Stanford CS193p

Developing Applications for iPhone 4, iPod Touch, & iPad

Fall 2010

Today: Grab Bag

- Finish Shutterbug Map Demo
  Add thumbnails
- `UITextField` and `UITextView`
  Editable text fields (unlike `UILabel` which is static text only)
- Modal View Controllers
  Temporary, "don't let the user do anything until he or she attends to this" views
- `UIView` Animation
  Animating changes in a few key `UIView` properties
- Core Motion
  Accelerometer and Gyro inputs

UITextField

- Like `UILabel`, but editable
  - Typing things in on an iPhone is secondary UI (keyboard is tiny).
  - More of a mainstream UI element on iPad.
  - Don't be fooled by your UI in the simulator (because you can use physical keyboard!).
  - You can set text color, alignment, font, etc., just like a `UILabel`.
- Keyboard appears when `UITextField` becomes "first responder"
  - It will do this automatically when the user taps on it.
  - Or you can make it the first responder by sending it the `becomeFirstResponder` message.
  - To make the keyboard go away, send `resignFirstResponder` to the `UITextField`.
- The text is obtained from the `UITextField` via its `delegate`

```
- (BOOL)textFieldShouldReturn:(UITextField *)sender;  // sent when return key is pressed
- (void)textFieldDidEndEditing:(UITextField *)sender;
```

  - This last one is sent when the text field resigns being first responder.
  - This is usually where you extract the text from the field using the `text` property.

Keyboard

- Controlling the appearance of the keyboard

  Set the properties defined in the `UITextInputTraits` protocol (which `UITextField` implements).

```
@property UITextAutocapitalizationType autocapitalizationType; // words, sentences, etc.
@property UITextAutocorrectionType autocorrectionType; // UITextAutocorrectionTypeYES/NO
@property UIReturnKeyType returnKeyType;               // Go, Search, Google, Done, etc.
@property BOOL secureTextEntry;                        // for passwords, for example
@property UIKeyboardType keyboardType;                 // ASCII, URL, PhonePad, etc.
```

- The keyboard comes up over other views

  So you need to adjust your view positioning (especially to keep the text field itself visible).

  You do this by reacting to the `UIKeyboard{Will,Did}{Show,Hide}Notifications` sent by `UIWindow`.

```
[[NSNotificationCenter defaultCenter] addObserver:self
                                         selector:@selector(theKeyboardAppeared:)
                                             name:UIKeyboardDidShowNotification
                                           object:self.view.window];
```

  Your `theKeyboardAppeared:` method will get called with an `NSNotification` as the argument.

  Inside the `NSNotification` is a `userInfo` which will have details about the appearance.

UITextField

- Other UITextField properties

```
@property BOOL clearsOnBeginEditing;
@property BOOL adjustsFontSizeToFitWidth;
@property CGFloat minimumFontSize;  // always set this if you set adjustsFontSizeToFitWidth
@property NSString *placeholder;    // drawn in gray when text field is empty
@property UIImage *background/disabledBackground;
```

- Other UITextField functionality

UITextFields have a "left" and "right" overlays (similar to accessory views in MKAnnotationView).
You can control in detail the layout of the text field (border, left/right view, clear button).

- Other Keyboard functionality

Keyboards can have accessory views that appear above the keyboard (custom toolbar, etc.).

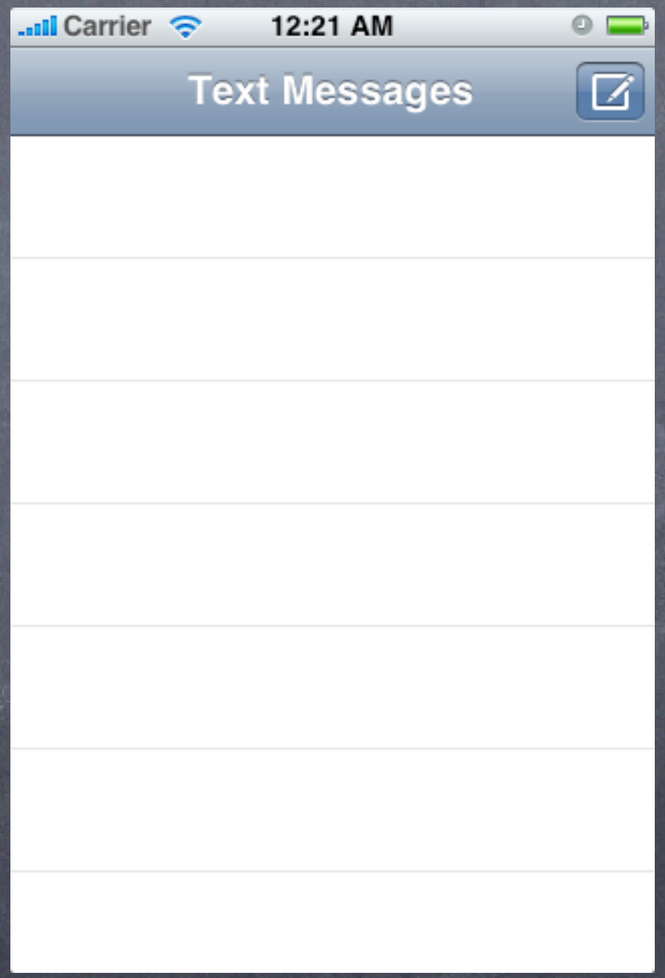
```
@property (retain) UIView *inputAccessoryView;  // UITextField method
```

UITextView

- `UITextView` is for multi-line, scrolling text
  Editable.
  Can set font and color of (the entire) text, of course.
  But does not support per-character formatting (use `UIWebView` and HTML for that).
- `UITextViewDelegate`
  Get notified when editing starts/ends.
  Control editing (prevent changes, etc.).
- It's a `UIScrollView`
  Has a text-specific scrolling method …
  `- (void)scrollRangeToVisible:(NSRange)rangeOfCharactersToScrollToVisible;`

Modal View Controllers

- Making a view controller's view appear temporarily
  And blocking all other "navigation" in the application until the user has dealt with this view.



Modal View Controllers

- One View Controller presents another View Controller modally

  Example. Putting up a modal view that asks the user to find an address.

```
- (void)lookupAddress  // this might be a target/action method in your view controller, e.g.
{
    AddressLookupViewController *alvc = [[AddressLookupViewController alloc] init];
    [self.presentModalViewController:alvc animated:YES];
    [alvc release];
}
```

  This method will fill the entire screen with `alvc`'s `view` and immediately return.

  The user will then not be able to do anything except interact with `alvc`'s `view`.

- So when does it all end?!

  It stays this way until someone sends this message to the view controller which put `alvc` up ...

```
- (void)dismissModalViewControllerAnimated:(BOOL)animated;
```

  You do NOT send this to `alvc`! You send it to the view controller that presented `alvc`
  (i.e. the one that implements the method `lookupAddress` above).

  Not only that, but `alvc` should NOT send it (since it should not have a "back" pointer to that VC).

Modal View Controllers

- So how is this conundrum resolved?

  Delegation.

```
- (void)lookupAddress
{
    AddressLookupViewController *alvc = [[AddressLookupViewController alloc] init];
    alvc.delegate = self;
    [self.presentModalViewController:alvc animated:YES];
    [alvc release];
}


// (One of) AddressLookupViewController's delegate method(s)
- (void)addressLookupViewController:(AddressLookupViewController *)sender
                    didSelectAddress:(Address *)anAddress
{
    // do something with the address the user selected (anAddress)
    [self.dismissModalViewControllerAnimated:YES];  // take sender off screen & release it
}
```

Modal View Controllers

- How is the modal view controller animated onto the screen?

  Depends on this property in the view controller that is being put up modally …

```
@property UIModalTransitionStyle modalTransitionStyle;

UIModalTransitionStyleCoverVertical  // slides up and down from bottom of screen
UIModalTransitionStyleFlipHorizontal // flips the current view controller view over for this one
UIModalTransitionStyleCrossDissolve  // old fades out as new fades in
UIModalTransitionStylePartialCurl    // only if presenter is full screen (and no more modal)
```

- What about iPad?

  Sometimes it might not look good for a presented view to take up the entire screen.

```
@property UIModalPresentationStyle modalPresentationStyle;  // in the modal view controller

UIModalPresentationFullScreen     // full screen anyway (always on iPhone/iPod Touch)
UIModalPresentationPageSheet      // full screen height, but portrait width even if landscape
UIModalPresentationFormSheet      // centered on the screen (all else dimmed)
UIModalPresentationCurrentContext // parent's context (e.g. in a popover)
```

UIView Animation

- Changes to certain `UIView` properties can be animated over time
  - View hierarchy (adding and removing subviews)
  - `hidden`
  - `frame`
  - `transform` (translation, rotation and scale)
  - `alpha` (opacity)
- Done with `UIView` class method and blocks
  - The class method takes animation parameters and an animation block as arguments.
  - The animation block contains the code that makes the changes to the `UIView`(s).
  - Most also have a "completion block" to be executed when the animation is done.
  - The changes inside the block are made immediately (even though they will appear "over time").
- Built on top of underlying Core Animation framework
  - We're not going to cover that in this course, but you should know it exists.

View Animation

- Animation class method in UIView

```
+ (void)animateWithDuration:(NSTimeInterval)duration
                      delay:(NSTimeInterval)delay
                    options:(UIViewAnimationOptions)options
                 animations:(void (^)(void))animations
                 completion:(void (^)(BOOL finished))completion;
```

- Example

```
[UIView animateWithDuration:3.0
                      delay:0.0
                    options:UIViewAnimationOptionBeginFromCurrentState
                 animations:^{ myView.alpha = 0.0; }
                 completion:^(BOOL fin) { if (fin) [myView removeFromSuperview]; }];
```

This would cause myView to "fade" out over 3 seconds (starting immediately).
Then it would remove myView from the view hierarchy (but only if the fade completed).
If, within the 3 seconds, someone animated the alpha to non-zero, the removal would not happen.

View Animation

- Another example

```
if (myView.alpha == 1.0) {
    [UIView animateWithDuration:3.0
                          delay:2.0
                        options:UIViewAnimationOptionBeginFromCurrentState
                     animations:^{ myView.alpha = 0.0; }
                     completion:nil];
    NSLog(@“alpha is %f.”, myView.alpha);
}
```

This would also cause `myView` to “fade” out over 3 seconds (starting in 2 seconds in this case).
The `NSLog()` would happen immediately (i.e. not after 3 or 5 seconds) and would print “alpha is 0.”
In other words, the animation block’s changes are executed immediately, but the animation itself (i.e. the visual appearance of the change to `alpha`) starts in 2 seconds and takes 3 seconds.

View Animation

- UIViewAnimationOptions

```
BeginFromCurrentState       // interrupt other, in-progress animations of these properties
AllowUserInteraction        // allow gestures to get process while animation is in progress
LayoutSubviews              // animate the relayout of subviews along with a parent's animation
Repeat                      // repeat indefinitely
Autoreverse                 // play animation forwards, then backwards
OverrideInheritedDuration   // if not set, use duration of any in-progress animation
OverrideInheritedCurve      // if not set, use curve (e.g. ease-in/out) of in-progress animation
AllowAnimatedContent        // if not set, just interpolate between current and end state image
CurveEaseInEaseOut          // slower at the beginning, normal throughout, then slow at end
CurveEaseIn                 // slower at the beginning, but then constant through the rest
CurveLinear                 // same speed throughout
TransitionFlipFromLeft/Right // only for hiding/removing views from the view hierarchy
TransitionCurlUp/Down       // only for hiding/removing views from the view hierarchy
```

View Animation

- Animating changes to the view hierarchy is slightly different

```
+ (void)transitionFromView:(UIView *)fromView
                    toView:(UIView *)toView
                  duration:(NSTimeInterval)duration
                   options:(UIViewAnimationOptions)options
                completion:(void (^)(BOOL finished))completion;
```

Include `UIViewAnimationOptionShowHideTransitionViews` if you want `hidden` property to be set. Otherwise it will actually remove `fromView` from the view hierarchy and add `toView`.

Or you can do the removing/adding/hiding yourself in a block with …

```
+ (void)transitionWithView:(UIView *)view
                  duration:(NSTimeInterval)duration
                   options:(UIViewAnimationOptions)options
                animations:(void (^)(void))animations
                completion:(void (^)(BOOL finished))completion;
```