IEEE Computer Graphics and Applications

www.computer.org/internet computing

Experimental Platforms for Computational Photography

Marc Levoy

Vol. 30, No. 5
September/October 2010

Editors: Lawrence Rosenblum and Simon Julier

Experimental Platforms for Computational Photography

Marc Levoy
Stanford University

The principles of photography have remained largely unchanged since its invention by Joseph Nicéphore Niépce in the 1820s. A lens focuses light from the scene onto a photosensitive plate, which records this information directly to form a picture. Because this picture is a simple copy of the optical image reaching the plate, improvements in image quality have been achieved primarily by refining the optics and the recording method. These refinements have been dramatic over the past few decades, particularly with the switchover from film to digital sensors, but they've been incremental.

Computational photography challenges this view. It instead considers the image the sensor gathers to be intermediate data, and it uses computation to form the picture. Often, it requires multiple images to be captured, which are combined in some way to produce the final picture. Representative techniques include high-dynamic-range (HDR) imaging, flash/no-flash imaging, coded-aperture and coded-exposure imaging, photography under structured illumination, multiperspective and panoramic stitching, digital photomontage, all-focus imaging, and light-field imaging.

In this article, I look at the lack of experimental platforms for computational photography (that is, cameras that are programmable), and I talk about one possible solution—the Frankencamera architecture designed in my laboratory at Stanford as part of our Camera 2.0 project (http://graphics.stanford.edu/projects/camera-2.0).

A Bit about Computational Photography

The term "computational photography" has been reinvented several times over the past 20 years. Its current definition stems from a symposium I co-organized in 2005 with Frédo Durand of the Massachusetts Institute of Technology's Computer Science and Artificial Intelligence Laboratory and Richard Szeliski of Microsoft Research (http://scpv.csail.mit.edu). Since then, the field has grown enormously. Of the 439 papers submitted to Siggraph 2009, nearly 40 percent were about 2D imaging applied to photography or video, making this area larger than the traditional areas of modeling, animation, and rendering.

The field has also evolved; it now spans computer vision and applied-optics topics, some of which were active research areas before anyone applied the term computational photography to them. Since 2009, the field has had its own yearly conference with peer-reviewed papers—the International Conference on Computational Photography—and it will soon have its first monograph.[1]

Why Can't I Buy a Computational Camera?

Despite this buzz, a demon haunts the computational photography community. Besides panoramic stitching, few of these techniques have found their way into commercial cameras. Why is this so? I believe four factors are involved.

Corporate Secrecy

First, the camera industry is secretive to an extent unfamiliar to those of us who grew up in computer graphics. There's little communication between companies and universities, as there has been, for example, between Silicon Graphics, Nvidia, ATI, and US academic graphics labs. Also, aside from image compression formats (such as JPEG) and color spaces (such as sRGB), the camera industry has few open standards or scholarly publications. Some of this secrecy is driven by culture, but it's also quietly acknowledged that camera companies are violating each other's patents. If you don't disclose your technology, nobody can sue you.

While pursuing the Camera 2.0 project, my colleagues and I have run into fear-driven secrecy among not only camera manufacturers but also the companies that make camera processing chips. This secrecy makes it hard to insert open source components into the value chain because these chip makers won't reveal their hardware interfaces.

The tangle of underlying patents, some of them quite broad, also constitutes a barrier to entry for new camera companies. When a patent war erupts, as it has between Nokia, Google, and Apple in the mobile phone space, it typically ends with negotiations and cross-licensing of patents, further raising the entry barrier. Starting a new camera company isn't impossible, but profit margins are razor-thin, partly due to licensing fees. So, unless you introduce technology that lets you charge significantly more for your camera, making money is difficult.

Although published computational photography techniques might appear ready for commercialization, key steps are sometimes missing.

Hardware versus Software

Second, traditional camera manufacturers are primarily hardware, not software, companies. Of course, digital cameras contain a tremendous amount of software, but these companies treat software as a necessity, not a point of departure. So, computational photography represents a threat because its value comes largely from algorithms. Fredo Durand likes to say that "software is the next optics"—meaning that future optical systems will be hybrids of glass and image processing. Besides placing more emphasis on software, computational photography results are being published in journals and either placed in the public domain or patented by universities, who often issue nonexclusive licenses. These trends run counter to the camera industry's hardware-intensive, secrecy-based structure.

Traditional camera companies are also uncomfortable with Internet ecosystems, which typically involve multiple vendors, permit (or encourage) user-generated content, and thrive or falter on the quality of their user interfaces. It's worth noting that, as of this writing, not a single point-and-shoot or single-lens reflex (SLR) camera offers a 3G connection (although a few offer Wi-Fi). This will change soon, driven by market pressure from the increasingly powerful and connected cameras on mobile devices. Also, no traditional camera manufacturer runs a photo-sharing site with significant penetration into Euro-American markets, although several are trying (for example, Nikon's My Picturetown or Kodak's Gallery).

Branding and Conservatism

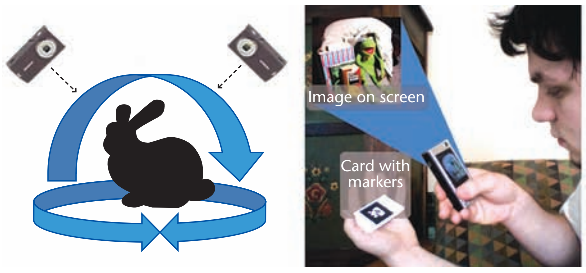
Third, traditional camera companies are inherently conservative. For example, every component in a Nikon or Canon SLR is treated by the company as reflecting on the brand's quality as a whole. So, new technologies are refined for years before they're introduced in a commercial product. This strategy yields reliable products but slow innovation.

The open source software community advocates exactly the opposite strategy—"release early, release often."[2] This strategy yields fast innovation and (eventually) high quality, as has been proven with Linux, the Apache Web server, the Thunderbird mail client, and most iPhone applications, even though the latter are not open source. When I proposed to a prominent camera manufacturer recently that they open their platform to user-generated plug-ins, they worried that if a photographer took a bad picture using a plug-in, he or she might return the camera to the store for warranty repair. Although this attitude was understandable 20 years ago, it's now antiquated; iPhone users know that if a third-party application crashes, it isn't Apple's fault.

As an example of this conservatism, although algorithms for HDR imaging have existed since the mid-1990s, except for a few two-exposure models from Fuji and Sony, no camera manufacturer offers an HDR camera. Instead, these manufacturers have been locked in a "megapixel war," leading to cameras with more pixels than most consumers need. This war is finally winding down, so companies are casting about for a new feature on which to compete. Once one of them offers such a feature (which might be HDR imaging), the rest will join in. They'll compete fiercely on that feature, to the exclusion of others that might be ready for commercialization and could be useful to consumers.

Research Gaps

Finally, while published computational photography techniques might appear ready for commercialization, key steps are sometimes missing. Although HDR imaging has a long history, the research community has never addressed the question of automatically deciding which exposures to capture—that is, metering for HDR. This omission undoubtedly arises from the lack of a programmable camera with access to a light meter. In addition, published techniques often aren't robust



(a)

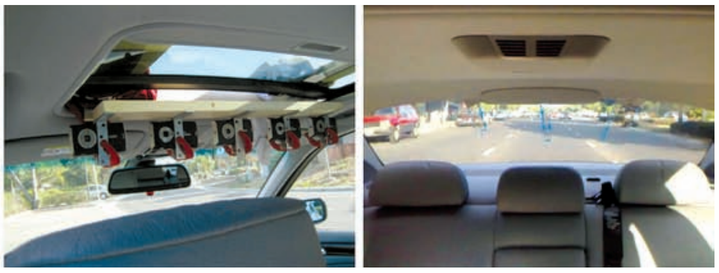

(b)

Figure 1. Two student projects with programmable cameras. (a) Abe Davis's system captured, transmitted, and displayed a 4D light field. (b) Derek Chen mounted five camera phones facing backward on his car's ceiling, creating a virtual rearview mirror with no blind spots.

enough for everyday photography. Flash/no-flash imaging is still an active research area partly because existing techniques produce visible artifacts in many common photographic situations. Again, I believe progress has been slow in these areas for lack of a portable, programmable camera.

Other Problems

Although most SLR cameras offer a software development toolkit (SDK), these SDKs give you no more control than the buttons on the camera. You can change the aperture, shutter speed, and ISO, but you can't change the metering or focusing algorithms or modify the pipeline that performs demosaicing, white balancing, denoising, sharpening, and image compression. Hackers have managed to run scripts on some cameras (see http://chdk.wikia.com/wiki/CHDK_for_Dummies), but these scripts mainly just fiddle with the user interface.

Another potential source of programmable cameras is development kits for the processing chips embedded in all cameras. But, except for Texas Instruments' OMAP (Open Multimedia Application Platform) platform, on which our cameras are based, these kits (and the hardware interfaces to the underlying chips) are either completely secret (as with Canon and Nikon) or carefully protected by nondisclosure agreements (such as Zoran and Ambarella).

Of course, you can always buy a machine vision camera (from Point Grey, Elphel, or others) and program everything yourself, but you won't enjoy trekking to Everest Base Camp tethered to a laptop.

Frankencamera: An Architecture for Programmable Cameras

My laboratory's interest in building programmable cameras grew out of computational photography courses we've offered at Stanford since 2004. In these courses, I found it frustrating that students could perform computations on photographs but couldn't perform them in a camera or use these computations' results to control the camera. I raised this Achilles' heel of computational photography in a town hall meeting at the 2005 symposium I mentioned earlier. But, at the time, it wasn't clear how to address it.

The pivotal event for us was a 2006 visit to my laboratory by Kari Pulli, a Nokia research manager. He pointed out that over the past five years, the cameras in cell phones have dramatically improved in resolution, optical quality, and photographic functionality. Moreover, camera phones offer features that dedicated cameras don't—wireless connectivity, a high-resolution display, 3D graphics, and high-quality audio. Perhaps most important, these platforms run real operating systems, which vendors have begun opening to third-party developers. With Nokia funding, Mark Horowitz (chair of Stanford's Department of Electrical Engineering), Kari, and I began developing computational photography applications for commercially available cell phones.

Early Experiments

Among our early experiments in this area was a real-time, in-camera algorithm for aligning successive frames captured by a Nokia N95 smartphone's video camera. Real-time image alignment is a low-level tool with many immediate applications, such as automated panorama capture and frame-averaged low-light photography.[3] In my spring 2008 computational photography course, we loaned N95s to every student. The resulting projects (see http://graphics.stanford.edu/courses/cs448a-08-spring) demonstrated the value of a programmable camera platform and the added value of having that platform portable and self-powered.

For example, Abe Davis developed a system for capturing, transmitting, and displaying a 4D light field (see Figure 1a). He waved a phone around a physical object to capture a light field. He then transmitted the light field to a second phone, on which you could view it by waving that phone. The second phone computed its pose in real time using its camera, displaying the appropriate slice from the light field so that the object appeared stationary behind the viewfinder.

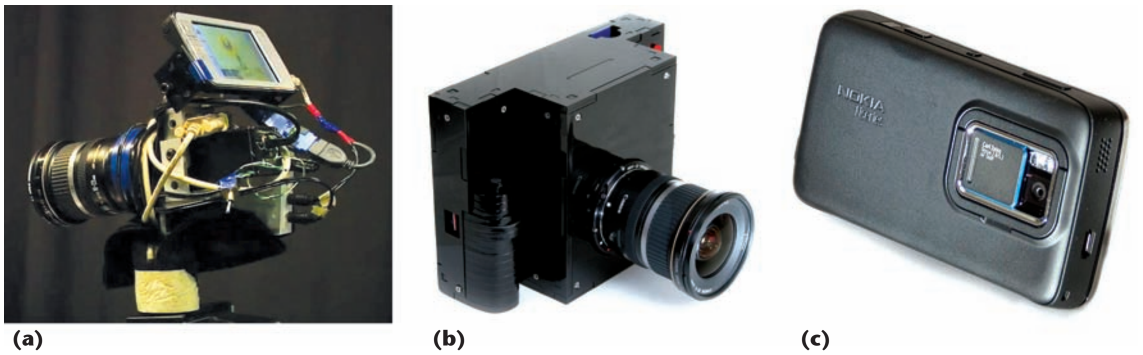

(a) (b) (c)

Figure 2. A gallery of Frankencameras. (a) Frankencamera F1, (b) Frankencamera F2, and (c) a Nokia N900 F—a retail smartphone with a custom software stack.

Derek Chen mounted five N95s facing backward on his car's ceiling (see Figure 1b). He combined the video streams to form a synthetic aperture picture that could, in principle, be displayed live on the rearview mirror. Because the aperture's baseline was large, Chen could blur out the posts supporting the car's roof, thereby providing a virtual rearview mirror with no blind spots. In the experiment shown in this figure, he blurs out wide strips of blue tape arranged on the rear window in a way that partly obscured his view. Note that you can barely see the tape in these synthetic pictures.

A Succession of Prototypes

Despite these successes, there are computational photography experiments that we can't implement on today's camera phones, because either the cameras' sensors or optics aren't good enough, the computing resources aren't powerful enough, or the APIs connecting the camera to the computing are too restrictive. To address this problem, we planned to define a programmable-camera architecture, then build a reference platform for it. Fortunately, Stanford PhD student Kayvon Fatahalian advised us to first build some cameras and see what their problems were, before trying to define an architecture. This approach seemed prudent.

So, in 2007, we began building a succession of cameras. Because we stitched together our early prototypes from parts of other devices, often dead ones, we called them "Frankencameras," after Frankenstein's monster in Mary Shelley's novel.

Figure 2 shows these prototypes, arranged in chronological order. Frankencamera F1 (see Figure 2a) employed an Elphel 353 network camera, a Nokia 800 Internet tablet as a viewfinder, and a Canon 10–22 mm EOS lens attached to a Birger EF-232 lens controller. The Elphel contains an Aptina MT9P031 sensor, the same 5-megapixel chip used in Nokia N95 cell phones. Unfortunately, separating the viewfinder from the camera forced us to burn most of the system's bandwidth feeding video to the viewfinder screen.

Frankencamera F2 (see Figure 2b) employed a Texas Instruments OMAP 3 system-on-a-chip mounted on a Mistral EVM (*e*valuation *m*odule). The Mistral has an integral touch screen display, thus solving the bandwidth problem, letting us encase the entire camera in a laser-cut plastic body. Attached to the Mistral is the Elphel 353's sensor tile and hence the same Aptina sensor.

The third camera (see Figure 2c) was a retail Nokia N900 smartphone with a Toshiba ET8EK8 5-megapixel CMOS (complementary metal-oxide semiconductor) sensor and a custom software stack.

The Frankencamera Architecture

After two years of building cameras and forcing them on the students in my computational photography courses, the key elements of our architecture began to take shape. Our most important insight was that to provide a live electronic viewfinder, digital cameras must have a pipeline of images in flight—the sensor is being configured to capture one image, while the previous image is still exposing and the one before that is being postprocessed.

However, current digital cameras have only a single state representing the camera's settings. This means that any change in settings will take effect at some unpredictable time in the future, and there's no way to know which image was captured with which settings. To capture a burst of images with different known settings, you must clear the pipeline, set up for the first image, trigger a frame, wait for it to pass through the pipeline, and set up for the next image. This incurs a latency equal to the pipeline's length for each image in the burst, as anyone who has driven an SLR from a laptop knows painfully well.

Our solution was to retain a pipeline but treat the camera as stateless. Instead, each frame in the pipeline specifies the recommended settings for producing that frame, a list of actions to synchronize to the shutter—such as if and when the flash should fire—and a frame ID. This makes it easy to program a burst of images having different exposures, ISOs, focus settings, and even different spatial resolutions or regions of interest in the field of view. This, in turn, facilitates many computational photography algorithms.

In the end, our architecture consisted of a hardware specification, a software stack based on Linux, and FCam, an API with bindings for C++. To demonstrate our architecture's viability, we implemented it on the Frankencamera F2 and our modified Nokia N900. We then programmed six applications: HDR viewfinding and capture, low-light viewfinding and capture, automated acquisition of extended-dynamic-range panoramas, foveal imaging, gyroscope-based hand-shake detection, and rephotography.[4]

As it happens, my favorite application was none of these six; it was a weekend hack in which PhD student David Jacobs mounted two Canon flash units on the Frankencamera F2 (see Figure 3a) and programmed them to obtain bizarre lighting effects on playing cards thrown into the air (see Figure 3b). What's cool about this application is that, beginning with Frankencamera F2 and FCam, Jacobs took only a few hours to assemble the hardware and write the code. It's exactly the sort of ad hoc experimentation we had hoped our architecture would enable.

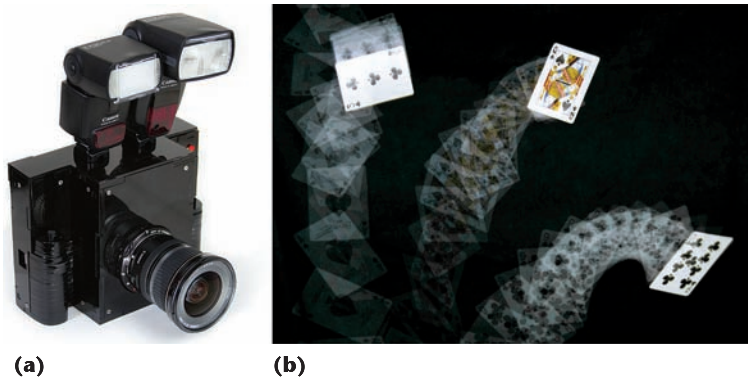


Figure 3. A weekend hack. (a) Frankencamera F2 with two flash units attached. (b) Using our FCam API, David Jacobs programmed one flash to strobe repeatedly during a one-second exposure, producing the card trails. Programming the other, more powerful flash to fire once at the end of the exposure produced the three cards' crisp images.

FCam's Pros and Cons

So, how useful is FCam? In my 2010 computational photography course, we loaned Nokia N900 Fs to every student and asked them to replace the autofocus algorithm. (The phone's camera, although small, has a movable lens.) We graded the assignment on the accuracy with which they could focus on a test scene we provided and on focusing speed in milliseconds. An assignment such as this would have been impossible before FCam. Two students submitted algorithms that were better than Nokia's; we presented these in Finland at the course's conclusion.

That said, FCam isn't perfect. While implementing it, we ran up against limitations in our reference platforms' hardware and low-level software.[4] For example, although our API supports changing spatial resolution (the total number of pixels) on every frame in a video stream, the Video for Linux 2 (V4L2) software layer on which our system is built has a pipeline with fixed-sized buffers. Changing resolutions requires flushing and resetting this pipeline, making it slower than we wanted.

Bootstrapping a World of Open Source Cameras

Defining a new architecture isn't a goal; it's only one step on a journey. Our goal is to create a community of photographer programmers who develop algorithms, applications, and hardware for computational cameras. By the time this article appears in print, we'll have published our architecture and released the source code for our implementation on the Nokia N900. We'll have also released a binary you can download to any retail N900 (without bricking the phone), thereby making its camera programmable. When we finish building Frankencamera F3 (based on the LUPA-4000 sensor), we'll make it available at cost to anyone who wants one—with luck, by December 2010.

My hope is that, once programmable cameras are widely available to the research community, camera technology can be studied at universities, published in conferences and journals, and fiddled with by hobbyists and photographers. This should foster a computational photography software industry, including plug-ins for cameras. Headed out to shoot your daughter's soccer game? Don't forget to download the new "soccer ball focus" application everybody's raving about! When I walk through the Siggraph 2020 exhibition, I expect to see a dizzying array of cameras, add-ons, software, photography asset management systems, and schools offering vocational training in computational photography.

Speaking of vocational training, I'd like to see our Frankencameras used for not only research but also education. Photography is a US$50 billion business—as large as moviemaking and video gaming combined. If we succeed in opening up the industry, we're going to need more university-trained engineers in this area. To seed this growth, my laboratory will loan a Frankencamera F3 and 10 Nokia N900s to any US university that uses them to teach a computational photography course. (This program, which will begin in 2011, is supported by a grant from the US National Science Foundation and a matching gift from Nokia.) In thinking through what it might take to triple the number of universities teaching computational photography, it's interesting to compare this nascent field to the older, larger field of computer graphics. I conjecture that the ubiquity and quality of university computer graphics curricula are due to three factors: the existence of good textbooks (such as Newman and Sproull, and Foley, van Dam, Feiner, and Hughes), experimental platforms with an open API (Silicon Graphics + OpenGL), and a high-quality publication venue (Siggraph). To accomplish this in computational photography, we'll need the same three ingredients. Good publication venues are already in place. The Frankencamera is a platform and an API, but it's only one—we need more.

For introductory textbooks, however, we're starting from zero. There are good photography books, but they don't cover its technical aspects. As a glaring example, not one how-to book on photography contains a formula for depth of field. To find one, you must turn to an optics monograph such as Rudolf Kingslake's classic *Optics in Photography* (SPIE, 1992). To fill this gap in my courses at Stanford, my students and I have written narrated Flash applets on the technical aspects of photography (see http://graphics.stanford.edu/courses/cs178/applets). However, these aren't an adequate substitute for a textbook. Somebody needs to write one.

Grand Challenges

Suppose the community succeeds in making cameras programmable and teaching a generation of students about camera technology. What should these students work on? What are the hard problems "on the dark side of the lens?"[5] Here are a few suggestions.

Shader Languages for Cameras

When students write camera applications using FCam, they typically do their image processing on the CPU, then complain about speed. It's not their fault. Although cameras and cell phones have other computing resources—GPUs, digital signal processors, and image signal processors—these aren't designed with computational photography in mind, and their hardware interfaces are often secret.

Fortunately, we believe this situation will change. Similar to the revolution occurring in graphics chips, we envision photographic image processors that use software-reconfigurable execution stages. To control such a processor, you could design a domain-specific language akin to CUDA (Compute Unified Device Architecture), OpenCL, or Adobe's PixelBender, but better suited to computational photography.

Single-Shot Computational Photography

Many papers on computational photography take this form: "Capture a burst of images varying camera setting X and combine them to produce a single image that exhibits better Y." It's a clever strategy. However, once researchers start doing this in a portable camera, they'll discover that few photographic situations lend themselves to this solution—people walk, breezes blow, foliage moves, and hands shake.

If we want to make computational photography robust, we must focus on single-frame solutions. But why must cameras record the world in discrete frames? Isn't this a holdover from film days? Using novel optics and sensor designs we can measure the light falling on the focal plane when and where we wish.

Computational Video

Although this article emphasizes still photography, the computational photography community is also interested in video. Representative techniques here include video stabilization, spatiotemporal upsampling, video summarization, and continuous versions of still-photography algorithms—for example, HDR video, light field videography, and object insertion and removal. Here, a special challenge is to ensure that the processing is temporally coherent—that is, it doesn't introduce jarring discontinuities from frame to frame.

Integration of In-Camera and In-Cloud Computing

Although high-end cameras don't yet contain 3G radios, this will change soon. Once cameras are connected to the cloud, you can imagine many applications that, for the foreseeable future, would take too long to run on the camera or burn too much battery power. A good example is video stabilization. I don't need this to run on the camera; just make sure my video is stable when it's posted to YouTube. You can also imagine using images from photo-sharing sites to help control the camera—my photo albums prove that my dog is black; why does my camera's light meter insist on making her look gray?

Progress in science and engineering is a dialogue between theory and experimentation. In this dialogue, the experimental platform's quality is paramount. We built our Camera 2.0 project on the premise that a tethered SLR isn't an adequate experimental platform for computational photography. In particular, an SLR's lack of controllability makes it hard to do innovative research, and its lack of transparency makes it hard to do definitive research. (What are the noise characteristics of a Canon 5D Mark II's image sensor? Only Canon knows.)

Michael Faraday, one of the greatest experimental scientists, built his own laboratory equipment. Faraday wasn't a fanatic; the devices available to him in the early 19th century were neither flexible enough nor accurate enough for his delicate experiments in electromagnetism. As Thomas Kuhn writes in his seminal study, *The Structure of Scientific Revolutions*, "Novelty ordinarily emerges only for the man who, knowing with precision what he should expect, is able to recognize that something has gone wrong."[6] Kuhn further hypothesizes that intellectual revolutions occur only when a scientist casts aside the tools of the dominant paradigm and begins to invent wild theories and experiment at random. The Camera 2.0 project's primary goal is to facilitate this sort of wild experimentation.

References

1. R. Raskar and J. Tumblin, *Computational Photography: Mastering New Techniques for Lenses, Lighting, and Sensors*, AK Peters, 2010.
2. E.S. Raymond, *The Cathedral and the Bazaar*, O'Reilly, 2001.
3. A. Adams, N. Gelfand, and K. Pulli, "Viewfinder Alignment," *Proc. Eurographics*, Eurographics Assoc., 2008, pp. 597–606.
4. A. Adams et al., "The Frankencamera: An Experimental Platform for Computational Photography," *ACM Trans. Graphics*, vol. 29, no. 4, 2010, article 29.
5. N. Goldberg, *Camera Technology: The Dark Side of the Lens*, Academic Press, 1992.
6. T. Kuhn, *The Structure of Scientific Revolutions*, Univ. of Chicago Press, 1962.

Marc Levoy *is a professor in Stanford University's Computer Science Department. Contact him at levoy@cs.stanford.edu.*

Contact the department editors at cga-vr@computer.org.