Improving Animation Performance

Core Animation is a great way to improve the frame rates for app-based animations but its use is not a guarantee of improved performance. Especially in OS X, you must still make choices about the most effective way to use Core Animation behaviors. And as with all performance-related issues, you should use Instruments to measure and track the performance of your app over time so that you can ensure that performance is improving and not regressing.

Choose the Best Redraw Policy for Your OS X Views

The default redraw policy for the `NSView` class preserves the original drawing behavior of that class, even if the view is layer-backed. If you are using layer-backed views in your app, you should examine the redraw policy choices and choose the one that offers the best performance for your app. In most cases, the default policy is not the one that is likely to offer the best performance. Instead, the `NSViewLayerContentsRedrawOnSetNeedsDisplay` policy is more likely to reduce the amount of drawing your app does and improve performance. Other policies might also offer better performance for specific types of views.

For more information on view redraw policies, see The Layer Redraw Policy for OS X Views Affects Performance.

Update Layers in OS X to Optimize Your Rendering Path

In OS X v10.8 and later, views have two options for updating the underlying layer's contents. When you update a layer-backed view in OS X v10.7 and earlier, the layer captures the drawing commands from the view's `drawRect:` method into the backing bitmap image. Caching the drawing commands is effective but is not the most efficient option in all cases. If you know how to provide the layer's contents directly without actually rendering them, you can use the `updateLayer` method to do so.

For information about the different paths for rendering, including those that involve the `updateLayer` method, see Using a Delegate to Provide the Layer's Content.

General Tips and Tricks

There are several ways to make your layer implementations more efficient. As with any such optimizations, though, you should always measure the current performance of your code before attempting to optimize. This gives you a baseline against that you can use to determine if the optimizations are working.

Use Opaque Layers Whenever Possible

Setting the `opaque` property of your layer to `YES` lets Core Animation know that it does not need to maintain an alpha channel for the layer. Not having an alpha channel means that the compositor does not need to blend the contents of your layer with its background content, which saves time during rendering. However, this property is relevant primarily for layers that are part of a layer-backed view or situations where Core Animation creates the underlying layer bitmap. If you assign an image directly to the layer's `contents` property, the alpha channel of that image is preserved regardless of the value in the `opaque` property.

Use Simpler Paths for CAShapeLayer Objects

The `CAShapeLayer` class creates its content by rendering the path you provide into a bitmap image at composite time. The advantage is that the layer always draws the path at the best possible resolution but that advantage comes at the cost of additional rendering time. If the path you provide is complex, rasterizing that path might get too expensive. And if the size of the layer changes frequently (and thus must be redrawn frequently), the amount of time spent drawing can add up and become a performance bottleneck.

One way to minimize drawing time for shape layers is to break up complex shapes into simpler shapes. Using simpler paths and layering multiple `CAShapeLayer` objects on top of one another in the compositor can be much faster than drawing one large complex path. That is because the drawing operations happen on the CPU whereas compositing takes place on the GPU. As with any simplifications of this nature, though, the potential performance gains are dependent on your content. Therefore, it is especially important to measure the performance of your code before optimizing so that you have a baseline to use for comparisons.

Set the Layer Contents Explicitly for Identical Layers

If you are using the same image in multiple layer objects, load the image yourself and assign it directly to the `contents` property of those layer objects. Assigning an image to the `contents` property prevents the layer from allocating memory for a backing store. Instead, the layer uses the image you provide as its backing store. When several layers use the same image, this means that all of those layers are sharing the same memory rather than allocating a copy of the image for themselves.

Always Set a Layer's Size to Integral Values

For best results, always set the width and height of your layer objects to integral values. Although you specify the width and height of your layer's bounds using floating-point numbers, the layer bounds are ultimately used to create a bitmap image. Specifying integral values for the width and height simplifies the work that Core Animation must do to create and manage the backing store and other layer information.

Use Asynchronous Layer Rendering As Needed

Any drawing that you do in your delegate's `drawLayer:inContext:` method or your view's `drawRect:` method normally occurs synchronously on your app's main thread. In some situations, though, drawing your content synchronously might not offer the best performance. If you notice that your animations are not performing well, you might try enabling the `drawsAsynchronously` property on your layer to move those operations to a background thread. If you do so, make sure your drawing code is thread safe. And as always, you should always measure the performance of drawing asynchronously before putting it into your production code.

Specify a Shadow Path When Adding a Shadow to Your Layer

Letting Core Animation determine the shape of a shadow can be expensive and impact your app's performance. Rather than letting Core Animation determine the shape of the shadow, specify the shadow shape explicitly using the `shadowPath` property of `CALayer`. When you specify a path object for this property, Core Animation uses that shape to draw and cache the shadow effect. For layers whose shape never changes or rarely changes, this greatly improves performance by reducing the amount of rendering done by Core Animation.