

# Multigrid and Multilevel Preconditioners for Computational Photography

Dilip Krishnan<sup>\*</sup>  
Department of Computer Science  
New York University



Richard Szeliski<sup>†</sup>  
Interactive Visual Media Group  
Microsoft Research



**Figure 1:** Colorization using a variety of multilevel techniques: (a) input gray image with color strokes overlaid; (b) “gold” (final) solution; (c) after one iteration of adaptive basis preconditioners (ABF) with partial sparsification; (d) after one iteration of regular coarsening algebraic multigrid (AMG) with Jacobi smoothing and (e) with four-color smoothing. The computational cost for the ABF approach is less than half of the AMG variants and the error with respect to the gold solution is lower (zoom in to see the differences). Four-color smoothing provides faster convergence (lower error) than Jacobi.

## Abstract

This paper unifies multigrid and multilevel (hierarchical) preconditioners, two widely-used approaches for solving computational photography and other computer graphics simulation problems. It provides detailed experimental comparisons of these techniques and their variants, including an analysis of relative computational costs and how these impact practical algorithm performance. We derive both theoretical convergence rates based on the condition numbers of the systems and their preconditioners, and empirical convergence rates drawn from real-world problems. We also develop new techniques for sparsifying higher connectivity problems, and compare our techniques to existing and newly developed variants such as algebraic and combinatorial multigrid. Our experimental results demonstrate that, except for highly irregular problems, adaptive hierarchical basis function preconditioners generally outperform alternative multigrid techniques, especially when computational complexity is taken into account.

**Keywords:** Computational photography, Poisson blending, colorization, multilevel techniques, fast PDE solution, parallel algorithms **Links:**  DL  PDF

## 1 Introduction

Multigrid and multilevel preconditioning techniques have long been widely used in computer graphics and computational photography

as a means of accelerating the solution of large gridded optimization problems such as geometric modeling [Gortler and Cohen 1995], high-dynamic range tone mapping [Fattal et al. 2002], Poisson and gradient-domain blending [Pérez et al. 2003; Levin et al. 2004b; Agarwala et al. 2004], colorization [Levin et al. 2004a] (Fig. 1), and natural image matting [Levin et al. 2008]. They have also found widespread application in the solution of computer vision problems such as surface interpolation, stereo matching, optical flow, and shape from shading [Terzopoulos 1986; Szeliski 1990; Pentland 1994], as well as large-scale finite element and finite difference modeling [Briggs et al. 2000; Trottenberg et al. 2000].

While the locally adaptive hierarchical basis function technique developed by Szeliski [Szeliski 2006] showed impressive speedups over earlier non-adaptive basis functions [Szeliski 1990], it was never adequately compared to state-of-the-art multigrid techniques such as algebraic multigrid [Briggs et al. 2000; Trottenberg et al. 2000] or to newer techniques such as combinatorial multigrid [Koutis et al. 2009]. Furthermore, the original technique was restricted to problems defined on four neighbor ( $\mathcal{N}_4$ ) grids.

In this paper, we generalize the sparsification method introduced in [Szeliski 2006] to handle a larger class of grid topologies, and show how multi-level preconditioners can be enhanced with smoothing to create hybrid algorithms that accrue the advantages of both adaptive basis preconditioning and multigrid relaxation. We also provide a detailed study of the convergence properties of all of these algorithms using both condition number analysis and empirical observations of convergence rates on real-world problems in computer graphics and computational photography.

Our experimental results demonstrate that locally adaptive hierarchical basis functions (ABF) [Szeliski 2006] combined with the extensions proposed in this paper generally outperform algebraic and combinatorial multigrid techniques, especially once computational complexity is taken into account. However, for highly irregular and inhomogeneous problems, techniques that use adaptive coarsening strategies (Section 3), which our approach does not currently use, may perform better.

In this paper, we consider general spatially varying quadratic cost functions. This allows the algorithms presented here to be applied

<sup>\*</sup>dilip@cs.nyu.edu

<sup>†</sup>szeliski@microsoft.com

to a variety of problems. In the existing literature, a number of optimized schemes have been developed focusing on specific problems. We give a brief survey of such schemes here.

[Agarwala 2007] considers the problem of Poisson blending when applied to the seamless stitching of very large images. The optimization proceeds by defining a variable-sized grid (large spacing in smooth regions and small spacing in non-smooth regions), resulting in a much smaller linear system. [McCann and Pollard 2008] use a standard V-cycle geometric multigrid method on a GPU, to achieve near real-time gradient field integration. [Farbman et al. 2009] solve the image cloning problem by replacing the linear system solver with an adaptive interpolation and mesh refinement strategy. This is similar in spirit to the work of [Agarwala 2007]. However, this approach cannot work for problems such as HDR image compression or Poisson blending with mixed gradients.

The algorithm in [Roberts 2001] is similar to that of [Szeliski 1990] but with some important differences in the preconditioner construction (there is no diagonal preconditioning of fine-level variables and Jacobi smoothing is used). This preconditioner-based solver shows better performance than geometric multigrid and extensions to 3D problems are given. However, since the interpolants aren't adapted to the problem, for inhomogeneous problems, both [Roberts 2001] and [Szeliski 1990] will underperform the adaptive basis functions presented in [Szeliski 2006] and this paper. [Wang et al. 2004] use the solver of [Roberts 2001] to solve a 3D version of the Poisson blending problem for video. [Jeschke et al. 2009] present a GPU solver for the integration of a homogeneous Poisson equation; this method uses Jacobi iterations with modified stencil sizes. [McAdams et al. 2010] present a parallelized Poisson solver for fluid simulation problems. This solver is based on geometric multigrid. The algorithms in [Wang et al. 2004; Jeschke et al. 2009; McAdams et al. 2010] are all restricted to homogeneous Poisson problems and their extension to inhomogeneous problems does not seem straightforward. [Kazhdan et al. 2010] develop a high-performance solver where their key technical contribution is to parallelize the raster-order Gauss-Seidel smoothing. Our ABF preconditioner and four-color Gauss-Seidel smoothers do not require such streaming implementations, as simpler overlapped (multi-resolution) tiles can be used to perform out-of-core computations.

The remainder of this paper is structured as follows. Section 2 presents the general class of regular gridded problems that we study, along with the associated quadratic energy we minimize and the linear systems of equations we aim to solve. In Section 3, we describe the various solvers we evaluate and qualitatively compare their characteristics. Section 4 describes how the eigenvalues of the iterative solvers and condition numbers of the preconditioned solvers can be used to predict the convergence rates of various algorithms. In Section 5, we show how to extend the sparsification step introduced in [Szeliski 2006] to a wider range of problems. In Section 6, we give descriptions of the sample problems that we study. Section 7 summarizes our experimental analysis of both the theoretical (condition number) and empirical convergence rates of the various algorithms and variants we consider. Section 8 contains conclusions and recommendations. To allow the community to better understand and use these solvers, we provide a MATLAB implementation of these algorithms at [www.cs.nyu.edu/~dilip/research/abf](http://www.cs.nyu.edu/~dilip/research/abf)

## 2 Problem formulation

Following [Szeliski 2006], we consider two-dimensional variational problems discretized on a regular grid. We seek to reconstruct a function  $f$  on a discrete domain  $\Omega$  given data  $d$  and, optionally, gradient terms  $g^x$  and  $g^y$ . Let  $(i, j) \in \Omega$  represent a point in this

domain. The problems we study involve finding the solution  $f$  that minimizes the quadratic energy

$$E_5 = \sum_{i,j \in \Omega} w_{i,j} (f_{i,j} - d_{i,j})^2 + s_{i,j}^x (f_{i+1,j} - f_{i,j} - g_{i,j}^x)^2 + s_{i,j}^y (f_{i,j+1} - f_{i,j} - g_{i,j}^y)^2. \quad (1)$$

The  $w_{i,j}$  are (non-negative, potentially spatially-varying) data term weights and the  $s_{i,j}^x$  and  $s_{i,j}^y$  are (non-negative, potentially spatially-varying) gradient term weights corresponding to the  $x$  and  $y$  directions respectively. These weights are used to balance the closeness of the values of  $f$  to  $d$  and the values of the gradients of  $f$  to  $g^x$  and  $g^y$ . The target gradient values  $g^x$  and  $g^y$  can be set to 0 if only smoothing is desired. (See Section 6 on how these weights map to various computer graphics and computational photography problems.)

If we represent the values in the function  $f$  by a vector  $x$ , we can then re-write Eqn. 1 as a quadratic energy,

$$E = x^T A x - 2b^T x + c, \quad (2)$$

where  $A$  is a sparse symmetric positive definite (SPD) and symmetric diagonally dominant (SDD) matrix with only non-positive off-diagonal terms. Taking the derivative of this energy with respect to  $x$  and setting it to 0 gives us the linear systems of equations to be solved,

$$A x = b. \quad (3)$$

The horizontal and vertical finite differences in Eqn. 1 give  $A$  a 5-banded structure.

If we add  $xy$  cross-derivatives,  $A$  has a 9-banded structure. The energy function becomes

$$E_9 = E_5 + \sum_{i,j \in \Omega} s_{i,j}^{xy} (f_{i+1,j-1} - f_{i,j} - g_{i,j}^{xy})^2 + s_{i,j}^{yx} (f_{i+1,j+1} - f_{i,j} - g_{i,j}^{yx})^2. \quad (4)$$

In this paper, we consider the solution of linear systems involving both 5-banded and 9-banded matrices. We often refer to these systems as 5-point stencils and 9-point stencils, respectively.

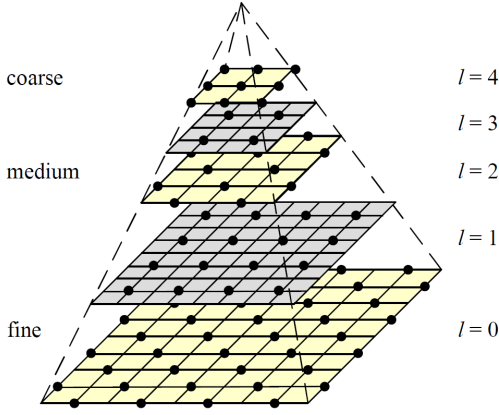
## 3 Solution techniques

In this section, we provide a summary of the methods used to solve systems of equations of the form  $Ax = b$  when  $A$  is a sparse, banded, symmetric positive definite matrix. For more details on these algorithms, please see [Krishnan and Szeliski 2011].

Traditional dense direct techniques such as Cholesky decomposition and Gaussian elimination have a cost of  $O(n^3)$ , where  $n$  is the number of variables. For gridded two-dimensional problems such as the ones we study in this paper, sparse direct methods such as nested dissection [Davis 2006] cost  $O(n^{3/2})$ . When  $n$  becomes large, such direct solvers incur too much memory and computational cost to be practical.

In many graphics and vision applications, it is not necessary to solve the linear systems exactly. Approximate solutions with a medium level of accuracy are often sufficient, owing to the eventual discretization of the output value  $f$  (e.g., a color image) into integral values. Hence, we can use iterative methods, which have the advantage of allowing termination when a pre-specified level of accuracy has been reached [Saad 2003].

Simple iterative methods such as Jacobi and Gauss-Seidel have  $O(n)$  cost per iteration. The number of iterations depends on the



**Figure 2:** Multilevel pyramid with half-octave sampling [Szeliski 2006].

condition number  $\kappa$  of  $A$  and on the desired level of accuracy (Section 4). For the kinds of reconstruction and interpolation problems studied in this paper, the condition numbers can be very large, requiring far too many iterations to be practical. However, iterative methods such as Jacobi and Gauss-Seidel, when used as *smoothers*, are an inexpensive way to discover high-frequency components of the correction in very few iterations. Hence they are used in multigrid methods in conjunction with a pyramid of grids. The tradeoff in using simple iterative methods as smoothers is their computational load versus their ability to discover high frequency components of the correction. Smoothers that best satisfy this tradeoff are the Gauss-Seidel family, including four-color Gauss-Seidel and raster-order Gauss-Seidel. See [Krishnan and Szeliski 2011] for details.

Conjugate Gradient (CG) algorithms typically exhibit much faster convergence than Jacobi or Gauss-Seidel methods for SPD problems [Shewchuk 1994; Saad 2003]. CG is almost always used with a preconditioner, and preconditioned CG (PCG) usually requires many fewer iterations than Jacobi to reach the same accuracy. The main challenge with PCG is the design of preconditioners that are computationally efficient and yet achieve a significant acceleration over unpreconditioned CG.

Hierarchical basis preconditioners [Szeliski 1990; Szeliski 2006] involve re-writing the original *nodal* variables  $x$  as a combination of *hierarchical* variables  $y$  that live on a multi-resolution pyramid with  $L$  levels. The relationship between  $x$  and  $y$  can be written as  $x = \hat{S}y$ , where the reconstruction matrix  $\hat{S}$  consists of recursively interpolating variables at a coarser level and adding in the finer-level variables. The original paper [Szeliski 1990] used a fixed set of full-octave interpolants, while the later paper [Szeliski 2006] used half-octave locally adaptive interpolants (Fig. 2), whose values are derived from the structure of the Hessian matrix  $A$  (in combination with the sparsification rules discussed in Section 5). In this paper, rather than simply using  $SS^T$  or  $SD^{-1}S^T$  (where  $D$  is the diagonal of the per-level Hessians) as the preconditioner, as in these previous publications, we invert the coarse-level Hessian using a sparse direct solver, as is commonly the case with multigrid techniques. More detailed descriptions of these techniques can be found in the extended version of this paper [Krishnan and Szeliski 2011].

Multigrid (MG) methods [Briggs et al. 2000; Trottenberg et al. 2000] are an alternative family of numerically stable and computationally efficient methods for iteratively solving SPD linear systems. Originally developed for homogeneous elliptic differential

equations, MG methods now constitute a family of methods under a common framework that can be used to solve both inhomogeneous elliptic and non-elliptic differential equations. This family includes algebraic multigrid (AMG) [Briggs et al. 2000; Trottenberg et al. 2000; Kushnir et al. 2010; Napov and Notay 2011] and combinatorial multigrid (CMG) techniques [Koutis et al. 2009].

Like hierarchical (multilevel) preconditioners, multigrid techniques use a pyramid to accelerate the reduction of low-frequency errors. However, in order to ensure that both high- and low-frequency errors are reduced, multigrid techniques perform additional smoothing at each level in the pyramid.

Geometric multigrid (GMG) approaches, like the original hierarchical basis technique, use fixed full-octave coarsening and interpolation schemes. Algebraic multigrid (AMG) techniques, like locally adaptive basis functions, derive their interpolation weights from the structure of the Hessian  $A$ , giving more weight to neighbors that have larger  $|a_{ij}|$  values. Algebraic multigrid techniques also use an adaptive subsampling of variables to define the coarse-level grid. Combinatorial multigrid (CMG) techniques resemble algebraic multigrid but use an *agglomerative* coarsening scheme in which clusters of fine-level variables get replaced by a single coarse-level variable [Koutis et al. 2009]. This has the advantage of simpler and faster interpolation and also supports deriving bounds on the growth in relative condition numbers as the number of levels increases.

While multigrid techniques can be used as stand-alone iterative solvers (like Jacobi and Gauss-Seidel), it is now common to use certain forms (those whose effects are equivalent to SPD transforms) as preconditioners in conjugate gradient. We evaluate both variants in this paper.

In Appendix A, we describe an algorithm that contains as special cases all of the algorithms described above. The input to the algorithm is the current residual  $r_k$ , and the output is a correction term  $e_k$ , which is added to the current iterate  $x_k$  to give the next iterate.  $r_k$  and  $x_k$  are related as  $r_k = b - Ax_k$ . The algorithm consists of pre-smoothing steps, a coarse level solve, fine-level diagonal preconditioning, and post-smoothing steps. Except for the coarse level solve, the other steps are optional. Setting  $\nu^{pre} = \nu^{post} = 0$  and  $d = 1$  gives the multilevel ABF and HBF preconditioners; setting  $d = 0$  and  $\nu^{pre} = \nu^{post} = 1$  gives the multigrid AMG, GMG and CMG algorithms.

Looking at the algorithm in Appendix A, one can see that it is possible to combine the smoothing elements of multigrid with the fine-level subspace solution (diagonal preconditioning) in multilevel preconditioners. This produces a new algorithm that benefits from both previous approaches, albeit at a higher computational cost. If we set  $\nu^{pre} = \nu^{post} = 1$  and  $d = 1$ , we get a hybrid algorithm that performs pre-smoothing, followed by a coarse-level splitting and preconditioning of the coarse-level and fine-level variables, followed by another step of post-smoothing. The performance of this new algorithm is analyzed in Section 7.

In our experiments, we use the implementation of CMG provided by the authors of [Koutis et al. 2009]. Because we have been unable to find a public domain implementation of algebraic multigrid, we have only implemented the adaptive interpolant portion of AMG, but still use a fixed regular full-octave coarsening scheme. More details about multigrid techniques and our own re-implementations can be found in [Briggs et al. 2000; Trottenberg et al. 2000; Kushnir et al. 2010; Koutis et al. 2009; Krishnan and Szeliski 2011].

## 4 Convergence analysis

To estimate or bound the asymptotic rate at which an iterative algorithm will converge, we can compute the *convergence rate*, which is the expected decrease in error per iteration.

For simple iterative algorithms such as Jacobi and Gauss Seidel [Saad 2003; Krishnan and Szeliski 2011], we get a general recurrence of the form

$$x_k = R_k b + H^k x_0 \quad (5)$$

$$= x^* + H^k (x_0 - x^*), \quad (6)$$

where  $x_0$  is the initial solution,  $x_k$  is the current solution,  $R_k = (I - H^k)A^{-1}$  is defined in [Krishnan and Szeliski 2011, Appendix A],  $x^* = A^{-1}b$  is the optimal (final) solution, and  $H$  is the algorithm-dependent *iteration matrix*. The *spectral radius* of the iteration matrix

$$\rho(H) = \max_{\lambda \in \sigma(H)} |\lambda| \quad (7)$$

is the eigenvalue of  $H$  that has the largest absolute value.

As the iteration progresses, the component of the error  $e = x_0 - x^*$  in the direction of the associated “largest” eigenvector  $v_{\max}$  of  $H$ ,  $\tilde{e} = v_{\max}^T e$ , gets reduced by a *convergence factor* of  $\rho$  at each iteration,

$$\tilde{e}_k = \rho^k \tilde{e}_0. \quad (8)$$

To reduce the error by a factor  $\epsilon$  (say  $\epsilon = 0.1$ ) from its current value, we require

$$\rho^k < \epsilon \quad \text{or} \quad k > \log_{\epsilon} \rho = -\log_{1/\epsilon} \rho. \quad (9)$$

It is more common [Saad 2003, p.113] to define the *convergence rate*  $\tau$  using the natural logarithm,

$$\tau = -\ln \rho, \quad (10)$$

which corresponds to how many iterations it takes to reduce the error by a factor  $\epsilon = 1/e \approx 0.37$ .

The convergence rate allows us to compare the *efficiency* of two alternative algorithms while taking their computational cost into account. We define the *effective convergence rate* of an algorithm as

$$\hat{\tau} = 100 \tau / C, \quad (11)$$

where  $C$  is the amount of computational cost required per iteration.<sup>1</sup>

The details on how to compute the computational cost  $C$  for each of the algorithms we evaluate are given in our companion paper [Krishnan and Szeliski 2011] and exact numbers are presented in the experimental results section. Roughly speaking, we find that because of the need for smoothing, traditional geometric and algebraic multigrid techniques (GMG, AMG) have about twice the computational cost of hierarchical basis techniques (HBF, ABF). Combinatorial multigrid (CMG) only uses additions in the restriction and prolongation steps but still does smoothing, and so its computation cost is between that of HBF/ABF and GMG/AMG techniques.

<sup>1</sup>To make these numbers more similar across problems and independent of grid size, we define  $C$  as the number of floating point operations per grid point in the fine grid. The scale factor of 100 makes  $\hat{\tau}$  easier to print and corresponds roughly to how many flops it takes to perform one multigrid cycle.

For conjugate gradient descent [Shewchuk 1994; Saad 2003], the asymptotic convergence rate is

$$\rho_{CG} \leq \left( \frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right), \quad (12)$$

where  $\kappa$  is the condition number of  $A$ ,

$$\kappa(A) = \frac{\lambda_{\max}}{\lambda_{\min}}, \quad (13)$$

i.e., the ratio of the largest and smallest eigenvalues of  $A$ . When the eigenvalues of  $A$  are highly clustered, the convergence rate can be even faster [Shewchuk 1994].

For preconditioned conjugate gradient, the convergence factor in Eqn. 12 depends on the condition number of  $B^{-1}A$ , where  $B = M^{-1}$  is the simple-to-invert approximation of  $A$  corresponding to the preconditioner  $M$ . The derivations of the formulas for  $M$  corresponding to various preconditioners and multigrid techniques are given in our longer companion report [Krishnan and Szeliski 2011].

An intuitive way to compute the generalized condition number is to use *generalized Rayleigh quotients*

$$\kappa(B^{-1}A) = \kappa(A, B) \equiv \max_x \frac{x^T A x}{x^T B x} \cdot \max_x \frac{x^T B x}{x^T A x} \quad (14)$$

[Koutis et al. 2009]. Each of the quadratic forms  $x^T A x$  can be interpreted as the power dissipated by network  $A$ , where the edge weights  $a_{ij}$  give the conductances and the vector  $x$  specifies the input voltages.

Thus, a good preconditioner  $B$  is one that dissipates neither significantly more nor significantly less power than the original matrix  $A$  across all possible voltages (both smooth and non-smooth). We use this observation to develop better sparsification rules in the next section.

To summarize, the convergence factor  $\rho$  for a standard iterative algorithm such as Jacobi, Gauss-Seidel, or multigrid, depends on the spectral radius of the associated iteration matrix  $H$ . For preconditioned conjugate gradient, it depends on the generalized condition number  $\kappa(A, B)$ , which can be thought of as the ratio of the greatest increase and decrease in relative power dissipation of the preconditioner matrix  $B$  with respect to the original matrix  $A$ . In our experimental results section, we compute the theoretical convergence factor  $\rho$ , the convergence rate  $\tau$ , and the effective convergence rate  $\hat{\tau}$  for problems that are sufficiently small to permit easy computation of their eigenvalues. For all problems, we also estimate an *empirical* convergence rate  $\tilde{\tau}$  by dividing the logarithmic decrease in the error  $|e_k|$  by the number of iterations,

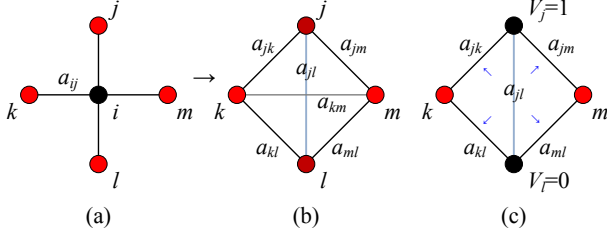
$$\tilde{\tau} = -\frac{1}{N} \ln \frac{|e_N|}{|e_0|}. \quad (15)$$

## 5 Sparsification

The adaptive hierarchical basis function algorithm of [Szeliski 2006] relies on removing unwanted “diagonal” links after each round of red-black multi-elimination, as shown in Fig. 3. The rule for re-distributing an unwanted connection  $a_{jl}$  to its neighbors is

$$a'_{jk} \leftarrow a_{jk} + s_N a_{jk} a_{jl} / S, \quad (16)$$

where  $S = a_{jk} + a_{kl} + a_{jm} + a_{lm}$  is the sum of the edge weights adjacent to  $a_{jl}$  and  $s_N = 2$  is a constant chosen to reproduce the finite element discretization at the coarser level. The diagonal entries



**Figure 3: Sparsification:** (a) after the black node  $i$  is eliminated, the extra “diagonal” links  $a_{jl}$  and  $a_{km}$  shown in (b) are introduced. (c) Since nodes  $j$  and  $l$  (now shown in black) are eliminated at the next round, only the  $a_{jl}$  edge needs to be eliminated and redistributed to the adjacent edges  $a_{jk}$ ,  $a_{kl}$ ,  $a_{jm}$ , and  $a_{ml}$ .

corresponding to the edges being eliminated and those being “fattened” are modified appropriately to maintain the same row sums as before, thereby resulting in the same energy for constant-valued solutions.<sup>2</sup>

In this paper, we introduce several extensions and improvements to this original formulation:

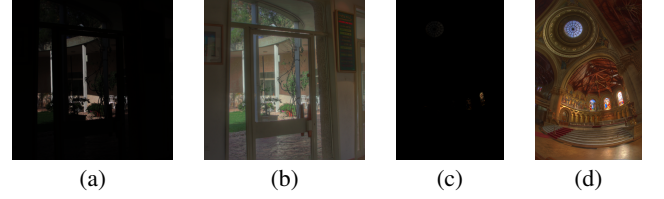
**$\mathcal{N}_8$  grid sparsification.** Although the original paper only described how to handle 4-neighbor (5-point stencil) discretizations, it is straightforward to apply the same sparsification rule used to eliminate unwanted diagonal elements from an 8-neighbor (9-point stencil) discretization. We demonstrate this in our experiments with colorization algorithms.

**Fine-fine sparsification.** The original algorithm eliminates both sets of “diagonal links”, i.e.,  $a_{jl}$  and  $a_{km}$  in Fig. 3b. Closer inspection reveals that this is unnecessary. At the next level of coarsening, only two out of the four nodes, say  $j$  and  $l$  (shown as dark red nodes in Fig. 3b and black in Fig. 3c), will be themselves eliminated. Therefore, it is only necessary to sparsify the links connecting such nodes, which results in a lower approximation error between the original and sparsified network. The other  $a_{km}$  link is left untouched and does not participate in the computation of the adaptive interpolants or cause the coarse-level Hessian matrix to grow in bandwidth. In the subsequent text, we refer to this sparsification as the “7-point stencil” sparsification, since the Hessian matrix after sparsification is on average 7-banded.

**Pre-sparsification.** The original algorithm in [Szeliski 2006] performs sparsification (elimination of “diagonal links”) each time a coarse-level Hessian is computed using the Galerkin condition [Szeliski 2006, Eqn. (13)]. However, unless the system is going to be further coarsened, i.e., unless another set of adaptive interpolants needs to be computed, this is unnecessary. The higher-bandwidth coarse-level matrix can be used as-is in the direct solver, at a slight increase in computational cost but potentially significant increase in accuracy (preconditioning efficacy). We call deferring the sparsification step to just before the interpolant construction as *pre-sparsification* and call the original approach *post-sparsification*. Both variants are evaluated in our experimental results section.

**The limits of sparsification.** Given that our sparsification rules are based on a sensible approximation of the low-frequency modes of one matrix with a sparser version, we can ask if this always result in a reasonably good approximation of the original problem. Unfortunately, this is not always the case. Consider the case where

<sup>2</sup>The modified incomplete LU (Cholesky) or MILU algorithm also maintains row sums, but it simply drops off-diagonal entries instead of redistributing them to adjacent edges [Saad 2003; Szeliski 2006].



**Figure 4: Two examples of High Dynamic Range compression [Fattal et al. 2002]:** “Belgium” (a) Input and (b) Compressed; “Memorial” (c) Input and (d) Compressed.

the value of  $|a_{jl}|$  is much larger than all of the adjacent edge values in Fig. 3c. In this case, the sparsification rule (Eqn. 16) can increase the values of the adjacent edges by an arbitrarily large ratio  $s_N a_{jl}/S \gg 1$ . The problem with this is that while the power dissipation of the sparsified system to the 0/1 voltage shown in Fig. 3c can be kept constant, if we then set all but one (say  $V_j = 1$ ) of the voltages in our network to zero, the power dissipated by the original and sparsified network can have an arbitrarily large Rayleigh quotient (Eqn. 14) and hence an arbitrarily bad generalized condition number.

A more intuitive way of describing this problem is to imagine a grid where neighbors are connected in long parallel chains (say a tight spiral structure, or two interleaved spirals with strongly differing values or voltages). If we use regular two-dimensional coarsening, there is no way to represent a reasonable approximation to this distribution over a coarser grid. Adaptive coarsening schemes such as CMG or full AMG, on the other hand, have no trouble determining that a good coarsening strategy is to drop every other element along the long linear chains. Therefore, an approach that combines sparsification rules with adaptive coarsening may perform better for such problems, and is an interesting area for future work.

## 6 Sample problems

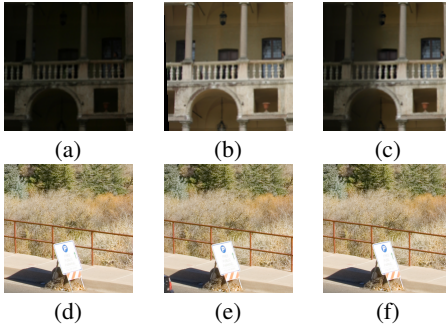
To measure the performance of the various techniques, we analyze a number of quadratic minimization problems that arise from the solution of two-dimensional computer graphics and computational photography problems.

The problems in this section are ordered from most regular to most irregular, which corresponds roughly to the level of difficulty in solving them. Regular problems have spatially uniform (homogeneous) data ( $w_{i,j}$ ) and smoothness ( $s_{i,j}$ ) weights and are well suited for traditional multigrid techniques. Moderately regular problems may have irregular data constraints and occasional tears or discontinuities in the solution. The most challenging problems are those where the smoothness can be very irregular and where elongated one-dimensional structures may arise. Techniques based on regular coarsening strategies generally do not fare well on such problems.

**High Dynamic Range (HDR) Compression** HDR image compression [Fattal et al. 2002] (also known as tone mapping) maps an input image whose values have a high dynamic range into an 8-bit output suitable for display on a monitor. The gradients of the log of the input luminance are compressed in a non-linear manner. An image is then reconstructed from the compressed gradients  $\{g_{i,j}^x, g_{i,j}^y\}$  by minimizing Eqn. 1. We follow the formulation given in [Fattal et al. 2002] to perform the HDR compression. Fig. 4 shows the two images used for our HDR compression tests.

**Poisson Blending** In Poisson blending [Pérez et al. 2003; Levin et al. 2004b; Agarwala et al. 2004], gradient domain operations are used to reduce visible discontinuities across seams when stitching together images with different exposures. There are a few differ-





**Figure 5:** Two examples of Poisson blending: (a,d) first source image; (b,e) second source image; (c,f) blended results.

ent variants of Poisson blending. In our implementation, we blend each of the RGB channels separately, using the formulas provided in Section 2 of [Szeliski et al. 2011]. Fig. 5 gives two examples of Poisson blending, where two input images of the same scene are stitched together along a seam. The seam is almost unnoticeable because the stitching is done in the gradient domain.

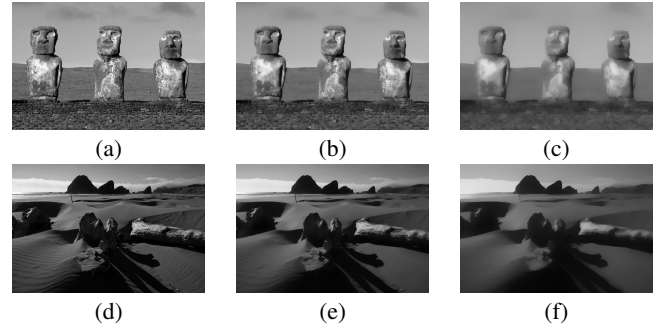
**2D membrane** A membrane refers to an energy-minimizing 2D piecewise smooth function used to interpolate a set of discrete data points [Terzopoulos 1986]. It is a good canonical problem to test the performance of techniques for solving spatially-varying (inhomogeneous) SPD linear systems. Our current test problem, modeled after those previously used in [Szeliski 1990; Szeliski 2006], uses four data points at different heights and a single tear (zeros in the  $s_{i,j}^x$  weights at appropriate locations) running halfway up the middle of the surface.

**Colorization** In colorization [Levin et al. 2004a], a gray-scale image is converted to color by propagating user-defined color strokes. In order to prevent color bleeding across edges, the propagation of the color is controlled by the strength of edges in the gray-scale image. Let  $Y$  be the gray-scale luminance image and  $S$  the stroke image. As in [Levin et al. 2004a], we work in YIQ color space. The system Eqn. 2 or Eqn. 4 is solved to recover both the I and Q color channels. This is combined with  $Y$  to give the output YIQ image. If  $s^{xy}$  and  $s^{yx}$  are set to 0, a 5-band system results, otherwise we have a 9-band system. Fig. 1 gives an example of colorization.

**Edge-preserving decomposition (EPD)** An edge-preserving multi-scale decomposition of an image can be created using non-linear filters to smooth an image while preserving sharp edges [Farbman et al. 2008]. The non-linear smoothing step is performed by minimizing a system of equations similar to those used in colorization, except that instead of sparse color strokes, the complete input image is used as a weak constraint on the final solution, i.e.,  $w_{i,j}$  is small but constant everywhere. We use this as another example of an inhomogeneous Poisson reconstruction problem, since the smoothness weights  $s_{i,j}$  are spatially varying. Fig. 6 gives an example of edge-preserving decomposition, with the original image being passed through 2 successive layers of edge-preserving smoothing.

## 7 Experiments

In this section, we summarize the results of our experimental comparisons of the solvers of Section 3 on the different sets of problems outlined in Section 6. Table 1 shows an example of the full set of results available in the full-length version of this paper [Krishnan and Szeliski 2011], while Fig. 7 shows sample convergence plots. Table 2 summarizes the empirical convergence rate  $\hat{\tau}$  applied to all



**Figure 6:** An example of edge-preserving decomposition [Farbman et al. 2008]: (a) input gray-scale image; (b) first layer of edge-preserving smoothing; (c) second layer of edge-preserving smoothing (by smoothing first layer). (d)-(f) input gray-scale image and two layers of smoothing for another gray-scale image;

| Algorithm     | theoretical |         |        |        |       |              | empirical    |              |           |              |
|---------------|-------------|---------|--------|--------|-------|--------------|--------------|--------------|-----------|--------------|
|               | $\kappa$    | $\nu$   | $\rho$ | $\tau$ | $C$   | $\hat{\tau}$ | $\bar{\rho}$ | $\bar{\tau}$ | $\bar{C}$ | $\hat{\tau}$ |
| HBf           | 23.59       | 356.83  | 0.66   | 0.42   | 32.5  | 1.28         | 0.76         | 0.27         | 32.5      | 0.84         |
| ABF-Pre7      | 3.05        | 8988.59 | 0.27   | 1.30   | 40.6  | 3.21         | 0.03         | 3.57         | 40.6      | 8.79         |
| ABF-Pre7-W    | 1.09        | 8988.59 | 0.02   | 3.81   | 141.3 | 2.69         | 0.01         | 4.98         | 141.3     | 3.53         |
| ABF-Pre7-4C   | 1.22        | 8988.59 | 0.05   | 3.01   | 97.4  | 3.09         | 0.00         | 11.17        | 97.4      | 11.47        |
| ABF-Pre7-4C-W | 1.95        | 7148.51 | 0.17   | 1.80   | 121.3 | 1.48         | 0.00         | 15.12        | 440.3     | 3.43         |
| AMG-J         | 3.12        | 7148.51 | 0.28   | 1.28   | 87.9  | 1.46         | 0.09         | 2.46         | 87.9      | 2.80         |
| AMG-4C        | 2.38        | 7148.51 | 0.21   | 1.54   | 83.4  | 1.85         | 0.01         | 4.70         | 83.4      | 5.64         |
| AMG-4C-W      | 1.95        | 7148.51 | 0.17   | 1.80   | 121.3 | 1.48         | 0.01         | 4.89         | 121.3     | 4.03         |
| GMG-J         | 9.46        | 356.83  | 0.51   | 0.67   | 91.2  | 0.74         | 0.32         | 1.15         | 91.2      | 1.26         |
| V(1,1)-4C     | 2.38        | 7148.51 | 0.21   | 1.54   | 79.4  | 1.95         | 0.01         | 4.63         | 79.4      | 5.83         |
| V(0,1)-4C     | 3.18        | 7148.51 | 0.28   | 1.27   | 44.2  | 2.87         | 0.12         | 2.09         | 44.2      | 4.73         |
| CMG           | 5.01        | 0.38    | 0.96   | 87.3   | 1.10  | 0.32         | 1.15         | 91.2         | 1.26      |              |

**Table 1:** Sample convergence results for three-level ( $L = 3$ ) preconditioners applied to a  $32 \times 32$  5-point stencil colorization problem. The convergence rates  $\kappa \dots \hat{\tau}$  are described in the section on convergence analysis and the algorithms are described in the accompanying text.  $\nu$  refers to the condition number of the coarsest level Hessian.

of our sample problems. The number of levels for each problem is fixed so that the coarsest level problem is always approximately the same size (consisting of between 4 to 8 unknowns). Table 3 summarizes empirical convergence rate  $\hat{\tau}$  for very large (multi megapixel) problems. The complete set of results is available in [Krishnan and Szeliski 2011].

In these tables and figures, the algorithms are denoted as follows:

- HBf: full-octave non-adaptive hierarchical basis preconditioning [Szeliski 1990];
- ABF-Pre7: half-octave adaptive hierarchical basis preconditioning [Szeliski 2006], where sparsification is applied before the interpolants and coarse-level Hessians are computed,
- ABF-Pre7-W: a W-cycle variant of ABF-Pre7;
- ABF-Pre7-4C: ABF-Pre7 with four-color Gauss-Seidel pre-smoothing and post-smoothing; this is the unified algorithm described in Appendix A with all components (smoothing, fine-level diagonal preconditioning and coarse-level solve) enabled;
- ABF-Pre7-4C-W: W-cycle variant of ABF-Pre7-4C;
- AMG-S: algebraic multigrid preconditioning with a fixed full-octave coarsening scheme and either a Jacobi (J) smoother or four-color Gauss-Seidel (4C) smoother; (our current implementation does not do full justice to AMG, since we were unable to find adaptive coarsening code);
- GMG-J: geometric multigrid preconditioning with full-octave

|               | HDR         |             |             | Poisson     |             |             | Membrane    |             |              | Color. 5-pt |             |             | Color. 9-pt |             |             | EPD         |             |
|---------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|--------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|
|               | 33<br>(4)   | 512<br>(8)  | 768<br>(7)  | 33<br>(4)   | 128<br>(6)  | 512<br>(7)  | 33<br>(4)   | 256<br>(7)  | 32<br>(4)    | 256<br>(7)  | 239<br>(6)  | 32<br>(4)   | 256<br>(7)  | 239<br>(6)  | 33<br>(4)   | 800<br>(8)  | 535<br>(7)  |
| HBf           | 3.35        | 2.09        | 2.78        | 3.54        | 2.80        | 2.41        | 0.94        | 0.7         | 0.92         | 1.18        | 0.93        | 0.96        | 1.06        | 0.82        | 1.97        | 1.78        | 1.37        |
| ABF-Pre7      | 5.42        | 2.81        | 2.88        | 5.39        | 4.48        | 4.83        | <b>6.41</b> | <b>5.19</b> | 8.75         | <b>9.97</b> | <b>5.65</b> | 4.55        | <b>4.17</b> | <b>3.68</b> | 3.82        | 3.09        | 2.54        |
| ABF-Pre7-W    | 3.15        | 2.42        | 2.56        | 3.41        | 2.86        | 2.42        | 3.98        | 3.15        | 2.72         | 3.43        | 0.39        | 1.22        | 0.65        | 0.43        | 0.57        | 0.25        | 0.26        |
| ABF-Pre7-4C   | 7.15        | <b>5.55</b> | 4.75        | 8.01        | <b>7.77</b> | <b>8.49</b> | 4.64        | 4.29        | <b>11.29</b> | 5.90        | 3.28        | 2.65        | 2.85        | 2.18        | 4.03        | 2.23        | 1.93        |
| ABF-Pre7-4C-W | 2.55        | 1.69        | 1.91        | 2.41        | 1.80        | 1.74        | 1.44        | 0.77        | 2.51         | 1.03        | 0.88        | 0.71        | 0.44        | 0.39        | 1.14        | 0.14        | 0.15        |
| AMG-J         | 4.37        | 3.20        | 3.73        | 4.82        | 4.40        | 4.46        | 3.99        | 3.32        | 2.77         | 2.23        | 1.10        | 3.13        | 1.83        | 1.07        | 1.47        | 0.93        | 0.72        |
| AMG-4C        | <b>7.53</b> | 4.56        | 4.95        | <b>8.25</b> | 6.73        | 7.21        | 5.35        | 4.29        | 5.58         | 3.30        | 1.68        | 4.02        | 2.75        | 1.17        | 1.91        | 1.19        | 0.92        |
| AMG-4C-W      | 4.66        | 5.32        | <b>5.16</b> | 5.21        | 5.25        | 5.19        | 3.94        | 3.30        | 3.74         | 2.95        | 1.81        | 3.34        | 2.68        | 1.70        | 1.57        | 1.28        | 1.16        |
| GMG-J         | 4.65        | 3.48        | 4.17        | 5.09        | 4.41        | 4.41        | 1.06        | 0.33        | 1.39         | 1.14        | 0.78        | 1.87        | 1.21        | 0.83        | 0.99        | 0.89        | 0.71        |
| V(1,1)-4C     | 6.95        | 4.59        | 4.93        | 7.67        | 5.03        | 6.34        | 2.93        | 3.68        | 5.76         | 2.93        | 1.43        | 3.97        | 2.22        | 1.00        | 0.93        | 0.82        | 0.39        |
| V(0,1)-4C     | 4.74        | 2.74        | 4.28        | 6.30        | 3.96        | 5.36        | 5.49        | 5.12        | 4.68         | 3.66        | 2.52        | <b>5.09</b> | 2.76        | 1.55        | 1.48        | 1.35        | 0.62        |
| CMG           | 1.36        | 1.16        | 1.25        | 1.92        | 1.87        | 2.18        | 1.82        | 1.97        | -            | 2.82        | 2.39        | 1.56        | 1.81        | 1.89        | <b>4.87</b> | <b>4.44</b> | <b>4.24</b> |

**Table 2:** Empirical effective convergence rates  $\hat{\tau}$  for preconditioners applied to all of our sample problems. The **boldface** numbers are the “winners” (fastest empirical convergence rate) in each column. For each problem, test results on two image sets and a small version of one of the sets are provided. The number of rows in the tested image is at the top of each column. Below that is the number of levels is given in parentheses.

|             | HDR<br>2048 × 1365<br>(10) |             | Poisson<br>2048 × 2048<br>(10) |             | Membrane<br>2048 × 2048<br>(10) |              | Color. 5-pt<br>1854 × 2048<br>(8) |             | Color. 9-pt<br>1854 × 2048<br>(8) |      | EPD<br>1365 × 2048<br>(8) |      |
|-------------|----------------------------|-------------|--------------------------------|-------------|---------------------------------|--------------|-----------------------------------|-------------|-----------------------------------|------|---------------------------|------|
| HBf         | 2.23                       | 2.23        | 2.58                           | 2.58        | 0.96                            | 0.41         | 0.41                              | 0.56        | 1.65                              | 1.65 | 2.60                      | 2.60 |
| ABF-Pre7    | 2.61                       | 2.61        | 5.52                           | 5.52        | 3.97                            | <b>20.35</b> | <b>16.48</b>                      | 7.56        | 1.74                              | 1.74 | 0.91                      | 0.91 |
| ABF-Pre7-4C | <b>5.75</b>                | <b>5.75</b> | <b>9.52</b>                    | <b>9.52</b> | 3.00                            | 12.17        | 7.56                              | 1.74        | 0.91                              | 0.91 | 0.91                      | 0.91 |
| AMG-J       | 2.93                       | 2.93        | 5.15                           | 5.15        | 3.35                            | 3.00         | 2.71                              | 0.60        | 0.87                              | 0.87 | 0.91                      | 0.91 |
| AMG-4C      | 4.16                       | 4.16        | 7.69                           | 7.69        | 3.33                            | 8.62         | 4.16                              | 0.87        | 0.91                              | 0.91 | 0.91                      | 0.91 |
| GMG-J       | 3.21                       | 3.21        | 4.98                           | 4.98        | 0.57                            | 1.05         | 0.98                              | 0.91        | 0.91                              | 0.91 | 0.91                      | 0.91 |
| V(1,1)-4C   | 4.06                       | 4.06        | 6.72                           | 6.72        | 3.45                            | 8.94         | 4.29                              | 0.37        | 0.91                              | 0.91 | 0.91                      | 0.91 |
| V(0,1)-4C   | 3.55                       | 3.55        | 5.48                           | 5.48        | <b>4.02</b>                     | 1.41         | 2.42                              | 0.59        | 0.91                              | 0.91 | 0.91                      | 0.91 |
| CMG         | -                          | -           | 2.29                           | 2.29        | 1.75                            | 5.58         | 3.55                              | <b>4.18</b> | 0.91                              | 0.91 | 0.91                      | 0.91 |

**Table 3:** Empirical effective convergence rates  $\hat{\tau}$  for preconditioners applied to all of our sample problems for multi-megapixel images. Next to problem size, the number of levels is given in parentheses. The **boldface** numbers are the “winners” (fastest empirical convergence rate) in each column. The size of the tested image is at the top of each column. Below that is the number of levels in parentheses.

bilinear interpolants and Jacobi smoothing;

- V(m,n)-4C: V-cycle multigrid with a fixed step size and four-color Gauss-Seidel  $m$  pre-smoothing and  $n$  post-smoothing steps; the interpolants and Hessians are computed using the same logic as for AMG-S; this variant tests the effectiveness of using stand-alone multigrid cycles instead of PCG;
- CMG: combinatorial multigrid, using the code provided by the authors of [Koutis et al. 2009].

There are other possible variants such as the original ABF 5-banded sparsification algorithm from [Szeliski 2006]. The new partial sparsification we introduce always works better than the original sparsification both empirically and theoretically, so we do not include these runs in our results.

Looking at the complete set of results available in the supplementary materials [Krishnan and Szeliski 2011], we see that the relative behavior of the various algorithms depends both on the amount of smoothness and the amount of inhomogeneity in the sample problems, i.e., how regular they are.

HDR and Poisson blending are homogeneous problems, ideally suited to geometric multigrid. Indeed GMG (and AMG, which reverts to GMG on uniform problems) have very fast convergence rate  $\tau$ . Furthermore,  $\tau$  is independent of the number of levels, as is predicted by multigrid theory. However, when we factor in the computational cost  $C$ , the effective convergence rate  $\hat{\tau}$  for the ABF-Pre7-4C is significantly better. Using the new partial (7-point stencil) sparsification rule consistently lowers the condition numbers and usually helps the empirical performance. Comparing the empirical rates for AMG-J and AMG-4C shows that the four-color smoother is a much better choice than Jacobi smoothing for all problems, but has an especially dramatic effect for the homogeneous problems.

The 2D membrane exhibits a moderate amount of inhomogeneity, including scattered data constraints and a strong discontinuity. Here, non-adaptive techniques such as HBF and GMG-J start to fall apart. ABF-Pre7 starts to pull away, with the additional smoothing

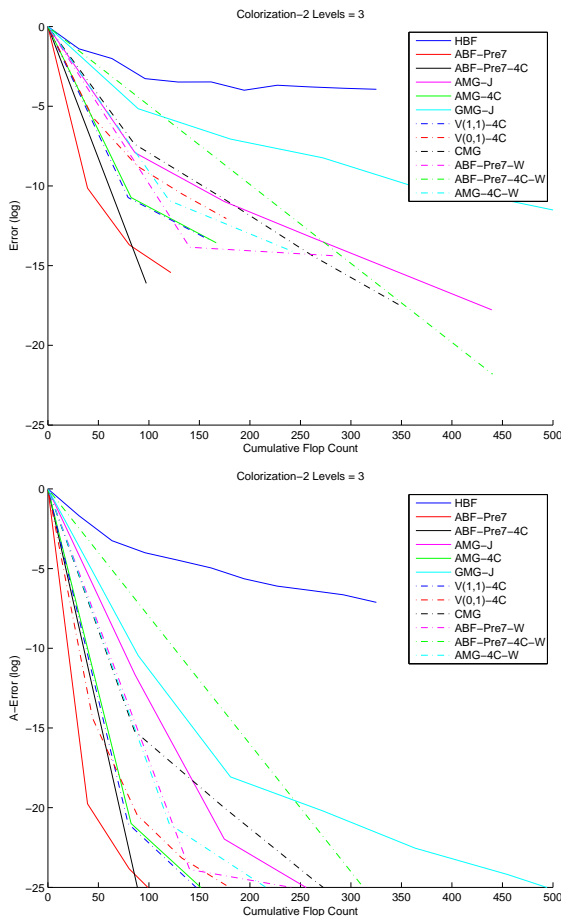
of ABF-Pre7-4C not providing significant improvement. AMG-4C is still competitive if we ignore computational costs. CMG, even though it’s designed for irregular problems, has slower convergence because it uses a piecewise-constant interpolator for what is essentially a mostly smooth problem.

The results on colorization (for both 5-point and 9-point fine-level stencil formulations) are similar to the 2D membrane, even though the number of discontinuities is quite a bit larger. Here, we would expect a full (adaptive coarsening) implementation of AMG-4C to do better, but we currently don’t have such an implementation to test. In terms of theoretical effective convergence rates, CMG is starting to look competitive, but still doesn’t do as well empirically.

The edge-preserving decomposition is much more challenging. Because of the large number of discontinuities and thinner elongated structures, CMG starts to perform very well, both in terms of theoretical condition numbers, and in terms of practical convergence rates. It will be interesting to compare its performance against true (adaptive coarsening) algebraic multigrid, and also to future extensions of our sparsifying approach that use adaptive coarsening schemes.

Note that for these problems, the theoretical convergence rates predicted by condition number analysis are much worse than the actual empirical convergence rates we observe. We believe the reason for this is the clustering of eigenvalues after preconditioning, whereas the theoretical condition number is computed using the largest and smallest eigenvalues, which may be outliers.

The unified algorithm ABF-Pre7-4C performs well on all the problems, even after taking into account computational load. Further research into combining adaptive coarsening with the unified algorithm may result in better performance for the less smooth problems such as edge-preserving decomposition.



**Figure 7:** Sample convergence plot for three-level ( $L = 3$ ) preconditioners applied to a  $32 \times 32$  5-point stencil colorization problem. Top: The horizontal axis plots the number of floating point operations (flops) performed per input (fine-level) variable, while the vertical axis plots the log error between the current and optimal solutions; Bottom: plot of flops against A-norm of error (defined as  $e^T A e$  where  $e$  is error and  $A$  is the Hessian).

## 8 Discussion and Conclusions

Our experimental results demonstrate that with our new extension to partial sparsification, adaptive hierarchical basis function preconditioning (ABF) outperforms traditional multigrid techniques on a wide range of computer graphics and computational photography problems. This runs contrary to the usual belief in the multigrid community that appropriate smoothing between level transfers is necessary for reasonable performance. Traditional adaptive hierarchical basis functions achieve slower (but still reasonable) convergence rates without using any smoothing steps, but use many fewer computations per level because they do not require any smoothing (other than that implicit in the recomputation of the residual at the beginning of each iteration).

For irregular problems, such as edge-preserving decomposition, adaptive coarsening schemes such as combinatorial and (adaptive) algebraic multigrid start to become important. Our experiments also show that taking into account only the number of iterations without incorporating flop counts gives a misleading picture of relative performance. However, we note that flop counts are themselves not a complete predictor of actual performance, since issues such as caching and memory access patterns will be important in

high-performance implementations of these algorithms.

Recently, a number of fast edge-aware smoothing techniques have been developed for solving tone mapping and colorization problems [Fattal 2009; Gastal and Oliveira 2011]. These techniques have linear complexity in the number of pixels and have extremely fast computational performance. However, the best-performing preconditioning methods in our paper also converge in a single iteration for tone mapping and colorization. Therefore the effective time for processing is the multilevel pyramid setup time and a single iteration of (preconditioned) conjugate gradient. Both these steps are a small constant multiple of the number of pixels in the image and are highly parallel operations. This leads us to believe that a carefully optimized GPU version of our MATLAB code would give very fast computational performance as well for the tone mapping and colorization algorithms.

For the other algorithms such as Poisson blending and edge-preserving decomposition, solving a large linear system is a necessity. Our MATLAB code has been written explicitly with GPU implementation in mind, since such an implementation would be of significant practical interest to the graphics and computer vision communities.

Currently, our adaptive hierarchical bases are defined on a fixed half-octave grid. However, we believe that the coarsening heuristics used in AMG could be adapted to our bases. The sparsification step used in constructing the adaptive interpolants requires a decomposition into coarse and fine variables, where the connections between fine variables (which need to be sparsified) are no stronger than those to their corresponding coarse-level “parents”. AMG coarsening strategies perform a very similar selection of nodes.

Another promising direction for future research is the extension to three-dimensional and more unstructured meshes. The advances available by combining all of these ideas together should lead to even more efficient techniques for the solution of large-scale two- and three-dimensional computer graphics and computational photography problems.

## References

- AGARWALA, A., ET AL. 2004. Interactive digital photomontage. *ACM Transactions on Graphics (Proc. SIGGRAPH 2004)* 23, 3 (August), 292–300.
- AGARWALA, A. 2007. Efficient gradient-domain compositing using quadrees. *ACM Transactions on Graphics (Proc. SIGGRAPH 2007)* 26, 3 (August), 94:1–94:5.
- BRIGGS, W. L., HENSON, V. E., AND MCCORMICK, S. F. 2000. *A Multigrid Tutorial*, second ed. Society for Industrial and Applied Mathematics, Philadelphia.
- 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 2008)* 27, 3 (August).
- FARBMAN, Z., HOFFER, G., LIPMAN, Y., COHEN-OR, D., AND LISCHINSKI, D. 2009. Coordinates for instant image cloning. *ACM Transactions on Graphics (Proc. SIGGRAPH 2009)* 28, 3 (August).
- FATTAL, R., LISCHINSKI, D., AND WERMAN, M. 2002. Gradient domain high dynamic range compression. *ACM Transactions on Graphics* 21, 3 (July), 249–256.



FATTAL, R. 2009. Edge-avoiding wavelets and their applications. *ACM Transactions on Graphics (Proc. SIGGRAPH 2009)* 28, 3 (August).

GASTAL, E. S. L., AND OLIVEIRA, M. M. 2011. Domain transform for edge-aware image and video processing. *ACM Transactions on Graphics (Proc. SIGGRAPH 2011)* 30, 4 (July).

GORTLER, S. J., AND COHEN, M. F. 1995. Hierarchical and variational geometric modeling with wavelets. In *Symposium on Interactive 3D Graphics*, 35–43.

JESCHKE, S., CLINE, D., AND WONKA, P. 2009. A GPU Laplacian solver for diffusion curves and Poisson image editing. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia 2009)* 28, 5 (December).

KAZHDAN, M., SURENDRAN, D., AND HOPPE, H. 2010. Distributed gradient-domain processing of planar and spherical images. *ACM Transactions on Graphics* 29, 2 (April).

KOUTIS, I., MILLER, G. L., AND TOLLIVER, D. 2009. Combinatorial preconditioners and multilevel solvers for problems in computer vision and image processing. In *5th International Symposium on Visual Computing (ISVC09)*, Springer.

KRISHNAN, D., AND SZELISKI, R. 2011. Multigrid and multilevel preconditioners for computational photography. Tech. Rep. NYU-TR-941, New York University, October.

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

LEVIN, A., LISCHINSKI, D., AND WEISS, Y. 2004. Colorization using optimization. *ACM Transactions on Graphics* 23, 3 (August), 689–694.

LEVIN, A., ZOMET, A., PELEG, S., AND WEISS, Y. 2004. Seamless image stitching in the gradient domain. In *Eighth European Conference on Computer Vision (ECCV 2004)*, Springer-Verlag, vol. IV, 377–389.

LEVIN, A., LISCHINSKI, D., AND WEISS, Y. 2008. A closed-form solution to natural image matting. *IEEE Trans. on Pattern Analysis and Machine Intelligence* 30, 2 (Feb.), 228–242.

MCADAMS, A., SIFAKIS, E., AND TERAN, J. 2010. A parallel multigrid Poisson solver for fluids simulation on large grids. *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 65–74.

MCCANN, J., AND POLLARD, N. 2008. Real-time gradient domain painting. *ACM Transactions on Graphics (Proc. SIGGRAPH 2008)* 27, 3 (August).

NAPOV, A., AND NOTAY, Y. 2011. Algebraic analysis of aggregation-based multigrid. *Numerical Linear Algebra with Applications* 18, 3 (May), 539–564.

PENTLAND, A. P. 1994. Interpolation using wavelet bases. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 16, 4 (April), 410–414.

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

ROBERTS, A. 2001. Simple and fast multigrid solutions of Poisson’s equation using diagonally oriented grids. *ANZIAM J.* 43 (July), E1–E36.

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

SHEWCHUK, J. R. 1994. An introduction to the conjugate gradient method without the agonizing pain. <http://www.cs.berkeley.edu/~jrs/>, August.

SZELISKI, R., UYTENDAELE, M., AND STEEDLY, D. 2011. Fast Poisson blending using multi-splines. In *International Conference on Computational Photography (ICCP 11)*.

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

SZELISKI, R. 2006. Locally adapted hierarchical basis preconditioning. *ACM Transactions on Graphics (Proc. SIGGRAPH 2006)* 25, 3 (August), 1135–1143.

TERZOPOULOS, D. 1986. Image analysis using multigrid relaxation methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-8*, 2 (March), 129–139.

TROTTEMBERG, U., OOSTERLEE, C. W., AND SCHULLER, A. 2000. *Multigrid*. Academic Press.

WANG, H., RASKAR, R., AND AHUJA, N. 2004. Seamless video editing. *Proceedings of the International Conference on Pattern Recognition*, 858–861.

## A Unified multilevel multigrid preconditioner

---

### Algorithm 1 Unified multigrid/multilevel algorithm

---

$[e_l] = \text{MGCYC}(l, r_l, \{A_1 \dots A_L\}, \{S_1 \dots S_L\}, MG)$   
 INPUT: Current level  $l$ , residual  $r_l$ , per-level Hessians  $A_l$ , per-level interpolation matrices  $S_l$ .  
 MG is a parameter structure that contains:  
 number of levels  $MG.L$ , damping factor  $MG.\omega$ ,  
 pre- and post-smoothing iterations  $MG.\nu^{pre}$   
 and  $MG.\nu^{post}$ , number of cycles  $MG.\gamma$ ,  
 flag for optional diagonal preconditioning  $MG.d$   
 OUTPUT: Correction at level  $l$   $e_l$

1. // Pre-smoothing correction  
 $e_l^{pre} = \text{Smooth}(0, A_l, r_l, \omega, \nu^{pre})$
2. // Update the residual  
 $\tilde{r}_l = r_l - A_l e_l^{pre}$
3. // Restrict residual to coarse level  
 $\tilde{r}_{l+1} = S_l^T \tilde{r}_l$
4. if  $l = L - 1$
5.    $e_{l+1} = A_L^{-1} \tilde{r}_{l+1}$
6. else
7.    $e_{l+1} = 0$
8.   //  $\gamma = 1$  is V-cycle;  $\gamma = 2$  is W-cycle  
   for  $j = 1, \dots, \gamma$
9.    // Update the residual  
      $\tilde{r}_{l+1} = \tilde{r}_{l+1} - A_{l+1} e_{l+1}$
10.     $\tilde{e}_{l+1} = \text{MGCYC}(l+1, \tilde{r}_{l+1}, \{A_1 \dots A_L\}, \{S_1 \dots S_L\}, MG)$
11.    // Add up corrections over the cycles  
      $e_{l+1} \leftarrow e_{l+1} + \tilde{e}_{l+1}$
12.   endfor
13. endif
14. // Prolong coarse-grid correction  
 $e_l^{cgc} = S_l e_{l+1}$
15. // Optional fine-level diagonal preconditioning  
   if ( $d = 1$ )
16.    // Precondition with inverse diagonal elements of  $A_l$ .  
      $e_l^d = \text{DiagPrecond}(\tilde{r}_l, A_l)$
17. else
18.    // No diagonal preconditioning  
      $e_l^d = 0$
19. endif
20. // Add up corrections: pre-smooth, coarse-level, and diagonal  
 $e_l^{sum} = e_l^{pre} + e_l^{cgc} + e_l^d$
21. // Post-smoothing  
 $e_l = \text{Smooth}(e_l^{sum}, A_l, r_l, \omega, \nu^{post})$

---