Efficient Preconditioning of Laplacian Matrices for Computer Graphics

Dilip Krishnan [*]
New York University

Raanan Fattal [†]
Hebrew University of Jerusalem

Richard Szeliski [‡]
Microsoft Research

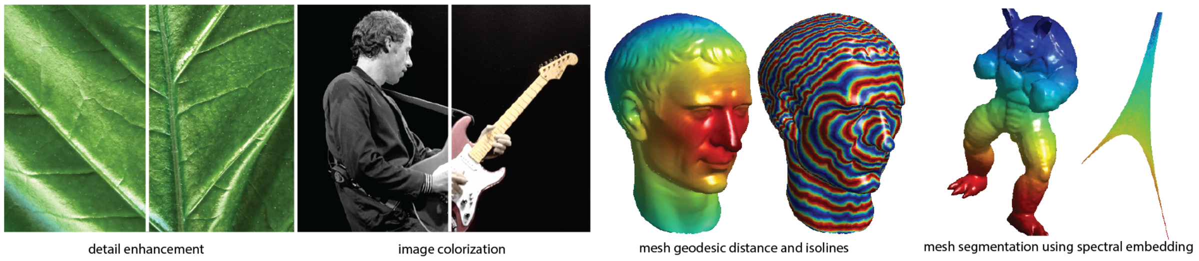
detail enhancement
image colorization
mesh geodesic distance and isolines
mesh segmentation using spectral embedding

Figure 1: *Example computer graphics applications that use Laplacian matrices.*

Abstract

We present a new multi-level preconditioning scheme for discrete Poisson equations that arise in various computer graphics applications such as colorization, edge-preserving decomposition for two-dimensional images, and geodesic distances and diffusion on three-dimensional meshes. Our approach interleaves the selection of fine- and coarse-level variables with the removal of weak connections between potential fine-level variables (*sparsification*) and the *compensation* for these changes by strengthening nearby connections. By applying these operations before each elimination step and repeating the procedure recursively on the resulting smaller systems, we obtain a highly efficient multi-level preconditioning scheme with linear time and memory requirements. Our experiments demonstrate that our new scheme outperforms or is comparable with other state-of-the-art methods, both in terms of operation count and wall-clock time. This speedup is achieved by the new method's ability to reduce the condition number of irregular Laplacian matrices as well as homogeneous systems. It can therefore be used for a wide variety of computational photography problems, as well as several 3D mesh processing tasks, without the need to carefully match the algorithm to the problem characteristics.

CR Categories: G.1.8 [Numerical Analysis]: Partial Differential Equations—Multigrid and Multilevel Methods; G.1.3 [Numerical Analysis]: Numerical Linear Algebra—Linear Systems Direct and Iterative Methods;

Keywords: matrix preconditioning, Laplacians, multigrid, mesh processing, computational photography

Links: DL PDF WEB

[*]e-mail: dilip@cs.nyu.edu
[†]e-mail:raananf@cs.huji.ac.il
[‡]e-mail:szeliski@microsoft.com

1 Introduction

A large number of problems in computer graphics and computational photography are formulated as norms over gradients and solved using discrete Poisson equations. Examples in computational photography include gradient-domain tone mapping [Fattal et al. 2002], Poisson blending [Pérez et al. 2003], alpha matting [Sun et al. 2004], image colorization [Levin et al. 2004], tonal adjustment [Lischinski et al. 2006], edge-preserving smoothing [Farbman et al. 2008], and image relighting and non-photorealistic rendering [Bhat et al. 2010]. Three-dimensional geometric processing applications include mesh segmentation [Liu and Zhang 2007] and geodesic distance computation [Crane et al. 2012], as well as mesh deformations and skinning weight computations. While the Poisson equation approach excels in terms of quality and mathematical conciseness, it comes at a considerable computational cost, as it requires solving very large and poorly-conditioned linear systems.

State of the art sparse direct solvers [Davis 2006] require approximately $O(n^{3/2})$ computer operations to solve planar (manifold) Poisson systems in n variables. Given the widespread use of Laplacian matrices in physical sciences, various preconditioning schemes have also been developed to accelerate their iterative solution. The geometric multigrid (GMG) method is one example, known to provide an optimal $O(n)$ running time for regular homogeneous Poisson equations [Trottenberg et al. 2001]. While classical GMG performs poorly on problems with strong spatial inhomogeneities, its generalization, the Algebraic Multigrid (AMG) method [Brandt 1986; Trottenberg et al. 2001], performs much better. AMG methods reformulate the problem over a hierarchy of adaptively selected coarser grids, related by interpolation weights tailored to capture the strong connections and stiff modes of the matrix. However, the irregular grid selection increases the bandwidth of non-zero elements in the coarser matrices and undermines AMG's efficiency. Using a more aggressive coarsening [Vaněk 1995] or regular subgrids with adaptive interpolation schemes [Zeeuw 1990; Szeliski 2006; Krishnan and Szeliski 2011] avoids this problem but also undermines the solver's effectiveness in the case of severe spatial inhomogeneities.

In this paper, we extend the method of Krishnan and Szeliski [2011] with adaptive coarsening to handle irregular meshes and inhomogeneous Laplacian matrices commonly used in computer graphics and computational photography. Our algorithm interleaves the adaptive selection of coarse and fine variables with a *sparsification* step, which carefully removes (relatively weak) connections while *com-*

pensating (strengthening) nearby connections in order to minimize the change in energy at low-frequency modes. By applying these steps repeatedly, we obtain a multi-level (hierarchical) approximation of the original matrix that provides an effective and efficient preconditioner. Our algorithm, which we call HSC, for *hierarchical sparsify and compensate*, is guided by formal insights into the effect of the sparsification and compensation steps on the condition number of the system.

Our new method is highly efficient and has an overall linear $O(n)$ construction time and memory complexity. Our experiments show that it outperforms or equals other state of the art methods, both in terms of operation count and wall-clock time. This speedup results from the new method's ability to dramatically reduce the matrix condition number. Our experiments also show that condition numbers of orders of 10^6 due to severe spatial irregularities are reduced to less than 10.

In the case of homogenous problems, our algorithm reduces to the GMG method and hence achieves optimal performance. For linear or tree-like inhomogeneous regions, our algorithm devolves to linear time cyclic reduction [Golub and Van Loan 1996] and (parallel) tree-based preconditioners. Performing well on these extreme cases makes our method well suited for mixed systems that contain large uniform regions separated by strong discontinuities, which often arise in graphics applications, such as edge-preserving smoothing [Farbman et al. 2008]. Our optimized MATLAB/Mex code is available for download at `http://www.cs.huji.ac.il/~raananf/projects/hsc/`.

We note that our algorithm is restricted to Laplacians which are M-matrices having non-positive off-diagonals. For those 3D mesh processing applications where cotangent Laplacians are necessary, it necessitates replacing the cotangent with a simpler M-matrix approximation.

2 Previous Work

Solving linear equations arising from discrete Laplace operators has received much attention from various scientific communities. We mention here the most popular methods for solving discrete Poisson equations and describe the main ideas behind them.

State of the art sparse direct solvers [Davis 2006] are based on the nested dissection method [Lipton et al. 1979]. From a theoretical perspective, nested dissection has strong guarantees on fill-in for systems arising from planar graphs. The best known algorithms for planar graphs run in time $O(n^{1.5})$ and are incorporated as direct solvers in software packages such as MATLAB. General dense direct solvers such as Gaussian elimination and LU factorization run in $O(n^3)$ time [Kincaid and Cheney 1991]. LU (and the related Cholesky) decomposition are often used for matrix preconditioning by dropping terms from the complete LU factorization [Saad 2003].

Iterative linear solvers such as Jacobi, Gauss-Seidel and CG are applicable to diagonally dominant and positive semi-definite matrices [Kincaid and Cheney 1991] including the Laplacian matrices we are considering. Since these methods consist of matrix-vector multiplications, each iteration runs in $O(n)$ time when solving sparse systems. However, as we noted earlier, in the case of Laplacian matrices, the number of iterations needed to achieve a particular accuracy depends on the matrix dimension and hence the complexity of these methods exceeds $O(n)$. Linear running time is achieved by applying these iterative solvers in a multi-level fashion, as in the geometric multigrid method [Brandt 1973]. The hierarchical basis method [Yserentant 1986; Szeliski 1990] use a similar approach, where a multi-level basis is used to precondition the matrix in conjunction with an iterative solver. The GMG method was extended to handle variable coefficients in [Alcouffe et al. 1981].

Several multigrid solvers have been adapted for specific computer graphics purposes. A streaming multigrid solver capable of solving very large problems, arising from processing gigapixel images, is described in [Kazhdan and Hoppe 2008]. Farbman et al. [2011] describe a highly optimized pyramid-based solver for tone mapping and interpolation. A multigrid framework for the simulation of high-resolution elastic deformable models, supporting linear and co-rotational linear elasticity, is described in [Zhu et al. 2010].

Shi et al. [2006] develop a multigrid solver to handle mesh deformation problems. They show significant speedup over direct solvers for meshes of upto 3 million vertices. The solver we present in this paper is faster in wall-clock time and also has higher accuracy. For example, the solver in [Shi et al. 2006] takes 2.8 seconds to process a mesh with 800K vertices on a Pentium 4, 3.8GHz with relative residual 10^{-3}. We process a mesh of the same size in about 0.5s on a single-core Xeon 2.7GHz with relative residual 10^{-6}. Bolz et al. [2003] present a GPU-based multigrid solver. However, they restrict their numerical experiments to small grids with less than 200K vertices.

The condition number of inhomogenous Laplacian matrices is often considerably higher than their homogeneous counterparts; clusters of strongly-connected variables that are weakly connected to the rest of the system introduce very weak modes, known as approximate zero modes, which increase the condition number [Trottenberg et al. 2001]. Spatially homogeneous solvers, such as GMG, fail to capture these spatially irregular modes and do not perform effectively on such problems.

The algebraic multigrid method (AMG) [Brandt 1986; Brandt 2001] generalizes its geometric counterpart and has a better ability to isolate and rescale the weak modes in spatially varying matrices. Adaptive coarse-grid selection plays a key role in AMG's success in capturing these modes, but at the same time, it leads to a growth in the bandwidth of non-zero matrix elements at coarse levels. The aggregation-based AMG [Vaněk 1995] and its smoothed version [Vaněk et al. 1996] limit the number of non-zero elements in the prolongation matrices that relate successive levels. This avoids the growth in the matrix bandwidth but also lowers its preconditioning abilities and increases the number of required cycles. The lean AMG solver [Livne and Brandt 2011] also falls into this category but offers a more sophisticated agglomeration rule as well as a correction step that improves the representation of weak modes at coarser levels. The use of fixed coarsening is another way to avoid growth in the matrix bandwidth [Zeeuw 1990; Moulton et al. 1998]. A hierarchical basis analogue of the AMG that also employs regular grid selection is described [Szeliski 2006]. A careful comparison, however, shows that methods that use adaptive grid selection offer better overall performance on highly irregular problems [Krishnan and Szeliski 2011]. The recently introduced Bootstrap AMG method [Brandt et al. 2011] automates the process of creating prolongation matrices using smoothing iterations over low-energy vectors.

Recent work in theoretical computer science has led to the development of preconditioners for graph-Laplacian matrices. A general umbrella term for these methods is *combinatorial preconditioning* [Spielman 2010]. The key is to construct a sparser approximation of the original matrix that is easy to invert and to then use it as a preconditioning matrix. For example, Vaidya [1990] suggested preconditioning with the Laplacian of a maximum spanning tree derived from the graph of the original Laplacian. This construction, however, does not offer an attractive bound on the resulting condition number. Boman and Hendrickson [2001] showed that better bounds are attained using low-stretch spanning trees. However, such constructions require $O(n \log n)$ operations and still do not guarantee that the resulting condition number is independent of n.

Vaidya [1990] also suggests improving the preconditioning by adding $O(n)$ edges to the spanning tree. This construction, which is known as an ultra-sparsifier, solves sparse Laplacian matrices in $O(n \log^{15} n)$ time and is constructed in nearly linear time [Spielman and Teng 2006]. The fastest known ultra-sparsifier based solver is described in [Koutis et al. 2010] and runs in $O(n \log^2 n (\log \log n)^2)$ time. Finally, Koutis et al. [2011a] use the notion of conductance from the support theory of graphs to derive an aggregation-based AMG method. Much progress has recently been made in the theory [Koutis et al. 2011b; Kelner et al. 2013], although practical solvers have yet to emerge from these efforts.

There are also specialized methods that work well in some applications normally solved using Laplacian matrices. The edge-avoiding wavelets in [Fattal 2009] offer fast running times for edge-preserving interpolation and tone mapping. However, unlike our approach, they consist of a regular sampling strategy and produce results of limited accuracy. Edit propagation using KD-trees in [Xu et al. 2009] is another example. Our solution applies to a larger family of Laplacian matrices, defined over arbitrary domains and boundary conditions, and is therefore applicable to a wider variety of applications.

3 Mathematical Background

In this section, we review Laplacian matrices and their connection to quadratic regularization problems. We use bold letters to denote vectors, e.g., $\mathbf{x} = (x_1, ..., x_n) \in \mathbb{R}^n$, capital letters to denote matrices, and calligraphic letters for sets. We use n as the number of variables and denote the set of indices by $\mathcal{I} = \{1..n\}$.

3.1 Laplacian Matrices

Laplacian matrices result from minimizing objective functions of the form

$$F(\mathbf{x}) = \sum_{i \in \mathcal{I}} \left[u_i (x_i - y_i)^2 + \sum_{j \in \mathcal{N}_i} w_{ij} (x_i - x_j - z_{ij})^2 \right]. \quad (1)$$

The first sum contains *data terms* that measure the proximity of $\mathbf{x}$ to a given input data vector $\mathbf{y}$. The second sum contains *smoothness terms* that measure the derivatives (pairwise differences) between every variable x_i and its neighbors x_j, $j \in \mathcal{N}_i$, with respect to (potentially zero) input derivatives z_{ij}. As we describe later, in the applications we are interested in, each set of neighbors $\mathcal{N}_i$ consists of a small number of variables that are geometrically close to x_i. Typical choices in the case of two-dimensional regular arrays of pixels are the four- or eight-nearest pixels. The weights u_i and w_{ij} define the cost for deviating from the data and smoothness objectives respectively and are non-negative. The problem becomes spatially homogeneous when u_i and w_{ij} are constant and is considered spatially inhomogeneous otherwise.

The objective $F(\mathbf{x})$ can be expressed in matrix-vector form as

$$F(\mathbf{x}) = (\mathbf{x} - \mathbf{y})^\top U (\mathbf{x} - \mathbf{y}) + (D\mathbf{x} - \mathbf{z})^\top W (D\mathbf{x} - \mathbf{z}), \quad (2)$$

where the matrix D is the discrete derivative operator whose rows correspond to pairs of neighboring variables i and $j \in \mathcal{N}_i$. The weights matrix W is diagonal and contains w_{ij} on the row that corresponds to the interaction between the i-th and j-th elements. The data weights matrix U is an n-by-n diagonal matrix with $U_{ii} = u_i$.

The minimum of this discrete quadratic form is obtained by setting $dF/d\mathbf{x} = \mathbf{0}$, which amounts to solving the following linear system

$$L\mathbf{x} = U\mathbf{y} + D^\top W \mathbf{z}, \quad (3)$$

where L is the discrete Laplacian matrix given by

$$L = U + D^\top W D. \quad (4)$$

The matrix L is, by construction, symmetric and positive semi-definite since $x^\top L x = x^\top U x + x^\top D^\top W D x = x^\top U x + (Dx)^\top W (Dx)$ and both U and W are diagonal with non-negative values. The off-diagonal elements of L are all non-positive and given by $L_{ij} = -w_{ij}$. The diagonal entries are given by $L_{ii} = u_i + \sum_{j \in \mathcal{N}_i} w_{ij}$ and are hence non-negative. These Laplacian matrices are associated with a graph whose vertices are the variables i and whose edges are weighted by w_{ij}. The data terms u_i can be considered as weights of edges connecting the vertices with a set of auxiliary variables.

The solver we describe in this paper applies for the general family of Laplacian matrices described by Eq. 1. This family includes a large portion of Laplacian matrices used in computer graphics and computational photography applications. For example. the image colorization of Levin et al. [2004] uses $u_i = 1$ at pixels i containing user input colors y_i and $u_i = 0$ elsewhere. The weights w_{ij} depend inversely on the difference between the i and j pixels gray-level values. The reference gradient field is set to zero, $z_{ij} = 0$. The weighted least squares edge-preserving smoothing of Farbman et al. [2008] uses a similar definition for the smoothness weights but sets *all* data terms $u_i = 1$ and provides the input image as the data $\mathbf{y}$. The dynamic range compression algorithm of Fattal et al. [2002] and Poisson blending of Perez et al. [2003] use $u_i = 0$ and set z_{ij} to the manipulated gradient field being integrated. This problem is homogeneous, i.e., $w_{ij} = 1$.

In 3D geometry processing applications, the Laplacian occurs in such problems as mesh segmentation [Liu and Zhang 2007] and geodesic distance computation [Crane et al. 2012]. The Laplacian can either be homogeneous or based on the local curvature or geometry (inter-vertex distances and angles) in the 3D mesh. Some of these Laplacian variants, e.g., the co-tangent Laplacian [Crane et al. 2012], may result in negative weights w_{ij}.

Given that most AMG methods, as well as our method, are applicable to Laplacian matrices with strictly negative off-diagonal (*M-matrices*), we restrict the formulation of 3D processing problems this class of matrices. In many cases, defining such matrices while preserving the same qualitative nature of the operator is possible. For example, Crane et al. [2012] describe their algorithm for arbitrary discretization of the Laplacian operator and then provide two alternatives. In Section 5, we show a particular discretization that results in an M-matrix and at the same time achieves the desired result.

In computer vision applications, problems of the form Eq. 1 arise from Gauss-Markov MRF models. The data terms are known as unary potentials and the smoothness terms as binary (or pairwise) potentials. Example application of such models is optical flow regularization [Baker et al. 2011].

Energy Function. The Laplacian matrix L assigns an *energy* value to each vector $\mathbf{x}$, defined by the Rayleigh quotient

$$E_L(\mathbf{x}) = (\mathbf{x}^\top L \mathbf{x}) / (\mathbf{x}^\top \mathbf{x}). \quad (5)$$

The energy values are always non-negative due to the positive semi-definiteness of L. The eigenvalues of L are the energy values assigned to their corresponding eigenvectors, since if $L\mathbf{x} = \lambda\mathbf{x}$, $(\mathbf{x}^\top L\mathbf{x})/(\mathbf{x}^\top\mathbf{x}) = (\mathbf{x}^\top\lambda\mathbf{x})/(\mathbf{x}^\top\mathbf{x}) = \lambda$.

The input data values y_i and derivatives z_{ij} contribute only to the right-hand side of Eq. 3 and hence do not affect the properties of L. The energy function E_L is therefore an intrinsic function of the Laplacian matrix and it is closely related to the solvability of Eq. 3.

3.2 Matrix preconditioning

The condition number of a symmetric positive definite matrix is defined as

$$\kappa(L) = \frac{\lambda_{\max}}{\lambda_{\min}} = \frac{E_L(\mathbf{x}_{\max})}{E_L(\mathbf{x}_{\min})} = \frac{\max_{\mathbf{x}} E_L(\mathbf{x})}{\min_{\mathbf{x}} E_L(\mathbf{x})}, \tag{6}$$

where $\lambda_{\max}$ and $\lambda_{\min}$ are the maximal and minimal non-zero eigenvalues of L and $\mathbf{x}_{\max}$ and $\mathbf{x}_{\min}$ are their corresponding eigenvectors. The last equality results from the fact that eigenvalues of L are the extremal values of E_L.

The number of iterations it takes iterative solvers to achieve a certain accuracy depends on $\kappa(L)$: $O(\kappa)$ iterations for Jacobi and Gauss-Seidel and $O(\sqrt{\kappa})$ for conjugate gradients [Saad 2003]. In the case of homogeneous matrices, $\kappa(L) = O(l^2)$, where l is the domain's length, $l \propto \sqrt[d]{n}$, and d is the spatial dimension ($d = 2$ for images). Inhomogenous Laplacians often contain approximate zero modes [Trottenberg et al. 2001], which, according to Eq. 6, lead to very high values of $\kappa(L)$.

The condition number of a matrix can be reduced by converting the linear system, $L\mathbf{x} = \mathbf{b}$, into a related problem by multiplying it with a *preconditioning matrix* Q^{-1} such that the condition number of $Q^{-1}L$ is significantly lower than that of L. In order to achieve effective preconditioning, the matrix Q^{-1} must meet several additional requirements. Iterative linear solvers such as Jacobi, Gauss-Seidel and CG consist of repeated matrix-vector multiplications with $Q^{-1}L$, which is typically computed in succession, with L and then with Q^{-1}. Therefore, multiplying a vector with Q^{-1} must not be significantly more expensive than multiplication with L. For example, in the case of sparse Laplacian matrices, these operations must cost $O(n)$. Note that this does not require Q^{-1} to be sparse; there just needs to be an efficient procedure for multiplying it with vectors, which is the case with various hierarchical schemes (Section 3.3). Another important aspect is that $Q^{-1}L$ must meet solver-specific requirements, for example maintain the positive definiteness of L in case of the Preconditioned Conjugate Gradient (PCG) method.

A fundamental theorem in combinatorial matrix preconditioning shows that the effectiveness of Q^{-1} as a preconditioning matrix depends on how well E_Q approximates E_L. More specifically, given $a, b > 0$ such that

$$\forall\mathbf{x},\ \ aE_Q(\mathbf{x}) \le E_L(\mathbf{x}) \le bE_Q(\mathbf{x}), \tag{7}$$

$\kappa(Q^{-1}L) \le b/a$ [Boman and Hendrickson 2003, Prop. 2.4]. We make use of this rule in our derivations.

3.3 Hierarchical preconditioning

Hierarchical preconditioners are constructed by formulating a smaller version of the original problem and using its solution as the approximate inverse for the original problem [Trottenberg et al. 2001; Saad 2003; Szeliski 2006]. Geometric multigrid techniques use a regular set of decimation rules (e.g., full octave decimation) and standard interpolation operators as their basis, and are particularly well suited for homogeneous problems [Trottenberg et al. 2001]. Algebraic multigrid techniques use both adaptive coarsening strategies and adaptive interpolation weights, which make them better suited for inhomogeneous problems. Unfortunately, the elimination of fine-level variables results in an increase in the matrix bandwidth, or, alternatively, a sub-exponential decrease in the matrix size (Figure 2). In order to reduce the bandwidth growth (fill-in) in coarser problems, AMG techniques drop small off-diagonal terms.

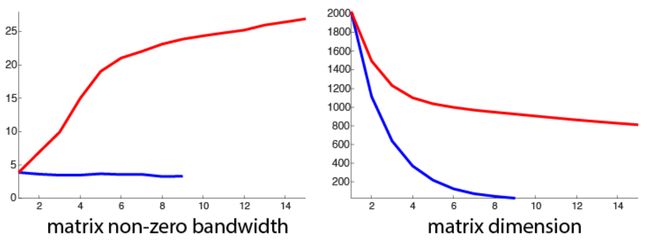

Figure 2: *Effect of Sparsification. The left graph shows how our sparsification step maintains an average bandwidth below 4 (blue curve) in all the hierarchy levels. Without the sparsification the (red curve) bandwidth grows until it reaches the matrix dimension. The right graph shows the scheme's ability to achieve an exponential reduction in the number of variables (blue curve), where without sparsification, fewer variables can be marked as fine and get eliminated (red curve). We used an inhomogeneous two-dimensional regular five-point Laplacian matrix in this example. Note that the sparsified hierarchy has fewer levels, as it more quickly reaches target coarse level size.*

Adaptive basis functions [Szeliski 2006; Krishnan and Szeliski 2011] perform the sparsification (element elimination) *before* creating the coarser-level (smaller) problem, which allows them to *compensate* for these eliminations by increasing nearby connections. In Section 4.2, we derive an alternative compensation strategy that is based on an analysis of the spaces spanned by the fine and coarse variables and hence produces a better preconditioner (slower growth in condition number).

Once the Laplacian matrix has been sparsified and compensated, we divide the variables into coarse $\mathcal{C}$ and fine $\mathcal{F}$ sets such that the coarse variables encode the low-frequency modes in the solutions, i.e., the modes that are not well solved by local smoothing or relaxation, and the fine level variables have no remaining connections between each other. The exact method for selecting these variables, which in our new algorithm is interleaved with the sparsification step, is described in Section 4.3.

Once the selection of coarse and fine variables has been done, we rearrange the indices in $\mathcal{I}$ such that $\mathcal{C}$ come first followed by $\mathcal{F}$. Under this permutation, the matrix L becomes

$$L = \begin{bmatrix} L_{\mathcal{CC}} & L_{\mathcal{CF}} \\ L_{\mathcal{FC}} & L_{\mathcal{FF}} \end{bmatrix}, \tag{8}$$

where $L_{\mathcal{CC}}$ contains only the connections between the coarse variables, $L_{\mathcal{FC}} = L_{\mathcal{CF}}^\top$ contains the connections between coarse and fine variables, and $L_{\mathcal{FF}}$ among the fine variables. Since the fine variables are uncoupled, $L_{\mathcal{FF}}$ is diagonal. The elimination of the fine variables is obtained by computing the Schur complement using the transformation matrix

$$P = \begin{bmatrix} I_{\mathcal{CC}} & 0 \\ -L_{\mathcal{FF}}^{-1}L_{\mathcal{FC}} & I_{\mathcal{FF}} \end{bmatrix}, \tag{9}$$

which is applied to L on both sides,

$$P^\top LP = \begin{bmatrix} L_{\mathcal{CC}} - L_{\mathcal{CF}}L_{\mathcal{FF}}^{-1}L_{\mathcal{FC}} & 0 \\ 0^\top & L_{\mathcal{FF}} \end{bmatrix}. \tag{10}$$

Note that the matrix $L_{\mathcal{FF}}^{-1}$ is an inverse of a diagonal matrix and is trivial to compute, and that $S = L_{\mathcal{FF}}^{-1} L_{\mathcal{FC}}$ is the *interpolation* (or *prolongation*) matrix used in hierarchically preconditioned conjugate gradient [Krishnan and Szeliski 2011, Algorithm 1]. Since L is a sparse matrix, so is P, and this elimination step is computed in linear time.

The resulting two-block matrix in Eq. 10 describes two systems that are solved independently. The fine system, $L_{\mathcal{FF}}$, is diagonal and solved exactly. In Appendix A (Lemma 1) we show that the coarser system, $L_{\mathcal{CC}} - L_{\mathcal{CF}} L_{\mathcal{FF}}^{-1} L_{\mathcal{FC}}$, is a Laplacian matrix like the original matrix L. This allows us to apply the same procedure again over this system, compute a Schur matrix, and repeat this process recursively over the resulting coarse system. At each level, we do not operate on the fine variables eliminated in the previous levels and hence obtain a sequence of of n-by-n transfer matrices $P^1, P^2, ..$ that contain the current level's prolongation matrix as the top-left block and are identity over the remaining coordinates (corresponding to the fine variables of all previous levels). The recursive elimination process is terminated once the number of coarse variables falls below some threshold (e.g., 1024, although changing this to 512 or 2048 does not affect our performance), since the direct solution of such small systems using Cholesky decomposition takes a negligible amount of time compared to finer-level operations.

4 Sparsification and Coloring

In the previous section, we presented a general framework encompassing previously developed hierarchical preconditioning algorithms for the solution of sparse Poisson equations. How does our new approach, *Hierarchical Sparsify and Compensate*, differ from these other techniques?

The key difference is that rather than using a fixed sparsification and coarsening scheme, as in ABF [Krishnan and Szeliski 2011], we adaptively select which edges to sparsify and which nodes to select as coarse and fine variables. (This process is often called *coloring* [Trottenberg et al. 2001].) Our extensions allows the ABF algorithm, which already performed well on a wide variety of computational photography applications, to now also perform well on more inhomogeneous problems as well as unstructured meshes. Compared to AMG [Brandt 1986] and CMG [Koutis et al. 2011a], our technique does a better job of creating a hierarchy of smaller approximate problems (because of our use of adaptive interpolants and compensations steps), and hence has better convergence and run-time properties.

Our sparsification and coloring algorithm tries to simultaneously satisfy two somewhat conflicting goals. The first is to produce a large number of disconnected fine variables, since the smaller we can make the coarse system, the easier it is to solve or invert. The second is to only sparsify connections that are already quite weak compared to the neighboring compensation paths. We solve this tension by developing a greedy algorithm that visits fine and unmarked variables and searches for connections to other variables that can be sparsified, which then enables these variables to also be colored as fine. The exact algorithm is described in Section 4.3. Before we get there, however, we first describe how to find good connections to sparsify and how to compensate for such operations.

4.1 Matrix Sparsification

We now explain how we avoid the growth in matrix bandwidth and increase the number of fine variables by carefully dropping off-diagonal elements at every level of the hierarchy, before executing the elimination. Unfortunately, eliminating a large number of off-diagonal elements introduces approximation errors, meaning that it cannot be used for computing the exact inverse of L. As we explained earlier, we use the approximated inverse for matrix preconditioning and accelerating different iterative solvers. In order to achieve low condition numbers, we need to keep the sparsified matrix 'close' to the original Laplacian. In this section, we explain how we carefully chose the off-diagonals to drop from L to produce $\tilde{L}$ based on its effect on the condition number $\kappa(L\tilde{L}^{-1})$. Let us first define more precisely what we mean by dropping connections in the matrix.

There are two properties that we need to preserve when sparsifying a matrix. We need to preserve its nature as a Laplacian matrix so that the following levels of the hierarchy are constructed in the same manner. To preserve its symmetry when setting L_{ij} to zero, we also set L_{ji} to zero. However this alone does not preserve the type of the problem, for example, in the case of purely regularization dataless problem, where $u_i = 0$, the row and column sums of L are zero. Setting off-diagonal elements to zero results in a matrix with positive row and column sums which corresponds to a problem *with* data terms.

We avoid this problem by replacing L_{ii} by $L_{ii} + L_{ij}$ and L_{jj} by $L_{jj} + L_{ij}$ when we drop the i, j element. From now on, we interchangeably say that we drop L_{ij} or the weight w_{ij} and in both cases refer to the same procedure where the diagonal elements are modified.

In order to decide which off-diagonal elements are dropped, we refer to the condition described in Section 3.2 which guarantees $\kappa(L\tilde{L}^{-1}) \leq b/a$, given lower and upper energy bounds b and a obeying Eq. 7. If the energies assigned by the sparsified matrix $\tilde{L}$ are close enough to those assigned by the original matrix L, the sparsified matrix will provide a good preconditioning. As we discussed in Section 3.1, the energy $E_L(\mathbf{x})$ (Eq. 5) does not depend on the norm of the vector $\|\mathbf{x}\|$ and hence we can restrict the discussion to unit vectors $\mathbf{x}$, in which case the energy function coincides with the objective function in Eq. 2. Dropping a weight w_{ij} means that the energy function ceases to account for changes between x_i and x_j. However, in case there is a third coordinate k such that both $w_{ik} > 0$ and $w_{jk} > 0$, any difference between x_i and x_j must be accompanied by a difference between x_i and x_k or between x_j and x_k. Thus, for example if $w_{ij} \ll w_{ik}, w_{jk}$, the penalty term $w_{ij}(x_i - x_j)^2$ is dominated and negligible compared to either $w_{ik}(x_i - x_k)^2$ or $w_{jk}(x_i - x_k)^2$. In Appendix A, Lemma 2, we prove that in such cases of triangular connectivity, when dropping the weakest weight, we get $a = 1$ and $b \leq 3$ and hence $\kappa(L\tilde{L}^{-1}) \leq 3$.

In view of these observations and the fact that we need to remove many connections to ensure rapid coarsening, we apply the following sparsification procedure. At each level of the hierarchy, we search for triplets of variables that form a triangle and remove the weakest edge in each triangle. This procedure is applied by scanning the matrix elements until no more triangles are found.

Figure 2 shows the non-zero matrix bandwidth and the matrix dimension at each hierarchy level with and without applying sparsification. In this experiment our strategy selects and eliminates between $40\% - 50\%$ of the variables at each level and avoids a growth in bandwidth. These actions are highly local, at the scale of three coupled variables, and hence run in time linear in the number of variables and connections in the matrix. As we explain next, we further improve the energy preservation by adjusting the two remaining weights of every sparsified triangle.

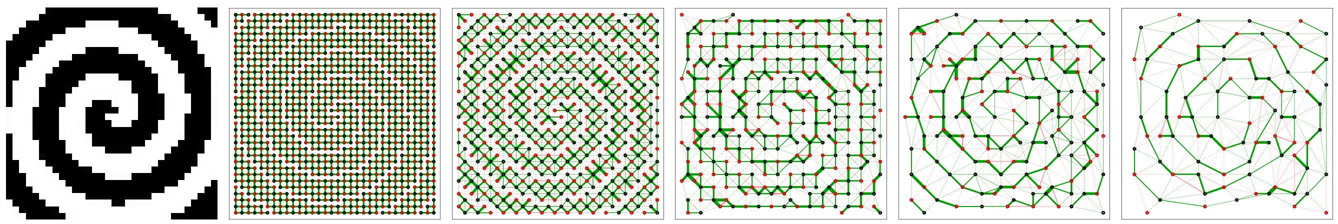
Figure 3: *Scheme Progression. We show the connectivity graphs at each level for a highly discontinuous EPS problem, starting from a four-point Laplacian matrix defined by the left image. Red circles show fine (eliminated) variables and black show the coarse ones. Green lines show the connections kept between variables and red show the ones removed during sparsification. The thickness of the lines indicates the corresponding weights strengths. Please see the supplementary results for additional examples.*

4.2 Compensation Step

When computing a single level of the hierarchy, low condition numbers are achieved even though multiple overlapping triangle are processed and the theoretical bound becomes $\kappa \le 3^r$ where r is a bound on the number of times the same edge participates in a triangle. However, when applying this strategy at every hierarchy level, the sparsification errors of the different levels accumulate. For example, there could be a vector $\mathbf{x}$ whose energy drops by a factor of 3 due to the sparsification that takes place at each level. Thus, after the expected number of $m = \log(n)$ hierarchy levels, its energy drops by a factor of $3^{\log(n)} = O(n)$, meaning that the condition number $\kappa(LQ^{-1}) = O(n)$, where Q represents the approximate operator obtained by the entire hierarchy. As we discussed earlier, this dependency of κ on n is observed in non-preconditioned Laplacian matrices.

The key observation that allows us to cope with this shortcoming is the fact that at each hierarchy level, about half the variables are eliminated and these variables define a linear subspace that *does not* experience the sparsification taking place in the following levels. This means that there is a hierarchy of subspaces that undergo a different number of sparsification steps. Similarly to Szeliski [2006], we alter the remaining matrix elements in order to compensate for the loss of those elements dropped during sparsification. Our compensation formula, however, is based on an analysis that characterizes vectors based on the number of sparsification steps they undergo. Thus, we both extend Szeliski's approach to an adaptive coarsening schemes as well as improve it by establishing the sense at which $\tilde{L}$ should best approximates L.

In Appendix B, we show that the linear subspace spanned by the columns of $P^1P^2..P^l$, which correspond to the fine variables eliminated in the l-th hierarchy level, is affected by the sparsification steps in the first l hierarchy levels. We further show that this subspace is characterized by having low energy values, since the elimination steps that define it correspond to the minimization of the energy over the eliminated variables. In view of this relation between the number of sparsification steps and low-energy vectors, whenever a triangle is sparsified, we compensate by adjusting its two remaining weights such that the triangle's energy contribution remains unchanged over low-energy vectors.

Assuming that w_{ij} is the weakest weight of a triangle which is set to zero, our compensation procedure consists of finding the sparsified matrix weights $\tilde{w}_{ik}$ and $\tilde{w}_{jk}$ of $\tilde{L}$ that satisfy

$$\begin{aligned} E_L(\mathbf{u}) &= w_{ij}(u_i-u_j)^2+w_{ik}(u_i-u_k)^2+w_{jk}(u_j-u_k)^2+E' \\ &= \tilde{w}_{ik}(u_i-u_k)^2+\tilde{w}_{jk}(u_j-u_k)^2+E' = E_{\tilde{L}}(\mathbf{u}), \end{aligned} \tag{11}$$

where $\mathbf{u}$ is one of two low-energy vectors. (The same equation is defined over the second vector $\mathbf{v}$.) The scalar E' accounts for the total cost of the rest of the energy terms in Eq. 2 that are unrelated to

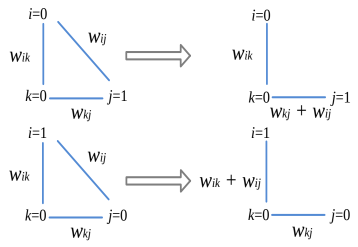

Figure 4: *Sparsification and Compensation in a Triangle. Left shows how the two vectors* $\mathbf{u}$ *and* $\mathbf{v}$ *are aligned with respect to the triangle and at the right we see the sparsified and compensated triangle weights which preserve the energy of* $\mathbf{u}$ *and* $\mathbf{v}$.

w_{ij}, w_{ik}, and w_{jk} and can therefore be omitted from Eq. 11. Since Eq. 11 and its counterpart for $\mathbf{v}$ consist only of differences between the variables, they are invariant to the addition of any constant to u_i, u_j, u_k and v_i, v_j, v_k.

These invariants leave us with only a single degree of freedom in $\mathbf{u}$ and $\mathbf{v}$, which we need to determine in order to model the shape of low-energy vectors at the coordinates i, j and k. We use the following rationale to obtain these two model low-energy vectors. Consider the scenario where w_{ik} is much greater than w_{jk} (and $w_{jk} > w_{ij}$). In this case, differences $(u_i - u_k)^2$ lead to a higher energy than $(u_i - u_j)^2$ and hence the former is expected to be much lower in the case of low-energy $\mathbf{u}$. Low-energy vectors will therefore attain similar values at i and k and acquire different values at u_j. Given the invariants we discussed, this situation can be modeled by choosing $[u_i, u_j, u_k] = [0, 1, 0]$. The reverse scenario where the j-th and k-th variables are strongly coupled and both are weakly connected to the i-th, can be modeled by the second vector $[v_i, v_j, v_k] = [1, 0, 0]$. These scenarios are often encountered in the two-dimensional computer graphics applications that we are interested in, where the weights are determined by pixel differences of some reference image. The two constellations result from strong edges in the reference image that pass through the i, j, k triangle and separate its pixels into two sets of distinct colors.

In this choice of $\mathbf{u}$ and $\mathbf{v}$, we dismiss a third scenario, where $w_{ik} \approx w_{jk}$ and both of them are much higher than w_{ij}. This scenario is less common in the applications that we are interested in since there is no assignment of pixel values that will lead to such weights. Furthermore, in such situations, the cost of $w_{ij}(u_i - u_j)^2$ will be penalized by either $w_{ik}(u_i - u_k)^2$ or $w_{jk}(u_j - u_k)^2$ since both w_{ik} and w_{jk} are assumed large. Lastly, the low-energy vectors are expected to be close to uniform at such triangle and, as we shall see below, we handle these situations properly with our current choice of $\mathbf{u}$ and $\mathbf{v}$.

Solving Eq. 11 for $\mathbf{u}$ and its analog for $\mathbf{v}$ boils down to simple and efficient update formulas for $\tilde{w}_{ik}$ and $\tilde{w}_{jk}$. The energy of

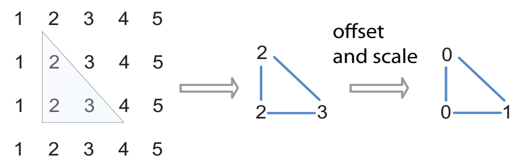

Figure 5: *Modeling Fourier Modes. Illustration shows how our choice of* $\mathbf{u}$ *models locally the shape of horizontal low-frequency Fourier modes. The transposed diagram applies for* $\mathbf{v}$.

$[u_i, u_j, u_k] = [0, 1, 0]$ before sparsification is given by $w_{ij} + w_{jk}$. After dropping w_{ij}, the energy becomes $\tilde{w}_{ik}$ and hence to preserve this energy, we need

$$\tilde{w}_{ik} = w_{ik} + w_{ij}, \tag{12}$$

and similarly, for $[v_i, v_j, v_k] = [1, 0, 0]$, we get

$$\tilde{w}_{jk} = w_{jk} + w_{ij}. \tag{13}$$

This action is illustrated in Figure 4.

The AMG method in [Livne and Brandt 2011] also corrects the matrices with respect to low-energy vectors. However, these vectors are computed globally by applying an iterative method to reduce the energy of random vectors. In order to achieve very low-energy vectors as our analysis suggests, many such iterations are needed for larger systems. Our method avoids this additional cost through its simple local operation.

Data Connections. The derivation of both the sparsification and compensation accounts only for the smoothness terms in Eq. 2. Non-zero data terms, u_i in Eq. 1 can be viewed as weighting differences with auxiliary variables of fixed values, namely y_i in Eq. 1. Since each of these auxiliary variables is connected to only a *single* variable x_i, it never participates in a triangle. Hence, these connections need not be involved in any sparsification and compensation steps. In practice, this means that the sparsification and compensation steps are applied on L after removing its excess diagonal values so that its rows and columns become zero-sum. Once these steps have been applied, the excess diagonal values are added back to $\tilde{L}$.

Homogeneous Systems. In the case of homogeneous Laplacian matrices, such as the spatially invariant Poisson used in [Fattal et al. 2002; Pérez et al. 2003; Sun et al. 2004], the eigenvectors of L are the Fourier modes [Trottenberg et al. 2001]. The two lowest energy (non-constant) eigenvectors are low-frequency horizontal and vertical sinusoids. From the perspective of three variables forming a triangle in the grid, these functions appear as two linear ramps. As shown in Figure 5, our compensation function preserves the energy in these Fourier modes.

If we make sure that our selection of coarse and fine variables as we drop matrix elements maintains spatial homogeneity, our construction boils down to a geometrical multigrid method for homogenous problems.

Our scheme is also important in the case of inhomogeneous problems, which often contain large homogeneous regions, e.g., due to nearly constant regions in the reference image. In the next section we explain how we make sure our scheme maintains spatial homogeneity in such regions.

Table 1 shows the effect that careful compensation has on condition numbers for homogenous Poisson matrices of increasing size. We compare our method with a sparsification-only version, where we drop the weakest edges in each triangle but do not perform any compensation. Running our method with compensation has the property of h-independence, while without it, the condition number increases with the problem size.

Algorithm 1 Sparsify and color

input: Laplacian matrix L and optionally the coordinates of the mesh vertices;
output: Fine $\mathcal{F}$ and coarse $\mathcal{C}$ variable indices and the sparsified matrix $\tilde{L}$, which, according to Eq. 9, determine the prolongation matrix S.

```
1. Remove the excess diagonals from $L$ and store them in $E$
2. Set all vertices as unmarked except for the first one, which we mark as fine
3. Flag variables as geometric or non-geometric
4. Cycle through vertices $i$ in the matrix
     if $i$ is marked as coarse, skip to the next vertex
     for each triangle in which vertex $i$ participates
       if all three vertices are flagged as geometric
         Cut longest edge in the triangle and compensate
         Mark the vertices in this triangle as coarse or fine
             according to the global red/black settings
       else
         Cut the weakest edge in the triangle and compensate
         If vertices on weakest edge are unmarked, mark them as fine
       endif
       Set unmarked neighbors of $i$ as coarse
     endfor
5. Set unmarked vertices that have fine neighbors to coarse, else fine
6. For any fine-fine connections, set one of the endpoints to coarse
7. Set any coarse variables connected only to coarse variables as fine.
8. Add the excess diagonal in $E$ back to $L$ to produce $\tilde{L}$
```

4.3 Coloring algorithm

With our sparsification and compensation steps in place, we need to decide how to color the original variables in the Laplacian as coarse $\mathcal{C}$ or $\mathcal{F}$ and how to interleave the process of sparsification/compensation into this procedure.

The algorithm we have developed is based on the observation that we want to produce a large number of fine variables (to obtain smaller problems), but that it is best to cut weak edges to limit the growth in condition number.

Our algorithm (Algorithm 1) visits nodes in their lexicographic order (how they were given in the original problems), ignoring nodes already marked as coarse, since we do not need to cut connections between these nodes and their neighbors. For each unmarked or fine node, we search its neighbors and neighbors' neighbors, looking for triangles where an edge emanating from the current node can be cut because it is no larger than the other edges in the triangle. When we find such an edge, we eliminate it, compensate the other two edges in the triangle, and mark both endpoints of the eliminated edge as fine. At the end of this procedure, we label any unmarked variables as fine or coarse (depending on their neighbors), fix fine-fine connections by marking one endpoint as coarse, and check for any coarse variables surrounded by coarse variables, which can be set to fine.

In homogeneous regions where we have additional geometric information, i.e., the (x, y) locations of variables, we modify the above edge selection and coloring procedure to produce a regular red/black coloring. To determine if a variable is in a homogenous region, we take the difference between the strongest and weakest connections of the variable and divide this difference by the strongest connection. We then find the mean of these ratios over all variables. Any variable whose ratio is below or equal to the mean ratio is marked as geometric. When sparsifying triangles, if all three vertices are marked as geometric, we cut the longest edge in the triangle based on its geometric distance. We also mark vertices according to a global red/black coarsening scheme, so that fine variables are disconnected from each other. These steps ensure that our scheme reduces to geometric multigrid for smoothly varying problems. In this approach, homogeneous regions in inhomogeneous problems are also processed using geometric coarsening.

Problem Size	Original CN	CN with comp.	CN with no comp.
1024	423	1.2	1.6
4096	1676.7	1.2	2.4
16384	6674	1.3	3.7
65536	26629	1.4	5.6
262144	106380	1.5	8.6
1048576	452500	1.5	13

Table 1: *Comparison of condition numbers with and without compensation. First columns gives the dimension of the homogenous Laplacian. Second column gives the unpreconditioned condition number. Third and fourth columns give the preconditioned condition number with and without compensation, respectively.*

Once our hierarchical preconditioner has been constructed, we use it in combination with conjugate gradients to precondition the solution of $L\mathbf{x} = \mathbf{b}$ using the multilevel V-cycle described in [Krishnan and Szeliski 2011, Algorithm 1]. In the experiments we report in this paper, we use a single step of post-smoothing, $\nu^{post} = 1$, and no pre-smoothing, $\nu^{pre} = 0$ and a V-cycle ($\gamma = 1$) since this resulted, on average, in the fastest running algorithms. For our smoothing algorithm, we use lexicographic-order Gauss-Seidel.

4.4 Updating the Preconditioner for Diagonal Shifts

In a number of applications [Farbman et al. 2008] and [Crane et al. 2012], diagonally shifted versions of the original laplacian L are considered. These perturbations result in Laplacians of the form $L + tI$, where I is the identity matrix and t is a non-negative scalar. For example, in edge-preserving decomposition [Farbman et al. 2008], these perturbations give rise to a multi-scale decomposition—larger values of t correspond to coarser scale versions of the original image. In smoothed geodesic computation on meshes [Crane et al. 2012], the scalar t controls the smoothness of geodesic distances on a mesh. The offset t is often not known in advance and needs experimental determination.

In order to avoid the expensive steps of sparsification, compensation, and coloring, we developed the following approximate method for computing a good preconditioner for $L + tI$ given our already constructed hierarchical preconditioner for L. We define the *excess diagonal* at the finest level as $E^0 = tI$. The finest-level Laplacian is modified from L^0 to $L^0 + E^0$. The excess diagonal for the next coarser level is then computed as $E^1 = diag({S^1}^T diag(E^0))$, where S^1 is the interpolation matrix at the next level as defined in Algorithm 1. The operator $diag$ is analogous to MATLAB: it extracts the main diagonal from a matrix, or embeds a vector into the main diagonal of a matrix. This results in E^1 being a diagonal matrix. The Laplacian at level 1 is then changed from L^1 to $L^1 + E^1$, where only the diagonal elements of L^1 are modified. This process is continued all the way through the hierarchy, and the modified Cholesky decomposition of the coarsest level Laplacian is also recomputed accordingly.

This update process does not require any new sparsification, compensation or Schur elimination of the coarse level matrices, and is hence extremely fast, typically an order of magnitude faster than recomputing the preconditioner. This updated preconditioner is also very accurate, usually only requiring only one extra CG iterations, as compared to using the recomputed hierarchy for $L + tI$. In Section 5, we give timings for an edge-preserving sharpening application on a multi-megapixel image.

We experimented with the same update scheme for the other preconditioning methods we compare against in Section 5. The resulting hierarchies proved very poor preconditioners.

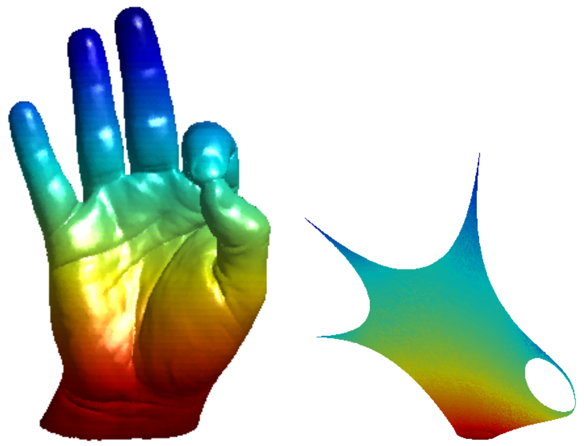

Figure 6: *Mesh Segmentation using Spectral Embedding: The 3D mesh is embedded into the 2D planar shape. Contour analysis is performed on this planar shape to determine segments in the original mesh. The fingers of the mesh are mapped to the spiky parts of the planar graph. Our solver is used to compute the lowest eigenvectors of the mesh Laplacian, to perform the spectral embedding.*

4.5 Efficient Multilevel Eigensolver

A number of applications such as mesh segmentation [Liu and Zhang 2007] and spectral matting [Levin et al. 2007] require the computation of a few lowest eigenvectors of a Laplacian. In mesh segmentation, a 3D mesh is spectrally projected onto the plane by projecting the mesh coordinates on the lowest two non-zero eigenvectors of the mesh Laplacian. Contour analysis is then performed on the projected planar shape (Figure 6). Two types of Laplacians are used in this application: a graph (homogenous) Laplacian and a geometric Laplacian designed to enhance concavity of the resulting planar shape.

Since our hierarchy explicitly preserves low energies, a natural strategy is to use a multi-scale approach to compute the lowest eigenvectors. First, we compute the lowest few eigenvectors at a coarse level. We then interpolate these eigenvectors to the finest level. This interpolation already gives us a very good approximation to the true lowest eigenvectors. To refine the vectors further, we perform a few iterations of block Davidson smoothing [Arbenz et al. 2003] at the finest level. Typically, less than a dozen iterations of block Davidson smoothing gives us a very accurate estimate with a relative error of 10^{-4}. This level of accuracy suffices for most graphics applications. The resulting multilevel eigensolver is twice as fast as MATLAB's built-in eigensolver for meshes with a million or more vertices, which uses the state of the art Lanczos methods to determine eigenvectors. In Section 5, we give timings for the eigensolver for meshes of different sizes. Kushnir et al. [2010] also de-

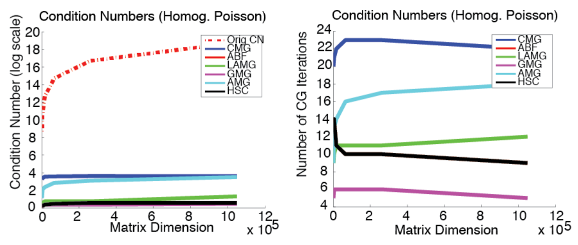

Figure 7: *Independence to system size. Left plot shows the condition number of the precondition systems achieved by various methods with respect to the matrix dimension n. Similarly, the right plot shows the number of CG iterations needed to achieve a fixed error or 10^{-4} with each method. Both the geometric multigrid method and our method show invariance to the scale.*

velop a multilevel eigensolver using algebraic multigrid methods. However, due to lack of available code, we are unable to compare with their scheme.

5 Results

In this section, we compare the performance of our preconditioning scheme against a number of other preconditioners, as well as MATLAB's direct solver, for a range of 2D and 3D problems. We embed the different preconditioners in the preconditioned conjugate gradient (PCG) method described in [Krishnan and Szeliski 2011, Algorithm 1] with $\nu^{pre} = 0$, $\nu^{post} = 1$, and $\gamma = 1$. Note that it is not essential to have a symmetric preconditioner for PCG to converge [Bouwmeester et al. 2012] and we choose to not have pre-smoothing since it does not help with convergence and adds computational complexity. We implement our solver in MATLAB/Mex. Our code is publicly available [1]. Please see the supplementary materials for additional visual results, e.g., the input and output images and meshes.

In our experiments, we measure the convergence of a solver for $Lx = b$ using two metrics: error with respect to the true solution computed by a direct solver, and relative residual, given by $\|Lx - b\|/\|b\|$. The relative residual is the measure used for terminating PCG iterations. For all experiments involving linear solvers, we use a target relative residual of 10^{-6}.

We compare the performance of our preconditioner against five other state of the art preconditioners. The first is combinatorial multigrid (CMG) [Koutis et al. 2011a]. This is an agglomerative multigrid method that clusters strongly coupled variables into a single coarse-level variable. The interpolation operators simply copy the value of the coarse variable into all fine variables that belong to the cluster. The second is another variant of this approach that takes the classic Ruge-Stuben AMG approach in [Brandt 1986; Trottenberg et al. 2001] and truncates the prolongation matrix to a single variable at every row (which makes it agglomerative as well) in order to avoid the growth in the matrix bandwidth. The third method is the lean algebraic multigrid (LAMG) [Livne and Brandt 2011], which was described in Section 2. The fourth is a standard geometric multigrid (GMG) method that employs first-order prolongation and restriction matrices [Trottenberg et al. 2001]. The fifth is the adaptive basis function (ABF-Pre7) method of [Krishnan and Szeliski 2011] that uses fixed coarsening. Except for AMG and GMG, we used code provided by the authors. For all these methods, we compare the reduction in error with respect to the number of iterations and floating-point operations (flops) and the decrease in relative residual with respect to iterations. We also report the wall-

[1]`http://www.cs.huji.ac.il/~raananf/projects/hsc/`

Problem	Size	Gauss-Seidel Its./Time	Jacobi Its./Time
HDR comp.	4.2M	10/6.6s	7/5.3s
Colorization	5.0M	3/4.8s	4/6.5s
EPD compress (3 scales)	4.2M	51/45.5	57/52.6s
EPD sharpen (5 scales)	4.2M	49/41.8s	55/42.4s

Table 4: *Comparing 2D problem solves with different smoothers: Gauss-Seidel and Jacobi. Both smoothers are used as post-smoothing only, the Jacobi smoother being used with a damping of* 0.8*. In each column we list the number of iterations (Its.) and wall-clock time in seconds to reach a relative residual of* 10^{-6}*. Neither smoother always outperforms the other. For the EPD problems, we report the total number of iterations and time over multiple scales. Setup times are the same for both smoothers and are not reported here.*

clock running times needed to achieve a sensible accuracy threshold. These latter (tabular) results include running time comparisons with MATLAB's direct solver, which is highly optimized.

5.1 2D Problems

We evaluate our algorithms on three kinds of 2D computational photography problems: homogeneous Poisson equations, non-homogeneous Poisson equations that arise in image colorization [Levin et al. 2004], and those that arise in EPD problems [Farbman et al. 2008].

Homogenous Poisson Problem. Homogenous Poisson matrices are used for various problems that require the integration of a manipulated gradient image field, e.g., tone mapping [Fattal et al. 2002] and Poisson blending [Pérez et al. 2003]. In Figure 7, we empirically verify h-independence we discussed in 4.2 by tone mapping the same image at various sizes. The plots show that our method, similarly to GMG, does not depend on the problem size. There are more efficient methods for solving this problem [Farbman et al. 2011]. However, ensuring that the h-independence property holds for our technique allows our method to scale well on problems with mixed coefficients that have large uniform regions within them.

Edge-Preserving Decomposition (EPD) and Image Colorization. In Figure 8, we compare the methods on the highly irregular matrices arising from the use of EPD for dynamic range compression and detail enhancement [Farbman et al. 2008]. Our method and CMG perform the best. Geometric methods such as GMG and ABF perform poorly because they use regular subsampling grids that fail to preserve thin regions at coarser levels. As described in Section 4.4, our preconditioner can be efficiently updated for problems such as Edge-Preserving decomposition, where a series of diagonally shifted Laplacians are generated.

In Table 2, we show the timing results for EPD compression and EPD sharpening for multiple such shifts. Neither CMG nor the direct solver have such an update, so their performance suffers compared to our solver. CMG is competitive with our solver for EPD-like problems if only a single solve is required. However, for multiple solves, our performance greatly improves due to our efficient preconditioner update.

Figure 8 also compares the methods on matrices arising from the colorization problem [Levin et al. 2004] of a very large image. This problem uses less irregular weights and hence the piecewise constant basis functions that CMG use undermine its performance.

In Table 2, we summarize the wall-clock running times of the dif-

Problem	Size	Direct	CMG	HSC	LAMG	AMG	ABF	GMG
HDR comp.	4.2M	31.1	29.8	**15.3**	107.4	24.5	**14.8**	19.9
Colorization	5.0M	118.9	19.7	**12.8**	129.7	26.1	**12.6**	27.3
EPD compress (3 scales)	4.2M	140.4	88.8	**70.4**	-	-	-	-
EPD sharpen (5 scales)	4.2M	125.0	97.7	**55.7**	-	-	75.8	-

Table 2: *Total wall-clock time taken (Setup + Solve) in seconds for problems arising on 2D grids. For each problem, winners are highlighted in bold. Timings within 15% of each other are considered a tie. A '-' means the iterative method did not converge to the target relative residual (10^{-6}) within 30 CG iterations. In all cases, our HSC method is faster than the direct solver by factors ranging from 1.2 to 9.3. The first column gives the number of unknowns. Our method performs the best over a range of problems with different levels of continuity.*

Problem	Size	Unpreconditioned	CMG	HSC	LAMG	AMG	ABF	GMG
HDR comp.	4.2M	1.7×10^6	12.4	**1.5**	5.7	14.0	**1.5**	**1.4**
Colorization	5.0M	2.2×10^7	11.6	**2.2**	**2.0**	19.9	**2.3**	20.8
EPD compress	4.2M	1.0×10^6	9.0	**5.9**	-	-	-	-
EPD sharpen	4.2M	6.7×10^5	10.7	**6.6**	-	**6.9**	-	-

Table 3: *Condition numbers achieved by the solvers for different 2D problems. The third column gives the condition number of the unpreconditioned Laplacian. The fourth column onwards list the condition number achieved by each method for that problem. For the EPD problems, the Laplacian at the finest level was used to compute the condition numbers. A '-' means that the method did not converge for a problem.*

ferent methods on different 2D problems. In Table 3, we present the condition numbers achieved by the different methods on these 2D problems.

Finally, in Table 4, we replace the Gauss-Seidel post-smoothing in our algorithm with a Jacobi smoother, to isolate the effect of the smoothing algorithm. The Gauss-Seidel smoother seems better for heterogenous problems, but Jacobi is better for homogenous problems. In our code we provide user options to switch between these two smoothers.

5.2 3D Meshes

We now consider applications of Laplacians arising in 3D mesh processing. The first is mesh segmentation, the second is geodesic distance computation on meshes, and the third is mesh denoising. We note that for both geodesic distance computation and mesh denoising, cotangent Laplacians are usually used. Since our preconditioner requires an M-matrix as input, we use instead the homogenous Laplacian for geodesic distance computation and a Laplacian with inverse Euclidean-based distance measure for mesh smoothing.

Mesh Segmentation. As explained in Section 4.5 and shown in Figure 6, the mesh segmentation method developed in [Liu and Zhang 2007] uses at its core an eigensolver to compute the three lowest eigenvectors of a Laplacian defined over the 3D mesh. Two types of Laplacians are described in the paper: a homogenous graph Laplacian and a geometric Laplacian. In Table 5, we compare the timings taken by our eigensolver, AMG, and MATLAB's built-in eigensolver, *eigs*, to compute the three lowest eigenvectors of homogenous Laplacians defined over 3D meshes of different sizes. For larger mesh sizes, we are between two and three times faster than MATLAB's eigensolver. We observe similar performance for the non-homogenous Laplacians defined in [Liu and Zhang 2007]. The CMG solver does not have an explicit preservation of low eigenvectors across the hierarchy. As a result, a multi-scale initialization works poorly for eigenvector computation.

Geodesic Distance Computation. In [Crane et al. 2012], a heat kernel is used to compute geodesic distance between points on a 3D mesh. The method introduced in their paper involves the solution of two linear systems. The Laplacians in these two systems are related to each other by a diagonal shift. Hence, our preconditioner

Mesh	Vertices	MATLAB *eigs*	AMG (Speedup)	HSC (Speedup)
Hand	50K	0.9	0.9 (1x)	0.9 (1x)
Lion	150K	3.9	3.0 (1.3x)	2.6 (1.5x)
Lago.	800K	27.1	16.1 (1.7x)	10 (2.7x)
Neptune	2M	60.9	35.6 (1.7x)	30.8 (2x)
Statuette	5M	154.8	82.3 (1.9x)	66.5 (2.3x)

Table 5: *Wall clock time (in seconds) to compute the lowest three eigenvectors of homogenous Laplacian defined on a 3D mesh, comparing our eigensolver, agglomerative AMG, and MATLAB's* eigs. *The speedup of our solver and AMG over the direct solver is given in parentheses.*

Mesh	Vertices	Direct Solver	HSC Setup/Solve (Speedup)
Lion	150K	7.5	1/0.5 (15.6x)
Lago.	800K	36.2	2.6/1.7 (21.3x)
Neptune	2M	104.4	11.3/6.2 (16.8x)
Statuette	5M	202	29.1/13.3 (15.2x)

Table 6: *Wall clock time (in seconds) to compute geodesic distances on 3D meshes of different sizes. We compare our solver and speedup over MATLAB's direct solver. The agglomerative methods AMG and CMG failed to converge to an accurate solution for these Laplacians (see text for details). The speedup ratios are given for the solve phase over the direct solver.*

update scheme (Section 4.4) helps to reduce the overall computation time. In Table 6, we compare our solver and MATLAB's direct solver on homogenous Laplacians defined over meshes of different sizes. CMG failed to construct a hierarchy (with a bug that we were unable to fix). AMG was very slow, taking over 100 PCG iterations to achieve an accuracy of 10^{-2}. Our solver provides a significant speedup over the direct solver. Figure 1 (right) shows the visualization of geodesic distances computed on the Caesar mesh. Here, the distances are computed from a point on Caesar's nose (red is closer to the source point, blue is farther). The isolines of the resulting distance function are also shown.

Mesh Smoothing. We perform Laplacian-based smoothing of noisy meshes, using the inverse Euclidean distance measure be-

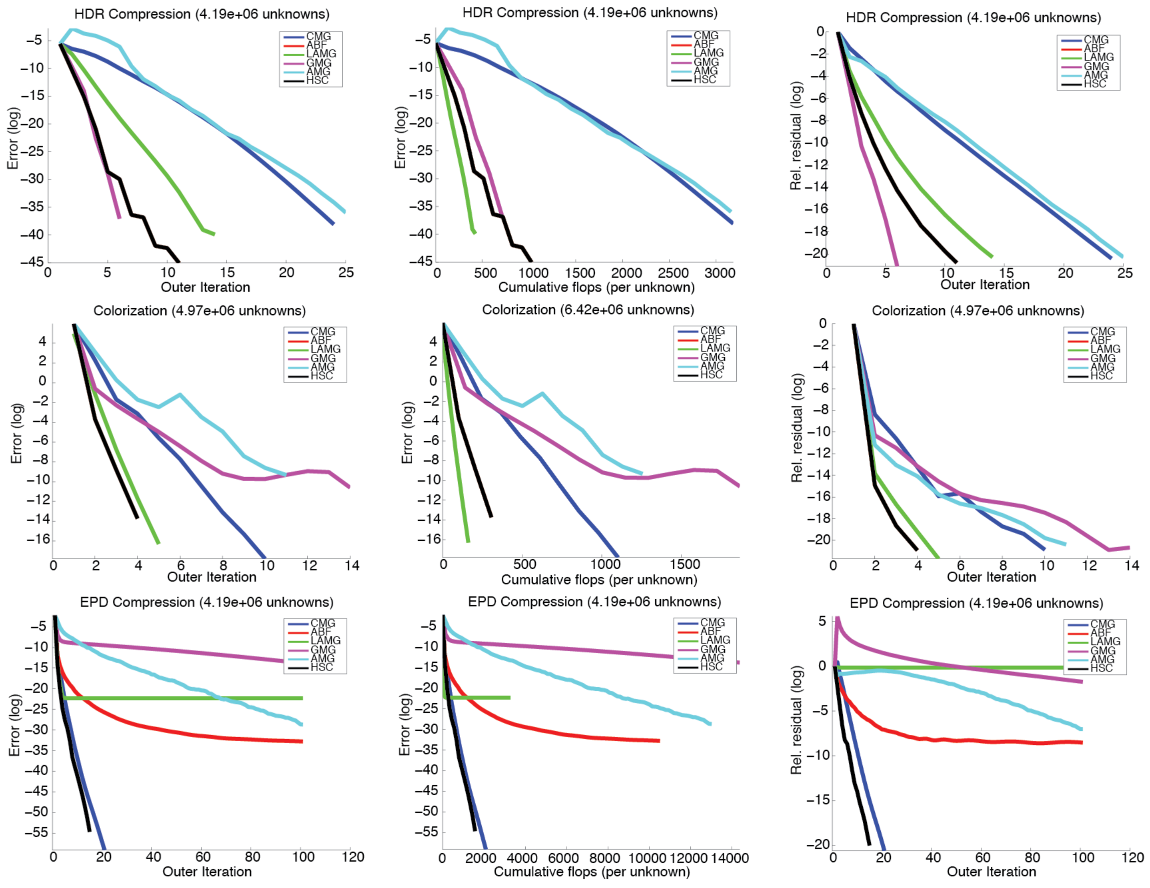

Figure 8: *Performance comparison for 2D problems with varying degrees of homogeneity: top: HDR Compression on a* 2048×2048 *image; middle: image colorization on a* 1962×2533 *size image; and Edge-Preserving Dynamic range compression on a* 2048×2048 *image (single level). Our method consistently ranks at or near the top on all the metrics considered: error with respect to iterations and flop count; and relative residual.*

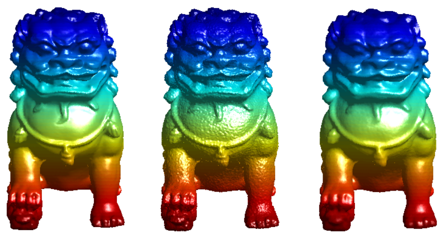

Figure 9: *Example of mesh smoothing using an in homogenous Laplacian based on an inverse distance measure between vertices. Left: original mesh; Middle: noisy mesh; Right: denoised mesh with our solver with smoothing parameter* $t = 0.1$.

Mesh	Vertices	Direct Solver	HSC Setup/Solve (Speedup)
Lion	150K	2.5	0.7/0.9 (2.8x)
Lago.	800K	26.3	2.1/0.5 (52x)
Neptune	2M	87.1	8.2/6.7 (13x)
Statuette	5M	176.2	18.9/50.4 (3.5x)
Lucy	14M	1246	80.6/76.9 (16.2x)

Table 7: *Wall clock time (in seconds) to to smooth noisy meshes of different sizes. We compare our solver and speedup over MATLAB's direct solver. The agglomerative methods AMG and CMG achieved poor accuracy in 50 PCG iterations for these inhomogenous Laplacians. The speedup ratios are given for our solve phase over the direct solver.*

tween vertices , $w_{ij} = \|\mathbf{v}_i - \mathbf{v}_j\|^{-2}$, as the entries in the Laplacian. Given the noisy vertices V_n of the original mesh, we compute smoothed vertices V_s, by solving the smoothing equation $(I + tL)V_s = V_n$. For comparison, on a mesh with 150K vertices, our method takes less than 1 second for the solve phase, whereas the bilateral filtering approach in [Fleishman et al. 2003] takes about 24 seconds on a mesh with 100K vertices (on results reported in 2003). Laplacians based on bilateral filtering-based similarity measures may also be used for denoising. In Figure 9, we give an example of mesh smoothing, and we report timing results in Table 7.

6 Discussion and Conclusions

We have presented an efficient and effective multi-level matrix preconditioning scheme that applies to a large class of Laplacian matrices used in computer graphics, including inhomogeneous computational photography problems and 3D mesh processing. Unlike existing hierarchical preconditioners [Szeliski 2006; Krishnan and Szeliski 2011] that use a fixed coarsening and sparsification scheme, our method adaptively selects the variables that are eliminated and the connections that are dropped and compensated, guided by principles that aim to maximize the preconditioning effectiveness. The derivation of our scheme is based on a formal analysis that ties the algorithmic decisions with their effect over the condition number of the preconditioned system. We derive a new compensation scheme based on this analysis that considers the interplay between levels in the hierarchy. This compensation helps to decouple the resulting condition number from the system size n.

The experiments we report show that our new preconditioner outperforms or equals other state-of-the-art iterative and direct methods in all scenarios, both in terms of operation count and wall-clock time. The ability to perform well on all applications makes it a more useful technique than specialized solvers such as GMG, CMG, or ABF, which only work well under certain conditions. Our performance results from our ability to reduce the condition number of highly irregular Laplacian matrices as well as our use of geometric coarsening in homogeneous regions, i.e., a mixed strategy well-suited for many computer graphics applications.

Our approach is inherently limited to Laplacian matrices that have non-positive off-diagonals entries. This is an important limitation as 3D mesh problems in graphics mostly use non M-matrices such as cotangent Laplacians. The multi-level solvers developed by Aksoylu et al. [2005] can handle cotangent Laplacians, but their hierarchical mesh simplification scheme leads to increased fill-in at coarser levels. Other than direct solvers, we are unaware of practical fast solvers to handle such matrices. Therefore, as future work, we plan to generalize our approach to non M-matrices. We also plan to extend our approach to 3D volumetric applications in computer graphics and simulation, and to develop parallel (multicore and GPU-based) implementations of our algorithms. Finally, we plan to study the theoretical relationship between our approach and existing algorithms such as GMG, AMG, and CMG, to see if we can derive formal proofs on the condition number and scaling properties of our approach.

Acknowledgements

We thank Alec Jacobson, Denis Zorin as well as the SIGGRAPH reviewers for useful suggestions and discussions, and Oren Livne for help with LAMG code and test data.

References

AKSOYLU, B., KHODAKOVSKY, A., AND SCHRÖDER, P. 2005. Multilevel solvers for unstructured surface meshes. *SIAM Journal on Scientific Computing 26*, 4, 1146–1165.

ALCOUFFE, R., BRANDT, A., DENDY, JR, J., AND PAINTER, J. 1981. The multi-grid method for the diffusion equation with strongly discontinuous coefficients. *SIAM Journal on Scientific and Statistical Computing 2*, 4, 430–454.

ARBENZ, P., HETMANUIK, U., LEHOUCQ, R., AND TUMINARO, R. 2003. A comparison of eigensolvers for large-scale 3D modal analysis using amg-preconditioned iterative methods. *Int. Journal for Numerical Methods in Engg. 1*.

BAKER, S., SCHARSTEIN, D., LEWIS, J. P., ROTH, S., BLACK, M. J., AND SZELISKI, R. 2011. A database and evaluation methodology for optical flow. *Int. J. Comput. Vision 92* (March), 1–31.

BHAT, P., ZITNICK, C. L., COHEN, M., AND CURLESS, B. 2010. Gradientshop: A gradient-domain optimization framework for image and video filtering. *ACM Trans. Graph. 29* (April), 10:1–10:14.

BOLZ, J., FARMER, I., GRINSPUN, E., AND SCHRÖODER, P. 2003. Sparse matrix solvers on the GPU: conjugate gradients and multigrid. In *ACM SIGGRAPH 2003 Papers*, ACM, New York, NY, USA, SIGGRAPH '03, 917–924.

BOMAN, E., G., AND HENDRICKSON, B. 2001. On spanning tree preconditioners. *Sandia National Labs*.

BOMAN, E., G., AND HENDRICKSON, B. 2003. Support theory for preconditioning. *SIAM J. Matrix Anal. Appl.*, 3, 694–717.

BOUWMEESTER, H., DOUGHERTY, A., AND KNYAZEV, A. V. 2012. Nonsymmetric multigrid preconditioning for conjugate gradient methods. *arXiv preprint arXiv:1212.6680*.

BRANDT, A., BRANNICK, J. J., KAHL, K., AND LIVSHITS, I. 2011. Bootstrap AMG. *SIAM J. Scientific Computing 33*, 2, 612–632.

BRANDT, A. 1973. Multi-level adaptive technique (MLAT) for fast numerical solution to boundary value problems. In *Proc. Conf. on Numerical Methods in Fluid Mechanics*, vol. 18 of *Lecture Notes in Physics*. Springer Berlin / Heidelberg, 82–89.

BRANDT, A. 1986. Algebraic multigrid theory: The symmetric case. *Applied Mathematics and Computation 19*, 14, 23 – 56.

BRANDT, A. 2001. Multiscale scientific computation: Review 2001. In *Multiscale and Multiresolution Methods*, Springer Verlag, 1–96.

CRANE, K., WEISCHEDEL, C., AND WARDETZKY, M. 2012. Geodesics in heat. *ACM Transactions on Graphics (Proc. SIGGRAPH) 31*, 4 (July).

DAVIS, T. A. 2006. *Direct Methods for Sparse Linear Systems*. SIAM.

FARBMAN, Z., FATTAL, R., LISCHINSKI, D., AND SZELISKI, R. 2008. Edge-preserving decompositions for multi-scale tone and detail manipulation. *ACM Transactions on Graphics (Proc. SIGGRAPH) 27*, 3 (August).

FARBMAN, Z., FATTAL, R., AND LISCHINSKI, D. 2011. Convolution pyramids. *ACM Trans. Graph. 30* (December), 175:1–175:8.

FATTAL, R., LISCHINSKI, D., AND WERMAN, M. 2002. Gradient domain high dynamic range compression. *ACM Transactions on Graphics*, 249–256.

FATTAL, R. 2009. Edge-avoiding wavelets and their applications. In *ACM SIGGRAPH papers*, ACM, New York, NY, USA, 22:1–22:10.

FLEISHMAN, S., DRORI, I., AND COHEN-OR, D. 2003. Bilateral mesh denoising. *ACM Transactions on Graphics (Proc. SIGGRAPH) 22*, 3.

GOLUB, G., AND VAN LOAN, C. F. 1996. *Matrix computation, third edition*. The John Hopkins University Press, Baltimore and London.

KAZHDAN, M., AND HOPPE, H. 2008. Streaming multigrid for gradient-domain operations on large images. *ACM Trans. Graph. 27*, 3, 21:1–21:10.

KELNER, J. A., ORECCHIA, L., SIDFORD, A., AND ZHU, Z. A. 2013. A simple, combinatorial algorithm for solving SDD systems in nearly-linear time. *arXiv preprint arXiv:1301.6628*.

KINCAID, D., AND CHENEY, W. 1991. *Numerical analysis: mathematics of scientific computing*. Brooks/Cole Publishing Co., Pacific Grove, CA, USA.

KOUTIS, I., MILLER, G. L., AND PENG, R. 2010. Approaching optimality for solving SDD linear systems. *Proceedings of FOCS 2010.*

KOUTIS, I., MILLER, G. L., AND TOLLIVER, D. 2011. Combinatorial preconditioners and multilevel solvers for problems in computer vision and image processing. *Computer Vision and Image Understanding*, 1638–1646.

KOUTIS, I., MILLER, G. L., AND PENG, R. 2011. A nearly-m log n time solver for SDD linear systems. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, IEEE, 590–598.

KRISHNAN, D., AND SZELISKI, R. 2011. Multigrid and multilevel preconditioners for computational photography. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)*, 5.

KUSHNIR, D., GALUN, M., AND BRANDT, A. 2010. Efficient multilevel eigensolvers with applications to data analysis tasks. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 32*, 8, 1377–1391.

LEVIN, A., LISCHINSKI, D., AND WEISS, Y. 2004. Colorization using optimization. *ACM Transactions on Graphics*, 689–694.

LEVIN, A., RAV-ACHA, A., AND LISCHINSKI, D. 2007. Spectral matting. *Proceedings of CVPR.*

LIPTON, R. J., ROSE, D. J., AND TARJAN, R. E. 1979. Generalized nested dissection. *SIAM J. Numer. Anal. 16*, 346–358.

LISCHINSKI, D., FARBMAN, Z., UYTTENDAELE, M., AND SZELISKI, R. 2006. Interactive local adjustment of tonal values. In *ACM SIGGRAPH Papers*, ACM, New York, NY, USA, 646–653.

LIU, R., AND ZHANG, H. 2007. Mesh segmentation via spectral embedding and contour analysis. *Eurographics 26*, 3.

LIVNE, O., AND BRANDT, A. 2011. Lean algebraic multigrid (LAMG): Fast graph laplacian solver. *arXiv:1108.0123v1.*

MOULTON, J. D., DENDY, JR., J. E., AND HYMAN, J. M. 1998. The black box multigrid numerical homogenization algorithm. *J. Comput. Phys. 142* (May), 80–108.

PÉREZ, P., GANGNET, M., AND BLAKE, A. 2003. Poisson image editing. *ACM Transactions on Graphics (Proc. SIGGRAPH) 22*, 3, 313–318.

SAAD, Y. 2003. *Iterative Methods for Sparse Linear Systems*, second ed. Society for Industrial and Applied Mathematics.

SHI, L., YU, Y., BELL, N., AND FENG, W.-W. 2006. A fast multigrid algorithm for mesh deformation. In *ACM SIGGRAPH 2006 Papers*, ACM, New York, NY, USA, SIGGRAPH '06, 1108–1117.

SPIELMAN, D. A., AND TENG, S.-H. 2006. Nearly-linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *CoRR abs/cs/0607105.*

SPIELMAN, D., A. 2010. Algorithms, graph theory and linear equations in Laplacian matrices. *Proceedings of the International Congress of Mathematicians (ICM).*

SUN, J., JIA, J., TANG, C.-K., AND SHUM, H.-Y. 2004. Poisson matting. In *ACM SIGGRAPH Papers*, ACM, New York, NY, USA, 315–321.

SZELISKI, R. 1990. Fast surface interpolation using hierarchical basis functions. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 12*, 6 (jun), 513 –528.

SZELISKI, R. 2006. Locally adapted hierarchical preconditioning. *Proceedings of ACM SIGGRAPH.*

TROTTENBERG, U., OOSTERLEE, C., AND SCHULLER, A. 2001. *Multigrid.* Academic Press.

VAIDYA, P. 1990. Solving linear equations with symmetric diagonally dominant matrices by constructing good preconditioners. Tech. rep., Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL.

VANĚK, P., MANDEL, J., AND BREZINA, M. 1996. Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems. *Computing 56*, 3, 179–196.

VANĚK, P. 1995. Fast multigrid solver. *Applications of Mathematics 40*, 1, 1–20.

XU, K., LI, Y., JU, T., HU, S.-M., AND LIU, T.-Q. 2009. Efficient affinity-based edit propagation using K-D tree. In *ACM SIGGRAPH Asia papers*, ACM, New York, NY, USA, 118:1–118:6.

YSERENTANT, H. 1986. On the multi-level splitting of finite element spaces. *Numerische Mathematik 49*, 379–412. 10.1007/BF01389538.

ZEEUW, P. D. 1990. Matrix-dependent prolongations and restrictions in a blackbox multigrid solver. *Journal of Computational and Applied Mathematics 33*, 1, 1–27.

ZHU, Y., SIFAKIS, E., TERAN, J., AND BRANDT, A. 2010. An efficient multigrid method for the simulation of high-resolution elastic solids. *ACM Trans. Graph. 29* (April), 16:1–16:18.

A Proofs of Lemmas

Lemma 1. *Let L be a Laplacian matrix with the following characteristics: it is symmetric, diagonally dominant and has non-positive off-diagonals. Let $\mathcal{C}$ and $\mathcal{F}$ be disjoint variables index sets such that no two variables in $\mathcal{F}$ are connected to each other in L. Then the coarse level system $L^1 = L_{\mathcal{CC}} - L_{\mathcal{CF}} L_{\mathcal{FF}}^{-1} L_{\mathcal{FC}}$ is also a Laplacian matrix which shares the same properties as L.*

Proof. Since no variable in $\mathcal{F}$ is connected to any other variables, $L_{\mathcal{FF}}$, is a diagonal matrix with positive diagonal entries (which was the case in L). Since the off-diagonal entries in L are all non-positive, all the entries in $L_{\mathcal{CF}}$ are also non-positive. Hence all the entries in $L_{\mathcal{CF}} L_{\mathcal{FF}}^{-1} L_{\mathcal{FC}}$ are positive. The matrix $L_{\mathcal{CC}}$ also has non-positive off-diagonals since it is a sub-matrix of L. Hence all the off-diagonal values of L^1 are non-positive. The diagonal elements in Laplace matrices are greater than or equal to the negative of the sum of the corresponding columns (without the diagonal elements). Hence $L_{\mathcal{FF}}^{-1} L_{\mathcal{FC}}$ consists of negative values greater than -1. Similarly, the row sums of $-L_{\mathcal{CF}}$ are smaller than the corresponding diagonal elements in $L_{\mathcal{CC}}$. Hence, all the diagonal elements in $L_{\mathcal{CF}} L_{\mathcal{FF}}^{-1} L_{\mathcal{FC}}$ (which are positive) are smaller than those in $L_{\mathcal{CC}}$ and therefore the diagonal elements of L^1 are all positive. □

Lemma 2. *Let T be a Laplacian matrix of three variables. Let w_{12}, w_{13} and w_{23} be $-T_{12}, -T_{13}$ and $-T_{23}$ respectively, where T_{ij} are the entries of the matrix T. Assume these weights are positive and $w_{12}, w_{13} \geq w_{23}$. The matrix $\tilde{T}$ produced by dropping w_{23} will obey*

$$E_{\tilde{T}} \leq E_T \leq bE_{\tilde{T}}, \tag{14}$$

with $b \leq 3$, where the energies E_T and $E_{\tilde{T}}$ are with respect to any vector x.

Proof. The definition of E_T in terms of the weights is given in Eq. 16. $E_{\tilde{T}} \le E_T$ follows immediately since $E_{\tilde{T}}$ has one less term, $w_{23}(x_2 - x_3)^2$, than E_T. Without loss of generality let us assume $\|\mathbf{x}\| = 1$, in which case $E_{\tilde{T}} = w_{12}(x_2 - x_1)^2 + w_{13}(x_1 - x_3)^2$. However, since $w_{12}, w_{13} \ge w_{23}$ we get $E_{\tilde{T}} \ge w_{23}\big((x_2 - x_1)^2 + (x_1 - x_3)^2\big)$ and since $s^2 + t^2 \ge (s+t)^2/2$ for any s and t, we get $E_{\tilde{T}} \ge w_{23}(x_2-x_3)^2/2$. Thus, $3E_{\tilde{T}} \ge w_{23}(x_2-x_3)^2+E_{\tilde{T}} = E_T$ and therefore $b = 3$ satisfies the upper bound. □

B Characterization of Sparsified Spaces and Compensation

As we explained in Section 4.2, the errors introduced by the sparsification done in each hierarchy level can add up and hurt the preconditioning such that $\kappa(LQ^{-1}) = O(n)$ as in the case of a non-preconditioned system.

The key observation that allows us to cope with this shortcoming is the that at each hierarchy level about half the variables are eliminated and these variables define a linear subspace that *does not* experience the sparsification taking place in the following levels. This means that there is a hierarchy of subspaces that undergo a different number of sparsification steps. Hence, when designing the compensation step, we focus on minimizing the error due to sparsification in the subspaces that are affected by the largest number of sparsification steps. In order to carry this out, we first establish the sense in which $\tilde{L}$ should best approximate L.

The sparsification that takes place at the finest level, over the input matrix L, is likely to affect most of $\mathbb{R}^n$. The second sparsification, which takes place at the second hierarchy level, operates over the sub-matrix that corresponds to the coarse variables selected at the finest level after nearly half of the variables were eliminated. This gives rise to two linear spaces, one of which is affected by this second sparsification step and one that is *not*. To characterize these vector spaces, let us ignore the sparsification done at the finest level and consider the system resulting after computing the first level of the hierarchy,

$$L\mathbf{x} = \mathbf{b} \;\;\Rightarrow\;\; (P^1)^\top L P^1 \mathbf{y}^1 = L^1\mathbf{y}^1 = (P^1)^\top \mathbf{b}, \qquad (15)$$

The solution $\mathbf{y}$ for this problem provides an exact solution, $\mathbf{x} = P^1\mathbf{y}$, for the original system. However, once we sparsify L^1, and get $\tilde{L}^1$, this procedure computes an approximate solution for $\mathbf{x}$. Equivalently, the matrix $P^1(\tilde{L}^1)^{-1}(P^1)^\top$ is the preconditioning matrix Q^{-1} that approximates L^{-1}in this one-level construction. As we see in Eq. 10, the resulting matrices L^1 and $\tilde{L}^1$ are block diagonal, meaning that the coarse and fine coordinates of $\mathbf{y}^1 = [\mathbf{y}^1_{\mathcal{C}}, \mathbf{y}^1_{\mathcal{F}}]^\top \in \mathbb{R}^n$ are uncoupled. Furthermore, since the sparsification at that level (producing $\tilde{L}^1$ from L^1) operates only over the top-left block that corresponds to the coarse variables $\mathbf{y}^1_{\mathcal{C}}$, we see that the fine coordinates $\mathbf{y}^1_{\mathcal{F}}$ are not affected by it. At the original coordinates $\mathbf{x}$, the latter subspace is given by $\{\mathbf{x} \,:\, ((P^1)^{-1}\mathbf{x})_{\mathcal{C}} = \mathbf{0}\}$ and the one affected by the sparsification by $\{\mathbf{x} \,:\, ((P^1)^{-1}\mathbf{x})_{\mathcal{F}} = \mathbf{0}\}$. In fact, according to the definition of an inverse matrix, these subspaces are given more explicitly by $\{P^1[\mathbf{0}, \mathbf{y}^1_{\mathcal{F}}]^\top : \forall \mathbf{y}^1_{\mathcal{F}}\}$ and $\{P^1[\mathbf{y}^1_{\mathcal{C}}, \mathbf{0}]^\top : \forall \mathbf{y}^1_{\mathcal{C}}\}$ respectively.

This rationale can be applied at any hierarchy level l, where we see that $\{P^1P^2..P^l[\mathbf{y}^l_{\mathcal{C}}, \mathbf{0}]^\top : \forall \mathbf{y}^l_{\mathcal{F}}\}$ is affected by the sparsification steps of the first $l+1$ levels (and the following levels computed), whereas $\{P^1P^2..P^l[\mathbf{0}, \mathbf{y}^l_{\mathcal{F}}, \mathbf{0}]^\top : \forall \mathbf{y}^l_{\mathcal{F}}\}$ by the sparsification of only the first l levels. In fact, by starting this analysis from an arbitrary level l, we get that $\{P^lP^{l+1}..P^{l'}[\mathbf{0}, \mathbf{y}^{l'}_{\mathcal{F}}, \mathbf{0}]^\top : \forall \mathbf{y}^{l'}_{\mathcal{F}}\}$ are vectors that are given in the coordinates of the l-th level and that undergo the following $l' - l$ sparsification steps.

This implies that in order to avoid excessive energy mismatch over vectors that undergo multiple sparsification steps, at every hierarchy level l, the compensation should improve the accuracy of the sparsified matrix over the column vectors of $P^lP^{l+1}..P^m$ that correspond to coarser levels. The problem is, however, that at the time we construct the l-th level, the matrix $P^lP^{l+1}..P^m$ is not yet defined and *depends* on our operations at the l-th level, including the compensation itself. Therefore, a more qualitative description of the coarse spaces is needed.

In order to achieve this, let us consider again a single level of the hierarchy and neglect the sparsification done at the finest level. The column vectors of P^1 that correspond to the coarse variables describe an interpolation from the coarser system $\mathbf{y}^1_{\mathcal{C}}$ to the original grid, i.e., $\mathbf{y}^1_{\mathcal{C}} \mapsto P^1[\mathbf{y}^1_{\mathcal{C}}, \mathbf{0}]^\top \in \mathbb{R}^n$. According to its definition in Eq. 9, the matrix P^1 has $I_{\mathcal{CC}}$ at its top-left block and hence the coarse variables in the original grid receive the same values they have at the coarser system $\mathbf{y}^1_{\mathcal{C}}$. The fine variables in the original grid are also determined by $\mathbf{y}^1_{\mathcal{C}}$ and, according to Eq. 9, they are given by $-(L^{-1}_{\mathcal{FF}}L_{\mathcal{FC}})\mathbf{y}^1_{\mathcal{C}}$.

On the other hand, in the case of null input values and derivatives ($y_i, z_{ij} = 0$), the energy is related to the objective function in Eq. 2 by

$$E_L(\mathbf{x}) = F(x)/(\mathbf{x}^\top\mathbf{x}) = \sum_{i\in\mathcal{I}} \big(u_i x_i^2 + w_{ij}(x_i - x_j)^2\big)/(\mathbf{x}^\top\mathbf{x}). \qquad (16)$$

The two coincide on unit-norm vectors, $\mathbf{x}^\top\mathbf{x} = \|\mathbf{x}\|^2 = 1$, and both are given by $\mathbf{x}^\top L\mathbf{x}$. Let us constrain the coarse coordinates of $\mathbf{x}_{\mathcal{C}}$ to be equal to $\mathbf{y}^1_{\mathcal{C}}$ and minimize the functional with respect to the remaining variables $\mathbf{x}_{\mathcal{F}}$. This constrained optimization is computed by $dF(\mathbf{x})/d\mathbf{x}_{\mathcal{F}} = d(x^\top L\mathbf{x})/d\mathbf{x}_{\mathcal{F}} = \mathbf{0}$ given $\mathbf{x}_{\mathcal{C}}$, and gives

$$\begin{bmatrix} I_{\mathcal{CC}} & 0 \\ L_{\mathcal{FC}} & L_{\mathcal{FF}} \end{bmatrix} \mathbf{x} = \begin{bmatrix} \mathbf{y}^1_{\mathcal{C}} \\ \mathbf{0} \end{bmatrix} \;\;\Rightarrow\;\; \begin{array}{c} \mathbf{x}_{\mathcal{C}} = \mathbf{y}^1_{\mathcal{C}} \\ L_{\mathcal{FC}}\mathbf{x}_{\mathcal{C}} + L_{\mathcal{FF}}\mathbf{x}_{\mathcal{F}} = \mathbf{0}, \end{array} \qquad (17)$$

implying that

$$\mathbf{x}_{\mathcal{F}} = -\big(L^{-1}_{\mathcal{FF}}L_{\mathcal{FC}}\big)\mathbf{y}^1_{\mathcal{C}}, \qquad (18)$$

which is what $P^1\mathbf{y}^1_{\mathcal{C}}$ produces at the fine variables of $\mathbf{x}$.

Thus, $P^1[\mathbf{y}^1_{\mathcal{C}}, \mathbf{0}]^\top$ are the minimal-cost vectors given the assignment $\mathbf{y}^1_{\mathcal{C}}$ over its coarse coordinates. If we assume no sparsification (and compensation) steps are applied throughout the hierarchy, computing multiple elimination steps in the hierarchy is equivalent to eliminating these variables at once. Hence, $P^1P^2..P^l[\mathbf{y}^l_{\mathcal{C}}, \mathbf{0}]^\top$ are the minimal-cost vectors given the assignment $\mathbf{y}^l_{\mathcal{C}}$ over its coarse coordinates. However, since there are far less variables in the l-th level as l grows, this optimization has fewer constraints and hence it is expected to achieve lower energies.

Based on this observation, we associate variables at coarser levels with vectors of low energy in the finer levels and focus the compensation step to preserve the energy of such vectors. As explained in Section 4.2, whenever a triangle is sparsified, we compensate by adjusting its two remaining weights such that the triangle's energy contribution remains unchanged over low-energy vectors. This requires modeling the profile of low-energy vectors locally, over three variables in a triangular connectivity, but it does not require knowing P^l of coarser levels. In Section 4.2, we explain how the low-energy vectors are modeled.

Finally, this reasoning did not take into account the sparsification (and compensation) applied to the matrices; the low-energy vectors that we use in this process are predicted based on a sparsified hierarchy. However, assuming we succeed in preserving the energy of low-energy vectors, in each step, the predictions we make in the next level will be reasonably accurate.