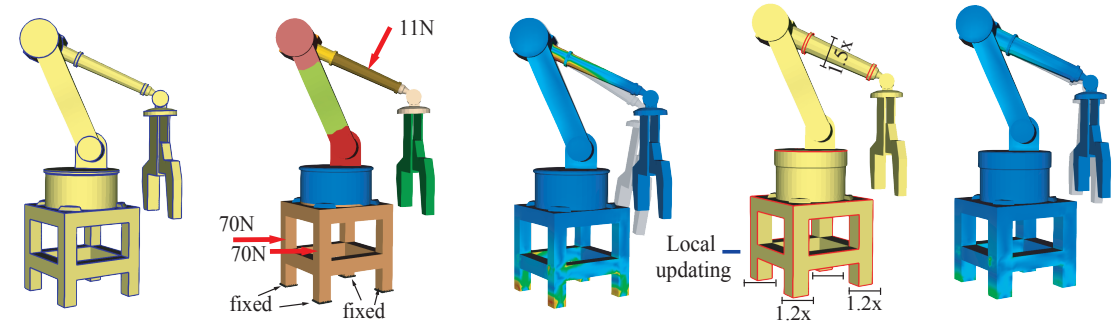
Submission to GMP 2018

Stress-aware Large-scale Mesh Editing using a Domain-decomposed Multigrid Solver

Submission number: 1049



(a) Manipulator mesh (b) Decomposed domains (c) Stress before editing (d) Thicken the basis (e) Stress after editing

Figure 1: The integration of IWires shape editing [19] and FEM stress analysis via domain decomposition for a 3D manipulator model with around five million DOFs. (a) Extract sharp edges, highlighted as blue lines, to form wires as the editing interface. (b) Decompose the model into domains. (c) The stress analysis result under the applied external forces, which are shown as red arrows in (b). (d)&(e) show how the editing operations reduce the stress peaks at manipulator's base to improve its structural stability. In our setting, high stress values are visualized in red and low stress values are in blue. The light gray shapes are the deformation results. Since the editing operation is to thicken the basis of the manipulator, we only need to update the subspace and FEM data structure for the corresponding domain. The local updating and solving time of our solver (updating + solving takes 174.27s) is roughly 70% faster than the parallel `PARDISO` solver from `Intel Math Kernel Library` (numerical factorization + solving takes 292.79s). Please also see the accompanied video for the editing and stress analysis procedure.

Abstract

In this paper, we develop a domain-decomposed subspace and multigrid solver to analyze the stress distribution for large-scale finite element meshes with millions of degrees of freedom. Through the domain decomposition technique, the shape editing directly updates the data structure of local finite element matrices. Doing so avoids the expensive factorization step in a direct solver and provides users with a progressive feedback of the stress distribution corresponding to the mesh operations: a fast preview is achieved through the subspace solver, and the multigrid solver refines the preview result if the user needs to examine the stress distribution carefully at certain design stages. Our system constructs the subspace for stress analysis using reduced constrained modes and builds a three-level multigrid solver through the algebraic multigrid method. We remove mid-edge nodes and lump unknowns with the Schur complement method. The updating and solving of the large global stiffness matrix are implemented in parallel after the domain decomposition. Experimental results show that our solver outperforms the parallel `Intel MKL` solver. Speedups of 50% - 100% can be achieved for large-scale meshes with reasonable pre-computation costs when setting the stopping criterion of the multigrid sover to be $1e-3$ relative error.

1 Introduction

In Computer Graphics, both geometric editing that modifies the shapes of 3D models and stress analysis using the finite element method (FEM) have been extensively studied. The application of these two techniques to improve the structural stability of 3D printouts also receives many research interests [37, 47]. Nevertheless, these two categories of research focus on distinctive objectives and their computing procedures differ from each other significantly. Existing design systems and commercial CAD/CAE packages only integrate the FEM analysis with the shape editing at the user interface level and use separate algorithmic subroutines to deal with each of them.

Clearly, a more in-depth coupling between the shape editing and FEM simulation will improve system efficiency and deliver better design experiences to the user. To this end, Nobuyuki and colleagues [41] re-use the linear FEM data structure to achieve a fast response of 2D designs. The problem of fabrication-aware shape editing was also explored in [44], where the stiffness matrix is directly parameterized using the editing parameters like scaling, rotating etc. to expedite the FEM matrix assemblage. However, those existing approaches adopt direct matrix solvers and the global stiffness matrix in FEM has to be re-factorized during shape editing, which is known to be expensive. Therefore, they are usually applied to meshes of moderate sizes i.e. thousands of degree of freedoms (DOFs). Indeed, stress analysis for large-scale meshes are more and more common in practical applications especially when 1) an as-accurate-as-possible stress distribution is needed and/or 2) 3D models are geometrically complex with a lot of local details. For instance, Akihiro and colleagues [24] use millions of DOFs to analyze the welding residual stress. How to achieve a fast and accurate design-simulation integration for system at such scale with high order elements remains a difficult problem.

As an echo to this challenge, we present a new algorithmic design of subspace and multigrid solvers to handle shape editing and stress analysis integration for large scale meshes. Our system efficiently handles 3D models with millions of DOFs on desktop PCs and utilizes quadratic elements in FEM simulation for an accurate stress analysis. The subspace solver provides a fast preview of stress distribution. The multigrid solver further refines the subspace solution if the user needs a careful inspection of stress values

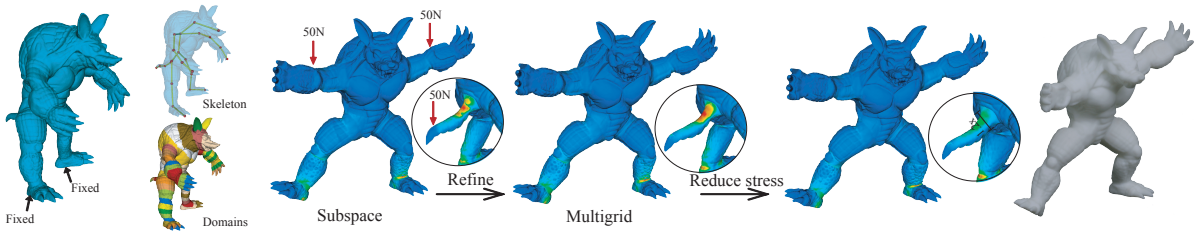


Figure 2: The system flowchart. Given an input Armadillo model, the system pre-computes its subspace basis and multigrid data structure. After the user edits its pose using the underlying skeleton, a quick preview of the stress distribution is immediately available (§ 5) and the user can choose to refine the subspace solution with the multigrid solver (§ 6) as shown in (b). The user can improve the structural stability by thickening the weak regions (c), and fabricate the final editing result with 3D printing (d).

at certain design stages. We adopt the domain decomposition method to localize the shape or topology editing and minimize the numerical update cost of the FEM data structure. That is, only the stiffness matrices and subspace basis of those domains influenced by the shape editing operation are updated. Thanks to the domain decomposed model, such local update can be done in parallel and is always synchronized with the user editing. More specifically, some unique technical features of our system include:

- Our system adopts constraint modes to build the subspace for a fast stress preview. According to the theory of component mode synthesis in structural mechanics [13], constraint modes represent the response of a domain interior nodes to the imposed unit displacement at a boundary node, and such modes provide a *complete space* of the static deformation of an object undertaking gravity and external forces at interfaces. Thus, the subspace solution using constraint modes also provides a good initial solution to speed up the multigrid solver.
- Instead of using the Lagrange multiplier method to stitch the domain boundaries which results in a none positive definite stiffness matrix, we further tweak constraint modes formulation to allow us to use the same set of unknown variables at domain boundaries. Doing so yields a symmetric positive definite (SPD) subspace system with domains' boundaries implicitly coupled, which can be fast solved using the Cholesky algorithm. Being free of the duplication of interface DOFs, the dimension of the system is also reduced.
- Our three-level algebraic multigrid solver is designed for quadratic FEM simulation. The first level is built on the original quadratic element mesh; the second level is built by removing mid-edge nodes; and the third level is built by lumping unknowns with Schur complement method to some key nodes at domains' interfaces (referred to as corner nodes). Such algorithmic design maintains the domain structure and results in a block-wise linear system at each level so that they can be locally updated in parallel to improve the simulation performance.

We currently let the stopping criterion of the multigrid solver be $1e-3$ relative residual error, which is enough in our experiments to achieve high-quality stress distribution results. With this stopping criterion, the computation time of our solver is much less than the commercial matrix solvers. It is straightforward to perform more multigrid iterations to achieve a smaller relative residual error when necessary. Furthermore, our subspace and the subsequent multigrid solver can provide progressive response in stress analysis, and the user can select where to stop to balance between the computation time and the accuracy of the results.

Two shape editing prototypes, namely IWires and skeleton driven shape editing, are implemented in our system to show the capability of our system for the shape editing and stress analysis integration. We have tested our solvers on a variety of large-scale 3D models from half millon to five million DOFs. Our solver provides a fast preview of stress analysis results to allow a fluent design experience with help of the subspace solver. Experimental results show that, with reasonable subspace and multigrid structure pre-computation costs, the proposed domain-decomposed multigrid achieves $50\% - 100\%$ **speedups** over the `Intel Math Kernel Library` or `MKL` [43], which is known as one of the best-optimized parallel direct solvers.

2 Related Work

Fabrication-aware design Computation design system aims to handle fabrication requirement in geometric design via constrained optimization or the integration of fast simulation techniques. This line of research allows the user to control physical properties of the design, such as appearance [14, 27, 9], deformation [5, 36, 32], articulation [28, 7, 2], acoustics [29], and mechanical motion [48, 11, 8]. To improve the structural stability of a design, a common method is to alter its shape based on the stress analysis, when the material properties are given. Stress relief operations, such as hollowing and thickening, are adopted in [37] to improve the structural stability of the 3D objects. Subspace technique are used to fast analyze worst load distribution that causes high stress in 3D printable objects [47]. Our goal instead, is to develop a structural analysis algorithm integrated with shape editing so that the user can predict the stress distribution while editing the model's geometry.

FEM domain decomposition The domain decomposition (DD) method is designed to solve large-scale partial differential equations by splitting the original problem into a set of small problems so that each sub-problem can be solved with manageable memory and computation costs [40]. It is particularly suitable for parallel or distributed environment and has wide applications in structure analysis and stress analysis [16, 44]. For non-overlapping DD methods, the domains only intersect at their boundaries, and the continuity of the solution across domains should be enforced. A seminal work in non-overlapping DD is the Finite Element Tearing and Interconnect (FETI) [17, 16]. It adopts Lagrange multiplier method to guarantee that the solution of each domain is equal at boundaries. DD integrated with subspace methods is developed for real-time deformable animation, where the deformation at domain boundaries are simplified to reduce the number of subspace basis, for example, the assumption of interface rigidity [4], penalty forces based soft coupling [26], or with reduced boundary freedoms [46].

Our work employs DD to localize the shape editing and FEM data structure updating, and it avoids the expensive factorization step in direct matrix solver by solving the system with a multigrid solver in parallel tailored for DD. The subspace and multigrid solvers, in our system, are primal solvers, which avoid the dual Lagrange multipliers by representing the domain boundaries with same unknowns.

Multigrid method The multigrid solver is powerful in solving linear and nonlinear systems. They have been extensively used for various problems in graphics, including shape modeling [35], cloth simulation [22, 38], fluid animation [30], and deformable models [49, 31, 33]. Multigrid methods leverage the fact that many iterative linear solvers, such as the Jacobi method, are able to effectively reduce or *smooth* high-frequency errors, but not for low-frequency errors. If a fine-level residual gets projected onto a coarse grid, the spectrum of the error vector is shifted from a high-frequency one to a relatively low-frequency one. Therefore, smoothing at coarse level becomes effective again. Multigrid methods often require dedicated efforts to design a good mapping between the coarse grid and the fine grid, also known as the prolongation and restriction operators. Therefore, regular grids are preferred [49]. In addition to creating the prolongation and restriction operators based on grid geometrically. Multigrid can also be constructed algebraically [45]. The key idea is to select a subset of unknown DOFs as a coarse problem and represent the other DOFs with them. Schur-complement or incomplete Gaussian elimination serve this purpose well [34, 42].

The multigrid solver is also popular in FEM problems [1, 23], and efficient GPU multigrid solvers are developed in [20]. However, the FEM data structures in such methods need to be globally updated. Our work can be viewed as a non-trivial extension of them in the case of shape editing and stress analysis integration, which further accelerates the updating and solving speed of the multigrid solver in parallel with domain decomposition technique.

3 System Overview

The primary goal of our system is to integrate IWires and skeleton editing methods [19, 44] with FEM simulation in the design of large-scale meshes. The system flowchart is illustrated in Fig. 2. After an editing operation is committed, the internal data structures for subspace and multigrid solvers are locally updated at influencing domains. Afterwards, the system computes new stress results for the edited mesh. The local updating operations and stress computation are implemented in parallel at domain level, thanks to the advantage of the domain decomposition technique.

The IWires shape editing method is well designed for shapes with salient feature lines, such as man-made objects [19]. It employs extracted feature lines as the editing interface and maintains the geometric relationships between them, such as planarity, orthogonality and parallelism, so that the edited shape reconstructed from user-edited feature lines still preserves its characteristic structure. We implement IWires method in our system for users to edit mechanical models, for instance, the manipulator model in Fig. 1. While a wire is translated or scaled, its editing is automatically propagated to other wires to form the editing result. Afterwards, the FEM data structure for domains influenced by the editing are updated to obtain the stress analysis results. The implemented skeleton editing interface is achieved by rigging the domains to the underlying skeleton. Therefore, when the user rotates the skeleton, the domains associated to it will be automatically rotated as well. Our implementation follows the algorithm in [44].

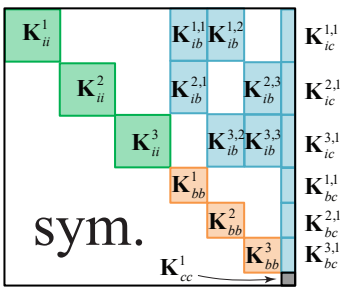
4 Preliminary

It is possible to permutate nodal DOFs and assemble the global stiffness matrix in a way reflecting the corresponding domain decomposition. In this section, we discuss how to leverage this block-wise structure and the matrix condensation technique to form a small-size coarse problem to warm start the full-scale equilibrium solution.

Classification of DOFs As illustrated in Fig.3, we define three types of nodes namely, internal nodes, boundary nodes and corner nodes on the domain decomposed mesh. Internal nodes are the ones inside a domain, and they do not have a direct adjacency towards nodes from other domains. Boundary nodes sit on the interface between two domains. It is possible that a node is incident to multiple (more than two) domains. In this case, the node is referred to as a corner node. We use subscripts $[\cdot]_i$, $[\cdot]_b$ and $[\cdot]_c$ to denote the internal node, boundary node and corner node respectively. While assembling the stiffness matrix, all the internal nodal DOFs are listed first, followed by boundary and corner ones. Therefore, the global equilibrium equation $\mathbf{Ku} = \mathbf{f}$ becomes in the format of:

$$\begin{bmatrix} \mathbf{K}_{ii} & \mathbf{K}_{ib} & \mathbf{K}_{ic} \\ \mathbf{K}_{ib}^\top & \mathbf{K}_{bb} & \mathbf{K}_{bc} \\ \mathbf{K}_{ic}^\top & \mathbf{K}_{bc}^\top & \mathbf{K}_{cc} \end{bmatrix} \begin{bmatrix} \mathbf{u}_i \\ \mathbf{u}_b \\ \mathbf{u}_c \end{bmatrix} = \begin{bmatrix} \mathbf{f}_i \\ \mathbf{f}_b \\ \mathbf{f}_c \end{bmatrix} \tag{1}$$

Block-wise stiffness matrix Nodes of the same type are also clustered according to their connectivity forming a connected component. Each cluster is assigned with a local index. Taking Fig. 3 an example, there are three internal nodes clusters (shadowed in green), three boundary node clusters (shadowed in orange) as well as one corner node cluster (shadowed in black). As shown on the right, each sub-matrix corresponding to a node cluster (i.e. a diagonal block) has a single superscript $[\cdot]^j$ denoting its local index and repetitive subscripts (i.e. $[\cdot]_{ii}$, $[\cdot]_{bb}$ or $[\cdot]_{cc}$). If subscripts do not match, a double superscript $[\cdot]^{j,l}$ will be used such that j and l are local indices of the linked DOF clusters. An off-diagonal sub-block is nonzero if and only if the corresponding DOF clusters are adjacent to each other on the mesh. The unknown displacement vector $\mathbf{u}$ and the external force $\mathbf{f}$ can be written similarly: $\mathbf{u} = \left[\mathbf{u}_i^{1\top}, \mathbf{u}_i^{2\top}, \mathbf{u}_i^{3\top}, \mathbf{u}_b^{1\top}, \mathbf{u}_b^{2\top}, \mathbf{u}_b^{3\top}, \mathbf{u}_c^{1\top}\right]^\top$ and $\mathbf{f} = \left[\mathbf{f}_i^{1\top}, \mathbf{f}_i^{2\top}, \mathbf{f}_i^{3\top}, \mathbf{f}_b^{1\top}, \mathbf{f}_b^{2\top}, \mathbf{f}_b^{3\top}, \mathbf{f}_c^{1\top}\right]^\top$.



Coarse problem via matrix condensation The block-wise structure matrix allows us to define a *coarse problem* on the DOFs of boundary and corner nodes through the matrix condensation technique [6, 39]. The number of DOFs in the coarse problem is much smaller than the original problem size. Therefore, it can be solved efficiently using a direct solver, and the full-scale solution can be obtained in parallel afterwards. Since the condensation technique is adopted to compute the reduced matrix in multigrid solver

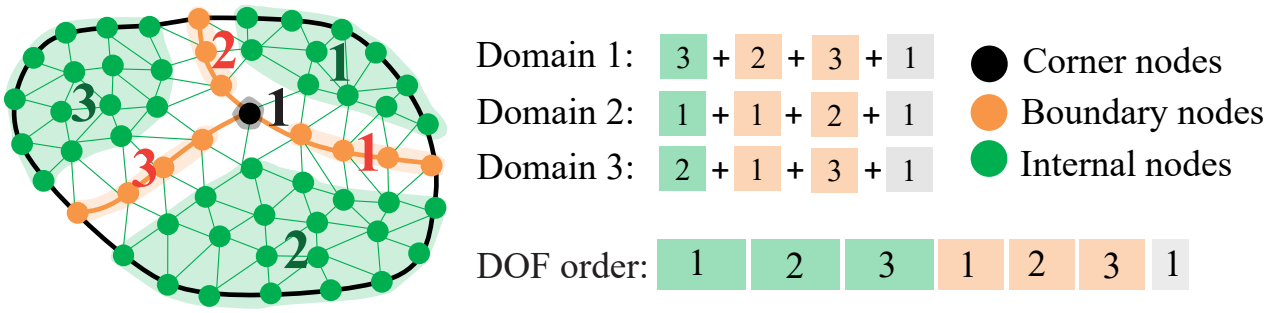


Figure 3: Domain decomposition, classification of DOFs and the corresponding DOF order.

for *each boundary cluster* i **do**
  find its two adjacent domain $\mathbf{D}_{d_1}$ and $\mathbf{D}_{d_2}$;
  for *each boundary cluster* j, $j \neq i$ *adjacent to* $\mathbf{D}_{d_1}$ **do**
    compute $\mathbf{G}_{bb}^{i,j} \leftarrow \mathbf{K}_{ib}^{d_1,i^\top} \mathbf{K}_{ii}^{d_1^{-1}} \mathbf{K}_{ib}^{d_1,j}$;
  end
  for *each corner cluster* k *adjacent to* $\mathbf{D}_{d_1}$ **do**
    compute $\mathbf{G}_{bc}^{i,k} \leftarrow \mathbf{K}_{ib}^{d_1,i^\top} \mathbf{K}_{ii}^{d_1^{-1}} \mathbf{K}_{ic}^{d_1,j}$;
  end
  compute $\mathbf{G}_{bb}^{i,j}$ and $\mathbf{G}_{bc}^{i,k}$ similarly for $\mathbf{D}_{d_2}$;
  compute $\mathbf{G}_{bb}^{i,i} \leftarrow \mathbf{K}_{bb}^{i} - \sum_{d=\{d_1,d_2\}} \mathbf{K}_{ib}^{d,i^\top} \mathbf{K}_{ii}^{d^{-1}} \mathbf{K}_{ib}^{d,i}$;
end
for *each corner cluster* i **do**
  for *each of its adjacency domain* $\mathbf{D}_d$ **do**
    for *each boundary cluster* j *adjacent to* $\mathbf{D}_d$ **do**
      compute $\mathbf{G}_{cb}^{i,j} \leftarrow \mathbf{K}_{ic}^{d,i^\top} \mathbf{K}_{ii}^{d^{-1}} \mathbf{K}_{ib}^{d,j}$;
    end
    for *each corner cluster* k *adjacent to* $\mathbf{D}_d$ **do**
      compute $\mathbf{G}_{cc}^{i,k} \leftarrow \mathbf{K}_{ic}^{d,i^\top} \mathbf{K}_{ii}^{d^{-1}} \mathbf{K}_{ic}^{d,k}$;
    end
  end
  compute $\mathbf{G}_{cc}^{i,i} \leftarrow \mathbf{K}_{cc}^{i} - \sum_{d} \mathbf{K}_{ic}^{d,i^\top} \mathbf{K}_{ii}^{d^{-1}} \mathbf{K}_{ic}^{d,i}$;
end

Algorithm 1: Assembling the coarse problem of Eq. (3).

at second-to-third level, we now detail how to obtain the matrices in coarse problem and their updating during the shape editing. Expanding Eq. (1) at internal nodes yields $\mathbf{K}_{ii}\mathbf{u}_i + \mathbf{K}_{ib}\mathbf{u}_b + \mathbf{K}_{ic}\mathbf{u}_c = \mathbf{f}_i$, and $\mathbf{u}_i$ can be *condensed* as:

$$\mathbf{u}_i = \mathbf{K}_{ii}^{-1}(\mathbf{f}_i - \mathbf{K}_{ib}\mathbf{u}_b - \mathbf{K}_{ic}\mathbf{u}_c). \tag{2}$$

Substituting Eq. (2) into Eq. (1) yields the coarse problem, which is essentially the equilibrium at boundary and corner nodes:

$$\begin{bmatrix} \mathbf{G}_{bb} & \mathbf{G}_{bc} \\ \mathbf{G}_{bc}^\top & \mathbf{G}_{cc} \end{bmatrix} \begin{bmatrix} \mathbf{u}_b \\ \mathbf{u}_c \end{bmatrix} = \begin{bmatrix} \widetilde{\mathbf{f}}_b \\ \widetilde{\mathbf{f}}_c \end{bmatrix}, \tag{3}$$

where

$$\begin{array}{rclcrcl} \mathbf{G}_{bb} &=& \mathbf{K}_{bb} - \mathbf{K}_{ib}^\top \mathbf{K}_{ii}^{-1} \mathbf{K}_{ib} & \qquad & \widetilde{\mathbf{f}}_b &=& \mathbf{f}_b - \mathbf{K}_{ib}^\top \mathbf{K}_{ii}^{-1} \mathbf{f}_i \\ \mathbf{G}_{bc} &=& \mathbf{K}_{bc} - \mathbf{K}_{ib}^\top \mathbf{K}_{ii}^{-1} \mathbf{K}_{ic} & & \widetilde{\mathbf{f}}_c &=& \mathbf{f}_c - \mathbf{K}_{ic}^\top \mathbf{K}_{ii}^{-1} \mathbf{f}_c. \\ \mathbf{G}_{cc} &=& \mathbf{K}_{cc} - \mathbf{K}_{ic}^\top \mathbf{K}_{ii}^{-1} \mathbf{K}_{ic}, & & & & \end{array} \tag{4}$$

Typically, $\mathbf{K}_{ii}$ is the biggest sub-block, however one can easily tell that it is also block-diagonal because each internal node cluster is isolated from each other by the boundary and corner nodes. Therefore, computing $\mathbf{K}_{ii}^{-1}\mathbf{K}_{ib}$ and $\mathbf{K}_{ii}^{-1}\mathbf{K}_{ic}$ in Eq. (4) is efficient and parallelizable. The coupling between boundaries and corners only exists when they are adjacent in one domain. Thus, the coarse problem is also sparse. If the geometry or topology of few domains are changed by the shape editing operation, we only need to update the corresponding $\mathbf{K}_{ii}^{-1}$ and compute the entries of the boundaries and corners adjacent to the edited domains as outlined in Alg. 1. Note that all the for-loops in Alg. 1 are parallelizable at domains.

5 Fast Subspace Stress Preview

According to the theory of component mode synthesis (CMS) [13], attachment or constraint modes define how a local domain response to external excitements applied via the boundary. If a domain only undertakes gravity and external forces at its interfaces,

constraint modes compose a complete set of its static deformation basis, meaning any static equilibrium can be exactly represented by the constraint mode set. Therefore, they are chosen as the subspace basis to analyze the stress distribution in our system. In this section, we describe how to further reduce the constraint modes set to construct a more compact subspace to achieve the fast preview of stress analysis results.

Constraint modes The constraint modes are defined for a local domain. Here, we drop the superscript in this section for succincter notations. In classic CMS, constraint modes describe domain's equilibrium shape when one of its boundary DOFs is prescribed by a unit displacement while the others are constrained. Mathematically, it can be expressed as a local equilibrium equation:

$$\begin{bmatrix} \mathbf{K}_{ii} & \mathbf{K}_{ib} & \mathbf{K}_{ic} \\ \mathbf{K}_{ib}^\top & \mathbf{K}_{bb} & \mathbf{K}_{bc} \\ \mathbf{K}_{ic}^\top & \mathbf{K}_{bc}^\top & \mathbf{K}_{cc} \end{bmatrix} \underbrace{\begin{bmatrix} \mathbf{\Phi}_i^b & \mathbf{\Phi}_i^c \\ \mathbf{I}_b & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_c \end{bmatrix}}_{\text{constraint modes } \mathbf{U}} = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{f}_b & \mathbf{0} \\ \mathbf{0} & \mathbf{f}_c \end{bmatrix}, \tag{5}$$

where identity matrices $\mathbf{I}_b$ and $\mathbf{I}_c$ represent unit displacements imposed. We would like to remind readers again that unlike Eq. (1), Eq. (5) is only for a local domain. The resulting displacements at internal nodes (i.e. $\mathbf{\Phi}_i^b$ and $\mathbf{\Phi}_i^c$) can be computed by solving the top two rows in Eq. (5) as: $\mathbf{\Phi}_i^b = \mathbf{K}_{ii}^{-1}\mathbf{K}_{ib}$ and $\mathbf{\Phi}_i^c = \mathbf{K}_{ii}^{-1}\mathbf{K}_{ic}$.

Reduced constraint modes An issue associated with constraint modes is that the total number of interface DOFs is still quite large, especially when we are dealing with high-resolution meshes of quadratic elements. As shown in the Fig.4.a, a quadratic tetrahedral element does not only have four regular *tetrahedral nodes* but also additional six *mid-edge nodes*. A large number of interface DOFs result in a noticeable delay when synchronizing the FEM analysis with mesh editing operations even within the constraint subspace. To further condense the dimension of the subspace, our reduced constraint modes are constructed only for tetrahedral nodes on the interface, each of which is still imposed by a unit displacement. Its adjacent mid-edge nodes are also prescribed by a half unit displacement as an estimate of the influence received from the tetrahedral nodes. In other words, $\mathbf{I}_b$ and $\mathbf{I}_c$ in Eq. (5) are no longer identity matrices and they are not square neither since the columns corresponding to mid-edge nodes are removed. Fig.4.b illustrates a 2D example of reduced constraint modes.

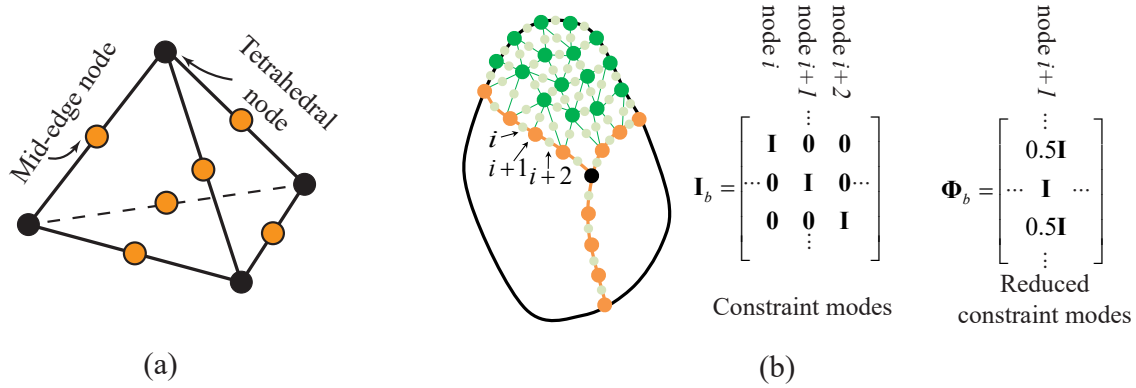


Figure 4: (a) 10 nodes quadratic element. (b) A toy 2D example showing how reduced constraint modes are constructed. Suppose there are three boundary nodes i, $i+1$, and $i+2$. Nodes i and $i+2$ are mid-edge nodes and node $i+1$ is a tetrahedral node. The classic constraint mode imposes a unit displacement at each of the boundary nodes yielding a 3×3 identity matrix at the sub-block corresponding to each node. In our reduced constraint mode, the unit displacement is only imposed at the tetrahedral node $i+1$. Therefore, columns corresponding to nodes i and $i+2$ are eliminated. Besides, we assume that the unite displacement at node $i+1$ also triggers small deflections at its adjacent mid-edge nodes i and $i+2$. Therefore, both of them are prescribed a half unit displacement $0.5\mathbf{I}$.

Let $\mathbf{\Phi}_b$ and $\mathbf{\Phi}_c$ denote the merged matrices whose row numbers equal to the total numbers of boundary DOFs and corner DOFs, but column numbers only reflect the number of DOFs at tetrahedral nodes. Accordingly, the corresponding internal displacements $\mathbf{\Phi}_i^b$ and $\mathbf{\Phi}_i^c$ are computed via:

$$\mathbf{\Phi}_i^b = \mathbf{K}_{ii}^{-1}\mathbf{K}_{ib}\mathbf{\Phi}_b, \qquad \mathbf{\Phi}_i^c = \mathbf{K}_{ii}^{-1}\mathbf{K}_{ic}\mathbf{\Phi}_c. \tag{6}$$

Reduced stiffness matrix A typical way of constructing domain mode vectors is to stack the displacements at internal, boundary and corner DOFs together. However, doing so also induces the so-called coupling problem at domain interfaces, which needs to be handled by using the Lagrange multiplier method to enforce a set of linear constraints so that the displacement is continuous across an interface [46]. Augmenting per-domain equilibrium with a Lagrange multiplier vector leaves zero entries at the diagonal of the system matrix (corresponding to the interface constraints), impedes its condition number and slows the follow-up multigrid iterations. To avoid additional Lagrange multiplier variables, we explicitly decouple nonzero displacements corresponding to the internal DOFs and boundary/corner DOFs in the constraint mode matrix, and we denote this reduced modes set with $\widetilde{\mathbf{U}}$:

$$\begin{bmatrix} \mathbf{\Phi}_i^b & \mathbf{\Phi}_i^c \\ \mathbf{\Phi}_b & \mathbf{0} \\ \mathbf{0} & \mathbf{\Phi}_c \end{bmatrix} \Rightarrow \underbrace{\begin{bmatrix} \mathbf{\Phi}_i^b & \mathbf{\Phi}_i^c & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{\Phi}_b & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{\Phi}_c \end{bmatrix}}_{\widetilde{\mathbf{U}}}. \tag{7}$$

It is easy to see that $\texttt{span}(\widetilde{\mathbf{U}}) \supset \texttt{span}(\mathbf{U})$ meaning any vector that can be represented with $\mathbf{U}$ can also be *losslessly* represented with $\widetilde{\mathbf{U}}$. This splitting allows us to put basis matrices for all the internals, boundary and corner nodes clusters into a big block diagonal matrix:

$$\mathbf{\Phi} = \texttt{diag}\left(\mathbf{\Phi}_i^{1,b\cup c}, \mathbf{\Phi}_i^{2,b\cup c}, ..., \mathbf{\Phi}_b^1, \mathbf{\Phi}_b^2, ..., \mathbf{\Phi}_c^1, ...\right). \tag{8}$$

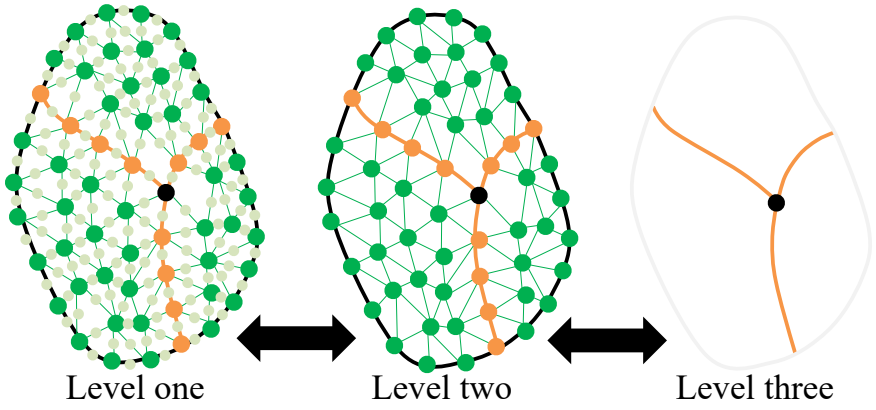


Figure 5: Left: A 2D illustration of the three-level multigrid construction. The 2^{nd} level is formed by removing the mid-edge points, and only boundary and corner nodes are kept in 3^{rd} level. Right: A quadric element. The yellow dots indicate the mid-edge points.

Here, we restore the superscript of the local index of a cluster. $\mathbf{\Phi}_i^{l,b\bigcup c}$ represents the union of the basis in $\mathbf{\Phi}_i^{l,b}$ and $\mathbf{\Phi}_i^{l,c}$ at l^{th} internal node cluster such that $\mathbf{\Phi}_i^{l,b\bigcup c} = \left[\mathbf{\Phi}_i^{l,b}, \mathbf{\Phi}_i^{l,c}\right]$. Because each domain only has one internal node cluster, the number of internal node clusters matches the number of decomposed domains. However the number of boundary and corner node clusters are independent to domains' number. In this setting, we can represent the reduced DOFs at boundary and corner node clusters with the same unknowns in the reduced system and avoid the usage of Lagrange multipliers.

Thanks to the block diagonal structure of $\mathbf{\Phi}$, the reduced stiffness matrix for d^{th} domain, which is also adjacent to boundary cluster j and corner cluster l can be written as:

$$\begin{cases} \widetilde{\mathbf{K}}_{ii}^{d} = \left[\mathbf{\Phi}_i^{d,b\bigcup c}\right]^{\top} \mathbf{K}_{ii}^{d} \left[\mathbf{\Phi}_i^{d,b\bigcup c}\right], \\ \widetilde{\mathbf{K}}_{ib}^{d,j} = \left[\mathbf{\Phi}_i^{d,b\bigcup c}\right]^{\top} \mathbf{K}_{ii}^{d} \mathbf{\Phi}_b^{j}, \qquad \widetilde{\mathbf{K}}_{ic}^{d,l} = \left[\mathbf{\Phi}_i^{d,b\bigcup c}\right]^{\top} \mathbf{K}_{ii}^{d} \mathbf{\Phi}_c^{l}, \\ \widetilde{\mathbf{K}}_{bb}^{j} = {\mathbf{\Phi}_b^{j}}^{\top} \mathbf{K}_{bb}^{j} \mathbf{\Phi}_b^{j}, \qquad \widetilde{\mathbf{K}}_{cc}^{l} = {\mathbf{\Phi}_c^{l}}^{\top} \mathbf{K}_{cc}^{j} \mathbf{\Phi}_c^{l}. \end{cases} \tag{9}$$

By avoiding the usage of Lagrange multipliers, our reduced stiffness matrix is still a symmetric positive definite (SPD) matrix, which can be solved efficiently using the Cholesky factorization method.

6 Refinement with Multigrid Solver

The goal of this step is to eliminate the residual error in the subspace solution to obtain an accurate stress analysis result. As we want to avoid the matrix factorization of the global stiffness matrix after user updates the geometry or topology of the original mesh, the multigrid solver is a reasonable choice due to its appealing properties of: 1) it converges fast for large-scale meshes, 2) its smoothing operator can be an iterative Jacobian/Gauss-Siedel solver, which is fast to compute [15, 35]. In this section, we show that the geometry of quadric elements allows us to construct a three-level multigrid solver that can be easily updated after user editing.

6.1 Three-level multigrid construction

It is well known that multigrid methods accelerate the convergence rate of iterative relaxation solver using a coarse to fine hierarchy formulation [15]. In multigrid methods, the low-frequency error that cannot be efficiently reduced at a fine level is restricted to the coarse level and solved recursively. Since the problems at coarse levels have much smaller size than the fine-level ones, such hierarchical formulation significantly speeds up the solver. For the problems defined on regular 2D or 3D grids, the hierarchy can be easily obtained by tweaking the grid spacing. In contrast, the construction of a multi-resolution representation of a finite element mesh is a non-trivial task, which necessitates the sophisticated mesh simplification or graph-based coarsening techniques. Besides, the hierarchy constructed from such techniques has many cross-domain elements which violates the domain decomposition. Our multigrid method constructs the prolongation/restriction operators at three different levels while maintaining the original domain partition, and we use the *V-cycle* in our implementation.

First-to-second level The first level is the original quadratic finite element mesh. The second level is constructed by removing the mid-edge nodes – it usually reduces $70\% - 80\%$ DOFs from the first level, where this percentage increases along with the mesh resolution. The prolongation operator from the second level to third level is designed as:

$$u_i^{(1)} = \begin{cases} \dfrac{u_j^{(2)} + u_k^{(2)}}{2} & \text{for mid-edge nodes} \\ u_i^{(2)} & \text{for tetrahedral nodes} \end{cases}, \tag{10}$$

where the superscript $[\cdot]^{(k)}$ indicates the grid level k. For the mid-edge nodes, indices j and k are two endpoints of the edge $< \mathbf{v}_j, \mathbf{v}_k >$, which the mid-point $\mathbf{v}_i$ belongs to. For a tetrahedral node, its value is directly copied to the second level in prolongation.

We now derive the coarse problem at the second level. Given the adjacent relationship in the first level mesh, a row of its stiffness matrix can be written as:

$$\sum_{j\in\mathcal{A}_i}\mathbf{S}_{ij}^{(1)}u_j^{(1)}+\mathbf{S}_{ii}^{(1)}u_i^{(1)}=f_i^{(1)}, \tag{11}$$

where $\mathbf{S}_{ij}^{(1)}$ is a 3×3 matrix extracted from the first level stiffness matrix. The set of nodes adjacent to node i is denoted by $\mathcal{A}_i$. $\mathcal{A}_i$ consists of both mid-edge and tetrahedral nodes sharing an element with node i. Thus, Eq. (11) can be re-written as:

$$\sum_{j\in\mathcal{M}_i}\mathbf{S}_{ij}^{(1)}u_j^{(1)}+\sum_{j\in\mathcal{N}_i}\mathbf{S}_{ij}^{(1)}u_j^{(1)}+\mathbf{S}_{ii}^{(1)}u_i^{(1)}=f_i^{(1)}, \tag{12}$$

where the set of mid-edge node are denoted by $\mathcal{M}_i\subset\mathcal{A}_i$, and the set of tetrahedral nodes are denoted by $\mathcal{N}_i\subset\mathcal{A}_i$. According to the prolongation operator in Eq. (10), the mid-edge node in $\mathcal{M}_i$ is represented by two tetrahedra nodes in $\mathcal{N}_i$. The proof is simple: if a mid-edge node $\mathbf{v}$ is in $\mathcal{M}_i$, both $\mathbf{v}$ and the node i must be in the same tetrahedra $\mathbf{t}_k$. Since the edge passing throught $\mathbf{v}$ is in $\mathbf{t}_k$, the two tetrahedral nodes used in the prolongation operator for $\mathbf{v}$ should be in $\mathcal{N}_i$. Accordingly, Eq. (12) can further be re-arranged as:

$$\begin{aligned}\sum_{j\in\mathcal{N}_i}\left(\mathbf{S}_{ij}^{(1)}+\frac{1}{2}\sum_{k\in\mathcal{A}_j\bigcap\mathcal{M}_i}\mathbf{S}_{ik}^{(1)}\right)u_j^{(1)}\\ +\left(\mathbf{S}_{ii}^{(1)}+\frac{1}{2}\sum_{k\in\mathcal{M}_i}\mathbf{S}_{ik}^{(1)}\right)u_i^{(1)}=f_i^{(1)}.\end{aligned} \tag{13}$$

Note $u_j^{(1)}$ in Eq. (13) only contains the nodes in $\mathcal{N}_i$ that will be restricted to the second multigrid level. Afterwards, simply adding all the equations at mid-edge nodes to the equations of the end nodes of its edge with weight 0.5 and discarding all the equations at mid-edge nodes forms the reduced problem at the second level. The restriction operator is the transpose of the prolongation operator. It guarantees that the problem at second level has a SPD matrix. An important advantage of Eq. (13) is that the block-wise structure arising from domain decomposition is well maintained, since the prolongation operator for mid-edge nodes *does not* break domain boundaries. It means that we can still perform local update at the second level.

Second-to-third Level Further coarsening or simplifying the second-level tetrahedral mesh may yield cross-domain elements, which downgrades the advantages of the domain decomposition technique: the classification of internal, boundary and corner nodes, as well as the corresponding block-wise structure are difficult to maintain.

Instead, we leverage the coarse problem as the third level of our multigrid solver using the Schur complement method, a widely used technique in algebraic multigrid methods [15]. Specifically, since the domain decomposition result is maintained at the second level, its equilibrium has a similar block-wise structure as of Eq. (1):

$$\mathbf{K}^{(2)}=\begin{bmatrix}\mathbf{K}_{ii}^{(2)} & \mathbf{K}_{ib}^{(2)} & \mathbf{K}_{ic}^{(2)}\\ \mathbf{K}_{ib}^{\top(2)} & \mathbf{K}_{bb}^{(2)} & \mathbf{K}_{bc}^{(2)}\\ \mathbf{K}_{ic}^{\top(2)} & \mathbf{K}_{bc}^{\top(2)} & \mathbf{K}_{cc}^{(2)}\end{bmatrix}. \tag{14}$$

The restriction and prolongation operators are defined as:

$$\mathbf{R}=\begin{bmatrix}-\mathbf{K}_{ib}^{\top(2)}\mathbf{K}_{ii}^{-1(2)} & \mathbf{I}_b\\ -\mathbf{K}_{ic}^{\top(2)}\mathbf{K}_{ii}^{-1(2)} & \mathbf{I}_c\end{bmatrix},\qquad \mathbf{P}=\mathbf{R}^{\top}. \tag{15}$$

We can then obtain the system matrix at the third level as follows $\mathbf{K}^{(3)}=\mathbf{R}\mathbf{K}^{(2)}\mathbf{P}$. In other words, the matrix $\mathbf{K}^{(3)}$ is the Schur complement of the matrix $\mathbf{K}_{ii}^{(2)}$, and *we compute it in parallel using domain decomposition at second level as described in Alg. 1*. Therefore, our prolongation/restriction operator can be considered as optimal operators in the theory of Schur complement, where only one iteration is required between second and third level.

Domain-level parallel Gauss-Siedel smoothing operator The stiffness matrix in linear elasticity is SPD, but we can not guarantee that it is a strictly diagonally dominant matrix. Therefore, we adopt the Gauss-Siedel iterative solver which is proved to converge for SPD matrices [21] instead of the Jacobi solver. The Gauss-siedel method has been used as the smoothing operator in FEM multigrid solver [25], and red-black ordering technique that avoids data race is used to implement a parallel Gauss-Siedel solver [12]. In our case, the domain decomposition provides a natural domain level red-black ordering, since decomposed domains are surrounded by boundary and corner nodes, their Gauss-Siedel iterations do not depend on each other. Therefore, we first perform Gauss-Siedel iteration for internal nodes for each domain in parallel, and then proceed to boundary and corner nodes in parallel.

7 Implementation Details

It is a nontrivial task for users to decompose an input model into domains manually. For the skeleton editing interface, the initial domains can be obtained through the automatic rigging [3], and we develop an algorithm to compute the cross section surface at wires to cut the input model into disconnected parts as domains. The cross section surface should be a manifold, consisting of triangles of the input tetrahedral mesh. The qualified wires are defined to be independent wires with two criterions: 1) They do not have common edges with other wires. 2) The cross section at them should have neighboring tetrahedra on its two sides.

As illustrated in Fig. 6, the independent wires are four wires that on the upper plane of the rectangular solid of the model, and we compute the cross section surface via graph cut. Specifically, a set of tetrahedrons intersected with a plane fitted at an independent wire is used as the seed and their 3-ring neighborhoods are nodes to form the graph. To figure out the cross section surface using graph cut, we assign source and sink nodes at the two boundary of the detected set of tetrahedrons without surface triangles, and set the edge weight to be the triangle area for two neighboring tetrahedrons sharing a common triangle. The triangles that have wire edges and are

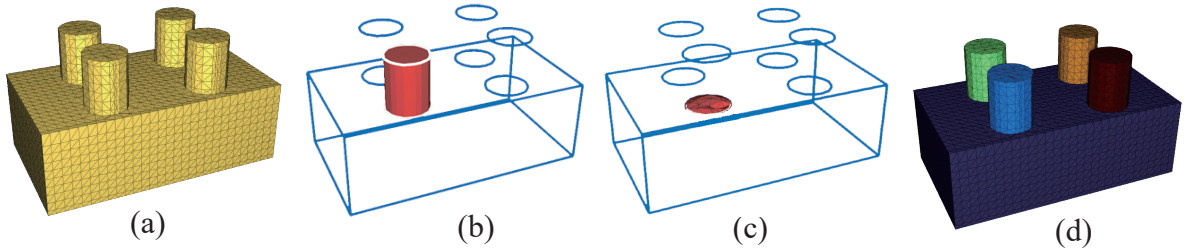


Figure 6: Decomposing the mesh into domains. (a) An input model. (b) The cross section surface computed via graph cut at one independent wires. (c) A detected domain by region growing. (d) The domain decomposition result. The largest domain is implicitly decomposed using variation shape segmentation algorithms into small sub-domains to balance the load in parallel implementation of our solver [10].

most parallel to be fitted plane at the wire is set to be on the cross section by hard constraint: its two neighboring tetrahedrons should belong to source and sink respectively. Our setting enables graph cut algorithm to find the optimal cross section surface as shown in Fig. 6 (b). The user can then refine the automatic domain decomposition result.

If the decomposed domains are of different sizes, it will influence the performance of the parallel implementation of our solver due to the unbalanced load. We adopt variational shape segmentation algorithm as in [10] but adapt it to the tetrahedral mesh to subdivide those large domains into sub-domains to balance the computational load. However, the editing is still maintained at the user refined domain level since it meets the semantic requirement of user editing.

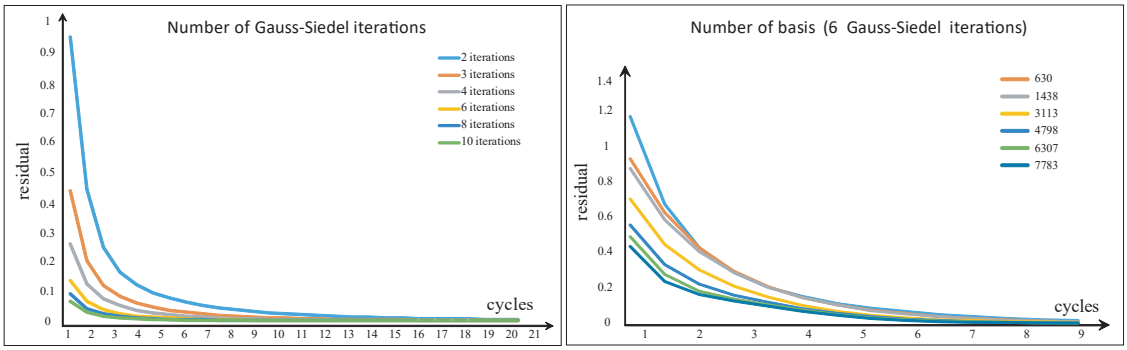


Figure 7: Convergence curves of the three-level multigrid solver. Top: How the initial solution from subspace solver influences the convergence speed of the multigrid solver. Bottom: The convergence speed of the multigrid solver with respect to the number of Gauss-Siedel iterations.

For large scale meshes, it is possible that the domain boundary has thousands of DOFs. The reduced constrained modes, in this case, might still have a significant memory footprint, since the modes are dense. Therefore, we resort to rigid and soft modes, which is actually the domain response to the rigid and harmonic deformation at boundaries to reduce the memory cost [13]. Please also refer to [46] for the details of these two kinds of modes. In our implementation, we limit the number of modes at each boundary to be at $200 - 300$.

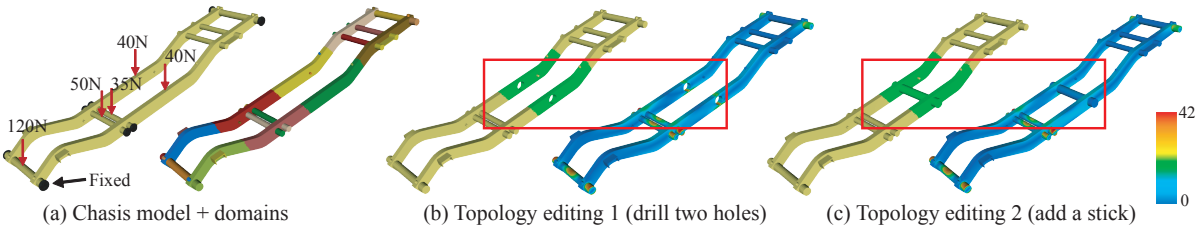


Figure 8: The integration of topology editing and stress analysis. Since the topology editing occurs at the selected domains (shown in (b) & (c)), our FEM data structure, such as subspace basis and multigrid restriction and propagation operators, can be easily updated at the edited domains, it accelerates the solver, achieving a 27.1% speedup (our method uses 8.72 s vs. `MKL`, which uses 11.09 s. See first row in Tab. 2). The chasis model has $515,142$ DOFs. The cylinders added is treated as a new domain in our implementation. The stress values are converted to color maps according to the rightmost legend, where the unit for its numbers is MPa.

8 Experiments

In this section, we report the results and time statistics of our system implemented on a desktop PC with `i7-5820`, a 6 core, 3.3 GHZ CPU and 64G memory. We set the material to be the photosensitive resin for stereolithography (SLA) 3D printing, whose Young's modulus is $2.5GPa$ and Poisson ratio is 0.41, for all the models used in the paper. Please also refer to the accompanied video for the editing and stress analysis demo.

Mesh editing and stress analysis integration Figs. 1, 2, 8, 9, 10 show several mesh editing results for stress relief. The editing are obtained through IWires or skeleton editing interfaces. The visualized stress is computed based on the definition of the *von Mises*

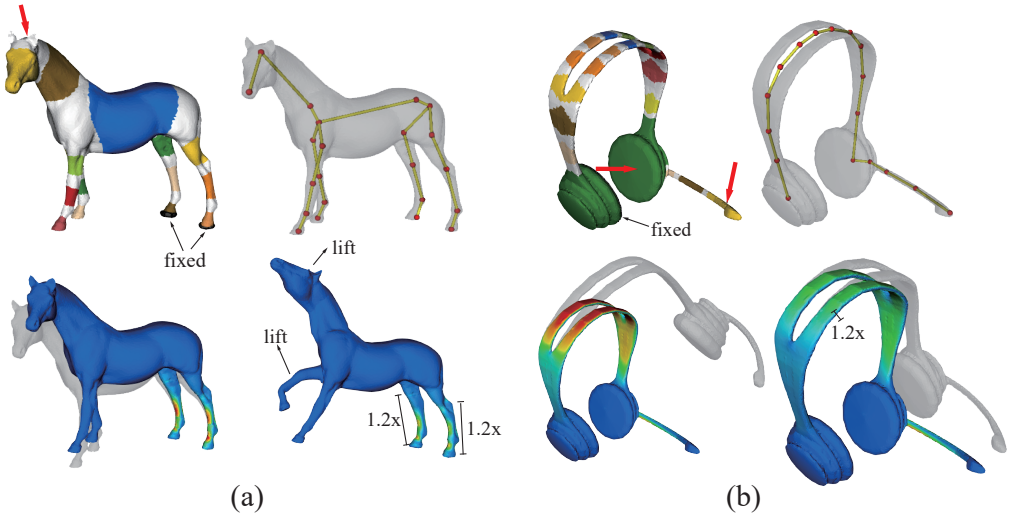


Figure 9: Stress relief oriented shape editing. The horse leg and earphone are thickened by scaling operation at rigid link associating to the underlying skeleton. The light gray shapes indicate the deformation results, and red regions imply high stress values.

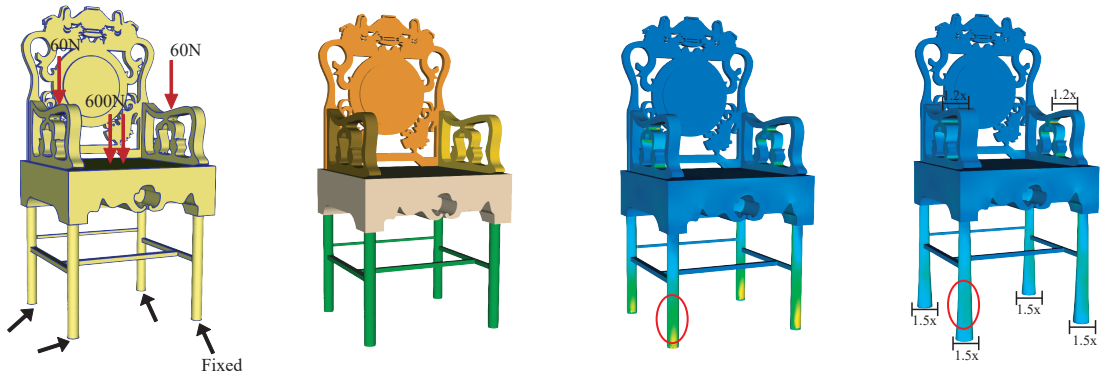


Figure 10: The integration of IWires editing and stress analysis on a chair model. The blue lines on the chair model indicate the extracted wires.

stress:

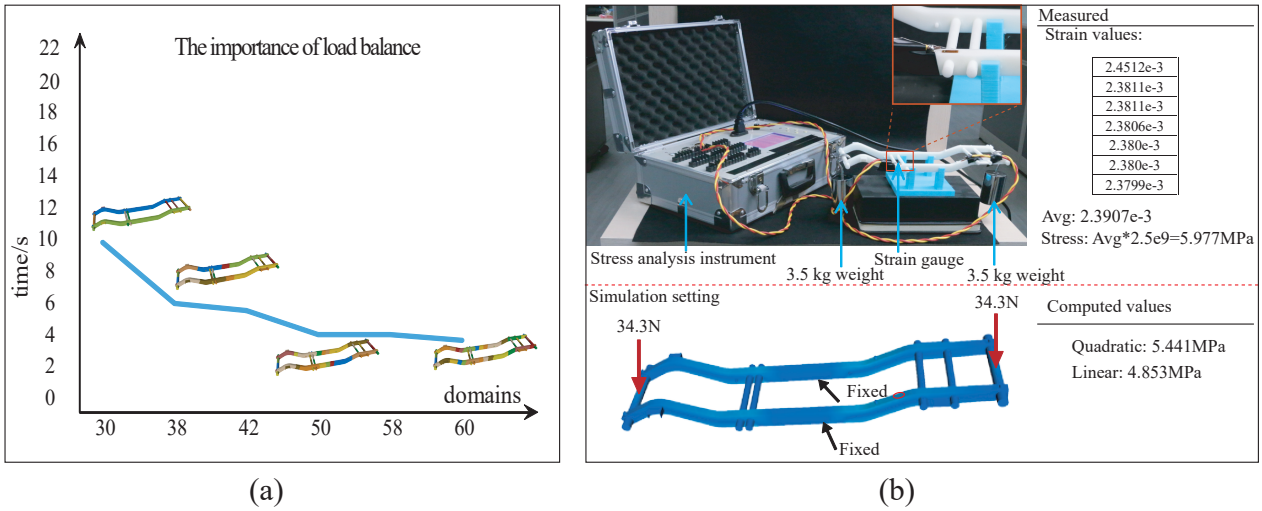
$$\sigma^2_{\text{von Mises}} = \frac{\sigma^2_{1,2} + \sigma^2_{2,3} + \sigma^2_{3,1} + 6(\sigma^2_{12} + \sigma^2_{23} + \sigma^2_{32})}{2}, \tag{16}$$

where σ is the 3×3 stress tensor and $\sigma_{i,j} = \sigma_{ii} - \sigma_{jj}$. Fig. 8 illustrates a topology editing result on a chasis model. Since the topology editing operations, such as drilling holes and add a supporting cylinder, are all local, we only need to update the subspace and multigrid solver data structure at the edited domains in green colors in Fig. 8 to obtain the updated stress analysis results. In contrast, the `PARDISO` solver from `MKL` has to perform symbolic and numerical factorization from scratch. Thus, it is 21% slower than our solver in this case.

The simulator provides a fast stress preview using reduced constrained modes. On the top of this, the multigrid solver further improves the subspace solution as shown in Fig. 2. The high frequency errors in subspace solution are sufficiently reduced after multigrid iterations.

Convergence of the multigrid solver The convergence speed is influenced by many factors, such as mesh quality, initial solution. In the top row of Fig. 7, we show that the initial solution from subspace is important to speed up the convergence of the multigrid solver, using the chasis model as an example. As the number of basis grows, the initial solution is more accurate and it reduces the convergence time of the multigrid solver. The initial solution for zero subspace basis is set to be a zero vector, and its convergence is around $2\times$ slower than the curve of subspace with reduced constraint modes ($8,613$ basis for the chasis model). The rest of curves are measured by increasingly sampling the reduced constraint modes. We also add rigid modes used in [46] in this test so that every boundary node has its displacements encoded in the subspace. The second row of Fig. 7 illustrates how the number of Gauss-Siedel iteration in the multigrid solver influences the convergence speed of the multigrid solver. Not surprisingly, smoothing the residual error with more iterations accelerate the convergence, since it further reduces the error at fine levels.

Fig. 11.a reports how the difference in domain size influences the performance of the multigrid solver, also tested on the chasis model. For 30 domains, the max/min domain sizes are $183,321/236$. We then try to reduce the difference in domain sizes by further subdividing the large domains into sub-domains using variational shape approximation method [10]. After that, there are 60 (sub-) domains, and the max/min domain sizes are $33,225/246$. The difference in domain sizes is greatly reduced, which leads to more balanced computational costs in the parallel implementation. On the other hand, the increasing number of domains also means more nodes at domain boundaries, and it yields a larger-size reduced stiffness matrix. In our implementation, we require the ratio of boundary nodes in the mesh to be less than 20% when subdividing the large domains.



(a) (b)

Figure 11: (a) The influence of domain size to the speed of the multigrid solver. With increasing number of domains, the difference in domain size is reduced. The computational load is more balanced to improve the performance of the parallel multigrid solver. This test is done on the Chasis model and its subdomains in the figures near the curve are automatically generated. (b) Physical validation using a stress measurement instrument. It shows that the stress error using linear elements is reduced by $0.59MPa$ after using quadratic elements for the chasis model.

Time performance Tab. 1 reports the geometric statistics of the models we have tested, and the timing statistics of the Intel MKL solver in stress analysis. The timing satistics for subspace and multigrid solvers for each model are listed in Tab. 2.

Model	Mesh Statistics			Domain Statistics		Single-Domain MKL	
	# ele	# node	# matrix size	# domain	# DOFs	Sym/Solving	Total
Chasis	102,452	171,714	515,142	60	8,121	3.91 s/7.18 s	11.09 s
Chair	209,443	351,059	1,053,177	28	37,613	9.1 s/41.85 s	50.95 s
Earphone	343,522	555,718	1,667,154	40	39,906	14.26 s/65.85 s	80.11 s
Armadillo	774,198	1,119,699	3,359,097	47	71,740	45.38 s/285.12 s	330.5 s
Horse	742,176	1,161,637	3,484,911	40	83,393	34.79 s/262.12 s	296.91 s
Manipulator	1,002,774	1,582,335	4,747,065	34	134,230	45.38 s/292.79 s	338.17 s

Table 1: Model statistics. Detailed statistic of the 3D models we used in the experiment. **# ele:** the number of quadric elements on the mesh; **# node:** the number of nodes on the mesh; **# domain:** the number of domains of the model; **# DOFs:** the average number of DOFs each domain holds. For the solving time of MKL pardiso sovler, we choose Cholesky matrix decomposition method and report its time in two parts **Sym/Solving**: symbolic factorization/numerial factorization+solving. **Total:** The sum of these two parts which is required in the integration of shape and topology editing and stress analysis. *Please note that only numerial factorization+solving time is required to obtain the stress analysis result if no topology editing. The timing is measured in seconds.*

Since we localize the updating of the solver data structure, our solver outperforms the `MKL PARDISO` solver for large-scale meshes. The stop criterion of the multigrid solver is set to be $1e-3$ relative residual error. For the integration of editing and stress analysis, our subspace and multigrid solver time to obtain the stress values can be calculated by the sum of their updating time and solving time, and the solving time for `MKL PARDISO` solver is just the sum of numerical factorization and solving time when shape editing only. In the case of topology editing, the symbolic factorization time has to be counted since the sparsity pattern of the stiffness matrix is changed.

From Tab. 2, except the chassis model with half million DOFs, the time required, including both the updating and solving time of the subspace and multigrid solvers, to compute the stress distribution after user editing for the models with million DOFs is around $50\% - 100\%$ faster than the `PARDISO` solver using Cholesky decomposition method (please see the total column in Tab. 1 and 2). It shows the advantage of localized updating enabled by domain decomposition. In addition, the subspace solver can be solved in a few seconds to provide fast stress previews, and the multigrid solver enables users to change its stopping criterion to balance between computation time and the accuracy.

Physical validation To show how the number DOFs influences the accuracy of the stress analysis, we print out the chasis model with SLA material and measure the strain on selected locations using the strain gauge, as shown in Fig. 11. The measured strain under the $3.5Kg$ load at the two ends of the model is $2.3907e-3$, the average of seven times measurements is used to remove the noise. Since the material properties of the fabricated model using SLA is isotropic [18], we multiply the measured strain with the Young's modulus to estimate the stress at the measured points, assuming that the shear strain is small under our applied load. Fig. 11.b shows that the stress error using linear elements is reduced by $0.59MPa$ after using quadratic elements. It indicates that the increased DOFs and quadratic shape functions in the quadratic elements improve the accuracy of the stress analysis results.

9 Conclusion and Future Work

In this paper, we explore how to combine subspace and multigrid solvers to provide users progressive response when analyzing the stress distribution, under static equilibrium, for meshes with millions of DOFs. Our system employs domain decomposition technique

Model	Multigrid			Subspace			Total
	I/U time	coarse size	Iteration	basis	I/U time	Sovling	
Chasis	$6.23\ s/0.60\ s$	$8,449$	$5.62\ s$	$8,613$	$69.87\ s/2.23\ s$	$0.27\ s$	$8.72\ s$
Chair	$5.58\ s/1.90\ s$	$9,336$	$20.88\ s$	700	$69.87\ s/2.02\ s$	$0.09\ s$	$24.89\ s$
Earphone	$11.63\ s/2.42\ s$	$20,076$	$39.12\ s$	$8,470$	$277.32\ s/20.6\ s$	$0.87\ s$	$63.01\ s$
Armadillo	$32.64\ s/15.25\ s$	$61,035$	$47.62\ s$	$4,776$	$462.59\ s/21.31\ s$	$0.82\ s$	$85.00s$
Horse	$27.8\ s/8.00\ s$	$41,094$	$67.98\ s$	$8,414$	$981.1\ s/75\ s$	$1.82\ s$	$152.80\ s$
Manipulator	$36.06\ s/12.34\ s$	$51,435$	$53.02\ s$	$10,470$	$975.03\ s/105.61\ s$	$3.2\ s$	$174.17\ s$

Table 2: The performance of the proposed subspace and multigrid solver. **I/U time:** The init and average updating time after user editing for multigrid and subspace solver. **coarse size:** the size of the third level problem; **Iteration**: The multigrid solver iteraion time after editing. **basis:** The number of subspace basis. **Solving**: the subspace solving time after its basis updating. **Total**: The total stress analysis time after shape and topology editing, which is the sum of subspace and multigrid solver updating and solving time. The timing is measured in seconds.

to integrate shape editing and FEM simulation. It avoids the expensive global matrix factorization of direct solver by localizing the shape editing and FEM data structure updating at edited domains. The system constructs the subspace for stress analysis in static equilibrium using reduced constrained modes and builds three-level multigrid solver through the algebraic multigrid method by removing mid-edge nodes and lumping unknowns with Schur complement method.

In the future, we want to further reduce the updating cost of FEM data structure according to the change of design parameters. The optimal Schur complement based multigrid construction in our implementation is expensive, and its efficient approximation is a reasonable choice to balance between updating cost and the convergence speed of the multigrid solver. We would also like to investigate how to develop fast simulation and design integration algorithms for transient or nonlinear stress analysis.

References

[1] M. Adams. Evaluation of three unstructured multigrid methods on 3d finite element problems in solid mechanics. *International Journal for Numerical Methods in Engineering*, 55(5):519–534, 2002.

[2] M. Bächer, B. Bickel, D. L. James, and H. Pfister. Fabricating articulated characters from skinned meshes. *ACM Trans. Graph.*, 31(4):47:1–47:9, July 2012.

[3] I. Baran and J. Popović. Automatic rigging and animation of 3d characters. *ACM Trans. Graph.*, 26(3), July 2007.

[4] J. Barbič and Y. Zhao. Real-time large-deformation substructuring. In *ACM SIGGRAPH 2011 Papers*, SIGGRAPH '11, pages 91:1–91:8, 2011.

[5] B. Bickel, M. Bächer, M. A. Otaduy, H. R. Lee, H. Pfister, M. Gross, and W. Matusik. Design and fabrication of materials with desired deformation behavior. *ACM Trans. Graph.*, 29(4):63:1–63:10, July 2010.

[6] M. Bro-Nielsen and S. Cotin. Real-time volumetric deformable models for surgery simulation using finite elements and condensation. *Computer Graphics Forum*, 15(3):57–66, 1996.

[7] J. Calì, D. A. Calian, C. Amati, R. Kleinberger, A. Steed, J. Kautz, and T. Weyrich. 3d-printing of non-assembly, articulated models. *ACM Trans. Graph.*, 31(6):130:1–130:8, Nov. 2012.

[8] D. Ceylan, W. Li, N. J. Mitra, M. Agrawala, and M. Pauly. Designing and fabricating mechanical automata from mocap sequences. *ACM Trans. Graph.*, 32(6):186:1–186:11, Nov. 2013.

[9] D. Chen, D. I. W. Levin, P. Didyk, P. Sitthi-Amorn, and W. Matusik. Spec2fab: A reducer-tuner model for translating specifications to 3d prints. *ACM Trans. Graph.*, 32(4):135:1–135:10, July 2013.

[10] D. Cohen-Steiner, P. Alliez, and M. Desbrun. Variational shape approximation. *ACM Trans. Graph.*, 23(3):905–914, Aug. 2004.

[11] S. Coros, B. Thomaszewski, G. Noris, S. Sueda, M. Forberg, R. W. Sumner, W. Matusik, and B. Bickel. Computational design of mechanical characters. *ACM Trans. Graph.*, 32(4):83:1–83:12, July 2013.

[12] H. Courtecuisse and J. Allard. Parallel dense gauss-seidel algorithm on many-core processors. In *HPCC*, pages 139–147. IEEE, 2009.

[13] R. Craig, R. A review of time-domain and frequency-domain component mode synthesis methods. *The International Journal of Analytical and Experimental Modal Analysis*, 2(2):59–72, Jan. 1987.

[14] Y. Dong, J. Wang, F. Pellacini, X. Tong, and B. Guo. Fabricating spatially-varying subsurface scattering. *ACM Trans. Graph.*, 29(4):62:1–62:10, July 2010.

[15] R. D. Falgout. An introduction to algebraic multigrid. *Computing in Science and Engineering*, 8(6):24–33, 2006.

[16] C. Farhat, M. Lesoinne, P. LeTallec, K. Pierson, and D. Rixen. Feti-dp: a dual¨cprimal unified feti method¡ªpart i: A faster alternative to the two-level feti method. *International Journal for Numerical Methods in Engineering*, 50(7):1523–1544, 2001.

[17] C. Farhat and F.-X. Roux. A method of finite element tearing and interconnecting and its parallel solution algorithm. *International Journal for Numerical Methods in Engineering*, 32(6):1205–1227, 1991.

[18] Formlabs. Validating isotropy in sla 3d printing. https://formlabs.com/blog/isotropy-in-SLA-3D-printing, October 2016.

[19] R. Gal, O. Sorkine, N. J. Mitra, and D. Cohen-Or. iwires: An analyze-and-edit approach to shape manipulation. *ACM Transactions on Graphics (Siggraph)*, 28(3):#33, 1–10, 2009.

[20] M. Geveler, D. Ribbrock, D. Göddeke, P. Zajac, and S. Turek. Towards a complete fem-based simulation toolkit on gpus: Unstructured grid finite element geometric multigrid solvers with strong smoothers based on sparse approximate inverses. *Computers & Fluids*, 80:327–332, 2013.

[21] G. H. Golub and C. F. V. Loan. *Matrix Computations*. The Johns Hopkins University Press, 3rd edition, 1996.

[22] I. Jeon, K.-J. Choi, T.-Y. Kim, B.-O. Choi, and H.-S. Ko. Constrainable multigrid for cloth. In *Computer Graphics Forum*, volume 32, pages 31–39. Wiley Online Library, 2013.

[23] E. Karer and J. K. Kraus. Algebraic multigrid for finite element elasticity equations: Determination of nodal dependence via edge-matrices and two-level convergence. *International Journal for Numerical Methods in Engineering*, 83(5):642–670, 2010.

[24] A. Kawaguchi, S. Itoh, M. Mochizuki, and M. Kameyama. Large-scale computation of welding residual stress. *Progress in Nuclear Science and Technology*, 2:613–619, 2011.

[25] K. Kim, K. H. Leem, P. G., and S. M. Algebraic multigrid preconditioner for a finite element method in tm electromagnetic scattering. *JOURNAL OF COMPUTATIONAL ANALYSIS AND APPLICATIONS*, 11(4):597–605, 2009.

[26] T. Kim and D. L. James. Physics-based character skinning using multi-domain subspace deformations. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '11, pages 63–72, 2011.

[27] Y. Lan, Y. Dong, F. Pellacini, and X. Tong. Bi-scale appearance fabrication. *ACM Trans. Graph.*, 32(4):145:1–145:12, July 2013.

[28] M. Lau, A. Ohgawara, J. Mitani, and T. Igarashi. Converting 3d furniture models to fabricatable parts and connectors. *ACM Trans. Graph.*, 30(4):85:1–85:6, July 2011.

[29] D. Li, D. I. W. Levin, W. Matusik, and C. Zheng. Acoustic voxels: Computational optimization of modular acoustic filters. *ACM Trans. Graph.*, 35(4):88:1–88:12, July 2016.

[30] A. McAdams, E. Sifakis, and J. Teran. A parallel multigrid poisson solver for fluids simulation on large grids. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 65–74. Eurographics Association, 2010.

[31] A. McAdams, Y. Zhu, A. Selle, M. Empey, R. Tamstorf, J. Teran, and E. Sifakis. Efficient elasticity for character skinning with contact and collisions. *ACM Trans. Graph. (SIGGRAPH)*, 30(4):37:1–37:12, July 2011.

[32] J. Panetta, Q. Zhou, L. Malomo, N. Pietroni, P. Cignoni, and D. Zorin. Elastic textures for additive fabrication. *ACM Trans. Graph.*, 34(4):135:1–135:12, July 2015.

[33] T. Patterson, N. Mitchell, and E. Sifakis. Simulation of complex nonlinear elastic bodies using lattice deformers. *ACM Trans. Graph. (SIGGRAPH Asia)*, 31(6):197:1–197:10, Nov. 2012.

[34] A. Reusken. A multigrid method based on incomplete gaussian elimination. *Numerical linear algebra with applications*, 3(5):369–390, 1996.

[35] L. Shi, Y. Yu, N. Bell, and W.-W. Feng. A fast multigrid algorithm for mesh deformation. In *ACM Transactions on Graphics (TOG)*, volume 25, pages 1108–1117. ACM, 2006.

[36] M. Skouras, B. Thomaszewski, S. Coros, B. Bickel, and M. Gross. Computational design of actuated deformable characters. *ACM Trans. Graph.*, 32(4):82:1–82:10, July 2013.

[37] O. Stava, J. Vanek, B. Benes, N. Carr, and R. Měch. Stress relief: Improving structural strength of 3d printable objects. *ACM Trans. Graph.*, 31(4):48:1–48:11, July 2012.

[38] R. Tamstorf, T. Jones, and S. F. McCormick. Smoothed aggregation multigrid for cloth simulation. *ACM Trans. Graph.*, 34(6):245:1–245:13, Oct. 2015.

[39] Y. Teng, M. Meyer, T. DeRose, and T. Kim. Subspace condensation: Full space adaptivity for subspace deformations. *ACM Trans. Graph.*, 34(4):76:1–76:9, July 2015.

[40] A. Toselli and O. Widlund. *Domain Decomposition Methods*. Springer, 2004.

[41] N. Umetani, K. Takayama, J. Mitani, and T. Igarashi. A responsive finite element method to aid interactive geometric modeling. *IEEE Comput. Graph. Appl.*, 31(5):43–53, Sept. 2011.

[42] C. Wagner, W. Kinzelbach, and G. Wittum. Schur-complement multigrid. *Numerische Mathematik*, 75(4):523–545, 1997.

[43] E. Wang, Q. Zhang, B. Shen, G. Zhang, X. Lu, Q. Wu, and Y. Wang. Intel math kernel library. In *High-Performance Computing on the Intel® Xeon Phi*, pages 167–188. Springer, 2014.

[44] Y. Xie, W. Xu, Y. Yang, X. Guo, and K. Zhou. Agile structural analysis for fabrication-aware shape editing. *Comput. Aided Geom. Des.*, 35(C):163–179, May 2015.

[45] U. M. Yang et al. Boomeramg: a parallel algebraic multigrid solver and preconditioner. *Applied Numerical Mathematics*, 41(1):155–177, 2002.

[46] Y. Yang, W. Xu, X. Guo, K. Zhou, and B. Guo. Boundary-aware multidomain subspace deformation. *Visualization and Computer Graphics, IEEE Transactions on*, 19(10):1633–1645, 2013.

[47] Q. Zhou, J. Panetta, and D. Zorin. Worst-case structural analysis. *ACM Trans. Graph.*, 32(4):137:1–137:12, July 2013.

[48] L. Zhu, W. Xu, J. Snyder, Y. Liu, G. Wang, and B. Guo. Motion-guided mechanical toy modeling. *ACM Trans. Graph.*, 31(6):127:1–127:10, Nov. 2012.

[49] Y. Zhu, E. Sifakis, J. Teran, and A. Brandt. An efficient multigrid method for the simulation of high-resolution elastic solids. *ACM Trans. Graph.*, 29(2):16:1–16:18, Apr. 2010.