Fast Surface Interpolation Using Hierarchical Basis Functions

RICHARD SZELISKI, MEMBER, IEEE

Abstract—**The rapid solution of surface interpolation and other regularization problems on massively parallel architectures is an important problem within computer vision. Fast relaxation algorithms can be used to integrate sparse data, resolve ambiguities in optic flow fields, and guide stereo matching algorithms. In the past, multigrid techniques have been used in order to speed up the relaxation. In this paper, we present an alternative to multigrid relaxation which is much easier to implement and more generally applicable. Our approach uses conjugate gradient descent in conjunction with a hierarchical (multiresolution) set of basis functions. The resulting algorithm uses a pyramid to smooth the residual vector before the new direction is computed. We present simulation results which show the speed of convergence and its dependence on the choice of interpolator, the number of smoothing levels, and other factors. We also discuss the relationship of this approach to other multiresolution relaxation and representation schemes.**

Index Terms—**Computer vision, conjugate gradient descent, hierarchial basis functions, multigrid relaxation, multiresolution methods, regularization, surface interpolation, visual reconstruction.**

I. Introduction

VISUAL surface interpolation [1] is an important component of low-level vision algorithms. It allows spare information—such as that obtained from feature-based stereo or motion—to be fused into a continuous visual surface. Surface interpolation is just one of a number of low-level vision algorithms that have been formalized using the theory or regularization [2], [3]. Other problems that have been studied using this formalism include scale-space stereo [4] and depth-from-motion [5].

The discrete formulation of surface interpolation and other regularization problems leads to the solution of a very large number of sparse linear equations (or equivalently, the minimization of an energy functional composed of many local energy terms). These equations map naturally onto massively parallel architectures such as the Connection Machine [6], [7], where each processor represents one node (variable) in the discrete formulation. Although iteration algorithms can be used to solve this system of equations, the convergence of these algorithms towards the true solution can be extremely slow. To speed up the convergence, multigrid techniques have been used successfully [8], [9]. Conjugate gradient descent and adaptive Chebychev acceleration methods have also been investigated [10], [11].

Recently, Yserentant devised a new relaxation technique which combines elements of both multigrid relaxation and conjugate gradient descent [12]. His approach uses a set of hierarchical basis functions, which are more global than the usual nodal basis set. This allows the algorithm to converge in $O(\log n)$ steps (where n is the number of nodes), rather than the usual $O(n)$. The hierarchical basis function representation is similar to the multiresolution pyramidal representations used in image processing [13].

In this paper, we extend Yserentant's approach to a wider variety of problem domains (such as piecewise-continuous thin plate models) and interpolants. We also develop an efficient version of the conjugate gradient algorithm which only requires a single sweep up and down a multiresolution pyramid. This algorithm works well in practice, is easy to implement, and can be applied to other problems such as 3-D surface modeling [14].

We begin this paper in Section II with a review of surface interpolation and other regularized problems and their discrete formulation. In Section III we present hierarchical basis functions, along with efficient parallel algorithms for converting between the hierarchical and nodal representations. In Section IV we review the usual conjugate gradient descent algorithm and present our new algorithm which uses smoothing of the residual to accelerate convergence. In Section V we show how to extend our algorithm to incorporate hard constraints on some of the surface points (this is used when true interpolation is desired). In Section VI we present some graphical examples of the new technique in action and numerical examples of convergence rates. In Section VII we discuss the advantages of the new approach over multigrid relaxation and discuss its relationship to other concurrent multigrid algorithms [15], [16]. We conclude that our new algorithm yields significant speedups over single-resolution relaxation techniques and is well suited to massively parallel architectures.

II. Surface Interpolation

Surface interpolation, or *visible surface reconstruction*, has been one of the most intensely studied problems in low-level computer vision [1], [17], [18], [19]. It plays a central role in the construction of a continuous $2\frac{1}{2}$-*dimen-*

Manuscript received January 20, 1989; revised January 5, 1990.
The author is with Digital Equipment Corporation, Cambridge Research Lab, One Kendall Square, Bldg. 700, Cambridge, MA 02139.
IEEE Log Number 9034825.

sional ($2\frac{1}{2}$-D) sketch from sparse visual data [20]. The solution techniques used in surface interpolation can often be applied to other low-level vision problems such as optic flow computation [21] and shape from shading [22].

A. The Variational (Continuous) Problem

The problem of interpolating a two-dimensional surface through a set of data points is usually underconstrained, i.e., there are many possible surfaces which pass through the given points. One way of resolving this problem is to heuristically choose a good interpolating algorithm, e.g., to triangulate the domain and use piecewise linear patches. Another possiblity is to cast surface interpolation as an optimization problem, e.g., to maximize the smoothness of the surface while minimizing the error of the fit to the data points. This latter approach, which is called variational spline fitting [23], is the one we examine here.

To formally characterize the optimization problem, we use *regularization* theory [24]. This mathematical technique imposes a weak smoothness constraint on the possible solutions. The functional to be minimized

$$\mathcal{E}(f) = \mathcal{E}_d(f; \{p_i\}) + \lambda \mathcal{E}_s(f) \tag{1}$$

is the weighted sum of two terms: the data compatibility constraint $\mathcal{E}_d(f; \{p_i\})$ and the smoothness constraint $\mathcal{E}_s(f)$. The regularization parameter λ is used to adjust the closeness of the fit between the surface and the data. Regularization has recently been applied to a wide range of problems in low-level computer vision [2], [9].

The data compatibility constraint measures the distance between the collection of depth values $\{p_i\} = \{(u_i, v_i, d_i)\}$ and the interpolated surface $f(u, v)$ using an energy measure

$$\mathcal{E}_d(f; \{p_i\}) = \tfrac{1}{2} \sum_i w_i (f(u_i, v_i) - d_i)^2 \tag{2}$$

where the weights w_i are inversely related to the variance of the measurements ($w_i = \sigma_i^{-2}$). This weighting of the data allows us to take into account the inherently uncertain nature of visual depth measurements [25].

The smoothness constraint (or *stabilizer* in regularization theory) is a functional or norm of $f(u, v)$ that encodes the *variation* in the surface. Two examples of possible smoothness functionals are the *membrane* model

$$\mathcal{E}_s(f) = \tfrac{1}{2} \int\int (f_u^2 + f_v^2)\, du\, dv,$$

which is a small deflection approximation of the surface area, and the *thin plate* model

$$\mathcal{E}_s(f) = \tfrac{1}{2} \int\int (f_{uu}^2 + 2f_{uv}^2 + f_{vv}^2)\, du\, dv,$$

which is a small deflection approximation of the surface curvature (the subscripts indicate partial derivatives) [3]. These two models can be combined into a single functional

$$\mathcal{E}_s(f) = \tfrac{1}{2} \int\int \rho(u, v) \left\{ [1 - \tau(u, v)] [f_u^2 + f_v^2] + \tau(u, v) [f_{uu}^2 + 2f_{uv}^2 + f_{vv}^2] \right\} du\, dv \tag{3}$$

where $\rho(u, v)$ is a *rigidity* function, and $\tau(u, v)$ is a *tension* function. The rigidity and tension functions can be used to allow depth ($\rho(u, v) = 0$) and orientation ($\tau(u, v) = 0$) discontinuities.

To see the effects of using various smoothness constraints, consider the nine data points shown in Fig. 1(a). The interpolated solution using a membrane model is shown in Fig. 1(b). Note how the surface has visible peaks and dips corresponding to the location of the data points. In practice, the membrane interpolator is not sufficiently smooth for visual surface interpolation. The interpolated solution using a thin plate model is shown in Fig. 1(c). This model may sometimes prove to be too smooth, but is often a good choice. The controlled-continuity spline is shown in Fig. 1(d). Note that a depth discontinuity has been introduced by hand along the left edge and an orientation discontinuity along the right.[1]

B. The Discrete Problem

To minimize (1) we apply the finite element method, which provides a systematic approach to the discretization and solution of variational spline problems [17]. We discretize $f(u, v)$ on a regular rectangular fine-grained mesh of *nodal variables*. The advantage of such a representation is that it is independent of the location of the data points. We can thus add more points or combine data from different resolutions without changing the algorithm. It is also easier to represent discontinuities and local variations in smoothness in this form. The regular fine-grained nature of the mesh leads naturally to simple local parallel algorithms which can execute on a massively parallel array of processors.[2]

Using a triangular conforming element for the membrane and a nonconforming rectangular element for the thin plate [17], we can derive the energy equations

$$E_p(\boldsymbol{x}) = \tfrac{1}{2} \sum_{(i,j)} \left[(x_{i+1,j} - x_{i,j})^2 + (x_{i,j+1} - x_{i,j})^2 \right] \tag{4}$$

for the membrane (the subscripts indicate spatial position) and

$$\begin{aligned} E_p(\boldsymbol{x}) = \tfrac{1}{2} h^2 \sum_{(i,j)} \big[& (x_{i+1,j} - 2x_{i,j} + x_{i-1,j})^2 \\ & + 2(x_{i+1,j+1} - x_{i,j+1} - x_{i+1,j} + x_{i,j})^2 \\ & + (x_{i,j+1} - 2x_{i,j} + x_{i,j-1})^2 \big] \end{aligned} \tag{5}$$

for the thin plate, where $h = |\Delta u| = |\Delta v|$ is the size of the mesh (isotropic in u and v). These equations hold at the interior of the surface. Near border points or discontinuities some of the energy terms are dropped or replaced by lower continuity terms [25]. The equation for the data

[1]The problem of automatically detecting discontinuities and creases in surfaces is an important problem that has been extensively studied in the field of computer vision [26], [18], [3], [27] but one that we will not address in this paper.

[2]The alternatives to our fine-grained discretization include finite elements defined over a data-dependent triangulation of the domain [28], [29] and kernel splines [30]. Nonregular or adaptive grids could also be used to give increased resolution where needed.

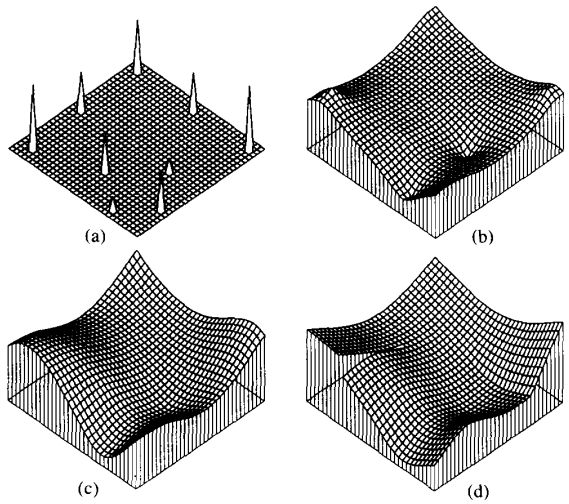


Fig. 1. Sample data points and interpolated solutions: (a) sample data points, (b) membrane interpolant, (c) thin plate interpolant, (d) controlled continuity spline (thin plate with discontinuities and creases).

compatibility energy is simply

$$E_d(\boldsymbol{x}, \boldsymbol{d}) = \tfrac{1}{2} \sum_{(i,j)} w_{i,j} (x_{i,j} - d_{i,j})^2, \tag{6}$$

with $w_{i,j} = d_{i,j} = 0$ at points where there is no input data.

If we concatenate all the nodal variables $\{x_{i,j}\}$ into one vector $\boldsymbol{x}$, we can write the prior energy model as one quadratic form

$$E_p(\boldsymbol{x}) = \tfrac{1}{2} \boldsymbol{x}^T \boldsymbol{A}_s \boldsymbol{x}, \tag{7}$$

where $\boldsymbol{x}$ is the vector of nodal variables, i.e., $\boldsymbol{x} = \{f(hi, hj)\}$. The *stiffness* matrix $\boldsymbol{A}_s$ is extremely sparse, having at most 13 entries per row. Similarly, the data constraint in (2) can be written as

$$E_d(\boldsymbol{x}, \boldsymbol{d}) = \tfrac{1}{2} (\boldsymbol{x} - \boldsymbol{d})^T \boldsymbol{A}_d (\boldsymbol{x} - \boldsymbol{d}), \tag{8}$$

where $\boldsymbol{d}$ is a zero-padded vector of data values, and the diagonal matrix $\boldsymbol{A}_d$ has entries c_i where data points coincide with nodal variables and zeros elsewhere.

Using (7) and (8), we write the combined energy (1) in discrete form as

$$E(\boldsymbol{x}) = \tfrac{1}{2} \boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x} - \boldsymbol{x}^T \boldsymbol{b} + c, \tag{9}$$

where

$$\boldsymbol{A} = \lambda \boldsymbol{A}_s + \boldsymbol{A}_d, \quad \boldsymbol{b} = \boldsymbol{A}_d \boldsymbol{d},$$

and c is a constant. This energy function has a minimum at $\boldsymbol{x} = \boldsymbol{x}^*$, the solution to the linear system of algebraic equations

$$\boldsymbol{A} \boldsymbol{x} = \boldsymbol{b}. \tag{10}$$

Other low-level vision problems which are formulated using regularization result in a similar set of equations.

C. Determining the Solution

In principle, the solution to (10) can be found using either direct or iterative numerical methods. Direct methods [31] are impractical for solving large systems associated with fine meshes because of excessive storage requirements.[3] Iterative methods [32], on the other hand, do not require any additional storage and are amenable to parallel implementations (see [33] for a survey of iterative methods applied to computer vision). The problem with straightforward relaxation schemes such as Gauss–Seidel [32] is that they are very slow to converge (Fig. 2). More sophisticated algorithms such as conjugate gradient [34] can give better performance (Fig. 3), but may still not be fast enough. Multigrid techniques [8], [35] have been applied successfully to many computer vision problems [9]. However, they are tricky to implement and require a fairly smooth solution to be effective (Section VII). What we present next is an alternative multiresolution technique that uses a different set of basis functions for the finite element discretization.

III. Hierarchical Basis Functions

The discrete equations we developed in the previous section were obtained by using *nodal basis functions*, which have local support [34]. This makes the computation of the discrete equations easier (more uniform), and results in a set of sparse equations, which are essential for a massively parallel implementation. An alternative to these nodal basis functions are the *hierarchical basis functions* developed by Yserentant [12]. In this approach, the usual nodal basis set $\boldsymbol{x}$ is replaced by a hierarchical basis set $\boldsymbol{y}$. Certain elements of the hierarchical basis set have larger support than the nodal basis elements, which allows the relaxation algorithm to converge more quickly when using the hierarchical set.

Consider the problem of representing a one-dimensional function on the interval [0, 4] using 5 nodal variables. If we used linear interpolation, the set of 5 nodal basis functions would be as shown in Fig. 4(a). Here, each basis function is a triangle function of extent 2 (except for the end functions, which are half-triangles). The hierarchical basis for this same domain would be the set of 5 functions shown in Fig. 4(b). Here, the basis functions are grouped into *levels*, with the functions at the higher (coarser) levels having a larger extent.

We can easily convert between the nodal basis representation $\boldsymbol{x}$ (a 5 element vector) and the hierarchical nodal basis $\boldsymbol{y}$ with a simple linear (matrix) transform

$$\boldsymbol{x} = \boldsymbol{S} \boldsymbol{y}. \tag{11}$$

Because of the structure of the basis function, the matrix $\boldsymbol{S}$ can be decomposed into a series of sparse matrices

$$\boldsymbol{S} = \boldsymbol{S}_1 \boldsymbol{S}_2 \cdots \boldsymbol{S}_{L-1}, \tag{12}$$

[3]For an $N \times N$ image, we require $O(N^3)$ storage and $O(N^4)$ operations for a direct solution.

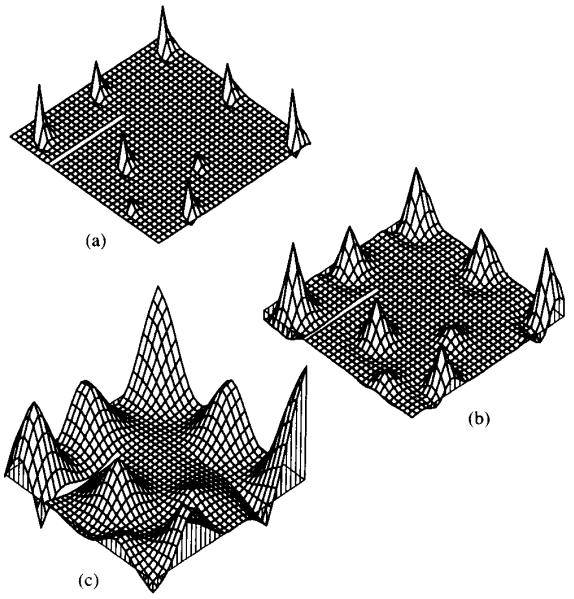


Fig. 2. Gauss-Seidel relaxation example after (a) 1 iteration, (b) 10 iterations, (c) 100 iterations.

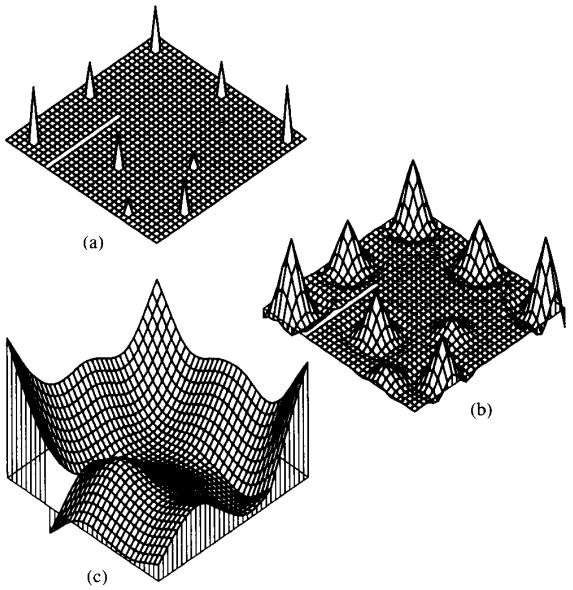


Fig. 3. Conjugate gradient relaxation example after (a) 1 iteration, (b) 10 iterations, (c) 100 iterations.

where L is the number of levels in the hierarchical basis set. Each of the component matrices S_l interpolates the nodes at level $l+1$ to level l and adds in the nodes corresponding to the new level.

For our 5 point example, we have

$$S_1 = \begin{pmatrix} 1 & & & & \\ \frac{1}{2} & 1 & \frac{1}{2} & & \\ & & 1 & & \\ & & \frac{1}{2} & 1 & \frac{1}{2} \\ & & & & 1 \end{pmatrix},$$

$$S_2 = \begin{pmatrix} 1 & & & & \\ & 1 & & & \\ \frac{1}{2} & & 1 & & \frac{1}{2} \\ & & & 1 & \\ & & & & 1 \end{pmatrix},$$

and

$$S = S_1 S_2 = \begin{pmatrix} 1 & & & & \\ \frac{3}{4} & 1 & \frac{1}{2} & & \frac{1}{4} \\ \frac{1}{2} & & 1 & & \frac{1}{2} \\ \frac{1}{4} & & \frac{1}{2} & 1 & \frac{3}{4} \\ & & & & 1 \end{pmatrix}.$$

The columns of S give the values of the hierarchical basis functions at the nodal variable locations. Note that while some of these basis functions may have global extent, the column of the component matrices S_l are of fixed size (e.g., for our 1-D linear interpolant, at most 2 off-diagonal element of a column of S_l are nonzero, while for a 2-D bilinear interpolant, at most 9 entries are nonzero).

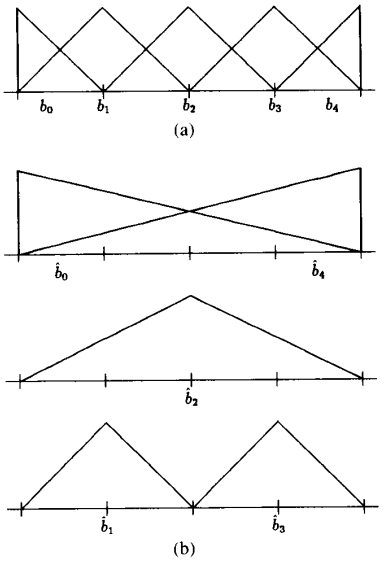


Fig. 4. Nodal and hierarchical basis functions. (a) Nodal. (b) Hierarchical.

In his paper, Yserentant uses recursive subdivision of triangles to obtain the nodal basis set. The corresponding hierarchical basis then consists of the top-level (coarse) triangulation, along with the subtriangles that are generated each time a larger triangle is subdivided. Linear in-

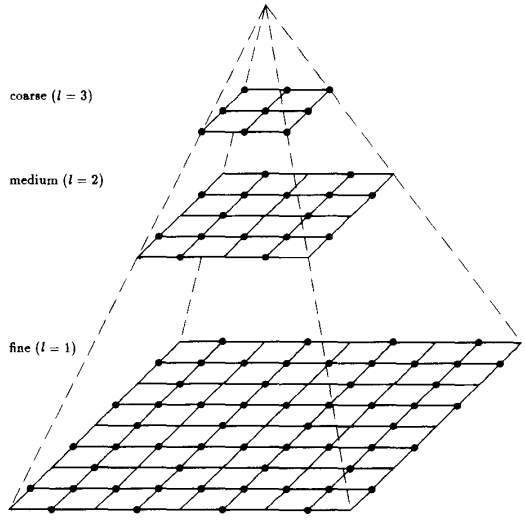


Fig. 5. Multiresolution pyramid. The circles indicate the nodes in the hierarchical basis.

terpolation is used on a triangle each time it is subdivided. In this paper, we generalize this notion to arbitrary interpolants defined over a rectangular grid. Each node in the hierarchical basis is assigned to the level in the multiresolution pyramid where it first appears (the solid dots in Fig. 5). This is similar to the multiresolution pyramid [36], [37], except that each level is only partially populated, and the total number of nodes is the same in both the nodal and hierarchical basis sets. To fully define the hierarchical basis set, we select an interpolation function that defines how each level is interpolated to the next finer level before the new node values are added in. An example of the hierarchical basis representation for the surface previously presented in Fig. 1(d) is shown in Fig. 6. In this representation, nodes that are coincident with higher level nodes have a zero value.

The resulting algorithms for mapping between the hierarchical and nodal basis sets are simple and efficient. We use

```
procedure S
  for l = L − 1 down to 1
    for i ∈ 𝔐_l
      for j ∈ 𝔑_i
        x(i) = x(i) + w(i; j) x(j)
end S
```

to convert from the hierarchical to the nodal basis set. In this procedure, which goes from the coarsest level ($l = L$) to the finest ($l = 1$), each node is assigned to one of the level collections $\mathfrak{M}_l$. Each node also has a number of neighboring nodes $\mathfrak{N}_i$ on the next coarser level that contribute to its value during the interpolation process. The $w(i; j)$ are the weighting functions that depend on the particular choice of interpolation function (these are the off-diagonal terms in the S_l matrices).

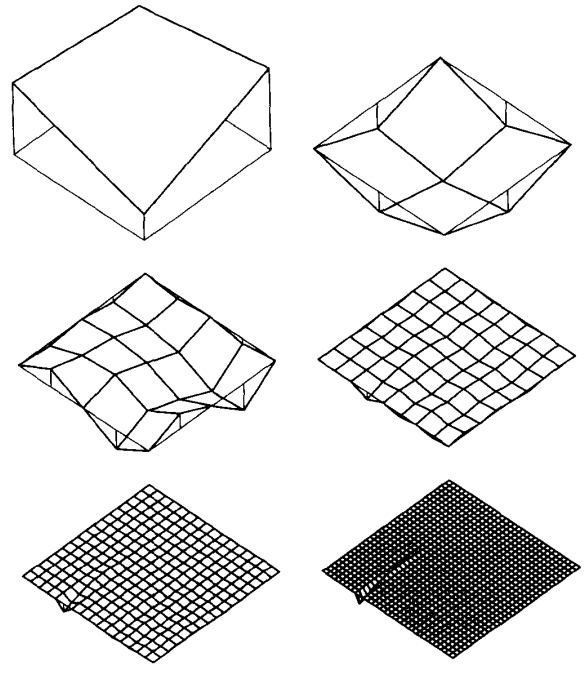


Fig. 6. Hierarchical basis representation of Fig. 1(d).

We also use the adjoint of this operation

```
procedure S^T
  for l = 1 to L − 1
    for i ∈ 𝔐_l
      for j ∈ 𝔑_i
        x(j) = x(j) + w(i; j) x(i)
end S^T
```

in the conjugate gradient descent algorithm that we develop in the next section. For completeness, we can also define the inverse mapping

```
procedure S^-1
  for l = 1 to L − 1
    for i ∈ 𝔐_l
      for j ∈ 𝔑_i
        x(i) = x(i) − w(i; j) x(j)
end S^-1
```

and its transpose

```
procedure S^-T
  for l = L − 1 down to 1
    for i ∈ 𝔐_l
      for j ∈ 𝔑_i
        x(j) = x(j) − w(i; j) x(i)
end S^-T,
```

although neither will be used in our implementation.

These mapping algorithms are easy to code (once the interpolation functions have been implemented) and require very few computations to perform. On a serial machine, each procedure uses $O(n)$ operations (multiplica-

tions and additions), where n is the number of nodes. This is because each of the local basis functions (the columns of S_l) are either the identity (no operation) or have fixed (constant) support.[4] Thus, while the basis functions in S can have global support, the hierarchical structure of S allows us to use efficient $O(N)$ recursive algorithms to compute the basis transforms. On a parallel machine, $O(L)$ parallel steps are required, where $L \leq \log_4 n$ is the number of levels in the pyramid. This is the same number of steps as is needed to perform the global summations (inner products) used in the conjugate gradient algorithm. Note that although we have defined the hierarchical basis over a pyramid, it can actually be represented in the same space as the usual nodal basis and the transformations between bases can be accomplished in place.

The hierarchical basis set allows us to minimize exactly the same energy as the one we derived from the discretization on the finest grid. Substituting $\boldsymbol{x} = \boldsymbol{S}\boldsymbol{y}$ into (9), we obtain the new energy equation

$$E(\boldsymbol{y}) = \tfrac{1}{2}\boldsymbol{y}^T(\boldsymbol{S}^T\boldsymbol{A}\boldsymbol{S})\boldsymbol{y} - \boldsymbol{y}^T(\boldsymbol{S}^T\boldsymbol{b}) + c$$
$$= \tfrac{1}{2}\boldsymbol{y}^T\hat{\boldsymbol{A}}\boldsymbol{y} - \boldsymbol{y}^T\hat{\boldsymbol{b}} + c \qquad (13)$$

where the $\hat{}$ identifies the hierarchical basis vectors and matrices. The advantage of minimizing this new equation is that the condition number of the matrix $\hat{\boldsymbol{A}}$ is much smaller than that of the original matrix $\boldsymbol{A}$ [12] (see Section VI-B). This means that iterative algorithms such as conjugate gradient will converge much faster [34].

Unfortunately, the $\hat{\boldsymbol{A}}$ matrix is not as sparse as the original matrix $\boldsymbol{A}$, so that a direct minimization of (13) is not practical. Instead, as we will see in the next section, we use the recursive mapping algorithms S and S^T in conjunction with the original matrix $\boldsymbol{A}$ and vector $\boldsymbol{b}$ to compute the required residuals and inner products.

IV. Conjugate Gradient Descent

Conjugate gradient descent is a numerical optimization technique closely related to steepest descent algorithms [34], [38]. At each step k, a direction $\boldsymbol{d}_k$ is selected in the state space, and an optimally sized step is taken in this direction. In steepest descent, the direction is always equal to the current negative gradient of the function being minimized. In conjugate gradient descent, we modify this direction so that successive directions are *conjugate* with respect to $\boldsymbol{A}$, i.e., $\boldsymbol{d}_{k+1}\boldsymbol{A}\boldsymbol{d}_k = 0$.

A description of the usual (nodal basis) conjugate gradient descent is shown in the left column of Fig. 7. We first compute the residual $\boldsymbol{r}_k$, which is the negative gradient of the energy. We then find the value of β_{k+1} that will make the new and old directions conjugate. Having selected a direction $\boldsymbol{d}_k$, we choose the optimal step size α_k so as to minimize $\Delta E(\boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k)$. This involves computing the product of the sparse matrix $\boldsymbol{A}$ and the direction $\boldsymbol{d}_k$, and computing the inner product of the resulting vector $\boldsymbol{w}_k$ and $\boldsymbol{d}_k$. On a fine-grained parallel architecture, the matrix operation is computable in time proportional to the size of the *molecules* in $\boldsymbol{A}$ (e.g., 13 for the thin plate), and the inner product summation is computable in $\log n$ steps using a summing pyramid.

For a quadratic energy equation such as (9) or (13), the conjugate gradient algorithm is guaranteed to converge to the correct solution in n steps, in the absence of roundoff error. As we mentioned in the previous section, however, we can obtain much faster convergence to an approximate solution if we minimize the energy expressed using the hierarchical basis (13) instead of the original energy (9). The resulting algorithm is shown in the right column of Fig. 7. In this algorithm, we update the state of the hierarchical basis vector $\boldsymbol{y}_k$ by computing the residual vector $\hat{\boldsymbol{r}}_k$ and direction vector $\hat{\boldsymbol{d}}_k$. To implement the matrix multiplications $\hat{\boldsymbol{A}}\hat{\boldsymbol{d}}_k$ and $\hat{\boldsymbol{A}}\boldsymbol{y}_k$, we use the mapping operations S and S^T before and after the matrix product with the original sparse matrix $\boldsymbol{A}$. In the process, we convert the quantities $\hat{\boldsymbol{d}}_k$ and $\boldsymbol{y}_k$ into the nodal representations $\boldsymbol{d}_k$ and $\boldsymbol{x}_k$. This implementation thus uses two calls to S and two calls to S^T to compute the required quantities for the conjugate gradient descent.

Inspection of the algorithm shown in Fig. 7 shows that it can be simplified to reduce the number of mappings required. We note that in steps 6 and 7 the quantities α_k^N and α_k^D (the numerator and denominator of α_k) can be computed just as easily using the regular nodal basis representation. Upon reflection, this is not surprising, since once the direction $\hat{\boldsymbol{d}}_k$ (or the equivalent $\boldsymbol{d}_k$) has been selected, the size of the step must be the same independent of which representation is used. Similarly, in step 2 we can replace the inner product $\hat{\boldsymbol{r}}_{k+1} \cdot \hat{\boldsymbol{w}}_{k+1}$ with the inner

Nodal basis conjugate gradient descent	Hierarchical basis conjugate gradient descent
0. if $k = 0$, use $\mathbf{d}_0 = \mathbf{r}_0$ in step 4.	0. if $k = 0$, use $\mathbf{d}_0 = \hat{\mathbf{r}}_0$ in step 4b.
1. $\mathbf{r}_k = \mathbf{b} - \mathbf{A}\mathbf{x}_k$	1a. $\mathbf{r}_k = \mathbf{b} - \mathbf{A}\mathbf{x}_k$
	1b. $\hat{\mathbf{r}}_k = \mathbf{S}^T\mathbf{b} - (\mathbf{S}^T\mathbf{A}\mathbf{S})\mathbf{y}_k = \mathbf{S}^T\mathbf{r}_k$
	1c. $\hat{\mathbf{r}}_k = \mathbf{S}\hat{\mathbf{r}}_k = \mathbf{S}\mathbf{S}^T\mathbf{r}_k$
2. $\beta_k^N = \mathbf{r}_k^T\mathbf{A}\mathbf{d}_{k-1} = \mathbf{r}_k \cdot \mathbf{w}_{k-1}$	2. $\beta_k^N = \hat{\mathbf{r}}_k \cdot \hat{\mathbf{w}}_{k-1} = \hat{\mathbf{r}}_k \cdot \mathbf{w}_{k-1}$
3.† $\beta_k = \beta_k^N/\alpha_{k-1}^D$	3. $\beta_k = \beta_k^N/\alpha_{k-1}^D$
	4a. $\hat{\mathbf{d}}_k = \hat{\mathbf{r}}_k - \beta_k\hat{\mathbf{d}}_{k-1}$
4. $\mathbf{d}_k = \mathbf{r}_k - \beta_k\mathbf{d}_{k-1}$	4b. $\mathbf{d}_k = \mathbf{S}\hat{\mathbf{d}}_k = \hat{\mathbf{r}}_k - \beta_k\mathbf{d}_{k-1}$
5. $\mathbf{w}_k = \mathbf{A}\mathbf{d}_k$	5a. $\mathbf{w}_k = \mathbf{A}\mathbf{d}_k = \mathbf{A}\mathbf{S}\hat{\mathbf{d}}_k$
	5b. $\hat{\mathbf{w}}_k = (\mathbf{S}^T\mathbf{A}\mathbf{S})\hat{\mathbf{d}}_k = \mathbf{S}^T\mathbf{w}_k$
6. $\alpha_k^D = \mathbf{d}_k^T\mathbf{A}\mathbf{d}_k = \mathbf{d}_k \cdot \mathbf{w}_k$	6. $\alpha_k^D = \hat{\mathbf{d}}_k \cdot \hat{\mathbf{w}}_k = \mathbf{d}_k \cdot \mathbf{w}_k$
7. $\alpha_k^N = \mathbf{d}_k \cdot \mathbf{r}_k$	7. $\alpha_k^N = \hat{\mathbf{d}}_k \cdot \hat{\mathbf{r}}_k = \mathbf{d}_k \cdot \mathbf{r}_k$
8.‡ $\alpha_k = \alpha_k^N/\alpha_k^D$	8. $\alpha_k = \alpha_k^N/\alpha_k^D$
	9a. $\mathbf{y}_{k+1} = \mathbf{y}_k + \alpha_k\hat{\mathbf{d}}_k$
9. $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k\mathbf{d}_k$	9b. $\mathbf{x}_{k+1} = \mathbf{S}\mathbf{y}_{k+1} = \mathbf{x}_k + \alpha_k\mathbf{d}_k$
10. increment k, loop to 1.	10. increment k, loop to 1a.

† ensures that $\mathbf{d}_k^T\mathbf{A}\mathbf{d}_{k-1} = 0$. We could also use
$\beta_k = \mathbf{d}_k \cdot \mathbf{d}_k/\mathbf{d}_{k-1} \cdot \mathbf{d}_{k-1}$ (Fletcher-Reeves method) *or*
$\beta_k = (\mathbf{d}_k - \mathbf{d}_{k-1}) \cdot \mathbf{d}_k/\mathbf{d}_{k-1} \cdot \mathbf{d}_{k-1}$ (Polak-Ribiere method)

‡ minimizes $\Delta E(\mathbf{x} + \alpha\mathbf{d}) = \frac{1}{2}\alpha^2\mathbf{d}^T\mathbf{A}\mathbf{d} - \alpha\mathbf{d}^T\mathbf{r}$

Fig. 7. Algorithms for nodal and hierarchical basis conjugate gradient descent.

[4] In other words, since the size of the neighborhood set $\mathfrak{N}_i$ does not change with the level, and each node i is in exactly one level set $\mathfrak{N}_l$, the number of operations is at most kN multiplications and additions, where $k = \max_i \text{card}(\mathfrak{N}_i)$ is fixed.

product $\tilde{r}_{k+1} \cdot w_{k+1}$, where $\tilde{r}_{k+1} = SS^T r_{k+1}$ is the *smoothed* residual vector.[5]

If we now use step 4b instead of 4a and use 9b instead of 9a, we obtain an algorithm that is nearly identical to the original conjugate gradient algorithm. The only difference is that we now smooth the residual vector r_{k+1} using a sweep up (S^T) and then back down (S) the pyramid to obtain the vector $\tilde{r}_{k+1}$. This smoothed vector dictates the new direction. The resulting algorithm is called the *untransformed preconditioned conjugate gradient method* [34]. We can further reduce the amount of computation at each iteration by replacing 1a with

$$r_k = r_{k-1} - \alpha_{k-1} A d_{k-1},$$

where $A d_{k-1}$ was already computed at step 5a of the previous iteration [34].

V. Hard Constraints

One of the attractions of the energy-based framework for surface interpolation presented in this paper is that we can accommodate both soft (spring-like) and hard (interpolating) data constraints. To implement a hard constraint at node i, we set $w_i = \sigma_i^{-2}$ in (2) to a very large number. Unfortunately, this increases the condition number of the A matrix, and slows down the convergence of gradient descent algorithms such as conjugate gradient and hierarchical conjugate gradient (an example of this is given in the next section).

One possibility for circumventing this is to re-write the set of linear equations $Ax = b$, replacing the ith row with the simple equation $x_i = d_i$. In the process of doing this, however, the symmetry of the A matrix is destroyed, and we can no longer minimize the old energy equation. Instead, we must minimize the squared residual $|Ax - b|^2$. The convergence of the conjugate gradient algorithm now depends on the condition number of A^TA, which is larger than that of the original matrix [34].

Another possibility is to partition the set of variables x into free variables x_1 and fixed variables (constants) x_2. We set the fixed variables to the desired values at the beginning of the program, and never change them subsequently. Let P be the matrix which permutes the nodes so that x_1 and x_2 are separated

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} P_1 x \\ P_2 x \end{bmatrix} = Px$$

(note that $P^T = P^{-1}$). We can then rewrite the energy equation (9) as

$$E(x_2) = \frac{1}{2} \begin{bmatrix} x_1^T & x_2^T \end{bmatrix} \begin{bmatrix} A_{11} & A_{12} \\ A_{12}^T & A_{22}^T \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} x_1^T & x_2^T \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} + c \qquad (14)$$

[5]Note that even though we are using a smoothed residual vector to select the direction, we are still guaranteed to be minimizing the same energy. The smoothing only affects the condition number of the A (Hessian) matrix, and hence the asymptotic rate of convergence.

where

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{12}^T & A_{22} \end{bmatrix} = PAP^T \text{ and } \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = Pb.$$

The residual (negative gradient) of the new energy equation is

$$r_1 = b_1 - (A_{11}x_1 + A_{12}x_2), = P_1 r,$$

i.e., we simply sample the residual from the original equation at the free variables. In practice, when using conjugate gradient it suffices to zero out the components of the residual and direction vectors that correspond to fixed variables.

When using the hierarchical basis approach, however, we cannot directly apply this simple technique. If we use the same set of basis functions as before (represented by the interpolation matrix S), we find that each hard constraint maps into a linear equation in y. Once again, we can no longer use a direct energy minimization approach. To overcome this problem, we partition the set of hierarchical variables into free variables y_1 and fixed variables y_2. The easiest way to do this is to make the hierarchical variables coincident with the fixed nodal variable be fixed. We can thus write $y_1 = P_1 y$ and $y_2 = P_2 y$.

Since some of the hierarchical basis variables are now fixed, we have to modify the interpolation matrix S so that the fixed nodal variables are computed using only fixed hierarchical variables. Again, the easiest way to accomplish this is to equate the two sets of variables, $y_2 = x_2$. The interpolation equation now has the form

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} S_{11} & S_{12} \\ 0 & I \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

where

$$\begin{bmatrix} S_{11} & S_{12} \\ 0 & I \end{bmatrix} = PS'P^T.$$

To compute the new interpolation matrix S', we change the rows in the S_l matrices corresponding to fixed variables to be unit vectors.

This change in the interpolation matrix (and hence the basis functions) is easy to implement in practice. In terms of the mapping algorithms presented in Section III, we simply remove the fixed nodes from the $\mathfrak{N}_l$ sets (or equivalently, we set $w(i, j)$ to zero if node i is fixed).

Using the new interpolation matrix, we can rewrite the energy equation in the hierarchical variables (13) as

$$E(y_2) = \frac{1}{2} \begin{bmatrix} y_1^T S_{11}^T + y_2^T S_{12}^T & y_2^T \end{bmatrix} \cdot \begin{bmatrix} A_{11} & A_{12} \\ A_{12}^T & A_{22}^T \end{bmatrix} \begin{bmatrix} S_{11}y_1 + S_{12}y_2 \\ y_2 \end{bmatrix} \qquad (15)$$

$$- \begin{bmatrix} y_1^T S_{11}^T + y_2^T S_{12}^T & y_2^T \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} + c. \qquad (16)$$

The residual of this energy equation is

$$\hat{r}_1 = S_{11}^T(b_1 - A_{11}S_{11}y_1 - A_{11}S_{12}y_2 - A_{12}y_2)$$
$$= S_{11}^T P_1 r. \quad (17)$$

This last expression is equivalent to applying S'^T to the residual vector with zeros at the fixed locations, and then sampling at the free variables.

The structure of the final hierarchical conjugate gradient algorithm with hard constraints is thus quite simple. We compute the residual using the usual equations, then zero out the components corresponding to the fixed variables. We then apply the new transform S'^T going up the pyramid (ignoring contributions from points which are fixed), and zero out the resulting hierarchical residual vector at fixed node locations. A sweep down the pyramid using S' (ignoring contributions to the fixed points) completes the computation of the smoothed residual $\hat{r}$. The algorithm can then proceed as before.

VI. Evaluation

A. Convergence of Iterative Algorithms

To evaluate the performance of our new algorithm, we ran a number of experiments on synthetic and real data sets. The set of points used in the first set of runs is the one shown in Fig. 1(a). For each of the three models tested (membrane, thin plate, and controlled-continuity thin plate), the optimal solution x^* for the 33 × 33 grid was computed by using 2000 iterations of conjugate gradient descent. For each experiment (defined by a suitable choice of interpolator and number of levels), the root mean squared (RMS) error

$$e_k = |x_k - x^*|/\sqrt{n}$$

was plotted as a function of the number of iterations k.

Fig. 8 shows the effects of varying the number of smoothing levels in the pyramid (L) on the convergence rate. The topmost curve ($L = 1$) is the convergence rate of the usual nodal basis conjugate gradient descent algorithm. The surprising result is that the fastest convergence is obtained when $L = 4$ and $L = 5$ instead of $L = 6$ (the full pyramid). Fig. 9 ($L = 4$) and 10 ($L = 6$) show the state of the hierarchical conjugate gradient algorithm after 1, 10, and 100 iterations. Initially, the larger number of smoothing levels used leads to a faster convergence. As time goes on, however, the large number of levels tends to over-smooth the residual vector. It seems that the optimum number of levels to be used is related to the density of the underlying data points (see Section VI-B). We are currently investigating methods for determining the optimal number of levels based on this information (i.e., based on the local structure of the stiffness matrix A). Another possibility might be to adjust the number of levels adaptively during the relaxation.

Fig. 11 shows the effects of using different interpolators on the convergence rate (for this plot, the best value of L, usually 4 or 5, was used). The bilinear interpolator and bilinear interpolator with discontinuities seem to work the best. The convergence rates for the thin plate without discontinuities (Fig. 12) and continuous membrane (Fig. 13) are even faster. Compared to coarse-to-fine Gauss-Seidel relaxation (Fig. 14), the hierarchical conjugate gradient algorithm is much faster.

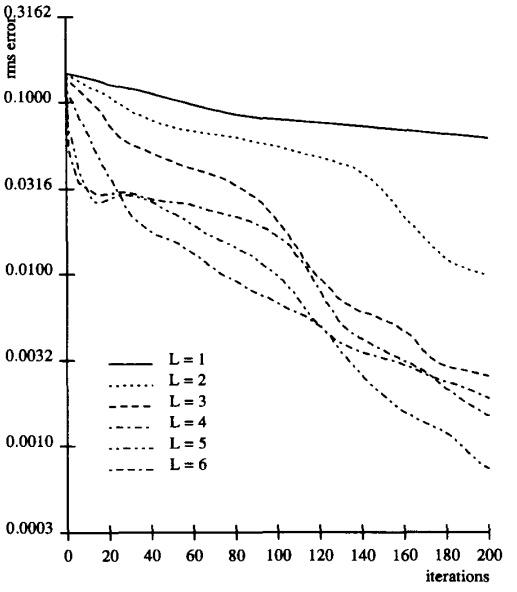


Fig. 8. Algorithm convergence as a function of L: controlled-continuity thin plate, bilinear interpolator.

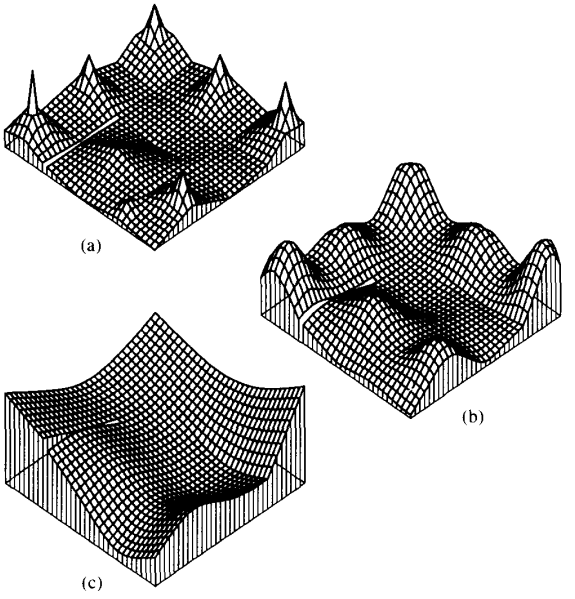


Fig. 9. Hierarchical conjugate gradient ($L = 4$) relaxation example after (a) 1 iteration, (b) 10 iterations, (c) 100 iterations.

The convergence rate of the hierarchical conjugate gradient algorithm also depends on the choice of the regularization parameter λ. As $\lambda \to 0$, the data component of the energy equation (9) starts to dominate and the condi-

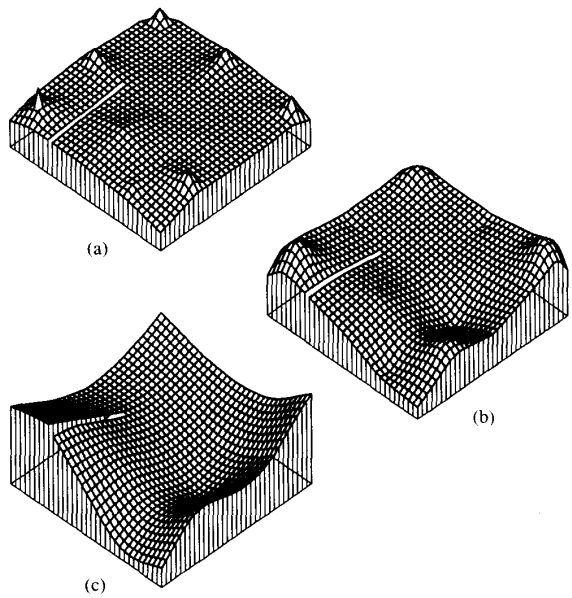


Fig. 10. Hierarchical conjugate gradient ($L = 6$) relaxation example after (a) 1 iteration, (b) 10 iterations, (c) 100 iterations.

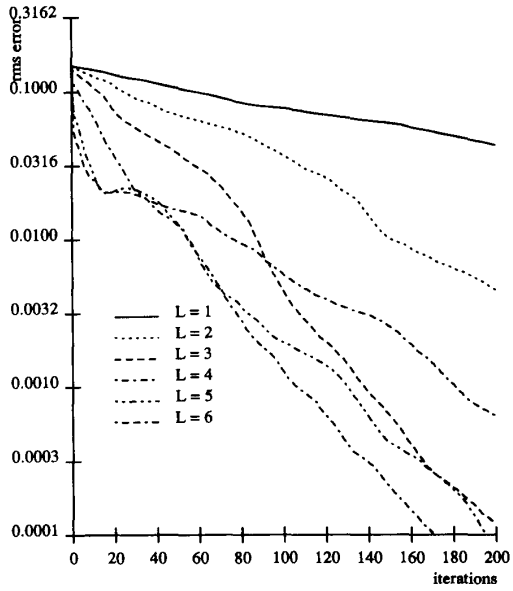


Fig. 12. Algorithm convergence as a function of L: thin plate, bilinear interpolator.

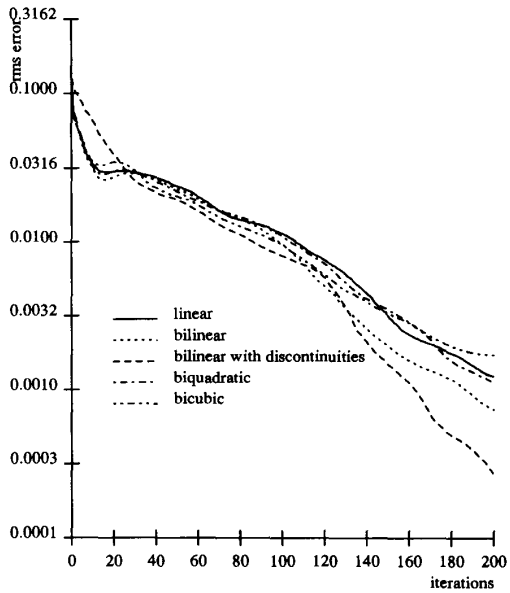


Fig. 11. Algorithm convergence as a function of interpolator: controlled-continuity thin plate, $L = 4$ or 5.

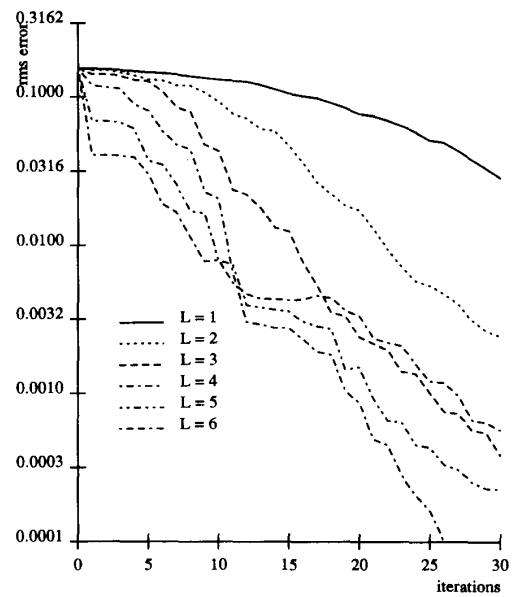


Fig. 13. Algorithm convergence as a function of L: membrane, bilinear interpolator.

tion number of the matrix A increases. To investigate this effect empirically, we ran the same controlled-continuity thin plate as before, but with smaller and larger values of λ. Fig. 15 shows the convergence rate of $\lambda = 0.1$. Compared to Fig. 8, we see that the algorithm converges more slowly. The opposite effect is seen if we set $\lambda = 10$ (Fig. 16).

As we discussed in the previous section, the alternative to using small λ when doing interpolation is to use hard constraints. The convergence of the resulting true interpolation algorithm is shown in Fig. 17. The convergence rates are slower than for $\lambda = 1$, but faster than for $\lambda = 0.1$. As λ is decreased further, the advantage of using hard constraints becomes more pronounced.

The hierarchical basis conjugate gradient algorithm has also been tested on real digital terrain data interpolated over large meshes. Fig. 18 shows an example of data that was obtained from matching isolated features in a stereo

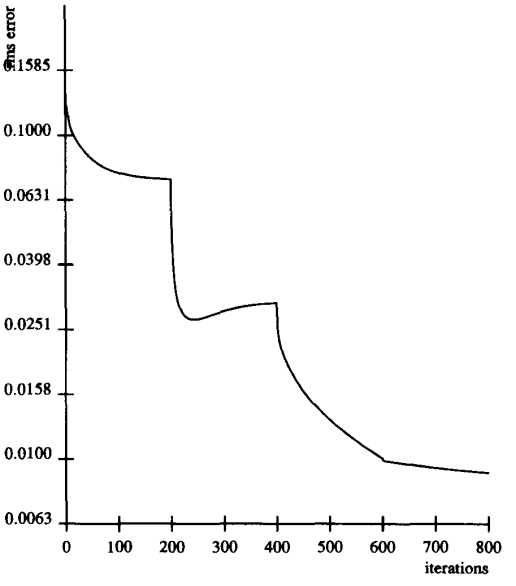


Fig. 14. Convergence of coarse-to-fine Gauss–Seidel relaxation. Four levels were used, with 200 iterations at each level, followed by bilinear interpolation. This is about the same amount of computation as 200 iterations of hierarchical conjugate gradient.

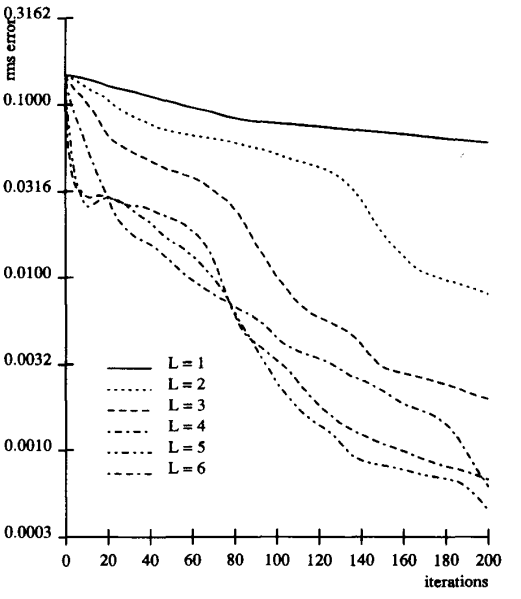


Fig. 16. Algorithm convergence for $\lambda = 10$: controlled-continuity thin plate, bilinear interpolator.

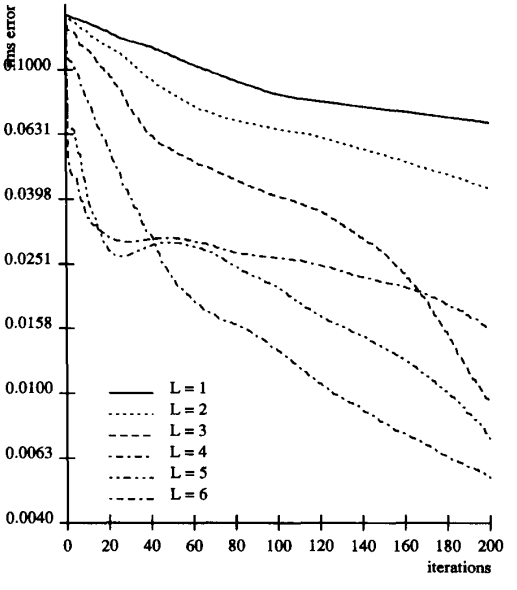


Fig. 15. Algorithm convergence for $\lambda = 0.1$: controlled-continuity thin plate, bilinear interpolator.

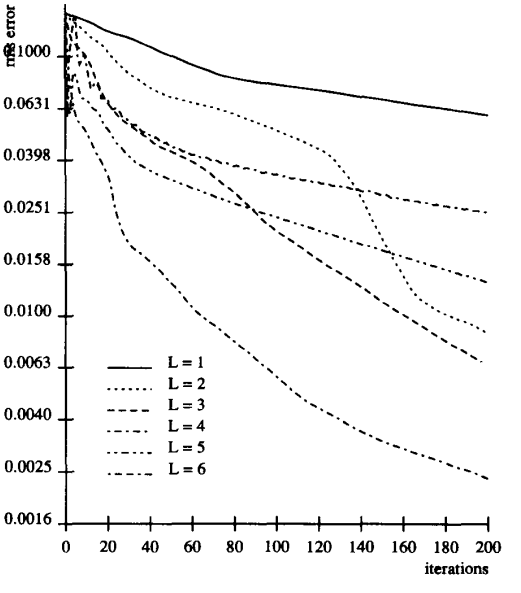


Fig. 17. Algorithm convergence for true interpolator (hard constraints): controlled-continuity thin plate, bilinear interpolator.

pair of aerial images. Fig. 18(a) shows one of the two intensity images that was input to the STEREOSYS stereo matching system [39]. Fig. 18(b) shows the set of 7696 matches that were produced by this algorithm in a subregion of this image. These points were then given as sparse elevation constraints to the thin plate interpolating program. The resulting dense elevation map (106×99 grid) is shown as a contour map in Fig. 18(c).

Fig. 19 shows the relative performance of a number of relaxation algorithms that were applied to the data set in Fig. 18 [40]. The topmost curve corresponds to weighted Jacobi relaxation ($\omega = 0.25$), which is the slowest of the algorithms tested. The second curve is for Gauss–Seidel, which initially converges faster but then also tails off. Conjugate gradient does better and continues to converge towards the solution at a reasonable rate. Using precon-

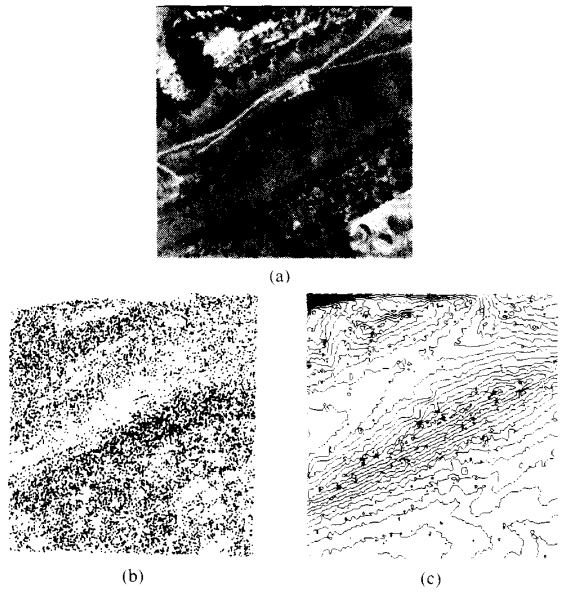


Fig. 18. Surface interpolated from sparse elevation data: (a) sample aerial photograph from stereo pair (b) matched points from stereo pair (c) interpolated thin plate solution on 106 × 99 grid.

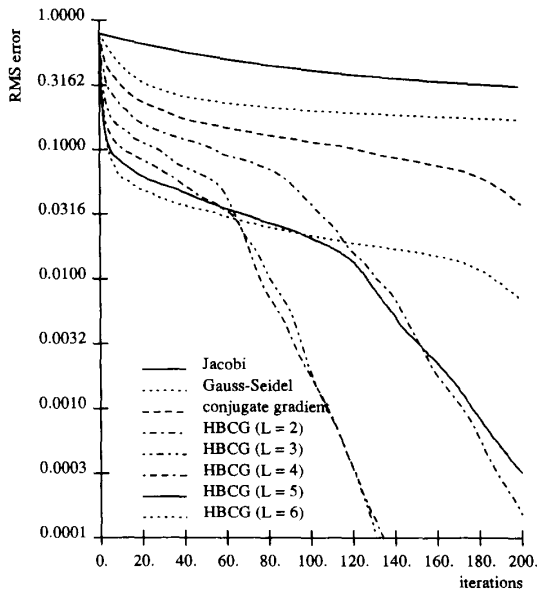


Fig. 19. Algorithm convergence for stereo data set: thin plate, bilinear interpolator.

ditioning with hierarchical basis functions dramatically speeds up the convergence of conjugate gradient. Initially, the speedup is proportional to the number of smoothing levels. As the iterations progress, however, the versions using an intermediate number of smoothing levels ($L = 3$ and $L = 4$) have the best performance. The algorithm was also tested on contour maps extracted from a complete (dense) terrain map and interpolated over a 256 × 256 grid [40].

B. Eigenvalues and Condition Numbers

The experimental results we have presented demonstrate that the hierarchical basis conjugate gradient algorithm offers dramatic speedups over traditional single-resolution algorithms. This speedup is due to the decrease in condition number as the stiffness matrix $\boldsymbol{A}$ is replaced by the preconditioned matrix $\hat{\boldsymbol{A}}$. Rather than simply relying on our experiments to demonstrate this effect, we can actually compute the condition numbers for $\boldsymbol{A}$ and $\hat{\boldsymbol{A}}$ and predict the theoretical speedups available with this approach.

The *condition number* of a symmetric positive definite matrix $\boldsymbol{A}$ is the ratio of its largest and smallest eigenvalues

$$\kappa(\boldsymbol{A}) = \lambda_n/\lambda_1. \tag{18}$$

In general, numerical methods for minimizing a functional tend to behave badly if the condition number of the Hessian $\boldsymbol{A}$ is large. In particular, it can be shown that the number of iterations required to reduce the solution error by a factor ϵ is bounded from above by some function of $\kappa(\boldsymbol{A})$ [34]. Let $p(\epsilon)$ be the smallest integer k such that

$$\|\boldsymbol{x}_k - \boldsymbol{x}^*\|_{\boldsymbol{A}} < \epsilon \|\boldsymbol{x}_0 - \boldsymbol{x}^*\|_{\boldsymbol{A}},$$

where $\|\boldsymbol{x}_{\boldsymbol{A}}\| = (\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x})^{1/2}$ is called the *energy norm*. For steepest descent methods, we have

$$p(\epsilon) \leq \tfrac{1}{2}\kappa(\boldsymbol{A}) \ln(1/\epsilon) + 1, \tag{19}$$

whereas for conjugate gradient we have

$$p(\epsilon) \leq \tfrac{1}{2}\sqrt{\kappa(\boldsymbol{A})} \ln(2/\epsilon) + 1. \tag{20}$$

The difference between these two formulas is significant. For a moderately large condition number (e.g., $\kappa = 10^4$), reducing the error by a factor of $\epsilon = 10^{-4}$ would take over 46,000 steps of steepest descent but only 426 steps of conjugate gradient descent.

To compute the minimum and maximum eigenvalues, we can use forward and inverse iteration [41]. Forward iteration,

$$\begin{aligned} \bar{\boldsymbol{x}}_k &= \boldsymbol{A}\boldsymbol{x}_{k-1} \\ \rho_k &= (\boldsymbol{x}_{k-1} \cdot \bar{\boldsymbol{x}}_k)/(\boldsymbol{x}_{k-1} \cdot \boldsymbol{x}_{k-1}) \\ \boldsymbol{x}_k &= \bar{\boldsymbol{x}}_k/(\boldsymbol{x}_{k-1} \cdot \boldsymbol{x}_{k-1})^{1/2}, \end{aligned}$$

involves repeatedly evaluating the sparse matrix/vector product $\boldsymbol{A}\boldsymbol{x}$ and provides an approximation to the largest eigenvalue $\lambda_n \doteq \rho_k$ and its associated eigenvector $\phi_n \doteq \boldsymbol{x}_k/(\boldsymbol{x}_k \cdot \boldsymbol{x}_k)^{1/2}$. Inverse iteration,

$$\begin{aligned} \boldsymbol{A}\bar{\boldsymbol{x}}_k &= \boldsymbol{x}_{k-1} \\ \rho_k &= (\bar{\boldsymbol{x}}_k \cdot \boldsymbol{x}_{k-1})/(\bar{\boldsymbol{x}}_k \cdot \bar{\boldsymbol{x}}_k) \\ \boldsymbol{x}_k &= \bar{\boldsymbol{x}}_k/(\bar{\boldsymbol{x}}_k \cdot \bar{\boldsymbol{x}}_k)^{1/2}, \end{aligned}$$

requires repeatedly solving the equations $\boldsymbol{A}\boldsymbol{x} = z$ and provides an approximation to the smallest eigenvalue $\lambda_1 \doteq \rho_k$ and its associated eigenvector $\phi_1 \doteq \boldsymbol{x}_k$. These algo-

rithms for computing the eigenvalues are easy to add to our existing surface interpolation algorithm since they do not involve any matrix factorization or determinant computation. Inverse iteration, however, can be quite slow.

When applied to our synthetic nine point example, the forward and inverse iteration procedures produce the two eigenvalues shown in Fig. 20(a) ($\lambda_1 = 0.000827$) and (b) ($\lambda_{1089} = 77.52$). The eigenvector corresponding to the smallest eigenvalue shows the low-frequency component, which is the slowest to die out in traditional single-resolution relaxation. The eigenvector corresponding to the largest eigenvalue shows the high-frequency component that dies out quickly. We can similarly compute the eigenvalues and eigenvectors for the preconditioned maxtrix $\hat{A}$ using a combination of forward/inverse iteration and the basis transform routines S, S^T, S^{-1}, and S^{-T}. Fig. 21(a) ($\lambda_1 = 0.003979$) and (b) ($\lambda_{1089} = 121.44$) show the eigenvectors[6] for the hierarchical basis discretization with $L = 3$. The low-frequency eigenvector has a larger eigenvalue associated with it than before (which is good, since it decreases the condition number). The high-frequency eigenvector shows some influence of multiresolution smoothing.[7]

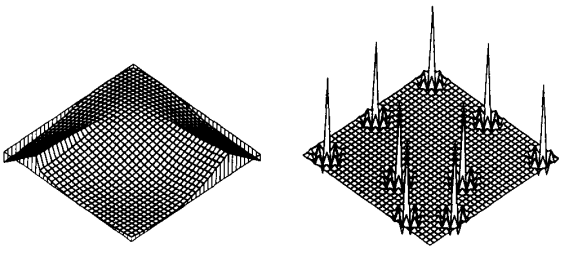
Table I shows the change in condition number and minimum/maximum eigenvalues as the number of smoothing levels in the hierarchical basis is changed. The smallest condition number is obtained for $L = 3$, which is close to the optimum $L = 4$ observed in Fig. 8. Table I also shows how the condition numbers vary with the size of the grid. As expected, the condition numbers increase for finer meshes. The optimum number of smoothing levels also increases with the grid size, which indicates that there is an optimum smoothing level independent of the fine grid size.

Table II shows the change in condition number as the interpolator and number of smoothing levels are changed. From the error plots in Fig. 11, it seemed that the rate of convergence was not very dependent on the choice of interpolator. Table II, on the other hand, shows that using the bilinear interpolator with discontinuities significantly reduces the condition number, and hence the theoretical rate of convergence. It is difficult to say if this difference will turn out to be significant in practice, where the convergence depends not only on the condition number, but also on the starting state.

The condition numbers for the thin plate and membrane interpolators without discontinuities are shown in Tables III and IV. The condition numbers for both of these cases are lower to start with and show a much more moderate growth with grid size when the hierarchical basis set is used. These two cases are more similar to the problem

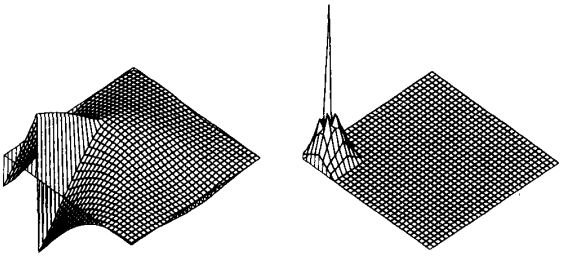
[6]Actually, the eigenvectors are in the hierarchical basis $\hat{\phi}$, so what we display are the eigenvectors transformed back into the nodal basis $\phi' = S\hat{\phi}$.

[7]Only a single data point appears in this eigenvector since it was the only one coincident with higher level grids. This points out some of the asymmetry of the hierarchical basis with the respect to data point placement.



(a) (b)

Fig. 20. Eigenvectors for original stiffness A: (a) $\lambda_1 = 0.000827$, (b) $\lambda_{1089} = 77.52$.



(a) (b)

Fig. 21. Eigenvectors for preconditioned stiffness $\hat{A}$ with $L = 3$: (a) $\lambda_1 = 0.003979$, (b) $\lambda_{1089} = 121.44$.

that Yserentant analyzed in his paper [12]. However, he analyzed a model problem where there were no inhomogeneous data constraints inside the solution domain, whereas in our case, we have sparse data constraints throughout. This explains why for our data sets, using the maximum number of smoothing levels possible is not the best choice.

We can also study the dependence of the condition numbers on the regularization parameter λ (not to be confused with the eigenvalues λ_1 and λ_n). Table V shows this dependence. The first line ($\lambda = 0.1$) shows the growth in condition number as the smoothness constraint is reduced (Section V). It also shows how the optimal number of smoothing levels grows with the overall smoothness of the solution.

VII. Discussion

A. Comparison to Multigrid

From the experiments described in the previous section, we see that the hierarchical basis conjugate gradient algorithm is dramatically faster than single-resolution conjugate gradient (which is itself much faster than Gauss–Seidel). Similar speedups have been reported using multigrid techniques [8]. Based on our experience in having implemented both algorithms, we believe that the hierarchical basis conjugate gradient is better suited for computer vision applications for three reasons: ease of implementation, suitability for massively parallel architectures, and wider range of applicability.

When designing a multigrid algorithm [35], we must

TABLE I
CONDITION NUMBER FOR CONTROLLED-CONTINUITY SPLINE AS A FUNCTION OF GRID SIZE AND NUMBER OF SMOOTHING LEVELS. BILINEAR INTERPOLATOR, $\lambda = 1$.

grid size	$L = 1$	$L = 2$	$L = 3$	$L = 4$	$L = 5$	$L = 6$
9×9	$\frac{51.24}{0.006741}$ 7601	$\frac{50.25}{0.011745}$ 4279	$\frac{170.73}{0.008643}$ 19752	$\frac{453.94}{0.008678}$ 52307	$\frac{\lambda_{min}}{\lambda_{max}}$ κ	$\frac{\lambda_{min}}{\lambda_{max}}$ κ
17×17	$\frac{55.31}{0.002629}$ 21036	$\frac{104.26}{0.006596}$ 15807	$\frac{162.68}{0.007258}$ 22415	$\frac{287.05}{0.008064}$ 47334	$\frac{614.49}{0.006114}$ 100509	$\frac{\lambda_{min}}{\lambda_{max}}$ κ
33×33	$\frac{77.52}{0.000827}$ 93788	$\frac{88.50}{0.002690}$ 32896	$\frac{121.44}{0.003979}$ 30524	$\frac{164.97}{0.004080}$ 40433	$\frac{289.62}{0.003729}$ 77664	$\frac{633.14}{0.003751}$ 168792
65×65	$\frac{255.49}{0.000221}$ 1155631	$\frac{203.39}{0.000830}$ 244980	$\frac{203.53}{0.001811}$ 112394	$\frac{204.27}{0.002177}$ 93843	$\frac{205.75}{0.002198}$ 93626	$\frac{211.07}{0.002102}$ 100409

TABLE II
CONDITION NUMBER FOR CONTROLLED-CONTINUITY SPLINE AS A FUNCTION OF INTERPOLATOR AND NUMBER OF SMOOTHING LEVELS. 33×33 GRID SIZE, $\lambda = 1$.

grid size	$L = 1$	$L = 2$	$L = 3$	$L = 4$	$L = 5$	$L = 6$
linear	$\frac{77.52}{0.000827}$ 93788	$\frac{128.28}{0.002642}$ 48559	$\frac{166.52}{0.003692}$ 45098	$\frac{230.82}{0.003654}$ 63168	$\frac{285.80}{0.003369}$ 84838	$\frac{626.86}{0.003388}$ 185042
bilinear	$\frac{77.52}{0.000827}$ 93788	$\frac{88.50}{0.002690}$ 32896	$\frac{121.44}{0.003979}$ 30524	$\frac{164.97}{0.004080}$ 40433	$\frac{289.62}{0.003729}$ 77664	$\frac{633.14}{0.003751}$ 168792
bilinear w/ discon.	$\frac{77.52}{0.000827}$ 93788	$\frac{88.50}{0.002977}$ 29725	$\frac{121.44}{0.008731}$ 13909	$\frac{164.95}{0.016276}$ 10135	$\frac{286.20}{0.008373}$ 34183	$\frac{621.97}{0.007940}$ 78336
biquadratic	$\frac{77.52}{0.000827}$ 93788	$\frac{113.95}{0.002841}$ 40102	$\frac{194.75}{0.005170}$ 37671	$\frac{259.92}{0.005637}$ 46111	$\frac{364.62}{0.004966}$ 73426	$\frac{736.69}{0.005001}$ 147319
bicubic	$\frac{77.52}{0.000827}$ 93788	$\frac{101.94}{0.002641}$ 38596	$\frac{197.06}{0.003769}$ 52279	$\frac{271.00}{0.003699}$ 73263	$\frac{408.37}{0.003399}$ 120146	$\frac{822.72}{0.003416}$ 240816

TABLE III
CONDITION NUMBER FOR THIN PLATE AS A FUNCTION OF GRID SIZE AND NUMBER OF SMOOTHING LEVELS. BILINEAR INTERPOLATOR, NO DISCONTINUITIES, $\lambda = 1$.

grid size	$L = 1$	$L = 2$	$L = 3$	$L = 4$	$L = 5$	$L = 6$
9×9	$\frac{51.27}{0.02240}$ 2289	$\frac{50.36}{0.04216}$ 1194	$\frac{170.76}{0.02998}$ 5695	$\frac{453.97}{0.02012}$ 22558	$\frac{\lambda_{min}}{\lambda_{max}}$ κ	$\frac{\lambda_{min}}{\lambda_{max}}$ κ
17×17	$\frac{55.36}{0.00608}$ 9105	$\frac{104.28}{0.02232}$ 4672	$\frac{162.70}{0.03890}$ 4182	$\frac{287.07}{0.02798}$ 10262	$\frac{614.55}{0.01908}$ 32210	$\frac{\lambda_{min}}{\lambda_{max}}$ κ
33×33	$\frac{77.65}{0.00156}$ 49691	$\frac{88.51}{0.00615}$ 14402	$\frac{121.57}{0.02211}$ 5498	$\frac{165.15}{0.03788}$ 4360	$\frac{289.62}{0.02689}$ 10769	$\frac{633.46}{0.01860}$ 34056

TABLE IV
CONDITION NUMBER FOR MEMBRANE AS A FUNCTION OF GRID SIZE AND NUMBER OF SMOOTHING LEVELS. BILINEAR INTERPOLATOR, NO DISCONTINUITIES, $\lambda = 1$.

grid size	$L = 1$	$L = 2$	$L = 3$	$L = 4$	$L = 5$	$L = 6$
9×9	$\frac{55.44}{0.4685}$ 118	$\frac{54.83}{1.0038}$ 55	$\frac{172.95}{0.6719}$ 257	$\frac{455.96}{0.5167}$ 882	$\frac{\lambda_{min}}{\lambda_{max}}$ κ	$\frac{\lambda_{min}}{\lambda_{max}}$ κ
17×17	$\frac{59.74}{0.0916}$ 652	$\frac{109.28}{0.3309}$ 330	$\frac{167.70}{0.7232}$ 232	$\frac{290.62}{0.4872}$ 597	$\frac{616.55}{0.3537}$ 1743	$\frac{\lambda_{min}}{\lambda_{max}}$ κ
33×33	$\frac{83.41}{0.0232}$ 3589	$\frac{92.57}{0.0906}$ 1022	$\frac{124.77}{0.3270}$ 382	$\frac{167.05}{0.5791}$ 288	$\frac{293.11}{0.3912}$ 749	$\frac{635.35}{0.2776}$ 2289

first devise a hierarchy of problems, i.e., for each level, we must rederive the finite element equations (this often involves averaging data from the finer level). We also have to specify both the injection (subsampling) and prolongation (interpolation) operations, as well as to choose an interlevel coordination scheme. With hierarchical basis functions, only a single interpolation function needs to be specified. There is no need to explicitly build a pyramid for representation or computation (this is also true for some multigrid techniques). Most of the computation proceeds in parallel at the fine level, with only occasional excursions up or down the virtual pyramid for summing or smoothing.[8] Since we can choose the interpolation function (hierarchical basis) independent of the problem being solved, we have much greater flexibility. For ex-

[8]Recently, *concurrent multigrid* techniques have been developed which also make full use of processors in a massively parallel architecture [42].

TABLE V
CONDITION NUMBER FOR CONTROLLED-CONTINUITY SPLINE AS A FUNCTION OF REGULARIZATON PARAMETER λ AND NUMBER OF SMOOTHING LEVELS. 33 × 33 GRID SIZE, BILINEAR INTERPOLATOR, $\lambda = 1$.

grid size	$L = 1$	$L = 2$	$L = 3$	$L = 4$	$L = 5$	$L = 6$
$\lambda = 0.1$	52.06 / 0.0000282 / 1843436	76.05 / 0.0002693 / 282408	120.51 / 0.0003982 / 302658	159.62 / 0.0004084 / 390847	288.70 / 0.0003732 / 773508	635.75 / 0.0003754 / 1693483
$\lambda = 1.0$	77.52 / 0.000827 / 93788	88.50 / 0.002690 / 32896	121.44 / 0.003979 / 30524	164.97 / 0.004080 / 40433	289.62 / 0.003729 / 77664	633.14 / 0.003751 / 168792
$\lambda = 10.0$	635.02 / 0.00810 / 78435	506.03 / 0.02659 / 19032	507.20 / 0.03944 / 12860	510.54 / 0.04042 / 12629	514.91 / 0.03699 / 13922	516.70 / 0.03721 / 13884
$\lambda = 100.0$	6345.70 / 0.0661398 / 95944	5060.18 / 0.231313 / 21876	5071.73 / 0.363838 / 13940	5103.24 / 0.372443 / 13702	5136.92 / 0.343573 / 14951	5140.44 / 0.346246 / 14846

ample, wavelets [43] could be used as an alternative to the polynomial bases that we have studied in this paper.

In our own implementation of multigrid, we were unable to achieve a convergence much better than that of the simple coarse-to-fine algorithm shown in Fig. 14. This is probably due to the inhomogeneous nature of the sparse data constraints, and could be remedied by carefully tailoring the injection and prolongation operations. This serves to point out one of the weaknesses of traditional multigrid, which is designed for solving smooth homogeneous partial differential equations with known boundary conditions. For computer vision problems, the task often involves integrating sparse data with varying degrees of uncertainty scattered over the visual field [25]. In such a situation, using conjugate gradient descent combined with hierarchical basis functions will still work when a simple implementation of multigrid might fail.[9]

B. Relative Representations

The hierarchical basis function representation is closely related to *relative* multiresolution representations, where the surface being modeled is obtained by summing the values at the various pyramid levels [25], [16]. In the hierarchical basis representation, the number of variables is the same as in the fine level representation, and the transformation between nodal and hierarchical bases is inevitable. It is therefore easy to formulate problems on the fine grid and solve them on the pyramid. An alternative to this approach is to use a full pyramid, which has more degrees of freedom than the original problem. To make the problem well-posed, we must equip each level with its own independent smoothness constraint, and require that the sum of the levels match the data constraints [25].

[9]If we were interpolating the data to an extremely fine grid, then the theoretical computation cost of full multigrid, which is $O(n)$ [35], would be better than that of hierarchical basis conjugate gradient, which is $O(n \log n)$ [12]. However, it is rare for computer vision problems to be so largely underconstrained and to require such high-resolution results.

This alternative is currently being investigated, along with other concurrent multigrid algorithms [16].

C. Extensions

The hierarchical basis $\boldsymbol{S}$ is an example of a *preconditioner*, which helps to reduce the condition number of the stiffness $\boldsymbol{A}$. Another popular choice for the preconditioner is $\boldsymbol{\Lambda}^{-1/2}$, where $\boldsymbol{\Lambda}$ is the diagonal matrix formed from $\boldsymbol{A}$. This *diagonal scaling* [34] produces a preconditioned stiffness matrix $\hat{\boldsymbol{A}}$ with all ones on the diagonal, and is quite effective at reducing the adverse effects of stiff data constraints (large w_i's or small λ). We are currently investigating how to combine diagonal scaling with hierarchical basis functions to obtain a method that automatically determines the best number of smoothing levels. Ideally, we would like our bases (preconditioners) to match the natural eigenvectors of the system as closely as possible, which would provide optimal speedups for our relaxation. In practice, we are constrained to finding hierarchical bases (i.e., ones which are recursively defined) that limit the amount of computation involved in the smoothing/scaling step.

The hierarchical basis function idea can easily be extended to domains other than two-dimensional surfaces. It could just as easily be applied to 3-D elastic models that use cylindrical coordinates [14], or to 3-D elastic net models built from a recursively tessellated sphere [25]. They are easy to implement because we do not have to create a whole family of related problems, as in multigrid. Instead, we can simply define a multiresolution set of basis functions (e.g., relying on the recursive tessellation of the method) and apply it as a preconditioner.

We are currently examining the application of our new approach to other computer vision problems such as shape from shading [44]. Here, the nonlinear nature of the data constraints (due to the nonlinear reflectance map) causes problems with traditional multigrid implementations [9], [45]. By using our hierarchical basis functions, we can solve the fine-level problem without any multiresolution

approximations, and simply use the hierarchical basis to speed up the relaxation.

VIII. Conclusions

In this paper, we have presented a new algorithm for efficiently solving surface interpolation and other regularized computer vision problems. The algorithm is based on the hierarchical basis functions devised by Yserentant. Because of the recursive formulation of the basis set, conversion between the usual nodal basis and the hierarchical basis is extremely efficient. This allows us to devise a new conjugate gradient descent algorithm which uses a smoothed version of the residual vector to determine the new candidate direction. This new algorithm is easy to implement, both on serial and parallel machines. On a serial machine, the smoothing step takes $O(n)$ operations, where n is the number of nodes. On a parallel machine, $O(\log n)$ parallel steps must be used.

The new conjugate gradient algorithm has been tested on a number of synthetic and natural data sets. It performs much better than single resolution schemes and coarse-to-fine relaxation. The optimal number of levels to be used in the smoothing pyramid and the optimal choice of interpolator seem to be problem dependent. Fortunately, this choice does not significantly affect the speedups available with this technique.

Because of the simplicity of the central idea in the algorithm (the multiresolution smoothing of the residual vector), this same algorithm is applicable to a wide variety of numerical relaxation problems, including those involving 3-D energy-based models. We are currently investigating other multiresolution relaxation schemes, in order to find efficient relaxation algorithms that can be implemented on parallel architectures and to build better multiresolution descriptions of the visual world.

Acknowledgment

I would like to thank O. Widlund for bringing Prof. Yserentant's paper to my attention, and D. Terzopoulos and the two anonymous reviewers for their comments on this paper.

References

[1] W. E. L. Grimson, "An implementation of a computational theory of visual surface interpolation," *Comput. Vision, Graphics, Image Processing*, vol. 22, pp. 39–69, 1983.

[2] T. Poggio, V. Torre, and C. Koch, "Computational vision and regularization theory," *Nature*, vol. 317, pp. 314–319, Sept. 26, 1985.

[3] D. Terzopoulos, "Regularization of inverse visual problems involving discontinuities," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-8, no. 4, pp. 413–424, July 1986.

[4] A. Witkin, D. Terzopoulos, and M. Kass, "Signal matching through scale space," *Int. J. Comput. Vision*, vol. 1, pp. 133–144, 1987.

[5] L. Matthies, T. Kanade, and R. Szeliski, "Kalman filter-based algorithms for estimating depth from image sequences," *Int. J. Comput. Vision*, vol. 3, pp. 209–236, 1989.

[6] J. L. Potter, Ed., *The Massively Parallel Processor*. Cambridge, MA: MIT Press, 1985.

[7] W. D. Hillis, *The Connection Machine*. Cambridge, MA: MIT Press, 1985.

[8] W. Hackbusch, *Multigrid Methods and Applications*. Berlin: Springer-Verlag, 1985.

[9] D. Terzopoulos, "Image analysis using multigrid relaxation methods," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-8, no. 2, pp. 129–139, Mar. 1986.

[10] D. J. Choi, "Solving the depth interpolation problem on a fine grained, mesh- and tree-connected SIMD machine," in *Proc. Image Understanding Workshop*, Los Angeles, CA, Feb. 1987. Los Altos, CA: Morgan Kaufmann, 1987, pp. 639–643.

[11] T. Simchony, R. Chellappa, and Z. Lichtenstein, "Pyramid implementation of optimal step conjugate search algorithms for some computer vision problems," *Proc. Second Int. Conf. Computer Vision (ICCV'88)*, Tampa, FL, Dec. 1988. Washington, DC: IEEE Computer Society Press, 1988, pp. 580–590.

[12] H. Yserentant, "On the multi-level splitting of finite element spaces," *Numerische Mathematik*, vol. 49, pp. 379–412, 1986.

[13] A. Rosenfeld, Ed., *Multiresolution Image Processing and Analysis*. New York: Springer-Verlag, 1984.

[14] D. Terzopoulos, A. Witkin, and M. Kass, "Symmetry-seeking models and 3D object reconstruction," *Int. J. Comput. Vision*, vol. 1, no. 3, pp. 211–221, Oct. 1987.

[15] D. Terzopoulos, "Concurrent multilevel relaxation," in *Proc. Image Understanding Workshop*, L. S. Baumann, Ed., Miami Beach, FL, Dec. 1985. Science Applications Int. Corp., 1985, pp. 156–162.

[16] R. Szeliski and D. Terzopoulos, "Parallel multigrid algorithms and applications to computer vision," in *Proc. Fourth Copper Mountain Conf. Multigrid Methods*, Copper Mountain, CO, Apr. 1989. Philadelphia, PA: Soc. Industrial and Applied Mathematics, 1989, pp. 383–398.

[17] D. Terzopoulos, "Multilevel computational processes for visual surface reconstruction," *Comput. Vision, Graphics, Image Processing*, vol. 24, pp. 52–96, 1983.

[18] A. Blake and A. Zisserman, *Visual Reconstruction*. Cambridge, MA: MIT Press, 1987.

[19] D. Terzopoulos, "The computation of visible-surface representations," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 10, no. 4, pp. 417–438, July 1988.

[20] D. Marr, "Representing visual information," in *Computer Vision Systems*, A. R. Hanson and E. M. Riseman, Eds. New York: Academic, 1978, pp. 61–80.

[21] B. K. P. Horn and B. G. Schunck, "Determining optical flow," *Artif. Intell.*, vol. 17, pp. 185–203, 1981.

[22] K. Ikeuchi and B. K. P. Horn, "Numerical shape from shading and occluding boundaries," *Artif. Intell.*, vol. 17, pp. 141–184, 1981.

[23] J. H. Ahlberg, E. N. Nilson, and J. L. Walsh, *The Theory of Splines and Their Applications*. New York: Academic, 1967.

[24] A. N. Tikhonov and V. Y. Arsenin, *Solutions of Ill-Posed Problems*. Washington, DC: Winston, 1977.

[25] R. Szeliski, *Bayesian Modeling of Uncertainty in Low-Level Vision*. Boston, MA: Kluwer Academic, 1989.

[26] J. L. Marroquin, "Surface reconstruction preserving discontinuities," Artif. Intell. Lab., Massachusetts Inst. Technol., A. I. Memo 792, Aug. 1984.

[27] Y. G. Leclerc, "Constructing simple stable descriptions for image partitioning," *Int. J. Comput. Vision*, vol. 3, no. 1, pp. 75–104, 1989.

[28] H. Fuchs, Z. M. Kedem, and S. P. Uselton, "Optimal surface reconstruction from planar contours," *Commun. ACM*, vol. 20, no. 10, pp. 693–702, Oct. 1977.

[29] Z. J. Cendes and S. H. Wong, "C^1 quadratic interpolation over arbitrary point sets," *IEEE Comput. Graphics Applications*, vol. 7, no. 11, pp. 8–16, Nov. 1987.

[30] T. E. Boult, "Information based complexity in non-linear equations and computer vision," Ph.D. dissertation, Columbia Univ., 1986.

[31] A. George and J. W. H. Liu, *Computer Solution of Large Sparse Positive Definite Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1981.

[32] D. M. Young, *Iterative Solution of Large Linear Systems*. New York: Academic, 1971.

[33] D. Terzopoulos, "Multiresolution computation of visible-surface representations," Ph.D. dissertation, Massachusetts Inst. Technol., Jan. 1984.

[34] O. Axelsson and V. A. Barker, *Finite Element Solution of Boundary*

Value Problems: Theory and Computation. Orlando, FL: Academic, 1984.
[35] W. L. Briggs, *A Multigrid Tutorial*. Philadelphia, PA: Soc. Industrial and Applied Mathematics, 1987.
[36] P. J. Burt and E. H. Adelson, "The Laplacian pyramid as a compact image code," *IEEE Trans. Commun.*, vol. COM-30, no. 4, pp. 532-540, Apr. 1983.
[37] J. L. Crowley and R. M. Stern, "Fast computation of the difference of low-pass transform," Robotics Inst., Carnegie-Mellon Univ., Tech. Rep. CMU-RI-TR-82-18, Nov. 1982.
[38] W. Press *et al.*, *Numerical Recipes: The Art of Scientific Computing*. Cambridge, England: Cambridge University Press, 1986.
[39] M. J. Hannah, "Test results from SRI's stereo system," in *Proc. Image Understanding Workshop*, Cambridge, MA, Apr. 1988. Los Altos, CA: Morgan Kaufmann, 1988, pp. 740-744.
[40] R. Szeliski, "Fast parallel surface interpolation with applications to digital cartography," Artif. Intell. Center, SRI International, Tech. Note 470, June 1989.
[41] K.-J. Bathe and E. L. Wilson, *Numerical Methods in Finite Element Analysis*. Englewood Cliffs, NJ: Prentice-Hall, 1976.
[42] O. McBryan, "The connection machine: PDE solution on 65,536 processors," Thinking Machines Corp., Cambridge, MA, Tech. Rep. NA86-3, 1986.
[43] S. G. Mallat, "A compact multiresolution representation: The wavelet model," in *Proc. IEEE Comput. Soc. Workshop Computer Vision*, Miami Beach, FL, Dec. 1987. Washington, DC: IEEE Computer Society Press, 1987, pp. 2-7.
[44] R. Szeliski, "Fast shape from shading," in *First European Conf. Computer Vision*, Antibes, France, Apr. 1990.
[45] G. Ron and S. Peleg, "Multiresolution shape from shading," in *Proc. IEEE Comput. Soc. Conf. Computer Vision and Pattern Recognition (CVPR'89)*, San Diego, CA, June 1989. Washington, DC: IEEE Computer Society Press, 1989, pp. 350-355.

Richard Szeliski (S'76-M'82-S'83-M'88) was born in Montreal, P.Q., Canada, in 1958. He received the B.Eng. degree in Honours electrical engineering from McGill University, Montreal, in 1979, the M.A.Sc. degree in electrical engineering from the University of British Columbia, Vancouver, B.C., in 1981, and the Ph.D. degree in computer science from Carnegie-Mellon University, Pittsburgh, PA, in 1988.

During his studies, he held summer positions at Bell-Northern Research, Montreal, in 1979, and at the NEC Corporation, Kawasaki, in 1985. From January 1982 to August 1983 he was a Member of Scientific Staff at Bell-Northern Research, Montreal, where he designed image coding and image processing systems. From September 1988 to January 1989 he was a Member of Scientific Staff at Schlumberger Palo Alto Research, Palo Alto, CA, where he worked on fast relaxation algorithms for computer vision and elastic surface modeling. From January 1989 to June 1989 he was a Visiting Scientist at the Artificial Intelligence Center of SRI International, Menlo Park, CA, where he developed fast parallel algorithms on the Connection Machine for digital cartography and terrain modeling. Since July 1989 he has been a Member of Research Staff at the Cambridge Research Lab of Digital Equipment Corporation, Cambridge, MA, where he is pursuing interests in intermediate-level computer vision and parallel programming and algorithms. He has published research papers in computer vision, computer graphics, neural nets, and parallel numerical algorithms.

Dr. Szeliski is a member of the American Association for Artificial Intelligence, the Association for Computing Machinery, and Sigma Xi.