Building an iPhone Application

448460-1
Fall 2011
10/06/2011
Kyoung Shin Park
Multimedia Engineering
Dankook University

Overview

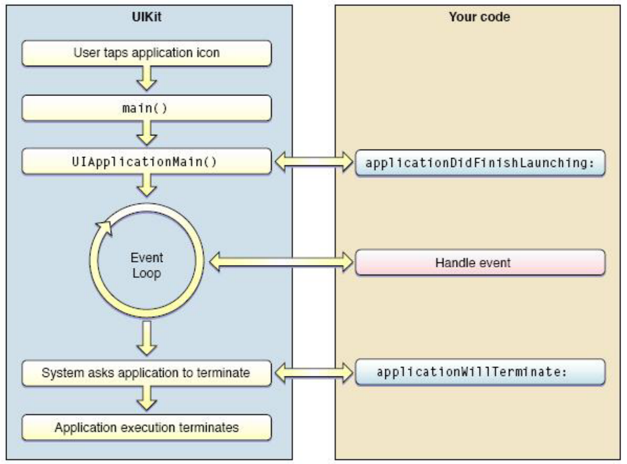
- Building an Application
- Model-View-Controller Design
- Interface Builder and Nib Files
- Controls and Target-Action
- Views & Custom Views
- Drawing with core Graphics
- Text & Images

Building an Application

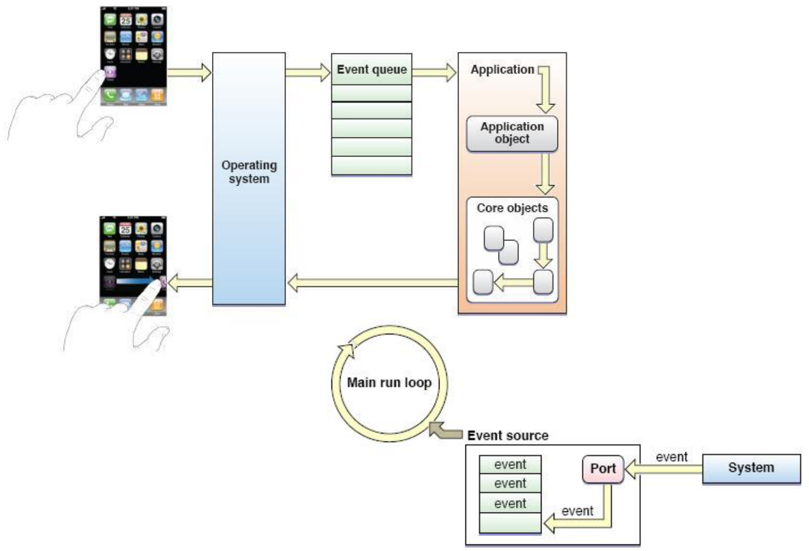
Anatomy of an Application

- Compiled code
  - Your code
  - Framework
- Nib files
  - UI elements and other objects
  - Details about object relationships
- Resources (images, sounds, strings, etc)
- Info.plist file (application configuration)

Application Lifecycle


UIKit
Your code
User taps application icon
main()
UIApplicationMain()
applicationDidFinishLaunching:
Event Loop
Handle event
System asks application to terminate
applicationWillTerminate:
Application execution terminates

Event-Handling Cycle



UIKit Framework

- UIKit provides standard interface elements
  - button, label, slider, tableview, etc
- Every application has a single instance of UIApplication
  - Singleton design pattern

    @interface UIApplication
    +(UIApplication *) sharedApplication
    @end
  - Orchestrates the lifecycle of an application
  - Dispatches events
  - Manages status bar, application icon badge
  - Rarely subclassed; **Uses delegation instead**

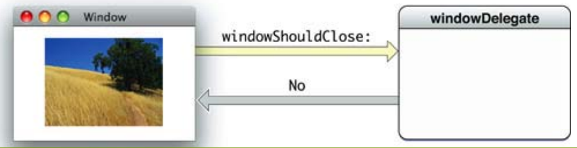
Main.m

```
#import <UIKit/UIKit.h>
int main(int argc, char *argv[])
{
    // create an autorelease pool
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];

    // call UIApplicationMain
    int retVal = UIApplicationMain(argc, argv, nil, nil);

    // release autorelease pool
    [pool release];

    return retVal;
}
```

UIKit uses a default UIApplication class.
UIKit loads a main Nib file which will load UIApplicationDelegate.

Delegation

- Delegate allows one object to act on behalf of another object
- Control passed to delegate objects to perform application specific behavior
- Avoids need to subclass complex objects
- Many UIKit classes use delegates
  - UIApplication
  - UITableView
  - UITextField



The delegate is automatically registered as an observer of notifications posted by the delegating object. The delegate need only implement a notification method declared by the framework class to receive a particular notification message. This window object posts an **NSWindowWillCloseNotification** to observers, but sends a **windowShouldClose:** message to its delegate.

ApplicationDelegate

- Xcode project templates have one set up by default
- Object you provide that participates in application lifecycle
- Many methods in the UIApplication object's delegate protocol

```
-(void) applicationDidFinishLaunching: (UIApplication *) application;
-(void) applicationWillTerminate: (UIApplication *) application;
-(void) applicationWillResignActive: (UIApplication *) application;
-(BOOL) application: (UIApplication *) application handleOpenURL:
(NSURL *) url;
-(void) applicationDidReceiveMemoryWarning: (UIApplication *)
application;
```

Application Delegate

```
@interface YourAppDelegate : NSObject<UIApplicationDelegate>{
   UIWindow *window;
   YourAppViewController * viewController;
}
@property (nonatomic, retain) IBOutlet UIWindow * window;
@property (nonatomic, retain) IBOutlet YourAppViewController *
   viewController;
@end
-(BOOL)application: (UIApplication *)application
         didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{    // override point for customization after application launch
     // add the view controller's view to the window and display
     [window addSubview:viewController.view];
     [window makeKeyAndVisible];
     return YES;
}
```

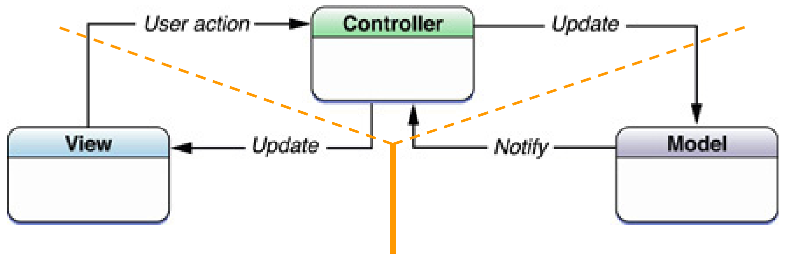
Info.plist file

- Property List (often XML), describing your application
  - Icon appearance
  - Status bar style (default, black, hidden)
  - Orientation
  - Uses Wifi networking
  - System Requirements
- Can edit most properties in Xcode

Model View Controller

Model View Controller

- The Model-View-Controller (MVC) design pattern assigns objects in an application one of three roles: model, view, or controller.



Model = **What** you application is (but **not how** it is displayed)
Controller = **How** your Model is presented to the user (UI logic)
View = Your Controller's minions

Model

- Manages the application data and state
- Not concerned with UI or presentation
- Often persists somewhere
- Same model should be reusable, unchanged in different interfaces

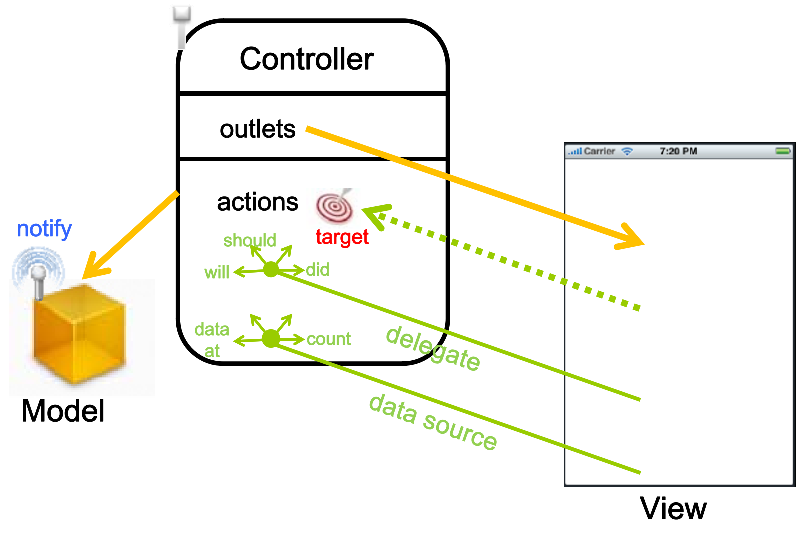
View

- Present the Model to the user in an appropriate interface
- Allows user to manipulate data
- Does not store any data (except to cache state)
- Easily reusable & configurable to display different data

Controller

- Intermediary between Model & View
- Updates the view when the model changes
- Updates the model when the user manipulates the view
- Typically where the application logic lives

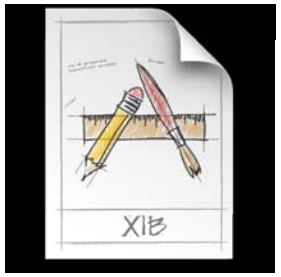
Model View Controller



Interface Builder and Nib

Nib Files

- Helps you design the View in MVC
  - Layout user interface elements
  - Add controller objects
  - Connect the controller and UI



http://developer.apple.com/library/ios/#documentation/iPhone/Conceptual
/iPhone101/Articles/04_InspectingNib.html#//apple_ref/doc/uid/TP40007514-CH6-SW1

Nib Loading

- At runtime, objects are unarchived
  - Values/settings in Interface Builder are restored
  - Ensures all outlets and actions are connected
  - Order of unarchiving is not defined
- If loading the nib automatically creates objects and order is undefined, how do I customize?
  - -awakeFromNib

-awakeFromNib

- Control point to implement any additional logic after nib loading
- Default empty implementation on NSObject
- You often implement it in your controller class
  - E.g. to restore previously saved application state
- Guaranteed everything has been unarchived from nib, and all connections are made before –awakeFromNib is called

```
- (void) awakeFromNib {
    // do customization here
}
```

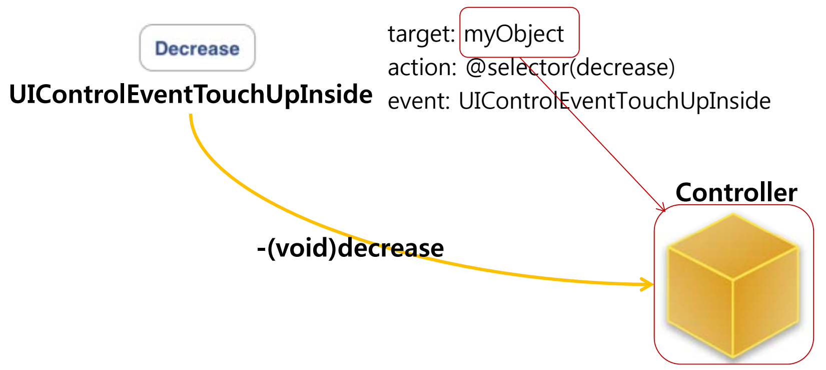
Controls and Target/Action

Controls – Events

- View objects that allows users to initiate some type of action
- Respond to variety of events
  - Touch events
    - touchDown
    - touchDragged (entered, exited, drag inside, drag outside)
    - touchUp
  - Value changed
  - Editing events
    - editing began
    - editing changed
    - editing ended

Controls – Target/Action

- When event occurs, actions is invoked on target object


Decrease
UIControlEventTouchUpInside
target: myObject
action: @selector(decrease)
event: UIControlEventTouchUpInside
Controller
-(void)decrease

Action Methods

- 3 different flavors of action method selector types
  - -(void) actionMethod;
  - -(void) actionMethod: (id) sender;
  - -(void) actionMethod: (id) sender withEvent: (UIEvent *) event;
- UIEvent contains details about the event that took place

Action Methods

- Simple no-argument selector

```
-(void) increase {
   // bump the number of sides of the polygon up
   polygon.numberOfSides += 1;
}
```

- Single argument selector –control is 'sender'

```
-(void) adjustNumberOfSides:(id) sender { // if control is a slider
   polygon.numberOfSides = [sender value];
}
```

- Two arguments in selector (sender & event)

```
-(void) adjustNumberOfSides:(id) sender withEvent:(UIEvent *) event {
   // could inspect event object if you needed to
}
```

Multiple Target-Actions

- Contols can trigger multiple actions on different targets in response to the same event
- Different than Cocoa on the desktop where only one target actions is supported
- Different events can be setup in Interface Builder

Manual Target-Action

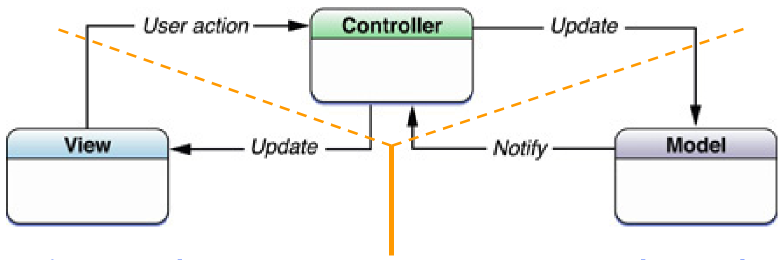
- Same information Interface Builder would use
- API and UIControlEvents found in UIControl.h
- UIControlEvents is a bitmask

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

-(void) removeTarget: (id)target action: (SEL) action
        forControlEvents: (UIControlEvents) controlEvents;
@end
```

Demo

HelloPolygon

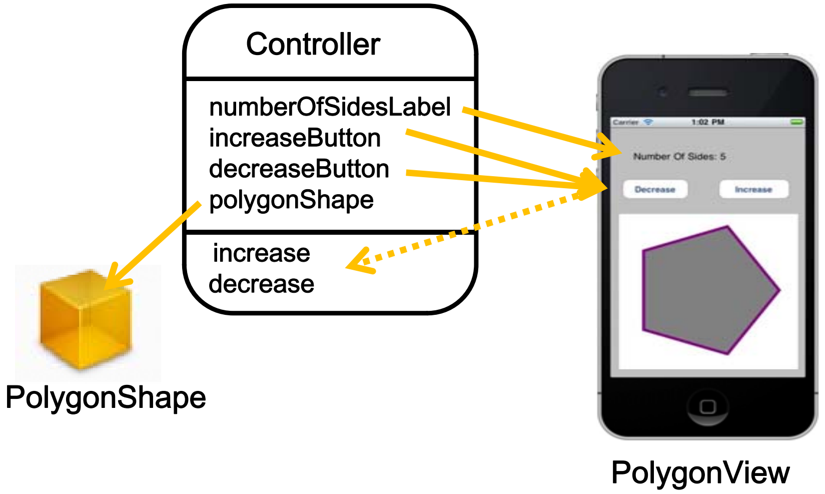
Controller



UIKit controls
PolygonView

PolygonShape

Model View Controller



Views

View Fundamentals

- A **view** (i.e., **UIView subclass**) represents a rectangular area on screen
- Draws content and handles events in that rectangle
- Subclass of **UIResponder** (event handling class)
- Views arranged hierarchically
  - Every view has **one superview – (UIView *)superview**
  - Every view has **zero or more subviews – (NSArray *)subviews**
  - Subview order (in that array) matters: those later in the array are on top of those earlier

View Hierarchy - UIWindow

- Views live inside of a window
- **UIWindow** is actually just a view
  - Adds some additional functionality specific to top level view
- **One UIWindow** for an iPhone application
  - Contains the entire view hierarchy
  - Set up by default in Xcode template project

View Hierarchy - Manipulation

- Add/remove views in Interface Builder or using UIView methods

```
-(void) addSubview: (UIView *)view;
-(void) removeFromSuperview;
```

- Manipulate the view hierarchy manually

```
-(void) insertSubview: (UIView *)view atIndex: (int)index;
-(void) insertSubview: (UIView *)view belowSubview: (UIView *)view;
-(void) insertSubview: (UIView *)view aboveSubview: (UIView *)view;
-(void) exchangeSubviewAtIndex: (int)index
            withSubviewAtIndex: (int)otherIndex;
```

View Hierarchy - Ownership

- **A superview retains its subviews**
  - Once you put a view into the view hierarchy, you can release your ownership if you want
- Be careful when you remove a view from the hierarchy
  - If you want to keep using a view, retain ownership before you send removeFromSuperview
  - Removing a view from the hierarchy immerdiately causes a release on it (not autorelease)
  - If there are no other owners, it will be immediately deallocated (and its subviews released)
  - So, retain subview before removing if you want to reuse it

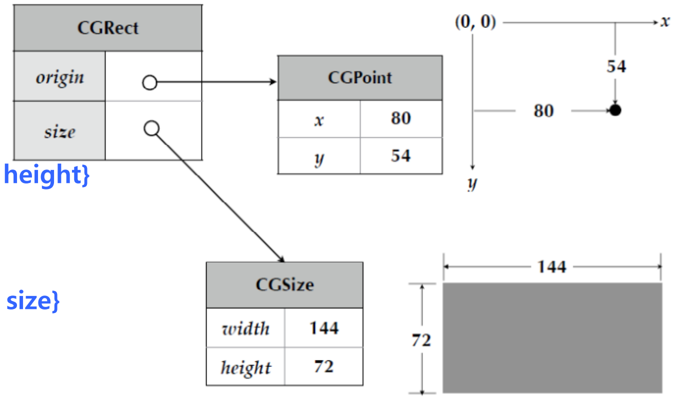
View Transparency

- What happens when views overlap?
  - **Subviews** list order determines who's in front
  - Lower ones can "show through" transparent views sitting on top of them though
- When you are drawing, you can draw with transparency
  - **By default, drawing is full opaque!**
- Also, you can hide a view completely by setting **hidden** property

```
@property BOOL hidden;
myView.hidden = YES; // view will not be on screen and
                     // will not handle events
```

View-related Structures

- CGPoint
  - {x, y}
- CGSize
  - {width, height}
- CGRect
  - {origin, size}



View-related Structures

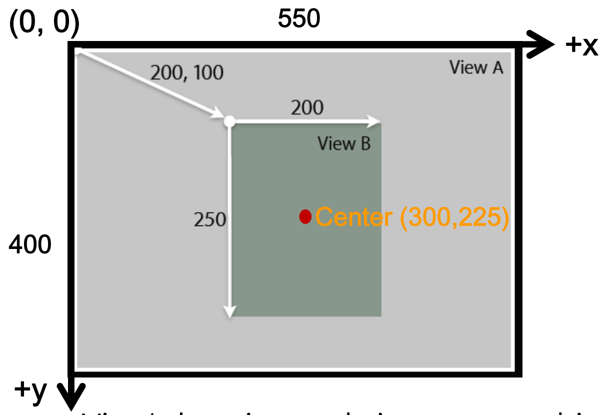
Function	Example

CGPointMake(x, y)	CGPoint point = CGPointMake(10.0, 20.0);
	point.x = 30.0;
	point.y += 30.0;

CGSizeMake(width, height)	CGSize size = CGSizeMake(40.0, 30.0);
	size.width = 300.0;
	size.height += 20.0;

CGRectMake(x, y, width, height)	CGRect rect = CGRectMake(100.0, 200.0, 40.0, 30.0);
	rect.origin.x = 0.0;
	rect.size.width = 50.0;

UIView Coordinate System



View A Frame:
- Origin: (0, 0)
- Size: 550 x 400

View A Bounds:
- Origin: (0, 0)
- Size: 550 x 400

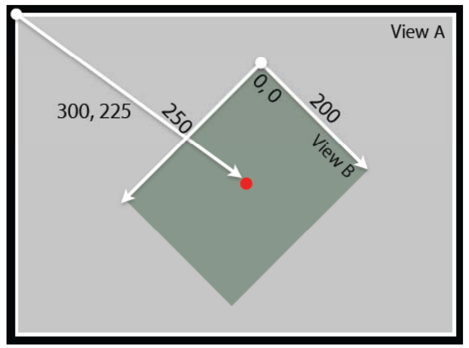
View B Frame:
- Origin: (200, 100)
- Size: 200 x 250

View B Bounds:
- Origin: (0, 0)
- Size: 200 x 250

- View's location and size expressed in two ways:
  - **Frame** is in superview's coordinate system
  - **Bounds** is in local coordinate system
  - **Center** is the center of your view in your superview's coordinates

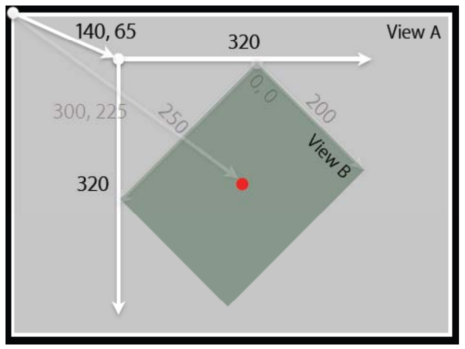
Transform

- 45º Rotation



Frame

- The smallest rectangle in the superview's coordinate system that fully encompasses the view itself



View B Center:
- Origin: (300, 225)

View B Frame:
- Origin: (145, 65)
- Size: 320 x 320

View B Bounds:
- Origin: (0, 0)
- Size: 200 x 250

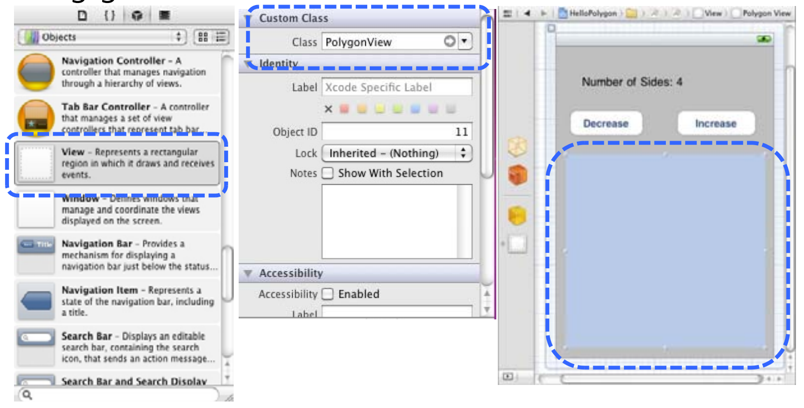
Frame and Bounds

- If you are using a view, typically you use frame
- If you are implementing a view, typically you use bounds
- Matter of perspective
  - From outside it's usually the frame
  - From inside it's usually the bounds
- Examples
  - **Creating a view, positioning a view in superview – use frame**
  - **Handling events, drawing a view – use bounds**

Creating Views

Where do views come from?

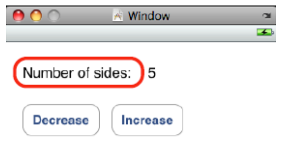
- Commonly Interface Builder
- Drag out any of the existing **view** objects (buttons, labels, etc)
- Drag generic UIView and set **custom class**


Objects
Navigation Controller - A controller that manages navigation through a hierarchy of views.
Tab Bar Controller - A controller that manages a set of view controllers that represent tab bar...
View - Represents a rectangular region in which it draws and receives events.
Window - Defines windows that manage and coordinate the views displayed on the screen.
Navigation Bar - Provides a mechanism for displaying a navigation bar just below the status...
Navigation Item - Represents a state of the navigation bar, including a title.
Search Bar - Displays an editable search bar, containing the search icon, that sends an action message...
Search Bar and Search Display
Custom Class
Identity
Accessibility
Number of Sides: 4
Decrease
Increase

Class: PolygonView
Label: Xcode Specific Label
Object ID: 11
Lock: Inherited - (Nothing)
Notes: Show With Selection
Accessibility: Enabled

Manual Creation

- Views are initialized using **–initWithFrame**
  - CGRect frame = CGRectMake(0, 0, 200, 150);
  - UIView *myView = [[UIView **alloc**] **initWithFrame**: frame];
- Example
  - CGRect frame = CGRectMake(20, 45, 140, 50);
  - UILabel *label = [[UILabel **alloc**] **initWithFrame**: frame];
  - **[window addSubview: label];**
  - [label setText:@"Number of sides:"];
  - **[label release];** // label now retained by window



Defining Custom Views

- When to create my own **UIView subclass**?
- For custom drawing, you override
  - **(void)drawRect: (CGRect) rect;**
- For event handling, you override
  - **(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;**

Drawing Views

-(void)drawRect: (CGRect)rect

- [UIView drawRect:] does nothing by default
  - If not overridden, then backgroundColor is used to fill
- **Override –drawRect: to draw a custom view**
  - **rect** argument is area to draw
- drawRect is invoked automatically
  - Don't call it directly!
- When a view needs to be redrawn, use:
  - **(void)setNeedsDisplay;**
- For example (PolygonView.m)

```
-(void)setNumberOfSides: (int)sides {
    numberOfSides = sides;
    [self setNeedsDisplay];
}
```

CoreGraphics and Quartz 2D

- UIKit offers very basic drawing functionality
  - **UIRectFill(CGRect rect);**
  - **UIRectFrame(CGRect rect);**
- CoreGraphics (CG): Drawing APIs
  - CG is a C-based API, not Objective-C
  - CG and Quartz 2D drawing engine define simple but powerful graphics primitives
    - Graphics context
    - Transformations
    - Paths
    - Colors
    - Fonts
    - Painting operations

Graphics Context

- All drawing is done into an opaque graphics context
- Draws to screen, bitmap buffer, printer, PDF, etc
- Graphics context setup automatically before invoking drawRect
  - Defines current path, line width, transform, etc
  - Access the graphics context within drawRect: by calling **(CGContextRef) UIGraphicsGetCurentContext(void);**
  - Use CG calls to change settings
- Context only valid for current call to drawRect
  - Do not cache the current graphics context in drawRect: to use later!

CG Wrappers

- Some CG functionality wrapped by UIKit
- UIColor
  - Convenience for common colors
  - Easily set the fill and/or stroke colors when drawing

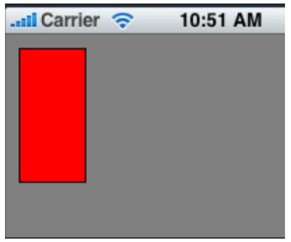
```
UIColor *redColor = [UIColor redColor];
[redColor set];
//drawing will be done in red
```

- UIFont
  - Access system font
  - Get font by name

```
UIFont *font = [UIFont systemFontOfSize:14.0];
[myLabel setFont:font];
```

Simple Rect Example

```
// draw a solid color and shape
-(void)drawRect: (CGRect)rect {
    CGRect bounds = [self bounds];
    [[UIColor grayColor] set];
    UIRectFill(bounds);
    CGRect square = CGRectMake(10, 10, 50, 100);
    [[UIColor redColor] set];
    UIRectFill(square);
    [[UIColor blackColor] set];
    UIRectFrame(square);
}
```



Drawing More Complex Shapes

- Common steps for drawRect: are
  - Get current graphics context
  - Define a path
  - Set a color
  - Stroke or fill path
  - Repeat, if necessary

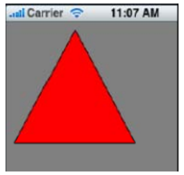
Paths

- CoreGraphics paths define shapes
- Made up of lines, arcs, curves and rectangles
- Creation and drawing of paths are two distinct operations
  - Define path first, then draw it
- Two parallel sets of functions for using paths
  - CGContext "convenience" throwaway functions
  - CGPath functions for creating reusable paths

CGContext	CGPath
CGContextMoveToPoint	CGPathMoveToPoint
CGContextAddLineToPoint	CGPathAddLineToPoint
CGContextAddArcToPoint	CGPathAddArcToPoint
CGContextClosePath	CGPathSubPath
and so on……	

Simple Path Example

```
// draw a shape and path
-(void)drawRect: (CGRect)rect {
    CGContextRef context = UIGraphicsGetCurrentContext();
    [[UIColor grayColor] set];
    UIRectFill([self bounds]);
    CGContextBeginPath(context);
    CGContextMoveToPoint(context, 75, 10);
    CGContextAddLineToPoint(context, 10, 150);
    CGContextAddLineToPoint(context, 160, 150);
    CGContextClosePath(context);
    [[UIColor redColor] setFill];
    [[UIColor blackColor] setStroke];
    CGContextDrawPath(context, kCGPathFillStroke);
}
```



Images & Text

UIImage

- UIKit class representing an image
- Creating UIImages (Fetching image in application bundle)
  - Use **+[UIImage imageNamed: (NSString *)name]**
  - Include file extension in file name, e.g. @"myimg.jpg"
- Creating UIImages (Read from file on disk)
  - Use **–[UIImage initWithContentsOfFile: (NSString *)path]**
- Creating UIImages (From data in memory)
  - Use **–[UIImage initWithData: (NSData *)data]**

Creating Images from a Context

- Need to dynamically generate a bitmap image
- Same as drawing a view
- General steps
  - Create a special CGGraphicsContext with a size
  - Draw
  - Capture the context as a bitmap
  - Clean up

Bitmap Image Example

```
// creating an image from a current graphics context
-(UIImage *)polygonImageOfSize: (CGSize)size {
    UIImage *result = nil;
    UIGraphicsBeginImageContext(size); // create CGGraphicsContext

    // call your drawing code ...

    result = UIGraphicsGetImageFromCurrentContext(); // capture
    UIGraphicsEndImageContext(); // clean up
    return result;
}
```

Getting Image Data

- Given UIImage, want PNG or JPG representation
  - **NSData *UIImagePNGRepresentation(UIImage * image);**
  - **NSData *UIImageJPGRepresentation(UIImage * image);**
- UIImage also has a CGImage property which will give you a CGImageRef to use with CG calls

Drawing Text & Images

- You can draw UIImages in -drawRect
  - **[UIImage drawAtPoint: (CGPoint)point]**
  - **[UIImage drawInRect: (CGRect)rect]**
  - **[UIImage drawAsPatternInRect: (CGRect)rect]**
- You can draw NSString in –drawRect
  - **[NSString drawAtPoint: (CGPoint)point withFont: (UIFont *)font]**

References

- Lecture 4 & 5 Slide from iPhone Application Development (Winter 2010) @Stanford University