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

Roundup of Tools and Techniques

- Choreographer
- StrictMode
- gfxinfo
- systrace
- Traceview
- Hierarchy View and uiautomatorviewer
- Overdraw

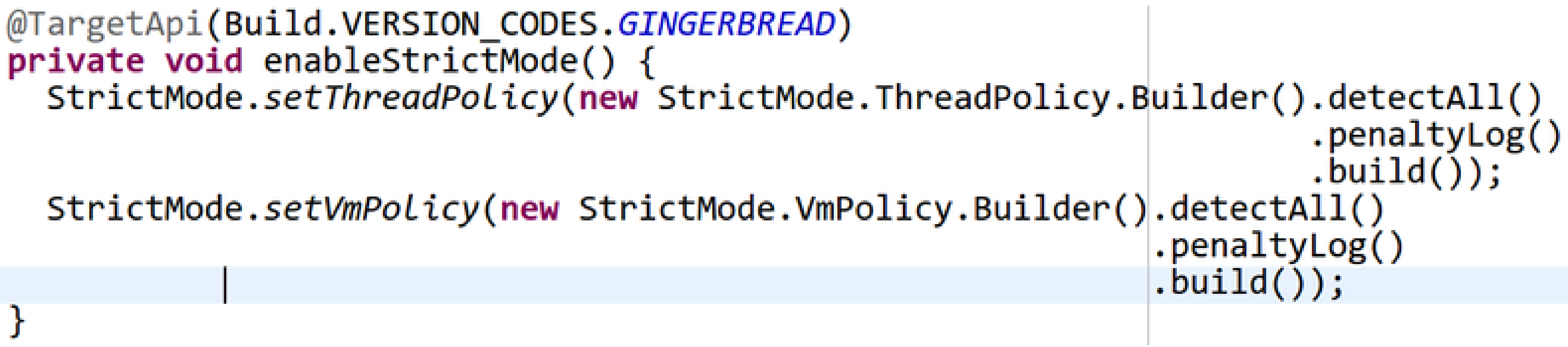
Choreographer

- 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

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
  - Typical: log to LogCat
  - Aggressive: crash the process
- Enable if `BuildConfig.DEBUG` and API Level 9+

```java
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.GINGERBREAD
    && BuildConfig.DEBUG) {
  enableStrictMode();
}
```



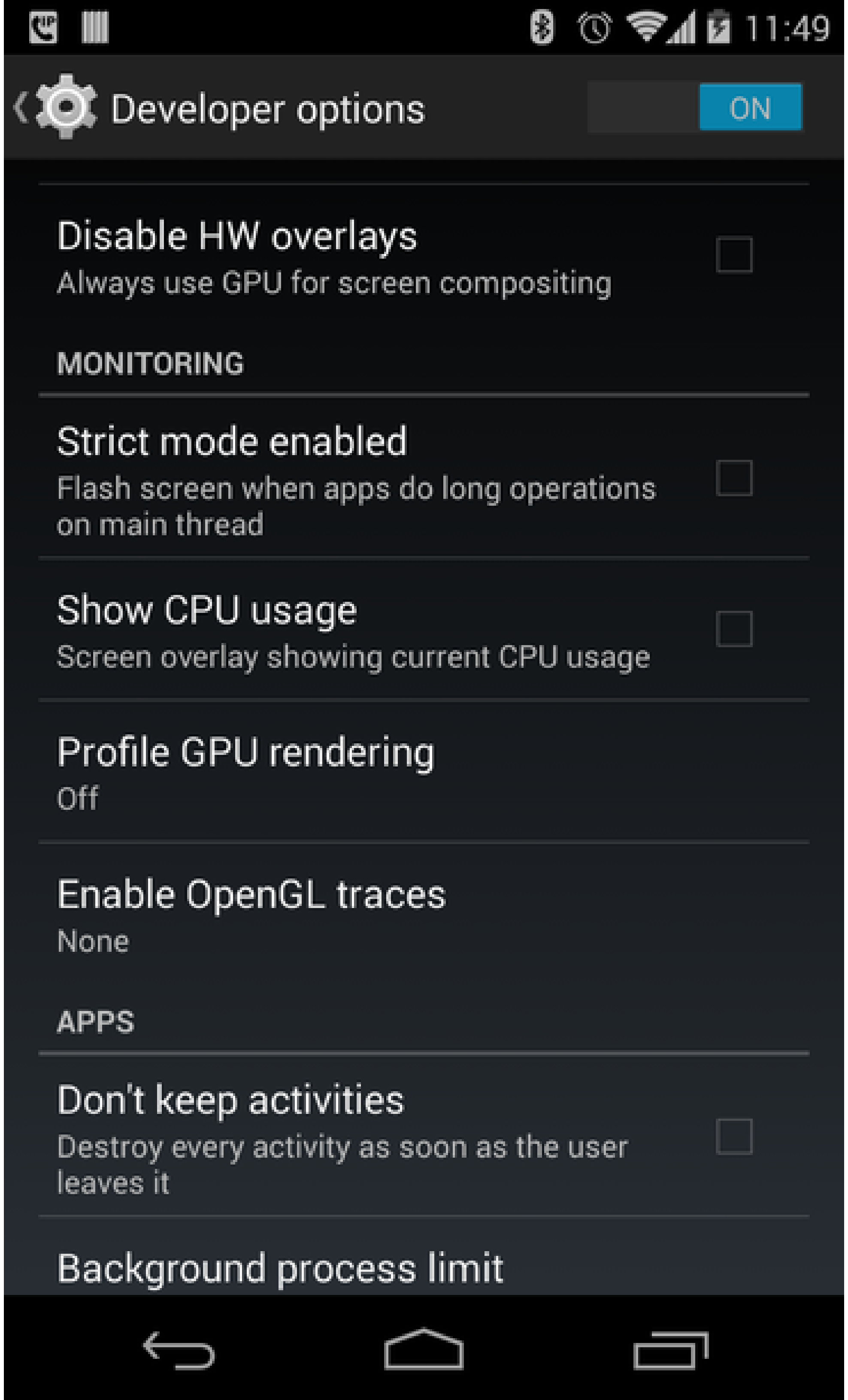
```java
@TargetApi(Build.VERSION_CODES.GINGERBREAD)
private void enableStrictMode() {
  StrictMode.setThreadPolicy(new StrictMode.ThreadPolicy.Builder().detectAll()
                                                                   .penaltyLog()
                                                                   .build());
  StrictMode.setVmPolicy(new StrictMode.VmPolicy.Builder().detectAll()
                                                          .penaltyLog()
                                                          .build());
}
```

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

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


11:49

Developer options ON

Disable HW overlays
Always use GPU for screen compositing ☐

MONITORING

Strict mode enabled
Flash screen when apps do long operations on main thread ☐

Show CPU usage
Screen overlay showing current CPU usage ☐

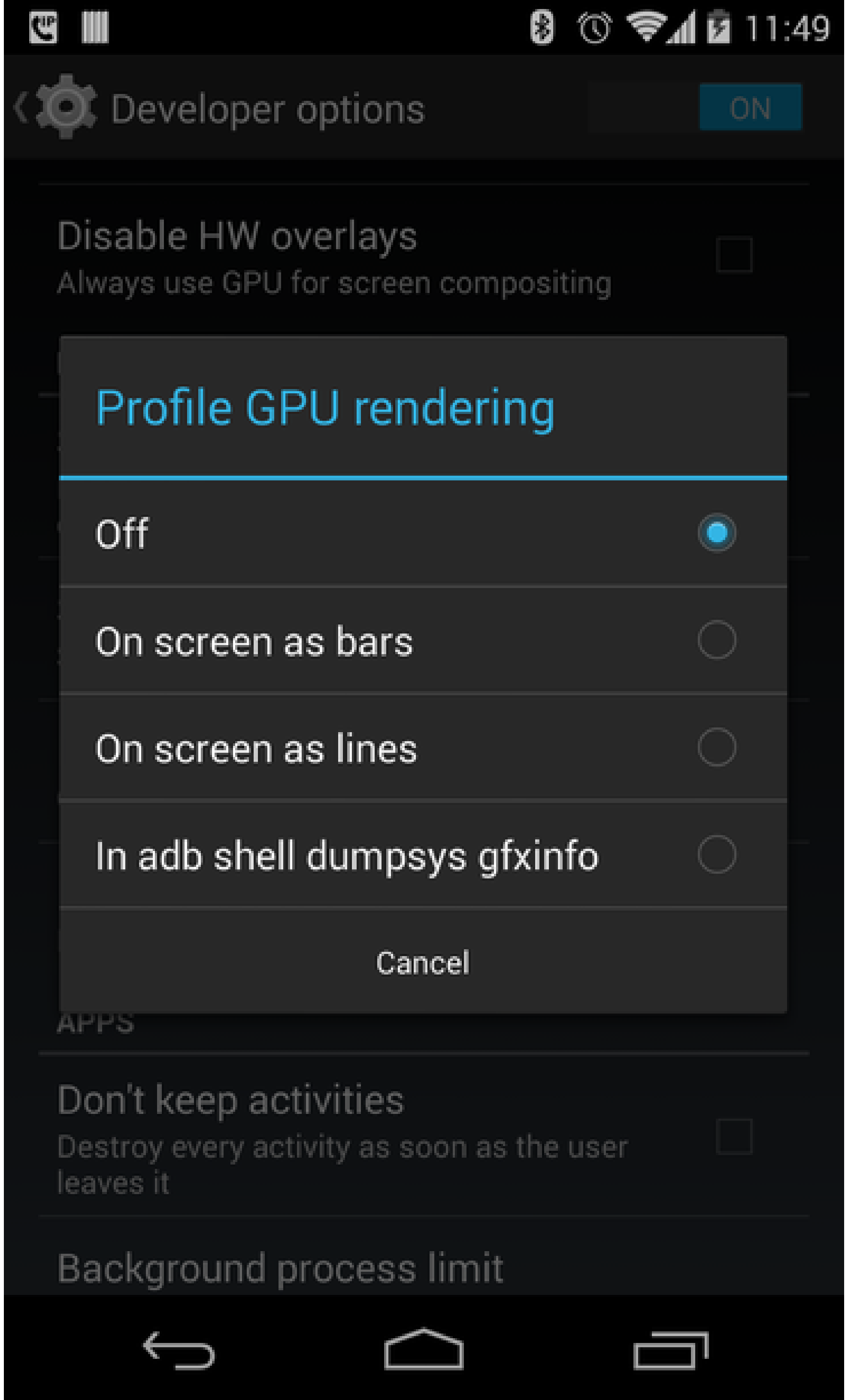
Profile GPU rendering
Off

Enable OpenGL traces
None

APPS

Don't keep activities
Destroy every activity as soon as the user leaves it ☐

Background process limit



11:49

Developer options ON

Disable HW overlays
Always use GPU for screen compositing

Profile GPU rendering

Off [✓]

On screen as bars

On screen as lines

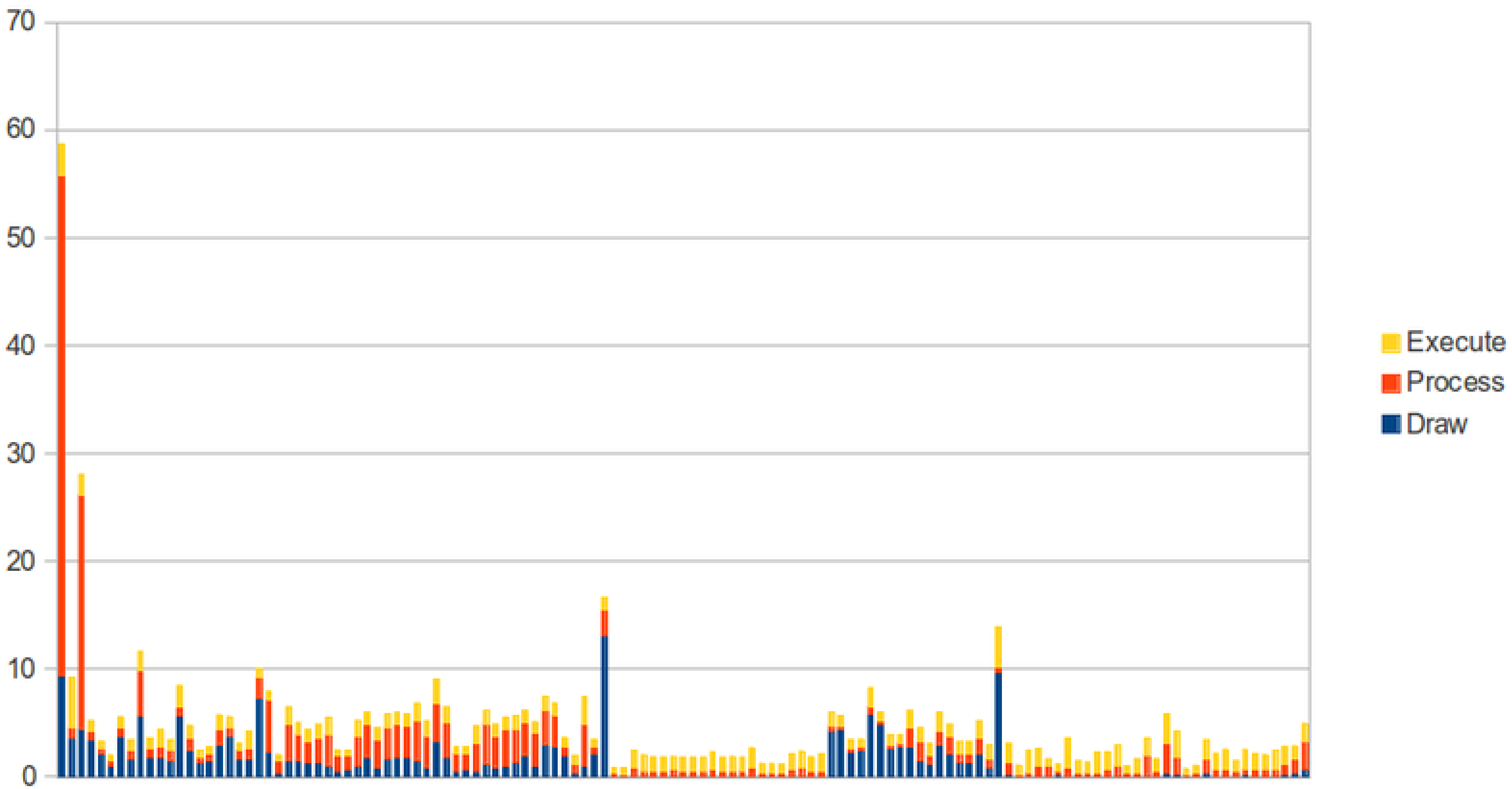
In adb shell dumpsys gfxinfo

Cancel

APPS

Don't keep activities
Destroy every activity as soon as the user leaves it

Background process limit


70
60
50
40
30
20
10
0
Execute
Process
Draw

systrace

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

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

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

Traceview

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

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

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

Hierarchy View and uiautomatorviewer

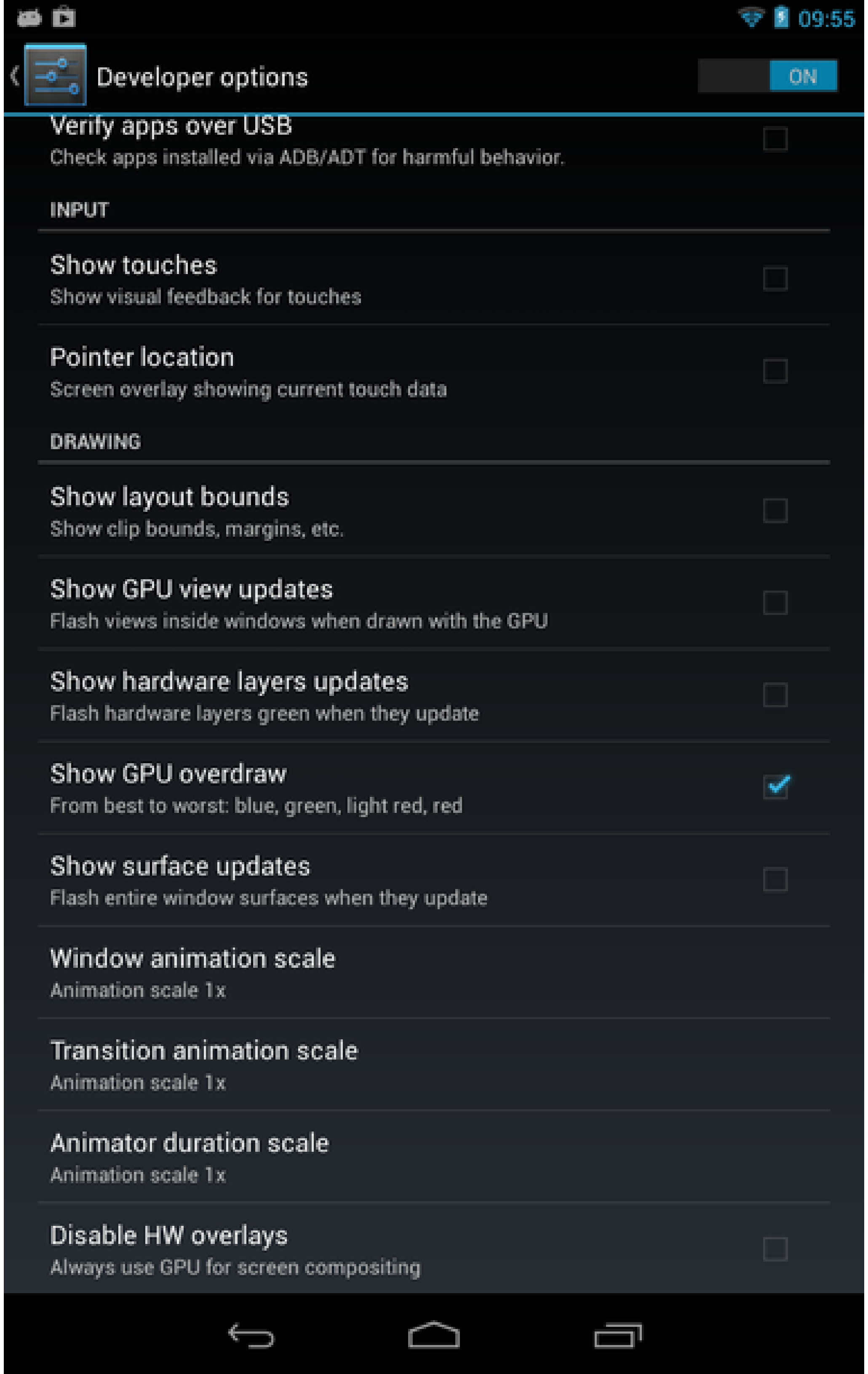
- What You Are Looking For
  - Too-deep nesting
    - Run risk of StackOverflowError at ~15 layers
    - Makes layout of containers expensive
  - Too many views
    - Each view takes time to draw, so too many views = too much time drawing
    - Also, each widget consumes minimum 1K heap space, so too many widgets = too much heap consumption
  - Overdraw

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

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)



09:55

Developer options ON

Verify apps over USB ☐
Check apps installed via ADB/ADT for harmful behavior.

INPUT

Show touches ☐
Show visual feedback for touches

Pointer location ☐
Screen overlay showing current touch data

DRAWING

Show layout bounds ☐
Show clip bounds, margins, etc.

Show GPU view updates ☐
Flash views inside windows when drawn with the GPU

Show hardware layers updates ☐
Flash hardware layers green when they update

Show GPU overdraw ✓
From best to worst: blue, green, light red, red

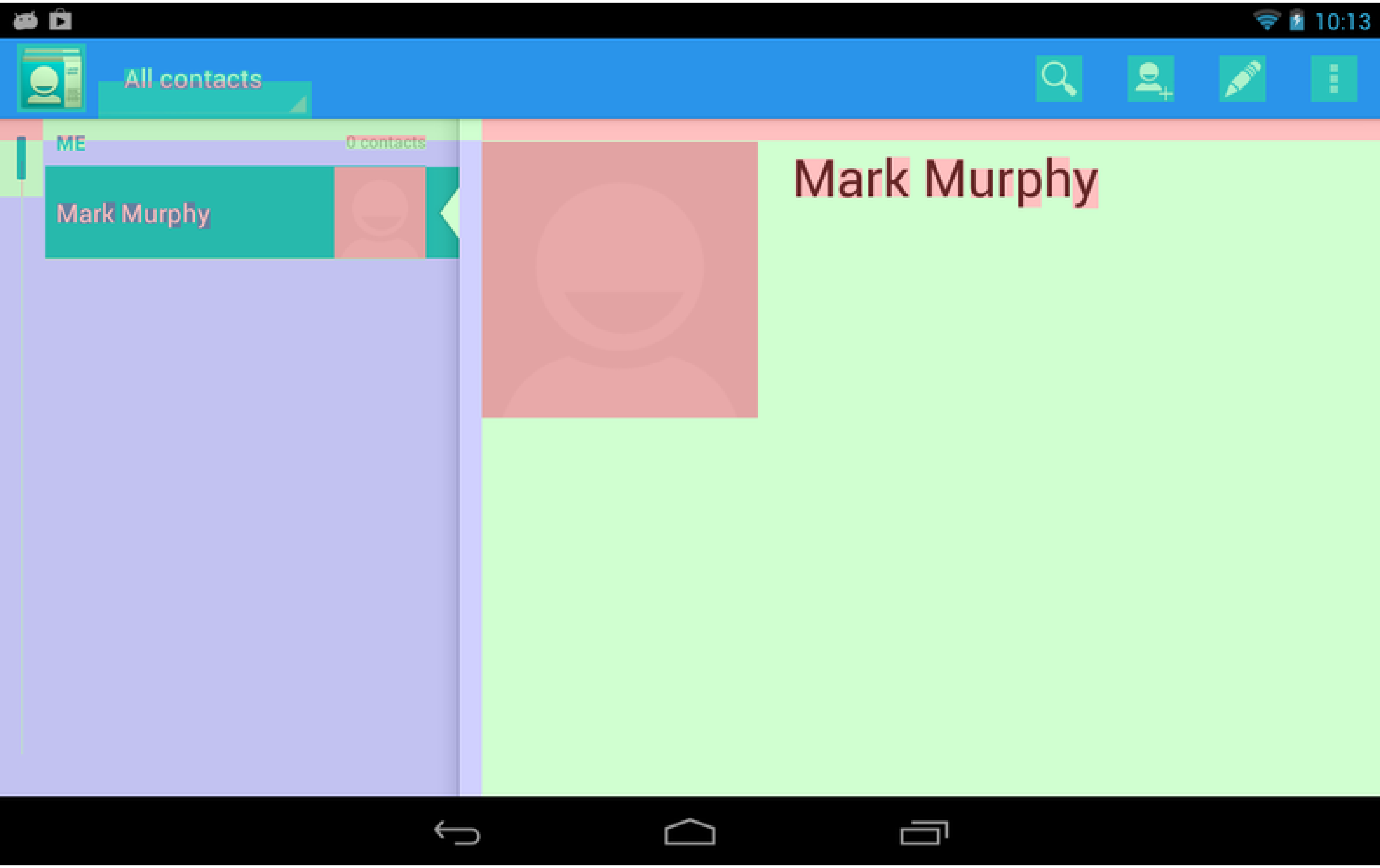
Show surface updates ☐
Flash entire window surfaces when they update

Window animation scale
Animation scale 1x

Transition animation scale
Animation scale 1x

Animator duration scale
Animation scale 1x

Disable HW overlays ☐
Always use GPU for screen compositing


10:13
All contacts
ME
0 contacts
Mark Murphy
Mark Murphy