Stanford CS193p

Developing Applications for iPhone 4, iPod Touch, & iPad

Fall 2010

Today

- Gesture Recognizers
  - How to get "input" into your UIView
- Demo
  - Universal Application
  - UISplitViewController
  - Handling pinch gesture
  - Handling device rotation (shouldAutorotate... and springs and struts)

UIGestureRecognizer

- We've seen how to draw in our `UIView`, how do we get touches?
  We can get notified of the raw touch events (touch down, moved, up).
  Or we can react to certain, predefined "gestures." This latter is the way to go.
- Gestures are handled by the class `UIGestureRecognizer`
  This class is "abstract." We only actually use "concrete subclasses" of it.
- There are two sides to using a gesture recognizer
  1. Adding a gesture recognizer to a `UIView` to ask it to recognize that gesture.
  2. Providing the implementation of a method to "handle" that gesture when it happens.
- Usually #1 is done by a Controller
  Though occasionally a `UIView` will do it to itself if it just doesn't make sense without that gesture.
- Usually #2 is provided by the `UIView` itself
  But it would not be unreasonable for the Controller to do it.
  Or for the Controller to decide it wants to handle a gesture differently than the view does.

UIGestureRecognizer

- Adding a gesture recognizer to a UIView from a Controller

```
- (void)viewDidLoad
{




}
```

UIGestureRecognizer

- Adding a gesture recognizer to a UIView from a Controller

```
- (void)viewDidLoad
{
    UIView *panView = ...;   // this is one of the Controller's views that we want to recognize "pan" gestures




}
```

UIGestureRecognizer

- Adding a gesture recognizer to a UIView from a Controller

```
- (void)viewDidLoad
{
    UIView *panView = ...;   // this is one of the Controller's views that we want to recognize "pan" gestures
    UIGestureRecognizer *pangr =
        [[UIPanGestureRecognizer alloc] initWithTarget:panView action:@selector(pan:)];


}
```

This is a concrete subclass of UIGestureRecognizer that recognizes "panning" (moving something around with your finger).

There are, of course, other concrete subclasses (for swipe, pinch, tap, etc.).

UIGestureRecognizer

- Adding a gesture recognizer to a UIView from a Controller

```
- (void)viewDidLoad
{
    UIView *panView = ...;   // this is one of the Controller's views that we want to recognize "pan" gestures
    UIGestureRecognizer *pangr =
        [[UIPanGestureRecognizer alloc] initWithTarget:panView action:@selector(pan:)];


}
```

Note that we are specifying the view itself as the target to handle a pan gesture when it is recognized. Thus the view will be both the recognizer and the handler of the gesture.

The UIView does not <u>have</u> to handle the gesture. It could be, for example, the controller that handles it. But it is most <u>often</u> that the view that handles the gestures that it recognizes.

UIGestureRecognizer

- Adding a gesture recognizer to a UIView from a Controller

```
- (void)viewDidLoad
{
    UIView *panView = ...;   // this is one of the Controller's views that we want to recognize "pan" gestures
    UIGestureRecognizer *pangr =
        [[UIPanGestureRecognizer alloc] initWithTarget:panView action:@selector(pan:)];


}
```

This is the action method that will be sent to the target (the panView) during the handling of the recognition of this gesture.

This version of the action message takes one argument (which is the UIGestureRecognizer that sends the action), but there is another version that takes no arguments if you'd prefer.

We'll look at the implementation of this method in a moment.

UIGestureRecognizer

- Adding a gesture recognizer to a UIView from a Controller

```
- (void)viewDidLoad
{
    UIView *panView = ...;   // this is one of the Controller's views that we want to recognize "pan" gestures
    UIGestureRecognizer *pangr =
        [[UIPanGestureRecognizer alloc] initWithTarget:panView action:@selector(pan:)];
    [panView addGestureRecognizer:pangr];

}
```

If we don't do this, then even though the panView implements pan:, it would never get called because we would have never added this gesture recognizer to the view's list of gestures that it recognizes.

Think of this as "turning the handling of this gesture on."

UIGestureRecognizer

- Adding a gesture recognizer to a UIView from a Controller

```
- (void)viewDidLoad
{
    UIView *panView = ...;   // this is one of the Controller's views that we want to recognize "pan" gestures
    UIGestureRecognizer *pangr =
        [[UIPanGestureRecognizer alloc] initWithTarget:panView action:@selector(pan:)];
    [panView addGestureRecognizer:pangr];
    [pangr release];
}
```

And, of course, we alloc'ed this concrete gesture recognizer, so we must release it (which we can do now because we've added it to the view's list of gestures to recognize, so the view has taken ownership of this object).

UIGestureRecognizer

- Adding a gesture recognizer to a UIView from a Controller

```
- (void)viewDidLoad
{
    UIView *panView = ...;   // this is one of the Controller's views that we want to recognize "pan" gestures
    UIGestureRecognizer *pangr =
        [[UIPanGestureRecognizer alloc] initWithTarget:panView action:@selector(pan:)];
    [panView addGestureRecognizer:pangr];
    [pangr release];
}
```

Only UIView instances can recognize a gesture (because UIViews handle all touch input).
But any object can tell a UIView to recognize a gesture (by adding a recognizer to the UIView).
And any object can handle the recognition of a gesture (by being the target of the gesture's action).
(But usually the UIView handles the gestures it is asked to recognize.)

UIGestureRecognizer

- How do we implement the target of a gesture recognizer?
  Each concrete class provides some methods to help you do that.

- For example, UIPanGestureRecognizer provides 3 methods

```
- (CGPoint)translationInView:(UIView *)aView;
- (CGPoint)velocityInView:(UIView *)aView;
- (void)setTranslation:(CGPoint)translation inView:(UIView *)aView;
```

- Also, the base class, UIGestureRecognizer provides this property

```
@property (readonly) UIGestureRecognizerState state;
```

  Gesture Recognizers sit around in the state `Possible` until they start to be recognized
  Then the either go to `Recognized` (for discrete gestures like a tap)
  Or they go to `Began` (for continuous gestures like a pan)
  At any time, the state can change to `Failed` (so watch out for that)
  If the gesture is continuous, it'll move on to the `Changed` and eventually the `Ended` state
  Continuous can also go to `Cancelled` state (if the recognizer realizes it's not this gesture after all)

UIGestureRecognizer

- So, given these methods, what would pan: look like?

```
- (void)pan:(UIPanGestureRecognizer *)recognizer
{








}
```

UIGestureRecognizer

- So, given these methods, what would `pan:` look like?

```
- (void)pan:(UIPanGestureRecognizer *)recognizer
{
    if ((sender.state == UIGestureRecognizerStateChanged) ||
        (sender.state == UIGestureRecognizerStateEnded)) {



    }
}
```

We're going to update our view every time the touch moves (and when the touch ends). This is "smooth panning."

UIGestureRecognizer

- So, given these methods, what would `pan:` look like?

```
- (void)pan:(UIPanGestureRecognizer *)recognizer
{
    if ((sender.state == UIGestureRecognizerStateChanged) ||
        (sender.state == UIGestureRecognizerStateEnded)) {
        CGPoint translation = [recognizer translationInView:self];



    }
}
```

This is how much the gesture moved.
The new movement is added to the translation that was there before.
Obviously the "translation that was there before" is zero at the start.

UIGestureRecognizer

- So, given these methods, what would pan: look like?

```
- (void)pan:(UIPanGestureRecognizer *)recognizer
{
    if ((sender.state == UIGestureRecognizerStateChanged) ||
        (sender.state == UIGestureRecognizerStateEnded)) {
        CGPoint translation = [recognizer translationInView:self];
        // move something in myself (I'm a UIView) by translation.x and translation.y
        // for example, if I were a graph and my origin was set by an @property called origin
        self.origin = CGPointMake(self.origin.x+translation.x, self.origin.y+translation.y);

    }
}
```

UIGestureRecognizer

- So, given these methods, what would pan: look like?

```
- (void)pan:(UIPanGestureRecognizer *)recognizer
{
    if ((sender.state == UIGestureRecognizerStateChanged) ||
        (sender.state == UIGestureRecognizerStateEnded)) {
        CGPoint translation = [recognizer translationInView:self];
        // move something in myself (I'm a UIView) by translation.x and translation.y
        // for example, if I were a graph and my origin was set by an @property called origin
        self.origin = CGPointMake(self.origin.x+translation.x, self.origin.y+translation.y);
        [recognizer setTranslation:CGPointZero inView:self];
    }
}
```

Here we are resetting the "translation that was there before" to zero.

Now each time this is called, we'll get the "incremental" movement of the gesture (which is what we want). If we wanted the "cumulative" movement of the gesture, we would not include this line of code.

UIGestureRecognizer

- So, given these methods, what would pan: look like?

```
- (void)pan:(UIPanGestureRecognizer *)recognizer
{
    if ((sender.state == UIGestureRecognizerStateChanged) ||
        (sender.state == UIGestureRecognizerStateEnded)) {
        CGPoint translation = [recognizer translationInView:self];
        // move something in myself (I'm a UIView) by translation.x and translation.y
        // for example, if I were a graph and my origin was set by an @property called origin
        self.origin = CGPointMake(self.origin.x+translation.x, self.origin.y+translation.y);
        [recognizer setTranslation:CGPointZero inView:self];
    }
}
```

Other Concrete Gestures

- UIPinchGestureRecognizer

  `@property CGFloat scale;` // note that this is NOT `readonly` (can reset each movement)

  `@property (readonly) CGFloat velocity;` // note that this IS `readonly`; scale factor per second

- UIRotationGestureRecognizer

  `@property CGFloat rotation;` // note that this is NOT `readonly`; in radians

  `@property (readonly) CGFloat velocity;` // note that this IS `readonly`; radians per second

- UISwipeGestureRecognizer

  This one you "set up" (w/the following) to find certain swipe types, then look for `Recognized state`

  `@property UISwipeGestureRecognizerDirection direction;` // what direction swipes you want

  `@property NSUInteger numberOfTouchesRequired;` // two finger swipes? or just one finger? more?

- UITapGestureRecognizer

  Set up (w/the following) then look for `Recognized state`

  `@property NSUInteger numberOfTapsRequired;` // single tap or double tap or triple tap, etc.

  `@property NSUInteger numberOfTouchesRequired;` // e.g., require two finger tap?

Coming Up

- Demo
  - Universal Application
  - UISplitViewController
  - Handling pinch gesture
  - Handling device rotation (shouldAutorotate... and springs and struts)
- Homework
  - Make your Calculator work on iPad (and still work on iPhone)
  - Use a UISplitViewController on iPad
  - Support pinch (change scale), pan (change origin) and tap (reset origin) gestures
  - Use NSUserDefaults to maintain some application state across launches
- Next Week
  - UITableView
  - UIScrollView
  - UIImageView