GRIDiron: An interactive authoring and cognitive training foundation for reconstructive plastic surgery procedures

Nathan Mitchell
University of Wisconsin-Madison

Court Cutting
New York University

Eftychios Sifakis
University of Wisconsin-Madison

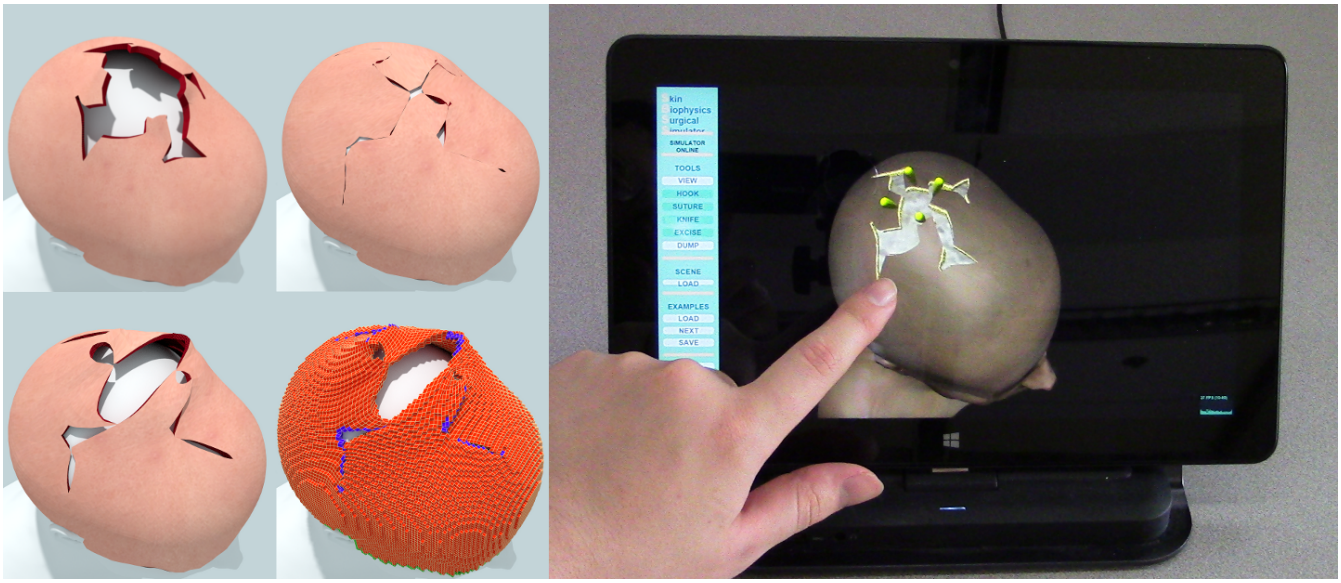

Figure 1: *Simulated **Dufourmentel-Mouly** repair (see [Baker 2014]) for a large gap of excised tissue on the scalp. From top to bottom, left to right: Rendered stages of procedure, embedding lattice, real time demo on a tablet running in a web browser.*

Abstract

We present an interactive simulation framework for authoring surgical procedures of soft tissue manipulation using physics-based simulation to animate the flesh. This interactive authoring tool can be used by clinical educators to craft three-dimensional illustrations of the intricate maneuvers involved in craniofacial repairs, in contrast to two-dimensional sketches and still photographs which are the medium used to describe these procedures in the traditional surgical curriculum. Our virtual environment also allows surgeons-in-training to develop cognitive skills for craniofacial surgery by experimenting with different approaches to reconstructive challenges, adapting stock techniques to flesh regions with nonstandard shape, and reach preliminary predictions about the feasibility of a given repair plan. We use a Cartesian grid-based embedded discretization of nonlinear elasticity to maximize regularity, and expose opportunities for aggressive multithreading and SIMD accelerations. Using a grid-based approach facilitates performance and scalability, but constrains our ability to capture the topology of thin surgical incisions. We circumvent this restriction by hybridizing the grid-based discretization with an explicit hexahedral mesh representation in regions where the embedding mesh necessitates overlap or nonmanifold connectivity. Finally, we detail how the front-end of our system can run on lightweight clients, while the core simulation capability can be hosted on a dedicated server and delivered as a network service.

CR Categories: I.3.5 [Computer Graphics]: Computational geometry and modeling—[Physically based modeling]

Keywords: Virtual surgery, finite elements, elasticity.

1 Introduction

The art and practice of plastic surgery is intimately tied with a lifelong learning experience both in establishing theoretical foundations, such as anatomy and pathology, as well as in acquiring and sharpening surgical skills. Although the requisite skill set overlaps with that of general surgery at large, plastic surgery has a distinct focus on the utilization of topology change as a treatment mechanism. Fortunately, a few opportunities exist for surgeons to acquire and develop these skills outside of the operating room. Some options involve operating on physical proxies, such as phantom materials, cadavers or animal tissues. Alternatively, computer technology can be leveraged to provide a non-invasive training testbed. The field of computer graphics is well positioned to contribute to this activity, given its vested interest in modeling virtual materials and digital models of anatomy. This application area, however, presents a unique set of challenges which may be uncharacteristic for traditional graphics applications. In particular, clinical measures of success (does the product improve quality of patient care?) may well differ from visual quality metrics in graphics. Additionally, given that computer-based surgical training is still an exploratory proposition, practical solutions need to be concerned with longevity, extensibility, deployment cost and ease of use.

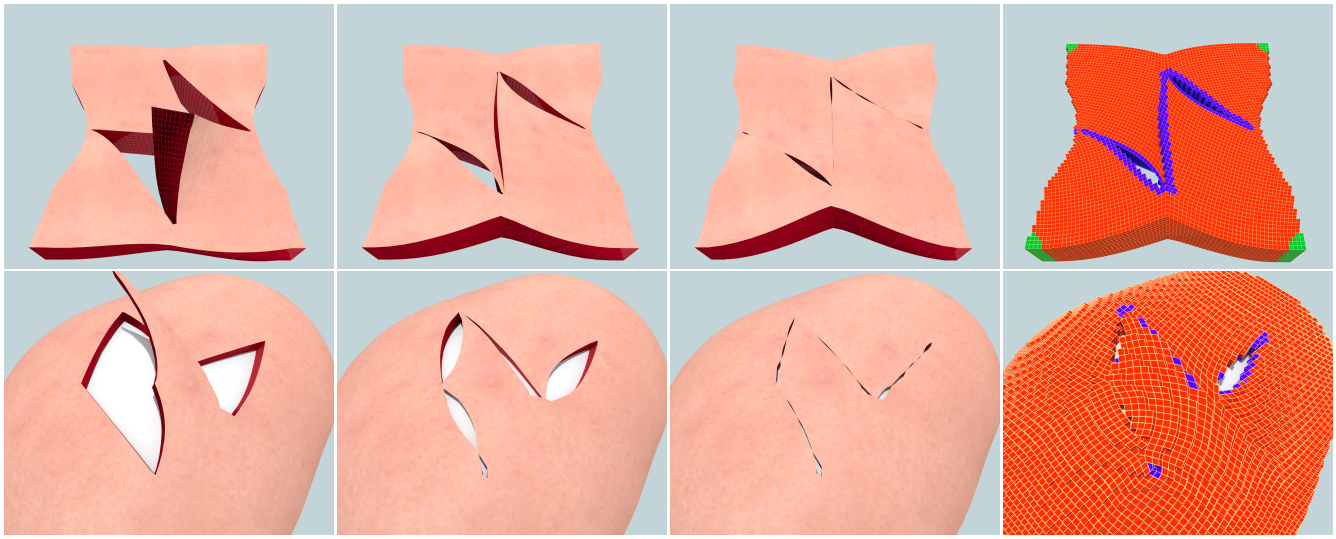

Figure 2: *Top: Simulation of **Z-Plasty**, a basic maneuver resulting in anisotropic scaling along two perpendicular directions. Z-Plasty is a common building block leveraged in more elaborate repairs. Bottom: **Rhomboid flap** procedure for closing a quadrilateral aperture of excised tissue on the scalp. The embedded simulation grids are shown on the right.*

Surgical skills targeted by computer-based training solutions have been classified [Gallagher et al. 2005] in two major categories. *Psychomotor* skills refer to the dexterous use of the surgeon's hands to manipulate instruments in the course of an operation. In plastic surgery, psychomotor training involves mechanical aspects of surgical tasks, such as the "feel" of tissue being cut or the nuances of manipulating a scalpel to enact a curved incision. For example, training for laparoscopic procedures requires a clinician to be familiar with the tactile response of pushing and pulling on organs and to practice coordination skills required for suturing and cauterization. A number of computer-based solutions focus on psychomotor training [Mendoza and Laugier 2003; De et al. 2005; Kim et al. 2007; Lindblad and Turkiyyah 2007]. In contrast to psychomotor training, *cognitive* skills and training are largely mental rather than dexterous exercises. For example, in the procedure shown in Figure 1, the surgeon needs to contemplate how to best repair a large square skin defect (i.e. area of excised tissue) by making auxiliary incisions that create properly shaped "puzzle pieces" which can be sutured together without creating excessive stress. Chentanez et al. [2009] described a cognitive training system for steerable needle insertion, where the mental challenge lies in planning a sequence of actions involving needle flexion and torsion, in order to achieve a desired insertion trajectory.

Cognitive training is paramount in plastic surgery and acquired through many years of practical experience. Most reconstructive procedures are not envisioned or taught as monolithic operations; instead, they are composed as a sequence of fundamental building blocks. Graphics practitioners would associate these elementary actions with geometric transforms: shear, rotation, uniform or anisotropic scaling. Of course, applying such transformation on live tissue is very different than their application on a geometric model. When stretching a tissue patch in one direction to 125% of its original length while contracting it in the transverse direction down to 80%, squeezing-and-stretching in-place is typically *not* the desired way to execute this transform. Real skin might not stretch that far or buckle in the transverse direction during the process. A maneuver called the *Z-plasty* (see Figure 2, top) achieves the same net effect, with a more graceful stress distribution and a smooth blend to the surrounding tissues. In plastic surgery, cognitive training addresses the mental challenge of how these elemental surgical puzzle-pieces are sequenced and adapted to craft a complex operation. Unfortunately, 3D computer-based cognitive training solutions for facial reconstructive surgery are virtually nonexistent; common educational materials are limited to 2D sketches and still photographs of procedures.

We present an interactive virtual surgery simulation framework using lattice-based embedded discretizations of elastic tissue, which can be used to author plastic surgery procedures and serve as a cognitive training tool for novice surgeons. Our system integrates a number of mature discretization and numerical solution techniques with new data structures, aggressive parallelization practices and a remote delivery mechanism for computation, in order to improve fidelity, interactivity and ease-of-use. Although we focus on surgical simulation, we believe the utility of our methods transcends the scope of this clinical application and can be leveraged in broader scenarios of interactive simulation.

Our key technical contributions include:

- A geometric data structure that hybridizes a grid-based discretization with an explicit hexahedral mesh in order to resolve non-manifold topological features, such as narrow incisions, while preserving as much regularity and parallel potential of uniform grids as possible.
- An object-oriented programming paradigm via which Cartesian grid-based embedded discretizations of elastic models can be systematically mapped on a spectrum of multithreaded and SIMD-oriented platforms. This is demonstrated on both the `x86` SSE/AVX instruction set, as well as on the Intel Xeon Phi platform.
- A tiered deployment strategy where the interactive front-end is mapped to a lightweight client device, geometrical modeling is handled by a remote server, and numerical solver by a many-core accelerator.

In addition to the technical merit of our methodology, we hope that our work serves to improve awareness of a highly rewarding emerging application which can greatly benefit from innovations in graphics and visual computing.

2 Related Work

Computer graphics research has produced several diverse techniques for model deformation. Procedural techniques [Joshi et al. 2007; Wang and Phillips 2002; Kavan et al. 2008] offer real-time performance for certain animation tasks but lack the physical accuracy needed in surgical simulations. Consequently, some research ventures into surgical simulation turned to elastic deformation models [Terzopoulos et al. 1987] that responded more realistically to scenarios of probing and cutting [Bro-nielsen and Cotin 1996; Mendoza and Laugier 2003; Nienhuys and van der Stappen 2001]. However, these early works were limited in their scope and effectiveness due to computational cost, geometric constraints and oversimplified material models. In this section we outline prior contributions that help address these limitations, and review a number of existing surgical simulation systems.

Virtual materials and anatomy modeling Approaches based on the Finite Element Method (FEM) have been particularly popular in the medical simulation community [Marchal et al. 2008] where the need for biologically accurate materials is more pronounced. In one of the earliest uses of advanced materials in computer animation, Chen and Zeltzer [1992] focused on anatomical structures such as muscles. FEM techniques were further leveraged in the animation literature for the discretization of linear elasticity for fracture modeling in a small-strain regime [O'Brien and Hodgins 1999]. Highly nonlinear materials such as active musculature [Teran et al. 2003] exposed challenges in robustness and numerical stability of FEM discretizations. Invertible FEM [Irving et al. 2004] improved simulation robustness in scenarios involving extreme compression, while modified Newton methods [Teran et al. 2005a] reduced the cost of implicit schemes with large time steps. Several of these algorithms have been incorporated in open-source modeling and simulation packages [Sin et al. 2013]. Solutions have also been proposed for material behaviors such as incompressibility [Irving et al. 2007] and viscoelasticity [Goktekin et al. 2004; Wojtan and Turk 2008], both of which can be found in typical biomaterials. Recent results in coupled Lagrangian-Eulerian simulation of solids have also facilitated the inclusion of intricate contact and collision handling in biomechanical modeling tasks [Sueda et al. 2008; Li et al. 2013; Fan et al. 2014].

Simulation of topology change A number of techniques have targetted topology change during simulation, due to cutting or fracture. Early work [Terzopoulos and Fleischer 1988] resorted to breaking connectivity of elements when stress limits were exceeded. Later methods [Nienhuys and van der Stappen 2001] split tetrahedra near cut boundaries and then used vertex snapping to more accurately approximate the cut. Local remeshing was also employed to simulate cracks in brittle materials [O'Brien and Hodgins 1999]. An issue with such subdivision schemes is the possible creation of poorly conditioned elements, which prompted a number of authors to pursue embedded simulation schemes [Molino et al. 2004; Teran et al. 2005b]. These techniques use non-conforming meshes with elements which are only partially covered by material, in lieu of conforming remeshing. Embedded simulation can provide a great degree of flexibility in cutting and fracture scenarios [Sifakis et al. 2007], although cutting meshes along arbitrary surfaces requires delicate book-keeping and careful handling of degeneracies.

Embedded simulation The use of non-conforming meshes has valuable benefits for fracture modeling, but embedded techniques have also been used in their own right for reasons of simplicity and performance. Mueller et al. [2004] employed an embedding scheme to perform volumetric simulation of objects described by their boundary mesh. The regularity of lattice embeddings has also been exploited in shape matching techniques [Rivers and James 2007] to achieve significant performance accelerations. Embedding has been combined with homogenization [Nesme et al. 2006; Kharevych et al. 2009] to resolve sub-element variation of material parameters, optionally with the use of non-manifold embedding lattices to support objects with a branching structure [Nesme et al. 2009]. Jerabkova et al. [2010] employed a method similar to our own, using a finer voxel grid to capture material topology to be embedded in a coarser, non-manifold voxel grid. Finally, Zhao and Barbič [2013] demonstrated the use of multiple voxel grid domains to segment a model hierarchically, which they used to simulate plants at interactive rates. Extended FEM (XFEM) formulations have also been explored [Jeřábková and Kuhlen 2009], where discontinuities are introduced into the element's shape functions, to model cutting. In a similar vein, Kaufmann et al. [2009] used discontinuous Galerkin FEM formulations. Others have dispensed with mesh based discretizations completely, preferring meshless methods [De and Bathe 2000] which were also used for surgical simulation [De et al. 2005].

Surgery simulation While much of the previously discussed work is geared towards general elastic body simulation in computer graphics, many relevant results originated in surgery-specific work. Pieper et al. [1995] demonstrated a very early surgical simulation platform for facial procedures, using FEM elastic shells. Many surgical simulation projects focus on the mechanical manipulation of organs and other soft internal objects [Nienhuys and van der Stappen 2001; Kim et al. 2007]. Even expensive commercial simulators like the Lap Mentor and GI Mentor primarily focus on pushing and cutting simulated internal organs [Simbionix USA Corporation 2002–2014b; Simbionix USA Corporation 2002–2014a]. These types of simulations are so common that several open source frameworks have been built to specifically support further development [Allard et al. 2007; Cavusoglu et al. 2006]. These provide easy access to common components like haptic feedback and APIs to connect multiple simulated components. Certain surgical simulation systems are tailored to specific skills, including interactive simulations of needle insertion [Chentanez et al. 2009].

Performance optimizations Improving simulation rates is a common challenge for many interactive modeling tasks, and even more so for accuracy-conscious applications such as virtual surgery. Attempts to improve performance have either relied on new data structures, faster solvers, or aggressive use of parallelization. The Boundary Element Method [James and Pai 1999] has been used to achieve interactive deformation rates for objects manipulated via their surface. Hermann et al. [2009] analyzed data flow in their simulations to inform a parallel scheduler for multicore systems. To avoid write hazards during parallel code execution, Kim et al. [2011] proposed a system of computation phases with coalesced memory writes, which allowed them to parallelize force computation. Related efforts by Courtecuisse and Allard [2009], developed a parallel version of the Gauss-Seidel algorithm that can run on GPUs. Regular discretizations have also been coupled with multigrid solvers to facilitate GPU accelerations for elastic skinning techniques [McAdams et al. 2010]. However, in spite of the efficiency of multigrid schemes, adapting them to the presence of incisions or other intricate topological features can be a nontrivial proposition.

3 Project scope and design decisions

Our target application carries a unique set of requirements related to accuracy, usability and effectiveness. We sought to produce a robust and usable prototype, appropriate for a pilot deployment (Figure 3). As a consequence, our design goals represent a balance between facilitating future extensions and leveraging all opportunities for optimizing performance and interactivity *now*. This section discusses the factors influencing our design decisions, including both long-term principles and short-term opportunities.

3.1 Clinical scope

In the current iteration of our framework, we focused on the virtual simulation of *local flaps*, as well as anatomical flaps on the *human cranium*. Local flaps are surgical designs defined on a flat tissue layer with uniform thickness, which is not attached to underlying bone or cartilage (Figure 2, top). Cranial flaps (Figure 2, bottom) are somewhat similar in that the tissue has relatively uniform thickness (of a few millimeters) and is not attached to the underlying bone. Of course, cranial flaps are not flat, and similar incision shapes can produce different end results, depending on the curvature of the cranium. Procedures on the cranium are a significant (and nontrivial) part of the clinical curriculum in a plastic surgery residency program.

Geometry of repairs Due to the small, uniform thickness of tissue in local and cranial flaps, incisions can be designed purely as curves on the skin surface, with the assumption that any incision will proceed through the entire thickness of the tissue (perpendicular to the surface). This specific scope greatly simplifies user interface, as cuts can be drawn on the tissue surface. In contrast, surgical procedures on anatomical parts with greater volumetric extent would necessitate complex input devices to prescribe 3D incision paths.

Contact scenarios Surgical repairs on the cranium certainly depend on proper contact between the elastic tissue and the underlying bone, but do not strictly require static contact between different tissue parts, as all incisions are thoroughly sutured after manipulation (not the case for operations in other parts of the body). Accordingly, our system only performs collisions between tissue and bone, forgoing expensive self-collision processing. We designed our system to be *extensible* to self-collision handling, but optimized our current deployment with the knowledge that only collisions with external bodies are needed at this juncture.

Timeline of incisions A large majority of surgical repairs on the cranium can be designed with all incisions made *at once*, in the beginning of the procedure. This allows us to improve the efficiency and robustness of incision modeling dramatically, applying all topological change at a single instance in time. This is certainly not the case for other procedures (e.g. cleft lip and palate repair or other procedures on the lower face) where incisions have to be applied in distinct time instances, while tissue is actively manipulated.

3.2 Physics-based modeling

Material properties of tissue Human tissue is a complex heterogeneous nonlinear material, with anisotropic, elastoplastic and viscoelastic traits. Nevertheless, although plasticity and viscoelasticity are important postoperative factors, our domain collaborators have suggested that these behaviors are of secondary importance within a *surgical timeframe* in determining whether a given repair is mechanically feasible or not. As a consequence, our framework is designed to accommodate heterogeneous *hyperelastic* materials, with no explicit provisions at this point for plasticity or viscoelasticity. Anisotropy is a much more pertinent trait; even on the cranium, the bottom of the scalp is lined with a thin fibrous sheet that is a significant source of material anisotropy (in-plane stiffness vs. normal to the scalp surface). Our design choice was to outfit our algorithmic framework with full support for heterogeneity and substantial support for anisotropy, including active anisotropic behaviors such as contractile muscle fibers. However, our interactive surgical examples were carried out with isotropic materials exclusively, and our support for anisotropy and active muscle elements is only leveraged to offline musculoskeletal simulations within our framework (see Section 7).

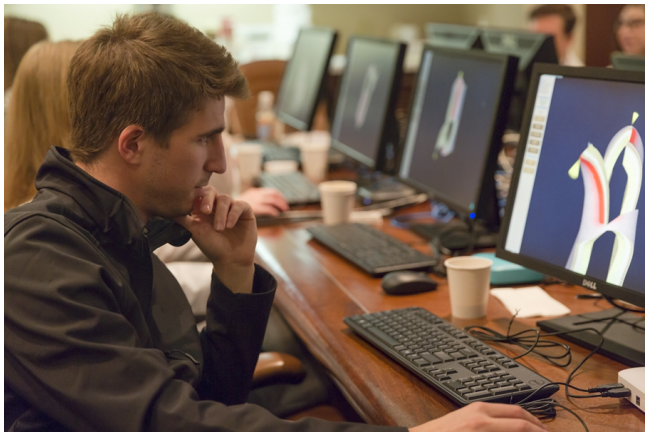

Figure 3: *Pilot deployment of our interactive web-based simulator with 14 Plastic & Reconstructive Surgery medical residents at the University of Wisconsin-Madison.*

Dynamics As a cognitive training platform, our framework's main priority is to capture the *stationary* shape that a given repair results in. As a consequence, an implicit or even quasistatic time evolution scheme is very well tolerated for our application, and aligns well with the interactivity requirements of our platform. The cost paid for this compromise is that transient deformation states during high-velocity maneuvers are not dynamically accurate, but merely iterations towards the solution of the implicit/quasistatic evolution scheme used.

Solvers, scalability and parallelization A number of authors have been successful in leveraging multigrid schemes [Dick et al. 2011; McAdams et al. 2011] in the simulation of elastic solids. For our specific application, the geometry of the computational domain (thin volumetric tissue sheets) would complicate the design of multigrid transfer operators, while future extensions to highly anisotropic and heterogeneous materials would require nontrivial adaptations to a performance-tuned multigrid scheme. Thus, we avoided committing to a multigrid solver at this point, and relied on iterative solvers such as Conjugate Gradients. That being said, we could not afford to forego parallelism and vectorization, as the performance benefit over a scalar single-threaded implementation would be dramatic. In light of this, we adopted a Cartesian-grid based embedded discretization that favors parallel accelerations while (i) designing a hybrid embedded structure that reconciles lattice embeddings with non-manifold topology near incisions and (ii) implementing a programming paradigm to systematically generate multi-threaded and vectorized versions of our algorithmic kernels as material properties might be adapted and refined.

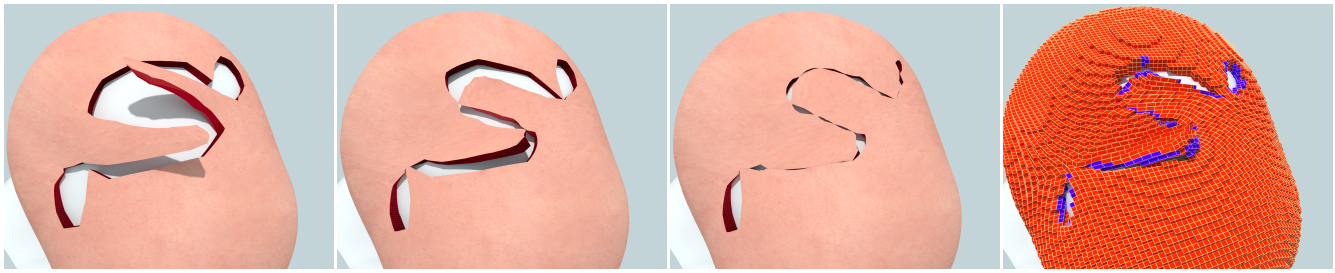

Figure 4: ***Dual S-Plasty*** *procedure on the scalp; a large oval-shape defect is patched by transposing two S-shaped flaps.*

3.3 User experience and ease of deployment

It is essential for the continued evolution of our framework to establish a communication channel between computer graphics developers and clinical end-users. Thus, we place great emphasis on a system that can be deployed to our domain testers quickly and inexpensively, provides a versatile upgrade path, and is intuitive and enjoyable to use as to elicit continued interest from our clinical collaborators.

Simple interface We aim to provide a simple, yet expressive and intuitive interface to clinical users of our system. Our current solution is centered around three basic interface "tools": (a) The *incision tool* is a drawing interface by which cuts are sketched as spline curves on the tissue surface. (b) The *hook tool* establishes soft spring-based constraints between a point on the visible surface of the tissue and a user-movable point in space. (c) The *suture tool* enacts a suture between two points on the tissue surface. The attachment is enforced as a stiff zero-length spring.

Remote deployment In order to improve simulation performance, we aggressively use parallel-optimized solvers on multicore systems (our platform uses an SMP server, with Intel Xeon Phi accelerators). It is, however, impractical to deploy such a server at the location of our end-users (who are surgical residents in teaching hospitals), and even more challenging to push algorithmic upgrades as our framework evolves. We have thus separated the graphical front-end into a lightweight application that runs on web-based clients (Figure 3), while the burden of numerical simulation is offloaded to a simulation server. This is possible as the front-end only needs to dispatch user manipulations (movements of hooks, placement of sutures) to the host simulator, and only receive *surface* geometry updates from the simulation server even if volumetric simulation is taking place.

4 Elasticity discretization

We start by describing our discretization of elastic forces and the time evolution scheme used in our methodology. We discretize the equations of elasticity on a regular Cartesian lattice, on which the simulated material is embedded. Although we actually allow our grids to contain duplicate cells and non-manifold features, as we detail in Section 5, each individual cell is a regular cube element, as in previously published lattice-based discretizations [McAdams et al. 2011; Patterson et al. 2012]. We review the relevant theory with a focus on exposing regular algorithmic kernels, which we feed into a stream processing pipeline in Section 6.

Quasistatic evolution As we are primarily concerned with the equilibrium shapes resulting from user manipulation and flesh/bone collision, we implement a quasistatic evolution based on a Newton-Raphson solver with Conjugate Gradients used to solve the linearized system at each step. For every configuration of constraints or kinematically prescribed degrees of freedom, we seek to solve for the nodal displacements x such that the force equilibrium equation $f(x) = 0$ is satisfied. At each iteration k we aim to refine our solution for $x_{(k)}$ by seeking a correction term dx such that:

$$0 = f(x_{(k)} + dx) \approx f(x_{(k)}) + \left.\frac{df}{dx}\right|_{x_{(k)}} dx \Rightarrow$$

$$\underbrace{\left(-\left.\frac{df}{dx}\right|_{x_{(k)}}\right)}_{\mathbf{K}} dx = f(x_{(k)}) \tag{1}$$

Equation 1 highlights the two terms we must be able to provide for each force component within our system, including elastic and contact forces. The right hand side is the total force as a function of the current position. If we had employed a direct method, the left hand side could have been represented by extracting the tangent stiffness matrix $\frac{df}{dx}$. However, our iterative Krylov solver only requires an algorithmic routine for computing products of the form $\mathbf{K}w$. We transform our problem into one of this form by using the notion of a differential. Interpreting w as a nodal displacement δx, we rewrite the left hand side of equation 1 as:

$$-\left.\frac{df}{dx}\right|_{x_{(k)}} \delta x := -\delta f[\delta x; x_{(k)}] \tag{2}$$

The left hand side is now in a form equivalent to $\mathbf{K}w$, suitable for use in our iterative solver. The following sections detail how forces $f(x_{(k)})$ and force differentials $\delta f[\delta x; x_{(k)}]$ are computed for every different force type in our method.

4.1 Elastic Forces

Isotropic elasticity Our system models the underlying mechanical behavior of soft tissue with an isotropic substrate, which is optionally augmented with additional anisotropic force contributions (either from passive fibrous structures, or active musculature). For the discretization of the isotropic component we derive forces and force differentials using FEM with trilinear shape functions and a one-point quadrature [McAdams et al. 2011; Patterson et al. 2012; Sifakis and Barbic 2012]. The exact process for forces and differentials is reiterated in Algorithm 1. We note the existence of two user-defined functions in the pseudocode:

```
DiagMatrix<3> P_Hat(DiagMatrix<3> Sigma, Params p)
Vector<12> dPdF_Hat(DiagMatrix<3> Sigma, Params p)
```

By providing a custom implementation of the above two functions, we can implement *any* isotropic material model without modifying any other part of the force computation pipeline. This is a flexibility that we strive to retain, to support more complex biomaterials in the future. Note that the

Algorithm 1 Procedures for computing elastic force and force differentials. Sections highlighted in gray indicate values which can be precomputed at the start of each Newton-Raphson iteration. The functions `P_Hat` and `dPdF_Hat` can be user-defined to implement arbitrary *isotropic* materials. The **reshape** operation concatenates eight nodal vectors in a cell (positions, displacements, forces, etc) into a 3×8 matrix and vice versa. $\mathbf{G}$ is a 3×8 gradient matrix, as defined in [McAdams et al. 2011]. $V_0 = h^3$ is the volume of the cartesian cell. $\mathcal{T}$ is a sparse 4th order tensor as defined in [Teran et al. 2005a]

procedure FELASTIC($\mathbf{x}$, params p)
  reshape $\mathbf{x} \rightarrow \mathbf{D}_s$
  compute $\mathbf{F} \leftarrow \mathbf{D}_s\mathbf{G}^T$
  compute $[U, \Sigma, V] \leftarrow \text{SVD}(\mathbf{F})$
  compute $\hat{\mathbf{P}} \leftarrow \text{P\_HAT}(\Sigma, p)$
  compute $\mathbf{P} \leftarrow \mathbf{U}\hat{\mathbf{P}}\mathbf{V}^T$
  set $\mathbf{H} \leftarrow V_0\mathbf{P}\mathbf{G}$
  reshape $\mathbf{H} \rightarrow \mathbf{f}$

$\mathbf{P}(\mathbf{F})$

procedure δFELASTIC($\mathbf{x}$,$\delta\mathbf{x}$, params p)
  reshape $\mathbf{x} \rightarrow \mathbf{D}_s$, $\delta\mathbf{x} \rightarrow \delta\mathbf{D}_s$
  compute $\mathbf{F} \leftarrow \mathbf{D}_s\mathbf{G}^T$, $\delta\mathbf{F} \leftarrow \delta\mathbf{D}_s\mathbf{G}^T$
  compute $[U, \Sigma, V] \leftarrow \text{SVD}(\mathbf{F})$
  compute $\mathcal{T} \leftarrow \text{DPDF\_HAT}(\Sigma, p)$
  compute $\delta\hat{\mathbf{F}} \leftarrow \mathbf{U}^T\delta\mathbf{F}V$
  compute $\delta\hat{\mathbf{P}} \leftarrow \mathcal{T} : \delta\hat{\mathbf{F}}$
  compute $\delta\mathbf{P} \leftarrow \mathbf{U}\delta\hat{\mathbf{P}}\mathbf{V}^T$
  set $\delta\mathbf{H} \leftarrow V_0\delta\mathbf{P}\mathbf{G}$
  reshape $\delta\mathbf{H} \rightarrow \delta\mathbf{f}$

$\delta\mathbf{P}(\mathbf{F}, \delta\mathbf{F})$

second function returns a sparse $3\times3\times3\times3$ tensor, which has 21 nonzero entries out of which only 12 are unique (reflected in the return type), as defined by Teran et al. [2005a]. Concrete formulas for the above functions have been previously given [Patterson et al. 2012] for Corotated, Neohookean and Mooney-Rivlin isotropic materials.

Anisotropy Our framework supports adding an anisotropic response on top of the base isotropic substrate. We allow custom anisotropic material definitions by accepting an alternative implementation of the Piola Stress $\mathbf{P}(\mathbf{F})$ and its differential $\delta\mathbf{P}(\mathbf{F};\delta\mathbf{F})$. Since our surgical examples do not currently make use of this capability, we defer to the supplemental document for details on how the muscle constitutive model of Teran et al [2003] can be cast in these terms. An offline musculoskeletal simulation using this feature is discussed in Section 7.

Stabilization For performance, our discretization method uses a single point quadrature scheme, which requires only one Singular Value Decomposition per cell as opposed to a more traditional Gauss quadrature which requires eight. As detailed in McAdams et al [2011], left uncorrected, this approach produces hourglass instabilities due to particular deformation modes which do not affect the discrete energy, and we employ their stabilization corrections to account for this issue. If additional accuracy is desired, a four point quadrature scheme proposed in Patterson et al [2012] could easily be substituted, at an additional computation cost.

4.2 Interaction Forces

In addition to internal elastic forces, our system supports interaction forces related to both user interaction and collision with kinematic objects. User interaction consists of manipulation *hooks*, which are spring attachments of moderate stiffness between a tissue surface location and a user-controlled point in space, as well as *sutures* which are significantly more stiff springs attaching two locations on the tissue surface. Both of these forces result in augmentation of the internal elastic energy by spring terms of the form:

$$E_{spring} = \sum \frac{k_i}{2} ||p_i^{(1)} - p_i^{(2)}||_2^2$$

where k_i is the spring constant and the endpoints can take one of two forms:

$$p_i^{(j)} = \mathbf{W}\mathbf{x} \quad \text{or} \quad p_i^{(j)} = \text{const}$$

In the first case, $p_i^{(j)}$ represents a embedded spring endpoint where $\mathbf{W}$ is a 3×24 weight matrix and $\mathbf{x}$ are the positions of the vertices from the embedding cell. The second case refers to a non-embedded target, where $p_i^{(j)}$ is simply a constant position in space. Hook constraints use one embedded and one fixed endpoint, while suture constraints have both endpoints embedded. In either case, forces and differentials are directly computed by differentiating the spring energy as $f_{spring} = -\partial E_{spring}/\partial \boldsymbol{x}$, $\delta f_{spring} = -\partial^2 E_{spring}/\partial \boldsymbol{x}^2 : \delta\boldsymbol{x}$.

Collisions with kinematic bodies are handled using the "hook" infrastructure; a number of collision proxies are seeded on the model surface, and when collision is detected their stiffness is set to a nonzero value, while their target endpoint is set to the projection on the surface of the colliding body. The suture infrastructure can be used to support self-collisions, following the paradigm of McAdams et al. [2011]. The only difference is that, for sutures, the embedding of both spring endpoints is known at the moment the suture is introduced. For self collisions, the embedding coordinates of one endpoint for every "self-collision spring" can be determined dynamically by the collision detection phase.

5 A hybrid embedding lattice structure

We employ an embedded simulation similar to other authors, who used regular lattice embeddings for performance [Müller et al. 2004; Rivers and James 2007; McAdams et al. 2011]. However, due to the presence of extremely thin incisions in our surgical models, standard lattice embedding would not be able to resolve the tissue topology, unless an extremely high resolution embedding was used. We thus adopt a non-manifold lattice-derived embedding discretization in the spirit of Virtual Node or XFEM methods [Molino et al. 2004; Sifakis et al. 2007; Nesme et al. 2009]. Although the use of non-manifold embedding meshes recovers much of the topological expressive ability of conforming meshes, it jeopardizes one of the most attractive features of regular embedding lattices, the fact that connectivity is implicit in the lattice structure as opposed to explicitly stored in a mesh. The performance impact of implicitly defined topology can be profound; the memory footprint of explicitly stored connectivity information can easily exceed the state variables themselves (e.g. vertex positions) and reduce effective memory bandwidth by necessitating indirect memory access. In our work, we strive to leverage the best of both worlds: We

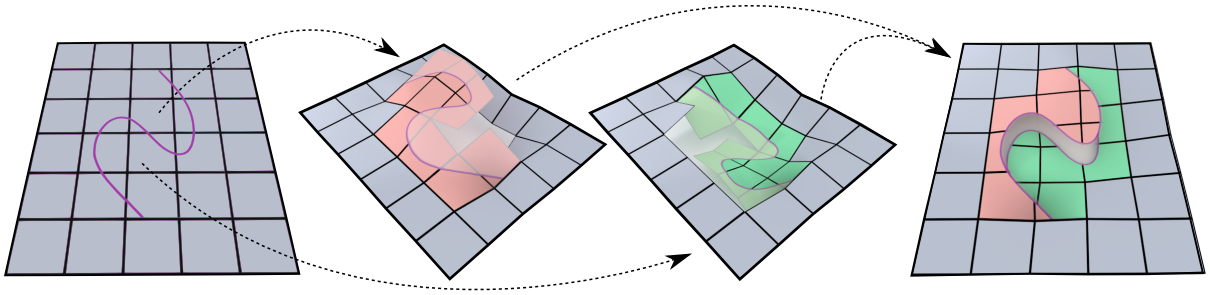

Figure 5: *Illustration of a cut generating a hybrid lattice. (a) A cut passing through the grid. (b) Mesh cells generated for the top half of the cut. (c) Mesh cells generated for the bottom half of the cut. (d) Cut surface is colored to show cell assignment.*

use an (implicit topology) Cartesian grid to capture the majority of the embedded model in regions where non-manifold duplication is not needed. We retain the topological flexibility of non-manifold embedding lattices by hybridizing this grid with an (explicit topology) hexahedral mesh used to describe regions in the vicinity of narrow slits and incisions.

5.1 Surface Model

Prior to elasticity discretization, a watertight surface model of the flesh, including any incisions, must be created. In our system, incisions are generated from user specified line segment curves, which guide constructive solid geometry (CSG) difference operations to produce cut surface meshes. We begin from a user specified line segment curve from which we construct prisms by thickening the line segments tangentially and perpendicularly along the surface normal. We then apply these prisms in a subtraction operation with the surface, resulting in a slightly thickened incision (Figure 6). Disconnected regions produced during this step can be marked and removed by the user. Note that for scenarios involving malignant tissue, discarding of excised tissue is commonplace.

5.2 Rasterization

Given a cut surface mesh, we first create a fine rasterization of the surface. The resolution of the rasterization is selected to capture all desired topological detail (typically an order of magnitude finer than simulation resolution). In Figure 7, it is possible to see the fine rasterization grid in contrast to the coarser simulation resolution. The rasterization is performed by detecting all voxels intersected by the object surface and flood-filling to mark the volumetric material region. Once the rasterization is complete, subsequent embedding operations are purely combinatorial, and not sensitive to poor conditioning of surface mesh elements. Additionally, this fine-grid embedding can also act as an interface layer to more complex embedding schemes, such as the non-manifold approach described next. We leverage this by translating any deformation results back to the fine-grid embedding prior to rendering, to hide non-manifold embedding or numerical solution details from the visual front-end.

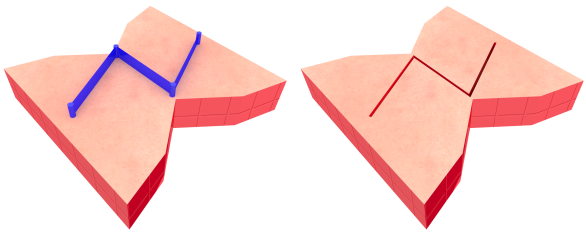

Figure 6: *Incisions in the flesh surface model are created by extruding and thickening user specified line segments.*

5.3 Non-Manifold mesh generation

We now seek to construct a coarser resolution explicit mesh discretization, which is allowed to be non-manifold in regions, as shown in Figure 5. We will extend the paradigm of non-manifold embedding proposed by Teran et al. [2005b] and Sifakis et al. [2007] using the precomputed fine grid rasterization to answer material connectivity predicates. Our non-manifold mesh generation process is outlined in Algorithm 2, and illustrated intuitively in Figure 5. Note that, in the pseudocode provided, there are two geometric predicates being used: (a) Determination of material components (line 3) requires the identification of all disconnected components of material present in the intersection of our domain with a given lattice cell. (b) Adjacent-element material continuity (line 10) is a predicate invoked to determine if two material fragments, associated with adjacent lattice cells, exhibit material continuity across their common face. These two geometric predicates are expressed in a fashion that is agnostic to the underlying geometric representation of material; in Teran et al. [2005b] the assumption is that a tetrahedralized model of the material is available, while Sifakis et al. [2007] define material fragments indirectly, by specifying cutting surfaces instead. In our case, the availability of the fine-grid rasterization makes both such operations purely combinatorial in nature. Material fragments within a coarse cell are computed via flood-fill, and fragments on adjacent cells are continuous if they contain adjacent *fine* cells on their rasterization. At the conclusion of this step, we have produced a coarse mesh (with explicitly stored connectivity), whose topology is as close as possible to the embedded geometry.

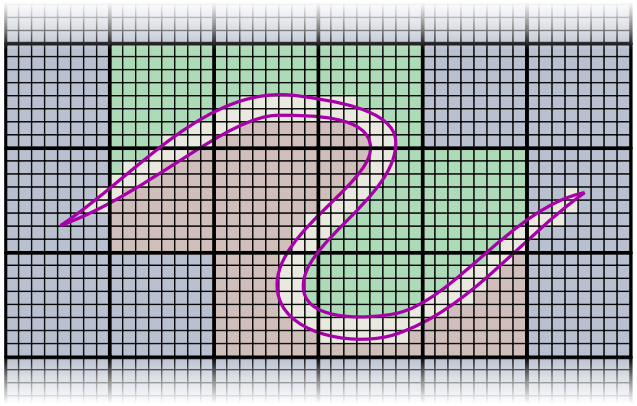

Figure 7: *Fine grid rasterization of a cut. Fine cells within the cut are empty, and colored to show material continuity.*

Algorithm 2 Algorithm to Construct Non-Manifold Mesh

Input: Coarse Resolution

```
function GENERATE_NONMANIFOLD_MESH
    for all Coarse Cells: i do
        C ← DETERMINE_MATERIAL_COMPONENTS(i)
        for all Components in C do
            Instance separate copy of i
            Generate unique, separate DOFs
            Assign descriptor of material content
    for all Geometrically adjacent cell pairs: (i, j) do
        for all Pairs of duplicates from i and j: (h, k) do
            if MATERIAL_IS_CONTINUOUS(h, k) then
                Mark shared vertices as equivalent
    for all Coarse Cells: i do
        Compare all duplicates of i
        Collapse duplicates with equivalent DOF's
```

Output: An explicit, possibly non-manifold mesh

5.4 Reduction and Remapping

For the final step in hybrid lattice construction, we attempt to map as much of the explicit-connectivity mesh as possible back onto an implicit-connectivity grid to recover regularity. From the explicit mesh, we have two geometric primitives to consider for remapping: nodes and cells. For simplicity, we will first consider nodes and then cells. Figure 8 illustrates the results of the remapping rules below:

- Nodes are mapped to the grid if and only if they possess no duplicates.
- Cells are mapped to the grid if and only if all of their vertices have been mapped to the grid.

Each mesh cell remaining in the hybrid lattice is associated with a coordinate from the grid. This mapping will become important later when we discuss the block-based acceleration structures. After these rules are applied, the new structure must adhere to several post-conditions.

- All grid cells are composed only of grid nodes.
- Mesh cells contain one or more mesh nodes.

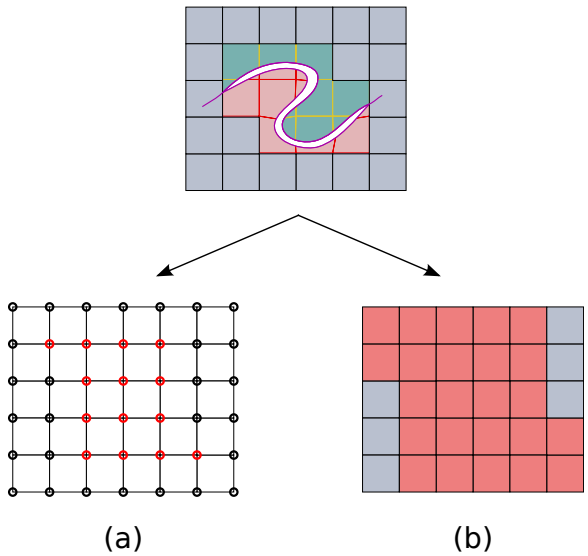

Figure 8: *From an explicit mesh (top), we generate (a) mesh mapped nodes in red, grid mapped nodes in black and (b) mesh mapped cells in red, grid mapped cells in gray.*

It should be noted that this set of rules is not strictly optimal in the sense of mapping the most elements into the grid. A more aggressive strategy would be to select one element from each set of geometrically co-located items and map it to the grid. We defer investigation of similar compaction heuristics to future work. **Note:** In all our surgical examples (Figures 1,2 and 4) mesh-mapped embedding cells are indicated with blue color, grid-mapped ones in red.Typically, only a minority of cells is mesh-mapped, allowing us to retain the bulk of performance benefits of implicit grid-mapped embeddings.

6 Software engineering and deployment

6.1 Parallelization

Our framework relies on both multithreading and vectorization (SIMD) to obtain the best possible performance. The fact that our Cartesian-based discretization consists of identically shaped elements offers a great opportunity to leverage both thread-level and data-level parallelism, due to the inherent regularity of the simulation kernels.

Blocking As described in Section 4 on discretization, forces are computed on a per cell basis. A naive multithreaded port would result in write hazards at nodal positions, unless expensive synchronization was used. Simple partitioning would eliminate this issue, but would not make efficient use of modern SIMD-enabled processors. Instead, we employ a blocking scheme to avoid write hazards while retaining a memory layout favorable for vectorization. Our objective is to redefine our "quantum" of computation from a single lattice cell, to a geometric neighborhood (or *block*) that is processed concurrently using vector operations. We adopt a block size of eight cells arranged as a $2 \times 2 \times 2$ cube. This formation allows us to fit blocks into eight-wide vectors and later we will demonstrate how we can adapt to larger and smaller vector widths. In this way, each cell in the block can be considered a "channel" in the vector. Blocks are tiled together to cover the extent of the lattice. However, restricting the contents of a non-manifold hybrid lattice to the spatial

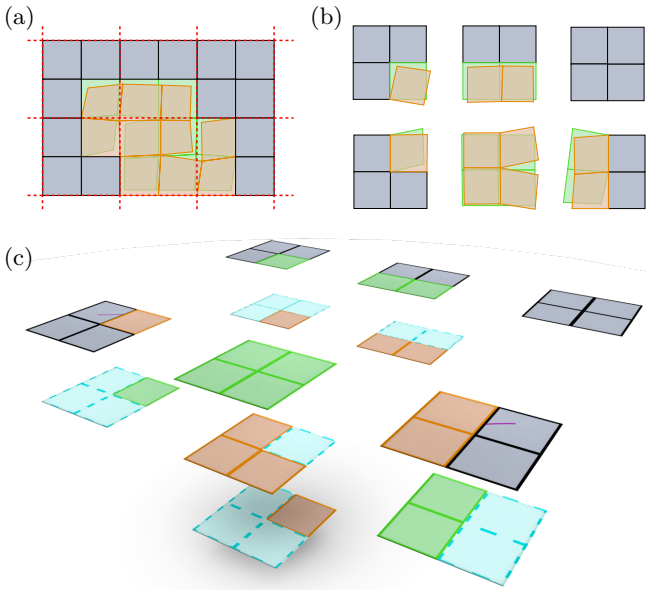

Figure 9: *Generating Blocks. (a) Block boundaries superimposed over hybrid lattice. (b) Non-manifold contents of each block region. (c) Final manifold blocks for each region.*

extent of a single block could easily yield more than one cell at each position in the block, as illustrated in Figure 9. To create blocks without overlapping cells, we employ a greedy algorithm which collects cells into manifold groupings along block boundaries, as seen in (Figure 9c). The full algorithm for this process is described in Algorithm 3.

We use the partitioning of our lattice into blocks to circumvent write dependencies during multithreaded execution. Prior to the execution of any kernel involving force computation, we copy the state variables from either the grid or mesh structures that natively store them, into duplicate copies for every block. We label this process a **Compaction** step, which is essentially a *gather* operation that yields a representation of the state variables into a flattened array of blocks (with shared variables duplicated across blocks). Subsequently, force computation can be executed in parallel on each block, without write dependencies, by allowing each block to record its own force contribution to the lattice nodes it touches. The reverse operation, labeled **Uncompaction** scatters and accumulates the contents of the per-block forces back to their native (non-duplicated) grid or mesh storage. Write dependencies can be avoided at this stage by partitioning this parallel operation on the grid or mesh variables that collect the per-block contributions. As a result, complex force computations can fully enjoy the benefits of thread- and SIMD-parallelism, without being concerned with data dependencies arising from the non-manifold mesh structure.

Guided Vectorization The high degree of regularity exposed by our blocking procedure naturally suggests using modern processor's SIMD capabilities to compute on all cells of a given block simultaneously. Although the performance potential is undeniable, porting code from a scalar implementation to a SIMD platform is a tedious task, one that auto-vectorization features of compilers have been traditionally ineffective in providing automatically. An example is the highly optimized SVD routines, published with the work of McAdams et al. [2011] which replicates almost instruction-by-instruction identical SIMD intrinsics to implement scalar, SSE and AVX versions; it can be easily verified that compiler auto-vectorization cannot provide competitive performance with these tediously hand-optimized kernels.

We have designed a programming paradigm called *guided vectorization*, with which we practically achieve the performance of hand-vectorized kernels, while only providing a single specification for scalar *and* vector variants. Our solution is object-oriented and based on the observation that the semantics of fundamental data types are very similar across scalar/vector platforms, even if the interface differs. Our system is rooted on two templatized C++ classes:

```
template<class scalar_arch> class Number;
template<class boolean_arch> class Mask;
```

Class `Number` is an abstraction of a single floating point number in a scalar platform (`scalar_arch==float`) or of a 4/8/16-wide vector register in SSE/AVX/Xeon Phi platforms (`scalar_arch:=__mm128|__mm256|__mm512`). Similarly, class `Mask` is an abstraction of the result of a comparison operation, in a form that can be used to perform a conditional assignment; thus `Mask<bool>` encapsulates a single C++ boolean variable, `Mask<__mm256>` captures a 256-bit mask usable in AVX `BLEND` instructions, while `Mask<__mmask16>` encapsulates the special concept in Intel Xeon Phi of a 16-bit *mask register* that is used in comparisons and conditional assignments. We provide enough overloaded operators in the interface of these classes to allow them to be used in algebraic expressions regardless of the encapsulating vector width. Ultimately we use them to construct macroscopic kernels of the form:

```
template<class scalar_arch,class T_DATA>
void Add_Force_Differential(
    const T_DATA (&dx)[3][8], ...
    const T_DATA (&V)[9],
    const T_DATA (&dPdF)[12],
    T_DATA (&df)[3][8]);
```

In this paradigm, we have separated the programmatic data width (type `T_DATA`, which could be `float`, for scalar code that computes forces on individual cells, or `float[8]`, for the force computation of all 8 cells of a block at once) from the architectural vector width. This allows us to design all of these kernels with the same semantics that would be followed for scalar execution, and automatically generate code that works on geometric blocks of any size, and vector architectures of different vector widths. For example the function call `Add_Force_Differential<__m128,float[16]>(....)` would use SSE instructions to compute force differentials of geometric blocks containing 16 cells each (e.g. blocks shaped like $4 \times 2 \times 2$ grid cells). A benchmark suite of *all* vectorized kernels in our solver is available from our project website.

Algorithm 3 Algorithm to Construct Blocks

Input: Block region i
1: **function** GENERATEBLOCKS
2: **for all** Cell c in i **do**
3: Create new empty block
4: Copy c into new block
5: Build connectivity graph between blocks
6: **repeat**
7: **for all** Symmetric connected block pairs **do**
8: Find pair with fewest neighbor mismatches
9: **if** Suitable pair found **then**
10: Collapse, merging block contents
11: **until** No further collapses occurred
12: **return** All remaining blocks
Output: A collection of one or more manifold Blocks

6.2 3-Tier Architecture

Our surgical platform is structured as a three-tier solution, with a web-based client (Tier 1), an SMP server for modeling (Tier 2), and a many-core accelerator for numerics (Tier 3). The *front-end client* serves primarily as the user interface to the system. The client is responsible for acquiring user input and visualizing the simulation results. Our initial client was written in C++ and continues to see use as a development platform. In order to support a wide variety of hardware at the client level, we have implemented a web browser client in HTML, Javascript, and WebGL. The web client is ideal for platform independence and does not require preinstallation. Communication is performed via the WebSockets standard, where a TCP connection is encapsulated by HTTP, which allows the client to operate under a push data model. Thus, the remote server sends updates to clients when they become available instead of requiring clients to poll for updates.

We refer to the *second-tier platform* as the CPU host. Its primary task is to perform all non-simulation computation that

can be offloaded from the client level. The host manages user sessions, stores and loads scene data from disk, performs geometric manipulations (non-manifold meshing and incision modeling) as a result of user actions, and runs collision detection. This tier requires large amounts of memory, beyond what is natively offered on a GPU or Many-Core accelerator. The *third tier* is the numerical solver, which executes all low-level compute intensive kernels (but not combinatorial tasks such as mesh generation). We have implementations that allow this layer to either run on the same CPU platform as the 2nd-tier code or run natively on a Xeon Phi accelerator, interfacing with the host over MPI.

7 Examples

For our clinical examples, we provided our senior domain collaborators with a brief tutorial of our authoring interface and asked them to craft typical repair scenarios on a prepared model of the human scalp, as well as on a stock model of a flat tissue sample. Our system captured a script of surgical manipulations, which was then re-run in an offline setting using a higher resolution embedding lattice, with the intent to re-enact the same manipulation with higher detail. As a proof of concept, we bench-marked our system with a high resolution human body model with anisotropic active musculature; details of the anisotropic material discretization can be found in the supplemental technical document. We report detail timings, including time taken at the individual kernels of our solver, in Table 1.

8 Evaluation and Discussion

Deployment To evaluate our system, we conducted a pilot deployment for the medical residents in the University of Wisconsin-Madison Plastic & Reconstructive Surgery program, as part of a workshop on craniofacial reconstruction techniques. We set up a local switched network at the location of the workshop, with a portable desktop-grade computer (with a 4-core Intel 4770R processor and 16GB RAM) acting as the Tier 2/3 simulation server, and a collection of eight Chromebox web-based thin clients as the user stations (our setup can be seen in Figure 3). Although our modest portable server did not have the computing power available to our many-core accelerated server systems which we used for our large-scale offline simulations, its (AVX-accelerated) performance was more than adequate to deliver an interactive user experience. Total cost of the entire deployment, including server, networking and client stations was $4,000.

Participants were initially given an extremely brief orientation on visual navigation within the application and its interface. In first part of the deployment exercise, the workshop instructor (a seasoned user of the system) authored several different reconstructive procedures on different anatomical models, while the participants were invited to follow the manipulations as they were taking place (without affecting them) by adjusting the view of the dynamic model on their own station. Subsequently, individual participants who had no prior exposure to the system were invited to drive the authoring process, which their colleagues would virtually follow. The workshop instructor provided guidance on clinical aspects of the repair being authored, while questions about the user interface would be recorded and addressed by the system developers. At the conclusion of the exercise, the participants were debriefed and given the opportunity to evaluate their experience and propose improvements.

Findings We found that our participants were extremely comfortable with aspects of 3D visual navigation, even with the very rudimentary orientation that was provided. Most participants found the visual examples of procedures demonstrated to be very enlightening, with many of them commenting that this visual illustration was the most informative exposure they had for procedures they only knew from reference literature (most of them had not witnessed these procedures in the operating room). Almost no participant volunteered the lack of self-collision processing as an observed omission, until the interviewer explicitly asked them about this aspect (all demonstrations in our workshop entailed full suturing of wound closure). On the contrary, several participants identified inaccuracies in the elastic behavior of the virtual tissues, finding that our models appeared to be more "permissive" to manipulation than real flesh tissue. An interface feature that was pointed out as lacking, is the inability to appreciate (simply by looking at the final sutured result) the deformation patterns that have resulted from a certain repair; it was suggested that adding a texture (grid lines or checkerboard patterns) on the skin surface would be much more useful in evaluating tissue strain and deformation. We also received requests for biologically inspired aids - in particular a visualization of anatomical elements such as blood vessels in the tissue being cut. Several users requested more traditional animation features, such as timeline scrubbing and history undo, as well as side-by-side views of different repair approaches for visual comparison.

Limitations and future work Section 3 discusses a number of conscious limitations in the interest of balancing performance and extensibility. From a research standpoint, our biggest perceived challenge is improving the accuracy of the constitutive models for the biomaterials involved. Full self-collision support will be essential for extending our work to procedures that rely on contact, in addition to sutures, to model properly. Finally, a large class of reconstructive procedures cannot be modeled with all incisions performed at the beginning of time; a flexible topology change modeling system would be necessary to incorporate a seamless ability for topology change, concurrent with deformation.

Acknowledgements

The authors are grateful to Dr. Timothy King, the medical residents of the Plastic & Reconstructive Surgery program, and the School of Medicine and Public Health at the University of Wisconsin-Madison for their support and participation in this activity. We would like to acknowledge Aaron Oliker and BioDigital Systems for supplying the models used in this project. Primary funding for this work was provided by the NSF/NIH Smart and Connected Health program (IIS-1407282). E.S. is supported in part by NSF Grants IIS-1253598, CNS-1218432, CCF-1423064, CCF-1438992.

References

Allard, J., Cotin, S., Faure, F., Bensoussan, P.-J., Poyer, F., Duriez, C., Delingette, H., Grisoni, L., et al. 2007. SOFA - an Open Source Framework for Medical Simulation. In *MMVR 15*, IOS Press.

Baker, S. R. 2014. *Local Flaps in Facial Reconstruction (3rd ed.)*. Saunders.

Bro-nielsen, M., and Cotin, S. 1996. Real-time Volumetric Deformable Models for Surgery Simulation using

Example	Plat.	Total Voxels	Grid Voxels	Mesh Voxels	Blocks	Newton Iteration (s)	Update State (ms)	Add Force (ms)	Add Differ. (ms)	Compact (ms)	UnCompact (ms)
Dufourmentel	Xeon	47689	47206	483	8821	0.3630	1.5	1.4	1.6	0.4	0.6
Mouly		294248	294015	233	47681	2.0648	7.8	5.1	4.2	2.1	3.0
	Phi	47689	47206	483	8821	0.4259	1.3	1.5	0.9	0.1	2.0
		294248	294015	233	47681	0.7349	2.8	3.3	2.2	0.7	4.7
Rhomboid Flap	Xeon	48529	47203	1326	6909	0.1620	1.1	0.9	0.7	0.2	0.4
		879571	877362	2209	112302	2.4131	18.4	9.6	9.0	3.2	4.0
	Phi	48529	47203	1326	6909	0.3735	1.3	1.4	1.0	0.1	1.5
		879571	877362	2209	112302	1.1323	5.3	6.2	3.9	1.2	9.6
ZPlasty	Xeon	54003	49306	3697	6943	0.2293	1.2	0.5	0.9	0.3	0.5
		960810	957908	2902	127070	2.7707	20.5	9.3	10.1	3.6	4.5
	Phi	54003	49306	3697	6943	0.3814	1.3	1.4	1.1	0.1	1.5
		960810	957908	2902	127070	1.2401	5.9	6.8	4.3	1.3	10.8
Human	Xeon	2006903	2006903	0	265078	12.112	35.9	21.5	20.8	8.4	12.0
	Phi	2006903	2006903	0	265078	3.9608	11.4	16.3	7.8	3.3	16.2

Table 1: *Performance results for various examples. Examples run on the Xeon platform used a 6-core Intel Xeon CPU E5-1650 machine with 64 GB of memory, while the Phi examples ran on a 60-core Xeon Phi 5110P card with 8GB of memory. All surgical examples were run with 50 conjugate gradient iterations while the human example was run with 100 iterations.*

Finite Elements and Condensation. In *Computer Graphics Forum*, 57–66.

Cavusoglu, M. C., Goktekin, T. G., and Tendick, F. 2006. GiPSi: A Framework for Open Source/Open Architecture Software Development for Organ-Level Surgical Simulation. *Information Technology in Biomedicine, IEEE Transactions on 10*, 2, 312–322.

Chen, D. T., and Zeltzer, D. 1992. Pump It Up: Computer Animation of a Biomechanically Based Model of Muscle Using the Finite Element Method. *SIGGRAPH Comput. Graph. 26*, 2 (July), 89–98.

Chentanez, N., Alterovitz, R., Ritchie, D., Cho, L., Hauser, K. K., Goldberg, K., Shewchuk, J. R., and O'Brien, J. F. 2009. Interactive Simulation of Surgical Needle Insertion and Steering. *ACM Trans. Graph. 28*, 3 (July), 88:1–88:10.

Courtecuisse, H., and Allard, J. 2009. Parallel Dense Gauss-Seidel Algorithm on Many-Core Processors. IEEE CS Press, HPCC '09, 139–147.

De, S., and Bathe, K. 2000. The method of finite spheres. *Computational Mechanics 25*, 329–345.

De, S., Kim, J., Lim, Y.-J., and Srinivasan, M. A. 2005. The point collocation-based method of finite spheres (PCMFS) for real time surgery simulation. *Computers & Structures 83*, 17 - 18, 1515 – 1525.

Dick, C., Georgii, J., and Westermann, R. 2011. A Hexahedral Multigrid Approach for Simulating Cuts in Deformable Objects. *IEEE Transactions on Visualization and Computer Graphics 17*, 11, 1663–1675.

Fan, Y., Litven, J., and Pai, D. K. 2014. Active Volumetric Musculoskeletal Systems. *ACM Trans. Graph. 33*, 4 (July), 152:1–152:9.

Gallagher, A. G., Ritter, E. M., Champion, H., Higgins, G., Fried, M. P., Moses, G., Smith, C. D., and Satava, R. M. 2005. Virtual Reality Simulation for the Operating Room: Proficiency-Based Training as a Paradigm Shift in Surgical Skills Training. *Annals of Surgery 241*, 2.

Goktekin, T. G., Bargteil, A. W., and O'Brien, J. F. 2004. A Method for Animating Viscoelastic Fluids. *ACM Trans. Graph. 23*, 3 (Aug.), 463–468.

Hermann, E., Raffin, B., and Faure, F. 2009. Interactive Physical Simulation on Multicore Architectures. Eurographics Association, EG PGV'09, 1–8.

Irving, G., Teran, J., and Fedkiw, R. 2004. Invertible Finite Elements for Robust Simulation of Large Deformation. Eurographics Association, SCA '04, 131–140.

Irving, G., Schroeder, C., and Fedkiw, R. 2007. Volume Conserving Finite Element Simulations of Deformable Models. *ACM Transactions on Graphics (SIGGRAPH Proc.) 26*, 3.

James, D. L., and Pai, D. K. 1999. ArtDefo: Accurate Real Time Deformable Objects. ACM Press/Addison-Wesley Publishing Co., SIGGRAPH '99, 65–72.

Jerabkova, L., Bousquet, G., Barbier, S., Faure, F., and Allard, J. 2010. Volumetric modeling and interactive cutting of deformable bodies. *Progress in Biophysics and Molecular Biology 103*, 2-3 (Dec.), 217–224. Special Issue on Biomechanical Modelling of Soft Tissue Motion.

Jeřábková, L., and Kuhlen, T. 2009. Stable Cutting of Deformable Objects in Virtual Environments Using XFEM. *IEEE Comput. Graph. Appl. 29*, 2, 61–71.

Joshi, P., Meyer, M., DeRose, T., Green, B., and Sanocki, T. 2007. Harmonic Coordinates for Character Articulation. *ACM Trans. Graph. 26*, 3 (July).

Kaufmann, P., Martin, S., Botsch, M., and Gross, M. 2009. Flexible Simulation of Deformable Models Using Discontinuous Galerkin FEM. *Graph. Models 71*, 4 (July), 153–167.

Kavan, L., Collins, S., Žára, J., and O'Sullivan, C. 2008. Geometric Skinning with Approximate Dual Quaternion Blending. *ACM Trans. Graph. 27*, 4 (Nov.), 105:1–105:23.

Kharevych, L., Mullen, P., Owhadi, H., and Desbrun, M. 2009. Numerical Coarsening of Inhomogeneous Elastic Materials. *ACM Trans. Graph. 28*, 3 (July), 51:1–51:8.

Kim, J., and Pollard, N. S. 2011. Fast Simulation of Skeleton-driven Deformable Body Characters. *ACM Trans. Graph. 30*, 5 (Oct.), 121:1–121:19.

Kim, J., Choi, C., De, S., and Srinivasan, M. A. 2007. Virtual surgery simulation for medical training using multi-resolution organ models. *The International Journal of Medical Robotics and Computer Assisted Surgery 3*, 2, 149–158.

Li, D., Sueda, S., Neog, D. R., and Pai, D. K. 2013. Thin Skin Elastodynamics. *ACM Trans. Graph. 32*, 4 (July), 49:1–49:10.

Lindblad, A., and Turkiyyah, G. 2007. A Physically-based Framework for Real-time Haptic Cutting and Interaction with 3D Continuum Models. ACM, SPM '07, 421–429.

Marchal, M., Allard, J., Duriez, C., and Cotin, S. 2008. Towards a Framework for Assessing Deformable Models in Medical Simulation. In *ISBMS '08*, P. J. E. Fernando Bello, Ed., vol. 5104 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 176–184.

McAdams, A., Sifakis, E., and Teran, J. 2010. A Parallel Multigrid Poisson Solver for Fluids Simulation on Large Grids. Eurographics Association, SCA '10, 65–74.

McAdams, A., Zhu, Y., Selle, A., Empey, M., Tamstorf, R., Teran, J., and Sifakis, E. 2011. Efficient Elasticity for Character Skinning with Contact and Collisions. *ACM Trans. Graph. 30*, 4 (July), 37:1–37:12.

Mendoza, C., and Laugier, C. 2003. Simulating Soft Tissue Cutting using Finite Element Models. vol. 1 of *ICRA '03*, IEEE, 1109–1114.

Molino, N., Bao, Z., and Fedkiw, R. 2004. A Virtual Node Algorithm for Changing Mesh Topology During Simulation. *ACM Trans. Graph. 23*, 3 (Aug.), 385–392.

Müller, M., Teschner, M., and Gross, M. 2004. Physically-Based simulation of Objects Represented by Surface Meshes. CGI '04, 156–165.

Nesme, M., Payan, Y., and Faure, F. 2006. Animating Shapes at Arbitrary Resolution with Non-Uniform Stiffness. EG VRIPHYS '06, Eurographics.

Nesme, M., Kry, P. G., Jeřábková, L., and Faure, F. 2009. Preserving Topology and Elasticity for Embedded Deformable Models. *ACM Trans. Graph. 28*, 3 (July), 52:1–52:9.

Nienhuys, H.-W., and van der Stappen, A. F. 2001. A Surgery Simulation Supporting Cuts and Finite Element Deformation. MICCAI '01, Springer, 145–152.

O'Brien, J., and Hodgins, J. 1999. Graphical Modeling and Animation of Brittle Fracture. In *Proc. of SIGGRAPH 1999*, 137–146.

Patterson, T., Mitchell, N., and Sifakis, E. 2012. Simulation of Complex Nonlinear Elastic Bodies Using Lattice Deformers. *ACM Trans. Graph. 31*, 6 (Nov.), 197:1–197:10.

Pieper, S. D., Laub Jr, D. R., and Rosen, J. M. 1995. A Finite-Element Facial Model for Simulating Plastic Surgery. *Plastic and Reconstructive Surgery 96*, 5, 1100–1105.

Rivers, A., and James, D. 2007. FastLSM: Fast lattice shape matching for robust real-time deformation. *ACM Transactions on Graphics (SIGGRAPH Proc.) 26*, 3.

Sifakis, E., and Barbic, J. 2012. FEM Simulation of 3D Deformable Solids: A Practitioner's Guide to Theory, Discretization and Model Reduction. In *ACM SIG. 2012 Courses*, ACM, SIGGRAPH '12, 20:1–20:50.

Sifakis, E., Der, K. G., and Fedkiw, R. 2007. Arbitrary Cutting of Deformable Tetrahedralized Objects. Eurographics Association, SCA '07, 73–80.

Simbionix USA Corporation, 2002–2014. Gastrointestinal Simulator - GI Mentor Simbionix. `http://simbionix.com/simulators/gi-bronch-gi-mentor`.

Simbionix USA Corporation, 2002–2014. Laparoscopic Simulator - LAP Mentor Simbionix. `http://simbionix.com/simulators/lap-mentor`.

Sin, F., Schroeder, D., and Barbic, J. 2013. Vega: Non-Linear FEM Deformable Object Simulator. *Comput. Graph. Forum 32*, 1, 36–48.

Sueda, S., Kaufman, A., and Pai, D. K. 2008. Musculotendon Simulation for Hand Animation. *ACM Trans. Graph. 27*, 3 (Aug.), 83:1–83:8.

Teran, J., Blemker, S., Hing, V. N. T., and Fedkiw, R. 2003. Finite Volume Methods for the Simulation of Skeletal Muscle. SCA '03, 68–74.

Teran, J., Sifakis, E., Irving, G., and Fedkiw, R. 2005. Robust Quasistatic Finite Elements and Flesh Simulation. *Proc. of the 2005 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, 181–190.

Teran, J., Sifakis, E., Blemker, S. S., Ng-Thow-Hing, V., Lau, C., and Fedkiw, R. 2005. Creating and Simulating Skeletal Muscle from the Visible Human Data Set. *Visualization and Computer Graphics, IEEE Transactions on 11*, 3, 317–328.

Terzopoulos, D., and Fleischer, K. 1988. Modeling Inelastic Deformation: Viscolelasticity, Plasticity, Fracture. *SIGGRAPH Comput. Graph. 22*, 4 (June), 269–278.

Terzopoulos, D., Platt, J., Barr, A., and Fleischer, K. 1987. Elastically Deformable Models. *SIGGRAPH Comput. Graph. 21*, 4 (Aug.), 205–214.

Wang, X. C., and Phillips, C. 2002. Multi-weight Enveloping: Least-squares Approximation Techniques for Skin Animation. ACM, SCA '02, 129–138.

Wojtan, C., and Turk, G. 2008. Fast Viscoelastic Behavior with Thin Features. *ACM Trans. Graph. 27*, 3 (Aug.), 47:1–47:8.

Zhao, Y., and Barbič, J. 2013. Interactive Authoring of Simulation-Ready Plants. *ACM Trans. on Graphics (SIGGRAPH 2013) 32*, 4, 84:1–84:12.