Handling User Interactions

448460-1
Fall 2011
11/24/2011
Kyoung Shin Park
Multimedia Engineering
Dankook University

Overview

- Gesture Recognizers
  - How to get "input" into your UIView
- Touch Events & Multi-Touch
  - Touch Sequences
  - Touch and Event Objects
  - Touch Delivery
  - Single Touch/Multiple Touches
  - Multiple Views
  - Touch Routing
- Hardware features
  - Image Picker & Camera
  - Location
  - Accelerometer

Gesture Recognizers

UIGestureRecognizer

- We've seen how to draw in our UIView, how do we get **touches**?
  - We can get notified of the raw **touch events** (e.g., touch down, moved, up).
  - Or we can react to certain, **predefined "gestures"**
- 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
  - Adding a gesture recognizer to a UIView to ask it to recognize that gesture.
  - Providing the implementation of a method to "handle" that gesture when it happens.

UIGestureRecognizer

- Adding a gesture recognizer to a UIView from a Controller

```
// UIPanGestureRecognizer is a concrete subclass of UIGestureRecognizer
that recognizes “panning” (moving something around with your finger)
-(void) viewDidLoad {
    UIView *panView = …; // a view for recognizing “pan” gestures
    UIGestureRecognizer *panGR =
                [[UIPanGestureRecognizer alloc] initWithTarget:panView
                                                action:@selector(pan:)];
    [panView addGestureRecognizer:panGR];
    [panGR release];
}
```

UIGestureRecognizer

- How do we implement the target of a gesture recognizer?
  - Each concrete class provides some methods to help you do that
- E.g., **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 **Begin** (for continuous gestures like pan)
  - At any time, the state can change to **Failed**

UIGestureRecognizer

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

```
-(void)pan: (UIPanGestureRecognizer *)recognizer {
    if ((sender.state == UIGestureRecognizerStateChanged) ||
        (sender.state == UIGestureRecognizerStateEnded)) {
        // how much the gesture moved
        CGPoint translation = [recognizer translationInView:self];
        // move something in myself by translation.x and translation.y
        self.origin = CGPointMake(self.origin.x+translation.x,
                                  self.origin.y+tranlation.y);
        [recognizer setTranslation:CGPointZero inView:self];
    }
}
```

Other Concrete Gesture Classes

- UIPinchGestureRecognizer

```
@property CGFloat scale;
@property (readonly) CGFloat velocity;
```

- UIRotationGestureRecognizer

```
@property CGFloat rotation; // (radians)
@property (readonly) CGFloat velocity; // (radians/second)
```

- UISwipeGestureRecognizer
  - Set up to find certain swipe types, then look for Recognized state

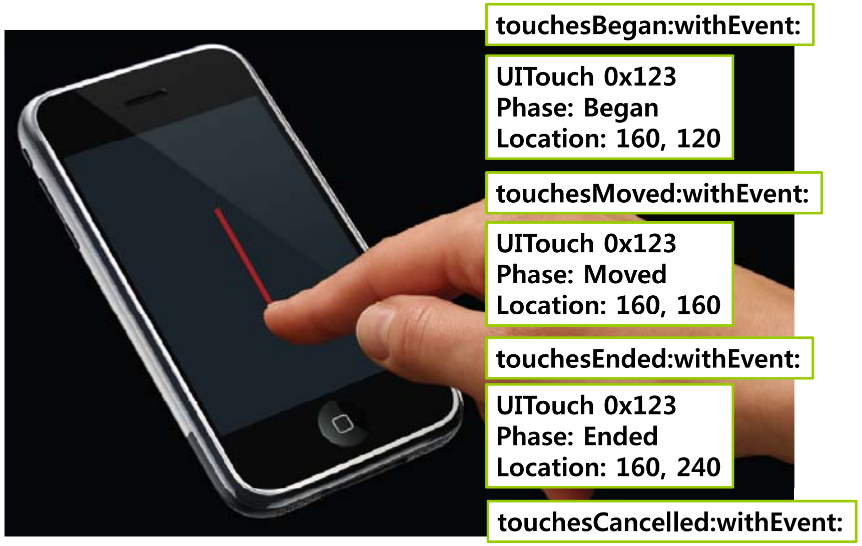
```
@property UISwipeGestureRecognizerDirection direction;
@property NSUInteger numbeOfTouchesRequired;
```

- UITapGestureRecognizer
  - Set up, then look for Recognized state

```
@property NSUInteger numberOfTapsRequired;
@property NSUInteger numberOfTouchesRequired;
```

Touch Events & Multi-Touch

Single Touch Sequence



touchesBegan:withEvent:

UITouch 0x123
Phase: Began
Location: 160, 120

touchesMoved:withEvent:

UITouch 0x123
Phase: Moved
Location: 160, 160

touchesEnded:withEvent:

UITouch 0x123
Phase: Ended
Location: 160, 240

touchesCancelled:withEvent:

UITouch

- Represents a single finger

```
@property (nonatomic, readonly) NSTimeInterval timestamp;
@property (nonatomic, readonly) UITouchPhase phase;
@property (nonatomic, readonly) NSUInteger tapCount;

@property (nonatomic, readonly, retain) UIWindow *window;
@property (nonatomic, readonly) UIView *view;

-(CGPoint)locationInView: (UIView *)view;
-(CGPoint)previousLocationInView: (UIView *)view;
```

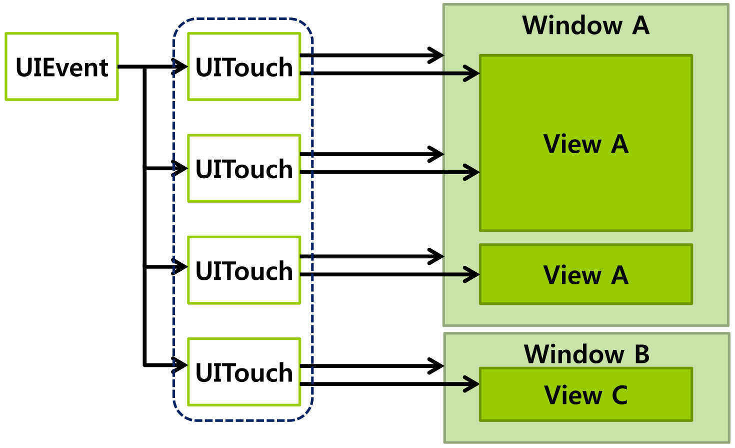
UIEvent

- A container for one or more touches

```
@property (nonatomic, readonly) NSTimeInterval timestamp;
-(NSSet *)allTouches;
-(NSSet *)touchesForWindow: (UIWindow *)window;
-(NSSet *)touchesForView: (UIView *)view;
```

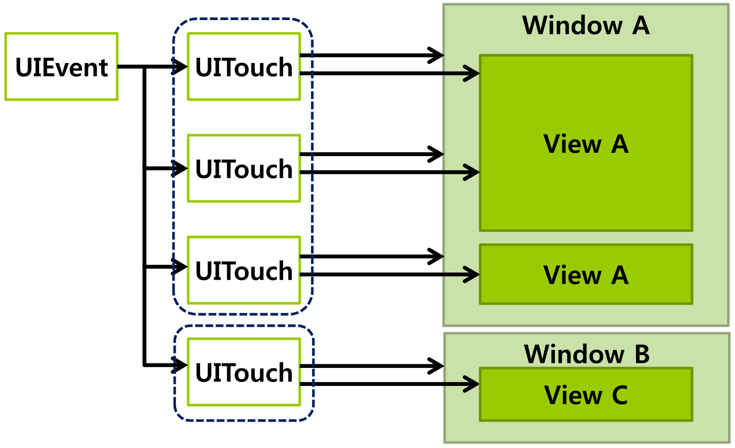
UIEvent

-(NSSet *)allTouches;


UIEvent
UITouch
UITouch
UITouch
UITouch
Window A
View A
View A
Window B
View C

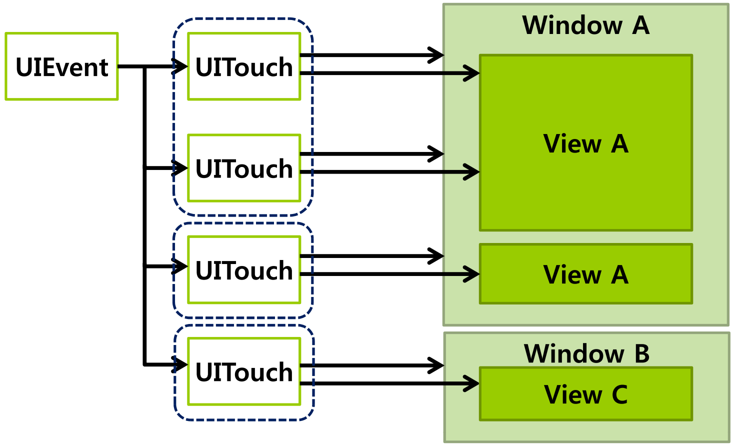
UIEvent

-(NSSet *)touchesForWindow: (UIWindow *)window;


UIEvent
UITouch
UITouch
UITouch
UITouch
Window A
View A
View A
Window B
View C

UIEvent

-(NSSet *)touchesForView: (UIView *)view;

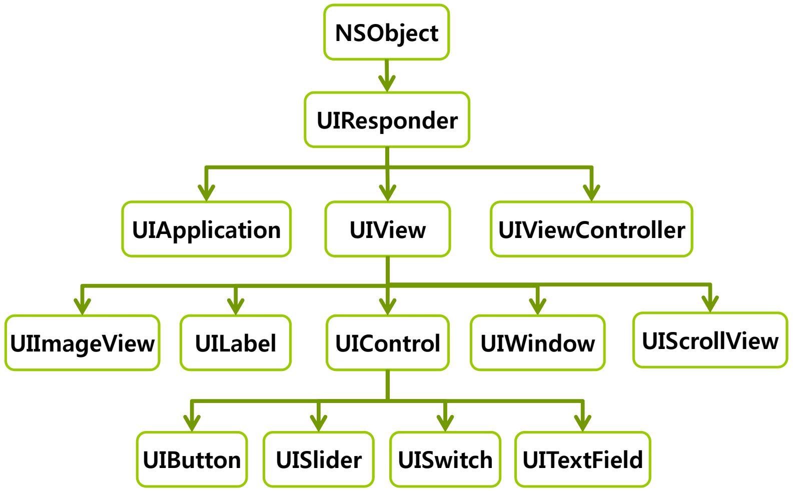

UIEvent
UITouch
UITouch
UITouch
UITouch
Window A
View A
View A
Window B
View C

UIResponder

- Receiving touches

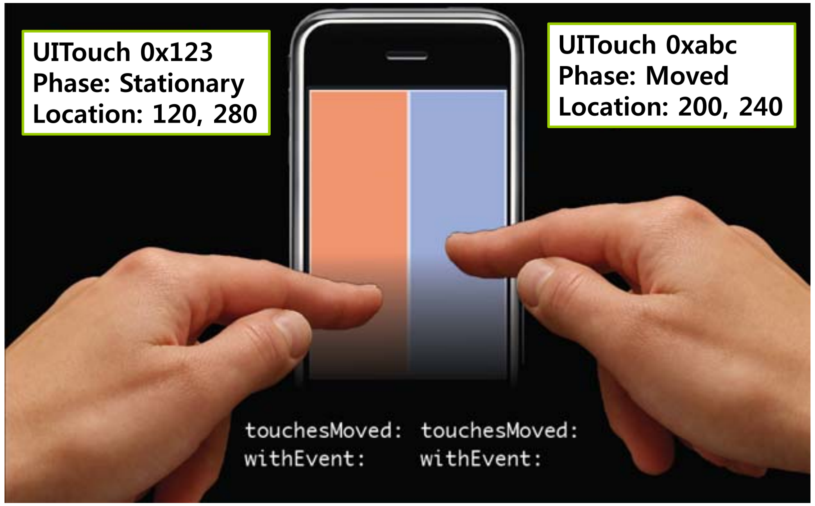
-(void)touchesBegan: (NSSet *)touches withEvent: (UIEvent *)event;
-(void)touchesMoved: (NSSet *)touches withEvent: (UIEvent *)event;
-(void)touchesEnded: (NSSet *)touches withEvent: (UIEvent *)event;
-(void)touchesCancelled: (NSSet *)touches withEvent: (UIEvent *) event;

UIResponder

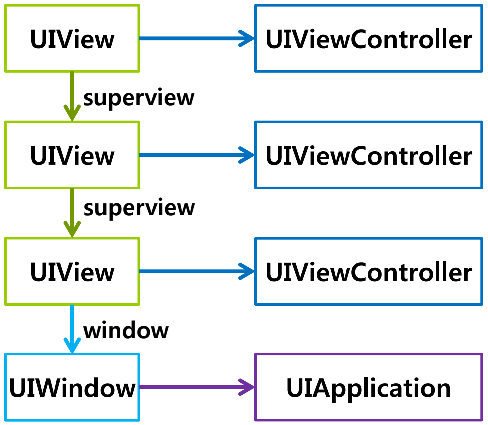


Multiple Touches

BOOL multipleTouchEnabled; // UIView property

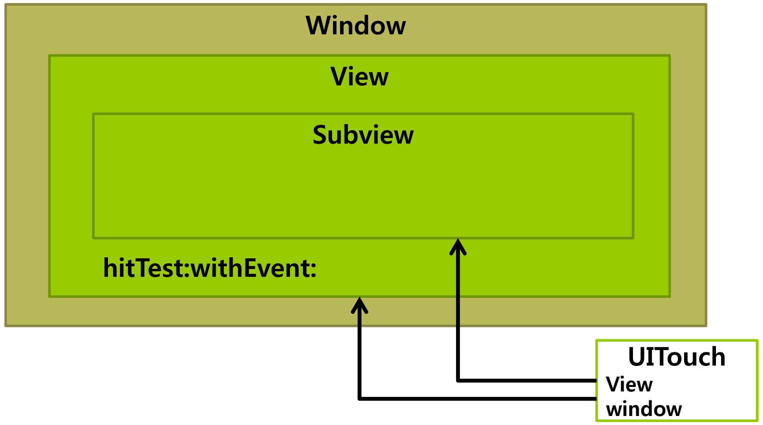


Responder Chain



Hit Testing

userInteractionEnabled;
Hidden/alpha
pointInside: withEvent:



UIControlEvents

```
UIControlEventTouchDown
UIControlEventTouchDownRepeat
UIControlEventTouchDragInside
UIControlEventTouchDragOutside
UIControlEventTouchDragEnter
UIControlEventTouchDragExit
UIControlEventTouchUpInside
UIControlEventTouchUpOutside
UIControlEventTouchCancel
```

Associating actions with UIControlEvents

- Add target and action for UIControlEvent

```
-(void)addTarget: (id)target
        action: (SEL)action
        forControlEvents: (UIControlEvents)controlEvents;
```

- Action signatures

```
-(void)performAction;
-(void)performAction: (id)sender;
-(void)performAction: (id)sender withEvent: (UIEvent *)event;
```

Associating actions with UIControlEvents

- UIControl touch tracking

```
-(BOOL)beginTrackingWithTouch: (UITouch *)touch
         withEvent: (UIEvent *) event;
-(BOOL)continueTrackingWithTouch: (UITouch *)touch
         withEvent: (UIEvent *) event;
-(BOOL)endTrackingWithTouch: (UITouch *)touch
         withEvent: (UIEvent *) event;
-(BOOL)cancelTrackingWithEvent: (UIEvent *) event;
```

Handling Touch Events

- The view returned by

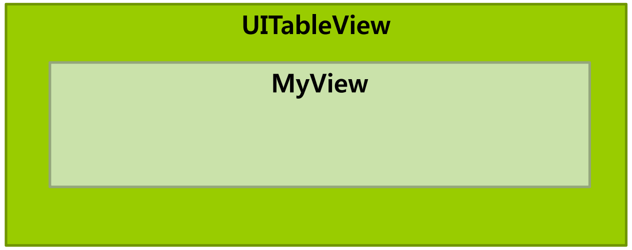
```
-(UIView *)hitTest: (CGPoint)point
         withEvent: (UIEvent *) event;
```

must handle all of the touch processing methods

```
-(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;
```

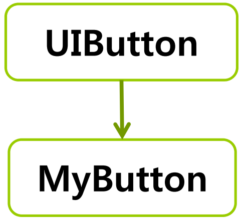
Subclassing UIView

- Subclasses of UIView **must implement all touch processing methods** and **must not call super.**



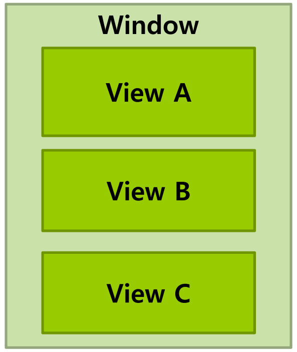
Subclassing Other UIKit Classes

- Subclasses of any other UIKit class can implement any or all touch processing methods but **must call super.**



Touch Forwaring

- If you need to conditionally send touches to various views, all of the views involved need to be your own custom subclasses of UIView.



Device Hardware

Device Hardware

Camera **Location** **Accelerometer & Gyro**

Image Picker Interface

- Image Picker Interface
  - Camera capture, Save photos, The photo library
- **UIImagePickerController** class
  - Use as-is (no subclassing)
  - Handles all user and device interactions
  - UIViewController Subclass
- **UIImagePickerControllerDelegate** protocol
  - Implemented by your delegate object

Displaying the Image Picker

- Steps for using
  - Check the source availability
  - Assign a delegate object
  - Present the controller modally
- Called from a view controller object

```
if ([UIImagePickerController isSourceTypeAvailable:
                    UIImagePickerControllerSourceTypeCamera]) {
    UIImagePickerController *picker =
        [[UIImagePickerController alloc] init];
    picker.sourceType =
UIImagePickerControllerSourceTypeCamera;
    picket.delegate = self;
    [self presentModalViewController:picker animated:YES];
}
```

Defining Your Delegate Object

- The UIImagePickerControllerDelegate protocol
  - The accept case:

```
-(void) imagePickerController: (UIImagePickerController *)picker
        didFinishPickingImage: (UIImage *)image
        editingInfo: (NSDictionary *)editingInfo {
    // save or use the image here

    // dismiss the image picker
    [self dismissModalViewControllerAnimated:YES];
    [picker release];
}
```

Defining Your Delegate Object

- The UIImagePickerControllerDelegate protocol
  - The cancel case

```
-(void) imagePickerControllerDidCancel:
          (UIImagePickerController *)picker {
    // dismiss the image picker
    [self dismissModalViewControllerAnimated:YES];
    [picker release];
}
```

Manipulating the Returned Image

- Allowing users to edit returned images
  - If **allowsImageEditing** property is YES:
    - User allowed to crop the returned image
    - Image metadata returned in editingInfo
  - The **editingInfo** dictionary
    - Original image in **UIImagePickerControllerOriginalImage** key
    - Crop rectangle in **UIImagePickerControllerCropRect** key

```
-(void) imagePickerController: (UIImagePickerController *)picker
        didFinishPickingImage: (UIImage *)image
        editingInfo: (NSDictionary *)editingInfo {
    // save or use the image here
    // dismiss the image picker
    [self dismissModalViewConrollerAnimated:YES];
    [picker release];
}
```

Manipulating the Returned Image

- Writing image/video to the photos album
  - **UIImageWriteToSavedPhotosAlbum**
    - Photos can be downloaded to iPhoto by user
    - Optional completion callback
  - **UIVideoAtPathIsCompatibleWithSavedPhotosAlbum**
  - **UISaveVideoAtPathToSavedPhotosAlbum**
    - Videos can be downloaded to iPhoto by user
    - Optional completion callback

Core Location

Core Location Framework

- The core classes and protocols
- Classes
  - CLLocationManager
  - CLLocation
- Protocol
  - CLLocationManagerDelegate

CLLocationManagerDelegate Protocol

- CLLocationManagerDelegate protocol

```
// 2 optional methods
-(void) locationManager: (CLLocationManager *)manager
        didUpdateToLocation: (CLLocation *)newLocation
        fromLocation: (CLLocation *)oldLocation;
-(void) locationManager: (CLLocationManager *)manager
        didFailWithError: (NSError *)error;
```

- Called asynchronously on main thread
- Issues movement-based updates

Getting a Location

- Starting the location service

```
CLLocationManager *manager = [[CLLocationManager alloc] init];
manager.delegate = self;
[manager startUpdatingLocation];
```

- Using the event data

```
-(void)locationManager: (CLLocationManager *)manager
        didUpdateToLocation: (CLLocation *)newLocation
        fromLocation: (CLLocation *)oldLocation {
   NSTimeInterval howRecent =
        [newLocation.timestamp timeIntervalSinceNow];
   if (howRecent < -10) return;
   if (newLocation.horizontalAccuracy > 100) return;
   double lat = newLocation.coordinate.latitude;
   double lon = newLocation.coordinate.longitude;
}
```

Getting a Heading

- Using the event data

```
-(void)locationManager: (CLLocationManager *)manager
        didUpdateHeading: (CLHeading *)newHeading {
   // use the coordinate data
   CLLocationDirection heading = newHeading.truHeading;
}
```

Desired Accuracy

- Choosing an appropriate accuracy level

```
CLLocationManager *manager = [[CLLocationManager alloc] init];
manager.desiredAccuracy = kCLLocationAccuracyBest;
```

  - Choosing an appropriate accuracy level
    - Higher accuracy impacts power consumption
    - Lower accuracy is "good enough" in most cases
  - Can change accuracy setting later if needed
  - Actual accuracy reported in **CLLocation** object

Distance Filter

- Choosing an appropriate update threshold

```
CLLocationManager *manager = [[CLLocationManager alloc] init];
manager.distanceFilter = 3000;
```

  - New events delivered when threshold exceeded

Stopping the Service

- Stopping the Service

```
CLLocationManager *manager = [[CLLocationManager alloc] init];
[manager startUpdatingLocation];
...
[manager stopUpdatingLocation];
```

  - Restart the service later as needed

Responding to Errors

- User may deny use of the location service
  - Results in a **kCLErrorDenied** error
  - Protects user privacy
  - Occurs on a per-application basis
- Location may be unavailable
  - Results in a **kCLErrorLocationUnknown** error
  - Likely just temporary
  - Scan continues in background

Core Motion

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.

Core Motion

- 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.
- 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;
```

  - This raw data includes acceleration due to gravity.

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

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

```
typedef struct { double x; double y; double z; } CMRotationRate;
```

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

```
@property (readonly) CMDeviceMoion *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;
typedef struct { double x; double y; double z; } CMAcceleration;
```

- Rotation Data in CMDeviceMotion

```
@property CMRotationRate rotationRate;
typedef struct { double x; double y; double z; } CMRotationRate;
@property CMAttitude *attitude; //device orientation in 3D space

@interface CMAttitude : NSObject
@property (readonly) double roll;
@property (readonly) double pitch;
@property (readonly) double yaw;
@end
```

Core Motion

- Registering a block to receive Accelerometer data

```
-(void) startAccelerometerUpdatesToQueue:
          (NSOperationQueue *)queue
          withHandler: (CMAccelerometerHandler)handler;
typedef void (^CMAccelerationHandler) (CMAccelerometerData *data, NSError *error);
```

- Registering a block to receive Gyro data

```
-(void) startGyroUpdatesToQueue:
          (NSOperationQueue *)queue
          withHandler: (CMGyroHandler)handler;
typedef void (^CMGyroHandler) (CMGyroData *data, NSError *error);
```

Core Motion

- Registering a block to receive combined Gyro / Accelerometer data

```
-(void) startDeviceMotionUpdatesToQueue:
          (NSOperationQueue *)queue
          withHandler: (CMDeviceHandler)handler;
typedef void (^CMDeviceMotionHandler) (CMDeviceMotion *data, NSError *error);
```

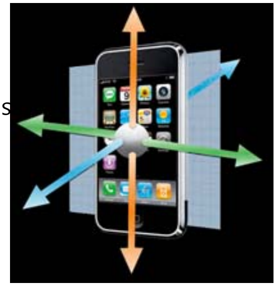
- Setting the rate at which your block gets executed

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

- It’s 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).

Orientation-related Changes

- Getting the physical orientation
- UIDevice class
  - Start notifications
    - beginGeneratingDeviceOrientationNotifications
  - Get orientation
    - UIDeviceOrientationDidChangeNotification delivered to registered observers
    - Orientation property
  - Stop notifications
    - endGeneratingDeviceOrientationNotifications



Orientation-related Changes

- Getting the interface orientation
- UIApplication class
  - statusBarOrientation property
  - Defines interface orientation, not device orientation
- UIViewController class
  - interfaceOrientation property

```
-(BOOL) shouldAutorotateToInterfaceOrientation:
        (UIInterfaceOrientation) interfaceOrientation
```

Shake

- UIEvent type

```
@property (readonly) UIEventType type;
@property (readonly) UIEventSubType subtype;
UIEventTypeMotion
UIEventSubTypeMotionShake
```

Accelerometer Interface

- Getting the raw accelerometer data
  - UIAccelerometer & UIAcceleration classes
  - UIAccelerometerDelegate protocol
- Starting the event delivery

```
-(void) enableAccelerometerEvent {
  UIAccelerometer *acc = [UIAccelerometer sharedAccelerometer];
  acc.updateInterval = 1/50; // 50 Hz
  acc.delegate = self;
}
```

Accelerometer Interface

- Processing the accelerometer data
  - Only one delegate per application
  - Delivered asynchronously to main thread

```
-(void) accelerometer: (UIAccelerometer *)accelerometer
          didAccelerate: (UIAcceleration *)acceleration {
   // get the event data
   UIAccelerationValue x, y, z;
   x = acceleration.x;
   y = acceleration.y;
   z = acceleration.z;
   // process the data
}
```

Accelerometer Interface

- Stopping the event delivery

```
-(void) disableAccelerometerEvents {
   UIAccelerometer * acc = [UIAccelerometer sharedAccelerometer]
   acc.delegate = nil;
}
```

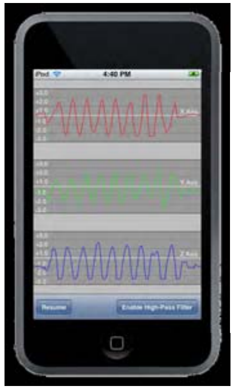
Filtering Accelerometer Data

- Use filters to isolate data components
  - **Low-pass filter**
    - Isolates constant acceleration
    - **Used to find the device orientation**
  - **High-pass filter**
    - Shows instantaneous movement only
    - **Used to identify user-initiated movement**

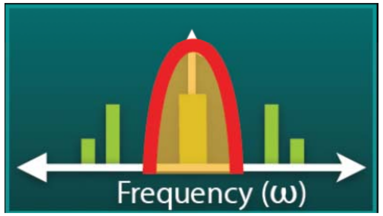
***f(t) => F(ω) :* Fourier Transform**

Filtering Accelerometer Data

- Applying a low-pass filter



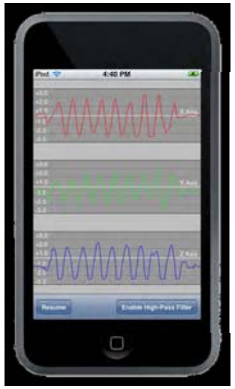
```
#define FILTERFACTOR 0.1
lowPassValue = (newAcceleration * FILTERFACTOR) +
        (previousLowPassValue * (1.0 – FILTERFACTOR));
previousLowPassValue = lowPassValue;
```



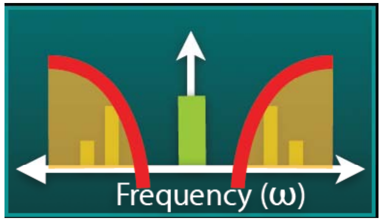
f(t) => F(ω)

Filtering Accelerometer Data

- Applying a high-pass filter



```
#define FILTERFACTOR 0.1
lowPassValue = (newAcceleration * FILTERFACTOR) +
        (previousLowPassValue * (1.0 – FILTERFACTOR));
previousLowPassValue = lowPassValue;
highPassValue = newAcceleration – lowPassValue;
```



f(t) => *F(ω)*

Filtering Accelerometer Data

- Bubble level sample **(low-pass filter)**

```
-(void) accelerometer: (UIAccelerometer *)accelerometer
        didAccelerate: (UIAcceleration *)acceleration {
    accelerationX = acceleration.x * kFilteringFactor +
                    accelerationX * (1.0 – kFilteringFactor);
    accelerationY = acceleration.y * kFilteringFactor +
                    accelerationY * (1.0 – kFilteringFactor);
    currentRawReading = atan2(accelerationY, accelerationX);
    float calibratedAngle = [self calibratedAngleFromAngle:
                                currentRawReading];
    [levelView updateToInclinationInRadians: calibratedAgle];
}
```

Using the Accelerometers Effectively

- Use UIViewControllers
- Use filters to isolate raw data components
- Disable accelerometer updates when not needed
  - Set your accelerometer delegate to nil

References

- Lecture 14 & 15 Slide from iPhone Application Development (Winter 2010) @Stanford University