Distributed Gradient-Domain Processing of Planar and Spherical Images

MICHAEL KAZHDAN
Johns Hopkins University
and
DINOJ SURENDRAN and HUGUES HOPPE
Microsoft Research

Gradient-domain processing is widely used to edit and combine images. In this paper we extend the framework in two directions. First, we adapt the gradient-domain approach to operate on a spherical domain, to enable operations such as seamless stitching, dynamic-range compression, and gradient-based sharpening over spherical imagery. An efficient streaming computation is obtained using a new spherical parameterization with bounded distortion and localized boundary constraints. Second, we design a distributed solver to efficiently process large planar or spherical images. The solver partitions images into bands, streams through these bands in parallel within a networked cluster, and schedules computation to hide the necessary synchronization latency. We demonstrate our contributions on several datasets including the Digitized Sky Survey, a terapixel spherical scan of the night sky.

Categories and Subject Descriptors: I.4.9 [**Image processing and computer vision**]: Applications

General Terms: Algorithms

Additional Key Words and Phrases: Panoramas, spherical parameterization, screened Poisson equation, streaming multigrid, distributed solver

ACM Reference Format:

Kazhdan, M., Surendran, D., and Hoppe, H. 2010. Distributed gradient-domain processing of planar and spherical images. ACM Trans. Graph. 29, 2, Article XXX (April 2010), XX pages.
DOI = 10.1145/1559755.1559763
http://doi.acm.org/10.1145/1559755.1559763

DOI 10.1145/1559755.1559763
http://doi.acm.org/10.1145/1559755.1559763

1 Introduction

Over the last decade, gradient-domain approaches have become increasingly prevalent for image processing in computer graphics and vision [Agrawal and Raskar 2007]. The basic strategy is to extract gradient fields from one or more images, modify the gradient values, and find the new image that best approximates the desired gradients. The least-squares solution of this over-constrained problem is obtained by solving a Poisson equation.

Many practical applications of gradient processing have been demonstrated, including removal of light and shadow effects [Horn 1974; Finlayson et al. 2002], reduction of dynamic range [Fattal et al. 2002; Weyrich et al. 2007], creation of intrinsic images [Weiss 2001], image stitching [Pérez et al. 2003; Agarwala et al. 2004; Levin et al. 2004], removal of reflections [Agrawal et al. 2005], gradient-based sharpening [Bhat et al. 2008], and most recently interactive authoring [McCann and Pollard 2008; Orzan et al. 2008].

Our work extends the framework of gradient-domain image processing in two directions:

Spherical domains. Prior techniques operate on Euclidean domains such as planar and cylindrical geometry, yet many images are inherently spherical, such as panoramas [Shum and Szeliski 2000], planetary surfaces [Google 2008; Microsoft 2008], and environment maps. We adapt the gradient-domain methodology to support the processing of spherical imagery. Our solution involves a new spherical parameterization, the periodic quincuncial (PQ) map, which has horizontal periodicity and bounded area distortion. These properties let us design a streaming multigrid solver whose boundary constraints are local to the stream front, and which converges in a small number of Gauss-Seidel iterations. We demonstrate gradient-domain operations on several examples (Section 4).

Distributed solver. Solving the Poisson equation over a large image can require significant computation and I/O, especially if the image does not fit in memory. We enable efficient parallel processing over large image domains by developing a distributed solver. Bands of the image are assigned to nodes in a computer cluster and processed in parallel. The network latency required to synchronize data across bands is masked behind computation. A unique aspect of our solver is that it handles images whose sizes exceed the aggregate memory of the cluster, because each distributed process still operates in a streaming fashion. We show significant speedups for gigapixel-size images now common on the Internet (Section 6).

To further showcase these two contributions, we combine them to address the problem of stitching images from the Digitized Sky Survey [DSS 2007]. The survey comprises 1790 individual 529-megapixel color plates acquired from two telescopes over more

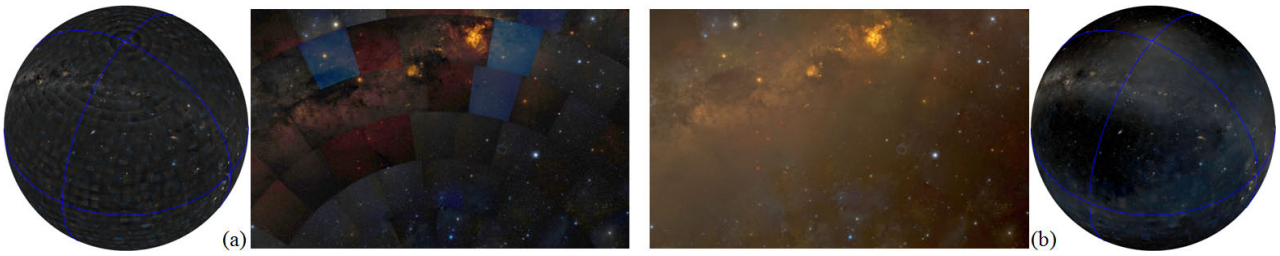

Fig. 1. Gradient-domain stitching of a terapixel panorama of the spherical night sky (viewed with gamma adjustment for printability). (a) Due to differences in exposure, simple juxtaposition of the photographic plates results in undesirable seams. (b) Using our distributed spherical Poisson solver, the gradients across the plate boundaries can be set to zero, resulting in a seamless spherical panorama.

than a decade (Figure 1). Together these overlapping plates form a terapixel, spherical image. However, significant exposure differences among the plates result in undesirable discontinuities. To eliminate these seams, our method solves a gradient-domain image problem with $3.3 \cdot 10^{12}$ degrees of freedom in 9 hours on a 16-node computer cluster (Section 7).

2 Related work

Spherical images. Traditional equirectangular parameterizations for spherical imagery can lead to sampling and continuity issues near the poles (Figure 2). To create spherical panoramas from photographs, Szeliski and Shum [1997; 2000] resample these into cube maps, and reduce seam discontinuities by applying a feathering algorithm. However, such feathering can lead to blurring or ghosting. By leveraging more recent, gradient-based techniques, we can remove seams more effectively. To our knowledge, gradient-domain approaches have not previously been applied to spherical domains.

Gradient-domain solutions on large images. Agarwala [2007] shows that for the application of gradient-based image stitching, the associated Poisson equation can be discretized adaptively along the stitching boundaries using a quadtree structure to yield good approximate solutions. Kazhdan and Hoppe [2008] solve the 2D Poisson equation efficiently using a multigrid V-cycle by interleaving its computations into two streaming passes on out-of-core data. McCann [2008] points out that the 2D Poisson equation can be solved exactly using the Fourier analysis and cyclic reduction (FACR) method [Press et al. 2007], which can be implemented in two streaming passes, with four times the temporary storage. We chose to base our distributed solver on a multigrid approach because it is more efficient and involves less network communication.

Parallel and distributed multigrid solvers. There is significant work on multigrid solvers for multiprocessors and computer clusters. An excellent recent survey is that of Chow et al. [2005]. For our case of structured grids, a common approach is multi-color (e.g. red-black) Gauss-Seidel relaxation, which allows parallel updates of independent grid elements [Briggs et al. 2000]. Wallin et al. [2006] improve caching efficiency by applying the coloring scheme to slices in a temporally blocked solver, and exploiting the shared cache of a chip multiprocessor. These approaches assume that the grid fits in the memory of a single machine.

Fig. 2. Close-ups of the South Pole in Google Earth, Virtual Earth, and Google Sky, showing either distortions or a "white cap".

Larger grids are typically partitioned into blocks that are solved on separate cluster nodes. One challenge in domain decomposition is to reduce the communication cost necessary to synchronize the data after each pass of Gauss-Seidel relaxation. Moreover, the simple approach of performing relaxation on all blocks independently corresponds to a block-Jacobi scheme and often results in poor solution convergence [Chow et al. 2005]. Adams [2001] improves convergence using a multi-colored block decomposition with Gauss-Seidel updates across blocks. Stookey et al. [2008] solve overlapping blocks independently and spatially blend the resulting approximations; because their problem involves hard constraints on a uniform subset of pixels, the Laplacian has local influence.

Our approach is to partition the image domain into bands, which the processors stream in lockstep. Similar to [Wallin et al. 2006], we improve cache friendliness by using temporal blocking, and like [Adams 2001] we hide network latency by introducing buffer regions and carefully scheduling computation. A unique aspect of our system is that it is both streaming and distributed at the same time, and therefore it is able to efficiently handle problems whose data sizes exceed the aggregate memory of the cluster.

3 Review of gradient-domain processing

In gradient-domain image processing, an application specifies desired color values and color differences, and the system solves for the image that best fits these constraints.

Formally, for each image color channel, one provides a vector field $V : \Omega \subset \mathbb{R}^2 \rightarrow \mathbb{R}^2$ specifying the desired gradients of an image, a scalar field $W : \Omega \subset \mathbb{R}^2 \rightarrow \mathbb{R}$ specifying the desired values, and an interpolation weight α, and the system solves for the function $U : \Omega \rightarrow \mathbb{R}$ that best fits the constraints:

$$\min_U \int_\Omega \|\nabla U - V\|^2 + \alpha (U - W)^2 .$$

Using the Euler-Lagrange equation, this minimization is equivalent to solving the screened Poisson equation [Bhat et al. 2008]:

$$(\Delta - \alpha) U \;=\; \nabla \cdot V \;-\; \alpha W \; .$$

Finite-element discretization. To make the solution tractable, the equation is transformed into a finite-dimensional system by only considering solutions within an N-dimensional subspace of functions spanned by elements $B_i(p)$ (with $1 \leq i \leq N$). We let the elements B_i be second-order bivariate B-splines centered on a regular grid [Christara and Smith 1997; Kazhdan and Hoppe 2008].

In this space, the problem of solving for the function U transforms into that of finding the coefficients $\{u_i\}$ such that the function $U(p) = \sum u_i B_i(p)$ minimizes the fitting error. Using the Galerkin formulation, this leads to the linear system $Lu = f$ where $u = (u_1 \dots u_N)^T$ is the vector of coefficients, L is the discretized screened Laplacian, and $f = (f_1 \dots f_N)^T$ are the constraints:

$$\begin{aligned} L_{i,j} &= -\int_\Omega \left(\nabla B_i\right)^T \nabla B_j + \alpha B_i B_j \\ f_i &= -\int_\Omega \left(\nabla B_i\right)^T V + \alpha B_i W \ . \end{aligned} \tag{1}$$

4 Spherical gradient-domain processing

To extend traditional image processing techniques to spherical imagery, we define a method for fitting an image to local constraints defined over the sphere. The challenge in doing this is defining the linear system that relates the constraints to the image.

We first formulate the problem directly on the sphere, then work toward a practical solution using a parameterization of the sphere onto a planar domain, and finally discuss our construction of the spherical parameterization and its impact on a multigrid solver.

4.1 Direct formulation on the sphere

Similar to the planar case, gradient-domain image processing can be performed when the image is defined over the sphere. The user provides a vector field $\breve{V} : S^2 \to TS^2$ specifying the desired spherical gradients of an image, a scalar field $\breve{W} : S^2 \to \mathbb{R}$ specifying the desired values, and an interpolation weight α, and the system finds the best-fitting function $\breve{U} : S^2 \to \mathbb{R}$:

$$\min_{\breve{U}} \int_{S^2} \|\nabla \breve{U} - \breve{V}\|^2 + \alpha (\breve{U} - \breve{W})^2. \tag{2}$$

As in the planar case, the function $\breve{U}$ minimizing the error is obtained by solving the screened Poisson equation:

$$(\Delta - \alpha)\breve{U} \ = \ \nabla \cdot \breve{V} \ - \ \alpha \breve{W} \ ,$$

where Δ is the Laplace-Beltrami operator which accounts for the geometry of the sphere.

Finite-element discretization. Once again, the solution is made tractable by defining an N-dimensional subspace of spherical functions spanned by the elements $\breve{B}_i(p) : S^2 \to \mathbb{R}$ and solving the system $Lu = f$ for the coefficients of $\breve{U}$, with:

$$\begin{aligned} L_{i,j} &= -\int_{S^2} \left(\nabla \breve{B}_i\right)^T \nabla \breve{B}_j + \alpha \breve{B}_i \breve{B}_j \\ f_i &= -\int_{S^2} \left(\nabla \breve{B}_i\right)^T \breve{V} + \alpha \breve{B}_i \breve{W} \ . \end{aligned} \tag{3}$$

4.2 Pulling back to a planar domain

To compute the linear system coefficients in Equation 3 we would have to choose a parameterization $\Phi : \Omega \subset \mathbb{R}^2 \to S^2$ and compute the integrals by (1) pulling back the finite elements and the constraint function to the parameterization domain and (2) applying the inverse of the push-forward map to transform the spherical vector field to a planar one.

Formally, denoting the Jacobian matrix of Φ by $J_\Phi = (\Phi_x \ \Phi_y)$, setting $B_i = \breve{B}_i \circ \Phi$ and $W = \breve{W} \circ \Phi$ to be the pull-back functions, and setting $V = J_\Phi^{-1} \breve{V}$ to be the vector field on the planar domain whose push-forward is the constraint vector field, the integrals become:

$$\begin{aligned} L_{i,j} &= -\int_\Omega \left(\nabla B_i^T \left(J_\Phi J_\Phi^T \right)^{-1} \nabla B_j + \alpha B_i B_j \right) \|\Phi_x \times \Phi_y\| \\ f_i &= -\int_\Omega \left(\nabla B_i^T V + \alpha B_i W \right) \|\Phi_x \times \Phi_y\| \ . \end{aligned} \tag{4}$$

Defining the system in this manner is the correct approach as it defines the screened Poisson equation in terms of the intrinsic spherical geometry and is independent of the parameterization Φ. However, it results in an inhomogeneous system of equations, where the stencils in the rows of matrix L contain spatially varying coefficients. The coefficients themselves are expensive to compute as they involve integrals, and moreover these integrals cannot be expressed in closed form for many parameterizations.

In this work, we instead resort to a non-intrinsic implementation in which we calculate the integrals directly in the planar domain, as in Equation 1. This presents concerns, since we only have

$$\int_{S^2} \left(\nabla \breve{B}_i\right)^T \nabla \breve{B}_j = \int_\Omega \left(\nabla B_i\right)^T \nabla B_j$$

when Φ is angle-preserving (conformal). And we only have

$$\int_{S^2} \breve{B}_i \breve{B}_j = \int_\Omega B_i B_j$$

when Φ is area-preserving. Since there is no parameterization that simultaneously preserves both angle and area (i.e. that is isometric), any non-intrinsic implementation must introduce some error.

Our strategy is to define a low-distortion parameterization, which compromises a small amount of accuracy for significantly improved performance. We will show that such an implementation gives rise to a solver that is significantly faster, while still generating images that are perceptually similar to the intrinsic solutions.

4.3 Choosing a spherical parameterization

The spherical parameterization should have low distortion to reduce the approximation error of our non-intrinsic system. Fortunately, this goal of minimal distortion is shared by many graphics and visualization applications. With too much area distortion, some surface regions are inevitably undersampled relative to others. And, with too much angle distortion (e.g. near the poles in any cylindrical projection), texture mapping requires costly anisotropic filtering to compensate for the poorly behaved surface parameterization.

The Digitized Sky Survey dataset in WorldWide Telescope [WWT 2008] is sampled using a so-called "tessellated octahedral adaptive subdivision transform" (TOAST) parameterization. It is defined as the composition of two simple maps:

(1) An octahedron is mapped to a sphere by recursive spherical subdivision (Figure 3a-b). This map is used in a number of applications in graphics [Fekete 1990; Schröder and Sweldens 1995] and astronomy [Kunszt et al. 2001]. The map is not differentiable, but provides a nice sample distribution over the sphere.

(2) The octahedron is projected into its equatorial plane to form a double-sided square, and the bottom side of this square is unfolded quincuncially (Figure 3c-d). Although this projection is not isometric, the samples of the subdivided octahedron map one-to-one with a 2D grid [Praun and Hoppe 2003].

For our purposes, however, the topology of the square's boundaries prevent a streaming solver. Our solution is to introduce a final step:

(3) The top half of the square is pivoted by 180° to form a rectangle domain whose width is four times its height (Figure 3e).

We refer to this result as the *periodic quincuncial* (PQ) parameterization. It is a map of the sphere onto a rectangle domain, with periodic boundary conditions in the horizontal directions, reflective boundary conditions along the top row, and twice-reflective boundary conditions in the bottom row. The crucial property is that all boundary constraints are local to a small window of rows in this rectangular domain. As shown in Figure 4, each row of the rectangle domain maps to a closed loop on the sphere, starting from a doubly covered 180° great arc (red), and ending at another doubly covered 180° great arc (orange).

Choice of Spherical Map. In developing a multigrid solver for spherical imagery, we have opted for the PQ parameterization because it has bounded area and and angle distortion and because it is well-suited to a streaming and distributed solver.

Although parameterizations such as HEALPix [Górski et al. 2005] and the rhombic-dodecahedron [Fu et al. 2009] have better distortion bounds (Figure 5) and can also be used to define efficient multigrid solvers (Table I), they are not suitable choices for streaming and distributed implementations. The limitation of both these parameterizations is that the quad-mesh over which they parameterize the sphere has vertices with odd valence, resulting in an angle defect that is a multiple of $\pi/2$. As a consequence, the neighbors of pixels within a stream-row may reside along a perpendicular column that is not contained within the working memory. In contrast, for our PQ parameterization, all extraordinary vertices have even valence, so the angle defect is a multiple of π, and the neighbors of a pixels within a stream-row always reside within an adjacent stream-row (though possibly with flipped orientation).

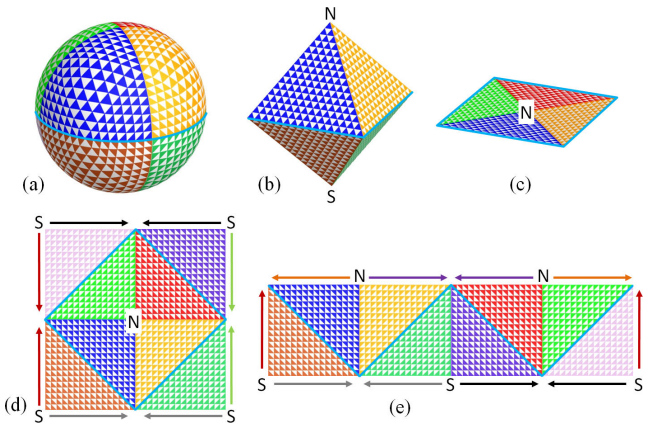

Fig. 3. The periodic quincuncial (PQ) parameterization is formed by mapping the sphere to an octahedron, a double-sided square, a larger square, and finally a rectangle with horizontal periodicity. The colored arrows indicate stitching boundaries in the domain.

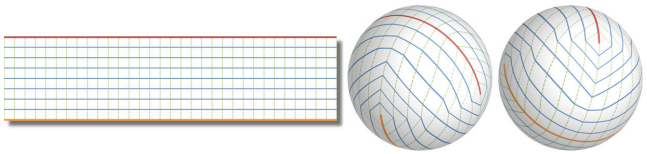

Fig. 4. Two views of the PQ spherical parameterization, showing how the domain grid maps to the sphere. The top and bottom boundaries map to doubly covered great arcs. The blue lines indicate the rows traversed by our streaming solver.

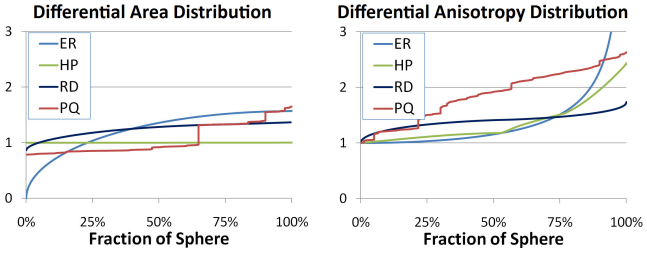

Fig. 5. In an equirectangular (ER) spherical map, area and anisotropic distortion are unbounded, whereas both these measures are bounded in the HEALPix (HP), rhombic-dodecahedron (RD), and periodic-quincuncial (PQ) parameterizations. (Differential area is the determinant of the Jacobian and anisotropy is the ratio of its singular values, so values of one correspond to no distortion.)

4.4 Adapting a planar multigrid solver

Using the PQ parameterization it is straightforward to adapt finite-element solvers to the processing of images over the sphere. Since a d-th order B-spline is supported within a radius of $(d+1)/2$, performing a restriction, prolongation, and Gauss-Seidel relaxation only requires access to pixel values within a small neighborhood (never larger than d). Since our implementation uses second-order B-splines, the processing of the pixels at a distance of more than two pixels from the parameterization boundary is unchanged.

As Figure 3e indicates, near the left and right boundaries processing must be adapted to support periodic boundary conditions, and near the top and bottom boundaries processing must be adapted to support reflective boundary conditions. Specifically, for an image of resolution $W \times H$, with $W = 4H$, the mapping is:

$$(i,j) \mapsto \begin{cases} (W\text{-}1\text{-}\tilde{i}, -j) & \text{if } j < 0 \\ (W/2\text{-}1\text{-}\tilde{i}, 2H\text{-}1\text{-}j) & \text{if } j \geq H \text{ and } i < W/2 \\ (3W/2\text{-}1\text{-}\tilde{i}, 2H\text{-}1\text{-}j) & \text{if } j \geq H \text{ and } i \geq W/2 \\ (\tilde{i}, j) & \text{otherwise,} \end{cases}$$

where $\tilde{i} = i \bmod W$.

The PQ parameterization has the desirable property that the two-ring neighborhood of pixel (i,j) always gets mapped into a set of pixels that are within rows $[j\text{-}2, j\text{+}2]$. Thus, a streaming solver needs to maintain only a small window of image rows as it streams through the image, allowing for the gradient-domain processing of spherical imagery that is larger than working memory.

4.5 Evaluation

To evaluate the quality of our periodic quincuncial (PQ) parameterization for spherical gradient-domain processing, we compare our results to those obtained using an intrinsic solver defined over an equirectangular (ER) parameterization.

We chose the ER parameterization because the associated metric tensor remains constant over the parallels of the sphere. As a result, defining the intrinsic system only requires that a new stencil be computed once per image row, rather than once per pixel. Note that though efficient to set up, the inhomogeneous nature of the system results in slow convergence, as shown in Table I. Consequently, we have only found this system to be effectual for establishing the baseline intrinsic solution, a context in which the solver can be allowed to run to completion and efficiency is not a concern.

Figure 6 shows results of our comparison for two different applications. For each evaluation, the columns show the original spherical image, the result obtained with the intrinsic ER solver and with our non-intrinsic PQ parameterization, and the error, magnified fivefold and visualized as darker regions on an off-white sphere.

Ground truth. We extract the gradient field of an image and test that both solvers accurately integrate the gradients to get back the original image (up to DC term). For the intrinsic solver, gradients are computed over the sphere, for our non-intrinsic solver gradients are computed over the parameterization domain. Note that since the two parameterizations sample the sphere at different points, the PQ solution had to be resampled to compare it against the ER solution, and small sampling errors can be seen, particularly in regions of high frequency detail.

Since the initial image is the correct solution we can also measure the performance by comparing the results of the solvers with the original. It is precisely these errors that are described in Table I.

Image stitching. We split the sphere with random great circles, perturbing half of the spherical image through brightening or gamma correction, and solve the Poisson equation to get back a seamless image. Constraints are defined by compositing gradients from the partitions and setting the seam-crossing gradients to zero.

Table I. Comparison of solver convergence using different spherical parameterizations

V-Cycles		1	2	3	4
ER	RMS	$6.7\cdot10^{-03}$	$4.6\cdot10^{-03}$	$3.5\cdot10^{-03}$	$2.9\cdot10^{-03}$
	Max	$2.1\cdot10^{-01}$	$1.7\cdot10^{-01}$	$1.6\cdot10^{-01}$	$1.5\cdot10^{-01}$
HP	RMS	$2.6\cdot10^{-04}$	$1.3\cdot10^{-06}$	$7.5\cdot10^{-09}$	$4.4\cdot10^{-11}$
	Max	$8.9\cdot10^{-04}$	$3.0\cdot10^{-06}$	$1.5\cdot10^{-08}$	$8.1\cdot10^{-11}$
PQ	RMS	$6.1\cdot10^{-05}$	$1.1\cdot10^{-07}$	$1.9\cdot10^{-10}$	$2.6\cdot10^{-12}$
	Max	$8.1\cdot10^{-04}$	$1.4\cdot10^{-06}$	$2.5\cdot10^{-09}$	$8.2\cdot10^{-12}$

Convergence of the intrinsic equirectangular (ER) solver compared with that of the non-intrinsic HEALPix (HP) and periodic-quincuncial (PQ) solvers, measured as rms and max solution error. (Since the base mesh for the rhombic-dodecahedron parameterization is identical to that of HEALPix, it defines the same finite-element system with identical convergence properties.)

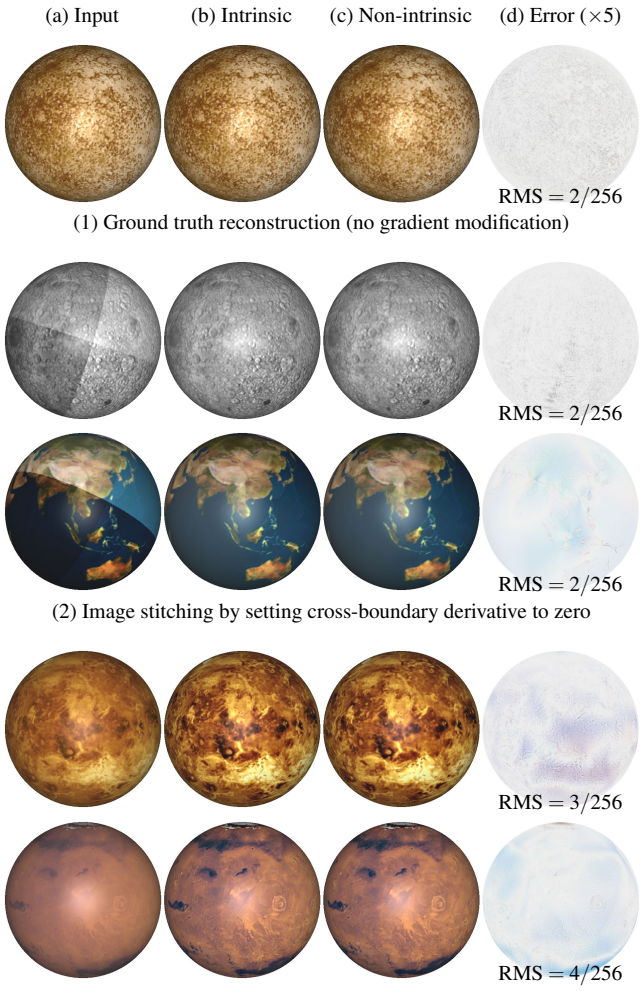

Fig. 6. Comparison of gradient-domain processing results using the costly intrinsic solution and our non-intrinsic approach. For visualization purposes, the difference is magnified 5 times.

Gradient sharpening. We constrain the solvers to simultaneously preserve the original color values and magnify the gradients. For input image I, this is done by setting the constraint values to the original image ($W = I$ and $\alpha > 0$) and setting the constraint vector field to the amplified gradients ($V = \beta\nabla I$ with $\beta > 1$) in Equation 2.

As Figure 6 shows, the low distortion of the PQ parameterization permits the non-intrinsic solver to attain results similar to the more expensive intrinsic solution. The greatest discrepancy arises in gradient sharpening, where the balancing of value and gradient constraints is known to be frequency-dependent [Bhat et al. 2008] and hence sensitive to the parametric distortions. But even in this case the error is primarily low-frequency, due to the bounded distortion of the parameterization, making the errors barely perceptible.

5 Review of streaming multigrid

The implementation of our solver (Section 6) builds on the streaming multigrid algorithm introduced by Kazhdan and Hoppe [2008], which we briefly summarize here. Using second-order elements, a

sufficiently accurate image solution is achieved in a single multigrid V-cycle, implemented as two sequential streaming passes over the data. The first pass is the restriction phase, in which the system is relaxed at finer resolutions and the residual is successively passed on to coarser resolutions for further processing. The second pass is the prolongation phase, in which the correction term from coarser resolutions is successively incorporated back into the finer resolutions and the system is further relaxed. The key components of the streaming multigrid solver are as follows.

Streaming Gauss-Seidel relaxation. Gauss-Seidel relaxation over the image is implemented as a streaming computation by maintaining a short window of pixel rows in working memory. As the stream is advanced, the pixels in the front row of the window are read in from disk, the pixels in the center row are relaxed, and then the pixels in the back row are written out to disk. Since a second-order stencil is used, relaxing the value at pixel (i, j) only requires access to the pixel values in the two-ring neighborhood $[i-2, i+2] \times [j-2, j+2]$, and a single relaxation of all the pixel values is performed in a working memory size of five rows.

Streaming multiple iterations of relaxation. To perform k successive Gauss-Seidel relaxations, the streaming computation is modified so that after reading row j into the front of the window, pixels in row $j-2$ are relaxed, followed by pixels in row $j-4$, all the way through pixels in row $j-2k$. The pixels in row $j-2k-2$ are then written back out to disk and the stream is advanced. Thus, by maintaining a window of $2k+3$ rows, a sequence of k successive Gauss-Seidel relaxations is performed in a single streaming pass through the image data.

Streaming the multigrid solver. The complete multigrid solver interleaves the relaxations of the different resolutions. As pixel updates are finalized at one resolution they are directly transferred into the next resolution, either as the restricted residual for the coarser resolution problem or as the prolonged correction term for the finer resolution. All inter-resolution data transfers occur via memory rather than disk I/O. The streams at the different resolutions are advanced synchronously, with each finer resolution advancing two rows for each row at the next-coarser resolution.

6 Distributing the streaming multigrid solver

The streaming solver of Kazhdan and Hoppe [2008] works on large, out-of-core images. However, due to limits on CPU and disk throughput, it requires about 90 minutes to process a 3.3-gigapixel panorama. By parallelizing this computation over multiple threads and processors, we significantly reduce this computation time. In addition, distributing the computation allows scaling to terapixel images, where the streaming window (spanning across more than a million pixels) would no longer fit into the cache of a single CPU.

Our distributed solver operates over many processors in a computer cluster. Each node of the cluster generally contains a few processors (e.g. 4 processors in a quad-core node), which access the same cache and memory. However, processors on different nodes do not share memory, and must communicate over a local network.

To reduce inter-processor communication, we exploit the fact that updates to the solution performed within a Gauss-Seidel relaxation only affect the computation of adjacent pixels. Our approach is to partition the image domain into contiguous bands, assign the bands to different processors, and have each processor compute its component of the overall solution (Figure 7).

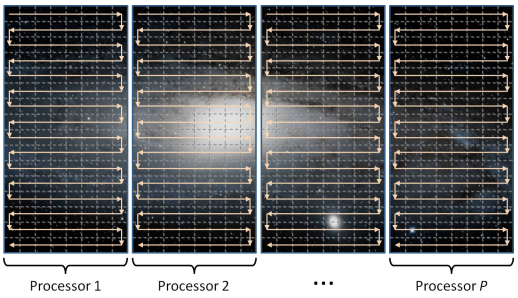

Fig. 7. Our solver decomposes the 2D image domain into vertical bands, assigns the bands to different processors, and lets all processors stream through the rows in lockstep.

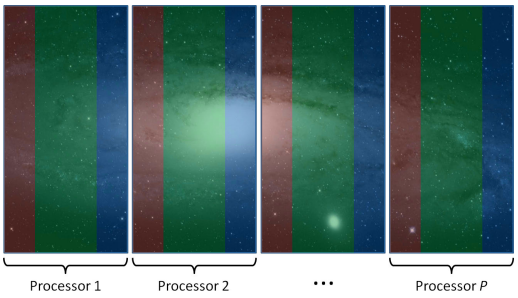

Fig. 8. We further decompose each image band into three lanes (shown by the red, green and blue colors), such that lanes of the same color can be processed in parallel.

An important challenge is the presence of data dependencies between adjacent pixels across a band boundary. Since updating these solution coefficients requires access to data contained in a different band, the processors must synchronize data to ensure that the correct values are used. This leads to two separate concerns. First, the sequence in which the solution coefficients are relaxed must allow processors to work in parallel, so that one processor does not stall waiting for its neighbors to finalize their solution values. Second, since updated coefficients near the band boundaries must be synchronized across adjacent processors after each Gauss-Seidel update, we must ensure that the associated inter-process communication does not bottleneck the system.

6.1 Parallelizing the Gauss-Seidel updates

To overcome the cycle of data dependencies between bands, and thereby enable parallelism across processors, we further divide each band into a set of three lanes, illustrated as red, green, and blue in Figure 8. This lets all processors update their lanes of the same color in parallel, since for example the neighborhood of any red pixel on one processor does not include any red pixels from another processor. In fact, just two lanes per processor would be sufficient, but having three lanes is useful for our synchronization algorithm described in the next section. The multi-lane structure is related to traditional multi-color partitioning used in parallelizing Gauss-Seidel relaxations [Briggs et al. 2000], except here it is used at a coarser granularity (i.e. over lanes rather than pixels) to account for the fact that memory is distributed.

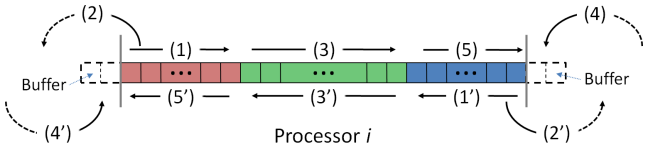

Fig. 9. Processing the green lane between the time that boundary data is sent and the time it is received allows for a non-blocking implementation of a distributed Gauss-Seidel relaxation.

6.2 Synchronizing band boundary values

To relax pixels near the boundaries of a band, each processor must have access to its neighbors' recently updated boundary values. We support this by assigning each processor a left and right *buffer* that stores copies of the neighbors' pixel values. After Gauss-Seidel updates are performed, neighboring processors communicate their updated boundary data into these buffers. The key to efficiency is a careful scheduling that allows inter-process communication to occur *while* the coefficients are relaxed. We first describe scheduling multiple Gauss-Seidel relaxations of a single row, and then present the implementation for the entire image.

Distributing the relaxation of a single row. To hide the latency of inter-process communication, we interject the processing of the center green lane between the sending and receiving of the recently relaxed boundary values, and symmetrize the processing by using boustrophedonic (alternating direction) traversal.

Figure 9 shows how this allows non-blocking computation of multiple Gauss-Seidel relaxations. In every even update, all processors update their pixels in a left-to-right traversal. Specifically, they (1) relax their red lanes, (2) send the updated boundary values to their left neighbors, (3) relax their green lanes, (4) receive the updated boundary values from their right neighbors, and (5) relax their blue neighbors. Then in every odd update, the update order is reversed, with steps (1')–(5') performed in a right-to-left traversal.

The data transfer initiated in step (2) and completed in step (4) occurs asynchronously with the relaxation of the green lane in step (3), so there is no need to block on inter-processor communication. The receipt of updated pixel values in step (4) ensures that as we process pixels near the right boundary of the blue lane in step (5), the accessed buffer pixels are all up-to-date.

Distributing the relaxation of multiple rows. We extend the same strategy to perform k Gauss-Seidel relaxations of all the image rows in a single streaming pass. Recall from Section 5 that the sequential algorithm maintains a window of $2k$+3 rows and updates every other row in front-to-back order. To obtain correct results in our distributed solver, we increase the height of the window by one row, extend the width of left/right buffers from two pixels to $2k$ pixels, and extend the range of the relaxation updates slightly into adjacent lanes and buffers.

Figure 10 shows an example of performing $k=3$ updates per pixel. The top diagram shows the left-to-right processing associated with an initial window, and the bottom diagram shows the right-to-left processing after advancing the window by one row. Due to the symmetry, we only provide details for the left-to-right traversal:

(1) Relaxing the red lane The pixels in the fourth row of the red lane are relaxed from left to right, followed by the pixels in the sixth row, and continuing all the way through the $(2k$+2)-nd row. For each row $2l$+2, relaxation is begun $2(k$-$l)$ pixels into the left buffer and continues out to $2(k$-$l)$ pixels into the green lane.

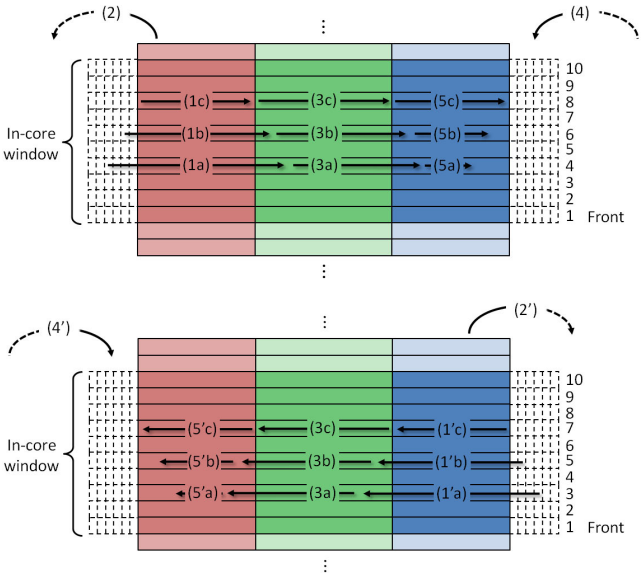

Fig. 10. The sequence of relaxations permitting a distributed implementation of $k=3$ Gauss-Seidel relaxations, without blocking on data synchronization between processors.

(2) Sending left boundary data The first $2k$ pixels of each in-core row are sent to the right buffer of the neighbor on the left. Additionally, for each row $2l$+2, the recently updated $2(k$-$l)$ pixels in the left buffer are sent as well.

(3) Relaxing the green lane Pixels in rows $4,6,\ldots(2k$+2) are successively relaxed. For each row $2l$+2, this relaxation begins $2(k$-$l)$+1 pixels into the green lane and continues out to $2(k$-$l)$ pixels into the blue lane.

(4) Receiving right boundary data Each row of the right buffer is updated with $2k$ pixels received from the neighbor to the right. Additionally, for each row $2l$+2, the last $2(k$-$l)$ pixels of the blue buffer are updated with the data from the neighbor to the right.

(5) Relaxing the blue lane Pixels in rows $4,6,\ldots(2k$+2) are relaxed. For each row $2l$+2, relaxation begins $2(k$-$l)$+1 pixels into the blue lane and continues up to $2(k$-$l)$ pixels from the end of the blue lane.

6.3 Correctness

To be a valid implementation of the Gauss-Seidel algorithm, each pixel must be updated k times and there must exist an ordering of the pixels such that the l-th update of pixel p reads neighboring pixels that have themselves been updated either l times if they lie earlier than p in the ordering or l-1 times if they lie later. Though cumbersome, it can be shown that our implementation exactly reproduces k iterations of Gauss-Seidel relaxation using an interleaved boustrophedonic ordering with strides determined by the widths of the bands.

6.4 Implementation details

Boundary conditions. The distributed solver can operate on both planar and spherical images. For planar images, we assume Neumann boundary conditions (zero cross-boundary derivatives), so all operations are local in the grid. For spherical images, the basis functions are reflected into the PQ parametric domain as described in Section 4.4. To handle the boundary reflections for the first and last two rows of the image, we maintain a 2-row buffer beyond the boundary, and repeatedly synchronize this buffer data across nodes after each update of the first and last two rows. The cost of this special processing is negligible for large domain sizes.

Lane sizes. To maximize latency hiding for inter-process communication, we maximize the width of the green lanes relative to the red and blue ones.

Improved cache temporal locality. The central green lane may be so wide that it causes the processor caches to thrash. Ideally, all $2k$+4 rows of the in-core window should reside in cache. Or, at a minimum, the caches should hold the three rows $2l$+2, $2l$+3, and $2l$+4 that are reused by the successive relaxations of rows $2l$+2 and $2l$+4. To improve cache friendliness, we partition the band into more than three lanes by dividing the green region into additional lanes. The computation for these interior lanes proceeds as before, using repeated instances of step (3).

Merging of bands at coarser solution levels. On the other hand, as the image size shrinks at coarser multigrid levels, the green lane may become so narrow that network latency is no longer masked effectively. To counter this, we merge bands at coarser levels. This reduced set of bands maps to fewer processors, and therefore reduces computational parallelism, but fortunately the number of image rows in these coarser levels decreases proportionally, so there is no asymptotic penalty.

Coarsest solution. At a sufficiently coarse level, data is transferred from all active nodes to a master process, the coarse solution is computed using a conjugate gradients solver, and then it is redistributed to the active nodes.

Network library. To communicate among the cluster nodes, we use TCP network sockets, much like the Message Passing Interface (MPI) protocol. A single process is launched on each cluster node, with separate threads for each processor (core), so the threads can communicate via shared memory.

Fig. 11. Seamless stitching of a 3.3-gigapixel image from 643 photographs, and close-ups of gradient-based sharpening. Both operations take about 6 minutes on a 4-node cluster.

Table II. Quantitative comparisons

		Time (s)				Max error			
Image	Pixels	SM	QT	FACR	Ours	SM	QT	FACR	Ours
Edinburgh	50M	79	122	190	17	5e-4	-	5e-12	1e-3
Redrock	87M	118	118	333	18	1e-3	-	3e-12	1e-3
Lobby	1G	2096	-	-	136	-	-	-	-
St. James	3.3G	5270	-	-	378	1e-3	-	-	2e-3
DSS data	64G	-	-	-	2,070	-	-	-	-
DSS data	1T	-	-	-	33,400	-	-	-	-

Comparison of streaming multigrid [Kazhdan and Hoppe 2008], quadtree adaptivity [Agarwala 2007], Fourier analysis cyclic reduction [McCann 2008], and our distributed parallel solver (with 4 nodes and 4 bands/node, except DSS with 16 nodes). Errors are fractions of the input range. The small accuracy difference between the SM results and ours is due to a minor implementation detail.

7 Results

We experiment with our distributed multigrid solver using different numbers of nodes within a 16-node computer cluster. Each node is an 8-core 2.5GHz Xeon L5420 with 16 GB memory and 1.6 TB disk space. The nodes are connected with gigabit Ethernet.

A surprising finding is that the JPEG decompression of the input images (and JPEG compression of the output) require a significant portion of the overall computation, so we allocate half of the processing threads just for these operations. We therefore consider the processing of 1, 2, or 4 image bands per node.

Figure 11 shows the result of stitching the 3.3-gigapixel St. James panorama of 643 photographs acquired in [Kopf et al. 2007]. It is solved in 378 seconds on 4 nodes with 4 bands per node. This is 14 times faster than the serial non-distributed computation in [Kazhdan and Hoppe 2008]. We verified that in stitching these photographs, our distributed solver has the same convergence properties as those reported for the non-distributed solver. Namely, the rms error is 0.1 and the maximum error is 0.6, in the color range [0,255]. Table II compares execution times and solution error with prior techniques for some small and medium-size images.

We also explore gradient-based sharpening, in which the image gradient is amplified but a soft constraint is introduced to preserve the original colors [Bhat et al. 2008]. Recall from Section 3 that this involves solving a screened Poisson equation in which the introduction of the soft constraint modifies the coefficients of the matrix L and the constraint vector f. However, the system has the same structure and therefore the solver has the same performance. Figure 12 shows gradient-based sharpening applied to a gigapixel spherical image, again using a solver distributed over 4 nodes.

Table III. Processing efficiency as a function of machine parallelism

Image size	32,768 × 8,192			131,072 × 32,768			524,288 × 131,072			2,097,152 × 524,288		
	Bands/node			Bands/node			Bands/node			Bands/node		
Nodes	1	2	4	1	2	4	1	2	4	1	2	4
1	0.98	0.75	0.42	0.86	0.66	0.47	0.86	0.70	0.41	†	†	†
2	0.96	0.78	0.51	0.87	0.75	0.47	0.83	0.68	0.38	†	†	†
4	0.90	0.70	0.41	0.89	0.74	0.42	0.85	0.68	0.39	†	†	†
8	0.84	0.61	0.38	0.89	0.50	0.35	0.71	0.73	0.36	†	†	†
16	0.68	0.52	0.29	0.78	0.76	0.42	0.89	0.50	0.26	0.75	0.66	0.46

Effective processing efficiency, measured as solution rate in megapixels per second per band, for various numbers of cluster nodes and bands per node. (Each band is assigned to a separate processor on a node.) †For the largest image, the solver required at least 16 nodes for sufficient temporary disk space.

Fig. 12. Gradient-based sharpening on a gigapixel spherical panorama, computed in 136 seconds on a 4-node cluster.

DSS dataset. The terapixel Digitized Sky Survey forms a 2048K×512K grid in the PQ parametric domain. Therefore, in the fully parallelized configuration of 16 cluster nodes and 4 bands per node, each processor is assigned a 32K×512K image band. To analyze scalability, we consider the same dataset downsampled at 6 more octaves, down to a resolution of 32K×8K. Figure 13 shows results of stitching on the full-size dataset. Due to our aggressive gamma adjustment, some low-frequency patterns can be seen in the results. These patterns are caused by vignetting artifacts in the source color plates. This reflects the general limitation of gradient-domain stitching in the presence of variations that are more complicated than simple offsetting or scaling, and would arise with any Poisson solver. It is not due to our approach or implementation.

To assemble a panorama, the input photographs must be mapped through some homography (even if very slight), so one resampling step is required. For viewing, our PQ-parameterized spherical image can be used directly as a texture map over a tessellated mesh, so no additional resampling is necessary. The rendered texture does not suffer from anisotropic sampling artifacts at the poles, as would be observed with a traditional equirectangular parameterization.

The solver requires 24 bytes/pixel to represent the image u and constraints f as 3-channel floating-point values. So without streaming computation, the full-resolution terapixel image would need 24 TB, more than 100 times the aggregate memory of the cluster. With streaming, memory usage is only 700 MB per node.

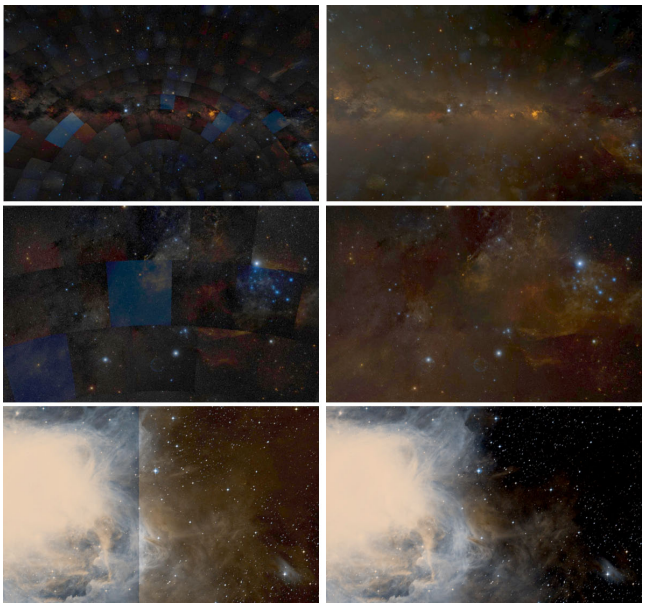

Fig. 13. Close-ups of the terapixel sky survey at different scales, showing the gradient-based stitching results (gamma-enhanced).

The solver requires 16 bytes/pixel of disk space to store the intermediate result between the restriction and prolongation phases at all multigrid levels. Due to this disk space requirement, the terapixel solution needs to run on a minimum of 16 nodes.

Table III shows the processing rates for the different datasets on various numbers of nodes, and with different numbers of bands (threads) per node. This rate is expressed as the number of solved pixels divided by the solution time and by the number of processors, and is therefore a measure of the efficiency of the distributed solver. Thus, for the terapixel dataset, the rate is $4.6{\cdot}10^5$ pixels/sec/processor, and with 16×4 processors running, the overall computation time is $1{\cdot}10^{12}/4.6{\cdot}10^5/64$=33,400 seconds or 9.3 hours. The 8-core processing nodes have about 60% CPU utilization when running 4 bands per node. The utilization beyond 50% is due to the asynchronous decompression and compression of the images. We did not reach 100% utilization with a higher number of bands per node due to synchronization inefficiencies between the image decompression/compression threads and the solver.

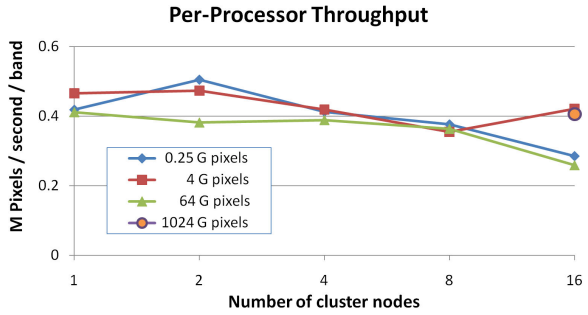

Fig. 14. Effective solution rate per processor, in megapixels per second, as a function of the number of cluster nodes and of the size of the image. (The number of bands per processor is fixed at 4.)

Figure 14 graphs this processing rate data for the case of 4 bands per node. The relatively flat curves in the graph indicate that the rate is relatively constant as we change the number of nodes in the range from 1 to 16, and as we change the size of the image from 250 megapixels to 1 terapixel — over nearly 5 orders of magnitude. This is strong confirmation of the solver's scalability.

8 Discussion

In designing a parallel approach for applying gradient-domain techniques to large images, we have developed a solver that implements classical multigrid with Gauss-Seidel relaxations within each level of the multigrid hierarchy. We have chosen to use such an approach over parallelized domain-decompositions [Smith et al. 2004] because the latter give rise to additive Schwarz methods that are known to converge less efficiently in general and are often only used as preconditioners for Krylov subspace methods [Xu 1992; Szyld and Frommer 1998; Nabben 2003]. (Note that the Schwarz alternating method is precluded by our assumption that the system does not fit into the aggregate memory of the cluster as such an implementation would require multiple disk reads/writes of the solution and constraints at each level of the multigrid hierarchy.)

Retrospectively, we have found that for gradient-domain processing, domain-decompositions may in fact provide sufficient accuracy. As an example, Table IV compares the magnitudes of the residual obtained using 5 Gauss-Seidel iterations to relax the solution on a 1024×1024 image. Residuals were obtained by running a multigrid solver in which the domain at each level of the hierarchy was partitioned into four sub-domains with 2, 5, and 10 pixels of overlap. Gauss-Seidel relaxation was performed on each of the sub-domains in parallel, and the individual solutions were then merged to obtain the relaxed solution for that level of the hierarchy.

While we have not explored such an extension in this work, our solver could be easily modified to support a domain-decomposition approach. In addition to partitioning the image columns into column-bands assigned to different processors, we would also partition the rows of the image into row-bands. The computation across the columns would proceed as before while the computation in the different row-bands would now also be performed in parallel. One advantage of such an approach might be that in distributing the parallelism across both the rows and the columns of the image, the column-bands become wider, allowing for more computation to occur between the sending and receiving of boundary information, alleviating a possible network I/O bottleneck.

Table IV. Comparison with domain decomposition

	Classical GS	Domain-Decomposition		
		Overlap=2	Overlap=5	Overlap=10
Stitching	$5.4 \cdot 10^{-2}$	$+7.3 \cdot 10^{-3}$	$+1.1 \cdot 10^{-3}$	$+8.8 \cdot 10^{-6}$
Sharpening	$1.3 \cdot 10^{-4}$	$+1.5 \cdot 10^{-2}$	$+1.0 \cdot 10^{-3}$	$+5.3 \cdot 10^{-7}$

Residual magnitudes obtained when applying classical Gauss-Seidel relaxation to the entire grid, and increases in residual magnitude when merging solutions obtained by separately relaxing over four sub-domains with different extents of pixel overlap.

9 Summary and future work

We extend gradient-domain processing to spherical domains. To maintain efficiency, we develop a spherical parameterization that allows stream processing yet avoids singularities that would lead to a poorly conditioned system. We use a non-intrinsic formulation, and show that the resulting approximation error is perceptually small and thus practical for graphics applications.

Applying gradient-domain approaches to expansive panoramas involves dauntingly large linear systems. We present a technique to solve such systems by parallelizing a multigrid computation over a distributed cluster. Due to its streaming design, our technique can handle images whose sizes far exceed the aggregate memory of the cluster. Results demonstrate excellent scalability with respect to both problem size and available parallelism.

In future work, it would be interesting to consider other spherical parameterizations, to further reduce distortion, and to allow closed-form evaluation and differentiability. Also, it would be desirable to approximate the intrinsic Laplace-Beltrami operator without incurring excessive computational cost. Another interesting direction is to consider the stitching of images having different resolutions, to form seamless transitions not only across stitching boundaries at a given resolution, but also across a mipmap pyramid.

ACKNOWLEDGMENTS

We are very grateful to Jonathan Fay and the Microsoft Research WorldWide Telescope project, the Digitized Sky Survey, Rick Szeliski, Michael Cohen, Aseem Agarwala, and Matt Uyttendaele for sharing their image data. This work was funded in part by the NSF Career Grant (#6801727).

REFERENCES

ADAMS, M. 2001. A distributed memory unstructured Gauss-Seidel algorithm for multigrid smoothers. In *Proc. of the ACM/IEEE SC2001 Conference*.

AGARWALA, A. 2007. Efficient gradient-domain compositing using quadtrees. *ACM Transactions on Graphics (SIGGRAPH '07)*.

AGARWALA, A., DONTCHEVA, M., AGRAWALA, M., DRUCKER, S., COLBURN, A., CURLESS, B., SALESIN, D., AND COHEN, M. 2004. Interactive digital photomontage. *ACM Transactions on Graphics (SIGGRAPH '04)*, 294–302.

AGRAWAL, A. AND RASKAR, R. 2007. Gradient domain manipulation techniques in vision and graphics. In *ICCV 2007 Course*.

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.

BHAT, P., CURLESS, B., COHEN, M., AND ZITNICK, L. 2008. Fourier analysis of the 2D screened Poisson equation for gradient domain problems. In *European Conference on Computer Vision*. 114–128.

BRIGGS, W., HENSON, V., AND MCCORMICK, S. 2000. *A Multigrid Tutorial*. Society for Industrial and Applied Mathematics.

CHOW, E., FALGOUT, R., HU, J., TUMINARO, R., AND YANG, U. 2005. A survey of parallelization techniques for multigrid solvers. In *Frontiers of Parallel Processing For Scientific Computing*. SIAM.

CHRISTARA, C. AND SMITH, B. 1997. Multigrid and multilevel methods for quadratic spline collocation. *BIT 37,* 4, 781–803.

DSS. 2007. Digitized Sky Survey. `http://archive.stsci.edu/dss/`.

FATTAL, R., LISCHINKSI, D., AND WERMAN, M. 2002. Gradient domain high dynamic range compression. *ACM Transactions on Graphics (SIGGRAPH '02)*, 249–256.

FEKETE, G. 1990. Rendering and managing spherical data with sphere quadtrees. In *VIS '90: Proceedings of the 1st conference on Visualization '90*. 176–186.

FINLAYSON, G., HORDLEY, S., AND DREW, M. 2002. Removing shadows from images. In *European Conference on Computer Vision*. 129–132.

FU, W., WAN, L., WONG, T., AND LEUNG, C. 2009. The rhombic dodecahedron map: An efficient scheme for encoding panoramic video. *IEEE Transactions on Multimedia 11,* 4, 634–644.

GOOGLE. 2008. Google Earth. `http://earth.google.com/`.

GÓRSKI, K., HIVON, E., BANDAY, A., WANDELT, B., HANSEN, F., REINECKE, M., AND BARTELMANN, M. 2005. HEALPix: A framework for high-resolution discretization and fast analysis of data distributed on the sphere. *The Astrophysical Journal 622*, 759–771.

HORN, B. 1974. Determining lightness from an image. *Computer Graphics and Image Processing 3*, 277–299.

KAZHDAN, M. AND HOPPE, H. 2008. Streaming multigrid for gradient-domain operations on large images. *ACM Transactions on Graphics (SIGGRAPH '08) 27*.

KOPF, J., UYTTENDAELE, M., DEUSSEN, O., AND COHEN, M. 2007. Capturing and viewing gigapixel images. *ACM Transactions on Graphics (SIGGRAPH '07)*.

KUNSZT, P., SZALAY, A., AND THAKAR, A. 2001. The hierarchical triangular mesh. In *Mining the Sky: Proceedings of the MPA/ESO/MPE Workshop*. 631–637.

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.

MCCANN, J. 2008. Recalling the single-FFT direct Poisson solve. In *SIGGRAPH 2008 Sketch*. ACM.

MCCANN, J. AND POLLARD, N. 2008. Real-time gradient-domain painting. *ACM Transactions on Graphics (SIGGRAPH '08) 27*.

MICROSOFT. 2008. Microsoft Virtual Earth. `http://www.microsoft.com/virtualearth/`.

NABBEN, R. 2003. Comparisons between multiplicative and additive Schwarz iterations in domain decomposition methods. *Numerische Mathematik 95*, 145–162.

ORZAN, A., BOUSSEAU, A., WINNEMÖLLER, H., BARLA, P., THOLLOT, J., AND SALESIN, D. 2008. Diffusion curves: A vector representation for smooth-shaded images. *ACM Transactions on Graphics (SIGGRAPH '08) 27*.

PÉREZ, P., GANGNET, M., AND BLAKE, A. 2003. Poisson image editing. *ACM Transactions on Graphics (SIGGRAPH '03)*, 313–318.

PRAUN, E. AND HOPPE, H. 2003. Spherical parametrization and remeshing. *ACM Transactions on Graphics (SIGGRAPH '03)*, 340–349.

PRESS, W., TEUKOLSKY, S., VETTERLING, W., AND FLANNERY, B. 2007. *Numerical Recipes*, 3rd ed. Cambridge.

SCHRÖDER, P. AND SWELDENS, W. 1995. Spherical wavelets: efficiently representing functions on the sphere. In *ACM SIGGRAPH Conference Proceedings*. 161–172.

SHUM, H. AND SZELISKI, R. 2000. Systems and experiment paper: Construction of panoramic mosaics with global and local alignment. *International Journal of Computer Vision 36*, 101–130.

SMITH, B., BJORSTAD, P., AND GROPP, W. 2004. *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, Cambridge, United Kingdom.

STOOKEY, J., XIE, Z., CUTLER, B., FRANKLIN, W., TRACY, D., AND ANDRADE, M. 2008. Parallel ODETLAP for terrain compression and reconstruction. In *ACM GIS*.

SZELISKI, R. AND SHUM, H. 1997. Creating full view panoramic image mosaics and environment maps. In *ACM SIGGRAPH Conference Proceedings*. 251–258.

SZYLD, D. AND FROMMER, A. 1998. Weighted max norms, splittings, and overlapping additive Schwarz iterations. *Numerische Mathematik 83*, 259–278.

WALLIN, D., LÖF, H., HAGERSTEN, E., AND HOLMGREN, S. 2006. Multigrid and Gauss-Seidel smoothers revisited: Parallelization on chip multiprocessors. In *Proceedings of ICS06*.

WEISS, Y. 2001. Deriving intrinsic images from image sequences. In *International Conference on Computer Vision*. 68–75.

WEYRICH, T., DENG, J., BARNES, C., RUSINKIEWICZ, S., AND FINKELSTEIN, A. 2007. Digital bas-relief from 3D scenes. *ACM Transactions on Graphics (SIGGRAPH '07) 26*.

WWT. 2008. WorldWide Telescope. `http://www.worldwidetelescope.org/`.

XU, J. 1992. Iterative methods by space decomposition and subspace correction. *SIAM Review 34*, 581–613.

Received September 2009; accepted December 2009