PRECONDITIONERS FOR INDEFINITE LINEAR SYSTEMS ARISING IN SURFACE PARAMETERIZATION

J. Liesen[1], E. de Sturler[2], A. Sheffer[1], Y. Aydin[3], and C. Siefert[2]

[1] *Computational Science and Engineering Program*
[2] *Department of Computer Science*
[3] *Department of Mathematics*
University of Illinois at Urbana-Champaign, Urbana, Illinois 61801, USA.
{ *liesen* | *sturler* | *sheffa* | *aydin* | *siefert* }*@uiuc.edu*

ABSTRACT

In [19] we introduced a new algorithm for computing planar triangulations of faceted surfaces for surface parameterization. Our algorithm computes a mapping that minimizes the distortion of the surface metric structures (lengths, angles, etc.). Compared with alternative approaches, the algorithm provides a significant improvement in robustness and applicability; it can handle more complicated surfaces and it does not require a convex or predefined planar domain boundary. However, our algorithm involves the solution of a constrained minimization problem. The potential high cost in solving the optimization problem has given rise to concerns about the applicability of the method, especially for very large problems. This paper is concerned with the efficient solution of the symmetric indefinite linear systems that arise when Newton's method is applied to the constrained minimization problem. In small to moderate size models the linear systems can be solved efficiently with a sparse direct method. We give examples from computations with the SuperLU package [6]. For larger models we have to use preconditioned iterative methods. We develop a new preconditioner that takes into account the structure of our linear systems. Some preliminary experimental results are shown that indicate the effectiveness of this approach.

Keywords: surface parameterization, triangulation, flattening, linear systems, Krylov subspace methods, preconditioning, indefinite problems

1. INTRODUCTION

In [19] we introduced a new algorithm for computing planar triangulations of faceted surfaces for surface parameterization. Our algorithm computes a mapping that minimizes the distortion of the surface metric structures (lengths, angles, etc.). Compared with alternative approaches, the algorithm provides a significant improvement in robustness and applicability; it can handle more complicated surfaces and it does not require a convex or predefined boundary. The qualities of our algorithm derive from two important choices. The first is to formulate the mapping from the three-dimensional tesselation to the planar one in terms of angles instead of node locations. The second choice is to formulate the problem in terms of a constrained optimization problem.

We minimize the relative deformation of the angles in the plane with respect to their counterparts in the three-dimensional surface, while satisfying a set of constraints on the angles that ensure the validity of the flat mesh.

The solution of a constrained minimization problem may be expensive, and in our earlier papers we did not concentrate on this aspect. We did introduce iterative solvers in [20], which will be necessary for solving large sparse linear systems in the optimization algorithm for large problems. In this paper we focus on fast methods to solve the linear systems arising in the optimization algorithm. We will discuss the algorithm in more detail later. It turns out that only a small number of nonlinear iterations are necessary to converge; see [19, 20]. In most of these nonlinear steps the convergence of the iterative solver is very rapid. However, for some problems in a few intermediate nonlinear steps the linear solver stagnates or converges very slowly. Hence, these iterations dominate the overall cost of the nonlinear iteration. We will analyze these problems in this paper and show how sparse direct solvers (for small to moderate size problems) and preconditioned iterative solvers can significantly improve the efficiency of the algorithm.

In the remainder of this introduction we describe the applications of our algorithm and alternative approaches, and we give an overview of the algorithm.

Surface parameterization serves a number of important applications, such as three-dimensional (surface) mesh generation [1], anisotropic meshing [2, 11] in cases where an analytic description of the surface does not exist, texture mapping [9, 23, 10], surface reconstruction, multiresolutional analysis [7], formation of ship hulls, generation of clothing patterns [13], and metal forming.

Many other approaches for the parameterization of tesselated surfaces have been proposed [13, 7, 8, 12, 9, 23, 10]. These approaches are restricted in their use by the requirement that the planar domain boundary has to be predefined and/or has to be convex. Moreover, some of these approaches either do not guarantee correctness of the resulting mesh, or they do not preserve the surface metric structures. For a brief overview of these alternative methods we refer to [20].

Our algorithm is based on the observation that for a triangular mesh preserving the size of the angles on each of the faces is sufficient to maintain the surface metric structures up to a global scaling factor. Therefore, the method defines the flattening problem in terms of angles. The algorithm minimizes the relative deformation of the angles in the plane with respect to their counterparts in the three-dimensional surface, while satisfying a set of constraints on the angles that ensure the validity of the flat mesh. In order to account for the 'curvature' at each node in the three-dimensional surface (the angles around an interior node do not sum to 2π), we apply a scaling to the angles in the three-dimensional surface, and we measure the deformation relative to these scaled angles. Our minimization problem is entirely formulated in terms of the angles; the locations of the mesh nodes do not play a role. Note that after the angles have been computed, the two-dimensional mesh is fully determined after fixing the position of one interior node and the length and direction of one edge connected to that node. This formulation does not require the two-dimensional boundary to be predefined and does not place any restrictions on the boundary shape or the surface curvature. At the same time the solution method based on the formulation is provably correct as was shown in [19, 20]. The result of the procedure is a valid two-dimensional mesh, which maintains the original surface connectivity and minimizes the distortion of the mesh angles resulting from the mapping to a plane.

The constrained minimization problem is transformed into a nonlinear system of equations using the Lagrange multiplier formulation. We solve this nonlinear system using a robust implementation of Newton's method. After the two-dimensional mesh is generated a spacing function is defined to account for the (minimum) deformation caused by the flattening. This spacing function can then be used to map the generated two-dimensional mesh back to the three-dimensional surface.

The main advantages of our method are that: (1) The method provides a parameterization for any surface for which a parameterization exists. (2) We proved that for any such surface a solution to the optimization problem exists and that the numerical algorithm converges to a solution. (3) The method guarantees that the resulting parameterization is valid. (4) We compute the boundary of the two-dimensional domain as part of the projection procedure; we do not need to define the boundary in advance. (5) The two-dimensional domain can have any shape; specifically, it does not have to be convex. (6) The robustness of the method is not affected by the input mesh quality due to the use of angles in the formulation. (7) The angle-based optimization avoids the scale problems often associated with working on meshes with different feature sizes, i.e. the tolerances and error measurements do not need to be modified when different mesh element sizes are used.

2. MINIMIZATION PROBLEM AND ALGORITHM

As mentioned in the Introduction, we formulate the mesh flattening problem in terms of the flat mesh angles. The constraints are the necessary and sufficient requirements for a valid two-dimensional mesh, and we minimize the (relative) deviation of the angles from their *optimal* two-dimensional projections. Our procedure for surface parameterization for meshing has three stages.

1. Solve the constrained minimization problem (defined below).
2. Check for intersections of the boundary. If a boundary intersection is found, augment the constraints and solve the augmented minimization problem again (starting with the current solution).
3. Compute the mesh spacing function based on the ratios between the areas of triangles in the three-dimensional surface and their counterparts in the flat surface.

Using the parameterization a surface mesh is generated by first meshing the two-dimensional domain using the spacing function and then mapping the resulting mesh to the three-dimensional surface. The process is illustrated in Figure 1. For the current paper only the constrained minimization problem and its solution are relevant.

We use the following notation to define the objective function to be minimized and the constraints. The index i always indicates faces, the index j indicates angles inside a face, and the index k indicates nodes.

- $f_i, i = 1 \dots P$, are the triangulation faces (either in the flat mesh or in the original mesh, as will be clear from the context).
- $\alpha_i^j, i = 1 \dots P, j = 1, 2, 3$, are the flat mesh angles; the angles in a face are numbered counterclockwise in increasing order. The vector of all angles is denoted α.
- $\beta_i^j, i = 1 \dots P, j = 1, 2, 3$, are the corresponding original mesh angles.
- $N_k, k = 1 \dots M$, are the mesh nodes, where $k = 1 \dots M_{int} (< M)$ indexes the interior nodes.
- $\alpha_i^{j(k)}$ is the angle in face f_i at node N_k.
- $\beta_i^{j(k)}$ is the angle in the original mesh corresponding to $\alpha_i^{j(k)}$.

In the following, we implicitly assume the obvious relations among the indices that derive from the connectivity of the mesh. So when we write $\sum_i \beta_i^{j(k)}$, the index i runs only over those faces that contain node N_k. The sum is therefore over all angles adjacent to node N_k. The objective function (to be minimized) is defined by

$$F(\alpha) = \sum_{i=1}^{P} \sum_{j=1}^{3} (\alpha_i^j - \phi_i^j)^2 w_i^j \tag{1}$$

where ϕ_i^j is the *optimal angle* for α_i^j in the two-dimensional mesh, and the $w_i^j > 0$ are weights. Our standard initial choice for the weights is $w_i^j = (\phi_i^j)^{-2}$. We derive the optimal angles ϕ_i^j from the angles β_i^j by computing a scaling factor per node:

$$\phi_i^j(k) = \begin{cases} \beta_i^{j(k)} \frac{2\pi}{\sum_i \beta_i^{j(k)}}, & N_k \text{ is an interior node,} \\ \beta_i^{j(k)}, & N_k \text{ is a boundary node,} \end{cases} \tag{2}$$

where N_k is the mesh node to which the face f_i is attached at the corner j. Since the input mesh is supposed to be valid, we assume that

$$\beta_i^j \quad \geq \quad \varepsilon_1 > 0, \tag{3}$$

where ε_1 is an arbitrarily small (input) parameter. The following constraints are necessary and sufficient to ensure that the resulting mesh is valid.

(1) $g_{i,j}^{(1)} \equiv \alpha_i^j \geq \varepsilon_2 > 0$, for $i = 1 \dots P, j = 1 \dots 3$, for some $\varepsilon_2 > 0$;

(2) $g_i^{(2)} \equiv \alpha_i^1 + \alpha_i^2 + \alpha_i^3 - \pi = 0$, for $i = 1 \dots P$;

(3) $g_k^{(3)} \equiv \sum_i \alpha_i^{j(k)} - 2\pi = 0$, for N_k: $k = 1 \dots M_{int}$ (interior nodes);

(4) $g_k^{(4)} \equiv \frac{\Pi_i \sin(\alpha_i^{j(k)+1})}{\Pi_i \sin(\alpha_i^{j(k)-1})} - 1 = 0$, for N_k: $k = 1 \dots M_{int}$, where $j(k) + 1 = 1$ if $j(k) = 3$ and $j(k) - 1 = 3$ if $j(k) = 1$. The symbol Π indicates the product of its arguments.

The first two constraints deal with the validity of individual faces. Constraint (1) maintains the orientation of a face (up or down) with respect to the mesh, and (2) ensures that each face is valid. Constraints (3) and (4) are necessary to ensure the topological validity of the mesh, because the connectivity of the mesh is not an explicit constraint. Again for more details we refer to [19].

We solve the constrained minimization problem as follows. As argued in [19], for any valid input to our algorithm a valid planar mesh exists. Since the optimal angles ϕ_i^j are strictly positive and our objective function measures the relative distance between the angles α_i^j and the optimal angles ϕ_i^j, we may prevent the algorithm from making angles too small by increasing the weight in the objective function for those angles rather than explicitly

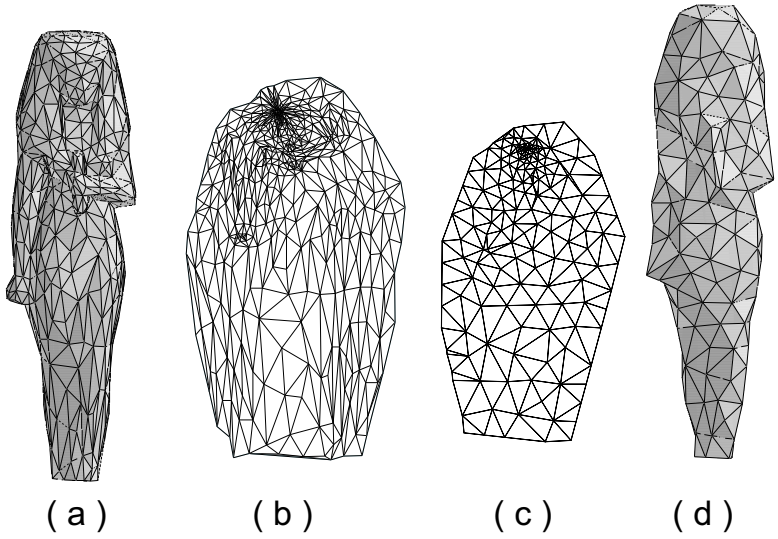

Figure 1: Overview of the different stages in the generation of a new surface mesh for the Isis statue. (a) Original surface mesh, (b) Generated flat mesh, (c) New two-dimensional mesh, (d) New surface mesh

preventing the algorithm from moving outside the feasible set. This corresponds to a change in the norm that measures the distance to our optimal angles. Moreover, because of their simple form, the inequality constraints are very easy to check. Therefore, we formulate the optimization problem without explicitly taking the inequality constraints into account. If the optimization algorithm makes a certain angle too small we reject the iterate, adjust the weight of that angle, and continue. The rationale for this choice is that, in our experience, such cases are extremely rare. So far in only one case, for a rather complicated model, a violation of the inequality constraints has occurred. Moreover, our algorithm converges very quickly if we take the vector of optimal angles, ϕ_i^j, as our initial guess in the Newton iteration. Note that this defines an initial guess that is guaranteed to be inside the feasible set. One alternative approach (followed in many packages) is to first treat the inequality constraint as an equality constraint and then (using possibly additional criteria) move from the boundary of the feasible set to an extremum. In our case, this is always more expensive. Furthermore, one important reason for such an approach is that for some problems it is hard to find a starting point inside the feasible set. Another alternative is to use so-called (logarithmic) barrier methods. Our approach is in fact close to this, but doesn't change the objective function until we actually get to the boundary.

We use a standard Lagrange multiplier formulation for the optimization problem with the equality constraints. The auxiliary objective function then becomes

$$F(\alpha)+\sum_{i=1}^{P}\lambda_i g_i^{(2)}(\alpha)+\sum_{k=1}^{M_{int}}\mu_k g_k^{(3)}(\alpha)+\sum_{k=1}^{M_{int}}\nu_k g_k^{(4)}(\alpha). \tag{4}$$

We use Newton's method to solve for a stationary point of the auxiliary function that satisfies the equality constraints. We modify Newton's method as indicated above to make sure we satisfy the inequality constraints. In most of our examples Newton's method converges in a few iterations and we do not need to adapt the weights.

Each step of Newton's method requires solving a linear system with the Jacobian. For small to moderate size problems we use a sparse direct solver with ordering for sparsity (SuperLU); see [5, 6]. We give some results for problems that were also reported on in [19] to show that using a well-tuned sparse direct solver makes the algorithm very fast. For large problems, however, this will be too expensive and we will have to use preconditioned iterative solvers. We are exploring the application of our algorithm to much larger problems, but in the present paper we use smaller problems to gain experience. From our experiments, see also [20], it turns out that the overall cost of the nonlinear solver is dominated by the cost of the linear solver in a few intermediate nonlinear steps. In these few nonlinear steps the iterative solver converges very poorly, and therefore (see analysis below) a proper choice of preconditioner is essential. In the next section we will derive a very effective preconditioner. The numerical results, which will focus on the *problem Ja-*

cobians from several models, will show that we can get very good convergence.

3. STRUCTURE OF THE MATRICES

To solve the constrained minimization problem

$$\begin{aligned} &\min\ F(\alpha) \ \text{ subject to} \\ &g(\alpha) \equiv [g^{(2)}(\alpha),\ g^{(3)}(\alpha),\ g^{(4)}(\alpha)]^T = 0, \end{aligned} \quad (5)$$

we search for a critical point of the Lagrangian

$$\mathcal{L}(\alpha, \lambda) \equiv F(\alpha) \ + \ \lambda^T \, g(\alpha) \quad (6)$$

using Newton's method, where λ is the vector that contains the Lagrange multipliers, λ_i, μ_k, and ν_k.

In step $l = 1, 2, \ldots$, of the Newton iteration we have to solve a linear system with coefficient matrix

$$\mathcal{M}_l \equiv \begin{bmatrix} \nabla_\alpha^2 [F + \lambda^T g] & J_g^T \\ J_g & 0 \end{bmatrix}, \quad (7)$$

where J_g denotes the Jacobi matrix of g with respect to α, and the functions in the right hand side of (7) are evaluated at the current Newton iterate, l. It is easy to see that

$$\nabla_\alpha^2 F \ = \ 2\,\mathrm{diag}(w_i^j) \ \equiv \ A. \quad (8)$$

The (1,1)-block of $\mathcal{M}_l$ also contains the term $\nabla_\alpha^2 \lambda^T g$. The constraints $g^{(2)}$ and $g^{(3)}$ are linear in α, so their second derivatives with respect to α are zero. The only contribution therefore is $\nabla_\alpha^2 \lambda^T g^{(4)} \ \equiv \ S_l$, which changes in every Newton step.

The (1,2)-block J_g^T consists of three separate blocks, each corresponding to one of the three constraints. Since $g^{(2)}$ and $g^{(3)}$ are linear, their derivatives with respect to α are constant, hence they do not change during the Newton iteration. The most efficient ordering in the matrix $\mathcal{M}_l$ is to consider the block corresponding to $g^{(2)}$ first. When the α's are ordered by faces, i.e.

$$[\alpha_1^1,\ \alpha_1^2,\ \alpha_1^3,\ \alpha_2^1,\ \alpha_2^2,\ \alpha_2^3,\ \ldots]^T,$$

the first block corresponding to the constraint $g^{(2)}$ is simply

$$B^T \equiv \begin{bmatrix} 1 & 1 & 1 & & & & \\ & & & 1 & 1 & 1 & \\ & & & & & & \ldots \end{bmatrix}^T. \quad (9)$$

With the ordering of the α's fixed in this way, the second and third block in J_g^T do not have a simple structure. The reason is that these blocks have nonzero entries only for the interior nodes. We group these blocks together and denote them by C_l^T. Again the subscript l is used to indicate that the block corresponding to $g^{(4)}$ changes in every step.

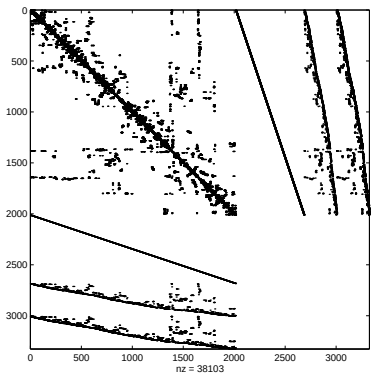

Figure 2: Structure of an example matrix $\mathcal{M}_l$ derived from the *full cat* model.

In general, if our model has P faces, then the matrix we have to solve for in each Newton step has the form

$$\mathcal{M}_l \equiv \begin{bmatrix} A + S_l & B^T & C_l^T \\ B & 0 & 0 \\ C_l & 0 & 0 \end{bmatrix}, \quad (10)$$

where $A + S_l \in \mathbf{R}^{3P \times 3P}$, $B^T \in \mathbf{R}^{3P \times P}$, and C_l^T of size $3P$ times twice the number of interior nodes, which is approximately equal to P. An example matrix for a model with $P = 671$ faces is shown in Figure 2. The (1,1)-block, i.e. the matrix $A + S_l$, is of order 2013, and the matrix B^T has size 2013-by-671. The block C_l^T has size 2013-by-642. The complete example matrix is of order 3326.

4. EFFICIENT SOLUTION OF SYSTEMS WITH $\mathcal{M}_l$

One way to solve a system with $\mathcal{M}_l$ is to use a sparse direct solver. As demonstrated by our numerical examples with the SuperLU package [5, 6], this is generally very efficient for smaller models (see Section 5).

Larger models require the use of preconditioned iterative methods. In [20] we reported on numerical results using the iterative solvers GMRES [18] and BiCGStab [21] with ILUT as preconditioner [16]. While this approach gave satisfactory results in most cases, for some problems the iterative solver stagnates in solving a few intermediate Jacobians. The reason for this stagnation is

that the matrices are indefinite and ill-conditioned, and hence convergence is extremely slow. Here we are concerned with exploiting the structure of the matrix $\mathcal{M}_l$ to construct preconditioners tailored to our needs. Although the preconditioners we will derive do not remove the indefiniteness, they tend to move all the eigenvalues close to the values 1 and $(1 \pm \sqrt{5})/2$. This removes the ill-conditioning and since all eigenvalues of the preconditioned matrix are close to these 3 values, convergence is rapid (see the Numerical Results section).

4.1 The Murphy-Golub-Wathen preconditioner

Several general purpose preconditioners for symmetric indefinite matrices of the form

$$\begin{bmatrix} M_1 & M_2^T \\ M_2 & 0 \end{bmatrix} \tag{11}$$

have been proposed in the literature. One of the most promising approaches was developed by Murphy, Golub and Wathen [14]: They show that multiplying the matrix (11) from the left (or right) by

$$\begin{bmatrix} M_1^{-1} & 0 \\ 0 & (M_2 M_1^{-1} M_2^T)^{-1} \end{bmatrix}, \tag{12}$$

results in a diagonalizable matrix with at most four distinct eigenvalues, namely $0, 1, (1 \pm \sqrt{5})/2$. The matrix (12) will be called the *exact MGW preconditioner* for matrix (11). Under the assumption that the preconditioned matrix is nonsingular, i.e. that there is no zero eigenvalue, this implies that any Krylov subspace method with an optimality or Galerkin type property, for example GMRES [18], will converge in at most three steps. Of course, the efficiency of this preconditioning technique crucially depends on an efficient solution of systems with the matrix M_1.

In case of our matrix $\mathcal{M}_l$, the role of M_1 in this approach could be played either by $A + S_l$, or by

$$\mathcal{A}_l \equiv \begin{bmatrix} A + S_l & B^T \\ B & 0 \end{bmatrix}. \tag{13}$$

Let us first consider the case that $M_1 = \mathcal{A}_l$. Each multiplication of the preconditioner (12) requires solving for $\mathcal{A}_l$ twice. Since GMRES will take three iterations and we need to precondition the right hand side, *we need to solve for $\mathcal{A}_l$ eight times*. In addition, we need four solves for the (2,2)-block of (12), which may be expensive as well. In our case $\mathcal{A}_l$ accounts for about 80% of the system matrix $\mathcal{M}_l$, so this approach will only be useful if $\mathcal{A}_l$ is of very special form.

Since the matrix $\mathcal{A}_l$ also is of the form (11), we actually can use the same preconditioning idea twice. The resulting method would use the preconditioner

$$\mathcal{P}_l^{(1)} \equiv \begin{bmatrix} (A + S_l)^{-1} & 0 \\ 0 & (B(A + S_l)^{-1} B^T)^{-1} \end{bmatrix}, \tag{14}$$

for the system with $\mathcal{A}_l$, and the preconditioner

$$\mathcal{P}_l^{(2)} \equiv \begin{bmatrix} \mathcal{A}_l^{-1} & 0 \\ 0 & ([C_l\ 0]\, \mathcal{A}_l^{-1}\, [C_l\ 0]^T)^{-1} \end{bmatrix}. \tag{15}$$

for the system with $\mathcal{M}_l$.

Solving with GMRES the preconditioned system with the matrix $\mathcal{P}_l^{(2)} \mathcal{M}_l$ requires four multiplications by $\mathcal{P}_l^{(2)}$ (again, one to precondition the right hand side, and three for the GMRES steps). Each of these multiplications requires solving a system with the $4P$-by-$4P$ matrix $\mathcal{A}_l$ and with the (approximately) P-by-P matrix $[C_l\ 0]\, \mathcal{A}_l^{-1}\, [C_l\ 0]^T$. The latter solve depends on the unknown matrix $\mathcal{A}_l^{-1}$, and can therefore only be performed iteratively. Next note that each multiplication with the preconditioner (14) requires one solve for a $3P$-by-$3P$ system with $A + S_l$ and one solve for an P-by-P system with $B(A + S_l)^{-1} B^T$. Again, the latter solves can only be performed iteratively since $(A+S_l)^{-1}$ is unknown. Altogether, solving for $\mathcal{P}_l^{(2)} \mathcal{M}_l$ with GMRES this approach requires *solving 64 systems with the matrix $A+S_l$*. Since this matrix is of order $3P$, which is about 60% of the order of $\mathcal{M}_l$, we cannot expect dramatic savings when using this preconditioner either.

The approach we propose in the following section consists of computing an approximation of the preconditioner (15) only. Specifically, we will use an approximation to the crucial block $A + S_l$.

4.2 The approximate Murphy-Golub-Wathen preconditioner

To explain our approximation of the preconditioner (15), we denote

$$\mathcal{M}_l \equiv \begin{bmatrix} \mathcal{A}_l & \mathcal{C}_l^T \\ \mathcal{C}_l & 0 \end{bmatrix}, \quad \mathcal{C}_l \equiv [C_l\ 0], \tag{16}$$

and additively split $\mathcal{A}_l$ into

$$\mathcal{A}_l = \begin{bmatrix} A & B^T \\ B & 0 \end{bmatrix} + \begin{bmatrix} S_l & 0 \\ 0 & 0 \end{bmatrix} \equiv \mathcal{A} + \mathcal{S}_l. \tag{17}$$

Then the inverse of $\mathcal{A}$ can be computed explicitly, and is given by

$$\mathcal{A}^{-1} \equiv \begin{bmatrix} (I_{3P} - Q)A^{-1} & \frac{1}{3} Q B^T \\ \frac{1}{3} B Q^T & -(BA^{-1}B^T)^{-1} \end{bmatrix}, \tag{18}$$

where $Q \equiv A^{-1}B^T(BA^{-1}B^T)^{-1}B$ is a projection, and we have used that $BB^T = 3I_P$. Note that

$$A^{-1} \;=\; \frac{1}{2}\,\mathrm{diag}(1/w_i^j), \text{ and}$$

$$BA^{-1}B^T \;=\; \frac{1}{2}\,\mathrm{diag}(1/w_i^1 + 1/w_i^2 + 1/w_i^3).$$

Thus, the (2,2)-block of $\mathcal{A}^{-1}$ is diagonal, and the (1,2)- and (2,1)-blocks have the same structure as B^T and B, respectively. In the (1,1)-block, the matrix $B^T(BA^{-1}B^T)^{-1}B$ is block-diagonal with 3-by-3 blocks, and hence the whole block has this structure. Altogether, $\mathcal{A}^{-1}$ has approximately $7P$ more nonzero entries than $\mathcal{A}$.

We now define the preconditioner

$$\begin{aligned}\mathcal{P}_l \;&\equiv\; \left[\begin{array}{cc} \mathcal{A}^{-1} & 0 \\ 0 & (\mathcal{C}_l\mathcal{A}^{-1}\mathcal{C}_l^T)^{-1} \end{array}\right] \\ &=\; \left[\begin{array}{cc} \mathcal{A}^{-1} & 0 \\ 0 & (C_l(I_{3P} - Q)A^{-1}C_l^T)^{-1} \end{array}\right], \end{aligned} \tag{19}$$

which for left preconditioning yields the system matrix

$$\mathcal{P}_l\,\mathcal{M}_l = \left[\begin{array}{cc} I_{4P} + \mathcal{A}^{-1}\mathcal{S}_l & \mathcal{A}^{-1}\mathcal{C}_l^T \\ (\mathcal{C}_l\mathcal{A}^{-1}\mathcal{C}_l^T)^{-1}\mathcal{C}_l & 0 \end{array}\right]. \tag{20}$$

Note that

$$\mathcal{M}_l\,\mathcal{P}_l = (\mathcal{P}_l\,\mathcal{M}_l)^T \tag{21}$$

since $\mathcal{M}_l$ and $\mathcal{P}_l$ are both symmetric. Since $\mathcal{A}^{-1}$ is known explicitly, the main work in applying the preconditioner (19) lies in solving a system with the (approximately) P-by-P matrix $\mathcal{C}_l\mathcal{A}^{-1}\mathcal{C}_l^T$. Thus the computation of the preconditioner (19) itself is considerably cheaper than the computation of the exact MGW preconditioner (cf. Section 4.1). In particular, the (1,1)-block $\mathcal{A}^{-1}$ in (19), which is of order $4P$, only has to be computed once and then stays constant during the entire Newton iteration. Moreover, in contrast to (14)–(15), the (2,2)-block $\mathcal{C}_l\mathcal{A}^{-1}\mathcal{C}_l^T$ is known explicitly, and hence can be solved by a direct or by an iterative method. We will refer to the preconditioner $\mathcal{P}_l$ (19) while explicitly computing or multiplying by $(\mathcal{C}_l\mathcal{A}^{-1}\mathcal{C}_l^T)^{-1}$ as the direct approximate MGW (DAMGW) preconditioner, and to the same preconditioner using an iterative method to multiply by $(\mathcal{C}_l\mathcal{A}^{-1}\mathcal{C}_l^T)^{-1}$ as the iterative approximate MGW (IAMGW) preconditioner.

For the DAMGW preconditioner we compute the LU-factorization of $\mathcal{C}_l\mathcal{A}^{-1}\mathcal{C}_l^T$ once for each Newton step l. We then solve either the left or the right preconditioned system with GMRES. Each step of this method requires one matrix-vector multiplication with the preconditioner (19), and thus one solve for (the L and the U factor of) $\mathcal{C}_l\mathcal{A}^{-1}\mathcal{C}_l^T$.

Even though using a direct solver to compute $\mathcal{C}_l\mathcal{A}^{-1}\mathcal{C}_l^T$ in the preconditioner is significantly cheaper than the MGW preconditioner, it will still become too expensive for really large problems. However, there is another possibility. Multiplication of a vector by $(\mathcal{C}_l\mathcal{A}^{-1}\mathcal{C}_l^T)^{-1}$ is the same as solving a linear system for $\mathcal{C}_l\mathcal{A}^{-1}\mathcal{C}_l^T$ with that vector as the right hand side. We can do this approximately using an iterative solver. In the IAMGW preconditioner we use (restarted) GMRES to approximate the multiplication by $(\mathcal{C}_l\mathcal{A}^{-1}\mathcal{C}_l^T)^{-1}$. If we solve to a very high tolerance the effect is the same as multiplying with the inverse; however, we also have the possibility to solve to low accuracy. This will reduce the cost of applying the preconditioner, but the convergence may be slower. Solving to low accuracy leads to so-called variable preconditioning; since the projection on the Krylov space in different iterations involves a different operator the preconditioner is different in every iteration. This means that the generated vectors generally do not span a Krylov space. However, if we precondition from the right we nevertheless get the image of some (approximate Krylov) subspace under multiplication by the linear operator, and we can minimize the residual over this subspace [22, 3, 15]. We use the Flexible GMRES (FGMRES) implementation outlined in [17]

Note that we have approximated $\mathcal{A}_l^{-1}$ in (15) by $\mathcal{A}^{-1}$. Hence the result of [14] does not guarantee convergence of GMRES for $\mathcal{P}_l\mathcal{M}_l$ or $\mathcal{M}_l\mathcal{P}_l$ in three steps; we no longer have exactly three distinct eigenvalues. A complete analysis of the properties of the preconditioned system is beyond the scope of this paper, but will be published elsewhere. However, for effective preconditioners one would expect the eigenvalues to be very close to the eigenvalues of the matrix $\mathcal{M}_l$ preconditioned by the exact MGW preconditioner. This should improve convergence significantly. We give the eigenvalues of one unpreconditioned Jacobian, $\mathcal{M}_l$, (from *half rabbit*) in Figure 3; the eigenvalues of the same Jacobian with preconditioner, $\mathcal{P}_l\mathcal{M}_l$, are given in Figure 4. Note left and right preconditioning yield the same eigenvalues. We see that the eigenvalues are indeed clustered around 1 and $(1\pm\sqrt{5})/2$, the three eigenvalues of the matrix $\mathcal{M}_l$ preconditioned by the exact MGW preconditioner. The effect of this change in the spectrum on the convergence can be seen in Figures 3-4.

5. NUMERICAL RESULTS

We first show a few results using the sparse direct solver (SuperLU) with ordering for sparsity. For comparison we also show the timings of a straightforward direct solution (from previous paper [19]). First, we note that the total runtime is drastically reduced by using a special reordering for sparsity. Second, we note that the time spent in

Model	Nr. of faces	Nr. Newton iterations	Runtime (s) standard	Runtime (s) reordered sparse
Cat head	257	4	36	7 (1.9)
Half rabbit	380	10	362	29 (1.87)
Isis statue	879	5	1355	222 (16.87)

Table 1: Comparison of total solution time for sparse direct solver without and with reordering for sparsity. The numbers in brackets in the *reordered sparse* column give the cumulative time for factorization of the Jacobian and the actual linear solve over all nonlinear iterations. The relatively large number of Newton iterations for *half rabbit* is caused by 4 additional iterations after the first solution had a nonlocal boundary intersection.

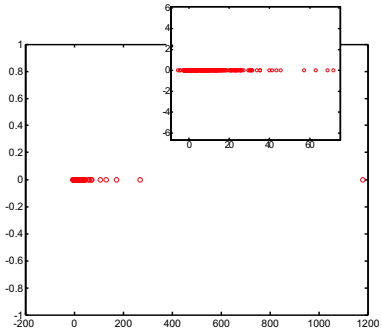

Figure 3: Eigenvalues of the unpreconditioned Jacobian, $\mathcal{M}_l$, from *half rabbit*. The upper right zoomed in box gives the eigenvalues near the origin.

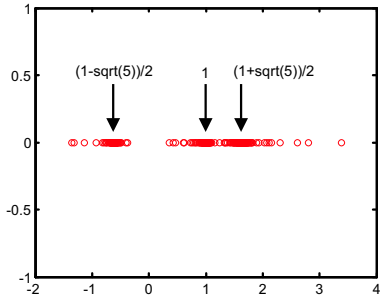

Figure 4: Eigenvalues of the preconditioned Jacobian, $\mathcal{P}_l\mathcal{M}_l$ from *half rabbit*.

the linear solver is now a small fraction of the overall solution time, especially for the larger two problems. The reason is the inefficient, but straightforward, computation of the Jacobian in the current implementation. In the old version this was not a bottleneck because of the high cost of the (standard) direct linear solver. Removing this overhead will be an obvious future improvement.

For the iterative solver we will show convergence results for each problem for one of the intermediate Jacobians, $\mathcal{M}_l$, for which convergence without a special preconditioner is problematic. We give results for three problems, *full cat*, *half rabbit*, and *Isis statue*. The three models are given in Figure 5. For comparison we will give the following convergence curves

- unpreconditioned GMRES,
- GMRES with the DAMGW preconditioner as left preconditioner,
- GMRES with the IAMGW preconditioner as right preconditioner solving to high accuracy (similar to DAMGW as right preconditioner),
- GMRES with the IAMGW preconditioner as right preconditioner solving to low accuracy (much cheaper),

We do not give the convergence for the DAMGW preconditioner as right preconditioner, because the convergence is virtually the same as for accurate IAMGW as right preconditioner. To keep the analysis simple we only use full GMRES to solve for the matrix $\mathcal{M}_l$ with some preconditioner. Note that in the low accuracy version of the IAMGW preconditioner we use restarted GMRES to solve for the $(2,2)$ block of the preconditioner, $\mathcal{C}_l\mathcal{A}^{-1}\mathcal{C}_l^T$. In practice, for large problems we will not use full GMRES because of its high cost in memory and CPU time. Most likely we will use a restarted or truncated version [4]. The high accuracy IAMGW right preconditioner uses full GMRES to solve the linear system with $\mathcal{C}_l\mathcal{A}^{-1}\mathcal{C}_l^T$, and we iterate till method reduces the (initial) residual by a factor of approximately 10^{-12}. The the low accuracy IAMGW right preconditioner uses restarted GMRES(50) (at most 5 cycles) and we iterate till the method reduces the (initial) residual norm by 10^{-2}.

In Figure 6 we give the convergence curves for the *full*

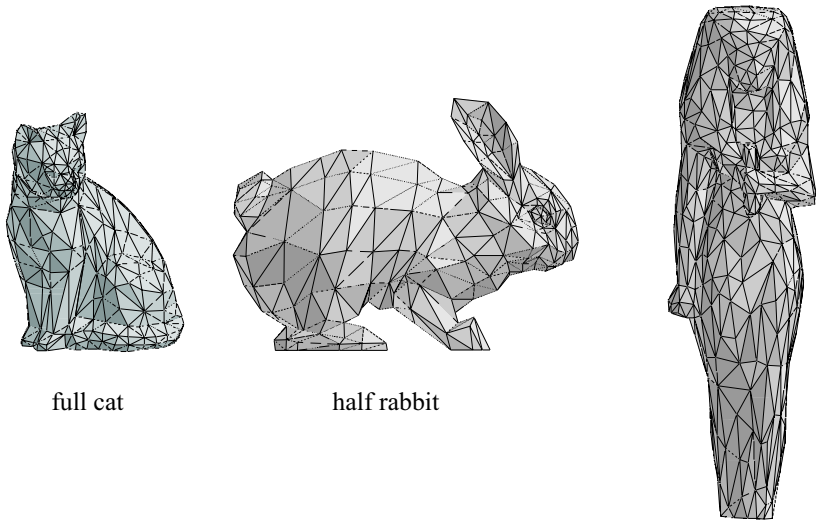

Figure 5: Our three model problems.

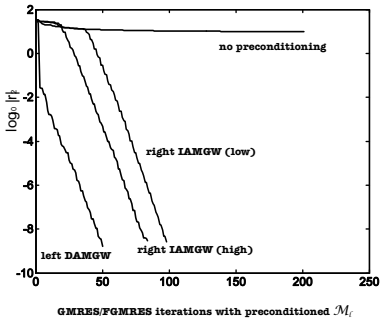

Figure 6: GMRES convergence curves for one Jacobian derived from the *full cat* model.

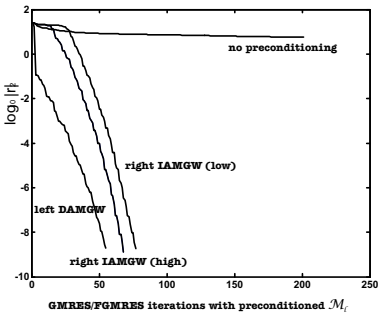

Figure 7: GMRES convergence curves for one Jacobian derived from the *half rabbit* model.

cat, opened at the base; see Figure 5. In Figure 7 we give the convergence curves for the *half rabbit*; see Figure 5. Finally, in Figure 8 we give the convergence curves for the *Isis statue*, opened at the back (seam); see Figure 5.

The results show that the preconditioned versions are much better than unpreconditioned versions. We also see that left preconditioning (for this problem) is more efficient than right preconditioning, especially at the start and (hence) for a large (inaccurate) convergence tolerance. For a small (accurate) tolerance the differences in numbers of iterations between left and right preconditioning are not very large. We also note that for two

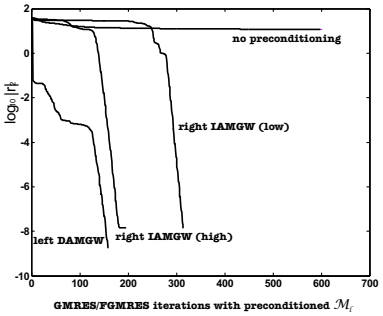

Figure 8: GMRES convergence curves for one Jacobian derived from the *Isis statue* model.

problems the convergence curves using a very large tolerance in the iterative preconditioner stay very close to the ones with very small tolerance. This is less the case for the convergence of the problems derived from the Isis statue. Nevertheless, even there the large tolerance preconditioner is much cheaper in time (not given); the same obviously holds for the other two problems.

6. CONCLUSIONS

We have derived a preconditioner that approximates the exact MGW preconditioner but is much cheaper to compute and use. There are two ways to use the exact MGW preconditioner. As argued at the end of Section 4.1, for our problems these are not cost effective. Therefore, we have derived two alternative approximate MGW preconditioners. We showed that the eigenvalues of the preconditioned matrix, using our preconditioner, form tight clusters around the eigenvalues of the matrix preconditioned by the exact MGW. This has a significant impact on the convergence rate, as could be seen in the numerical examples. We realize that a full evaluation of the usefulness of our preconditioners for this application requires run times in addition to convergence curves. Unfortunately, at the deadline for this paper those timings were not yet available. However, the convergence results suggest that with these preconditioners the method proposed by Sheffer and de Sturler [19] is viable for large problems.

In addition, our timings demonstrate that a sparse direct solver is very effective for small to moderately sized problems.

ACKNOWLEDGEMENTS

This work was supported, in part, by Sandia National Laboratories under Contract SNL 16806, the Center for Simulation of Advanced Rockets (DOE LLNL B341494), and the Center for Process Simulation and Design (NSF DMS 98-73945). The work of the first author was also supported by an Emmy Noether stipend of the Deutsche Forschungsgemeinschaft.

References

[1] M. Bern and D. Eppstein. Quadrilateral meshing by circle packing. *6th International Meshing Roundtable*, pages 7–19, 1997.

[2] H. Borouchaki and P. J. Frey. Adaptive triangular-quadrilateral mesh generation. *International Journal of Numerical Methods in Engineering*, 41:915–934, 1998.

[3] E. De Sturler. Nested Krylov methods based on GCR. *J. Comput. Appl. Math.*, 67:15–41, 1996.

[4] E. De Sturler. Truncation strategies for optimal Krylov subspace methods. *SIAM J. Numer. Anal.*, 36:864–889, 1999. electronically available from http://epubs.siam.org.

[5] J. W. Demmel, Stanley C. Eisenstat, John R. Gilbert, Xiaoye S. Li, and Joseph W. H. Liu. A supernodal approach to sparse partial pivoting. *SIAM J. Matrix Anal. Appl.*, 20(3):720–755, 1999.

[6] James W. Demmel, John R. Gilbert, and Xiaoye S. Li. SuperLU user's guide, 1999. available from http://www.nersc.gov/ xiaoye/SuperLU/index.html.

[7] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. Multiresolution analysis of arbitrary meshes. *Computer Graphics (Annual Conference Series, 1995. SIGGRAPH '95)*, pages 173–182, 1995.

[8] M. S. Floater. Parametrization and smooth approximation of surface triangulations. *Computer Aided Geometric Design*, 14:231–250, 1997.

[9] S. Haker, S. Angenent, A. Tannenbaum, R. Kikinis, G. Sapiro, and M. Halle. Conformal surface parameterization for texture mapping. *IEEE Transactions on Visualization and Computer Graphics*, 6 (2):181–189, 2000.

[10] B. Levi and J.L. Mallet. Non-distorted texture mapping for sheared triangulated meshes. *Proc. SIGGRAPH 1998*, pages 343–352, 1998.

[11] X. Y. Li, S. H. Teng, and A. Üngör. Biting ellipses to generate anisotropic mesh. *8th International Meshing Roundtable*, pages 97–108, 1999.

[12] D. L. Marcum and J. A. Gaiter. Unstructured surface grid generation using global mapping and physical space approximation. *8th International Meshing Roundtable*, pages 397–406, 1999.

[13] J. McCartney, B. K. Hinds, and B. L. Seow. The flattening of triangulated surfaces incorporating darts and gussets. *Computer-Aided Design (CAD)*, 31:249–260, 1999.

[14] Malcolm F. Murphy, Gene H. Golub, and Andrew J. Wathen. A note on preconditioning for indefinite linear systems. *SIAM J. Sci. Comput.*, 21(6):1969–1972 (electronic), 2000.

[15] Y. Saad. A flexible inner-outer preconditioned GMRES algorithm. *SIAM J. Sci. Statist. Comput.*, 14:461–469, 1993.

[16] Y. Saad. Ilut: a dual threshold incomplete ilu factorization. *Numerical Linear Algebra and Applications*, 1:387–402, 1994.

[17] Y. Saad. *Iterative Methods for Sparse Linear Systems*. PWS Publishing, Boston, 1996.

[18] Youssef Saad and Martin H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 7:856–869, 1986.

[19] A. Sheffer and E. de Sturler. Surface parameterization for meshing by triangulation flattening. *Proc. 9th International Meshing Roundtable*, pages 161–172, 2000.

[20] Alla Sheffer and Eric de Sturler. Parameterization of faceted surfaces for meshing using angle based flattening. *Engineering with Computers (Springer)*. accepted.

[21] H. A. Van der Vorst. BI-CGSTAB: A fast and smoothly converging variant of BI-CG for the solution of nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 13:631–644, 1992.

[22] H. A. Van der Vorst and C. Vuik. GMRESR: A family of nested GMRES methods. *Num. Lin. Alg. with Appl.*, 1:369–386, 1994.

[23] G. Zigelman, R. Kimmel, and N. Kiryati. Texture mapping using surface flattening via multi-dimensional scaling. *CIS report CIS-2000-01, submitted to IEEE Trans. on Visualization and Computer Graphics*, 2000.