

The Design and Implementation of a Smartphone Illuminance Meter

Devon D. Sparks

November 2012

Abstract

The proliferation of consumer smartphones has offered new opportunities for environmental sensing and mobile computation. Recent smartphone models – equipped with GPS trackers, accelerometers, multi-megapixel cameras and rich software stacks – offer the possibility of emulated specialized tool completely in software. This paper documents recent efforts to build a robust, smartphone-based illuminance meter application, and describes its prototype implementation. Though API restrictions prevent the complete development of such a prototype, its use and potential are demonstrated.

1 Introduction

Rapid advances in consumer electronics and advertising have transformed the mobile device market. Single-function mobile phones of the early-1990’s were overtaken by “feature phones”, only to be usurped by the powerful, touchscreen smartphones of the last few years. Now standard issue for 74% of Americans aged 24-35[6], smartphones have been ceaselessly recast as tour guides, news readers, desktop calculators, event planners, and sound mixers. When one machine has the ability become most others with the swipe of a finger, there are two practical issues to consider: what is the upper bound on a smartphone’s ability to replace specialized tools, and how might this be changed? This paper explores both questions in the domain of building science, using a smartphone-based light meter as its test application. Its conclusions are straightforward: that despite their power, existing smartphones are not yet fit-for-purpose as digital light meters, and that some of their failings can be fixed. It examines the limitations of current smartphone software stacks in supporting accurate illuminance calculation, and speculates on the potential use of smartphones as tools for teaching building science and architectural daylighting.

2 Review of Existing Light Meter Apps

The promise of smartphones as tools for environmental monitoring has been widely published[2][4]. Light meter applications for two leading smartphone software stacks – Google’s Android and Apple’s iOS – make use of a smartphone’s internal light sensor to estimate scene illuminance. User reviews are consistently poor, due in large part to the limitations of the phone’s ambient light sensor. Though a photodiode may be an appropriate tool for adjusting an LCD backlight, it makes a poor illuminance calculator. Of the 12 purported illuminance meters available in the Google Play and Apple Store, all depend on the accuracy of phone’s ambient light sensor. Fortunately, the large number of disappointed reviews demonstrates the clear interest in a more robust smartphone light meter, and the release of increasingly sophisticated software stacks lays fertile ground for their development.

3 Hardware Limitations of Smartphones

Methods for computing scene illuminance using mass-market digital cameras are well known[7]. Though a smartphone camera has no mechanically adjustable aperture, its sensor array can transform the number of photons impinging each sensor into a proportional voltage, perceived as a pixel brightness in the final image. Many smartphone cameras now support multi-megapixel sensors, but remain bound to software APIs that encode 24-bit image formats. One byte for each red, green and blue color channel allows for 16.7 million colors per pixel, but only 256 values per channel. In scenes with some combination of sunlighting ($10^5 cd/m^2$) and artificial lighting ($10^2 cd/m^2$), one byte per channel per pixel will almost always produce an image that is under- or overexposed. However, capturing multiple images of a static scene with varying exposure times provides an array of RGB values for each pixel, at least one of which will be properly exposed. By choosing proper exposures on a pixel-by-pixel basis, a high dynamic range (HDR) image can be generated that removes the limitations of standard, low dynamic range (LDR) imaging.

If accurately captured, each pixel in a source image can then undergo a change of basis from RGB into the CIE XYZ color space, where the CIE Y value closely matches spectral luminous efficiency of the human eye for photopic vision, as defined in ITU-R BT.709.

$$Y = 0.2162R_{linear} + 0.7152G_{linear} + 0.0722B_{linear} \quad (1)$$

Assuming an ideal, linear image sensor, each spot luminance can be normalized by dividing out its exposure time[5, p. 117]. As a practical matter, most digital cameras do not have an linear response; manufacturers often introduce gamma correction into their image processing pipelines in order to generate more pleasing images for monitors and print. Restoring this linear mapping requires recovery of the camera’s unique response curve, called the Opto-Electronic Conversion Function (OECF), and described in International Standard ISO 14524.

Thankfully, once captured and stored, the camera response curve may can be used for subsequent exposures.

One popular algorithm for recovering this mapping, due to Debevec and Malik[3], leverages the fact that by capturing different exposures of a static scene, “one is effectively sampling the camera response function at each pixel.” [5, p. 137] When the exposure values are plotted against each pixel value, Debevec and Malik use linear optimization to find a smooth curve that minimizes the mean-squared error over each of the three red, green, and blue color channels. If the dynamic range provided by the camera’s exposure parameters is too narrow, linear optimization on one or more channels will fail, and an accurate response curve won’t be found.

Response recovery on a Samsung Galaxy Nexus, a Google-endorsed smartphone running Android 4.1, proved unsuccessful for just this reason. Anywhere Software’s *Photosphere* failed to produce an accurate response curve on all of seven independent trials, each using seven input images taken with Android’s built-in camera application (one for each of the four available *capture modes*, and three additional images using the built-in *area metering* feature). Few image sets converged at all, and those that did gave polynomials of the first or second degree – less accurate than a generic sRGB response curve. Response recovery on images taken with the manual exposure settings of a Canon XXYY, in contrast, were consistently successful, implying that the lack of manual exposure control in Android’s camera parameters was stunting response recovery.

Existing Android camera parameters – like area metering, and exposure compensation – are not appropriate substitutes for manual exposure: the former for its inability to reliably alter the automatic exposure settings of the camera firmware, and the latter for its application as a post-processing step, after pixel values have been encoded. Apple’s iOS camera APIs, which provide similar features, have similar shortcomings. Unfortunately, use of a generic response curve is also ill-fated, as Reinhard, Ward et. al. note:

“Assuming an sRGB response curve is unwise, because most makers boost image contrast beyond the standard sRGB gamma to produce a livelier image. There is often some modification as well at the ends of the curves, to provide software highlights and reduce noise visibility in shadows.”

It is a surprising fact that, at the time of writing, neither Google Android nor Apple’s iOS provide official support for manual exposure control – requisite for HDR generation, response curve recovery, and accurate illuminance calculation. This restriction is well documented for the Android platform in Google Android Issue #5692, and on iOS in the *AVCaptureDevice* Class Reference. Given that illuminance calculation with mass-market digital cameras is well understood[7], it is clear this is purely a restriction of existing software stacks, and not a fundamental limitation of smartphone hardware. The limited field of view of smartphone cameras, and tight memory restrictions of their software APIs, create additional challenges.

4 Proposal for a New Light Meter

A proper smartphone light meter should embody the basic qualities of a specialized meter (small size, easy of use), but leverage the symbol processing power and network connectivity of mobile phones. To accurately measure scene illuminance, it must use a fine sensor, recover spot luminance from encoded pixel values, and give results in reasonable time (less than one-two minutes) in side-, as well as toplit, spaces. These requirements demand that illuminance calculation be converted into (at least) a three-stage pipeline: capture, merging, and calculation. Each module, along with its prototype implementation built on a Samsung Galaxy Nexus running Android 4.1, will be described in the following sections.

4.1 The Capture Pipeline

To accurately estimate scene illuminance, a smartphone light meter must have a field of view large enough to account for top and side lighting. As most smartphone cameras (including the Galaxy Nexus) have a relatively narrow field of view (50-60 degrees), multiple images must be captured to allow for panorama generation in the following stage. However, allowing the user to take any scene images he wishes is not advised, especially when the camera must remain steady between successive shots. To guide the image capture process, a new interface is needed.

In the prototype implementation, the user is presented with a "mass and spring" simulation, controlled by the phone's orientation sensors and a software-based Verlet path integrator. As the phone shifts orientation, the virtual mass stretches away from the screen's center proportional to the tilt. The virtual mass can take one of several colors, each corresponding to one on-screen target. Once in contact with a target, the phone's front-facing camera captures several lossy (i.e. JPEG encoded) images of the current scene – ideally with varying exposure times – and writes them to internal storage for later processing. The targets are positioned to maximize the chance of feature matching. As the simulation is rotationally invariant, it is assumed the user remains stationary throughout the capture exercise.

[CAPTURE IMAGE INTERFACE IMAGE]

The intent behind a mass-and-spring simulation is two-fold: first, to force the user to slow his movement, and focus his attention, so that crisp scene fragments can be captured for stitching and analysis; and second, to reduce his perception of elapsed time while background threads complete image analysis. If illuminance calculations must be computationally intensive, the best an application designer can do is make the wait an imperceptible one.

[SECOND INTERFACE IMAGE (?)]

4.2 The Merging Pipeline

Once captured, the application aligns images pairwise, storing the homographies calculated using the algorithms of BoofCV (a pure-Java implementation of OpenCV). It then attempts to collapse these images into a single mosaic with a 90° effective field of view. These images become the input for the next pipeline stage, which implements fisheye projection using a cubic environment mapper.

Generating large image mosaics on a small phone is not easy. It is computationally expensive (both in time, and space), especially on a machine with a 16-24MB memory limit per application¹. Fortunately, experiments with the prototype image stitching module were largely successful, generating complete, 90° field of view planar images in 20-30 seconds each in more than 75% of trials. With some adjustment to the Capture module's on-screen target positions, and a reimplement at the OS level, this could be improved.

[SAMPLE MOSAIC]

However, there are strong caveats to this approach. Images used to stitch a mosaic with an effective 90° field of view must also undergo HDR processing. It is unnecessary to process every pixel of every image, as most image pairs will have substantial overlap. At minimum, a lookup table between the pixels of source images and an output mosaic must be generated, so that HDR generation is efficient. Without manual exposure control, the prototype implementation had to skip HDR processing, and focus on image alignment and merging only.

Fundamental properties of this workflow that make it a poor match for smartphones. Successful feature matching of the overhead scene required 8-12 source images, pairwise homography calculation, and repeated reading from external storage to avoid Android's internal memory constraints. Homography calculation depends on the success of feature correspondence between overlapping images, a challenge if the overhead scene has few distinguishing features. In order to generate a full 180° fisheye perspective, four additional mosaics must be generated, each representing the upper half of the user's view in each of the cardinal directions. This says nothing of HDR generation, or response curve recovery. At least one research-grade mobile software stack can automate this process[1], at the cost of specific smartphone hardware, a custom operating system installation, and modified firmware.

Though each of the above obstacles can be overcome, they cannot be reduced to a few seconds of calculation. The price to an all-digital, purchase-with-a-swipe illuminance meter is time; what a fisheye lens can do at light speed, a mobile computer must do sequentially, and stochastically. A lens will always merge its inputs, but a panorama generator makes no such guarantee.

¹See the section *Displaying Bitmaps Efficiently* of the Android Training Manual for details: <http://developer.android.com/training/displaying-bitmaps/index.html>. There is some discussion that this hard limit can be overcome through use of Google's Native Development Kit (NDK), though this option was not pursued in this paper; tight resource constraints are not the only count against a smartphone-based illuminance meter.

Other efforts to perform multi-directional HDR panorama generation have either offloaded the feature correspondence calculations to back-end servers, or used one of the two image registration routines outlined in Reinhard, Ward et. al.[5, p. 122]. The faster of these routines – *the mean threshold bitmap alignment routine* – is unsuited for images taken at different view angles, making it a poor match for smartphone deployment. It is a fair criticism to claim that any smartphone application intending to bridge the gap between computation and intuition ought to give incremental results in less than 30 seconds. Designers of future smartphone-based building science tools should keep this point front-of-mind.


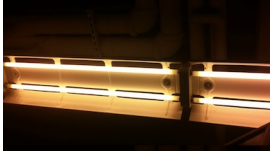


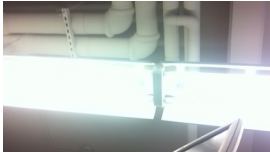

4.3 The Calculation Pipeline

Under the assumption that an HDR image mosaic can be generated from scene fragments, an implementation of an luminance integral – the central calculation in a smartphone light meter – is trivial. Once a response curve is known, integration involves a linear transformation, lookup, and Reimann sum for all pixels in the input image. In the prototype implementation, luminance integration on sample, low dynamic range images never took more than a few seconds.

While outputting a single illuminance value would fulfill all the requirements of a fit-for-purpose light meter, the high resolution display and network connectivity of smartphones offers new opportunities. With user permission, illuminance calculations could be uploaded to an online database, capable of archiving HDR images and GPS coordinates along with a qualitative description of the scene. Such an archive could become the basis of further work in the human factors associated with architectural daylighting, and improve student intuition of lighting fundamentals. Such collaborative tools have been deployed in other fields (like the SoundAroundYou.com environmental audio archive from the University of Salford), and there’s little reason to think their success could not be replicated by daylighting practioners and students.

5 Apple’s Undocumented Manual Exposure APIs

As of late 2012, the iOS and Android software stacks provide no official support for manual exposure control for the internal camera. However, beginning with iPhone 4, HDR image generation is possible within Apple’s built-in camera application. If HDR generation occurs, manual exposure must be implemented. A decompilation of the *AVCaptureDevice* class headers in iOS 6.0 reveals several *undocumented* method calls for manual exposure, including one **setExposureDuration:** that takes a single C structure with the signature $\{long\ long\ x1; int\ x2; unsigned\ int\ x3; long\ long\ x4\}$. Unfortunately, the decompiler can only provide information on argument *types* required by this struct, and can say nothing about their *meaning*. With some experimentation, the meaning and use of these undocumented methods can be recovered. To activate manual exposure control

| Captured Image | Exposure Time |
|---|---------------|
|  | 0.01 ms |
|  | 0.02 ms |
|  | 0.05 ms |
|  | 1.0 ms |
|  | 10.0 ms |
|  | 100.0 ms |

in iOS 6.0, first enable manual exposure support:

```
[avdevice setManualExposureSupportEnabled:YES];
```

This will allow use of a new exposure mode flag in **setExposure:**. To set the camera to manual exposure mode, invoke the new flag constant (3):

```
[avdevice setExposureMode: 3];
```

Finally, set the camera shutter speed by sending an Exposure struct to *avdevice* via **setExposureDuration:**. Though the author is still unclear on the exact meaning of each entry in *Exposure*, it is thought that x1, x4 are exposure times in milliseconds, x2 is f-stop, and x3 is the ISO. Varying the exposure duration as described generated the following image set:

This is, as far as the author is aware, the first time use of manual exposure control on iOS has been publicly documented. Unfortunately, such information cannot be used by eager app designers without violating Apple’s *iPhone Developer Program License Agreement*:

”3.3.1 Applications may only use Documented APIs in the manner prescribed by Apple and must not use or call any private APIs.”

While the Android licenses do not impose such restrictions, it is unclear when manual exposure control or RAW image capture support will be added to the software stack. As Google’s Dave Sparks noted on the *[android-developers]* mailing list in 2009².

On the G1, no data is returned [from the RAW picture callback] - only a null pointer. The original intent was to return an uncompressed RGB565 frame, but this proved to be impractical...Applications are restricted to a heap of 16MB. An 8MP image in RG888 will use 32MB...I’m talking about deprecating the raw picture callback that has never worked.

6 Challenges and Possibilities

The inability to control the underlying camera hardware directly from the iOS or Android software stack regrettably stall efforts to develop a light meter application suitable for design science teaching. Thankfully, this is purely a limitation of the existing software stacks, and not a conceptual or hardware-based constraint. The above workflow can be run piecewise on a traditional workstation, using a group of low dynamic range images with varied exposure settings. The short half-life of smartphones (source) also opens new opportunities for porting these workstation-based implementations (including the underlying kernel) to older smartphone models.

[Spread of manual exposure settings on Canon, with HDR image, and illuminance calculation compared to specialized meter]

Building a robust, smartphone-based light meter appears to be – for the most part – technically possible. In the worst case, the motivated smartphone user could modify his phone’s firmware, install a basic Linux installation on its drive, buy an inexpensive, mountable fisheye lens, and (assuming he’s able to find or write appropriate drivers for the camera hardware), emulate the HDR generation and illuminance calculation workflow normally performed on workstations. While *technically* possible, this approach is hardly *practical*. Smartphone apps appeal because they’re configuration-free, instantly available, and device agnostic. An application that required an external lens, top-of-the-line hardware, and

²See <http://www.mail-archive.com/android-developers@googlegroups.com/msg23290.html> for full discussion transcript.

two-minute wait time fails on all three counts, and has little to offer relative to even the most inexpensive specialized meters.

This does not rule out the possibility – and great potential – of a fit-for-purpose smartphone light meter in the near future. The greatest obstacle to adoption is the lack of manual exposure control in the iOS and Android software stack. But this is a fast-moving market, and the demonstration above reveals that at least one vendor already supports manual exposure, albeit in a private API. With a clear statement of purpose, there’s little reason to think that the major vendors could not be convinced to add manual exposure support.

Once manual exposure control is available, true HDR generation becomes possible. Some technical cleverness may be required to circumvent application memory limits, but this is a surmountable challenge. Panoramic stitching of wide-angle scenes – important in spaces with considerable side-lighting – will remain technically feasible, if unwieldy. API changes providing native support for panorama generation may improve this lot, though it remains to be seen whether it will be enough to offset the wait time required between image capture, and illuminance calculation. Though it may be a slight kludge, the use of an external, mountable fisheye lens (available for less than the price of some lunches) avoids many of the problems outlined in previous sections. If, at the highest level, the goal of a smartphone-based light meter is to develop intuition, and offer first-order accuracy of calculation, it is a boon to avoid computationally expensive image stitching at all costs, and use a basic lens. Then users, developers, and researchers can focus more on the practical implications of calculation, and less of the technical wizardry that made it possible.

References

- [1] A. Adams, E.-V. Talvala, S. H. Park, D. E. Jacobs, B. Ajdin, N. Gelfand, J. Dolson, D. Vaquero, J. Baek, M. Tico, H. P. A. Lensch, W. Matusik, K. Pulli, M. Horowitz, and M. Levoy. The frankencamera: an experimental platform for computational photography. *ACM Trans. Graph.*, 29(4):29:1–29:12, July 2010.
- [2] S. Aram, A. Troiano, and E. Pasero. Environment sensing using smartphone. In *Sensors Applications Symposium (SAS), 2012 IEEE*, pages 1–4. IEEE, 2012.
- [3] P. E. Debevec and J. Malik. Recovering high dynamic range radiance maps from photographs. *SIGGRAPH 97*, August 1997.
- [4] N. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A. Campbell. A survey of mobile phone sensing. *Communications Magazine, IEEE*, 48(9):140–150, September 2010.
- [5] E. Reinhard, G. Ward, S. Pattanaik, and P. Debevec. *High Dynamic Range Imaging: Acquisition, Display, and Image-Based Lighting (The Morgan*

Kaufmann Series in Computer Graphics). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

- [6] N. M. Resarch. Young adults and teens lead growth among smartphone owners, September 2012.
- [7] D. Wüller and H. Gabele. The usage of digital cameras as illuminance meters. *Electronic Imaging Conference*, 2007.