AMPS: REAL-TIME MESH CUTTING WITH AUGMENTED MATRICES FOR SURGICAL SIMULATIONS[*]

YU-HONG YEUNG[†], ALEX POTHEN[†], AND JESSICA CROUCH[‡]

Abstract. We present the augmented matrix for principal submatrix update (AMPS) algorithm, a finite element solution method that combines principal submatrix updates and Schur complement techniques, well-suited for interactive simulations of deformation and cutting of finite element meshes. Our approach features real-time solutions to the updated stiffness matrix systems to account for interactive changes in mesh connectivity and boundary conditions. Updates are accomplished by an augmented matrix formulation of the stiffness equations to maintain its consistency with changes to the underlying model without refactorization at each timestep. As changes accumulate over multiple simulation timesteps, the augmented solution algorithm enables tens or hundreds of updates per second. Acceleration schemes that exploit sparsity, memoization and parallelization lead to the updates being computed in real-time. The complexity analysis and experimental results for this method demonstrate that it scales linearly with the number of nonzeros of the factors of the stiffness matrix. Results for cutting and deformation of 3D elastic models are reported for meshes with up to 50,000 nodes, and involve models of surgery for astigmatism and the brain.

Key words. finite element, surgery simulation, real-time, deformable model, cutting

AMS subject classifications. 65F50, 65F10, 65F05, 65Y20

1. Introduction.

We present an algorithm that enables us to deform and cut solid finite element models in real time by solving the resulting time-varying equations fast. Modifications of the mesh topology and changes in the boundary conditions are the basic operations of many simulation scenarios, particularly surgical simulations. A real-time finite element solution method for mesh cutting is computationally demanding, first because graphic and haptic rendering require accurate solutions at real-time update rates, and second because connectivity changes due to cutting and remeshing modify the underlying matrix equations. Such modifications invalidate previous factorizations or inverse computations for the stiffness matrix, requiring either computationally expensive update procedures or a solution via an iterative method.

Interactive simulations often involve unpredictable cutting paths to allow flexibility to the user inputs. This feature requires that the internal deformation of a solid model be computed and tracked so that accurate cut surfaces are exposed as cuts progress into the potentially heterogeneous interior of a model. The cutting of a 3D mesh results in pushing and pulling forces being applied to the nodes, and new Dirichlet boundary conditions being imposed on the nodes by different fixation scenarios. Consequently an algorithm to compute the displacements of all nodes under these changes in real time is essential to make the simulations practical.

Observing that the aforementioned changes to the meshes result in a principal submatrix update and a change in dimensions to the underlying equations, we propose a new solution approach to reflect both the update and the dimension change in a modified augmented matrix formulation. This approach, called Augmented Matrix for Principal Submatrix update (AMPS), is similar to other augmented matrix methods in that the matrix is represented in a block matrix form in which the (1,1) block

[*]This work was supported in part by NSF grant CCF-1637534; the U.S. Department of Energy through grant DE-FG02-13ER26135; and the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the DOE Office of Science and the NNSA.

[†]Department of Computer Science, Purdue University, West Lafayette, IN (yyeung@purdue.edu, apothen@purdue.edu).

[‡]Department of Computer Science, Old Dominion University, Norfolk, VA (jrcrouch@cs.odu.edu).

is the fixed original matrix and the other blocks are either zero or vary according to the changes. The Schur complement operation is then applied to decouple the augmentation from the remaining part of the system, and the Schur complement system is solved in two phases. Our current solution combines a one-time sparse matrix-factorization for the (1,1) block with an explicit computation of a principal submatrix of the inverse of the original matrix and a direct solution of the Schur complement system. Sparsity in the matrix, solution vector, and the right-hand-side vector are carefully exploited throughout the computations and intermediate results are stored for subsequent changes in later cutting steps. The time complexity of the algorithm shows that performance scales well with model size and various cutting lengths, while supporting arbitrary cutting of any valid finite element mesh.

Different algorithms for mesh generation [5, 10, 15, 18], collision detection [20, 23, 26], and mesh refinement [8, 19, 21] can be paired with our solution algorithm to produce a complete simulation platform. The scope of this paper does not include algorithms for simulation tasks other than solving the finite element system of equations. 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.

The three main contributions of this work are:

- An augmented matrix formulation of the stiffness system of equations from a finite element model, specific for principal submatrix updates and dimension changes resulting from 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 re-factorization whenever a change occurs.
- A direct solution approach that provides fast and accurate solutions to both the updated portion and unchanged portion, when the percentage of mesh elements affected by topological changes is small.
- Acceleration of the solution method by exploiting sparsity, memoization and parallelization. We analyze the time complexity of the accelerated solution method using concepts from graph theory.

This paper 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 with principal submatrix update 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 size. Finally, Section 5 discusses conclusions and directions for future work.

2. Previous Work.

The augmented matrix algorithm described in this paper is related to earlier algorithms designed by us and our colleagues in [24, 25]. In the first paper, we formed an augmented system to replace columns in the original matrix, and solved the Schur complement system using GMRES implicitly and the rest of the system directly using precomputed $LDL^\top$ factors of the original matrix. Unfortunately the symmetry present in the system was destroyed during this method for updating the solution, and two closures needed to be computed to exploit the sparsities in both the matrix and the right-hand-side vector. The convergence of the iterative solver depended on the condition of the Schur complement of the system, and a preconditioner was sometimes needed for faster convergence. However, since the Schur complement was not computed explicitly, it was difficult to find an effective preconditioner fast enough to provide real-time updates.

We overcome these shortcomings by following an approach similar to that presented in the second paper [25]. By observing that the only change to the original matrix is within a principal submatrix, with our co-authors we showed that symmetry could be preserved during the update of the solution. We presented two approaches to solve the Schur complement system, an iterative method and a direct method. However, the application to contingency analysis of power grids considered there retained the size of the system for any contingency scenario. Thus the augmented system considered there was for applications in which the matrix update did not change the dimension of the matrix. This is not the case with surgical simulations, in which new vertices are added to the mesh along the cutting surface. The additional vertices increase the overall dimension of the modified system. Therefore an extension is presented in this paper to generalize the augmented matrix approach to systems where their dimensions change. We also improve the computation of the principal submatrix of the matrix inverse to further accelerate the solution.

In [25], CHOLMOD [6], an algorithm to update or downdate the Cholesky factor of the matrix with low-rank matrices, was compared to our augmented matrix formulation. It was shown that our approach outperformed CHOLMOD for the power contigency application. However, SuiteSparse, the software package that includes CHOLMOD, does not provide functionality to increase the dimension of the modified system. Therefore, we cannot compare our method with CHOLMOD for the surgical simulation application in this paper.

Other related papers were surveyed in the two aforementioned papers and hence we do not repeat them here. Specifically, in [25] we have provided a summary of the state of the art in algorithms that update the solution in surgical simulations.

Linear elastic models are useful in biomechanical contexts when the deformations are small and limited forces are applied, e.g., eye surgery for astigmatism. Here we list a few more papers that employ linear elastic models for surgery and for the study of biomechanical properties. Chabanas et al. [3] examined the applicability of a linear elastic model for maxillofacial surgery, compared it to other models, and concluded that the linear model is appropriate for some types of surgery simulations. Liu et al. [16] used a linear elastic model for soft tissues during minimally invasive surgery. Using data collected by their probe during surgery, they used the model to estimate the elastic modulus of different areas of tissue and demonstrated the ability to detect firm tumors. Crouch et al. [5] modeled prostate brachytherapy using a linear elastic material model. Andreaus et al. [1] have modeled the interaction between bone tissue and resorbable biomaterial as linear elastic materials with voids. Juszczyk et al. [12] have shown that the femur can be modeled under physiological loading conditions using linear elasticity.

3. Methods.

Navier's equation of equilibrium, which describes the deformation of an isotropic, homogeneous elastostatic body, is

$$f + \mu \nabla^2 a + (\lambda + \mu)\, \nabla\nabla \cdot a = 0. \tag{3.1}$$

In this equation f represents applied force, a represents displacement, and λ and μ are elastic moduli constants that describe material properties [14]. They are related to the Young's modulus E and the the Poisson ratio ν by the equations $\mu = E/\left(2(1+\nu)\right)$, and $\lambda = \nu E/\left((1+\nu)(1-2\nu)\right)$. We have used $E = 10$ kPa and $\nu = 0.3$. The results presented in this paper were generated for finite element models that represent approximate solutions to this equation. Our computational method also applies when the material model is anisotropic or inhomogeneous. The relationship between the strong form of Navier's equation, shown in Equation 3.1, and the weak form of the equation is reviewed in [22], and the matrix equation of the finite element model is derived in [2].

The system of stiffness equations associated with a finite element model of an organ or tissue is

$$K_0 a_0 = f_0, \tag{3.2}$$

where the stiffness matrix K_0 has dimension n. The finite element mesh then undergoes a series of discrete cutting steps, and the mesh and the associated stiffness equations are updated after each cutting step. New nodes and elements might be inserted or some nodes and elements deleted at each step. At step t, we have the system

$$K_t a_t = f_t, \tag{3.3}$$

where K_t is now of order $(n+d_t)\times(n+d_t)$, and the displacement vector a_t and the force vector f_t are of order $(n+d_t)$. By considering the difference between the original $n\times n$ stiffness matrix K_0 and the modified stiffness matrix K_t after cutting at step t, we observe that K_t can be expressed as the result of a principal submatrix update to K_0 augmented by an identity matrix of size d_t:

$$K_t = \underbrace{\begin{bmatrix} K_0 & \\ & I_{d_t}\end{bmatrix}}_{\bar{K}_t} - \underbrace{\begin{bmatrix} H_t & \\ & I_{d_t}\end{bmatrix}}_{\bar{H}_t} \underbrace{\left(E_t + \begin{bmatrix} 0_{m_t} & \\ & I_{d_t}\end{bmatrix}\right)}_{\bar{E}_t} \underbrace{\begin{bmatrix} H_t^\top & \\ & I_{d_t}\end{bmatrix}}_{\bar{H}_t^\top}, \tag{3.4}$$

where H_t comprises the m_t columns of the identity matrix of order n whose indices correspond to the rows and columns of K_0 to be updated; and E_t is an $(m_t+d_t)\times(m_t+d_t)$ principal submatrix update to $\bar{K}_t$. Here H_t has dimension $n\times m_t$, $\bar{H}_t$ has dimension $(n+d_t)\times(m_t+d_t)$; and $\bar{E}_t$ has dimension $(m_t+d_t)\times(m_t+d_t)$, the same as that of E_t. Note that $\bar{H}_t^\top \bar{H}_t = I_{(m_t+d_t)}$. Here, m_t is the number of its rows and columns replaced at step t, and d_t is the change in dimension of the modified matrix at step t. In the context of the finite element model used in the surgical simulation, m_t corresponds to the degrees of freedom (DOFs) of the modified vertices and their neighbors, and d_t corresponds to the DOFs of the newly added vertices with respect to the original system. In general, $m_t \gg d_t$. An example is shown in Figure 1.

For the sake of simplicity, the subscripts t are dropped hereafter throughout the paper except in Section 3.3 where the subscript t is necessary.

We can express a as the sum of two independent terms:

$$a = a^{(1)} + \bar{H}a^{(2)}, \tag{3.5}$$

where $a^{(1)}$ is an $(n+d)$-vector and $a^{(2)}$ is an $(m+d)$-vector such that

$$\bar{H}^\top a^{(1)} = 0; \tag{3.6}$$

it follows that $\bar{H}^\top a = a^{(2)}$. Substituting Equation 3.5 into Equation 3.3, we have

$$K a^{(1)} + K\bar{H}a^{(2)} = f. \tag{3.7}$$

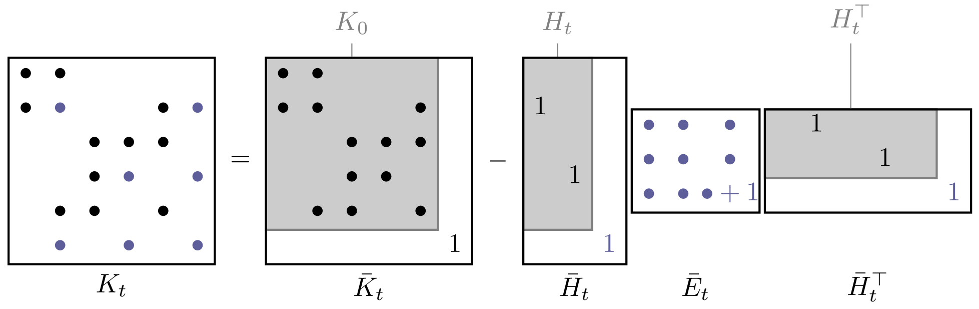


FIG. 1. *Example of a modified* (6×6)*-matrix* K_t *formed by a principal submatrix update* E_t *of size* 3×3 *colored in blue to the original* (5×5)*-matrix* K_0 *with dimension change. Here,* $n = 5$*,* $m_t = 2$ *and* $d_t = 1$*.*

Substituting Equation 3.4 into Equation 3.7, we have

$$\bar{K}a^{(1)} - \bar{H}\bar{E}\bar{H}^\top a^{(1)} + \left(\bar{K}\bar{H} - \bar{H}\bar{E}\bar{H}^\top\bar{H}\right)a^{(2)} = f,$$

$$\bar{K}a^{(1)} + \left(\bar{K}\bar{H} - \bar{H}\bar{E}\right)a^{(2)} = f. \tag{3.8}$$

We have made use of Equation 3.6 and the orthogonality of $\bar{H}$ to obtain the second equation from the first. If we denote $a^{(3)} = -\bar{E}a^{(2)}$, then Equation 3.8 becomes

$$\bar{K}a^{(1)} + \bar{K}\bar{H}a^{(2)} + \bar{H}a^{(3)} = f. \tag{3.9}$$

Premultiplying Equation 3.8 by $\bar{H}^\top$ yields

$$\bar{H}^\top\bar{K}a^{(1)} + \left(\bar{H}^\top\bar{K}\bar{H} - \bar{E}\right)a^{(2)} = \bar{H}^\top f. \tag{3.10}$$

Assembling Equations 3.6, 3.9 and 3.10, we can form an augmented system of equations

$$\begin{bmatrix} \bar{K} & \bar{K}\bar{H} & \bar{H} \\ \bar{H}^\top\bar{K} & \bar{H}^\top\bar{K}\bar{H} - \bar{E} & 0 \\ \bar{H}^\top & 0 & 0 \end{bmatrix} \begin{bmatrix} a^{(1)} \\ a^{(2)} \\ a^{(3)} \end{bmatrix} = \begin{bmatrix} f \\ \bar{H}^\top f \\ 0 \end{bmatrix}. \tag{3.11}$$

Using $\bar{K}$ as the block pivot, Equation 3.11 can be reduced to a smaller system involving the Schur complement of $\bar{K}$, S_1. After multiplying that system with -1 we obtain:

$$\underbrace{\begin{bmatrix} \bar{E} & I \\ I & \bar{H}^\top\bar{K}^{-1}\bar{H} \end{bmatrix}}_{S_1} \begin{bmatrix} a^{(2)} \\ a^{(3)} \end{bmatrix} = \begin{bmatrix} 0 \\ \bar{H}^\top\bar{K}^{-1}f \end{bmatrix}, \tag{3.12}$$

in which

$$\bar{K}^{-1} = \begin{bmatrix} K_0^{-1} & \\ & I_d \end{bmatrix}. \tag{3.13}$$

Note that the matrix S_1 is symmetric. Equation 3.12 can be further reduced with a second Schur complement using the $(1,2)$-block of S_1 as the block pivot:

$$\underbrace{\left(I - \bar{H}^\top\bar{K}^{-1}\bar{H}\bar{E}\right)}_{S_2} a^{(2)} = \bar{H}^\top\bar{K}^{-1}f. \tag{3.14}$$

Note that the matrix S_2 is not symmetric. If f only differs from f_0 at the newly added vertices, i.e.,

$$f - \begin{bmatrix} f_0 \\ 0_d \end{bmatrix} = \begin{bmatrix} 0_n \\ \bar{f} \end{bmatrix}, \tag{3.15}$$

then the right-hand-side vector of Equation 3.14 can be simplified to

$$\bar{H}^\top \bar{K}^{-1} f = \begin{bmatrix} H^\top a_0 \\ \bar{f} \end{bmatrix}, \tag{3.16}$$

where a_0 is the solution to the original system 3.2 and $\bar{f}$ is the force applied to the d newly added vertices.

After solving Equation 3.14 for $a^{(2)}$ using a direct solver, we can solve for a in the modified system 3.3 directly using the following observation. Premultiplying Equation 3.8 by $\bar{K}^{-1}$ and rearranging the terms yields

$$a^{(1)} = \bar{K}^{-1} f + \left(\bar{K}^{-1} \bar{H} \bar{E} - \bar{H} \right) a^{(2)}. \tag{3.17}$$

Again, if f satisfies the condition of Equation 3.15, Equation 3.17 can be simplified to

$$a^{(1)} = \begin{bmatrix} a_0 \\ \bar{f} \end{bmatrix} + \left(\bar{K}^{-1} \bar{H} \bar{E} - \bar{H} \right) a^{(2)}. \tag{3.18}$$

Substituting Equation 3.18 into Equation 3.5 yields

$$a = \begin{bmatrix} a_0 \\ \bar{f} \end{bmatrix} + \bar{K}^{-1} \bar{H} \bar{E} a^{(2)}, \tag{3.19}$$

thus completing the solution.

An alternative Schur complement formulation is possible. One can use the $(2,1)$-block in Equation 3.12 as the block pivot for the Schur complement and get

$$\begin{aligned} \left(\bar{E} \bar{H}^\top \bar{K}^{-1} \bar{H} - I \right) a^{(3)} &= \bar{E} \bar{H}^\top \bar{K}^{-1} f. \\ &= \begin{bmatrix} 0 \\ \bar{f} \end{bmatrix} + \begin{bmatrix} E_{11} \\ E_{12}^\top \end{bmatrix} H^\top a_0, \end{aligned} \tag{3.20}$$

assuming that the condition in Equation 3.15 is satisfied. Again the coefficient matrix is not symmetric. After solving for $a^{(3)}$ using Equation 3.20, the solution a can be obtained as follows:

$$a = \begin{bmatrix} a_0 \\ \bar{f} \end{bmatrix} - \bar{K}^{-1} \bar{H} a^{(3)}. \tag{3.21}$$

3.1. Improving numerical accuracy.

We can improve the numerical accuracy of the solutions by avoiding numerical errors as follows. From the third row block of Equation 3.11, we have

$$\bar{H}^\top a^{(1)} = 0. \tag{3.22}$$

Premultiplying Equation 3.5 by $\bar{H}^\top$ yields

$$\bar{H}^\top a = \bar{H}^\top a^{(1)} + \bar{H}^\top \bar{H} a^{(2)} = a^{(2)}. \tag{3.23}$$

Note that the components of a picked out by $\bar{H}^\top$ correspond to $a^{(2)}$, which are arithmetically identical to the same components computed using Equation 3.19 but with higher accuracy. If we denote $\mathcal{H}$ as the set of indices for which the rows and columns of K_0 are updated including the newly added ones, combining the two equations, we have

$$a[i] = \begin{cases} \left(\bar{H}a^{(2)}\right)[i] & \text{for } i \in \mathcal{H}, \\ \left(\begin{bmatrix} a_0 \\ \bar{f} \end{bmatrix} + \bar{K}^{-1}\bar{H}\bar{E}a^{(2)}\right)[i] & \text{for } i \notin \mathcal{H}. \end{cases} \tag{3.24}$$

Skipping the computations of those components in a that are in $\mathcal{H}$ also improves the performance of the algorithm.

3.2. Computing the Schur Complement Matrix. Our augmented algorithm involves solving Equations 3.14 and 3.24. Unlike [24] both equations are solved using a direct solver. The Schur complement matrix S_2 in Equation 3.14 can be expressed in block matrix form using Equations 3.4, 3.13 and 3.16 to obtain

$$\left(\begin{bmatrix} I_m & \\ & 0_d \end{bmatrix} - \begin{bmatrix} H^\top K_0^{-1} H & \\ & I_d \end{bmatrix} E\right) a^{(2)} = \begin{bmatrix} H^\top a_0 \\ \bar{f} \end{bmatrix}. \tag{3.25}$$

Solving Equation 3.25 involves computing the principal submatrix of the inverse $H^\top K_0^{-1} H$. Assuming that $K_0 = L_0 D_0 L_0^\top$ is a factorization of K, we have $H^\top K_0^{-1} H = H^\top L_0^{-\top} D_0^{-1} L_0^{-1} H$. If we denote $V \equiv L_0^{-1} H$, then $H^\top K_0^{-1} H = V^\top D^{-1} V$, which can be computed by first solving for V using forward substitution, then scaling V to obtain $U \equiv D_0^{-1} V$ and finally premultiplying U by $V^\top$. The computation of the rest of the matrix in Equation 3.25 is straightforward.

3.3. Memoization. For an efficient computation of the principal submatrix of the inverse $H_t^\top K_0^{-1} H_t$ at step t, we observe that since the vertices removed during the cutting are accumulating and H is the submatrix of the identity corresponding to the replaced rows and columns in K_0, the matrix H_{t-1} at the previous step $t-1$ is a submatrix of the first m_{t-1} columns of matrix H_t at step t, i.e.,

$$H_t = \left[\begin{array}{c|c} H_{t-1} & H_{\Delta t} \end{array}\right], \tag{3.26}$$

where $H_{\Delta t}$ is the $(m_t - m_{t-1})$ columns of the identity matrix corresponding to the newly removed columns at step t. Consequently, the matrix V_{t-1} is also the first m_{t-1} columns of V_t since each column of V_t is independently solved, i.e.,

$$V_t = \left[\begin{array}{c|c} V_{t-1} & V_{\Delta t} \end{array}\right], \tag{3.27}$$

where $V_{\Delta t} = L_0^{-1} H_{\Delta t}$, which are the only columns of V_t that need to be computed. Furthermore, the top-left $(m_{t-1} \times m_{t-1})$ submatrix of $H_t^\top K_0^{-1} H_t$ is identical to

$H_{t-1}^\top K_0^{-1} H_{t-1}$ because

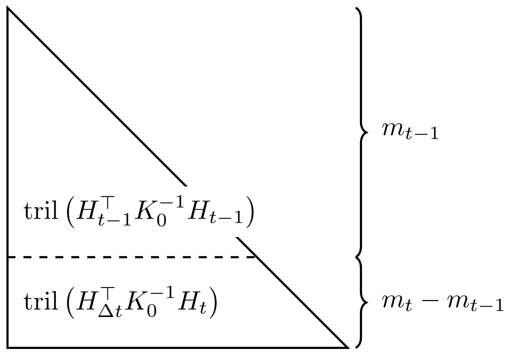
$$H_t^\top K_0^{-1} H_t = V_t^\top D_0^{-1} V_t = \left[\begin{array}{c} V_{t-1}^\top \\ \hline V_{\Delta t}^\top \end{array}\right] D_0^{-1} \left[\begin{array}{c|c} V_{t-1} & V_{\Delta t} \end{array}\right]$$

$$= \left[\begin{array}{c|c} V_{t-1}^\top D_0^{-1} V_{t-1} & V_{t-1}^\top D_0^{-1} V_{\Delta t} \\ \hline V_{\Delta t}^\top D_0^{-1} V_{t-1} & V_{\Delta t}^\top D_0^{-1} V_{\Delta t} \end{array}\right]$$

$$= \left[\begin{array}{c|c} H_{t-1}^\top K_0^{-1} H_{t-1} & V_{t-1}^\top D_0^{-1} V_{\Delta t} \\ \hline V_{\Delta t}^\top D_0^{-1} V_{t-1} & V_{\Delta t}^\top D_0^{-1} V_{\Delta t} \end{array}\right]. \tag{3.28}$$

Furthermore, it can be observed from Equation 3.28 that $H_t^\top K_0^{-1} H_t$ is also symmetric and only the lower or upper triangular part needs to be computed and stored, and subsequent updates can be done sequentially by trapezoidal augmentations to $\mathrm{tril}\left(H_{t-1}^\top K_0^{-1} H_{t-1}\right)$:

(3.29) $\mathrm{tril}\left(H_t^\top K_0^{-1} H_t\right) =$


$\mathrm{tril}\left(H_{t-1}^\top K_0^{-1} H_{t-1}\right)$
$\mathrm{tril}\left(H_{\Delta t}^\top K_0^{-1} H_t\right)$
m_{t-1}
$m_t - m_{t-1}$

,

where $\mathrm{tril}\left(\bullet\right)$ is the lower triangular part of the matrix and the augmentation part, $\mathrm{tril}\left(H_{\Delta t}^\top K_0^{-1} H_t\right)$, can be computed as

$$\mathrm{tril}\left(H_{\Delta t}^\top K_0^{-1} H_t\right)[i,j] = \left(V_{\Delta t}[i,*]\right)^\top D_0^{-1}\left(V_t[j,*]\right) \tag{3.30}$$
$$\text{for } i \in [m_{t-1}+1, m_t]; j \in [1, i].$$

It is obvious that Equation 3.30 can be computed in parallel for all i's and j's since they are independent of each other.

3.4. Dimension Shrinking.

In the case of the imposition of Dirichlet boundary conditions, the dimension of the system is shrunk instead of expanded, unlike the case of cutting. The authors in [24] have shown that an augmented matrix system similar to Equation 3.11 is equivalent to the modified system of equations:

$$\begin{bmatrix} K_0 & H \\ H^\top & 0 \end{bmatrix} \begin{bmatrix} a^{(1)} \\ a^{(2)} \end{bmatrix} = \begin{bmatrix} f \\ 0 \end{bmatrix}, \tag{3.31}$$

where H is the matrix whose column j correspond to the indicator vector of the j-th Dirichlet degree of freedom, $a^{(2)} = -H^\top f_0$ is the newly unknown force and f is given by

$$f[i] = \begin{cases} f_0[i] - \sum_{j \in \mathcal{H}} K_0[i,j] a_0[j] & j \notin \mathcal{H}, \\ -\sum_{j \in \mathcal{H}} K_0[i,j] a_0[j] & j \in \mathcal{H}. \end{cases} \tag{3.32}$$

Similar to Equation 3.11, we can reduce Equation 3.31 to a smaller system using K_0 as the pivot:

$$H^\top K_0^{-1} H a^{(2)} = H^\top K_0^{-1} f. \tag{3.33}$$

Note that the matrix on the left-hand side is the principal submatrix of the inverse $HK_0^{-1}H^\top$, which can be efficiently computed as described in previous subsections. The right-hand side can be computed using $V \equiv L_0^{-1}H$ as

$$H^\top K_0^{-1} f = V^\top \underbrace{D_0^{-1} L_0^{-1} f}_{g}. \tag{3.34}$$

After computing $a^{(2)}$, $a^{(1)}$ can be computed using the first row block of Equation 3.31 as

$$\begin{aligned} a^{(1)} &= K_0^{-1}\left(f - H^\top a^{(2)}\right) \\ &= L_0^{-\top} D_0^{-1} L_0^{-1}\left(f - H^\top a^{(2)}\right) \\ &= L_0^{-\top}\left(g - D_0^{-1} V a^{(2)}\right), \end{aligned} \tag{3.35}$$

in which g is already computed in Equation 3.34 and can be reused.

3.5. Complexity Analysis. The time complexity of principal submatrix updates using the symmetric augmented formulation can be summarized in Table 1. Both per cut and total update times are provided. The one-time factorization costs assume meshes with good separators for both 2D and 3D-meshes, of size $O(n^{1/2})$ and $O(n^{2/3})$, respectively. In the table, variables with subscript t are the values at step t, those with subscript Δt are the newly added values at step t, whereas their hatted counterparts are their maxima over all t. Recall that n is the size of the original matrix K_0, m is the size of the principal submatrix update H, d is the dimension change. In addition, $\mathcal{H}$ is the set of indices of the nonzero rows of H, $|L_0|$ is the number of nonzeros in L_0, and $v = \max_j |V_{*,j}|$ is the maximum number of nonzeros in any column of V, which is equivalent to the maximum closure size of any vertex in $\mathcal{H}$ in the graph of $G(L_0)$. For a detailed discussion on the concepts of closure and the relations between sparse matrix computations and its corresponding graph, we refer the readers to [24]. The authors have also included there the theorems used to prove the upper bounds of the complexity of the AMPS algorithms.

The overall time complexity of the algorithm is dominated by either Step 2 (computing $\operatorname{tril}\left(H_{\Delta t}^\top K_0^{-1} H_t\right)$) or Step 6 (solving for a). The update steps in the AMPS algorithm have an overall time complexity of

$$O\left(\hat{m}^2 \hat{v} + c \cdot |L_0|\right). \tag{3.36}$$

Note that the second term of the time complexity comes from the final triangular solve, which is common among all direct methods. Moreover, our algorithm only requires one triangular solve, compared to two triangular solves in a typical direct solver.

3.6. Parallelization. We can observe that Steps 1–4 in the update steps in Table 1 are easily parallelizable from the facts that in Step 1 each columns of $V_{\Delta t}$ are independently solved, both Steps 2 and 3 involve matrix-matrix multiplications, and

	Computation	Complexity	
Amortized initialization:			
1	Compute LDL$^\top$ factorization of K_0	$O(n^2)$ for 3D meshes; $O(n^{3/2})$ for 2D meshes	
2	Compute $a_0 = K_0^{-1} f_0$	$O(\lvert L_0 \rvert)$	
Real-time update steps:		per step	total
1	Solve for $V_{\Delta t}$	$O\left(\sum_{h \in \mathcal{H}_{\Delta t}} \text{closure}_{L_0}(h)\right)$	$O\left(\sum_{h \in \hat{\mathcal{H}}} \text{closure}_{L_0}(h)\right)$
2	Compute tril $\left(H_{\Delta t}^\top K_0^{-1} H_t\right)$	$O\left((m_{t-1} + 1 + m_t) m_{\Delta t} \cdot v_t\right)$	$O(\hat{m}^2 \hat{v})$
3	Form S_2	$O\left(m_t^2 (m_t + d_t) + m_t\right)$	$O\left(\hat{m}^2 (\hat{m} + \hat{d})\right)$
4	Form R.H.S. of Equation 3.16	$O(m_t)$	$O(\hat{m})$
5	Solve for $a^{(2)}$ in Equation 3.14	$O\left((m_t + d_t)^3\right)$	$O\left(c \cdot (\hat{m} + \hat{d})^3\right)$
6	Solve for a in Equation 3.24	$O(m_t \cdot v_t + \lvert L_0 \rvert)$	$O\left(c \cdot (\hat{m}\hat{v} + \lvert L_0 \rvert)\right)$

TABLE 1
Summary of the algorithm and its time complexity

in Step 4 the R.H.S. of Equation 3.16 is formed by mapping. The parallelization of Step 5 and 6 is non-trivial, which is out of the scope of this paper. The parallel time complexity of the update steps in the algorithm for p processors is

$$O\left(\frac{\hat{m}^2 \hat{v}}{p} + c \cdot \lvert L_0 \rvert\right). \tag{3.37}$$

3.7. Relation to previous augmented formulation. In earlier work [24] we have presented a hybrid unsymmetric augmented algorithm to perform surgical simulations using finite element models. In this earlier formulation, the system was augmented in an unsymmetric manner:

$$\begin{bmatrix} \bar{K} & J \\ \bar{H}^\top & 0 \end{bmatrix} \begin{bmatrix} a^{(1)} \\ a^{(2)} \end{bmatrix} = \begin{bmatrix} f \\ 0 \end{bmatrix}, \tag{3.38}$$

where J consists of the $(m + d)$ columns of K to replace the corresponding columns of $\bar{K}$. Note that we use $\bar{H}^\top$ here for matrices with more columns than rows instead. Equation 3.38 was then split into two parts to solve for $a^{(1)}$ and $a^{(2)}$ respectively:

$$\bar{H}^\top \bar{K}^{-1} J a^{(2)} = \bar{H}^\top \bar{K}^{-1} f \quad \text{and} \tag{3.39a}$$

$$a^{(1)} = \bar{K}^{-1} \left(f - J a^{(2)}\right), \tag{3.39b}$$

in which the first equation is solved by using GMRES whereas the second one is solved using a direct solver.

Since J is a submatrix of K, it can be expressed in terms of K as

$$J = K\bar{H}. \tag{3.40}$$

Substituting Equation 3.4 into Equation 3.40 yields

$$\begin{aligned} J &= \left(\bar{K} - \bar{H}\bar{E}\bar{H}^\top\right)\bar{H} \\ &= \bar{K}\bar{H} - \bar{H}\bar{E}. \end{aligned} \tag{3.41}$$

Substituting Equation 3.41 into Equations 3.39a and 3.39b yields

$$\left(I - \bar{H}^\top\bar{K}^{-1}\bar{H}\bar{E}\right)a^{(2)} = \bar{H}^\top\bar{K}^{-1}f \quad \text{and} \tag{3.42a}$$

$$a^{(1)} = \bar{K}^{-1}f - \bar{H}a^{(2)} + \bar{K}^{-1}\bar{H}\bar{E}a^{(2)}, \tag{3.42b}$$

in which the first equation is identical to Equation 3.14 and the second equation is identical to Equation 3.17. Hence the two augmented formulations are mathematically equivalent.

4. Results. The augmented matrix method for principal submatrix updates was evaluated through finite element cutting experiments with five model types. This section provides relevant implementation details and presents experimental data. We compare the performances of the following three approaches:

- AMPS algorithm presented in Section 3;
- Unsymmetric augmented matrix methods presented in [24] using a GMRES iterative solver, without preconditioning, and with two kinds of preconditioners: sparse approximate inverse (SPAI) and the diagonal matrix D_0 from the initial LDL$^\top$ factorization of the initial stiffness matrix; and
- Jacobi preconditioned or nonpreconditioned conjugate gradient (CG) iterative solver applied on $Ka = f$.

For the latter two approaches, only the best performing versions are included in the figures.

4.1. Implementation. All experiments were conducted on a compute node with two 16-core Intel Xeon Processors E5-2698 v3 ("Haswell") at 2.3 GHz, and each core equipped with 64 KB L1 cache (32 KB instruction cache, 32 KB data cache) and 256 KB L2 cache; as well as a 40-MB shared L3 cache per socket. In addition, there are 128 GB DDR4 2133 MHz memory. All data represent times averaged over 20 runs unless overall time exceeds 30 minutes, in which case we averaged over 10 runs.

The precomputed LDL$^\top$ factorizations of the stiffness matrices were computed using OBLIO, a sparse direct solver library [7]. All other basic linear algebra subroutines including matrix-vector products, dense matrix factorization and solves, as well as the GMRES iterative solver used in the unsymmetric augmented matrix methods and the CG solver used for comparison purposes were from the Intel Math Kernel Library (MKL) [11]. The remainder of the code, including the computation of the closure in K_0 induced by $H_{\Delta t}$, the matrices $V_{\Delta t}$ and tril $\left(H_{\Delta t}K_0^{-1}H_t\right)$ in Equation 3.30, and the overall algorithm, was written by the authors in C++.

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, the matrices K_0, L_0 and V were stored in compressed sparse column matrix (CSC) format for efficient column access. The diagonal matrix D_0 is stored in a vector of size n. The principal submatrix update E, the principal submatrix of the inverse $H^\top K_0^{-1}H$ and the Schur complement S_2 were stored in dense matrix format for fast computations. The matrix H and its transpose were represented as an array of indices and their multiplications with other matrices were done by index mappings. All vectors were stored in dense format.

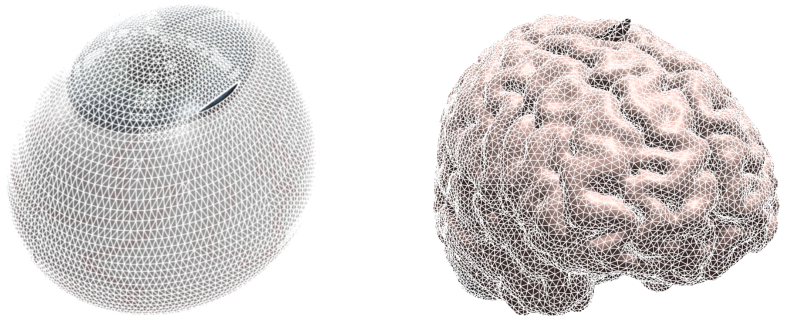


Fig. 2. *Renderings of (a) the eye mesh and (b) the brain mesh*

Mesh	$\|V\|$	Estimated condition number	Factorization time (s)
Beam	$100 - 25,600$	$1.14 \times 10^3 - 3.29 \times 10^{12}$	$0.02 - 1.62$
Brick	$250 - 18,081$	$2.19 \times 10^3 - 1.18 \times 10^5$	$0.1 - 5.42$
Eye	$17,821$	7.73×10^6	1.6
Brain	$50,737$	failed to estimate	7.77

Table 2
Numerical properties of the meshes.

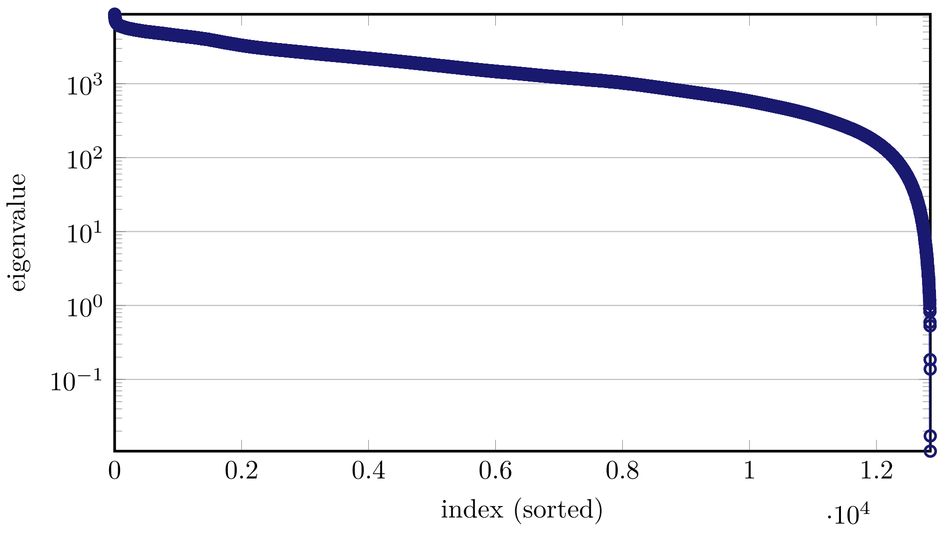


Fig. 3. *Eigenspectrum of the eye mesh of* $4,444$ *nodes.*

4.2. Model Meshes. Four types of solid tetrahedral meshes were used for performance evaluation. Table 2 lists for each mesh its number of vertices, the estimated condition number computed using Matlab's `condest` function, and the factorization times computed using OBLIO. Since the models are 3-dimensional, the total degrees of freedom (DOFs) in each system are 3 times the number of vertices minus the DOFs constrained by the Dirichlet boundary conditions $\mathcal{D}$, which is also the dimension of the matrix, i.e., $n = 3|V| - |\mathcal{D}|$.

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. 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 solver performance varies with stiffness matrix conditioning.
2. *Brick:* A group of five rectangular brick solids with varying mesh resolutions were generated. Each of the models had the same compact physical dimension of $1 \times 1 \times 2$. An initial good-quality mesh was uniformly subdivided to produce meshes of increasingly fine resolution. These meshes allowed us to examine solver performance relative to node count for fixed model geometry. Similar to the beam meshes, zero-displacement boundary conditions were applied to one face of the block.
3. *Eye:* A human eye model [4] with a clear corneal cataract incision was used in a simulation of corrective surgery for astigmatism. Zero displacement boundary conditions were applied to the posterior portion of the globe. Figure 3 shows the eigenspectrum of an eye mesh of 4,444 nodes, a downsampled mesh of the eye model. As it can be seen, the model has a wide range of eigenvalues, and hence a large condition number.
4. *Brain:* A human brain model (contributed by INRIA to the AIM@SHAPE Shape Repository) was used to demonstrate applicability to surgical simulation on an organ of complicated structure. Zero displacement boundary conditions were applied to the interior portion of the brain. The condition number could not be estimated with Matlab due to insufficient memory.

The eye and the brain mesh renderings are shown in Figures 2 and the renderings of other meshes can be found in [24].

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 present in the beam models versus interior nodes in the brick models. The increased connectivity leads to a higher percentage of nonzeros in the stiffness matrix factors and larger sizes for the closures referenced in Table 1. These differences have a significant impact on the relative performance of the solution methods.

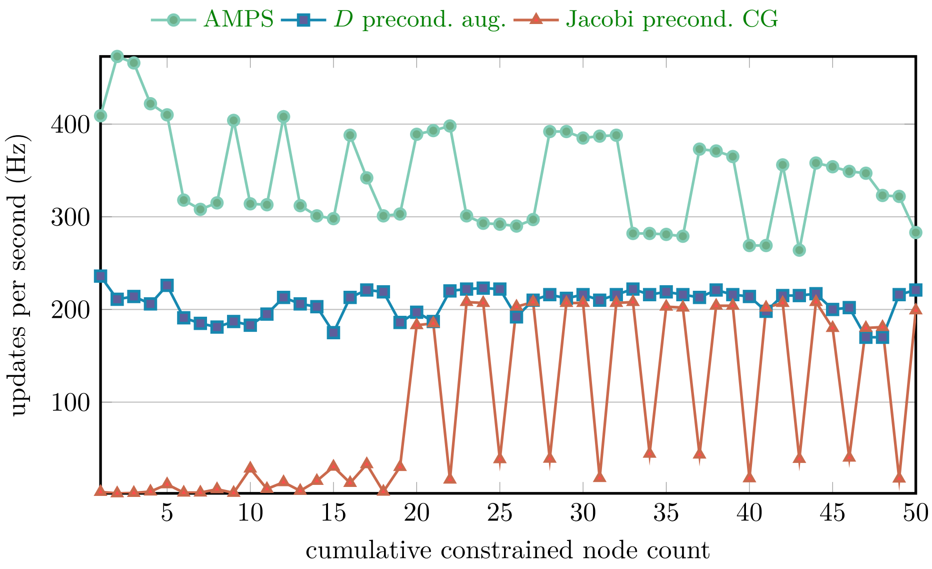
4.3. Experiments.

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

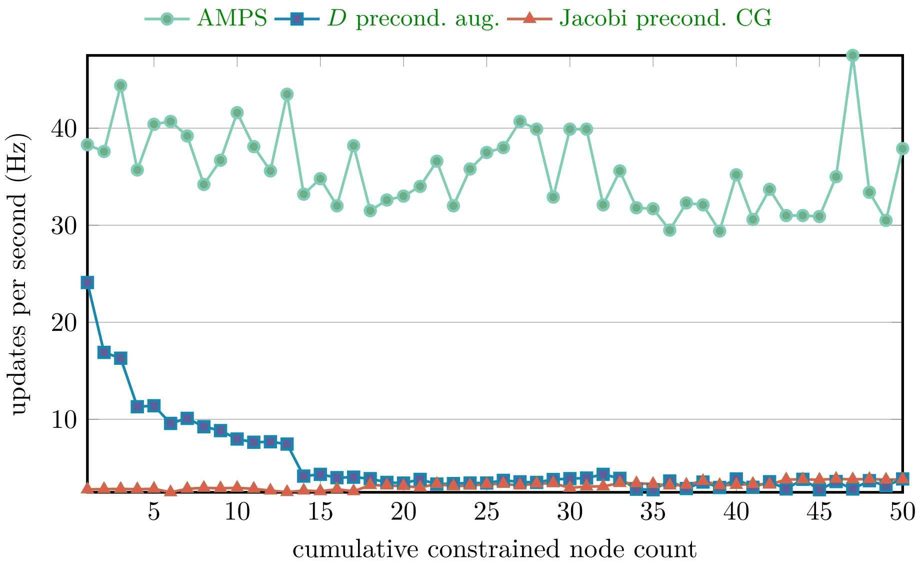
4.3.1. Deformation of Intact Meshes.

In this group of experiments, we applied an increasing number of non-zero essential boundary conditions to mesh nodes to create deformation. Figure. 4 shows how solution time varied with the number of constrained nodes for instances of the beam and brick meshes.

For the beam mesh, AMPS outperformed the unsymmetric augmented matrix method by a factor of 1.65 and the CG method by 3.63, while maintaining a high average update rate of 343 Hz (updates/sec) throughout. The unsymmetric augmented matrix method came second, maintaining update rates around 200 Hz. CG performed the worst, providing updates in the range of 1.6–33 Hz for the first 19 cutting steps, and experienced a zig-zag pattern afterwards caused by the connectivity pattern of



(a) Deformation of Beam Mesh: 6, 400 Nodes

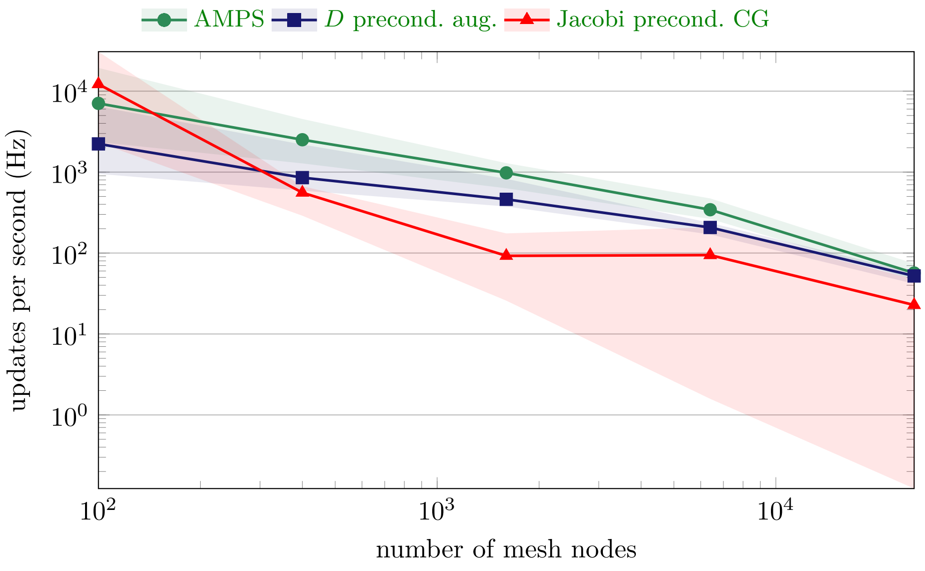


(b) Deformation of Brick Meshes: 9, 537 Nodes

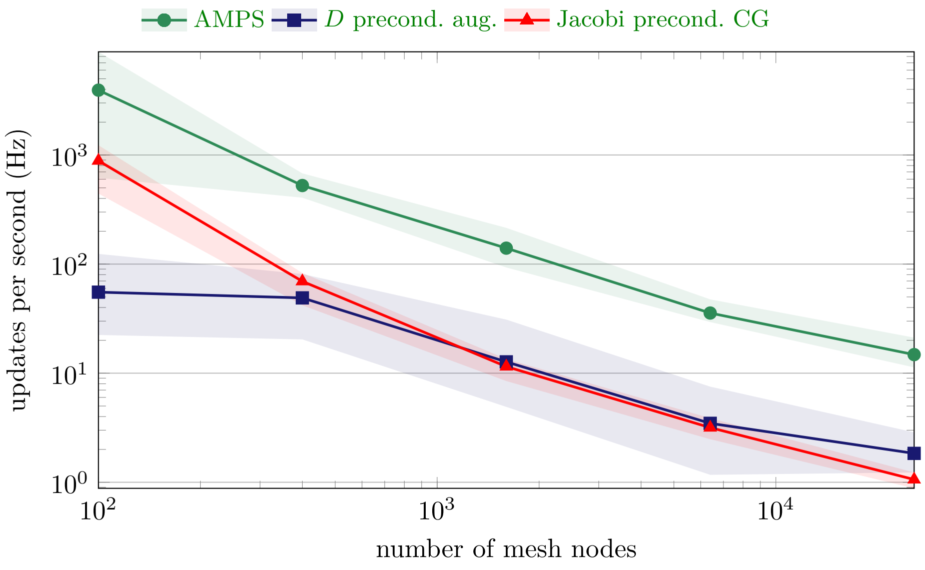
FIG. 4. *Deformation update rates are shown for AMPS, the preconditioned augmented and CG methods as constraints are progressively added to an increasing number of nodes in (a) beam and (b) brick meshes.*

nodes in the tetrahedral brick mesh as explained in [24]. This pattern also appeared in the results of the cutting experiments of the beam and brick meshes, as well as the eye mesh as they have a structural pattern in the ellipsoidal shapes.

For brick meshes, AMPS vastly outperformed the unsymmetric augmented matrix method by a factor of 6.37, and the CG method by 11.2. AMPS maintained



(a) Deformation of Beam Meshes



(b) Deformation of Brick Meshes

FIG. 5. *Average update rates and ranges are shown for the deformation experiments of the series of (a) beam and (b) brick meshes. AMPS results are shown in green, SPAI preconditioned augmented method results are shown in blue, and Jacobi preconditinoed CG results are shown in red.*

relatively stable average update rates at 35.6 Hz. The unsymmetric augmented matrix method outperformed CG as constraints were applied to the first dozen nodes, but performance degrades as the number of constrained nodes increased, eventually resulting in similar update rates between the augmented method and CG. Overall, the unsymmetric augmented matrix method achieved an average update rate of 5.59

Hz while the preconditioned CG method only had an average update rate of 3.17 Hz.

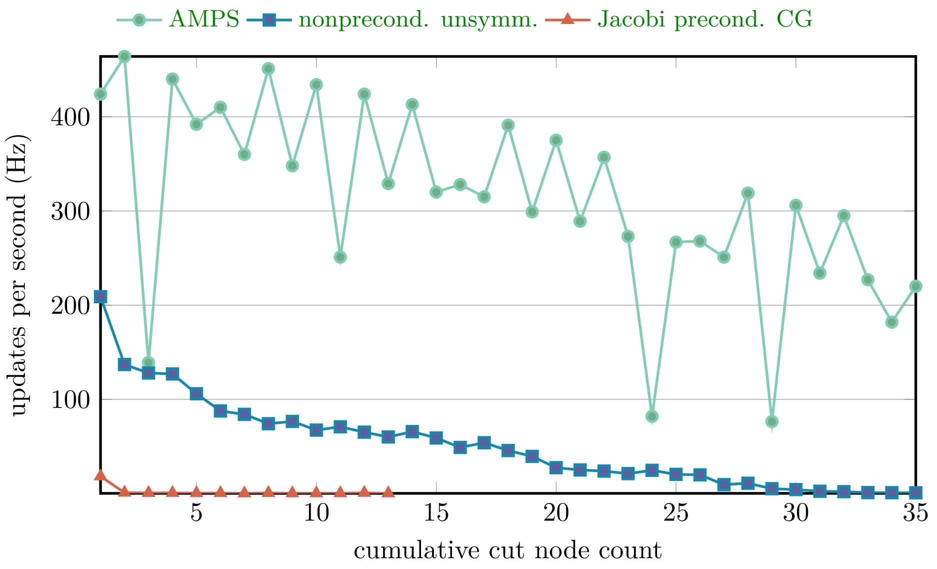
Figure 5 is a log-log plot that shows how solution times varied for different sizes of beam and brick meshes. The lines show the trend of the average times for various methods and the shaded areas are the ranges of the solution times. These graphs show that AMPS ran faster than both the augmented and CG methods for the beam meshes except for the very smallest instance that had only 100 nodes. It can also be observed that CG has the largest ranges among all methods especially for the larger beam meshes. This means that the CG solution times increased a lot while the deformation progressed. For the brick meshes, AMPS also outperformed both the augmented and CG methods with smaller solution time ranges than the other methods.

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 copy 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. The newly added node causes the linear system to increase in dimension, and the remeshing associated with the duplicated node and all its neighboring nodes results in a principal submatrix update to the stiffness matrix. In the results, the cut node count corresponds to the number of duplicated nodes resulting from the cut. Opposing force vectors were applied to selected surface nodes to pull the cut faces apart. Figure 2 shows the the eye mesh at the initial stages of cutting.

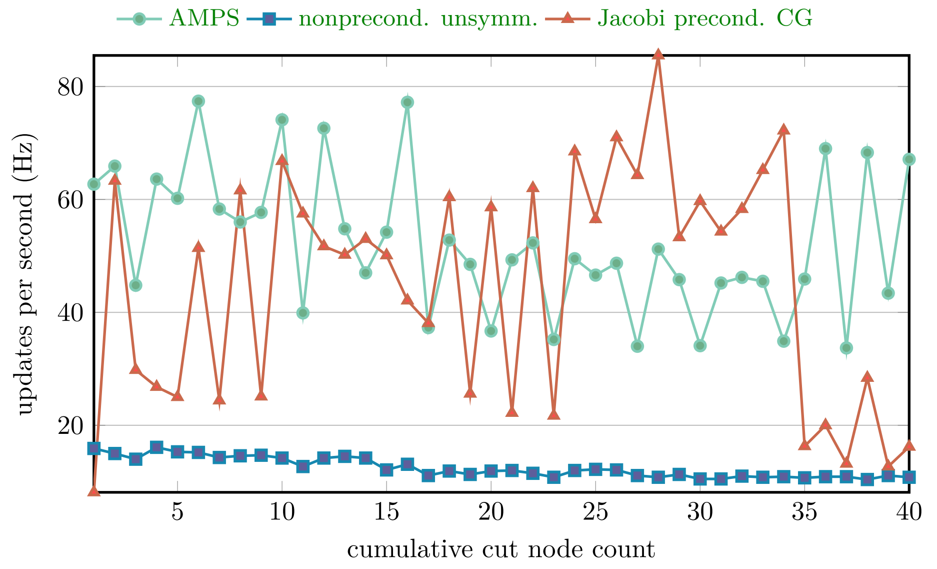
While the other methods behaved differently for the cutting and deformation experiments for the beam and brick meshes, AMPS performed similarly in the two experiments as shown in Figure 6a compared to Figure 4. AMPS outperformed the nonpreconditioned unsymmetric augmented matrix method by a factor of 6.06, and Jacobi preconditioned CG method by 216 in the beam cutting experiments, providing updates in the range 167–479 Hz. The unsymmetric augmented method provided 0.83–209 Hz whereas preconditioned CG needed more than 1 second for most of the cutting steps except for the first one, and failed to converge to any solution after the 18th step. D preconditioned and SPAI preconditioned variants ran 15.6 and 12.5 times slower than AMPS respectively. On the other hand, AMPS performed on par with CG for the brick mesh cutting experiment, providing 52.2 Hz and 44.8 Hz update rates; while the unsymmetric augmented matrix method underperformed for this mesh, providing only an average of 12.5 Hz update rate, as shown in Figure 6b. The D preconditioned and SPAI preconditioned variants ran 11.4 and 13.1 times slower than AMPS for the cutting of the brick mesh.

Figure 8 shows the breakdown of the solution times for individual steps of the AMPS algorithm for the brain mesh of $50,737$ nodes. The most computational expensive step was the triangular solve for the final solution a, accounting for roughly 80% of the time, followed by the computation of the principal submatrix of the inverse, accounting for roughly 20% of the time. The remaining steps are less significant. The valleys in the area plot are due to the fact that at some cuts no additional neighboring vertices were included in $\mathcal{H}_{\Delta t}$ and thus $\operatorname{tril}\left(H_{\Delta t}^\top K_0^{-1} H_t\right)$ is empty and the principal submatrix of the inverse of K_0 need not be updated.

Results from the eye and brain mesh cutting experiments are shown in Figure 7. Here we show that for the astigmatism surgical simulation experiment AMPS vastly outperformed the D preconditioned unsymmetric augmented matrix method by a factor of 10.8 and Jacobi preconditioned CG method by 12.2. For the brain model, AMPS ran 10.2 times faster than the SPAI preconditioned unsymmetric augmented



(a) Cutting of Beam Mesh: 6,400 Nodes

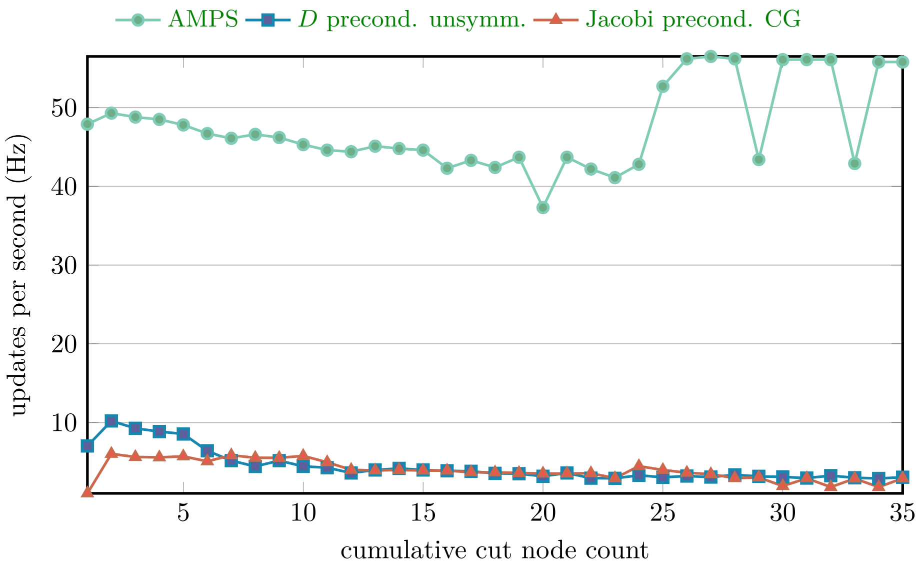


(b) Cutting of Brick Meshes: 9,537 Nodes

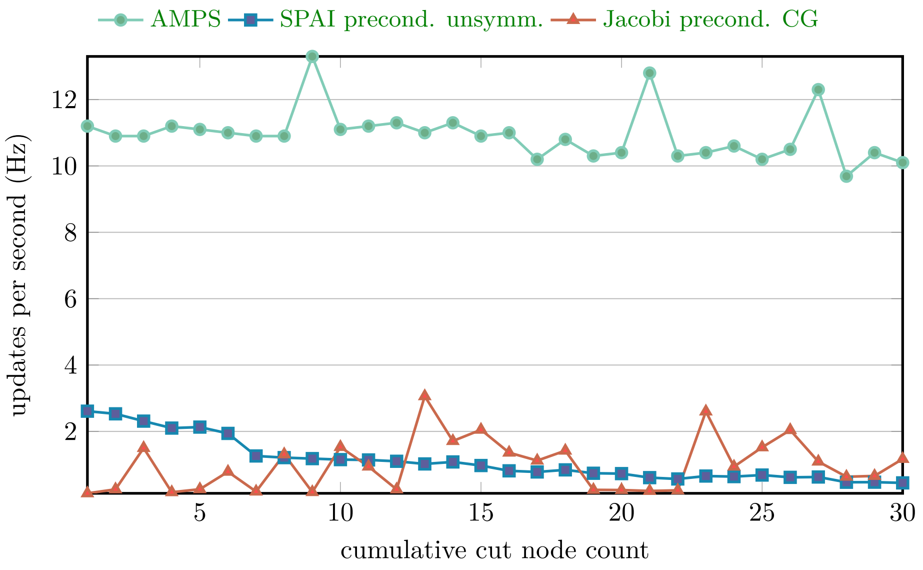
FIG. 6. *Update rates are shown for AMPS, the preconditioned augmented and CG methods as a cut is advanced through (a) a beam mesh and (b) a brick mesh.*

matrix method, 18.5 times faster than the D preconditioned variant, 11.5 times faster than the nonpreconditioned variant, and 11.5 times faster than Jacobi preconditioned CG method. The average update rates of 47.5 Hz and 11.4 Hz achieved by AMPS on both the eye and brain meshes respectively make interactive stimulation feasible.

Figure 9 shows the brain mesh cutting experiments using AMPS on a single core versus 32 cores. Speedups vary for different cuts due to the various numbers



(a) Astigmatism Surgical Simulation of Eye Mesh: $17,821$ Nodes



(b) Cutting of Brain Meshes: $50,737$ Nodes

Fig. 7. *Timing results are provided for (a) the eye mesh of* $17,821$ *nodes and (b) the brain mesh of* $50,737$ *nodes.*

of new neighboring nodes of the node being cut. For cuts that do not involve new neighboring nodes, the single-core results are even better than those using 32 cores due to the multi-core overheads. The geometric mean of the speedups is 1.58.

Since AMPS uses direct solver in both augmented part and the whole solutions, the solution accuracy of AMPS is only affected by the rounding errors amplified by the matrix condition number. Hence, AMPS not only provided faster update times than

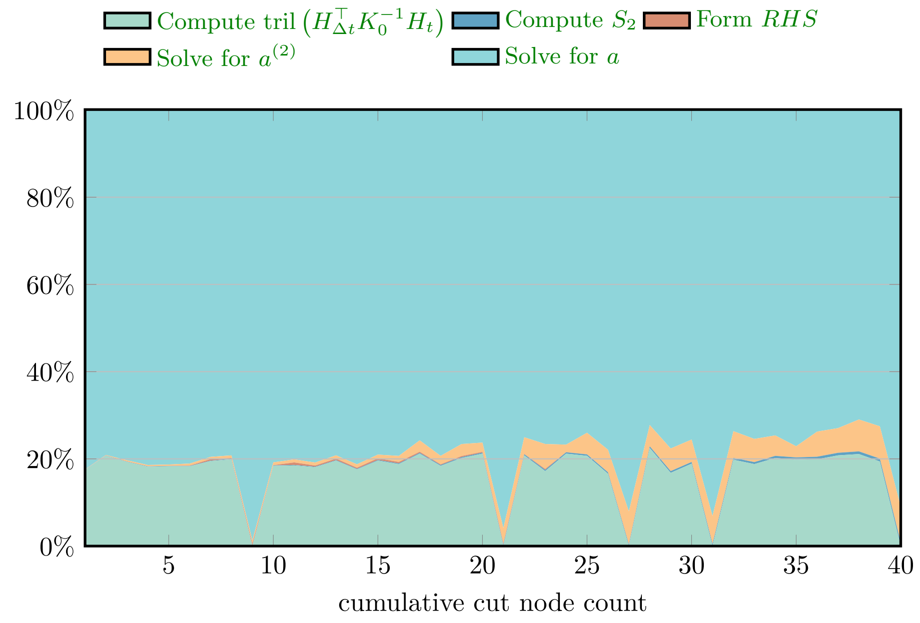


FIG. 8. *The breakdown of computation time to steps of AMPS for the brain mesh of 50,737 nodes.*

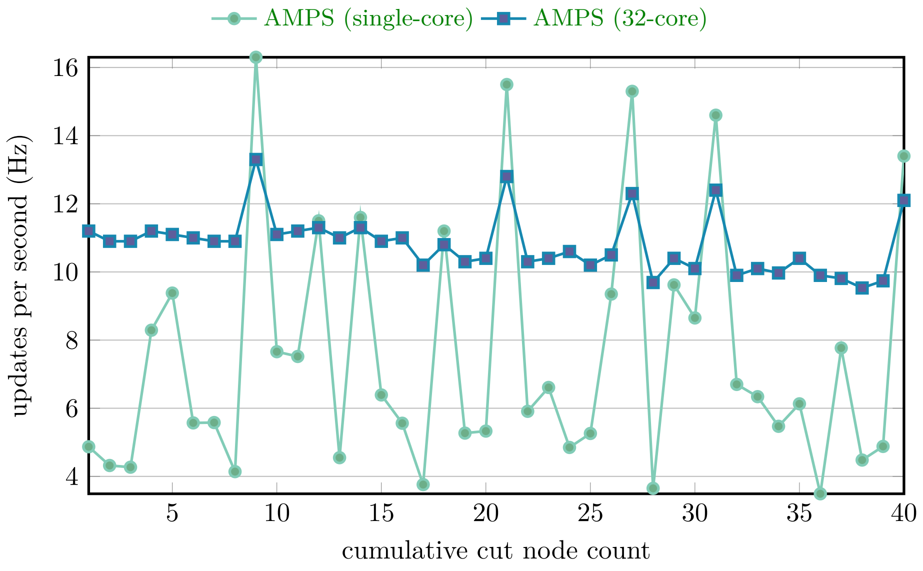


FIG. 9. *Single-core and 32-core results are provided for the brain mesh of 50,737 nodes.*

both the unsymmetric augmented matrix method and CG methods, but also higher accuracy. Table 3 compares the relative residual norms of the computed solutions of the tested methods. The absolute tolerances listed were set such that the computed relative residual norms were less than 10^{-3}. If lower tolerances were set, the number of iterations and thus the solution time would increase. It can be observed that the

Mesh	$\|V\|$	AMPS	SPAI precond. unsymm. aug.	Jacobi precond. CG
Beam	25,600	7×10^{-11}	$1 \times 10^{-4}(10^{-4})$	failed to converge
Brick	18,081	3×10^{-14}	$5 \times 10^{-5}(10^{-5})$	$5 \times 10^{-5}(10^{-8})$
Eye	17,821	4×10^{-14}	$7 \times 10^{-4}(10^{-5})$	$1 \times 10^{-5}(10^{-7})$
Brain	50,737	9×10^{-14}	$7 \times 10^{-5}(10^{-4})$	$1 \times 10^{-5}(10^{-5})$

Table 3
Comparison of relative residual norms ($\|\hat{K}\hat{a} - \hat{f}\|_2 / \|\hat{f}\|_2$). Absolute tolerances for the iterative solvers are listed in parentheses.

solutions computed by AMPS are much more accurate than the others.

5. Conclusions and Future Work.

When meshes are cut, new nodes and elements are inserted during the remeshing, and new boundary conditions are imposed. These changes result in principal submatrix updates to the stiffness system of equations, and we have demonstrated that the solutions of the modified systems can be computed in real-time with high accuracy even for large meshes. Our new AMPS algorithm has outperformed an earlier unsymmetric augmented method and CG in almost every deformation and cutting experiment, often by a factor of ten or more. We have also observed that unlike the unsymmetric augmented method, for most meshes, the update rates of AMPS do not deteriorate while the number of constrained nodes increases, or the cutting is being advanced in the meshes (Figure 7). These properties of AMPS are crucial for making real-time surgical simulation feasible as it requires accurate, fast and stable updates to the meshes. Refactorization would not be needed when AMPS is applied.

As we observed from the experimental results, the computation time for the augmentation is no longer the dominating factor of the total solution time for large meshes. More time was spent on the triangular solves in the solution. Hence, in the future one could incorporate the parallelization of the triangular solves into the AMPS algorithm. For more complicated and larger meshes, GPU and distributed parallelism could also explored.

The surgical simulations community has found the linear elastic model to be useful for biomechanical modeling when deformations are small and limited forces are applied, although linear elasticity does not adequately model organs and tissue types under heavier loading scenarios. Nonlinear models are not considered in this article, but could be investigated in the future for a broader range of surgical simulation problems, since there is evidence that viscoelastic and hyperelastic material models are often appropriate for modeling soft tissues [9, 13, 17].

REFERENCES

[1] U. Andreaus, I. Giorgio, and A. Madeo, *Modeling of the interaction between bone tissue and resorbable biomaterial as linear elastic materials with voids*, Zeitschrift für Angewandte Mathematik und Physik, 66 (2014), pp. 209–237.
[2] K. Bathe, *Finite Element Procedures*, Prentice-Hall, New Jersey, 1996.
[3] M. Chabanas, Y. Payan, C. Marécaux, P. Swider, and F. Boutault, *Comparison of linear and non-linear soft tissue models with post-operative CT scan in maxillofacial surgery*, in Medical Simulation. ISMS, S. Cotin and D. Metaxas, eds., vol. 3078 of Lecture Notes in Computer Science, Springer, Berlin, 2004.
[4] J. Crouch and A. Cherry, *Parametric eye models*, in Medicine meets virtual reality, J. West-

wood, R. Haluck, H. Hoffman, G. Mogel, R. Phillips, R. Robb, and K. Vosburgh, eds., vol. 15, Jan. 2007, pp. 91–93.

[5] J. Crouch, S. Pizer, E. Chaney, Y.-C. Hu, G. Mageras, and M. Zaider, *Automated finite element analysis for deformable registration of prostate images*, IEEE Trans. on Med. Imag., 26 (2007), pp. 1379–1390, https://doi.org/10.1109/TMI.2007.898810.

[6] T. A. Davis and W. W. Hager, *Row modifications of a sparse Cholesky factorization*, SIAM Journal on Matrix Analysis and Applications, 26 (2005), pp. 621–639, https://doi.org/10.1137/S089547980343641X.

[7] F. Dobrian and A. Pothen, *Oblio: Design and performance*, in Applied Parallel Computing. State of the Art in Scientific Computing, J. Dongarra, K. Madsen, and J. Wasniewski, eds., vol. 3732 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2006, pp. 758–767, https://doi.org/10.1007/11558958_92.

[8] C. Forest, H. Delingette, and N. Ayache, *Cutting simulation of manifold volumetric meshes*, in Proc. of Int. Conf. Medical Image Computing and Computer-Assisted Intervention, Part II, London, UK, 2002, Springer-Verlag, pp. 235–244.

[9] Y. Fung, *Biomechanics: Mechanical Properties of Living Tissues*, Springer-Verlag, 1993.

[10] O. Goksel and S. Salcudean, *Image-based variational meshing*, IEEE Trans. on Medical Imaging, 30 (2011), pp. 11–21, https://doi.org/10.1109/TMI.2010.2055884.

[11] Intel Corporation, *Math Kernel Library Developer Reference*, 2015, https://software.intel.com/en-us/articles/mkl-reference-manual.

[12] M. M. Juszczyk, L. Cristofolini, and M. Viceconti, *The human proximal femur behaves linearly elastic up to failure under physiological loading conditions*, Journal of Biomechanics, 44 (2011), p. 2259–2266.

[13] R. Lapeer, P. Gasson, and V. Karri, *Simulating plastic surgery: From human skin tensile tests, through hyperelastic finite element models to real-time haptics*, Progress in Biophysics & Molecular Biology, 103 (2010), pp. 208–216, https://doi.org/10.1016/j.pbiomolbio.2010.09.013.

[14] B. Lautrup, *Physics of Continuous Matter: Exotic and Everyday Phenomena in the Macroscopic World*, CRC Press, 2011, https://books.google.com/books?id=ohbSBQAAQBAJ.

[15] C. Lederman, A. Joshi, I. Dinov, J. Van Horn, L. Vese, and A. Toga, *Tetrahedral mesh generation for medical images with multiple regions using active surfaces*, in IEEE Int. Symp. Biomedical Imaging: From Nano to Macro, Apr. 2010, pp. 436–439, https://doi.org/10.1109/ISBI.2010.5490317.

[16] H. Liu, J. Li, X. Song, L. D. Seneviratne, and K. Althoefer, *Rolling indentation probe for tissue abnormality identification during minimally invasive surgery*, IEEE Transactions on Robotics, 27 (2011), pp. 450–460.

[17] S. Marchesseau, T. Heimann, S. Chatelin, R. Willinger, and H. Delingette, *Fast porous visco-hyperelastic soft tissue model for surgery simulation: Application to liver surgery*, Progress in Biophysics & Molecular Biology, 103 (2010), pp. 185–196, https://doi.org/10.1016/j.pbiomolbio.2010.09.005.

[18] A. Mohamed and C. Davatzikos, *Finite element mesh generation and remeshing from segmented medical images*, in IEEE Int. Symp. Biomedical Imaging: Nano to Macro, vol. 1, Apr. 2004, pp. 420–423, https://doi.org/10.1109/ISBI.2004.1398564.

[19] A. Mor and T. Kanade, *Modifying soft tissue models: Progressive cutting with minimal new element creation*, in Medical Image Computing and Computer-Assisted Intervention, S. Delp, A. DiGoia, and B. Jaramaz, eds., vol. 1935 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2000, pp. CH412–CH412.

[20] J. Spillmann and M. Harders, *Robust interactive collision handling between tools and thin volumetric objects*, IEEE Trans. on Visualization and Computer Graphics, 18 (2012), pp. 1241–1254, https://doi.org/10.1109/TVCG.2011.151.

[21] D. Steinemann, M. Harders, M. Gross, and G. Szekely, *Hybrid cutting of deformable solids*, in Prof. of IEEE Virtual Reality, Mar. 2006, pp. 35–42, https://doi.org/10.1109/VR.2006.74.

[22] M. Tchonkova and S. Sture, *Classical and recent formulations for linear elasticity*, Archives of Computational Methods in Engineering, 8 (2001), pp. 41–74, https://doi.org/10.1007/BF02736684, https://doi.org/10.1007/BF02736684.

[23] 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, *Collision detection for deformable objects*, 2005, https://doi.org/10.1111/j.1467-8659.2005.00829.x.

[24] Y.-H. Yeung, J. Crouch, and A. Pothen, *Interactively cutting and constraining vertices in meshes using augmented matrices*, ACM Trans. Graph., 35 (2016), pp. 18:1–18:17,

https://doi.org/10.1145/2856317.

[25] Y.-H. Yeung, A. Pothen, M. Halappanavar, and Z. Huang, *AMPS: An augmented matrix formulation for principal submatrix updates with application to power grids*, SIAM J. Scientific Computing, (2017). to appear.

[26] X. Zhang and Y. Kim, *Simple culling methods for continuous collision detection of deforming triangles*, IEEE Trans. on Visualization and Computer Graphics, 18 (2012), pp. 1146–1155, https://doi.org/10.1109/TVCG.2011.120.