Locally Adapted Hierarchical Basis Preconditioning

Richard Szeliski

Microsoft Research

May 2006

Technical Report

MSR-TR-2006-38

This paper develops locally adapted hierarchical basis functions for effectively preconditioning large optimization problems that arise in computer vision, computer graphics, and computational photography applications such as surface interpolation, optic flow, tone mapping, gradient-domain blending, and colorization. By looking at the local structure of the coefficient matrix and performing a recursive set of variable eliminations, combined with a simplification of the resulting coarse level problems, we obtain bases better suited for problems with inhomogeneous (spatially varying) data, smoothness, and boundary constraints. Our approach removes the need to heuristically adjust the optimal number of preconditioning levels, significantly outperforms previous approaches, and also maps cleanly onto data-parallel architectures such as modern GPUs. *[Errata in (8) and (9) fixed October, 2007]*

Microsoft Research

Microsoft Corporation

One Microsoft Way

Redmond, WA 98052

`http://www.research.microsoft.com`

1 Introduction

A large number of computer graphics problems can be formulated as the solution of a set of spatially-varying linear or non-linear partial differential equations (PDEs). Such problems include computational photography algorithms such as high-dynamic range tone mapping (Fattal *et al.* 2002), Poisson and gradient-domain blending (Pérez *et al.* 2003, Levin *et al.* 2004, Agarwala *et al.* 2004), colorization (Levin *et al.* 2004), and natural image matting (Levin *et al.* 2006). They also include various physically-based modeling problems such as elastic surface dynamics (Terzopoulos and Witkin 1988), as well as the regularized solution of computer vision problems such as surface reconstruction, optic flow, and shape from shading (Poggio *et al.* 1985, Terzopoulos 1986, Brox *et al.* 2004).

The discretization of these continuous variational problems using finite element or finite difference techniques yields large sparse systems of linear equations. Direct methods for sparse systems (Duff *et al.* 1986) become inefficient for large multi-dimensional problems because of excessive amounts of *fill-in* that occurs during the solution. In most cases, iterative solution are a better choice (Saad 2003). Unfortunately, as the problems become larger, the performance of the iterative algorithms usually degrades due to the increased condition number of the associated systems.

Two different classes of techniques, both based on multi-level (hierarchical) representations, have traditionally been used to accelerate the convergence of iterative techniques (Saad 2003). The first are multigrid techniques (Briggs *et al.* 2000), which interpolate between different levels of resolution in an effort to alternately reduce the low- and high-frequency components of the error. Unfortunately, these techniques work best for smoothly varying (homogeneous) problems, which is often not the case with low-level vision algorithms.

The second class of techniques are optimization algorithms such as conjugate gradient preconditioned using a variety of parallel or multi-level techniques (Yserentant 1986, Szeliski 1990c, Saad 2003). Among these techniques, multi-level preconditioners such as hierarchical basis functions and wavelets (Yserentant 1986, Szeliski 1990c, Pentland 1994, Gortler and Cohen 1995, Yaou and Chang 1994, Lai and Vemuri 1997) are a particularly good match to vision problems because they exploit the natural multi-scale nature of many visual problems.

Another big advantage of such techniques is their suitability for implementation on data-parallel architectures. Older work on low-level vision often stressed this connection (Poggio 1985), and several low-level algorithms were ported to data parallel architectures such as the Connection Machine (Poggio *et al.* 1988). While such computers were very specialized and have mostly disappeared, the last few years has seen the re-emergence of data-parallel computation with the advent of programmable graphical processing units (GPUs), which can now be found in almost every computer. In fact, many vision and numerical analysis algorithms have recently been ported to GPUs (Yang and Pollefeys 2003, Bolz *et al.* 2003).

While multi-level preconditioners have proven to be quite effective at accelerating the solution of many low-level vision problems (Szeliski 1990c, Lai and Vemuri 1997), some problems remain. First, the choice of basis functions and the number of levels is problem-dependent and is still mostly a heuristic art (Szeliski 1990c). Modulating the size of the basis functions based on the local stiffness of the regularized problem can help quite a bit, especially in reducing the need to find an optimal number of preconditioning levels (Lai and Vemuri 1997). However, this requires the temporal adjustment of a weighting function $\sigma(i)$ which blends between the data-dependent and smoothness-dependent modulation functions (Lai and Vemuri 1997). Furthermore, these algorithms perform poorly on problems with large amounts of inhomogeneity, such as local discontinuities or irregularly spaced data, or irregularly shaped boundaries.

In this paper, we go back to basics and ask the question: is there an *optimal* set of hierarchical basis functions that can perfectly precondition an arbitrary multi-dimensional variational problem? It turns out that for a one-dimensional first order system, we can construct such a multi-resolution basis. For higher order or higher-dimensional problems, we can use a similar methodology to derive locally adapted hierarchical basis functions that *approximate* an optimal preconditioning of arbitrary problems and significantly outperform previously published techniques. The key to our approach is to carefully examine the the local structure of the coefficient matrix to perform a recursive set of variable eliminations combined with a simplification of the resulting coarse level problems.

We begin this paper with a review of variational problems and their discretization (Section 2).

In Section 3, we review previously developed techniques, both direct and iterative, for solving such linear systems, including the use of multigrid and multi-level preconditioners. In Section 4, we derive optimal hierarchical bases for the one-dimensional first order problem. In Section 5, we develop a methodology for constructing locally adaptive hierarchical bases for two-dimensional first order problems such as Poisson image blending, HDR tone mapping, and colorization. We also discuss how to map these algorithms onto GPUs. Section 6 presents some experimental results, including comparisons with previous approaches, while Section 7 discusses areas for further research.

2 Problem formulation

Many problems in low-level computer vision can be formulated using a *variational approach*, which involves defining a continuous two-dimensional optimization problem, adding appropriate *regularization terms*, and then discretizing the problem on a regular grid (Terzopoulos 1983, Poggio *et al.* 1985, Horn and Brooks 1986). There are also deep connections between such approaches and the use of Markov Random Fields as priors for low-level inference problems (Szeliski 1990a).

Variational approaches were first applied in computer vision to solve surface interpolation problems (Terzopoulos 1983), including surfaces with first and second order discontinuities (tears and creases) (Terzopoulos 1986, Zhang *et al.* 2002). They were also applied to problems such as optic flow (Horn and Brooks 1989, Brox *et al.* 2004), shape from shading (Horn and Brooks 1986, Szeliski 1990b), stereo matching (Poggio *et al.* 1985), and edge detection (Poggio *et al.* 1985).

More recently, similar approaches have been used in computational photography to perform various image editing operations such as single view modeling (Zhang *et al.* 2002), seam hiding (Pérez *et al.* 2003, Levin *et al.* 2004, Agarwala *et al.* 2004), high dynamic range tone mapping (Fattal *et al.* 2002), and colorization (Levin *et al.* 2004). What all of these problems have in common is that the discretization of these variational formulations leads to quadratic forms with large sparse symmetric positive definite (SPD) coefficient (a.k.a. Hessian or stiffness) matrices that have a natural embedding in a two-dimensional grid and are hence amenable to multi-level approaches.[1]

2.1 1D problems

Let us begin with the simplest possible variational problem, namely the controlled continuity (discontinuity preserving) approximating spline. The problem is to reconstruct a one-dimensional function $f(x)$ that approximately interpolates a set of data points $\{d_k\}$ at locations $\{x_k\}$ with associated weights $\{w_k\}$.[2] To make the problem well-posed, we add a regularization term, namely a smoothness functional, which can either be first order,

$$\mathcal{E}_s = \int s(x) f_x^2(x) dx, \tag{1}$$

or second order

$$\mathcal{E}_c = \int c(x) f_{xx}^2(x) dx, \tag{2}$$

or some blend between the two. The spatially varying *smoothness* and *curvature* functions $s(x)$ and $c(x)$ determine the amount of smoothness imposed and the locations of tears ($s(x) = 0$) and creases ($c(x) = 0$). For the remainder of this paper, we focus on first order problems and leave the solution of second order problems to future work.

To discretize the above problem, we choose a regular grid spacing of size h and place all of our data constraints at discrete grid locations, which results in the discrete energy

$$E_{\mathrm{d}} = \sum_i w_i (f_i - d_i)^2, \tag{3}$$

with $w_i = 0$ where there are no data points. Figure 1 shows a sample set of input data values d_i and weights w_i, which we can think of points pulling on the final surface f_i with strong springs. (Weaker springs connect the remaining points to the 0 baseline.) The smoothness function s_i is constant everywhere except for one "tear" at $s_{11} = 0$.

[1]We will not deal in this paper with vision problems such as stereo and shape from shading that have non-quadratic energies, although our techniques could be extended in this direction (Szeliski 1990b).

[2]We can set $w_k \to \infty$ to implement a *hard constraint* and our preconditioner can deal with this case.

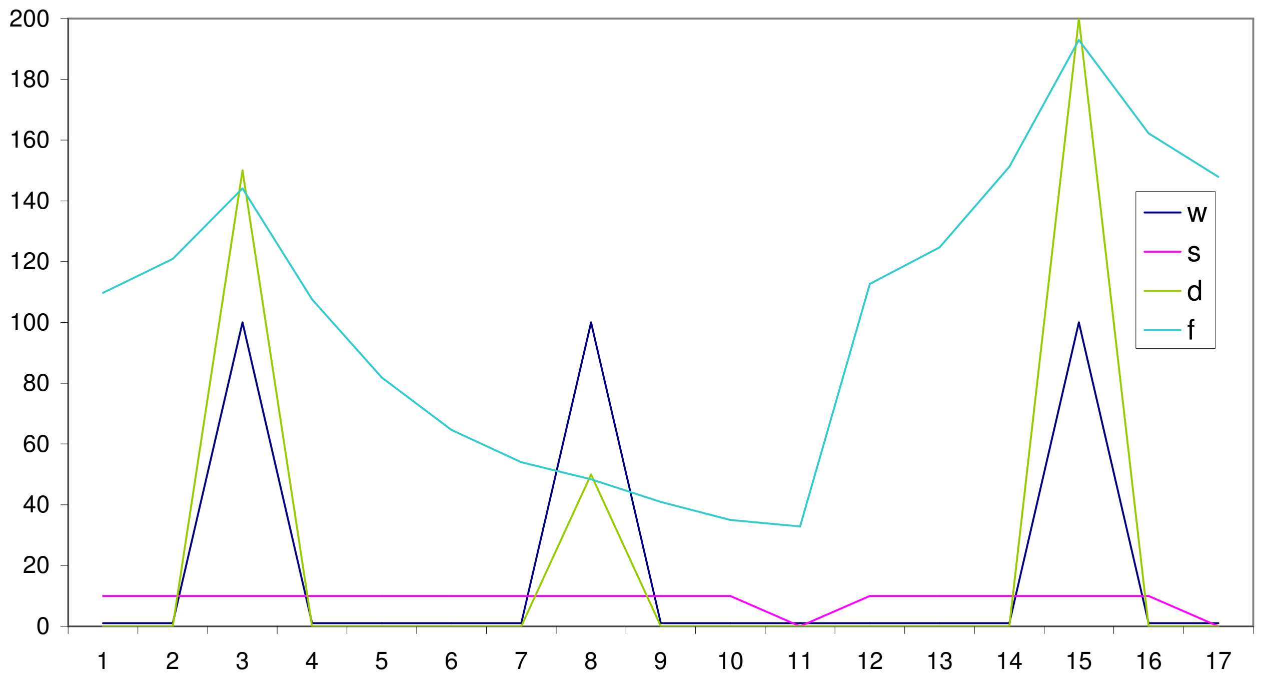


Figure 1: *Input data and output solution. The input data values are* $d_i = \{150, 50, 200\}$ *at* $i = \{3, 8, 15\}$ *and 0 elsewhere. The data weights are* $w_i = 100$ *at* $i = \{3, 8, 15\}$ *and 1 elsewhere, which causes the interpolated curve* f *to sag towards 0. The smoothness function is* $s_i = 10$ *except for* $s_{11} = 0$*, which causes a "tear" in the reconstructed* f *curve between* $i = \{11, 12\}$*. (The tear is not readily apparent because of the linear interpolation used by the plotting software.)*

To discretize the smoothness functional, we can use either a finite difference or a finite element approach. In finite differences, we replace the first order derivative term with a collection of $(f_{i+1} - f_i)$ terms, each of which becomes squared and weighted independently to obtain

$$E_{\rm s} = \sum_i s_i (f_{i+1} - f_i)^2. \tag{4}$$

The finite element approach (Bathe and Wilson 1976) models the function $f(x)$ as a piecewise linear spline, i.e.,

$$f(x) = f_i + \frac{(x - x_i)}{h}[f_{i+1} - f_i], x \in [x_i, x_{i+1}] \tag{5}$$

(Terzopoulos 1983). After analytically integrating the energy functional $\mathcal{E}_s$, we obtain the same formula as (4), with the value for s_i given by $s_i = h^{-2} \int_{x_i}^{x_{i+1}} s(x)dx = s/h$, for a constant $s(x) = s$.

Thus, at first glance, it appears that the finite element approach holds no real advantage over finite differences. However, it makes the dependence of the energy on the grid size h explicit, which makes it easier to reason about relationships between different discretizations and discontinuities.

The second order *curvature* functional can similarly be discretized as

$$E_{\rm c} = \sum_i c_i (f_{i+1} - 2f_i + f_{i-1})^2. \tag{6}$$

In this case, the finite element approach to discretizing the second derivative functional is to approximate $f(x)$ with a piecewise quadratic spline (Terzopoulos 1983). Observing that for the quadratic spline,

$$f_{xx} = \frac{f_{i+1} - 2f_i + f_{i-1}}{2h^2},$$

we obtain for constant $c(x)$ the relationship $c_i = h^{-3}c/4$.

2.2 2D problems

Two-dimensional variational problems can be formulated in a similar manner. Again, we wish to reconstruct a function $f(x, y)$ through a set of data points $\{d_k\}$ at locations $\{(x_k, y_k)\}$ with associated weights $\{w_k\}$. The two-dimensional first order smoothness functional (*membrane* model

(Terzopoulos 1983)) can be written as

$$\mathcal{E}_s = \int s^x(x,y) f_x^2(x,y) + s^y(x,y) f_y^2(x,y) dx\, dy, \tag{7}$$

while the second order *thin plate* model is

$$\mathcal{E}_s = \int c(x,y)[f_{xx}^2(x,y) + 2f_{xy}^2(x,y) + f_{yy}^2(x,y)] dx\, dy, \tag{8}$$

where the mixed $2f_{xy}^2$ term is needed to make the energy rotationally symmetric. Discretizing these functionals leads to a first order *smoothness* term,

$$E_{\text{s}} = \sum_{i,j} s_{i,j}^x (f_{i+1,j} - f_{i,j} - g_{i,j}^x)^2 + s_{i,j}^y (f_{i,j+1} - f_{i,j} - g_{i,j}^y)^2 \tag{9}$$

and a second order *curvature* term

$$\begin{aligned} E_{\text{c}} &= \sum_{i,j} c_{i,j}^x (f_{i+1,j} - 2f_{i,j} + f_{i-1,j})^2 \\ &\quad + 2c_{i,j}^m (f_{i+1,j+1} - f_{i+1,j} - f_{i,j+1} + f_{i,j})^2 \\ &\quad + c_{i,j}^y (f_{i,j+1} - 2f_{i,j} + f_{i,j-1})^2. \end{aligned} \tag{10}$$

The data values $g_{i,j}^x$ and $g_{i,j}^y$ are gradient data terms (constraints) used by algorithms such as HDR tone mapping, Poisson blending, and gradient-domain blending (Fattal *et al.* 2002, Pérez *et al.* 2003, Levin *et al.* 2004). (They are 0 when just discretizing the conventional first order smoothness functional (7).) Note how separate smoothness and curvature terms can be imposed in the x, y, and mixed directions to produce local tears or creases (Terzopoulos 1988, Szeliski 1990a). The two dimensional discrete data energy is written as

$$E_{\text{d}} = \sum_{i,j} w_{i,j} (f_{i,j} - d_{i,j})^2. \tag{11}$$

To obtain the discrete energies using finite element analysis, Terzopoulos uses piecewise linear right-angled *triangular* spline elements for the membrane model and square quadratic patches for the second order thin-plate model (Terzopoulos 1983). Because the squared derivatives get integrated over local two-dimensional patches of area h^2, the discrete first order squared smoothness terms become independent of h, while the second order terms scale as h^{-2}.

For both the one and two-dimensional problems, the total energy of the discretized problem can be written as a *quadratic form*

$$E = E_{\mathrm{d}} + E_{\mathrm{s}} = \boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x} - 2\boldsymbol{x}^T \boldsymbol{b} + c, \tag{12}$$

where $\boldsymbol{x} = [f_0 \dots f_{n-1}]$ is called the *state vector*.[3]

The sparse symmetric positive-definite matrix $\boldsymbol{A}$ is called the *Hessian* since it encodes the second derivative of the energy function.[4] For the one-dimensional first order problem, $\boldsymbol{A}$ is tridiagonal, while for the two-dimensional problem, it is multi-banded with 5 non-zero entries per row. We call $\boldsymbol{b}$ the *weighted data vector*. Minimizing the above quadratic form is equivalent to solving the sparse linear system

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}. \tag{13}$$

3 Direct and iterative solution techniques

The solution of sparse positive definite (SPD) linear systems of equations breaks down into two major classes of algorithms: direct and iterative.

Direct methods use sparse matrix representations together with stable factorization algorithms such as LU (lower-upper) or Cholesky decomposition to obtain sparse symmetric factor matrices for which forward and backward substitution can be efficiently performed (Duff *et al.* 1986). Unfortunately, for two-dimensional problems, the factored matrices suffer from *fill-in*, i.e., the zero entries separating the main diagonal band and the off-diagonal bands (which result when the 2D variables are placed in raster order) get replaced with non-zero values. Thus, for two-dimensional problems discretized on an $n \times m$ grid, the cost of linear system solution is $O(nm^3)$ rather than the $O(n)$ complexity of one-dimensional direct solution techniques. Furthermore, direct solvers, being sequential in nature, are not directly amenable to data-parallel execution on GPUs.

[3]We use $\boldsymbol{x}$ instead of $\boldsymbol{f}$ because this is the more common form in the numerical analysis literature (Golub and Van Loan 1996).

[4]In numerical analysis, $\boldsymbol{A}$ is called the *coefficient* matrix (Saad 2003), while in finite element analysis (Bathe and Wilson 1976), it is called the *stiffness* matrix.

Iterative relaxation algorithms such as gradient descent, successive over-relaxation (SOR), and conjugate gradient descent minimize the quadratic energy function (12) using a series of steps that successively refine the solution by reducing its energy (Saad 2003). While in theory, conjugate gradient descent requires $N = nm$ steps to converge to the final solution, in practice, the rate of convergence depends on the *condition number* of the coefficient matrix $\boldsymbol{A}$, which depends on the ratio of its minimum and maximum eigenvalues (Saad 2003). Basic iterative algorithms can be accelerated using a variety of techniques, including multigrid, hierarchical basis preconditioning, and tree-based preconditioning.

3.1 Multigrid

A more intuitive way to analyze the convergence of these iterative algorithms is to note that gradient descent is equivalent to local relaxation, which effectively reduces the high-frequency component of the error, while being less effective at reducing the low-frequency error (Briggs *et al.* 2000). Based on this observation, *multigrid* methods have been developed which alternate between solving the problem on the original (fine) level and projecting the error to a coarser level where the lower-frequency components can be reduced (Briggs *et al.* 2000, Szeliski and Terzopoulos 1989, Saad 2003). This alternation between fine and coarse levels can be extended recursively by attacking the coarse level problem with its own set of inter-level transfers, which results in the commonly used *V-cycle* and *W-cycle* algorithms (Briggs *et al.* 2000). Multigrid techniques were first introduced to the vision community by Terzopoulos (Terzopoulos 1983) and are now commonly used to solve newer problems such as blending and colorization (Fattal *et al.* 2002, Pérez *et al.* 2003, Levin *et al.* 2004, Levin *et al.* 2004). The algorithm has also been ported to GPUs (Bolz *et al.* 2003).

While multigrid techniques are provably optimal for simple homogeneous problems, their performance degrades as the problem becomes more irregular or inhomogeneous (Saad 2003). Algebraic multigrid solvers (AMG), which locally adapt the interpolation functions to the local structure of the coefficient matrix, can be more effective in these cases (Briggs *et al.* 2000, Saad 2003). However, to our knowledge, AMG solvers have not previously been applied to computer graphics and computer vision problems. The techniques we develop are related to AMG (as we discuss later), although we use preconditioned conjugate gradient as our basic algorithm rather than multigrid, since it can compensate for the imperfect coarsening that is endemic in most real-world problems (Saad 2003).

3.2 Hierarchical basis preconditioning

An alternative approach to solving sparse multi-dimensional relaxation problems also draws its inspiration from multi-level techniques. However, instead of solving a series of multi-resolution sub-problems interspersed with inter-level transfers, preconditioned conjugate gradient uses the multi-level hierarchy to *precondition* the original system, i.e., to make the search directions more independent and better scaled. These techniques were first developed in the numerical analysis community by Yserentant (1986) and applied to computer vision problems by Szeliski (1990c). More recent versions using different basis functions (such as wavelets) and some amount of local adaptation have been proposed by Pentland (1994), Yaou and Chang (1994), Gortler and Cohen (1995), and Lai and Vemuri (1997).

The basic idea is to replace the original *nodal* variables $\boldsymbol{x} = [f_0 \ldots f_{n-1}]$ with a set of *hierarchical* variables $\boldsymbol{y}$ that lie on a multi-resolution pyramid, which has been sub-sampled so that it has the same number of samples as the original problem (Figure 2) (Szeliski 1990c). Because some of these new variables have larger bases (influence regions), the solver can take larger low-frequency steps as it iteratively searches for the optimal solution.

To convert between the hierarchical (multi-resolution) and nodal bases, we use a set of local interpolation steps,

$$\boldsymbol{x} = \boldsymbol{S}\boldsymbol{y} = \boldsymbol{S}_1\boldsymbol{S}_2 \cdots \boldsymbol{S}_{L-1}\boldsymbol{y}. \tag{14}$$

The columns of $\boldsymbol{S}$ can be interpreted as the *hierarchical basis functions* corresponding to the nodal variables (Yserentant 1986, Szeliski 1990c). In the wavelet community, this operation is called the *lifting scheme* for *second generation wavelets* (Schröder and Sweldens 1995, Sweldens 1997). The form of the individual interlevel interpolation functions $\boldsymbol{S}_l$ is usually similar to that used in multigrid, e.g., linear interpolation for one-dimensional problems and bilinear interpolation for

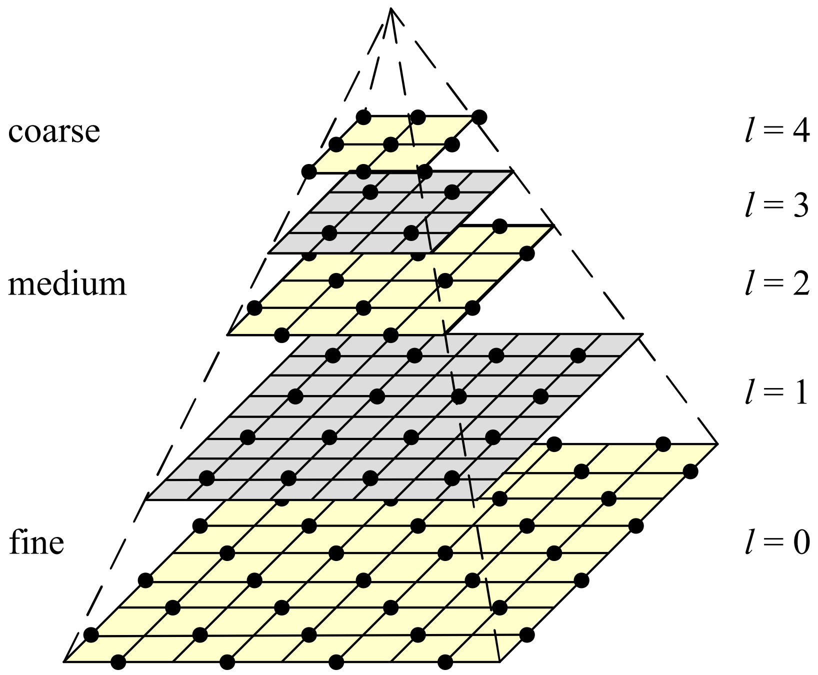


Figure 2: *Multiresolution pyramid with half-octave (*quincunx *sampling (odd levels are colored gray for easier visibility). Hierarchical basis function control variables are shown as black dots.*

two-dimensional problems. Figure 3 shows a set of hierarchical bases that correspond to using uniform linear interpolation at each stage.

The hierarchical bases are used to transform the original quadratic form in $\boldsymbol{x}$, (12), into a new form in $\boldsymbol{y}$,

$$E' = \boldsymbol{y}^T(\boldsymbol{S}^T\boldsymbol{A}\boldsymbol{S})\boldsymbol{y} - 2\boldsymbol{y}^T(\boldsymbol{S}^T)\boldsymbol{b} + c = \boldsymbol{y}^T\hat{\boldsymbol{A}}\boldsymbol{y} - \hat{\boldsymbol{b}} + c. \tag{15}$$

The advantage of performing this transformation is that the condition number of the matrix $\hat{\boldsymbol{A}}$ is much smaller than that of $\boldsymbol{A}$ (Szeliski 1990c). The matrix $\hat{\boldsymbol{A}}$ is not nearly as sparse as the original matrix $\boldsymbol{A}$, but this does not impact the performance of the conjugate gradient descent algorithm, since $\hat{\boldsymbol{A}}$ is never explicitly formed. Instread, the *search direction* for the origional (unpredonditioned) problem is multiplied by $\boldsymbol{S}$ and then $\boldsymbol{S}^T$, each of which can be implemented as a recursive set of coarsening or interpolation steps. (See (Szeliski 1990c) for a full derivation and pseudocode of the preconditioned conjugate gradient algorithm.)

Hierarchical basis preconditioning often outperforms multigrid relaxation on the kind of non-uniform scattered data interpolation problems that arise in low-level vision (Szeliski 1990c). However, as more levels of preconditioning are used, the performance starts to degrade once the hierarchical bases become larger than the "natural" local spacing of the data (Figure 5). This is symptomatic of a more general problem observed by Lai and Vemuri, namely that regular HBFs ignore the structure of the coefficient matrix $\boldsymbol{A}$ in formulating the bases (Lai and Vemuri 1997).

In their paper, Lai and Vemuri show how to exploit this information by *modulating* the size of the basis function at each level based on the spectral (frequency) characteristics of the smoothing operator (*regularization filter*), which depends on the relative values of the data weighting, first order smoothness, and curvature terms w, s, and c. This results in an algorithm that significantly outperforms the technique originally proposed in (Szeliski 1990c) and does not require the tuning of the number of levels. However, the *shape* of the basis functions themselves is still determined heuristically and does not adapt to the local continuity structure of the underlying problem. (See, however, (Szeliski 1990c, Zhang *et al.* 2002) for some heuristic rules to adapt the interpolation functions to local discontinuities.) Because the spectral estimation in (Lai and Vemuri 1997) is based on a wavelet transform of the *diagonal* of the data stiffness matrix, they also need to modify

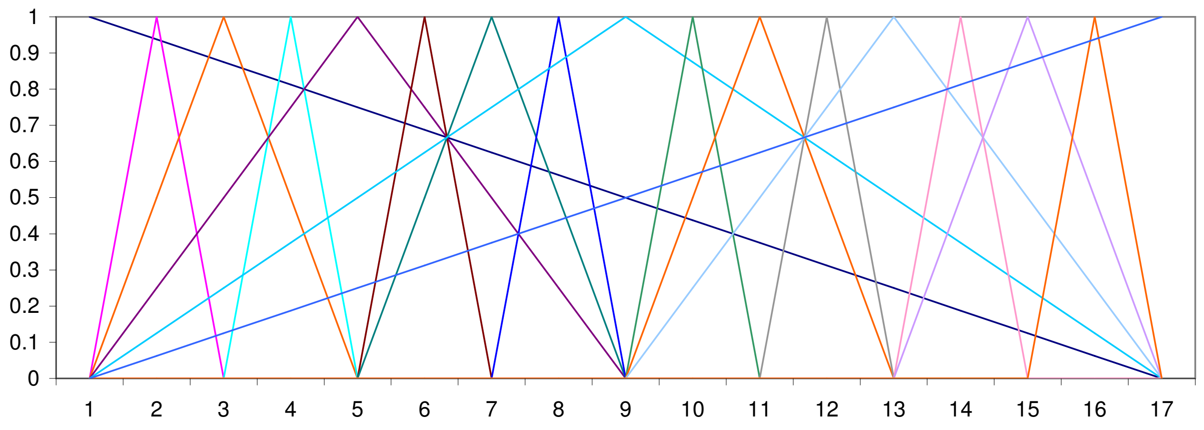


Figure 3: *Regular (non-adaptive) hierarchical basis functions. Note how some of the functions (at the ends and in the middle) are very broad, corresponding to the coarse-level bases, while others become progressively narrower.*

the basis modulation function at each iteration to originally ignore the data term and to later give it more weight.

These enhancements to the original HBF idea result in algorithms that can solve regularized low-level vision problems more effectively than previously developed techniques. However, they still leave the question open as to whether it is possible to derive an *optimal* set of hierarchical basis functions, or at least a more principled way to adapt them to the local structure of the underlying problem. We next turn out attention to answering these questions.

4 One-dimensional problems

We begin by analyzing the first order one-dimensional problem (4), since it is the easiest to understand and since we can actually derive an optimal hierarchical preconditioner. In practice, one-dimensional problems can usually be solved more efficiently using direct methods (Duff *et al.* 1986). However, if we wish to exploit the data-parallelism (and higher computational throughput) available on GPUs, the techniques presented here could be used.

The basic idea in using a hierarchical basis is to interpolate the solution obtained with a coarser level approximation and to then add a local *correction* term. However, if we were given the exact solution at the coarser level, could we then *perfectly* predict the solution at the finer level?

For the first order one-dimensional problem, it turns out that we can because each variable only interacts with its two nearest neighbors. Thus, if we place all the even variables in the coarse level and keep the odd variables at the fine level, we can solve for each fine level variable independent of the others. Consider the ith row from the tridiagonal system of linear equations (13),

$$a_{i,i-1}x_{i-1} + a_{i,i}x_i + a_{i,i+1}x_{i+1} = b_i. \tag{16}$$

We can write the solution for x_i as

$$x_i = a_{i,i}^{-1}[b_i - a_{i,i-1}x_{i-1} - a_{i,i+1}x_{i+1}]. \tag{17}$$

Note that this “interpolant” does *not*, in general, interpolate the two parents x_{i-1} and x_{i+1} since $a_{i,i-1}/a_{i,i} + a_{i,i+1}/a_{i,i} < 1$ whenever $w_i > 0$, i.e., whenever there is any data present at a node. In fact, as the data weight at a point $w_i \to \infty$, we get $x_i \to b_i/a_{i,i} = d_i$ and the target data value is interpolated exactly without using *any* smoothing. This is an example of what Sweldens calls *weighted wavelets* (Sweldens 1997).

We can now substitute the above formula for x_i for all odd values of i simultaneously and obtain a new energy function that does not involve the odd variables and yet which has the exact same dependence on the even variables as the original system (assuming that the odd variables are always placed in their minimal energy configuration). This process is commonly known as *cyclic reduction* (Golub and Van Loan 1996).

Another way to view this operation is to say that (17) defines a locally adapted basis (interpolation) function (Kobbelt and Schröder 1998). Figure 4 shows the locally adapted hierarchical basis functions for the data weighting and smoothness function introduced in Figure 1. Notice how none of the functions span the discontinuity in the smoothness function and how the local shape of the functions is controlled by the data spacing and strength.

If we permute the entries in $\boldsymbol{A}$ and $\boldsymbol{S}$ so that the fine level variables come first and the coarse level variables come second, we can re-write these matrices as

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{D} & \boldsymbol{E} \\ \boldsymbol{E}^T & \boldsymbol{F} \end{bmatrix} \quad \text{and} \quad \boldsymbol{S}_1 = \begin{bmatrix} \boldsymbol{I} & -\boldsymbol{D}^{-1}\boldsymbol{E} \\ 0^T & \boldsymbol{I} \end{bmatrix}. \tag{18}$$

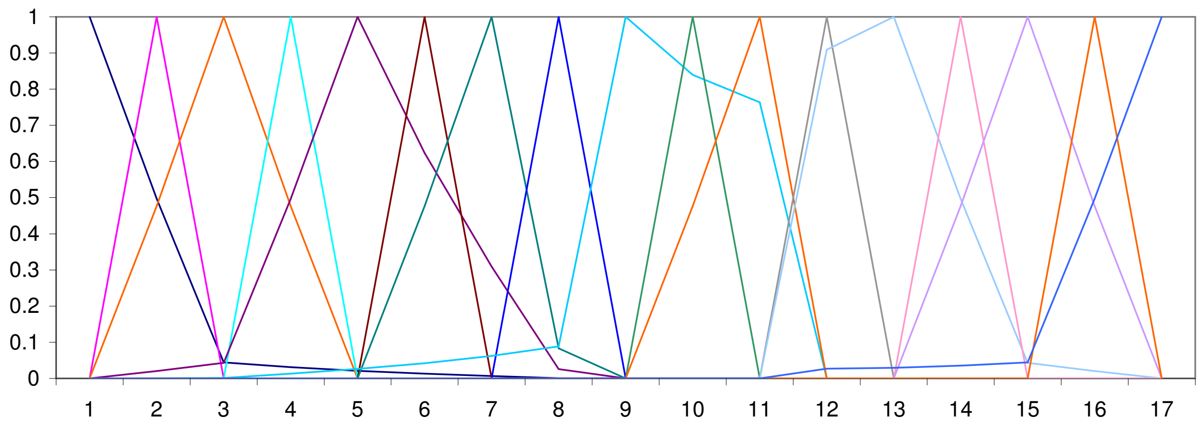


Figure 4: *Locally adapted hierarchical basis functions. Note how none of the functions span the break at* $i = \{11, 12\}$ *and how the shape of individual functions varies according to the local data weights and smoothness values. This particular set of bases is an exact preconditioner for the 1-D problem (when combined with the appropriate scaling).*

The resulting *preconditioned* coefficient matrix can therefore be written as

$$\hat{\boldsymbol{A}}_1 = \boldsymbol{S}_1^T \boldsymbol{A} \boldsymbol{S}_1 = \begin{bmatrix} \boldsymbol{D} & 0 \\ 0^T & \boldsymbol{F} - \boldsymbol{E}^T \boldsymbol{D}^{-1} \boldsymbol{E} \end{bmatrix} = \begin{bmatrix} \boldsymbol{D} & 0 \\ 0^T & \hat{\boldsymbol{A}}_1^c \end{bmatrix}. \tag{19}$$

For the one-dimensional first order problem, $\boldsymbol{D}$ is a diagonal matrix, $\boldsymbol{E}$ has a bandwidth of two, and $\boldsymbol{F}$ is also diagonal. The resulting (smaller) coarse-level coefficient matrix $\hat{\boldsymbol{A}}_1^c = \boldsymbol{F} - \boldsymbol{E}^T \boldsymbol{D}^{-1} \boldsymbol{E}$ is therefore tri-diagonal, just like the original coefficient matrix $\boldsymbol{A}$. We can therefore recurse on this matrix to construct a locally adapted set of hierarchical basis functions that that convert the original coefficient into a diagonal matrix $\hat{\boldsymbol{A}}$. In the iterative methods community, the above algorithm is called a *complete factorization* of $\hat{\boldsymbol{A}}$, since no approximations are used in this derivation (Ciarlet Jr. 1994). It is a special instance of the more general *incomplete LU factorization with multi-elimination* (ILUM) algorithm (Saad 2003, §12.5), which we describe in the next section.

The construction described above leads to the following re-interpretation of the well known cyclic reduction algorithm (Kobbelt and Schröder 1998):

For a one-dimensional first-order problem, using the preconditioner $\boldsymbol{S}\hat{\boldsymbol{A}}^{-1}\boldsymbol{S}^T$, *where* $\boldsymbol{S}$ *is*

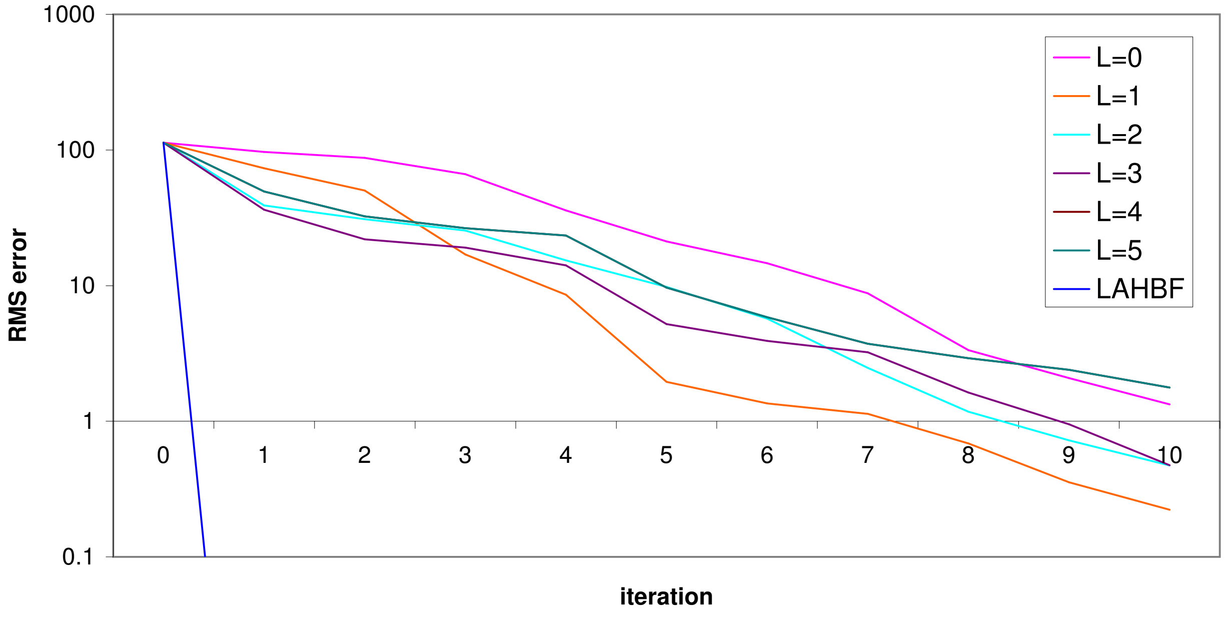


Figure 5: *Convergence of preconditioned conjugate gradient on the sample 1st order 1D problem, where L indicates the number of preconditioning levels. For non-adapted hierarchical bases, more levels initially speeds up the convergence but eventually slows it down. For the locally adapted bases (LAHBF), the algorithm converges in a single iteration.*

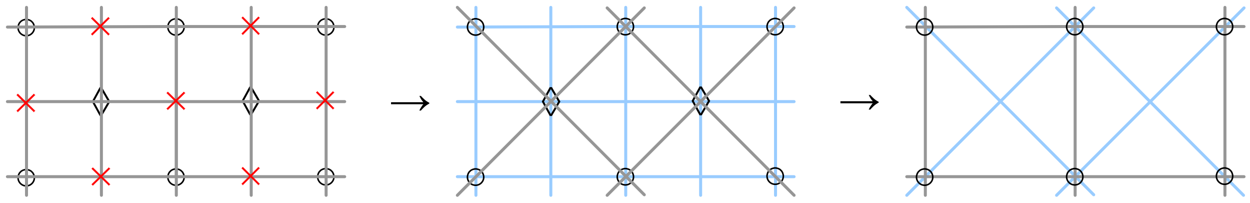


Figure 6: Red-black elimination of variables, and the resulting undesired (extra bandwidth) connections shown in blue.

defined using the set of locally adapted hierarchical basis functions $\boldsymbol{S}_l = \begin{bmatrix} \boldsymbol{I} & -\boldsymbol{D}_l^{-1}\boldsymbol{E}_l \end{bmatrix}$ *as described above and* $\hat{\boldsymbol{A}}$ *is the resulting diagonal (preconditioned) multi-level coefficient matrix, results in perfect preconditioning and hence a one-step solution.*

Looking at Figure 5, we see that indeed, the locally adapted hierarchical basis functions algorithm (LAHBF) converges in exactly one step.

Unfortunately, this result only holds for one-dimensional first order problems. However, the general methodology developed above can be modified so that it can be applied to two-dimensional problems, as we describe next.

5 Two-dimensional problems

Let us now turn our attention to two-dimensional problems, which are generally more relevant to computer vision and computer graphics.

In two dimensions, most hierarchical basis function approaches (as well as pyramid-based algorithms) eliminate odd rows and columns simultaneously, so that each coarser level has one quarter as many variables as the finer one (Szeliski 1990c). However, the fine-level variables being eliminated are not independent, which makes it difficult to apply the previous methodology.

Instead, we eliminate the red elements in a red-black checkerboard first, and then eliminate the other half of the variables to move a full level up in the pyramid. In the numerical analysis community, this is known at *repeated red-black (RRB) ordering* (or elimination) (Brand 1992, Ciarlet Jr. 1994, Notay and Amar 1997), and is a particular instance of more general *multicoloring* strategies that can be applied in conjunction with *incomplete LU with multi-elimination* (ILUM) preconditioning (Saad 2003).[5] Pictorially, we can view this as shown in Figure 6, where the red $\times$'s are eliminated in the first pass and the diamonds are eliminated in the second. The gray lines indicate the connections (non-zero entries in $\boldsymbol{A}$) between variables, while the blue lines indicate undesirable connections, which must be eliminated, as described below.

Using this scheme, the $\boldsymbol{D}$ matrix is diagonal, the $\boldsymbol{E}$ matrix is 4-banded (4 nearest neighbors), and the $\boldsymbol{F}$ matrix is diagonal (crosses and diamonds are independent of each other). The resulting $\hat{\boldsymbol{A}}_1^c$ matrix is 9-banded, i.e., each circle (or diamond) depends on not only its immediate 4-neighbors (gray lines in Figures 6 and 7), but also on its diagonal 2-D neighbors (light blue lines).

In order to keep the bandwidth from growing, we would like to eliminate these diagonal connections. But how? The usual answer (Brand 1992, Saad 2003) is to perform *incomplete factorization*, which means simply dropping the undesired element (ILU0), or perhaps adding these dropped values to the diagonal to preserve affine/interpolatory properties of the resulting system (known as *modified ILU* or MILU). (A blend of these two approaches, known *relaxed ILU* or RILU, is to add a fraction ω of the dropped elements to the diagonal (Saad 2003).)

This paper takes an alternative approach. Rather than thinking of the undesired matrix elements as abstract quantities, we view them as springs whose impact on the system must be approximated by stiffening adjacent (non-eliminated) springs. The goal is to make the coarsened *energy function* (quadratic form with enforced zero entries) be as good as possible an approximation to the original energy.

Let us start with some special cases. If the data constraint is so strong that the connections in $\boldsymbol{D}^{-1}\boldsymbol{E}$ are essentially zero, we do not have a problem. The same is true when a circle only depends on an adjacent pair of neighbors (Figure 7a), since this induces a single new link (dependence).

When a circle has a non-adjacent pair of neighbors (not shown in Figure 7), we ignore this connection, since modeling it would otherwise induce a connection between neighbors that does

[5]Half-octave pyramids are also used in image processing (Crowley and Stern 1984), where they are sometimes called *quincunx* sampling (Feilner *et al.* 2005).

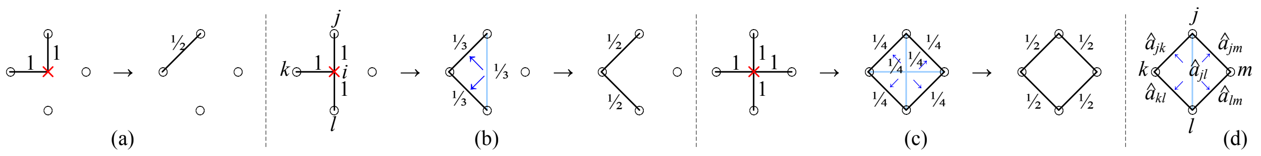


Figure 7: *Two-dimensional cliques of neighbors and how they are treated during the simplification process. (a–c): The left hand side of each arrow diagram shows the original (non-zero) connections between the crosses and the circle along with their original weights. The right hand side shows the connections after variable elimination. The non-diagonal connections shown in blue are eliminated (redistributed to the black ones) in the final step. (d): General case of diagonal edge weight redistribution.*

not exist in the fine-level model.

When three out of the four connections are non-zero (Figure 7b), we can view this configuration as a triangular finite element. If the original weights between neighbors are 1, it is easy to demonstrate (by substituting the average value $(x_j + x_k + x_l)/3$ for x_i)[6] that the resulting energy has connections of strengths $1/3$ between the neighbors shown in Figure 7b (right size of the first arrow). For a triangular element of area h^2 with unit smoothness $s = 1$, the diagonal connections should be $1/2$, so that they add up to 1 after the other side of each link is also eliminated. (Recall that the finite element analysis in Section 2.2 indicates that the membrane energy in 2D does not scale with h because the h^{-2} growth in derivative values is cancelled by the h^2 growth in area.) Therefore, a sensible heuristic is to "distribute" half of the strength along the vertical blue line to each of its two neighbors, which results in the desired pair of values $(1/2, 1/2)$.

For the fully four-connected case (Figure 7c), a similar analysis shows that the vertical and horizontal strength should be evenly distributed between the adjacent neighbors *after being scaled up by a factor of 2*, which results in a final strength $1/2$ on each of the diagonal (black) edges.

What about the case when the strengths are not all the same? A general heuristic rule can be formulated that reduces to the above special cases:

Heuristic 1: *For the two-dimensional membrane, after multi-elimination of the red (cross)*

[6]Note how we now use single subscripts to index the variables

variables, distribute any "diagonal" connection strengths in the rotated grid to each connection's nearest neighbors, using the relative weighting of the original 4-connections, with appropriate scaling.

More formally, if a variable x_i originally has connections to its 4 neighbors of weight $a_{ij} \dots a_{im}$ (Figure 7d), and the interpolation (adapted hierarchical basis) weights are $s_{ij} \dots s_{im},\ s_{ij} = -a_{ij}/a_{ii}$, the entries in the new coarser-level coefficient matrix $\hat{\boldsymbol{A}}_1$ (19) corresponding to that finite element will be

$$\hat{a}_{jk} = a_{ii} s_{ij} s_{ik}, \quad \text{etc.} \tag{20}$$

(The entry $\hat{a}_{jk}$ will also receive a contribution from the finite element bordering it on the other side, but such contributions are ignored in the formulas below, at least in our current implementation.)

This formula works fine, except that it introduces undesired entries between nodes j and l (and also k and m, not shown in Figure 7d), which must be re-distributed to the neighboring pairs $jk \dots jm$. To do this, we first compute the relative weights for the four allowable links, e.g.,

$$w_{jk} = \hat{a}_{jk}/W_{jklm}, \quad \text{with} \quad W_{jklm} = \sum_{\alpha\beta \in \{jk,kl,jm,ml\}} \hat{a}_{\alpha\beta}. \tag{21}$$

We then update each of the allowable link weights by

$$\hat{a}_{jk} \leftarrow \hat{a}_{jk} + s_{\mathrm{N}} w_{jk} \hat{a}_{jl} = (1 + s_{\mathrm{N}} \hat{a}_{jl}/W_{jklm}) \hat{a}_{jk}. \tag{22}$$

(The diagonal entries are also adjusted to maintain the balance between off-diagonal and diagonal entries in $\hat{\boldsymbol{A}}_1$.) The scaling factor $s_{\mathrm{N}} = 2$ is chosen so that the coarse level coefficient matrix in homogenous (constant smoothness) regions is the same as would be obtained by applying finite element analysis at the coarser level.

The important thing to notice about this scheme is that it simultaneously satisfies two goals:

1. It incorporates the local relative weighting between the data constraints and the smoothness constraints, as well as local variability and discontinuities in the smoothness, since the interpolation/basis functions $\boldsymbol{S}_l$ are derived from the problem itself and do a perfect job of coarsening the original problem.

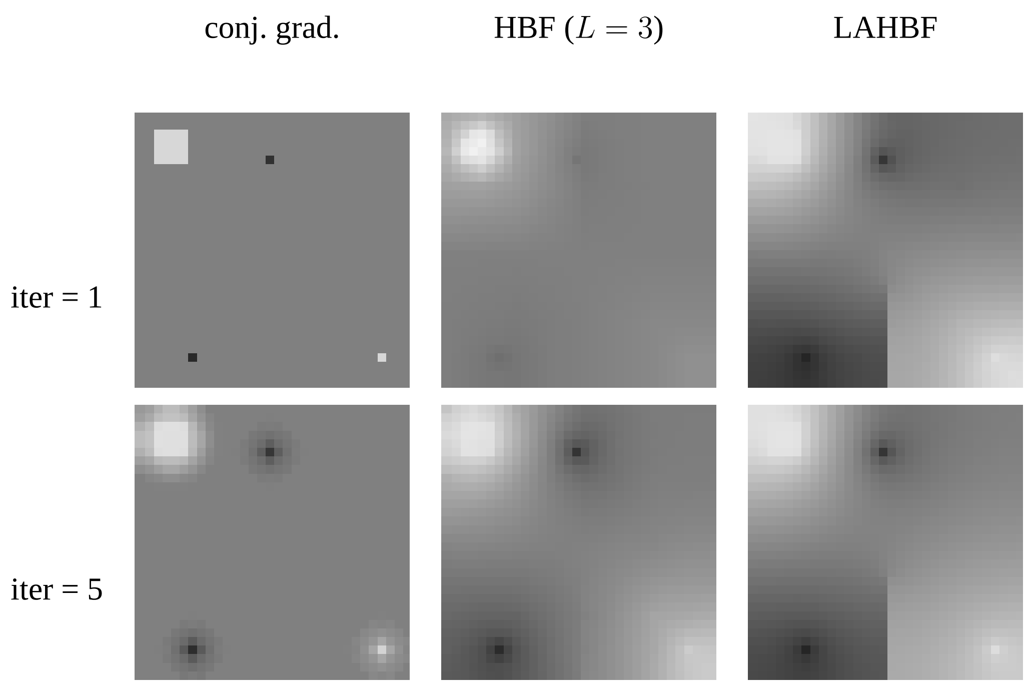


Figure 8: *Simple two-dimensional discontinuous interpolation problem. The two rows show the solution after the first and fifth iterations, and the three columns show regular conjugate gradient, hierarchical basis preconditioning (3 levels), and LAHBF preconditioning. Note how conjugate gradient hardly makes any progress, while regular HBFs over-smooth the solution.*

2. It re-distributes unwanted entries in the new coefficient matrix in a way that preserves the desired finite element discretization as well as possible in smooth regions.

These two properties have not, to our knowledge, been previously used to construct parallel preconditioners for these kinds of problems.

5.1 GPU implementation

Implementing the locally adaptive hierarchical basis preconditioned conjugate gradient algorithm is straightforward, as it involves only a small extension to the conjugate gradient and multigrid algorithms already developed by Bolz *et al.* (2003). The basic operations in conjugate gradient, namely the evaluation of the residual vector $\boldsymbol{r}$, which involves the sparse matrix multiplication $\boldsymbol{Ax} - \boldsymbol{b}$, and the computation of the new conjugated direction $\boldsymbol{d}$ and step size, which involve the computation of two inner products, map naturally onto the GPU. (Note that the $\boldsymbol{A}$ matrix, which has only 4 non-zero entries per row, is stored naturally alongside all of the other vector (image) quantities as a 4-banded float image.) Similarly, the transformation of the current direction vector by the $\boldsymbol{S}^T$ and $\boldsymbol{S}$ matrices (Szeliski 1990c) involves nothing more than the inter-level transfer operations present in multigrid, where the spatially varying entries in each $\boldsymbol{S}_l$ matrix can be again stored in a 4-banded image aligned with the quantities they are being applied to.

The final algorithm therefore consists of the following steps (see (Szeliski 1990c) for more details):

1. Compute the current residual vector $\boldsymbol{r}$
2. Compute the hierarchical version of the residual by applying the coarsening operations $\boldsymbol{S}^T$
3. Divide this residual by the hierarchical coefficient matrix diagonal
4. Compute the nodal (smoothed) version of the residual by applying the interpolation operations $\boldsymbol{S}$
5. Compute the conjugated direction as a linear combination of the new (preconditioned) and previous directions
6. Compute the optimal step size using an inner product and update the state

Each of these steps involves a separate rendering pass, with one pass per resolution level in the coarsening and smoothing stages. Because of the currently large overhead in initiating a new pass, it is probably more effective to use a limited number of levels of preconditioning, which is fine, as for most computer graphics applications, constraints only need to be propagated a finite distance. Alternatively, a coarsened version of the problem could be uploaded to the CPU for solution with a direct solver as part of the preconditioning step.

6 Experimental results

To evaluate the performance of our new algorithm, we have compared it to regular conjugate gradient descent, regular (non-adaptive) hierarchical basis preconditioning, and three different variants of incomplete factorization (ILU0, MILU, and RILU with $\omega = 0.5$). Figure 8 shows a simple 2D interpolation problem that consists of interpolating a set of 4 sparse constraints on a square grid with a tear halfway across the bottom edge. As you can see, conjugate gradient hardly makes any progress, even after 20 iterations. Regular hierarchical basis preconditioning starts off strong, but after 5 iterations, it has not yet accurately modeled the discontinuity. Locally adaptive hierarchical basis preconditioning, on the other hand, achieves a visually acceptable solution in a single iteration.

Figure 9 shows a plot of the RMS error (relative to the minimum energy solution, expressed in gray levels) for all of these algorithms. As you can see, LAHBF significantly outperforms all previously published algorithms, which can also be verified by looking at the semi-log plots of the error curves.

6.1 Computer Graphics applications

We next evaluate the performance of our algorithm on a number of computer graphics applications.

Colorization Colorization is the process of propagating a small number of color strokes to a complete gray-scale image, while attempting to preserve discontinuities in the image (Levin *et al.* 2004). As such, is it a perfect match to the controlled-continuity interpolators developed in this paper. The actual smoothness term being minimized in (Levin *et al.* 2004) involves differences between a pixel's value and the average of its eight neighbors. As such, it is not a first-order smoothness term and is not directly amenable to our acceleration technique.[7] Instead, we replace the term used in (Levin *et al.* 2004) with a simpler term, i.e., we set the horizontal and vertical smoothness strengths inversely proportional to the horizontal and vertical grayscale gradients.

[7]Levin *et al.* (2006) also use a related formula for solving the natural image matting problem in closed form.

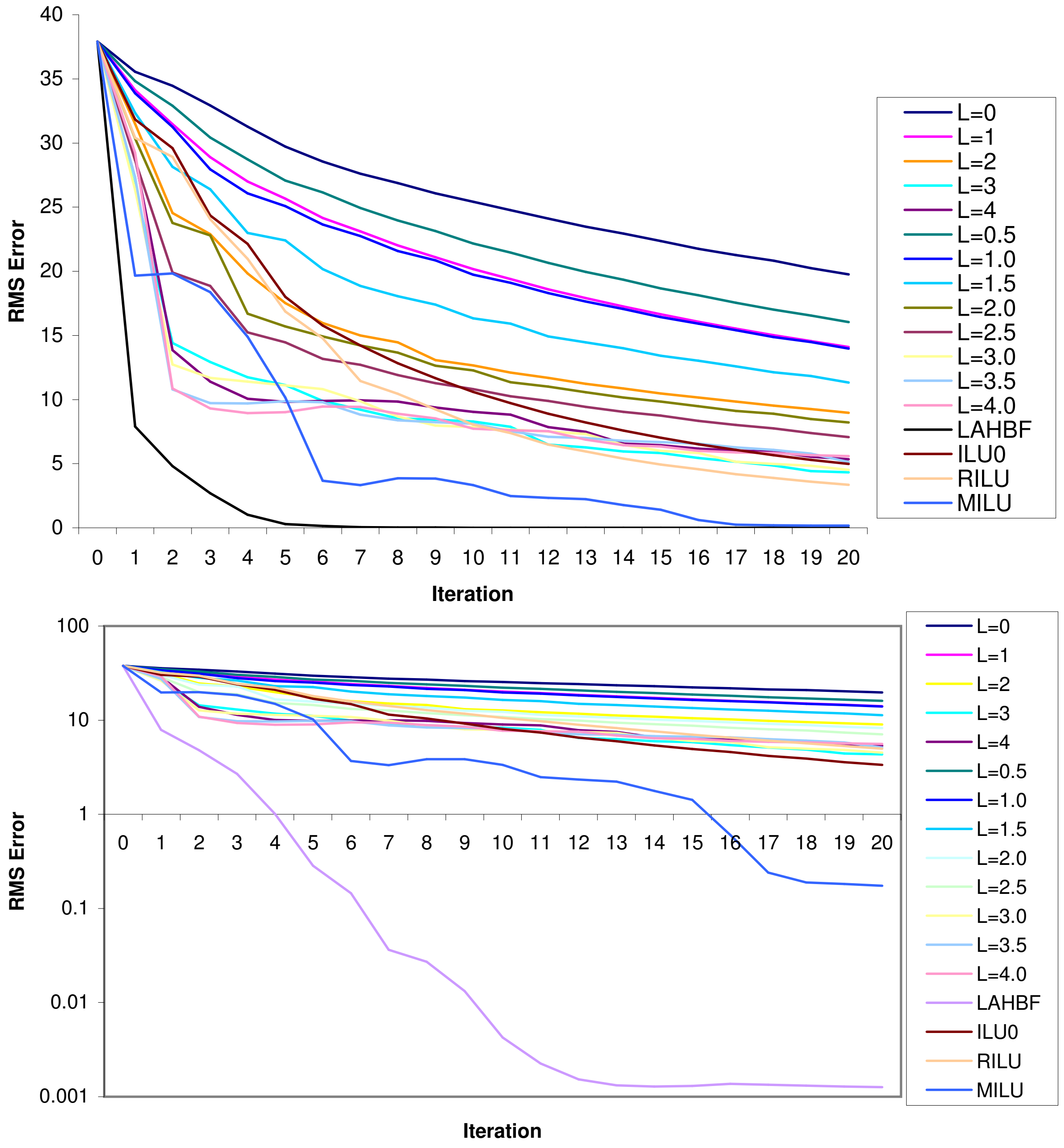


Figure 9: *Error plot for the simple two-dimensional problem. The LAHBF preconditioned solver converges in just a few iterations, while regular HBFs converge more slowly and require manual selection of the number of levels for best performance. The MILU algorithm performs better than non-adapted bases, but still significantly underperforms the new algortihm.*

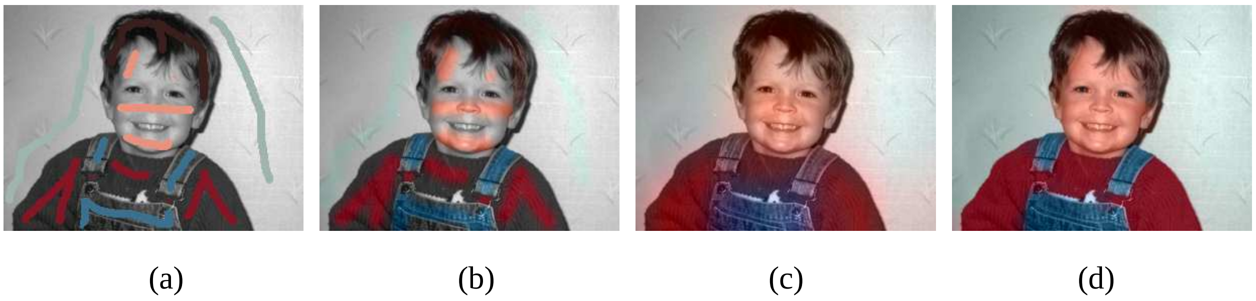


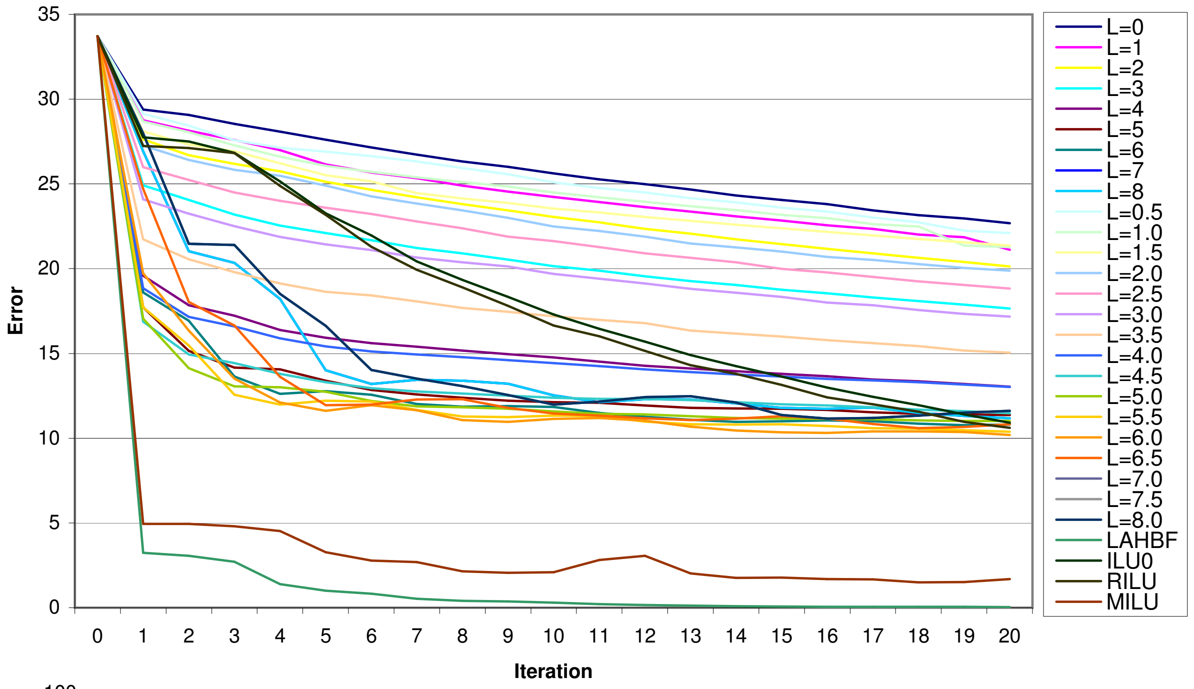
Figure 10: *Piecewise smooth color interpolation in a colorization application: (a) input gray image with color strokes overlaid; (b) solution after 20 iterations of conjugate gradient; (c) using 1 iteration of hierarchical basis function preconditioning; (d) using 1 iteration of locally adapted hierarchical basis functions.*

This still does a good job of preventing colors from bleeding across region boundaries, while being amenable to our current acceleration technique.

Figure 10 visually shows the result of running our algorithm, as well as conventional preconditioned conjugate gradient on the sparse set of color strokes shown in Figure 10a. As you can see, conjugate gradient hardly makes any progress, even after 20 iterations, while LAHBF preconditioned conjugate gradient converges in just a few iterations. Figure 11 shows the convergence of these algorithms (RMS error in the reconstructed chrominance signals) as a function of the number of iterations.

Poisson (gradient domain) blending Poisson blending is a technique originally developed to remove visual artifacts due to strong intensity or color differences when a cutout is pasted into a new background (Pérez *et al.* 2003). It has since been extended to non-linear variants (Levin *et al.* 2004), as well as applied to image stitching, where it is used to reduce visual discontinuities across image seams (Agarwala *et al.* 2004). Poisson blending relies on reconstructing an image that matches, in a least-squares sense, a gradient field that has been computed from the various images that contribute to the final mosaic.

Figure 12 shows an original two-image stitch where blending has been applied. The left image shows the unblended result. The middle image shows the label field, indicating where each pixel



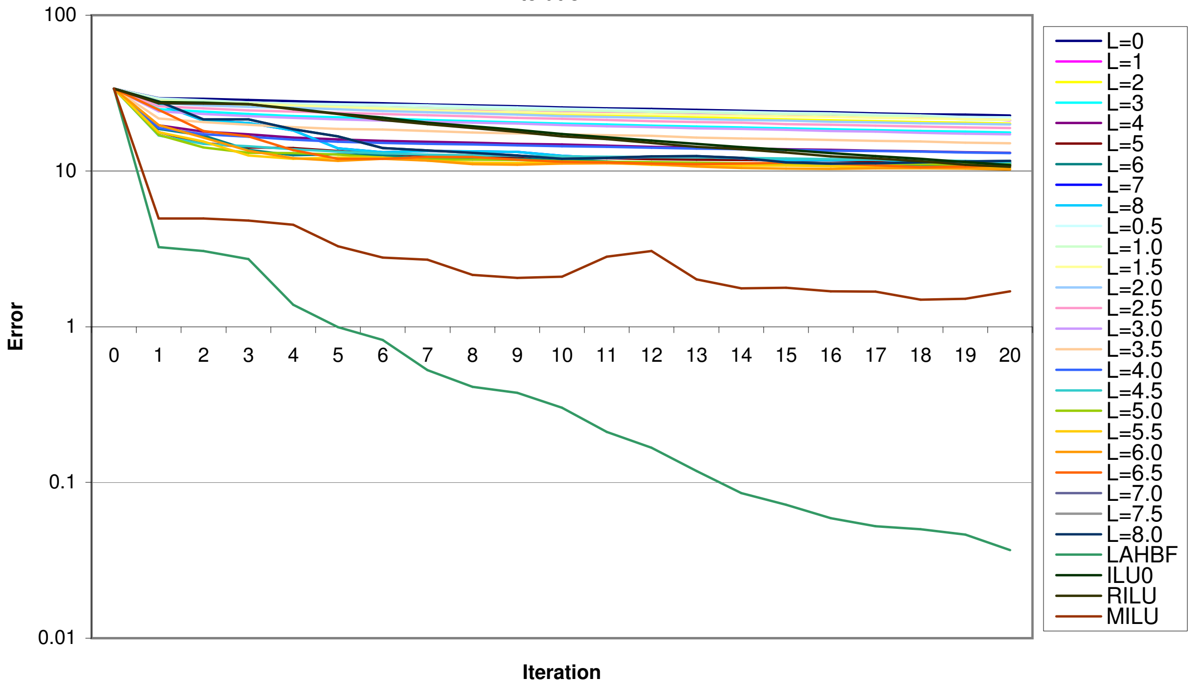


Figure 11: *Error plot for the simple colorization problem. The LAHBF preconditioned solver converges in just a few iterations, while regular HBFs and MILU converge more slowly.*

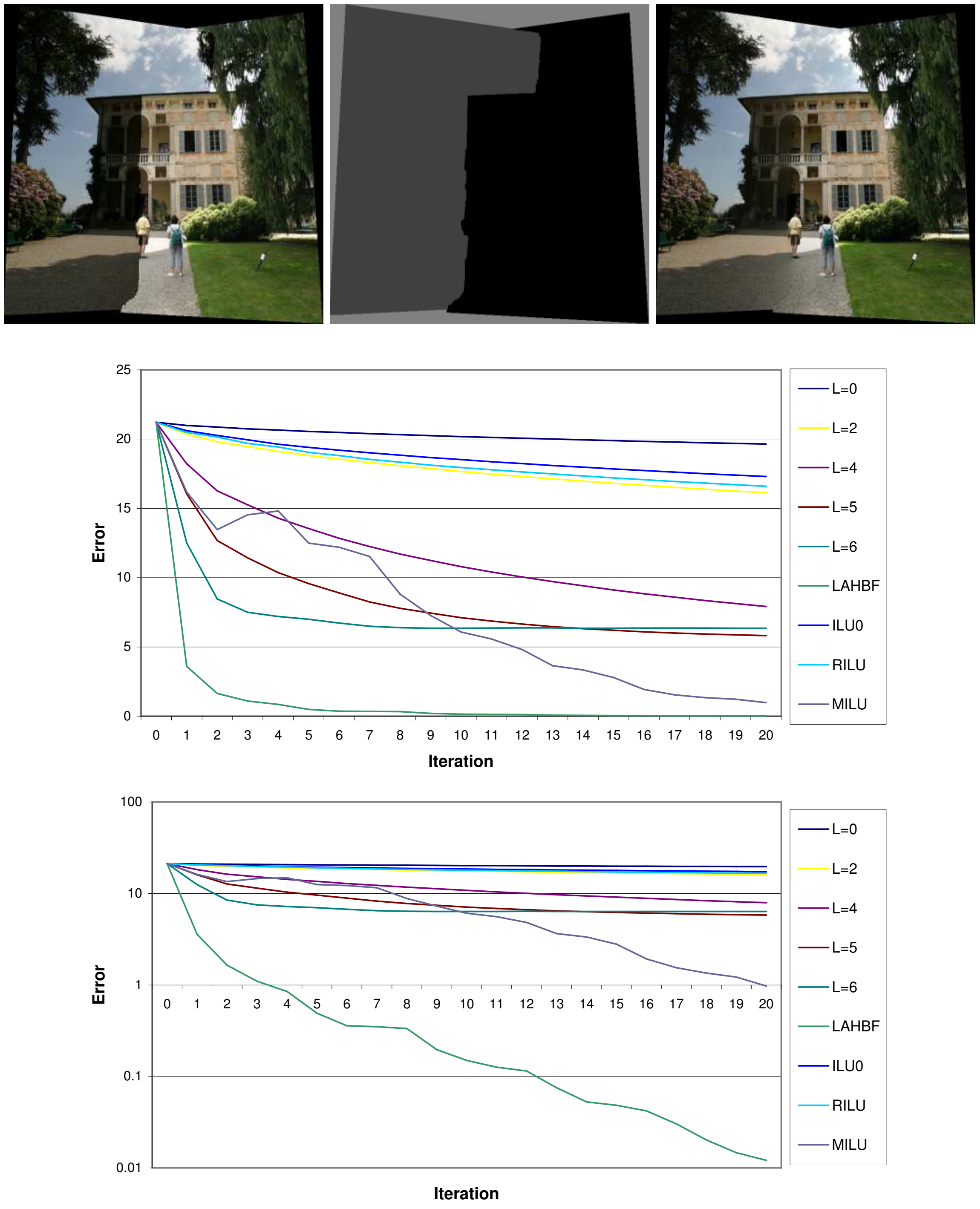
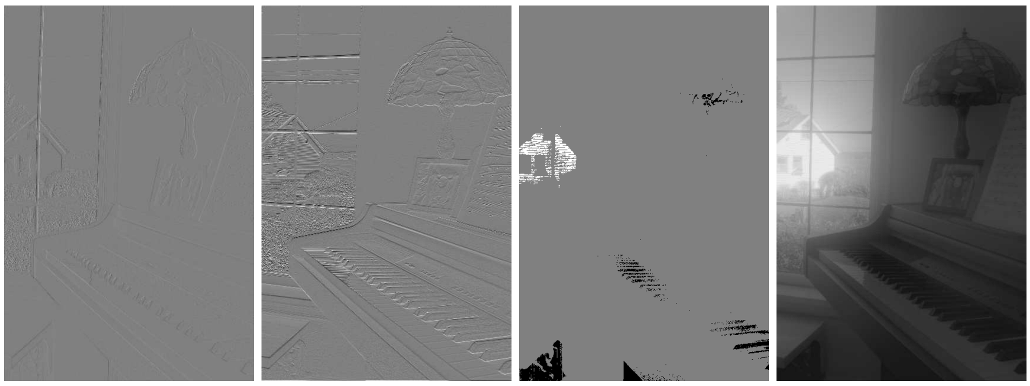


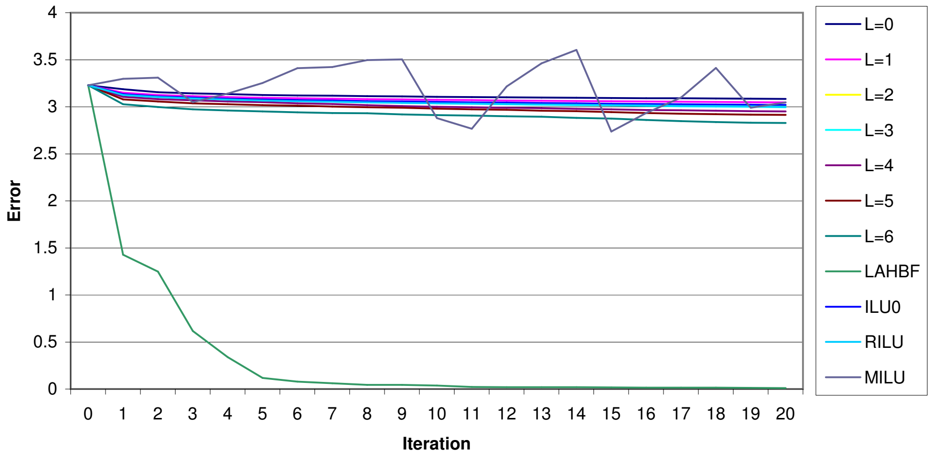
Figure 12: *Poisson blending example. The left image shows the unblended result, the middle image shows the pixel labels, and the right image shows the final blended image. The regular and semi-log error plots are below.*

is drawn from. The right image shows the blended result, obtained using just a few iteration of LAHBF preconditioned conjugate gradient. (Notice that a slight seam is still visible, because the *contrast* is not properly matched across images. This suggest that blending in the log-luminance domain might be more appropriate.) Figure 12 also shows the corresponding error plots. As before, we see that LAHBF converges in just a handful of steps. In these examples, rather than selecting a single pixel as a hard constraint to remove the overall shift ambiguity, we used a weak constraint towards the unblended original image.

Tone mapping Our final graphics application is gradient domain high dynamic range compression (Fattal *et al.* 2002). In this technique, the gradients of a log-luminance function are first computed and then compressed through a non-linearity (Figure 13a–b), A compressed log-luminance function is then reconstructed from these gradients. Figure 13c shows the sparse set of constraints used to clamp the dark and light values, Figure 13d shows the reconstructed log-luminance function, and Figure 13e shows the convergence plots. As before, we see that our new approach offers significant performance improvements. We have also applied our algorithm to a newly developed interactive tone mapping algorithm (Lischinski *et al.* 2006) with good results.

Additional applications The solution of data and gradient interpolation/reconstruction tasks occurs in many other computer vision, computer graphics, and computational photography applications. For example, Agrawal *et al.* (2005) remove reflections from photographs by selectively nulling out certain gradients and then reconstructing the image from the resulting gradient field. In more recent work (Agrawal *et al.* 2006), they solve shape from shading using locally weighted gradient reconstruction, which is specifically the kinds of tasks our technique is designed to handle. It would be interesting to compare the speed of the approach described in this paper to the sine-transform approach used in their work.





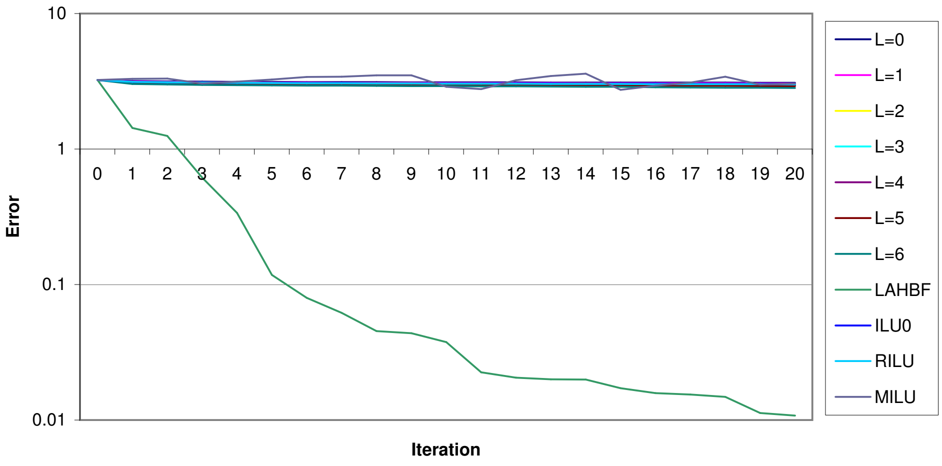


Figure 13: *Gradient domain high dynamic range compression: Leftmost two images: horizontal and vertical compressed gradient fields; middle image: light and dark value constraints; right image: reconstructed compressed log-luminance function. The regular and semi-log error plots are below.*

Applic.	Prob. size	Setup	LAHBF	CG	Prec.	Mem (MB)
Color.	$298 \times 224 \times 2$	15	153	30	77	14 / 26
Blend.	$686 \times 686 \times 3$	262	797	280	563	106 / 220
Compr.	$846 \times 1271 \times 1$	359	1984	150	1200	174 / 373

Table 1: *Performance of LAHBF on three sample problems. The problem size is given in pixels (width $\times$ height) by the number functions (r.h.s. columns) being reconstructed. The setup (coefficient matrix computation), locally adaptive HBF pre-computation, conjugate gradient iteration, and hierarchical preconditioning steps are given in milliseconds (msec). The memory footprint is in MB, and shows the memory without and with hierarchical preconditioning. As noted in the text, we expect these numbers given to drop significantly with code optimization.*

6.2 Performance

The performance of our algorithm on the three real-world examples discussed above is summarized in Table 1. These numbers were obtained on a Xeon 2.8GHz PC running on unoptimized C++ code. (We suspect that we can squeeze out a factor of $2 - -3\times$ in both speed and memory with careful optimization.) The time taken and memory size are roughly proportional to the problem size. Computing the locally adaptive basis functions takes a little longer than actually using them as a preconditioner, which is about twice as slow as the basic conjugate gradient update step. The convergence plots shown previously should therefore have their top lines (regular conjugate gradient $L = 0$) squished left by a factor of three to make the comparisons sensible in terms of run times. However, the advantages of LAHBF over conjugate gradient still remain quite dramatic.

We also compared the time taken with our solver against the sparse linear solver in MatLab. For the Colorization application (298×224 image), our preconditioned iterative technique uses 153 msec precomputation time and 107 msec per iteration, for a total of 688 msec for 5 iterations. The MatLab code (after loading the sparse linear system) takes 2534 msec. For the larger problems listed in Table 1, we were unable to load the sparse matrices into Matlab, so we cannot get comparative numbers.

7 Discussion

Our experimental results demonstrate that our locally adapted hierarchical basis functions significantly outperform unadapted hierarchical bases as preconditioners for the kinds of first-order optimization problems presented in the previous section.

As we mentioned before, our technique is closely related to both algebraic multigrid and ILUM. Incomplete LU (lower-upper) factorization with multi-elimination (ILUM) on a red-black (multicolor) grid performs the same first step of problem reduction/coarsening as our algorithm. The major difference comes in how the undesired off-diagonal entries are treated. In ILUM, it is traditional to simply drop coefficient matrix entries that are of small magnitude relative to other entries in a given row. (This is known as ILU with thresholding, or ILUT (Saad 2003, p. 306).) In our approach, rather than dropping such terms we *re-distribute* these entries to other off-diagonal and on-diagonal entries in a way that *preserves* a good finite-element approximation to the original variational problem.

Algebraic multigrid techniques attempt to remedy some of the shortcomings of traditional multigrid techniques, which assume a fixed uniform set of coarsening and interpolation kernels, by adapting these kernel to the underlying structure (continuity and stiffness) of the finite element problem being solved. As discussed in (Saad 2003, p. 444), more recent AMG techniques use the same basic multi-elimination equations to define the interlevel transfer operators as ILUM. As with ILUM, small terms in the resulting coarser level coefficient matrix are again neglected in order to keep the system bandwidth from growing excessively. Again, the approach presented in this paper suggests a more principled and effective way for re-distributing these neglected terms.

Another class of recently developed pre-conditioning techniques relies on *trees* defined over the original connectivity graph (Gremban 1996, Boman *et al.* 2004, Spielman and Teng 2004). These techniques, also known as *combinatorial preconditioners* because of their close connection to graph theoretic algorithms, are known to perform well on sparse irregular linear systems. (http://www.preconditioners.com contains a good tutorial introduction as well as pointers to the most recent literature.) While we have not yet had a chance to compare the performance of our approach to these new techniques, we believe that for the 2D grid-based first order problems that we address in the paper, our technique will perform better because it does not neglect a subset of connections in the original problem, as the tree-based techniques do.

7.1 Future work

The basic approach developed in this paper can be extended in a number of ways. We are currently working on extending our technique to second order problems in both one and two dimensions. Because even-odd and red-black ordering do not lead to isolated sets of variables that can be eliminated, these problems are significantly more challenging to handle.

We are also interested in extending our approach to non-quadratic energy minimization problems such as dynamic programming (which can be used for stereo matching or speech recognition). While 1-D versions of these problems are known to have effective sequential solutions, our approach can be used to parallelize these algorithms onto architectures such as GPUs. The extension to 2-D MRFs is likely to be quite challenging, but could have large payoffs for problems such as stereo matching and graph cut segmentation.

Finally, we would like to apply our techniques to a wider range of computer graphics applications such as fluid simulations.

8 Conclusions

In this paper, we have shown how to locally adapt hierarchical basis functions to inhomogeneous problems so that they are more effective at preconditioning large first-order optimization problems that arise in computer graphics and computer vision. Our approach is based on an extension of repeated red-black (RRB) ILUM preconditioners that uses finite element analysis to redistribute off-diagonal terms in a manner that preserves good approximations to the underlying variational problem. The resulting algorithm performs significantly better than previously proposed approaches, is simple to implement, and also maps naturally onto a GPU. We have demonstrated that our current algorithm has a wide range of applications in computer graphics, and we plan to extend its range of applicability in the future.

References

Agarwala, A. *et al.*. (2004). Interactive digital photomontage. *ACM Transactions on Graphics*, *23(3)*, 292–300.

Agrawal, A. *et al.*. (2005). Removing photography artifacts using gradient projection and flash-exposure sampling. *ACM Transactions on Graphics*, *24(3)*, 828–835.

Agrawal, A., Raskar, R., and Chellappa, R. (2006). What is the range of surface reconstructions from a gradient field? In Leonardis, A., Bischof, H., and Pinz, A., editors, *Computer Vision – ECCV 2006*, pages 578–591, Springer.

Bathe, K.-J. and Wilson, E. L. (1976). *Numerical Methods in Finite Element Analysis*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Bolz, J. *et al.*. (2003). Sparse matrix solvers on the GPU: Conjugate gradients and multigrid. *ACM Transactions on Graphics*, *22(3)*, 917–924.

Boman, E. G. *et al.*. (2004). Maximum-weight-basis preconditioners. *Numerical Linear Algebra with Applications*, *11*, 695–721.

Brand, C. W. (1992). An incomplete factorization preconditioning using repeated red-black ordering. *Numer. Math.*, *61*, 433–454.

Briggs, W. L., Henson, V. E., and McCormick, S. F. (2000). *A Multigrid Tutorial*. Society for Industrial and Applied Mathematics, Philadelphia, second edition.

Brox, T. *et al.*. (2004). High accuracy optical flow estimation based on a theory for warping. In *Eighth European Conference on Computer Vision (ECCV 2004)*, pages 25–36, Springer-Verlag, Prague.

Ciarlet Jr., P. (1994). Repeated red-black ordering: a new approach. *Numerical Algorithms*, *7*, 295–324.

Crowley, J. L. and Stern, R. M. (1984). Fast computation of the difference of low-pass transform. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *6(2)*, 212–222.

Duff, I. S., Erisman, A. M., and Reid, J. K. (1986). *Direct Methods for Sparse Matrices*. Clarendon Press, Oxford.

Fattal, R., Lischinski, D., and Werman, M. (2002). Gradient domain high dynamic range compression. *ACM Transactions on Graphics (TOG)*, *21(3)*, 249–256.

Feilner, M. *et al.*. (2005). An orthogonal family of quincunx wavelets with continuously adjustable order. *IEEE Transactions on Image Processing*, *14(4)*, 499–520.

Golub, G. and Van Loan, C. F. (1996). *Matrix Computation, third edition*. The John Hopkins University Press, Baltimore and London.

Gortler, S. J. and Cohen, M. F. (1995). Hierarchical and variational geometric modeling with wavelets. In *Symposium on Interactive 3D Graphics*, pages 35–43, Monterey, CA.

Gremban, K. D. (1996). *Combinatorial Preconditioners for Sparse, Symmetric, Diagonally Dominant Linear Systems*. Ph.D. thesis, Carnegie Mellon University. CMU-CS-96-123.

Horn, B. K. P. and Brooks, M. J. (1986). The variational approach to shape from shading. *Computer Vision, Graphics, and Image Processing*, *33*, 174–208.

Horn, B. K. P. and Brooks, M. J. (1989). *Shape from Shading*. MIT Press, Cambridge, Massachusetts.

Kobbelt, L. and Schröder, P. (1998). A multiresolution framework for variational subdivision. *ACM Transactions on Graphics*, *17(4)*, 209–237.

Lai, S.-H. and Vemuri, B. C. (1997). Physically based adaptive preconditioning for early vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *19(6)*, 594–607.

Levin, A., Lischinski, D., and Weiss, Y. (2004). Colorization using optimization. *ACM Transactions on Graphics*, *23(3)*, 689–694.

Levin, A., Lischinski, D., and Weiss, Y. (2006). A closed form solution to natural image matting. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2006)*, pages 61–68, New York City, NY.

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)*, pages 377–389, Springer-Verlag, Prague.

Lischinski, D., Farbman, Z., Uytendaelle, M., and Szeliski, R. (2006). Interactive local adjustment of tonal values. *ACM Transactions on Graphics*, *25(3)*.

Notay, Y. and Amar, Z. O. (1997). A nearly optimal preconditioning based on recursive red-black ordering. *Numerical Linear Alg. Appl.*, *4*, 369–391.

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

Pérez, P., Gangnet, M., and Blake, A. (2003). Poisson image editing. *ACM Transactions on Graphics*, *22(3)*, 313–318.

Poggio, T. (1985). Early vision: From computational structure to algorithms and parallel hardware. *Computer Vision, Graphics, and Image Processing*, *31*, 139–155.

Poggio, T. *et al.*. (1988). The MIT vision machine. In *Image Understanding Workshop*, pages 177–198, Morgan Kaufmann Publishers, Boston.

Poggio, T., Torre, V., and Koch, C. (1985). Computational vision and regularization theory. *Nature*, *317(6035)*, 314–319.

Saad, Y. (2003). *Iterative Methods for Sparse Linear Systems*. SIAM, second edition.

Schröder, P. and Sweldens, W. (1995). Spherical wavelets: Efficiently representing functions on the sphere. In *Proceedings of SIGGRAPH 95*, pages 161–172.

Spielman, D. A. and Teng, S.-H. (2004). Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *36th Annual ACM Symposium on Theory of Computing*, pages 81–90. Full version available at http://arxiv.org/abs/cs.DS/0310051.

Sweldens, W. (1997). The lifting scheme: A construction of second generation wavelets. *SIAM J. Math. Anal., 29(2)*, 511–546.

Szeliski, R. (1990a). Bayesian modeling of uncertainty in low-level vision. *International Journal of Computer Vision, 5(3)*, 271–301.

Szeliski, R. (1990b). Fast shape from shading. In *First European Conference on Computer Vision (ECCV'90)*, pages 359–368, Springer-Verlag, Antibes, France.

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

Szeliski, R. and Terzopoulos, D. (1989). Parallel multigrid algorithms and computer vision applications. In Mandel, J. *et al.*, editors, *Fourth Copper Mountain Conference on Multigrid Methods*, pages 383–398, Society for Industrial and Applied Mathematics, Copper Mountain, Colorado.

Terzopoulos, D. (1983). Multilevel computational processes for visual surface reconstruction. *Computer Vision, Graphics, and Image Processing*, *24*, 52–96.

Terzopoulos, D. (1986). Regularization of inverse visual problems involving discontinuities. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *PAMI-8(4)*, 413–424.

Terzopoulos, D. (1988). The computation of visible-surface representations. *IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-10(4)*, 417–438.

Terzopoulos, D. and Witkin, A. (1988). Physically-based models with rigid and deformable components. *IEEE Computer Graphics and Applications, 8(6)*, 41–51.

Yang, R. and Pollefeys, M. (2003). Multi-resolution real-time stereo on commodity graphics hardware. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2003)*, pages 211–218, Madison, WI.

Yaou, M.-H. and Chang, W.-T. (1994). Fast surface interpolation using multiresolution wavelets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *16(7)*, 673–689.

Yserentant, H. (1986). On the multi-level splitting of finite element spaces. *Numerische Mathematik*, *49*, 379–412.

Zhang, L. *et al.*. (2002). Single view modeling of free-form scenes. *Journal of Visualization and Computer Animation*, *13(4)*, 225–235.