Custom Transitions
Using View Controllers

New capabilities, APIs and enhancements

Session 218
Bruce D. Nilo
View Controller Mechanic

These are confidential sessions—please refrain from streaming, blogging, or taking pictures
Roadmap
Roadmap

• New animation tools
• Custom view controller transitions
• Interactive view controller transitions
• Canceling and coordinating transitions
Roadmap

New animation tools
Roadmap

New animation tools

• Quick review of the block based UIView animation API
Roadmap

New animation tools

• Quick review of the block based UIView animation API
• Spring animations
Roadmap

New animation tools

• Quick review of the block based UIView animation API
• Spring animations
• Key-frame animations
New animation tools

• Quick review of the block based UIView animation API
• Spring animations
• Key-frame animations
• UIKit Dynamics
Roadmap

Custom view controller transitions
Roadmap

Custom view controller transitions

• Which transitions can be customized?
  ▪ Presentations and dismissals
  ▪ UITabBarController
  ▪ UINavigationController
  ▪ UICollectionViewController layout-to-layout transitions
Roadmap

Custom view controller transitions

• Which transitions can be customized?
  ▪ Presentations and dismissals
  ▪ UITabBarController
  ▪ UINavigationController
  ▪ UICollectionViewController layout-to-layout transitions

• What is a transition?
  ▪ Anatomy and generalizations
Roadmap

Custom view controller transitions

• Which transitions can be customized?
  ▪ Presentations and dismissals
  ▪ UITabBarController
  ▪ UINavigationController
  ▪ UICollectionViewController layout-to-layout transitions

• What is a transition?
  ▪ Anatomy and generalizations

• API discussion with some examples
Roadmap

Interactive view controller transitions
Roadmap

Interactive view controller transitions

• Adding interactivity to custom transitions
Roadmap

Interactive view controller transitions

- Adding interactivity to custom transitions
- Special support for UICollectionViews
  - UICollectionViewTransitionLayout
Roadmap

Interactive view controller transitions

• Adding interactivity to custom transitions
• Special support for UICollectionViews
  • UICollectionViewTransitionLayout
• Canceling transitions
Roadmap
Interactive view controller transitions

• Adding interactivity to custom transitions
• Special support for UICollectionViews
  ▪ UICollectionViewTransitionLayout
• Canceling transitions
• UITransitionCoordinator
  ▪ Animating alongside transitions
  ▪ Specifying a completion handler
    ▪ This can be used for all UINavigationController transitions!
New UIView Animation APIs
Create compelling, custom, view controller transitions
New UIView Animation APIs
Quick review: Existing UIView block based API
New UIView Animation APIs

Quick review: Existing UIView block based API

+ (void) beginAnimations:context:
+ (void) commitAnimations
New UIView Animation APIs

Quick review: Existing UIView block based API

+ (void) beginAnimations:context:
+ (void) commitAnimations

+ (void)-animateWithDuration:(NSTimeInterval)duration
delay:(NSTimeInterval)delay
options:(UIViewAnimationOptions)options
animations:(void (^)(void))animations
completion:(void (^)(BOOL finished))completion;
New UIView Animation APIs
Quick review: Relationship to core animation

```objectivec
[UIView animationWithDuration:delay:options:animations:^{

} completion: nil];
```
New UIView Animation APIs

Quick review: Relationship to core animation

```swift
[UIView animationWithDuration:delay:options:animations:^{

} completion: nil];
```
New UIView Animation APIs

Quick review: Relationship to core animation

```swift
[UIView animationWithDuration:delay:options:animations:^{

} completion: nil];
```
New UIView Animation APIs
Quick review: Relationship to core animation

**[UIView animationWithDuration:delay:options:animations:^{**

{ completion: nil};
New UIView Animation APIs

Relationship to core animation

```swift
[UIView animationWithDuration:...animations:^{
    // Update properties
    UIView
    CALayer
}
} completion: nil];
```
New UIView Animation APIs

Relationship to core animation

```swift
UIView animationWithDuration:...animations:^{
    // Update properties
    UIView
    CALayer
    CAAnimations
    added to layers
}

} completion: nil];
```
New UIView Animation APIs

UIView block based API

• Disabling and enabling animations
  + (void)setAnimationsEnabled:(BOOL)
New UIView Animation APIs

UIView block based API

+ (void)performWithoutAnimation:(void ^)(void))actions;
New UIView Animation APIs

UIView block based API

+ (void)performWithoutAnimation:(void (^)(void))actions;
New UIView Animation APIs

Spring animations
New UIButton Animation APIs

Spring animations

• Two new parameters
  ▪ DampingRatio
    ▪ 1.0 >= r > 0.0
  ▪ Initial Spring Velocity

• Composes nicely with other UIButton animation methods
New UIView Animation APIs

Spring animations

+ (void)animateWithDuration:(NSTimeInterval)duration
delay:(NSTimeInterval)delay
usingSpringWithDamping:(CGFloat)dampingRatio
initialSpringVelocity:(CGFloat)velocity
options:(UIViewAnimationOptions)options
animations:(void (^)(void))animations
completion:(void (^)(BOOL finished))completion;
New UIView Animation APIs

Spring animations

+ (void)animateWithDuration:(NSTimeInterval)duration
delay:(NSTimeInterval)delay
  usingSpringWithDamping:(CGFloat)dampingRatio
  initialSpringVelocity:(CGFloat)velocity
  options:(UIViewAnimationOptions)options
  animations:(void (^)(void))animations
  completion:(void (^)(BOOL finished))completion;
New UIView Animation APIs

Key-frame animations
New UIView Animation APIs

Key-frame animations

- `animateKeyframesWithDuration...`
  - is to `CAKeyframeAnimation`
New UIView Animation APIs

Key-frame animations

• `animateKeyframesWithDuration…`
  ▪ is to CAKeyframeAnimation
• `as animateWithDuration…`
  ▪ is to CABasicAnimation
New UIView Animation APIs

Key-frame animations

• `animateKeyframesWithDuration…`
  ▪ is to CAKeyframeAnimation
• `as animateWithDuration…`
  ▪ is to CABasicAnimation
• Specify keyframes within the animation block
  ▪ Options augmented to include calculation mode
New UIView Animation APIs

Key-frame animations

• animateKeyframesWithDuration…
  ▪ is to CAKeyframeAnimation
• as animateWithDuration…
  ▪ is to CABasicAnimation
• Specify keyframes within the animation block
  ▪ Options augmented to include calculation mode
• Composes nicely with other UIView animation methods
New View Based Animation APIs

Key-frame animations

+ (void)animateKeyframesWithDuration:(NSTimeInterval)duration
delay:(NSTimeInterval)delay
options:(UIViewKeyframeAnimationOptions)options
animations:(void (^)(void))animations
completion:(void (^)(BOOL finished))completion;

+ (void)addKeyframeWithRelativeStartTime:(double)frameStartTime
relativeDuration:(double)frameDuration
animations:(void (^)(void))animations
New View Based Animation APIs

Key-frame animations

+ (void)animateKeyframesWithDuration:(NSTimeInterval)duration
delay:(NSTimeInterval)delay
options:(UIViewKeyframeAnimationOptions)options
animations:(void (^)(void))animations
completion:(void (^)(BOOL finished))completion;

+ (void)addKeyframeWithRelativeStartTime:(double)frameStartTime
relativeDuration:(double)frameDuration
animations:(void (^)(void))animations
New View Based Animation APIs

Key-frame animations

+ (void)animateKeyframesWithDuration:(NSTimeInterval)duration
delay:(NSTimeInterval)delay
options:(UIViewKeyframeAnimationOptions)options
animations:(void (^)(void))animations
completion:(void (^)(BOOL finished))completion;

+ (void)addKeyframeWithRelativeStartTime:(double)frameStartTime
relativeDuration:(double)frameDuration
animations:(void (^)(void))animations
New View Based Animation APIs

Key-frame animations

```swift
[UIView animateKeyframesWithDuration: .35
delay: 0.0
options:0
animations:^{
    [UIView addKeyframe... animations: ^{...}];
    [UIView addKeyframe... animations:^{...}];
    [UIView addKeyframe... animations:^{
        [someView setPosition:...];
        // etc.
    }];
}
completion:^(BOOL finished) {...}];
```
Key-frame animations

```objective-c
[UIView animateKeyframesWithDuration: .35
delay: 0.0
options:0
animations:^{
    [UIView addKeyframe... animations: ^{...}];
    [UIView addKeyframe... animations: ^{...}];
    [UIView addKeyframe... animations: ^{
        [someView setPosition:...];
        // etc.
    }];
}
completion:^(BOOL finished) {...}];
```
Key-frame animations

```swift
[UIView animateKeyframesWithDuration: .35
delay: 0.0
options:0
animations:^{
   [UIView addKeyframe... animations: ^{...}];
   [UIView addKeyframe... animations:^{...}];
   [UIView addKeyframe... animations:^{
       [someView setPosition:...];
       // etc.
   }];
}
completion:^:(BOOL finished) {...}];
```
Improved and Simplified Snapshot API
UIView snapshots
Improved and Simplified Snapshot API

UIView snapshots

• Snapshot API
  - (UIView *)[UIView snapshotView]
  - (UIView *)[UIView resizableSnapshotViewFromRect:(CGRect)rect
    withCapInsets:(UIEdgeInsets)capInsets]
Improved and Simplified Snapshot API

**UIView snapshots**

- **Snapshot API**
  - (UIView *)[UIView snapshotView]
  - (UIView *)[UIView resizableSnapshotViewFromRect:(CGRect)rect withCapInsets:(UIEdgeInsets)capInsets]

- Creating snapshots from snapshots is supported
New View Based Animation APIs
UIKit Dynamics
New View Based Animation APIs

UIKit Dynamics

• Distinct from UIView animation APIs
New View Based Animation APIs

UIKit Dynamics

• Distinct from UIView animation APIs
  ▪ Compatible with the new transitioning APIs
  ▪ More in this afternoon’s talk
Customizing Your View
Controller Transitions

It’s easy to use
Custom View Controller Transitions

Which transitions can be customized?
Custom View Controller Transitions

Which transitions can be customized?

• Presentations and dismissals
  ▪ Supported presentation styles
    ▪ UIModalPresentationFullScreen
    ▪ UIModalPresentationCustom
Custom View Controller Transitions

Which transitions can be customized?

• Presentations and dismissals
  ▪ Supported presentation styles
    ▪ UIModalPresentationFullScreen
    ▪ UIModalPresentationCustom

The from view controller is not removed from the window hierarchy
Custom View Controller Transitions

Which transitions can be customized?

- Presentations and dismissals
  - Supported presentation styles
    - UIModalPresentationFullScreen
    - UIModalPresentationCustom

The from view controller is not removed from the window hierarchy

```swift
UIViewController *vc = ...;
id <UIViewControllerTransitioningDelegate> transitioningDelegate;
vc.modalPresentationStyle = UIModalPresentationCustom;
[vc setTransitioningDelegate: transitioningDelegate];
[self presentViewController:vc animated: YES completion: nil];
```
Custom View Controller Transitions
Which transitions can be customized?

- Presentations and dismissals
  - Supported presentation styles
    - UIModalPresentationFullScreen
    - UIModalPresentationCustom

The from view controller is not removed from the window hierarchy

```swift
UIViewController *vc = ...;
id <UIViewControllerTransitioningDelegate> transitioningDelegate;
vc.modalPresentationStyle = UIModalPresentationCustom;
[vc setTransitioningDelegate: transitioningDelegate];
[self presentViewController:vc animated: YES completion: nil];
```
Custom View Controller Transitions

Which transitions can be customized?
Custom View Controller Transitions

Which transitions can be customized?

• UITabBarController

```
setSelectedViewController:(UIViewController *)vc;
setSelectedIndex:(NSUInteger)index;
```
Custom View Controller Transitions
Which transitions can be customized?

-UITabBarController
  
  ```swift
  setSelectedViewController:(UIViewController *)vc;
  setSelectedIndex:(NSUInteger)idx;
  
  NSUInteger secondTab = 1;
  self.delegate = tabBarControllerDelegate;
  [self setSelectedIndex:secondTab];
  ```
Custom View Controller Transitions

Which transitions can be customized?
Custom View Controller Transitions

Which transitions can be customized?

• UINavigationController
  pushViewController:animated:
  popViewControllerAnimated:
  setViewControllers:animated:
Custom View Controller Transitions

Which transitions can be customized?

• UINavigationController
  pushViewController:animated:
  popViewControllerAnimated:
  setViewControllers:animated:

  self.delegate = navigationControllerDelegate;
  [self pushViewController:vc animated:YES];
Custom View Controller Transitions

UIKitViewController meets UICollectionViewViewController
Custom View Controller Transitions

UINavigationController meets UICollectionViewViewController

• Layout-to-layout navigation transitions
Custom View Controller Transitions

UIApplicationDelegateViewController meets UICollectionViewViewController

• Layout-to-layout navigation transitions

UICollectionViewLayout *layout1, *layout2, *layout3;
UICollectionViewController *cvc1, *cvc2, *cvc3;
cvc1 = [cvc1 initWithCollectionViewLayout:layout1];
...

[nav pushViewController:cvc1 animated:YES]
cvc2.useLayoutToLayoutNavigationTransitions = YES;
cvc3.useLayoutToLayoutNavigationTransitions = YES;
[nav pushViewController:cvc2 animated:YES];
[nav pushViewController:cvc3 animated:YES];
[nav popViewControllerAnimated:YES];
Demo

Some examples of custom transitions
Customizing Your View
Controller Transitions
Concepts and APIs
Custom View Controller Transitions

The anatomy of a transition

Start State

End State

Legend

Objects:
- View Controller (orange)
- View (blue)

Relationships:
- VC-View (orange arrow)
- Containment (yellow arrow)
- Superview (blue arrow)
Custom View Controller Transitions
The anatomy of a transition

Intermediate State
Custom View Controller Transitions

The anatomy of a transition

Intermediate State
Custom View Controller Transitions
The anatomy of a transition

Intermediate State
Custom View Controller Transitions
The anatomy of a transition

Intermediate State
Custom View Controller Transitions
The anatomy of a transition

Intermediate State
Custom View Controller Transitions
The anatomy of a transition

End State
Custom View Controller Transitions

The anatomy of a transition
Custom View Controller Transitions

The anatomy of a transition

• Start state
  • Consistent view controller hierarchy and view hierarchy
Custom View Controller Transitions
The anatomy of a transition

• Start state
  ▪ Consistent view controller hierarchy and view hierarchy
• User or programmatic transition commences
Custom View Controller Transitions

The anatomy of a transition

• Start state
  ▪ Consistent view controller hierarchy and view hierarchy
• User or programmatic transition commences
• Internal structures are updated, callbacks made, etc.
Custom View Controller Transitions

The anatomy of a transition

• Start state
  ▪ Consistent view controller hierarchy and view hierarchy
• User or programmatic transition commences
• Internal structures are updated, callbacks made, etc.
• Container view, and start and final view positions are computed
Custom View Controller Transitions
The anatomy of a transition

• Start state
  ▪ Consistent view controller hierarchy and view hierarchy
• User or programmatic transition commences
• Internal structures are updated, callbacks made, etc.
• Container view, and start and final view positions are computed
• Optional animation to end state view hierarchy is run
Custom View Controller Transitions
The anatomy of a transition

• Start state
  ▪ Consistent view controller hierarchy and view hierarchy
• User or programmatic transition commences
• Internal structures are updated, callbacks made, etc.
• Container view, and start and final view positions are computed
• Optional animation to end state view hierarchy is run
• Animation completes
  ▪ Internal structures are updated, callbacks made, etc.
Custom View Controller Transitions

The anatomy of a transition

• Start state
  ▪ Consistent view controller hierarchy and view hierarchy
• User or programmatic transition commences
• Internal structures are updated, callbacks made, etc.
• Container view, and start and final view positions are computed
• Optional animation to end state view hierarchy is run
• Animation completes
  ▪ Internal structures are updated, callbacks made, etc.
• End State
  ▪ Consistent view controller hierarchy and view hierarchy
Custom View Controller Transitions
The anatomy of a transition

• Start state
  ▪ Consistent view controller hierarchy and view hierarchy
• User or programmatic transition commences
• Internal structures are updated, callbacks made, etc.
• Container view, and start and final view positions are computed
• Optional animation to end state view hierarchy is run
• Animation completes
  ▪ Internal structures are updated, callbacks made, etc.
• End State
  ▪ Consistent view controller hierarchy and view hierarchy
Custom View Controller Transitions

@protocol UIViewControllerContextTransitioning <NSObject>

// The view in which the animated transition should take place.
- (UIView *)containerView;

// Two keys for the method below are currently defined by the system
// UITransitionContextToViewControllerKey, and
// UITransitionContextFromViewControllerKey.

- (UIViewController *) viewControllerForKey:(NSString *)key;
- (CGRect) initialFrameForViewController:(UIViewController *)vc;
- (CGRect) finalFrameForViewController:(UIViewController *)vc;

// This MUST be called whenever a transition completes (or is cancelled.)
- (void)completeTransition:(BOOL)didComplete;
...
@end
Custom View Controller Transitions

@protocol UIViewControllerContextTransitioning <NSObject>

// The view in which the animated transition should take place.
- (UIView *)containerView;

// Two keys for the method below are currently defined by the system
// UITransitionContextToViewControllerKey, and
// UITransitionContextFromViewControllerKey.
- (UIViewController *)viewControllerForKey:(NSString *)key;
- (CGRect) initialFrameForViewController:(UIViewController *)vc;
- (CGRect) finalFrameForViewController:(UIViewController *)vc;

// This MUST be called whenever a transition completes (or is cancelled.)
- (void)completeTransition:(BOOL)didComplete;
...
@end
@protocol UIViewControllerContextTransitioning <NSObject>

// The view in which the animated transition should take place.
- (UIView *)containerView;

// Two keys for the method below are currently defined by the system
// UITransitionContextToViewControllerKey, and
// UITransitionContextFromViewControllerKey.

- (UIViewController *) viewControllerForKey:(NSString *)key;
- (CGRect) initialFrameForViewController:(UIViewController *)vc;
- (CGRect) finalFrameForViewController:(UIViewController *)vc;

// This MUST be called whenever a transition completes (or is cancelled.)
- (void)completeTransition:(BOOL)didComplete;
...
@end
@protocol UIViewControllerContextTransitioning <NSObject>

// The view in which the animated transition should take place.
- (UIView *)containerView;

// Two keys for the method below are currently defined by the system
// UITransitionContextToViewControllerKey, and
// UITransitionContextFromViewControllerKey.
- (UIViewController *)viewControllerForKey:(NSString *)key;
- (CGRect) initialFrameForViewController:(UIViewController *)vc;
- (CGRect) finalFrameForViewController:(UIViewController *)vc;

// This MUST be called whenever a transition completes (or is cancelled.)
- (void)completeTransition:(BOOL)didComplete;

@end
Custom View Controller Transitions
<UIViewControllerAnimatedTransitioning>

@protocol UIViewControllerAnimatedTransitioning <NSObject>
-
(NSTimeInterval)transitionDuration:(id <UIViewControllerContextTransitioning>)ctx;

// This method can only be a nop if the transition is interactive and not a percentDriven interactive transition.
- (void)animateTransition:(id <UIViewControllerContextTransitioning>)ctx;

@optional

// This is a convenience and if implemented will be invoked by the system when the transition context's completeTransition: method is invoked.
- (void)animationEnded:(BOOL)transitionCompleted;
@end
Custom View Controller Transitions
<UIViewControllerAnimatedTransitioning>

@protocol UIViewControllerAnimatedTransitioning <NSObject>

- (NSTimeInterval)transitionDuration:(id <UIViewControllerContextTransitioning>)ctx;

// This method can only be a nop if the transition is interactive and not a percentDriven interactive transition.
- (void)animateTransition:(id <UIViewControllerContextTransitioning>)ctx;

@optional

// This is a convenience and if implemented will be invoked by the system when the transition context's completeTransition: method is invoked.
- (void)animationEnded:(BOOL)transitionCompleted;

@end
Custom View Controller Transitions

@protocol UIViewControllerAnimatedTransitioning <NSObject>

- (NSTimeInterval)transitionDuration:(id <UIViewControllerContextTransitioning>)ctx;

// This method can only be a nop if the transition is interactive and not a percentDriven interactive transition.
- (void)animateTransition:(id <UIViewControllerContextTransitioning>)ctx;

@optional

// This is a convenience and if implemented will be invoked by the system when the transition context's completeTransition: method is invoked.
- (void)animationEnded:(BOOL) transitionCompleted;

@end
Custom View Controller Transitions
The anatomy of a transition
Custom View Controller Transitions

The anatomy of a transition

(id <UIViewControllerContextTransitioning>) context;
[animationController animateTransition: context];
Custom View Controller Transitions

The anatomy of a transition

(id <UIViewControllerContexTransitioning>) context;
[animationController animateTransition: context];
Custom View Controller Transitions
The anatomy of a transition

(id <UIViewControllerContexTransitioning>) context;
[context completeTransition: YES];
Custom View Controller Transitions

The anatomy of a transition
Custom View Controller Transitions
Wiring it all together
Custom View Controller Transitions
Wiring it all together

- Animation and interaction controllers are vended by delegates

  `<UIViewControllerTransitioningDelegate>`
  `<UINavigationControllerDelegate>`
  `<UITabBarControllerDelegate>`
Custom View Controller Transitions

Wiring it all together

• Animation and interaction controllers are vended by delegates

<UIViewControllerTransitioningDelegate>
<UINavigationControllerDelegate>
<UITabBarControllerDelegate>

• Animation controllers conform to a protocol

<UIViewControllerAnimatedTransitioning>
Custom View Controller Transitions

Wiring it all together

• Animation and interaction controllers are vended by delegates
  `<UIViewControllerTransitioningDelegate>`
  `<UINavigationControllerDelegate>`
  `<UITabBarControllerDelegate>`

• Animation controllers conform to a protocol
  `<UIViewControllerAnimatedTransitioning>`

• Interaction controllers conform to a protocol
  `<UIViewControllerInteractiveTransitioning>`
Custom View Controller Transitions

Wiring it all together

• Animation and interaction controllers are vended by delegates
  <UIViewControllerTransitioningDelegate>
  <UINavigationControllerDelegate>
  <UITabBarControllerDelegate>

• Animation controllers conform to a protocol
  <UIViewControllerAnimatedTransitioning>

• Interaction controllers conform to a protocol
  <UIViewControllerInteractiveTransitioning>

• A system object passed to the controllers conforms to
  <UIViewControllerContextTransitioning>
Custom View Controller Transitions

Start of a custom presentation

Presented Controller

Presenting Controller

transitionDelegate
Custom View Controller Transitions

Start of a custom presentation

Presented Controller

setTransitioningDelegate:

<UIViewControllerTransitioningDelegate>
(transitionDelegate)

Presenting Controller

transitionDelegate
Custom View Controller Transitions

Start of a custom presentation

- Presented Controller
  - setTransitioningDelegate:
- Presenting Controller
  - presentViewController:
- transitionDelegate

<UIViewControllerTransitioningDelegate>
(transitionDelegate)

Presented Controller

Presented Controller
Custom View Controller Transitions

Start of a custom presentation

Presented Controller

- setTransitioningDelegate:

Presenting Controller

- presentViewController:

transitionDelegate

- animationControllerForPresentingController: presentedController:sourceController:

Presented Controller

- <UIViewControllerTransitioningDelegate> (transitionDelegate)

- <UIViewControllerAnimatedTransitioning> (animationController)
Custom View Controller Transitions
End of a custom presentation
Custom View Controller Transitions

End of a custom presentation

animationController

transitionDuration:

<UIViewControllerContextTransitioning>(context)

context
Custom View Controller Transitions

End of a custom presentation

animationController

transitionDuration:

<UIViewControllerContextTransitioning>(context)

animationController

animateTransition:

<UIViewControllerContextTransitioning>(context)

context
Custom View Controller Transitions

End of a custom presentation

<table>
<thead>
<tr>
<th>animationController</th>
<th>transitionDuration:</th>
</tr>
</thead>
<tbody>
<tr>
<td>animationController</td>
<td>animateTransition:</td>
</tr>
<tr>
<td>context</td>
<td>completeTransition:</td>
</tr>
</tbody>
</table>

```
<UIViewControllerContextTransitioning>
<context>
transitionDuration: [animationController]
animateTransition: [animationController]
completeTransition: [context]
```
Custom View Controller Transitions

Pseudo-code of a custom presentation
Custom View Controller Transitions

Pseudo-code of a custom presentation

```swift
id <UIViewControllerTransitioningDelegate> delegate;
[presentedController setTransitioningDelegate:delegate];
[presentedController setModalPresentationStyle: UIModalPresentationCustom];
[self presentViewController:presentedController animated: YES completion:nil];
```
Custom View Controller Transitions

Pseudo-code of a custom presentation

```swift
id <UIViewControllerTransitioningDelegate> delegate;
[presentedController setTransitioningDelegate:delegate];
[presentedController setModalPresentationStyle: UIModalPresentationCustom];
[self presentViewController:presentedController animated: YES completion:nil];

id <UIViewControllerAnimatedTransitioning> animationController =
[delegate animationControllerForPresentedController: presented
    presentingController: presenter
    sourceController: target];
```
Custom View Controller Transitions

Pseudo-code of a custom presentation

id <UIViewControllerContextTransitioning>)ctx;
NSTimeInterval duration = [animationController transitionDuration:ctx];
[animationController animateTransitionTransition:ctx];
Custom View Controller Transitions

Pseudo-code of a custom presentation

```cpp
id <UIViewControllerContextTransitioning>)ctx;
NSTimeInterval duration = [animationController transitionDuration:ctx];
[animationController animateTransition:ctx];
```
```swift
- (void)animateTransition:(id <UIViewControllerContextTransitioning>ctx) {
    UIView *inView = [ctx containerView];
    UIView *toView = [[ctx viewControllerForKey:...] view];
    UIView *fromView = [[ctx viewControllerForKey: ...];
    CGSize size = toEndFrame.size;

    if(self.isPresentation) {
        ...
        [inView addSubview: toView];
    } else {
        ...
        [inView insertSubview:toView belowSubview: [fromVC view]];
    }

    [UIView animateWithDuration: self.transitionDuration animations: ^ {
        if(self.isPresentation) {
            toView.center = newCenter;
            toView.bounds = newBounds;
        } else {
            ...
        } completion: ^(BOOL finished) { [ctx completeTransition: YES];}}];
```
Custom View Controller Transitions

Pseudo-code of a custom presentation

- (void/animateTransition:(id <UIViewControllerContextTransitioning>ctx { 
  UIView *inView = [ctx containerView];
  UIView *toView = [[ctx viewControllerForKey:...] view];
  UIView *fromView = [[ctx viewControllerForKey:...];
  CGSize size = toEndFrame.size;

  if(self.isPresentation) {
    ... 
    [inView addSubview: toView];
  } else {
    ...
    [inView insertSubview:toView belowSubview: [fromVC view]];
  }

  [UIView animateWithDuration: self.transitionDuration animations: ^ {
    if(self.isPresentation) {
      toView.center = newCenter;
      toView.bounds = newBounds;
    } else {
      ...
    } completion: ^(BOOL finished) { [ctx completeTransition: YES];}];
}
Custom View Controller Transitions
Pseudo-code of a custom presentation

-(void)animateTransition:(id <UIViewControllerContextTransitioning>CTX {  
    UIView *InView = [CTX containerView];
    UIView *ToView = [[CTX viewControllerForKey:... view];
    UIView *FromView = [[CTX viewControllerForKey: ...];
    CGSize size = toEndFrame.size;
    if(self.isPresentation) {
        ...
        [InView addSubview: toView];
    } else {
        ...
        [InView insertSubview:toView belowSubview: [fromVC view]];
    }

    [UIView animateWithDuration: self.transitionDuration animations: ^ {
        if(self.isPresentation) {
            toView.center = newCenter;
            toView.bounds = newBounds;
        } else {
            ...
        } completion: ^(BOOL finished) { [CTX completeTransition: YES];};
    }];
}
Custom View Controller Transitions

Pseudo-code of a custom presentation

```cpp
id <UIViewControllerContextTransitioning>)ctx;
[ctx completeTransition:YES];
```
@protocol UIViewControllerTransitioningDelegate <NSObject>

@optional

- (id <UIViewControllerAnimatedTransitioning>)
  animationControllerForPresentedController:(UIVC *)presented
  presentingController:(UIVC *)presenting
  sourceController:(UIVC *)source;

- (id <UIViewControllerAnimatedTransitioning>)
  animationControllerForDismissedController:(UIVC *)dismissed;

- (id <UIViewControllerInteractiveTransitioning>)
  interactionControllerForPresentation:(id <UIViewControllerAnimatedTransitioning>)a;

- (id <UIViewControllerInteractiveTransitioning>)
  interactionControllerForDismissal:(id <UIViewControllerAnimatedTransitioning>)a;

@end
UIViewControllerTransitioningDelegate

@protocol UIViewControllerTransitioningDelegate <NSObject>

@optional

- (id <UIViewControllerAnimatedTransitioning>) animationControllerForPresentedController:(UIVC *)presented
  presentingController:(UIVC *)presenting
  sourceController:(UIVC *)source;

- (id <UIViewControllerAnimatedTransitioning>) animationControllerForDismissedController:(UIVC *)dismissed;

- (id <UIViewControllerInteractiveTransitioning>) interactionControllerForPresentation:(id <UIViewControllerAnimatedTransitioning>) a;

- (id <UIViewControllerInteractiveTransitioning>) interactionControllerForDismissal:(id <UIViewControllerAnimatedTransitioning>) a;

@end
Custom View Controller Transitions

UIViewControllerTransitioningDelegate

@protocol UIViewControllerTransitioningDelegate <NSObject>

@optional

- (id <UIViewControllerAnimatedTransitioning>)
  animationControllerForPresentedController:(UIVC *)presented
  presentingController:(UIVC *)presenting
  sourceController:(UIVC *)source;

- (id <UIViewControllerAnimatedTransitioning>)
  animationControllerForDismissedController:(UIVC *)dismissed;

- (id <UIViewControllerInteractiveTransitioning>)
  interactionControllerForPresentation:(id <UIViewControllerAnimatedTransitioning>)a;

- (id <UIViewControllerInteractiveTransitioning>)
  interactionControllerForDismissal:(id <UIViewControllerAnimatedTransitioning>)a;

@end
Custom View Controller Transitions

UIViewControllerTransitioningDelegate

@interface UIViewController(CustomTransitioning)

@property (nonatomic,retain) id <UIViewControllerTransitioningDelegate>transitionDelegate;

@end
Custom View Controller Transitions
UIViewControllerTransitioningDelegate

@interface UIViewController(CustomTransitioning)

@property (nonatomic,retain) id <UIViewControllerTransitioningDelegate>transitionDelegate;

@end
Custom View Controller Transitions

- (id <UIViewControllerAnimatedTransitioning>)navigationController: (UINavigationController *)nc
  animationControllerForOperation: (UINavigationControllerOperation)op
  fromViewController: (UIViewController *)fromVC
  toViewController: (UIViewController *)toVC;

- (id <UIViewControllerInteractiveTransitioning>)navigationController: (UINavigationController *)nc
  interactionControllerForAnimationController: (id <UIViewControllerAnimatedTransitioning>)a;
Custom View Controller Transitions
UITabBarControllerDelegate Extensions

- (id <UIViewControllerAnimatedTransitioning>)tabBarController: (UITABC *)tbc
  animationControllerForTransitionFromViewController:(UIVC *)fromVC
toViewController:(UIVC *)toVC;

- (id <UIViewControllerInteractiveTransitioning>)tabBarController: (UITABC *)tbc
  interactionControllerForAnimationController: (id <UIViewControllerAnimatedTransitioning>)a;
Custom View Controller Transitions

Responsibilities of the animation controller
Custom View Controller Transitions
Responsibilities of the animation controller

• Implementation of `animateTransition:` and `transitionDuration:`
  ▪ Insertion of “to” view controller’s view into the container view
Custom View Controller Transitions

Responsibilities of the animation controller

• Implementation of `animateTransition:` and `transitionDuration:`
  ▪ Insertion of “to” view controller’s view into the container view

• When the transition animation completes
  ▪ The “to” and “from” view controller’s views need to be in their designated positions
  ▪ The context’s `completeTransition:` method must be invoked
Interactive View Controller
Transitions

Introduction
Interactive View Controller Transitions

Running transition animations interactively

- UINavigationController
  - Interactive pop gesture is pervasive on iOS 7.0
Interactive View Controller Transitions

Running transition animations interactively

- UINavigationController
  - Interactive pop gesture is pervasive on iOS 7.0
- Applications can define their own interactive transitions
Interactive View Controller Transitions
Running transition animations interactively

• UINavigationController
  ▪ Interactive pop gesture is pervasive on iOS 7.0
• Applications can define their own interactive transitions
  ▪ Interactive transitions need not be gesture driven
Interactive View Controller Transitions

Running transition animations interactively

• UINavigationController
  ▪ Interactive pop gesture is pervasive on iOS 7.0
• Applications can define their own interactive transitions
  ▪ Interactive transitions need not be gesture driven
  ▪ Interactive transitions usually run forwards and backwards
    ▪ Often a transition can start and be cancelled
Interactive View Controller Transitions
Running transition animations interactively

• UINavigationController
  - Interactive pop gesture is pervasive on iOS 7.0
• Applications can define their own interactive transitions
  - Interactive transitions need not be gesture driven
  - Interactive transitions usually run forwards and backwards
    - Often a transition can start and be cancelled
• UIKit provides a concrete interaction controller class
  - UIPercentDrivenInteractiveTransition
Interactive View Controller Transitions
<UIViewControllerInteractiveTransitioning>

@protocol UIViewControllerInteractiveTransitioning <NSObject>

- (void)startInteractiveTransition:(id <UIViewControllerContextTransitioning>)ctx;

@optional

- (CGFloat)completionSpeed;
- (UIViewAnimationCurve)completionCurve;

@end
Interactive View Controller Transitions
<UIViewControllerInteractiveTransitioning>

@protocol UIViewControllerInteractiveTransitioning <NSObject>

- (void)startInteractiveTransition:(id <UIViewControllerContextTransitioning>)ctx;

@optional

- (CGFloat)completionSpeed;
- (UIVIEWAnimationCurve)completionCurve;

@end
Interactive View Controller Transitions

Start of an interactive presentation

- Presented Controller
- Presenting Controller
- transitionDelegate
Interactive View Controller Transitions
Start of an interactive presentation

Presented Controller
setTransitioningDelegate:

Presenting Controller
transitionDelegate

<UIViewControllerTransitioningDelegate>
(transitionDelegate)
Interactive View Controller Transitions

Start of an interactive presentation

Presented Controller
- setTransitioningDelegate:

Presenting Controller
- presentViewController:

transitionDelegate

<UIViewControllerTransitioningDelegate>
(transitionDelegate)

Presented Controller
Interactive View Controller Transitions

Start of an interactive presentation

- **Presented Controller**
  - setTransitioningDelegate:
  - `<UIViewControllerTransitioningDelegate>(transitionDelegate)`

- **Presenting Controller**
  - presentViewController:
  - animationControllerForPresentingController: presentedController:sourceController:

- **transitionDelegate**
  - interactionControllerForPresentation:

- **Presented Controller**
  - `<UIViewControllerInteractiveTransitioning>(interactionController)`

- **Presenting Controller**
  - `<UIViewControllerAnimatedTransitioning>(animationController)`
UIViewControllerTransitioning

Interactive transitioning states

States

- No Transition
- Start Interactive Transition
- Update Interactive Transition
- Interaction Transition End
- Transition Canceling
- Transition Complete
- Transition Finishing
- No Transition

Agents

- Interactive Event Handler
- Interactor
- Animator
- Context
UIViewControllerTransitioning
Interactive transitioning states

States

- No Transition
- Start Interactive Transition
- Update Interactive Transition
- Interaction Transition End
- Transition Canceling
- Transition Finishing
- Transition Complete
- No Transition

Agents

- Interactive Event Handler
- Interactor
- Animator
- Context
UIViewControllerTransitioning
Interactive transitioning states

States
- No Transition
- Start Interactive Transition
- Update Interactive Transition
- Interaction Transition End
- Transition Canceling
- Transition Finishing
- Transition Complete
- No Transition

Agents
- Interactive Event Handler
- Interactor
- Animator
- Context
UIViewControllerTransitioning
Interactive transitioning states

States

- No Transition
- Start Interactive Transition
- Update Interactive Transition
- Interaction Transition End
- Transition Canceling
- Transition Finishing
- Transition Complete
- No Transition

Agents

- Interactive Event Handler
- Interactor
- Animator
- Context
UIViewControllerTransitioning

Interactive transitioning states

States

- No Transition
- Start Interactive Transition
- Update Interactive Transition
- Interaction Transition End
- Transition Canceling
- Transition Complete
- Transition Finishing
- No Transition

Agents

- finishInteractiveTransition
- Interactive Event Handler
- Interactor
- Animator
- Context
UIViewControllerTransitioning

Interactive transitioning states

States

- No Transition
- Start Interactive Transition
- Update Interactive Transition
- Interaction Transition End
- Transition Canceling
- Transition Complete
- No Transition
- Transition Finishing

Agents

- Interactive Event Handler
- Interactor
- Animator
- Context
UIViewControllerTransitioning
Interactive transitioning states

States
- No Transition
- Start Interactive Transition
- Update Interactive Transition
- Interaction Transition End
- Transition Canceling
- Transition Complete
- No Transition

Agents
- Interactive Event Handler
- Interactor
- Animator
- Context
UIViewControllerTransitioning
Interactive transitioning states

States
- No Transition
- Start Interactive Transition
- Update Interactive Transition
- Interaction Transition End
- Transition Canceling
- Transition Complete
- Transition Finishing
- No Transition

Agents
- Interactive Event Handler
- Interactor
- Animator
- Context
**UIViewControllerTransitioning**

Interactive transitioning states

**States**

- No Transition
- Start Interactive Transition
- Update Interactive Transition
- Interaction Transition End
- Transition Canceling
- Transition Finishing
- Transition Complete
- No Transition

**Agents**

- Interactive Event Handler
- Interactor
- Animator
- Context
Interactive View Controller Transitions
The easy way—use UIViewControllerPercentDrivenTransition
Interactive View Controller Transitions

The easy way—use UIViewControllerPercentDrivenTransition

• Implement the animation controller
  • animatePresentation: must be implemented using the UIView animation block APIs
Interactive View Controller Transitions
The easy way—use UIViewControllerPercentDrivenTransition

• Implement the animation controller
  ▪ animatePresentation: must be implemented using the UIView animation block APIs

• Implement the logic that will drive the interaction
  ▪ e.g. The target of a gesture recognizer
  ▪ Often this target is a subclass of UIViewControllerPercentDrivenTransition
Interactive View Controller Transitions
The easy way—use UIViewControllerPercentDrivenTransition

• Implement the animation controller
  ▪ `animatePresentation:` must be implemented using the UIView animation block APIs

• Implement the logic that will drive the interaction
  ▪ e.g. The target of a gesture recognizer
  ▪ Often this target is a subclass of UIViewControllerPercentDrivenTransition
  ▪ The interaction logic will call
    ▪ `updateInteractiveTransition:(CGFloat)percent`
    ▪ `completeInteractiveTransition` or `cancelInteractiveTransition`
    ▪ (Note that `startInteractiveTransition` is handled automatically)
Interactive View Controller Transitions

UIPercentDrivenInteractiveTransition

// The associated animation controller must animate its transition using UIView animation APIs.
@interface UIPercentDrivenInteractiveTransition : NSObject <UIViewControllerInteractiveTransitioning>

@property (readonly) CGFloat duration;
// The last percentComplete value specified by updateInteractiveTransition:
@property (readonly) CGFloat percentComplete;

// completionSpeed defaults to 1.0 which corresponds to a completion duration of
// (1 - percentComplete)*duration. It must be greater than 0.0.
@property (nonatomic,assign) CGFloat completionSpeed;

// When the interactive part of the transition has completed, this property can
// be set to indicate a different animation curve.
@property (nonatomic,assign) UIViewAnimationCurve completionCurve;

// Used instead of the corresponding context methods.
- (void)updateInteractiveTransition:(CGFloat)percentComplete;
- (void)cancelInteractiveTransition;
- (void)finishInteractiveTransition;
@end
Interactive View Controller Transitions

UIPercentDrivenInteractiveTransition

// The associated animation controller must animate its transition using UIView animation APIs.
@interface UIPercentDrivenInteractiveTransition : NSObject <UIViewControllerInteractiveTransitioning>

@property (readonly) CGFloat duration;
// The last percentComplete value specified by updateInteractiveTransition:
@property (readonly) CGFloat percentComplete;

// completionSpeed defaults to 1.0 which corresponds to a completion duration of
// (1 - percentComplete)*duration. It must be greater than 0.0.
@property (nonatomic,assign) CGFloat completionSpeed;

// When the interactive part of the transition has completed, this property can
// be set to indicate a different animation curve.
@property (nonatomic,assign) UIViewAnimationCurve completionCurve;

// Used instead of the corresponding context methods.
- (void)updateInteractiveTransition:(CGFloat)percentComplete;
- (void)cancelInteractiveTransition;
- (void)finishInteractiveTransition;
@end
Interactive View Controller Transitions

The easy way—use UIViewControllerPercentDrivenTransition
Interactive View Controller Transitions
The easy way—use UIViewControllerPercentDrivenTransition
Interactive View Controller Transitions
UIPercentDrivenInteractiveTransition

@interface YYSlideInteractor : UIPercentDrivenInteractiveTransition
-
(instancetype)initWithNavigationController:(UINavigationController *)nc;

@propertynonatomic, assign) UINavigationController *parent;
@property nonatomic, assign, getter = isInteractive) BOOL interactive;
@end
Interactive View Controller Transitions

UIPercentDrivenInteractiveTransition

-(void)handlePinch:(UIPinchGestureRecognizer *)gr {
    CGFloat scale = [gr scale];
    switch ([gr state]) {
        case UIGestureRecognizerStateBegan:
            self.interactive = YES; _startScale = scale;
            [self.parent pushViewController:Animated:YES];
            break;
        case UIGestureRecognizerStateChanged: {
            CGFloat percent = (1.0 - scale/_startScale);
            [self updateInteractiveTransition: (percent <= 0.0) ? 0.0 : percent];
            break;
        }
        case UIGestureRecognizerStateEnded:
        case UIGestureRecognizerStateCancelled:
            if([gr velocity] >= 0.0 || [gr state] == UIGestureRecognizerStateCancelled)
                [self cancelInteractiveTransition];
            else
                [self finishInteractiveTransition];
            self.interactive = NO;
            break;
    }
}
- (void)handlePinch:(UIPinchGestureRecognizer *)gr {
    CGFloat scale = [gr scale];
    switch ([gr state]) {
    case UIGestureRecognizerStateBegan:
        self.interactive = YES; _startScale = scale;
        [self.parent popViewControllerAnimated:YES];
        break;
    case UIGestureRecognizerStateChanged: {
        CGFloat percent = (1.0 - scale/_startScale);
        [self updateInteractiveTransition: (percent <= 0.0) ? 0.0 : percent];
        break;
    }
    case UIGestureRecognizerStateEnded:
    case UIGestureRecognizerStateCancelled:
        if([gr velocity] >= 0.0 || [gr state] == UIGestureRecognizerStateCancelled)
            [self cancelInteractiveTransition];
        else
            [self finishInteractiveTransition];
        self.interactive = NO;
        break;
    }
}
Interactive View Controller Transitions

UIPercentDrivenInteractiveTransition

- (void)handlePinch:(UIPinchGestureRecognizer *)gr {
    CGFloat scale = [gr scale];
    switch ([gr state]) {
    case UIGestureRecognizerStateBegan:
        self.interactive = YES; _startScale = scale;
        [self.parent popViewControllerAnimated:YES];
        break;
    case UIGestureRecognizerStateChanged: {
        CGFloat percent = (1.0 - scale/_startScale);
        [self updateInteractiveTransition: (percent <= 0.0) ? 0.0 : percent];
        break;
    }
    case UIGestureRecognizerStateChanged:
    case UIGestureRecognizerStateEnded:
    case UIGestureRecognizerStateCancelled:
        if([gr velocity] >= 0.0 || [gr state] == UIGestureRecognizerStateCancelled)
            [self cancelInteractiveTransition];
        else
            [self finishInteractiveTransition];
        self.interactive = NO;
        break;
    }
}
Interactive View Controller Transitions
UIPercentDrivenInteractiveTransition

- (void)handlePinch:(UIPinchGestureRecognizer *)gr {
  CGFloat scale = [gr scale];
  switch ([gr state]) {
    case UIGestureRecognizerStateBegan:
      self.interactive = YES; _startScale = scale;
      [self.parent popViewControllerAnimated:YES];
      break;
    case UIGestureRecognizerStateChanged: {
      CGFloat percent = (1.0 - scale/_startScale);
      [self updateInteractiveTransition: (percent <= 0.0) ? 0.0 : percent];
      break;
    }
    case UIGestureRecognizerStateEnded:
    case UIGestureRecognizerStateCancelled:
      if([gr velocity] >= 0.0 || [gr state] == UIGestureRecognizerStateCancelled) {
        [self cancelInteractiveTransition];
      } else {
        [self finishInteractiveTransition];
        self.interactive = NO;
      }
      break;
  }
}
Interactive Collection View Layout Transitions

Olivier Gutknecht
UICollectionViewTransitionLayout
UICollectionViewTransitionLayout

• A new layout interpolating between two layouts
UICollectionViewTransitionLayout

• A new layout *interpolating* between two layouts
• Interactively or not, with the `transitionProgress` property
UICollectionViewTransitionLayout

• A new layout **interpolating** between two layouts
• Interactively or not, with the `transitionProgress` property
• Subclassable
UICollectionViewTransitionLayout

• A new layout **interpolating** between two layouts
• Interactively or not, with the `transitionProgress` property
• Subclassable
• Simple integration with view controller transitions
Collection Views and Transition Layout

UICollectionViewTransitionLayout

Layout A

Layout B
Collection Views and Transition Layout

UICollectionViewTransitionLayout

Layout A

Transition Layout

transitionProgress

0.0

Layout B
Collection Views and Transition Layout

UICollectionViewTransitionLayout

Layout A

Transition Layout

transitionProgress

0.5

Layout B
Collection Views and Transition Layout

UICollectionViewTransitionLayout

Layout A

Transition Layout

transitionProgress

1.0

Layout B
Collection Views and Transition Layout

UICollectionViewTransitionLayout

Layout A

Transition Layout

transitionProgress

Layout B
Collection Views and Transition Layout

UICollectionViewTransitionLayout

Layout A

Transition Layout

Layout B

transitionProgress

1.0
Interactive Transitions
UICollectionView
Interactive Transitions

UICollectionView

• New methods in UICollectionView
  - (UICollectionViewTransitionLayout *)
    startInteractiveTransitionToCollectionViewLayout:completion:
  - (void)finishInteractiveTransition
  - (void)cancelInteractiveTransition
Interactive Transitions

UICollectionView

• New methods in UICollectionView
  - (UICollectionViewTransitionLayout *)
    startInteractiveTransitionToCollectionViewLayout:completion:
  - (void)finishInteractiveTransition
  - (void)cancelInteractiveTransition

• New delegate method
  - (UICollectionViewTransitionLayout *)collectionView:(UICollectionView*)v
    transitionLayoutForOldLayout:(UICollectionViewLayout*)o
    newLayout:(UICollectionViewLayout*)n
Interactive Transitions

UICollectionView

• New methods in UICollectionView
  – (UICollectionViewTransitionLayout *)
    startInteractiveTransitionToCollectionViewLayout:completion:
  – (void)finishInteractiveTransition
  – (void)cancelInteractiveTransition

• New delegate method
  – (UICollectionViewTransitionLayout *)collectionView:(UICollectionView*)v
    transitionLayoutForOldLayout:(UICollectionViewLayout*)o
    newLayout:(UICollectionViewLayout*)n

• Does not replace
  – (void)setCollectionViewLayout:animated:
Subclassing Transition Layout
Subclassing Transition Layout

• Implement your UICollectionViewTransitionLayout subclass
  ▪ e.g. update cell positions based on gesture position
Subclassing Transition Layout

• Implement your UICollectionViewTransitionLayout subclass
  - e.g. update cell positions based on gesture position
• Create an instance of your own class in your delegate method
Subclassing Transition Layout

• Implement your UICollectionViewTransitionLayout subclass
  ▪ e.g. update cell positions based on gesture position
• Create an instance of your own class in your delegate method
• On finish or cancel, UIKit animates at correct velocity
  ▪ Track your own parameters
    `updateValue:forAnimatedKey:`
  ▪ UIKit will monitor velocity
  ▪ UIKit gives you in-sync values on completion and cancel with
    `valueForAnimatedKey:`
Demo
Collection Views Transitions

Other enhancements
Collection Views Transitions

Other enhancements

• Better control of target offsets everywhere
targetContentOffsetForProposedContentOffset:
Collection Views Transitions

Other enhancements

• Better control of target offsets everywhere
  targetContentOffsetForProposedContentOffset:

• Layouts are now notified on transitions
  prepareForTransitionToLayout:
  prepareForTransitionFromLayout:
  finalizeLayoutTransition
Collection Views Transitions

Other enhancements

• Better control of target offsets everywhere
  targetContentOffsetForProposedContentOffset:

• Layouts are now notified on transitions
  prepareForTransitionToLayout:
  prepareForTransitionFromLayout:
  finalizeLayoutTransition

• Better animations
  • Initial and final layout attributes are now supported
  • A new completion handler
    setCollectionViewLayout:animated:completion:
Interactive View Controller
Transitions
Cancellation and coordinators
UIViewControllerTransitioning
Interactive transitioning states

No Transition
Start Interactive Transition
Update Interactive Transition
Interaction Transition End
Transition Canceling
Transition Complete
Transition Finishing
No Transition
UIViewControllerTransitioning
Interactive transitioning states

No Transition  Start Interactive Transition  Update Interactive Transition  Interaction Transition End  Transition Canceling  Transition Complete  No Transition

Transition Canceling  Transition Finishing
UIViewControllerTransitioning
Cancelling and appearance callbacks

Interactive transitioning states
- Interaction Transition End
- Transition Canceling
- Transition Finishing
- Transition Complete

View controller appearance states
- Appearing
- Appeared
- Disappeared
- Disappearing
UIViewControllerTransitioning
Cancellation and appearance callbacks

Interactive transitioning states
- Interaction Transition End
- Transition Canceling
- Transition Finishing
- Transition Complete

View controller appearance states
- Appearing
- Appeared
- Disappearing
- Disappeared
UIViewControllerTransitioning
Cancellation and appearance callbacks

Interactive transitioning states
- Interaction Transition End
- Transition Canceling
- Transition Finishing
- Transition Complete

View controller appearance states
- Appearing
- Appeared
- Disappeared
- Disappearing
Interactive View Controller Transitions

Canceling an interactive transition
Interactive View Controller Transitions

Canceling an interactive transition

• Don’t assume that viewDidAppear follows viewWillAppear
Interactive View Controller Transitions

Canceling an interactive transition

• Don’t assume that `viewDidAppear` follows `viewWillAppear`
• Make sure to undo any side effects
  ▪ There is new API to help manage this
Interactive View Controller Transitions

@interface UIViewController(TransitionCoordinator)

@property (nonatomic,retain) id <UIViewControllerTransitionCoordinator> transitionCoordinator;

@end
Interactive View Controller Transitions

@interface UIViewController(TransitionCoordinator)

@property (nonatomic,retain) id <UIViewControllerTransitionCoordinator>
transitionCoordinator;
@end
Interactive View Controller Transitions

@protocol UIViewControllerTransitionCoordinator
    <UIViewControllerTransitionCoordinatorContext>
@end

@optional

- (BOOL) notifyWhenInteractionEndsUsingBlock:
    (void (^ (id<UIViewControllerTransitionCoordinatorContext>handler;

- (BOOL) animatorAlongsideTransition:
    (void (^) (id <UIViewControllerTransitionCoordinatorContext)a;
    completion:(void (^)(id<UIViewControllerTransitionCoordinatorContext>c;

- (BOOL) animatorAlongsideTransitionInView:(UIView *)view
    animation: (void (^) (id <UIViewControllerTransitionCoordinatorContext)a;
    completion:(void (^)(id<UIViewControllerTransitionCoordinatorContext>c;
@end
@protocol UIViewControllerTransitionCoordinator
    <UIViewControllerTransitionCoordinatorContext>
@optional

- (BOOL) notifyWhenInteractionEndsUsingBlock:
    (void (^ (id<UIViewControllerTransitionCoordinatorContext>)handler;

- (BOOL) animatorAlongsideTransition:
    (void (^)(id <UIViewControllerTransitionCoordinatorContext>)a;
    completion:(void (^)(id<UIViewControllerTransitionCoordinatorContext>)c;

- (BOOL) animatorAlongsideTransitionInView:(UIView *)view
    animation: (void (^)(id <UIViewControllerTransitionCoordinatorContext>)a;
    completion:(void (^)(id<UIViewControllerTransitionCoordinatorContext>)c;
@end
Interactive View Controller Transitions

@protocol UIViewControllerTransitionCoordinatorContext <NSObject>

- (UIView *)containerView;
- (UIViewController *) viewControllerForKey:(NSString *)key;
- (CGRect) initialFrameForViewController:(UIViewController *)vc;
- (CGRect) finalFrameForViewController:(UIViewController *)vc;

- (BOOL) isCancelled;
- (BOOL) initiallyInteractive;
- (BOOL) isInteractive;

@end
Interactive View Controller Transitions

@protocol UIViewControllerTransitionCoordinatorContext <NSObject>

- (UIView *)containerView;
- (UIViewController *) viewControllerForKey:(NSString *)key;
- (CGRect) initialFrameForViewController:(UIViewController *)vc;
- (CGRect) finalFrameForViewController:(UIViewController *)vc;
- (BOOL) isCancelled;
- (BOOL) initiallyInteractive;
- (BOOL) isInteractive;

@end
Interactive View Controller Transitions
Canceling an interactive transition
Interactive View Controller Transitions

Canceling an interactive transition

• Don’t assume that viewDidLoad follows viewWillAppear:
Interactive View Controller Transitions

Canceling an interactive transition

• Don’t assume that viewDidLoad follows viewWillAppear:

    -(void)viewWillAppear: {
        [self doSomeSideEffectsAssumingViewDidLoadIsGoingToBeCalled];

        id <UIViewControllerTransitionCoordinator> coordinator;
        coordinator = [self transitionCoordinator];

        if(coordinator && [coordinator initiallyInteractive]) {
            [transitionCoordinator notifyWhenInteractionEndsUsingBlock:
                ^(id <UIViewControllerTransitionCoordinatorContext> ctx) {
                    if(ctx.isCancelled) {
                        [self undoSideEffects];
                    }
                }];
        }
    }
Interactive View Controller Transitions

Canceling an interactive transition

• Don’t assume that viewDidAppear follows viewWillAppear:
  
  ```
  (void) viewWillAppear: {
    [self doSomeSideEffectsAssumingViewDidAppearIsGoingToBeCalled];
  }
  ```

  ```
  id <UIViewControllerTransitionCoordinator> coordinator;
  coordinator = [self transitionCoordinator];

  if(coordinator && [coordinator initiallyInteractive]) {
    [transitionCoordinator notifyWhenInteractionEndsUsingBlock:
    ^(id <UIViewControllerTransitionCoordinatorContext> ctx) {
      if(ctx.isCancelled) {
        [self undoSideEffects];
      }
    }];
  }
  ```
Interactive View Controller Transitions

Canceling an interactive transition

• Don’t assume that `viewDidAppear` follows `viewWillAppear`:
  
  ```
  - (void)viewWillAppear: {
    [self doSomeSideEffectsAssumingViewDidAppearIsGoingToBeCalled];

  id <UIViewControllerTransitionCoordinator> coordinator;
  coordinator = [self transitionCoordinator];

  if(coordinator && [coordinator initiallyInteractive]) {
    [transitionCoordinator notifyWhenInteractionEndsUsingBlock:
      ^(id <UIViewControllerTransitionCoordinatorContext> ctx) {
        if(ctx.isCancelled) {
          [self undoSideEffects];
        }
      }];
  }
  }
  ```
Interactive View Controller Transitions

Don’t assume that viewDidLoad follows viewDidAppear:

```swift
-(void) viewWillAppear: {
    [self doSomeSideEffectsAssumingViewDidAppearIsGoingToBeCalled];

    id <UIViewControllerTransitionCoordinator> coordinator = [self transitionCoordinator];
    if(coordinator && [coordinator initiallyInteractive]) {
        [transitionCoordinator notifyWhenInteractionEndsUsingBlock:
            ^(id <UIViewControllerTransitionCoordinatorContext> ctx) {
                if(ctx.isCancelled) {
                    [self undoSideEffects];
                }
            }];
    }
}
```

Canceling an interactive transition

Interactive View Controller Transitions
Interactive View Controller Transitions

<UIViewControllerTransitionCoordinator>
Interactive View Controller Transitions

- The transitionCoordinator does even more
  - Allows completion handlers to be registered for transitions
Interactive View Controller Transitions

• The transitionCoordinator does even more
  ▪ Allows completion handlers to be registered for transitions
  ▪ Allows other animations to run alongside the transition animation
Interactive View Controller Transitions

 UIViewControllerTransitionCoordinator

• The transitionCoordinator does even more
  ▪ Allows completion handlers to be registered for transitions
  ▪ Allows other animations to run alongside the transition animation

• In addition to custom transitions on iOS 7
  ▪ UINavigationController transitions have an associated transition coordinator
Interactive View Controller Transitions

<UIViewControllerTransitionCoordinator>

• The transitionCoordinator does even more
  ▪ Allows completion handlers to be registered for transitions
  ▪ Allows other animations to run alongside the transition animation

• In addition to custom transitions on iOS 7
  ▪ UINavigationController transitions have an associated transition coordinator
  ▪ Present and Dismiss transitions have an associated coordinator
Interactive View Controller Transitions

- (BOOL)
  animatorAlongsideTransition:(void (^) (id <UIViewControllerTransitionCoordinatorContext>)a;
  completion:(void (^) (id<UIViewControllerTransitionCoordinatorContext>)c;
- (BOOL)
  animatorAlongsideTransitionInView:(UIView *)view
  animation: (void (^) (id <UIViewControllerTransitionCoordinatorContext>)a;
  completion:(void (^) (id<UIViewControllerTransitionCoordinatorContext>)c;
Interactive View Controller Transitions
<UIViewControllerTransitionCoordinator>

- (BOOL)
  animatorAlongsideTransition:(void (^)(id <UIViewControllerTransitionCoordinatorContext>)a;
  completion:(void (^)(id<UIViewControllerTransitionCoordinatorContext>)c;

- (BOOL)
  animatorAlongsideTransitionInView:(UIView *)view
  animation: (void (^)(id <UIViewControllerTransitionCoordinatorContext>)a;
  completion: (void (^)(id<UIViewControllerTransitionCoordinatorContext>)c;
UIViewController *vc;
[self pushViewController:vc animated: YES];

id <UIViewControllerTransitionCoordinator>coordinator;
coordinator = [viewController transitionCoordinator];

[coordinator animateAlongsideTransition:
 ^ (id <UIViewControllerTransitionCoordinatorContext> c) {
  ;; some animation
 }]
completion:(id <UIViewControllerTransitionCoordinatorContext> c) {
  ;; Code to run after your push transition has finished.
};
Interactive View Controller Transitions
Fun with transition coordinators

```objective-c
UIViewController *vc;
[self pushViewController:vc animated: YES];

id <UIViewControllerTransitionCoordinator>coordinator;
coordinator = [viewController transitionCoordinator];

[coordinator animateAlongsideTransition: ^(id <UIViewControllerTransitionCoordinatorContext> c) {
    ;; some animation
}]
completion:(id <UIViewControllerTransitionCoordinatorContext> c) {
    ;; Code to run after your push transition has finished.
}
```
Interactive View Controller Transitions

Fun with transition coordinators

UIViewController *vc;
[self pushViewController:vc animated: YES];

id <UIViewControllerTransitionCoordinator>coordinator;
coordinator = [viewController transitionCoordinator];

[coordinator animateAlongsideTransition: ^(id <UIViewControllerTransitionCoordinatorContext> c) {
    ;; some animation
}
completion:(id <UIViewControllerTransitionCoordinatorContext> c) {
    ;; Code to run after your push transition has finished.
}];
Interactive View Controller Transitions
Fun with transition coordinators

UIViewController *vc;
[self pushViewController:vc animated: YES];

id <UIViewControllerTransitionCoordinator>coordinator;
coordinator = [viewController transitionCoordinator];

[coordinator animateAlongsideTransition:
 ^{(id <UIViewControllerTransitionCoordinatorContext> c) {
     ;; some animation
 };
 completion:(id <UIViewControllerTransitionCoordinatorContext> c) {
     ;; Code to run after your push transition has finished.
 }}];
Concluding Remarks

“With great power comes greater responsibility”
Concluding Remarks

“With great power comes greater responsibility”
Concluding Remarks

“With great power comes greater responsibility”
Concluding Remarks

“With great power comes greater responsibility”
Concluding Remarks

“With great power comes greater responsibility”

• Powerful new animation and snapshot APIs can be used to create awesome transition animations
Concluding Remarks

“With great power comes greater responsibility”

• Powerful new animation and snapshot APIs can be used to create awesome transition animations
• Many view controller transition animations can be customized
  ▪ UICollectionViewControllers and UICollectionViewViews can be easily used to define custom transitions
  ▪ The protocol based expression of this API is very flexible
Concluding Remarks

“With great power comes greater responsibility”

• Powerful new animation and snapshot APIs can be used to create awesome transition animations

• Many view controller transition animations can be customized
  ▪ UICollectionViewControllers and UICollectionViews can be easily used to define custom transitions
  ▪ The protocol based expression of this API is very flexible

• View controller transitions can be interactive
  ▪ Is it viewWillAppear: Or viewWillAppear:?
Concluding Remarks

“With great power comes greater responsibility”

- Powerful new animation and snapshot APIs can be used to create awesome transition animations
- Many view controller transition animations can be customized
  - UICollectionViewControllers and UICollectionViews can be easily used to define custom transitions
  - The protocol based expression of this API is very flexible
- View controller transitions can be interactive
  - Is it `viewWillAppear:` or `viewWillProbablyAppear:`?
- A transition coordinator can be used with/without custom transitions
  - `animateAlongsideTransition:completion:`
  - `notifyWhenInteractionEndsUsingBlock:`
More Information

Jake Behrens
App Frameworks Evangelist
behrens@apple.com

Documentation and Sample Code
iOS Dev Center
http://developer.apple.com

Apple Developer Forums
http://devforums.apple.com
## Related Sessions

<table>
<thead>
<tr>
<th>Session</th>
<th>Location</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Building User Interfaces for iOS 7</td>
<td>Presidio</td>
<td>Tuesday 10:15AM</td>
</tr>
<tr>
<td>Getting Started with UIKit Dynamics</td>
<td>Presidio</td>
<td>Tuesday 4:30PM</td>
</tr>
<tr>
<td>Advance Techniques with UIKit Dynamics</td>
<td>Presidio</td>
<td>Thursday 3:15PM</td>
</tr>
<tr>
<td>Best Practices for Great iOS UI Design</td>
<td>Presidio</td>
<td>Friday 10:15AM</td>
</tr>
<tr>
<td>Labs</td>
<td>Frameworks Lab B</td>
<td></td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------------------------</td>
<td></td>
</tr>
<tr>
<td>Cocoa Touch Animation Lab</td>
<td>Thursday 2:00PM</td>
<td></td>
</tr>
</tbody>
</table>