Emerging Technologies: Mobile Development for Android Devices

Performance

Introduction

- With more and more apps being developed on Android devices performance is starting to matter to users.
- This is in spite of the fact that the latest devices have considerably more processing capability.
- In this lecture we will explore some things that can improve the performance of Android applications.
- To make them run faster, smoother, respond quicker.

Performance Matters

- Performance matters for two main reasons:
- Higher performance means a smoother more responsive application that quickly deals with user input.
- Apps that exhibit higher performance on the same tasks will complete in a shorter time leading to less use of the battery.
- Thus it is in your interest to make your applications perform well.

Rules of Performance

- There are two basic rules to improving performance.
- Don’t do work that is unnecessary.
- Avoid memory allocation/deallocation wherever possible.
- By observing these rules you will create an application that exhibits high performance and will place little drain on the battery.

Measuring Performance

- Most of the performance numbers that you will see here have come from experimentation on Android code.
- Also through code profiler usage. A code profiler is a tool for monitoring the execution time of code.
- A profiler will produce a range of statistics that will show where a program spends the majority of its time.
- Indicates where the code can be improved to gain better performance.

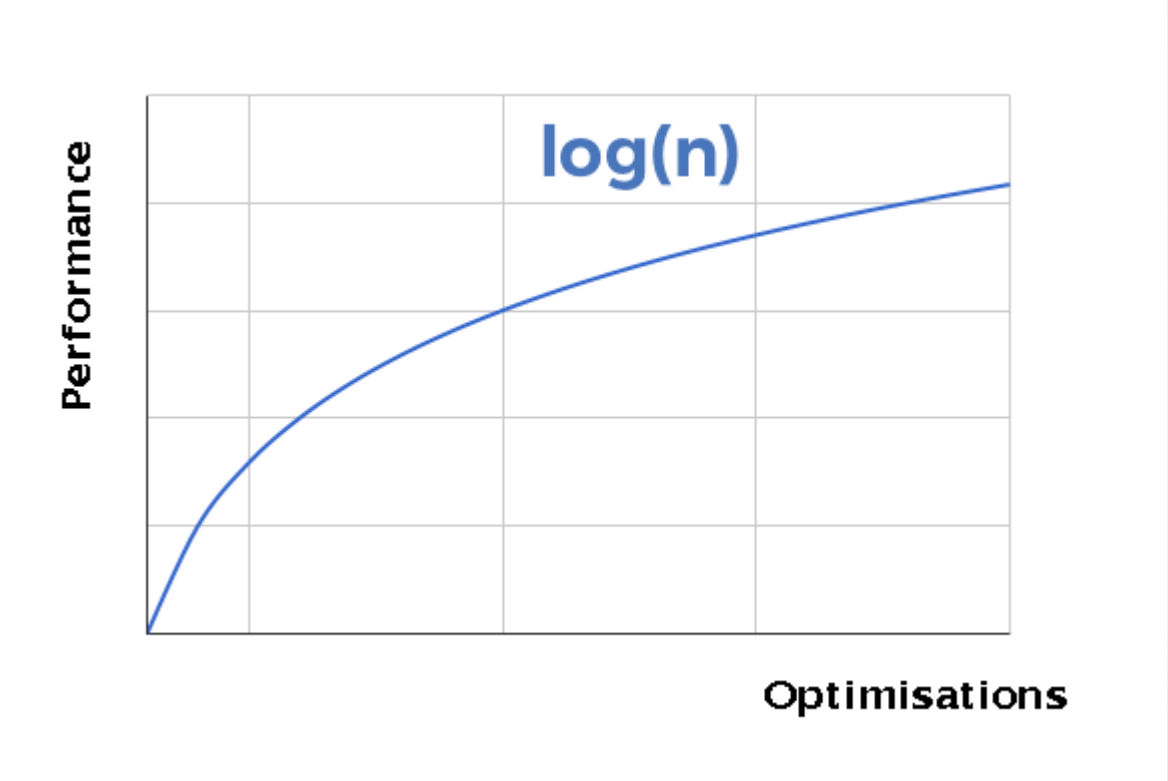
Code Profiler

- A code profiler will instrument code with timing information in order to determine how much time an application spends in each part of the code.
- First the application will be run to determine how long each part of the code runs for.
- This will be arranged in descending order from longest to shortest.
- One of the longest parts of code will be chosen and an attempt will be made to optimise it. The profiler will then be run again to evaluate the changes in code.
- If the change has reduced the execution time of the program it will be kept and another optimisation will be looked at.
- If the change has increased the execution time of the program it will be reverted and another optimisation will be looked at.
- The process is then repeated until an acceptable outcome is achieved.

Measuring Performance

- When performing optimisation, you must remember that you will run into the law of diminishing returns.
- Each optimisation will result in a smaller performance gain.
- The goal therefore is to only do optimisation to the point where performance is perceived as being good enough.
- This is because bottlenecks in code simply tend to get moved from one place to another rather than being removed entirely.

Diminishing Returns



Optimisation Goal

- When performing optimisation, you must remember that you will run into the law of diminishing returns.
- Each optimisation will result in a smaller performance gain.
- The goal therefore is to only do optimisation to the point where performance is perceived as being good enough.
- This is because bottlenecks in code simply tend to get moved from one place to another rather than being removed entirely.

Mobile Performance is more Complex

- You also have an additional issue in that your code has to support multiple CPU architectures.
- Thus CPU specific optimisations that improve performance for one CPU architecture may result in significantly deteriorated performance on others.
- The majority of desktop computers have similar x86 based architectures.
- To complicate matters further there is a JIT (Just In Time) compiler that is also trying to optimise you code at run time.

Try to Avoid Memory Allocation

- Memory allocation/deallocation are expensive operations in any OS.
- Requires a switch from user mode to kernel mode execution to allocate the memory and another switch back to continue running the application.
- Also, the more you allocate memory the more the garbage collector will get involved and will reduce your application performance.
- There are a couple of examples involving string buffers and parallel single dimension arrays that can help reduce allocation.

String Buffers

- If you have a method that returns a string that is to be used immediately with a StringBuffer or StringBuilder it would be better just to append the string to the StringBuilder directly if you can do so.
- Prevents the need to allocate and deallocate a temporary string on each method call.
- Another option when working with small portions of strings is to use a substring instead
of making a copy.
- Instead of copying the data across the substring will use the same shared character
data, thus saving the need to copy values.

String Buffers

- If you have a method that returns a string that is to be used immediately with a StringBuffer or StringBuilder it would be better just to append the string to the StringBuilder directly if you can do so.
- Prevents the need to allocate and deallocate a temporary string on each method call.
- Another option when working with small portions of strings is to use a substring instead of making a copy.
- Instead of copying the data across the substring will use the same shared character data, thus saving the need to copy values.

Static Methods

- Use static methods where possible as they are much quicker to call compared to virtual methods.
- Particularly if you don’t need to access an object’s fields.
- Invocations on static methods are about 15-20% quicker than object methods.
- Difficult to find such opportunities but take them whenever possible.

Static Final

- Using the static final declaration with constants will improve the lookup and reference times of these constants.
- The constants will be computed once and stored in the dex homogeneous pools.
- Where they can be referenced and looked up relatively inexpensively.
- Precomputed data like this removes computations from your program before it even runs.

Further Ideas

- Use integers rather than floating point numbers where possible.
- Optimise layout hierarchies. Remove nested levels Flatten.
- When drawing custom views keep all memory allocation in the shared init() method instead of the onDraw() method.
- As stated before onDraw() may be called many times a second and the allocation routines will slow things down.
- Finally, avoid feature creep and software bloat.
- Remember your application should do a single job and should do it well. Resist the urge to add more and more features as this will slow down your application performance.