

Streaming Multigrid for Gradient-Domain Operations on Large Images

Michael Kazhdan
Johns Hopkins University

Hugues Hoppe
Microsoft Research

Abstract

We introduce a new tool to solve the large linear systems arising from gradient-domain image processing. Specifically, we develop a streaming multigrid solver, which needs just two sequential passes over out-of-core data. This fast solution is enabled by a combination of three techniques: (1) use of second-order finite elements (rather than traditional finite differences) to reach sufficient accuracy in a single V-cycle, (2) temporally blocked relaxation, and (3) multi-level streaming to pipeline the restriction and prolongation phases into single streaming passes. A key contribution is the extension of the B-spline finite-element method to be compatible with the forward-difference gradient representation commonly used with images. Our streaming solver is also efficient for in-memory images, due to its fast convergence and excellent cache behavior. Remarkably, it can outperform spatially adaptive solvers that exploit application-specific knowledge. We demonstrate seamless stitching and tone-mapping of gigapixel images in about an hour on a notebook PC.

Keywords: out-of-core multigrid solver, B-spline finite elements, Poisson equation, gigapixel images, multi-level streaming.

1 Introduction

Many recent image processing techniques operate in the gradient domain. They extract gradient fields from one or more images, process the data to construct a desired gradient field, and solve for a new image whose pixel differences best fit the desired gradients.

For example, lighting is removed from an image by zeroing small gradients [Horn 1974], or by selecting the median of gradients from multiple exposures [Weiss 2001]. A high dynamic range (HDR) image is tone-mapped by adaptively attenuating luminance gradients [Fattal et al. 2002]. Overlapping images are stitched seamlessly by merging their gradients [Pérez et al. 2003; Agarwala et al. 2004; Levin et al. 2004]. Shadows are removed by zeroing large luminance gradients in regions of constant chromaticity [Finlayson et al. 2002]. Undesirable reflections are removed in flash and ambient image pairs [Agrawal et al. 2005]. Photographic tone management is improved using gradient constraints [Bae et al. 2006]. Novel painterly effects are possible with interactive gradient-domain modeling [McCann and Pollard 2008]. In all these applications, the final image is recovered from the processed gradient by solving a Poisson equation, which is discretized to form a sparse linear system.

Recent work has begun to address the processing of large (e.g. gigapixel) images [Kopf et al. 2007b]. In this case the resulting linear systems, and often the images themselves, are too large to fit in main memory. Consequently, direct solution techniques like Cholesky factorization become impractical, and traditional relaxation techniques like conjugate gradients and multigrid are ineffi-

cient because they require many iterations over out-of-core data. As reviewed in Section 2, the associated Poisson problem can be made tractable for some specific applications through adaptive discretization, and has also been addressed using heuristic approximations.

Our contribution We introduce a general, efficient, accurate solver for Poisson equations over large images. We combine multigrid computation with a data streaming framework, to allow out-of-core processing on gigapixel images. Our scheme maintains small moving windows of data in memory (and cache) by sequentially advancing through the out-of-core data. Because the solver computation outpaces disk I/O, the speed bottleneck is the number of streaming passes through the image data. We are able to obtain sufficient accuracy in just 2 passes, even on gigapixel images. Three key components enable this efficient solution:

- Whereas prior image solvers use finite differences, we discretize the Poisson equation with *second-order finite elements*, allowing us to reach sufficiently low error in a single multigrid V-cycle.
- Within a grid level, *temporally blocked relaxation* evaluates several Gauss-Seidel relaxations as a single streaming operation.
- We interleave these multiresolution relaxations using *multi-level streaming* so that the restriction and prolongation phases of a V-cycle can each be pipelined into a single streaming pass.

Compared to finite differences, second-order finite elements require larger stencils for relaxation, restriction, and prolongation. Though these larger stencils increase computational load, they improve the solver convergence rate significantly. On most images, a second-order solver with a single V-cycle finds a solution with maximum error markedly less than $1/256$ of the pixel value range, i.e. well within the requisite precision for 8-bit/channel images. We show that additional V-cycles further decrease error at a steady rate, and moreover each such cycle requires just one extra streaming pass.

Since forward-difference representations of gradient fields are commonly used as constraints in image processing, it is essential that our system support them. Although finite differences and finite elements provide alternate interpretations of image derivatives, we show that the use of B-spline elements lets us correctly solve for images subject to finite-difference constraints.

In this paper we consider instances of the Poisson equation with unconstrained (more precisely, Neumann) boundary conditions, which supports many gradient-domain techniques. The Poisson equation corresponds to an elliptical partial differential equation with spatially constant coefficients. In Section 10 we discuss future extensions for more general boundary conditions or PDEs.

While our main motivation is the processing of huge images, we find that the streaming multigrid approach is also beneficial for smaller in-memory images due to its fast convergence and excellent cache behavior. On a single CPU core, we are able to solve a 3-channel, 16-megapixel gradient-domain problem with an rms error on the order of 10^{-5} in 15 seconds. The ongoing transition to many-core architectures will likely favor methods with a high ratio of local computation to memory bandwidth. In effect, memory access may become the next I/O bottleneck. Because our streaming scheme offers many opportunities for computational parallelism on local data, with few accesses to global memory, it is well suited for future scalability.

2 Related work

Due to the numerous applications of the Poisson equation, many solution techniques have been explored. The fast Fourier transform is an elegant scheme, but has $O(n \log n)$ time complexity. A direct solution can also be obtained by sparse matrix factorization, but the resulting factors may require significant temporary memory. Iterative solvers like Gauss-Seidel and conjugate gradients are memory-efficient but usually require many iterations over the data. The number of iterations can be reduced using either multiresolution preconditioners [Gortler and Cohen 1995; Szeliski 2006] or multigrid solvers [Brandt 1977; Briggs et al. 2000]. Such techniques have also been implemented efficiently on the GPU [Bolz et al. 2003; Goodnight et al. 2003; Göddeke et al. 2008].

Most previous work has focused on Poisson systems that fit in memory. Toledo [1999] presents an excellent survey of out-of-core algorithms for large linear systems. Most algorithms assume that while the system matrix may be stored on disk, the solution vector itself (i.e. the image in our case) still fits in memory. In general, it was previously thought that advanced iterative techniques, including multigrid, are difficult to schedule out-of-core.

One approach for out-of-core images is to solve the problem on a coarser-resolution grid and then upsample the resulting approximation [Kopf et al. 2007a]. However, maintaining sharp features involves heuristics that may not always be robust.

In some cases where the solution is known to have low frequency almost everywhere, the problem size can be reduced by adaptively partitioning the domain. For image stitching, Agarwala [2007] exploits the fact that if an initial guess is generated by copying pixels from the input images, the residual has low frequency away from the image seams and the Poisson equation can be solved over a quadtree adapted to the seams. Such adaptive partitions have also been used to solve the Poisson equation in the context of fluid flow simulation [Losasso et al. 2004] and surface reconstruction [Kazhdan et al. 2006].

Our method addresses the general case where (1) the Poisson equation must be solved accurately everywhere, (2) the problem size cannot be reduced, and (3) no initial solution guess is available.

Bolitho et al. [2007] introduce multi-level streaming for an out-of-core Poisson solution. Their method, which implements a cascadic (prolongation-only) solver, has adequate accuracy for surface reconstruction but not for image processing. In this work, we show that the multi-level streaming idea can be integrated into a finite-element multigrid solver that performs multiple Gauss-Seidel iterations at each resolution and implements a complete V-cycle in only two streaming passes. We demonstrate that the resulting solution is accurate enough for gradient-domain image processing.

3 Gradient-domain problem as Poisson solution

In the continuous setting, we seek an image $U(x, y)$ on a domain $\Omega = [0, 1]^2$ whose gradient is closest to a desired gradient field $\vec{G}(x, y)$, i.e. to find U minimizing $\|\nabla U - \vec{G}\|$. Using the normal equation, this minimization is equivalent to solving the Poisson equation $\Delta U = F$, where $\Delta = \nabla \cdot \nabla$ is the Laplacian operator and $F = \nabla \cdot \vec{G}$ is the divergence of the desired gradient.

Of course, an image is typically represented by a discrete $N \times N$ grid u of pixel values $u_{i,j}$. One of the challenges in setting up the Poisson equation in the context of image processing is that the Poisson equation is a continuous beast requiring the computation of derivatives while images are inherently discrete. There are two ways to address this. The first is to discretize derivatives, resulting

in the traditional finite-difference approach, often used in image processing (Section 4). The second is to treat images as continuous functions, resulting in the finite-element approach pursued here (Section 5).

4 Review of finite-difference multigrid

Both the image gradient ∇U and desired gradient \vec{G} are commonly expressed as forward differences of adjacent pixels values, i.e. $\nabla u_{i,j} = (u_{i+1,j} - u_{i,j}, u_{i,j+1} - u_{i,j})$ and similarly for the forward differences $\vec{g}_{i,j} = (g_{i,j}^x, g_{i,j}^y)$. With such a discretization, one seeks to minimize $\|\nabla u - \vec{g}\|$, which is equivalent to solving a sparse linear system $Lu = f$. Each row of the matrix L corresponds to the five-point Laplacian stencil with weight -4 at the current pixel and weight 1 at the four adjacent pixels. And, the Laplacian constraint vector f has entries defined in terms of backward differences $f_{i,j} = g_{i,j}^x - g_{i-1,j}^x + g_{i,j}^y - g_{i,j-1}^y$. For simplicity we use u (and similarly f) to denote both an $N \times N$ array of pixels $\{u_{i,j}\}$ and a vector $(u_0 \dots u_{N^2-1})^T$ of the same pixels in raster order.

To avoid constraining boundary values in the Poisson equation, the definition of ∇u and the construction of f are modified at domain boundaries to assume trivial Neumann conditions, i.e. zero cross-boundary derivatives. The resulting matrix L does not have full rank, since the solution is only determined up to an additive constant. However, a solution always exists because the construction of f using the divergence operator guarantees compatibility.

A simple approach for solving the linear system $Lu = f$ is to apply repeated iterations of Gauss-Seidel relaxation. In each iteration, the image u is updated by successively modifying each pixel u_i while holding the other pixels constant:

$$u_i = \left(f_i - \sum_{j \neq i} L_{i,j} u_j \right) / L_{i,i}.$$

Because the Laplacian operator L has local 2D support, these relaxation updates only require access to local data.

However, a limitation of Gauss-Seidel relaxation is that it converges slowly on the low-frequency components of the solution. Multigrid is introduced to overcome this [Briggs et al. 2000]. The single system $Lu = f$ is replaced by a multiresolution set of systems $\{L^l u^l = f^l\}$, where $u^l \in \mathbb{R}^{N_l^2}$ with $N_l = 2^l$. Restriction and prolongation linear operators are defined to transition between levels:

$$R_{l+1}^l : \mathbb{R}^{N_{l+1}^2} \rightarrow \mathbb{R}^{N_l^2} \quad \text{and} \quad P_l^{l+1} : \mathbb{R}^{N_l^2} \rightarrow \mathbb{R}^{N_{l+1}^2},$$

typically using local weighted averaging and bilinear interpolation.

The restriction and prolongation operators are combined to generate the standard V-cycle (Figure 1), consisting of a fine-to-coarse restriction phase followed by a coarse-to-fine prolongation phase:

- **Restriction:** Assuming an initial guess u^l for the solution at level l , relaxation is performed to obtain an approximate solution u_R^l . Then the *residual* $r^l = f^l - L^l u_R^l$ is restricted, giving the right-hand side for a coarser-grid correction problem:

$$f^{l-1} = R_{l-1}^l r^l.$$

This process is repeated with level $l-1$ replacing level l and setting the initial guess $u^{l-1} = 0$.

- **Base Solution:** At a sufficiently coarse level l_{\min} , the linear system $L^{l_{\min}} u^{l_{\min}} = f^{l_{\min}}$ is solved directly.

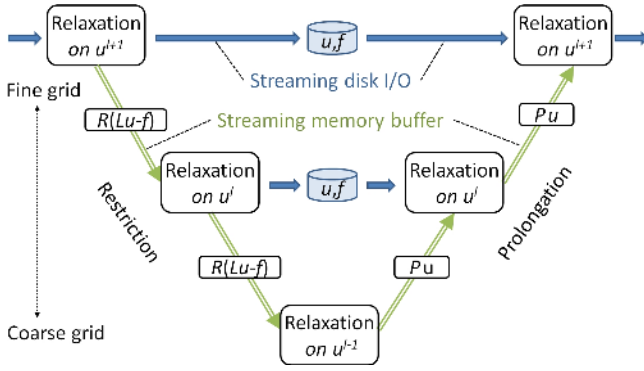


Figure 1: Standard multigrid V-cycle, consisting of a restriction phase followed by a prolongation phase. This diagram also shows the data flow in our streaming solver described later in Section 6.

- *Prolongation*: Assuming a correction solution u_p^{l-1} at level $l-1$, an initial guess is obtained at level l by adding the prolongation of that solution to the solution obtained in the restriction phase:

$$u^l = P_{l-1}^l u_p^{l-1} + u_R^l.$$

Then, relaxation is performed to obtain the solution u_p^l , and the process is repeated with level $l+1$ replacing level l .

5 Our finite-element multigrid approach

We represent an image as a continuous function using a finite-element approach, with individual pixel values acting as coefficients of continuous basis functions. We have chosen to use B-splines as they are differentiable functions with minimal local support, and satisfy essential nesting conditions. Specifically, we interpret pixel values as coefficients of tensor-product B-spline basis functions $B(x, y)$ centered at the pixel positions. As discussed later in Section 7, we find that the choice of second-order (quadratic) B-splines gives the best performance tradeoff for gradient-domain image processing.

For simplicity we begin our description of the approach in 1D. In this setting the discrete image u defines the continuous function

$$U(x) \equiv \sum_i u_i B_i(x),$$

where $B_i(x)$ is the B-spline basis $B(x)$ translated to the i -th pixel. B-spline functions of *even* degree are nested according to a dual subdivision structure, i.e. $B_i(x) = B(Nx - i - 0.5)$, $i \in \{0 \dots N-1\}$ are centered at different positions on different levels, whereas B-splines of *odd* degree are nested according to a primal subdivision structure, i.e. $B_i(x) = B(Nx - i)$, $i \in \{0 \dots N\}$.

5.1 Representing the Poisson equation

The B-spline finite elements define a finite-dimensional vector space, $\mathcal{B} = \text{Span}\{B_i(x)\}$, over which we can solve the Poisson equation. Interpreting an image as a continuous function lets us exactly compute its Laplacian, $\Delta U(x) \equiv \sum_i u_i \Delta B_i(x)$. However, there are still two difficulties in discretizing the Poisson equation $\Delta U = F$. First, the divergence $F = \nabla \cdot \vec{G}$ of the desired gradient may not reside in the spanning space \mathcal{B} . Second, even though the B-spline functions $B_i(x)$ are in \mathcal{B} , their Laplacians are not. (The derivative of a B-spline is the difference of two lower-degree B-splines, which lie in a different space.)

These difficulties are addressed using the Galerkin method (e.g. [Fletcher 1984]). We reformulate the Poisson equation to solve for the image U with the property that the *projection* of its Laplacian onto \mathcal{B} is equal to the *projection* of F onto \mathcal{B} . Equivalently, the image $U(x)$ must satisfy the matrix form

$$\langle \Delta U, B_j \rangle = \langle F, B_j \rangle \quad \text{for all } 0 \leq j < N,$$

where $\langle \cdot, \cdot \rangle$ denotes the integral of the product of two functions over the domain Ω .

Thus, if we form

- L as the $N \times N$ matrix with $L_{i,j} = \langle \Delta B_i(x), B_j(x) \rangle$, and
- f as the vector with $f_j = \langle F(x), B_j(x) \rangle$,

solving the Poisson equation reduces to solving $Lu = f$.

An alternate derivation is to apply the Galerkin method to the optimization $\min_U \|\nabla U - \vec{G}\|$ using $-\nabla B_j$ as test functions. This results in the set of linear equations

$$\langle \nabla U, -\nabla B_j \rangle = \langle \vec{G}, -\nabla B_j \rangle \quad \text{for all } 0 \leq j < N,$$

where $\langle \cdot, \cdot \rangle$ is also used to denote the integral of the pointwise dot product of two vector fields over Ω . In the presence of trivial Neumann boundary conditions, the two linear systems can be shown to be identical using the Gauss divergence theorem. As a result, matrix L can be expressed as $L_{i,j} = \langle \nabla B_i(x), -\nabla B_j(x) \rangle$, which only requires that the basis function be once-differentiable.

The linear system has similar structure to that obtained from finite differences. However, the matrix L is less sparse. For B-spline basis functions of order n , each row of L represents a stencil with $2n+1$ nonzero entries in 1D, or $(2n+1)^2$ nonzero entries in 2D.

5.2 Fitting forward-difference gradient constraints

To solve for the image $U(x, y)$ whose gradient is closest to a desired gradient field $\vec{G}(x, y)$ we must solve the associated normal equation:

$$Lu = f \quad \text{where} \quad f_j = \langle \vec{G}, -\nabla B_j \rangle.$$

However, in most image processing applications we are not given a continuous field \vec{G} , but rather a discrete set of values $\vec{g}_{i,j}$ representing forward differences of pixel values. So again we face the problem of interpreting discrete representations as continuous ones.

Although there are many ways to interpret \vec{g} as a continuous gradient field, the correct definition must conform to the forward-difference representation. Specifically, in the case that \vec{g} is the gradient of an (unknown) image v :

$$\vec{g}_{i,j} = (v_{i+1,j} - v_{i,j}, v_{i,j+1} - v_{i,j}),$$

our definition of \vec{G} should equal the gradient of the *equivalent* continuous image, i.e. $\vec{G} = \nabla V$ where $V = \sum v_{i,j} B_{i,j}$.

A crucial property for deriving a conforming continuous interpretation is that the derivative of a B-spline of degree n can be expressed as the difference of two shifted B-splines of degree $n-1$ (Figure 2):

$$\frac{d}{dx} B^n(x) = B^{n-1}(x+0.5) - B^{n-1}(x-0.5).$$

We first continue the analysis in 1D, and then extend it to 2D.

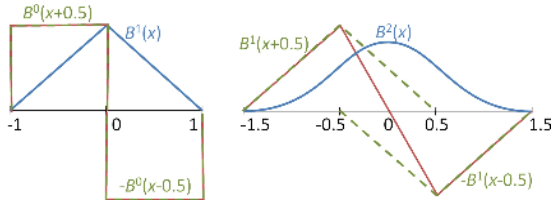


Figure 2: Given a B-spline of degree n (blue), its derivative (red) can be expressed as the difference of two offset B-splines of degree $n-1$ (dashed green), as shown here for $n=1,2$.

1D case Recall that the discrete signal v expresses a continuous function in terms of quadratic B-splines: $V(x) = \sum_i v_i B_i^2(x)$, where $B_i^2(x)$ is centered in the middle of the i -th lattice cell.

To compute the derivative of V we observe that:

$$\begin{aligned} \nabla V(x) &= N \cdot \sum_i \left(B_i^1(x) - B_{i+1}^1(x) \right) v_i \\ &= N \cdot \sum_i B_{i+1}^1(x) (v_{i+1} - v_i) = N \cdot \sum_i B_{i+1}^1(x) g_i. \end{aligned}$$

Thus, even though we are not given the original image V , we can compute its derivative from the forward-difference representation by interpreting the values g_i as coefficients of *first-order* B-splines.

Using this interpretation, we can directly compute the Laplacian constraints on the right-hand side:

$$f_j = \langle \nabla V(x), -\nabla B_j^2(x) \rangle = -N \cdot \sum_i g_i \left\langle B_{i+1}^1(x), \frac{d}{dx} B_j^2(x) \right\rangle.$$

2D case Recall that in the 2D case, the discrete array v defines a continuous function using tensor products of quadratic B-splines: $V(x, y) = \sum_{i,j} v_{i,j} B_i^2(x) B_j^2(y)$. As above, we obtain

$$\nabla V(x, y) = N \cdot \left(\sum_{i,j} B_{i+1}^1(x) B_j^2(y) g_{i,j}^x, \sum_{i,j} B_i^2(x) B_{j+1}^1(y) g_{i,j}^y \right).$$

And again, even though we are not given the original image V , we can compute its gradient from the forward-difference representation by interpreting the values $g_{i,j}$ as coefficients of *mixed* tensor products of first- and second-order B-splines.

Using this interpretation, we can directly compute the Laplacian constraints on the right hand side:

$$\begin{aligned} f_{i,j} &= -N \cdot \sum_{s,t} g_{s,t}^x \left\langle B_{s+1}^1(x), \frac{d}{dx} B_i^2(x) \right\rangle \left\langle B_t^2(y), B_j^2(y) \right\rangle \\ &\quad - N \cdot \sum_{s,t} g_{s,t}^y \left\langle B_s^2(x), B_i^2(x) \right\rangle \left\langle B_{t+1}^1(y), \frac{d}{dy} B_j^2(y) \right\rangle. \end{aligned}$$

Note that the inner products defining the Laplacian constraints are nonzero only if $|s-i|, |t-j| \leq 2$, so the computation can be expressed using two 5×5 stencils. Also, these stencils can be pre-computed and are constant within the domain interior.

For trivial Neumann boundary conditions, the B-spline basis functions must have zero derivatives at the domain boundaries $x = 0, 1$. This is easy to achieve by modifying the bases using reflection, as $B_i^2(x) \leftarrow B_i^2(x) + B_i^2(-x) + B_i^2(1-x)$ which also preserves essential nesting properties. The B-splines used to represent derivatives are modified accordingly to have trivial Dirichlet conditions using skew reflection, as $B_i^1(x) \leftarrow B_i^1(x) - B_i^1(-x) - B_i^1(1-x)$.

5.3 Defining the multigrid operators

B-splines are an effective multigrid basis for two reasons. First, they provide nested subspaces under grid subdivision, ensuring that solutions found at coarser resolutions can be (losslessly) realized in the finer resolution bases. Second, their local support allows relaxation and prolongation to be efficiently computed with compact stencils [Christara and Smith 1997].

Using B-splines in a multigrid setting, the prolongation operator becomes the linear map nesting B-splines at resolution N in the space of B-splines at resolution $2 \cdot N$. By duality, the restriction operator is defined as the transpose of the prolongation operator.

Thus, for quadratic B-splines in 1D, prolongation matrices P have local weights $\frac{1}{4}(3 \ 1)$ and $\frac{1}{4}(1 \ 3)$ on alternating rows; the rows of restriction matrices R have weights $\frac{1}{4}(1 \ 3 \ 3 \ 1)$; and, rows of Laplacian matrices L have weights $\frac{1}{6}(1 \ 2 \ -6 \ 2 \ 1)$.

For bi-quadratic B-splines, the local 2D stencils for prolongation, restriction, and the Laplacian are respectively:

$$\frac{1}{16} \begin{pmatrix} 9 & 3 \\ 3 & 1 \end{pmatrix}, \frac{1}{16} \begin{pmatrix} 1 & 3 & 3 & 1 \\ 3 & 9 & 9 & 3 \\ 3 & 9 & 9 & 3 \\ 1 & 3 & 3 & 1 \end{pmatrix}, \text{ and } \frac{1}{360} \begin{pmatrix} 1 & 14 & 30 & 14 & 1 \\ 14 & 52 & -12 & 52 & 14 \\ 30 & -12 & -396 & -12 & 30 \\ 14 & 52 & -12 & 52 & 14 \\ 1 & 14 & 30 & 14 & 1 \end{pmatrix}.$$

6 Streaming multigrid solver

Traditional implementations of multigrid perform restriction, prolongation, and relaxation iterations as separate passes, and therefore traverse the vectors u^l and f^l at the various levels many times. While each operation is computationally fast, especially for the small stencils of finite-difference schemes, such an approach is slow for out-of-core images. Also, it may hinder performance even for in-memory images due to lack of cache locality. Indeed, Bolz et al. [2003] and Goodnight et al. [2003] both report that their CPU and GPU multigrid implementations are bandwidth-limited.

Our insight is to perform *all* operations as streaming computations to maintain a small working set, and to group together as many computations as possible to minimize the number of data passes and thereby reduce memory and disk bandwidth. Our plan is three-fold. First, we implement the Gauss-Seidel solver so that all of its updates occur in one streaming pass. Second, we perform restriction and prolongation between levels in a streaming fashion. Finally, we interleave all multigrid operations across levels so that data can be transferred directly between the multiresolution solvers without having to be stored temporarily (to memory or disk).

Temporally blocked relaxation We perform k iterations of Gauss-Seidel relaxation as a single streaming operation. This is largely inspired by the works of [Pfeifer 1963; Douglas et al. 2000]. They show that, with careful attention to data-dependencies, several finite-difference Gauss-Seidel updates can be performed together by maintaining a moving block/tile of data values in the L1 cache.

We have found that the cache prefetcher in the Intel Core 2 processor, which automatically reads sequentially accessed data from the L2 to the L1 cache, is efficient enough that we can use a similar strategy to advance a window spanning entire rows of the image. The idea is to apply relaxation updates on all image pixels k times by processing pixels in a moving window tall enough to respect data dependencies. As this window sweeps down the image, its pixels are updated in “counter-current” order, as shown in Figure 3.

In adapting this approach to the second-order finite-element setting, we must ensure that two properties are satisfied. First, when updating the pixels in row j , pixels in rows $\{j-2, \dots, j+2\}$ must be

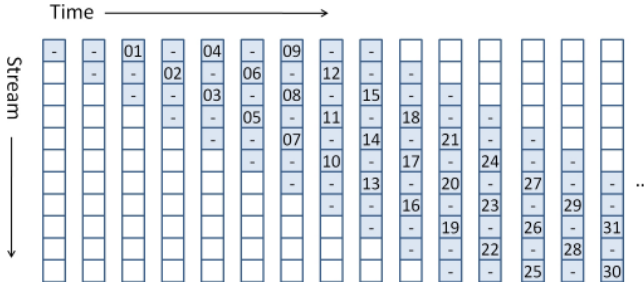


Figure 3: Visualization of the temporally blocked solver performing $k=3$ Gauss-Seidel iterations as a single streaming operation. Each square represents a row of pixels in the image; blue squares indicate the pixel rows that are memory-resident (i.e. in the image window); numbers identify the temporal sequence of the relaxations.

contained in the window, since the Laplacian stencil has size 5×5 . Second, as we update pixels in row j for the m -th time, pixels in rows $j-2$ and $j-1$ must have already been updated m times and pixels in rows $j+1$ and $j+2$ must have been updated $m-1$ times.

To satisfy these requirements, as the window sweep index i is advanced, we maintain a window consisting of rows $[i-1, i+2k+1]$, and perform a skipping, counter-current relaxation sweep, updating pixels in rows $\{i+2k-1, i+2k-3, \dots, i+1\}$. For instance, in the $k=3$ case shown in Figure 3, the second relaxation of the fourth row of pixels, labeled $\{11\}$, can be performed because the two prior rows have already been relaxed twice (as indicated by labels $\{02, 06\}$ and $\{03, 08\}$) and the two subsequent rows have already been relaxed once (as indicated by $\{07\}$ and $\{10\}$).

Streaming restriction and prolongation Because both these operations involve local stencils, they can be performed as streaming computations. It is most efficient to iterate over the destination level and accumulate stencil-weighted values from the source level.

Multi-level streaming We intersperse the restriction and relaxation operations across levels to realize the restriction phase as a single multi-streaming pass. And similarly, we intersperse the prolongation and relaxation operations to realize the prolongation phase as another multi-streaming pass. Therefore we are able to compute an entire multigrid V-cycle in just two streaming passes, as shown in Figure 1.

At each level of the restriction phase, the approximate solution u_R^l and the Laplacian constraint vector f^l are streamed to disk, while the restricted residual $f^{l-1} = R_{l-1}^l f^l$ is transferred to the next-coarser level via a streaming memory buffer. And at each level of the prolongation phase, u_R^l and f^l are streamed back from disk, and combined with the prolonged correction which is transferred from the coarser level via a streaming memory buffer.

For both the restriction and prolongation phases, we advance a sweep index i_l at each level l , with finer index i_{l+1} advancing twice for every advancement of coarser index i_l . We maintain windows of image rows on both u^l and f^l , much as in the temporally blocked relaxation solver. Each level performs several streaming computations, so the windows must be larger to account for the additional dependencies. The coarser-level relaxations trail behind those in finer levels during the restriction phase, whereas the finer-level relaxations trail behind those in coarser levels during the prolongation phase. Table 1 summarizes the parameters necessary for the multi-level streaming implementation.

| Restriction phase | | Prolongation phase | |
|--|---------------------|-------------------------|---------------------|
| $i_{l-1}+2k+1 < \lfloor (i_l-1)/2 \rfloor$ | | $i_{l+1}+2k+1 < 2i_l-1$ | |
| Data window | $[i_l-3, i_l+2k+1]$ | Data window | $[i_l-1, i_l+2k+1]$ |
| Restr. from $l+1$ | $[i_l+2k+1]$ | Prolong. from $l-1$ | $[i_l+2k+1]$ |
| Relaxation | $[i_l-1, i_l+2k+1]$ | Relaxation | $[i_l-1, i_l+2k+1]$ |
| Residual | $[i_l-3, i_l+1]$ | | |

Table 1: Inter-level dependencies on the sweep indices i_l over the image rows, extents of the streaming windows at each level, and window ranges accessed by each of the streaming operations.

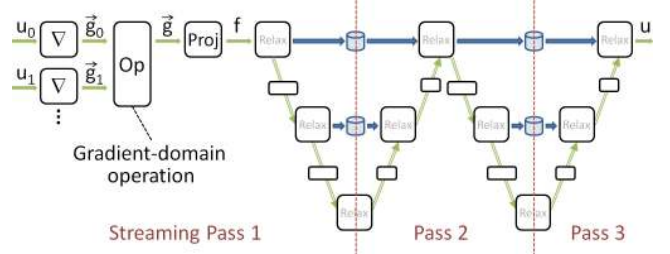


Figure 4: Full data pipeline for gradient-domain processing, showing two successive V-cycles. Typically a single V-cycle is sufficient.

Number of V-cycles In traditional multigrid, several V-cycles are necessary to ensure accurate convergence, resulting in performance degradation due to low I/O throughput on out-of-core data [Toledo 1999]. With the improved accuracy of the degree $n=2$ finite elements and the larger number $k \sim 5$ of relaxation iterations done per level, a single V-cycle often becomes sufficient (Section 7).

If additional V-cycles are necessary, these can be chained together as in Figure 4. A surprising fact is that the prolongation phase of a first V-cycle can be directly streamed into the restriction phase of a second V-cycle, to form a single combined streaming pass. Thus, the intermediate result between V-cycles need not be stored to disk.

Full streaming pipeline Figure 4 summarizes the pipeline, starting from one or more original images $\{u_i\}$, and resulting in a new image u . We extract the finite-difference gradient fields \vec{g}_i , and process them into a desired gradient field \vec{g} . Next we compute the Laplacian constraint vector f from \vec{g} as explained in Section 5.2. Finally, we solve the system $Lu = f$ using one or more multigrid V-cycles. With a single V-cycle, the full pipeline needs just two streaming passes. With v V-cycles, $v+1$ streaming passes are sufficient (rather than the expected $2v$), as shown for $v=2$.

Many image compression libraries like JPEG support streaming input and output, so we can directly write the final image in compressed form to disk. We find that such compression is faster than disk I/O, so it actually speeds up the overall process.

Setting the image mean Recall that with Neumann boundary conditions the solution u has an unconstrained mean value. Often one would like to prescribe this mean value, for instance to equal the average color of the original images $\{u_i\}$. The obvious scheme would require an extra post-processing pass. We have found a practical workaround as follows. We maintain the coarser levels up to 1024^2 pixels entirely in memory rather than disk. Then during the prolongation phase, at the finest memory-resident level we delay streaming to finer levels to allow a quick pass that assigns the desired mean value. Although relaxation at finer levels may still change the solution's mean, this change is small. For example, to stitch together the 19588×4457 pixel Red Rock image in Figure 8, we maintain resolutions 2560×576 and coarser entirely in core. By setting the mean value before proceeding to the prolongation of the

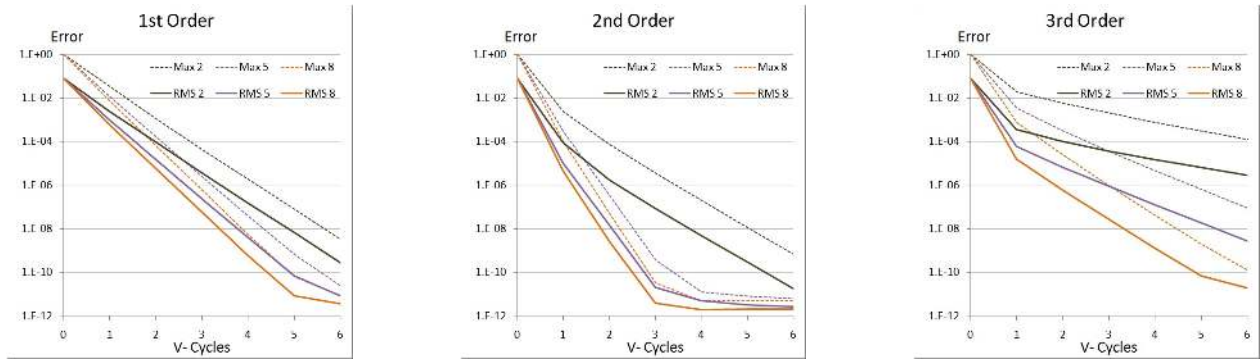


Figure 5: Plot of the rms and maximum errors as a function of the number of multigrid V-cycles, for elements of degree $n=1,2,3$ and using $k=2,5,8$ Gauss-Seidel updates. Errors are expressed as fractions of the value range. Second-order elements give the fastest convergence.

out-of-core resolutions, we obtain a final image whose mean value is within 0.1% of the prescribed value. Alternatively or in addition, the final mean can be computed during prolongation and stored in the output, for adjustment by any later stream processing.

Memory analysis We implement the active windows on u_R^l and f^l as circular memory buffers of image rows. These windows have constant height ($2k+5$ for the restriction phase, and $2k+3$ for prolongation), but their widths decrease geometrically at coarser levels along with the images. Therefore, if the input image has size $N_x \times N_y$, the total memory usage is $O(N_x)$. Interestingly, for the wide images shown in Section 9, it would be more memory-efficient to sweep horizontally rather than vertically, but unfortunately this is incompatible with the row-major ordering of image file formats.

Time analysis The amount of work at each multigrid level is linear on the image size at that level. Because the system size is reduced by a factor of 4 between successive levels, the overall time complexity is linear on the input size.

7 Efficient convergence of second-order elements

The efficiency of the Poisson solver depends on several parameters: the degree n of the finite elements, the number ν of V-cycles, and the number k of Gauss-Seidel updates. These parameters affect performance by dictating the number of image rows that must be memory-resident ($\sim kn$), the number of times that image data must be transferred to/from disk ($\nu+1$), and the overall per-pixel computational cost ($\nu k(2n+1)^2$).

In this section, we show that second-order elements provide the best convergence rate. We first demonstrate this empirically, and next provide theoretical justification by analyzing the spectral properties of the solver.

Empirical evaluation We examine the errors obtained in reconstructing an image from its Laplacian for a variety of different settings. Although this reconstruction problem does not represent a practical application, we should note that it is fundamentally as difficult as constructing an image from desired gradients, because in both cases the solver is presented with Laplacian constraints that are the Laplacian of some possibly unknown image.

Figure 5 plots rms and maximum error as functions of the number of V-cycles, for B-spline elements of degree $n=1,2,3$, using $k=2,5,8$ Gauss-Seidel updates. When no relaxations are performed (at 0 V-cycle), the zero initial solution has a maximum error that realizes the worst possible error (1.0), but an rms error (0.1) that happens



Figure 6: Visualization of the reconstruction errors for elements of order $n=1,2,3$ after a single V-cycle and $k=2$ Gauss-Seidel updates.

to be 10 times smaller. As expected, the error decreases steadily with the number of V-cycles, and shows a steeper slope with more relaxation updates. We used double-precision numbers in these experiments, so the error reaches a lower bound of about 10^{-12} .

We observe that the second-order elements provide the fastest convergence rate, with a per-cycle convergence factor of about 0.0014 in this example when using 5 relaxation updates. The maximum error appears to have the same asymptotic convergence rate.

As shown in Figure 6, using the ($n=2$) second-order finite elements, the error becomes nearly invisible after a single V-cycle and just $k=2$ Gauss-Seidel steps.

Spectral analysis To better understand the empirical behavior of the different solvers, we compute the spectrum of the two-grid operator T , which characterizes both the convergence of the relaxation and the effectiveness of the inter-grid transfer (restriction and prolongation) between the finer and coarser resolutions.

In our experiments, T is defined as the operator that (1) performs k Gauss-Seidel relaxation steps at the finer resolution, (2) restricts the residual to the lower resolution, (3) performs k relaxation steps at the coarser resolution, (4) adds the prolongation of the coarser solution to the finer solution, and (5) performs k more relaxations at the finer resolution. We assume simple periodic boundary conditions.

Figure 7 plots the magnitudes of the 64 largest eigenvalues of the two-grid operator T computed for a 32×32 grid using elements of degree $n=1,2,3$ with $k=2,5,8$ Gauss-Seidel updates. Even though the operator T is not symmetric and the eigenvalues may be complex, the fact that all eigenvalues have magnitude less than one guarantees that the solvers converge to the correct solution. The relative magnitudes of these spectra explain the empirical results seen above. Independent of the number of Gauss-Seidel updates used, the solvers using second-order elements have smaller spectra

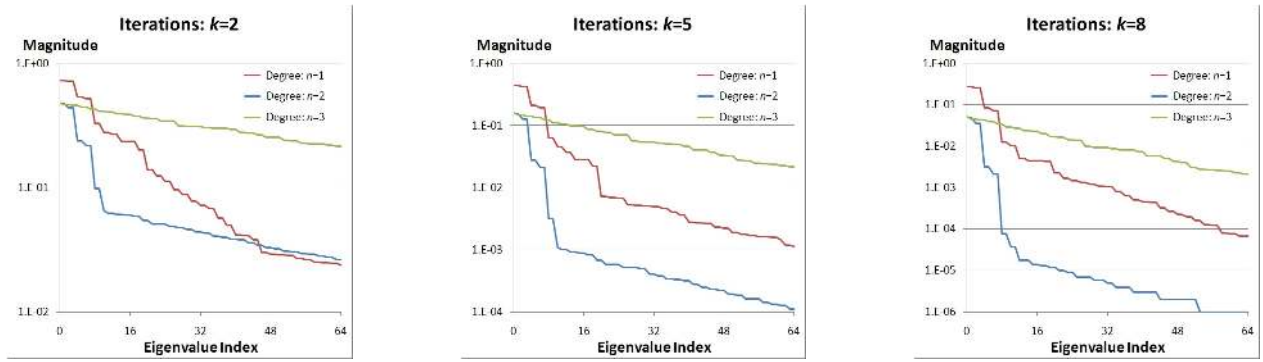


Figure 7: Plots of the 64 largest eigenvalue magnitudes for the two-grid linear operators computed on a 32×32 grid using B-spline elements of degree $n=1,2,3$ with $k=2,5,8$ Gauss-Seidel updates. Independent of the number of Gauss-Seidel updates used, the operators associated with the second-order solvers have the smallest magnitude eigenvalues, resulting in the fastest convergence.

than either first- or third-order solvers, thus explaining their faster convergence in Figure 5.

We believe that the slow convergence of the third-order finite elements is due to the increased support size of the third-order elements. When performing the Gauss-Seidel updates, the update of the i -th coefficient is performed under the assumption that all other coefficients of the solution are fixed. However, a subsequent update of the j -th coefficient can adversely affect the solution at the i -th coefficient if the value of the (i, j) -th entry in the Laplacian matrix is nonzero. Since the entry is nonzero precisely when the supports of the associated functions overlap, increased support size can detrimentally affect the solver by reducing the optimality of the individual coefficient updates.

Parameter selection In our out-of-core streaming implementation, we find that the runtime bottleneck is disk I/O rather than computation. So, we select parameters that provide a sufficiently accurate solution with a minimum number of V-cycles. In the context of 8-bit/channel image processing, we desire a solution with maximum error that is less than $1/256 \approx 4 \times 10^{-3}$, a condition that can be satisfied using second-order elements with a single V-cycle. In all our application results, we use second-order elements, with a single V-cycle, and five Gauss-Seidel updates.

8 Implementation

Maximizing disk throughput We transfer data to and from disk in large blocks (4MB) to minimize disk latency overhead, and thereby reach maximum disk bandwidth (about 30-40MByte/sec on a laptop PC). To obtain large block transfers, we initiate all I/O asynchronously in one separate thread. Because all streams advance sequentially at constant rates, we can precisely determine which data block will result in the earliest buffer underflow or overflow, and initiate the corresponding data read or write.

In addition, we obtain a 2X speed improvement by storing the intermediate u, f floating-point values on disk at half precision. Though this results in some data loss due to quantization, in Section 9 we show that sufficient accuracy is retained for performing gradient-domain image processing – giving rise to solutions that differ by no more than $\sim 0.03\%$ from the true solution.

Relaxation optimization For in-memory images, computation is the bottleneck. In particular, the slowest processing step is the streaming Gauss-Seidel relaxation. Each Gauss-Seidel pixel update involves a dot product of the 5×5 neighboring pixels with the 5×5

Laplacian stencil presented in Section 5.3. We optimize this computation by leveraging the vertical and horizontal symmetries of the stencil, and making use of the CPU SSE2 4-vector instructions.

Non-power-of-two images Currently we handle non-power-of-two images by padding the input image to a resolution $2^{l_{\max} - l_{\min}}$ times the resolution at the coarsest level. Since our only constraint on the resolution at the coarsest level is that it be small enough so that using a direct solver is not computationally expensive, we choose the coarsest resolution to be in the range $[32, 64]^2$. This is still sufficiently small to be solved quickly with a conjugate-gradients solver, while ensuring that the total size of the padding is no larger than $(33/32)^2 - 1 \approx 6\%$ of the original image.

Multi-channel images Some applications such as tone-mapping solve a Poisson equation on just the luminance of the image, while others like stitching require a separate Poisson equation for each color channel. Because gradient-domain processing may access the magnitude of the full-color gradient, we interleave the per-channel solutions to reduce the total number of passes.

9 Application results

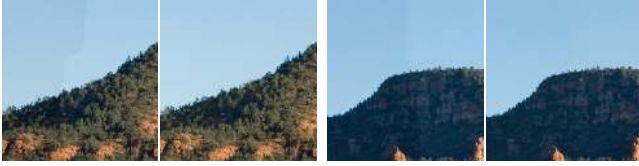
We evaluate our streaming solver in two practical applications, image stitching and tone-mapping. In both sets of experiments, no ground-truth solution is available a priori, so we obtain one by running our solver “to completion” with double precision both in-core and out-of-core, and using a large number of Gauss-Seidel updates across multiple V-cycles. To validate these ground truth solutions, we compare the size of the solution residual $r = f - Lu$ with the size of the constraint vector f . For all of the examples, the ratio of the norms $\|r\|/\|f\|$ is on the order of 10^{-10} .

All experiments are run on a notebook PC with a 2.2GHz Core 2 Duo processor and 4 GB of RAM. Timings include the I/O to read the gradient field \vec{g} from disk and to write the JPEG-compressed output image to disk, except for results labeled “in-core”. The initial solution u is set to zero in all experiments.

Image stitching Given a set of registered, segmented images, we combine them seamlessly into a single coherent image. We follow the approach of [Pérez et al. 2003; Agarwala et al. 2004], generating a vector field by copying the gradient values from the constituent images (zeroing out the gradients across the seams) and then solving the Poisson equation to get back the single image whose gradient most closely approximates the vector field.



19,588×4,457 (87-megapixel) panorama stitched from 9 photos



Close-ups, comparing direct copying of source image pixels

Figure 8: Example result of image stitching, obtained by copying the image gradients and solving the Poisson equation. (Data is courtesy of Aseem Agarwala.)

| Image name | Size (MP) | Rms error | | Max error | | Mem. (MB) | | Time (s) | |
|------------------------------|--------------|-------------------|-------------------|-------------------|-------------------|-----------|-----|----------|-----|
| | | SM | QT | SM | QT | SM | QT | SM | QT |
| Streaming in-core solver | | | | | | | | | |
| Beynac | 12 | $4 \cdot 10^{-5}$ | $4 \cdot 10^{-5}$ | $2 \cdot 10^{-4}$ | $5 \cdot 10^{-3}$ | 670 | 16 | 10 | 8 |
| Rainier | 23 | $2 \cdot 10^{-5}$ | $6 \cdot 10^{-5}$ | $2 \cdot 10^{-4}$ | $4 \cdot 10^{-3}$ | 1311 | 27 | 21 | 14 |
| Streaming out-of-core solver | | | | | | | | | |
| Beynac | 12 | $4 \cdot 10^{-5}$ | $4 \cdot 10^{-5}$ | $3 \cdot 10^{-4}$ | $5 \cdot 10^{-3}$ | 190 | 16 | 17 | 8 |
| Rainier | 23 | $3 \cdot 10^{-5}$ | $6 \cdot 10^{-5}$ | $3 \cdot 10^{-4}$ | $4 \cdot 10^{-3}$ | 110 | 27 | 33 | 14 |
| Edinburgh | 50 | $2 \cdot 10^{-5}$ | † | $3 \cdot 10^{-4}$ | † | 203 | 123 | 79 | 122 |
| Redrock | 87 | $5 \cdot 10^{-5}$ | † | $4 \cdot 10^{-4}$ | † | 133 | 112 | 118 | 118 |
| St. James | 3,342 | $2 \cdot 10^{-4}$ | | $6 \cdot 10^{-4}$ | | 408 | | 5,270 | |
| St. James _(log) | 3,342 | $1 \cdot 10^{-4}$ | | $1 \cdot 10^{-3}$ | | 408 | | 5,310 | |

Table 2: Solution error, peak memory usage, and running times for several image stitching experiments, comparing the results of our streaming multigrid (SM) solver with the quadtree (QT) solver of Agarwala [2007]. †Obtaining ground truth solutions on large images was impractical prior to our streaming multigrid scheme.

An example result is presented in Figure 8. As highlighted by the zoomed regions, stitching in the gradient domain eliminates the seam discontinuities at the image boundaries that would otherwise be apparent in the color composite (e.g. due to change in atmospheric conditions, exposure settings, etc. between snapshots).

We compute the solution errors, memory usage, and running times for a number of stitching examples. The in-core results store the temporary data values in 32-bit floats, while the out-of-core results use 16-bit floats. Table 2 summarizes the results, and presents a performance comparison of our streaming multigrid (SM) solver with the domain-specific quadtree (QT) solver of Agarwala [2007]. Despite the fact that the streaming multigrid solver is tailored towards solving a more general Poisson equation and therefore has time complexity that is linear on the number of pixels, it compares favorably with Agarwala’s adaptive method – always providing a solution with smaller maximum error, and having comparable running times at higher resolutions.

The results also show that our out-of-core streaming algorithm operates in a small memory footprint, approximately the same size as that used by the quadtree solver for the higher resolutions. The memory footprint of the streaming algorithm is linear on just the width of the image, i.e. usually the square-root of the number of pixels. In particular, for this application, the memory footprint is independent of the seam complexity, unlike in the QT solver.



Input image (16,950×2,956; 50 megapixels)



Output image



Close-up comparisons

Figure 9: Example result of tone-mapping, obtained by nonlinearly modulating log-luminance gradients.

| Image name | Size (MP) | Rms error | Max error | Memory (MB) | Time (s) |
|--------------------|-----------|-------------------|-------------------|-------------|----------|
| Ocean [†] | 1 | $7 \cdot 10^{-5}$ | $2 \cdot 10^{-3}$ | 29 | 0.3 |
| Edinburgh | 50 | $2 \cdot 10^{-3}$ | $5 \cdot 10^{-3}$ | 279 | 37 |
| St. James | 3,342 | $3 \cdot 10^{-4}$ | $2 \cdot 10^{-3}$ | 224 | 2,710 |

Table 3: Solution error, peak memory usage, and running times for several tone-mapping experiments. All errors are fractions of the input range. †For small images, results are solved entirely in-core.

Tone-mapping Given a single (possibly HDR) image, we perform tone-mapping to locally adjust the image contrast. We follow the approach presented in [Fattal et al. 2002], computing the log-luminance gradients, non-linearly attenuating them, solving the Poisson equation, and exponentiating the result to obtain the new image luminance.

The tone mapping problem is more challenging than image stitching because the difference image, obtained by subtracting the original image from the tone-mapped image, can have high-frequency components at arbitrary pixel positions. Consequently, adaptive methods cannot be applied, and the Poisson system must be solved at full resolution over the entire grid. In the past, this fact made it computationally infeasible to perform tone mapping on high-resolution images. Our streaming multigrid solver is the first system that can solve the associated Poisson equation in time that is linear on the number of pixels, and in a memory footprint that is linear on the image width.

Figure 9 shows an example result. As is highlighted by the zoomed-in regions, performing this type of non-linear attenuation provides a way to adaptively bring out detail in low-contrast regions. Table 3 summarizes the results on several examples. The execution times are faster than in the image stitching application for the same number of pixels because here the Poisson equation involves only a single channel (luminance).

Tone-mapping is performed in log-luminance space to better approximate perceived image brightness, so any error in the Poisson solution is exponentiated in the final image. Despite the fact that we measure errors in the final linear-space RGB image, we still obtain an accurate solution, with maximum errors on the order of $1/255$.



(a) Mosaic of 643 input photographs with differing exposures



(b) Result of stitching the images in linear RGB space



(c) Result of stitching the images in log-RGB space



(d) Result of tone-mapping (c) in log-luminance space

Figure 10: Result of stitching and tone-mapping at full resolution on a $88,309 \times 37,842$ (3.3-gigapixel) domain. (Data is courtesy of Matthew Uyttendaele.)

Gigapixel stitching and tone-mapping A challenge in stitching gigapixel images composed of hundreds of photographs is that due to the extreme lighting variations across the panorama, the photographs are usually taken under different exposure settings. Figure 10a shows an example of such a panorama, consisting of 643 photographs forming a $88,309 \times 37,842$ image (same data as in [Kopf et al. 2007b]). Automatic exposure settings are revealed by large brightness variations between adjacent photographs.

Applying our streaming multigrid technique, we can obtain three different seamless panoramas.

Performing traditional image stitching in the RGB space of the input photographs successfully removes the seam boundaries between the photographs, as shown in Figure 10b. However, the solution image may not capture the true scene contrast.

A better approach is to perform image stitching in log-RGB space [Szeliski et al. 2008]. To an approximation, changing the exposure of an image amounts to multiplying the pixels’ RGB values by a fixed constant. Because this multiplicative factor becomes a constant additive offset in log-RGB space, the effects of exposure vanish after gradient computation. After stitching the gradients of the log-RGB values, the exponentiated Poisson solution gives the HDR image shown in Figure 10c. We observe that the rightmost portion of the image seems overly bright, and believe that this is due to saturation in the source photographs.

Figure 10d shows the result of applying a tone-mapping operation to convert the stitched HDR result to an LDR image, using a second application of the streaming multigrid solver. The error, memory size, and execution time are shown in the last row of Table 3.

For this example dataset which has a row size of 88,000 pixels, our implementation can perform the entirety of the computation in under 500 MB of memory. Extrapolating this, a commodity PC with 2 GB memory should be able to handle a row size of 352,000 pixels, i.e. a 53-gigapixel image with the same aspect ratio.

10 Conclusions and future work

Streaming multigrid is a novel out-of-core technique for solving large global linear systems while requiring only local access and a few passes of sequential I/O, as demonstrated here for gradient-domain processing of huge images. It is able to solve general Poisson equations that cannot be adapted to a quadtree.

We have shown that a finite-element formulation, in which the image is interpreted as a continuous function, is compatible with the traditional setting of desired pixel differences. The resulting relaxation and inter-grid transfer operations require more local computation but achieve faster convergence. Thus a single V-cycle often suffices for an accurate solution in the context of image processing.

Our current system provides a solution to the Poisson equation on a 2D domain with Neumann boundary conditions. Some gradient-domain applications require that the solution conform to given boundary colors. We think that our approach could be extended to support such Dirichlet boundary value-constraints by appropriately modifying the 2D stencils and the construction of the constraint vector f . Introducing a soft constraint to match some original image u_0 , i.e. $\min \|\nabla u - \vec{g}\|^2 + \lambda \|u - u_0\|^2$, may also be achievable by modifying the 2D stencils and letting u_0 contribute to f . Supporting Dirichlet boundaries with irregular shape might be approached using a method akin to web-splines [Höllig et al. 2001].

Other gradient-domain techniques involve a weighted minimization $\|w(\nabla u - \vec{g})\|$ where $w(x, y)$ is a spatially varying 2×2 diagonal matrix that weights differences in x and y independently. We believe that our general philosophy of using streaming to minimize memory bandwidth could still prove useful in this setting.

The efficiency of streaming multigrid relies on the local support of the relaxation operator. While our work has focused on the Laplacian $\Delta U = U_{xx} + U_{yy}$ (also used for wave propagation, diffusion, electrostatics, etc.), streaming multigrid should extend to other differential operators, such as convection-diffusion $-\varepsilon \Delta U + a U_x$, and the Bilaplacian $\Delta^2 U = U_{xxxx} + 2U_{xxyy} + U_{yyyy}$ (for linear elasticity, Stokes flows).

It would be interesting to reduce disk bandwidth by using (very fast) compression and decompression of the streamed temporary data. Finally, there are exciting opportunities for parallelization of the streaming computation on many-core CPUs or GPUs, for instance by partitioning image rows.

Acknowledgments

We are very grateful to Aseem Agarwala and Matt Uyttendaele for sharing their aligned panorama images. We also thank Ketan Dalal and Bill Bolosky for valuable advice on implementing fast asynchronous I/O.

References

- AGARWALA, A., DONTCHEVA, M., AGRAWALA, M., DRUCKER, S., COLBURN, A., CULLESS, B., SALESIN, D., AND COHEN, M. 2004. Interactive digital photomontage. *ACM Transactions on Graphics (SIGGRAPH '04)*, 294–302.
- AGARWALA, A. 2007. Efficient gradient-domain compositing using quadrees. *ACM Transactions on Graphics (SIGGRAPH '07)*.
- AGRAWAL, A., RASKAR, R., NAYAR, S. K., AND LI, Y. 2005. Removing photography artifacts using gradient projection and flash-exposure sampling. *ACM Transactions on Graphics (SIGGRAPH '05)*, 828–835.
- BAE, S., PARIS, S., AND DURAND, F. 2006. Two-scale tone management for photographic look. *ACM Transactions on Graphics (SIGGRAPH '06)*.
- BOLITHO, M., KAZHDAN, M., BURNS, R., AND HOPPE, H. 2007. Multilevel streaming for out-of-core surface reconstruction. In *Symposium on Geometry Processing*, 69–78.
- BOLZ, J., FARMER, I., GRINSPUN, E., AND SCHRÖDER, P. 2003. Sparse matrix solvers on the GPU: Conjugate gradients and multigrid. *ACM Transactions on Graphics (SIGGRAPH '03)*, 917–924.
- BRANDT, A. 1977. Multi-level adaptive solutions to boundary-value problems. *Mathematics of Computation* 31, 333–390.
- BRIGGS, W., HENSON, V., AND MCCORMICK, S. 2000. *A Multigrid Tutorial*. Society for Industrial and Applied Mathematics.
- CHRISTARA, C., AND SMITH, B. 1997. Multigrid and multilevel methods for quadratic spline collocation. *BIT* 37, 4, 781–803.
- DOUGLAS, C., HU, J., KOWARSCHIK, M., RÜDE, U., AND WEISS, C. 2000. Cache optimization for structured and unstructured grid multigrid. *Electronic Transactions on Numerical Analysis* 10, 21–40.
- FATTAL, R., LISCHINSKI, D., AND WERMAN, M. 2002. Gradient domain high dynamic range compression. In *ACM SIGGRAPH*, 249–256.
- FINLAYSON, G., HORDLEY, S., AND DREW, M. 2002. Removing shadows from images. In *European Conference on Computer Vision*, 129–132.
- FLETCHER, C. 1984. *Computational Galerkin Methods*. Springer.
- GÖDDEKE, D., STRZODKA, R., MOHD-YUSOF, J., MCCORMICK, P., WOBKER, H., BECKER, C., AND TUREK, S. 2008. Using GPUs to improve multigrid solver performance on a cluster. *Intl. J. of Computational Science and Engineering*.
- GOODNIGHT, N., WOOLLEY, C., LEWIN, G., LUEBKE, D., AND HUMPHREYS, G. 2003. A multigrid solver for boundary value problems using programmable graphics hardware. In *Proc. of Graphics Hardware*, 102–111.
- GORTLER, S., AND COHEN, M. 1995. Variational modeling with wavelets. In *Symposium on Interactive 3D Graphics*, 35–42.
- HÖLLIG, K., REIF, U., AND WIPPER, J. 2001. Weighted extended B-spline approximation of Dirichlet problems. *SIAM Journal on Numerical Analysis* 39, 442–462.
- HORN, B. 1974. Determining lightness from an image. *Computer Graphics and Image Processing* 3, 277–299.
- KAZHDAN, M., BOLITHO, M., AND HOPPE, H. 2006. Poisson surface reconstruction. In *Symposium on Geometry Processing*, 73–82.
- KOPF, J., COHEN, M., LISCHINSKI, D., AND UYTTENDAELE, M. 2007. Joint bilateral upsampling. *ACM Transactions on Graphics (SIGGRAPH '07)*.
- KOPF, J., UYTTENDAELE, M., DEUSSEN, O., AND COHEN, M. 2007. Capturing and viewing gigapixel images. *ACM Transactions on Graphics (SIGGRAPH '07)*.
- LEVIN, A., ZOMET, A., PELEG, S., AND WEISS, Y. 2004. Seamless image stitching in the gradient domain. In *European Conference on Computer Vision*, 377–389.
- LOSASSO, F., GIBOU, F., AND FEDKIW, R. 2004. Simulating water and smoke with an octree data structure. *ACM Transactions on Graphics (SIGGRAPH '04)*, 457–462.
- MCCANN, J., AND POLLARD, N. 2008. Real-time gradient-domain painting. *ACM Transactions on Graphics (SIGGRAPH '08)*.
- PÉREZ, P., GANGNET, M., AND BLAKE, A. 2003. Poisson image editing. *ACM Transactions on Graphics (SIGGRAPH '03)*, 313–318.
- PFEIFER, C. 1963. Data flow and storage allocation for the PDQ-5 program on the Philco-2000. *Communications of the ACM* 6, 7, 365–366.
- SZELISKI, R., UYTTENDAELE, M., AND STEEDLY, D. 2008. Fast Poisson blending using multi-splines. Tech. Rep. MSR-TR-2008-58, Microsoft Research.
- SZELISKI, R. 2006. Locally adapted hierarchical basis preconditioning. *ACM Transactions on Graphics (SIGGRAPH '06)*, 1135–1143.
- TOLEDO, S. 1999. A survey of out-of-core algorithms in numerical linear algebra. In *External Memory Algorithms and Visualization*, J. Abello and J. S. Vitter, Eds. American Mathematical Society Press, Providence, RI, 161–180.
- WEISS, Y. 2001. Deriving intrinsic images from image sequences. In *International Conference on Computer Vision*, 68–75.