International Journal of Computer Applications (0975 - 8887)
Volume 60 - No. 12, December 2012

Performance Analysis of Mobile Memory with Optimization

Rohit Maurya
Gameloft India Pvt. Ltd
Hydrabad

Ajay Shankar Shukla
ABV-IIITM
Gwalior

Manish Kumar Sharma
GNIT
Greater Noida

ABSTRACT

As it known that mobile phones has low end configuration so we need memory optimization and network bandwidth reduction. These are very important factor for mobile developer because storage memory is very limited ranging from 64K to 200K bytes and the heap memory is ranging from 200K to500K bytes. The maximum size of application is fixed so we cannot run the application which exceeds that maximum size. Computing power is also limited so only few instructions can be processed per unit time. The problem is, unlike the PC platform where hardware can be easily upgraded whenever required; in mobile it is almost impossible to update the hardware requirement dynamically. Games developed for a particular mobile platform may or may not run on the other mobile platform due to difference of hardware capability. So in mobile platform only chance is to play with the source code, it is developer who has to write the code in such a way that will consume less resource and run faster.In this paper we present various techniques to optimize mobile memory for mobile games.

Keywords:

Optimization, Games, Mobile, Memoryifx

1. INTRODUCTION

The evolution of mobile phones is very rapid and the use of mobile phones increases ranging from communication to entertainment especially in gaming. The mobile has now become more than the medium of communication. It is now equipped with more multimedia and gaming capability. The hardware capability is also increasing but less rapid than the usage of broad range of multimedia application like 3D gaming, social networking etc. So mobile phones are needed to perform more and faster computation than its predecessors. In PC environment hardware can be upgraded easily as the requirement changes but it is not practical for the mobile platform so instead of changing the hardware requirement we have to change or optimize our algorithms to meet the requirement. The paper is organized as follows, mobile platform constraints Section II, followed various optimization techniques in Section III , Section IV shows simulated results and finally Section V presents the conclusion.

2. MOBILE PLATFORM CONSTRAINTS

When we are developing or porting games from PC platform to mobile platform, special care has to be taken because of limitation of mobile platform because in case of porting the game from PC environment to mobile environment the hardware capability will not be the same as the PC platform[4].
It will be less as compared to PC platform so chances are very few to run the same game on mobile platform without optimization. So first we discuss in this section what the constraints of mobile platform are and then we will discuss how to optimize the game to fit in those constraints in the next section. The limitation of mobile platform can be given as follows.

2.1 Memory Limitation

Memory limitation is a major issue with the mobile game developer. This is no longer a critical issue for developer due to falling price of flash memory. But it still needs to handle carefully to avoid memory leaks, crashes and fragmentation. There are three types of memory in typical java games:-
Storage Memory: Storage memory is the memory where jar application is stored. Every mobile platform has a fixed maximum jar size so if our jar application has a size greater than this maximum size, we need to minimize it. By doing this we also reduce the storage size as well as the download time and cost charged by the network [2].
Stack and Heap:Stack stores primitive data types such as int, long and Boolean. Heap stores the runtime objects. The heap size is lesser than the stack so we should avoid creating runtime objects unless it is necessary.
Persistent Storage: Persistent Storage is a non-volatile storage for data. J2ME applications use the Record Management System to store the persistent data.

2.2 Control Processing Units

Most of the phone will have very slow clock speed. So complex calculation like 3d graphics calculations are limited in the speed with which they can be performed. Size of cache is also limited which will put extra burden on the processor due to excessive data demand.

2.3 Low Bus Bandwidth

In mobile phones the data flow between CPU, memory and I/O devices is limited due to low bus bandwidth. So developers have to be careful while programming and try not to flood the data flow because it will affect the throughput and efficiency.

3. TECHNIQUES FOR OPTIMIZATION

Optimization is required to make the application run within the constraints. In case we are short of memory and our application is larger than the maximum size we have to reduce it to lower than the maximum allowed size. Most useful and effective approach is to compress or reduce the graphics and jar size. If we are short of heap memory or device cannot allocate the sufficient memory to run the application, we have to optimize our source code as well as graphics also because a bad source code can cause the heap problem. Example of heap crisis is loading of unnecessary data at one time which is larger than the heap memory. So we discuss two main optimization techniques: first is memory optimization and another is source code optimization in the following optimization techniques.

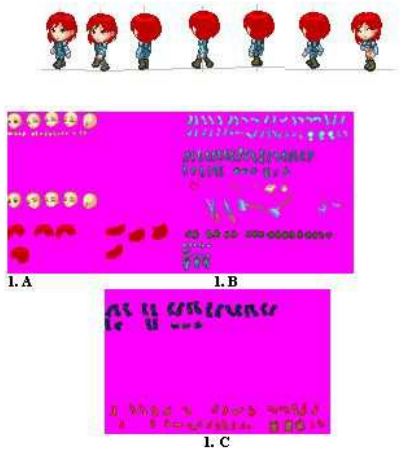
3.1 Memory Optimization

Mobile platform faces two types of memory optimization issues first jar size optimization and another is heap memory issue.

3.1.1 Jar Size Optimization. A mobile application consists of class files and resources so jar size greatly depends on the size of these resources and class files. Jar uses zip compression to reduce the size of jar. Sometimes it is not sufficient we need to further reduce the size of jar. The method of jar optimization is discussed in the following sections.

3.1.1.1 Graphics Optimization

1 **Combine small images into big image:**
In mobile games graphics consume lots of spaces. So graphics size can be reduced by combining the lots of small images into single image. By doing this we can eliminate the header from the small images and size will be reduced.
If there are various avatar of the actor or hero in the game, we do not take separate images for the different avatar we only save the difference between two avatars because it will reduce the graphics size. Consider the following 7 avatars first two avatar is almost same except the change in leg movements and avatar 3 to 6 has difference in only hand and leg movements. So by saving the avatar 1, 2, 3, 4, 5, 6 we are saving their common parts too which is obviously increase the size of graphics. But if we store only the changes in the avatar as shown in 1.A, 1.B, 1.C we can eliminate the duplication and able to reduce the size graphics.
As shown in the figure 1 first we combine the different image in one single image and draw the avatars using that single image. For example if we want to draw avatar, we select the face from the image and then the hair and other parts according the type of avatar. By doing this we only need to store those parts which are changing. We can see from the above figure if we store different avatar as a single image how much space can be saved.
As we see in figure 2 we can draw the avatar by taking the different required part from the single image which is denoted in rectangle.
This concept also applied in the different levels because the different levels differ by only a slight graphics change and we dont need to store the common parts again and again.
By reducing the graphics we can also optimize the heap memory.



(The above images is not three separate images its same as shown in 1.D, its shown in three separate part only for better visibility)

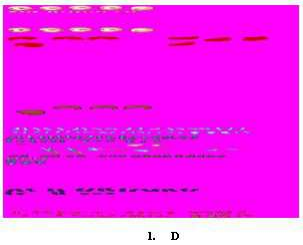


Fig. 1.

(Image 1.D is a big single image consists of several small images)

Fig. 2.

2 **Using Optimization Tools**
We can compress the graphics using graphics optimization tools like punyPNG, WebP and etc which will reduce the graphics size to a great extent.

3 **Combine Image Files into a Data File**
We can combine the all the images of our games in to single data file to reduce the jar size because compressing the bigger file will give the better compression than compressing small files separately. In this way we can also hide the images used from the decompression of the jar.

3.1.2 Heap Memory Optimization. In J2me application memory management like garbage collection and allocation is handled by JVM itself. Whenever we want to free the memory we can do it by using Runtime.getRuntime().freeMemory() and System.gc() [3].But it is not necessary that every time we use System.gc() it will free the memory because its jvm who decides when to actually free the memory. The implementation ofJ2ME VM is differ in the different handset models, due to differences in low level I/O and memory management of different hardware. The same application that runs in two different handsets and emulator may have a big difference in memory consumption. Memory measurement in emulator is easy, but memory measure is very difficult in real handsets [3].
One solution to this problem is given as following code.

```
static int[] allocMemory = null;
staticintminHeap=1024 * 200;

static void GC()
{

if (Runtime.getRuntime().freeMemory() <minHeap)
```

```
if(allocMemory == null)

allocMemory = new int[250000];
if(allocMemory != null)

allocMemory = null;

System.gc();
}
```

Instead of using System.gc() we will call the above GC() function. In the above code minHeap is the size of heap which can vary from device to device. What the above code is doing that whenever the available memory is less than the heap memory it will allot the large value to the integer array allocate Memory and then set it to null and then call the actual System.gc(). After calling this jvm thinks that there is some huge memory is being allocated and deallocated so it will free the garbage memory
Another approach of heap memory optimization is to handle the game data carefully. Suppose there is five levels of game so a nice way of heap optimization is not to load all the data at first level because at start we dont need the graphics and other data of another level. After a level ends and data belonging to particular level is not required anymore, must be freed.

3.2 Source code optimization

Our games performance depends on how we code so by optimizing our coding style we improve the performance of mobile game

3.2.1 Initialize objects at declaration time. Performing the declaration and initialization on separate lines it will take two operation one invoking the objects default constructor and another is assignment operator.

```
UserClassobj = data; ( one operation )

UserClassobj;
( Two Operation )
Obj = data;
```

First code is taking one operation where second code is taking two operations so obviously first one will be faster.

3.2.2 Avoid unnecessary declaration. For Example

```
for(int i = COUNT; i>=0; –i )
{
Int a = 2*COUNT //unnecessary here
/*
Some code here
*/
}
```

In the above code the declaration, initialization and calculation is unnecessary because every time loop executes these operation will be performed and processor cycle will be wasted.

3.2.3 For loop optimization. Use following code

```
for(i = n-1; i>= 0; –i ) { }
instead of
for(i = 0; i< n; ++i ) { }
```

Because post increment/decrement operators are more expensive than the pre increment/ decrement operators[1].

Testing against 0 is always faster than testing against any other value [5]..

3.2.4 Do not load the data until its required. Never load the data until its required because it will consume unnecessary heap memory. So only load the data when its required.

3.2.5 Do not create unnecessary runtime objects. As we know runtime objects are stored in heap. So never create unnecessary runtime object because it will consume heap memory. Its better to declare the object before we use it.

```
public String OutString = ”Game Update”;
Public void game_update()
{
//// some code

System.out.println(Game Update called);
g.drawString(OutString + called ,0,0,0);

// some code

}
```

In the above code the System.out.println is printing the temporary string object and g.drawstring is also uses the called as temporary object. Which is taking unnecessary heap space.

We should better use the following code instead of above code

```
public String OutString = ”Game Update called”;
Public void game_update()
{
//// some code

System.out.println(+OutString);
g.drawString(OutString ,0,0,0);
// some code

}
```

3.2.6 Switch versus if-else. In general switch is faster than the if-else. ”switch” uses two different byte code instructions named ”tableswitch” and ”lookup switch”. ”table switch” is used for contiguous cases, while the slower ”lookup switch” is used for more spread-out values. In case of table switch the compiler translates them into a jump table instead of a comparison chain. A jump table improves performance because it reduces the number of branches to a single procedure call and reduces the size of the control-flow code [1].

3.2.7 Declare functions as static. Always declare local and member functions that are not to be used outside the file they are defined in as static. Static functions are faster than non-static function. Unlike the non-static function there is only one copy of the static function in memory and compiler doesnt have to worry which copy to call.

3.2.8 Declare local variables as static. It is good to declare static local variables in the routines that are called frequently. This will remove the overhead of declaring and initializing them every time the function executes. It also helps to solve memory fragmentation problem.

3.2.9 Avoid synchronization when possible. Synchronization makes code run slower than the ordinary code. A lock must be acquired and released upon an object every time a synchronized context is entered and exit. Thread which cannot acquire the lock

must have to wait until lock is released. This makes slow execution of code.

3.2.10 Look-Up Table. Use of look-up table which is an array of pre computed values makes code run faster because we do not have to compute the required expression every time. So, if we have a function whose values can be pre-computed and stored in a table, we can trade memory for speed by using values stored in that table.

```
// pre computed array

double My_SIN[360];

for (i = 0; i< 360; i++)
{
My_SIN[i] = Math.sin(i);
}

/* we simply use My_SIN pre-computed array instead of
Math.sin(angle) for faster execution */
result = My_SIN[angle];
```

3.2.11 Use of array. Optimize multi-dimensional array by making them uni-dimensional.

```
Use int [] array; // 1X16
Instead of int [] [] array; // 4X4
```

3.2.12 For-Loop Unrolling. For loop is very convenient but if it is not used properly it will make the execution worse.

```
For(i=0;i<10;i++)

Syste.out.println(For Loop Optimization);

Is equivalent to

i=0;
Syste.out.println(For Loop Optimization);
If(i>=10)
..... 1st time Return;
i++;
.
.
.
.
.
.
Syste.out.println(For Loop Optimization);
If(i>=10)
return;
10th time i++;
```

The above code is doing unnecessary 10 increment and 11 comparison which waste CPU cycle. The above can be written as 10 times println statement.

```
Syste.out.println(For Loop Optimization); 1st
.
.
.
Syste.out.println(For Loop Optimization); 10th
Or
For (i=0;i<10;i+=5)
Syste.out.println(For Loop Optimization);
```

Which includes only 2 increment and 3 comparison operations.

3.2.13 Use of Mathematical Operator. Addition and subtraction are faster than the multiplication, division and modulus operators. Also use of bitwise operator can make operations faster.

```
intvar = 10*2; // slowest
intvar = 10+10; // slow
intvar = 10 <<1; // faster
intvar = 20; // do not compute anything which is already known as design time
```

4. RESULTS

The results are based on different game developed and played by Game tester and comparison results are shown in table 1.

Table 1. Comparison Results

Feature	Without Optimization	With Optimization
Cost(memory)	High	Medium, Low
Processing Time	High	Low
Game Startup Time	More	Less
Game graphics Quality	High	Good, More likely to be low
Crash Rate	High	Low
Frame Rate issue	High	Low
Storage Size	More	Less
Network transfer time	More	Less
Start Up at low end devices	More failure	Less failure

5. CONCLUSION

The optimization techniques do not guarantee that our application will be always optimized. It is depend on the programmer how he chooses those techniques to optimize the program. It may happen that optimizing one thing will affect the other like if we are compressing the graphics, we have to be sure that it does not affect the quality. Another example is case of heap optimization in which we are restricting our application to use lesser heaps to avoid out of memory exception but we have not implemented it properly. This may create a frame rate or lag issue throughout the games because we may have restricted some graphics and other resources to load into memory. So in this case the data is loaded before we use it which will create the lag and frame rate issue. So optimization should be done very carefully because it might make the performance worse.

6. REFERENCES

[1] F. Chehimi, P. Coulton, and R. Edwards. C++ optimizations for mobile applications. In *Consumer Electronics, 2006. ISCE '06. 2006 IEEE Tenth International Symposium on*, pages 1 –6, 0-0 2006.

[2] V.-M. Hartikainen, P.P. Liimatainen, and T. Mikkonen. On mobile java memory consumption. In *Parallel, Distributed, and Network-Based Processing, 2006. PDP 2006. 14th Euromicro International Conference on*, page 7 pp., feb. 2006.

[3] Wang Jianmin, Zheng Zibin, Peter Tam, and Liu Jianping. Optimization technique for commercial mobile mmorpg. In Xiaopeng Zhang, Shaochun Zhong, Zhigeng Pan, Kevin Wong, and Ruwei Yun, editors, *Entertainment for Education. Digital Techniques and Systems*, volume 6249 of *Lecture Notes in Computer Science*, pages 34–45. Springer Berlin Heidelberg, 2010.

[4] Bailin Yang and Zhiyong Zhang. Design and implementation of high performance mobile game on embedded device. In *Computer Application and System Modeling (ICCASM), 2010 International Conference on*, volume 8, pages V8–196 –V8–199, oct. 2010.

[5] Weishan Zhang, Dong Han, T. Kunz, and K.M. Hansen. Mobile game development: Object-orientation or not. In *Computer Software and Applications Conference, 2007. COMPSAC 2007. 31st Annual International*, volume 1, pages 601 –608, july 2007.