CSC 471 / 371 Mobile Application Development for iOS

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

1

Multi-Touch Events & Gestures

2

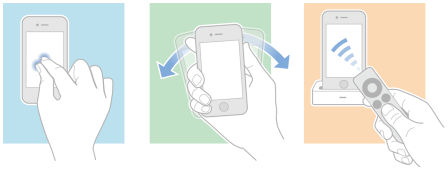
Outline

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

3

Event Types in iOS

- Event types
  - Multi-touch events, e.g., swipe, pinch
  - Motion events, e.g., accelerometer, gyroscope
  - Remote control events, e.g., play/pause, volume up/down



4

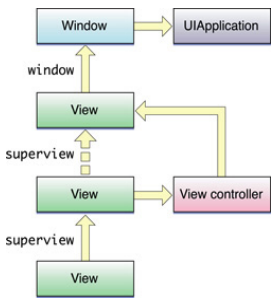
Responder Objects

- 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

5

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

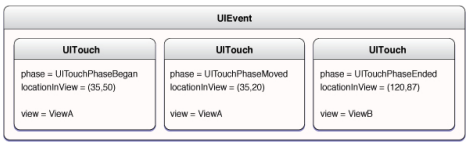


6

Multi-Touch Events

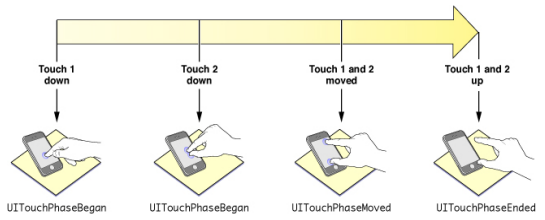
- `UIEvent`
  - A container for one or more touches
- `UITouch`
  - Represents a single finger
  - Properties

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



7

Multi-Touch Event Phases



8

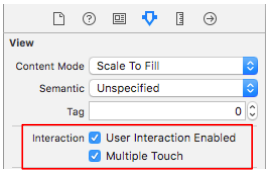
Handling Multi-Touch Events

- 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

9

Enabling Multi-Touch Events

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



10

Receiving Touch Events

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

- One or more fingers touched down on the screen.

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

- One or more fingers moved.

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

- One or more fingers lifted up from the screen.

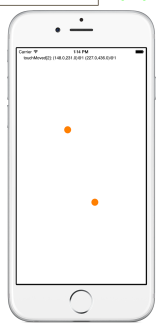
```
func touchesCancelled(_ touches: Set<UITouch>,
                  with event: UIEvent?)
```

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

11

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



12

Multi-Touch Demo App

- Start with an 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 draw(_ rect: CGRect) { … }
    …
    Handle touch events
}
```

The locations of the touches

DePaul University 13

13

Handle Touch Events

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

15

Handle Touch Events

```
func handleTouches(_ method: String,
                   touches: Set<UITouch>) {
    message = method + "[\(touches?.count)]:"
    points.removeAll(keepCapacity: true)
    for touch in touches {
        let p = touch.location(in: self)
        message += String(format: " (%.2f, %.2f)",
                          p.x, p.y)
        points.append(p)
    }
    setNeedsDisplay()
}
```

DePaul University 15

16

Draw Touch View

```
override func draw(_ rect: CGRect) {
    message.draw(at: CGPoint(x: 20, y: 20),
                 withAttributes: nil)

    if let context = UIGraphicsGetCurrentContext() {
        context.setFillColor(UIColor.orange.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)
            context.fillEllipse(in: rect)
        }
    }
}
```

DePaul University 16

17

Handle Tap Gesture – 1st Attempt

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

```
override func touchesEnded(_ touches: Set<UITouch>,
                           with 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

18

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

19

Handle Tap Gesture – 2nd Attempt, Part 1

```
var timer: Timer?
override func touchesEnded(_ touches: Set<UITouch>,
                           with event: UIEvent?) {
    if let touch = touches.first {
        if touch.tapCount == 2 {
            handleDoubleTap()
        } else {
            timer = Timer.scheduledTimer(
                    withTimeInterval: 0.3, repeats: false) {
                _ in self.handleSingleTap()
            }
        }
    }
}
```

Call to the handleSingleTap method is delayed with a timer

20

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()
}
```


21

Handle Tap Gesture – 2nd Attempt, Part 2

- Invalidate the timer to cancel the method call

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


22

Draw Touch View – The Tap Message

```
override func draw(_ rect: CGRect) {
    message.draw(at: CGPoint(x: 20, y: 20),
                 withAttributes: nil)
    tapMessage.draw(at: CGPoint(x: 20, y: 40),
                    withAttributes: nil)
}
```

Drawing the solid circles at touch locations (slide #16)

23

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

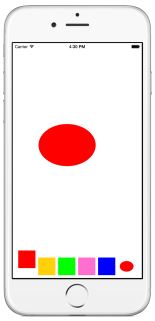
24

A Drawing App

25

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

26

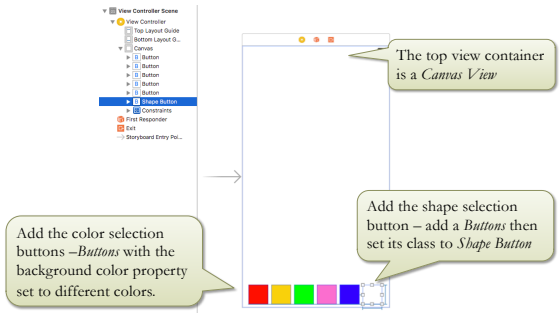
Create the Drawing App

- Start with an *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

27

Drawing Pad – The UI Design



DePaul University 27

28

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

29

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

30

The Shape Button

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

```
class ShapeButton: UIButton {
    var shape: ShapeType = .Line
    var color: UIColor = UIColor.blue

    override func draw(_ rect: CGRect) { … }
}
```

Override the `draw` method

DePaul University 30

31

Shape Button – Drawing Shapes

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

```
override func draw(_ rect: CGRect) {
    if let context = UIGraphicsGetCurrentContext() {
        context.setStrokeColor(color.cgColor)
        context.setFillColor(color.cgColor)
        context.setLineWidth(2)
        Draw shapes (next 3 slides)
    }
}
```

DePaul University 31

32

Shape Button – Drawing Line Shape

```
override func draw(_ rect: CGRect) {
    if let context = UIGraphicsGetCurrentContext() {
        …
        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:
            context.move(to: CGPoint(x: x1, y: y1))
            context.addLine(to: CGPoint(x: x2, y: y2))
            context.strokePath()
        …
```

DePaul University 32

33

Shape Button – Drawing Ellipse and Rectangle

```
override func draw(_ rect: CGRect) {
    if let context = UIGraphicsGetCurrentContext() {
        …
        switch shape {
        case .Line: …
        case .Ellipse:
            context.strokeEllipse(in: rect)
        case .Rectangle:
            context.stroke(rect)
        case .FilledEllipse:
            context.fillEllipse(in: rect)
        case .FilledRectangle:
            context.fill(rect)
        …
```

DePaul University 33

34

Shape Button – Drawing the Scribble Shape

```
override func draw(_ rect: CGRect) {
    if let context = UIGraphicsGetCurrentContext() {
        …
        switch shape {
        …
        case .Scribble:
            context.move(to: CGPoint(x: x1, y: y1))
            context.addCurve(to: CGPoint(x: x2, y: y2),
                control1: CGPoint(x: x1 + 80, y: y1 – 10),
                control2: CGPoint(x: x2 – 80, y: y2 + 10))
            context.strokePath()
        }
    }
}
```

A cubic Bézier curve.

DePaul University 34

35

Drawing Pad – Incremental Implementation

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

DePaul University 35

36

Drawing Pad – Iteration 1 The Canvas View

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

The first and last touch locations

DePaul University 36

37

Drawing Pad – The Canvas View Handle Touches

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

Keep track of the first and last touch locations.

DePaul University 37

38

Drawing Pad – The Canvas View Handle Touches

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

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

DePaul University 38

39

Drawing Pad – The Canvas View Draw Lines

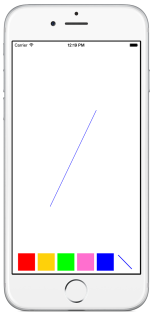
```
override func draw(_ rect: CGRect) {
    if let context = UIGraphicsGetCurrentContext() {
        context.setStrokeColor(color.cgColor)
        context.setFillColor(color.cgColor)
        switch shape {
        case .Line:
            context.move(to: CGPoint(x: first.x, y: first.y))
            context.addLine(to: CGPoint(x: last.x, y: last.y))
            context.strokePath()
        case …
        }
    }
}
```

DePaul University 39

40

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 an ellipse and a rectangle shapes in the *Canvas View*
  - Handle selection of shapes in the *View Controller*



DePaul University 40

41

Drawing Pad – The Canvas View Draw Ellipses & Rectangles

```
override func draw(_ rect: CGRect) {
    if let context = … {
        let rect = CGRect(x: first.x, y: first.y,
            width: last.x - first.x, height: last.y - first.y)
        switch shape { …
        case .Ellipse:
            context.strokeEllipse(in: rect)
        case .Rectangle:
            context.stroke(rect)
        case .FilledEllipse:
            context.fillEllipse(in: rect)
        case .FilledRectangle:
            context.fill(rect)
        case .Scribble: …
        }
    }
}
```

Will handle scribbles later.

42

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

43

Drawing Pad – View Controller

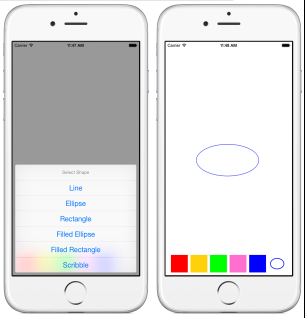
```
@IBAction func selectShape(_ sender: ShapeButton) {
    let title = "Select Shape"
    let alertController = UIAlertController(title: title,
        message: nil, preferredStyle: .actionSheet)
    for shape in shapes {
        let action = UIAlertAction(title: shape.rawValue,
            style: .default)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

Action for selecting a shape

44

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 an ellipse or a rectangle shape



45

Drawing Pad – Iteration 3

- Draw a 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

46

Drawing Pad – Handle Scribble

```
class CanvasView: UIView {
    var shape: ShapeType = .Line
    var color: UIColor = UIColor.blue
    var first: CGPoint = CGPoint.zero
    var last : CGPoint = CGPoint.zero
    var points: [CGPoint] = []
    override func draw(_ rect: CGRect) { … }
    override func touchesBegan(_ touches: Set<UITouch>,
                               with event: UIEvent?) { … }
    override func touchesMoved(_ touches: Set<UITouch>,
                               with event: UIEvent?) { … }
    override func touchesEnded(_ touches: Set<UITouchт>,
                               with event: UIEvent?) { … }
    override func touchesCancelled(_ touches: Set<UITouch>,
                                   with event: UIEvent?) {}
}
```

47

Drawing Pad – The Canvas View Handle Scribble Touches

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


48

Drawing Pad – The Canvas View Handle Scribble Touches

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


49

Drawing Pad – The Canvas View Handle Scribble Touches

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

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


50

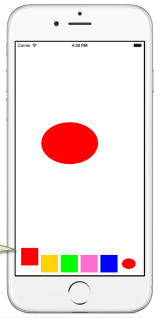
Drawing Pad – The Canvas View Draw Scribble

```
override func draw(_ rect: CGRect) {
    if let context = … {
        switch shape {
        case …
        case .Scribble:
            context.move(to: CGPoint(x: first.x, y: first.y))
            for p in points {
                context.addLine(to: CGPoint(x: p.x, y: p.y))
            }
            context.strokePath()
        }
    }
}
```


51

Drawing Pad – Iteration 4

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



52

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

53

Drawing Pad – View Controller

```
@IBAction func selectColor(_ sender: UIButton) {
    UIView.animate(withDuration: 0.5, delay: 0.0,
        usingSpringWithDamping: CGFloat(0.25),
        initialSpringVelocity: CGFloat(0.25),
        options: UIView.AnimationOptions(),
        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.

54

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

55

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 old value. Default: `oldValue`

DePaul University 55

56

Drawing Pad – Shape Button v.2

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

    override func draw(_ rect: CGRect) { … }
}
```

DePaul University 56

57

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)
    }
    present(alertController, animated: true, completion: nil)
}
```

DePaul University 57

59

Drawing Pad – Set Color, v.2

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

}
```

DePaul University 58

61

Gesture Recognizers

62

Gesture Recognizers

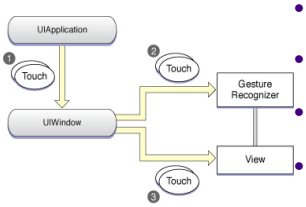
- UIKit provides gesture recognizers for common gestures.
  - Tapping (any number of taps)
    - `UITapGestureRecognizer`
  - Pinching in and out (for zooming a view)
    - `UIPinchGestureRecognizer`
  - Panning or dragging
    - `UIPanGestureRecognizer`
  - Swiping (in a given direction)
    - `UISwipeGestureRecognizer`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

DePaul University 60

63

Gesture Recognizer

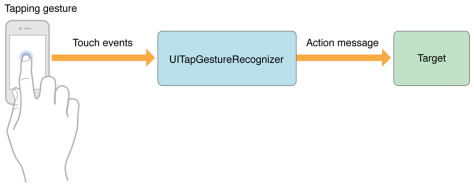
- Observers of touch objects
- Not part of that view hierarchy
- Do not participate in the responder chain.
- May delay the delivery of touch objects to the view
- Cancel delivery of remaining touch objects to the view once they recognize their gesture.



64

Discrete and Continuous Gestures

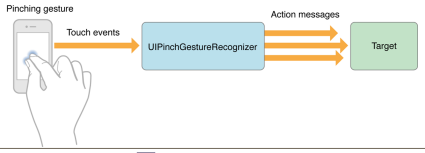
- A discrete gesture happens just once, such as a double-tap
  - The gesture recognizer sends its target a single action message.



65

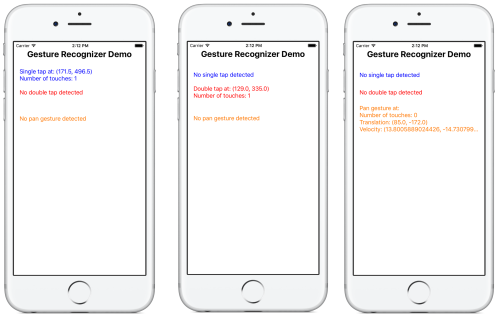
Discrete and Continuous Gestures

- 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.



66

Gesture Recognizer Demo – Single & Double Taps, Panning



67

Detect Single Tap

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


68

Selectors

- A *selector* defines a reference to a method of a class
  - i.e., turns a method call (a statement) into an object reference (an expression)
- Typical use:
  - passing a reference to a method as an argument
- Basic syntax: `#selector(ClassName.methodName)`
  - e.g., `#selector(ViewController.handleSingleTap)`
- Bridged from Objective-C
  - The referenced method must be annotated as `@objc`
  - An alternative to closures, widely used API's of Objective-C

69

Handle Single Tap

```
@objc
func handleSingleTap(_ sender: UITapGestureRecognizer) {
    let n = sender.numberOfTouches
    var message = ""
    for i in 0 ..< n {
        message +=
          " \(sender.location(ofTouch: i, in: view))"
    }
    singleTapLabel.text = "Single tap at:" + message +
        "\nNumber of touches: \(n)"
    DispatchQueue.main.asyncAfter(deadline: .now() + .seconds(3)) {
        self.singleTapLabel.text =  "No single tap detected"
    }
}
```

DePaul University 67

70

Detect Double Tap

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

        singleTapRecognizer.require(
            toFail: doubleTapRecognizer)
    }
}
```

DePaul University 68

71

Handle Double Tap

```
@objc
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)"
    DispatchQueue.main.asyncAfter(deadline: .now() + .seconds(3)) {
        self.singleTapLabel.text =  "No double tap detected"
    }
}
```

DePaul University 69

72

Detect Pan Gesture

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

@objc
func handlePanGesture(_ sender: UIPanGestureRecognizer) {
    …
}
```

DePaul University 70

73

Swipe & Pinch Gesture Recognizer Demo

DePaul University 71

74

Detect Swipe Gestures – Up Swipe

```
override func viewDidLoad() {
    super.viewDidLoad()
    …
    let upSwipeRecognizer =
        UISwipeGestureRecognizer(target: self,
            action: #selector(ViewController.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.transition(from: view1, to: view2,
        duration: 2.0, options: .transitionCurlUp,
        completion: nil)
}
```

75

Detect Swipe Gestures – Down Swipe

```
override func viewDidLoad() {
    super.viewDidLoad()
    …
    let downSwipeRecognizer =
        UISwipeGestureRecognizer(target: self,
            action: #selector(ViewController.handleDownSwipe))
    downSwipeRecognizer.numberOfTouchesRequired = 1
    downSwipeRecognizer.direction = .down
    view.addGestureRecognizer(downSwipeRecognizer)
    …
}
@objc func handleDownSwipe(_ sender: UISwipeGestureRecognizer) {
    let view1 = big_ben.superview != nil ? big_ben : eiffel
    let view2 = big_ben.superview != nil ? eiffel : big_ben
    UIView.transition(from: view1, to: view2,
        duration: 2.0, options: .transitionCurlDown,
        completion: nil)
}
```

76

Detect Swipe Gestures – Left & Right Swipes

```
override func viewDidLoad() {
    super.viewDidLoad()
    …
    let horizontalSwipeRecognizer =
        UISwipeGestureRecognizer(target: self,
        action: #selector(ViewController.handleHorizontalSwipe))
    horizontalSwipeRecognizer.numberOfTouchesRequired = 1
    horizontalSwipeRecognizer.direction = [ .left, .right ]
    view.addGestureRecognizer(horizontalSwipeRecognizer)
    …
}
@objc func handleDownSwipe(_ sender: UISwipeGestureRecognizer) {
    let view1 = big_ben.superview != nil ? big_ben : eiffel
    let view2 = big_ben.superview != nil ? eiffel : big_ben
    UIView.transition(with: self.container, duration: 2.0,
        options: .transitionCrossDissolve, animations: {
            view1.removeFromSuperview()
            self.container.addSubview(view2)
        }, completion: nil)
}
```

77

Detect Pinch Gesture

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

@objc 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 75

78

Sample Code

- Multi-Touch Demo – SB.zip
- Drawing Pad – SB.zip
- Gesture Recognizers Demo – SB.zip
- Swipe Gesture Demo – SB.zip

DePaul University 76

79

Next …

- To infinity and beyond …

- Research project/thesis
- Independent studies
- *Advanced Mobile Application Development*

✧ iOS is a trademark of Apple Inc.

DePaul University 77

80