Need for Speed:
Optimizing your App

Steven R. Bagley

Why Speed…

- Mobile devices are like PCs, but they aren't PCs
- PCs tend to be powerful…
- As we saw in lecture two though, Mobile Devices aren't…
- So the Mobile experience should be slower…

PCS (Fast CPU, lots of corse, lots of RAM, lots of disk, fast network)
Movbile (slow-ish CPU <~1GHz, limited ram <~512MB limited storage, 3G/Wifi)

"The iPad is a far slower machine than a modern MacBook in terms of raw hardware performance, but it feels faster in many ways, because you never have to wait for it"

John Gruber,
Daring Fireball

http://daringfireball.net/2010/04/the_ipad

Why Speed?

- The mobile experience isn't slow…
- And people don't want it to be slow
- Mobiles are often used on the move
- If your app is slow, then they won't use it
- Optimize our apps so they are responsive

E.g. to look up the times for a bus train... To send an email/text/tweet when running late

Power

- Another reason to optimize our apps is to maximize battery life
- People don't want the phone's juice running out when they need it
- The more work the CPU does the more battery life it uses…

CPU and Battery life

- CPU uses most power when it is doing things
- Uses the least power when it is doing nothing
- Therefore, we want our programs to let the CPU idle as much as possible
- How do we do that?
- By making our programs run fast…

CPU and Battery life

- If code runs as quickly as possible
- Then it will be finished in less time…
- Which means the CPU can idle for longer

- Most things that optimize for speed will also optimize battery life
- Additional things we can do for power too…

Game plan

- Today
  - Optimize for speed
  - Optimize for Power
- Tomorrow
  - Google lecture on optimization
- Next week, optimizing network access…

Measure, don't Guess

- A word of warning
- Don't try and guess what needs optimizing
- Our assumptions are often built on understandings of different hardware
- Measure it…

Measuring time

- Tools with the SDK (code profilers)
- But can also just use `Log.i()`
- Print time before and after part of the code

- Use your app – does it feel responsive
- Measure on the device, not the simulator…
- Should be able to show that your optimization made a difference

Simulator isn't as accurate

Responsiveness

- Apps should feel fast
- It doesn't have to actually be fast
- What this means is that it should be responsive to the user
- If it takes longer than 100-200ms to do something, the user will decide its laggy
- And buy something else…

Startup

- One place the app should be fast is at startup
- Users won't use an app that takes forever to startup
- And it doesn't look good when they demo it to a friend (potential customer)

Slow startup

- Why is an app slow to startup?
- Usually because its initializing itself
- But how much of this has to be done before the app starts?
- What can be done in the background after the app is running?
- Move anything that can be into the background

Twitter Example

- A Twitter app could fetch all new tweets at startup
- But this would slow startup down considerably
- Better, startup quickly with cached tweets
- Then download new tweets in background and update display

YOu are going to be doing it in the background later anyway

In App responsiveness

- Same applies for in app stuff
- Push any processing into the background, rather than doing it in the main thread
- Remember if it takes more than 100ms to do something, your app will feel laggy

Optimize for speed

- There are some optimizations that are specific to mobile devices/OSs
- But these are small-scale optimizations — although they can have huge differences
- Far more important is that you chose the right algorithms and the right data structures

Small optimizations in tight loops can have a huge effect

Procrastinate

- Delay doing things until as late as possible
- It may well turn out that you don't need to do it at all!
- For example, don't redraw graphics that don't need redrawing
  - i.e. don't have a 3D chess game that is redrawing at 60fps

When does a game need to save its state?

Memory

- Mobiles have a limited amount of RAM
- Typically <~512MB
- A lot of that is used for other things
- iPhone3G used 104MB of 128MB for system
- Be frugal in memory usage
- No virtual memory

Don’t allocate what you don’t need
No VM (in conventional sense) -- somethings can still be unloaded

Memory

- Avoid allocating objects if at all possible
- Takes time to allocate memory, so don't waste it
- Especially in the UI thread — causes the garbage collector to fire more often
- Will be visible as hiccups in the UI

Memory

- Remember a `String` is an object, so try and avoid allocating them unnecessarily
- E.g. If a method returns a `String` that is then added to a `StringBuffer`
- Change the method so it takes the `StringBuffer` as a parameter and adds it directly

Memory and Strings

- Return substrings rather than creating new `String`s
- Substrings share memory with the original `String`
- Danger, can also cause the long `String` to stay around unnecessarily
- Think about the usage pattern…

Arrays

- Don't box things — an array of `int`s is far better than an array of `Integer` objects
- No objects created in the first
- Can be more radical than this
- Slice up arrays of objects into parallel arrays of basic types

Kinda goes against OO principles, but necessary at times

Points

- Take this class:

```
class Point
{
int x;
int y;
}
```

- And this array

```
Point ps[] = new Point[1024];
```

- This creates 1024 objects…

Sliced points

- Better solution would be to store the data as two arrays of ints:

```
int x[1024], y[1024];
```

- Faster and uses less memory
- But care must be taken to ensure you don't get the arrays out of sync…
- In general, avoid creating short lived objects

Prefer Static over Virtual

- Two types of methods in a class
- static methods — belong to the class
  `public static void foo() { … }`
- virtual methods — act on an object
  `public void bar() { … }`
- If a method doesn't need to be virtual, make it `static`...

Prefer Static over Virtual

- When can it be `static`?
- When it doesn't access any member variables
- If it does, can it be converted to not by passing those variables in as parameters?
- Will make the method invocation 15%—20% faster…

Also good practice…

Constants

- If you define constants, make sure you declare them as `static final`
- If you just make them `static`, you cause a class initializer method to be run (more unnecessary code)
- If you make them `final`, the variable lookup vanishes (replaced by the constant)

WIN-WIN situation

Getter/Setter methods

- Good OO practice to provide these at the class interface
- But don't use them internally — access the variables directly
- They are virtual method calls and much more expensive than accessing the data
- Looking at 3x (without JIT) to 7x faster…

For loops

- Used the enhanced for-each loops wherever possible:

```
for(Foo a : mArray)
```

- Over:

```
for(int i =0; i < mArray.length; i++)
{
   Foo a = mArray[i];
   …
}
```

For loops

- The exception is for `ArrayList`, where a handwritten loop is much faster.
- But pull the call to get the array size into a local variable
- In general, use for-each — except for `ArrayList`, when a hand-written counted loop is faster…

Floating-point

- FPU is about 2x slower on an Android device
- True on both devices with or without and FPU or JIT
- No difference between `float` and `double` (Except space)
- Some CPUs don't have a hardware divide for integers

So beware of using division and modulus operators...