iPhone Programming

CMSC 498i – Spring 2010

Animation

Lecture #13 – Chuck Pisula

Today's Topics

- Animation
  - UIKit Animation Essentials
  - Coordinating multiple animations
  - Advanced animations – keyframes, animation curves, etc…
- Touch Events Handling

Animation

Introduction

Introduction To Animation

- Naive Approach
  - Create a view, implement `-drawRect:`
  - Set up a repeating timer (firing every 1/30th of a second)
  - Redraw when timer fires
- Drawbacks
  - Redrawing is slow and should be avoided
  - Hardcoded frame rate
  - Complexity – Coordinating multiple animations, state tracking, timer management

Core Animation

- Problem – Dynamic, animated UI can be hard to create
- Solution – Core Animation (CA)
  - Simple programming model
  - Familiar view-like hierarchies
  - Hardware accelerated rendering engine
    - Concurrent animations
    - Animate hundreds of views / layers simultaneously
  - Available on iPhone OS, Mac OS X

Core Animation

- Easy To Use
  - State of animation automatically tracked
  - Animation run automatically in a separate thread
    - No more interaction required by you
    - No need to manage timers
- `UIView` integration
  - `UIView` covers most animation needs
  - Use Core Animation directly when more options are needed

Core Animation Performance

- UIView's are backed by Core Animation layers
  - Drawing results are "cached" in a `CALayer`
  - Your `-drawRect:` is called only when contents change
- High Performance
  - Layers stored in process responsible for compositing
  - Animation can be achieved by direct manipulation of the layer
    - "Drawing is slow" – this avoids unnecessary redraws
    - Animation and compositing happens all in the same process
  - Layers hardware accelerated without any work on your part

Core Animation Features

- Properties – interpolates between ‘to’ and ‘from’ value
  - Examples: opacity, position, backgroundColor
  - By default property changes are animated
- Transitions – provides stock effects like fade, reveal, etc…
- Grouping – multiple animations may run together
- Coordination – easy to coordinate multiple animations
- Timing Functions – provide custom functions to customize the interpolation curve
- Keyframe Animations – specify property, array of values and time for each value

UIView Animations

- Properties – interpolates between ‘to’ and ‘from’ value
  - Examples: opacity, position, backgroundColor
  - By default property changes are animated
- Transitions – provides stock effects like fade, reveal, etc…
- Grouping – multiple animations may run together
- Coordination – easy to coordinate multiple animations
- Timing Functions – provide custom functions to customize the interpolation curve
- Keyframe Animations – specify property, array of values and time for each value

UIView Animations

- ✓ Properties – interpolates between ‘to’ and ‘from’ value
  - Examples: opacity, position, backgroundColor
  - By default property changes are animated
- Transitions – provides stock effects like fade, reveal, etc…
- Grouping – multiple animations may run together
- Coordination – easy to coordinate multiple animations
- Timing Functions – provide custom functions to customize the interpolation curve
- Keyframe Animations – specify property, array of values and time for each value

UIView Animations

- ✓ Properties – interpolates between ‘to’ and ‘from’ value
  - Examples: opacity, position, backgroundColor
  - By default property changes are animated
- Transitions – provides stock effects like fade, reveal, etc…
- Grouping – multiple animations may run together
- Coordination – easy to coordinate multiple animations
- Timing Functions – provide custom functions to customize the interpolation curve
- Keyframe Animations – specify property, array of values and time for each value

UIView Animations

- ✓ Properties – interpolates between ‘to’ and ‘from’ value
  - Examples: opacity, position, backgroundColor
  - By default property changes are **NOT** animated
- Transitions – provides stock effects like fade, reveal, etc…
- Grouping – multiple animations may run together
- Coordination – easy to coordinate multiple animations
- Timing Functions – provide custom functions to customize the interpolation curve
- Keyframe Animations – specify property, array of values and time for each value

UIView Animations

- ✓ Properties – interpolates between ‘to’ and ‘from’ value
  - Examples: opacity, position, backgroundColor
  - By default property changes are **NOT** animated
- ✓ Transitions – provides stock effects like fade, reveal, etc…
- Grouping – multiple animations may run together
- Coordination – easy to coordinate multiple animations
- Timing Functions – provide custom functions to customize the interpolation curve
- Keyframe Animations – specify property, array of values and time for each value

UIView Animations

- ✓ Properties – interpolates between ‘to’ and ‘from’ value
  - Examples: opacity, position, backgroundColor
  - By default property changes are **NOT** animated
- ✓ Transitions – provides stock effects like fade, reveal, etc…
- ✓ Grouping – multiple animations may run together
- Coordination – easy to coordinate multiple animations
- Timing Functions – provide custom functions to customize the interpolation curve
- Keyframe Animations – specify property, array of values and time for each value

UIView Animations

- ✓ Properties – interpolates between ‘to’ and ‘from’ value
  - Examples: opacity, position, backgroundColor
  - By default property changes are **NOT** animated
- ✓ Transitions – provides stock effects like fade, reveal, etc…
- ✓ Grouping – multiple animations may run together
- ✓ Coordination – easy to coordinate multiple animations
- Timing Functions – provide custom functions to customize the interpolation curve
- Keyframe Animations – specify property, array of values and time for each value

UIView Animations

- ✓ Properties – interpolates between ‘to’ and ‘from’ value
  - Examples: opacity, position, backgroundColor
  - By default property changes are **NOT** animated
- ✓ Transitions – provides stock effects like fade, reveal, etc…
- ✓ Grouping – multiple animations may run together
- ✓ Coordination – easy to coordinate multiple animations
- ✓ Timing Functions – simplified API to customize the animation curve used
- Keyframe Animations – specify property, array of values and time for each value

UIView Animations

- ✓ Properties – interpolates between ‘to’ and ‘from’ value
  - Examples: opacity, position, backgroundColor
  - By default property changes are **NOT** animated
- ✓ Transitions – provides stock effects like fade, reveal, etc…
- ✓ Grouping – multiple animations may run together
- ✓ Coordination – easy to coordinate multiple animations
- ✓ Timing Functions – simplified API to customize the animation curve used
- X Keyframe Animations – Use CoreAnimation

UIView Animations

- ✓ Properties – interpolates between ‘to’ and ‘from’ value
  - Examples: opacity, position, backgroundColor
  - By default property changes are **NOT** animated
- ✓ Transitions – provides stock effects like fade, reveal, etc…
- ✓ Grouping – multiple animations may run together
- ✓ Coordination – easy to coordinate multiple animations
- ✓ Timing Functions – simplified API to customize the animation curve used
- X Keyframe Animations – Use CoreAnimation

UIView Animations

Details

Animation Use

- Create “blocks” around changes you wish to animate

```
[UIView beginAnimations:@”animationID” context:NULL]
```

- Make changes to animatable properties

```
[view1 setFrame:newFrame]; // ‘currFrame’ → ‘newFrame’
[view2 setAlpha:newAlpha]; // ‘currAlpha‘ → ‘newAlpha’
```

- Close animation block to run animation

```
[UIView commitAnimations];
```

Animation Use

- Create “blocks” around changes you wish to animate

```
[UIView beginAnimations:@”animationID” context:NULL]
```

- Make changes to animatable properties

```
[view1 setFrame:newFrame]; // ‘currFrame’ → ‘newFrame’
[view2 setAlpha:newAlpha]; // ‘currAlpha‘ → ‘newAlpha’
```

- Close animation block to run animation

```
[UIView commitAnimations];
```

- Temporarily disable animating properties in a block

```
[UIView setAnimationsEnabled:NO];
```

Animatable Properties

- `CGFloat alpha`
- `CGRect bounds, frame`
  - what happens depends on `view.contentMode`
- `CGPoint center`
- `CGAffineTransformation transform`
- `UIColor *backgroundColor`

Coordination

- Changes within same block animate together
- Each block may define delegate and callbacks

```
+ (void)setAnimationDelegate:(id)delegate;
+ (void)setAnimationWillStartSelector:(SEL)selector;
+ (void)setAnimationDidStopSelector:(SEL)selector;
```

Coordination

- Changes within same block animate together
- Each block may define delegate and callbacks

```
+ (void)setAnimationDelegate:(id)delegate;
+ (void)setAnimationWillStartSelector:(SEL)selector;
+ (void)setAnimationDidStopSelector:(SEL)selector;
```

- Delegate messaged when animation begins

```
- (void)animationWillStart:(NSString *)animationID
                   context:(void *)context;
```

- Delegate messaged when animation ends

```
- (void)animationDidStop:(NSString *)animationID
                finished:(NSNumber *)finishedByCompleting
                 context:(void *)context;
```

Additional Options

- Customizable duration

```
+ (void)setAnimationDuration:(NSTimeInterval)duration; // default = 0.2
```

- Customizable starting time

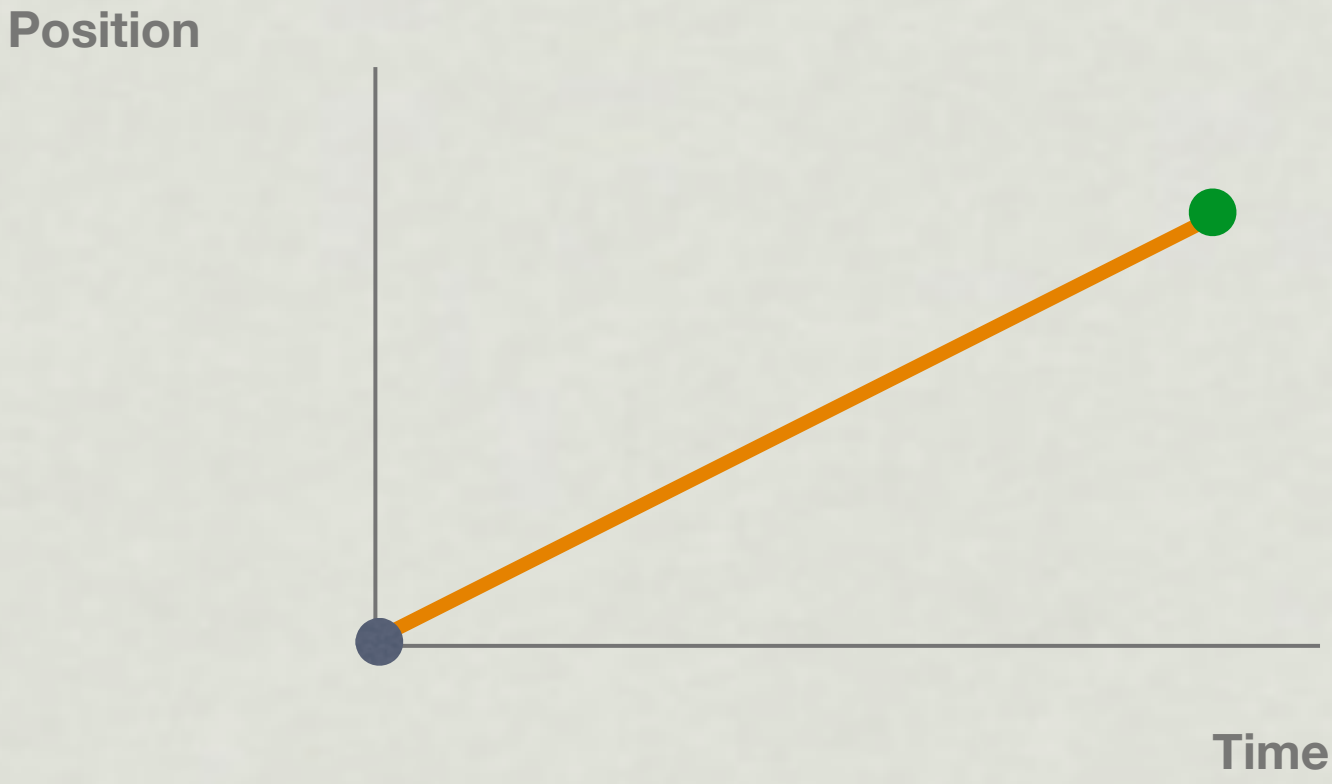
```
+ (void)setAnimationDelay:(NSTimeInterval)delay; // default = 0.0
+ (void)setAnimationStartDate:(NSDate *)startDate; // default = 'now'
```

- Interpolation curve setting

```
+ (void)setAnimationCurve:(UIViewAnimationCurve)curve; // default = 'in/out'
```

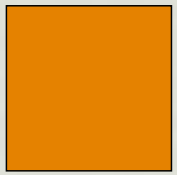
```
typedef enum {
    UIViewAnimationCurveEaseInOut,          // slow at beginning and end
    UIViewAnimationCurveEaseIn,             // slow at beginning
    UIViewAnimationCurveEaseOut,            // slow at end
    UIViewAnimationCurveLinear
} UIViewAnimationCurve;
```

Curves Linear



Curves

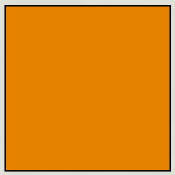
Linear

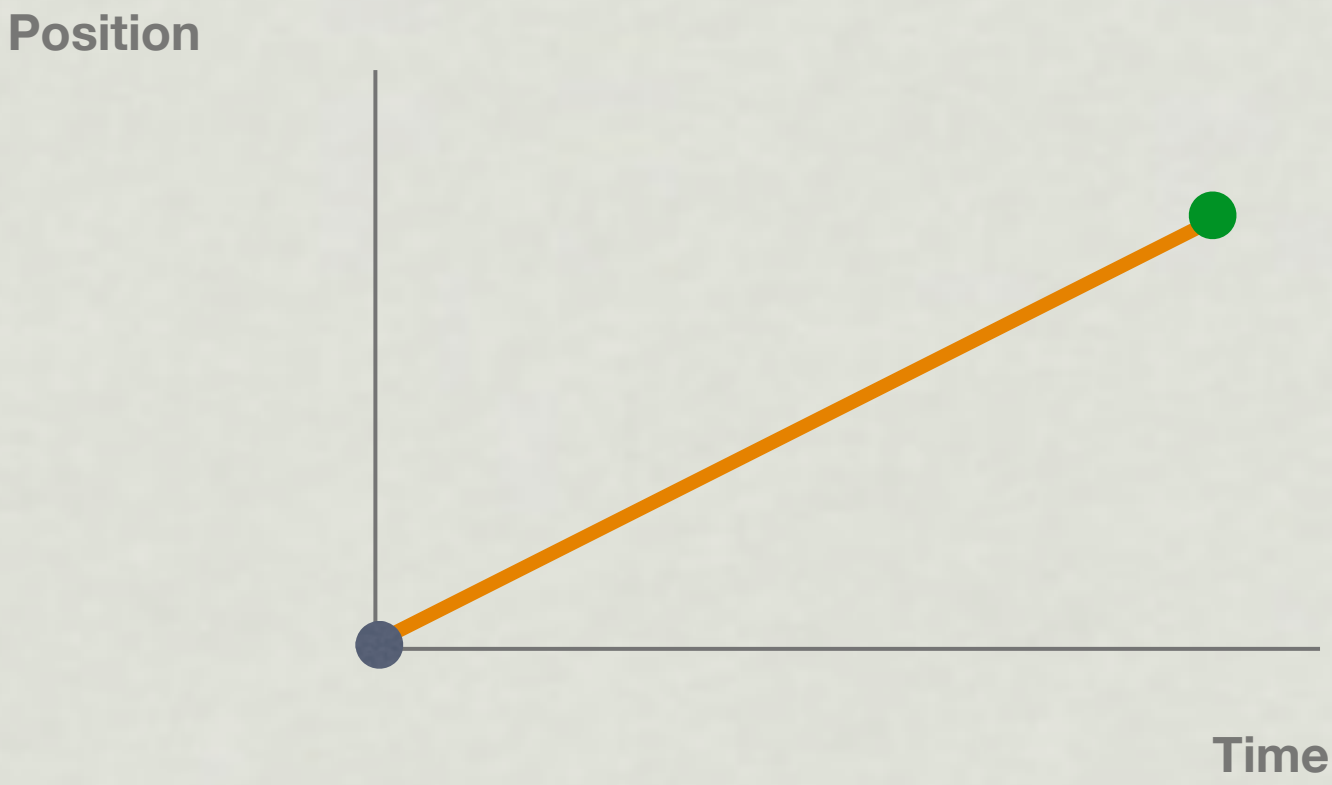


Position

Time

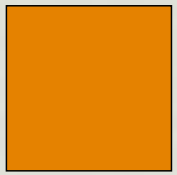
Curves Linear



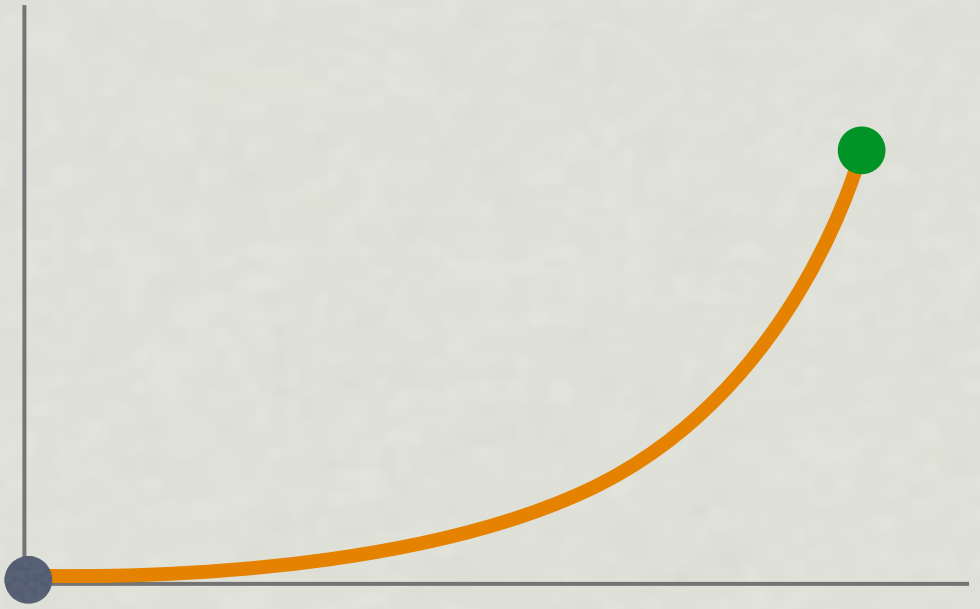


Curves

EaseIn

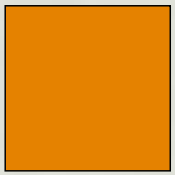


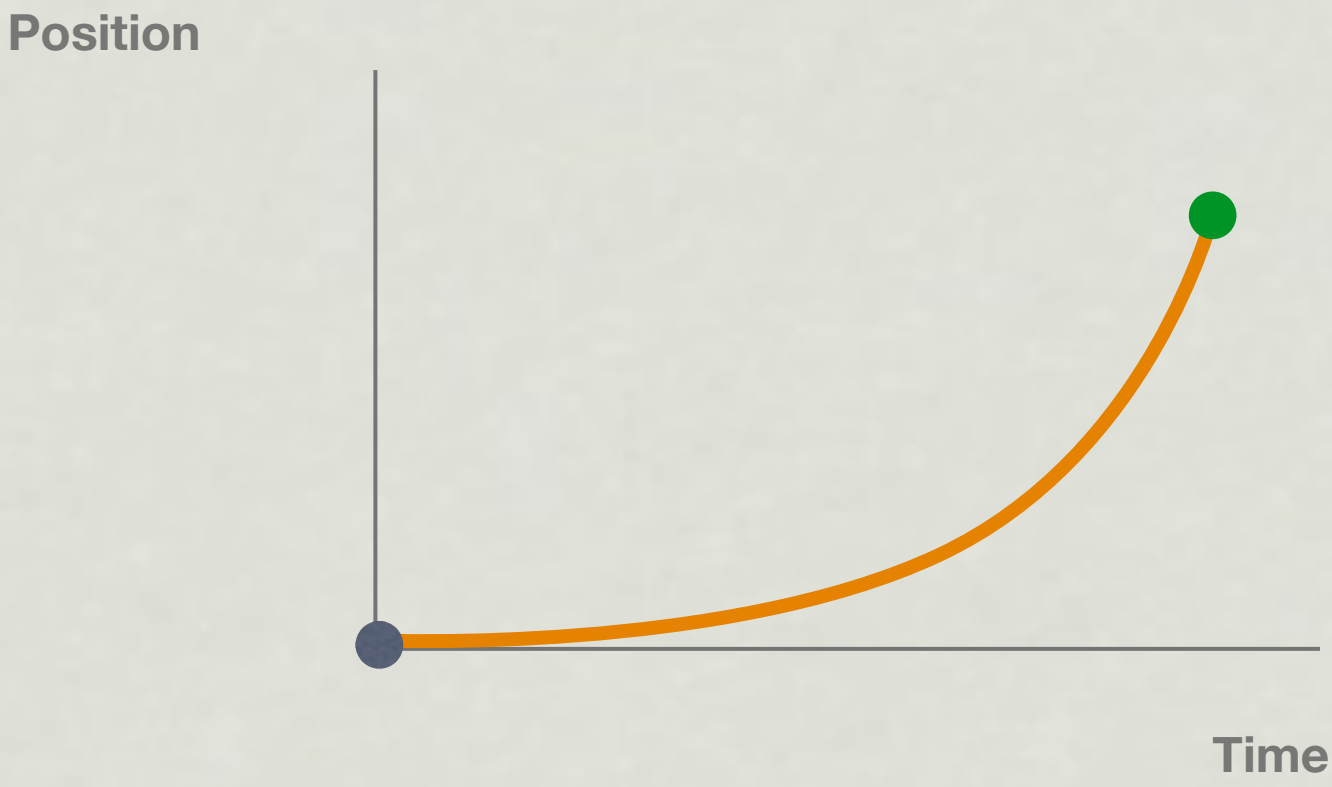
Position



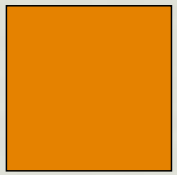
Time

Curves EaseIn





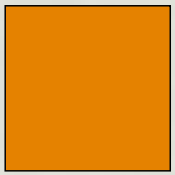
Curves EaseOut

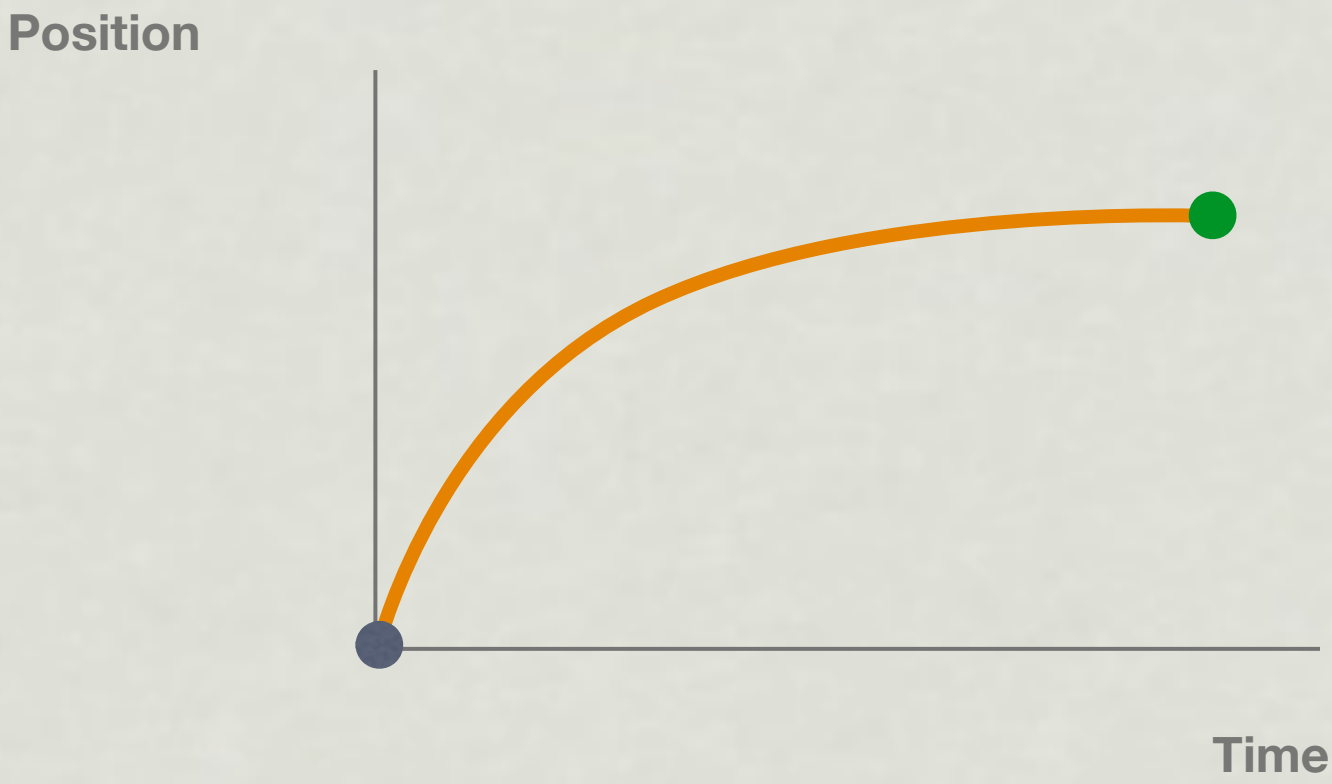


Position

Time

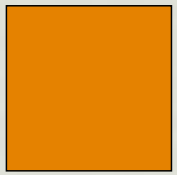
Curves EaseOut



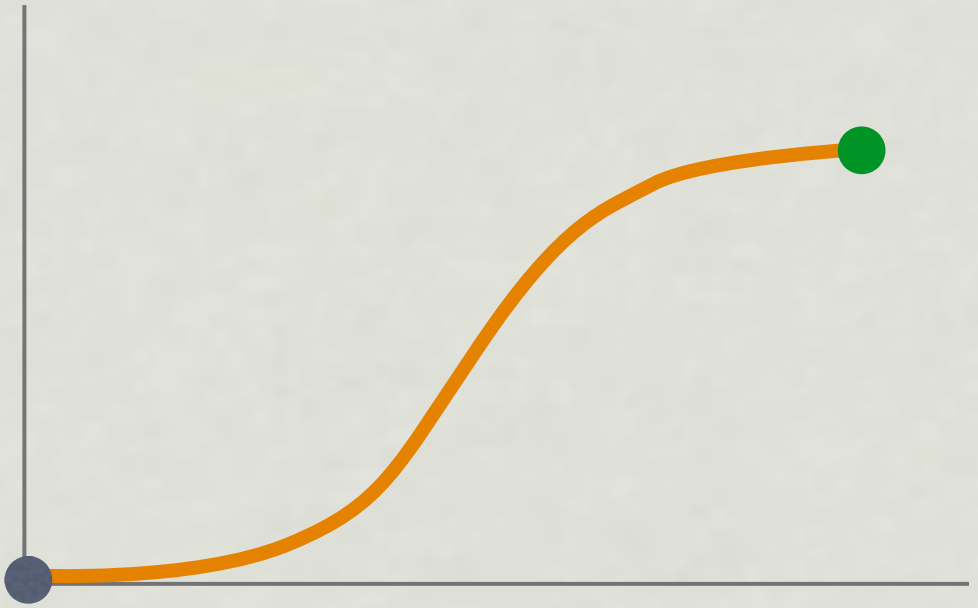


Curves

EaseIn, EaseOut



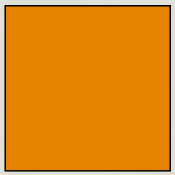
Position

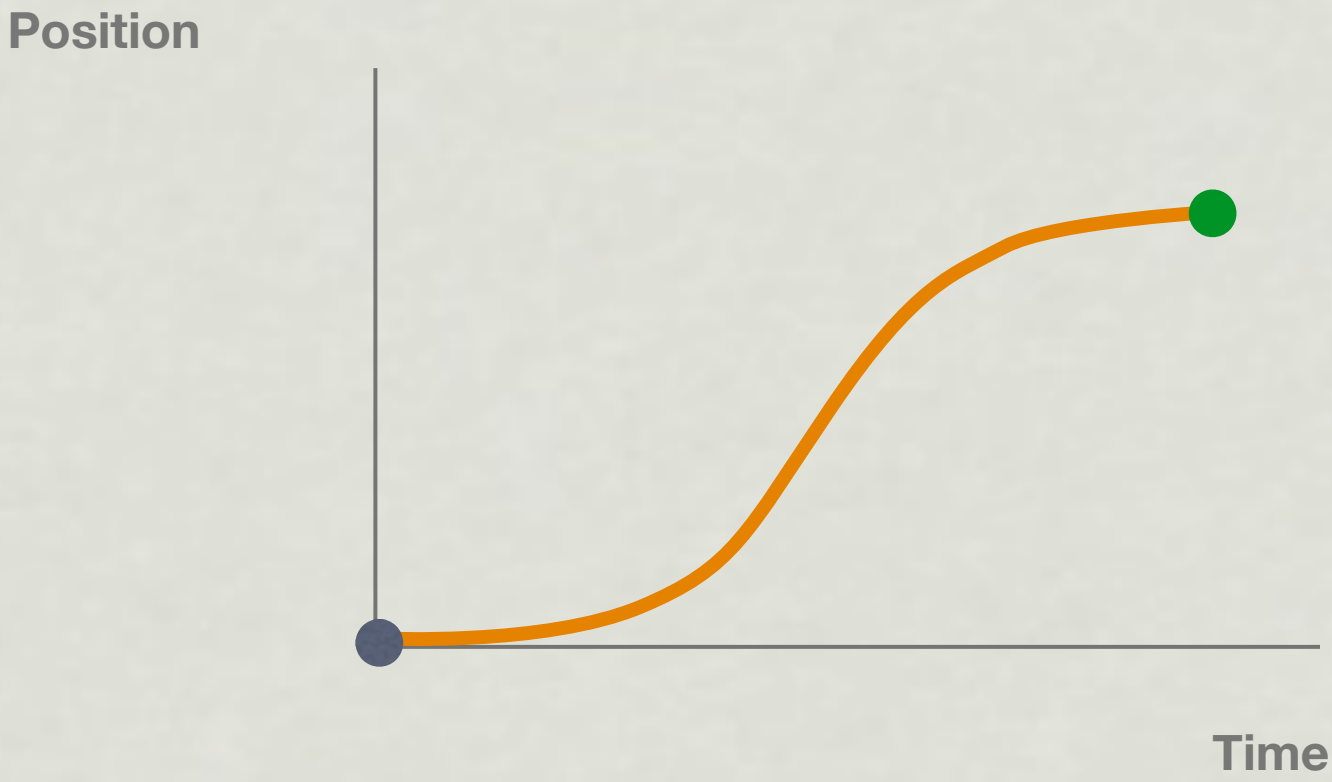


Time

Curves

EaseIn, EaseOut





Additional Options

- Repeat an animation

```
+ (void)setAnimationRepeatCount:(float)repeatCount; // fractional values OK
```

- Autoreverse when repeating – e.g. ‘start‘ → ‘end’ → ‘start‘ → …

```
+ (void)setAnimationRepeatAutoreverses:(BOOL)repeatAutoreverses;
```

- Stopping an animation
  - Normally you don’t need to deal with this
  - However, say you have an “infinitely” repeating animation...

```
// CALayer has API for stopping animations
[[view layer] removeAllAnimations]
```

In Flight Animations

- Animations that have not yet started or finished
- Accessing **in–flight** properties will return the end value

```
// Set an initial ‘startFrame’
[theView setFrame: CGRectMake(0, 0, 10, 10)];

// Animate to ‘endFrame’
[UIView beginAnimations:@"move" context:NULL]
[theView setFrame:CGRectMake(0, 100, 10, 10)];
[UIView commitAnimations];
```

In Flight Animations

- Animations that have not yet started or finished
- Accessing **in–flight** properties will return the end value

```
// Set an initial ‘startFrame’
[theView setFrame: CGRectMake(0, 0, 10, 10)];

// Animate to ‘endFrame’
[UIView beginAnimations:@"move" context:NULL]
[theView setFrame:CGRectMake(0, 100, 10, 10)];
[UIView commitAnimations];
```

- Before animation even starts, or has finished…

```
NSLog(@”frame origin is %@”, NSStringFromCGPoint([theView frame].origin));
```

In Flight Animations

- Animations that have not yet started or finished
- Accessing **in–flight** properties will return the end value

```
// Set an initial ‘startFrame’
[theView setFrame: CGRectMake(0, 0, 10, 10)];

// Animate to ‘endFrame’
[UIView beginAnimations:@"move" context:NULL]
[theView setFrame:CGRectMake(0, 100, 10, 10)];
[UIView commitAnimations];
```

- Before animation even starts, or has finished…

```
NSLog(@”frame origin is %@”, NSStringFromCGPoint([theView frame].origin));
```

- Output

```
frame origin is { 0, 100 }
```

In Flight Animations

- When starting an animation for a view already animating…
- Starting property value determined by

```
+ (void)setAnimationBeginsFromCurrentState:(BOOL)fromCurrentState;
```

- To start from end (goal) value
  - `fromCurrentState = NO`
  - in-flight animation will end before next begins
- To start from current intermediate value
  - `fromCurrentState = YES`
  - animation begin immediately from current position

Sample Code

- Animate a view from an onscreen location
- Animate to an offscreen location
- Fade the view from opaque to transparent at the same time
- After the animation finishes, remove the view and release it

Sample Code

```
// make sure someView is not freed before the end callback
[someView retain];

// start up an animation and set the “did end” callback, pass “someView” as context
[UIView beginAnimations:@"animateFromLeftToRight" context:someView];
[UIView setAnimationDelegate:self];
[UIView setAnimationDidStopSelector:@selector(animateOffscreen:finished:context:)];

// slide the view to the right and offscreen
CGPoint center = [someView center];
center.x = CGRectGetMaxX(bounds) + CGRectGetWidth(someView) / 2.0;
[someView setCenter:center];

[UIView commitAnimations]; // commit
```

```
- (void)animateOffscreen:(NSString *)animationID
                finished:(NSNumber *)finished
                 context:(void *)context
{
    // ‘context’ is the view passed when the animation began
    UIView *view = (UIView *)context;

    // The view is now offscreen remove, and release it
    [view removeFromSuperview];
    [view release];
}
```

Sample Code

```
// make sure someView is not freed before the end callback
[someView retain];

// start up an animation and set the “did end” callback, pass “someView” as context
[UIView beginAnimations:@"animateFromLeftToRight" context:someView];
[UIView setAnimationDelegate:self];
[UIView setAnimationDidStopSelector:@selector(animateOffscreen:finished:context:)];

// slide the view to the right and offscreen
CGPoint center = [someView center];
center.x = CGRectGetMaxX(bounds) + CGRectGetWidth(someView) / 2.0;
[someView setCenter:center];

[UIView commitAnimations]; // commit
```

```
- (void)animateOffscreen:(NSString *)animationID
                finished:(NSNumber *)finished
                 context:(void *)context
{
    // ‘context’ is the view passed when the animation began
    UIView *view = (UIView *)context;

    // The view is now offscreen remove, and release it
    [view removeFromSuperview];
    [view release];
}
```

Sample Code

```
// make sure someView is not freed before the end callback
[someView retain];

// start up an animation and set the “did end” callback, pass “someView” as context
[UIView beginAnimations:@"animateFromLeftToRight" context:someView];
[UIView setAnimationDelegate:self];
[UIView setAnimationDidStopSelector:@selector(animateOffscreen:finished:context:)];

// slide the view to the right and offscreen
CGPoint center = [someView center];
center.x = CGRectGetMaxX(bounds) + CGRectGetWidth(someView) / 2.0;
[someView setCenter:center];

[UIView commitAnimations]; // commit
```

```
- (void)animateOffscreen:(NSString *)animationID
                finished:(NSNumber *)finished
                 context:(void *)context
{
    // ‘context’ is the view passed when the animation began
    UIView *view = (UIView *)context;

    // The view is now offscreen remove, and release it
    [view removeFromSuperview];
    [view release];
}
```

Sample Code

```
// make sure someView is not freed before the end callback
[someView retain];

// start up an animation and set the “did end” callback, pass “someView” as context
[UIView beginAnimations:@"animateFromLeftToRight" context:someView];
[UIView setAnimationDelegate:self];
[UIView setAnimationDidStopSelector:@selector(animateOffscreen:finished:context:)];

// slide the view to the right and offscreen
CGPoint center = [someView center];
center.x = CGRectGetMaxX(bounds) + CGRectGetWidth(someView) / 2.0;
[someView setCenter:center];

[UIView commitAnimations]; // commit
```

```
- (void)animateOffscreen:(NSString *)animationID
                finished:(NSNumber *)finished
                 context:(void *)context
{
    // ‘context’ is the view passed when the animation began
    UIView *view = (UIView *)context;

    // The view is now offscreen remove, and release it
    [view removeFromSuperview];
    [view release];
}
```

Sample Code

```
// make sure someView is not freed before the end callback
[someView retain];

// start up an animation and set the “did end” callback, pass “someView” as context
[UIView beginAnimations:@"animateFromLeftToRight" context:someView];
[UIView setAnimationDelegate:self];
[UIView setAnimationDidStopSelector:@selector(animateOffscreen:finished:context:)];

// slide the view to the right and offscreen
CGPoint center = [someView center];
center.x = CGRectGetMaxX(bounds) + CGRectGetWidth(someView) / 2.0;
[someView setCenter:center];

[UIView commitAnimations]; // commit
```

```
- (void)animateOffscreen:(NSString *)animationID
                finished:(NSNumber *)finished
                 context:(void *)context
{
    // ‘context’ is the view passed when the animation began
    UIView *view = (UIView *)context;

    // The view is now offscreen remove, and release it
    [view removeFromSuperview];
    [view release];
}
```

Sample Code

```
// make sure someView is not freed before the end callback
[someView retain];

// start up an animation and set the “did end” callback, pass “someView” as context
[UIView beginAnimations:@"animateFromLeftToRight" context:someView];
[UIView setAnimationDelegate:self];
[UIView setAnimationDidStopSelector:@selector(animateOffscreen:finished:context:)];

// slide the view to the right and offscreen
CGPoint center = [someView center];
center.x = CGRectGetMaxX(bounds) + CGRectGetWidth(someView) / 2.0;
[someView setCenter:center];

[UIView commitAnimations]; // commit
```

```
- (void)animateOffscreen:(NSString *)animationID
                finished:(NSNumber *)finished
                 context:(void *)context
{
    // ‘context’ is the view passed when the animation began
    UIView *view = (UIView *)context;

    // The view is now offscreen remove, and release it
    [view removeFromSuperview];
    [view release];
}
```

Sample Code

```
// make sure someView is not freed before the end callback
[someView retain];

// start up an animation and set the “did end” callback, pass “someView” as context
[UIView beginAnimations:@"animateFromLeftToRight" context:someView];
[UIView setAnimationDelegate:self];
[UIView setAnimationDidStopSelector:@selector(animateOffscreen:finished:context:)];

// slide the view to the right and offscreen
CGPoint center = [someView center];
center.x = CGRectGetMaxX(bounds) + CGRectGetWidth(someView) / 2.0;
[someView setCenter:center];

[UIView commitAnimations]; // commit
```

```
- (void)animateOffscreen:(NSString *)animationID
                finished:(NSNumber *)finished
                 context:(void *)context
{
    // ‘context’ is the view passed when the animation began
    UIView *view = (UIView *)context;

    // The view is now offscreen remove, and release it
    [view removeFromSuperview];
    [view release];
}
```

Sample Code

```
// make sure someView is not freed before the end callback
[someView retain];

// start up an animation and set the “did end” callback, pass “someView” as context
[UIView beginAnimations:@"animateFromLeftToRight" context:someView];
[UIView setAnimationDelegate:self];
[UIView setAnimationDidStopSelector:@selector(animateOffscreen:finished:context:)];

// slide the view to the right and offscreen
CGPoint center = [someView center];
center.x = CGRectGetMaxX(bounds) + CGRectGetWidth(someView) / 2.0;
[someView setCenter:center];

[UIView commitAnimations]; // commit
```

```
- (void)animateOffscreen:(NSString *)animationID
                finished:(NSNumber *)finished
                 context:(void *)context
{
    // ‘context’ is the view passed when the animation began
    UIView *view = (UIView *)context;

    // The view is now offscreen remove, and release it
    [view removeFromSuperview];
    [view release];
}
```

Sample Code

```
// make sure someView is not freed before the end callback
[someView retain];

// start up an animation and set the “did end” callback, pass “someView” as context
[UIView beginAnimations:@"animateFromLeftToRight" context:someView];
[UIView setAnimationDelegate:self];
[UIView setAnimationDidStopSelector:@selector(animateOffscreen:finished:context:)];

// slide the view to the right and offscreen
CGPoint center = [someView center];
center.x = CGRectGetMaxX(bounds) + CGRectGetWidth(someView) / 2.0;
[someView setCenter:center];

[UIView commitAnimations]; // commit
```

```
- (void)animateOffscreen:(NSString *)animationID
                finished:(NSNumber *)finished
                 context:(void *)context
{
    // ‘context’ is the view passed when the animation began
    UIView *view = (UIView *)context;

    // The view is now offscreen remove, and release it
    [view removeFromSuperview];
    [view release];
}
```

Sample Code

```
// make sure someView is not freed before the end callback
[someView retain];

// start up an animation and set the “did end” callback, pass “someView” as context
[UIView beginAnimations:@"animateFromLeftToRight" context:someView];
[UIView setAnimationDelegate:self];
[UIView setAnimationDidStopSelector:@selector(animateOffscreen:finished:context:)];

// slide the view to the right and offscreen
CGPoint center = [someView center];
center.x = CGRectGetMaxX(bounds) + CGRectGetWidth(someView) / 2.0;
[someView setCenter:center];

[UIView commitAnimations]; // commit
```

```
- (void)animateOffscreen:(NSString *)animationID
                finished:(NSNumber *)finished
                 context:(void *)context
{
    // ‘context’ is the view passed when the animation began
    UIView *view = (UIView *)context;

    // The view is now offscreen remove, and release it
    [view removeFromSuperview];
    [view release];
}
```

Sample Code

```
// make sure someView is not freed before the end callback
[someView retain];

// start up an animation and set the “did end” callback, pass “someView” as context
[UIView beginAnimations:@"animateFromLeftToRight" context:someView];
[UIView setAnimationDelegate:self];
[UIView setAnimationDidStopSelector:@selector(animateOffscreen:finished:context:)];

// slide the view to the right and offscreen
CGPoint center = [someView center];
center.x = CGRectGetMaxX(bounds) + CGRectGetWidth(someView) / 2.0;
[someView setCenter:center];

[UIView commitAnimations]; // commit
```

```
- (void)animateOffscreen:(NSString *)animationID
                finished:(NSNumber *)finished
                 context:(void *)context
{
    // ‘context’ is the view passed when the animation began
    UIView *view = (UIView *)context;

    // The view is now offscreen remove, and release it
    [view removeFromSuperview];
    [view release];
}
```

Sample Code

```
- (void)animateOffscreen:(NSString *)animationID
                finished:(NSNumber *)finished
                 context:(void *)context
{
    // ‘context’ is the view passed when the animation began
    UIView *view = (UIView *)context;

    // The view is now offscreen remove, and release it
    [view removeFromSuperview];
    [view release];
}
```

Sample Code

```
- (void)animateOffscreen:(NSString *)animationID
                finished:(NSNumber *)finished
                 context:(UIView *)view
{
    // 'context' is the view passed when the animation began
    UIView *view = (UIView *)context;

    // The view is now offscreen remove, and release it
    [view removeFromSuperview];
    [view release];
}
```

Demo

Simple Animation

Redraw During Animation

- Property animation handled without need for re-drawing
- Sometimes, you need to though
  - Think twice to be sure, redrawing is much more expensive
    - Why? – drawing code runs, multiple processes involved, …
- Most often wanted for redrawing in response to bounds changes
  - To redisplay whenever the bounds changes

```
view.contentMode = UIViewContentModeRedraw;
```

View Transitions

Full screen animation

Full View Transition Effects

- Perform advanced view transitions: flip, curl
  - Start an animation block
  - Tell animation block to animate view hierarchy changes

```
typedef enum {
    UIViewAnimationTransitionNone,
    UIViewAnimationTransitionFlipFromLeft,
    UIViewAnimationTransitionFlipFromRight,
    UIViewAnimationTransitionCurlUp,
    UIViewAnimationTransitionCurlDown,
} UIViewAnimationTransition;

+ (void)setAnimationTransition:(UIViewAnimationTransition)transition
                       forView:(UIView *)view cache:(BOOL)cache;
```

  - Add / remove a single subview in an animation block

Demo

View Transition

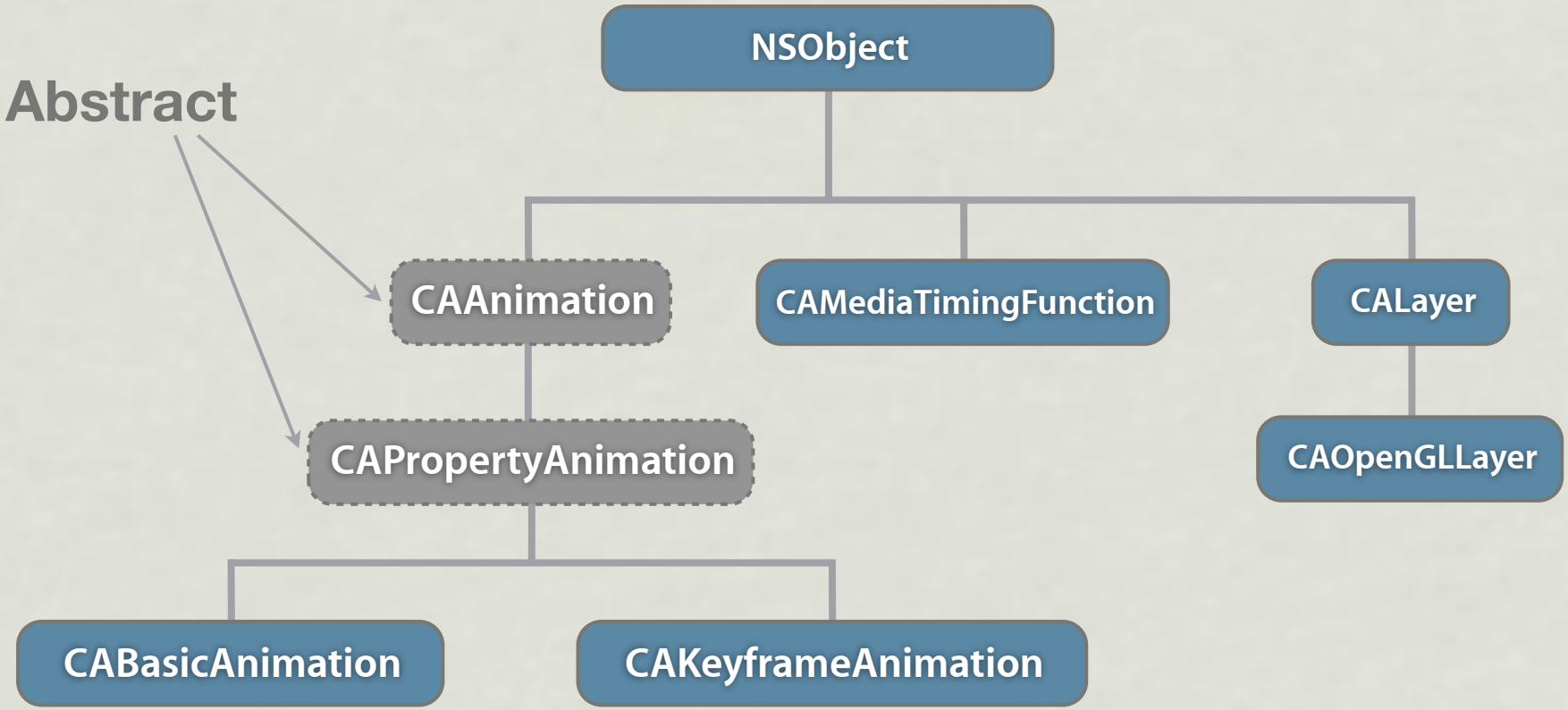
Core Animation

Features

Core Animation

- Discussion of features and capabilities only
- API details can be found online
- Prerequisites for using Core Animation directly
  - Key–Value Coding Understanding
  - CG / Quartz 2D Knowledge

Important Classes



Features

- Abstract Base Classes
  - `CAAnimation` is the abstract base class of all animations
  - `CAPropertyAnimation` allows animation of KVC properties
- Animations work on any `CALayer`
  - `CALayers`, `CAOpenGLLayer`
- Simple Animation
  - `CABasicAnimation` provides interpolation for layer properties
    - Basic – one property animated to one goal value
  - UIKit animations are essentially basic animations

Features

- Key Frame Animations
  - One property animated through multiple key-frames
  - Specified by the following
    - array of values for each animation frame, or a `CGPathRef`
    - optional key frame times
    - optional timing functions for each segment
- Customizing Timing
  - `CAAnimation` allows you to set a custom pacing
  - `CAKeyframeAnimation` allows custom pacing for each segment

Animatable Properties

- Colors – background, border, shadow
- Geometry – bounds, position, z position
- Shadow – color, opacity, offset
- Transforms – 3D transforms
- Filters – Compositing filters and other advanced effects
- Layer Opacity
- More…

Examples

Core Animation


1:40 AM
Mailboxes
Inbox
Drafts
Sent
Trash
Apple Mail To Do
Drafts (Home)
Moving Stuff

Event Handling

UIView and UITouch

Handling Touches

- To handle events, override

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

- “touches” has those associated with a particular phase
  - E.g. touchesBegan touches are those that just touched down
  - To get **all** active touches

```
// UIEvent
- (NSSet *)allTouches;
- (NSSet *)touchesForView:(UIView *)view;
- (NSSet *)touchesForWindow:(UIWindow *)window;
```

Handling Touches

- Must declare multi-touch awareness

```
view.multipleTouchEnabled = YES; // defaults to NO...
```

- Sometimes want to ignore taps while animations and state changes are “inflight”

```
view.userInteractionDisabled = YES;
```

Handling Touches

- Must declare multi-touch awareness

```
view.multipleTouchEnabled = YES; // defaults to NO...
```

- Sometimes want to ignore taps while animations and state changes are “inflight”

```
view.userInteractionDisabled = YES;
```

Handling Touches

- Must declare multi-touch awareness

```
view.multipleTouchEnabled = YES; // defaults to NO...
```

- Sometimes want to ignore taps while animations and state changes are “inflight”

```
view.userInteractionDisabled = YES;
```

```
view.userInteractionDisabled = NO;
```

Handling Touches

- Sometimes want to ignore taps while tracking in other views

```
view.exclusiveTouch = YES;
```

- Example
  - User presses fast-forward button
  - What happens if they press rewind in the middle?
    - Does this make sense?
    - Is your code ready to function properly when one state is interrupted

Touch Point

- UITouch
  - Provides the touched view
  - Provides the centroid of the touch location in the touched view's coordinates
- Finding where the touch happened

```
UITouch *touch = ...;
CGPoint viewRelativePt = [touch locationInView:self];
```

- Determining single vs. double tap

```
// touch down within a certain point within a certain amount of time
@property(nonatomic,readonly) NSUInteger tapCount;
```

Handling Tap Gesture

```
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
    // Nothing to do
}

- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event {
    // Nothing to do
}

- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event {
    for (UITouch *touch in touches) {
        if (touch.tapCount == 1) {
            // Handle the tap
        }
    }
}

- (void)touchesCancelled:(NSSet *)touches withEvent:(UIEvent *)event {
    // Nothing to do
}
```

Single And Double Tap

- Slightly more involved, but not much
- Trick
  - You don’t know if a second tap is going to come
  - Instead of handling the single tap, schedule it to be handled
  - If a double tap comes in, cancel the single handler

Mail Bag!

- Email Chuck Pisula – chuck@cs.umd.edu – “Mailbag: ... ”
  - Topics you’d like discussed
    - In greater details?
    - Stuff that hasn’t been covered
  - What isn’t clear?

- I’ll try to address whatever possible, and may use as topics in upcoming lectures…

Reading

- iPhone Application Programming Guide
  - Animating Views – p.67 - 70
  - Touch Events – p.81 - 94

    ** Especially the section on complex multi-touch sequences
- Core Animation Programming Guide
  - What is Core Animation? – p.13 - 19
  - Layer Style Properties – p. 59 - 65