Extracted from:

iPad Programming

A Quick-Start Guide for iPhone Developers

The Pragmatic Programmers

iPad Programming

A Quick-Start Guide for iPhone Developers

Daniel H Steinberg
Eric T Freeman

Edited by Colleen Toporek

ISBN-10: 1-934356-57-3

ISBN-13: 978-1-934356-57-9

Printed on acid-free paper.

B1.0 printing, April 3, 2010

Version: 2010-4-14

Chapter 3

Using Gestures

The iPad's large surface isn't just for viewing: it's for getting your hands in the mix and using gestures to interact with your applications. Of course, the iPhone also uses multi-touch as its primary means of interaction, but its small screen constrains us from fully exploring gestures, and ultimately limits our ability to create an immersive experience.

Another aspect of the iPhone has made using gestures in your applications challenging—the SDK. To write gestures, you had to cut and paste multi-touch code between classes and applications, without any convenient means for reuse. Beginning in iPhone OS 3.2, Apple has refined the way gestures fit into the UIKit architecture and given you the ability to abstract and reuse gestures. Apple's also provided some ready-made gestures for you to use, including taps, pans, pitches, rotation, and swipes—and, if the built-in gestures don't meet your needs, then you can easily extend the new architecture and write your own.

As you might expect, there is a fair bit more to writing and using gesture recognizers, and in this chapter we'll explore the API in detail; we'll start by using and configuring pre-built recognizers, and then add more sophisticated behavior, customizing and controlling how the recognizers behave. Finally, we'll create own recognizer to look for our own custom gesture. Let's get started!

3.1 iPad Virtual Bubble Wrap

Who would have thought when the Sealed Air Corporation invented BubbleWrap™ in 1957, it would take on a life of its own as an addictive and satisfying activity for all ages?[1] After all, when presented with a fresh sheet of all those air-filled bubbles, who doesn't want to start popping them?

Of course, another invention, the Internet, came along and brought with it *virtual* bubble wrap; and today, all you have to say is "bubble wrap" to hear the response "There's an App for That!" and bubble wrap has found its way onto the iPhone. What's next? Virtual bubble wrap for the iPad, of course—just imagine all that screen space and multi-touch gestures devoted to popping simulated, air-filled bubbles.

In this chapter, we're going to create our own virtual bubble wrap application and fully explore the new gesture API in the process. Rather than present a conventional sheet of bubbles that can be popped, we're going to use gestures to make things a little more interesting: in this application, you'll create your own custom bubble wrap layouts (through tap gestures), pop all the bubbles you want (using the tap gesture in a slightly different manner), resize bubbles (with a pinch gesture), swipe the screen and start over (through a swipe gesture), and even implement a custom gesture that will allow you to delete bubbles.

To to get started, we're going to:

1. Create a simple view-based application to act as a "bubble sheet."
2. Use a built-in gesture to capture a "tap" and place a bubble on the sheet for each tap.
3. Reuse this tap gesture to pop the bubble.

Once we have the basics, we'll circle back and add resizing, swiping, and deletion—all through gestures.

3.2 Using Simple Tap Gestures

Before we dive in, we have time for a quick pep talk—if you have experience with the UIKit views and you have even a passing familiarity of a multi-touch events, then understanding the new gesture design is straightforward: the new architecture resolves around gesture recognizers that you can instantiate and attached to any view. These recognizers

1. If you aren't familiar with BubbleWrap, it is a transparent sheet of plastic embedded with air-filled bubbles that is used as a protective means of packaging items for shipping. Many hours of entertainment have resulted from squeezing the air bubbles (and producing the loud pop that accompanies it).

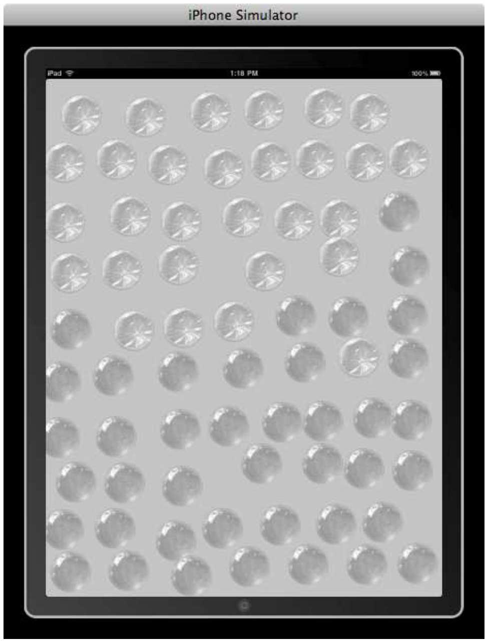

Figure 3.1: The Bubble Wrap Interface

act as observers of multi-touch events and get a peek at them before your views do. Recognizers process multi-touch events until their gesture is recognized and then send an action message to a target that you specify; if your recognizer doesn't find a gesture in a sequence of multi-touch events, then the events are passed on to other recognizers (if you have any) and ultimately, if no gesture is recognized, to the view itself. With that short summary out of the way, the best way to understand gestures is to see them in action, so let's get the implementation started.

To get this thing rolling, we've already created a project named Bubbles1; to give the application more of a bubble wrap feel, we've taken the liberty of opening the Bubbles1ViewController.xib Nib file in Interface Builder and changed the background of the main view to a nice medium gray. For dramatic effect (that you'll see later on), we also changed the main window's background color to solid black in the MainWindow.xib Nib file. Finally, we've placed two images in the project, one of an inflated bubble (named bubble.png), and one of a popped bubble (named popped.png).

Instantiating the Gesture

Our first coding task is to create a tap gesture that, when recognized, will result in a bubble being placed on the main application view. To accomplish that, we override viewDidLoad in the Bubbles1ViewController and instantiate a tap recognizer. Let's take a look:

Download gestures/Bubbles1/Classes/Bubbles1ViewController.m

```
- (void)viewDidLoad {
        [super viewDidLoad];

        UITapGestureRecognizer *tapRecognizer =
                [[UITapGestureRecognizer alloc]
                 initWithTarget:self
                 action:@selector(handleTapFrom:)];
}
```

Here we instantiate a tap recognizer by calling UITapGestureRecognizer's initWithTarget:action: method. Gesture recognizers use the common Cocoa target/action pattern, whereby a target will be sent an action message when the gesture is recognized. We've designated self as the target and handleTapFrom: as the action, so when this gesture is recognized, the handleTapFrom method will be called on our instance of Bubbles1ViewController.

Attaching the Gesture to a View

With any gesture, after you instantiate it, you're going to need to configure it through its properties, and you're going to have to attach it to a view for it to be useful. Here's how we do that with our tap recognizer:

Download gestures/Bubbles1/Classes/Bubbles1ViewController.m

```
  - (void)viewDidLoad {
          [super viewDidLoad];

          UITapGestureRecognizer *tapRecognizer =
                  [[UITapGestureRecognizer alloc]
                   initWithTarget:self
                   action:@selector(handleTapFrom:)];
▶         [tapRecognizer setNumberOfTapsRequired:1];
▶         [self.view addGestureRecognizer:tapRecognizer];
▶         [tapRecognizer release];
  }
```

We first configure the recognizer through its setNumberOfTapsRequired: method, which allows us to specify how many taps the recognizer should look for. In this case, one tap is probably going to be the most natural gesture for our users. As you can probably guess, the setNumberOfTapsRequired: method is specific to a tap recognizer; we're going to see that other types of recognizers have their own specific configuration properties.

We then take the Bubbles1ViewController's view and attach the tapRecognizer using the addGestureRecognizer method—a brand new method added to UIView's interface in iPhone OS 3.2. Once the recognizer is attached to a view it gets to observe all multi-touch events that hit the view to look for gestures.

Finally, as good citizens, we release the recognizer now that the view owns it.

Acting on the Gesture

Now that we have the tap recognizer instantiated and attached to a view, we need to implement the specific behavior for the action—placing a bubble on the view. How do we approach that? We're going to need to determine the location of the tap, and then we'll need to create an image representing the bubble and add it to the Bubbles1ViewController's view. So, let's write a handleTapFrom: method that determines the location of the tap and creates a bubble:

Download gestures/Bubbles1/Classes/Bubbles1ViewController.m

```
- (void)handleTapFrom:(UITapGestureRecognizer *)recognizer {
        CGPoint location = [recognizer locationInView:self.view];

        CGRect rect = CGRectMake(location.x - 40,
                location.y - 40, 80.0f, 80.0f);
        UIImageView *image = [[UIImageView alloc]
                initWithFrame:rect];
        [image setImage:[UIImage imageNamed:@"bubble.png"]];

        [self.view addSubview:image];

        [image release];
}
```

Let's step through this code: first, handleTapFrom: is passed the recognizer object that accepted the gesture.[2] We can use the recognizer to determine the location where the tap occurred by using its locationIn-

2. We'll discuss precisely what it means to accept a gesture later in the chapter, for now, when a tap recognizer sees a multi-touch set of events that look like a tap gesture, it recognizes the gesture and invokes the action on the target.

View: method, which, given a view, will tell us the point within that view where the gesture occurred. Here we're most interested in the location of the tap within the Bubbles1ViewController's view, so we pass self.view as the view.

Now that we have the point where the tap occurred, we need to create a bubble and place it at that location. We do that by computing a bounding box for the image (using the tap location as a guide), instantiating an image with that bounding box, and setting the source of the image to bubbles.png. Finally, all we need to do is add the image as a subview to our self.view.

Creating Some Bubbles

At this point, you're ready to test this code. Go ahead and build and run the project, and you should see a blank, gray screen—simply tap within the view (or click using the simulator), and you should see bubbles appear under your taps.

Before we go further, let's step back to think about what we've done:

- We created a tap gesture recognizer and configured it to recognize a one-tap gesture.
- We took that gesture object and attached it to our Bubbles1ViewController's view, so anytime multi-touch events occur on that view, our gesture recognizer gets a peek at those events and can decide whether they constitute a tap.
- And, assuming there is a tap, we told the gesture recognizer to call the handleTapFrom: method on our Bubbles1ViewController.
- We also wrote the code for the handleTapFrom: method, which determines the location of the tap from the recognizer object, and then creates an image that looks like a bubble and places it at that location.

Not bad for such a small amount of code, huh? Now let's make things more interesting by enhancing the bubbles so they can be popped.

3.3 Multi-Touch Events and the View Hierarchy

To create a popped-bubble effect, we need to know when a user taps on a bubble. How do we accomplish that? You might be thinking that one way of recognizing when a bubble (or in our case an UIImageView