MOTODEV

The Motorola developer network

OPTIMIZING A JAVA ME APPLICATION PART 3: CANVAS PERFORMANCE IMPROVEMENT

TECHNICAL ARTICLE

Copyright © 2006, Motorola, Inc. All rights reserved. This documentation may be printed and copied solely for use in developing products for Motorola products. In addition, two (2) copies of this documentation may be made for archival and backup purposes. Except for the foregoing, no part of this documentation may be reproduced or transmitted in any form or by any means or used to make any derivative work (such as translation, transformation, or adaptation) without express written consent from Motorola, Inc.

Motorola reserves the right to make changes without notice to any products or services described herein. "Typical" parameters, which may be provided in Motorola Data sheets and/or specifications, can and do vary in different applications and actual performance may vary. Customer's technical experts will validate all "Typicals" for each customer application.

Motorola makes no warranty in regard to the products or services contained herein. Implied warranties, including without limitation, the implied warranties of merchantability and fitness for a particular purpose, are given only if specifically required by applicable law. Otherwise, they are specifically excluded.

No warranty is made as to coverage, availability, or grade of service provided by the products or services, whether through a service provider or otherwise. No warranty is made that the software will meet your requirements or will work in combination with any hardware or application software products provided by third parties, that the operation of the software products will be uninterrupted or error free, or that all defects in the software products will be corrected.

In no event shall Motorola be liable, whether in contract or tort (including negligence), for any damages resulting from use of a product or service described herein, or for any indirect, incidental, special or consequential damages of any kind, or loss of revenue or profits, loss of business, loss of information or data, or other financial loss arising out of or in connection with the ability or inability to use the Products, to the full extent these damages may be disclaimed by law.

Some states and other jurisdictions do not allow the exclusion or limitation of incidental or consequential damages, or limitation on the length of an implied warranty, therefore the above limitations or exclusions may not apply to you.

This warranty gives you specific legal rights, and you may also have other rights, which vary from jurisdiction to jurisdiction.

Motorola products or services are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Motorola product or service could create a situation where personal injury or death may occur.

Should the buyer purchase or use Motorola products or services for any such unintended or unauthorized application, the buyer shall release, indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the designing or manufacturing of the product or service.

Motorola recommends that if you are not the author or creator of the graphics, video, or sound, you obtain sufficient license rights, including the rights under all patents, trademarks, trade names, copyrights, and other third party proprietary rights.

If this documentation is provided on compact disc, the other software and documentation on the compact disc are subject to the license agreement accompanying the compact disc.

Optimizing a Java ME Application Part 3: Canvas Performance Improvement

2/1/2007

For the latest version of this document, visit http://developer.motorola.com

Motorola, Inc.

http://www.motorola.com

Contents

List of Figures

List of Tables

About this Document

Document Overview

Introduction: In Java ME devices, processing speed and memory space are limited; performance improvement will make the application run in a fluid and robust way. This document includes two parts,

- **Introduction to Motorola SDK's profiler:** Introduction about how to use the profiler in the Motorola SDK. The profiler makes possible the monitoring of the running state of a Java ME application. It is a vital tool in helping to optimize a Java ME application.
- **Performance improvement tips:** Overview of the CLDC Hotspot mechanism and some technical tips for performance improvement. An example is provided to show how to optimize a Java ME application with the profiler.

Definitions, Abbreviations, Acronyms

SDK	Software Development Kit

An Introduction to the Motorola SDK's profiler

Profiling function in Motorola Linux OS SDK

The profiler in Motorola SDK for Linux OS Products can be invoked with the menu Profiling->Active Mode. It has a memory monitor, which shows the total memory and free memory in KVM.

Figure 1: Motorola Linux OS product SDK profiling menu

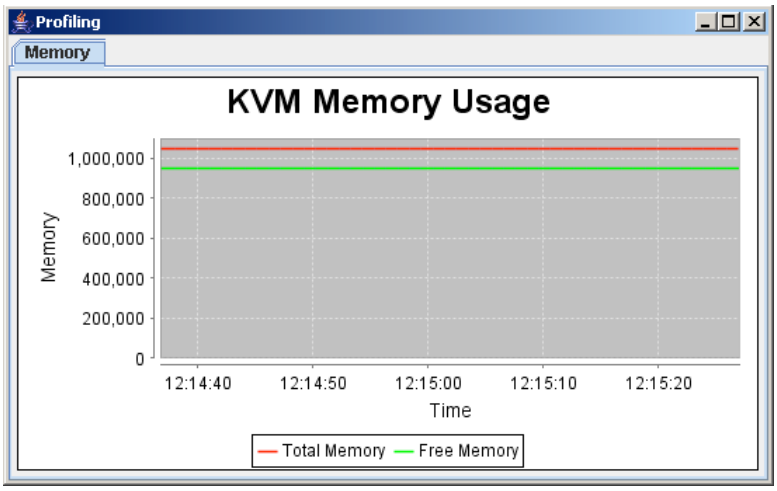

Figure 2, Motorola Linux OS product SDK memory monitor

Code to get total and free memory in real device,

```
Runtime.getRuntime().totalMemory();
Runtime.getRuntime().freeMemory();
```

Performance improvement tips and example

CLDC Hotspot mechanism overview

In the Motorola A1200 and ROKR E2, the Java virtual machine is Sun's CLDC Hotspot 1.1.2. The Hotspot Java virtual machine has a dynamic compiler, an interpreter. With a profiler, the CLDC Hotspot implementation finds the most frequently used code pieces and compiles those byte codes into machine instructions; the remaining parts of byte code are executed by the interpreter at runtime. The compiled code is about an order of magnitude faster than byte code and occupies 4~8 times the space. Normally, the boundary of hot code pieces is identified by function.

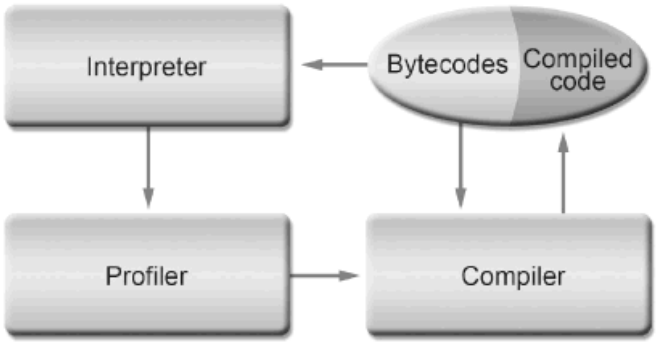

Figure 3: CLDC HotSpot Implementation Architecture (graphic courtesy of Sun)

Performance improvement tips

Technical tips:

1. Reuse code and objects if possible, avoid similar code in different places.
2. For unused objects, set their values to null. The memory management system will collect the garbage memory in a proper way and time. Calling System.gc() is likely add overhead without any benefit. For example, press the "Run GC" button in the SUN WTK's memory monitor will impair the application performance.
3. Use arrays instead of vectors where possible.
4. Abstract most frequently used code pieces and put them into independent functions, this will make it easy to be compiled and optimized by the CLDC Hotspot implementation.
5. Using large classes instead of small classes. Class header will increase the memory consumption and management complexity.
6. Using final/static methods.
7. Avoid using too many exceptions, because exceptions are much slower than method invocation.
8. Obfuscator like proguard can be used to compress the jar file size and thus reduce the midlet loading time.

Performance improvement example

The example used in performance improvement is a multi thread demo application in Motorola Java ME SDK v6.1 for Linux OS Products, <SDK path>\demo\com\mot\j2me\midlets\Bounce. The demo bounces rectangles on the screen, and the purpose is to demonstrate the application framework and animation using multiple threads.

FRAME_DELAY is the thread sleeping time in every cycle of the program, first set as

```
FRAME_DELAY = 100
```

in the source code, thus the application will make the emulator and real device keep running in the full loading state. This will make the performance improvement obvious.

Then test the time interval for 1000 cycles in one thread. This time interval is used as the time interval before optimization.

```
long initTime = System.currentTimeMillis();

System.out.println("Time Interval:"+ (System.currentTimeMillis()-
initTime));
```

For application optimization, the point is to focus on the methods and objects, which consume most system resources. The `serviceRepaints()` function forces any repaint requests in the queue to be serviced immediately. This method blocks until the pending requests have been serviced. `ServiceRepaints()` will make the animation look very fluid, but it has very big impact on performance. Because too many `serviceRepaint()` methods in synchronized part blocks every thread. In the modified code, `serviceRepaints()` is running in only one thread. (Note: This works well in real device, but some frames were missing when the application running with emulator in some slower PC.)

Original code,

```
myCanvas.serviceRepaints();
```

Modified code, `serviceRepaints()` is being called in only one thread, the rectangle size is used like the thread id.

```
if(this.size==10){
    myCanvas.serviceRepaints();
}
```

Table 1. Example performance improvement step 1

	Time Interval in Emulator (ms)	%Improvement in Emulator	Time Interval in real device(ms)	%Improvement in real device
Before optimization	12578		6848	
Modify serviceRepaints	11438	9.06%	6486	5.28%

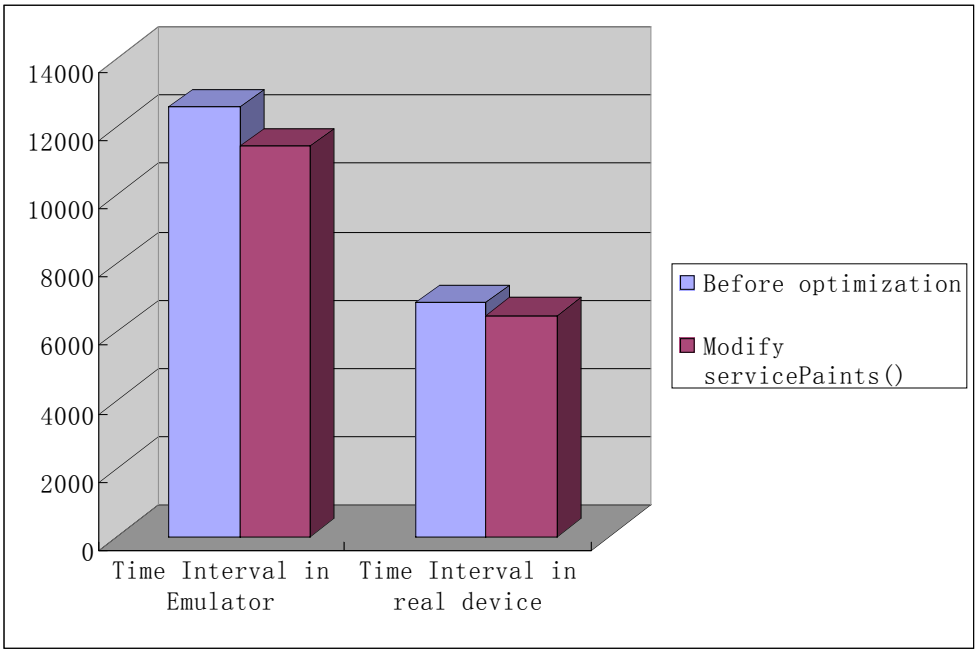

Figure 4: performance improvement step 1

It seems that the `fillRect` function is being called too many times in BounceCanvas.

In the original code, every thread calls `paint(g)` of all the four threads. `Paint(g)` is being called 4x4x1000=16000 times in 1000 cycles;

```
for (int i = 0; i < rectThreads.length; i++) {
     rectThreads[i].paint(g);
}
```

In the modified code, every thread only calls `paint(g)` for its own thread. `Paint(g)` is being called 4x1000=4000 times in 1000 cycles;

```
switch(activeSize){
    case 8: rectThreads[0].paint(g);break;
    case 4: rectThreads[1].paint(g);break;
    case 10: rectThreads[2].paint(g);break;
    case 2: rectThreads[3].paint(g);break;
}
```

Table 2. Example performance improvement step 2

	Time Interval in Emulator (ms)	%Improvement in Emulator	Time Interval in real device(ms)	%Improvement in real device
Before optimization	11438		6486	
Modify BounceCanvas.paint(g)	7906	30.8%	5769	11.05%

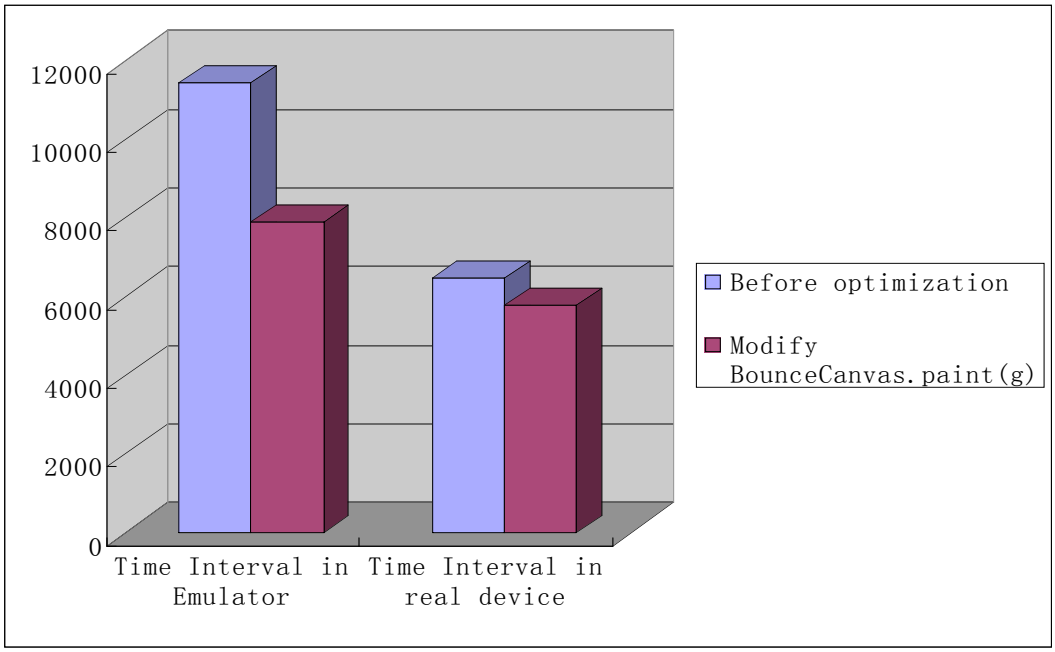

Figure 5: performance improvement step 2

Because the emulator cannot simulate the real hardware environment, the result in emulator's profiler can only be used as a reference to find the performance improvement direction. The last performance improvement result still depends on testing on the real device.

Appendix A: Additional Resources

- Motorola J2ME SDK Users Guide
- Sun CLDC Hotspot 1.1.2 Implementation Virtual Machine white paper