Jonathan Ragan-Kelley - Research Statement

My research develops new programming languages, compilers, systems, architectures, and algorithms that enable graphics, imaging, and vision applications orders of magnitude more rich than any we have today. For example, replacing video with real-time 4D light fields, synthesizing trillions of voxels to rapidly print large 3D objects with micron level details, rendering interactive worlds indistinguishable from reality, and building machines which pervasively understand the visual world all require orders of magnitude more computational power than current systems offer. With current technology, lightfield VR capture requires a rack of computers to process the video feed from a single camera rig, while modern convolutional neural networks require a 250 Watt GPU to classify 0.1 megapixel images at video rates. Orders of magnitude more performance and energy efficiency will let us leverage cheap, high data rate cameras for everything from human-computer interaction to search, and move powerful rendering and image processing into every device we touch, where it will transform how we see, think, remember, and are entertained, while globally mining the rivers of visual information created every day will let computers, and ultimately humans, understand and analyze the visual world in ways never before possible.

Superficially, the data parallelism inherent in graphics and imaging computations would seem to make them easy to scale on future hardware. Real graphics and imaging computations, however, have complex dependencies, and are limited by locality (the distance over which data has to move, e.g., from nearby caches or far away main memory) and synchronization. Increasingly, the cost of communication—both within chips and over networks—dominates computation and power consumption, and limits the gains realized from shrinking transistors. Relative to performing an ALU operation on 32 bits of data, moving that data a few millimeters across a chip requires an order of magnitude more energy, moving it to or from off-chip DRAM requires four orders of magnitude more, and sending it over a cellular radio or across a data center at least six orders of magnitude more.

The efficiency and performance of an application are determined not only by the algorithm and hardware architecture, but critically also by the organization of computations and data. For algorithms with the same complexity—even the exact same set of arithmetic operations and data—executing on the same hardware, the order and granularity of execution and placement of data can easily change performance by an order of magnitude because of locality and parallelism. Especially in multi-stage algorithms like image processing pipelines, this difference comes not from the optimization of individual stages in isolation, but from the global interleaving of computations and data. For example, computing each stage completely before the next—even with the optimal inner loop, spread over thousands of threads on a GPU—destroys producer-consumer locality, constantly pushing data to and from main memory. As a result, libraries of even the best optimized subroutines do not compose into efficient pipelines. This effect only grows larger as the hierarchy of communication deepens across an entire data center.

I believe the key to turning ongoing exponential growth in hardware into orders of magnitude more performance and efficiency in real applications is to apply domain knowledge to model and understand the dependencies both in data structures and in the tasks linking them. The dependencies between computations and data define the topology of the problem. In graphics and imaging applications, the scale of both tasks and data are large, and the dependencies among them complex, making locality and parallelism essential but challenging to exploit. Modeling and understanding the dependencies allows us to:

- restructure the computations and data—especially task and data granularity, and data movement—to simultaneously maximize locality and coherence, and minimize redundant computation;
- map them to efficient hardware primitives, and design new hardware architectures to exploit the structure of the resulting computation;
- and evolve the design of algorithms to best exploit these opportunities for efficiency.

Using this philosophy, I will create systems and algorithms to scale imaging, vision, and 3D graphics; abstract the patterns exploited by these systems to build more efficient general purpose architectures; and improve the efficiency of data-intensive applications in other domains, such as machine learning and scientific computation.

Selected Past Research

I have worked on systems, programming languages, compilers, architectures, and algorithms for applications throughout graphics, imaging, and vision. My research has focussed primarily on two major types of visual computing systems: high-performance image processing, for computational photography and vision; and 3D graphics, from offline rendering, to the real-time graphics pipeline, to 3D printing.

High-performance image processing systems

Image processing pipelines are simultaneously wide and deep, combining the challenges of stencil computations and stream programs. They have abundant data parallelism, but whole pipelines consist of long sequences of different opera-

tions which individually have low arithmetic intensity (the ratio of computation performed to data read from prior stages and written to later stages). Because of this structure, simply executing each stage in isolation before moving on to the next stage is extremely inefficient, limited by the bandwidth to read and write intermediate data flowing through the pipeline. For each stage, an implementation has to balance competing demands for locality, parallelism, and reusing rather than redundantly recomputing shared values. The performance difference between an implementation with even the best inner loop optimizations in each stage, and a pipeline globally restructured for efficient computation and communication, is often an order of magnitude or more.

The Halide image processing language [4, 5]: I designed a new programming language for high performance image processing, called Halide, and a compiler that is able to globally reorganize computation to maximize locality and control the granularity of parallelism. The key to Halide's design is strictly separating the intrinsic *algorithm* defining a pipeline from choices about the order and granularity of allocation, execution, and communication for each stage. I call these choices its *schedule*. I defined a systematic model of the tradeoff space fundamental to stencil pipelines, and a schedule representation that describes concrete points in this space for each stage in an image processing pipeline. This model was the essential domain specific design which lets terse, portable, and composable algorithms compile to state-of-the-art implementations on many different architectures. It naturally and compactly expresses the choices fundamental to optimizing image processing pipelines, providing a powerful representation for both human programmers and automatic algorithms to express and explore the choice space, balancing and orchestrating locality, parallelism, and selective redundant recomputation at all scales of the resulting pipeline. The compiler synthesizes high performance implementations from a Halide algorithm and a schedule, often delivering performance many times faster than the best prior hand-tuned C, assembly, and GPU implementations. In recent work, we also showed how even higher-level scheduling transforms could outperform traditionally optimized implementations of common *recursive filters* by an order of magnitude [6].

The productivity offered by Halide has sparked rapid adoption in industry, including at Google, Facebook/Instagram, Adobe, and Qualcomm. At Google, at least 60 engineers have committed production Halide code including the Android HDR+ pipeline, tens of thousands of lines of Halide code process every image uploaded to Google Photos, and a stock Android 6 phone ships with over 100 Halide pipelines. Qualcomm has released a backend for their latest Snapdragon's image processor, using Halide as the language for one of the first programmable image signal processors (ISPs). Halide has also been explored for everything from survey astronomy to exascale simulation code. In all, production Halide code runs on hundreds of millions of phones, tens of thousands of servers, and has processed billions of images.

Helium: lifting stencil kernels from stripped x86 binaries to Halide code [7]: While domain-specific languages (DSLs) like Halide provide enormous performance and productivity benefits when writing new code, but legacy code—even legacy binaries—is also important to re-optimize for new architectures. We built a tool to automatically re-optimize ("rejuvenate") existing binaries by reconstructing high-level Halide code for their core kernels. The original optimized binary code is nearly impossible to analyze statically. Instead, we rely on dynamic traces to regenerate the kernels, reconstructing buffer structure and abstracting from a forest of concrete dependency trees to symbolic stencils suitable for high-level code generation. Helium can handle highly optimized, complex stencil kernels with input-dependent control flow. We lifted kernels from several commercial binaries, including seven kernels from Adobe Photoshop where re-optimization provided a 75% performance improvement on modern x86. Others have shown similar results automatically lifting Fortran source into Halide [8].

Darkroom: compiling high-level image processing code into hardware pipelines [9]: By restricting programs to a subset of the Halide model, Darkroom automatically schedules image processing pipelines for optimal buffering using a pattern called *line buffering*. Then, given an optimally scheduled pipeline, it automatically synthesizes hardware designs for ASICs or FPGAs. We demonstrated simulated ASIC performance similar to hand-designed commercial ISPs (gigapixels/sec in 100s of mW), and 100 megapixel/sec throughput in a few watts on real FPGAs, all synthesized from a few hundred lines of high-level DSL code. More recently, we are broadening this to support all of Halide, and hybrid FGPA+CPU execution.

Transform Recipes for efficient cloud photo enhancement [10]: Offloading image processing to the cloud is often proposed as a solution to the limited resources of mobile devices, but this overlooks the time and energy cost of transferring the input and output: including transfer, most common image processing is *slower and more expensive* to offload to a remote server than run locally on a phone. We aimed to shift this balance in the case of image enhancements that preserve the overall content of an image. Our key insight is that, in this case, the server can compute and transmit a description of the *transformation* from input to output, which we call a *transform recipe*. This is a kind of domain-specific compression. At equivalent quality, recipes are much smaller than JPEG images, and they can also be computed from highly compressed *inputs*, which significantly reduces the data sent both to and from the server. On real mobile phones, we showed transform recipe-based pipelines running 2-4x faster and using 2-7x less energy than either local or naive cloud computation.

3D graphics and rendering systems

The Simit simulation and sparse linear algebra language [11]: Using existing programming tools, high-performance simulation code is extremely difficult to write. Simulations naturally describe the behavior of an entire physical system using the language of linear algebra, but must also manipulate individual geometric elements, which are best represented using linked data structures like meshes. Translating between the linked data structures and linear algebra comes at significant cost, both to the programmer and the machine. To address this, collaborators and I recently built Simit, a new language for physical simulations that lets the programmer view the system simultaneously both as a linked data structure in the form of a hypergraph, and as a set of global vectors, matrices and tensors depending on what is convenient at any given time. The compiler unifies these two abstractions to generate efficient in-place computation on the graph. Simit is easy to use (typically shorter than a MATLAB program), high-performance (comparable to hand-optimized simulations), and portable (Simit programs can be compiled for GPUs with no change to the program, delivering 5-25x speedups over optimized CPU code).

The OpenFab programmable pipeline for multi-material fabrication [12] tries to imagine what should be "the RenderMan of 3D printing". State-of-the-art 3D printing hardware is capable of mixing many materials at 1000 DPI resolution. This poses an enormous computational challenge as the resulting data is far too large to directly precompute and store (a single cubic foot at this resolution requires at least 10^{11} voxels, and terabytes of storage). At the same time, it is challenging for users to specify continuous multi-material mixtures at high resolution. Current printer software is designed to process polygon meshes with a single discrete material per object. We proposed an entirely different way to drive multi-material 3D printers, as a programmable synthesis pipeline akin to the rendering pipeline. Instead of a static mesh per piece of material, OpenFab describes a procedural method to synthesize the final voxels of material at full printer resolution, on demand. This provides efficient, static storage and communication, resolution independence, and it decouples material definition from geometry. A DSL and pipeline for 3D printing make it much easier for users to specify many types of procedurally printed output than they could by writing standalone programs for every different material or fabrication application. This system drives the state-of-the-art printer developed by the MIT computational fabrication group [13].

The Lightspeed Lighting Preview System [14] aimed to radically improve the productivity of artists designing lighting and materials for 3D scenes in special effects and feature animation. Historically, artists waited hours for the final frame renderer to recompute an image from scratch every time they changed a few lighting parameters. My system accelerates this process by three to four orders of magnitude using knowledge that only certain shading computations tend to change. Given the parameters the user wants to edit, it automatically analyzes and splits the computation into a caching step, which precomputes as much as possible of the rendering, and a preview step—translated to the GPU—which recomputes the final image interactively. We introduced new adaptive data structures for caching and recomputing both shading and visibility information orders of magnitude more efficiently than regular encodings. The system has been deployed in production at Industrial Light & Magic since 2007. It has been used by dozens of artists on almost 20 films, and was a finalist for an Academy of Motion Picture Arts and Sciences technical achievement award.

Decoupled sampling for real-time graphics pipelines [15]: Drawing on key ideas and data structures from Lightspeed, I proposed a fundamental extension to the architecture of the real-time graphics pipeline. By exploiting coherence in shading computations, it reduced the overall cost of rendering by many times in the presence of previously unattainable effects including realistic motion and defocus blur. This directly led to a large volume of ongoing work in adaptive shading for real-time rendering.

Ongoing and Future Research Directions

The changing pressures brought by Moore's Law, making data movement and parallelism dominate performance and efficiency where once we worried about instructions and inner loops, require that we break through the traditional barriers of abstraction and reinvent our systems from hardware to algorithms. At the same time, the shift to internet-scale computing to analyze and understand the world is driving a transformation of computing systems from individual machines to large, distributed, and heterogeneous platforms. My research applies domain knowledge to build systems which capture, model, and exploit the global structure of computations and data in real applications. I will develop:

- programming language and system design, coupled with program analysis, to automatically model and extract the structure of computations;
- code generation and runtime scheduling systems to mechanize the transformation of computations and data to maximize locality, coherence, and parallelism, while minimizing wasted work;
- architectural primitives and specialized hardware to exploit these characteristics for efficient execution;
- algorithms to expose structure more amenable to exploitation at all these levels, and designed with explicit consideration of the rapidly diverging costs of computation and data movement.

By working together for end-to-end efficiency, these techniques provide orders of magnitude increases in computational power and energy efficiency, from embedded devices to data centers, while at the same time providing better abstractions which dramatically increase programmer productivity.

The Visual Computing Database

The past decade of progress in vision has been characterized by the growth of ever-larger image data sets. Popular curated academic collections contain tens of millions of images [16], while in the wider world Facebook and YouTube ingest at least 100 million photos and 100 billion new video frames per day, 250 million security cameras were installed worldwide in 2014 alone, and the Large Synoptic Survey Telescope will soon capture terapixels per night for 10 years.

These data sources provide an amazing opportunity to analyze and understand the world, for everything from computer vision, to urban planning, to large-scale science, but actually working with all this visual data is extremely challenging. Just storing or browsing 100M images is prohibitively difficult. Finding a selection of relevant images for a given task is daunting, and given a selection, real analyses often require the Cartesian product of this already large set with many different transformations of the raw data. Processing these collections is even more challenging. For example, a common fundamental task is to build a feature descriptor for every image. Given an optimized feature extraction kernel that took just 100 milliseconds per image at all scales of interest, it would take over 24 hours per million images every time you update the feature descriptor (e.g., reconfigure or re-tune a deep network). As a result, real implementations need to be highly optimized, *and* scaled to run on large clusters or data centers.

To address this, I recently started a major project, which I expect to pursue for several years, in collaboration with Kayvon Fatahalian and Pat Hanrahan [17]. Our goal is to make it much easier to work with internet-scale image, video, and visual data. We want to bridge the gap between the 1 hour MATLAB sketch of a visual analysis concept and the 1 month distributed cluster implementation necessary to run experiments on realistic data sets. To do this, we are building a *database system* for visual data. I see our query system as a domain specific language, the database of visual data as its core abstract data structure, and the task of query planning and scheduling a domain-specific compilation problem. A key challenge is the too-large-to-store scale of intermediate results, and the orders of magnitude increase in computation per element relative to traditional analytics systems, shifting the balance from I/O to compute-bound. This presents many challenges at the intersection of image processing, database systems, and high-performance computing, including: defining programming abstractions for expressing complex queries on visual data collections; enabling the database system to efficiently manage derived datasets often many times larger than the input data, but expensive to recompute; and designing the system to efficiently integrate non-relational distributed computation (e.g., for machine learning) with visual and relational operations. Looking forward, there are numerous opportunities beyond the initial system, from architecting data center hardware to most efficiently support large-scale visual analysis, to supporting streaming queries, which run directly on the global firehose of visual data, or even at the edges of the network directly on data sources, to natively integrating sampling for accuracy-performance tradeoffs in interactive analysis, and human computation which is still critical to many visual analysis tasks.

A DSL for real-time optimization on visual data

Many problems in graphics and vision are naturally described as optimizations over visual data, like images and meshes, with objectives based on visual operators, like convolution, noise models, patch similarity, or deformation. The mathematical description of these optimizations is often extremely concise, but their realization in code is anything but, especially when hand-optimized for performance. Hand-written solvers specialized to specific graphics applications can exploit domain-specific knowledge of the key operations (e.g., stencils, convolution) and structure of unknowns (e.g., pixels, voxels, meshes) to achieve 2-3 orders of magnitude higher performance than general-purpose solvers using opaque matrix representations for operations and data [18], but they require painstaking manual derivation of differential forms of the objective, and complex fusion into thousand lines of custom CUDA code.

To address this, I am building a DSL and compiler for several important classes of nonlinear optimization on visual data. In preliminary results, we have found that, by writing just a few lines of energy function code directly encoding the objective, we can compile custom solvers which solve real graphics and vision problems on multi-million variable unknowns in real-time, outperforming the state-of-the-art Ceres solver by 100× while employing the same Gauss-Newton solver method [19]. But this is just the first step in a longer-term agenda: optimization is a ubiquitous and powerful tool, but it takes different forms in different domains. I believe there is a large opportunity for compilation and specialization of optimizations to specific problems to radically improve performance and scalability, across many types of objectives, many structures of unknowns, and many different solver methods. This is a collaboration with Steven Boyd's group, which I hope will lead to entirely new backends for CVX and other systems.

High Performance Image Processing and Vision Systems

There remains enormous opportunity to improve image processing and vision by orders of magnitude, from algorithms down to hardware. I will build new systems for high performance image processing and vision. In initial work, I have found that a simple model of locality in convolutional neural networks allows an exhaustive search of blocking choices to synthesize Halide code which outperforms state-of-the-art (CAFFE/MKL) implementations by several times; combined with a memory access energy model, this allows co-optimization of network implementation and hardware cache allocation to create specialized CNN processors more efficient than any previously shown. Further, modeling sparse, irregular, and streaming data structures will open up a wide range of higher-level vision, computational photography, and video algorithms. The challenge will be to model the more complex dependencies within these data structures, and their interaction with the data parallel grids at the core of image processing.

I will work to make fixed-function image processing pipelines (ISPs) programmable in much the way the graphics pipeline became 10 years ago. Mobile computational photography and vision applications stand to improve performance and energy efficiency by orders of magnitude over current implementations on general-purpose processors, but it will require careful co-design of the architectural abstractions, programming models, and compilers to satisfy demands for increased generality and programmability without sacrificing the extreme efficiency of current fixed-function hardware.

Finally, we will need new algorithms to best exploit these systems. In a world where machines no longer simply get faster for existing code, and optimized subroutines can't compose into optimized systems, it will be increasingly valuable to apply systems and architecture thinking to fundamentally improving the efficiency of vision and image processing pipelines. I will design new algorithms based on what these high performance systems are able to do well. A likely theme will be more regular algorithms winning out over "smarter," more complex alternatives at scale. For example, segmentation algorithms derived from simple belief propagation may yield better systems than any heroic attempts to adaptively parallelize highly sequential algorithms like graph cuts. An algorithm like PatchMatch [20] can be reinterpreted as a high-level randomized search strategy rather than a specific incremental algorithm, which can be implemented with many different communication and synchronization strategies to balance sequential information propagation with locality and parallelism. I am interested not just in individual algorithms, but in the design of whole applications like 3D reconstruction pipelines, where granularity, the dependence of data, and the order of computation dominate scalability to internet-scale datasets [21].

Throughput Architectures

I will work to abstract patterns essential to efficient execution on domain-specific architectures, like GPUs and ISPs, to make general purpose throughput architectures more efficient for many more applications. When programmed as general-purpose machines in CUDA or OpenCL, existing GPUs expose only a few of the primitives essential to making the 3D graphics pipeline efficient. Other critical features—like buffering intermediate data between stages for producer-consumer locality, fine-grained dynamic work creation to manage irregular data rates, and fast synchronization on spatial data structures—are only available inside the fixed-function graphics pipeline. Similarly, ISPs and DSPs exploit fixed communication patterns to minimize expensive data movement through static portions of image processing and other pipelines. I believe it is possible to extract composable primitives from which systems can exploit these same opportunities for efficiency in general programs, such that Direct3D and ISP pipelines, physical simulation, and computer vision could all become software applications atop general programmable architectures, while retaining much of the efficiency of specialized processors. These abstractions must be co-designed with compilers and programming models, in much the way OpenGL and Direct3D have mutually evolved with GPUs.

Domain-Specific Language Foundations

Finally, if domain-specific programming systems are key to delivering productivity, performance, and efficiency on future hardware, I want to address two fundamental challenges to their spread. First, enabling computations from multiple domain-specific systems to effectively compose with each other is essential. High-performance DSLs are primarily built around abstract collection data structures specialized to a given domain (images, matrices, power-law graphs, etc.), and optimized by controlling the concrete implementation of those structures and the aggregate operations over them. In addition to generalizing the concepts of a scheduling algebra like that in Halide, I think the work of data representation synthesis points towards building these specialized primitives on a common foundation [22], using the shared language of relational data models and physical decompositions to connect them. Second, I will work to abstract the patterns of programming model and domain-specific compiler design into reusable tools, to make it much easier to build systems that analyze and exploit domain-specific application structure to schedule computation and synthesize code on both specialized and general purpose processors.

Opportunities for Collaboration

I strongly believe in the power of domain-specific system building, but it requires deep collaboration with users and domain experts. Outside graphics and vision, I have recently been fortunate to work with astronomers building the image processing pipelines for the Large Synoptic Survey Telescope. I am excited to continue collaborating with experts in vision, graphics, and other domains from physical simulation to machine learning, to develop higher-performance systems based on domain-specific abstractions, and new techniques enabled by the resulting radical increases in computational capacity and energy efficiency. For example, applications of vision from health, to image recognition and search, to HCI (e.g., Microsoft Kinect) use recognition databases to understand and act on visual input. Scaling these training databases to orders of magnitude more examples, features, classification categories, and resolution presents enormous challenges in managing the granularity and scheduling of computation and communication. Ultimately, all of the data-intensive applications demanding far more computational power than we have today present complex systems challenges like these. Designing the radically different end-to-end systems required to deliver orders of magnitude more computational power and efficiency will be a large undertaking, and I look for collaborators at every level, from application and domain experts, to researchers in compilers, programming languages, systems, and architecture.

References

1. A hierarchical volumetric shadow algorithm for single scattering. I. Baran, J. Chen, J. Ragan-Kelley, F. Durand, J. Lehtinen. *ACM Trans. Graph.* 29(6), 2010.
2. Portable performance on heterogeneous architectures. P. M. Phothilimthana, J. Ansel, J. Ragan-Kelley, S. Amarasinghe. *SIGARCH Computer Architecture News (Proc. ASPLOS)* 41(1), 2013.
3. OpenTuner: An extensible framework for program autotuning. J. Ansel, S. Kamil, K. Veeramachaneni, J. Ragan-Kelley, J. Bosboom, O'ReillyU., S. Amarasinghe. *Proc. PACT*, 2014.
4. Decoupling algorithms from schedules for easy optimization of image processing pipelines. J. Ragan-Kelley, A. Adams, S. Paris, M. Levoy, S. Amarasinghe, F. Durand. *ACM Trans. Graph.* 31(4), 2012.
5. Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. J. Ragan-Kelley, C. Barnes, A. Adams, S. Paris, F. Durand, S. Amarasinghe. *Proc. ACM PLDI*, 2013.
6. Compiling high performance recursive filters. G. Chaurasia, J. Ragan-Kelley, S. Paris, G. Drettakis, F. Durand. *Proc. High-Performance Graphics*, 2015.
7. Helium: Lifting high-performance stencil kernels from stripped x86 binaries to Halide DSL code. C. Mendis, J. Bosboom, K. Wu, S. Kamil, J. Ragan-Kelley, S. Paris, Q. Zhao, S. Amarasinghe. *Proc. ACM PLDI*, 2015.
8. Bridging the gap between general-purpose and domain-specific compilers with synthesis. A. Cheung, S. Kamil, A. Solar-Lezama. *Summit oN Advances in Programming Languages (SNAPL)*, 2015.
9. Darkroom: Compiling high-level image processing code into hardware pipelines. J. Hegarty, J. Brunhaver, Z. DeVito, J. Ragan-Kelley, N. Cohen, S. Bell, A. Vasilyev, M. Horowitz, P. Hanrahan. *ACM Trans. Graph.* 33(4), 2014.
10. Transform recipes for efficient cloud photo enhancement. M. Gharbi, Y. Shih, G. Chaurasia, J. Ragan-Kelley, S. Paris, F. Durand. *ACM Trans. Graph.* 34(6), 2015.
11. Simit: A language for physical simulation. F. Kjølstad, S. Kamil, J. Ragan-Kelley, D. Levin, S. Sueda, D. Chen, E. Vouga, D. Kaufman, G. Kanwar, W. Matusik, S. Amarasinghe. *ACM Trans. Graph. (to appear)*
12. OpenFab: A programmable pipeline for multi-material fabrication. K. Vidimče, S.-P. Wang, J. Ragan-Kelley, W. Matusik. *ACM Trans. Graph.* 32(4), 2013.
13. MultiFab: A machine vision assisted platform for multi-material 3D printing. P. Sitthi-Amorn, J. Ramos, Y. Wang, J. Kwan, J. Lan, W. Wang, W. Matusik. *ACM Trans. Graph.* 34(3), 2015.
14. The Lightspeed automatic interactive lighting preview system. J. Ragan-Kelley, C. Kilpatrick, B. Smith, D. Epps, P. Green, C. Hery, F. Durand. *ACM Trans. Graph.* 26(3), 2007.
15. Decoupled sampling for graphics pipelines. J. Ragan-Kelley, J. Lehtinen, J. Chen, M. Doggett, F. Durand. *ACM Trans. Graph.* 30(3), 2011.
16. ImageNet: A large-scale hierarchical image database. J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei. *Proc. of CVPR*, 2009.
17. The Visual Computing Database: A platform for visual data processing and analysis at internet scale (NSF 1539069). K. Fatahalian, P. Hanrahan, J. Ragan-Kelley. http://people.csail.mit.edu/jrk/visualdb_shareable.pdf
18. Real-time shading-based refinement for consumer depth cameras. C. Wu, M. Zollhöfer, M. Nießner, M. Stamminger, S. Izadi, C. Theobalt. *ACM Trans. Graph.* 33(6), 2014.
19. Ceres solver. S. Agarwal, K. Mierle, Others.
20. PatchMatch: a randomized correspondence algorithm for structural image editing. C. Barnes, E. Shechtman, A. Finkelstein, D. B. Goldman. *ACM Trans. Graph.* 28(3), 2009.
21. Building rome in a day. S. Agarwal, N. Snavely, I. Simon, S. M. Seitz, R. Szeliski. *Proc. of ICCV*, 2009.
22. Data representation synthesis. P. Hawkins, A. Aiken, K. Fisher, M. Rinard, M. Sagiv. *Proc. ACM PLDI*, 2011.