Practical Guide to Optimization for Mobiles

This guide is for developers new to mobile game development, who are probably feeling overwhelmed, and are either planning and prototyping a new mobile game or porting an existing project to run smoothly on a mobile device. It should also be useful as a reference for anyone making mobile games or browser games which target old PCs and netbooks.

Optimization is a broad topic, and how you do it depends a lot on your game, so this guide is best read as an introduction or reference rather than a step-by-step guide that guarantees a smooth product.

All mobile devices are not created equal

The information here assumes hardware around the level of the Apple A4 chipset, which is used on the original iPad, the iPhone 3GS, and the 3rd generation iPod Touch. On the Android side, that would mean an Android phone such as the Nexus One, or most phones that run Android 2.3 Gingerbread. On average, these devices were released in early 2010. Out of the app-hungry market, these devices are the older, slower portion. But they should be supported, because they represent a large portion of the market.

There are much slower, and much faster phones out there as well. The computational capability of mobile devices is increasing at an alarming rate. It's not unheard of for a new generation of a mobile GPU to be five times faster than its predecessor. That's ***fast***, when compared to the PC industry.

For an overview of Apple mobile device tech specs, see the Hardware page.

If you want to develop for mobile devices which will be popular in the future, or exclusively for high end devices right now, you will be able to get away with doing more. See Future Mobile Devices.

The very low end, such as the iPhone 3G and the first and second generation iPod touches, are extremely limited and even more care must be taken to optimize for them. However, there is some question to whether consumers who have not upgraded their device will be buying apps. So unless you are making a free app, it might not be worthwhile to support the old hardware.

Make optimization a design consideration, not a final step

British computer scientist Michael A. Jackson is often quoted for his Rules of Program Optimization:

The First Rule of Program Optimization: Don't do it. The Second Rule of Program Optimization (for experts only!): Don't do it yet.

His rationale was that, considering how fast computers are, and how quickly their speed is increasing, there is a good chance that if you program something it will run fast enough. Besides that, if you try to optimize too heavily, you might over-complicate things, limit yourself, or create tons of bugs.

However, if you are developing mobile games, there is another consideration: The hardware that is on the market right now is very limited compared to the computers we are used to working with, so the risk of creating something that simply won't run on the device balances out the risk of over-complication that comes with optimizing from the start.

Throughout this guide we will try to point out situations where an optimization would help a lot, versus situations where it would be frivolous.

Optimization: Not just for programmers

Artists also need to know the limitations of the platform and the methods that are used to get around them, so they can make creative choices that will pay off, and don't have to redo work.

- More responsibility can fall on the artist if the game design calls for atmosphere and lighting to be drawn into textures instead of being baked.
- Whenever anything can be baked, artists can produce content for baking, instead of real-time rendering. This allows them to ignore technical limitations and work freely.

Design your game to make a smooth runtime fall into your lap

These two pages detail general trends in game performance, and will explain how you can best design your game to be optimized, or how you can intuitively figure out which things need to be optimized if you've already gone into production.

- Practical Methods for Optimized Rendering
- Practical Methods for Optimized Scripting and Gameplay

Profile early and often

Profiling is important because it helps you discern which optimizations will pay off with big performance increases and which ones are a waste of your time. Because of the way that rendering is handled on a separate chip (GPU), the time it takes to render a frame is not the time that the CPU takes plus the time that the time that the GPU takes, instead it is the longer of the two. That means that if the CPU is slowing things down, optimizing your shaders won't increase the frame rate at all, and if the GPU is slowing things down, optimizing physics and scripts won't help at all.

Often different parts of the game and different situations perform differently as well, so one part of the game might cause 100 millisecond frames entirely due to a script, and another part of the game might cause the same slowdown, but because of something that is being rendered. So, at very least, you need to know where all the bottlenecks are if you are going to optimize your game.

Unity Profiler (Pro only)

The main Profiler in Unity can be used when targeting iOS or Android. See the Profiler guide for basic instructions on how to use it.

Internal Profiler

The internal profiler spews out text every 30 frames. It can help you figure out which aspects of your game are slowing things down, be it physics, scripts, or rendering, but it doesn't go into much detail, for example, which script or which renderer is the culprit.

See the Internal Profiler page for more details on how it works and how to turn it on.

Profiler indicates most of time spent rendering

- Rendering Optimizations

Profiler indicates most of time spent outside of rendering

- Scripting Optimizations

Table of Contents

- Practical Guide to Optimization for Mobiles - Scripting and Gameplay Methods
- Practical Guide to Optimization for Mobiles - Rendering Optimizations
- Practical Guide to Optimization for Mobiles - Optimizing Scripts

Page last updated: 2012-08-24