Jonathan Ragan-Kelley - Teaching Statement

I value teaching most for selfish reasons: I find it uniquely powerful for helping me continually distill ideas and improve the communication skills which are essential to high impact research and central to the role of academics in science. I have also always found it especially satisfying to help clarify someone's understanding of a complex idea, and to see them take it on as their own. I think it is always possible to extract essential patterns and structure from complex ideas, helping students learn far more than a single example.

Much of computer science can be understood as design problems: balancing competing goals, often by applying or deriving from abstract design patterns. I focus on teaching this way by distilling and abstracting as much as possible to key concepts and patterns, and revisiting them through different solutions at different scales. And just as abstraction of implementation is among the most fundamental skills of programming, I ultimately strive to teach students to abstract patterns themselves, recognize them in other problems and solutions, and apply them in new areas.

Computer architecture, for example, has the opportunity to deeply inform not just basic hardware design, but the design of software, and students' use of and expectations for computing systems now and in the future. For example, patterns for finding and exploiting parallelism, locality, and coherence, and minimizing or hiding latency, appear at many scales to address similar challenges. In graphics pipeline architectures, similar patterns have emerged to manage irregular data, parallelism, locality, large working sets, and other design problems, across real-time and offline renderers, in both hardware and pure software implementations, over four decades. This makes clear just how much "architecture" is about fundamental design tradeoffs, regardless of whether they are addressed in hardware or software.

In the same way, I strive to teach patterns over specific methods whenever possible. I want to avoid leaving students with a book of black magic incantations but no confidence or understanding to dig deeper and build their own. In computer science education, programming presents a particular challenge: we want to teach students practical skills and tools, which too often requires them to memorize their way around a huge amount of accidental complexity in real languages and tools before they can even begin to practice algorithmic thinking. Educational languages, libraries, tools, and environments are an area where I think computer science as a field can keep improving, and where I have spent much of my energy when preparing courses. Giving students rapid and tangible ways of interacting with, viewing, and sharing their work is especially powerful.

Better supporting programming in education also offers a rich opportunity for research, both in HCI and tools, and in programming language design[1]. The same conceptual improvements in tools and languages which can make programming more quickly accessible to a broader range of students will also make professional programmers more productive, and even broaden the range of people who can usefully program to automate their work[2].

Tools are also important for improving our work as teachers. I am excited by the opportunities of the "flipped classroom" model, and enjoyed working with Frédo Durand on tools to make creating and collaboratively developing online lectures much easier and more powerful.

Teaching Experience and Interests

Throughout my academic career, I have appreciated the opportunity teaching offers to learn and better distill my own understanding of the key ideas with which I work.

This fall, I designed and co-taught a new course on domain-specific languages for graphics, imaging, and beyond[3]. In it, we brought students from the basics of compilers and code generation, through IR design, to a survey of real systems in a half dozen different domains. Students' final projects built new languages for domains from machine learning to hardware design, of which some will likely result in published research.

In 2011, I was the teaching assistant (for 60 students, from sophomores to PhD candidates) for Frédo Durand's computational photography course. Based on that experience, I helped completely redesign the programming assignments and environment for the following year, and have learned immensely from teaching the students Halide. For the past two years, I have also taught half- and full-day courses on Halide to researchers and practitioners at industry conferences including SIGGRAPH and CVPR. I have also taught courses on the architecture of the graphics pipeline to hundreds of practitioners and academics at several recent SIGGRAPH conferences, and to students in MIT's introductory graphics course most years I have been in graduate school. In 2010, taught a 12 hour graduate seminar course on graphics architectures and micropolygon research at Lund University in Sweden.

I would enjoy the opportunity to teach and help design curriculum for a range of subjects, including:

- domain-specific languages
- computer graphics
- computer architecture
- graphics and imaging architectures
- computational photography and imaging
- parallel systems, programming
- compilers
- introductory computer science/programming

[1] Learnable Programming. B. Victor, 2012. http://worrydream.com/LearnableProgramming/.
[2] Eve. C. Granger, et al., 2015. https://github.com/witheve/Eve.
[3] http://cs448h.stanford.edu