CSC 471 / 371 Mobile Application Development for iOS

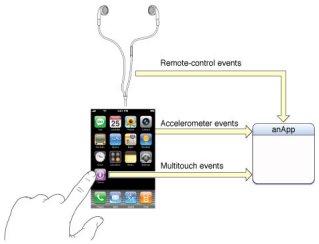
Prof. Xiaoping Jia
School of Computing, CDM
DePaul University
xjia@cdm.depaul.edu
@DePaulSWEng

Multi-Touch Events & Gestures

Outline

- Events
- Multi-touch events
- Gestures
  - Taps
  - Multi-touches
  - Swipe
  - Pinch
- Gesture recognizers

Event Types in iOS



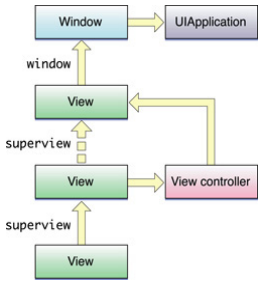
- Event types
  - Touches
  - Motion
  - Remote control

Responder Objects

- Objects that can respond to events and handle them.
  - Also known as, simply, *responders*.
  - `UIResponder` is the base class for all responders
- The *first responder*
  - The first in a chain to respond to touch events, and
  - The responder to receive *untargeted* events
    - All events except touch events
    - Motion, remote-control, etc.
  - Usually a `UIView` object
  - Automatically maintained by the UIKit

Responder Chain

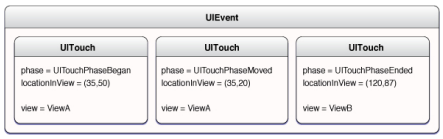
- The **responder chain**
  - a series of responders
  - first responder at the head
- An event proceeds up the responder chain to look for a responder capable of handling the event.
- The responder chain
  - Maintain the *next* responder
  - Default: the superview



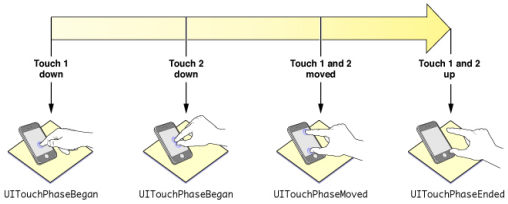
Multi-Touch Events

- `UIEvent`
  - A container for one or more touches
- `UITouch`
  - Represents a single finger
  - Properties

```
var timestamp: NSTimeInterval
var phase: UITouchPhase
var tapCount: Int
```



Multi-Touch Event Phases

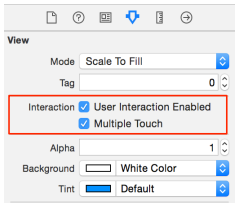


Handling Multi-Touch Events

- Create a subclass of a responder class
  - View controller, custom view, etc.
- Typically the view controller associated with the view.
- Enable user interaction and multi-touch
- Implement one or more *UIResponder* methods to handle the multi-touch events

Enabling Multi-Touch Events

- In *Interface Builder*, check
  - User Interaction Enabled
  - Multiple Touch
- Programmatically set properties of view objects
  - `userInteractionEnabled`
  - `multipleTouchEnabled`



Receiving Touch Events

```
func touchesBegan(touches: Set<UITouch>,
                  withEvent event: UIEvent?)
```

- One or more fingers touched down on the screen.

```
func touchesMoved(touches: Set<UITouch>,
                  withEvent event: UIEvent?)
```

- One or more fingers moved.

```
func touchesEnded(touches: Set<UITouch>,
                  withEvent event: UIEvent?)
```

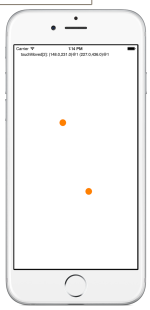
- One or more fingers lifted up from the screen.

```
func touchesCancelled(touches: Set<UITouch>?,
                  withEvent event: UIEvent?)
```

- The touch sequence is cancelled by a system event, such as an incoming phone call.

Multi-Touch Demo App

- Handle multi-touch events
- Display a simple message
  - Touch location
  - Tap count
- Draw solid circles at the locations of the touches
- Handle single and double taps



Multi-Touch Demo App

- Single view app
- A custom view class: *Touch View*
- Change the root view to *Touch View*

```
class TouchView: UIView {
    var points : [CGPoint] = []
    var message : String = "Touch view"

    override func drawRect(rect: CGRect) { … }
    …
    Handle touch events
}
```

The locations of the touches

DePaul University 13

Handle Touch Events

```
override func touchesBegan(touches: Set<UITouch>,
                           withEvent event: UIEvent?) {
    handleTouches("touchBegan", touches: touches)
}
override func touchesMoved(touches: Set<UITouch>,
                           withEvent event: UIEvent?) {
    handleTouches("touchMoved", touches: touches)
}
override func touchesEnded(touches: Set<UITouch>,
                           withEvent event: UIEvent?) {
    handleTouches("touchEnded", touches: touches)
}
override func touchesCancelled(touches: Set<UITouch>?,
                           withEvent event: UIEvent?) {
    handleTouches("touchCancelled", touches: touches)
}
```

Handle Touch Events

```
func handleTouches(method: String,
                   touches: Set<UITouch>?) {
    message = method + "[\(touches?.count ?? 0)]:"
    points.removeAll(keepCapacity: true)
    if let touches = touches {
        for touch in touches {
            let p = touch.locationInView(self)
            message += " (\(p.x),\(p.y))"
            points.append(p)
        }
    }
    setNeedsDisplay()
}
```

DePaul University 15

Draw Touch View

```
override func drawRect(rect: CGRect) {
    (message as NSString).drawAtPoint(CGPointMake(20, 20),
        withAttributes: nil)

    let context = UIGraphicsGetCurrentContext()
    CGContextSetFillColorWithColor(context,
        UIColor.orangeColor().CGColor)
    let r: CGFloat = 10
    for p in points {
        let rect = CGRect(x: p.x - r, y: p.y - r,
            width: 2 * r, height: 2 * r)
        CGContextFillEllipseInRect(context, rect)
    }
}
```

DePaul University 16

Handle Tap Gesture – 1st Attempt

- Number of taps: `tapCount` in a *UITouch* object
- Place to handle tap: `touchesEnded`

```
override func touchesEnded(touches: Set<UITouch>,
                           withEvent event: UIEvent?) {
    if let touch = touches.first {
        if touch.tapCount == 1 {
            Handle single tap
        } else if touch.tapCount == 2 {
            Handle double tap
        }
    }
}
```

DePaul University 17

Handle Single and Double Taps

- A complication
  - When you receive a single tap
  - Is it just a single tap?
  - Or is it the first tap of a double tap
- Handling of single tap must be delayed until you are certain that it is just a single tap.

DePaul University 18

Handle Tap Gesture – 2nd Attempt, Part 1

```
var timer: NSTimer?
override func touchesEnded(touches: Set<UITouch>,
                           withEvent event: UIEvent?) {
    if let touch = touches.first {
        if touch.tapCount == 2 {
            handleDoubleTap()
        } else {
            timer =
              NSTimer.scheduledTimerWithTimeInterval(0.3,
                target: self, selector: "handleSingleTap",
                userInfo: nil, repeats: false)
        }
    }
}
```

Call to the `handleSingleTap` method is delayed with a timer

Handle Tap Gesture – Tap Messages

```
var tapMessage : String = ""

func handleSingleTap() {
    tapMessage = "Single tap!"
    print("Single tap!")
    setNeedsDisplay()
}

func handleDoubleTap() {
    tapMessage = "Double tap!!"
    print("Double tap!!")
    setNeedsDisplay()
}
```


Handle Tap Gesture – 2nd Attempt, Part 2

- Invalidate the timer to cancel the method call

```
override func touchesBegan(touches: Set<UITouch>,
                           withEvent event: UIEvent?) {
    handleTouches("touchBegan", touches: touches)
    tapMessage = ""
    if let touch = touches.first {
        if touch.tapCount >= 2 {
            timer?.invalidate()
        }
    }
}
```


Draw Touch View – The Tap Message

```
override func drawRect(rect: CGRect) {
    (message as NSString).drawAtPoint(
        CGPointMake(20, 20), withAttributes: nil)
    (tapMessage as NSString).drawAtPoint(
        CGPointMake(20, 40), withAttributes: nil)
}
```

Drawing the solid circles at touch locations (slide 16)

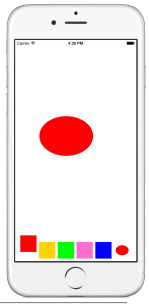
Handling Multi-Touch Events Best Practices

- Implement all of the event-handling methods
  - Even if it is a null implementation.
  - Do not call the superclass implementation of the methods.
- Always implement the event-cancellation methods.
  - Restore the state of the view

A Drawing App

A Simple Drawing App

- A touch drawing app
- Draw various shapes
  - Line, Ellipse, Filled Ellipse, Rectangle, Filled Rectangle, Scribble
  - Select shapes with the *shape button*
- Using different colors
  - Select color with the *color buttons*



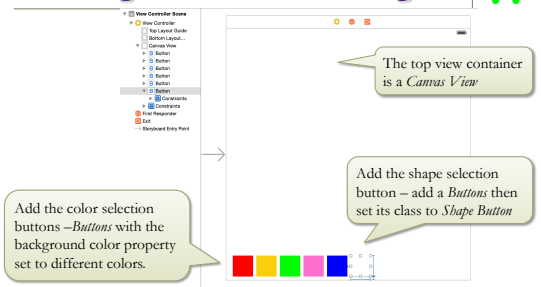
DePaul University 25

Create the Drawing App

- Start with a *Single View app*
- Add two new custom view class
  - *Canvas View* – for drawing
    - A subclass of *UIView*
    - Set the top view container class to *Canvas View*
  - *Shape Button* – for selecting shapes
    - A subclass of *UIButton*
    - A button that displays the currently selected shape

DePaul University 26

Drawing Pad – The UI Design



DePaul University 27

Drawing Pad – Connect the View and the View Controller

- An outlet collection of all the color buttons
- Outlets of the canvas and the shape button
- An action to select colors, connected to all color buttons
- An action to select shapes, connected to the shape button

```
class ViewController: UIViewController {
    @IBOutlet var colorButtons: [UIButton]!
    @IBOutlet weak var canvas: CanvasView!
    @IBOutlet weak var shapeButton: ShapeButton!
    @IBAction func selectColor(sender: UIButton) { … }
    @IBAction func selectShape(sender: ShapeButton) { … }
    …
}
```

DePaul University 28

Drawing Pad – The Shape Type

- An enum type
- A constant array of all available shapes

```
enum ShapeType: String {
    case Line = "Line"
    case Ellipse = "Ellipse"
    case Rectangle = "Rectangle"
    case FilledEllipse = "Filled Ellipse"
    case FilledRectangle = "Filled Rectangle"
    case Scribble = "Scribble"
}

let shapes: [ShapeType] = [ .Line, .Ellipse, .Rectangle,
        .FilledEllipse, .FilledRectangle, .Scribble ]
```

DePaul University 29

The Shape Button

- A custom view class, subclass of `UIButton`
  - Visually show the selected shape with the selected color

Override the `drawRect` method

```
class ShapeButton: UIButton {
    var shape: ShapeType = .Line
    var color: UIColor = UIColor.blueColor()

    override func drawRect(rect: CGRect) { … }
}
```

DePaul University 30

Shape Button – Drawing Shapes

- Override the `drawRect` method
  - Set up the the graphics context with the selected color

```
override func drawRect(rect: CGRect) {
    let context = UIGraphicsGetCurrentContext()
    CGContextSetStrokeColorWithColor(context,
        color.CGColor)
    CGContextSetFillColorWithColor(context,
        color.CGColor)
    CGContextSetLineWidth(context, 2)
    Draw shapes (next 3 slides)
}
```


Shape Button – Drawing Line Shape

```
override func drawRect(rect: CGRect) {
    …
    let x1: CGFloat = 5
    let y1: CGFloat = 5
    let x2: CGFloat = frame.width - 5
    let y2: CGFloat = frame.height - 5
    let rect = CGRect(x: x1, y: y1 + 5,
        width: frame.width - 10, height: frame.height - 20)
    switch shape {
    case .Line:
            CGContextMoveToPoint(context, x1, y1)
            CGContextAddLineToPoint(context, x2, y2)
            CGContextStrokePath(context)
    …
```

Shape Button – Drawing Ellipse and Rectangle

```
override func drawRect(rect: CGRect) {
    …
    switch shape {
    case .Line: …
    case .Ellipse:
            CGContextStrokeEllipseInRect(context, rect)
    case .Rectangle:
            CGContextStrokeRect(context, rect)
    case .FilledEllipse:
            CGContextFillEllipseInRect(context, rect)
    case .FilledRectangle:
            CGContextFillRect(context, rect)
    …
```

Shape Button – Drawing the Scribble Shape

```
override func drawRect(rect: CGRect) {
    …
    switch shape {
    …
    case .Scribble:
        CGContextMoveToPoint(context, x1, y1)
        CGContextAddCurveToPoint(context,
            x1 + 80, y1 - 10,   // the 1st control point
            x2 - 80, y2 + 10,   // the 2nd control point
            x2, y2)             // the end point
        CGContextStrokePath(context)
    }
}
```

A cubic Bézier curve.

Drawing Pad – Incremental Implementation

- Iteration 1: drawing line only, with the default color
  - Tracking the first and last touch points
- Iteration 2: drawing ellipses and rectangles
  - Handling selection of shapes
- Iteration 3: drawing scribble
  - Tracking all touch points
- Iteration 4: using different colors
  - Handling color selection
  - Property observers

Drawing Pad – Iteration 1 The Canvas View

```
class CanvasView: UIView {
    var shape: ShapeType = .Line
    var color: UIColor = UIColor.blueColor()
    var first :CGPoint = CGPointZero
    var last  :CGPoint = CGPointZero
    override func drawRect(rect: CGRect) { … }
    override func touchesBegan(touches: Set<UITouch>,
                  withEvent event: UIEvent?) { … }
    override func touchesMoved(touches: Set<UITouch>,
                  withEvent event: UIEvent?) { … }
    override func touchesEnded(touches: Set<UITouch>,
                  withEvent event: UIEvent?) { … }
    override func touchesCancelled(touches: Set<UITouch>?,
                  withEvent event: UIEvent?) {}
}
```

The first and last touch locations

Drawing Pad – The Canvas View Handle Touches

```
override func touchesBegan(touches: Set<UITouch>,
              withEvent event: UIEvent?) {
    if let touch = touches.first {
        first = touch.locationInView(self)
        last = first
        setNeedsDisplay()
    }
}
override func touchesMoved(touches: Set<UITouch>,
              withEvent event: UIEvent?) {
    if let touch = touches.first {
        last = touch.locationInView(self)
        setNeedsDisplay()
    }
}
```

Keep track of the first and last touch locations.

DePaul University 37

Drawing Pad – The Canvas View Handle Touches

```
override func touchesEnded(touches: Set<UITouch>,
              withEvent event: UIEvent?) {
    if let touch = touches.first {
        last = touch.locationInView(self)
        setNeedsDisplay()
    }
}

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

DePaul University 38

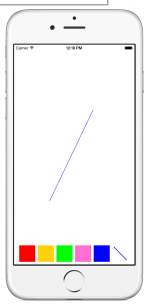
Drawing Pad – The Canvas View Draw Lines

```
override func drawRect(rect: CGRect) {
    let context = UIGraphicsGetCurrentContext()
    CGContextSetStrokeColorWithColor(context, color.CGColor)
    CGContextSetFillColorWithColor(context, color.CGColor)
    switch shape {
    case .Line:
        CGContextMoveToPoint(context, first.x, first.y)
        CGContextAddLineToPoint(context, last.x, last.y)
        CGContextStrokePath(context)
    case …
    }
}
```

DePaul University 39

Drawing Pad – First Test Run

- Handle touch events for drawing
  - a line segment, the default shape
  - using the default color
- Next: iteration 2
  - Handle drawing of ellipse and rectangle shapes in the *Canvas View*
  - Handle selection of shapes in the *View Controller*



DePaul University 40

Drawing Pad – The Canvas View Draw Ellipses & Rectangles

```
override func drawRect(rect: CGRect) {
    …
    let rect = CGRect(x: first.x, y: first.y,
        width: last.x - first.x, height: last.y - first.y)
    switch shape { …
    case .Ellipse:
            CGContextStrokeEllipseInRect(context, rect)
    case .Rectangle:
            CGContextStrokeRect(context, rect)
    case .FilledEllipse:
            CGContextFillEllipseInRect(context, rect)
    case .FilledRectangle:
            CGContextFillRect(context, rect)
    case .Scribble: …
    }
}
```

Will handle scribbles later.

Drawing Pad – View Controller

- Let's deal with the selection of shapes first
  - Select shape action displays an *Action Sheet* popup

```
class ViewController: UIViewController {

    @IBOutlet var colorButtons: [UIButton]!
    @IBOutlet weak var canvas: CanvasView!
    @IBOutlet weak var shapeButton: ShapeButton!

    @IBAction func selectColor(sender: UIButton) { … }
    @IBAction func selectShape(sender: ShapeButton) { … }
    …
}
```

DePaul University 42

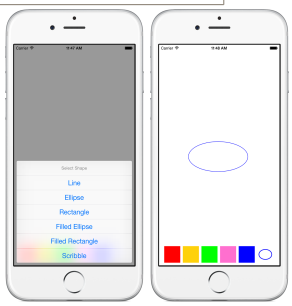
Drawing Pad – View Controller

```
@IBAction func selectShape(sender: ShapeButton) {
    let title = "Select Shape"
    let alertController = UIAlertController(title: title,
        message: nil, preferredStyle: .ActionSheet)
    for shape in shapes {
        let action = UIAlertAction(title: shape.rawValue,
            style: .Default) { action in
            sender.shape = shape
            sender.setNeedsDisplay()
            self.canvas.shape = shape
        }
        alertController.addAction(action)
    }
    presentViewController(alertController, animated: true,
        completion: nil)
}
```

Action for selecting a shape

Drawing Pad – Second Test Run

- Tap the *Shape Button*
- Select an ellipse or rectangle shape from the *Action Sheet*
  - Notice the change of shapes displayed in the *Shape Button*
- Draw ellipse or rectangle shapes



Drawing Pad – Iteration 3

- Draw scribble
- Need to keep track the locations of all touch events
  - Use an array of `CGPoint`
- **Do not save/store touch events**
  - Event objects are recycled
  - `CGPoint` is a struct, i.e., a value type
    - The values are copied, safe to store

Drawing Pad – Handle Scribble

```
class CanvasView: UIView {
    var shape: ShapeType = .Line
    var color: UIColor = UIColor.blueColor()
    var first :CGPoint = CGPointZero
    var last  :CGPoint = CGPointZero
    var points: [CGPoint] = []
    override func drawRect(rect: CGRect) { … }
    override func touchesBegan(touches: Set<UITouch>,
                  withEvent event: UIEvent?) { … }
    override func touchesMoved(touches: Set<UITouch>,
                  withEvent event: UIEvent?) { … }
    override func touchesEnded(touches: Set<UIToucht>,
                  withEvent event: UIEvent?) { … }
    override func touchesCancelled(touches: Set<UITouch>?,
                  withEvent event: UIEvent?) {}
}
```

Drawing Pad – The Canvas View Handle Scribble Touches

```
override func touchesBegan(touches: Set<UITouch>,
                withEvent event: UIEvent?) {
    if let touch = touches.first {
        first = touch.locationInView(self)
        last = first
        points.removeAll(keepCapacity: true)
        if shape == .Scribble {
            points.append(first)
        }
        setNeedsDisplay()
    }
}
```


Drawing Pad – The Canvas View Handle Scribble Touches

```
override func touchesMoved(touches: Set<UITouch>,
                withEvent event: UIEvent?) {
    if let touch = touches.first {
        last = touch.locationInView(self)
        if shape == .Scribble {
            points.append(last)
        }
        setNeedsDisplay()
    }
}
```

Drawing Pad – The Canvas View Handle Scribble Touches

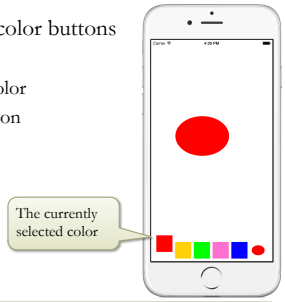
```
override func touchesEnded(touches: Set<UITouch>,
                withEvent event: UIEvent?) {
    if let touch = touches.first {
        last = touch.locationInView(self)
        if shape == .Scribble {
            points.append(last)
        }
        setNeedsDisplay()
    }
}

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


Drawing Pad – The Canvas View Draw Scribble

```
override func drawRect(rect: CGRect) {
    …
    switch shape {
    case …
    case .Scribble:
        CGContextMoveToPoint(context, first.x, first.y)
        for p in points {
            CGContextAddLineToPoint(context, p.x, p.y)
        }
        CGContextStrokePath(context)
    }
}
```


Drawing Pad – Iteration 4

- Select color using the color buttons
  - Select color
  - Indicate the selected color
  - Animate the color button



Drawing Pad – View Controller

- The `selectColor` action is connected to all the color buttons

```
@IBAction func selectColor(sender: UIButton) {
    canvas.color = sender.backgroundColor!
    shapeButton.color = sender.backgroundColor!
    shapeButton.setNeedsDisplay()
}
```

- This will select the color, but we also want to indicate which color is currently selected
  - Raise the button corresponding to the selected color

Drawing Pad – View Controller

```
@IBAction func selectColor(sender: UIButton) {
    UIView.animateWithDuration(0.5, delay: 0.0,
        usingSpringWithDamping: CGFloat(0.25),
        initialSpringVelocity: CGFloat(0.25),
        options: UIViewAnimationOptions.CurveEaseInOut,
        animations: {
        for button in self.colorButtons {
            button.frame.origin.y = self.view.bounds.height - 58
        }
        sender.frame.origin.y -= 20
    }, completion: nil)
    canvas.color = sender.backgroundColor!
    shapeButton.color = sender.backgroundColor!
    shapeButton.setNeedsDisplay()
}
```

Animate the selection

Raise the selected color button.

Another Refinement

- When a new color or shape is selected, the shape button `color` and `shape` properties are updated
- The property updates must be immediately followed by

```
shapeButton.setNeedsDisplay()
```

  - To update the display on the Shape Button, with the newly selected color or shape.
  - Without this, the display would be inconsistent with the selection

Property Observers

- Observe and respond to changes in a *stored* property's value.
  - Called every time a property's value is set
  - Even if the new value is the same as the property's current value.
- You may define either or both observers:
  - `willSet` – called just before the value is set
    - Parameter: the new value. Default name: `newValue`
  - `didSet` – called immediately after the new value is set
    - Parameter: the oldValue. Default: `oldValue`

DePaul University 55

Drawing Pad – Shape Button v.2

```
class ShapeButton: UIButton {
    var shape: ShapeType = .Line {
        didSet {
            setNeedsDisplay()
        }
    }
    var color: UIColor = UIColor.blueColor() {
        didSet {
            setNeedsDisplay()
        }
    }

    override func drawRect(rect: CGRect) { … }
}
```

DePaul University 56

Drawing Pad – Select Shape, v.2

```
@IBAction func selectShape(sender: ShapeButton) {
    let title = "Select Shape"
    let alertController = UIAlertController(title: title,
        message: nil, preferredStyle: .ActionSheet)
    for shape in shapes {
        let action = UIAlertAction(title: shape.rawValue,
            style: .Default) { action in
            sender.shape = shape

            self.canvas.shape = shape
        }
        alertController.addAction(action)
    }
    presentViewController(alertController, animated: true,
        completion: nil)
}
```

DePaul University 57

Drawing Pad – Set Color, v.2

```
@IBAction func selectColor(sender: UIButton) {
    canvas.color = sender.backgroundColor!
    shapeButton.color = sender.backgroundColor!

}
```

DePaul University 58

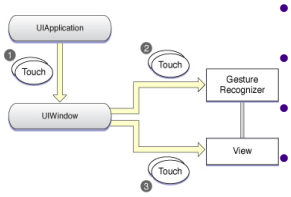
Gesture Recognizers

Gesture Recognizers

- UIKit provides gesture recognizers for common gestures.
  - Tapping (any number of taps)
    - `UITapGestureRecognizer`
  - Pinching in and out (for zooming a view)
    - `UIPinchGestureRecognizer`
  - Panning or dragging
    - `UIPanGestureRecognizer`
  - Swiping (in a given direction)
    - `UISwipeGestureRecognizer`
  - Rotating (fingers moving in opposite directions)
    - `UIRotationGestureRecognizer`
  - Long press (also known as "touch and hold")
    - `UILongPressGestureRecognizer`

DePaul University 60

Gesture Recognizer

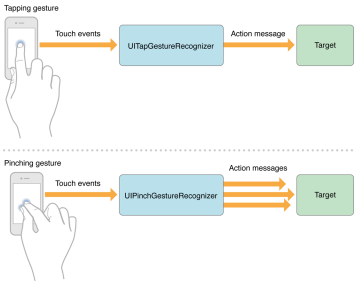


- Observers of touch objects
- Not part of that view hierarchy
- Do not participate in the responder chain.
- May delay the delivery of touch objects to the view
- Cancel delivery of remaining touch objects to the view once they recognize their gesture.

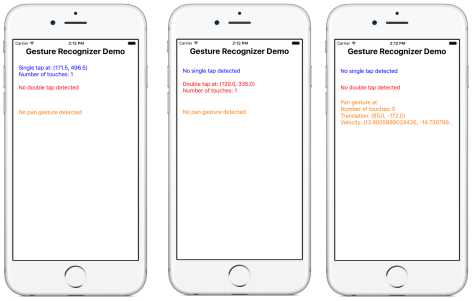
Discrete and Continuous Gestures

- A discrete gesture happens just once.
  - such as a double-tap
  - The gesture recognizer sends its target a single action message.
- A continuous gesture takes place over a period
  - such as pinching
  - Ends when the user lifts the final finger in the multi-touch sequence.
  - The gesture recognizer sends action messages to its target at short intervals until the multi-touch sequence ends.

Discrete and Continuous Gestures



Gesture Recognizer Demo – Single & Double Taps, Panning



Detect Single Tap

```
override func viewDidLoad() {
    super.viewDidLoad()
    for t in 1...3 {
        let singleTapRecognizer =
           UITapGestureRecognizer(target: self,
                action: "handleSingleTap:")
        singleTapRecognizer.numberOfTapsRequired = 1
        singleTapRecognizer.numberOfTouchesRequired = t
        view.addGestureRecognizer(singleTapRecognizer)
        …
    }
}
```


Handle Single Tap

```
func handleSingleTap(sender: UITapGestureRecognizer) {
    let n = sender.numberOfTouches()
    var message = ""
    for i in 0 ..< n {
        message +=
          " \(sender.locationOfTouch(i, inView: view))"
    }
    singleTapLabel.text = "Single tap at:" + message +
        "\nNumber of touches: \(n)"
    dispatch_after(dispatch_time(DISPATCH_TIME_NOW,
                                 Int64(3 * NSEC_PER_SEC)),
        dispatch_get_main_queue() {
            self.singleTapLabel.text =  "No single tap detected"
        }
}
```

Detect Double Tap

```
override func viewDidLoad() {
    super.viewDidLoad()
    for t in 1...3 {
        …
        let doubleTapRecognizer =
            UITapGestureRecognizer(target: self,
                action: "handleDoubleTap:")
        doubleTapRecognizer.numberOfTapsRequired = 2
        doubleTapRecognizer.numberOfTouchesRequired = t
        view.addGestureRecognizer(doubleTapRecognizer)

        singleTapRecognizer.requireGestureRecognizerToFail(
            doubleTapRecognizer)
    }
}
```


Handle Double Tap

```
func handleDoubleTap(sender: UITapGestureRecognizer) {
    let n = sender.numberOfTouches()
    var message = ""
    for i in 0 ..< n {
        message +=
          " \(sender.locationOfTouch(i, inView: view))"
    }
    singleTapLabel.text = "Double tap at:" + message +
        "\nNumber of touches: \(n)"
    dispatch_after(dispatch_time(DISPATCH_TIME_NOW,
                                 Int64(3 * NSEC_PER_SEC)),
        dispatch_get_main_queue() {
            self.singleTapLabel.text =  "No double tap detected"
        }
}
```


Detect Pan Gesture

```
override func viewDidLoad() {
    super.viewDidLoad()
    …
    let panRecognizer = UIPanGestureRecognizer(target: self,
                            action: "handlePanGesture:")
    panRecognizer.minimumNumberOfTouches = 1
    panRecognizer.maximumNumberOfTouches = 3
    view.addGestureRecognizer(panRecognizer)
}

func handlePanGesture(sender: UIPanGestureRecognizer) {
    …
}
```


Swipe & Pinch Gesture Recognizer Demo

Detect Swipe Gestures – Up Swipe

```
override func viewDidLoad() {
    super.viewDidLoad()
    …
    let upSwipeRecognizer = UISwipeGestureRecognizer(target: self,
                                action: "handleUpSwipe:")
    upSwipeRecognizer.numberOfTouchesRequired = 1
    upSwipeRecognizer.direction = .Up
    view.addGestureRecognizer(upSwipeRecognizer)
    …
}
func handleUpSwipe(sender: UISwipeGestureRecognizer) {
    let view1 = big_ben.superview != nil ? big_ben : eiffel
    let view2 = big_ben.superview != nil ? eiffel : big_ben
    UIView.transitionFromView(view1, toView: view2,
        duration: 2.0, options: .TransitionCurlUp,
        completion: nil)
}
```


Detect Swipe Gestures – Down Swipe

```
override func viewDidLoad() {
    super.viewDidLoad()
    …
    let downSwipeRecognizer = UISwipeGestureRecognizer(target: self,
                                action: "handleDownSwipe:")
    downSwipeRecognizer.numberOfTouchesRequired = 1
    downSwipeRecognizer.direction = .Down
    view.addGestureRecognizer(downSwipeRecognizer)
    …
}
func handleDownSwipe(sender: UISwipeGestureRecognizer) {
    let view1 = big_ben.superview != nil ? big_ben : eiffel
    let view2 = big_ben.superview != nil ? eiffel : big_ben
    UIView.transitionFromView(view1, toView: view2,
        duration: 2.0, options: .TransitionCurlDown,
        completion: nil)
}
```

Detect Swipe Gestures – Left & Right Swipes

```
override func viewDidLoad() {
    super.viewDidLoad()
    …
    let horizontalSwipeRecognizer =
        UISwipeGestureRecognizer(target: self,
                                 action: "handleHorizontalSwipe:")
    horizontalSwipeRecognizer.numberOfTouchesRequired = 1
    horizontalSwipeRecognizer.direction = [ .Left, .Right ]
    view.addGestureRecognizer(horizontalSwipeRecognizer)
    …
}
func handleDownSwipe(sender: UISwipeGestureRecognizer) {
    let view1 = big_ben.superview != nil ? big_ben : eiffel
    let view2 = big_ben.superview != nil ? eiffel : big_ben
    UIView.transitionFromView(view1, toView: view2,
        duration: 2.0, options: .TransitionCrossDissolve,
        completion: nil)
}
```

Detect Pinch Gesture

```
override func viewDidLoad() {
    super.viewDidLoad()
    …
    let pinchGestureRecognizer =
        UIPinchGestureRecognizer(target: self,
                                 action: "handlePinch:")
    view.addGestureRecognizer(pinchGestureRecognizer)
}

func handlePinch(sender: UIPinchGestureRecognizer) {
    let s = sender.scale
    container.transform =
        CGAffineTransform(a: s, b: 0, c: 0, d: s, tx: 0, ty: 0)
}
```

DePaul University 74

Sample Code

- Multi-Touch Demo.zip
- Drawing Pad.zip
- Gesture Recognizer.zip
- Swipe Gestures.zip

DePaul University 75

Next …

- Motion events
- Accelerometers
- Gyroscopes

✧ iOS is a trademark of Apple Inc.

DePaul University 76