Animation Overview

Graphics & Imaging > Quartz

2008-10-15

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Simultaneously published in the United States and Canada.

Contents

Figures

Introduction to Animation Overview

This document is intended as a general introduction to the animation capabilities provided to programmers in Mac OS X.

This document is for all developers interested in using animation in Cocoa applications. It is a good starting place if you have no prior knowledge of animation, but you should be familiar with the basic concepts of using views in a Cocoa application. You do not need any understanding of graphics programming in general, although such knowledge would be helpful.

If you are an advanced Cocoa developer and already familiar with animation techniques, you should read this book for tips on using animation efficiently in a Cocoa application, and how you can ensure that the animations you use integrate effectively with the Mac OS X user experience.

Organization of This Document

This document has the following chapters:

- "What Is Animation?" (page 9) describes what animation is and how it is used in Mac OS X.
- "Animation Basics" (page 13) describes the basic components of an animation.
- "Mac OS X Animation Technologies" (page 15) describes the technologies that Mac OS X provides for animation.
- "Choosing the Animation Technology for Your Application" (page 21) explores the strong points of Core Animation layers and layer-backed views and provides guidelines for choosing which is appropriate for your application.

See Also

For more information about Core Animation and the Cocoa animation classes, see the following resources:

- *Core Animation Programming Guide* provides detailed information on integrating Core Animation in your application.
- *Animation Programming Guide for Cocoa* describes the animation classes and protocols provided by the Application Kit.

What Is Animation?

Animation is a visual technique that provides the illusion of motion by displaying a collection of images in rapid sequence. Each image contains a small change, for example a leg moves slightly, or the wheel of a car turns. When the images are viewed rapidly, your eye fills in the details and the illusion of movement is complete.

When used appropriately in your application's user interface, animation can enhance the user experience while providing a more dynamic look and feel. Moving user interface elements smoothly around the screen, gradually fading them in and out, and creating new custom controls with special visual effects can combine to create a cinematic computing experience for your users.

Animation in Mac OS X

There are many examples of animation in the Mac OS X user interface:

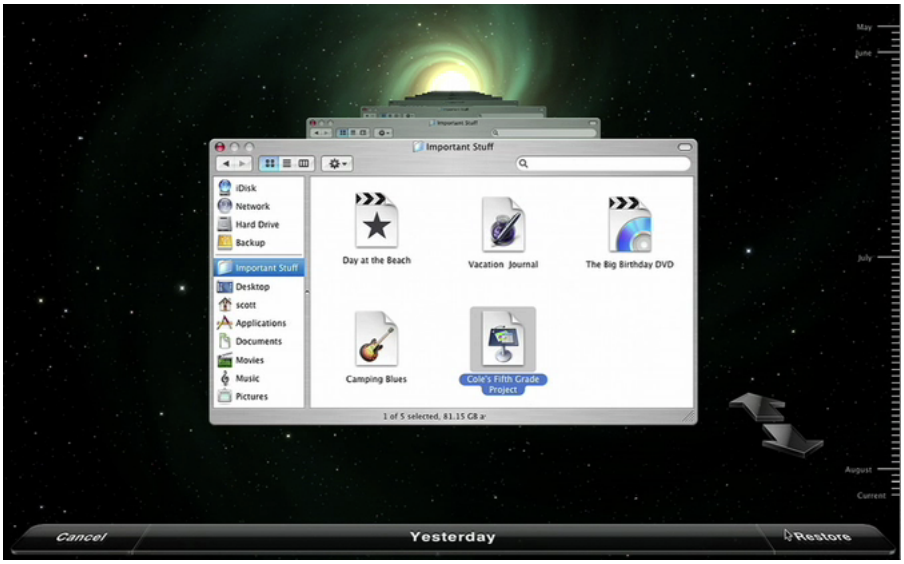
- Dock icons bounce to indicate that an application requires the user's attention.
- Sheets and drawers use animation as they are displayed and hidden.
- Progress indicators animate to indicate that a lengthy operation is in progress.
- Dragging icons to the dock causes icons to "part" to allow the new icon to be inserted.
- The default button in a panel pulses.

Applications also use animation effectively to provide a rich, dynamic user experience. For example, iChat plays a sound, and then uses animation to show when a buddy is no longer available. The sound alerts you to the change in status, and the animation provides you a chance to focus on the application to analyze the event.

Front Row uses animation throughout its user interface. For example, when Front Row is activated a transition is used to show that the mode is changing from the Mac OS Aqua interface to the Front Row interface. As you navigate deeper into the hierarchy of menus, new menus push in from the right, pushing the old menu off to the left. As you ascend the menus, the transitions reverse, again helping you maintain context.

In Time Machine perceived distance provides an intuitive metaphor for a linear progression in time. Older file system snapshots are shown further away, allowing you to move through them to find the version you want to restore.

Figure 1-1 Time Machine user interface



The iTunes 7.0 CoverFlow interface shows album and movie covers in an engaging manner. As you browse through the content the images animate to face the user directly. This is a great example of the cinematic computing experience. And the CoverFlow user interface allows more content to be displayed in a smaller area than if the images were placed side by side.

Figure 1-2 iTunes 7.0 CoverFlow user interface

Even small uses of animation can communicate well. The iSync menu extra animates to show the syncing is in progress (Figure Figure 1-3.) In all these cases the applications provide additional information and context to the user through the use of animation.

Figure 1-3 iSync menu status item

Using Animation in Your Applications

How you incorporate animation into your own application largely depends on the type of interface your application provides. Applications that use the Aqua user interface can best integrate animation by creating custom controls or views. Applications that create their own user interface, such as educational software, casual games, or full-screen applications such as Front Row, have much greater leeway in determining how much animation is appropriate for their users.

With judicious use of animation and visual effects, the most mundane system utility can become a rich visual experience for users, thus providing a compelling competitive advantage for your application.

Animation Basics

There are several fundamental attributes required of all animations: They must be associated with an object to animate, and they must define what type of animation will be performed and how long the animation will last.

This chapter discusses, in abstract terms, the basic animation techniques that are common to the Mac OS X animation technologies.

Animation Target Object

Each animation must be associated with a visual element that the animation will affect. You can think of this as the animation target object. The animation target object provides the content that is displayed to the user. An animation will act either on the animation target object as a whole, or on a specific property of the target, for example, its location in the coordinate system or the color that it is drawn with.

Animations are associated with an animation target object. You don't explicitly start an animation; the animation target object starts and stops the animation.

Types of Animation

Mac OS X animation supports three distinct types of animation: basic animation, keyframe animation, and transition animations.

Basic Animation

Basic animation—also known as simple animation or single keyframe animation—progresses from a starting value to a target value through a series of intermediate values. These intermediate values are calculated, or interpolated, such that the animation occurs over a specified duration. Basic animation requires that you specify a property of the animation target object to animate.

Basic animation can be used with any value types that can be interpolated, including:

- integers and doubles
- `CGRect`, `CGPoint`, `CGSize`, and `CGAffineTransform` structures
- `CATransform3D` data structures
- `CGColor` and `CGImage` references

Keyframe Animation

Keyframe animation is similar to basic animation; however it allows you to specify an array of target values. Each of these target values is reached, in turn, over the duration of the animation. The keyframes of a keyframe animation can be any of the types supported by basic animation, or a Core Graphics path which is decomposed into a series of `CGPoint` object by the animation. Like a basic animation, a keyframe animation requires that the animation act on a specific property of the animation target object.

Transition Animation

Transition animations specify a visual effect that determines how the animation target object is displayed as it is made visible, or hidden. For example, making an animation target object visible may cause it to be pushed into view from the side of the display. Transition animations are performed by using **Core Image** filters.

Because transition animations affect the animation target object as a whole, it is not necessary to specify a property of the target object.

Animation Timing

The overall timing of an animation is determined by several factors: duration, pacing, and the repeating behavior.

Duration is the length of time that an animation takes to get from the starting or current state to the target state, measured in seconds.

Pacing of an animation determines how the interpolated values are distributed over the duration of the animation. For example, a single keyframe animation from 0 to 5 with a duration of 5 seconds and linear pacing causes the intermediate values to be spread out evenly over the duration. The same animation with a pacing of "EaseIn" causes the animation to begin slowly, and then speed up as the animation progresses. The animation takes the same duration and reaches the same values, but when each of the intermediate values are reached differs.

Animation repetition can be specified by in two ways: through a simple count of how many times the animation should repeat, or by setting a duration that the animation should repeat for. For example, specifying a repeat duration of 15 seconds would cause a 5-second animation to repeat three times. You can also specify that an animation should play forward, and then again in reverse, as it repeats.

Mac OS X Animation Technologies

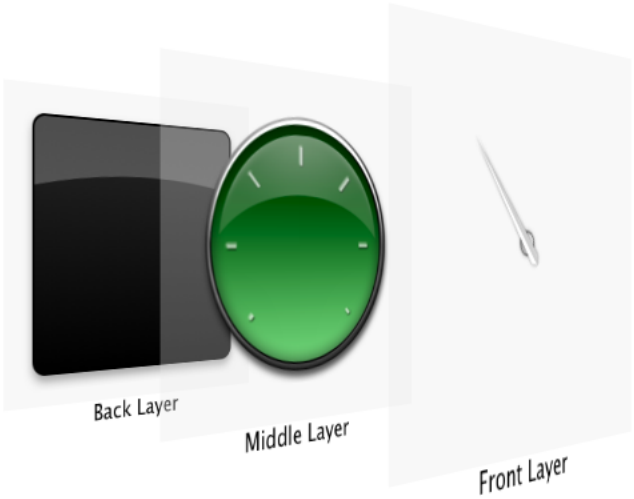
Incorporating sophisticated animations and visual effects into an application can be difficult, often requiring developers to use low-level graphics APIs such as OpenGL to get acceptable animation performance. Mac OS X provides several technologies that make it easier to create rich, animated application interfaces from high-level APIs that are available to both Cocoa and Carbon applications.

Core Animation

Core Animation is an Objective-C framework, first introduced in Mac OS X v10.5, that allows you to dramatically increase the production value of your application by adding real-time animations and visual transitions without needing to know esoteric graphics and math techniques. Using Core Animation you can dynamically render and animate text, 2D graphics, OpenGL, Quartz Composer compositions and QuickTime video simultaneously, complete with transparency effects and Core Image filters and effects.

At its heart, Core Animation is a high-speed, 2D layering engine. You create a Core Animation interface by dividing the user interface into separate layers. By compositing those layers together you create the finished user interface. Figure 3-1 shows a user interface animating its content, then splitting into its decomposed Core Animation layers to show how the layers are combined.

Figure 3-1 A Core Animation user interface decomposed into layers



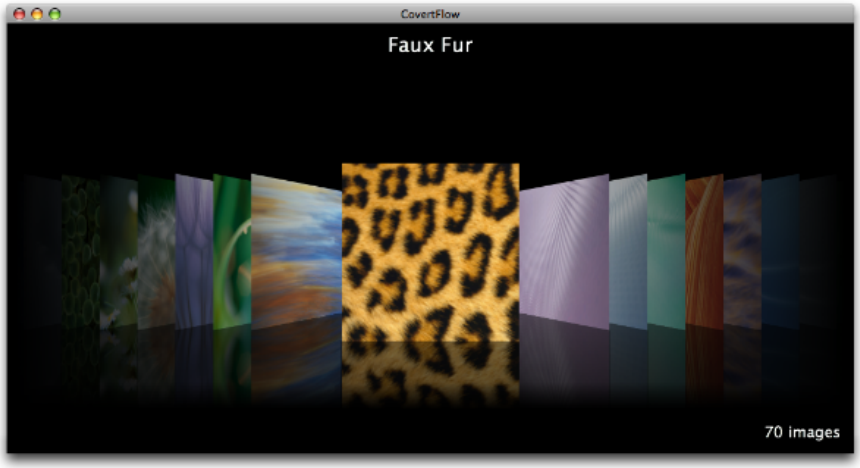
Layers are arranged in a nested layer tree—an abstraction that should be familiar to developers who have used views in their Cocoa application. Each visible layer tree is backed by a corresponding render tree, which is responsible for caching the layer content and rapidly compositing it to the screen as required. Your application doesn't need to perform costly redraw operations unless the content changes.

In addition to the layer's basic position and geometry, each layer also provides optional visual properties that are applied when a layer's content is rendered, including:

- An optional background color
- An optional corner radius, allowing layers to display with rounded corners
- An optional array of Core Image filters that are applied to the content behind a layer before its content is composited
- A Core Image filter used to composite the layer's contents with the background
- An optional array of Core Image filters that are applied to the contents of the layer and its sublayers
- Control over a layer's opacity
- Parameters that are used to draw an optional shadow including the color, offset, opacity and blur radius
- An optional border drawn using the specified line width and color

Although Core Animation is a 2D layering engine, it provides support for convincing 3D scenes using projective transformations. Every layer has a three-dimensional transform matrix that is applied to its content, and a second three-dimensional transform matrix that is applied to its sublayers. Using these projections, you can create stunning user interfaces that communicate depth to the user (see Figure 3-2).

Figure 3-2 Core Animation 3D interface



Animating Layers

Core Animation provides support for animating a layer's visual properties in two ways, implicitly and explicitly.

Implicit animations occur automatically in response to setting a new value for an animatable layer property. Core Animation assumes full responsibility for running the animation, at frame rate, in a separate thread, freeing your application to handle other events. For example, by assigning the `frame` property the `newFrame` value, the `textLayer` object animates smoothly to the new location.

```
textLayer.frame=newFrame;
```

Each of a layer's animatable properties has a related implicit animation. You can override a default animation to supply your own custom animations for a layer property. You can also disable the animation of any layer property, either temporarily or permanently.

Explicit animation is achieved by creating an instance of a Core Animation animation class and specifying a target layer and, optionally, a property. Explicitly animating a layer property affects only the presentation value of the property; the actual value of the layer property does not change. For example, to draw attention to a layer, you might make it spin 360° by animating the transformation matrix. This animation affects the display of the layer, but its transform matrix remains untouched.

Layout Management

Core Animation layers support the classic Cocoa view model of positioning layers relative to their superlayer—a style known as "springs and struts". In addition Core Animation also provides a more general layout manager mechanism that allows you to write your own layout managers. A custom layout manager assumes responsibility for providing layout of the associated layer's sublayers.

Using a custom layout manager, your application can create complex animations by taking advantage of implicit animations. Updating the position, size, or transform matrix of a layer causes it to animate to the new settings. The CoverFlow style of animation is accomplished with a custom layout manager.

Core Animation provides a constraint layout manager class that arranges layers using a set of constraints. Each specified constraint describes the relationship of one geometric attribute of a layer (the left, right, top, or bottom edge or the horizontal or vertical center) to a geometric attribute of one of its sibling layers or its superlayer. For example, using the constraint layout manager you can define a layout where layer A is always centered and 5 points below layer B. Moving layer B to a new position automatically causes layer A" to reposition itself relative to layer B.

NSView Integration with Core Animation

The Cocoa `NSView` class is integrated with Core Animation and layers in two ways. The first type of integration is **layer hosting**. A layer-hosting view displays a Core Animation layer tree that is set by an application. It is the application's responsibility to interact with the layer content by manipulating the layers directly. A layer hosting view is how an application displays Core Animation user interfaces.

You specify that the view `aView` is the layer host of the Core Animation layer `rootLayer` as follows:

```
// aView is an existing view in a window
// rootLayer is the root layer of a layer tree

[aView setLayer:rootLayer];
[aView setWantsLayer:YES];
```

When working with a layer-hosting view, you typically subclass `NSView` and handle user-generated events such as keypresses and mouse clicks in that subclass, manipulating the Core Animation layers as necessary.

The second type of `NSView` and Core Animation integration is **layer-backed views**. Layer-backed views use Core Animation layers as their backing store, freeing the views from the responsibility of refreshing the screen. The views need to redraw only when the view content actually changes. Enabling layer-backing for a view and its descendants is done using the following code:

```
// aView is an existing view in a window
[aView setWantsLayer:YES];
```

When layer backing is enabled for a view, the view and all its subviews are mirrored by a Core Animation layer tree. The views can be standard Aqua controls or custom views. The view and its subviews still take part in the responder-chain, still receive events, and act as any other view. However, when redrawing needs to be done and the content has not changed, the render tree handles the redraw rather than the application.

In addition to providing cached redrawing, layer-backed views expose a number of the advanced visual properties of Core Animation layer properties, including:

- Control over the view's opacity
- An optional shadow, specified using an `NSShadow` object
- An optional array of Core Image filters that are applied to the content behind a view before its content is composited
- A Core Image filter used to composite the view's contents with the background
- An optional array of Core Image filters that are applied to the contents of the view and its subviews

Note: When using a view in layer-backed mode, you should not directly manipulate the layers, instead use the layer methods exposed by the `NSView` class.

For more information on layer-backed views, see *View Programming Guide for Cocoa*.

Cocoa Animation Proxies

Taking a cue from Core Animation, the Application Kit supports the concept of a generalized animation facility through the use of **animation proxies**.

Classes that support the `NSAnimatablePropertyContainer` protocol provide implicit animation support for any property that is key-value coding compliant. By operating through the animation proxy, you can perform implicit animation just as you can in Core Animation.

For example, the following code snippet causes `aView` to be displayed at its new location:

```
[aView setFrame:NSMakeRect(100.0,100.0,300.0,300.0)];
```

However, using an animation proxy and the same view method, `aView` animates smoothly to its new location.

```
[[aView animator] setFrame:NSMakeRect(100.0,100.0,300.0,300.0)];
```

Unlike Core Animation, which allows only animation of properties that have a direct mapping to a render-tree property, the Application Kit allows any key-value coding compliant property to be animated if its class implements the `NSAnimatablePropertyContainer` protocol. Animating these properties does require the view to redraw its contents, but is still much easier than animating these properties manually. As with Core Animation layers you can change the default implicit animation for the animated properties.

Note: View properties of the types float, double, `NSPoint`, `NSSize`, or `NSRect` are animatable using the proxy returned the `NSAnimatablePropertyContainer` method `animator` without requiring that the view have layer-backing enabled.

For additional information on Cocoa animation see *Animation Programming Guide for Cocoa*.

Transitioning to and from Full Screen Mode

The `NSView` class provides methods that allow you to easily provide a transition animation that is used when a view transitions to full screen mode and back again. The method `enterFullScreenMode:withOptions:` allows you to specify the target screen that the view should take over.. Exiting full screen mode is done using `exitFullScreenModeWithOptions:`. You can test whether a view is in full screen mode using the method `isInFullScreenMode`.

Additional Animation Support

The `NSAnimation` and `NSViewAnimation` classes provide a much simpler, and less powerful, animation capability than the `NSAnimatablePropertyContainer` protocol.

`NSAnimation` provides basic timing functionality and progress curve computation, a delegate mechanism for influencing animation behavior in simple ways, and facilities for chaining component animations together.

`NSViewAnimation` is a concrete subclass of `NSAnimation` that provides for animating view frame changes (thus, both position and dimensions) and performing simple fade-in and fade-out effects, using any of four predefined progress curves. View animations may be blocking, asynchronous, or threaded asynchronous, but successive incremental changes as view frame coordinates are interpolated take effect immediately in the view instances themselves.

If your application is intended for deployment on Mac OS X v10.5 or later, you'll find that the animation proxies provided by the `NSWindow` and `NSView` classes are typically more convenient to use than using `NSAnimation` directly.

`NSAnimation` and `NSViewAnimation` are available in Mac OS X v10.4 and later. For additional information on `NSAnimation` and `NSViewAnimation`, see *Animation Programming Guide for Cocoa*.

Choosing the Animation Technology for Your Application

Choosing which animation technology your application should use can be challenging.

This chapter provides guidelines that will help you determine whether your application should use Core Animation, non-layer-backed Cocoa views, layer-backed Cocoa views, or a combination of the technologies. It also provides a summary of the capabilities offered by Core Animation layers and layer-backed Cocoa views.

Guidance

When adding animation to your applications, the following guidelines will help you choose the appropriate technology:

- For the portions of your user interface that only require static, non-rotated, Aqua controls, use Cocoa views.
- For the portions of your user interface that require simple animation of a view or window's frame, consider using the animator proxy support provided by Cocoa views and windows.
- For the portions of your user interface that require the ability to display and interact with rotated Aqua controls, use layer-backed Cocoa views.
- If performance testing indicates that your custom view is processor or time intensive, consider using layer-backed Cocoa views. Remember that layer-backed views use more memory than non-layer-backed views.
- If your custom views requires extensive user events handling consider using layer-backed Cocoa views.
- For the portions of your user interface that have considerable graphics and animation requirements and does not rely on Aqua controls, consider using Core Animation layers directly. Using layers directly does require you to handle all user events (mouse and keyboard interaction) yourself by overriding the view that hosts the root layer.

Core Animation layers, on the other hand, have the advantage of being computationally much lighter-weight than instances of Cocoa views. This allows you to create interfaces from smaller constituent parts when using Core Animation layers. This can also give Core Animation layers the edge when your application needs to display hierarchies consisting of thousands of layer objects.

Hybrid View/Core Animation Applications

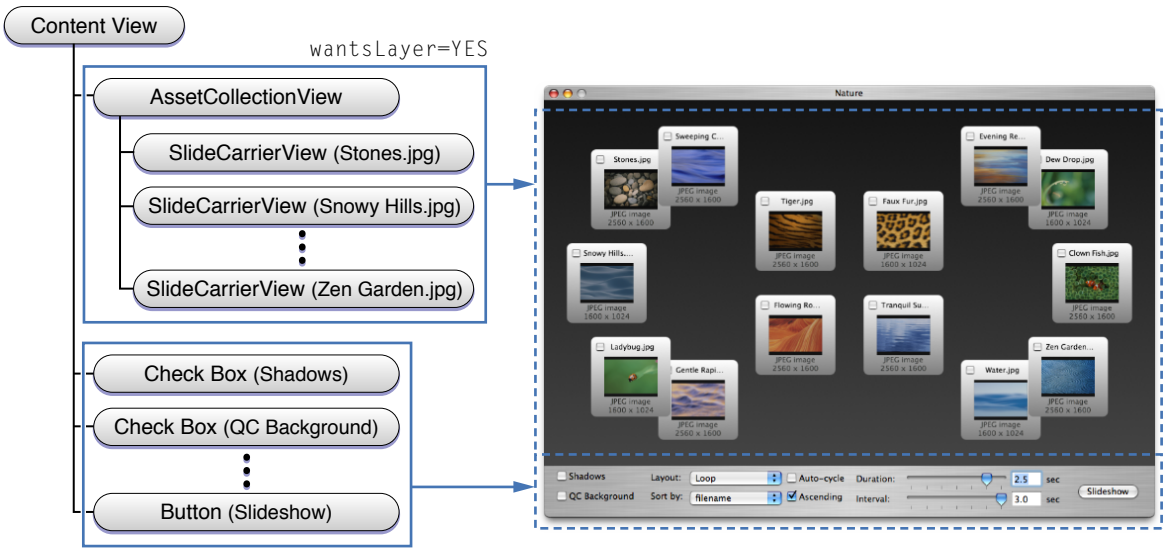
You can use a combination of layer-backed views, non-layer-backed views, and layer-hosting views in a single user interface with two caveats:

- All the subviews of layer-backed views are automatically layer backed. To improve performance, you should avoid making views descendants of layer-backed views if they don't require animation or cached drawing.
- You must host your custom Core Animation layers in a single layer-hosting view rather than inserting them into a layer hierarchy managed by a layer-backed view.

Example: A Hybrid Application

The *CocoaSlides* sample is an example of a hybrid view/layer-backed view user interface. Figure 4-1 shows the relevant portion of the view hierarchy.

Figure 4-1 Hybrid layer-backed view and conventional view user interface



The `AssetCollectionView` view is a layer-backed view, and as a result all the `SlideCarrierView` subviews are also layer-backed. Layer-backed views are used for this portion of the user interface because:

- The slides require animation that includes rotation; this requirement precludes using non-layer-backed views, because the view animation proxies do not support rotation unless the view is layer-backed.
- The slides incorporate a standard Aqua checkbox; this fact precludes using Core Animation directly because it doesn't provide Aqua controls. Replicating the look and feel of an Aqua control in a Core Animation layer is not recommended; this may change in a future version of Mac OS X.

  Using a non-layer-backed view is not an option; The Aqua checkbox must function, even when rotated, and that is not supported.
- The slides are complex to draw; by using a layer-backed view; the slide content is redrawn by the application only when it changes.

The portion of the user interface at the bottom of the window is made up of standard Aqua controls that are not layer-backed. There is no need for these controls to be in a layer-backed view hierarchy; they are never rotated and would not benefit significantly by using cached drawing.

Summary of Layer and View Capabilities

The following is a brief summary of the shared capabilities of Core Animation layers and layer-backed views, including the advantages of one over the other.

Common Capabilities

- Both layers and layer-backed views support a nested hierarchy model with parent-relative coordinate systems.
- Both layers and layer-backed views support cached contents that require no application interaction for damage redrawing.
- Both layers and layer-backed views support the full range of media types.
- Both layers and layer-backed views support overlapping of layers/views with other layers/views outside of the sublayer/subview.

Layer Advantages

- Layers support several visual properties that are not exposed in layer-backed views. For example, corner radius and borders.
- Sublayers can reside outside of a layer's bounds.
- Layers support complex masking.
- Layers are computationally lightweight.
- Layers support complex transforms.
- Layout managers provide more flexible control over layer placement.

Advantages of Layer-Backed Views

- Layer-backed views participate in the responder chain.
- Layer-backed views receive user events.
- Layer-backed views support drag and drop.
- Layer-backed views support accessibility.
- Layer-backed views support all the standard Aqua controls and views.
- Layer-backed views support scrolling.
- Layer -backed views support text input.

Glossary

animation Animation is a visual technique that provides the illusion of motion by displaying a collection of images in rapid sequence.

animation proxy An animation proxy "stands in" for an object and provides animation capabilities without significantly impacting the original objects API.

basic animation A simple animation from a start value to a target value.

Core Image The framework that provides image processing filters used to process still and video images.

duration The length of time, in seconds, it takes for an animation to complete.

keyframe animation An animation that specifies an array of values that an animation uses as sequential targets.

interpolation The calculation of intermediate values relative to known beginning and ending values.

layer-backed view An instance of an `NSView` object that uses a Core Animation layer to cache its drawing content. The view is responsible for managing the layer tree, the developer should not manipulate the layer tree directly.

layer-hosting view An instance of an `NSView` object that hosts a Core Animation layer tree. The developer is responsible for managing the layer tree directly.

OpenGL An open source graphics library. For more information see http://www.opengl.org/.

pacing The distribution of the interpolated values of an animation across the duration of the animation.

transition animation An animation that uses a Core Image filter to apply a visual effect to an animation object being displayed or hidden.

Document Revision History

This table describes the changes to *Animation Overview*.

Date	Notes
2008-10-15	Corrected typos.
2008-04-08	Corrected typos.
2007-10-31	Added QuickTime movie sample. Added new terms to the glossary.
2007-05-26	New document that describes the animation facilities provided by Mac OS X.

Document Revision History