I spent some time yesterday fixing some performance issues related to EveryCollegCal (my iOS app). Basically, I was seeing very slow, choppy segues between the first two views (one of which is a welcome screen, the other a table view). On top of that, the scrolling performance of the latter view was equally gross.

In the end, the fixes were not too involved—i.e., they didn’t involve a spectacular amount of code (of course, this can’t be said of all performance issues, so I count myself lucky). And there were only two necessary changes: the choppy segues were fixed by swapping out my glossy button library, which was apparently very inefficient. But the second change demonstrates something cool about iOS performance tweaks: while they’re often subtle, the under-the-hood explanations are always logical.

Drop Shadows

I thought for a while that my poor scrolling performance was due to the cells themselves, which use resizable assets to create a gradient background. But after rolling the app with certain features of the table disabled, **I figured out that the issue was caused by the drop shadow in my cells’ accessories—such a tiny detail.**

It’s very easy to add drop shadows to your views in iOS. What’s not so easy is making sure that they have good performance.

Previously, I was adding a drop shadow with CALayer.

disclosureIndicator.layer.shadowColor = [UIColor colorWithRed:0.2 green:0.2 blue:0.2 alpha:0.5].CGColor;

disclosureIndicator.layer.shadowOffset = CGSizeMake(0, 1);

disclosureIndicator.layer.shadowOpacity = 1.0;

disclosureIndicator.layer.shadowRadius = 1;

disclosureIndicator.layer.masksToBounds = NO;

This looks great and requires minimal code but—as outlined here—there’s a huge expense associated the process, namely because you must render (offscreen) the exact shape of the view; i.e., your device will spend time figuring out if your view is circular, rectangular, hexagonal, etc.

The fix: specify the shape of the shadow using a **Bezier path**.

disclosureIndicator.layer.shadowPath = [UIBezierPath bezierPathWithOvalInRect:disclosureIndicator.layer.bounds].CGPath;

It’s a simple fix, but I can’t emphasize enough how drastically it improved my table’s scroll performance. And the reasoning behind it is simple enough: with the latter method, you save the machine having to calculate the shape of an object.

This is a great example of iOS performance tweaks: subtle, but logical.