2016

Interactively Cutting and Constraining Vertices in Meshes Using Augmented Matrices

Yu-Hong Yeung
Jessica Crouch
Old Dominion University
Alex Pothen

Follow this and additional works at: https://digitalcommons.odu.edu/computerscience_fac_pubs

Part of the Software Engineering Commons

Repository Citation
Yeung, Yu-Hong; Crouch, Jessica; and Pothen, Alex, "Interactively Cutting and Constraining Vertices in Meshes Using Augmented Matrices" (2016). Computer Science Faculty Publications. 97.
https://digitalcommons.odu.edu/computerscience_fac_pubs/97

Original Publication Citation
We present a finite-element solution method that is well suited for interactive simulations of cutting meshes in the regime of linear elastic models. Our approach features fast updates to the solution of the stiffness system of equations to account for real-time changes in mesh connectivity and boundary conditions. Updates are accomplished by augmenting the stiffness matrix to keep it consistent with changes to the underlying model, without refactoring the matrix at each step of cutting. The initial stiffness matrix and its Cholesky factors are used to implicitly form and solve a Schur complement system using an iterative solver. As changes accumulate over many simulation timesteps, the augmented solution method slows down due to the size of the augmented matrix. However, by periodically refactoring the stiffness matrix in a concurrent background process, fresh Cholesky factors that incorporate recent model changes can replace the initial factors. This controls the size of the augmented matrices and provides a way to maintain a fast solution rate as the number of changes to a model grows. We exploit sparsity in the stiffness matrix, the right-hand-side vectors and the solution vectors to compute the solutions fast, and show that the time complexity of the update steps is bounded linearly by the size of the augmented matrices. However, by periodically refactoring the stiffness matrix in a concurrent background process, fresh Cholesky factors that incorporate recent model changes can replace the initial factors. This controls the size of the augmented matrices and provides a way to maintain a fast solution rate as the number of changes to a model grows. We exploit sparsity in the stiffness matrix, the right-hand-side vectors and the solution vectors to compute the solutions fast, and show that the time complexity of the update steps is bounded linearly by the size of the Cholesky factor of the initial matrix. Our complexity analysis and experimental results demonstrate that this approach scales well with problem size. Results for cutting and deformation of 3D linear elastic models are reported for meshes representing the brain, eye, and model problems with element counts up to 167,000; these show the potential of this method for real-time interactivity. An application to limbal incisions for surgical correction of astigmatism, for which linear elastic models and small deformations are sufficient, is included.

We acknowledge support from DOE grant no. 13SC-003242, NSF grants no. CCF-1218916 and no. CCF-1552323, and the Purdue Research Foundation. Authors’ addresses: Y.-H. Yeung and A. Pothen, Department of Computer Science, Purdue University, West Lafayette, IN 47907; emails: {yyeung, apothen}@cs.purdue.edu; J. Crouch, Department of Computer Science, Old Dominion University, Norfolk, VA 23529; email: jcroch@cs.odu.edu. Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org. © 2016 ACM 0730-0301/2016/02-ART18 $15.00 DOI: http://dx.doi.org/10.1145/2856317


1. INTRODUCTION

A method to support interactively cutting or deforming solid finite element models by solving the time-varying equations quickly is presented in this article. Topological mesh modifications and boundary condition changes are essential parts of many simulation scenarios, particularly surgical simulations. Integrating support for cutting with real-time finite-element solution methods is a computational challenge: first, because graphic and haptic rendering requires demanding update rates and, second, because connectivity changes due to cutting necessitate corresponding changes to the underlying matrix equations. Such changes invalidate previous factorizations or inverse computations for the stiffness matrix, requiring either computationally expensive update procedures or solution via an iterative method.

Many simulations that involve cutting would ideally support unpredictable cutting paths. Enabling unpredictable cutting can require that the internal deformation of a solid model be computed and tracked so that accurate cut surfaces are exposed as cuts progress into a model’s potentially inhomogeneous interior. Thus, a desirable solution method would quickly compute the displacement of all nodes in a 3D mesh while accommodating changes to the mesh and equations due to cutting, variable pushing and pulling forces, and changes to the fixed displacements (Dirichlet boundary conditions) created by different fixation scenarios.

Our solution approach is to represent the changing mesh of a linear elastic model with an augmented 2-by-2 block matrix in which the (1,1) block is fixed, the (2,2) block is zero, and the other blocks vary. We then use an implicit solution approach to
the Schur complement system, in which we exploit the sparsity of the matrices involved. Our current solution approach combines a matrix-factorization-based method for the (1,1) block, with a Krylov space-based iterative solver for the Schur complement. A detailed treatment of the algorithm’s complexity shows that performance scales well with model size while supporting arbitrary cutting of any valid finite-element mesh. Periodic refactorizations of the stiffness matrix are performed concurrently with the real-time solution loop so that changes to the model eventually become directly incorporated into the nonaugmented stiffness matrix factors. This allows the list of nodes affected by accumulated changes to be periodically reduced, limiting the growth in solution time that can occur as the list of mesh modifications grows longer over time.

A variety of existing algorithms for mesh generation [Goksel and Salcudean 2011; Mohamed and Davatzikos 2004; Lederman et al. 2010], collision detection [Teschner et al. 2005; Spillmann and Harders 2012; Zhang and Kim 2012], and mesh refinement [Steinemann et al. 2006; Mor and Kanade 2000; Forest et al. 2002] are available. Such algorithms that work for finite-element models can be paired with our solution algorithm to construct a simulation platform. Thus, the scope of this article does not include algorithms for simulation tasks other than solving the finite-element system of equations. In addition, although only node cutting is demonstrated in this article, many cutting and remeshing algorithms cut edges and surfaces; these can also be solved using our method. A feature of the solution algorithm presented is its flexibility to work with structured and unstructured meshes as well as a number of different methods for adapting mesh geometry to respect a cut surface.

Our results show that the augmented approach works well on linear models exhibiting small deformations. This is valuable for the important subset of medical applications that involve small-magnitude, but medically significant, deformations. For these applications, linear elasticity can be an appropriate material model because it realistically simulates deformation at a lower computational cost than more complex models. A variety of simulation results have been published based on linear elastic models in both the computer graphics and biomedical engineering literatures, in many cases with validation of model accuracy against empirical data derived from medical images or mechanical experiments. Examples include models of the lens [Mikielewicz et al. 2013] and cornea [Gefen et al. 2009] of the eye, trabecular bone [Jahya et al. 2014], prostate biopsy [Crouch et al. 2007] and prostate brachytherapy [Keveny et al. 1994; Andreasson et al. 2014; Juszczyk et al. 2011]. We recognize that linear elasticity will not adequately model these organs and tissue types under all loading scenarios, but the community has found the linear elastic model to be useful for biomechanical modeling when limited forces are applied. Nonlinear models are not considered in this article, but will be investigated in the future.

1.1 Our Contributions

The three main contributions of this work are:

—A unified augmented matrix formulation of a finite element model that allows both continuous, unpredictable cutting, and imposition of new boundary conditions. This formulation keeps the original stiffness matrix as a submatrix to eliminate the necessity of refactorization at each timestep.

—A hybrid solution approach that utilizes a direct solver and an iterative solver. The solution of the updated portion of a mesh is decoupled from the solution of the unchanged portion, facilitating fast updates when the percentage of mesh elements affected by topological changes is small. Preconditioning techniques for the iterative part of the solution method are also discussed.

—Acceleration of the solution algorithms by exploiting sparsity in both the matrices and the vectors. A complexity analysis is presented using graph theory concepts applied to the accelerated solution method.

1.2 Article Organization

This article is organized as follows. Section 2 reviews previous work on the real-time solution of physics-based models and finite-element equations. Section 3 presents our new augmented method for assembling a finite-element system of equations and accounting for changes in mesh connectivity and boundary conditions via updates to stiffness matrix factors. Section 4 presents speed and accuracy results from finite-element deformation and cutting experiments with models of various sizes. Section 5 discusses conclusions and directions for future work.

2. PREVIOUS WORK

Beginning with Terzopoulos et al. [1987] and Terzopoulos and Fleischer [1988], physics-based deformable models have been used for animation and simulation. By the mid-1990s, a variety of work specific to surgery simulation began to appear [Cover et al. 1993; Bro-Nielsen and Cotin 1996]. This section reviews existing approaches for computing physics-based deformation solutions, with a particular focus on methods that involve finite-element analysis and cutting. Methods are categorized according to whether they use a direct-solution approach with precomputation, an iterative solver, or a combination of both.

2.1 Precomputation Approaches

Precomputation strategies accelerate the solution step of a simulation by shifting the bulk of the computational burden to a preprocessing stage. The bottleneck in a finite-element simulation is the solution of a system of linear equations, $Ka = f$, where $K$ is the stiffness matrix, $a$ is a vector of nodal displacements, and $f$ is a vector of nodal forces. Precomputation methods such as Zhong et al. [2005] minimize the time required to calculate the solution vector by inverting $K$ before a simulation begins so that $a$ can be directly computed via the multiplication $a = K^{-1}f$ during the simulation. Since $K^{-1}$ is dense, condensation methods such as Bro-Nielsen and Cotin [1996], Bro-Nielsen [1998], Berkley et al. [2004], and Lee et al. [2010] further reduce computation time by producing from the full inverse matrix a smaller dimension one that contains only the equations necessary to compute a solution for a small subset of the nodes, such as a set of surface nodes. The inability to compute a solution for nodes not included in the preselected subset poses a problem for applications that involve cutting. The Sherman-Morrison-Woodbury update formula [Hager 1989] has been used to address this by allowing selected degrees of freedom to be added back into a condensed stiffness matrix as they are needed. This approach was suggested by James and Pai [1999], and later was used in needle-insertion simulation [DiMaio and Salcudean 2002; DiMaio 2003], and in a cutting simulation [Zhong et al. 2005]. The approach is most successful when access to a small number of degrees of freedom needs to be added to an already condensed system. It becomes computationally intensive and slow as the added number of degrees of freedom increases, and is impractical for applications that require cutting with nontrivial remeshing.

A variation on the precomputed inverse approach sharing some similarities with our work is the precomputed stiffness matrix factorization described by Turkiyyah [2011], which updated a Cholesky factorization to accommodate the addition and modification of
discontinuous basis functions along a cutting path. While our work also updates the solution to a system of equations involving the stiffness matrix, it does so without updating the factors by solving a Schur complement system with an iterative solver. Hence, our method supports any local mesh modification, whether from remeshing or addition of basis functions; consequently, our method’s update process substantially differs from Turkiiyah’s work.

Solution techniques that rely on the superposition principle such as Cotin et al. [1999], Picinbono et al. [2002], and Sedef et al. [2006] precompute and record the set of node displacements that result from a constraint being applied to a single node. This computation is repeated for every node that might be subject to a constraint, and all the results are stored. At runtime, nodal displacements are computed as a linear combination of the stored results. For some applications, this approach can closely approximate the ideal solution, but without modification, it cannot handle changes in mesh connectivity.

Precomputed Green’s functions have also been used by Nikitin et al. [2002] and James and Pai [2003] to quickly compute deformation solutions for subsets of mesh nodes. Similarly, the banded matrix method proposed by Berkley et al. [1999] prioritizes and re-arranges the rows and columns of a stiffness matrix based on node type, then factors the stiffness matrix in such a way that a fast update is provided only for the highest-priority nodes. In both cases, solutions for internal nodes are generally not computed.

A limitation generally shared by precomputation approaches is that results produced in the precomputation phase are invalidated when the topology of the mesh changes; thus, cutting and remeshing require special consideration. Constraint removal is a precomputation approach that requires cutting paths to be known a priori. Lindblad and Turkiiyah [2007] and Sela et al. [2007] have demonstrated how duplicate nodes along a cutting path can be constrained to move together until they are cut, at which time the constraint is removed to open up a predefined cut.

Discontinuous basis functions provide a more flexible cutting scheme that has been used in concert with precomputation. This approach was originally introduced in the engineering literature as a way of studying crack formation [Mos et al. 1999] and more recently has been applied to the problem of cutting in surgical simulations [Vigneron et al. 2004; Turkiiyah et al. 2011]. Unpredictable and arbitrary cutting paths are accommodated through the addition of new degrees of freedom that use discontinuous interpolation functions to account for mid-element breaks in nodal influence. The Turkiiyah work has important similarities to our work, in that both approaches progressively update the solution of the stiffness matrix equation. However, an important distinction is that our work maintains a finite-element mesh that respects cut surfaces by remeshing areas as needed, while the Turkiiyah work maintains separate, distinct meshes for graphic rendering and for computation of the finite-element solution. In their method, rendered surfaces are remeshed as needed, but the finite element equations accommodate cutting through the addition of discontinuous basis functions without remeshing. As shown in Lindblad and Turkiiyah [2007], an update procedure similar to the Sherman-Morris-Woodbury update can be employed to update a precomputed inverse stiffness matrix to account for the new degrees of freedom. Because the complexity of the Sherman-Morris-Woodbury update is cubic with respect to the number of matrix rows and columns changed, this works well only when the modifications are very limited.

Nonlinear elastodynamics problems, in which the nonlinearities are due to rotations within an object, have been solved using a corotational approach in Hecht et al. [2012]. They approximate the rotation by applying an average of the rotations of the surround-
2.3 Hybrid Solution Methods

Some simulations have been implemented using hybrid approaches that employ two or more solution methods. Typically, a portion of a model is designated as susceptible to cuts and deformation, while the remainder is subject only to deformation. The strategy is to apply a fast precomputation approach to the portion of a model that cannot be cut and apply a slower method that supports cutting to the remainder. For example, Wu and Heng [2005] and Heng et al. [2004] combine the use of condensation and CG solvers, while Cotin et al. [2000] combine the use of a linear superposition method with explicit integration applied selectively to the dynamic, cuttable portion of a mesh. Koçak et al. [2009] provided further support for this approach by describing a framework for building a consistent finite-element simulation when different regions of the mesh are solved at different update rates.

3. METHODS

The augmented matrix approach presented here is a hybrid solution method that employs both direct and iterative solution algorithms without restricting cutting to a specific part of a model. An LDL^T factorization computed by a direct solver is used in conjunction with the generalized minimal residual method (GMRES), an iterative algorithm. Applied together, these methods compute fast and accurate solution updates for a finite element model as it undergoes stiffness matrix changes, including topological changes due to cutting.

This section is organized into three parts. First, we show how matrix augmentation can be used to express changes to a stiffness matrix. Next, we outline the steps required to solve an augmented system of finite-element equations. Finally, we detail how the sparsity inherent in the equations can be exploited to maximize the efficiency of the implementation.

3.1 Augmented Finite-Element Matrices

In an elastostatic finite-element model, an object is represented by a discrete mesh governed by a system of linear equations \( Ka = f \), where \( K \) is the \( n \times n \) global stiffness matrix, \( a \) represents nodal displacements, and \( f \) represents forces applied to mesh nodes. These are finite-element matrices and vectors that are constructed using standard finite-element methods [Bathe 1996]. Here, \( n \) is the number of degrees of freedom in the model; for a 3D solid model, \( n \) equals three times the number of mesh nodes.

If the \( i^{th} \) degree of freedom is involved in a change to the model, the \( i^{th} \) row and the \( i^{th} \) column of \( K \) will be modified to reflect the change in its relationship with the rest of the mesh. Changing any portion of \( K \) invalidates a previous factorization, necessitating a refactorization or an update. For a 3D finite-element model represented on a mesh with good aspect ratios, stiffness matrix refactorization can be performed in \( O(n^3) \) operations [Lipton et al. 1979]. While this is better than the \( O(n^3) \) complexity of matrix inversion, it does not provide the solution speed needed for interactive simulations.

The augmented formulation reflects changes to any limited portion of \( K \) while preserving the utility of its precomputed \( LDL^T \) factors. We rely on an effective column-replacement procedure applied to a matrix as follows. Suppose that we want to replace the third column of a matrix \( K^0 \) in a system with a vector \( k \) and compute the solution to the modified system. We can form the following augmented system of equations that, if exact arithmetic is used, will yield the same solution as the modified system after appropriate permutation of the solution vector.

\[
\begin{bmatrix}
K^0 & [e_3] \\
\v & a_3 \\
0 & a_4
\end{bmatrix}
\begin{bmatrix}
f_1 \\
f_2 \\
f_3 \\
f_4
\end{bmatrix}
= \begin{bmatrix}
f_1 \\
f_2 \\
f_3 \\
f_4
\end{bmatrix},
\]

where \( e_3^T \) is the third row vector of the identity matrix, and \( a_3 \) is a placeholder variable at the third component of the solution vector.

Note that multiplying the \( a \) vector by the last row of the augmented matrix constrains \( a_3 \) to be 0; thus, the whole third column of \( K^0 \) is multiplied by 0, canceling its effect on the system of equations. As the variable \( a_3 \) is multiplied by \( k \), this column acts as a replacement for the third column of \( K^0 \). This augmentation can be cascaded to replace multiple columns at the same time.

Suppose that \( K \), the global stiffness matrix at time \( t > t_0 \), differs from \( K^0 \), the initial stiffness matrix at time \( t_0 \), by \( m \) columns. We can use the effective column-replacement procedure mentioned earlier on these \( m \) columns to form an equivalent augmented system of equations

\[
K^A a^A = f^A,
\]

where superscript \( A \) suggests that the matrix and vectors are in augmented forms.

All topological mesh changes, including those resulting from cutting or element subdivision, can be represented in a finite element system of equations by replacing or deleting existing stiffness matrix columns and expanding the matrix to accommodate new columns. As described in Bathe [1996], the global stiffness matrix for a model is assembled by summing the contributions of the stiffness matrices of the individual elements. When a mesh is cut, the affected elements can be removed from the mesh by subtracting their contributions from the global stiffness matrix. Then, a
The set of replacement elements that respect the cut can be added to
the global stiffness matrix using the standard assembly procedure. For
a large mesh with a localized cut, this results in a small percentage of
columns in the global stiffness matrix being changed. These
changes can be implemented for a previously factored matrix using
the matrix augmentation technique presented previously.

Similarly, the imposition of Dirichlet boundary conditions that
specify the displacements of selected mesh nodes is accomplished
through the removal of the associated degrees of freedom from the
finite-element equations. In the standard formulation, this is ac-
complished by deleting the associated rows and columns from the
stiffness matrix. In the augmented matrix formulation, degrees of
freedom are removed via steps that resemble the effective column
replacement procedure in Equation (1). The following example il-
lustrates removal of the third degree of freedom from the augmented
system.

\[
\begin{bmatrix}
K^0 & e_3 \\
e_3^T & 0
\end{bmatrix}
\begin{bmatrix}
a_1 \\
a_2 \\
z_3 \\
a_4 \\
\vdots \\
- f_3 \\
\vdots \\
0
\end{bmatrix}
= 
\begin{bmatrix}
f_1 \\
f_2 \\
0 \\
f_4 \\
\vdots \\
0
\end{bmatrix}
- 
\begin{bmatrix}
a_1 K e_3 \\
0 \\
0 \\
0
\end{bmatrix}.
\]

As seen from Equation (1), the last row constrains \( z_3 \) to be 0. Per-
forming the multiplication of the third row yields

\[ K^0_{31} a_1 + K^0_{32} a_2 + K^0_{33} z_3 + K^0_{34} a_4 + \cdots - f_3 = -K^0_{33} a_3. \]

Substituting \( z_3 = 0 \) and rearranging the terms would get back
the third row of the standard formulation. Similarly, performing the
multiplication of the \( i \)th row other than the third row yields

\[ K^0_{i1} a_1 + K^0_{i2} a_2 + K^0_{i3} z_3 + K^0_{i4} a_4 + \cdots = f_i - K^0_{i3} a_3, \]

which is identical to the \( i \)th row of the standard formulation after
substitution of \( z_3 = 0 \) and rearrangements of terms.

Here \( f_3 \) is moved from the right-hand-side vector to the solution
vector since the force applied to the third degree of freedom becomes
unknown after the imposition of the Dirichlet boundary condition.
Note the similar structure of Equations (1) and (3), demon-
strating that both topological changes and imposition of Dirichlet
boundary conditions can be accomplished using a unified augmen-
tation procedure. Next, we provide the complete algorithm for formulating the
augmented system that supports both replacement and expansion
in a large mesh with a localized cut, this results in a small percentage
of degrees of freedom added to \( K \) at times \( t > t_0 \) corresponding to
possible new nodes added to the mesh due to the cut. The columns
of \( J \) inserted into \( K^A \) are effectively replaced by new columns in
\( K \). As shown here, \( J \) contains a copy of all columns of \( K \) that
have been added or changed, and \( H \) contains rows from the identity
matrix. The matrices \( J \) and \( H \) are defined as

\[
J_{i,j} = \begin{cases} 
K_{i,j} & \text{if } \mathcal{L}_i \notin \mathcal{D} \\
I_{i,j} & \text{if } \mathcal{L}_i \in \mathcal{D}
\end{cases} \quad \text{and} \quad (5)
\]

\[
H_{i,*} = I_{L,*} \quad (6)
\]

Here, \( \mathcal{D} \) is the set of degrees of freedom constrained by Dirichlet
boundary conditions, and \( \mathcal{L} \) is an accessory data structure that maps
the indices of columns and rows in \( J \) and \( H \) to the indices of
columns in \( K^0 \) to be replaced, that is, the \( i \)th column of \( J \) replaces
the \( L^0_i \) column of \( K^0 \). Hence, the \( i \)th column of \( J \) contains a copy
of the \( L^0_i \) column of \( K \).

Augmented displacement and force vectors must have sizes and
degree-of-freedom orderings consistent with the augmented stiff-
ness matrix. The augmented displacement vector, \( a^A \), can be par-
tioned into two parts denoted \( a \), a vector of length \( n \), and \( a_2 \), a
vector of length \( m \). Here,

\[
a^A = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} \quad (7)
\]

\[
(a_1)_i = \begin{cases} a_i & \text{if } i \notin \mathcal{L}, \\
z_i & \text{if } i \in \mathcal{L}
\end{cases} \quad (8)
\]

\[
(a_2)_i = \begin{cases} a_{L_i} & \text{if } i \notin \mathcal{D}, \\
-f_{L_i} & \text{if } i \in \mathcal{D}
\end{cases} \quad (9)
\]

As in Equations (1) and (3), the \( z \) terms are constrained to have a
value of zero, the \( a \) terms represent unknown nodal displacements,
and the \( f \) terms represent the unknown nodal forces when a new
Dirichlet boundary condition is imposed.

The augmented force vector is also partitioned into two parts: \( \hat{f} \)
of length \( n \), and a zero vector of length \( m \). Some components of
\( \hat{f} \) have terms subtracted to account for imposition of new Dirichlet
boundary conditions. Here,

\[
\hat{f}^A = \begin{bmatrix} \hat{f} \\ 0 \end{bmatrix} \quad (10)
\]

\[
\hat{f}_i = \begin{cases} f_i - \sum_{j \in \mathcal{D}} K_{i,j} a_j & i \notin \mathcal{D}, \\
- \sum_{j \in \mathcal{D}} K_{i,j} a_j & i \in \mathcal{D}
\end{cases} \quad (11)
\]

The augmentation procedure can be summarized by the following
four steps.

1. Construct the accessory data structures \( \mathcal{L} \) and \( \mathcal{D} \).
2. Form matrices \( J \) and \( H \) using Equations (5) and (6). Append
\( J \) to the right side of the stiffness matrix, \( K^0 \), and append \( H \) to
its bottom, as shown in Equation (4).
3. Form the right-hand-side vector \( f^A \) using Equations (10) and
(11).
4. After computing the solution, copy terms in \( a^A \) to the appro-
priate positions in the nodal displacement and force vectors as
indicated by Equations (8) and (9), discarding the \( z \) terms.
3.2 Solution Method

Since we assume a conservative material model, the reduced stiffness matrix $K$ is guaranteed to be symmetric positive definite. Thus, it can be factored as $K = L L^T D$, where $L$ is a lower triangular matrix and $D$ is a diagonal matrix. If new degrees of freedom are added to the model in subsequent timesteps, the factors can be padded as follows.

$$
\begin{bmatrix}
  K & 0 \\
  R & I
\end{bmatrix} = \begin{bmatrix}
  L_0 & 0 & 0 \\
  L & 0 & 0 \\
  D & 0 & 0 \\
  L^T & 0 & 0
\end{bmatrix},
$$

where $I$ is the identity matrix with dimension equal to the number of columns added to $K$, as in Equation (4). Using the matrix names shown earlier, the equation can be stated more compactly as $R = L D L^T$. Note, particularly, that $R$ refers to the global-stiffness matrix from time $t_0$ padded so that its dimension matches that of $K$ in the current timestep.

Substituting the definitions provided in Equations (4), (7), (10), and (12) into the augmented system of equations given in Equation (2) gives

$$
\begin{bmatrix}
  R & J \\
  H & 0
\end{bmatrix} \begin{bmatrix}
  a_1 \\
  a_2
\end{bmatrix} = \begin{bmatrix}
  \hat{f} \\
  0
\end{bmatrix}.
$$

(13)

Performing the multiplication in Equation (13) yields the following two equations:

$$
Ra_1 + Ja_2 = \hat{f},
$$

(14)

$$
Ha_1 = 0.
$$

(15)

Equation (14) can be rewritten as

$$a_1 = R^{-1}(\hat{f} - Ja_2).$$

(16)

Substituting Equation (16) into Equation (15) yields

$$HR^{-1}Ja_2 = HR^{-1}\hat{f}.$$  

(17)

The solution of Equation (13) can then be broken into three steps.

1. **Calculate the right-hand side of Equation (17).** Each occurrence of the multiplication $R^{-1}x$, for any vector $x$, can be efficiently calculated using the precomputed $LDL^T$ factors of $R$ via triangular matrix solves with forward and back substitution. We use this observation to first calculate the vector $y$, where $y = R^{-1}\hat{f}$, then calculate the product $Hy$ to arrive at the vector on the right-hand side of Equation (17).

2. **Solve Equation (17) to find $a_2$ using an iterative solver.** Since the right-hand side of Equation (17) is known from Step 1, an iterative solver can be used to successively improve estimates of $a_2$ if the multiplication $(HR^{-1})a_2$ can be performed. $J$ and $H$ have known values, and by making use of the $LDL^T$ factors of $R$ again, multiplication with $R^{-1}$ can be accomplished. As in Step 1, performing this multiplication requires triangular matrix solves with forward and back substitution. We use GMRES, a Krylov space iterative solver that requires only one or two matrix-vector multiplications per iteration. This choice minimizes the number of triangular solves needed.

3. **Substitute $a_2$ into Equation (16), then solve for $a_1$.** This step requires the use of the $LDL^T$ factors a final time to perform the multiplication of $R^{-1}$ with the known vector $(\hat{f} - Ja_2)$.

3.3 Accelerated Implementation Using Sparsity

3.3.1 Exploiting Sparsity in the Solution Steps. A careful examination of the sparsity of the matrices and vectors in Equations (17) and (16) allows us to maximize the efficiency of our implementation. The sparsity analysis is expressed using concepts from graph theory that are outlined in the Appendix. Each of the three solution steps outlined for Equations (16) and (17) in Section 3.2 involves computation with sparse vectors and matrices. We carefully exploit this sparsity to avoid unnecessary computation.

**Sparsity of Solution Step 1:** The right-hand side of Equation (17) is evaluated by computing

$$HR^{-1}f = HL^{-1}D^{-1}(L^{-1}f).$$

(18)

By applying Theorem 1 given in the Appendix, we find that $\text{struct}(L^{-1}f) \subseteq \text{closure}_L(f)$. This result says that the nonzero components in the vector in the left-hand side are given by components that can reach the nonzero components in the vector $f$ by an edge in a directed graph representation of the matrix $L$. Details are in the Appendix. Hence, only the submatrix of $L$ corresponding to $\text{closure}_L(f)$ is needed to evaluate the term $L^{-1}f$. We observe that the vector $f$ is typically sparse because external forces are applied to only a small fraction of the nodes while a mesh is being cut or deformed.

Consider that after we have computed the vector $y$ in Equation (18), it is projected through multiplication by the sparse matrix $H$. $H$ is composed of a few rows from an identity matrix, and has many more columns than rows. Only $m$ columns of $H$ contain nonzero components; consequently, all but $m$ components of $y$ are multiplied by 0 and do not contribute to the value of the right-hand-side vector. Let the components of $y$ necessary for the calculation be denoted $\tilde{y}$, such that $H\tilde{y} = H\bar{y}$. Then,

$$\bar{y}_i = \begin{cases} y_i & \text{if } H_{i,i} \neq 0, \\ 0 & \text{otherwise}. \end{cases}$$

(19)

By applying Theorem 2 from the Appendix, we find that only the submatrix of $L$ corresponding to $\text{closure}_L(\bar{y})$ is needed to complete the evaluation of the right-hand side of Equation (17).

**Sparsity of Solution Step 2:** During each GMRES iteration in solution Step 2, the solution estimate $a_2$ is projected by a sparse matrix $J$ to a larger space.

$$HR^{-1}Ja_2 = HL^{-1}D^{-1}L^{-1}(Ja_2).$$

(20)

Because the product matrix $Ja_2$ is sparse, only the submatrix of $L$ corresponding to $\text{closure}_L(Ja_2)$ is useful. As in Step 1, the vector $w$ is projected through multiplication by $H$. Hence, during the backward substitution, only a submatrix of $L$ corresponding to $\text{closure}_L(w)$ is needed for the computation.

**Sparsity of Solution Step 3:** Since in Step 3 both $f$ and $Ja_2$ are sparse, the difference vector $f - Ja_2$ is also sparse. Hence, the forward substitution can be sped up by considering only those needed rows of $L$. However, since the solution vector $a_1$ is not projected by a sparse matrix, the whole matrix $L$ is needed in the backward substitution.

We note that, in both Steps 1 and 3, the triangular solves are only done once; thus, we modify these routines to accept two additional inputs that indicate the sparsity of the right-hand-side vector and the
indices of needed components in the solution vector. However, the situation is different for Step 2, in which GMRES executes multiple iterations before converging. In this case, to reduce the overhead of indirect indexing, we explicitly form the needed submatrices by copying the needed rows and columns from $L$ and $L^\top$.

### 3.4 Complexity Analysis

Key parts of the complexity analysis hinge on the sparsity of the stiffness matrix, the complexity of the $LDL^\top$ factorization, and the sparsity of the factors. The number of nonzeros in each column of the global stiffness matrix is dependent on the connectivity between nodes. Since in a well-formed mesh the number of edges incident on a single node is limited by geometric considerations, the number of nonzeros per column can be bounded by a constant that is independent of the total number of degrees of freedom in the model. Due to this assumption and since sparse matrix data structures are used in this work, the complexity of all the steps in the augmented algorithm that update or otherwise operate on stiffness matrix columns is dependent only on the number of columns affected, not the number of rows in the matrix.

The derivation of complexity bounds for the $LDL^\top$ factorization can be found in Lipton et al. [1979]. Their work shows that using efficient sparse matrix algorithms, an $LDL^\top$ factorization of an $n \times n$ stiffness matrix for a 3D finite-element model can be accomplished with $O(n^2)$ operations; the resulting lower triangular matrix $L$ will contain $O(n^{4/3})$ nonzeros if the mesh elements have good aspect ratios. Therefore, without imposing any restrictions on the force vector, we can say that the triangular solves needed in Steps 1 and 3 of the solution algorithm described in Section 3.2 can be completed in $O(n^{4/3})$ operations.

Considering the sparsity analysis in the previous section raises the question of whether sparsity could provide a basis for a tighter complexity bound. In solution Step 1, the complexity depends on $|\text{closure}_f(f)|$ and $|\text{closure}_f(\tilde{y})|$. Theorem 3 from the Appendix informs us that the sizes of the closures depend on the length of the path from the root of the elimination tree to the nodes in $\text{struct}(f)$ and $\text{struct}(\tilde{y})$. If those nodes are close to the root, the sizes would be small constants, independent of $n$. On the other hand, if they are leaves of the elimination tree, the sizes would be close to $n$, and the cost of the triangular solve step would be linear in the number of nonzeros in $L$, $O(n^{4/3})$. In general, the cost lies in between these two extremes, and the upper bound of $O(n^{4/3})$ is not tight.

The complexity for each step in the algorithm, including the precomputation phase and the real-time update loop, is detailed in Table I. Summing the complexity of each of the real-time update steps for a 3D model and simplifying the expression to retain only the dominant terms results in a complexity bound for an update.

3.5 Preconditioning

The GMRES iteration in Step 2 can be preconditioned to reduce the number of iterations. One possible preconditioner is a matrix $M$ that approximates $HR^{-1}J$ in Equation (17). However, there are two drawbacks of this approach: neither forming the matrix $HR^{-1}J$ nor minimizing $\|M(HR^{-1}J) - I\|$ is computationally cheap. Also, $M$ has to be recomputed whenever there is a change to the mesh. Another possible preconditioner is a matrix product $HSJ$ such that $S$ approximates $R^{-1}$. In this case, $S$ needs to be computed only once and can be reused in later timesteps, even after changes to the mesh. In this article, we use two approximations of $S$: the inverse of $D$ in the precomputed $LDL^\top$ factors of $R$ (for all meshes), and a tridiagonal sparse approximate inverse (SPAI) [Benson and Frederickson 1982] of the matrix $R$ (for Stanford Bunny, brain, and eye meshes).

3.6 Refactorization

The augmented matrix approach produces a solution for a finite-element model with a time-varying stiffness matrix more quickly than a full refactorization would allow so long as $m$, the number of modified columns, is sufficiently small. As changes to a stiffness matrix accumulate across a growing number of columns, the augmented method begins to slow down because the size of the $oz$ vector also increases. To maintain fast solution speeds for an interactive simulation, we prevent $m$ from growing indefinitely by periodically recomputing a full $LDL^\top$ factorization of $K$ in a process that runs concurrently with the simulation loop. Figure 1 shows the changes in the equations before and after refactorization. Let $m'$ denote the number of columns modified due to cutting after the refactorization was initiated. When freshly computed factors are used to replace the

---

Table I. A Summary of the Calculation Steps Required by the Augmented Method, along with a Complexity Bound for Each Step

<table>
<thead>
<tr>
<th>Computation</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialization: $t = t_0$</td>
<td>$O(m)$</td>
</tr>
<tr>
<td>Real-time update steps: $t &gt; t_0$</td>
<td>$O(n^2)$ for 3D meshes; $O(n^{3/2})$ for 2D meshes</td>
</tr>
</tbody>
</table>

Note: $n$ is the order of the initial stiffness matrix, $m$ is the number of columns changed by cutting, and $|\text{closure}_f(f)|$ is the number of columns changed by cutting, and $|\text{closure}_f(\tilde{y})|$ is the number of columns modified due to cutting after the refactorization. Let $m'$ denote the number of GMRES iterations needed for convergence.
original factors, the size $m + m'$ is reduced to $m'$. The rate at which matrix changes accumulate will vary widely and depend both on the nature of a simulation and how quickly and aggressively a user manipulates a model. Even considering a single user and a single simulation, the growth rate of $m$ will vary unevenly across time as an interactive task progresses through moments of cutting, grasping, and pulling. Since the speed of our solution method is dependent on $m + m'$, the simulation update rate it provides is affected by the speed of mesh cutting and other manipulations. The best way to address this issue will depend on the application, but in some contexts, it is reasonable to limit the rate at which cutting can occur in order to maintain a desired update rate. Variability in the update rate arising from the refactorization process can be smoothed by buffering the computed solutions.

To provide some context for how refactorization will impact simulation speed, the results in Figures 5, 7, 10(a), 12, and 13 indicate the simulation step at which the cumulative time for mesh updates equals the time for matrix factorization, assuming one newly cut node per update and beginning with an empty list of mesh modifications. In practice, the list of recent mesh modifications will be nonempty when a refactorization step completes, there will be update steps that involve changes to multiple nodes, and many update steps will not involve any topological mesh changes. Depending on these factors, actual simulation update rates could be faster or slower when refactorization concludes than the times shown in the graphs. It is also feasible to run multiple refactorization processes concurrently, so that mesh changes get incorporated into the factors as quickly as possible and the size of $m$ is kept to a minimum. If multiple processors are available, this is one way to maximize the update rate since it is not necessary for one factorization process to complete before another begins.

4. RESULTS

The augmented matrix solution method was evaluated through finite-element deformation and cutting experiments with five model types. This section provides relevant implementation details and presents experimental data, including comparisons with both non-preconditioned and Jacobi-preconditioned CG solvers. ILU0 and ILUT preconditioners for CG were tested, but the reduction in number of iterations did not compensate for the increased computation complexity per iteration.

4.1 Implementation

All experiments were conducted on a desktop computer with four 8-core Intel Xeon E5-2670 processors running at 2.6GHz with 20GB cache and 256GB RAM. All data represent an average timing from 20 runs.

The precomputed $LDL^T$ factorizations of the stiffness matrices were computed using OBLIO, a sparse direct solver library [Dobrian and Pothen 2006]. Both the GMRES iterative solver used in solution Step 2 and the CG solver used for comparison purposes were from the Intel Math Kernel Library (MKL). The remainder of the code was written by the authors.

All matrices were stored in sparse matrix format to reduce both the storage space and access time. Since the closure of a set of indices in the graph of a triangular matrix can be found effectively column by column, and OBLIO uses supernodes in matrix factorization, all matrices were stored in compressed sparse column (CSC) format for efficient column access.

4.2 Model Meshes

Five types of solid tetrahedral meshes were used to evaluate the augmented matrix solution method in comparison to a traditional CG method. Meshes are shown in Figures 2 and 3.

(1) Elongated Beam: A group of five elongated rectangular solids with varying lengths were generated. Nodes were placed at regularly spaced grid points on a $5 \times 5 \times h$ grid, where $h$ ranged from 4 to 1024. The largest beam mesh has 25,600 nodes and 81,840 elements. Each block mesh was anchored at one end of the solid. All elements had good aspect ratios and were arranged in a regular pattern. However, models with greater degrees of elongation produced more poorly conditioned systems of equations, as fixation at only one end meant that longer structures were less stable. Thus, experiments with this group of meshes illuminates the way that solver performance varies with stiffness matrix conditioning. The estimated condition numbers of the beam mesh stiffness matrices range from $1.14 \times 10^3$ to $3.29 \times 10^{12}$.

(2) Brick: A group of five rectangular brick solids with varying mesh resolutions were generated. Each of the models had the same compact physical dimension of $1 \times 1 \times 2$. An initial good-quality mesh was uniformly subdivided to produce meshes of increasingly fine resolution. These meshes allowed
us to examine solver performance relative to node count for fixed model geometry. Similar to the beam meshes, zero-displacement boundary conditions were applied to one face of the block. The largest brick mesh has 18,081 nodes and 80,000 elements. The estimated condition numbers of the brick mesh stiffness matrices range from $2.19 \times 10^3$ to $1.18 \times 10^6$.

3) **Stanford bunny**: A 20,133 node, 62,698 element mesh of the Stanford bunny [Turk and Levoy 1994] is used to demonstrate solver performance on an irregular mesh. Zero-displacement boundary conditions are applied to nodes on the bottom of the bunny’s feet. The bunny mesh stiffness matrix has an estimated condition number of $6.17 \times 10^7$.

4) **Eye**: Incisions into a human eye model [Crouch and Cherry 2007] were used to demonstrate applicability to surgery simulation. Clear cornea cataract incisions were made into two models with resolutions containing 4,444 nodes and 14,841 elements, and 16,176 nodes and 52,772 elements. Zero-displacement boundary conditions were applied to the posterior portion of the globe. The eye mesh stiffness matrices have estimated condition numbers of $2.66 \times 10^6$ and $1.62 \times 10^7$, respectively.

Relaxing limbal incisions used to treat severe astigmatism were also simulated using the eye models. Simulation of the relaxing limbal incision procedure is of particular interest because the deformation induced by the incisions is not incidental to the procedure but rather is the motivating reason for performing the procedure. Astigmatism causes blurred vision due to an aspherical corneal surface, meaning that the corneal curvature is higher along some cross-sections than others. This variation in curvature can be reduced for patients through limbal incisions that are carefully placed around the periphery of the cornea to create a flattening effect along the meridian of highest curvature. Although guidelines exist for selecting appropriate placement, depth, and length for these incisions, such guidelines make a number of assumptions about a patient’s eye anatomy and cannot fully account for individual variations in corneal topography and thickness. Thus, simulations of this procedure might be useful both for individualizing treatment plans and as a teaching tool in medical education. While the tissue motion that is induced by relaxing limbal incisions is measured in millimeters, the resulting change in the optics of the cornea can be very significant, correcting up to 3 diopters of astigmatism. Since the cornea is responsible for two-thirds of the focusing power of the eye, small changes in corneal curvature can have a large impact on visual acuity. A linear elastic material model is appropriate for this application because the deformations are small in absolute terms. However, large, medically important changes in patients’ refractive error result from these small deformations.

5) **Brain**: Two resolutions of a human brain model (contributed by INRIA to the AIM@SHAPE Shape Repository) were used to demonstrate applicability to surgery simulation on an organ of complicated structure. The models contained 23,734 nodes and 81,746 elements, and 50,737 nodes and 167,366 elements. Zero-displacement boundary conditions were applied to the interior portion of the brain. The small brain mesh stiffness matrix has an estimated condition number of $4.64 \times 10^7$. The condition estimation failed for the large brain mesh due to insufficient memory.

On average, the nodes in the brick meshes have a higher degree of connectivity than those in the elongated beam meshes. This is due to a greater proportion of surface nodes in the beam models versus interior nodes in the brick models. The increased connectivity leads of the focusing power of the eye, small changes in corneal curvature can have a large impact on visual acuity. A linear elastic material model is appropriate for this application because the deformations are small in absolute terms. However, large, medically important changes in patients’ refractive error result from these small deformations.
Deformation update rates are shown for both the augmented and CG methods as constraints are progressively added to an increasing number of nodes in (a) beam and (b) brick meshes.

to a higher percentage of nonzeros in the stiffness matrix factors and larger sizes for the closures referenced in Table I. These differences have a significant impact on the relative performance of the solution methods. Figure 4 compares $|\text{closure}_L(J_{\text{all}})|$ for the different test meshes during the cutting experiments. The set closure$_L(J_{\text{all}})$ is the largest of the closures referenced in the complexity analysis in Table I, and is a measure of the size of the triangular system to be solved. As expected, brick meshes have larger closures than the other two meshes.

4.3 Experiments

Performance was examined through two types of experiments: deformation of intact meshes, and deformation of meshes undergoing cutting.

4.3.1 Deformation of Intact Meshes. In this group of experiments, we applied an increasing number of nonzero essential boundary conditions to mesh nodes to create deformation. Figure 5 shows how solution time varied with the number of constrained nodes for

Fig. 5. Deformation update rates are shown for both the augmented and CG methods as constraints are progressively added to an increasing number of nodes in (a) beam and (b) brick meshes.
Interactively Cutting Meshes Using Augmented Matrices

Fig. 7. Update rates are shown for the augmented and CG methods as a cut is advanced through a (a) beam mesh and (b) brick mesh.

Fig. 8. A portion of the tetrahedral brick test mesh. Node A has 13 connected nodes (colored in orange) whereas Node B only has 5 (colored in red).

completed after approximately 16 deformation steps. Thus, we see that the augmented method outperformed CG by a modest margin in the brick deformation experiment.

Figure 6 is a log-log plot that shows how solution times varied for different sizes of beam and brick meshes. These graphs show that the augmented method ran significantly faster than CG for the beam meshes except for the very smallest instance that had only 100 nodes. Most strikingly, on the largest beam mesh, which had 25,600 nodes, the augmented method provided updates at a rate of 113Hz, while CG ran at $3 \times 10^{-5}$Hz. For the brick meshes, the augmented method ran faster than CG, although the margin was smaller.

4.3.2 Deformation of Meshes Undergoing Cutting. In this group of experiments, we made an advancing planar cut into the volume of each mesh. As a cut progressed, a duplicate of each node along the cut path was added to the mesh, and connectivity was modified so that elements on opposite sides of the cut became separated. These changes required expanding the stiffness matrix and modifying existing entries in the stiffness matrix at dozens of locations each time a node was duplicated. Opposing force vectors were applied to selected surface nodes to pull the cut faces apart. Figure 2 shows the three test meshes at the initial stages of cutting.

Fig. 10. Timing results are provided for the bunny mesh cutting experiment. (a) Update rates are shown for the augmented and CG methods as a cut is advanced. (b) The allocation of computation time to steps of the augmented method is shown.

The differences between the results for the beam and brick meshes are even more pronounced for the cutting experiment than for the deformation experiment. Figure 7(a) shows that the augmented method outperformed CG in the beam-cutting experiments, providing updates in the range 49–145Hz in the time period before refactorization completed. CG provided updates in the range 0.26–172Hz for the same cutting steps, but failed to converge to any solution for seven of those steps. However, CG provided consistently better performance for the brick mesh cutting experiment, as shown in Figure 7(b). The zig-zag appearance of the CG results was caused by the connectivity pattern of nodes in the tetrahedral brick mesh. Periodically, nodes with a higher degree of connectivity were cut. These cutting steps required a larger number of changes to the stiffness matrix and resulted in periodically slower CG solution times. The connectivity pattern is illustrated in Figure 8.

Figure 9 shows that the beam vs. brick performance trend held over a variety of mesh sizes. The augmented method provided the fastest updates when cutting a beam mesh, maintaining an update rate over 50Hz even with a relatively large cut in a 25,600 node mesh. Particularly for the larger beam meshes, CG was often unable to provide any solution. However, CG reliably provided the fastest updates when cutting a brick mesh.

Results from the bunny mesh-cutting experiment are shown in Figure 10. Here, we find that the nonpreconditioned augmented method performed best, with a minimum update rate of 14Hz during the period before refactorization completes. As seen in some of the previous experiments, the update rate provided by the augmented method diminishes as the size of the cut and complexity of the attendant remeshing grows. However, the augmented method is still faster than the 0.3–6.8Hz update rate provided by preconditioned CG in this experiment. Figure 10(b) shows that the bulk of the computation time is spent in the GMRES iteration of Step 2 in the bunny mesh-cutting experiment. The dominance of the GMRES iterations in the distribution of computing time is also a feature of the experiments with beam and brick meshes. However, Figure 11(a) demonstrates that the number of GMRES iterations needed for convergence does not grow with model size.
Results from the eye mesh-cutting experiments are shown in Figure 12, and those from the brain mesh-cutting experiments are shown in Figure 13. Here, we show that the augmented method outperformed the CG method with and without preconditioning. However, the update rate for the brain meshes remains lower than desired for interactive simulation. Further reduction of the solution times for large, dense meshes is a priority for future work.

The experimental results also indicate that the augmented solution method does not lead to problems with solution accuracy. Figure 11(b) shows that the relative error of the computed solutions remains flat as a brick mesh is cut and increases only gradually as the less stable beam mesh is cut.

5. CONCLUSIONS AND FUTURE WORK

There are two primary reasons for the disparity between the beam mesh and brick mesh results. First, the beam meshes have a higher percentage of surface nodes, resulting in sparser matrix factors and smaller closure sizes, as shown in Figure 4. Smaller closures result in faster execution of the augmented solution steps, particularly the GMRES iterations in Step 2. Thus, we see that the structure of a mesh is an important factor in determining whether the augmented method will be a particularly efficient solution method for a given problem. In general, the augmented method is particularly attractive for meshes that have greater amounts of surface area relative to their volume. The second reason for the wide disparity in results is that the brick meshes had particularly well-conditioned stiffness matrices, while the beam meshes had more poorly conditioned stiffness matrices. Iterative methods can converge very slowly or fail to converge at all when systems are not sufficiently well conditioned. In contrast, the direct solution approach provided by the augmented factors is more robust in poorly conditioned scenarios. We conclude that the augmented method is particularly appropriate when a problem would benefit from the robustness of a direct solution approach, but also needs the flexibility to update the system due to cutting or other changes.

In summary, we have demonstrated the feasibility of using augmented matrices to provide fast updates for finite-element models undergoing cutting and deformation. The augmented method has been experimentally shown to offer advantages both in speed and...
reliability for certain classes of problems. We plan to explore the applicability of this method to a wider range of problems in future work. One particular application to investigate is surgery simulation, for which there is evidence that viscoelastic and hyperelastic material models are often appropriate for soft-tissue modeling [Fung 1993; Lapeer et al. 2010; Marchesseau et al. 2010]. Nonlinear material models can require stiffness matrix updates at each timestep, even without cutting. However, in the case of tool-tissue interaction, acceptable nonlinear accuracy might possibly be achieved by updating the stiffness of a subset of only the most deformed elements or those closest to the contact area. This raises the interesting possibility of using the augmented matrix method for fast updates of nonlinear materials.

Another direction for future investigation is inspired by the variety of recent publications that have reported acceleration of solution methods via GPU implementations [Dick et al. 2011b; Courtecuisse et al. 2010b; Joldes et al. 2010; NVIDIA 2015]. Our augmented matrix solution method could likely be similarly accelerated if the triangular solves and/or GMRES algorithm were implemented in a way that makes efficient use of GPU processing.

APPENDIX

Graph theory concepts relied on in the discussions of sparsity and complexity are outlined here. Included are the definitions and theorems referenced in Section 3.3. Note that, in this discussion, the matrix A is nonsymmetric. We apply these results to the lower and upper triangular factors of the stiffness matrix K, although the results here are more general.

Definition 1. An $n \times n$ sparse matrix A can be represented by a directed graph $G(A)$ whose vertices are the integers 1, ..., $n$ and whose edges are

$$[(i, j) : i \neq j, \text{ and } A_{ij} \neq 0].$$

This set of indices is called the structure of A.

Definition 2. The transitive reduction of a directed graph $G(L)$ is the graph obtained by removing edges (i, j) whenever there is a directed path (that does not use the edge (i, j)) joining vertices i and j. An elimination tree of a Cholesky factor L is the transitive reduction of the directed graph $G(L)$ (in this case, it is a tree rather than a directed acyclic graph) [Liu 1990].

Definition 3. The structure of a vector x with n components is

$$\text{struct}(x) := \{i : x_i \neq 0\},$$

which can be interpreted as a set of vertices, $W_r$, of the directed graph of $G(A)$ such that $i \in W$ if and only if $x_i \neq 0$ when solving $Ax = b$ or $Ay = x$. In this article, for a vector x, closure$_\ell(x)$ refers to closure$_\ell(\text{struct}(x)).$

Definition 4. Given a directed graph $G(A)$ and a subset of its vertices denoted by $W_r$, we say that $W_r$ is closed with respect to A if there is no edge of $G(A)$ that joins a vertex not in $W_r$ to a vertex in $W_r$; that is, $v_j \in W_r$ and $A_{ij} \neq 0$ implies that $v_i \in W_r$. The closure of $W_r$ with respect to A is the smallest closed set containing $W_r$,

$$\text{closure}_\ell(W_r) := \bigcap\{U : W_r \subseteq U, \text{ and } U \text{ is closed}\},$$

which is the set of vertices of $G(A)$ from which there are directed paths in $G(A)$ to vertices in $W_r$.

THEOREM 1. Let the structures of A and b be given. Whatever the values of the nonzeros in A and b, if A is nonsingular, then

$$\text{struct}(A^{-1}b) \subseteq \text{closure}_\ell(b).$$

The proof of Theorem 1 can be found in Gilbert [1994].

THEOREM 2. Suppose that we need only some of the components of the solution vector x of the system $Ax = b$. Denote the needed components by $\ell$. If A is nonsingular, then the set of components in b needed is $\text{closure}_\ell(x)$.

Proof. Let values be given for which $A$ is nonsingular. Renumber the vertices of $G(A')$ so that $\text{closure}_\ell(x) = \{1, 2, \ldots, k\}$ for some $k \leq n$. Then, $Ax = b$ can be partitioned as

$$\begin{bmatrix} B & D \\ C & E \end{bmatrix} \begin{bmatrix} y \\ z \end{bmatrix} = \begin{bmatrix} d \\ e \end{bmatrix},$$

where $B$ is $k \times k$. By the definition of $\text{closure}_\ell(x)$, there is no edge $(i, j)$ with $i \in \text{closure}_\ell(x)$ and $j \notin \text{closure}_\ell(x)$. Therefore, $D = 0$. Then, $By = d$. Since A is nonsingular, B is also nonsingular. Thus, $\hat{x}$ can be computed by solving only $By = d$, which implies that only $\text{closure}_\ell(x)$ is needed to compute the components in $\hat{x}$. □

THEOREM 3. Let $A = LL^T$ be a Cholesky factorization and $\mathbb{W}$ be a subset of vertices in $G(L)$. If r is the root of the elimination tree $T$ of $L$, then

$$\text{closure}_\ell(\mathbb{W}) = \bigcup_{v \in \mathbb{W}} [r \to v],$$

where $r \Rightarrow v$ is the path from r to v in $T$, including all intermediate vertices along the path.

Proof. (i) $\bigcup_{v \in \mathbb{W}} [r \Rightarrow v] \subseteq \text{closure}_\ell(\mathbb{W})$:

For any edge between a node v and its parent u in $T$, there is an edge $(u, v)$ in $G(L)$. By definition, if $v \in \text{closure}_\ell(\mathbb{W})$, then $u \in \text{closure}_\ell(\mathbb{W})$. Since $\mathbb{W} \subseteq \text{closure}_\ell(\mathbb{W})$, all ancestors of $\mathbb{W}$ must be in $\text{closure}_\ell(\mathbb{W})$.

(ii) $\text{closure}_\ell(\mathbb{W}) \subseteq \bigcup_{v \in \mathbb{W}} [r \Rightarrow v]$:

If a node $u \notin \bigcup_{v \in \mathbb{W}} [r \Rightarrow v]$, there must be a path from a node $w \in \bigcup_{v \in \mathbb{W}} [r \Rightarrow v]$ to u. Hence, there is also a directed path from v to u in $G(L)$. Since L is lower triangular, there is no cycle in $G(L)$. Hence, there is no directed path from $u$ to $w$ and $u \notin \text{closure}_\ell(\mathbb{W})$. □

ACKNOWLEDGMENTS

We are grateful to the owners of the models used in the experiments. The Stanford bunny model is provided courtesy of the Stanford Computer Graphics Laboratory by the Stanford 3D Scanning Repository. The human eye model was created by one of us (JRC). The human brain model is provided courtesy of INRIA by the AIM@SHAPE Shape Repository. We thank the reviewers for their careful reading and helpful suggestions.

REFERENCES


Received April 2014; revised September 2015; accepted December 2015