Stanford CS193p

Developing Applications for iPhone 4, iPod Touch, & iPad

Fall 2010

Today: Grab Bag

- `UIView` Animation
  One more thing: animating view hierarchy changes
- `UISegmentedControl`
  Compact "radio button"-like control
- Core Motion
  Accelerometer and Gyro inputs
- Alerts
  `UIAlertView` & `UIActionSheet`
- `NSTimer`

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;
```

Core Motion

- API to access motion sensing hardware on your device
- Two primary inputs: Accelerometer and Gyro
  Currently only iPhone4 and newest iPod Touch have a gyro.
- Primary class used to get input is `CMMotionManager`
  Create with `alloc`/`init`, but only one instance allowed per application.
  It is a "global resource," so getting one via an application delegate method or class method is okay.
- Usage
  1. Check to see what hardware is available.
  2. Start the sampling going and poll the motion manager for the latest sample it has.

  ... or ...

  1. Check to see what hardware is available.
  2. Set the rate at which you want data to be reported from the hardware,
  3. Register a block (and a dispatch queue to run it on) each time a sample is taken.

Core Motion

- Checking availability of hardware sensors
  ```
  @property (readonly) BOOL {accelerometer,gyro,deviceMotion}Available;
  ```
  The "device motion" is a combination of accelerometer and gyro.
  We'll talk more about that in a couple of slides.
- Starting the hardware sensors collecting data
  You only need to do this if you are going to poll for data.
  ```
  - (void)start{Accelerometer,Gyro,DeviceMotion}Updates;
  ```
- Is the hardware currently collecting data?
  ```
  @property (readonly) BOOL {accelerometer,gyro,deviceMotion}Active;
  ```
- Stop the hardware collecting data
  It is a performance hit to be collecting data, so stop during times you don't need the data.
  ```
  - (void)stop{Accelerometer,Gyro,DeviceMotion}Updates;
  ```

Core Motion

- Actually polling the data

```
@property (readonly) CMAccelerometerData *accelerometerData;
```

`CMAccelerometerData` object provides `@property (readonly) CMAcceleration acceleration;`

```
typedef struct { double x; double y; double z; } CMAcceleration; // x, y, z in "g"
```

This raw data includes acceleration due to gravity.

```
@property (readonly) CMGyroData *gyroData;
```

`CMGyroData` object has one property `@property (readonly) CMRotationRate rotationRate;`

```
typedef struct { double x; double y; double z; } CMRotationRate; // x, y, z in radians/second
```

Sign of rotation rate follows right hand rule. This raw data will be biased.

```
@property (readonly) CMDeviceMotion *deviceMotion;
```

`CMDeviceMotion` is an intelligent combination of gyro and acceleration.
If you have both devices, you can report better information about each.

CMDeviceMotion

- Acceleration Data in CMDeviceMotion

```
@property (readonly) CMAcceleration gravity;
@property (readonly) CMAcceleration userAcceleration;  // gravity factored out using gyro
typedef struct { double x; double y; double z; } CMAcceleration; // x, y, z in "g"
```

- Rotation Data in CMDeviceMotion

```
@property CMRotationRate rotationRate;  // bias removed from raw data using accelerometer
typedef struct { double x; double y; double z; } CMRotationRate; // x, y, z in radians/second

@property CMAttitude *attitude;  // device's attitude (orientation) in 3D space

@interface CMAttitude : NSObject     // roll, pitch and yaw are in radians
@property (readonly) double roll;    // around longitudinal axis passing through top/bottom
@property (readonly) double pitch;   // around lateral axis passing through sides
@property (readonly) double yaw;     // around axis with origin at center of gravity and
                                     // perpendicular to screen directed down
   // other mathematical representations of the device's attitude also available
@end
```

Core Motion

- Registering a block to receive Accelerometer data

```
- (void)startAccelerometerUpdatesToQueue:(NSOperationQueue *)queue
                             withHandler:(CMAccelerometerHandler)handler;

typedef void (^CMAccelerationHandler)(CMAccelerometerData *data, NSError *error);
```

We haven't talked about `NSOperationQueue`, but think of it as an OO `dispatch_queue_t`.
There is also a corresponding OO version of blocks (sort of) called `NSOperation`.

- Registering a block to receive Gyro data

```
- (void)startGyroUpdatesToQueue:(NSOperationQueue *)queue
                    withHandler:(CMGyroHandler)handler;

typedef void (^CMGyroHandler)(CMGyroData *data, NSError *error);
```

- Registering a block to receive combined Gyro/Accelerometer data

```
- (void)startDeviceMotionUpdatesToQueue:(NSOperationQueue *)queue
                            withHandler:(CMDeviceMotionHandler)handler;

typedef void (^CMDeviceMotionHandler)(CMDeviceMotion *motion, NSError *error);
```

Core Motion

- Setting the rate at which your block gets executed

```
@property NSTimeInterval accelerometerUpdateInterval;
@property NSTimeInterval gyroUpdateInterval;
@property NSTimeInterval deviceMotionUpdateInterval;
```

- It is okay to add multiple handler blocks

  Even though you are only allowed one `CMMotionManager`.
  However, each of the blocks will receive the data at the same rate (as set above).
  (Multiple objects are allowed to poll at the same time as well, of course.)

UISegmentedControl

- Three different styles

```
@property UISegmentedControlStyle segmentedControlStyle;
UISegmentedControlStylePlain/Bordered/Bar
```

- Designated initalizer takes an NSArray of NSStrings or UIImages

```
NSArray *itemsArray = [NSArray arrayWithObjects:@“First”, @“Second”, nil];
UISegmentedControl *myControl = [[UISegmentedControl alloc] initWithItems:itemsArray];
```

- Or you can get/set items individually

```
- (void)setImage:(UIImage *)image forSegmentAtIndex:(int)index;
- (NSString *)titleForSegmentAtIndex:(int)index;
```

- Set or get which item is selected

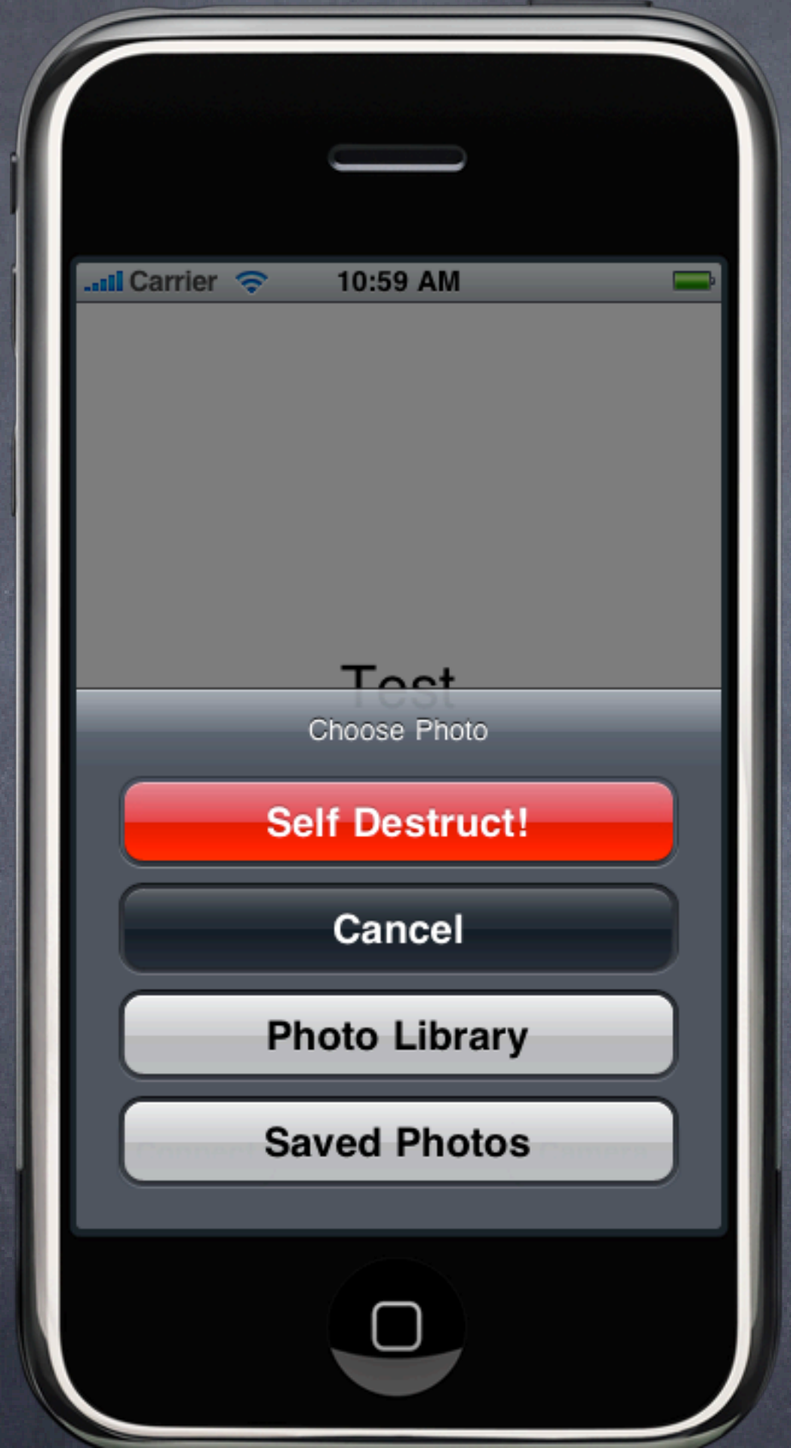
```
@property NSInteger selectedSegmentIndex;
```

Will be UISegmentedControlNoSegment if nothing is selected.

- It’s a UIControl, so use target/action to monitor changes

Alerts

- Two kinds of "pop up and ask the user something" mechanisms
  - Action Sheets
  - Alerts
- Action Sheets
  - Usually slides in from the bottom of the screen on iPhone/iPod Touch, and in a popover on iPad.
  - Can be displayed from a tab bar, toolbar, bar button item or from a rectangular area in a view.
  - Usually asks questions that have more than two answers.
  - Think of action sheets as presenting "branching decisions" to the user (i.e. what next?).
- Alerts
  - Pop up in the middle of the screen.
  - Usually ask questions with only two (or one) answers (e.g. OK/Cancel, Yes/No, etc.).
  - Very disruptive to your user-interface, so use carefully.
  - Often used for "asynchronous" problems ("connection reset" or "network fetch failed").



Stanford
CS193p
Fall 2010

UIActionSheet

- Initializer

```
-(id)initWithTitle:(NSString *)title
          delegate:(id <UIActionSheetDelegate>)delegate
 cancelButtonTitle:(NSString *)cancelButtonTitle
destructiveButtonTitle:(NSString *)destructiveButtonTitle
 otherButtonTitles:(NSString *)otherButtonTitles, ...;
```

- And you can add more buttons programmatically

```
- (void)addButtonWithTitle:(NSString *)buttonTitle;
```

- Displaying the Action Sheet

```
UIActionSheet *actionSheet = [[UIActionSheet alloc] initWithTitle:...];
[actionSheet showInView:(UIView *)];  // centers the view on iPad (so don't use this on iPad)
[actionSheet showFromRect:(CGRect) inView:(UIView *) animated:(BOOL)];  // good on iPad
[actionSheet showFromBarButtonItem:(UIBarButtonItem *) animated:(BOOL)];// good on iPad
```

UIActionSheet

- Finding out what the user has chosen via the delegate

```
- (void)actionSheet:(UIAlertView *)sender clickedButtonAtIndex:(NSInteger)index;
```

- Remember from initializer that Cancel/Destructive are special

```
@property NSInteger cancelButtonIndex;
@property NSInteger destructiveButtonIndex;
```

- Other indexes

```
@property NSInteger firstOtherButtonIndex;
@property NSInteger numberOfButtons;
- (NSString *)buttonTitleAtIndex:(NSInteger)index;
```

The "other button" indexes are in the order you specified them in initializer and/or added them

- You can programmatically dismiss the action sheet as well

```
- (void)dismissWithClickedButtonIndex:(NSInteger)index animated:(BOOL)animated;
```

It is generally recommended to call this on UIApplicationDidEnterBackgroundNotification. Remember also that you might be terminated while you are in the background, so be ready.

UIAlertView

- Initializer

```
-(id)initWithTitle:(NSString *)title
           message:(NSString *)message // different from UIActionSheet
          delegate:(id <UIActionSheetDelegate>)delegate
 cancelButtonTitle:(NSString *)cancelButtonTitle
 otherButtonTitles:(NSString *)otherButtonTitles, ...;
```

- And you can add more buttons programmatically

```
- (void)addButtonWithTitle:(NSString *)buttonTitle;
```

- Displaying the Action Sheet

```
UIAlertView *alertView = [[UIAlertView alloc] initWithTitle:...];
[alertView show];   // different from UIActionSheet, always appears in center of screen
```

- Rest of the mechanism is the same as UIActionSheet

NSTimer

- Scheduled invocation of a method in the main queue

```
NSTimer *timer = [NSTimer scheduledTimerWithTimeInterval:(NSTimeInterval)seconds
                                                  target:self
                                                selector:@selector(doSomething:)
                                                userInfo:(id)anyObject
                                                 repeats:(BOOL)yesOrNo];
```

- Not "real time" since it can run only each time around run loop
- Don't do anything too time consuming in the main thread
  You could dispatch another thread to do something time consuming, though.
- Check documentation for more
  For example, you can invalidate a repeating timer when you want it to stop.
  Or you can create a timer that will fire at a specific time (NSDate) in the future.

Delayed Perform

- Delayed invocation in the current queue

```
[self performSelector:@selector(aMethod:)
           withObject:(id)argumentToAMethod
           afterDelay:(NSTimeInterval)wait];
```

- Cancel previous requests

```
+ (void)cancelPreviousPerformRequestsWithTarget:(id)objectPerformWasSentTo
                                       selector:(SEL)aSelector
                                         object:(id)argument];
```

- Example usage: batching up database changes into a single save

  Instead of calling `save:` after each little change, we call the method `delayedSave:` below.
  This way we only `save:` (via a `doSave:` method) after no changes have occured for 1s.

```
- (void)delayedSave:(NSManagedObjectContext *)ctxt {
    [NSObject cancelPreviousPerformRequestsWithTarget:self selector:@selector(doSave:) object:ctxt];
    [self performSelector:@selector(doSave:) withObject:ctxt afterDelay:1.0];
}
```