ES3 Lecture 1

Introduction to mobile and embedded development

Course overview

- Introduction to mobile development
- Development platform overview
- iPhone development (4 lectures)
- Python/Qt/Maemo development (3 lectures)
- Introduction to mobile HCI
- Mobile networking
- OpenGLES -- 2D and 3D accelerated graphics
- Sensors

Course Topic

- Mobile and embedded systems
- Focus on mobile development
  - You should be a competent iPhone and Maemo developer by the end of the course
  - Understand the challenges of developing for mobile platforms
  - Have a clear understanding of the core technologies involved
    - mobile networking, mobile graphics, mobile interface design, mobile sensors
  - Be able to go from a blank screen to the App store!
- It's assumed you can program *reasonably* well in C and Python and have a decent knowledge of object-oriented programming

Course Overview

- 1 lecture (Mondays @ 12)
- 2 hour lab (Tuesdays @ 2)
- 1 lecture (sometimes) (Wednesday @ 12)

- Email: **jhw@dcs.gla.ac.uk**

- **Assessment will be coursework alone! There is NO exam**

- 3 pieces of minor coursework **(10% each)**
- 1 individual project **(70%)**

Schedule

Week	Monday	Tuesday	Wednesday
1	Intro to mobile dev.	---	Mobile platforms
2	iPhone I	iPhone lab	iPhone II
3	iPhone III	iPhone Lab	---
4	iPhone IV	iPhone Lab	---
5	Maemo I	Maemo Lab	Maemo II
6	Maemo III	Maemo Lab	---
7	Mobile HCI	Maemo Lab	---
8	OpenGLES	OpenGLES lab	---
9	Sensors	iPhone sensors lab	---
10	Networking	Project Lab	---

Individual Project

- **Your** project
  - Must be on Maemo or iPhone platform
  - Should be challenging -- choose something interesting!
- Proposal by week 5
  - Start work by week 6 (at the very latest)
- Hand in at end of term
- Assessed on code + report
- Focus on quality and ambitiousness of product

Minor Coursework

- 3 minor pieces
  - Week **4** basic iPhone application
  - Week **7** basic Maemo application
  - Week **9** OpenGLES + sensors
- Should take around 2 hours each
  - Not massive development
  - Skeleton code will be provided

What's special about mobile devices?

- They move around
  - The context of use is highly variable
- Connectivity is usually very important
  - Communication and networking are key
- Usually very constrained in many ways
  - Limited hardware -- processing power, storage, battery life
  - Limited software -- OS, language, debugging tools, libraries
  - Development is hard -- often slow, clumsy and frustrating

What's different

- Coding, interface design, evaluation are all very different from desktops
- Need to work with tricky constraints
- But be able to work in a wide range of difficult environments!
- Testing and evaluating can be very difficult
  - Networking complicates testing enormously
  - Development tools are sometimes primitive
  - Getting coverage of usage environments is hard
  - Capturing and analysing data can be challenging

Context

- Devices aren't used in a fixed context like a desktop
  - An idea which seems good in the lab might be awful on the bus
- Can't rely on user attention
  - User attention will vary during use
  - e.g. while walking or while talking to friends
- Constant risk of interrupts from phone calls, SMS, etc.
  - Applications must be able to deal with this
  - State must persist in a reliable way

Platforms and cost

- Lots of mobile platforms
  - often just one API
  - write once, debug everywhere
  - you probably won't ever *see* most of the platforms your software is likely to run on, never mind test them
- Platforms are constantly changing
  - New features will be added or removed
  - There is a constant struggle to be up-to-date

Impact of poor coding

- Mistakes can have really serious consequences
  - Financial cost impact -- mobile data can be incredibly expensive
  - Imagine a bug that sent SMS sending into an infinite loop
    - Could cost end-user thousands of pounds
    - **And the developer would be liable!**
- Safety issues
  - Devices are used in unknown contexts, possibly dangerous
    - Driving, walking, etc
  - Attention must be managed
  - Think of impact of draining battery life (no ability to make emergency calls...)

Hardware constraints

- Limited CPU power
  - Maybe an ARM chip (620MHz ARM in iPhone, 600MHz ARM in N900, 434MHz on N97)
    - powerful, but not exactly on par with a desktop
  - Or maybe a much more limited device
    - Like an 8-bit 40MHz PIC or Atmel chip on an embedded device
    - Just a few integer instructions on 8 bit values
  - Or something unusual
    - Like a Parallax Propeller
    - 20MHz but on 8 parallel 32 bit cores

Processing Constraints

- In general, CPU power will be limited
- Very often no floating point unit **(not true on the iPhone!)**
- This means operations on floating point numbers will be VERY slow

- GPU acceleration may or may not be available
  - But it will be slower than desktop systems
  - Recently, OpenGLES becoming common. We'll cover it later...

- Sometimes calls for lower level languages (C, even assembly)
  - But most work can still be done in high-level code with occasional custom modules

- Writing in fixed point is sometimes necessary (and unpleasant and bug prone)

Interesting resources

- Doing mathematical operations on limited devices?
  - **Math Toolkit for Real-Time Programming / Jack Crenshaw**
    - **This is a pretty good guide to clever ways of computing numerical functions like log, sin, polynomials quickly and accurately**
  - **FXTBook / Jorg Arndt**
    - **Hardcore but powerful numerical algorithms (e.g. CORDIC for fast sin/cos/tan, FFT algorithms...)**
  - **Hacker's Delight / Henry S. Warren**
    - **Nothing to do with hacking, lots of algorithms for low-level fast integer operations**

Storage Constraints

- Limited RAM the norm
  - Maybe 128Mb on a mobile phone
  - Maybe 128 **bytes** on an embedded system
  - being lazy with storage isn't an option...

- Backing storage can be slow
  - Flash memory is ubiquitous
  - But it often has huge write time (20--100ms) and it has to write whole blocks at a time
  - Wears out after 10,000 -- 100,000 writes
    - Wear levelling -- which tries to spread out writes -- can help improve device lifespan
    - But means that Flash memory slows down over time!

Human Computer Interaction

- Devices are small: this means their screens are too
  - Complex GUI elements like overlapping windows aren't reasonable
  - Keypads are small too, and might only have a small number of keys (e.g. numeric)
- Interaction often based around a few buttons, touchscreen interactions or using other sensors
  - Same application may need to support many input options in a consistent way
- Optimizing interfaces becomes important
  - massive menu hierarchies become unreasonable, for example
- Lots of interesting possibilities
  - capacitive sensors, accelerometers, pressure sensors, vibrotactile output....

Power

- **Power consumption is critical**
- Everything revolves around battery life
- This impacts computational power -- slower CPUs use less power
- Many devices spend almost all their time "sleeping" in a very low power mode.
  - Only waking up for maybe 10ms in a 2 second cycle (e.g. to handle GSM connections)
    - This has a big impact on how applications can do computations!
- Networking is a massive power draw, especially transmitting
  - Using efficient protocols and minimizing unnecessary communication is essential

Power (II)

- Screen illumination is very expensive too
  - So using visual feedback for everything is not necessarily the best choice
- Battery life is hard to estimate without testing on a device
  - This adds to the complexity of building software
- Building systems based around operating only with very short timeslices can be difficult
  - Getting it wrong doesn't result in any errors -- except the battery will die when you test it.

Networking

- Networking is essential to many mobile services
- Network access varies continuously
  - Must be robust to gaining/losing network connections
  - And work as well as possible when networking goes out
- Must be parsimonious in use of data transfer
  - Big power draw, and can be very expensive (in monetary cost to end user) for some transport options (like 3G or GPRS)
- Can be complicated and messy to use, especially Bluetooth
  - Bluetooth is guaranteed to cause problems
  - Security, pairing, synchronizing devices
  - Usually will fail mysteriously

Limited Operating Systems

- Generally, libraries and operating systems cut down
  - Limited multi-threading support
  - Limited GUI components
  - No command line or console (this is sometimes painful when debugging)

- Your favourite library won't be there
  - For computational geometry, numerical methods, database access, PDF writing, whatever
  - Either roll your own, try and port an existing piece of software, or just don't implement that functionality

Emulators and debugging tools

- Debugging and testing is the worst thing about developing for embedded and mobile systems

- Testing in emulators is great
  - Except they never work like real devices
  - Some are pretty good (iPhone) and some are awful (Symbian)
    - If it works on the Symbian emulator, it's just about guaranteed not to work on the phone
  - The features that make devices exciting, like GPS, sensors, multitouch can't be tested because they can't be realistically simulated
  - Battery life also can't be tested

Debugging

- Remote debuggers allow testing of software on real device
  - Communicate remotely with desktop, allowing normal debugger functions
- Usually incur a big speed penalty
- Occasionally behave in strange ways
  - Bugs may only occur when the debugger is **not** running!
- But if available, make testing much easier

Documentation

- Documentation for mobile devices is usually bad and incomplete
- Devices change **very** fast
  - Phones completely change in a year
  - And no-one likes writing documentation
- Apple manage quite well by having a very similar platform for OS X and the iPhone
  - Not many versions of the iPhone
- Android, Windows Mobile, Symbian suffer from device variability
  - One API, but endless implementations
  - Most of which are incorrect or incomplete with respect to the API
- Internet forums are often the best or only source of reliable information
  - Unless you have some other access to gurus...

Licensing and security issues

- Many platforms require signing for devices
  - This aids security -- packages can be traced to individual developers and capabilities can be restricted
  - Makes writing viruses and other malware much harder
  - But it can be a real pain for developers
- Signing can be difficult, can be expensive and may be arbitrarily restricted by vendors (e.g. Apple!) or network operators
- Usually developers can get limited rights to sign applications for their own device
  - Even still, this is time-consuming and can be fraught with problems like expiring certificates
  - Capabilities of the device (full file system access, DRM operations) may be restricted to certain "trusted" vendors (like network operators)

Memory and power management

- Memory is very constrained
  - Management of resources is essential
  - This can be time-consuming (and sometimes leads to unpleasant API's which enforce resource management, such as Symbian's model)
- Applications must be designed with power consumption in mind
  - Can be restrictive
  - Busy waits or polling are out!
- Tools available to profile power use
  - Unexpected results sometimes
    - pressing a key results in a big power spike on Nokia N95s
    - some phones will draw massive power futilely trying to get GPS fix indoors for example

Summary

- Hardware is limited: processing power and storage become much more relevant
- **Always think about power consumption**
  - You cannot rely on having 100% CPU time
- Interface design needs to be done differently with mobile devices
- Networking is essential, but robustness is critical
- Platforms, operating systems and libraries are often lacking
- Debugging can be hard
  - Emulators help, but they are far from a panacea