Gesture Recognizers

Elizabeth Reid
Finger != Mouse

- You have more than one finger
- A finger touches more than a single point on a screen
- When user puts their finger on something, they can’t see it any more
- Poor performance is more obvious to a user when they are “touching” it
iOS Touch Processing

- iOS takes the area of contact and turns it into a single point
- Area of contact is usually an oval
- Position and angle of the oval used to estimate where the user “meant” to touch
- The takeaway: you get a single point, but that isn’t the only place the user touched, and may not be where they meant. But it’s probably close.
Don’t Expect Precision

- Don’t design any UI that requires very precise finger motions, or the user to hit a small target
- Remember the user can’t see what’s directly under their finger
Show What They Hit
You need to make anything that the user touches big enough for them to hit. Or you need to “help” them if they’re close.
Discoverability

- Non-mobile OSes have menus to look through
- Hotkeys for power users
- Tooltips
- No menus for iOS apps — how does the user know what they can do?
- UI feedback to clue users in about gestures
Gesture Collision

- There are often fewer simple gestures than important actions.

- Be careful that you don’t overload the same motion to mean different things.
  - e.g. drag vs. pan

- If it’s hard to explain a gesture out loud, the user will never figure it out.
Gestures Inform UI

- Sometimes your choice of what gesture to use can make it clearer what your UI should be
- e.g. pull down to update
A Physical Response

- Gestures feel more natural when the response matches users’ physical expectations
  - Pinch to zoom
  - Bounce on scroll
- Try to keep the same point under the user’s finger
Standard Gestures

- Tapping
- Pinching
- Swiping
- Panning/dragging
- Rotating
- Long press
Gesture Recognizers

Note: This chapter contains information that used to be in iPad Programming Guide. The information in this chapter has not been updated specifically for iOS 4.0.

Applications for iOS are driven largely through events generated when users touch buttons, toolbars, table-view rows and other objects in an application's user interface. The classes of the UIKit framework provide default event-handling behavior for most of these objects. However, some applications, primarily those with custom views, have to do their own event handling. They have to analyze the stream of touch objects in a multitouch sequence and determine the intention of the user.

Most event-handling views seek to detect common gestures that users make on their surface—things such as triple-tap, touch-and-hold (also called long press), pinching, and rotating gestures. The code for examining a raw stream of multitouch events and detecting one or more gestures is often complex. Prior to iOS 3.2, you cannot reuse the code except by copying it to another project and modifying it appropriately.

To help applications detect gestures, iOS 3.2 introduces gesture recognizers, objects that inherit directly from the UIGestureRecognizer class. The following sections tell you about how these objects work, how to use them, and how to create custom gesture recognizers that you can reuse among your applications.

Gesture Recognizers Simplify Event Handling

UIGestureRecognizer is the abstract base class for concrete gesture-recognizer subclasses (or, simply, gesture recognizers). The
Basic Behavior

- UIGestureRecognizers attach to UIViews
- Receives touches on that view and its subviews
- Multiple gesture recognizers can watch touches on a given view
- View may still receive touches directly even when it has gesture recognizers

- Probably simpler to not bother, though
Action Messages

• When a gesture recognizes, it fires one or more action messages to its target(s)

• You initialize a gesture with a target and action pair, but more can be added later

• Actions fire in a nondeterministic order
Add a GR to a UIView

// make the tap gesture
UITapGestureRecognizer* doubleTapGR =
    [[UITapGestureRecognizer alloc]
    initWithTarget:target
    action:@selector(handleDoubleTap:)];

// specify that it is a double tap
doubleTapGR.numberOfTapsRequired = 2;

// add it to a view
[view addGestureRecognizer:doubleTapGR];
Target Action

-(void) handleDoubleTap:
(UITapGestureRecognizer*)doubleTapGR
{
    CGPoint tapPoint = [doubleTapGR
        locationInView:doubleTapGR.view];
    // Do something with this information
    ...
    // Profit!
}


Basic Touch Processing

- When a touch begins, each gesture is given a chance to see it.
- The order in which gestures are shown the touch is nondeterministic.
- Once all of the gestures have seen the touch, the OS updates its internal bookkeeping and fires any appropriate target actions.
- Lather, rinse, repeat as necessary.
When to Fire an Action

- UIGestureRecognizerState property.
- Possible states: Possible, Recognized, Began, Changed, Ended, Failed, Cancelled
- A gesture always starts in state Possible
- Through analysis of touches, a gesture may leave state Possible
- Actions fire from states Recognized, Began, Changed, Ended, Cancelled
Discrete Gestures

- Tap, double tap, triple tap, swipe
- Remains in state Possible until it sees the entire series of events, then enters state Recognized
- Its target action(s) fire only once.
Double Tap

Finger down

Finger up

Finger down

Finger up

state Possible

point of recognition

state Recognized

target actions fire
Continuous Gestures

- Pan, rotate, pinch, long press
- Remains in state Possible until it sees a series of touch events needed to identify the gesture
- Target action fires once it enters state Began, continues firing in state Changed and Ended
- Touch information continues to be given to the gesture until it reaches state Ended or Cancelled
Pinch

First Finger down
Second Finger down
Fingers move apart

Fingers keep moving

Fingers up

state Possible

state Began

state Changed

point of recognition

target actions fire

state Ended
Failure - Pinch

Finger down → Finger down → Finger up → No more touch information sent

state Possible → state Failed
Reset the Logic

- At the end of a sequence, a gesture will be in state Recognized, Ended, Failed or Cancelled
- It needs to start a sequence in state Possible
- OS keeps track of this, and will reset the logic of gestures that are done
- Allows gestures to clear any internal state
Inter-Gesture Interaction
Inter-Gesture Interaction

- Default OS behavior is that no two gestures can recognize at the same time.
Inter-Gesture Interaction

- Default OS behavior is that no two gestures can recognize at the same time.
- What if we have two gestures on a view that want to recognize?
Inter-Gesture Interaction

• Default OS behavior is that no two gestures can recognize at the same time
• What if we have two gestures on a view that want to recognize?
• Only one will be allowed.
Inter-Gesture Interaction

• Default OS behavior is that no two gestures can recognize at the same time.

• What if we have two gestures on a view that want to recognize?

• Only one will be allowed.

• Which one?
Inter-Gesture Interaction

• Default OS behavior is that no two gestures can recognize at the same time.

• What if we have two gestures on a view that want to recognize?
  • Only one will be allowed.

• Which one?
  • Gesture most recently added to the view.
Inter-Gesture Interaction

- Default OS behavior is that no two gestures can recognize at the same time.
- What if we have two gestures on a view that want to recognize?
  - Only one will be allowed.
- Which one?
  - Gesture most recently added to the view.
- What if we want both to recognize?
• A gesture can be assigned a delegate to control some of its behavior

- (BOOL) gestureRecognizer:(UIGestureRecognizer*)gesture shouldRecognizeSimultaneouslyWithGestureRecognizer:(UIGestureRecognizer*)otherGesture;

- (BOOL) gestureRecognizer:(UIGestureRecognizer*)gesture shouldReceiveTouch:(UITouch*)touch;

- (BOOL) gestureRecognizerShouldBegin:(UIGestureRecognizer*)gesture;
- (BOOL) gestureRecognizer: 
  shouldRecognizeSimultaneouslyWithGestureRecognizer:

• Called when a gesture recognizes, for it and each remaining gesture on the view

• If either delegate returns YES, the unrecognized gesture remains in state Possible

• If no delegate, or delegate returns NO, unrecognized gesture is moved to state Failed
- (BOOL) gestureRecognizer:shouldReceiveTouch:

• Called before a gesture is told about a new touch

• If the delegate returns NO, the gesture does not see the touch

• OS assumes YES if no delegate implementation
<UIGestureDelegate>

-(BOOL) gestureRecognizerShouldBegin:

- Called when the gesture attempts to transition out of state Possible into state Began
- If the delegate returns NO, the gesture will move to state Failed instead
- OS assumes YES if no delegate implementation
Failure Requirement

• What if we want the opposite: a gesture should only recognize if another gesture fails
  
• e.g. double tap prevents single tap
  
• [A requireGestureRecognizerToFail:B] adds a failure dependency between gestures A and B
  
• A’s target actions won’t fire until B fails
  
• If B recognizes, A will fail
  
• OS won’t reset A or B until both are ready
Failure Requirement
Implications

- If a single tap requires that a double tap fails, how does that affect behavior?
- Single tap can’t fire immediately, which creates a noticeable delay
Taps Without Failure Dependency

- Finger down
- Finger up
- Finger down
- Finger up

Single tap recognizes & fires

Point of recognition

Target actions fire

Double & single taps recognize & fire
Taps With Failure Dependency

- Finger down
- Finger up
- No Finger down

- Single tap recognizes
- Single tap fires

Point of recognition
Target actions fire
Failure Requirement
Implications

• Also true for continuous gestures: pan vs. pinch

• User may put one finger down first and move it some before a second finger lands to pinch

• If you respond to the pan gesture immediately, you may have to undo its action in order to pinch

• This feels very unnatural
Failure Requirement Implications

- Aim for additive gesture systems
- e.g. single tap action should not conflict with double tap action
- Makes entire system feel more responsive
- Easier for users to understand
- This isn’t always possible to achieve, but always keep it in mind
Custom Gestures

• You can make a UIGestureRecognizer subclass
• Manage the gesture’s state explicitly
• Methods to override:

- (void)reset;
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event;
- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event;
- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event;
- (void)touchesCancelled:(NSSet *)Touches withEvent:
  (UIEvent *)event;
Custom Gestures

- Fail custom gestures as early as you possibly can.
- This is important because of possible failure requirements.
- Sometimes this means you need to use a timer or delayed callback rather than wait for a new touch event.
- Use `dispatch_after`, `performSelectorAfterDelay` or `NSTimer`.
Custom Gestures

• Can also control simultaneous recognition with a subclass

• Works similarly to the delegate, but gives class-wide control

• e.g. UITapGestureRecognizer always allows taps with a higher tap count (single tap does not prevent double or triple tap)

  - (BOOL)canBePreventedByGestureRecognizer:
    (UIGestureRecognizer *)preventingGestureRecognizer;
  - (BOOL)canPreventGestureRecognizer:
    (UIGestureRecognizer *)preventedGestureRecognizer;
Demo!