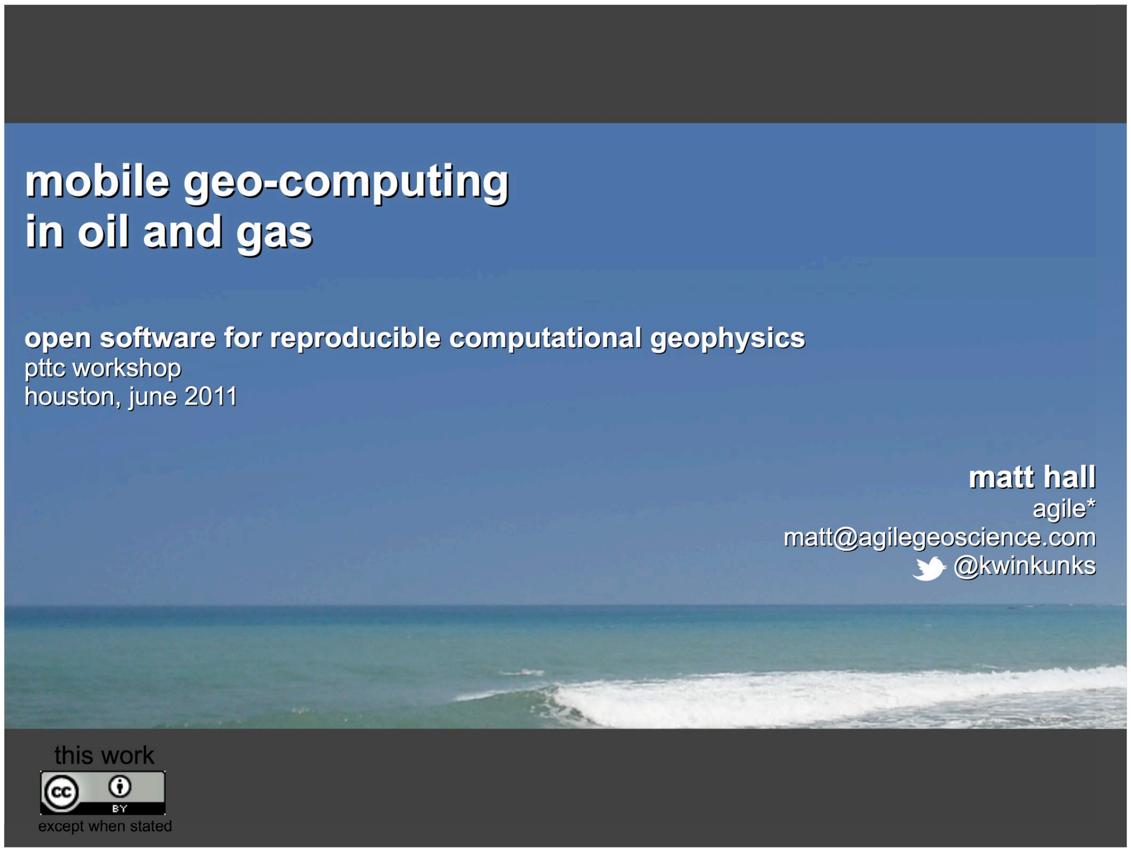


mobile geo-computing in oil and gas

open software for reproducible computational geophysics
pttc workshop
houston, june 2011

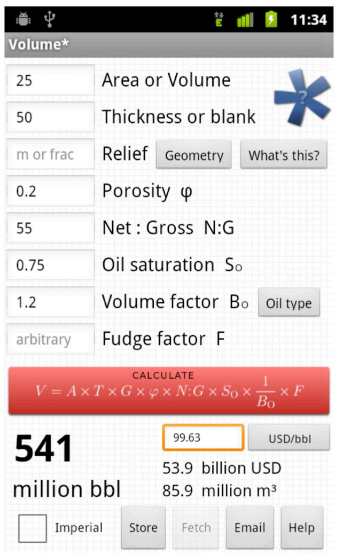
matt hall
agile*
matt@agilegeoscience.com
@kwinkunks

This is a version of a talk I gave at the Petroleum Technology Transfer Council's workshop **Open software for reproducible computational geophysics**, being held at the Bureau of Economic Geology's Houston Research Center and organized skillfully by Karl Schleicher and Robert Newsham of the University of Texas at Austin and BEG.

I have blogged a bit about this workshop at http://www.agilegeoscience.com/journal/2011/6/16/open-seismic-processing-and-dolphins.html (and the next post after that).

I was invited along because of my interest in, and writing about, open source software. I rashly decided not to talk about that, but to take a look at mobile computing platforms instead. You can find my two previous talks about open source on the blog: http://www.agilegeoscience.com/journal/2011/5/16/why-we-should-embrace-openness.html.

s/iphone/android/g



25	Area or Volume
50	Thickness or blank
m or frac	Relief Geometry What's this?
0.2	Porosity φ
55	Net : Gross N:G
0.75	Oil saturation S_O
1.2	Volume factor B_O Oil type
arbitrary	Fudge factor F

CALCULATE

$V = A \times T \times G \times \varphi \times N{:}G \times S_O \times \frac{1}{B_O} \times F$

541 million bbl

99.63 USD/bbl

53.9 billion USD

85.9 million m³

Imperial Store Fetch Email Help

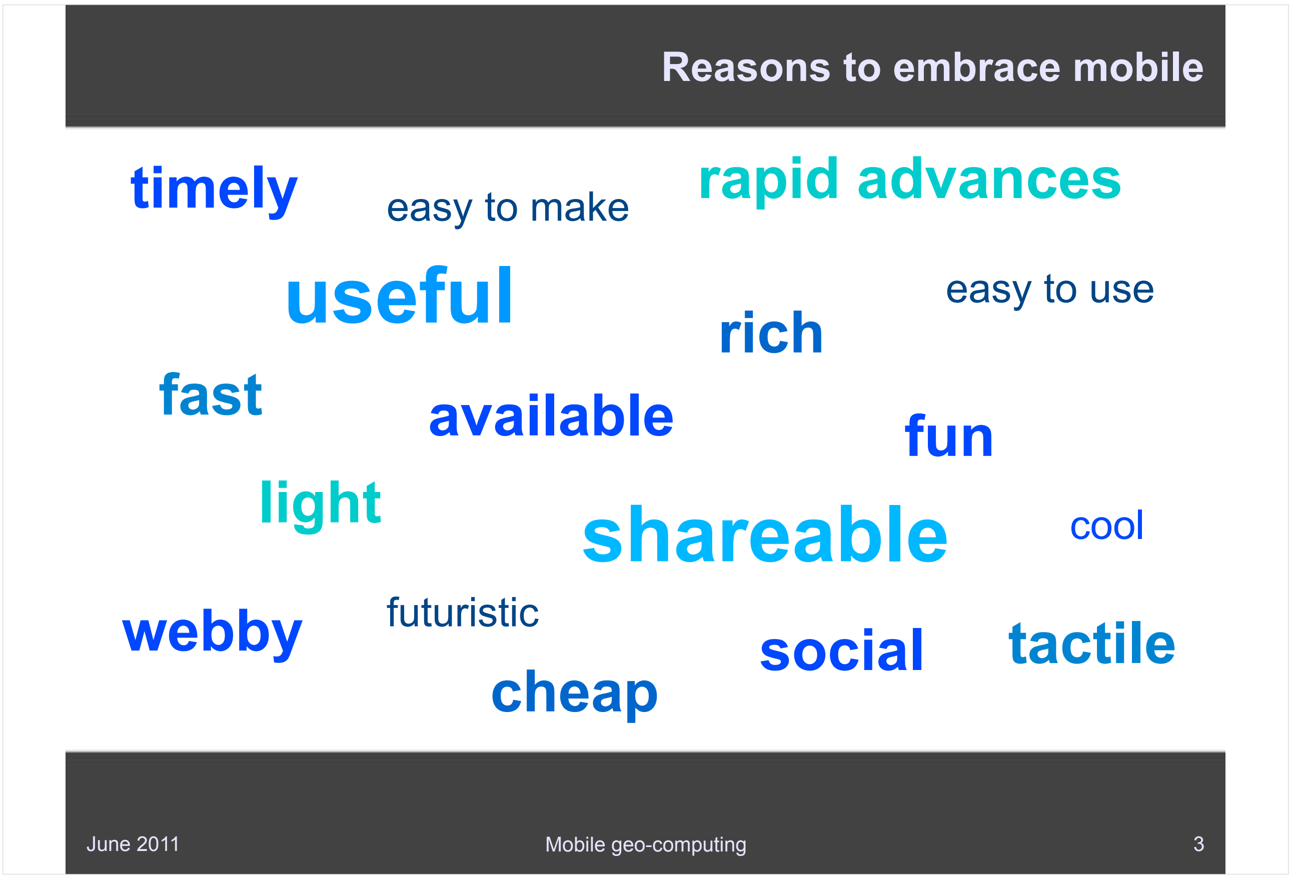
Until 20 April I was an iPhone user. Those phones are a great way to explore mobile technology, because the user experience is so amazing and everything is very easy. But of course the system is closed (notwithstanding jailbreaking), and there are complexities with developing native apps if you're a novice like me. Web apps are simpler, but then there are other complexities...

But on 20 April my new Nexus S arrived. This is Google's second 'pure' phone: no tweaks to the OS by the manufacturer or carrier, and fast upgrades when new versions are released.

Like other modern smart phones, it's an amazing device and much faster of course than my old iPhone 3G. But this isn't why I was excited about it. I was excited about it because I had been dying to try out Google's App Inventor software — a tool for non-programmers to build mobile apps on their Android devices. I'll tell you a bit more about it later.

My dreams came true! Less than a week after getting the phone, I'd built three apps, one of which (volume*, here) was quite functional and robust.

There's no going back now.



Reasons to embrace mobile

timely easy to make rapid advances useful rich easy to use fast available fun light shareable cool webby futuristic social tactile cheap

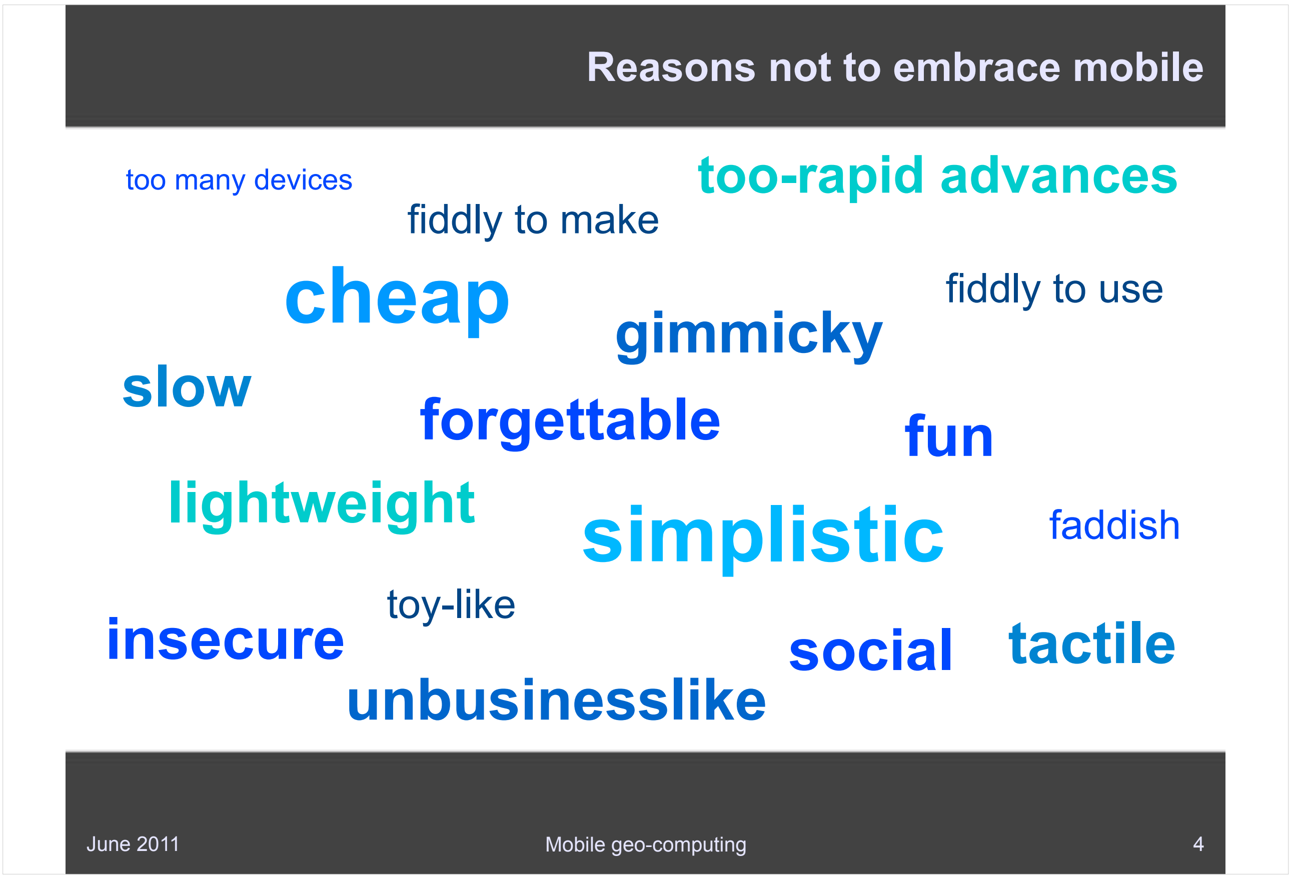
June 2011 Mobile geo-computing 3

So what? What's so great about mobile devices?

Well, here are some reasons I've heard. I'd subscribe to most of them myself. When it comes to the workplace, I think it's worth building these little apps because lots of people have these devices with them all the time, in meetings, at coffee, on the train, etc. I like the idea that we can put a little bit of science in the geoscientists pocket this way, and help people follow through on ideas or questions *right now*. Not later when you're back at your desk with Excel or whatever, but now the problem doesn't seem as pertinent or urgent...

And it's fun to be able to wonder about something like an AVO response, or volumetric scenario, and check it immediately and easily. The back-of-the-digital-envelope is what I called it in a blog post recently.

There are always doubters, of course. Sometimes you can see their point if view: these devices are light and cheap and fun...



Reasons not to embrace mobile

too many devices

too-rapid advances

fiddly to make

cheap

fiddly to use

gimmicky

slow

forgettable

fun

lightweight

simplistic

faddish

toy-like

insecure

social

tactile

unbusinesslike

June 2011 Mobile geo-computing 4

… but they're light, cheap and fun. They're not serious, not for business, not for science. They're gimmicky toys. A fad.

“Put it away and stop fiddling at the table.”

From a developer's point of view, there are issues too: lots of devices, display resolutions, a very rapid development cycle for the OS, tweaked operating environments.

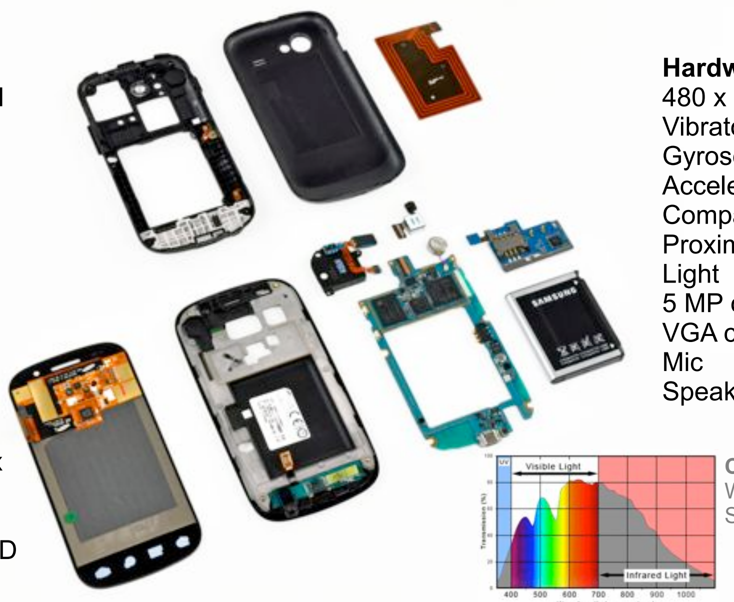
And everything has to be free.

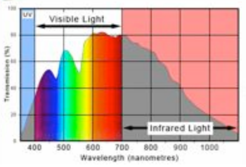
Components in 84 cm^3

Comms
Quad-band GSM
WiFi
Bluetooth
NFC
GPS
μUSB

Compute
1 GHz A8 Cortex
VPower GPU
512 MB mDDR
16GB flash NAND

Hardware
480 x 800 display
Vibrator
Gyroscope
Accelerometer
Compass
Proximity
Light
5 MP camera
VGA camera
Mic
Speaker





CCD reponse
Well into IR
Some UV

Nexus S teardown: iFixIt.com, licensed under CC-NC-SA

Spectrum: http://www.customs.gov.au/site/page5973.asp

So what's in these things? What are they made of?

You don't need me to tell you how incredible the technology is.

Around 8 or 10 sensors, depending on what you count. A GPS with compass and gyroscope. Two cameras. 16 GB storage. Four communications protocols. This thing can even make phone calls. To Australia!

The cameras are pretty cool because their response spectra go well into the infrared. Try pointing one at a remote control or warm object. I have no idea what use this is, but it's awesome.

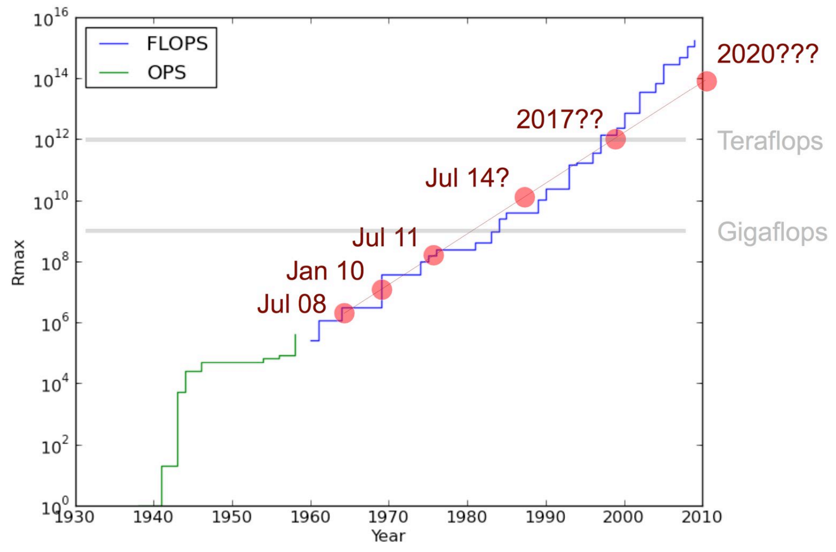
The really interesting parts are the compute power and the display.

The 1 GHz Hummingbird processor is a so-called 'system on a chip'. It's a CPU and graphics chip in one. Note that these things are often underclocked to save power. It only has half a Gig of RAM, but tablets are appearing with 1 GB (eg Motorola Xoom, but not iPad yet).

These CPUs are getting pretty powerful...

Mobile supercomputing

Data from various Wikipedia articles on supercomputers and mobile devices



http://en.wikipedia.org/wiki/File:Supercomputing-rmax-graph.png

Now, these data are highly suspect and the whole idea of measuring compute performance in FLOPS is rejected by some people.

But if we can be at all consistent in how it's measured, which we can't, maybe it's vaguely interesting to look at a trend over time.

The chart shows supercomputer performance over time. It looks a bit like these mobile chips are progressing about four times more quickly than supercomputers did. I have projected the trend brazenly into the near future... We might hit a GFLOPS at the end of next year.

For reference, my iMac does about 40 GFLOPS or so. No doubt there will be limitations like power consumption, etc, but there is progress with batteries too...

Most new phones have a dual core chip. (Not quite accurate, as far as I understand it, but most people seem to describe it this way). The new iPad 2 chip, the A9, is quad core.

Certainly serious work can be done on these devices.

Displays

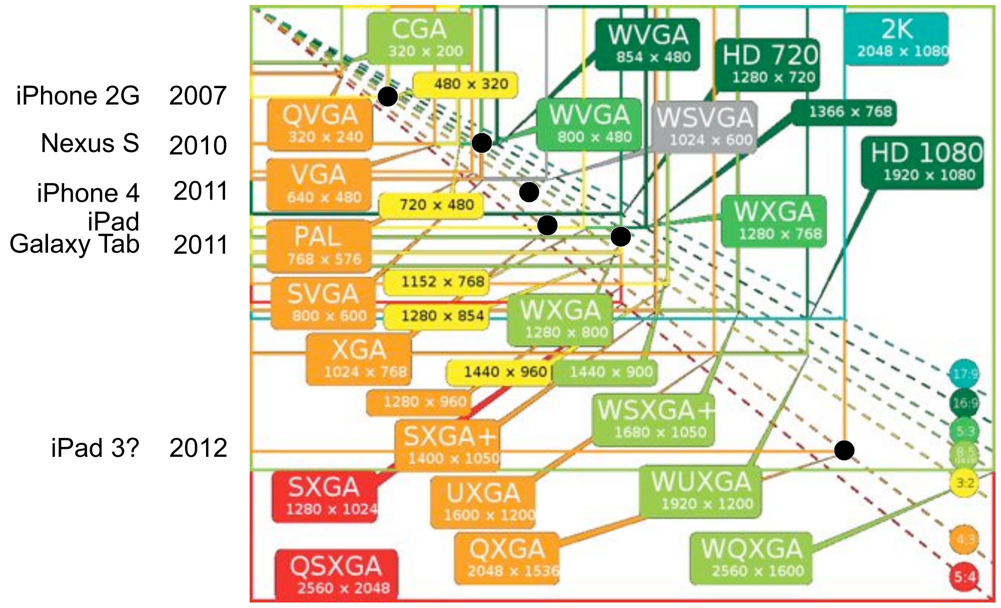


Image: User Flooch at Wikimedia Commons – CC-BY-SA

The other big factor for serious work is the display, of course.

This is a noisy figure, but I've tried to show how these devices are mapping to standard display sizes. In about the last year, these screens have jumped from the very small end firmly into the middle range. The industry leader in terms of technology is the iPad, which has a very high pixel density and thus the sharpest image. I've yet to see a Galaxy Tab 10.1.

The next iPad is rumoured to have double the pixels in each dimension. So by the end of next year, we may have a hand-held desktop-sized display (in terms of pixels, not dimensions), running at a billion FLOPS.

Incredible.

But what can you actually do with this sort of technology?

Highly creative apps

REQUISITOS MINIMOS RESPUESTA EN 2 DIAS
REQUIREMENTS MINIMUM ANSWER EN 2 DAYS

word lens

These low-res images are proprietary and used for illustrative purposes only; I am claiming fair use for them

videosurf

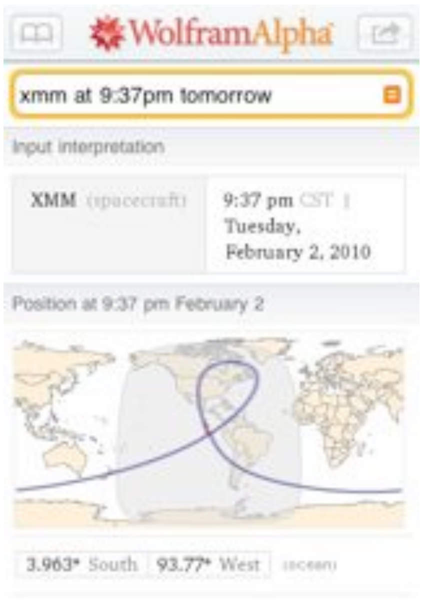
Most of the interaction with the device is through *apps*.

These two apps are interesting. Look at Word Lens. Point the camera at a foreign language, and it translates what it sees. In real time. And then it makes a new image of the translated words. In real time. It's an augmented reality machine and it's brilliant.

Videosurf is equally amazing. Point it at some videofor a few seconds and it does a reverse internet search, finding metadata about the content: what the clip is, who's in it, what others are saying about it, etc. If you've seen Shazam for music or TinEye for images (or now Google Image Search), this is the same idea, but for video.

Notice how these apps, and now many others too, assume you are online all the time. Even my little volumetric app needs to be online for one function — to grab the oil price.

Geek tools



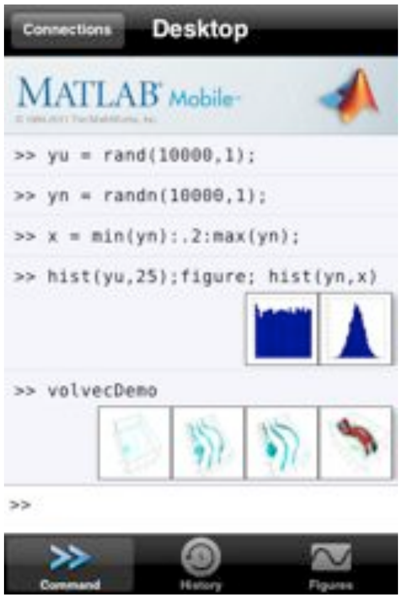
xmm at 9:37pm tomorrow

Input interpretation

XMM (spacecraft) | 9:37 pm CST | Tuesday, February 2, 2010

Position at 9:37 pm February 2

3.963° South | 93.77° West | (ocean)



```
>> yu = rand(10000,1);
>> yn = randn(10000,1);
>> x = min(yn):.2:max(yn);
>> hist(yu,25);figure; hist(yn,x)
>> volvecDemo
>>
```

Various low-res screen shots, claimed as fair use

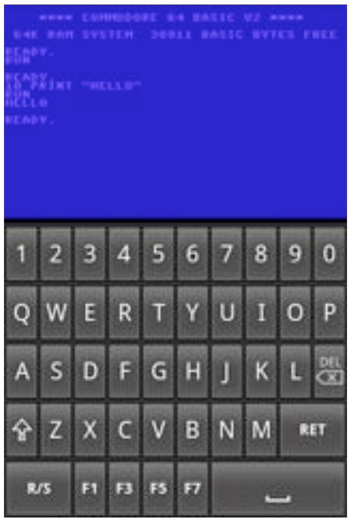
These devices aren't just used for fun, obviously. There are sciencey, geeky tools out there.

The Wolfram Alpha app is great. It offers more than the website version, and is fun to play with and explore. When it came out it cost $50 (!), and was in the top paid apps list for a while so people will pay cool stuff... for a while. It costs $2 now.

The MATLAB app, before you get excited, is really just an interface for your workstation... you can't actually build and run MATLAB programs in there. It's free.

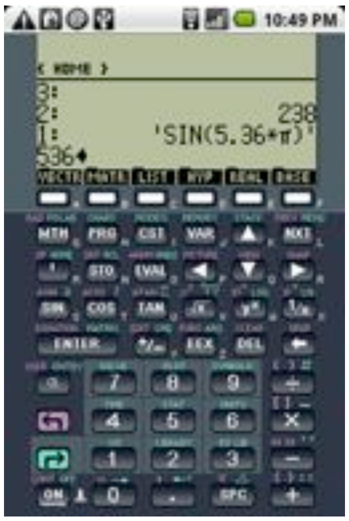
But there are even geekier tools out there too...

Really geek tools



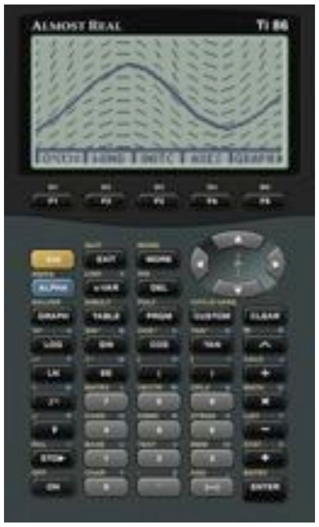
```
**** COMMODORE 64 BASIC V2 ****
 64K RAM SYSTEM  38911 BASIC BYTES FREE
READY.
RUN
READY.
10 PRINT "HELLO"
RUN
HELLO
READY.
```

Frodo C64



```
< HOME >
3:
2:                238
1:    'SIN(5.36*π)'
536♦
```

Droid48



Ti86

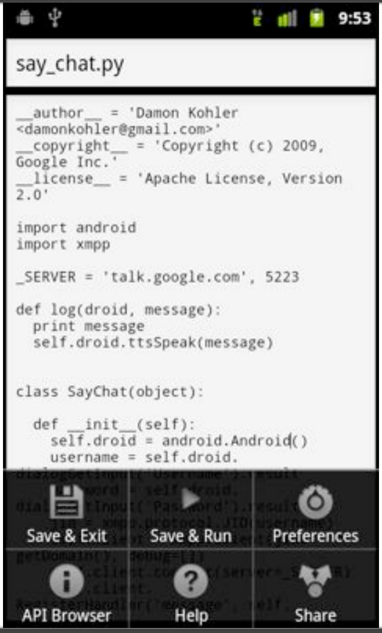
Various low-res screen shots, claimed as fair use

...in case you ever need a Commodore 64 command line. Or an HP48 graphing calculator.

One thing that makes me a bit sad is the developer effort being expended on these things. I mean, they're fun and everything, but... I can't help wondering what would happen if these hours were spent on something a little closer to the world of 'relevant and useful'.

You should see how many seismographs there are on the market!

Really, really geek tools



```
__author__ = 'Damon Kohler
<damonkohler@gmail.com>'
__copyright__ = 'Copyright (c) 2009,
Google Inc.'
__license__ = 'Apache License, Version
2.0'

import android
import xmpp

_SERVER = 'talk.google.com', 5223

def log(droid, message):
  print message
  self.droid.ttsSpeak(message)


class SayChat(object):

  def __init__(self):
    self.droid = android.Android()
    username = self.droid.
```

Scripting layer for Android SL4A

Python, Lua, JRuby, Perl, JavaScript, BeanShell, Tcl, shell

Prototyping

Sensor logging

Hacking

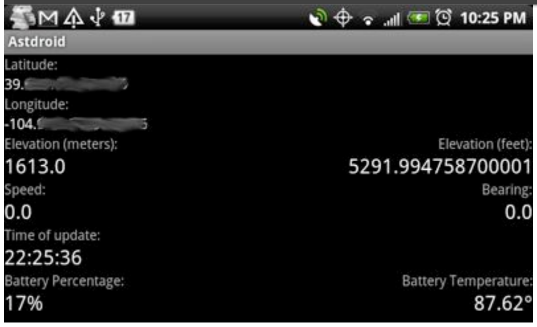
This is one for this audience: Android Scripting Layer. Build and run scripts right on your phone. You can interface any of the phone's functions. No need to build apps!

The tool already supports Python, JavaScript, Jruby, Perl, and shell... and other languages too.

This sort of thing reflects the beauty of having an open platform.

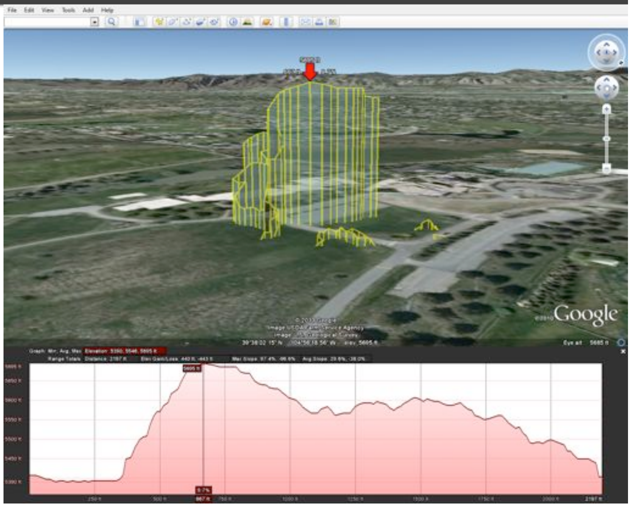
This is the sort of thing you'd need if you were going to, say, send your phone into space and monitor its systems to see how the GPS, comms, and other sensors were coping with the altitude and location.

Androids in spaaace.....



Astdroid

Latitude:
39.[redacted]
Longitude:
-104.[redacted]5
Elevation (meters):
1613.0
Elevation (feet):
5291.994758700001
Speed:
0.0
Bearing:
0.0
Time of update:
22:25:36
Battery Percentage:
17%
Battery Temperature:
87.62°



ASTDROID

Danny Pier's Astdroid project
http://www.astdroid.com/

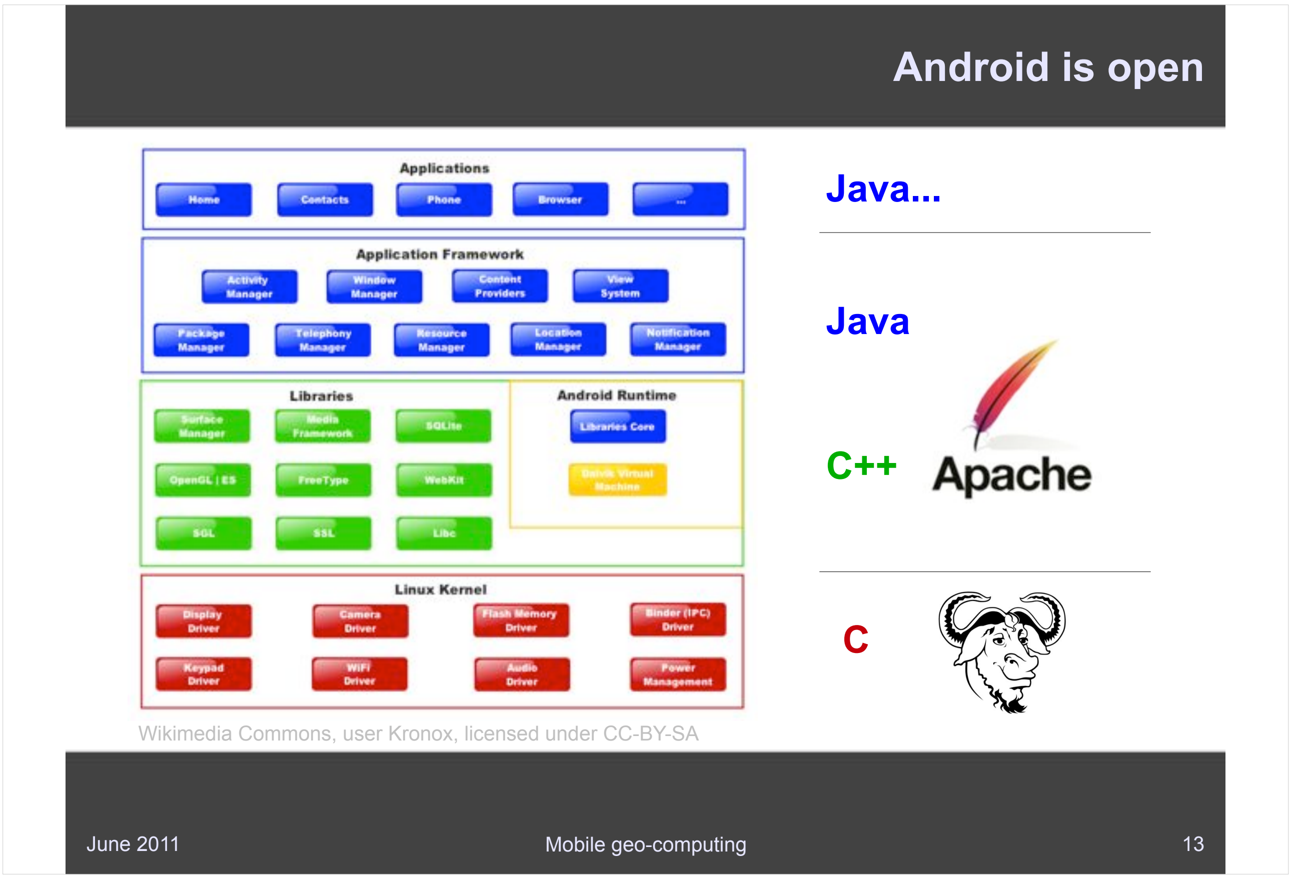
Danny Pier

So the Google team that did send a few phones into space did just that. They blogged about it — it's worth reading.

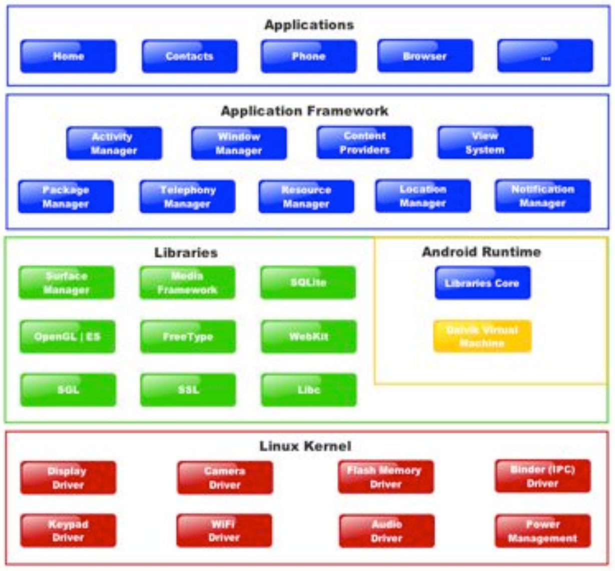
http://googlemobile.blogspot.com/2010/12/android-in-spaaaace.html

I love how Google are always thinking about innovation, but also about having fun with technology, and integrating with their other programs, like here showing the phones' trajectories in Google Earth.

I can't show pictures from the Google space shot, because everything is copyright, but here are some pictures from Danny Pier's wonderful Astdroid project. I think he was the 'pilot' of the first phone in space (well, on its own). I don't know if he used the Android Scripting Layer or not.



Android is open



Wikimedia Commons, user Kronox, licensed under CC-BY-SA

Java...

Java

C++ Apache

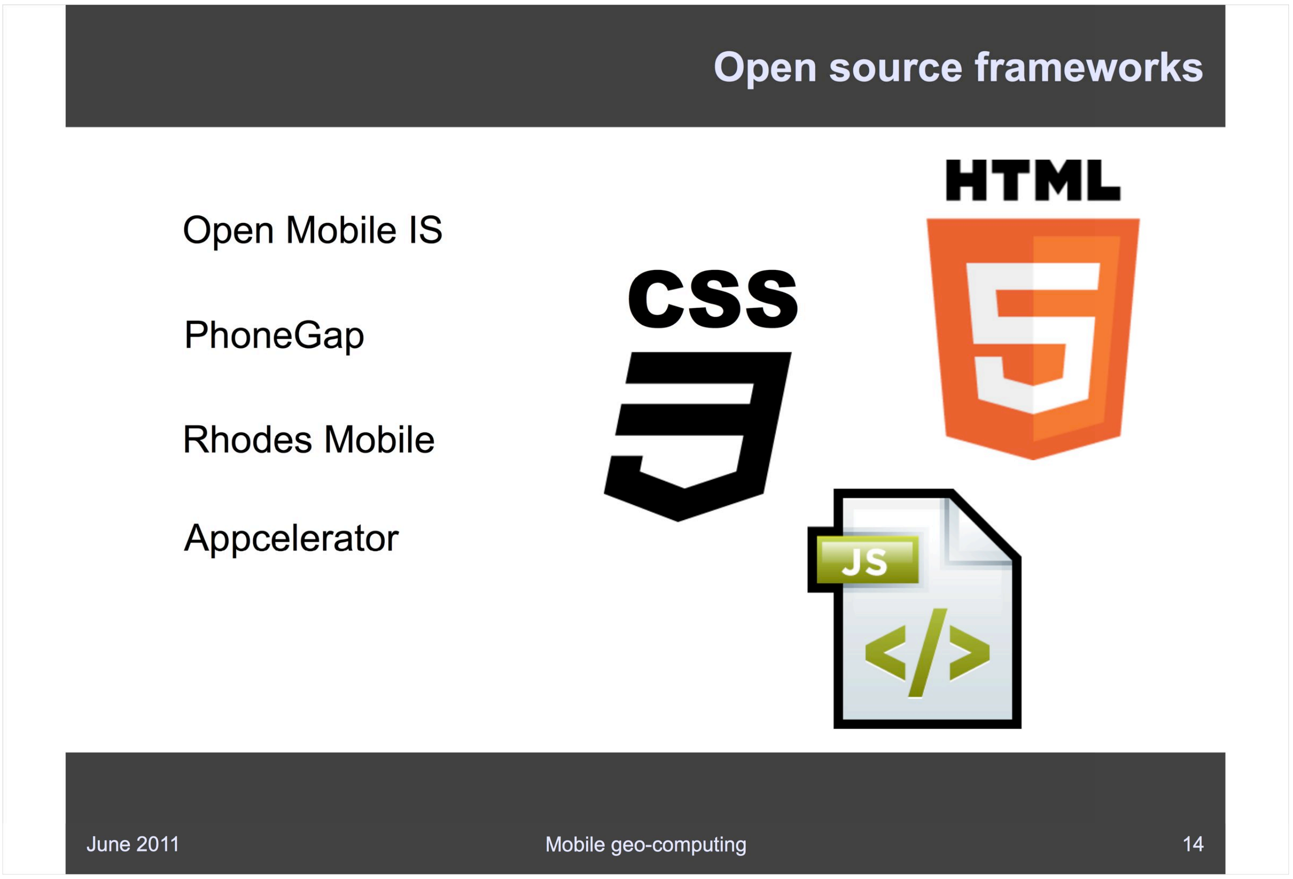
C

So Android is open... what does that mean exactly?

Well, most of the versions of the entire operating system are completely open source and licensed under the Apache license, a permissive license used for about 25% of the projects on Google Code.

And Android is built on top of the Linux kernel, licensed by the GPL of course. Software is usually written in Java, and runs on the Dalvik Virtual Machine, analogous to the Java Virtual Machine on PCs. (It's a register-based architecture rather than stack-based and no, I don't know what that means).

App Inventor apps, just like other apps, live up here in this top layer, with whatever license you like. While they are not written in Java exactly, they are converted to Java from the visual programming environment you build them in.



Open source frameworks

Open Mobile IS

PhoneGap

Rhodes Mobile

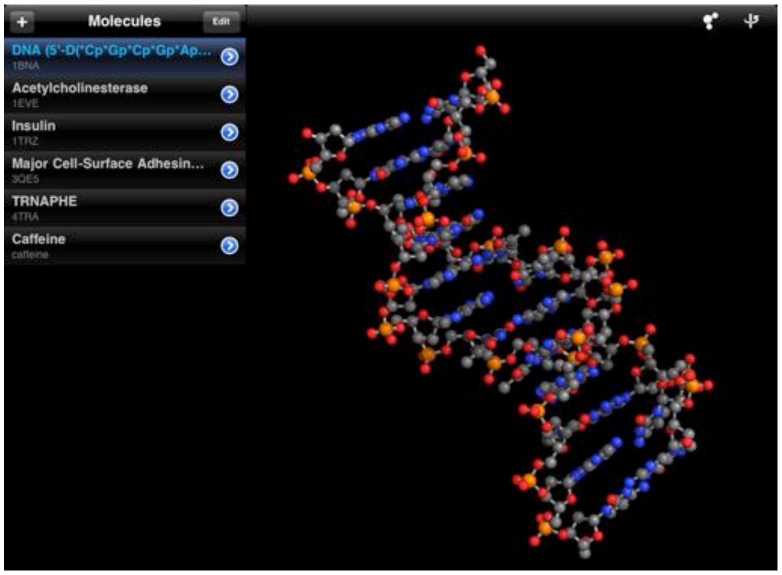
Appcelerator

June 2011 Mobile geo-computing 14

There are alternatives to App Inventor of course. The developer has lots of options for building apps on these devices. Many just write Java in Eclipse, others use frameworks like these (slide).

Some are frustrated with the multi-platform problem. But these open source frameworks exploit HTML5 and JavaScript to build device-agnostic apps. I am just getting into these. It's amazing how they've exploded over the last 3 years.

Open source apps

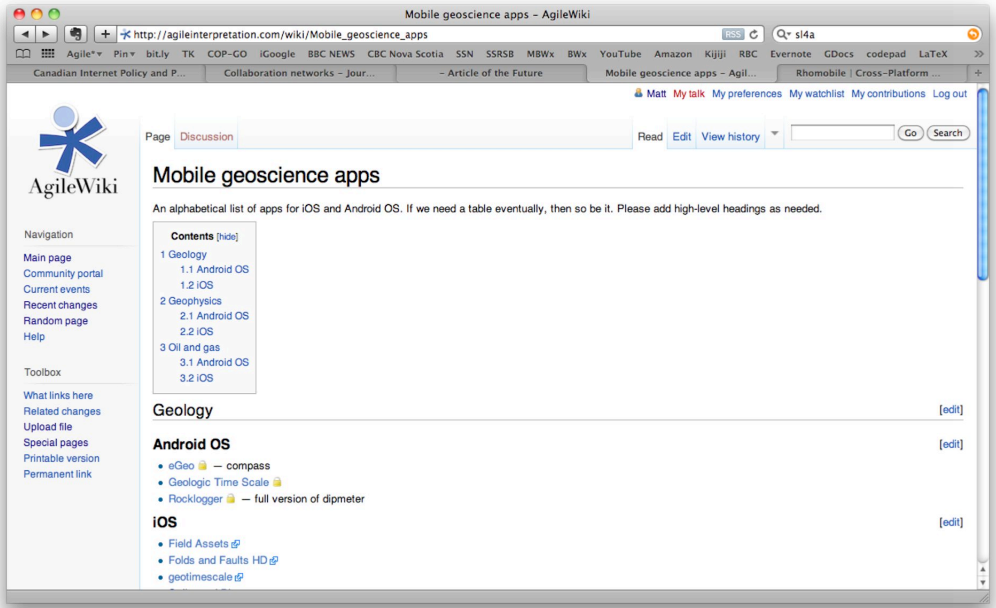


Various low-res screen shots, claimed as fair use

There is also open software for iPhone and Android devices. Just like regular softsare, you'll find them in SVN and other repositories, with all sorts of licenses. Here are a couple of examples: a molecule visualizer and a marine compass toy app.

I am still learning about how to license and publish code, and it may be more complex with App Inventor 'code'.

List of mobile geoscience apps



Mobile geoscience apps – AgileWiki

http://agileinterpretation.com/wiki/Mobile_geoscience_apps

Matt My talk My preferences My watchlist My contributions Log out

Page Discussion Read Edit View history

AgileWiki

Navigation
- Main page
- Community portal
- Current events
- Recent changes
- Random page
- Help

Toolbox
- What links here
- Related changes
- Upload file
- Special pages
- Printable version
- Permanent link

Mobile geoscience apps

An alphabetical list of apps for iOS and Android OS. If we need a table eventually, then so be it. Please add high-level headings as needed.

Contents [hide]

Geology [edit]

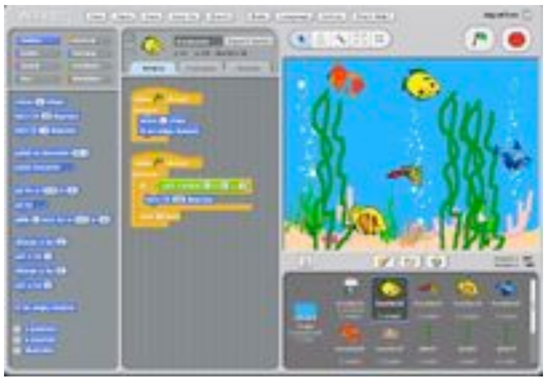
Android OS [edit]
- eGeo — compass
- Geologic Time Scale
- Rocklogger — full version of dipmeter

iOS [edit]
- Field Assets
- Folds and Faults HD
- geotimescale

For what it's worth, I have started a list of geoscience mobile apps in my wiki, AgileWiki... You are free to add to it.

http://agileinterpretation.com/wiki/Mobile_geoscience_apps

MIT Media Lab and Logo



Various low-res screen shots, claimed as fair use

So, what the heck is App Inventor?

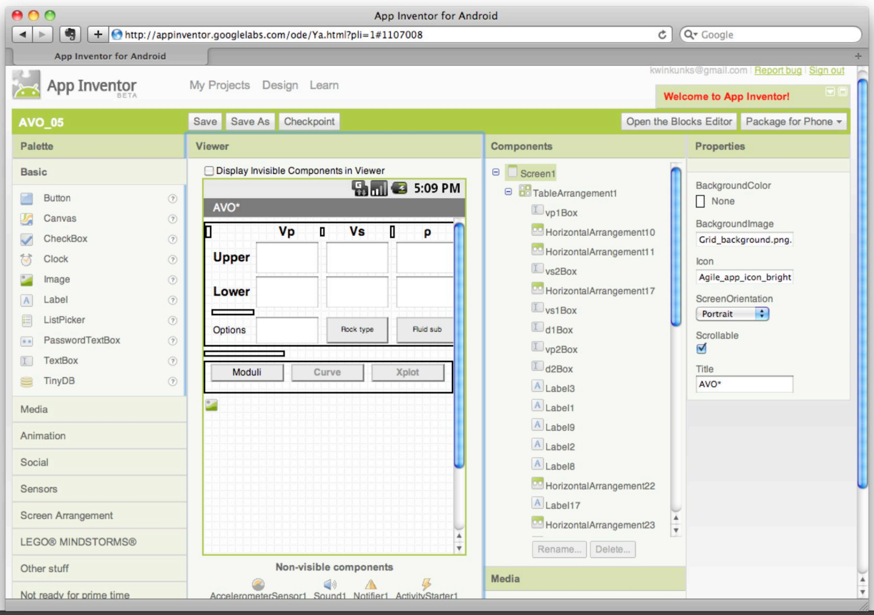
It's based on the logo progamming langauge from MIT Media Lab. This is sometimes referred to as Lisp without parentheses. There is an open source version called OpenStarLogo. It has a graphical version called StarLogoTNG (The Next Generation).

StarLogoTNG is closely related, though I'm not sure how exactly, to OpenBlocks, a completely open source blocks-based programming language. This is the foundation of MIT Scratch, a learning environment for young programmers. These are ancestors of App Inventor, the development of which was largely driven by Hal Abelson of MIT, whilst on secondment to Google.

Another environment in this paradigm you might have seen is Lego Mindstorms, which I think has some common ancestry here too.

These all look like toys to real programmers like this audience. But you can build real, functional, software with these software applications. Indeed, I personally have found the experience profoundly liberating and exciting.

App Inventor for Android



If you have a Google account, you can already just navigate to appinventor.googlelabs.com and start using it. It runs in the browser and stores all your projects in the cloud.

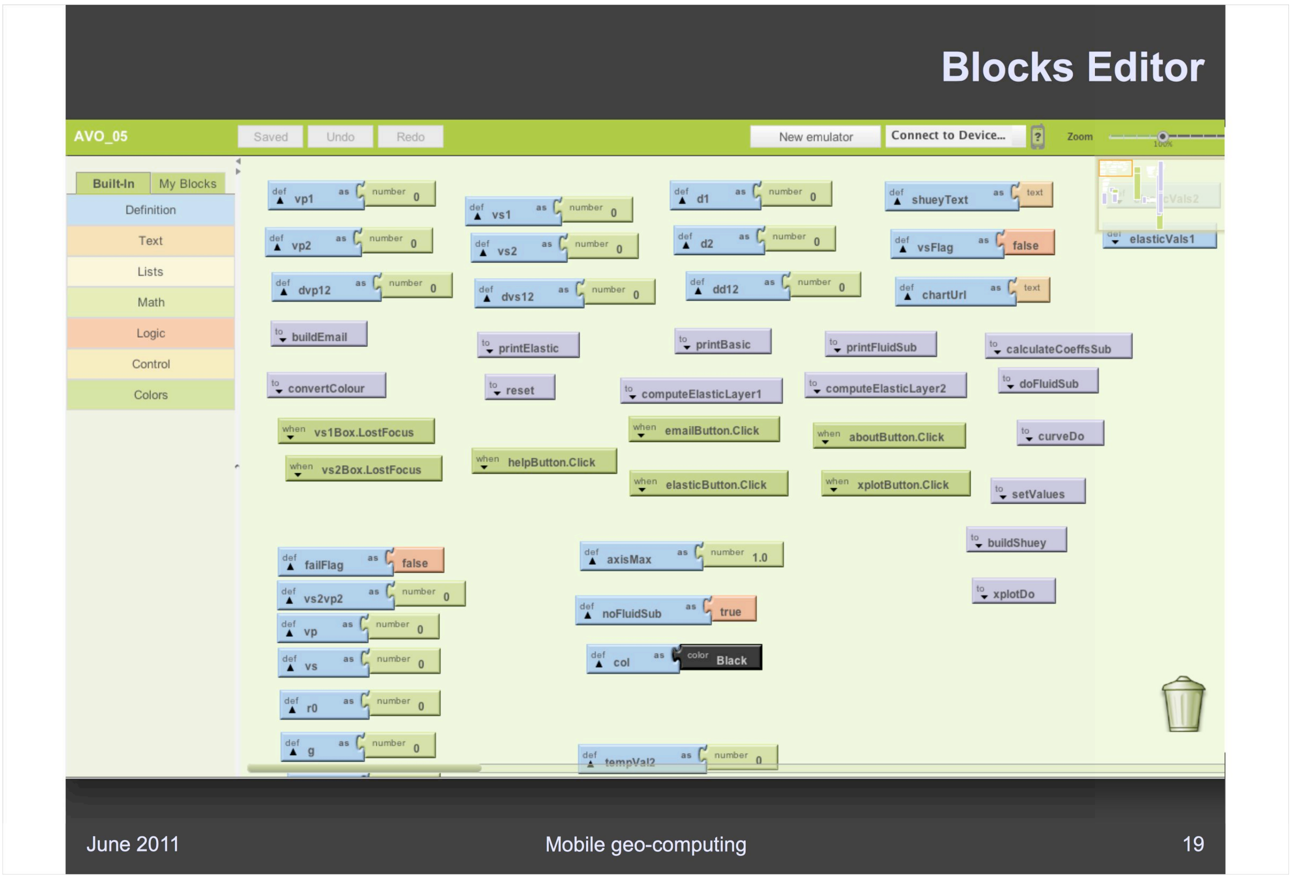
There are three components:

The screen viewer — shown here — where you build the app and specify how it will look and the default parameters for the components

The blocks editor — which I'll show in a sec — where you build the logic for the program, specifying what it will *do*

The device itself, which you tether all the way through development, so that you can run the app on there. All changes are instantly live, so testing is very easy, with no compiling or running.

You can run an emulator instead of running on the device. So you can actually build these things without having an Android device of your own.



Blocks Editor

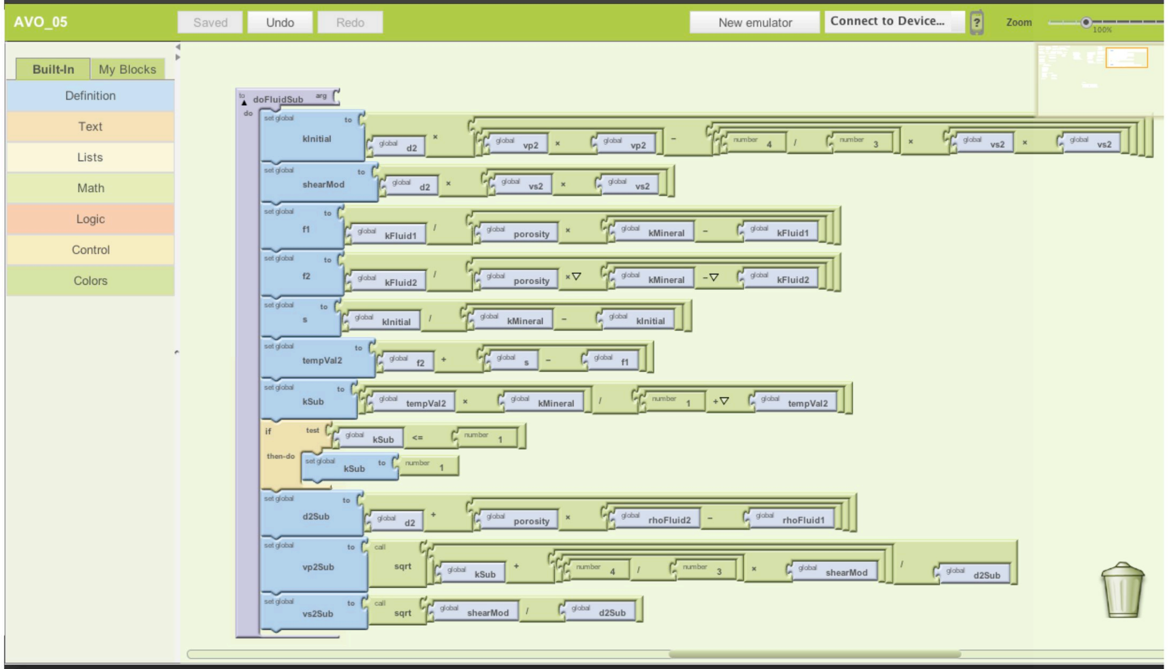
June 2011 Mobile geo-computing 19

This is the blocks editor. It looks a bit chaotic, and indeed that's my only complaint: there's no way to make your code blocks more organized, other than personal discipline.

Some of these blocks initialize variables (point), others define procedures (the purple blocks). And these green blocks are event handlers.

So for example, if I add a button to the viewer screen I showed you before, then I get some event handlers in the blocks editor, like **newButton1.Clicked** for example. There's no need to remember which methods are available, because they all appear in a menu, and I select the ones I want.

Blocks: fluid substitution

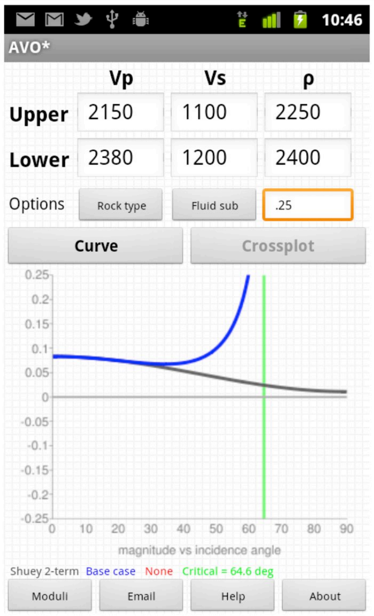


```
to doFluidSub arg
do
  set global kInitial to  global d2 × ((global vp2 × global vp2) − ((number 4 / number 3) × (global vs2 × global vs2)))
  set global shearMod to  global d2 × (global vs2 × global vs2)
  set global f1 to  global kFluid1 / (global porosity × (global kMineral − global kFluid1))
  set global f2 to  global kFluid2 / (global porosity ×▽ (global kMineral −▽ global kFluid2))
  set global s to  global kInitial / (global kMineral − global kInitial)
  set global tempVal2 to  global f2 + (global s − global f1)
  set global kSub to  (global tempVal2 × global kMineral) / (number 1 +▽ global tempVal2)
  if test  global kSub <= number 1
     then-do  set global kSub to number 1
  set global d2Sub to  global d2 + (global porosity × (global rhoFluid2 − global rhoFluid1))
  set global vp2Sub to  call sqrt ((global kSub + ((number 4 / number 3) × global shearMod)) / global d2Sub)
  set global vs2Sub to  call sqrt (global shearMod / global d2Sub)
```

Another example, this time an expanded block — you see all the logic. The math is clunky because you can only do operations in pairs. All these variables are global — I gather some programmers don't like this, but it's great for novices because you don't have to track which methods 'know' about a parameter.

If you want to see one of these apps up close, you can visit http://www.agilegeoscience.com/download and look for Elastic.zip. You will need to upload this to App Inventor, where you can inspect the 'code'.

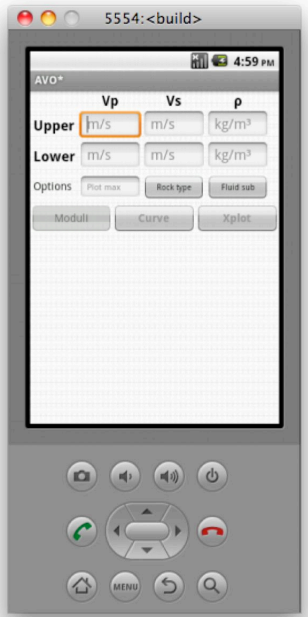
On the phone


AVO*

	Vp	Vs	ρ
Upper	2150	1100	2250
Lower	2380	1200	2400
Options	Rock type	Fluid sub	.25

Shuey 2-term Base case None Critical = 64.6 deg

screen shot


5554:<build>

AVO*

	Vp	Vs	ρ
Upper	m/s	m/s	kg/m³
Lower	m/s	m/s	kg/m³
Options	Plot max	Rock type	Fluid sub

emulator

Here's how one of my apps looks on my phone (left) and in the emulator. It's just a simple calculator for amplitude-vs-offset modeling. The nifty thing here is the graph.

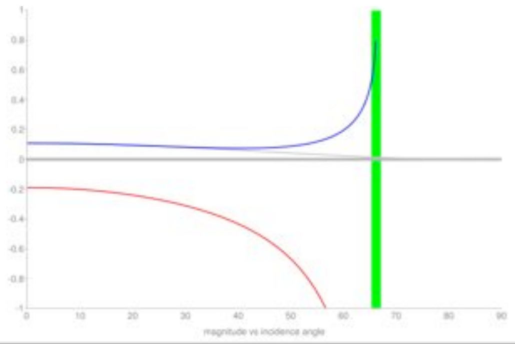
I was initially stumped by drawing a graph, and didn't really know where to turn. Did I need to draw one with the raster tools? Maybe use a package like agraphengine, but that's only for 'real' Android coding — you can't have arbitrary code blocks in App Inventor yet.

Google to the rescue with their Charts web API. I can simply add an image to my app, then instead of calling on an in-app file (a PNG or JPG, say), I can provide a URL.

It's not pretty...

The programmable web

http://chart.googleapis.com/chart?cht=lc&chd=t:-1l-1l-1&chds=-1,1&chs=600x400&chf=bg,s,FFFFFF&chco=BBBBBB,FF0000,0000FF&chxr=0,0,90,10l1,0,100l2,-1,1&chxt=x,x,y&chxl=1:lmagnitude%20vs%20incidence%20anglel&chxp=1,50&chxs=0N*f0*&chm=R,00FF00,0,0.72545,0.74545lr,AAAAAA,0,0.495,0.505&chfd=0,x,0,90,0.1,0.10827-0.11312*(sin(x*0.0174533))%5E2l1,x,0,90,0.1,0.01786-0.02044*(sin(x*0.0174533))%5E2-0.2075/(cos(0.5*x*0.0174533%2B0.5*asin(sin(x*0.0174533)*1.09302)))%5E2-0.15206*(sin(x*0.0174533))%5E2l2,x,0,90,0.1,0.06383-0.07305*(sin(x*0.0174533))%5E2%2B0.04444/(cos(0.5*x*0.0174533%2B0.5*asin(sin(x*0.0174533)*1.09302)))%5E2-0.09951*(sin(x*0.0174533))%5E2



...but a URL like this one returns a graph, as here.

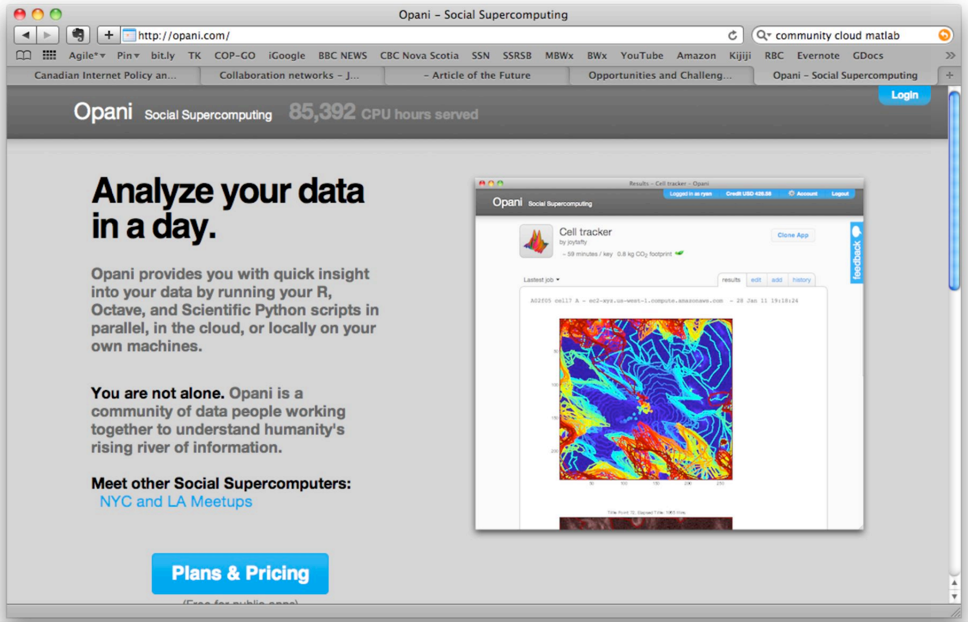
Everything after chfd (right in the middle) is just the function definition: one for the two-term Shuey equation, and two for the Aki-Richards equation. So my charting problem is reduced to simply contructing this URL as a text object, which is pretty easy.

Of course, it means the user has to be online, but that's increasingly common, at least in cities and offices.

This maybe be trivial to you, but for me, this is a huge epiphany. Not only is there an ecosystem of code snippets and code APIs, but now this ecology of web APIs, communicating through simple URLs, is mid-bogglingly inspiring to me. If you've heard of the mashable web, or the programmable web, this is what that is.

One of my new projects is to build a web API of my own to serve up synthetic gathers. Provide a wavelet type, scale, etc, maybe a reflectivity series, and I can pass back a PNG of a gather... so now maybe I can really do some quite hard, sciencey, things on the phone, because the compute and rendering is happening somewhere else.

Social, web-based supercomputing



Opani Social Supercomputing 85,392 CPU hours served

Analyze your data in a day.

Opani provides you with quick insight into your data by running your R, Octave, and Scientific Python scripts in parallel, in the cloud, or locally on your own machines.

You are not alone. Opani is a community of data people working together to understand humanity's rising river of information.

Meet other Social Supercomputers:
NYC and LA Meetups

Plans & Pricing

Once you know about these web APIs it starts you thinking about what else you can do in the cloud. Quite a lot, it turns out. Of course, there's massive storage in the cloud. I run my wiki on Amazon's Elastic Compute EC2 service... which is basically just a Linux box in the sky. Or 10 Linux boxes... or 100, or 1000: it's elastic!

Another service built on EC2 is Opani (shown). You can run Python, MATLAB, Octave, R, etc. Prices are scalable too: $5/month plus 9 cents per job gets you unlimited storage on Amazon S3 (at their rates), cloud processing at 10 cents per core hour. Awesome.

Really, if you can dream it up, there's probably already a way to do it with a mobile device.

The only problematic piece is the large data we often use in geophysics. Both moving it around, and the possible privacy issues some might have with the idea of moving it around. I don't buy this privacy point though... we already let service companies host our data online, we email all sorts of things daily, and send data back ad forth with JIPs. It's just a smokescreen objection and it will go away as people get used to the idea. The practical issues are another matter.

Projects

Meta-app launcher

Cheatsheets

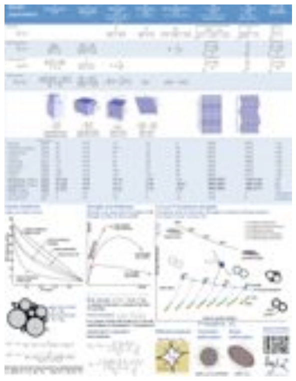
Web API for forward model

Seismic attributes from photo

Networked sensors, arrays

2D model from photo

Optimization games



So where am I going with all this... I don't really know. I started this exploration about two months ago, as a sort of professional hobbyist. I am very excited about this technology, especially Android and App Inventor, as a democratization of experiment-driven innovation.

We have a few projects on the go (shown)... I don't know which will work out first, if any. We want to expose more of our popular cheatsheets in apps, saving people from having to remember details. I want to try building a web API to deliver gathers. I think it would be fun to be able to take a photo of something and apply horizon attributes to the picture. Or turn it into an impedance model and make a synthetic from it. And there are lots of things you could try with a class of geophysics students, like games, or building sensor arrays with cell phones. So many possibilities!

My real message is this: don't underestimate the power of toy-like tools to energize and inspire people. The next time you're thinking about an interface for your high-science algorithms, think about using a visual programming interface, so that 'ordinary geophysicists' can play with your algorithms.