2016

Interactively Cutting and Constraining Vertices in Meshes Using Augmented Matrices

Yu-Hong Yeung

Jessica Crouch
Old Dominion University

Alex Pothen

Original Publication Citation

Yeung, Y. H., Crouch, J., & Pothen, A. (2016). Interactively cutting and constraining vertices in meshes using augmented matrices. *ACM Transactions on Graphics, 35*(2), 18. doi:10.1145/2856317

Interactively Cutting and Constraining Vertices in Meshes Using Augmented Matrices

YU-HONG YEUNG
Purdue University
JESSICA CROUCH
Old Dominion University
and
ALEX POTHEN
Purdue University

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 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.

Categories and Subject Descriptors: I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Physically based modeling*; I.6.3 [**Simulation and Modeling**]: Applications; I.1.2 [**Symbolic and Algebraic Manipulation**]: Algorithms—*Algebraic algorithms*

General Terms: Theory, Performance

Additional Key Words and Phrases: Finite element, surgery simulation, real-time, deformable model, cutting

ACM Reference Format:

Yu-Hong Yeung, Jessica Crouch, and Alex Pothen. 2016. Interactively cutting and constraining vertices in meshes using augmented matrices. ACM Trans. Graph. 35, 2, Article 18 (February 2016), 17 pages.
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

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: jrcrouch@cs.odu.edu.

DOI: http://dx.doi.org/10.1145/2856317

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 literature, 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 [Keaveny et al. 1994; Andreas 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 Turkiyyah'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 rearranges 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 Turkiyyah [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; Turkiyyah 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 Turkiyyah 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 Turkiyyah 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 Turkiyyah [2007], an update procedure similar to the Sherman-Morrison-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-Morrison-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 surrounding elements to a node. They compute the solution by updating the Cholesky factors of the system matrix by exploiting the nonzero pattern of the factor due to a nested dissection ordering, observing as we have done here that only the submatrices of the factor that lie on a path in the elimination tree from a submatrix to the root of the tree are affected. In order to have the simulations run fast, they trade off an increased error tolerance for time by choosing which submatrices in the factors to update. There are major differences between our work and theirs. They have applied their work to nonlinear problems in which rotations are the major source of the nonlinearity, while our work in this article applies to linear problems. Our augmented matrix approach models the stiffness system exactly, and the solutions should be identical to the original system in exact arithmetic. They have not applied their work to cutting problems addressed in this work, which would require the ability to handle changes in the mesh topology. The way that the two approaches exploit sparsity is also different. Hecht et al. [2012] have chosen to update selected submatrices of the Cholesky factors, which requires dynamic updates to the large data structure that stores the Cholesky factor and the update matrices, increasing the storage needs. Our approach is to update the solution, but not the Cholesky factors, by *implicitly* solving a Schur complement system with a Krylov space solver, without forming the Schur complement matrix. We exploit the sparsity not only in the factor, but also in the right-hand-side vectors and the solutions, as described in Section 3.3 and the Appendix. The time complexity of the update step is bounded linearly by the number of nonzeros in the (static) Cholesky factor (and the number of iterations of the Krylov space solver) in our case, but the corotational approach cannot be bounded in this manner since the factors are updated.

Finally, we consider the CHOLMOD approach of Chen et al. [2008] for updating the Cholesky factors when rows or columns are added or deleted from the matrix. This algorithm relies on dynamic supernodal updates of the Cholesky factor. Unfortunately, the number of columns to be added or deleted (change in the rank of the factor) during the cutting of meshes is much larger than can be efficiently performed with this software since we need to remesh around the cut. Hecht et al. [2012] have come to similar conclusions for their problem. Furthermore, dynamic updates to the large sparse Cholesky factor and update matrix data structure are expensive; instead, we work with an implicit Schur complement approach whose time complexity can be bounded linearly by the size of the Cholesky factor.

2.2 Iterative Solvers

Iterative solvers do not share the same limitations as precomputation methods because all of the calculations needed to produce a solution occur at runtime. Thus, iterative solvers can be successfully applied when stiffness matrix updates are caused by topological mesh changes. However, using an iterative solver does require that attention be paid to issues of convergence and stability.

Conjugate Gradient (CG) solvers have been frequently used with finite-element simulations [Frank et al. 2001; Nienhuys and van der Stappen 2001; Courtecuisse et al. 2010b]. The popularity and relatively straightforward implementation of the CG algorithm make the method a good benchmark for comparisons with alternative solution methods. CG implementations that take advantage of sparse matrix-vector multiplication have been used for interactive applications and can be accelerated with parallel [Chentanez et al. 2009] and GPU implementations [Wu and Heng 2004]. However, the simulation community continues to seek solution methods that outperform CG, as real-time performance on higher-resolution models promises improved realism.

Some of the most recent work in interactive finite-element simulation has explored the use of multigrid solution methods [Dick et al. 2011a; Zhu et al. 2010; Georgii and Westermann 2006; Wu and Tendick 2004]. Multigrid methods are among the most efficient iterative solution approaches and accelerate convergence by reducing error at multiple spatial resolutions. However, they also have higher fixed overhead costs than methods such as Conjugate Gradient. A GPU implementation of multigrid by Dick et al. [2011b] has demonstrated further speed improvement. Preconditioning can also be used to speed up the iterative solvers. Courtecuisse et al. [2010a] have demonstrated the asynchronous update of Cholesky factors for preconditioning on a separate thread. The asynchronous update of the factors is similar to our refactorization scheme. The need for a multiresolution mesh makes multigrid naturally suited for structured meshes, with hexahedral grids typically being used. Although hexahedral grids do not lend themselves to smooth cutting surfaces, recent work by Zhu et al. [2010] has demonstrated a way to incorporate cutting into a simulation with a multigrid solver.

A final category of iterative solvers is explicit integration methods. Explained in detail in Bathe [1996], it was originally suggested for use in surgery simulation by Bro-Nielsen [1998] and also implemented in the software suite described in Joldes et al. [2009]. Explicit integration has been successfully used in real-time simulation with dynamic and nonlinear finite-element models [Wu et al. 2001], and has been used in simulations involving surgery [Wittek et al. 2010] and cutting [Serby et al. 2001]. Care must be taken in selecting the timestep size for explicit integration because it can be numerically unstable if the timesteps are too large.

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 $LDL^\top$ 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 $O(n^2)$ 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 $LDL^\top$ 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.

$$\left[\begin{array}{ccc|c} & K^0 & & k \\ \hline & e_3^\top & & 0 \end{array}\right] \begin{bmatrix} a_1 \\ a_2 \\ z_3 \\ a_4 \\ \vdots \\ a_3 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ \vdots \\ 0 \end{bmatrix}, \tag{1}$$

where $e_3^\top$ is the third row vector of the identity matrix, and z_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 z_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

$$K^A a^A = f^A, \tag{2}$$

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

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 accomplished 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 illustrates removal of the third degree of freedom from the augmented system.

$$\left[\begin{array}{c|c} K^0 & e_3 \\ \hline e_3^\top & 0 \end{array}\right]\left[\begin{array}{c} a_1 \\ a_2 \\ z_3 \\ a_4 \\ \vdots \\ -f_3 \end{array}\right] = \left[\begin{array}{c} f_1 \\ f_2 \\ 0 \\ f_4 \\ \vdots \\ 0 \end{array}\right] - \left[\begin{array}{c} a_3 K e_3 \\ \\ \\ \\ \\ 0 \end{array}\right]. \quad (3)$$

As seen from Equation (1), the last row constrains z_3 to be 0. Performing 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), demonstrating that both topological changes and imposition of Dirichlet boundary conditions can be accomplished using a unified augmentation procedure. Next, we provide the complete algorithm for formulating the augmented system that supports both replacement and expansion affecting multiple matrix columns.

In the ensuing discussion, any matrix or vector without a 0 superscript is assumed to reference the model at some time $t > t_0$. At the beginning of a simulation, the augmented system is identical to the standard finite-element system at time t_0. For timesteps $t > t_0$, K^A retains K^0 as a submatrix so that precomputed factors of K^0 remain useful, and new rows and columns contained in rectangular matrices J and H are appended to account for the updates in K. Mathematically, K^A has the form

$$K^A = \underbrace{\left[\begin{array}{cc|c} K^0 & 0 & J \\ 0 & I & \\ \hline H & & 0 \end{array}\right]}_{n \quad m} \begin{array}{l} \left.\begin{array}{l} \\ \\ \end{array}\right\} n \\ \left.\begin{array}{l} \end{array}\right\} m \end{array}. \quad (4)$$

Here, I is the identity matrix with dimension equal to the number 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 I 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

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

$$H_{i,*} = I_{\mathcal{L}_i,*}. \quad (6)$$

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 $\mathcal{L}_i^{\text{th}}$ column of K^0. Hence, the i^{th} column of J contains a copy of the $\mathcal{L}_i^{\text{th}}$ column of K.

Augmented displacement and force vectors must have sizes and degree-of-freedom orderings consistent with the augmented stiffness matrix. The augmented displacement vector, a^A, can be partitioned into two parts denoted a_1, a vector of length n, and a_2, a vector of length m. Here,

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

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

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

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,

$$f_i^A = \begin{bmatrix} \hat{f} \\ 0 \end{bmatrix}, \quad (10)$$

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

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 appropriate positions in the nodal displacement and force vectors as indicated by Equations (8) and (9), discarding the $z_{\mathcal{L}_i}$ 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 $K^0 = L_0\, D_0\, L_0^\top$, where L_0 is a lower triangular matrix and D_0 is a diagonal matrix. If new degrees of freedom are added to the model in subsequent timesteps, the factors can be padded as follows.

$$\underbrace{\begin{bmatrix} K^0 & 0 \\ 0 & I \end{bmatrix}}_{R} = \underbrace{\begin{bmatrix} L_0 & 0 \\ 0 & I \end{bmatrix}}_{L} \underbrace{\begin{bmatrix} D_0 & 0 \\ 0 & I \end{bmatrix}}_{D} \underbrace{\begin{bmatrix} L_0^\top & 0 \\ 0 & I \end{bmatrix}}_{L^\top}, \tag{12}$$

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 $R = LDL^\top$. 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

$$\begin{bmatrix} R & J \\ H & 0 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \hat{f} \\ 0 \end{bmatrix}. \tag{13}$$

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

$$R\,a_1 + J\,a_2 = \hat{f}, \tag{14}$$

$$H\,a_1 = 0. \tag{15}$$

Equation (14) can be rewritten as

$$a_1 = R^{-1}\big(\hat{f} - J\,a_2\big). \tag{16}$$

Substituting Equation (16) into Equation (15) yields

$$\big(HR^{-1}J\big)\,a_2 = HR^{-1}\hat{f}. \tag{17}$$

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 $LDL^\top$ 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 $H\,y$ 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 $(HR^{-1}J)a_2$ can be performed. J and H have known values, and by making use of the $LDL^\top$ 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 $LDL^\top$ factors a final time to perform the multiplication of R^{-1} with the known vector $(\hat{f} - J\,a_2)$.

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

$$HR^{-1}f = H\underbrace{L^{-\top}D^{-1}\left(L^{-1}f\right)}_{\textcircled{y}}. \tag{18}$$

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 $\hat{y}$, such that $H\hat{y} = Hy$. Then,

$$\hat{y}_i = \begin{cases} y_i & \text{if } H_{*,i} \neq 0, \\ 0 & \text{otherwise.} \end{cases} \tag{19}$$

By applying Theorem 2 from the Appendix, we find that only the submatrix of $L^\top$ corresponding to $\text{closure}_L(\hat{y})$ 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.

$$\big(HR^{-1}J\big)\,a_2 = H\underbrace{L^{-\top}D^{-1}L^{-1}\,(Ja_2)}_{\textcircled{w}}. \tag{20}$$

Because the product vector 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 $L^\top$ 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 α_1 is not projected by a sparse matrix, the whole matrix $L^\top$ 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

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

	Computation	Complexity
Initialization: $t = t_0$		
1	Compute $LDL^\top$ factorization of K^{A_0}	$O(n^2)$ for 3D meshes; $O(n^{3/2})$ for 2D meshes
Real-time update steps: $t > t_0$		
1	Update K	$O(m)$
2	Compute J and H	$O(m)$
3	Solution Step 1	$O(\lvert \text{closure}_L(f) \rvert + \lvert \text{closure}_L(\hat{y}) \rvert) \leq O(\lvert L \rvert)$
4	Solution Step 2	$O(\lvert \text{closure}_L(Ja_2) \rvert + \lvert \text{closure}_L(\hat{y}) \rvert) \cdot n_{iter} \leq O(\lvert L \rvert \cdot n_{iter})$
5	Solution Step 3	$O(\lvert L \rvert)$

Note: n is the order of the initial stiffness matrix, m is the number of columns changed by cutting, and $|L|$ in solution Steps 1, 2, and 3 is the number of nonzeros in L, which is bounded by $o(n^{4/3})$ for 3D meshes and $O(n \log n)$ for 2D meshes. The complexity upper bound for an entire update iteration is $O(|L| \cdot n_{iter})$, where n_{iter} is the number of GMRES iterations needed for convergence.

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 $|\text{closure}_L(f)|$ and $|\text{closure}_L(\hat{y})|$. 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 struct(f) and struct($\hat{y}$). 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

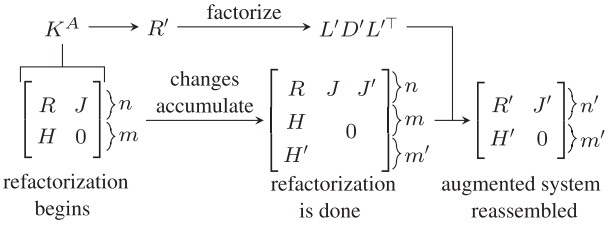


Fig. 1. Refactorization process.

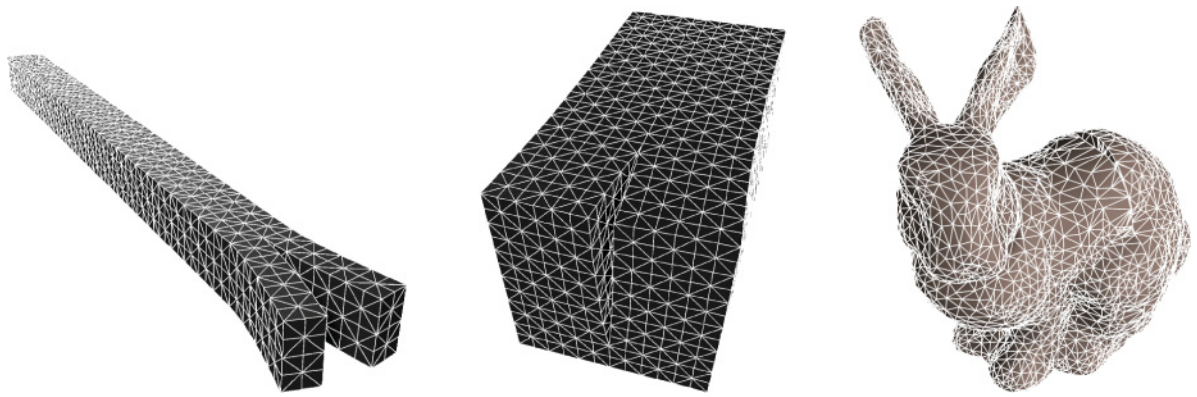
iteration of $O(n^{4/3} \cdot n_{iter})$, where n_{iter} is the number of GMRES iterations needed for convergence. Note that n_{iter} is influenced by m, the number of columns updated, rather than n, the dimension of the stiffness matrix.

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 Federickson 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 a_2 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



(a) (b) (c)

Fig. 2. Test meshes are shown after cuts described in the experiments in Section 4.2.

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 $LDL^\top$ 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 1.14×10^3 to 3.29×10^{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

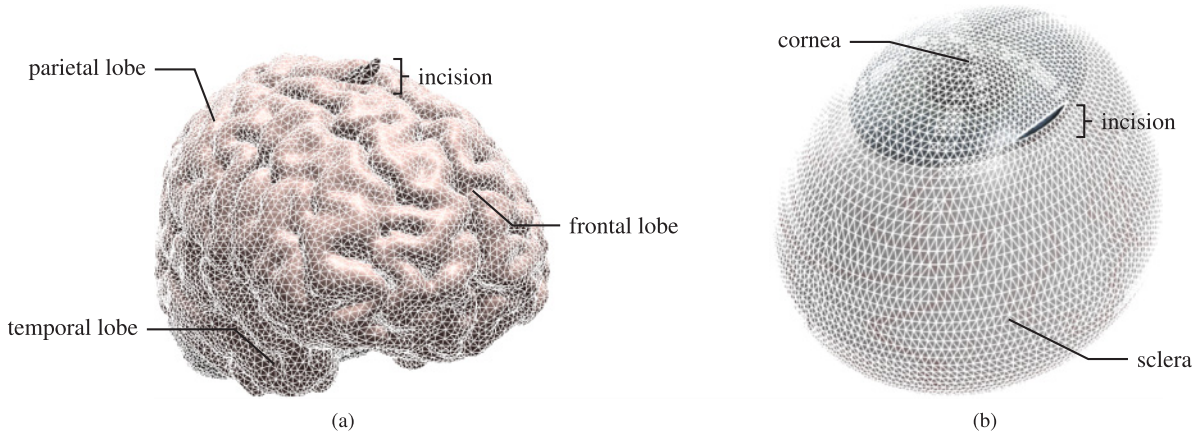


Fig. 3. Renderings of brain and eye models are shown with incisions used in the experiments reported in Section 4.2. (a) The incision on the brain model shown is on the superior portion of the right frontal lobe. (b) The incision on the eye model shown is along the corneal limbus, to correct for astigmatism.

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 2.19×10^3 to 1.18×10^5.

(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 6.17×10^7.

(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 2.66×10^6 and 1.62×10^7, 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

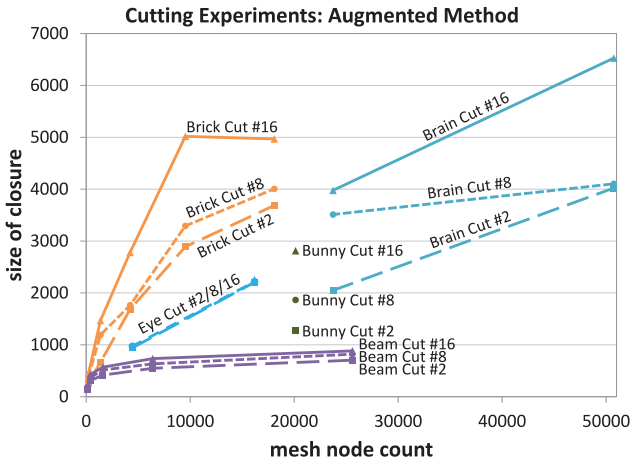
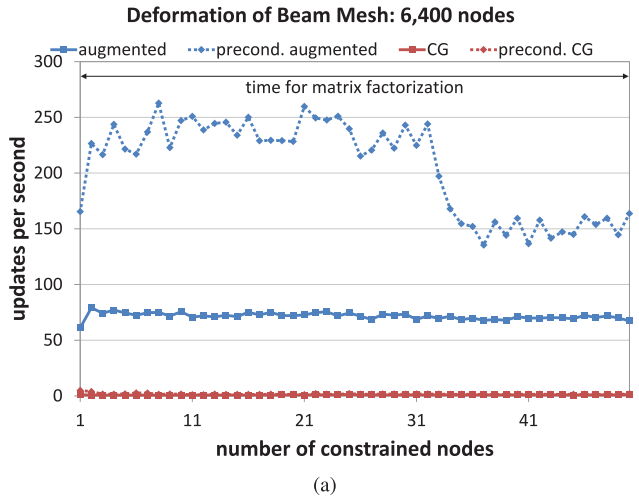


Fig. 4. $|\text{closure}_L(Ja2)|$ versus node count, shown for cutting Steps 2, 8, and 16.

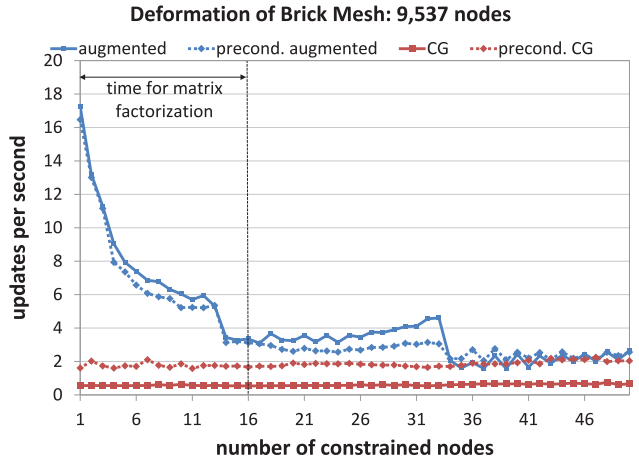
of the focusing power of the eye, small changes in corneal curvature can have a large impact on visual acuity. A linear elastic material model is appropriate for this application because the deformations are small in absolute terms. However, large, medically important changes in patients' refractive 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 4.64×10^7. 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

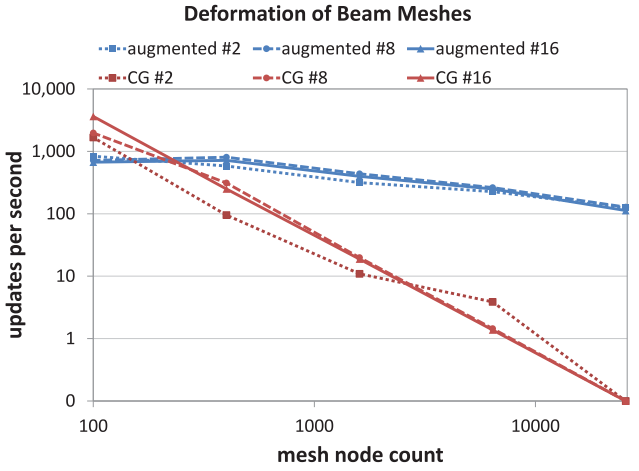


(a)

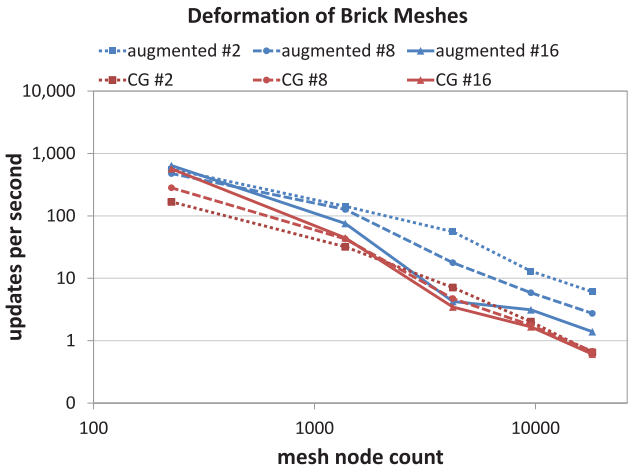


(b)

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.



(a)



(b)

Fig. 6. Update rates are shown for the series of (a) beam and (b) brick meshes. CG results are shown with red lines, and augmented method results are shown with blue lines. Dotted lines show the results for deformation Step 2 across the series of test mesh sizes. Dashed lines show results for deformation Step 8, and solid lines for deformation Step 16.

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 $|\text{closure}_L(Ja_2)|$ for the different test meshes during the cutting experiments. The set $\text{closure}_L(Ja_2)$ 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 instances of the beam and brick meshes. It is interesting to note the dramatically different results for the beam meshes versus the brick meshes in these experiments. As shown in Figure 5(a), the augmented method maintained a high update rate for the beam meshes throughout, and vastly outperformed the CG method. The beam deformation experiments ran so fast with the augmented method that the experiments concluded before there was time to compute a refactorization. In the example shown in the figure, the update cycles ran at rates between 137–263Hz.

For brick meshes, the augmented method outperformed CG as constraints were applied to the first one to two dozen nodes, but performance dropped as the number of constrained nodes increased, eventually resulting in similar update rates between the augmented method and CG. However, since the brick mesh experiments ran more slowly overall, refactorization played a meaningful role in the augmented solution process. In the results shown in Figure 5(b), a refactorization process running concurrently with the solution loop

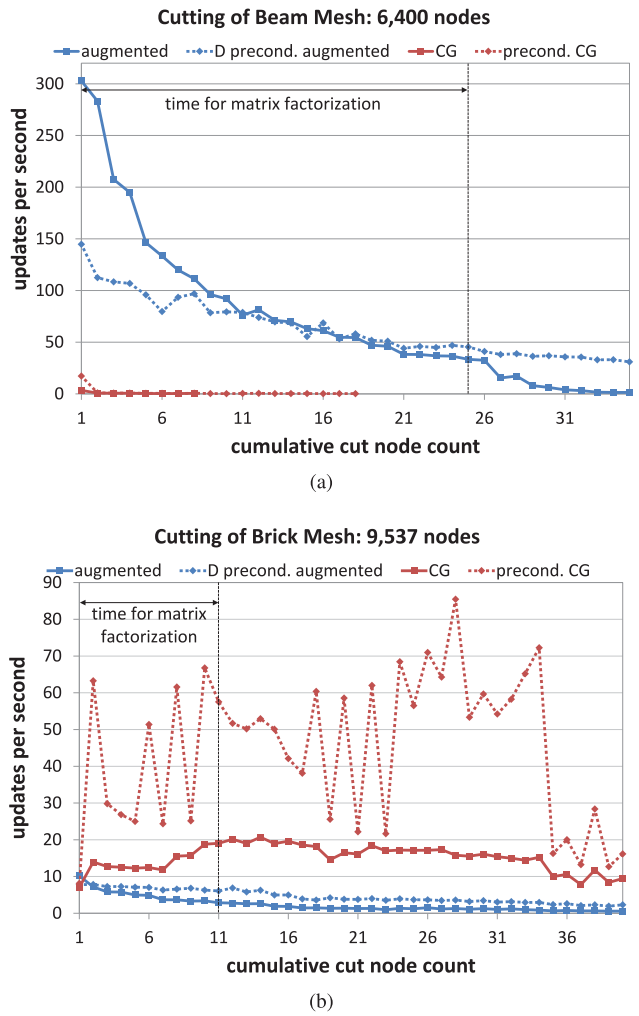


Fig. 7. Update rates are shown for the augmented and CG methods as a cut is advanced through a (a) beam mesh and (b) brick mesh.

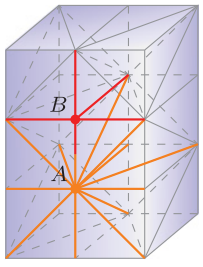


Fig. 8. A portion of the tetrahedral brick test mesh. Node *A* has 13 connected nodes (colored in orange) whereas Node *B* only has 5 (colored in red).

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

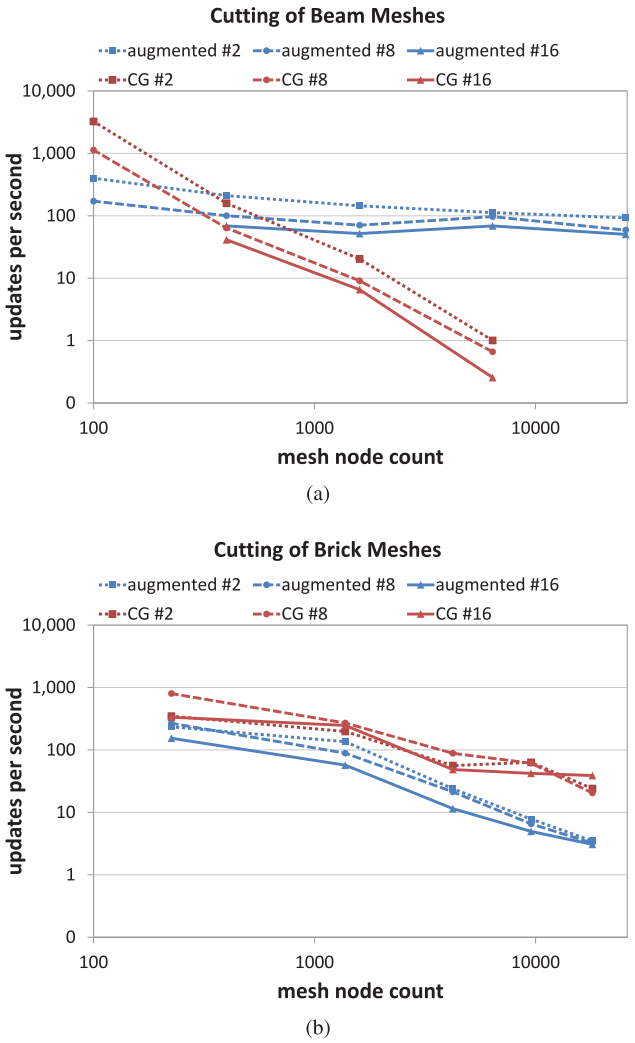
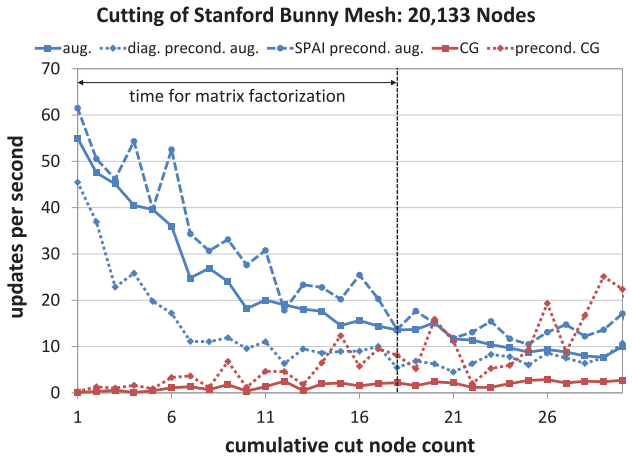


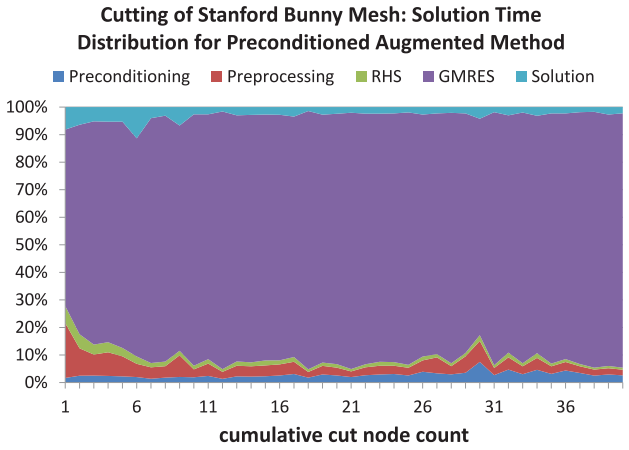
Fig. 9. Update rates are shown for the series of (a) beam and (b) brick meshes. CG results are shown with red lines, and augmented method results are shown with blue lines. Dotted lines show the results for cutting Step 2 across the series of test mesh sizes. Dashed lines show results for cutting Step 8, and solid lines for cutting Step 16.

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 3×10^{-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.

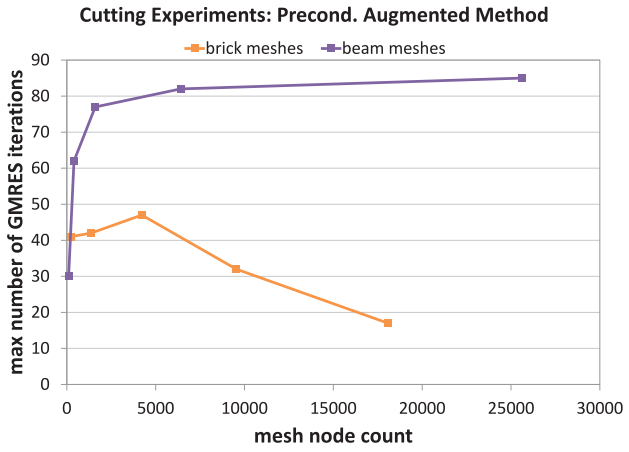


(a)

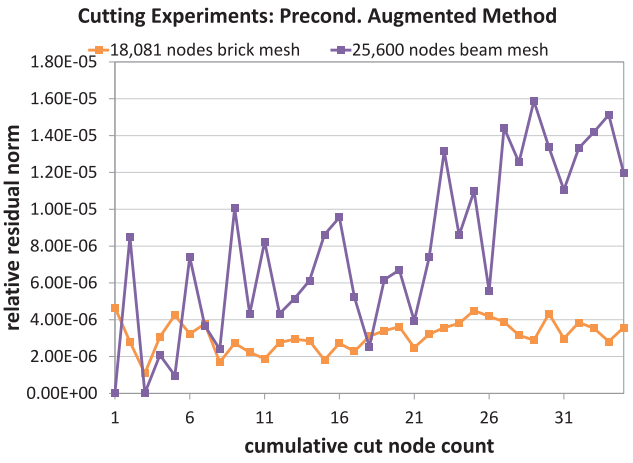


(b)

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.



(a)



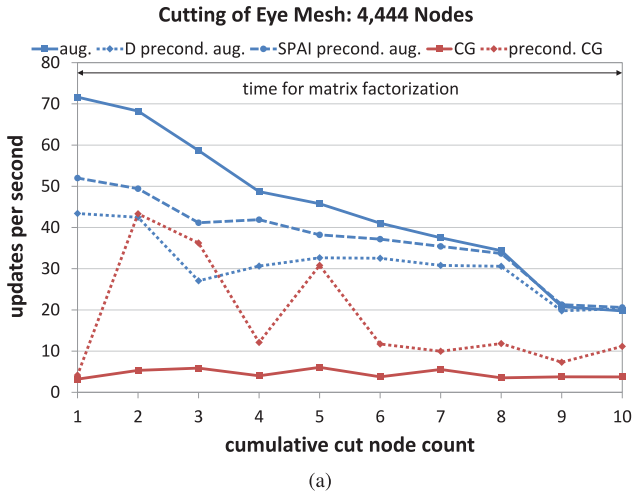
(b)

Fig. 11. (a) The maximum number of GMRES iterations required by the beam and brick meshes of a specific size. (b) Relative residual norm versus cut depth. For a solution $\hat{x}$ to the system $Ax = b$, relative residual norm is defined as $\|A\hat{x} - b\|_2/\|b\|_2$.

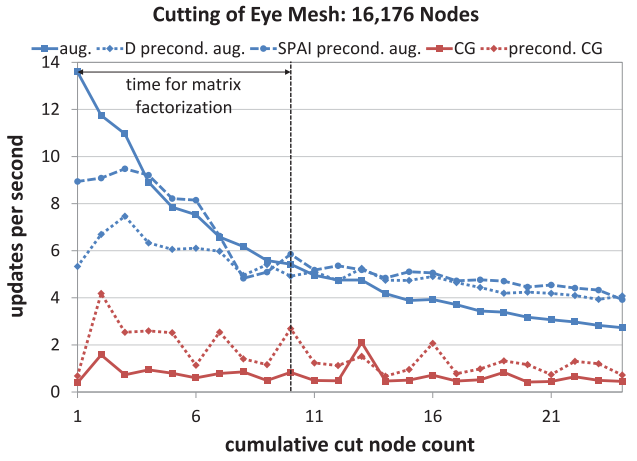
The differences between the results for the beam and brick meshes are even more pronounced for the cutting experiment than for the deformation experiment. Figure 7(a) shows that the augmented method outperformed CG in the beam-cutting experiments, providing updates in the range 49–145Hz in the time period before the 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.

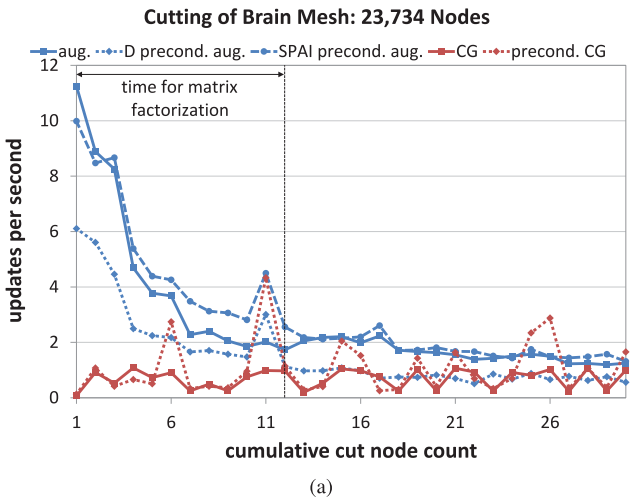


(a)

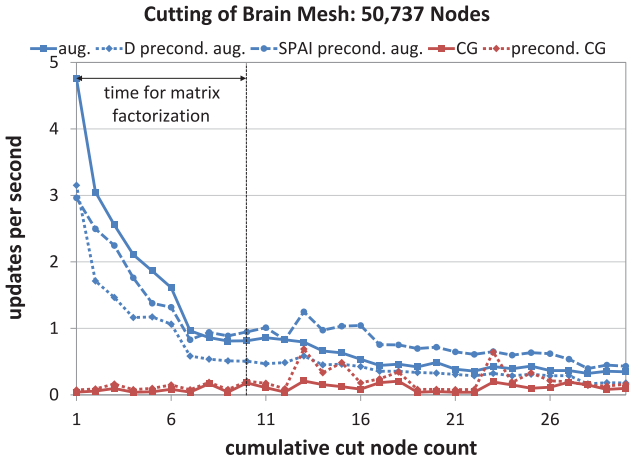


(b)

Fig. 12. Timing results are provided for the eye meshes of (a) 4,444 nodes and (b) 16,176 nodes.



(a)



(b)

Fig. 13. Timing results are provided for the brain meshes of (a) 23,734 nodes and (b) 50,737 nodes.

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 $1, \ldots, n$ 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, $\mathbb{W}$, of the directed graph of $G(A)$ such that $i \in \mathbb{W}$ if and only if $x_i \neq 0$ when solving $Ax = b$ or $Ay = x$. In this article, for a vector x, $\text{closure}_A(x)$ refers to $\text{closure}_A(\text{struct}(x))$.

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

$$\text{closure}_A(\mathbb{W}) := \bigcap \{\mathbb{U} : \mathbb{W} \subseteq \mathbb{U}, \text{ and } \mathbb{U} \text{ is closed}\},$$

which is the set of vertices of $G(A)$ from which there are directed paths in $G(A)$ to vertices in $\mathbb{W}$.

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*

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

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 $\hat{x}$. If A is nonsingular, then the set of components in b needed is* $\text{closure}_{A^\top}(\hat{x})$.

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

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

where B is $k \times k$. By the definition of $\text{closure}_{A^\top}(\hat{x})$, there is no edge (i, j) with $i \in \text{closure}_{A^\top}(\hat{x})$ and $j \notin \text{closure}_{A^\top}(\hat{x})$. 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 $\text{closure}_{A^\top}(\hat{x})$ is needed to compute the components in $\hat{x}$. □

THEOREM 3. *Let $A = LL^\top$ 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*

$$\text{closure}_L(\mathbb{W}) = \bigcup_{v \in \mathbb{W}} \{r \overset{T}{\Rightarrow} v\},$$

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

PROOF.

(i) $\bigcup_{v \in \mathbb{W}} \{r \Rightarrow^T v\} \subseteq \text{closure}_L(\mathbb{W})$:
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 $v \in \text{closure}_L(\mathbb{W})$, then $u \in \text{closure}_L(\mathbb{W})$. Since $\mathbb{W} \in \text{closure}_L(\mathbb{W})$, all ancestors of W must be in $\text{closure}_L(\mathbb{W})$.

(ii) $\text{closure}_L(\mathbb{W}) \subseteq \bigcup_{v \in \mathbb{W}} \{r \Rightarrow^T v\}$:
If a node $u \notin \bigcup_{v \in \mathbb{W}} \{r \Rightarrow^T v\}$, there must be a path from a node $w \in \bigcup_{v \in \mathbb{W}} \{r \Rightarrow^T v\}$ to u. Hence, there is also a directed path from w 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 $u \notin \text{closure}_L(\mathbb{W})$. □

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

Ugo Andreaus, Ivan Giorgio, and Angela Madeo. 2014. Modeling of the interaction between bone tissue and resorbable biomaterial as linear elastic materials with voids. *Zeitschrift für angewandte Mathematik und Physik* 66, 1, 209–237.

K. Bathe. 1996. *Finite Element Procedures*. Prentice-Hall, Upper Saddle River, NJ.

M. W. Benson and P. O. Federickson. 1982. Iterative solution of large sparse linear systems arising in certain multidimensional approximation problems. *Utilitas Mathematica* 22, 127–140.

J. Berkley, G. Turkiyyah, D. Berg, M. Ganter, and S. Weghorst. 2004. Real-time finite element modeling for surgery simulation: An application to virtual suturing. *IEEE Transactions on Visualization and Computer Graphics* 10, 3, 314–325. DOI:http://dx.doi.org/10.1109/TVCG.2004.1272730

J. Berkley, S. Weghorst, H. Gladstone, G. Raugi, D. Berg, and M. Ganter. 1999. Banded matrix approach to finite element modeling for soft tissue simulation. *Virtual Reality: Research, Development & Applications* 4, 203–212.

M. Bro-Nielsen. 1998. Finite element modeling in surgery simulation. *Proceedings of the IEEE* 86, 3, 490–503. DOI:http://dx.doi.org/10.1109/5.662874

Morten Bro-Nielsen and Stephane Cotin. 1996. Real-time volumetric deformable models for surgery simulation using finite elements and condensation. *Computer Graphics Forum* 15, 3, 57–66. DOI:http://dx.doi.org/10.1111/1467-8659.1530057

Yanqing Chen, Timothy A. Davis, William W. Hager, and Sivasankaran Rajamanickam. 2008. Algorithm 887: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate. *ACM Transactions on Mathematical Software* 35, 22:1–22:14.

Nuttapong Chentanez, Ron Alterovitz, Daniel Ritchie, Lita Cho, Kris K. Hauser, Ken Goldberg, Jonathan R. Shewchuk, and James F. O'Brien. 2009. Interactive simulation of surgical needle insertion and steering. *ACM Transactions on Graphics* 28, 3, Article 88, 10 pages. DOI:http://dx.doi.org/10.1145/1531326.1531394

Stephane Cotin, Herve Delingette, and Nicholas Ayache. 1999. Real-time elastic deformations of soft tissues for surgery simulation. *IEEE Transactions on Visualization and Computer Graphics* 5, 1, 62–73.

Stephane Cotin, Herve Delingette, and Nicholas Ayache. 2000. A hybrid elastic model for real-time cutting, deformations, and force feedback for surgery training and simulation. *The Visual Computer* 16, 7, 437–452.

Hadrien Courtecuisse, Jeremie Allard, Christian Duriez, and Stephane Cotin. 2010a. Asynchronous preconditioners for efficient solving of non-linear deformations. In *Proceedings of Virtual Reality Interaction and Physical Simulation*. 59–68.

Hadrien Courtecuisse, Hoeryong Jung, Jeremie Allard, Christian Duriez, Doo Yong Lee, and Stphane Cotin. 2010b. GPU-based real-time soft tissue deformation with cutting and haptic feedback. *Progress in Biophysics & Molecular Biology* 103, 23, 159–168. DOI:http://dx.doi.org/10.1016/j.pbiomolbio.2010.09.016

S. A. Cover, N. F. Ezquerra, J. F. O'Brien, R. Rowe, T. Gadacz, and E. Palm. 1993. Interactively deformable models for surgery simulation. *IEEE Computer Graphics and Applicattions* 13, 6, 68–75. DOI:http://dx.doi.org/10.1109/38.252559

J. R. Crouch and A. Cherry. 2007. Parametric eye models. In *Medicine meets virtual reality*, J. D. Westwood, R. S. Haluck, H. M. Hoffman, G. T. Mogel, R. Phillips, R. A. Robb, and K. G. Vosburgh (Eds.), Vol. 15. 91–93.

J. R. Crouch, S. M. Pizer, E. L. Chaney, Yu-Chi Hu, G. S. Mageras, and M. Zaider. 2007. Automated finite element analysis for deformable registration of prostate images. *IEEE Transactions on Medical Imaging* 26, 10, 1379–1390. DOI:http://dx.doi.org/10.1109/TMI.2007.898810

C. Dick, J. Georgii, and R. Westermann. 2011a. A hexahedral multigrid approach for simulating cuts in deformable objects. *IEEE Transactions on Visualization and Computer Graphics* 17, 11, 1663–1675. DOI:http://dx.doi.org/10.1109/TVCG.2010.268

Christian Dick, Joachim Georgii, and Rüdiger Westermann. 2011b. A real-time multigrid finite hexahedra method for elasticity simulation using CUDA. *Simulation Modelling Practice & Theory* 19, 2, 801–816.

S. E. DiMaio and S. P. Salcudean. 2003. Needle insertion modeling and simulation. *IEEE Transactions on Robotics and Automation* 19, 5, 864–875. DOI:http://dx.doi.org/10.1109/TRA.2003.817044

Simon P. DiMaio and S. E. Salcudean. 2002. Simulated interactive needle insertion. In *Proceedings of IEEE Symposium on Haptic Interfaces Virtual Environment Teleoperator Systems*, 344.

Florin Dobrian and Alex Pothen. 2006. Oblio: Design and performance. In *Applied Parallel Computing. State of the Art in Scientific Computing*, Jack Dongarra, Kaj Madsen, and Jerzy Wasniewski (Eds.). Lecture Notes in Computer Science, Vol. 3732. Springer, Berlin, 758–767. DOI:http://dx.doi.org/10.1007/11558958_92

C. Forest, Herve Delingette, and Nicholas Ayache. 2002. Cutting simulation of manifold volumetric meshes. In *Proceedings of International Conference on Medical Image Computing and Computer-Assisted Intervention, Part II*. Springer-Verlag, London, 235–244.

Andreas O. Frank, I. Alexander Twombly, Timothy J. Barth, and Jeffrey D. Smith. 2001. Finite element methods for real-time haptic feedback of soft-tissue models in virtual reality simulators. In *Proceedings of IEEE Virtual Reality*. 257.

Y. C. Fung. 1993. *Biomechanics: Mechanical Properties of Living Tissues*. Springer-Verlag.

Amit Gefen, Ran Shalom, David Elad, and Yossi Mandel. 2009. Biomechanical analysis of the keratoconic cornea. *Journal of the Mechanical Behavior of Biomedical Materials* 2, 3, 224–236.

Joachim Georgii and Rdiger Westermann. 2006. A multigrid framework for real-time simulation of deformable bodies. *Computer and Graphics* 30, 3, 408–415. DOI:http://dx.doi.org/10.1016/j.cag.2006.02.016

J. Gilbert. 1994. Predicting structure in sparse matrix computations. *SIAM Journal on Matrix Analysis and Applications* 15, 1, 62–79.

O. Goksel and S. E. Salcudean. 2011. Image-based variational meshing. *IEEE Transactions on Medical Imaging* 30, 1, 11–21. DOI:http://dx.doi.org/10.1109/TMI.2010.2055884

W. W. Hager. 1989. Updating the inverse of a matrix. *SIAM Review* 31, 2, 221–239. DOI:http://dx.doi.org/10.1137/1031049

Florian Hecht, Yeon Jin Lee, Jonathan R. Shewchuk, and James F. O'Brien. 2012. Updated sparse Cholesky factors for corotational elastodynamics. *ACM Transactions on Graphics* 31, 1–13.

P.-A. Heng, Chun-Yiu Cheng, Tien-Tsin Wong, Yangsheng Xu, Yim-Pan Chui, Kai-Ming Chan, and Shiu-Kit Tso. 2004. A virtual-reality training system for knee arthroscopic surgery. *IEEE Transactions on Information Technology in Biomedicine* 8, 2, 217–227. DOI:http://dx.doi.org/10.1109/TITB.2004.826720

Alex Jahya, Martijn G. Schouten, Jurgen J. Fütterer, and Sarthak Misra. 2014. On the importance of modelling organ geometry and boundary conditions for predicting three-dimensional prostate deformation. *Computer Methods in Biomechanics and Biomedical Engineering* 17, 5, 497–506.

Doug L. James and Dinesh K. Pai. 1999. ArtDefo: Accurate real time deformable objects. In *Proceedings of ACM SIGGRAPH*. 65–72. DOI:http://dx.doi.org/10.1145/311535.311542

Doug L. James and Dinesh K. Pai. 2003. Multiresolution Green's function methods for interactive simulation of large-scale elastostatic objects. *ACM Transactions on Graphics* 22, 47–82.

Grand Roman Joldes, Adam Wittek, and Karol Miller. 2009. Suite of finite element algorithms for accurate computation of soft tissue deformation for surgical simulation. *Medical Image Analysis* 13, 6, 912–919. DOI:http://dx.doi.org/10.1016/j.media.2008.12.001

Grand Roman Joldes, Adam Wittek, and Karol Miller. 2010. Real-time nonlinear finite element computations on GPU applications to neurosurgical simulation. *Computer Methods on Applied Mechanics and Engineering* 199, 4952, 3305–3314. DOI:http://dx.doi.org/10.1016/j.cma.2010.06.037

Mateusz Maria Juszczyk, Luca Cristofolini, and Marco Viceconti. 2011. The human proximal femur behaves linearly elastic up to failure under physiological loading conditions. *Journal of Biomechanics* 44, 12, 2259–2266.

Tony M. Keaveny, X. Edward Guo, Edward F. Wachtel, Thomas A. McMahon, and Wilson C. Hayes. 1994. Trabecular bone exhibits fully linear elastic behavior and yields at low strains. *Journal of Biomechanics* 27, 9, 1127–1136.

Umut Koçak, Karljohan Lundin Palmerius, and Matthew Cooper. 2009. Dynamic deformation using adaptable, linked asynchronous FEM regions. In *Proceedings of ACM Spring Conference on Computer Graphics*. 197–204. DOI:http://dx.doi.org/10.1145/1980462.1980500

R. J. Lapeer, P. D. Gasson, and V. Karri. 2010. Simulating plastic surgery: From human skin tensile tests, through hyperelastic finite element models to real-time haptics. *Progress in Biophysics & Molecular Biology* 103, 23, 208–216. DOI:http://dx.doi.org/10.1016/j.pbiomolbio.2010.09.013

C. Lederman, A. Joshi, I. Dinov, J. D. Van Horn, L. Vese, and A. Toga. 2010. Tetrahedral mesh generation for medical images with multiple regions using active surfaces. In *IEEE International Symposium on Biomedical Imaging: From Nano to Macro*. 436–439. DOI:http://dx.doi.org/10.1109/ISBI.2010.5490317

Bryan Lee, Dan C. Popescu, and Sebastien Ourselin. 2010. Topology modification for surgical simulation using precomputed finite element models based on linear elasticity. *Progress in Biophysics & Molecular Biology* 103, 23, 236–251. DOI:http://dx.doi.org/10.1016/j.pbiomolbio.2010.09.011

Alex Lindblad and George Turkiyyah. 2007. A physically-based framework for real-time haptic cutting and interaction with 3D continuum models. In *Proceedings of ACM Symposium on Solid and Physical Modeling*. 421–429. DOI:http://dx.doi.org/10.1145/1236246.1236307

Richard J. Lipton, Donald J. Rose, and Robert E. Tarjan. 1979. Generalized nested dissection. *SIAM Journal on Numerical Analysis* 16, 2, 346–358.

Joseph W. H. Liu. 1990. The role of elimination trees in sparse factorization. *SIAM Journal on Matrix Analysis and Applications* 11, 1, 134–172.

Stephanie Marchesseau, Tobias Heimann, Simon Chatelin, Remy Willinger, and Herv Delingette. 2010. Fast porous visco-hyperelastic soft tissue model for surgery simulation: Application to liver surgery. *Progress in Biophysics and Molecular Biology* 103, 23, 185–196. DOI:http://dx.doi.org/10.1016/j.pbiomolbio.2010.09.005

M. Mikielewicz, R. Michael, G. Montenegro, L. Pinilla Cortes, and R. I. Barraquer. 2013. Elastic properties of human lens zonules as a function of age in presbyopes. *Acta Ophthalmologica* 91, s252, 0–0.

A. Mohamed and C. Davatzikos. 2004. Finite element mesh generation and remeshing from segmented medical images. In *IEEE International Symposium on Biomedical Imaging: Nano to Macro*, Vol. 1. 420–423. DOI:http://dx.doi.org/10.1109/ISBI.2004.1398564

Andrew Mor and Takeo Kanade. 2000. Modifying soft tissue models: Progressive cutting with minimal new element creation. In *Medical Image Computing and Computer-Assisted Intervention*, Scott Delp, Anthony DiGoia, and Branislav Jaramaz (Eds.). Lecture Notes in Computer Science, Vol. 1935. Springer, Berlin, CH412–CH412.

N. Mos, J. Dolbow, and T. Belytschko. 1999. A finite element method for crack growth without remeshing. *International Journal for Numerical Methods in Engineering* 46, 1, 131–150.

Han-Wen Nienhuys and A. Frank van der Stappen. 2001. A surgery simulation supporting cuts and finite element deformation. In *Proceedings of the 4th International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer-Verlag, London, 145–152.

Igor Nikitin, Lialia Nikitina, Pavel Frolov, Gernot Goebbels, Martin Göbel, Stanislav Klimenko, and Gregory M. Nielson. 2002. Real-time simulation of elastic objects in virtual environments using finite element method and precomputed Green's functions. In *Proceedings of Eurographics Workshop on Virtual Environments*. 47–52.

NVIDIA. 2015. CUDA 7.0 Performance Report. http://developer.download.nvidia.com/compute/cuda/compute-docs/cuda-performance-report.pdf.

Guillaume Picinbono, Jean-Christophe Lombardo, Herv Delingette, and Nicholas Ayache. 2002. Improving realism of a surgery simulator: linear anisotropic elasticity, complex interactions and force extrapolation. *The Journal of Visualization and Computer Animation* 13, 3, 147–167. DOI:http://dx.doi.org/10.1002/vis.257

M. Sedef, E. Samur, and C. Basdogan. 2006. Real-time finite-element simulation of linear viscoelastic tissue behavior based on experimental data. *IEEE Computer Graphics and Applications* 26, 6, 58–68. DOI:http://dx.doi.org/10.1109/MCG.2006.135

Guy Sela, Jacob Subag, Alex Lindblad, Dan Albocher, Sagi Schein, and Gershon Elber. 2007. Real-time haptic incision simulation using FEM-based discontinuous free-form deformation. *Computer Aided Design* 39, 8, 685–693. DOI:http://dx.doi.org/10.1016/j.cad.2007.05.011

D. Serby, Matthias Harders, and Gábor Székely. 2001. A new approach to cutting into finite element models. In *Proceedings of the 4th International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer-Verlag, London, 425–433.

J. Spillmann and M. Harders. 2012. Robust interactive collision handling between tools and thin volumetric objects. *IEEE Transactions on Visualization and Computer Graphics* 18, 8, 1241–1254. DOI:http://dx.doi.org/10.1109/TVCG.2011.151

D. Steinemann, M. Harders, M. Gross, and G. Szekely. 2006. Hybrid cutting of deformable solids. In *IEEE Virtual Reality Conference*. 35–42. DOI:http://dx.doi.org/10.1109/VR.2006.74

Demetri Terzopoulos and Kurt Fleischer. 1988. Modeling inelastic deformation: Viscoelasticity, plasticity, fracture. *SIGGRAPH Computer Graphics* 22, 4, 269–278. DOI:http://dx.doi.org/10.1145/378456.378522

Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. 1987. Elastically deformable models. *SIGGRAPH Computer Graphics* 21, 4, 205–214. DOI:http://dx.doi.org/10.1145/37402.37427

M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M.-P. Cani, F. Faure, N. Magnenat-Thalmann, W. Strasser, and P. Volino. 2005. Collision detection for deformable objects. *Computer Graphics Forum* 24, 1, 61–81. DOI:http://dx.doi.org/10.1111/j.1467-8659.2005.00829.x

Greg Turk and Marc Levoy. 1994. Zippered polygon meshes from range images. In *Proceedings of ACM SIGGRAPH*. 311–318. DOI:http://dx.doi.org/10.1145/192161.192241

George M. Turkiyyah, Wajih Bou Karam, Zeina Ajami, and Ahmad Nasri. 2011. Mesh cutting during real-time physical simulation. *Computer Aided Design* 43, 7, 809–819. DOI:http://dx.doi.org/10.1016/j.cad.2010.10.005

Lara M. Vigneron, Jacques G. Verly, and Simon K. Warfield. 2004. Modelling surgical cuts, retractions, and resections via extended finite element method. In *Proceedings of the 7th International Conference on Medical Image Computing and Computer-Assisted Intervention, Part II*. Christian Barillot, David R. Haynor, and Pierre Hellier (Eds.). Lecture Notes in Computer Science, Vol. 3217. Springer, Berlin, 311–318.

Adam Wittek, Grand Joldes, Mathieu Couton, Simon K. Warfield, and Karol Miller. 2010. Patient-specific non-linear finite element modelling for predicting soft organ deformation in real-time; Application to non-rigid neuroimage registration. *Progress in Biophysics and Molecular Biology* 103, 23, 292–303. DOI:http://dx.doi.org/10.1016/j.pbiomolbio.2010.09.001

Wen Wu and Pheng Ann Heng. 2004. A hybrid condensed finite element model with GPU acceleration for interactive 3D soft tissue cutting:

Research Articles. *Computer Animation and Virtual Worlds* 15, 3–4, 219–227. DOI:http://dx.doi.org/10.1002/cav.v15:3/4

Wen Wu and Pheng Ann Heng. 2005. An improved scheme of an interactive finite element model for 3D soft-tissue cutting and deformation. *The Visual Computer* 21, 8, 707–716. DOI:http://dx.doi.org/10.1007/s00371-005-0310-6

Xunlei Wu, Michael S. Downes, Tolga Goktekin, and Frank Tendick. 2001. Adaptive nonlinear finite elements for deformable body simulation using dynamic progressive meshes. In *Proceedings of Eurographics*, A. Chalmers and T.-M. Rhyne (Eds.). Vol. 20(3). Blackwell Publishing, 349–358.

Xunlei Wu and Frank Tendick. 2004. Multigrid integration for interactive deformable body simulation. In *Medical Simulation*, Stephane Cotin and Dimitris Metaxas (Eds.). Lecture Notes in Computer Science, Vol. 3078. Springer, Berlin, 92–104.

Xinyu Zhang and Y. J. Kim. 2012. Simple culling methods for continuous collision detection of deforming triangles. *IEEE Transactions on Visualization and Computer Graphics* 18, 7, 1146–1155. DOI:http://dx.doi.org/10.1109/TVCG.2011.120

Hualiang Zhong, Mark P. Wachowiak, and Terry M. Peters. 2005. Adaptive finite element technique for cutting in surgical simulation. *Medical Imaging 2005: Visualization, Image-Guided Procedures, and Display* 5744, 1, 604–611. DOI:http://dx.doi.org/10.1117/12.594379

Yongning Zhu, Eftychios Sifakis, Joseph Teran, and Achi Brandt. 2010. An efficient multigrid method for the simulation of high-resolution elastic solids. *ACM Transactions on Graphics* 29, 2, Article 16, 18 pages. DOI:http://dx.doi.org/10.1145/1731047.1731054

Received April 2014; revised September 2015; accepted December 2015