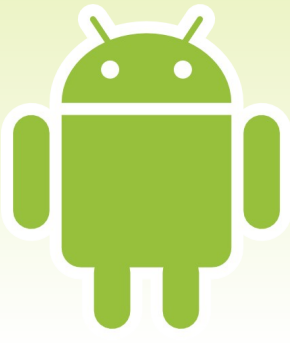
Android Builders Summit

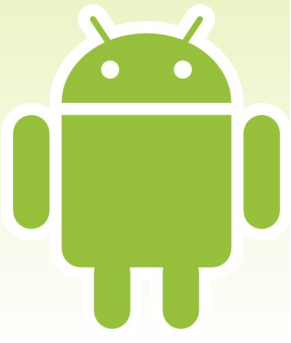
Android App Tuning Techniques

Part Two: Jank Busting



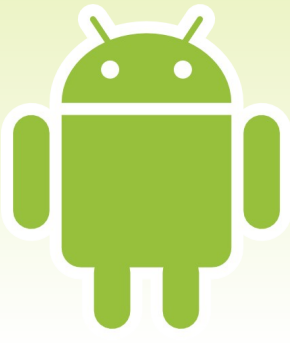
What Is Jank?

- In UI, Delays in Animations
  - E.g., instead of smoothly scrolling, the UI pauses and jerks the scrolling
- Source: Too Much Work on Main Application Thread
  - Project Butter = 60fps = 16ms/frame
  - Individual callbacks must be cheap
  - Popular callbacks must be crazy cheap
    - E.g., `getView()` of a ListAdapter



Do We Have Jank?

- Subjective
  - "You know jank when you see it"
  - Problem: differing tolerance
- Objective
  - Dropped frames = jank
  - If achieving 60fps without drops, should be no noticeable jank



The #A4C739 Mantra

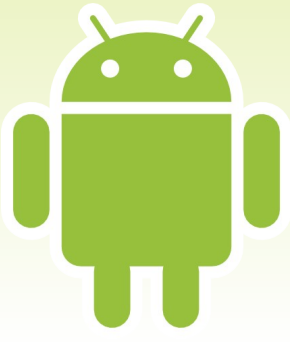
- Reckon
- Reduce
- ~~Reorder~~
- ~~Reuse~~
- ~~Recycle~~
- ~~Cheat~~

Reckon: Logging

- Simple LogCat Messages
- TimingLogger
  - Collects time splits, dumps results to LogCat
- Hugo
  - Adds @DebugLog annotation for methods
  - Logs inputs, outputs, times of method execution
  - Requires Gradle
  - From Jake Wharton

Reckon: Choreographer

- In Java, API For Tying Into Frames
  - E.g., `postFrameCallback()`
- In LogCat, Complaints About Dropped Frames
  - If Choreographer says that you are dropping frames, you have jank
  - There may be a *de minimis* threshold where Choreographer does not complain



Reckon: StrictMode

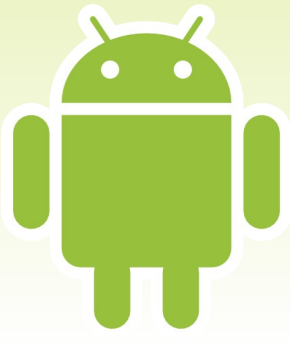
- Detects Common Runtime Problems
  - Disk I/O on main application thread
  - Network I/O on main application thread
    - Enabled by default on Android 4.0+
- Custom Penalty
  - Log to LogCat
  - Flash the screen
  - Crash the process
- Enable if `BuildConfig.DEBUG` and API Level 9+

Reduce: StrictMode

- Move I/O to Background
  - AsyncTask for cheap, disposable stuff
  - IntentService for stuff that should survive a configuration change
  - WakefulBroadcastReceiver/WakefulIntentService for stuff that should happen even if device would ordinarily power down

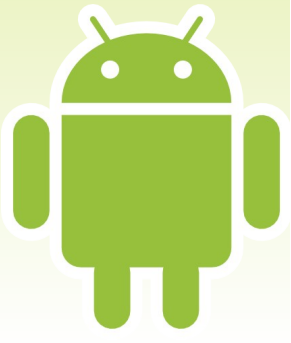
Reckon: gfxinfo

- Utility to Capture Frame Times
  - Ties into developer options on device, so must be opted-into, as capturing adds a bit of overhead
- Command-Line Tool
  - No IDE integration at this time
- Raw Output
  - Dump of per-frame time for draw, execute, process
  - Need to import into a spreadsheet or otherwise analyze to learn anything



Reckon: gfxinfo

- Instructions
  - Enable Developer Options (7 taps on build number)
  - Enable "Profile GPU Rendering" in Developer Options
  - Try your app
  - **`adb shell dumpsys gfxinfo`**
  - Disable "Profile GPU Rendering"
  - Create a stacked bar chart or something…
  - Look for frames that took > 12ms



Reckon: systrace

- System-Level Tracing
  - CPU, GPU, etc.
- Benefits Over gfxinfo
  - Report generated in HTML for you
- Downsides Over gfxinfo
  - Cryptic output
  - Requires Python or IDE / `monitor`

Reckon: systrace

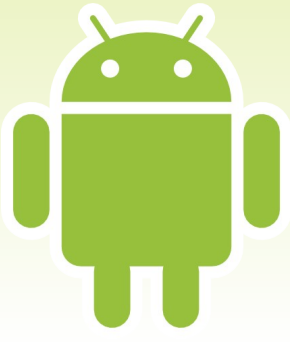
- Instructions
  - Enable Developer Options, "Enable Traces"
  - Command Line
    - Android 4.3+: `python systrace.py --time=10 -o mynewtrace.html sched gfx view wm`
    - Android 4.2 and below: more complicated…
  - DDMS/monitor
    - Toolbar button and dialog replacement for above
  - Run test (then clean up "Enable Traces")
  - Examine results

Reckon: Traceview

- Method Tracing
  - Records each method invocation, time it took
  - Builds tree of calls, to drill down into slow spots
  - Helps you identify more specifically what you are doing

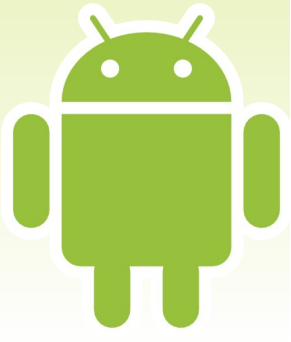
Reckon: Traceview

- Interactive
  - Start/stop method tracing toolbar buttons
    - DDMS
    - Monitor
  - Automatically opens results in Traceview view (Debug perspective)
- Programmatic
  - Debug class, `startMethodTracing()` and `stopMethodTracing()`
  - Must pull trace file off device, load manually



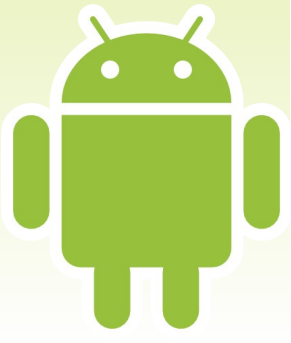
Reckon: Traceview

- Results
  - Top pane = threads (rows) versus time
  - Bottom pane = call details
    - Inclusive time = call and all downstream calls
    - Exclusive time = time spent purely in the method itself, not counting downstream calls
- Techniques
  - Find expensive or frequent stuff
  - Drill down to find something you recognize



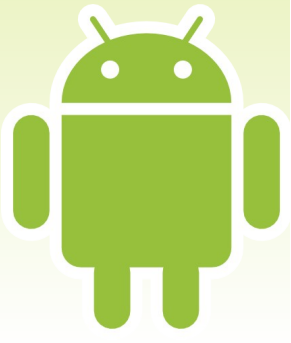
Reduce: systrace and Traceview

- Avoid Stupid Stuff
  - Example: animating widget sizes
- Move Logic Off Main Application Thread
  - Helps with jank, not with overall CPU consumption
- Cache Computation Results
- Micro-optimizations
  - Loop invariants, etc.



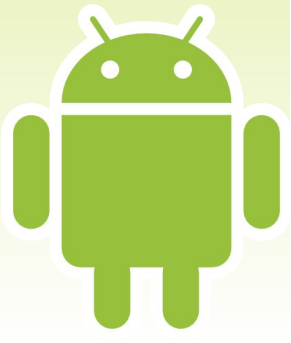
Reduce: systrace and Traceview

- Less-Frequent Processing
  - Example: syntax highlighting
- Faster Implementations
  - Better algorithms?
    - Example: syntax highlighting
  - Renderscript?
  - NDK?
- Offload to Server
  - Always or optional



Reckon: Hierarchy View and uiautomatorviewer

- Show View Hierarchy
  - Containers and children
- Hierarchy View
  - Integrated into IDEs, plus `monitor`
  - Full details of widgets
  - Only works with emulator or modified project
- `uiautomatorviewer`
  - Works on production devices, with any app
  - No widget IDs, less information overall

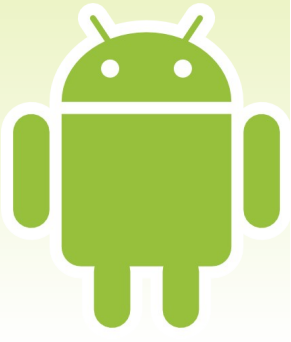


Reduce: Hierarchy View and uiautomatorviewer

- Too-Deep Nesting
  - Run risk of StackOverflowError at ~15 layers
  - Makes layout of containers expensive
- Too Many Views
  - Each view takes time to draw, so too many views = too much time drawing
  - Also, each widget consumes minimum 1K heap space, so too many widgets = too much heap consumption
- Overdraw

Reckon and Reduce: Overdraw

- Drawing the Same Pixel, Over and Over
- Common Sources
  - Z-axis ordering (widgets on top of widgets)
  - Foreground and background both set
  - Activity theme and occluding content
    - E.g., setting a background color for the activity, then hiding all of it with a ListView that has no transparent parts



Reckon and Reduce: Overdraw

- Instructions
  - Enable Developer Options
  - Enable "Show GPU overdraw" in Developer Options
  - Interpret colors
    - Blue = 1x overdraw (OK, though may show a "quick win" for large areas)
    - Green = 2x overdraw (OK in medium patches)
    - Light red = 3x overdraw (OK only in tiny patches)
    - Dark red = 4x overdraw (OMG)