



Optimizing Content for Flash Lite 2.0

Josh Ulm
Principal Designer
Experience Lead, Mobile and Devices

Contributors:
Rosalind Morrison, Adobe Consulting; Walter Luh, Flash Lite Engineering; Jian Zheng, Flash Lite Engineering; Jeremy Clark, XD

Contents

INTRODUCTION.....	3
KNOWN ISSUES	4
<i>Frame rate</i>	4
THINGS TO REMEMBER	5
<i>Stage size and publishing profiles</i>	5
<i>Intervals and listeners</i>	5
<i>Skipping frames</i>	5
<i>Global and local variables</i>	5
<i>Anonymous functions</i>	5
<i>Scaling and rotating bitmaps</i>	5
<i>Gradient banding</i>	5
OPTIMIZING CONTENT	6
<i>Artwork formats</i>	6
<i>Compressed JPGs</i>	6
<i>Transparent PNGs</i>	6
<i>Gradients</i>	6
<i>Vector complexity</i>	6
<i>Vector corners and curves</i>	6
<i>Shape outlines</i>	6
<i>Embedded text</i>	6
<i>Multiline dynamic text</i>	7
<i>Hiding movie clips</i>	7
<i>Tween regions</i>	7
<i>Layer types</i>	7
<i>Clean source files</i>	7
<i>First frame initialization</i>	7
<i>Math and floating point numbers</i>	7
<i>Math routine data</i>	8
<i>Loop iteration</i>	8
<i>XML data</i>	8
ABOUT THE AUTHOR.....	9

Introduction

This document brings together many tips and insights for creating lean and fast Flash content for mobile devices using Flash Lite 2.0.

Most Flash developers are well aware of the universal truths for optimizing Flash content, such as don't animate huge complex artwork, don't tween a zillion things at once, and don't overuse transparency. On the desktop Flash 8 (finally!) solves these performance issues. However, Flash Lite developers have a more complicated situation to deal with. On devices we just don't have the oomph we need to kick out the crazy Hollywood effects, at least not yet. Furthermore, some devices perform better than others, sometimes dramatically. And because mobile authoring often requires that we publish to many different devices, it may be the case that we have to author for the lowest common denominator. That being said, a byte here and there may make all the difference.

Many recommendations will suggest that you avoid using some common authoring techniques. Clearly Flash wouldn't be Flash without vectors, dynamic text, and animation, so don't walk away distressed that these features are off-limits – they aren't. But you will have to experiment with what works and what doesn't before you know how much is too much. As you go through this document, keep in mind that optimizing mobile content often requires measuring the tradeoffs. Technique "A" may look better, while technique "B" results in much better performance. To get there, you'll need to understand and expect that the process of optimizing mobile content requires a lot of back and forth on a target device. There is no substitute for testing your files on real hardware; there is no other way to confirm actual performance, true colors, text readability, physical interactions, UI responsiveness, and ultimately, the real mobile experience.

Known issues

Frame rate

A host's timer-resolution fundamentally affects the maximum frame rates achievable by Flash Lite on that device. Therefore, the fps specified at the authoring level (expected fps) will **not** translate into an actual fps, even for SWFs that are empty (no rendering or AS)! There is an adjustment that needs to be made to the expected fps to obtain the true, maximum fps you should expect from your content.

Therefore, set the frame rate to an fps that is native to the target device. If you are not sure of your target device's ideal frame rates, you should test different frame rates to see if adjusting them significantly affects the achievable frame rate of your movie.

An example will help illustrate this: On Symbian Series 60v2-based phones, the timer resolution is 1/64 second. That means *DoPlay* can get called at the following time intervals: 1/64, 2/64, 3/64, 4/64, 5/64, etc. Hence, the frame rates (fps) that are available on such devices are: 64, 32, 21.3, 16, 12.8, etc.

So when content authored at 20 fps runs, it theoretically should be called every 1/20 second. However, Symbian only has a 1/64 second resolution, so 1/20 falls between the 3/64 and 4/64 intervals that Symbian can provide. The player is conservative and will only advance frames when the 1/20 interval is exceeded, i.e. after 4/64 seconds has elapsed. This corresponds to 16 fps! That's a 20% adjustment – or loss in **perceived** performance!

Things to remember

Stage size and publishing profiles

When creating a new movie, be sure your document is set up correctly. Although Flash movies can scale smoothly, there is a performance hit if the movie is not running at its native stage size and has to scale in the player. Make sure you set the document's stage size to match your target device's resolution. Also be sure to set your Flash Player to the correct version of Flash Lite under Publish Settings and select an appropriate device profile in Device Central.

Intervals and listeners

SWF data memory will not be re-collected if any ActionScript functions are still referring to the SWF data when the movie clip is unloaded. Using intervals and listeners are two such cases where the data will not be freed unless the ActionScript is cleared first. Be sure to clear any active intervals by using *clearInterval* and remove any active listeners using *removeListener* before removing content with *unloadMovie* or *removeMovieClip*.

Skipping frames

When using *gotoAndPlay* remember that every frame in between the current frame and the requested frame will need to be initialized prior to the requested frame being played. If every frame in between the two frames contains different content, it may be more efficient to use different movie clips rather than the timeline.

Global and local variables

Within functions, local variables are registered in a way that makes them much faster than global variables for the player to get and set. Use *var* whenever possible.

Anonymous functions

Avoid defining functions using anonymous syntax: *myObj.eventName = function() { ... };*
Explicitly defined functions are more efficient: *function myFunc() { ... }; myObj.eventName = myFunc;*

Scaling and rotating bitmaps

The Flash Lite player does not support bitmap smoothing. This means that if you scale or rotate a bitmap, it will have a chunky appearance. It is best to use bitmaps at their native rotation and resolution. If you have a graphic that you need to scale or rotate, consider using a vector graphic instead.

Gradient banding

The majority of devices currently on the market still only support 16-bit color (thousands), not 24 or 32-bit (millions). That means that gradients will often appear as banded stripes of solid color instead of as a smooth graduated transition. One way to deal with this is to post-process your bitmaps with a free 3rd party Photoshop filter by Telegraphics called *5_6_5*, which reduces the color depth of a bitmap to 16-bit and dithers it. That filter can be found here: <http://www.telegraphics.com.au/sw/>.

Optimizing content

Artwork formats

Whenever possible, use bitmaps instead of vectors for artwork. The rendering of multiple vectors reduces performance, whereas bitmaps can be rendered much faster.

Compressed JPGs

The decompression of JPGs will impede performance. Therefore, if memory allows, try to use PNGs.

Transparent PNGs

Be sure to minimize the amount of transparency in PNG files – the player has to calculate redraws even for the transparent portions of the bitmap. For example, if you have a transparent PNG that represents a foreground element, don't export the transparent PNG at the full size of the screen. Export it at the actual size of the foreground element.

Gradients

Minimize the use of vector gradients. They are costly for the player to calculate and render. If you must use them, note that linear gradients render faster than radial gradients.

Vector complexity

When you absolutely must use vector shapes, optimize them as much as possible. The more dimension a shape has, the more Flash will have to do to render that shape. Optimization can be especially helpful with small vector shapes such as icons. Your icon might be so small that lots of the detail is being lost, yet if the complexity of the shape is retained that's extra work for the player to render. In most cases using bitmaps instead of vectors is a better solution.

Vector corners and curves

Corners can be mathematically simpler to render than curves. When possible, stick with flat edges, especially with very small vector shapes.

Shape outlines

Whereas a fill has only an outside shape to render, outlines have an inside and an outside to render. This means twice as much work to draw a line versus a fill. Avoid them when possible.

Embedded text

Text is essentially just a very complex vector shape. This makes type one of the more complicated shapes for Flash to render. Of course text is often essential, so it can rarely be avoided entirely. When you do use text avoid animating it or placing it over an animation, and consider using text as a bitmap.

Multiline dynamic text

Line breaking in the player is a very time consuming process. Static text fields are not problematic because the line breaking is pre-calculated at compile time. But for multiline dynamic and input text, the line breaking of the text string is not cached. Line breaking is done in the player at runtime and is recalculated every time the text field needs to be redrawn. For dynamic content, using dynamic text fields is unavoidable, but when possible consider using static text fields instead.

Hiding movie clips

Avoid using `_alpha = 0` and `_visible = false` to hide onscreen movie clips. If you simply turn a movie clip's visibility off or change its alpha to zero, it is still included in the player's scanline rendering calculations which can affect performance. Similarly, do not try to hide a movie clip by obscuring it behind another piece of artwork. It will still be included in the player's calculations. Instead, move movie clips completely off-stage or remove them using `removeMovieClip`.

Tween regions

When Flash draws an animated region, it does so by defining a rectangular bounding box around the area. You can optimize this drawing by making that rectangle as small as possible. This means that if you can, you should avoid overlapping tweens since Flash will see the merged area as a single rectangle resulting in a larger total region. Use Flash 8's 'show redraw region' feature within the player to optimize your animation.

Layer types

Try to arrange bitmap layers and vector layers near each other whenever possible. As the player renders the movie it needs to implement different renderers depending on the type of content. That switch between renderers takes time, not a lot, but if it happens a lot, it could add up. By ordering the layers that have vector content on them near each other and doing the same for bitmaps the player can render them faster by switching less.

Clean source files

This should go without saying, but certainly you'll want to keep your movie as small as possible and remove any extraneous content and code before compiling. Make sure unused movie clips get removed, kill unnecessary frame and code loops, and avoid too many or extraneous frames. Believe it or not those empty frames can really add up!

First frame initialization

Although preloading all your content by putting it at the beginning of your movie makes sense on the desktop, doing this on a mobile device can result in a movie that is slow to start. Space your content naturally throughout your movie so that movie clips are initialized as they are used.

Math and floating point numbers

Minimize the use of Math functions and floating point numbers. The calculation of these functions will slow the performance of the content.

Math routine data

If you have to use the Math routines, consider pre-calculating the values and storing them in an array of variables. Pulling those values from a data table is much faster than having the player calculate them at runtime.

Loop iteration

Running *for* loops can be expensive because of the overhead incurred while the condition is checked with each iteration. When the costs of the iteration and the loop overhead are comparable, unroll your loops to execute multiple operations individually. This will lead to larger code size, but faster performance.

XML data

Avoid loading and parsing XML files if possible; these consume the processor and affect performance. If possible, store data in simple name/value pairs and load them from a text file using *loadVars* or from precompiled SWF files.

About the author

Josh Ulm is Principal Designer and Design Lead of Mobile and Devices for Adobe Systems, Inc. Since joining Macromedia in 2004 and following with Adobe, he has worked predominately with Mobile and Devices to define the mobile experience platform and works directly with developers and customers to create engaging Flash Lite experiences. His work has driven the successful adoption of many products and technologies for the combined companies and their customers; he is frequently asked to present and develop the company's experience vision; and is an active and respected veteran within the Flash and mobile developer communities.