# Scratching the Itch

How to attack slow applications, with particular reference to Scratch on a Raspberry Pi

tim Rowledge
tim@rowledge.org
for the Raspberry Pi Foundation
ESUG ’14 Cambridge UK

Scratch lives at scratch.mit.edu - though be warned, the latest version is not Smalltalk based at all.

Raspberry Pi lives at www.raspberrypi.org

tim lives at www.rowledge.org/tim

“How many of you like the Ocean?”

For the full musical experience of the following five slides, see the first 35 seconds of https://www.youtube.com/watch?v=hQeeQjdU2Bk And then enjoy the rest of Steam Powered Giraffe.

“It’s pretty wet, huh?”

“And how many of you like Fishing?”

“That’s right, *nobody*.
Because it’s *Boring*”

“But how many of you like fighting
Skeleton Pirates on the back of Killer Whales?”

–Steam Powered Giraffe

## Making code better is that exciting!

- Scratch is an important application, perhaps the most used Smalltalk code out there
- Raspberry Pi is wildly popular as an educational tool, a toy, a personal dynamic medium
- Scratch is slow on a Pi :-(
- We need to fix that.
- Saddle up my Orca!

Scratch on Pi has probably over a million regular users. That’s a scary large number of customers.

- Squeak… isn't that some sort of old fashioned interpreted language?
- Scratch is highly graphical, so it must be slow because of poor graphics in Squeak
- Pi is ARM cpu based, so can't possibly be fast
- Pi is cheap… yada-yada

## We all know what to do

- Dream up some benchmarks
- Run some tests
- Do a few profiles
- Fix problems
- Profit!!!!

This is what we normally do, or at least what we get told to do. Like most of what we are taught, there is a lot more to it.

# Except…

- Benchmarks are hard to devise
- Tests don’t, often
- Profiles only show where time is spent, not if it is spent wisely
- Fix which problems? Code quality, algorithm choice, system design…

Benchmarks are a morass of bad code, stupid ideas, inadequate thinking, simplistic approaches and general wrongness. And then we add all the numbers and pretend it means something.

Tests can be worse.Even well thought out tests go out of date.

Profiling is perhaps the most misunderstood issue of all.

# Specifically…

- Scratch/Pi was claimed to spend most time doing bitblt. Not actually true even for a totally graphical test.
- Code to do a simple swap between two backgrounds involved many bitblts, mostly of ‘paint’ type.
- Surrounding graphic ‘frame’ used several complex translucency bitblts for a virtually invisible effect
- Multiple (ab)uses of #isKindOf:. Sigh.

- Actual bitblt time was 22% with a 6fps display rate
- Pre-composite graphics to avoid multiple paint
- Dump invisible prettiness
- Avoid #isKindOf:
- 10fps

Here are some example changes made in the early part of the project.

# BitBlt actually *is* slow on a Pi

- The Pi SoC was originally intended for phones - essentially an iPhone 1 logic board
- Memory bandwidth to the ARM is not huge
- We improved bitblt performance by a large factor with carefully written ARM assembler to replace some important routines; thanks to Ben Avison
- Use ARM PLD instruction to preload scanlines
- Make better use of ARM magic than C compiler can typically manage plus some SIMD trickery.
- A lot of 1@1 sized blts; turned out to be a way of testing for A touching B. Replace with a very simple single-pixel test prim. Good case of a better algorithm rather than makee-faster

The ARM bitblt improvements are in the trunk VM code, as is the pixel testing prim.

# They did what?

- Scratch uses an interesting and clever execution simulator to run the scripts; kind of a virtual machine running on the virtual machine.
- Some really, really bad code in some important places.
- #isKindOf: all over the place plus lots of state based behaviour instead of class based, thus wasting the benefits of the message sending VM.
- Specialise some key parts of the Scratch engine to know what they do and some important sections run 8x faster. Suddenly Asteroids is playable.
- Still a lot that could be done in this area
  - number/string conversion/polymorphism within Scratch costs a lot of time
  - still at least 50 extraneous #isKindOf: at last count

Scratch’s execution mechanism is an excellent example of a very clever idea that didn't get looked after as well as it deserved. We’re trying to fix that.

# Script Editing

- Users drag script elements from palette to editor and around editor to build and change a script
- Dragging a large stack of script blocks involves creating a large bitmap, rendering all the blocks into it and then animating dragging a mostly transparent bitmap.
- Added caching of the fiddly bits, rounded or <> ends of argument morphs to speed up this rendering
- In many cases it's almost as fast to (re)draw each script block individually at each step to save so much transparency work.
- Perhaps we will change to dragging a symbolic script chunk
- Changes in place improved the system from 'unusable by my students' to 'wow, amazing'

# Moving to a modern image

- Scratch was originally written in a 2.4 era image and much hacking was done
- Lots has changed
  - importantly the old image could not run on latest VMs
  - StackVM as a first step; major work
  - Required extracting code from old system, porting forward to 4.5 image
  - Oh my - look at the list ->

This is what happens when code gets left to mould. It becomes a major effort to bring it back to a modern world.

- Huge changes to Morph
- Changes to event handling - #mouseDown: etc
- Scratch relies upon recorded keyboard/mouse state as well as event changes
- Rewrote (with much help from Nic Cellier) the core text scanning to remove pointless duplication and speed up quite a bit
- #copy-ing chaos
- Custom i18n solution
- Message name clashes as facilities added to general Morph that were Scratch specific
- Morphic layout changes
- Morphic drag/drop infrastructure
- Dictionary>>do: changed semantics!

And here is a list of at least some of the problem areas that have been tackled.

# Going Faster

- NuScratch beta is now near final and runs on the StackVM
- Next step is CogVM/ARMv6
- Eliot is freakin' amazing
- Tantalisingly close to running in simulator
- Huge amount of subtle tricky stuff
- Sends, SICs, PICs and most returns done
- One day, ARMv8

# Where can I get this wonderful stuff?

- Beta release for ARM is on
  - http://github.com/raspberrypi/scratch
- Beta8 tgz is latest
  - extract the image and run on OS X, Win-whatever, other linuxen.
- Feedback!