Animation in iOS

Alexis Goldstein
@alexisgoldstein

Agenda for Today

- Animation Sample Code
- UIView Animations:
  - Animatable Properties
  - Animation Options
  - Transitions
- CoreAnimation:
  - Implicit Animations
  - Explicit Animations
- Where to go to learn more

Sample Code

- I've created some basic sample code to go along with this talk.
  - I leveraged Nathan Eror's CA360 project as a base, and then added all new examples for this talk.
- You can find my sample code at: http://github.com/alexisgo/AnimationTalk
- You can find these slides at: http://bit.ly/alexisAnim

About Me

- Spent seven years in technology on Wall Street.
- Quit to pursue my dreams of going indie.
- Launched **aut faciam**, an iOS software house, in July 2010.

- Director of Operations for **Girl Develop IT**:
  - Aim is to provide low-cost, non-intimidating programming classes for women.

About You

- How long have you been working with iOS?
- What do you want to animate / what kinds of things have you used animation for?

UIView Animations

UIView Animations

- Powered by CoreAnimation.
- Hardware Accelerated:
  - Animations happen on the GPU, rather than the CPU, to improve performance.

UIView Animations

- How you perform a UIView animation depends on the iOS version.
- iOS 4.0 and later:
  - block-based animation methods, such as **animateWithDuration: animations: completion:**
- iOS 3.2 and earlier:
  - **beginAnimation** and **commitAnimations**

Blocks (in 30 seconds)

- Blocks are objects that encapsulate a section of code that can be executed at any time.
- ^ introduces a block.
- When used as method or function arguments, blocks are a type of callback.
  - "When invoked, the method or function performs some work and, at the appropriate moments, calls back to the invoking code—via the block—to request additional information or to obtain application-specific behavior from it." -- A Short Practical Guide to Blocks
- Blocks have access to the local variables, parameters & even stack variables of the method you define it in.
- Blocks also have access to functions and global variables, including instance variables.

UIView Animations: what we'll focus on today

- How you perform a UIView animation depends on the OS version.
  - iOS 4.0 and later:
    - block-based animation methods, such as **animateWithDuration: animations: completion:**
  - iOS 3.2 and earlier:
    - **beginAnimation** and **commitAnimations**

UIView Animations

- There are certain properties in UIView that are animatable.
- You create UIView animations by changing the value of these animatable properties from within an animation block (iOS 4.0+)

animations:^ { ... }

UIView Animatable Properties

Property	Description
frame	The view's frame rectangle
bounds	The view's bounding rectangle
center	The center of the frame
transform	The transform applied to the view, relative to the center of its bounds
alpha	The view's level of transparency

UIView's animateWithDuration

- The most basic method to use to create UIView animations is

animateWithDuration:
animations:

Animating a change in Alpha

```
[UIView animateWithDuration:1.0
        animations:^
        {
            [myView setAlpha:1.0];
        }];
```

Demo: SetAlpha.m in the sample code

Animating a change in Center

```
[UIView animateWithDuration:1.0
      animations:^
       {
          [myView setCenter:
              CGPointMake(200, 200)];
}];
```

Animating a Transform

```
[UIView animateWithDuration:1.0
    animations:^
      {
        [myView setTransform:
          CGAffineTransformMakeRotatio
          n(M_PI)];
}];
```

Nesting Animation Blocks

- You can nest animations inside one another
- See NestedAnimations.m in the sample code
- http://github.com/alexisgo/AnimationTalk

UIView Animation Options

- You can also configure the animation through UIViewAnimationOptions.
- In iOS 4.0+, these are set through the **options:** argument of **animateWithDuration: delay: options: animations: completion:**
    - In iOS 3.2 and earlier: Setter methods in several class methods of UIView.

Animation Options

```
[UIView animateWithDuration: 2
        delay: 1.2
        options: (UIViewAnimationOptions)options
        animations:^{ ... }
        completion:^(BOOL finished) { ... } ];
```

Animation Options

- UIViewAnimationOptionAllowUserInteraction
  - Allows user interaction during an animation (off by default).
- UIViewAnimationOptionAutoreverse
  - Runs the animation back and forth, in a loop
- UIViewAnimationOptionBeginFromCurrentState
  - To avoid jumps between animations, if a second animation starts midway through another animation.
- For the full list of all options, search for UIViewAnimationOptions at http://bit.ly/cBMPMg

Transitions

- iOS 4.0+:
  - transitionFromView:toView:duration:options:completion:
  - transitionWithView:duration:options:animations:completion
- iOS 3.2 and earlier:
  - setAnimationTransition:forView:cache:

Transitions: CurlUp

To create a curling effect on a view:

```
[UIView transitionWithView: myView
                  duration:2.0
options:UIViewAnimationOptionTransitionCurlUp
                animations:^{ ... }
                completion:nil
];
```

Transitions: CurlUp

- Please BEWARE your Code Sense!
- Make sure you are using the correct option
  - DO NOT WANT UIViewAnimationTransitionCurlUp!
    - This is the old enum used with the (pre-blocks) setAnimationTransition method.
- You want: UIViewAnimation**Option**TransitionCurlUp

Transitions: CurlUp

- If you'd like to transition from an old view to a new view, and want the old view to be removed from the view hierarchy, and the new one added, use:

```
[UIView transitionFromView:myView
                    toView:newView
                  duration:2.0
                   options:UIViewAnimationOptionTransitionCurlUp
                completion:nil];
```

Transitions: Flip

```
[UIView transitionWithView:transitionView
                  duration:2.0
options:UIViewAnimationOptionTransitionFlipFromLeft
                animations: nil
                completion:^(BOOL finished)
{
    ...
}];
```

UIView Resources

Apple Docs:

- Getting Started with Graphics and Animation
- "Animating Views" section of the View Programming Guide for iOS

WWDC 2010 Talks & Sample Code:

- Building Animation Driven Interfaces
- iPlant

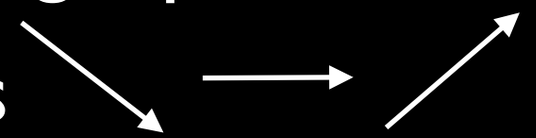
CoreAnimation

UIView or CoreAnimation?

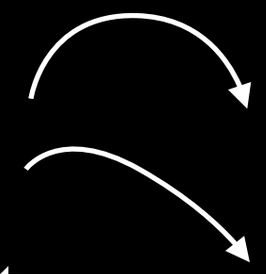
- Almost always, you should use UIView animations.

When should you use Core Animation?

- Use CoreAnimation when you can't accomplish what you need with UIView animations:
  - You want to specify motion along a path
    - UIView gives you linear paths



    - CoreAnimation lets you specify any path



  - You want to animate properties that are not animatable in UIView, such as: zPosition, cornerRadius, borderWidth, or borderColor.
  - You have something very lightweight or short-lived.

When should you use Core Animation?

- A great example of a good time to use Core Animation is if want to rotate a UIView.
  - With UIView animations, we can only rotate around the **default** anchorPoint of the UIView, which is the middle of the view.
  - But you can modify that default anchor point through the CALayer of that viewew:

    [myView.layer.anchorPoint = CGPointMake (0,0);
- DEMO: Great visualization of this is in LayerTransforms.m class in Nathan Eror's CA360 project (http://github.com/neror/CA360)

About Core Animation

- *What is it?* A compositing and animation API.
- Uses CALayer:
  - Layers have hierarchies, just like UIViews.
  - Can add, remove and reorder CALayers in the same way you add/remove/reorder UIViews.
- Hardware Accelerated.
- Interpolation of animations happens away from the main thread.

Jargon Alert! Compositing

COMPOSITING:

"The combining of visual elements from separate sources into single images, often to create the illusion that all those elements are parts of the same scene."

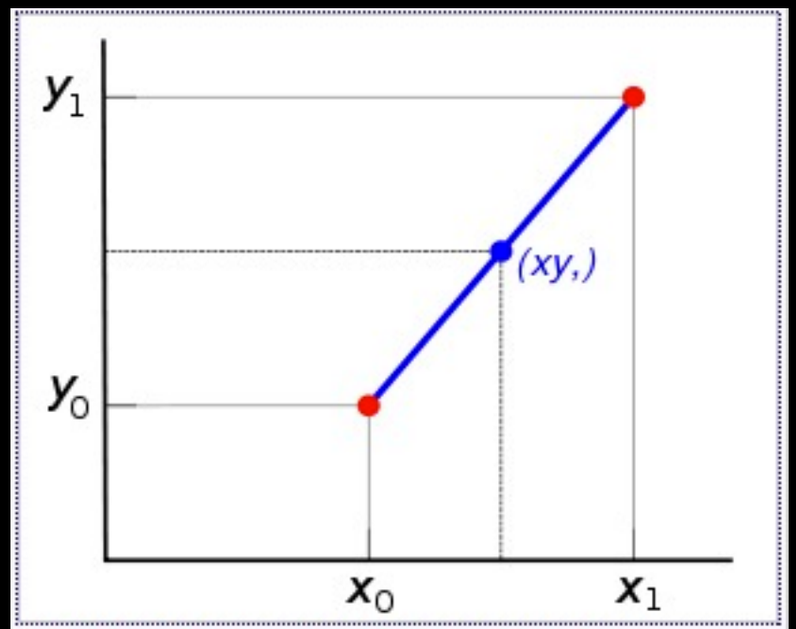
--http://en.wikipedia.org/wiki/Compositing

INTERPOLATION:

Deriving the value of one or more **unknown** intermediate points based **known** start and end points.

Image source:

http://en.wikipedia.org/wiki/Linear_interpolation



Given the two red points, the blue line is the linear interpolant between the points, and the value *y* at *x* may be found by linear interpolation.

Using Core Animation

Remember to

#import <QuartzCore/QuartzCore.h>

and to add the QuartzCore framework to your project.

First Step! Adding Content to your Layer

There are three ways to add content to a CALayer:

1. Set the **contents** property on the layer using a CGImageRef
2. Use a delegate to provide or draw the content
3. Subclass CALayer and override one of the display methods

Adding Layer Content

- Excellent details on the different ways to do this can be found at:
  - http://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/CoreAnimation_guide/Articles/ProvidingCALayerContent.html

Types of Animation: Implicit vs Explicit

- CoreAnimation has both implicit and explicit animations.
- Implicit Animations:
  - Implicit animations occur *automatically* whenever you modify an animatable layer property. (Sweet!)
- Explicit Animations:
  - Explicit animations are created and then explicitly added to your layer by calling addAnimation:

Implicit Animation

- When you set layer properties, the next time the run-loop gets control, it implements a transaction.
  - That transaction will cause all the layer properties that have changed to animate according to certain defaults.
  - To override these defaults, you can use CATransaction.
- Sample code: BasicAnimation.m

Overriding Implicit Animation Defaults

- Let's say we have the following implicit animation:

```
myLayer.position = CGMakePoint(100,
200);
```

- If we want this to take a bit longer, we can add:

```
[CATransaction setAnimationDuration: 5];
[CATransaction
    setAnimationTimingFunction: [CAMediaTimingFunction
    functionWithName:kCAMediaTimingFunctionEaseInEaseOut]
myLayer.position = CGMakePoint(200,100);
```

Adding Drop Shadows

- @property CGPathRef shadowPath
- You specify where the layer is opaque
- The compositor takes care of creating the drop shadow, and it does so by caching a shadow bitmap.

> COMPOSITING:
>
> The combining of visual elements from separate sources into single images, often to create the illusion that all those elements are parts of the same scene.
>
> Source: *http://en.wikipedia.org/wiki/Compositing*

- Sample code: ShadowPath.m

Explicit Animations

- If you want more control over your animations, use Explicit Animations:
  - CABasicAnimation
  - CAKeyframeAnimation
  - CAAnimationGroup
  - CATransition

How to Create Explicit Animations

- Use keyPath to specify **which property you are changing**.
  - @"position.x"
  - @"transform.scale"
- Specify a from value and a to value (make something change).
- Add the animation to the layer.

```
[CABasicAnimation *basic =
[CABasicAnimation
    animationWithKeyPath:@"opacity"];
...
[myView.layer addAnimation:basic];
```

CABasicAnimation

- Provides simple interpolation for a layer property.
- Simple example: BasicAnimation.m

Explicit Animations

- Please note: The model value DOES NOT CHANGE! The animation is changing the presentation only.

Explicit Animations: Making it Stick

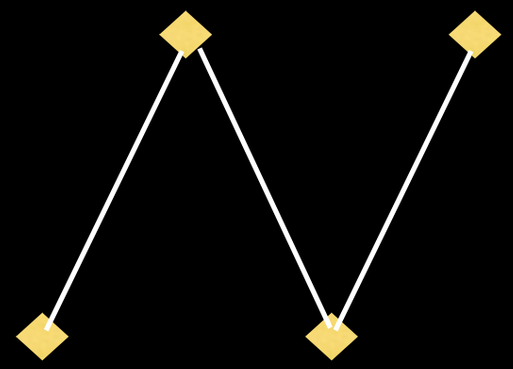
- Because you are not changing the underlying model, the layer will go back to where it was before the animation--unless you also change the model.
- Let's examine BasicAnimation.m from the sample code.
- For more on this topic:
    - http://bit.ly/bKxQPJ
    - WWDC10's "Core Animation in

Keyframe Animations

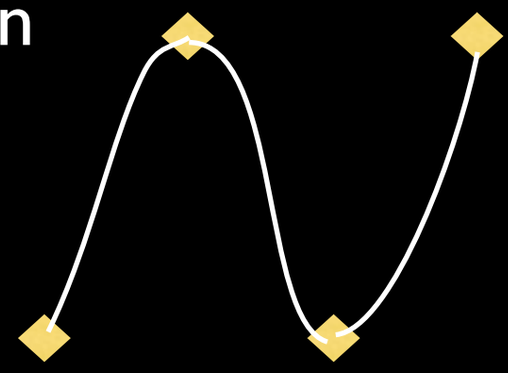
- You provide a set of key frames, and Core Animation fills in the gaps.
- You can provide the frames through either:
  - An array of values
  - Series of points in a CGPathRef

Cubic Keyframe Animations

- Interpolation is usually linear -- so interpolating between two points will get you a straight line.



- Starting in iOS 4.0, a new calculationMode was added, which calculates the interpolation based on the two points AS WELL AS the points surrounding is--giving a cubic interpolation.



- animation.calculationMode = **kCAAnimationCubic**
- Sample code: Keyframe.m

CA360 Project

- Nathan Eror (@neror) has an excellent resource on GitHub, CA360, which I used as a base for my sample code
- I highly recommend checking it out
  - http://github.com/neror/CA360

Resources to Learn More

Apple Docs

- Getting Started with Graphics and Animation
- Core Animation Programming Guide
- "Animating Views" section of the View Programming Guide for iOS

WWDC 2010 Talks:

- Building Animation Driven Interfaces
- Core Animation in Practice: Part I
- Core Animation in Practice: Part II

Nathan Eror, Freetime Studios

- Talk from 360 iDev 2009
- 360 sample code: http://github.com/neror/CA360
- FTUtils: http://ftutils.com/