# Writing graphics software gets much easier

August 1 2012, by Larry Hardesty

An image undergoing basic processing steps; exposure adjustments, then noise reduction, to arrive at the final image (bottom). Graphic: Christine Daniloff

(Phys.org) -- Image-processing software is a hot commodity: Just look at Instagram, a company built around image processing that Facebook is trying to buy for a billion dollars. Image processing is also going mobile, as more and more people are sending cellphone photos directly to the Web, without transferring them to a computer first.

At the same time, digital-photo files are getting so big that, without a lot of clever software engineering, processing them would take a painfully long time on a desktop computer, let alone a cellphone. Unfortunately, the tricks that engineers use to speed up their image-processing algorithms make their code almost unreadable, and rarely reusable.

Adding a new function to an image-processing program, or modifying it to run on a different device, often requires rethinking and revising it from top to bottom.

Researchers at MIT's Computer Science and Artificial Intelligence Laboratory (CSAIL) aim to change that, with a new programming language called Halide. Not only are Halide programs easier to read, write and revise than image-processing programs written in a conventional language, but because Halide automates code-optimization procedures that would ordinarily take hours to perform by hand, they're also significantly faster.

In tests, the MIT researchers used Halide to rewrite several common image-processing algorithms whose performance had already been optimized by seasoned programmers. The Halide versions were typically about one-third as long but offered significant performance gains — two-, three-, or even six-fold speedups. In one instance, the Halide program was actually longer than the original — but the speedup was 70-fold.

Jonathan Ragan-Kelley, a graduate student in the Department of Electrical Engineering and Computer Science (EECS), and Andrew Adams, a CSAIL postdoc, led the development of Halide, and they've released the code online. At this month's Siggraph, the premier graphics conference, they'll present a paper on Halide, which they co-wrote with MIT computer science professors Saman Amarasinghe and Fredo Durand and with colleagues at Adobe and Stanford University.

## Parallel pipelines

One reason that image processing is so computationally intensive is that it generally requires a succession of discrete operations. After light strikes the sensor in a cellphone camera, the phone combs through the

image data for values that indicate malfunctioning sensor pixels and corrects them. Then it correlates the readings from pixels sensitive to different colors to deduce the actual colors of image regions. Then it does some color correction, and then some contrast adjustment, to make the image colors better correspond to what the human eye sees. At this point, the phone has done so much processing that it takes another pass through the data to clean it up.

And that's just to display the image on the phone screen. Software that does anything more complicated, like removing red eye, or softening shadows, or boosting color saturation — or making the image look like an old Polaroid photo — introduces still more layers of processing. Moreover, high-level modifications often require the software to go back and recompute prior stages in the pipeline.

In today's multicore chips, distributing different segments of the image to cores working in parallel can make image processing more efficient. But the way parallel processing is usually done, after each step in the image-processing pipeline, the cores would send the results of their computations back to main memory. Because data transfer is much slower than computation, this can eat up all the performance gains offered by parallelization.

So software engineers try to keep the individual cores busy for as long as possible before they have to ship their results to memory. That means that the cores have to execute several steps in the processing pipeline on their separate chunks of data without aggregating their results. Keeping track of all the dependencies between pixels being processed on separate cores is what makes the code for efficient image processors so complicated. Moreover, the trade-offs between the number of cores, the processing power of the cores, the amount of local memory available to each core, and the time it takes to move data off-core varies from machine to machine, so a program optimized for one device may offer

no speed advantages on a different one.

## Divide and conquer

Halide doesn't spare the programmer from thinking about how to parallelize efficiently on particular machines, but it splits that problem off from the description of the image-processing algorithms. A Halide program has two sections: one for the algorithms, and one for the processing "schedule." The schedule can specify the size and shape of the image chunks that each core needs to process at each step in the pipeline, and it can specify data dependencies — for instance, that steps being executed on particular cores will need access to the results of previous steps on different cores. Once the schedule is drawn up, however, Halide handles all the accounting automatically.

A programmer who wants to export a program to a different machine just changes the schedule, not the algorithm description. A programmer who wants to add a new processing step to the pipeline just plugs in a description of the new procedure, without having to modify the existing ones. (A new step in the pipeline will require a corresponding specification in the schedule, however.)

"When you have the idea that you might want to parallelize something a certain way or use stages a certain way, when writing that manually, it's really hard to express that idea correctly," Ragan-Kelley says. "If you have a new optimization idea that you want to apply, chances are you're going to spend three days debugging it because you've broken it in the process. With this, you change one line that expresses that idea, and it synthesizes the correct thing."

Although Halide programs are simpler to write and to read than ordinary image-processing programs, because the scheduling is handled automatically, they still frequently offer performance gains over even

the most carefully hand-engineered code. Moreover, Halide code is so easy to modify that programmers could simply experiment with half-baked ideas to see if they improve performance.

“You can just flail around and try different things at random, and you’ll often find something really good,” Adams says. “Only much later, when you’ve thought about it very hard, will you figure out why it’s good.”

**More information:** The Halide project site: people.csail.mit.edu/jrk/halide12/

Provided by Massachusetts Institute of Technology