Computer Aided Geometric Design 35–36 (2015) 163–179

Contents lists available at ScienceDirect

Computer Aided Geometric Design

www.elsevier.com/locate/cagd

Agile structural analysis for fabrication-aware shape editing

Yue Xie [a], Weiwei Xu [b,*], Yin Yang [c], Xiaohu Guo [d], Kun Zhou [a]

[a] State Key Lab of CAD&CG, Zhejiang University, China
[b] Hangzhou Normal University, China
[c] The University of New Mexico, United States
[d] The University of Texas at Dallas, United States

ARTICLE INFO

Article history:
Available online 3 April 2015

Keywords:
Shape editing
FEM
Stress analysis
3D printing

ABSTRACT

This paper presents an agile simulation-aided shape editing system for personal fabrication applications. The finite element structural analysis and geometric design are seamlessly integrated within our system to provide users interactive structure analysis feedback during mesh editing. Observing the fact that most editing operations are actually local, a domain decomposition framework is employed to provide unified interface for shape editing, FEM system updating and shape optimization. We parameterize entries of the stiffness matrix as polynomial-like functions of geometry editing parameters thus the underlying FEM system can be rapidly synchronized once edits are made. A local update scheme is devised to re-use the untouched parts of the FEM system thus a lot repetitive calculations are avoided. Our system can also perform shape optimizations to reduce high stresses in model while preserving the appearance of the model as much as possible. Experiments show our system provides users a smooth editing experience and accurate feedback.

1. Introduction

The fast developing of rapid prototyping technologies, such as 3D printing, enables convenient manufacturing of real objects from complex 3D digital models. As a prominent example, 3D printing has been extensively adopted in various areas such as architecture construction, industrial design and medical industries. The 3D printing technology converts an input digital model into layers, manufactures each layer and glues them together to shape a real object with various types of technologies, such as selective inhibition sintering (SIS), stereolithography (SLA) and fused deposition modeling (FDM) (Dutta et al., 2000). People can handily fabricate their own designs with emerging low-cost 3D printers and open-source softwares (Reprap, 2010; Vidimče et al., 2013).

To provide novice users feasible controls over the physical properties of the designed objects, many computational design tools have been developed in computer graphics community, i.e. devising objects with desirable structural stability (Stava et al., 2012; Zhou et al., 2013), deformation behaviors (Bickel et al., 2010; Chen et al., 2013) or kinetic constraints (Coros et al., 2013; Zhu et al., 2012). The finite element method (FEM) is widely adopted in such tools for accurate structural analysis, but it becomes time-consuming when the input digital models are complex. This can downgrade the system usability, especially in cases that users need to iterate the editing–simulating process. Moreover, most existing methods or commercial CAD/CAE packages only combine design and FEM simulation at the interface layer and users have to do the shape design and

* Corresponding author.
E-mail addresses: xie.full@gmail.com (Y. Xie), weiwei.xu.g@gmail.com (W. Xu), yangy@unm.edu (Y. Yang), xguo@utdallas.edu (X. Guo), kunzhou@acm.org (K. Zhou).

http://dx.doi.org/10.1016/j.cagd.2015.03.019
0167-8396/© 2015 Elsevier B.V. All rights reserved.

FEM simulation in separate stages. A recent contribution aims to update FEM simulation data structures during geometry editing (Umetani et al., 2011b). However, it only focuses on the design problems of moderate-size 2D models using linear elements. Fast design and simulation integrated system for large-scale 3D models using high order elements remains a technical challenge under investigation.

In this paper, we present a fabrication-aware shape editing system which can provide structure analysis feedback interactively for large 3D models. The main feature of our system is the seamless integration of FEM simulation and shape editing operations at domain level via the domain decomposition method, a well-known technique in FEM simulation for solving large scale matrix systems (Farhat and Roux, 1991). In our case, such integration enables to assemble the stiffness matrix of the FEM system locally at each domain. Furthermore, the entries of the stiffness matrix can be parameterized as closed-form expressions of the parameters of editing operations at domain-level, such as scaling and rotating. Thus, the FEM system updating speed is largely improved. The editing interface of our system is mainly a skeleton-driven editing interface where domains are connected to form kinematic chains. Scaling operations are also supported at each domain to support stress-relief operations (Stava et al., 2012). We also develop a domain-based optimizer that can optimize the domain geometry parameters to reduce the maximal stress value to a required threshold while preserving model shape. The parameterized entries allow us to compute the derivatives of stiffness matrix with respect to editing parameters easily in the optimization. The optimization algorithm can release the user efforts to manually adjust shapes to improve the structural stability.

The distinctive features of our system are:

- Observing that users often perform editing operations locally, such as posing, scaling and thickening parts of a mesh, we propose to adopt the non-overlapping domain decomposition as a unified interface for geometry editing, FEM system updating and shape optimization. Therefore, when only a part of the mesh is modified, our system can locally re-assemble the sub-matrices of FEM system belonging to the affected domains while leaving the rest parts untouched. Compared to re-assembling the whole system, a lot unnecessary calculations can be avoided. Our shape optimizer can also take advantage of domain decomposition to decrease the number of optimization variables for fast convergence.
- We derive closed-form formulas to parameterize each entry of the stiffness matrix as a polynomial-alike function of the domain scaling factors. With coefficients pre-computed, when a domain is scaled our system could update each entry of its stiffness matrix through 3 multiplications plus 2 additions. Compared to assembling stiffness matrices directly from vertex coordinates, our parametrization method is 2–3× faster.
- We propose a domain-based shape optimization algorithm that can perform local optimizations to reduce high stresses while preserving the model shape. Observing that the shapes of the mesh parts far away from high stresses are almost not changed during optimizations, our system only chooses the domains close to high stresses as optimizing variables. A constraint list is also dynamically maintained to reduce the cost of constraints enforcement. With the described optimization algorithm, our system can obtain the optimized results within 1–2 minutes.

We have tested our system with a variety of 3D models and the results show that our system provides users agile feedbacks. The accuracy of our system is also verified through two physical experiments on 3D print-out objects.

2. Related work

3D printing: Since 3D printing technology provides users the opportunities to interact with the designed 3D object in real world, it receives a significant amount of research interests in the computer graphics community. In recent years, many techniques are invented to facilitate the printing process, such as adding scaffoldings as support structures (Dumas et al., 2014), decomposing a printable model to separate parts (Hu et al., 2014; Luo et al., 2012), hollowing printable models (Lu et al., 2014) and using skin-frame structures as internal supports (Wang et al., 2013). Research efforts have also been devoted to design algorithms to let the printed 3D objects possess desirable physical properties, such as deformation (Bickel et al., 2010; Skouras et al., 2013), articulation (Bächer et al., 2012; Calì et al., 2012), mechanical motion (Bächer et al., 2014; Ceylan et al., 2013; Coros et al., 2013; Zhu et al., 2012) and appearance (Chen et al., 2013, 2014; Dong et al., 2010; Lan et al., 2013; Prévost et al., 2013). 3D printing technology has been adopted in many works as a convenient method of fabrication, for example, in face cloning (Bickel et al., 2012), self-supporting structures (Deuss et al., 2014) and appearance-mimicking surfaces (Schüller et al., 2014).

Our work is most related to the structural stability analysis of the 3D printable design. Stress relief operations, such as hollowing and thickening, are adopted in Stava et al. (2012) to improve the structural stability of the 3D objects, once areas with high stress are detected. A fast method to analyze worst load distribution that will cause high stress in printed object is developed in Zhou et al. (2013). These two algorithms can be viewed as a post-processing of the 3D design. In contrast, given the material properties which will be used in 3D printing, our goal is to develop an interactive structural analysis algorithm to let the user predict the stress distribution during designing. The structural analysis algorithm can also efficiently handle various external load constraints.

Fabrication-aware design: Fabrication-aware design focuses on developing geometric design algorithms that facilitate fabrication. For instance, in architecture geometry, Liu et al. (2006, 2011) developed algorithms to design planar quad mesh for freeform architectural surface to reduce the fabrication cost. To facilitate the 3D shape fabrication, algorithms have been designed to convert the 3D shape into planar slices (Hildebrand et al., 2012; McCrae et al., 2011). Given a 3D furniture model,

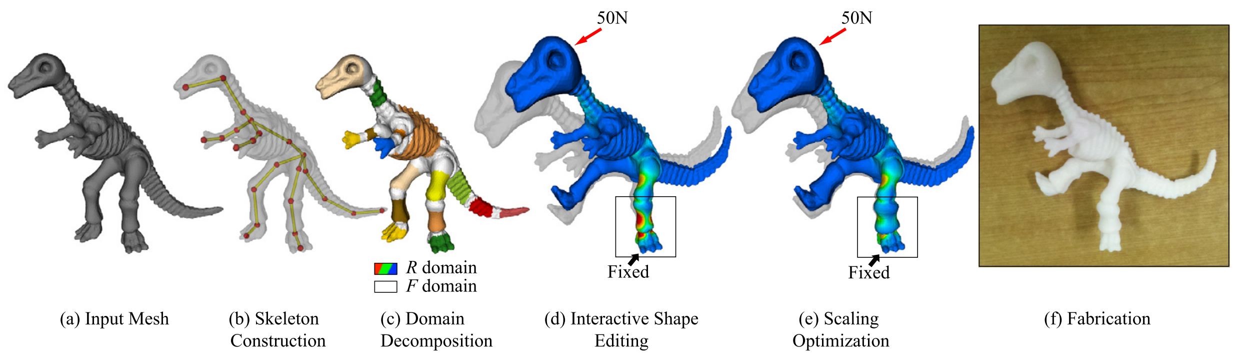

Fig. 1. The workflow of our system. (a) The input of our system is a tetrahedral mesh. (b) A skeleton is constructed from the input mesh. (c) The mesh is decomposed to domains according to the skeleton, and the decomposed domain is associated with the bones. (d) Users can edit the mesh and see the influence on stress distribution interactively. (e) Our system also can optimize the shape to reduce stresses. (f) After the stresses are reduced to a safe level, the edited mesh is used to print real objects.

Lau et al. (2011) proposed a method to decompose it into parts and connectors according to fabrication constraints. Recently, Koo et al. (2014) introduced a system for creating works-like prototypes of mechanical objects and Thomaszewski et al. (2014) proposed a system for designing linkage-based characters computationally. The technologies that perform physical simulations during shape design also fall into the area of concurrent engineering, where its philosophy is to take different targets of product design into account in parallel (Liu et al., 2010).

In order to allow users to predict the physical properties of the designed object, fast simulation techniques are integrated into design system so that the influences of changing geometric parameters on the physical properties can be efficiently computed. Its typical applications include plush toy design (Mori and Igarashi, 2007) and sensitivity analysis based cloth design (Umetani et al., 2011a). Sensitivity analysis technique is also adopted in furniture design to fast compute the structural stability (Umetani et al., 2012). Our work can be viewed as a fabrication-aware design algorithm, as we provide users interactive tools to edit the shape of the printable models, and integrate FEM simulation into our system to provide user structure analysis feedback.

Domain decomposition: Domain Decomposition (DD) is a numerical technique originally designed for solving partial differential equations (PDEs) (Toselli and Widlund, 2004) inspired by the idea of divide and conquer. Farhat et al. (2001, 1991) propose large-scale iterative DD solvers, which use Lagrange multipliers to enforce continuity constraints on the interface between domains and can be executed on multiple cores in parallel. In the computer graphics community, similar ideas have also been adopted in FEM based animation techniques. Nevertheless, in order to achieve a real-time simulation performance, simplifications have been applied, for example the assumption of interface rigidity (Barbič and Zhao, 2011), penalty forces based soft coupling (Kim and James, 2011), or with reduced boundary freedoms (Yang et al., 2013).

Instead of using DD as a computing technique, we use DD as a strategy to take advantage of locality. Our system adopts DD as a unified interface for shape editing, FEM system updating and shape optimization, while still solves the FEM system using direct factorization.

3. System overview

In this section, we briefly introduce each stage of our system. The pipeline of our system is illustrated in Fig. 1.

Skeleton construction: The input of our system is a tetrahedral mesh which consists of a set of vertices and a set of tetrahedrons. A set of bones $\mathcal{B} = \{b_1, b_2, \ldots, b_n\}$ is constructed from the mesh to form a skeleton which is used as the editing interface. An example of the constructed skeleton can be seen in Fig. 1(b).

Domain decomposition: According to the skeleton, the tetrahedral mesh is decomposed into a set of domains $\mathcal{D} = \{D_1, D_2, \ldots, D_n\}$. Each domain comprises its own sets of vertices and tetrahedrons. Each tetrahedron of the mesh belongs to one and only one domain, while the vertices on interfaces between domains are duplicated. Each domain has its own local coordinate system and its vertices are transformed into the local coordinate system.

Based on the associations between domains and bones, we classify the domains to two types: type $\mathcal{R}$ and type $\mathcal{F}$. A domain of type $\mathcal{R}$ is associated with only one bone. Its local coordinate system is built on the bone and its mesh is completely translated, rotated and scaled with the bone. In its local coordinate system, the vertex coordinates could be viewed as functions of the three scaling factors, thus the entries of the stiffness matrix assembled from the vertex coordinates are functions of the scaling factors too. We derive closed-form formulas to parameterize these entries, using the scaling factors as the parameters. A domain of the other type $\mathcal{F}$ is associated with multiple bones simultaneously. Its local coordinate system can be built on an arbitrary associated bone. Its vertex coordinates are transformed with each associated bone then blended. We do not parameterize the stiffness matrices of domains of type $\mathcal{F}$. An example of domain decomposition can be seen in Fig. 1(c). As can be seen from the example, most parts of the mesh are in domains of type $\mathcal{R}$.

Shape editing: After all domains associated to the bones, users can begin to manipulate the mesh by adjusting the skeleton. Our system integrates the Forward Kinematic technique (FK) and the Inverse Kinematic technique (IK) to let users translate and rotate the bones, consequently translate, rotate and blend the associated domains. Our system also let users scale the bones associated with domains of type $\mathcal{R}$, which consequently scale the associated type $\mathcal{R}$ domains and blend the adjacent type $\mathcal{F}$ domains. Please watch the accompanied video to see the effects of editing.

FEM simulation integration: When users are editing the mesh, our system runs FEM simulations in background to provide structure analysis feedback interactively. Same to Stava et al. (2012), our system solves the standard static equilibrium equation for the vertex displacements:

$$\mathbf{K}\vec{\mathbf{u}} = \vec{\mathbf{f}} \tag{1}$$

where $\mathbf{K}$ is the stiffness matrix assembled from the mesh using quadric tetrahedral elements, $\vec{\mathbf{f}}$ is the external loads, and the solution $\vec{\mathbf{u}}$ is the vertex displacements reacting to the external loads.

As soon as the mesh is edited by users, our system updates the stiffness matrix $\mathbf{K}$ to synchronize the FEM system with the edited mesh. Instead of re-assembling the whole $\mathbf{K}$, a domain-based local update scheme is used. Our system maintains local stiffness matrices for each domain, and $\mathbf{K}$ is assembled from these local stiffness matrices. When the mesh is edited, our system detects the affected domains and only re-assembles the local stiffness matrices of these domains. Then the entries in $\mathbf{K}$ contributed by these domains are updated with the re-assembled local stiffness matrices. The local stiffness matrices of domains of type $\mathcal{R}$ are parameterized and thus can be re-assembled very efficiently.

After $\mathbf{K}$ is updated and (1) is solved, our system calculates the Von-Mises stress of every vertex using (21) and visualizes the Von-Mises stress distribution on the surface of the mesh. The magnitudes of the Von-Mises stresses are represented by different colors, as in Fig. 1(d). From such feedback, users are aware of the influences of their edits on the stress distribution, thus can avoid editing operations causing high stresses in the model.

Our system is also able to provide error estimations of the computed Von-Mises stress values to users. We adopt the method in Zienkiewicz and Zhu (1987), where an improved stress distribution is first computed on vertices and then accuracy errors are calculated in percentage with the difference between the original and improved stress values.

Scaling optimization: In addition to providing feedback interactively during editing, our system can also perform a domain-based shape optimization to automatically reduce the maximal stress in the model. Users can set a Von-Mises stress threshold based on the material to be used in 3D printing. Our system detects a set of domains having Von-Mises stresses larger than the threshold and optimizes their scaling factors. The result is a mesh most close to the mesh before optimization and satisfies the stress constraints, as in Fig. 1(e).

Fabrication: After users finish their editing and ensure there are no structure stability problems may be caused from high stresses, the edited mesh can be used to print real objects. An example is shown in Fig. 1(f).

In the following sections, we describe each technique used in our system in detail. Section 4 introduces the methods for skeleton construction and domain decomposition. Section 5 derives the formulas for stiffness matrix parametrization. Section 6 describes the details of the FEM system updating. Section 7 discusses the scaling optimization. Section 8 describes the method we used for error estimation. Finally we demonstrate our system with several results in Section 9, conclude and discuss the limitations and possible future works in Section 10.

4. Pre-processing

In pre-processing, we construct a skeleton from the input mesh for the subsequent editing. The input mesh is decomposed to domains based on the skeleton structure, and the domains are associated with the bones.

We adopt the automatic rigging system, Pinocchio, in Baran and Popović (2007) for skeleton construction. The Pinocchio system pre-defines a set of skeletons for common animal and human models, and automatically embeds these skeletons into the input meshes. Our system allows users to edit the skeleton outputted from the Pinocchio system, like inserting new joints and adjusting joint positions. An example of the constructed skeleton can be seen in Fig. 1(b).

The Pinocchio system also computes the skinning weights for the surface vertices of the mesh, which can be used for domain decomposition. We extend the skinning weights of the surface vertices to the interior vertices by solving the Laplace equation:

$$\Delta\vec{\mathbf{w}} = \vec{\mathbf{0}} \tag{2}$$

where the components of $\vec{\mathbf{w}}$ are the weights of the internal vertices and the Laplace operator Δ is calculated using Mean Value Coordinates (Tao et al., 2005). The solved weights are consistent with the weights of the surface vertices and vary smoothly inside the tetrahedral mesh.

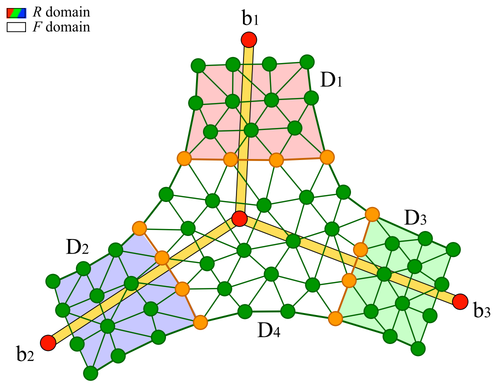

Fig. 2. Domain decomposition. D_1 (red), D_2 (blue) and D_3 (green) are type $\mathcal{R}$ domains and are associated with bones b_1, b_2 and b_3 respectively. D_4 (white) is a type $\mathcal{F}$ domain adjacent to D_1, D_2 and D_3, and is associated with b_1, b_2 and b_3 simultaneously. The orange points represent vertices on the interface between domains and are duplicated into the domains on both sides. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

For each bone b_i, we perform a region growing to detect tetrahedrons completely transformed with b_i. The region grows through face adjacency, which means a tetrahedron could be merged only if it shares a neighboring face with any already merged tetrahedron. We set a default weight threshold (0.95) such that a tetrahedron is merged only if the average weight of its four vertices is larger than the threshold. The tetrahedrons detected in this way form a domain of type $\mathcal{R}$ and is associated with b_i. We allow users to modify the domain by tuning the weight threshold and adding/removing individual tetrahedrons, provided the face adjacency property is preserved.

After all domains of type $\mathcal{R}$ are detected, the left tetrahedrons are grouped to domains of type $\mathcal{F}$, according to face adjacency. Such domains are simultaneously associated with multiple bones, an example could be seen in Fig. 2. The mesh of a type $\mathcal{F}$ domain is transformed with each associated bone then blended. To compute the blend weights for each type $\mathcal{F}$ domain, any mesh blend method can be used, providing the mesh continuities between domains of type $\mathcal{R}$ and domains of type $\mathcal{F}$ are enforced. For the sake of simplicity, we use linear blend and calculate the blend weights using the Laplace equation (2) with suitable boundary conditions. For example, as in Fig. 2, to calculate the blend weights of vertices of D_4 for b_1, we assign the weights on the interface between D_4 and D_1 all 1 and the weights on the interfaces between D_4 and D_2, D_4 and D_3 all 0, then solve the Laplace equation on all internal vertices of D_4. With the blend weights solved in this way, the continuities between domains of $\mathcal{R}$ and domains of $\mathcal{F}$ are enforced and the meshes in domains of $\mathcal{F}$ are smoothly blended.

5. Stiffness matrix parametrization

In this section, we show that each entry of the stiffness matrix of a quadratic tetrahedral mesh can be expressed as a polynomial. All these polynomials share a few same variables, which are arithmetic expressions formed by the scaling factors s_x, s_y and s_z along XYZ axes. The coefficients in these polynomials only depend on the vertex coordinates of the unedited mesh and the material parameters, thus can be pre-computed. When the mesh is scaled along XYZ axes, we can update the stiffness matrix by first computing these variables from s_x, s_y and s_z then re-evaluating the polynomial for each entry.

We use the quadratic tetrahedrons of our input mesh as the finite elements, and each element has 10 nodes which are the 10 vertices forming the tetrahedron, as shown in Fig. 3. Following the convention in FEM textbook (Zienkiewicz et al., 2008), the stiffness matrix $\mathbf{K}^e$ of an element e can be assembled as:

$$\mathbf{K}^e = \iiint \mathbf{B}^T \mathbf{D} \mathbf{B} \mathrm{d}x\mathrm{d}y\mathrm{d}z \tag{3}$$

$\mathbf{D}$ is the 6×6 stress–strain relation matrix formed from the material properties. In our application, we use isotropic material model. $\mathbf{B}$ is the 6×30 strain–displacement relation matrix:

$$\mathbf{B} = \left[\, \mathbf{B}_1 \middle| \ldots \middle| \mathbf{B}_{10} \,\right] \tag{4}$$

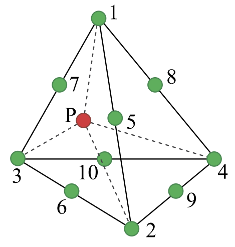

Fig. 3. Quadratic tetrahedral element.

where

$$\mathbf{B}_i = \begin{bmatrix} \frac{\partial N_i}{\partial x} & & \\ & \frac{\partial N_i}{\partial y} & \\ & & \frac{\partial N_i}{\partial z} \\ \frac{\partial N_i}{\partial y} & \frac{\partial N_i}{\partial x} & \\ & \frac{\partial N_i}{\partial z} & \frac{\partial N_i}{\partial y} \\ \frac{\partial N_i}{\partial z} & & \frac{\partial N_i}{\partial x} \end{bmatrix} \tag{5}$$

N_i denotes the shape function of node i.

The entries of the 3×3 sub-matrix $\mathbf{B}_i^T \mathbf{D} \mathbf{B}_j$ can be calculated with simple matrix multiplications. For example, the expression of the top-left entry is:

$$(\mathbf{B}_i^T \mathbf{D} \mathbf{B}_j)_{00} = \frac{E}{(1+v)(1-2v)}((1-v)\frac{\partial N_i}{\partial x}\frac{\partial N_j}{\partial x} + \frac{(1-2v)}{2}\frac{\partial N_i}{\partial y}\frac{\partial N_j}{\partial y} + \frac{(1-2v)}{2}\frac{\partial N_i}{\partial z}\frac{\partial N_j}{\partial z}) \tag{6}$$

where E and v are Young's modulus and Poisson's ratio of the material, both are constants.

Substitute (6) into (3); one can observe that the basic terms in the expressions of entries in $\mathbf{K}^e$ share the same form:

$$\mathbf{c}(i, j, \theta_1, \theta_2) = \iiint \frac{\partial N_i}{\partial \theta_1}\frac{\partial N_j}{\partial \theta_2} \mathrm{d}x\mathrm{d}y\mathrm{d}z, \quad \theta_1, \theta_2 = x, y, z \tag{7}$$

where θ_1 and θ_2 are dummy variables which can be x, y or z.

As showed in Zienkiewicz et al. (2008), within a quadratic tetrahedral element, the shape functions can be represented as:

$$\begin{aligned} &N_i = (2L_i - 1)L_i, \quad i = 1, 2, 3, 4 \\ &N_5 = 4L_1L_2, \quad etc. \end{aligned} \tag{8}$$

where L_i is the 3D barycentric coordinate with respect to the i-th node of the element. When $i > 4$, the node is on an edge and its shape function N_i is the product of the shape functions of the two end nodes of the edge.

The expression of $\mathbf{c}$ can be derived by substituting (8) into (7). For different combinations of the parameters, $\mathbf{c}$ has different forms, but the derivations are similar. Here we derive $\mathbf{c}(1, 1, x, x)$ as an example, the complete formulas can be found in Appendix A at the end of the paper. Suppose we are given an edited tetrahedron $t = \{\vec{\mathbf{v}}_i, \ i = 1 \ldots 10\}$, where $\vec{\mathbf{v}}_i = (x_i, y_i, z_i)^T$ is the position of the i-th node and is computed by:

$$x_i = s_x x_i', \ y_i = s_y y_i', \ z_i = s_z z_i' \tag{9}$$

where $(x_i', y_i', z_i')^T$ are the unedited position coordinates and are treated as constants. The barycentric coordinate of a point P (as in Fig. 3) with respect to the 1-th node is computed as:

$$L_1 = \frac{Vol\ P234}{Vol\ 1234} = \frac{a_1 + b_1 x + c_1 y + d_1 z}{6V} \tag{10}$$

Here Vol represents the volume computation for the tetrahedron formed by the following 4 points. V is the volume of the edited tetrahedron and can be computed by $V = s_x s_y s_z V'$, where V' is the volume of the unedited tetrahedron. $(x, y, z)^T$ are the coordinates of the point P. a_1 will not appear in the final expression, while b_1, c_1 and d_1 are calculated as (Zienkiewicz et al., 2008):

$$b_1 = -\begin{vmatrix} 1 & y_2 & z_2 \\ 1 & y_3 & z_3 \\ 1 & y_4 & z_4 \end{vmatrix} = -\begin{vmatrix} 1 & y_2' & z_2' \\ 1 & y_3' & z_3' \\ 1 & y_4' & z_4' \end{vmatrix} s_y s_z = b_1' s_y s_z,$$

$$c_1 = +\begin{vmatrix} 1 & x_2 & z_2 \\ 1 & x_3 & z_3 \\ 1 & x_4 & z_4 \end{vmatrix} = +\begin{vmatrix} 1 & x'_2 & z'_2 \\ 1 & x'_3 & z'_3 \\ 1 & x'_4 & z'_4 \end{vmatrix} s_x s_z = c'_1 s_x s_z,$$
$$d_1 = -\begin{vmatrix} 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \\ 1 & x_4 & y_4 \end{vmatrix} = -\begin{vmatrix} 1 & x'_2 & y'_2 \\ 1 & x'_3 & y'_3 \\ 1 & x'_4 & y'_4 \end{vmatrix} s_x s_y = d'_1 s_x s_y \tag{11}$$

Using (10) and (11), the partial derivative $\frac{\partial N_1}{\partial x}$ in the integrand of $c(1,1,x,x)$ is calculated as:

$$\frac{\partial N_1}{\partial x} = (4L_1 - 1)\frac{b'_1 s_y s_z}{6V' s_x s_y s_z} \tag{12}$$

Substitute (12) into (7), and use the integration equation for 3D barycentric coordinate functions:

$$\iiint L_1^p L_2^q L_3^l L_4^m \mathrm{d}x\mathrm{d}y\mathrm{d}z = \frac{p!q!l!m!}{(p+q+l+m+3)!}6V \tag{13}$$

we finally have:

$$\mathbf{c}(1,1,x,x) = \iiint \frac{\partial N_1}{\partial x}\frac{\partial N_1}{\partial x}\mathrm{d}x\mathrm{d}y\mathrm{d}z = \frac{1}{60}\frac{{b'_1}^2}{V'}\frac{s_y^2 s_z^2}{s_x s_y s_z} \tag{14}$$

Thus $\mathbf{c}(1,1,x,x)$ could be regarded as a one-term polynomial with coefficient α and variable $\mathbf{m}_{(x,x)}$:

$$\mathbf{c}(1,1,x,x) = \alpha\mathbf{m}_{(x,x)}, \ \alpha = \frac{1}{60}\frac{{b'_1}^2}{V'}, \ \mathbf{m}_{(x,x)} = \frac{s_y^2 s_z^2}{s_x s_y s_z} \tag{15}$$

where α only depends on the unedited mesh and can be pre-computed.

All types of $\mathbf{c}$ can be expressed as the product of a coefficient α and a variable $\mathbf{m}$. According to the different combinations of the dummy variable θ_1 and θ_2, there are totally 6 types of $\mathbf{m}$ and all of them are similar to $\mathbf{m}_{(x,x)}$, only formed by the three scaling factors. This indicates that all entries of $\mathbf{K}^e$ can be expressed as 1-order polynomials of 6 variables. Then if all the tetrahedrons in a domain are scaled with the same scaling factors, as in domains of type $\mathcal{R}$, the entries in the stiffness matrix of the whole domain are also 1-order polynomials of the 6 variables, and can be computed through polynomial summations.

Checking the expression of each entry of $\mathbf{B}_i^T\mathbf{D}\mathbf{B}_j$ (such as (6)), we can see that only three types of $\mathbf{m}$ are used for an entry on the diagonal of $\mathbf{B}_i^T\mathbf{D}\mathbf{B}_j$ while only two are used for an entry off the diagonal. And the pattern of variable types is same for all $\mathbf{B}_i^T\mathbf{D}\mathbf{B}_j$. Thus for every entry of the stiffness matrix, only two or three coefficients are needed to be stored. With these coefficients pre-computed from the unedited mesh, only 2–3 multiplications plus 1–2 additions are needed to evaluate an entry.

The same coefficients can also be used to evaluate the partial derivatives of each entry with respect to the scaling factors, as the formulas for the partial derivatives can be derived by simply differentiating each $\mathbf{m}$ with respect to s_x, s_y and s_z. Thus our parametrization also can be used to compute the partial derivatives of the stiffness matrix, which are required in the shape optimization algorithm described in Section 7.

For domains of type $\mathcal{F}$, the derivations in this section also provide a closed-form method to assemble the stiffness matrix directly from vertex coordinates, with no needs for numerical quadratures.

6. FEM system updating

As stated in Section 3, our system solves the standard equilibrium equation (1) for the vertex displacements. Whenever an editing operation is committed, our system needs to update the stiffness matrix $\mathbf{K}$ in (1) to synchronize with the edited mesh. In this section, we describe the local update scheme used in our system and the details of updating $\mathbf{K}$.

6.1. Local update scheme

We observe that most editing operations only affect a few parts of the mesh, thus we can update $\mathbf{K}$ by only re-evaluating a small fraction of entries in $\mathbf{K}$. We demonstrate this idea with a simple 1-D example.

Consider the simple 1-D structure formed by two springs in Fig. 4. Assume the stiffness matrices of the two elements e_1 and e_2 are:

$$\mathbf{K}^{e_1} = \begin{pmatrix} k_{11}^{e_1} & k_{12}^{e_1} \\ k_{21}^{e_1} & k_{22}^{e_1} \end{pmatrix}, \quad \mathbf{K}^{e_2} = \begin{pmatrix} k_{11}^{e_2} & k_{12}^{e_2} \\ k_{21}^{e_2} & k_{22}^{e_2} \end{pmatrix} \tag{16}$$

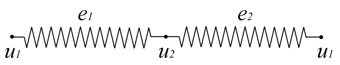

Fig. 4. A 1-D two elements example.

Then the global stiffness matrix $\mathbf{K}^g$ for the whole structure is assembled by adding the entries of $\mathbf{K}^{e_1}$ and $\mathbf{K}^{e_2}$ to the corresponding entries in $\mathbf{K}^g$:

$$\mathbf{K}^g = \begin{pmatrix} k_{11}^{e_1} & k_{12}^{e_1} & \\ k_{21}^{e_1} & k_{22}^{e_1} + k_{11}^{e_2} & k_{12}^{e_2} \\ & k_{21}^{e_2} & k_{22}^{e_2} \end{pmatrix} \tag{17}$$

When only the element e_2 is changed, $\mathbf{K}^{e_2}$ is changed while $\mathbf{K}^{e_1}$ remains the same. Consequently, $\mathbf{K}^g$ is changed to:

$$\hat{\mathbf{K}}^g = \begin{pmatrix} k_{11}^{e_1} & k_{12}^{e_1} & \\ k_{21}^{e_1} & k_{22}^{e_1} + \hat{k}_{11}^{e_2} & \hat{k}_{12}^{e_2} \\ & \hat{k}_{21}^{e_2} & \hat{k}_{22}^{e_2} \end{pmatrix} \tag{18}$$

where the hat on a symbol means its value is changed.

Compare $\mathbf{K}^g$ with $\hat{\mathbf{K}}^g$; we can see that only the four entries in the bottom-right corner which are contributed by e_2 are changed. Most of these changed entries can be updated by just re-assembling $\hat{\mathbf{K}}^{e_2}$. The only exception is the entry $k_{22}^{e_1} + \hat{k}_{11}^{e_2}$ that is related to the vertex on the interface between e_1 and e_2, which also needs the entry $k_{22}^{e_1}$ in $\mathbf{K}^{e_1}$. $\mathbf{K}^{e_1}$ is not changed so that $k_{22}^{e_1}$ can be re-used and only an extra addition is needed. This means when only a few elements of a structure are changed, the global stiffness matrix of the structure can be updated by only re-assembling the stiffness matrices of the modified elements plus performing extra additions for the entries related to the vertices on the interfaces between these elements.

The same idea can be extended to our domain decomposition case. The global stiffness matrix $\mathbf{K}$ can be updated locally in units of domains. We divide the entries in $\mathbf{K}$ to two types: entries contributed by only one domain and entries contributed by multiple domains simultaneously. When a domain is modified, the affected entries of the first type can be updated by just re-assembling the local stiffness matrix of the modified domain, while the affected entries of the second type require extra additions. The entries of the second type are only related to vertices on the interfaces between domains, thus only count a small fraction in the whole matrix.

We design special data structures to facilitate the local updating scheme. For each domain, we store the positions of the entries in $\mathbf{K}$ that are contributed by this domain. For each non-zero entry k in $\mathbf{K}$, we use a list $\mathcal{L}$ to record the domains contributing to it. Further, for each domain recorded in $\mathcal{L}$, the position of the entry in its local stiffness matrix $\mathbf{K}_i$ that is added to k is also stored. For most entries, $\mathcal{L}$ only contains one domain. When a set of domains are edited, our system first updates the $\mathbf{K}_i$ of each edited domain, then locates the affected entries in $\mathbf{K}$ and updates these entries by summing the contributions from the domains recorded in $\mathcal{L}$.

6.2. System updating

Instead of assembling $\mathbf{K}$ directly from the mesh in a straightforward manner, our system organizes the assembling of $\mathbf{K}$ in units of domains. Each domain maintains its local stiffness matrices and the global stiffness matrix is assembled and updated from these local stiffness matrices using the local update scheme described in Section 6.1.

For each domain D_i, our system stores three matrices: $\mathbf{K}'_i$, $\mathbf{R}_i$ and $\mathbf{K}_i$. $\mathbf{K}'_i$ is the local stiffness matrix and is assembled from the mesh decomposed into D_i in the local coordinate system. For all domains of type $\mathcal{R}$, $\mathbf{K}'_i$ is parameterized with the three scaling factors along the XYZ axes of the local coordinate system; for all domains of type $\mathcal{F}$, $\mathbf{K}'_i$ is directly assembled from the vertex coordinates. $\mathbf{R}_i$ is a rotation matrix transforming a 3-D vector from the local coordinate system of D_i to the global coordinate system. $\mathbf{K}_i$ is the local stiffness matrix of D_i in the global coordinate system, and is calculated by:

$$\mathbf{K}_i = \mathbf{R}_i \mathbf{K}'_i \mathbf{R}_i^T \tag{19}$$

$\mathbf{K}_i$ is the matrix used to update $\mathbf{K}$ in the local update scheme.

With the editing interface of our system, a domain can be modified in four manners: translation, rotation, scaling and blend. Only domains of type $\mathcal{R}$ will be scaled while only domains of type $\mathcal{F}$ will be blended. Translating a domain has no effects on its local stiffness matrices. Rotating a domain D_i changes $\mathbf{R}_i$. Scaling a domain of type $\mathcal{R}$ causes our system to re-assemble $\mathbf{K}'_i$ using parametrization. Blending a domain of type $\mathcal{F}$ causes our system to re-assemble $\mathbf{K}'_i$ directly from the new-blended vertex coordinates. If either $\mathbf{R}_i$ or $\mathbf{K}'_i$ is changed, $\mathbf{K}_i$ needs to be re-calculated.

Before users begin editing, our system performs a pre-computation to initialize the simulation system. For each domain D_i, our system allocates its $\mathbf{K}'_i$, $\mathbf{R}_i$ and $\mathbf{K}_i$. For each domain of type $\mathcal{R}$, our system pre-computes the coefficients used in the parametrization of $\mathbf{K}'_i$. And the global stiffness matrix $\mathbf{K}$ is allocated and symbolically factorized. The data structure

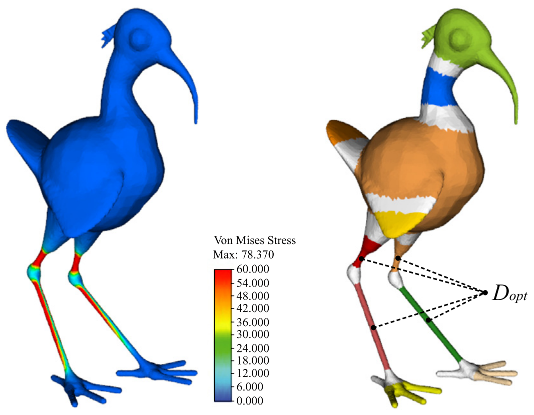

Fig. 5. Example of $\mathcal{D}_{opt}$. Only the four domains having Von-Mises stresses above the threshold (60 MPa) are added into $\mathcal{D}_{opt}$.

used for local update scheme is also built. As the editing operations in our system don't change the topology of the mesh, the sparse matrix structures of the stiffness matrices are re-used in subsequent updates.

When users make an edit, our system first determines the affected domains. Then for each affected domain, our system maintains its local stiffness matrices in the manner described above. Finally **K** is updated using the local update scheme.

7. Scaling optimization

In addition to providing the editing interface, our system can also optimize the scaling factors of domains of type $\mathcal{R}$ to reduce the Von-Mises (VM) stresses in the model. The objective function of the optimization is formulated as the difference between the vertex coordinates before optimization and the vertex coordinates after optimization, and is minimized to preserve the shape of the mesh. Inequalities constraints are used to enforce that the VM stresses of the vertices are under a user specific threshold after optimization.

High stresses usually occur at the thin parts of the mesh, and can be reduced by adjusting the thicknesses of the mesh parts near the high stresses. With our system, thickening and thinning parts of the mesh can be easily achieved by adjusting the scaling factors of the type $\mathcal{R}$ domains, thus their scaling factors become a nature choice for optimization variables. We also observe that adjusting the thickness of a part of the mesh far away from the high stresses has very little effects on reducing the high stresses, so we only choose the domains near the high stresses. We denote the set of domains used in the optimization by $\mathcal{D}_{opt}$, a domain D_i of type $\mathcal{R}$ is added into $\mathcal{D}_{opt}$ if: (1) D_i itself has stresses larger than the user specific threshold; or (2) D_i is adjacent to another type $\mathcal{F}$ domain D_j while D_j has stresses larger than the threshold. Fig. 5 shows an example of $\mathcal{D}_{opt}$. Further, for each domain in $\mathcal{D}_{opt}$, we only use the two scaling factors along the two axes perpendicular to the associated bone.

We also found that directly constraining the VM stress on every vertex is inefficient, even slows down the performance of the system to an unacceptable degree. Thus instead of constraining every vertices, we only monitor the vertices having the maximal VM stresses. Our system dynamically maintains a set of constrained vertices $\mathcal{V}_{constrain}$ during the optimization and only enforces constraints on the vertices in $\mathcal{V}_{constrain}$. At the beginning of the optimization, $\mathcal{V}_{constrain}$ only contains one vertex that has the maximal VM stress in the whole mesh. In each iteration, whenever a vertex not in $\mathcal{V}_{constrain}$ becomes the vertex having the maximal VM stress, it is added into $\mathcal{V}_{constrain}$. In the worst case, all vertices of the mesh could be added into $\mathcal{V}_{constrain}$, which is equivalent to constraining all vertices. From our test results, there are always only several vertices in $\mathcal{V}_{constrain}$ at the end of the optimization.

With the set of chosen domains $\mathcal{D}_{opt}$ and the set of constrained vertices $\mathcal{V}_{constrain}$, we formulate the optimization as:

$$\begin{aligned} &min\ \|\vec{\mathbf{v}}(\vec{\mathbf{s}}) - \vec{\mathbf{v}}(\vec{\mathbf{s}}_0)\|^2 \\ &\quad \text{s.t.} \quad \forall v \in \mathcal{V}_{constrain},\ \sigma_{vm}^{v} < \sigma_{vm}^{threshold} \end{aligned} \tag{20}$$

where $\vec{\mathbf{s}}$ is a vector containing the scaling factors of the domains in $\mathcal{D}_{opt}$, $\vec{\mathbf{s}}_0$ is a vector containing the same scaling factors before the optimization, $\vec{\mathbf{v}}(\vec{\mathbf{s}})$ and $\vec{\mathbf{v}}(\vec{\mathbf{s}}_0)$ are the vector coordinates of the mesh scaled with $\vec{\mathbf{s}}$ and $\vec{\mathbf{s}}_0$. For a vertex v in $\mathcal{V}_{constrain}$, σ_{vm}^{v} is its VM stress. $\sigma_{vm}^{threshold}$ is the user specific VM stress threshold.

For a vertex v, σ_{vm}^{v} is calculated as (Logan, 2012):

$$\sigma_{vm}^{v} = \frac{1}{\sqrt{2}}((\sigma_x - \sigma_y)^2 + (\sigma_y - \sigma_z)^2 + (\sigma_z - \sigma_x)^2 + 6(\tau_{xy}^2 + \tau_{yz}^2 + \tau_{zx}^2))^{\frac{1}{2}} \tag{21}$$

The terms σ and τ are the components of vertex stress $\vec{\boldsymbol{\sigma}}^{v}$. Within an element e, $\vec{\boldsymbol{\sigma}}^{v}$ is calculated as:

$$\vec{\boldsymbol{\sigma}}^{v} = (\sigma_x \quad \sigma_y \quad \sigma_z \quad \tau_{xy} \quad \tau_{yz} \quad \tau_{zx})^T = \mathbf{DB}\vec{\mathbf{u}}^{e} \tag{22}$$

where **D** and **B** are the stress–strain relation matrix and the strain–displacement relation matrix same as in (3). $\vec{\mathbf{u}}^e$ is the vector containing the displacements of the vertices of e. Using quadratic tetrahedral elements, $\vec{\boldsymbol{\sigma}}^v$ is discontinuous on the interface between two elements, thus for every vertex we calculate the average value of its stresses within its incident elements.

As the objective function is quadratic and the constraints are non-linear, we adopt the Sequential Quadratic Programming (SQP) method. We linearize σ^v_{vm} as:

$$\sigma^v_{vm}(\vec{\mathbf{s}}+\Delta\vec{\mathbf{s}}) \approx \sigma^v_{vm}(\vec{\mathbf{s}}) + \frac{\partial \sigma^v_{vm}}{\partial \vec{\mathbf{s}}}(\vec{\mathbf{s}})\Delta\vec{\mathbf{s}} \tag{23}$$

Then during the k-iteration, we solve the Quadratic Programming (QP) problem:

$$\begin{aligned} &min\ \|\vec{\mathbf{v}}(\vec{\mathbf{s}}_k+\Delta\vec{\mathbf{s}}) - \vec{\mathbf{v}}(\vec{\mathbf{s}}_0)\|^2 \\ &\quad \text{s.t.}\quad \forall v \in \mathcal{V}_{constrain}, \sigma^v_{vm}(\vec{\mathbf{s}}_k) + \frac{\partial \sigma^v_{vm}}{\partial \vec{\mathbf{s}}}(\vec{\mathbf{s}}_k)\Delta\vec{\mathbf{s}} < \sigma^{threshold}_{vm} \end{aligned} \tag{24}$$

for $\Delta\vec{\mathbf{s}}$ and update the scaling factors as:

$$\vec{\mathbf{s}}_{k+1} = \vec{\mathbf{s}}_k + \Delta\vec{\mathbf{s}} \tag{25}$$

The partial derivative in (23) can be calculated by using the chain rule:

$$\frac{\partial \sigma^v_{vm}}{\partial \vec{\mathbf{s}}}(\vec{\mathbf{s}}_k) = \frac{\partial \sigma^v_{vm}}{\partial \vec{\boldsymbol{\sigma}}^v}(\vec{\mathbf{s}}_k)\frac{\partial \vec{\boldsymbol{\sigma}}^v}{\partial \vec{\mathbf{s}}}(\vec{\mathbf{s}}_k) \tag{26}$$

The first term in (26) is straightforward and we describe how to compute the second term. From (22), the partial derivative of $\vec{\boldsymbol{\sigma}}^v$ with respect to a component s of $\vec{\mathbf{s}}$ is computed as:

$$\frac{\partial \vec{\boldsymbol{\sigma}}^v}{\partial s}(\vec{\mathbf{s}}_k) = \mathbf{D}\frac{\partial \mathbf{B}}{\partial s}(\vec{\mathbf{s}}_k)\vec{\mathbf{u}}^e(\vec{\mathbf{s}}_k) + \mathbf{DB}(\vec{\mathbf{s}}_k)\frac{\partial \vec{\mathbf{u}}^e}{\partial s}(\vec{\mathbf{s}}_k) \tag{27}$$

D is constant. **B** changes with the vertex coordinates of the element e, thus is only related to these s affecting e. Its partial derivative is 0 for other s and we compute the non-zero ones by finite differences.

$\vec{\mathbf{u}}^e$ is computed as a part of $\vec{\mathbf{u}}$ which contains the displacements of the whole mesh. Similarly $\frac{\partial \vec{\mathbf{u}}^e}{\partial s}$ is computed as a part of $\frac{\partial \vec{\mathbf{u}}}{\partial s}$. $\vec{\mathbf{u}}$ is obtained by solving (1), and we adopt the sensitivity analysis technique (Arnout et al., 2012) to calculate its partial derivatives: differentiate both sides of the equilibrium equation (1) with respect to s, then after some simple re-arrangements we get:

$$\frac{\partial \vec{\mathbf{u}}}{\partial s}(\vec{\mathbf{s}}_k) = \mathbf{K}(\vec{\mathbf{s}}_k)^{-1}\left(\frac{\partial \vec{\mathbf{f}}}{\partial s}(\vec{\mathbf{s}}_k) - \frac{\partial \mathbf{K}}{\partial s}(\vec{\mathbf{s}}_k)\vec{\mathbf{u}}(\vec{\mathbf{s}}_k)\right) \tag{28}$$

Different entries in $\vec{\mathbf{f}}$ and **K** are contributed by different domains, thus for each s only the entries contributed by the domains changing with s have non-zero partial derivatives with respect to s. Such entries in **K** can be located using the data structure for the local update scheme, as described in Section 6.1. For entries contributed by domains of type $\mathcal{R}$, their partial derivatives can be computed using the parametrization, as described in Section 5. For entries contributed by domains of type $\mathcal{F}$, we compute their partial derivatives by finite differences. $\frac{\partial \vec{\mathbf{f}}}{\partial s}$ is also computed by finite differences.

The inverse matrix $\mathbf{K}(\vec{\mathbf{s}}_k)^{-1}$ need not to be computed explicitly, as computing $\mathbf{A}^{-1}\vec{\mathbf{y}}$ equals solving $\mathbf{A}\vec{\mathbf{x}} = \vec{\mathbf{y}}$ for $\vec{\mathbf{x}}$. $\mathbf{K}(\vec{\mathbf{s}}_k)$ could be updated using the local update scheme, and only needs to be factorized once for each iteration.

The iterating of the optimization is determined as converged when two conditions are satisfied simultaneously: (1) no more new vertex is added into $\mathcal{V}_{constrain}$, and (2) the relative change of the objective function between two successive iterations is below a threshold ϵ. In our tests, we set ϵ as 1e–4. The whole algorithm is described in Algorithm 1.

8. Error estimation

For the purpose of interactive performance and the simple formulations of the derivative of stress to editing parameters in optimization, our system computes the stress on each vertex by averaging operation. To let users predict the accuracy of the analysis, our system is also able to estimate the error using the method in Zienkiewicz and Zhu (1987). First, an improved vertex stresses $\vec{\boldsymbol{\sigma}}^\star$ are computed by solving the equation:

$$\int_\Omega \mathbf{N}^T\mathbf{N}d\Omega\,\vec{\boldsymbol{\sigma}}^\star = \int_\Omega \mathbf{N}^T\mathbf{DB}d\Omega\vec{\mathbf{u}} \tag{29}$$

```
   Data: vertices set $\mathcal{V}$, tetrahedrons set $\mathcal{T}$, scaling factors $\vec{\mathbf{s}}_0$
   Result: optimized $\vec{\mathbf{s}}$
 1 initialize $\mathcal{D}_{opt}$, $\mathcal{V}_{constrain}$;
 2 $\vec{\mathbf{s}} = \vec{\mathbf{s}}_0$;
 3 while not converge do
 4     calculate $\frac{\partial \sigma_{vm}^{v}}{\partial \vec{\mathbf{s}}}(\vec{\mathbf{s}}_k)$ for every $v$ in $\mathcal{V}_{constrain}$ using (23);
 5     solve the QP problem (24) for $\Delta\vec{\mathbf{s}}$;
 6     $\vec{\mathbf{s}} = \vec{\mathbf{s}} + \Delta\vec{\mathbf{s}}$;
 7     update vertex coordinates $\mathcal{V}$ with $\vec{\mathbf{s}}$;
 8     calculate Von-Mises stresses of all vertices;
 9     update $\mathcal{V}_{constrain}$;
10     update the objective function (20);
11 end
```

Algorithm 1: The SQP algorithm.

where Ω represents the elements of the mesh, $\mathbf{N}$ is the matrix of shape functions, $\mathbf{D}$ and $\mathbf{B}$ are the stress–strain relation matrix and strain–displacement relation matrix, and $\vec{\mathbf{u}}$ is the vertex displacements solved from (1). The percentage error η in energy norm is then computed as:

$$\eta = \frac{\|\vec{\mathbf{e}}\|}{\|\vec{\mathbf{u}}\|} \tag{30}$$

where

$$\|\vec{\mathbf{e}}\| = \left[\int\limits_{\Omega} \vec{\mathbf{e}}_{\sigma}^{T}\mathbf{D}^{-1}\vec{\mathbf{e}}_{\sigma}\,d\Omega\right]^{\frac{1}{2}}, \quad \vec{\mathbf{e}}_{\sigma} = \mathbf{N}\vec{\sigma}^{\star} - \mathbf{DB}\vec{\mathbf{u}} \tag{31}$$

and

$$\|\vec{\mathbf{u}}\| = \left[\int\limits_{\Omega} \vec{\mathbf{u}}^{T}\mathbf{B}^{T}\mathbf{DB}\vec{\mathbf{u}}\,d\Omega\right]^{\frac{1}{2}} \tag{32}$$

Finally η is multiplied by an empirical correction multiplying factor. Consulting the factors used in Zienkiewicz and Zhu (1987), we use 1.5 for our quadratic tetrahedral element.

9. Results

We implement our system on a desktop PC equipping an Intel i7-4770 3.4 GHz CPU and 16 GB RAM. The standard equilibrium equation (1) is solved with the Direct Sparse Solver (DSS) module of the MKL library, through matrix factorization and back-substitution. The Quadratic Programming problem (24) is solved using the ALGLIB library. Except the parallelism in MKL, all computations are executed in a single thread.

In all tests we use the typical parameters of Polylactic Acid (PLA) material, which is widely used in 3D printing. Young's modulus is set as 2.3e9 N/m^2, Poisson's ratio is set as 0.35, and density is set as 1300 kg/m^3. In addition to the user-specified forces, the gravity is also taken into account and the gravity acceleration is set as 9.8 m/s^2. The yielding strength of PLA is about 60 MPa and we use 55 MPa as the Von-Mises stress threshold in optimization for safety. The real objects are printed by an FDM 3D printer using the PLA material.

Fig. 6 shows our experiment on an opener model. We measured the loads that an opener undertakes when opening a beer bottle, and simulated the same loads in our system (a, b). From the analysis results provided by our system, we were aware that the printed object might break when opening a bottle, because the high stresses occur at these red regions exceeded the yielding strength of the material (d). With the editing interface of our system, we thickened parts of the mesh, balancing the structure stabilities against appearances. After editing, our system predicted the stresses had been reduced to a safe level (f). We printed two real objects from the unedited mesh and the edited mesh respectively, and tested them by opening real bottles. The opener printed form the unedited mesh broke in the test, and the position of the crack matches the high-stress region in the analysis result (c). The opener printed from the edited mesh survived the test (e).

Fig. 7 shows another experiment on a bird model. The leg parts of the model are thin and are easy to break. From the analysis result provided by our system, we were aware that very high stresses would occur if a 5 N force were exerted on a leg of the model. This time we let our system optimize the mesh to enable the printed model to undertake the force. The experiments on the printed objects match the prediction of our system, as the object printed from the unedited mesh broke during the experiment, while the object printed from the edited mesh survived the experiment.

The dinosaur model in Fig. 1 is another example. We adjusted its pose using the IK tool, and scaled its head. From the structure analysis feedback provided by our system, we knew high stresses might occur on the left leg of the model (d).

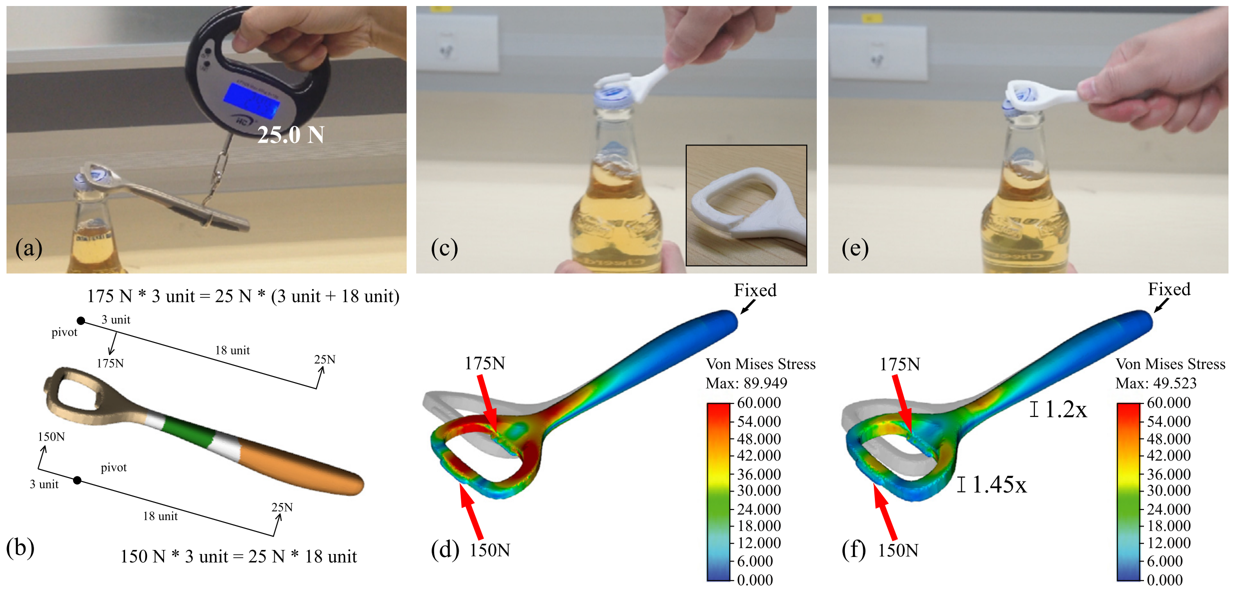

Fig. 6. Opener experiment. (a, b) We measured the loads with an electric spring scale. The magnitude of the forces exerted on different parts of the opener are computed by torque equilibrium. (c, d) Before editing, our system predicted that the maximal Von-Mises stress in the model exceeded the yielding strength of the material, and the printed object broke in the experiment. (e, f) After editing, our system predicted that the high stresses had been reduced, and the printed object survived the experiment. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.)

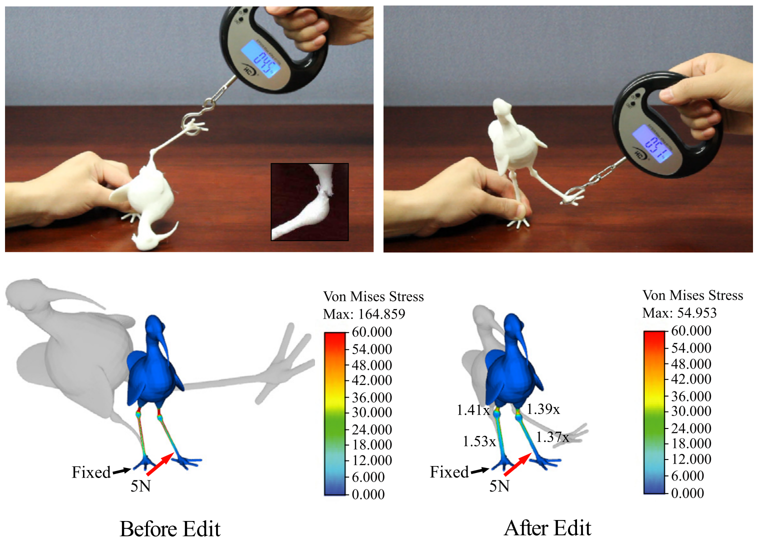

Fig. 7. Bird experiment. Left: the object printed from the unedited model broke when a 5 N force was exerted on the leg, as predicted by our system. Right: after optimization, the printed object survived the physical experiment. The larger ones of the two optimized scaling factors of each leg domain are marked on the figure.

Then we let our system optimize the mesh and the left leg be thickened to reduce the stresses. For aesthetic reason, we manually scale the right leg to match the left leg. The edited mesh is used to print a real object as in (f). Other two similar examples are showed in Fig. 8.

Table 1 lists the size of each test 3D model. Table 2 reports the performance of our system's underlying solver, which is updated with the method described in Section 6.2, taking advantages of stiffness matrix parametrization (Section 5) and local update scheme (Section 6.1). In addition to model size, the performance also depends on the number of tetrahedrons and vertices decomposed into domains of type $\mathcal{R}$. From the table we can see that assembling the stiffness matrix using parametrization is usually 2–3 times faster than assembling the matrix directly from vertex coordinates using the closed-form formulas, and local updating can be 6–20 times faster. For all test models, our system can provide the structure analysis feedback within few seconds after users make an edit.

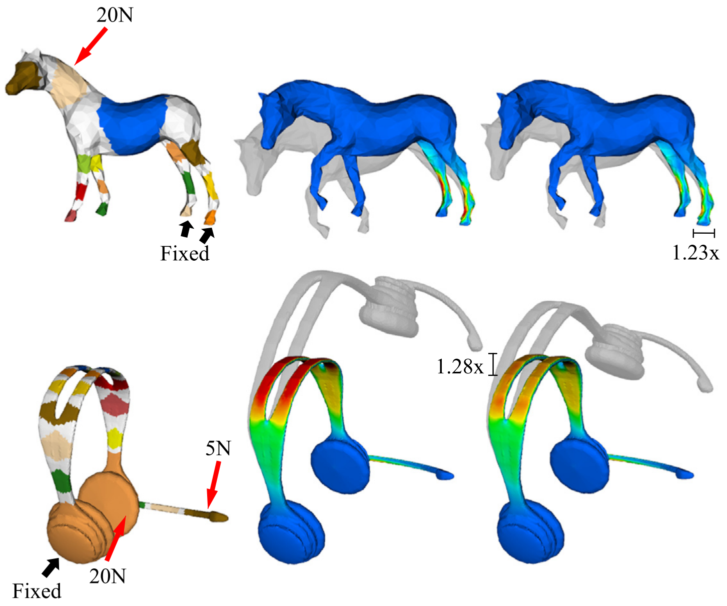

Fig. 8. More scenes. Left: domain decompositions and load configurations. Middle: shape editing results, high Von-Mises stresses are presented by red color. Right: meshes after optimization, the maximal scaling are marked on the figure. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 1
Scene sizes. Total, $\mathcal{R}$ and $\mathcal{F}$ means number in the whole mesh, number in all domains of $\mathcal{R}$ and number in all domains of type $\mathcal{F}$ respectively.

Scene	Domain #			Tetrahedron #			Vertex #		
	Total	$\mathcal{R}$	$\mathcal{F}$	Total	$\mathcal{R}$	$\mathcal{F}$	Total	$\mathcal{R}$	$\mathcal{F}$
Opener	5	3	2	38,729	33,293	5436	73,640	64,092	10,652
Bird	23	12	11	58,911	46,996	11,915	115,232	94,019	25,919
Horse	26	15	11	15,869	7813	8056	29,522	15,840	15,802
Dinosaur	27	16	11	31,409	21,158	10,251	61,407	43,163	21,549
Earphone	34	15	19	42,944	37,180	5764	81,977	71,811	13,248

Table 2
The performance of our system's underlying solver. From left to right, Sparse structure: time of allocating the local stiffness matrices of every domain and the global stiffness matrix, building the data structure for local update scheme, and symbolically factorizing the global stiffness matrix. Calc Poly Coef: time of pre-computing the polynomial coefficients of the stiffness matrix parametrization. Direct assemble: time of evaluating all entries in the global stiffness matrix directly from vertex coordinates. DD assemble: the first number is the time of evaluating all entries in the global stiffness matrix, using parametrization for domains of type $\mathcal{R}$; the second number is the time of updating the global stiffness matrix with the local update scheme, using parametrization for domains of type $\mathcal{R}$. As system updating varies according to the editing operations, the second number is an average value of a sequence of edits similar to those editing operations recorded in the video. Solve: time of solving the system, including numerical factorization and back substitution.

Scene	Sparse structure (s)	Calc Poly Coef (s)	Direct assemble (ms)	DD assemble (ms)	Solve (s)
Opener	3.48	0.35	387	103/50	1.11
Bird	5.46	0.49	634	199/23	2.93
Horse	1.47	0.08	170	121/28.6	0.41
Dinosaur	3.15	0.23	338	155/31.8	0.91
Earphone	3.82	0.41	458	119/20.9	1.38

Table 3 reports the performance of our scaling optimization algorithm described in Section 7. Our optimization algorithm can robustly handle a large range of Von-Mises stresses, as in the bird model the maximal Von-Mises stress is dropped from 164.86 MPa to 55 MPa, reduced by 66.6%. For a model of tens of thousands of vertices, our optimizer is able to converge in tens of seconds.

Fig. 9 shows the distributions of the per-element errors of two test models, which are estimated using the method described in Section 8. The bird model is a very fine model. The force configuration and the analysis result is same as Fig. 7 (left), and the error of the whole mesh is 6.24% while the maximal element error is 127.3%. The horse model is relatively coarse. The force configuration and the analysis result is same as Fig. 8 (middle), and the error of the whole mesh is 14.6% while the maximal element error is 338.5%. As most of our test models are relatively fine and quadratic elements are used, the estimated errors of our models are usually 5%–15%.

Table 3
The performance of scaling optimization. Max VM: the maximal Von-Mises stress of the model before optimization. $\mathcal{D}_{opt}$: the number of the domains in $\mathcal{D}_{opt}$. Cstr: the number of the vertices in $\mathcal{V}_{constrain}$ when the optimization completes. For all these 3D models, we set the Von-Mises stress threshold as 55 MPa.

Scene	Max VM (MPa)	$\mathcal{D}_{opt}$	Cstr	Iter	Time (s)
Opener	89.95	2	3	6	20.78
Bird	164.86	4	4	10	77.89
Horse	91.40	4	3	6	9.55
Dinosaur	110.62	3	2	6	18.89
Earphone	87.83	8	7	9	52.90

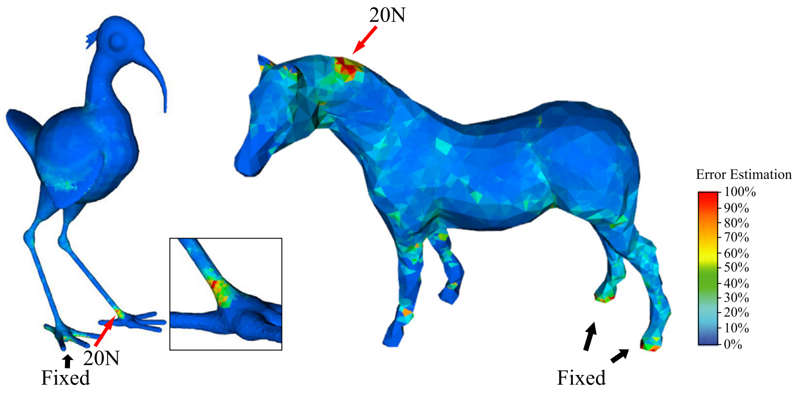

Fig. 9. The distribution of the percentage error in energy norm on elements.

10. Conclusion

In this paper, we described a shape editing system for models used in 3D printing. Our system integrates FEM simulation to provide stress distribution feedback during mesh editing. Domain decomposition is adopted as a unified interface for shape editing, FEM system updating and shape optimization. Parametrization is used to efficiently synchronize stiffness matrices with the edited mesh, and the FEM system is updated with a local update scheme to avoid repetitive computations. A domain-based scaling optimization algorithm is also devised to automatically reduce high stresses while preserving mesh shape. We tested our system with a variety of 3D models and verified its accuracy with two physical experiments.

The major limitation of our system is that it only supports the editing of skeleton-based models, and the editing interface is confined to domain-level translation, rotation and scaling. We are considering supporting more editing operations and parameterizing the stiffness matrix with more geometry parameters. In addition, our stiffness matrix parametrization only handles isotropic material. Therefore, integrating anisotropic material properties can be an interesting future work. In some large scale editing cases, we observe that large deformation may degenerate or even invert some tetrahedrons, which would downgrade the accuracy of the analysis or cause solver failure. Another possible future work can be to preserve qualities of the tetrahedral mesh or even apply re-meshing during mesh editing.

Acknowledgements

We would like to thank the anonymous reviewers for their constructive comments; Changke Zhang for improving the mesh quality. This work is partially supported by NSFC (No. 61272392, No. 61272305, No. 61322204). Yin Yang is partially supported by National Science Foundation CRII 1464306 and open project of State Key Lab of China of CAD&CG (A1403), and Xiaohu Guo is partially supported by Cancer Prevention & Research Institute of Texas (CPRIT) under Grant No. RP110329, and National Science Foundation (NSF) under Grant Nos. IIS-1149737 and CNS-1012975.

Appendix A. Complete formulas in stiffness matrix parametrization

In this appendix we list the complete formulas derived in Section 5.
All the 9 elements of the sub-matrix $\mathbf{B_i}^T\mathbf{DB_j}$ are:

$$(\mathbf{B}_i^T\mathbf{DB}_j)_{00} = \frac{E}{(1+v)(1-2v)}((1-v)\frac{\partial N_i}{\partial x}\frac{\partial N_j}{\partial x} + \frac{(1-2v)}{2}\frac{\partial N_i}{\partial y}\frac{\partial N_j}{\partial y} + \frac{(1-2v)}{2}\frac{\partial N_i}{\partial z}\frac{\partial N_j}{\partial z})$$

$$(\mathbf{B}_i^T\mathbf{DB}_j)_{01} = \frac{E}{(1+v)(1-2v)}(v\frac{\partial N_i}{\partial x}\frac{\partial N_j}{\partial y} + \frac{1-2v}{2}\frac{\partial N_i}{\partial y}\frac{\partial N_j}{\partial x})$$

$$
\begin{aligned}
(\mathbf{B}_i^T\mathbf{D}\mathbf{B}_j)_{02} &= \frac{E}{(1+v)(1-2v)}(v\frac{\partial N_i}{\partial x}\frac{\partial N_j}{\partial z}+\frac{1-2v}{2}\frac{\partial N_i}{\partial z}\frac{\partial N_j}{\partial x})\\
(\mathbf{B}_i^T\mathbf{D}\mathbf{B}_j)_{10} &= \frac{E}{(1+v)(1-2v)}(v\frac{\partial N_i}{\partial y}\frac{\partial N_j}{\partial x}+\frac{1-2v}{2}\frac{\partial N_i}{\partial x}\frac{\partial N_j}{\partial y})\\
(\mathbf{B}_i^T\mathbf{D}\mathbf{B}_j)_{11} &= \frac{E}{(1+v)(1-2v)}(\frac{1-2v}{2}\frac{\partial N_i}{\partial x}\frac{\partial N_j}{\partial x}+(1-v)\frac{\partial N_i}{\partial y}\frac{\partial N_j}{\partial y}+\frac{1-2v}{2}\frac{\partial N_i}{\partial z}\frac{\partial N_j}{\partial z})\\
(\mathbf{B}_i^T\mathbf{D}\mathbf{B}_j)_{12} &= \frac{E}{(1+v)(1-2v)}(v\frac{\partial N_i}{\partial y}\frac{\partial N_j}{\partial z}+\frac{1-2v}{2}\frac{\partial N_i}{\partial z}\frac{\partial N_j}{\partial y})\\
(\mathbf{B}_i^T\mathbf{D}\mathbf{B}_j)_{20} &= \frac{E}{(1+v)(1-2v)}(v\frac{\partial N_i}{\partial z}\frac{\partial N_j}{\partial x}+\frac{1-2v}{2}\frac{\partial N_i}{\partial x}\frac{\partial N_j}{\partial z})\\
(\mathbf{B}_i^T\mathbf{D}\mathbf{B}_j)_{21} &= \frac{E}{(1+v)(1-2v)}(v\frac{\partial N_i}{\partial z}\frac{\partial N_j}{\partial y}+\frac{1-2v}{2}\frac{\partial N_i}{\partial y}\frac{\partial N_j}{\partial z})\\
(\mathbf{B}_i^T\mathbf{D}\mathbf{B}_j)_{22} &= \frac{E}{(1+v)(1-2v)}(\frac{1-2v}{2}\frac{\partial N_i}{\partial x}\frac{\partial N_j}{\partial x}+\frac{1-2v}{2}\frac{\partial N_i}{\partial y}\frac{\partial N_j}{\partial y}+(1-v)\frac{\partial N_i}{\partial z}\frac{\partial N_j}{\partial z})
\end{aligned}
\tag{A.1}
$$

So we can see the basic term in $\iiint \mathbf{B_i}^T\mathbf{DB_j}\mathrm{d}x\mathrm{d}y\mathrm{d}z$ is

$$\mathbf{c}(i,j,\theta_1,\theta_2)=\iiint \frac{\partial N_i}{\partial \theta_1}\frac{\partial N_j}{\partial \theta_2}\mathrm{d}x\mathrm{d}y\mathrm{d}z, \quad \theta_1,\theta_2=x,y,z \tag{A.2}$$

As derived in Section 5, after handling the integration using (13) $\mathbf{c}(i,j,\theta_1,\theta_2)$ can be expressed as

$$\mathbf{c}(i,j,\theta_1,\theta_2)=\alpha_{(i,j,\theta_1,\theta_2)}\mathbf{m}_{(\theta_1,\theta_2)} \tag{A.3}$$

where $\alpha_{(i,j,\theta_1,\theta_2)}$ is a constant coefficient computed from the rest pose mesh, and $\mathbf{m}_{(\theta_1,\theta_2)}$ is the term formed from scaling factor s_x, s_y and s_z. To express $\alpha_{(i,j,\theta_1,\theta_2)}$, we first define some helper functions. Let

$$i=\mathrm{mid}(m,n) \tag{A.4}$$

express that the i-th node is the middle node between the m-th node and the n-th node (see Fig. 3), and

$$s_{(i,j)}=\begin{cases}\frac{1}{10} & \text{if } i=j\\ \frac{1}{20} & \text{if } i\neq j\end{cases} \tag{A.5}$$

and

$$l_{(k,\theta)}=\begin{cases} b_k & \text{if } \theta=x\\ c_k & \text{if } \theta=y\\ d_k & \text{if } \theta=z\end{cases} \tag{A.6}$$

where b_k, c_k and d_k are computed as in (11).

Then

$$\alpha_{(i,j,\theta_1,\theta_2)}=\begin{cases}
\frac{4}{9}s_{(i,j)}l_{(i,\theta_1)}l_{(j,\theta_2)}-\frac{1}{36}l_{(i,\theta_1)}l_{(j,\theta_2)} & \text{if } i,j\in[1,4]\\
\frac{4}{9}s_{(n,q)}l_{(m,\theta_1)}l_{(p,\theta_2)}+\frac{4}{9}s_{(m,q)}l_{(n,\theta_1)}l_{(p,\theta_2)} & \text{if } i,j\in[5,10],\\
\quad+\frac{4}{9}s_{(n,p)}l_{(m,\theta_1)}l_{(q,\theta_2)}+\frac{4}{9}s_{(m,p)}l_{(n,\theta_1)}l_{(q,\theta_2)} & \text{and } i=\mathrm{mid}(m,n),\ j=\mathrm{mid}(p,q)\\
\frac{4}{9}s_{(i,q)}l_{(i,\theta_1)}l_{(p,\theta_2)}-\frac{1}{36}l_{(i,\theta_1)}l_{(p,\theta_2)} & \text{if } i\in[1,4],\ j\in[5,10],\\
\quad+\frac{4}{9}s_{(i,p)}l_{(i,\theta_1)}l_{(q,\theta_2)}-\frac{1}{36}l_{(i,\theta_1)}l_{(q,\theta_2)} & \text{and } j=\mathrm{mid}(p,q)\\
\frac{4}{9}s_{(n,j)}l_{(m,\theta_1)}l_{(j,\theta_2)}+\frac{4}{9}s_{(m,j)}l_{(n,\theta_1)}l_{(j,\theta_2)} & \text{if } i\in[5,10],\ j\in[1,4],\\
\quad-\frac{1}{36}l_{(m,\theta_1)}l_{(j,\theta_2)}-\frac{1}{36}l_{(n,\theta_1)}l_{(j,\theta_2)} & \text{and } i=\mathrm{mid}(m,n)
\end{cases} \tag{A.7}$$

Finally, $\mathbf{m}_{(\theta_1,\theta_2)}$ is expressed as:

$$\mathbf{m}_{(\theta_1,\theta_2)} = \begin{cases} \frac{s_y^2 s_z^2}{s_x s_y s_z} = \frac{s_y s_z}{s_x} & \text{if } \theta_1 = x, \theta_2 = x \\ \frac{s_x^2 s_z^2}{s_x s_y s_z} = \frac{s_x s_z}{s_y} & \text{if } \theta_1 = y, \theta_2 = y \\ \frac{s_x^2 s_y^2}{s_x s_y s_z} = \frac{s_x s_y}{s_z} & \text{if } \theta_1 = z, \theta_2 = z \\ \frac{s_x s_y s_z^2}{s_x s_y s_z} = s_z & \text{if } \theta_1 = x, \theta_2 = y \text{ or } \theta_1 = y, \theta_2 = x \\ \frac{s_x^2 s_y s_z}{s_x s_y s_z} = s_x & \text{if } \theta_1 = y, \theta_2 = z \text{ or } \theta_1 = z, \theta_2 = y \\ \frac{s_x s_y^2 s_z}{s_x s_y s_z} = s_y & \text{if } \theta_1 = z, \theta_2 = x \text{ or } \theta_1 = x, \theta_2 = z \end{cases} \tag{A.8}$$

Appendix B. Supplementary material

Supplementary material related to this article can be found online at http://dx.doi.org/10.1016/j.cagd.2015.03.019.

References

Arnout, S., Firl, M., Bletzinger, K.-U., 2012. Parameter free shape and thickness optimisation considering stress response. Struct. Multidiscip. Optim. 45, 801–814.

Bächer, M., Bickel, B., James, D.L., Pfister, H., 2012. Fabricating articulated characters from skinned meshes. ACM Trans. Graph. 31 (4). http://dx.doi.org/10.1145/2185520.2185543. 47, 9 pp.

Bächer, M., Whiting, E., Bickel, B., Sorkine-Hornung, O., 2014. Spin-it: optimizing moment of inertia for spinnable objects. ACM Trans. Graph. 33 (4), 96.

Baran, I., Popović, J., 2007. Automatic rigging and animation of 3d characters. ACM Trans. Graph. 26 (3). http://dx.doi.org/10.1145/1276377.1276467.

Barbič, J., Zhao, Y., 2011. Real-time large-deformation substructuring. In: ACM SIGGRAPH 2011 Papers. SIGGRAPH'11. 91, 8 pp.

Bickel, B., Bächer, M., Otaduy, M.A., Lee, H.R., Pfister, H., Gross, M., Matusik, W., 2010. Design and fabrication of materials with desired deformation behavior. ACM Trans. Graph. 29 (4). http://dx.doi.org/10.1145/1778765.1778800. 63, 10 pp.

Bickel, B., Kaufmann, P., Skouras, M., Thomaszewski, B., Bradley, D., Beeler, T., Jackson, P., Marschner, S., Matusik, W., Gross, M., 2012. Physical face cloning. ACM Trans. Graph. 31 (4), 118.

Calì, J., Calian, D.A., Amati, C., Kleinberger, R., Steed, A., Kautz, J., Weyrich, T., 2012. 3d-printing of non-assembly, articulated models. ACM Trans. Graph. 31 (6). http://dx.doi.org/10.1145/2366145.2366149. 130, 8 pp.

Ceylan, D., Li, W., Mitra, N.J., Agrawala, M., Pauly, M., 2013. Designing and fabricating mechanical automata from mocap sequences. ACM Trans. Graph. 32 (6). http://dx.doi.org/10.1145/2508363.2508400. 186, 11 pp.

Chen, D., Levin, D.I.W., Didyk, P., Sitthi-Amorn, P., Matusik, W., 2013. Spec2fab: a reducer-tuner model for translating specifications to 3d prints. ACM Trans. Graph. 32 (4). http://dx.doi.org/10.1145/2461912.2461994. 135, 10 pp.

Chen, X., Zheng, C., Xu, W., Zhou, K., 2014. An asymptotic numerical method for inverse elastic shape design. In: Proc. of SIGGRAPH. ACM Trans. Graph. 33 (4).

Coros, S., Thomaszewski, B., Noris, G., Sueda, S., Forberg, M., Sumner, R.W., Matusik, W., Bickel, B., 2013. Computational design of mechanical characters. ACM Trans. Graph. 32 (4). http://dx.doi.org/10.1145/2461912.2461953. 83, 12 pp.

Deuss, M., Panozzo, D., Whiting, E., Liu, Y., Block, P., Sokrine-Hornung, O., Pauly, M., 2014. Assembling self-supporting structures. ACM Trans. Graph. 33. EPFL-ARTICLE-201940.

Dong, Y., Wang, J., Pellacini, F., Tong, X., Guo, B., 2010. Fabricating spatially-varying subsurface scattering. ACM Trans. Graph. 29 (4). http://dx.doi.org/10.1145/1778765.1778799. 62, 10 pp.

Dumas, J., Hergel, J., Lefebvre, S., 2014. Bridging the gap: automated steady scaffoldings for 3d printing. ACM Trans. Graph. 33 (4). http://dx.doi.org/10.1145/2601097.2601153. 98, 10 pp.

Dutta, D., Prinz, F.B., Rosen, D., Weiss, L., 2000. Layered manufacturing: current status and future trends. J. Comput. Inf. Sci. Eng. 1 (1), 60–71.

Farhat, C., Roux, F.-X., 1991. A method of finite element tearing and interconnecting and its parallel solution algorithm. Int. J. Numer. Methods Eng. 32 (6), 1205–1227.

Farhat, C., Lesoinne, M., LeTallec, P., Pierson, K., Rixen, D., 2001. Feti-dp: a dual–primal unified feti method, part I: a faster alternative to the two-level feti method. Int. J. Numer. Methods Eng. 50 (7), 1523–1544. http://dx.doi.org/10.1002/nme.76.

Hildebrand, K., Bickel, B., Alexa, M., 2012. Crdbrd: shape fabrication by sliding planar slices. Comput. Graph. Forum 31 (2pt3), 583–592. http://dx.doi.org/10.1111/j.1467-8659.2012.03037.x.

Hu, R., Li, H., Zhang, H., Cohen-Or, D., 2014. Approximate pyramidal shape decomposition. In: Siggraph Asia 2014.

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

Koo, B., Li, W., Yao, J., Agrawala, M., Mitra, N.J., 2014. Creating works-like prototypes of mechanical objects. In: Proceedings of ACM SIGGRAPH Asia. ACM Trans. Graph. 33 (6). Article No. 217.

Lan, Y., Dong, Y., Pellacini, F., Tong, X., 2013. Bi-scale appearance fabrication. ACM Trans. Graph. 32 (4). http://dx.doi.org/10.1145/2461912.2461989. 145, 12 pp.

Lau, M., Ohgawara, A., Mitani, J., Igarashi, T., 2011. Converting 3d furniture models to fabricatable parts and connectors. ACM Trans. Graph. 30 (4). http://dx.doi.org/10.1145/2010324.1964980. 85, 6 pp.

Lu, L., Sharf, A., Zhao, H., Wei, Y., Fan, Q., Chen, X., Savoye, Y., Tu, C., Cohen-Or, D., Chen, B., 2014. Build-to-last: strength to weight 3d printed objects. In: Proc. SIGGRAPH. ACM Trans. Graph. 33 (4). 97, 10 pp.

Liu, Y., Pottmann, H., Wallner, J., Yang, Y.-L., Wang, W., 2006. Geometric modeling with conical meshes and developable surfaces. ACM Trans. Graph. 25 (3), 681–689. http://dx.doi.org/10.1145/1141911.1141941.

Liu, Y.-J., Lai, K.-L., Dai, G., Yuen, M.-F., 2010. A semantic feature model in concurrent engineering. IEEE Trans. Autom. Sci. Eng. 7 (3), 659–665.
Liu, Y., Xu, W., Wang, J., Zhu, L., Guo, B., Chen, F., Wang, G., 2011. General planar quadrilateral mesh design using conjugate direction field. ACM Trans. Graph. 30 (6). http://dx.doi.org/10.1145/2070781.2024174. 140, 10 pp.
Logan, D.L., 2012. A First Course in the Finite Element Method, fifth edition. Cengage Learning.
Luo, L., Baran, I., Rusinkiewicz, S., Matusik, W., 2012. Chopper: partitioning models into 3d-printable parts. ACM Trans. Graph. 31 (6), 129.
McCrae, J., Singh, K., Mitra, N.J., 2011. Slices: a shape-proxy based on planar sections. ACM Trans. Graph. 30 (6). http://dx.doi.org/10.1145/2070781.2024202. 168, 12 pp.
Mori, Y., Igarashi, T., 2007. Plushie: an interactive design system for plush toys. ACM Trans. Graph. 26 (3). http://dx.doi.org/10.1145/1276377.1276433.
Prévost, R., Whiting, E., Lefebvre, S., Sorkine-Hornung, O., 2013. Make it stand: balancing shapes for 3D fabrication. In: Proceedings of ACM SIGGRAPH. ACM Trans. Graph. 32 (4). 81, 10 pp.
Reprap, 2010. Open-source reprap project. http://reprap.org/wiki/RepRap.
Schüller, C., Panozzo, D., Sorkine-Hornung, O., 2014. Appearance-mimicking surfaces. ACM Trans. Graph. 33 (6), 216.
Skouras, M., Thomaszewski, B., Coros, S., Bickel, B., Gross, M., 2013. Computational design of actuated deformable characters. ACM Trans. Graph. 32 (4). http://dx.doi.org/10.1145/2461912.2461979. 82, 10 pp.
Stava, O., Vanek, J., Benes, B., Carr, N., Měch, R., 2012. Stress relief: improving structural strength of 3d printable objects. ACM Trans. Graph. 31 (4). http://dx.doi.org/10.1145/2185520.2185544. 48, 11 pp.
Tao, J., Scott, S., Joe, W., 2005. Mean value coordinates for closed triangular meshes. In: Proceedings of ACM SIGGRAPH.
Thomaszewski, B., Coros, S., Gauge, D., Megaro, V., Grinspun, E., Gross, M., 2014. Computational design of linkage-based characters. ACM Trans. Graph. 33 (4), 64.
Toselli, A., Widlund, O., 2004. Domain Decomposition Methods. Springer.
Umetani, N., Kaufman, D.M., Igarashi, T., Grinspun, E., 2011a. Sensitive couture for interactive garment modeling and editing. ACM Trans. Graph. 30 (4). http://dx.doi.org/10.1145/2010324.1964985. 90, 12 pp.
Umetani, N., Takayama, K., Mitani, J., Igarashi, T., 2011b. A responsive finite element method to aid interactive geometric modeling. IEEE Comput. Graph. Appl. 31 (5), 43–53.
Umetani, N., Igarashi, T., Mitra, N.J., 2012. Guided exploration of physically valid shapes for furniture design. ACM Trans. Graph. 31 (4). http://dx.doi.org/10.1145/2185520.2185582. 86, 11 pp.
Vidimče, K., Wang, S.-P., Ragan-Kelley, J., Matusik, W., 2013. Openfab: a programmable pipeline for multi-material fabrication. ACM Trans. Graph. 32 (4). 136, 12 pp.
Wang, W., Wang, T.Y., Yang, Z., Liu, L., Tong, X., Tong, W., Deng, J., Chen, F., Liu, X., 2013. Cost-effective printing of 3d objects with skin-frame structures. In: Proc. SIGGRAPH Asia. ACM Trans. Graph. 32 (5). Article 177, 10 pp.
Yang, Y., Xu, W., Guo, X., Zhou, K., Guo, B., 2013. Boundary-aware multidomain subspace deformation. IEEE Trans. Vis. Comput. Graph. 19 (10), 1633–1645.
Zhou, Q., Panetta, J., Zorin, D., 2013. Worst-case structural analysis. ACM Trans. Graph. 32 (4). http://dx.doi.org/10.1145/2461912.2461967. 137, 12 pp.
Zhu, L., Xu, W., Snyder, J., Liu, Y., Wang, G., Guo, B., 2012. Motion-guided mechanical toy modeling. ACM Trans. Graph. 31 (6). http://dx.doi.org/10.1145/2366145.2366146. 127, 10 pp.
Zienkiewicz, O.C., Zhu, J.Z., 1987. A simple error estimator and adaptive procedure for practical engineering analysis. Int. J. Numer. Methods Eng. 24 (2), 337–357.
Zienkiewicz, O., Taylor, R., Zhu, J., 2008. The Finite Element Method: Its Basis & Fundamentals. Elsevier.