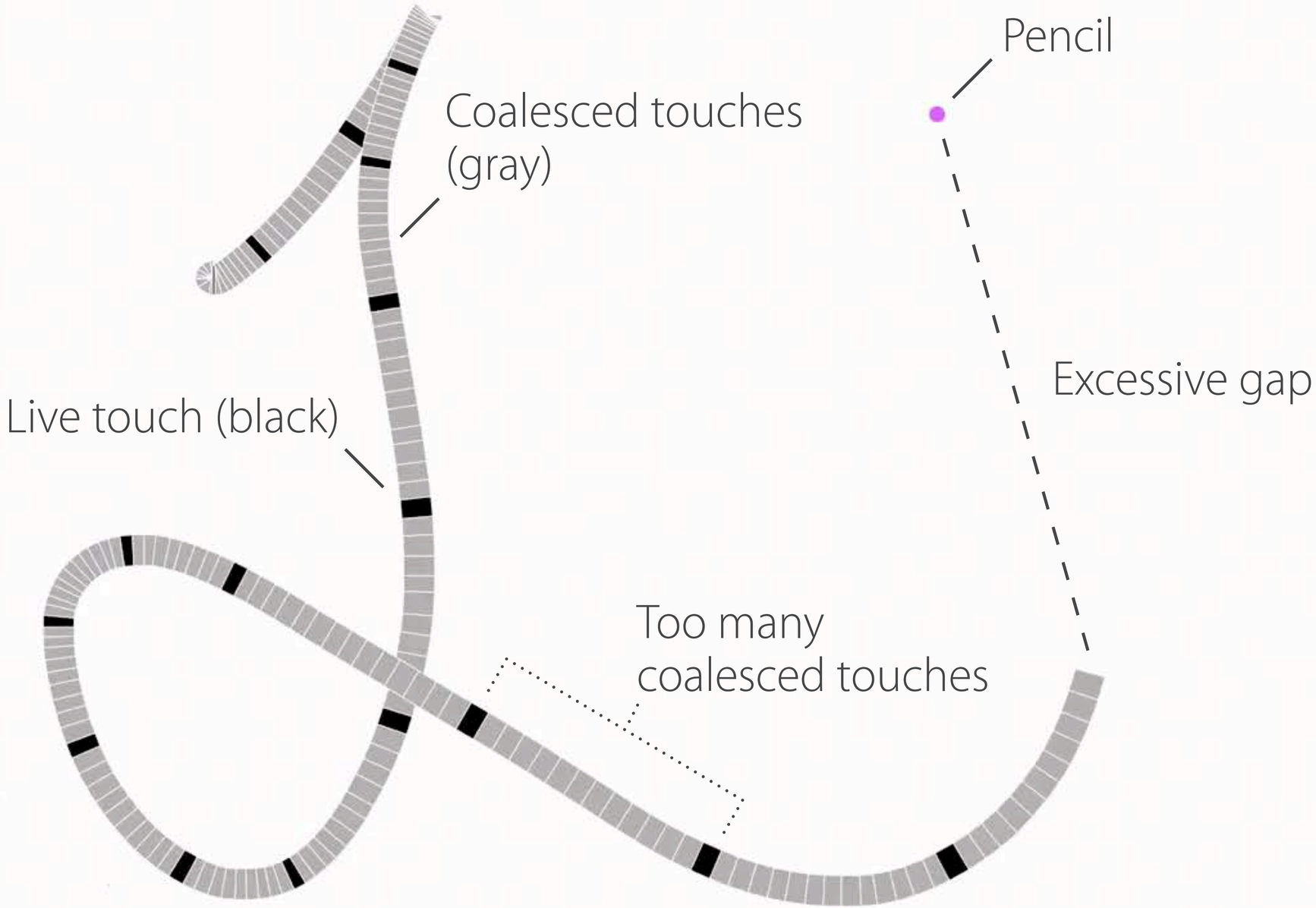
Coalesced Touches

```
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)
            }
        }
    }
}
```








Coalesced touches
(gray)
Pencil
Excessive gap
Live touch (black)
Too many
coalesced touches

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()
        }
    }


}
```

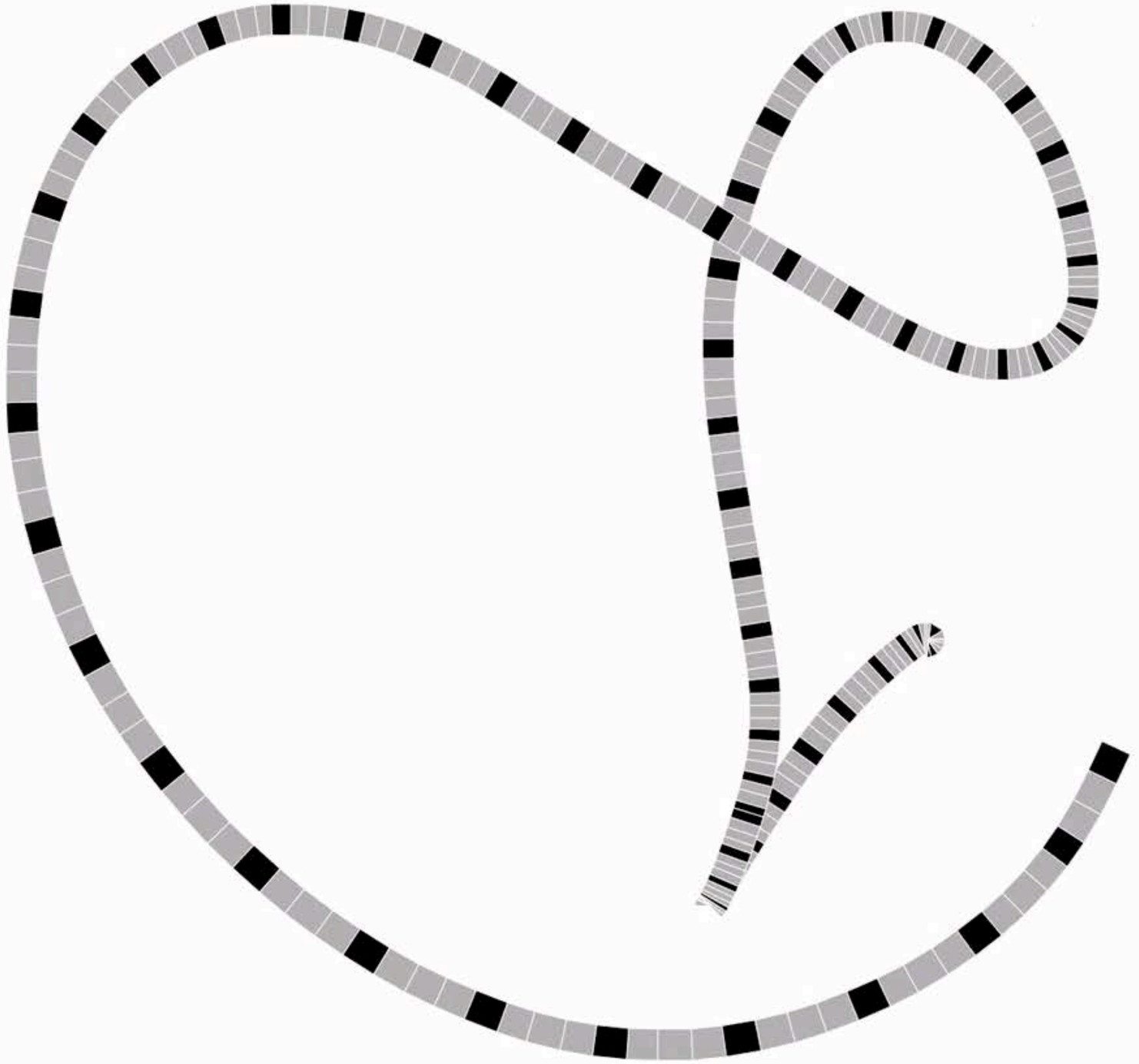
Even Better

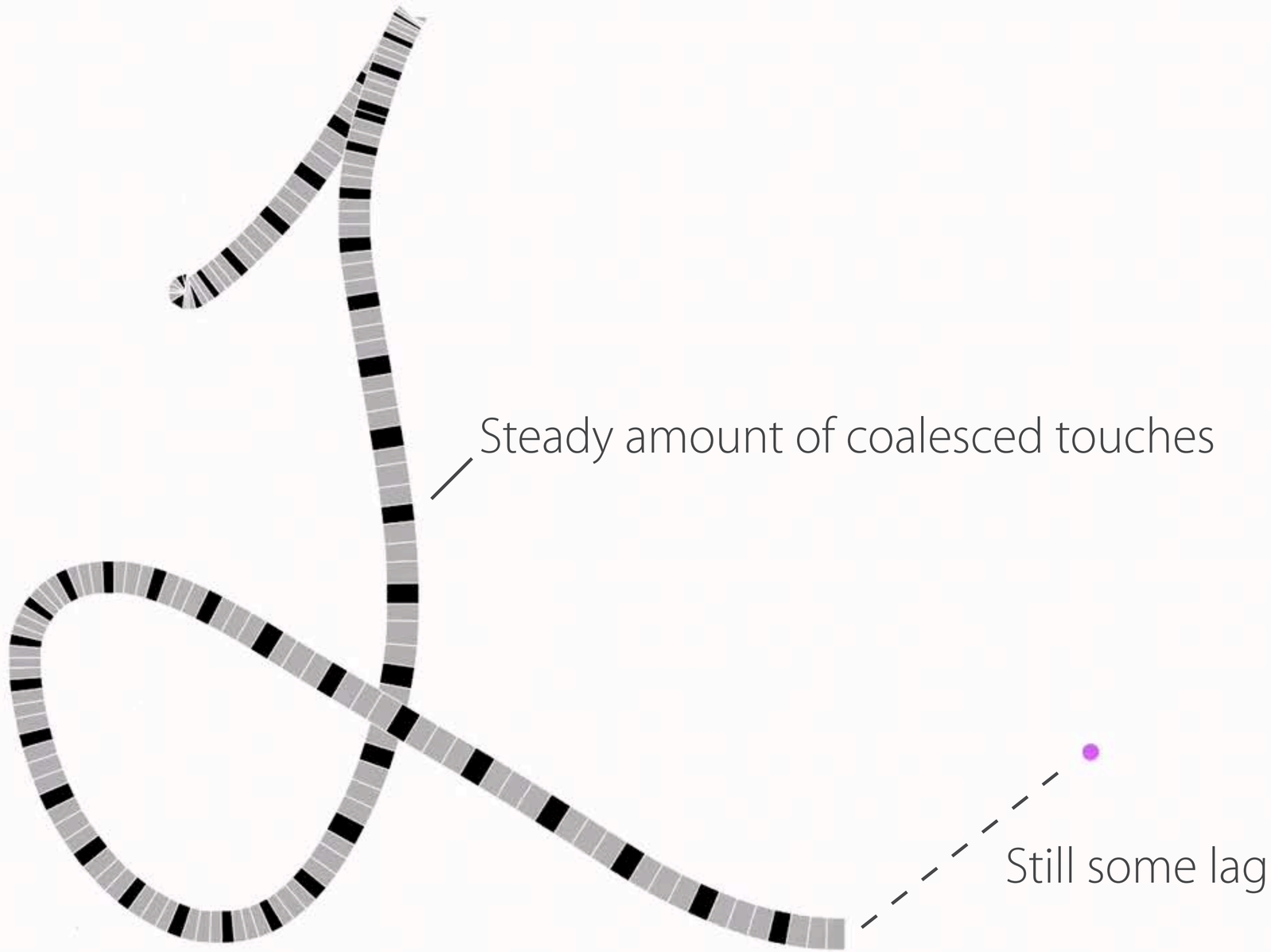
Mark only the changed areas

- `setNeedsDisplayIn(rect: changedRect)`

Activate `drawsAsynchronously` on the layer

```
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

Predicted Touches

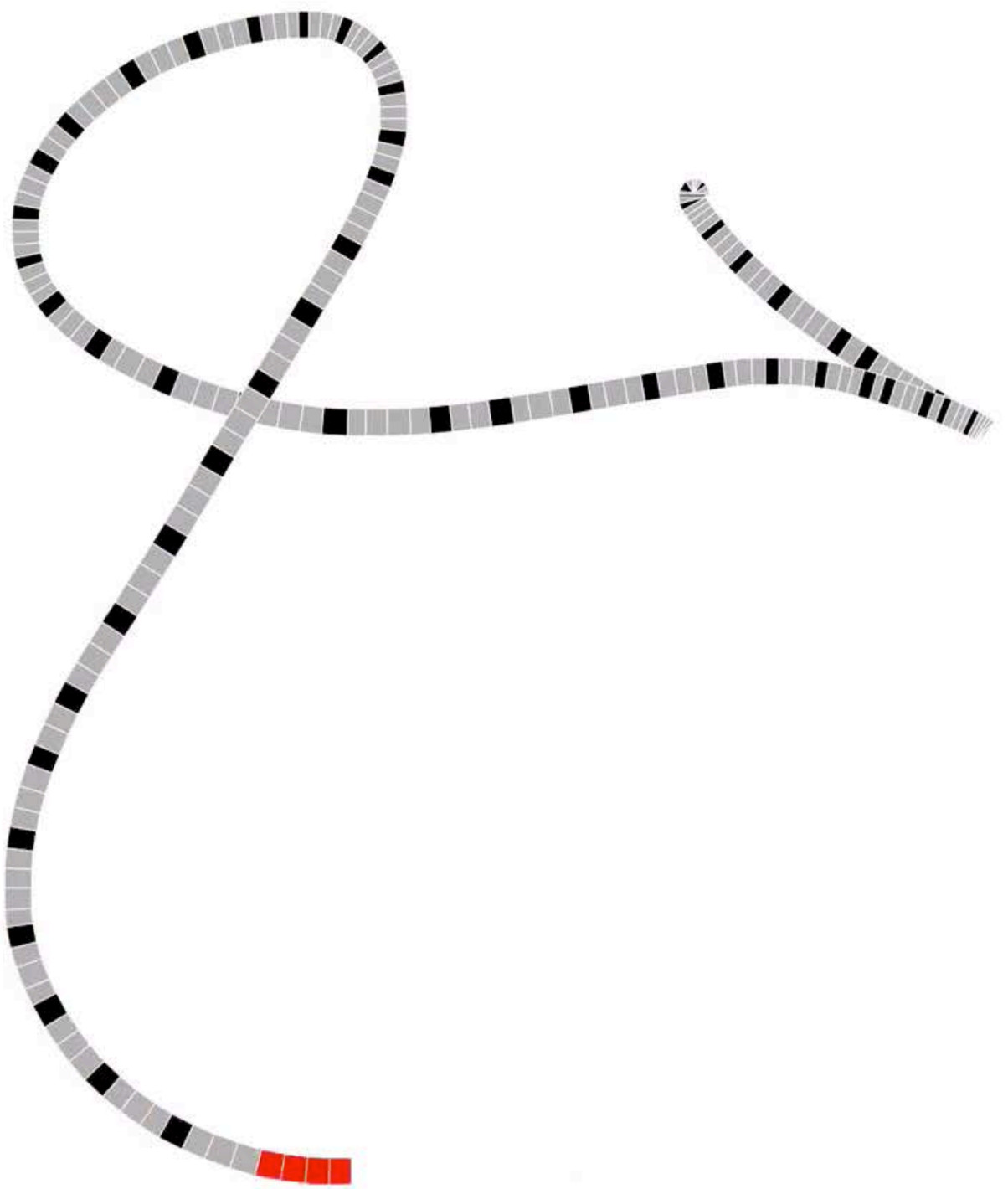
```
class StrokeGestureRecognizer: UIGestureRecognizer {
    override func touchesMoved(_ touches: Set<UITouch>, with event: UIEvent?) {
        if let touch = touches.first {

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




        }
    }
}
```

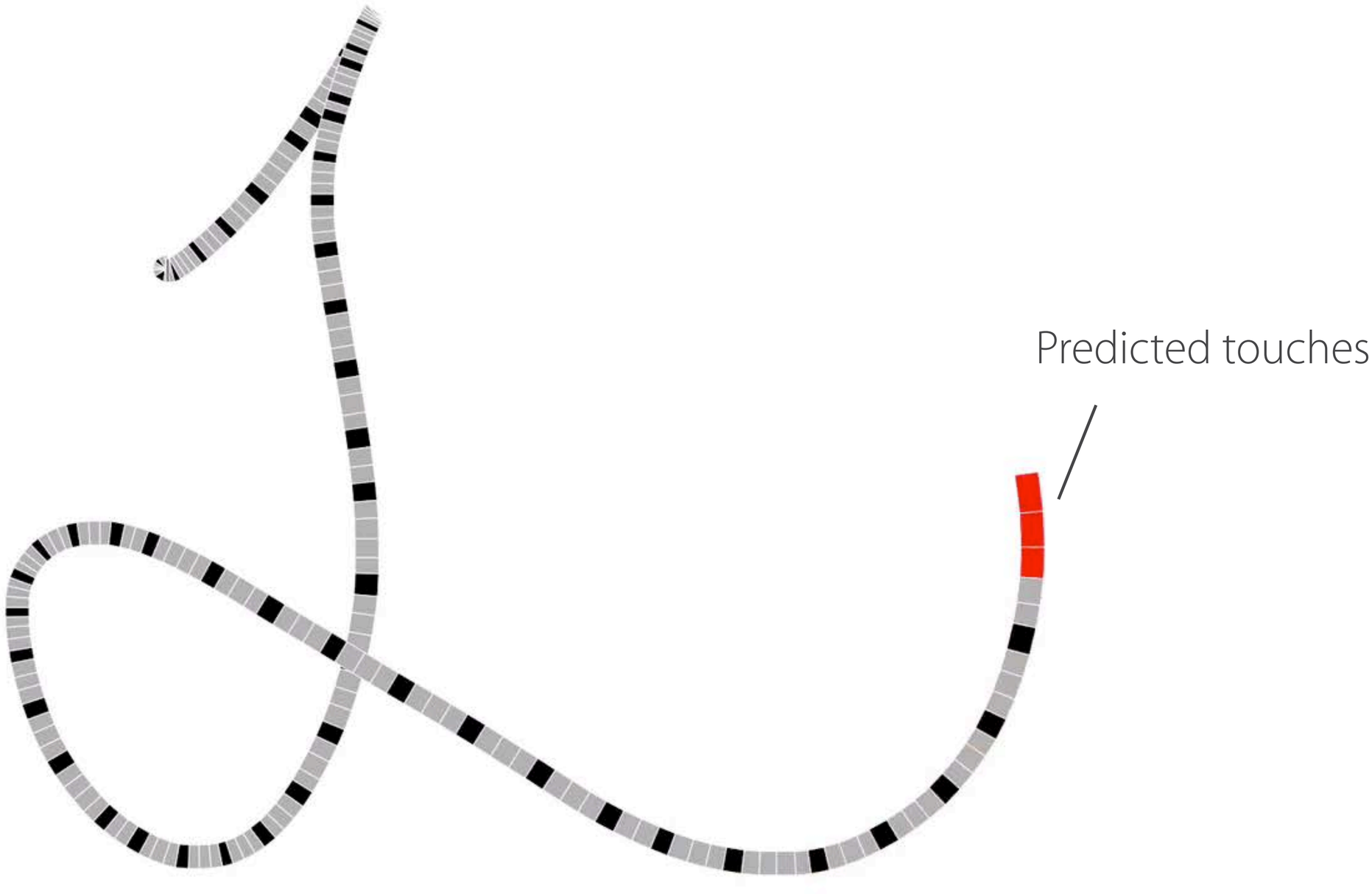
Predicted Touches

```
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

```
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

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

Apple Pencil

Touch types

```
public var type: UITouchType { get }

enum UITouchType : Int {
   case direct
   case indirect
   case stylus
}
```

Apple Pencil

Higher precision

```
func preciseLocation(in view: UIView?) -> CGPoint
func precisePreviousLocation(in view: UIView?) -> CGPoint
```

Apple Pencil and 3D Touch

Force

```
public var force: CGFloat { get }
public var maximumPossibleForce: CGFloat { get }
```

Device	Range
iPhone 6s (Plus)	0.0 - `maximumPossibleForce`
Apple Pencil on iPad Pro	0.0 - `maximumPossibleForce`
Touch on iPad Pro and all previous devices	0.0

Apple Pencil and 3D Touch

Force

```
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

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

Use UITapGestureRecognizer

Force

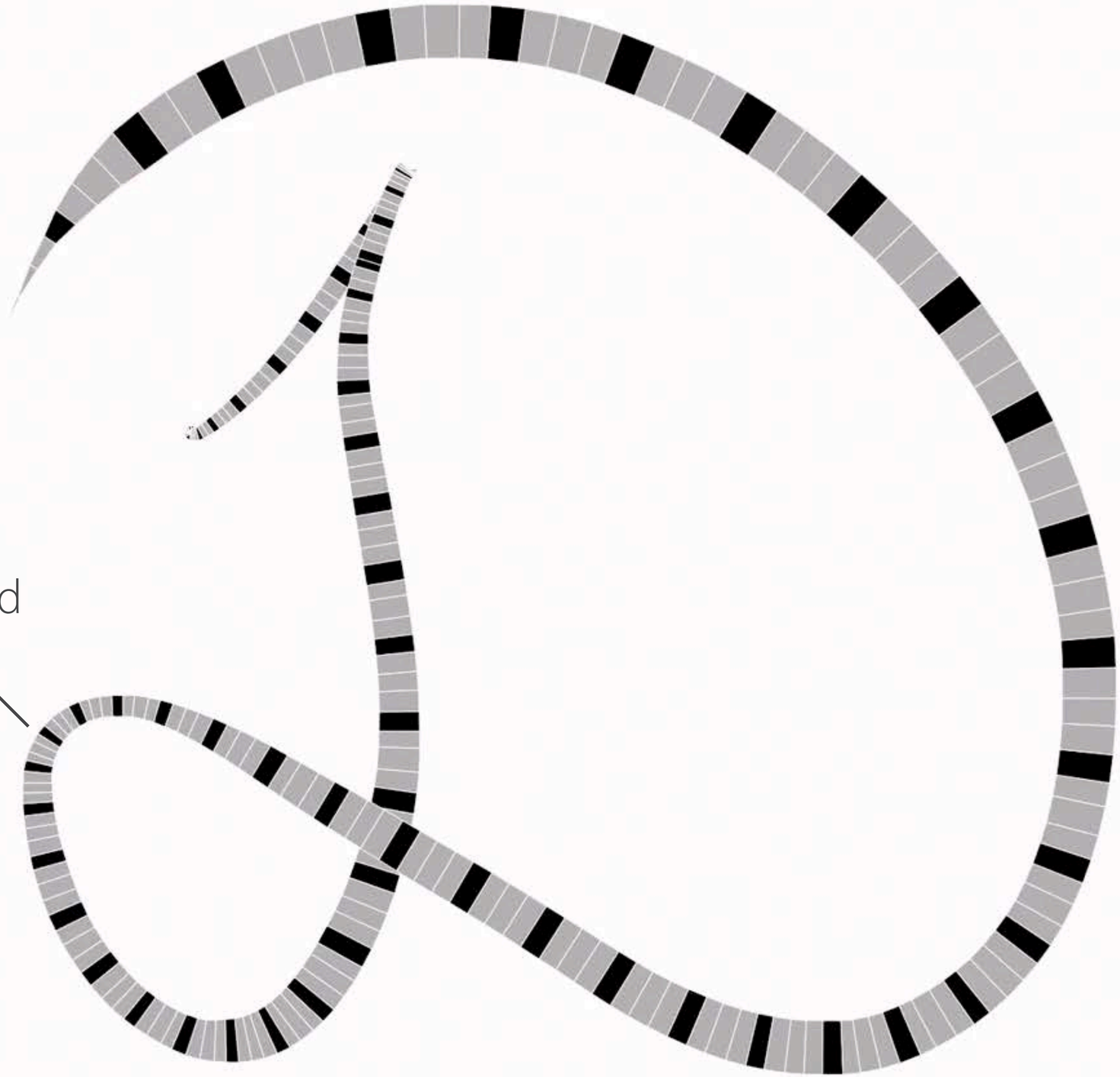
Add to the model

```
struct StrokeSample {
    let location: CGPoint

}
```

Force

Add to the model

```
struct StrokeSample {
    let location: CGPoint
    var force: CGFloat?
}
```

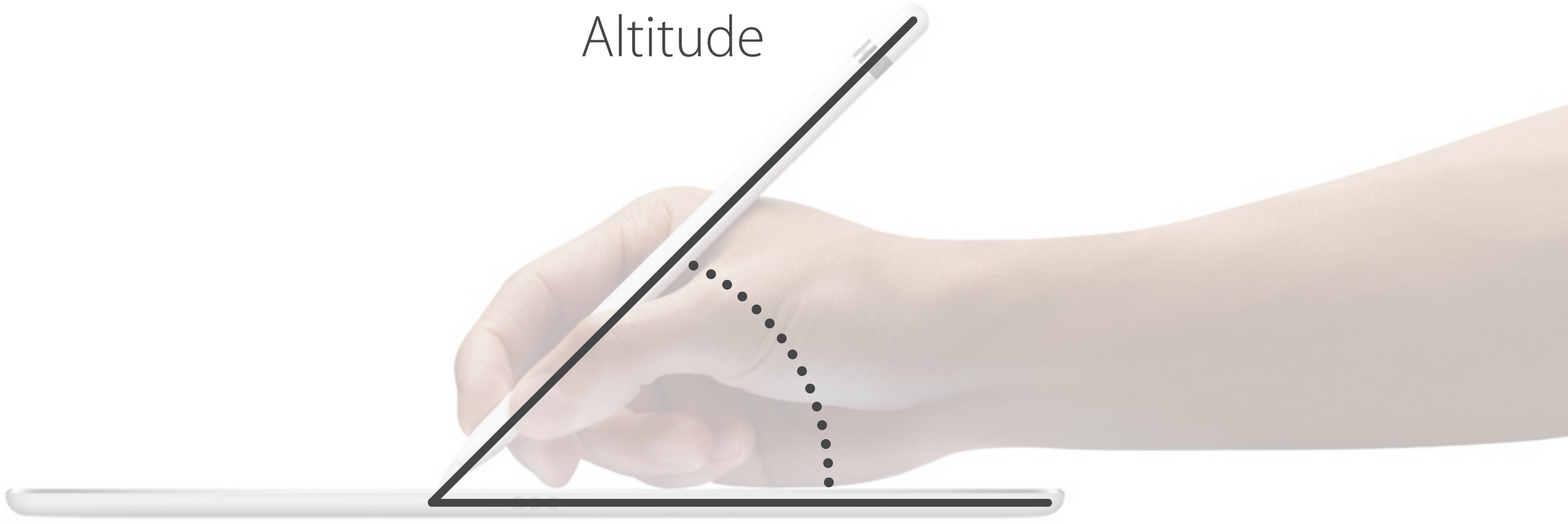

Width based
on force

Apple Pencil

Tilt

Apple Pencil

Tilt



Apple Pencil

Tilt

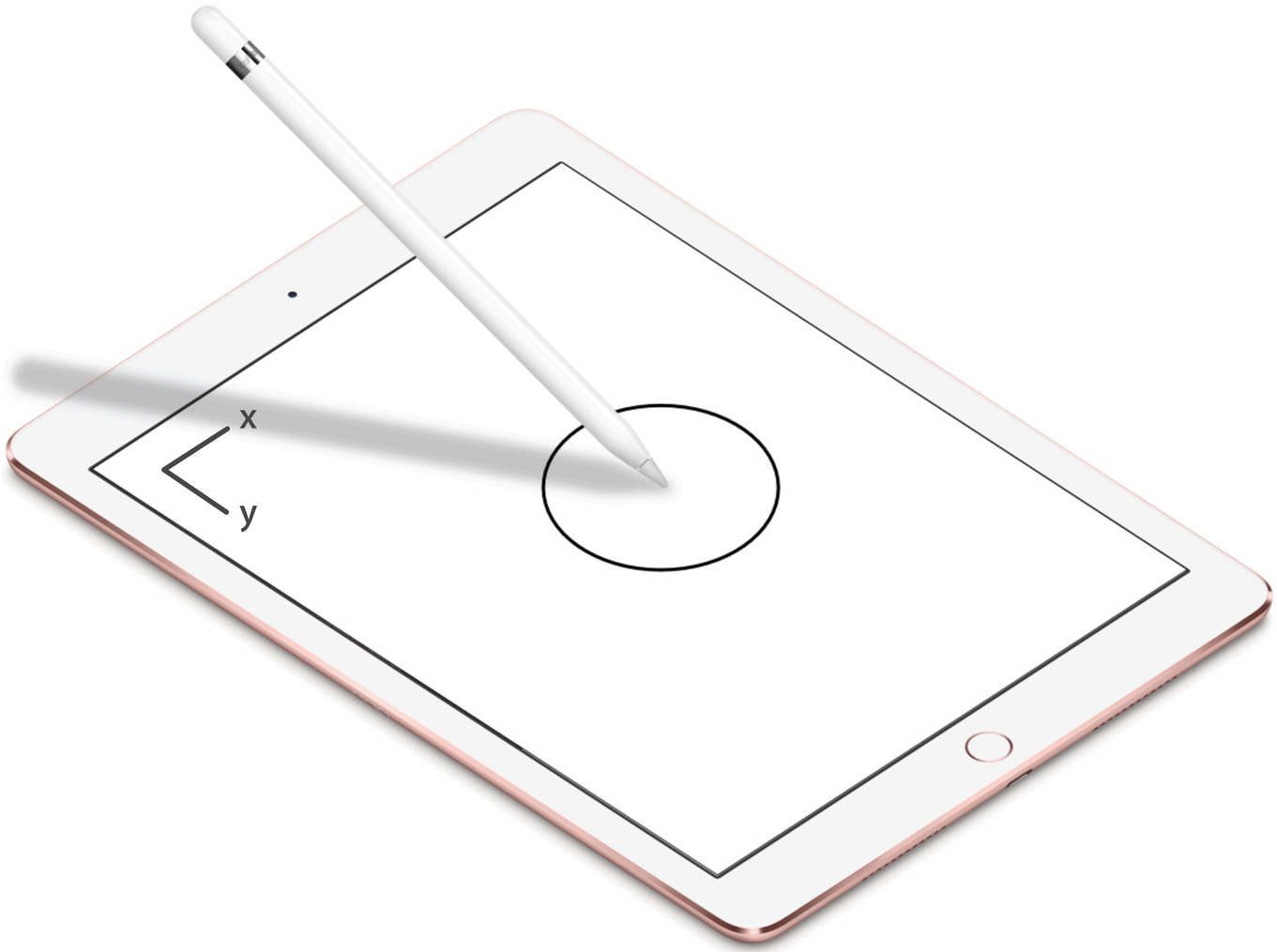
```
var altitudeAngle: CGFloat { get }
```

Apple Pencil

Orientation

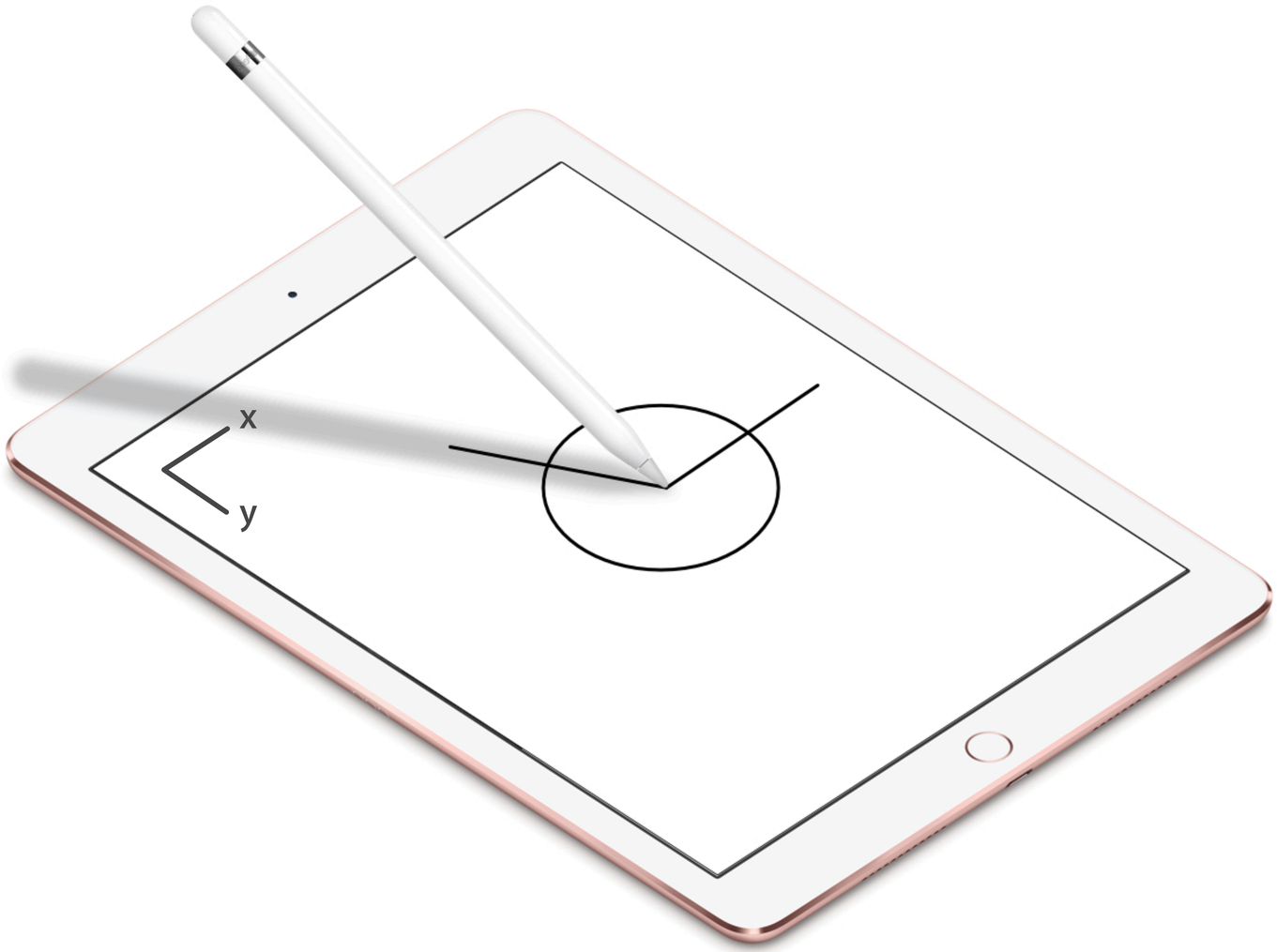
Apple Pencil

Orientation



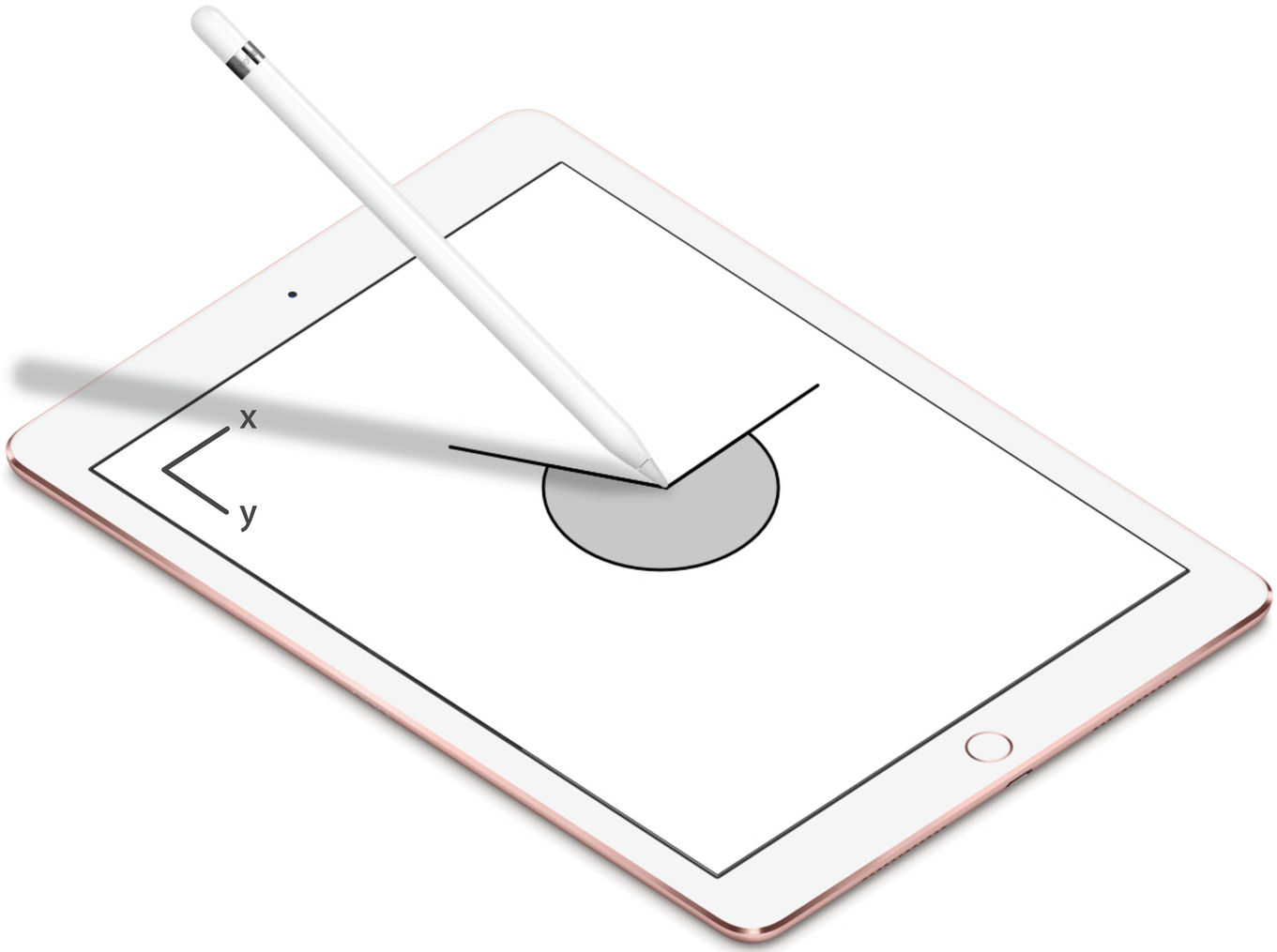
Apple Pencil

Orientation



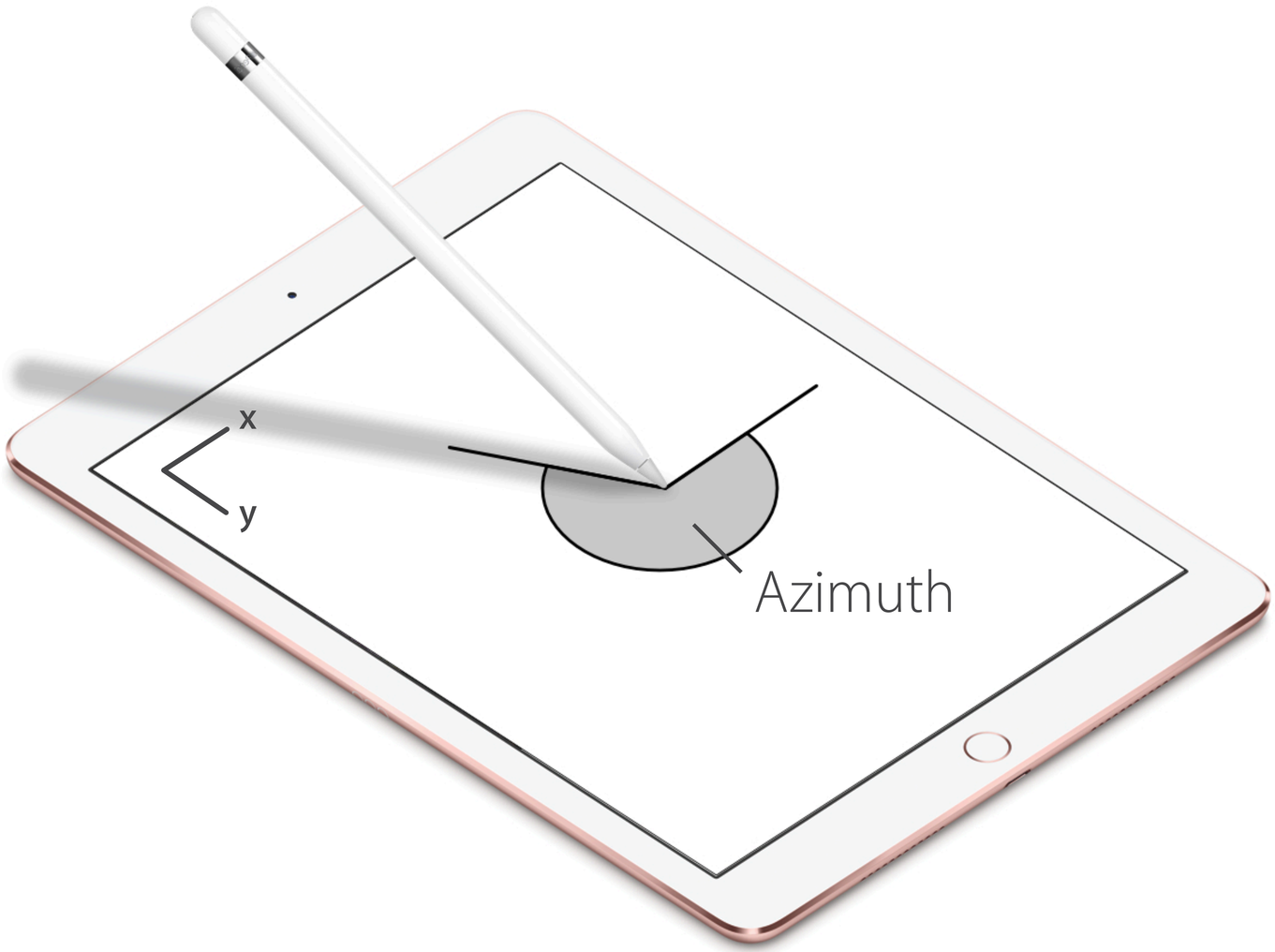
Apple Pencil

Orientation



Apple Pencil

Orientation



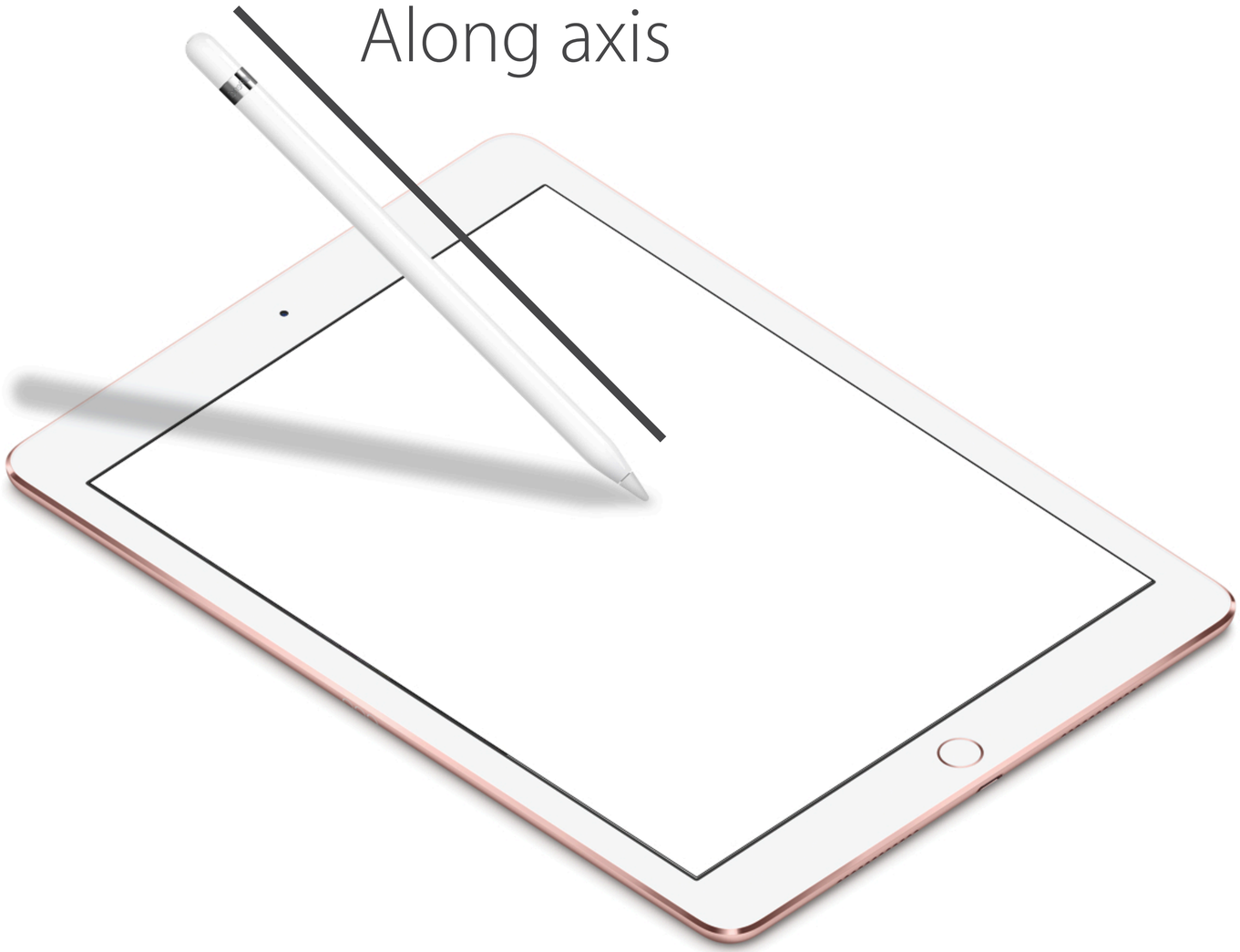
Apple Pencil

Azimuth

```
func azimuthAngle(in view: UIView?) -> CGFloat
func azimuthUnitVector(in view: UIView?) -> CGVector
```

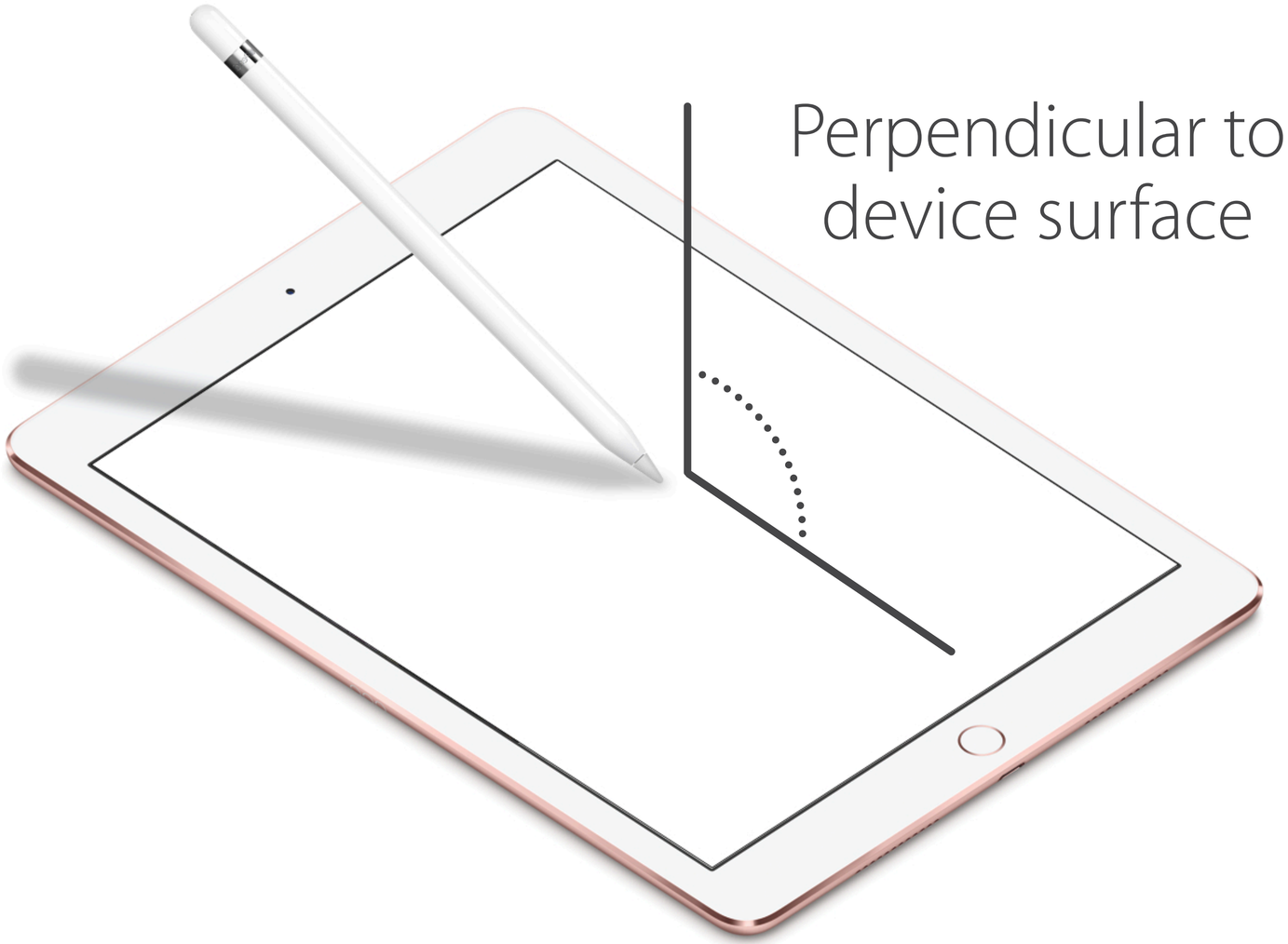
Apple Pencil

Force



Apple Pencil

Force



Apple Pencil

Force

```
var perpendicularForce: CGFloat {
    get {
        return     force / sin(altitudeAngle)
    }
}
```

Apple Pencil

Force

```
var perpendicularForce: CGFloat {
    get {
        return min(force / sin(altitudeAngle), maximumPossibleForce)
    }
}
```

Apple Pencil

Force

Apple Pencil

Estimated properties

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

Apple Pencil

Estimated properties

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

```
struct UITouchProperties : OptionSet {



}
```

Apple Pencil

Estimated properties

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

```
struct UITouchProperties : OptionSet {
    static var force: UITouchProperties { get }

}
```

Apple Pencil

Estimated properties

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

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

}
```

Apple Pencil

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 }
}
```

Apple Pencil

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 estimatedPropertiesExpectingUpdates: UITouchProperties { get }
```

Apple Pencil

Estimated properties with updates

```
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

```
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)

func 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:`

Apple Pencil

Estimated properties with updates

```
override func touchesEstimatedPropertiesUpdated(_ touches: Set<UITouch>) {
    for touch in touches {



    }
}
```

Apple Pencil

Estimated properties with updates

```
override func touchesEstimatedPropertiesUpdated(_ touches: Set<UITouch>) {
    for touch in touches {
        let estimationIndex = touch.estimationUpdateIndex!



    }
}
```

Apple Pencil

Estimated properties with updates

```
override func touchesEstimatedPropertiesUpdated(_ touches: Set<UITouch>) {
    for touch in touches {
        let estimationIndex = touch.estimationUpdateIndex!
        let (sample, sampleIndex) = samplesExpectingUpdates[estimationIndex]



    }
}
```

Apple Pencil

Estimated properties with updates

```
override func touchesEstimatedPropertiesUpdated(_ touches: Set<UITouch>) {
    for touch in touches {
        let estimationIndex = touch.estimationUpdateIndex!
        let (sample, sampleIndex) = samplesExpectingUpdates[estimationIndex]
        let updatedSample = updatedSample(with: touch)



    }
}
```

Apple Pencil

Estimated properties with updates

```
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)



    }
}
```

Apple Pencil

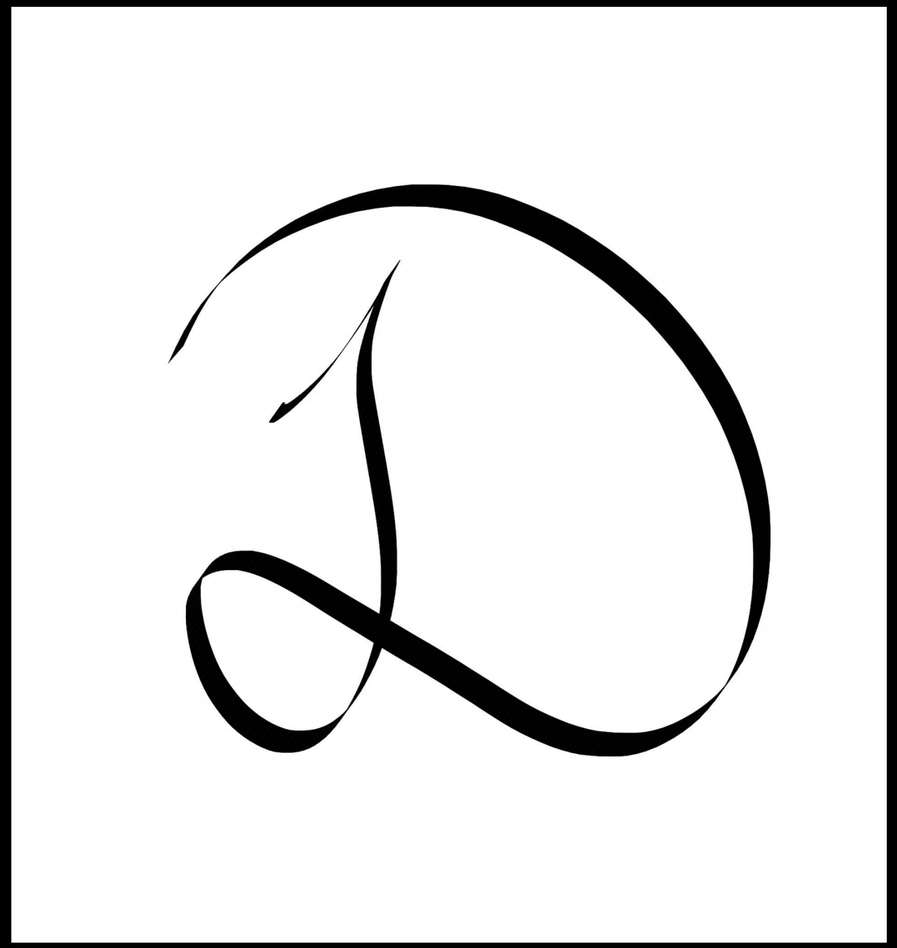
Estimated properties with updates

```
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)
        if touch.estimatedPropertiesExpectingUpdates == [] {
            samplesExpectingUpdates.removeValue(forKey: sampleIndex)
        }
    }
}
```


Azimuth

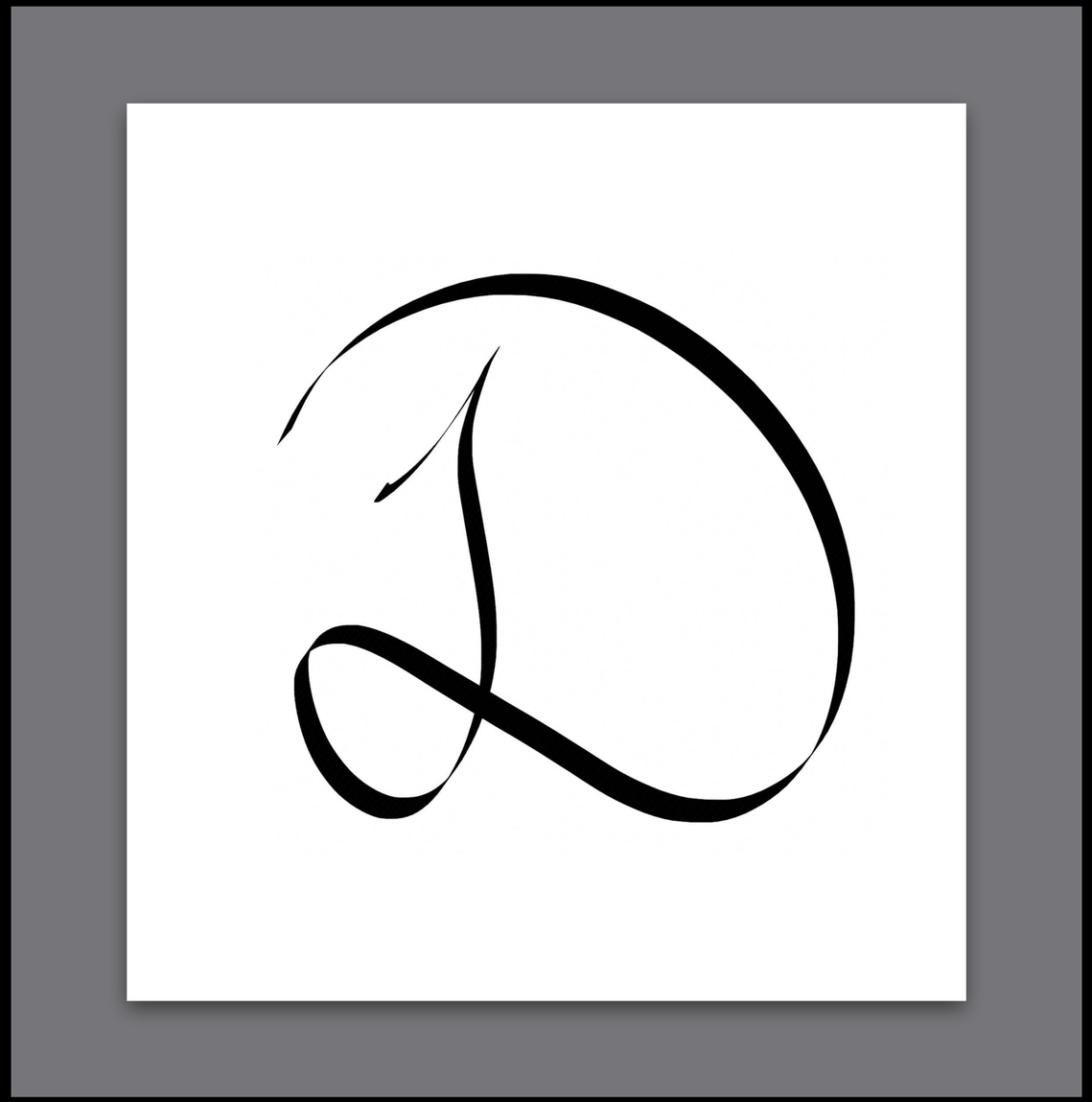
Finishing Touches

Canvas



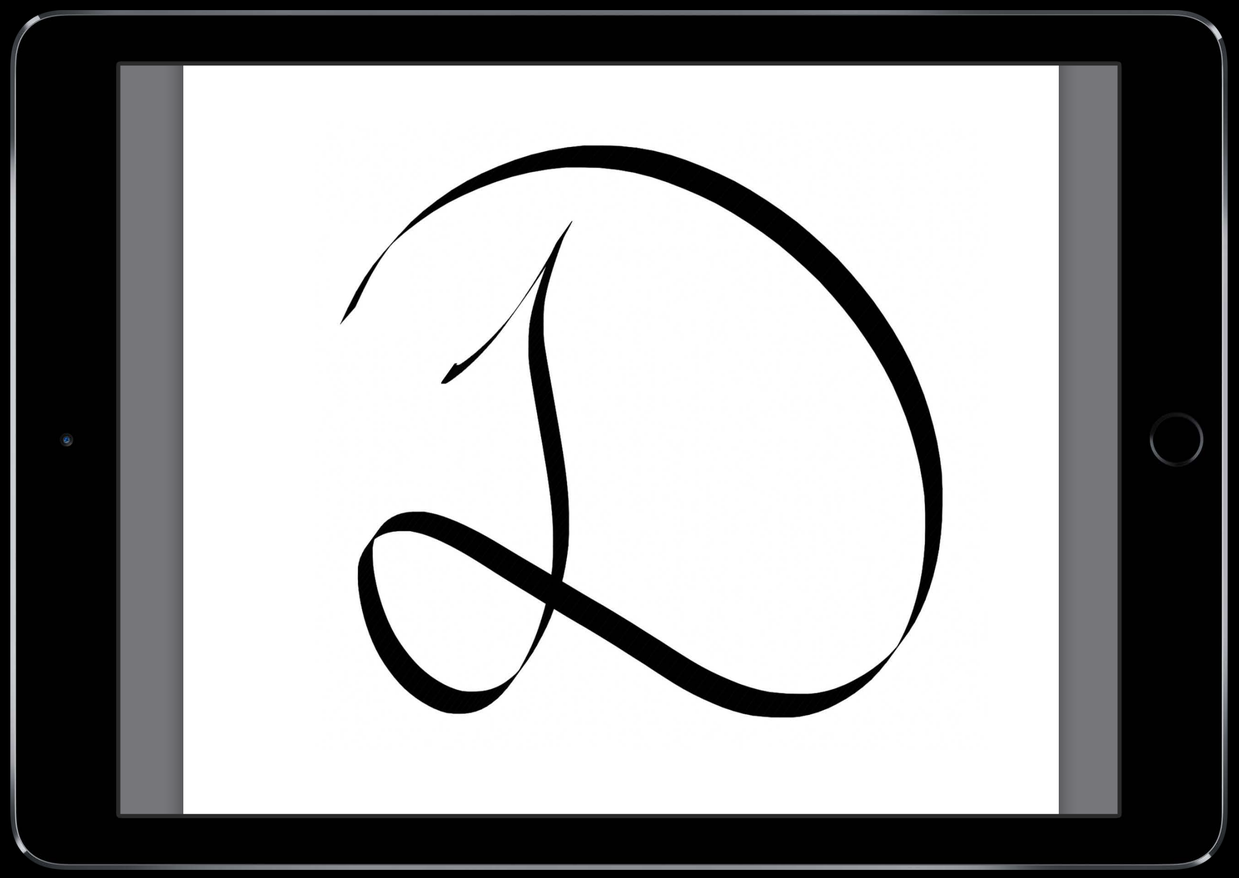
Finishing Touches

Canvas



Finishing Touches

Canvas



Finishing Touches

Adjusting gestures

Scroll view UIPanGestureRecognizer vs. StrokeGestureRecognizer

Disable scrolling with Apple Pencil

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

```
let pan = scrollView.panGestureRecognizer
pan.allowedTouchTypes = [UITouchType.direct.rawValue as NSNumber]
strokeRecognizer.allowedTouchTypes = [UITouchType.stylus.rawValue as NSNumber]
```

Finishing Touches

Adjusting gestures

```
class UIGestureRecognizer {
    public var requiresExclusiveTouchType: Bool
}
```

Summary

New properties of UITouch

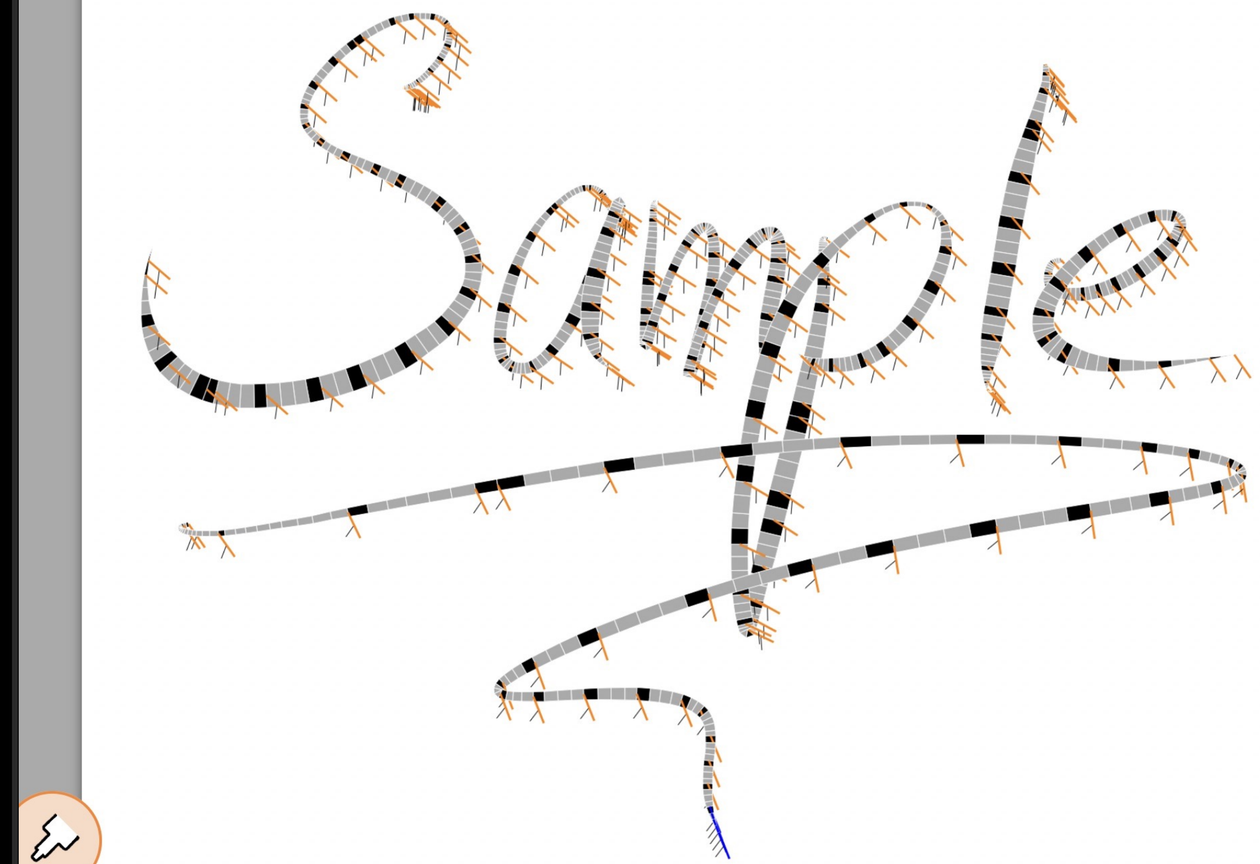
Coalesced and predicted touches

Property estimation

Adjusting gestures


pencil
clear
Sample

pencil clear



More Information

https://developer.apple.com/wwdc16/220

Related Sessions

Controlling Game Input for Apple TV	Mission	Wednesday 5:00PM
A Peek at 3D Touch	Presidio	Thursday 4:00PM
Advanced Touch Input on iOS	WWDC 2015	

Labs

UIKit and UIKit Animations Lab	Frameworks Lab C	Thursday 1:00PM
Cocoa Touch and 3D Touch Lab	Frameworks Lab C	Friday 10:30AM

WWDC16