Gesture and Motion Controls

Dr. Sarah Abraham

University of Texas at Austin
CS329e
Fall 2016
Controller Interfaces

- Allow humans to issue commands to computer
  - Mouse
  - Keyboard
  - Microphone
  - Tablet
Touch-based Interfaces

- User interacts with screen using touch
- Touch has position and direction
- Pattern of touch recognized by hardware and OS
- App provides context to patterns to create use-cases
Gestures

- Touch patterns based on iOS Human Interface Guidelines
- Gestures have expected responses that apps should try to conform to
Receiving Gestures as Events

1. Hardware notifies operating system about user action
2. OS builds `UITouch` object from touch information
3. `UITouch` object placed inside `UIEvent` object
4. `UIEvent` sent to application
5. `UIResponder` handles event
iOS Events

- Touch events contain one or more finger gestures on-screen
- Motion events process orientation and accelerometer information
- Remote control events receive commands from device accessories
UIResponder objects can handle events and recognize touch gestures

- Includes UIApplication, UIView, and UIViewController
Gesture Recognizers

- **UIGestureRecognizer** class associated with a view
- Monitor for predefined gestures made in view
- Perform an action once a valid gesture is detected
1. Add gesture recognizer to view in Storyboard

2. Add event handling to view’s view controller in Xcode Editor
   ✤ @IBAction func respondToGesture (recognizer: [UIGestureRecognizer subclass]) {}

3. Associate gesture recognizer in Storyboard with event handler
   ✤ Control-click Recognizer
   ✤ Select its view controller to choose function
Some Details

- Gesture recognizer associated with a single view
- Notifies callback function if touch matches a gesture on this view
- Multiple gesture recognizers needed for multiple views
- Callback handling can be shared between multiple gesture recognizers
- View must have “User Interaction Enabled” to recognize gestures
- View must have “Multiple Touch” Enabled to handle multitouch gestures
UITapGestureRecognizer

- Discrete touch to screen
- Single or multiple taps
- One or two fingers
- Can set number of fingers and number of taps
- Often used to select an item
UISwipeGestureRecognizer

- Straight movement in one direction (up, down, left, right)
- Direction property indicates type of swipe
  - Right is default
- Each swipe gesture recognizer only recognizes one type of swipe in one direction
- Used to switch between views or rapidly scroll in a given direction
UIPanGestureRecognizer

- Continuous touch across screen
- Drag based on one or more fingers moving in a direction
  - User specified minimum and maximum number of fingers to activate gesture
- Can track translation and velocity of gesture
- Used to move an element on the screen
UIPinchGestureRecognizer

- Continuous touch across screen
- Requires two fingers
- Scale factor based on finger position
- Used to zoom in (pinch apart) or zoom out (pinch together)
UIRotationGestureRecognizer

- Continuous touch across screen
- Requires two fingers
- Rotation (in radians) based on finger position and circular movement relative to each other
- Used to rotate a view around a common center
UILongPressGestureRecognizer

- Continuous touch across screen
- One or more fingers
- Press must be held a minimum amount of time before action triggers
- Fingers may not move beyond a specified distance
- Used to select for editing
Motion Events

- Generated when user moves, shakes or tilts the device

- Detected by accelerometer:
  - One in each axis (X, Y, Z)
  - Measures velocity over time along a linear path

- And gyroscope:
  - Measures rate of rotation around three axes (X, Y, Z)
Device Orientation

- Basic physical orientations available in UIDevice class
  - UIDeviceOrientationLandscapeLeft
  - UIDeviceOrientationLandscapeRight
  - UIDeviceOrientationPortrait
  - UIDeviceOrientationPortraitUpsideDown
  - UIDeviceOrientationFaceUp
  - UIDeviceOrientationFaceDown
  - UIDeviceOrientationUnknown
Device vs Interface Orientations

- Device orientation is related to the physical orientation of the device
- Interface orientation is related to the interface display’s orientation for the viewer:
  - UIInterfaceOrientationPortrait
  - UIInterfaceOrientationPortraitUpsideDown
  - UIInterfaceOrientationLandscapeLeft
  - UIInterfaceOrientationLandscapeRight
- Use device orientation for motion events, use interface orientation for designing displays
Shake Gesture

- Accelerometer determines that shake gesture occurred
- Operating system creates `UIEvent` to pass to active apps
- Event includes:
  - Motion start
  - Motion stop
  - Timestamp
- Object in app designated the “first responder” handles this event
**Motion Event Handling**

- Appropriate view controller made first responder:
  ```swift
  func canBecomeFirstResponder() -> Bool { return true }
  ```

- Implement motion handling:
  ```swift
  func motionBegan(motion: UIEventSubtype, withEvent event: UIEvent)
  func motionEnded(motion: UIEventSubtype, withEvent event: UIEvent)
  func motionCancelled(motion: UIEventSubtype, withEvent event: UIEvent)
  ```
Core Motion

- Framework for handling more generalized motion inputs
- Supports access to both raw and processed accelerometer data
- Wide range of sources
  - Accelerometer, pedometer, magnetometer, altitude, attitude, motion activity etc
- Not available to test in simulator — must use a device
Euler Angles

- Pitch (rotation around the X-axis)
- Roll (rotation around the Y-axis)
- Yaw (rotation around the Z-axis)
CMMotionManager

- Shared instance throughout app to handle motion data
- Provides interface for four motion data types:
  - Accelerometer
  - Gyro
  - Magnetometer
  - deviceMotion
Using CMMotionManager

1. Declare import CoreMotion

2. Instantiate CMMotionManager as a property within the necessary view controller
   ✤ let manager = CMMotionManager()

3. Check for data on given operation queue
   ✤ Uses closure functionality
Checking for Accelerometer Data

```swift
if manager.accelerometerAvailable {
    manager.accelerometerUpdateInterval = 0.1
    manager.startAccelerometerUpdatesToQueue(NSOperationQueue.mainQueue()) {
        [weak self] (data: CMAccelerometerData!, error: NSError!) in
            /* Process data.acceleration here */
    }
}
```

<http://nshipster.com/cmdevicemotion/>