Automatic Preparation, Calibration, and Simulation of Deformable Objects

Dan Morris
Stanford University Robotics Lab
Computer Science Department
Stanford, CA 94305-9010
dmorris@cs.stanford.edu

ABSTRACT

Many simulation environments – particularly those intended for medical simulation – require solid objects to deform at interactive rates, with deformation properties that correspond to real materials. Furthermore, new objects may be created frequently (for example, each time a new patient's data is processed), prohibiting manual intervention in the model preparation process. This paper provides a pipeline for rapid preparation of deformable objects with no manual intervention, specifically focusing on mesh generation (preparing solid meshes from surface models), automated calibration of models to finite element reference analyses (including a novel approach to reducing the complexity of calibrating nonhomogeneous objects), and automated skinning of meshes for interactive simulation.

Categories and Subject Descriptors

I.6.0 [Simulation and Modeling]: General; I.6.1 [Simulation and Modeling]: Model Validation and Analysis

General Terms

Algorithms

Keywords

Interactive simulation, real-time deformation, deformable models, finite element modeling, mesh skinning, model calibration, simulated annealing, mesh generation, medical simulation, soft tissue simulation

1. INTRODUCTION AND RELATED WORK

1.1 Background

Interactive physical simulation has become a critical aspect of many virtual environments. Computer games are increasingly using physical simulation to allow players a wider range of interactions with their surroundings; this has become such a prevalent phenomenon that dedicated hardware has become available for rigid body mechanics [29], and software libraries are becoming available to dedicate graphics resources to physical simulation [30]. Simulation for games currently focuses primarily on rigid body dynamics and particle systems (for fluid, smoke, explosions, etc.), but will likely move toward deformable solid simulation as the standard for realism increases.

Many medical simulation environments – both commercial ([31],[32],[33],[34],[35],[36]) and academic [37],[38],[39],[40],[41]) – already depend on modeling deformable solids. The vast majority of tasks performed during surgery involve interaction with deformable bodies, so a medical simulator is expected to not only represent deformation, but to model it with sufficient accuracy for effective training. Force/deformation curves of virtual organs should correspond to their real counterparts, and deformation should vary realistically among patients, among tissue types, and even within a tissue type.

Currently many of these simulators focus on canonical cases, whose creation requires significant manual intervention by developers, technicians, or manufacturers. As surgical simulation enters mainstream medical practice, the use of patient-specific data in place of canonical cases is likely to become common, allowing a much broader range of applications and training cases. This scenario prohibits the use of tedious manual procedures for data preprocessing. Similarly, as games incorporate more sophisticated simulation techniques, rapid preparation of deformable models will be required to continue the current trend toward player-generated and custom content.

This paper addresses this need: automatic preparation of realistic deformable models for medical simulation and computer games. We restrict our discussion to a particular simulation method in the interest of focusing on *automation* of model preparation (rather than simulation), but the techniques presented here can be generalized to other models.

We assume that the user provides a surface model of the desired structure; this is a reasonable assumption, as surface models are the standard object representation in games and are easily derived from automatically-segmented medical images. We further assume that the user provides constitutive properties describing the material they are attempting to represent; this is also a reasonable assumption, as constitutive properties for a wide variety of materials are available in engineering handbooks. Constitutive properties for biological tissues can be measured experimentally ([42],[43],[44]).

Section 2 discusses the generation of volumetric (tetrahedral) meshes from surface meshes. Section 3 discusses the use of a finite element reference model to calibrate an interactive simulation. Section 4 discusses simulation and rendering, focusing on a geometric interpretation of the simulation technique presented in [16] and a mesh skinning technique that is suitable for our deformation model. The remainder of Section 1 discusses work related to each of these three topics.

1.2 Related Work: Mesh generation

"Mesh generation" generally refers to the process of discretizing a space into volumetric elements. The space is frequently defined by either an implicit or explicit surface boundary, and the elements are generally explicit solid units, commonly tetrahedra or hexahedra when the space is three-dimensional.

Ho-Le [5] provides a summary of core methods in mesh generation for finite element analysis, and Zhang [4] provides a summary of more recent work in this area. Si [2] describes a common, public-domain package for mesh generation, specifically targeted toward finite element analysis applications. Recent work on mesh generation employs physical simulation in the meshing process (e.g. [3]).

The work most closely related to the approach presented in Section 3 of this paper is that of Mueller [1], which also focuses on generating approximate, non-conformal meshes for interactive simulation.

1.3 Related Work: Deformation Calibration

Early work exploring the relationship between non-constitutive simulations (generally mass-spring systems) and finite element analyses began with Deussen et al [10], who optimize a 2D mass-spring system to behave like an analytically-deformed single 2D constitutive element. Similarly, van Gelder [13] analytically derives spring constants from constitutive properties for a 2D mass-spring system. This work also includes a theoretical proof that a mass-spring system cannot exactly represent the deformation properties of a constitutive finite element model.

While most work in this area has been oriented toward volumetric solid deformation using simulation results as a ground truth, Bhat et al [6] use video of moving cloth to calibrate simulation parameters for a cloth simulation. Similarly, Etzmuss et al [9] extend the theoretical approach of van Gelder [13] to derive a mass-spring system from a constitutive model of cloth.

Bianchi et al [7] demonstrate that a calibration procedure can enable a 2D mass-spring system to recover the connectivity of another 2D mass-spring system; deformation constants are held constant. Bianchi et al [8] later demonstrate the recovery of spring constants, and the 2D calibration of a mass-spring system to a finite element reference model. They do not extend their calibration to 3D, and do not provide a mechanism for handling the exponential growth in optimization complexity associated with 3D objects and complex topologies. Choi et al [11] use a similar approach to calibrate a homogeneous mass-spring system, and Mosegaard [12] uses a similar optimization for simple models but takes dynamic behavior into account during optimization.

1.4 Related Work: Mesh Skinning

Mesh skinning describes the process of animating the vertices of a rendered mesh to correspond to the behavior of an underlying skeleton. This has become a very common technique for rendering characters in games and video animation; the skeleton often literally represents a character's skeleton and the rendered mesh generally represents the character's skin. Skinning is easily implemented in graphics hardware [45], making it suitable for a variety of simulation environments.

Recent work on mesh skinning has focused on correcting the inaccuracies that result from naïve blending, as per [15], and on automatically associating vertex movements with an implicit underlying skeleton [14] as a form of animation compression. However, bones are generally defined and associated with vertices manually by content developers, as part of the modeling/animation process.

2. MESH GENERATION

This section discusses our approach to generating tetrahedral meshes from surface meshes for interactive deformation.

2.1 Background

Previous approaches to generating tetrahedral meshes (e.g. [1],[2],[3],[4]) from surface meshes have generally focused on generating conformal meshes (meshes whose bounding surface matches the target surface precisely) for high-precision finite element simulation. Consequently, the resulting meshes are generally highly complex, particularly near complex surface regions.

Interactive simulation presents a different set of requirements and priorities for mesh generation. Since the use of interactive simulation techniques comes with an intrinsic loss in precision, some discrepancy between the target surface mesh and the resulting volumetric mesh is generally acceptable. In particular, the computational expense of increased tetrahedron count does not justify the benefits of a conformal mesh. This is particularly true for applications in games, where physical plausibility and interactivity take precedence over perfect accuracy. For most applications, the surface used for interactive rendering is decoupled from the simulation mesh, so the nonconformality of the mesh will not affect the rendered results (see Section 4).

Like finite element simulation, most interactive simulation techniques have difficulties when tetrahedral aspect ratios approach zero. In other words, "sliver" tets are generally undesirable, since they are easily inverted and do not have well-defined axes for volume restoration forces.

The behavior of interactive simulation techniques is often visibly affected by topology, so a homogeneous material is generally most effectively simulated by a mesh with homogeneous topological properties. Thus there is an intrinsic advantage to *regularity* in deformable meshes.

Thus the goal of the technique presented here is to automatically generate nonconformal, regular meshes with large tetrahedral aspect ratios.

It is also desirable for the process to proceed at nearly interactive rates for meshes of typical complexity, so the process can easily be repeated following topology changes or plastic deformation during interactive simulation.

2.2 Mesh Generation

Our mesh generation procedure begins with a surface mesh (Figure 1a), for which we build an axis-aligned bounding box (AABB) hierarchy (Figure 1b).

The AABB tree is used to rapidly floodfill (voxelize) the surface (Figure 1c). The floodfilling begins with a seed voxel, identified by stepping a short distance along the inward-pointing surface normal of a mesh triangle. This voxel is considered to be an internal voxel. Floodfilling sequentially pulls internal voxels from a queue. A ray is cast from each known internal voxel to each of its neighbors; the AABB hierarchy is used to determine whether this ray crosses the object boundary, with spatial coherence exploited as per [26]. If the ray does not cross the surface, the neighbor is marked as an internal voxel and is placed on the queue. If the ray does cross the surface, the neighbor is

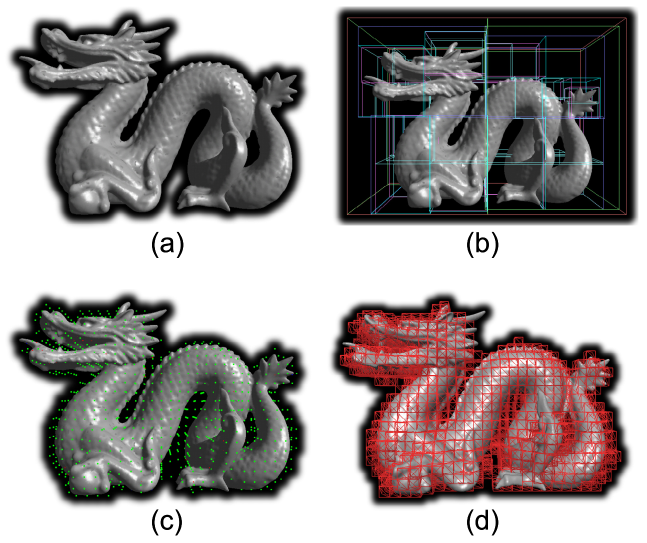


Figure 1. Stages of the mesh generation process: (a) initial surface mesh, (b) axis-aligned bounding box hierarchy for rapid voxelization (c, with voxel centers in green), and (d) splitting of voxels into tetrahedra.

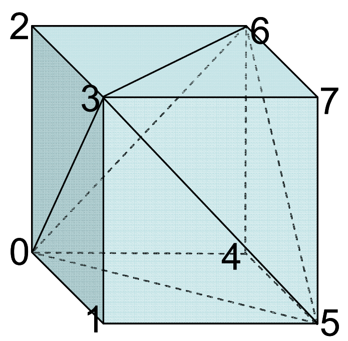
marked as a border voxel and is not considered for further processing. Floodfilling proceeds until the queue is empty.

The resolution of voxelization – which determines the resolution of the output tet mesh – is user-specified. Since voxels are isotropic, the user need only specify the voxel resolution of the mesh's longest axis, a simple precision metric that a user can intuitively relate to the target application. Voxelization is allowed to proceed one voxel outside the surface; for interactive simulation techniques that include collision-detection and penalty-based collision response, it is generally desirable to slightly overestimate object volume at this stage.

Each resulting voxel (defined by its center point) is then used to create a cube of eight vertices. Vertices are stored by position in a hash table; existing vertices can thus be recalled (rather than re-created) when creating a voxel cube, allowing shared vertices in the output mesh. Each resulting cube is then divided into five tetrahedra (Figure 2), yielding the final tetrahedral mesh (Figure 1d).

2.3 Implementation and Results

The mesh generation approach presented here was incorporated into the *voxelizer* package, available online and discussed in more detail in [26]. The package is written in C++ and uses CHAI [27] for visualization and collision detection (AABB tree construction). Files are output in a format compatible with TetGen [2].



Tet	Nodes
0	0, 1, 3, 5
1	3, 5, 6, 7
2	0, 2, 3, 6
3	0, 4, 5, 6
4	0, 3, 5, 6

Figure 2. Splitting a cube (voxel) into five tetrahedra.

To evaluate the computational cost of our approach, and thus its suitability for real-time re-meshing, we generated tetrahedral meshes for a variety of meshes (Figure 3) at a variety of resolutions on a 1.5GHz Pentium 4. Resolutions were specified as "long axis resolution", i.e. the number of tetrahedra along the output mesh's longest axis (Section 2.2).

Table 1 summarizes these results. Mesh generation time is almost precisely linear in output tet count (Figure 5), and mesh generation time is below one second for meshes up to approximately 250,000 tets. Mesh generation proceeds at graphically interactive rates (>10Hz) for meshes up to approximately 20,000 tets. Current parallel simulation techniques ([46],[47]) allow simulation of over 100,000 tets interactively; mesh generation for meshes at this scale is not real-time (about 500ms), but would be sufficiently fast – even at these extremely high resolutions – to allow nearly-interactive background remeshing in cases of topology changes and large deformations.

Figure 4 shows mesh generation times as a function of the user-specified precision variable: long axis mesh resolution.

A binary version of our mesh generation approach is publicly available at:

http://cs.stanford.edu/~dmorris/voxelizer

3. Calibration to Ground Truth Deformation

This section discusses the automated calibration of non-constitutive deformation properties using known constitutive properties and a finite-element-based reference deformation.

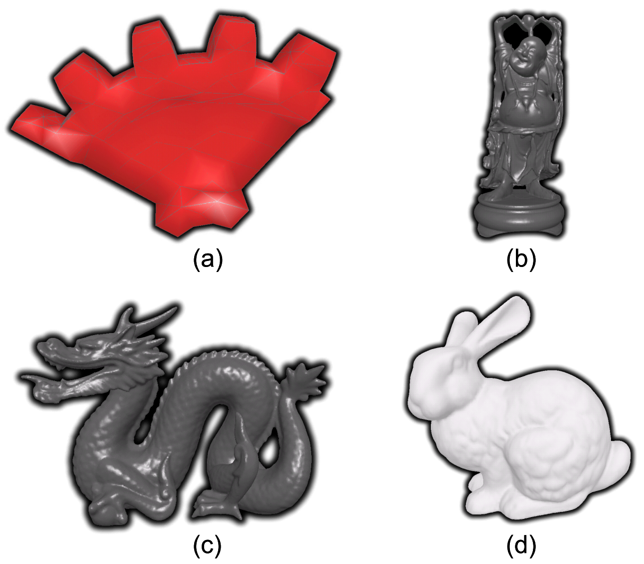


Figure 3. Meshes used for evaluating mesh generation. (a) Gear: 1000 triangles. (b) Happy: 16000 triangles. (c) Dragon: 203,000 triangles (d) Bunny: 70,000 triangles.

Input mesh	Input mesh size (triangles)	Long axis resolution (tets)	Output mesh size (tets)	Tetrahedralization time (s)
bunny	70k	30	35840	0.153
bunny	70k	75	478140	1.98
bunny	70k	135	2645120	10.139
bunny	70k	165	4769080	18.287
gear	1k	30	20780	0.101
gear	1k	75	271350	1.132
gear	1k	135	1434065	5.789
gear	1k	165	2504240	9.961
happy	16k	30	10100	0.057
happy	16k	75	126610	0.562
happy	16k	135	662745	2.7
happy	16k	165	1178725	4.74
dragon	203k	30	12750	0.083
dragon	203k	75	158370	0.772
dragon	203k	135	820305	3.57
dragon	203k	165	1453270	6.042

Table 1. Tetrahedralization time for the meshes shown in Figure 3, at various output mesh resolutions.

3.1 Background

Techniques for simulating deformable materials can be classified coarsely into two categories: constitutive and non-constitutive models.

Approaches based on constitutive models (e.g. [19],[20],[21],[22],[23],[24],[25]) generally use equations from physics to describe how a material will behave in terms of physical constants that describe real materials – e.g. Poisson's coefficient, Young's modulus, etc. These constants can generally be looked up in an engineering handbook or determined experimentally for a particular material. Many methods in this category are variants on finite element analysis (e.g. [19],[20],[21]), which uses known constitutive relationships between force and deformation to predict how a material will deform. These methods are traditionally very accurate, and are used for computing stresses and strains for critical applications in structural mechanics, civil engineering, automotive engineering, etc. However, these methods are generally associated with significant computational overhead, often requiring solutions to large linear systems, and thus cannot generally be run at interactive rates. When these approaches are adapted to run at interactive rates, they are generally limited in the mesh resolutions they can process in real-time.

In contrast, many approaches to material deformation are non-constitutive, e.g. [16],[17],[18],[46],[47]. Rather than using physical constants (e.g. elastic moduli) to describe a material, such approaches describe objects in terms of constants that are particular to the simulation technique employed. Many approaches in this category are variants on the network of masses and springs, whose behavior is governed by spring constants that can't be directly determined for real materials. In general, these methods are thus not accurate in an absolute sense. However,

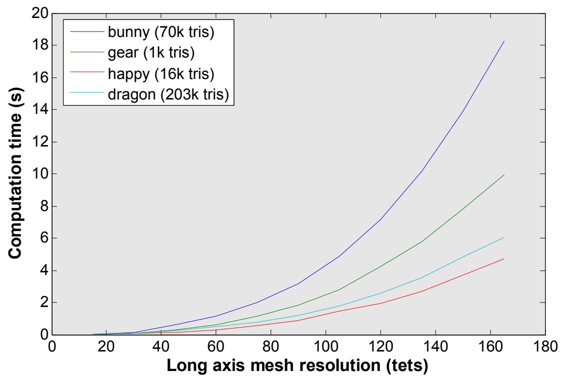


Figure 4. Mesh generation times at various output mesh resolutions. Long axis resolution, rather than output tet count, is used as the dependent variable; this is an intuitive metric for user-specified mesh precision.

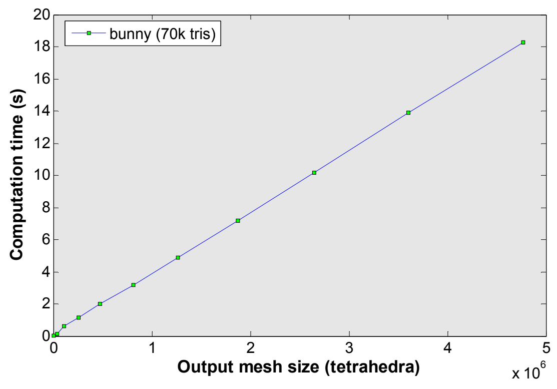


Figure 5. Mesh generation times at various output mesh resolutions.

many approaches in this category can be simulated at interactive rates even for high-resolution data, and these approaches often parallelize extremely well, offering further potential speedup as parallel hardware becomes increasingly common.

In short, the decision to use one approach or the other for a particular application is a tradeoff between realism and performance, and interactive simulations are often constrained to use non-constitutive techniques.

For applications in entertainment or visualization, simulation based on hand-calibrated constants may be adequate. But for high-precision applications, particularly applications in virtual surgery, a deformation model is expected to behave like a specific real material. It is often critical, for example, to teach absolute levels of force that are necessary to achieve certain deformations, and it is often critical to differentiate among tissue types based on compliance. Thus roughly-calibrated material properties are insufficient for medical applications.

Furthermore, traditional mass-spring systems are usually expressed in terms of stiffnesses for each spring, so the only way to vary the behavior of a material is to vary those stiffnesses. For any significant model, this translates into many more free parameters than a content developer could reasonably calibrate by hand.

Even if sufficient manual labor is available to manually calibrate canonical models, this calibration would generally be object-specific, as much of the deformation properties of a mass-spring network are embedded in the topology and geometry of the network [48]. Therefore calibrated spring constants cannot be directly transferred among objects, even objects that are intended to represent the same material.

The present work aims to run this calibration automatically, using the result of a finite element analysis as a ground truth. While calibration results still cannot be generalized across objects, the calibration runs with no manual intervention and can thus be rapidly repeated for arbitrary sets of objects.

3.2 Homogeneous Calibration

The following are assumed as inputs for the calibration process:

- A known geometry for the object to be deformed, generated according to the procedure outlined in Section 2.
- A known set of loads – defined as constant forces applied at one or more mesh vertices – that are representative of the deformations that will be applied to the object interactively. In practice, these loads are acquired using a haptic simulation environment and an uncalibrated object. Note that a single "load" may refer to multiple forces applied to multiple (potentially disjoint) regions of the mesh.
- Constitutive elastic properties (Poisson's coefficient and Young's modulus) for the material that is to be represented.

The supplied constitutive properties are used to model the application of the specified loads using an implicit finite element analysis, providing a ground truth deformation to which non-constitutive results can be compared. This quasi-static analysis neglects dynamic effects; extension to dynamics is an area for future work.

The same loads are then applied to the same geometry using a non-constitutive simulation, and the simulation is allowed to come to steady-state (a configuration in which elastic forces precisely negate the applied forces). For the implementation presented in Section 3.3 we use the deformation model presented in [16], but for this discussion we will treat the simulation technique as a black box with a set of adjustable parameters.

There are, in most cases, large subsets of the parameter space that will not yield stable deformations. In traditional mass-spring systems, for example, inappropriately high constants result in instability and oscillation, while inappropriately low constants result in structural collapse. In either case, local variation in parameters cannot be reliably related to variation in deformation. Optimization will proceed most rapidly if it begins with a baseline deformation that can be used to quickly discard such regions in the parameter space. Therefore, before beginning our optimization, we coarsely sample the parameter space for a fixed number of simulations (generally 100) and begin our optimization with the optimal set among these samples, as per [6] (our optimality metric follows). If none of our samples yield a stable deformation, we randomly sample the space until a stable deformation is obtained.

We then compute an error metric describing the accuracy of this parameter set as the surface distance between the meshes resulting from constitutive and non-constitutive deformation:

$$e_L(\varphi) = \sqrt{\frac{\sum_{i=1}^{nvertices} \left| p_{const}(i) - p_{nonconst}(i) \right|^2}{nvertices}}$$

…where $e_L(\varphi)$ is the error (inaccuracy) for a parameter set φ and load L, *nvertices* is the number of vertices in our mesh, $p_{const}(i)$ is the position of vertex *i* following constitutive deformation, and $p_{nonconst}(i)$ is the position of vertex *i* following non-constitutive deformation with parameter set φ. Note that the non-constitutive deformation is computed once at the beginning of the optimization procedure and is not repeated.

This error metric assumes a one-to-one correspondence between vertices in the two meshes; in practice this is the case for the implementation presented in Section 3.3, but were this not the case, the lower-resolution mesh could be resampled at the locations of the higher-resolution mesh's vertices. The deformed positions of the resampled vertices could then be obtained by interpolating the deformed positions of the neighboring vertices in the lower-resolution mesh after deformation (this is analogous to interpolating displacements by free-form deformation [49]).

When multiple loads (to be applied separately) have been defined, we average the resulting errors over those loads to define an accuracy metric for a parameter set:

$$E(\varphi) = \sum_{L=1}^{nloads} e_L(\varphi)$$

…where E(φ) is the average error for the parameter set φ and *nloads* is the number of separate loads to apply. In practice *nloads* is often 1, but we will continue to use the more general E(φ) notation that allows for multiple loads.

The goal of our optimization is thus to find the parameter set φ that minimizes E(φ):

$$\Phi = \arg\min_{\varphi} E(\varphi)$$

…where Φ is our output parameter set, representing the best match to the supplied constitutive parameters for the specified deformations, and φ is bounded by user-specified upper- and lower-bounds, which generally do not vary from problem to problem.

We solve this constrained minimization problem through simulated annealing [50] (SA), a stochastic optimization technique that follows local gradients in a problem space to arrive at minima of the energy function, but periodically jumps against the gradient to avoid local minima. In particular, we use the adaptive simulated annealing [28] (ASA) variant on SA, which automatically adjusts the annealing parameters over time to converge more quickly than traditional SA.

For very simple linear problems, such as identifying the optimal spring constant for a single tetrahedron being stretched in a single direction, we have also employed gradient descent, which is extremely efficient, but complex error landscapes prevent this approach for significant problems. We will discuss further applications for simpler approaches in Section 5.

At the completion of the simulated annealing procedure, we will have a non-constitutive parameter set Φ that optimally matches our non-constitutive deformation to our constitutive deformation. The annealing procedure may take a significant amount of time to complete, but it proceeds with no manual intervention and can thus be efficiently used to prepare numerous deformable models.

3.3 Implementation

We have implemented the described calibration process using an implicit solver for our constitutive deformation and the method of [16] for our non-constitutive deformation. The finite element package Abaqus [51] is used for reference deformations, and our interactive deformation model is implemented in C++ using CHAI [27] for visualization. Deformation results from both packages are collected in Matlab [52], and optimization is performed with the ASA package [54] through the ASAmin wrapper [53]. Gradients are estimated by finite differencing.

The selected deformation model is described in more detail in Section 4; the key point for this discussion is that nodal forces are computed based on four deformation parameters: a volume preservation constant k_v (defined for each tetrahedron), an area preservation constant k_a (defined for each face), a length preservation constant k_d (defined for each edge), and a viscous damping force k_{damp}. These four values are the free parameters for our optimization. For the results presented in Section 3.4, they are taken to be homogeneous throughout the material. Nonhomogeneity will be introduced in Section 3.5. In practice, the viscous damping force is always uniform and is allowed to vary only coarsely; once it is calibrated to a reasonable value for a problem, variations should affect the time required to reach steady-state but not the final deformation.

Since we use a quasi-static, implicit simulation for constitutive deformation, we require steady-state results from our non-constitutive simulation as well. A simulation is determined to be at steady-state when the maximum and mean vertex velocities and accelerations are below threshold values for a predetermined amount of time. These values are defined manually but do not vary from problem to problem. Simulations that do not reach steady-state within a specified interval are assigned an error of DBL_MAX.

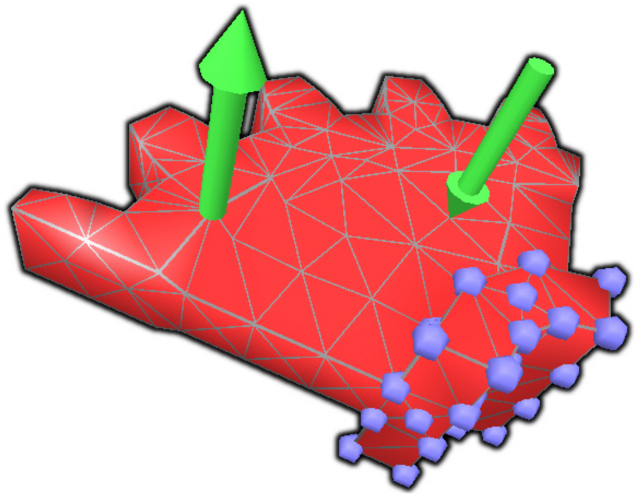


Figure 6. The deformation problem analyzed in section 3.4. Nodes highlighted in blue are fixed in place; green arrows define the applied load.

3.4 Results: Homogeneous Calibration

We will demonstrate the effectiveness of this approach through a case study, using the problem depicted in Figure 6. Here the base of the gear model is fixed in place (nodes indicated in blue), and opposing forces are applied to the "front" of the gear. This load will tend to "twist" the gear around its vertical axis. The simulated object is defined to be approximately 2 meters wide, with 50 pounds of force applied at each of the two load application points. The constitutive simulation uses a Young's modulus of 100kPa and a Poisson's coefficient of 0.45 .

Figure 7 graphically displays the results of the calibration procedure for this problem. The undeformed mesh is shown in

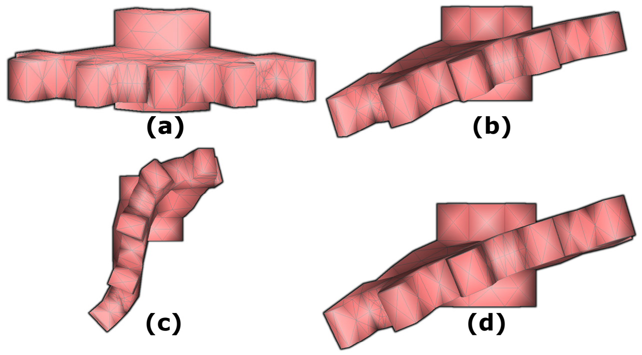


Figure 7. Results after calibration for the problem shown in Figure 6. Each subfigure shows a "top view" of the model introduced in Figure 6. (a) Undeformed model. (b) Ground truth deformation (resulting from finite element analysis). (c) Baseline non-constitutive deformation (hand-selected constants). (d) Calibrated non-constitutive deformation. (b) and (d) are nearly identical, indicating successful calibration.

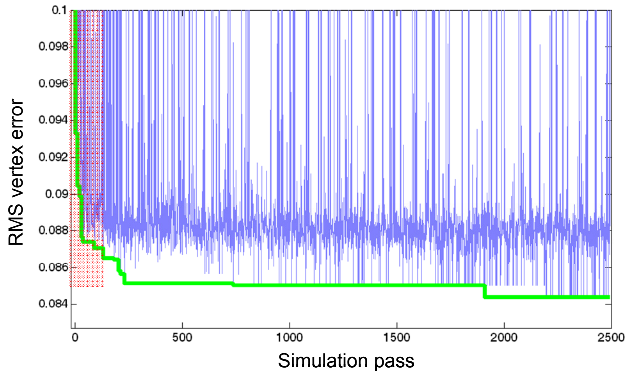


Figure 8. Optimization trajectory for the calibration shown in Figure 7. Each error value is shown in blue; the green line represents the lower envelope of the blue line, or in other words the best result found so far at any point in the optimization process. The region highlighted in red indicates the rapid initial gradient descent. The y-axis is compressed to improve visibility; the initial error is 0.9, and the maximum error (assigned to non-terminating simulations) is DBL_MAX.

Figure 7a. For comparison, the result of a non-constitutive deformation using constants selected through several minutes of manual calibration is presented in Figure 7c. Note that this is not an unreasonable or inconsistent response to the applied loads. Figures 7b and 7d show the results of constitutive deformation and calibrated non-constitutive deformation, respectively. The two models are nearly identical, indicating a successful calibration. Using the error metric described in section 3.2, the error was reduced from 0.9 (uncalibrated) to 0.08 (calibrated).

Figure 8 looks more closely at the optimization trajectory during this calibration. The optimization proceeds from left to right, with each point representing a simulation pass. Higher values on the y-axis indicate less accurate deformation. The highlighted area indicates the optimization's efficient use of the error gradient for rapid descent from the initial error result. This indicates that a bounded optimization, for which the user specified an acceptable error bound, rather than waiting for a global optimum, would proceed extremely rapidly. This is likely to be the most practical usage model for this approach.

The "jittery" appearance of the error plot, with numerous simulations resulting in very large errors, results from the annealing process's tendency to occasionally jump from a "good" region of the parameter space to an unexplored region of the parameter space. These jumps often result in unstable simulations, which are assigned a high error.

Having obtained calibrated constants for this problem, we would like to demonstrate that these constants translate to another load applied to the same object; i.e. we'd like to confirm that our results are not overfit to the particular load on which the system was calibrated.

Figure 9 demonstrates a new load applied to the same model, which will produce an entirely different deformation and will stress the mesh along a different axis. Figure 10 shows the result of transferring the calibration to this problem. Again we present the undeformed mesh and a "baseline" mesh (constants selected

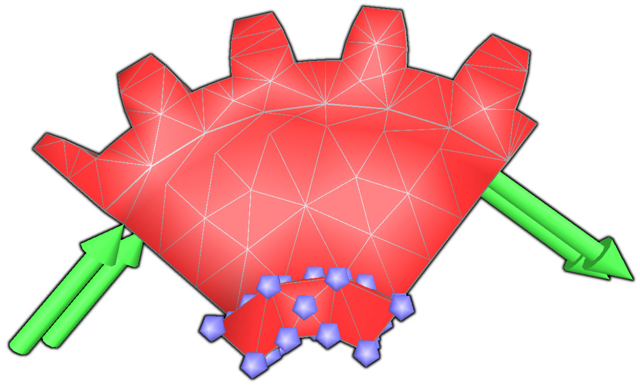


Figure 9. Calibration verification problem. Nodes highlighted in blue are fixed in place; green arrows define the applied load.

quickly by hand) for comparison. We again see an excellent correlation between Figure 10b and Figure 10d, indicating a successful calibration transfer. The RMS vertex error was reduced from 1.0 to 0.1 in this case. The resulting error was thus only slightly higher than the residual self-calibration error represented in Figures 7 and 8.

3.5 Nonhomogeneous Calibration

The results presented so far were based on homogeneous materials, i.e. the four calibrated constants were uniform throughout the object. There are, however, two motivations for allowing inhomogeneous deformation constants.

The first is to allow calibration to inhomogeneous reference objects. An object whose material properties vary in space clearly cannot be represented with homogeneous deformation parameters. This is particularly relevant for applications in virtual surgery, where tissues may have material properties that vary according to microanatomy or pathology, or may represent

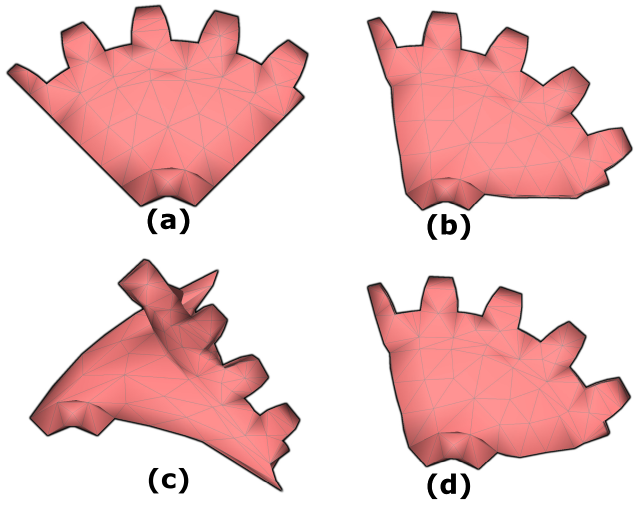


Figure 10. Calibration verification results. (a) Undeformed model. (b) Ground truth deformation (resulting from finite element analysis). (c) Baseline non-constitutive deformation (hand-selected constants). (d) Calibrated non-constitutive deformation, using the results obtained from the problem presented in Figure 6. (b) and (d) are nearly identical, indicating successful calibration transfer to the new problem.

compound materials such as muscle coupled to bone.

A second motivation for allowing inhomogeneous deformation constants is to compensate for deformation properties that are artificially introduced by the geometry and topology of the simulation mesh. van Gelder has shown, for the two-dimensional case, that uniform stiffness properties fail to simulate a uniform object accurately [13]. It is also known that mesh geometry and topology can introduce undesired deformation properties into mass-spring simulations [48]. We would thus like to allow constants to vary within our calibrated mesh, even when it is intended to represent a homogeneous object.

Previous approaches to nonhomogeneous deformation calibration (e.g. [8],[12]) have allowed stiffness constants to vary at each node, which links optimization complexity directly to mesh resolution and presents an enormous optimization landscape.

We present a novel approach to nonhomogeneous parameter optimization, which decouples optimization complexity from simulation complexity and mesh resolution. Specifically, rather than presenting the per-node deformation parameters directly to the optimizer, we allow the optimizer to manipulate deformation parameters defined on a fixed grid; those parameters are then interpolated by trilinear interpolation to each node before every simulation pass. This imposes some continuity constraints on the resulting parameter set (nearby vertices will have similar parameter values), but can greatly speed up the optimization process, making possible the calibration of large meshes that would be prohibitively expensive to optimize per node.

Figure 11 shows an example of the decoupling of the optimization and simulation meshes. The optimization mesh can be arbitrarily simplified to allow, for example, variation of parameters along only one axis of the object (using a k × 1 × 1 optimization grid).

As a preprocessing step, each simulation vertex is associated with a set of weights defining the optimization nodes that affect its parameter set. Specifically, we assign weights to the eight optimization nodes that form a cube around each simulation vertex. We will refer to the coordinates of those nodes as $\lfloor x \rfloor, \lceil x \rceil, \lfloor y \rfloor, \lceil y \rceil, \lfloor z \rfloor, \lceil z \rceil$, representing the upper and lower bounds of this vertex's cell in the optimization grid. The coordinates of the eight nodes of this cell are thus:

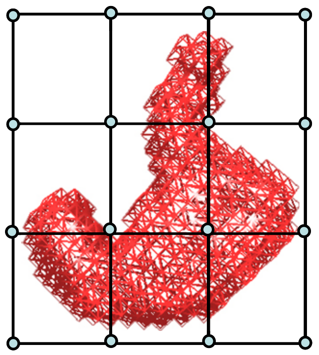


Figure 11. Decoupled simulation and optimization meshes. The optimizer adjusts constants on the larger grid (blue nodes), which are interpolated to the simulation mesh (red) before each simulation pass.

0	$\lfloor x \rfloor, \lfloor y \rfloor, \lfloor z \rfloor$
1	$\lfloor x \rfloor, \lfloor y \rfloor, \lceil z \rceil$
2	$\lfloor x \rfloor, \lceil y \rceil, \lfloor z \rfloor$
3	$\lfloor x \rfloor, \lceil y \rceil, \lceil z \rceil$
4	$\lceil x \rceil, \lfloor y \rfloor, \lfloor z \rfloor$
5	$\lceil x \rceil, \lfloor y \rfloor, \lceil z \rceil$
6	$\lceil x \rceil, \lceil y \rceil, \lfloor z \rfloor$
7	$\lceil x \rceil, \lceil y \rceil, \lceil z \rceil$

We then define the cell-relative position of vertex v along each axis as:

$$v_{xrel} = (v.x - \lfloor x \rfloor)/(\lceil x \rceil - \lfloor x \rfloor)$$
$$v_{yrel} = (v.y - \lfloor y \rfloor)/(\lceil y \rceil - \lfloor y \rfloor)$$
$$v_{zrel} = (v.z - \lfloor z \rfloor)/(\lceil z \rceil - \lfloor z \rfloor)$$

And the trilinear interpolation weights for this vertex associated with each optimization node are:

0	$(1-v_{xrel})(1-v_{yrel})(1-v_{zrel})$
1	$(1-v_{xrel})(1-v_{yrel})(v_{zrel})$
2	$(1-v_{xrel})(v_{yrel})(1-v_{zrel})$
3	$(1-v_{xrel})(v_{yrel})(v_{zrel})$
4	$(v_{xrel})(1-v_{yrel})(1-v_{zrel})$
5	$(v_{xrel})(1-v_{yrel})(v_{zrel})$
6	$(v_{xrel})(v_{yrel})(1-v_{zrel})$
7	$(v_{xrel})(v_{yrel})(v_{zrel})$

Calibration nodes that do not affect parameter values at any vertex (for example, the upper-left calibration node in Figure 11) are discarded and are not used for optimization. In practice, weights are assembled into a (highly sparse) matrix of size [number of calibration_nodes] × [number of vertices] that can be multiplied by a vector of values of length [number of calibration nodes] for each parameter to quickly compute the parameter value at each vertex by matrix-vector multiplication.

Parameter values defined on the optimization grid cannot be used directly for simulation, so to compute a parameter value p_v for a particular simulation vertex v before a simulation pass, we compute the weighted sum:

$$p_v = \sum_{i=0}^{7} w_i p_i$$

…where w_i is the weight associated with node i, as defined above (node numbering here is within a cell, not over the entire grid) and p_i is the parameter value at node i (supplied by the optimizer).

In summary, we learn deformation parameters on a fixed grid, which is generally more sparse than the simulation mesh, and interpolate values to simulation nodes at each evaluation of the error metric. This decouples optimization complexity from mesh complexity.

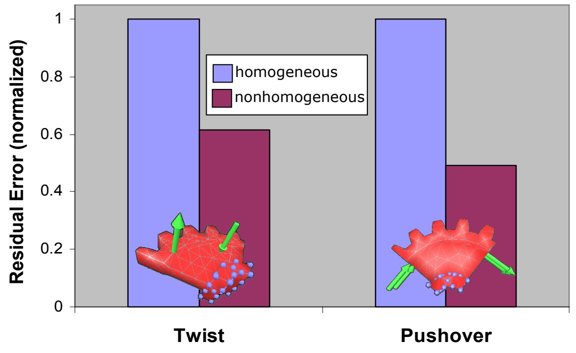


Figure 12. Improvement in self-calibration due to nonhomgeneous calibration. For each problem, the residual calibration errors following homogeneous and nonhomogeneous calibration are indicated in blue and purple, respectively.

3.6 Results: Nonhomogeneous Calibration

We suggested in Section 3.5 that nonhomogeneous calibration should improve calibration results even for homogeneous objects. We will thus revisit the problems presented in Section 3.4 and assess the benefit of nonhomogeneous calibration.

Figure 12 shows the error reduction for "self-calibration" (the residual error at the completion of optimization) for the two "gear" problems introduced in Section 3.5. A significant error reduction is observed in both cases, indicating that the optimizer is able to use the additional degrees of freedom provided through nonhomogeneity. In both cases, a grid resolution of 5 × 5 × 3 was used, where the shortest axis of the gear (the vertical axis in Figure 7a) was assigned to the shortest axis (3 nodes) of the calibration grid.

Having established the benefit of nonhomogeneous calibration for homogeneous objects, we would like to demonstrate the ability of our calibration technique to learn variations in material properties within nonhomogeneous objects.

Figure 13 shows the results of a nonuniform calibration for a cube that was modeled with a uniform Poisson's coefficient (0.499) but included two regions with different Young's moduli (50kPa and 1000kPa) (Figure 13a). An applied load (Figure 13b) resulted in virtually no displacement in the "hard" (bottom) portion of the object according to finite element analysis (Figure 13c). This reference deformation was used to learn constants on an interpolation grid, which converged to the results shown in Figure 13f (values are interpolated to vertices in the figure). Figure 13f shows the distribution of k_d; k_a and k_v showed similar distributions, and the damping constant was treated as uniform for this calibration. The resulting non-constitutive deformation (Figure 13e) can be contrasted with the optimal result obtained using *homogeneous* values for all four constants (Figure 13d).

Figure 14 summarizes the calibration of deformation parameters for a more complex model. Here a grid of 54 nodes is used to calibrate a simulation mesh of over 700 nodes.

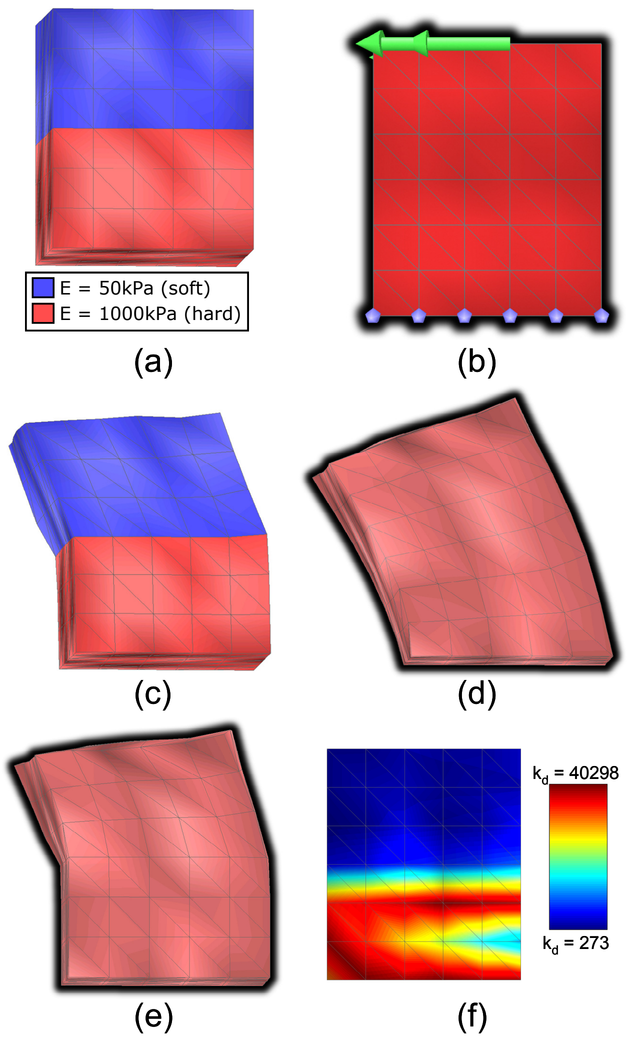


Figure 13. Results following a nonhomogeneous calibration. The object was modeled with a nonuniform Young's modulus (a), and subjected to the load indicated in (b), with blue highlights indicating zero-displacement constraints. (c) The resulting deformation according to finite element analysis (note that the load is absorbed almost entirely in the "soft" region). (d) The resulting deformation according to a calibrated, non-constitutive model with *homogeneous* parameter values. (e) The resulting deformation according to a calibrated, non-constitutive model with *nonhomogeneous* parameter values; note that the load is correctly absorbed in the top part of the object. (f) The distribution of k_d values after calibration.

4. Rendering and Simulation

We have now discussed our approaches to preparing and calibrating tetrahedral meshes and deformation parameters for interactive simulation. This section reviews our selected simulation approach, a reformulation of the method presented in [16]) and discusses our approach to mesh skinning during simulation.

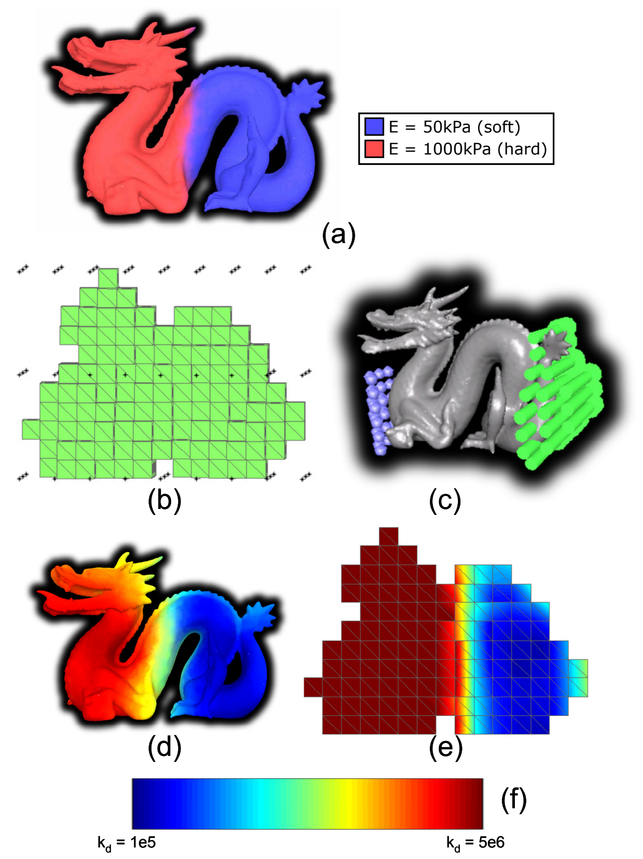


Figure 14. Calibrated material properties for a more complex mesh. (a) The ground truth (constitutive) set of material properties for the object; the two regions have different values for Young's modulus. (b) The tetrahedral mesh used to simulate this model, generated as per Section 1, and the coarse optimization grid used to calibrate this mesh; the grid is 9×3×3 nodes, or 8×2×2 cells. (c) The problem used to calibrate this model; blue highlights indicate zero-displacement constraints, and green arrows indicate applied forces. (d) and (e) show the calibrated values for k_d (k_a and k_v showed similar patterns), displayed on the rendering and simulation meshes, respectively, with the scale of k_d values indicated in (f). The learned regions of high and low k_d correspond closely to the regions of high and low Young's modulus in (a).

4.1 Simulation

The deformation model presented in [16] addresses important limitations in traditional mass-spring systems. In particular, local volume-preservation and area-preservation forces, computed for each tetrahedron and each tetrahedral face, respectively, complement traditional length-preservation forces computed along each tetrahedral edge. This model enforces local volume conservation, which approximately constrains global volume, and allows a much wider variety of material behaviors to be expressed than a traditional mass-spring system.

The original presentation of this approach [16] presents these constraint forces as analytic derivatives of energy functions. We will present equivalent geometric interpretations; our

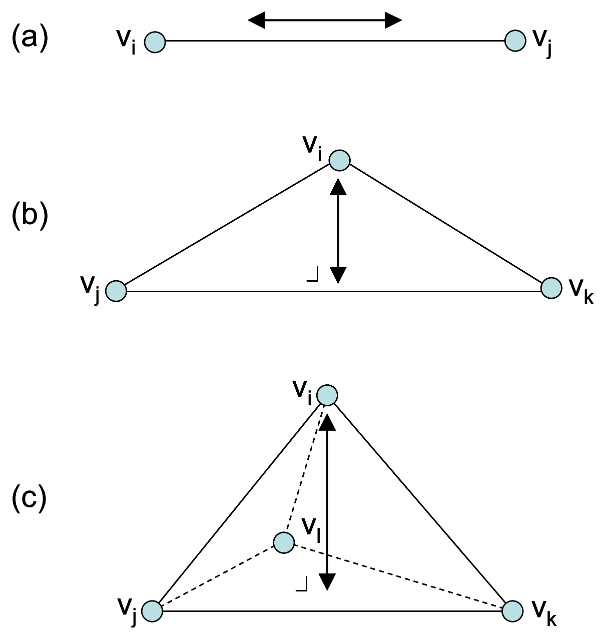


Figure 15. Geometric representations of energy derivatives with respect to vertex v_i, i.e. the direction in which each force should be applied to vertex v_i. (a) Distance preservation. (b) Area preservation. (c) Volume preservation. The double-headed arrow indicates force direction in each case.

implementation is based on these geometric representations of the constraint forces.

Distance Preservation

The energy function E_D associated with the distance-preservation force between two connected vertices v_i and v_j is [16]:

$$E_D(v_i, v_j) = \frac{1}{2} k_d \left(\frac{|v_j - v_i| - D_0}{D_0} \right)^2$$

…where D_0 is the rest length of this edge (computed in preprocessing) and k_d is the distance-preservation constant associated with this edge.

The force applied to vertex v_i to minimize this energy is the traditional spring force:

$$F_D(v_i) = k_d \left(|v_j - v_i| - D_0 \right) \left(\frac{v_j - v_i}{|v_j - v_i|} \right)$$

Intuitively, energy is minimized by shortening or lengthening the spring to its rest length, so we apply a force toward the opposing vertex, whose magnitude depends on the edge's deviation from rest length (Figure 15a).

In practice, edge lengths are computed before any forces are calculated, so they can be accessed by each vertex without recomputation.

Area Preservation

The energy function E_A associated with the area-preservation force for the triangle consisting of vertices v_i, v_j, and v_k is [16]:

$$E_A(v_i,v_j,v_k)=\frac{1}{2}k_a\left(\frac{\frac{1}{2}\left|(v_j-v_i)\times(v_k-v_i)\right|-A_0}{A_0}\right)^2$$

…where A_0 is the rest area of this triangle (computed in preprocessing) and k_a is the area-preservation constant associated with this triangle.

To understand the force we should apply to vertex v_i to minimize this energy, we will view this triangle with the edge (v_j,v_k) on the horizontal axis (Figure 15b). The area of this triangle is equal to ½ times its baseline ($|v_k - v_j|$) times its height. Since the baseline of the triangle cannot be affected by moving vertex v_i, the gradient of the triangle area in terms of the position of v_i is clearly along the vertical axis (maximally affecting the height of the triangle). We thus compute this perpendicular explicitly to find the *direction* of the area-preservation force to apply to vertex v_i:

$$areagradient(v_i)=(v_i-v_j)-\left((v_k-v_j)\bullet\frac{(v_k-v_j)\bullet(v_i-v_j)}{(v_k-v_j)\bullet(v_k-v_j)}\right)$$

$$forcedir_A(v_i)=\frac{F_A(v_i)}{|F_A(v_i)|}=\frac{areagradient(v_i)}{|areagradient(v_i)|}$$

Here we have just decomposed the vector $(v_i - v_j)$ into components parallel to and perpendicular to $(v_k - v_j)$ and discarded the parallel component, then normalized the result (*areagradient*) to get our force direction.

The *magnitude* of this force should be proportional to the difference between the current and rest areas of the triangle. We compute the area as half the cross-product of the edges, i.e.:

$$forcemag_A(v_i)=\frac{1}{2}\left((v_k-v_i)\times(v_j-v_i)\right)-A_0$$

…where A_0 is the rest area of this triangle, computed in preprocessing.

And we scale the final force by the area-preservation constant k_a associated with this triangle:

$$F_A(v_i)=k_a\bullet forcemag_A(v_i)\bullet forcedir_A(v_i)$$

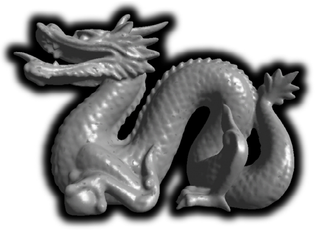
In practice, triangle areas are computed before any forces are calculated, so they can be accessed by each vertex without recomputation (area computation also yields triangle normals, which are used for computing volume-preservation forces).

Volume Preservation

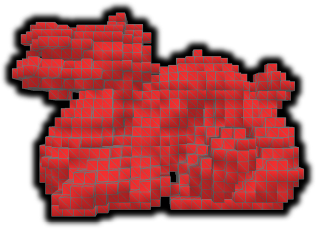
The energy function E_V associated with the volume-preservation force for the tetrahedron consisting of vertices v_i, v_j, v_k, and v_l is [16]:

$$E_V(v_i,v_j,v_k,v_l)=\frac{1}{2}k_v\left(\frac{\frac{1}{6}(v_j-v_i)\bullet\left((v_k-v_i)\times(v_l-v_i)\right)-V_0}{V_0}\right)^2$$

(a)



(b)



(c)

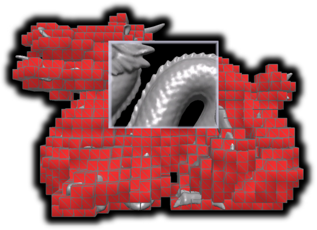


Figure 16. Skinning a rendering mesh on a simulation mesh. (a) Original mesh, used for interactive rendering. (b) Tetrahedral mesh, used for interactive simulation. (c) Rendering mesh skinned on simulation mesh (with cutaway view).

…where V_0 is the rest volume of this tetrahedron (computed in preprocessing) and k_v is the volume-preservation constant associated with this tetrahedron.

To understand the force we should apply to vertex v_i to minimize this energy, we will view this tetrahedron with the face (v_j,v_k,v_l) on the horizontal plane (Figure 15c). The volume of this tetrahedron is equal to 1/3 times its base area (½ ($(v_l - v_j) \times (v_k - v_j)$)) times its height. Since the base area of the tetrahedron cannot be affected by moving vertex v_i, the gradient of the tetrahedron volume in terms of the position of v_i is clearly along the vertical axis (maximally affecting the *height* of the tetrahedron). We thus compute this perpendicular (the normal to the triangle (v_j,v_k,v_l)) explicitly to find the *direction* of the volume-preservation force to apply to vertex v_i:

$$volumegradient(v_i)=\left((v_k-v_j)\times(v_l-v_j)\right)$$

$$forcedir_v(v_i)=\frac{F_v(v_i)}{|F_v(v_i)|}=\frac{volumegradient(v_i)}{|volumegradient(v_i)|}$$

Here we have just computed a vector normal to the triangle (v_j,v_k,v_l) (*volumegradient*) and normalized the result.

The *magnitude* of this force should be proportional to the difference between the current and rest volumes of the tetrahedron. We compute the volume of the tetrahedron and subtract the rest volume:

$$forcemag_V(v_i)=\frac{1}{6}\left((v_j-v_i)\bullet\left((v_k-v_i)\times(v_l-v_i)\right)\right)-V_0$$

…where V_0 is the rest area of this triangle, computed in preprocessing.

And we scale the final force by the volume-preservation constant k_v associated with this tetrahedron:

$$F_V(v_i) = k_v \bullet forcemag_v(v_i) \bullet forcedir_v(v_i)$$

In practice, tetrahedral volumes are computed before any forces are calculated, so they can be accessed by each vertex without recomputation.

As is described in [16], these forces are accumulated for each vertex and integrated explicitly using Verlet integration. A viscous damping force is also applied to each vertex according to a fourth material constant k_{damp}.

4.2 Mesh Skinning

The tetrahedral mesh used for simulation will generally present a lower-resolution surface than the original mesh; rendering this surface directly significantly limits rendering quality (compare Figure 16a to Figure 16b). It is thus desirable to decouple the rendering and simulation meshes by "skinning" a rendering mesh onto a simulation mesh (Figure 16c).

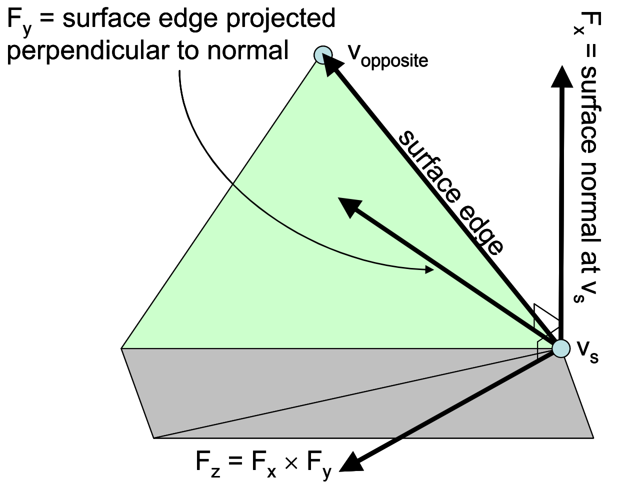
This type of mesh skinning is common for applications that have a low-resolution rigid skeleton for animation and wish to deform a rendering mesh to reflect the movements of the underlying bones, an operation that can be performed on commodity graphics hardware [45]. However, such approaches assume a low-degree-of-freedom underlying skeleton and are thus not suitable for skinning complex meshes. Furthermore, mesh skinning usually involves manual assignment of vertices to one or more bones, which is not practical when the set of independently deforming components is very large. In other words, manually assigning vertices to be controlled by specific tetrahedra would be prohibitively time-consuming.

We thus present an automatic mechanism for skinning a rendering mesh onto a simulation mesh. Our approach is similar to free-form deformation [49], which determines the movement of vertices in a deforming space defined by a grid of control points. In our case, the physically-based deformation of the tetrahedral mesh defines a deforming space, and the vertices of the rendering mesh are translated accordingly.

Specifically, we perform a preprocessing step that begins with defining a "vertex-space" coordinate frame for each vertex v_s on the surface of the simulation mesh. We assume that surface vertices in the simulation mesh are tagged as such during the mesh generation process (Section 2). The vertex-space coordinate frame $\mathbf{F_{vs}}$, with origin at v_s, is defined by the three reference vectors $\mathbf{F_x}$, $\mathbf{F_y}$, and $\mathbf{F_z}$, which are created as follows and are orthogonal by construct (Figure 17):

- $\mathbf{F_x}$: the surface normal at v_s. Surface normals are computed before and during simulation by averaging the face normals of all triangles that contain v_s.
- $\mathbf{F_y}$: The component of first "surface edge" connected to v_s that is perpendicular to the normal at v_s. A "surface edge" is defined as an edge that connects to another vertex that is on the surface of the mesh. This component is computed as follows:

$$F_y = (v_{opposite} - v_s) - (((v_{opposite} - v_s) \bullet N)N)$$



F_y = surface edge projected perpendicular to normal
$v_{opposite}$
surface edge
F_x = surface normal at v_s
v_s
$F_z = F_x \times F_y$

Figure 17. Vertex-space coordinate frame definition. The triangles shown in gray, and their edges, are not used explicitly for defining this vertex's coordinate frame, but will influence the frame through their influence on the surface normal.

…where v_s is the vertex at which we're defining a frame, $v_{opposite}$ is the simulation vertex at the other side of the selected surface edge, and $\mathbf{N}$ is the unit normal vector at V_s. $\mathbf{F_y}$ approximates a local surface tangent vector.

- $\mathbf{F_z}$: The cross-product of $\mathbf{F_x}$ and $\mathbf{F_y}$.

$\mathbf{F_x}$, $\mathbf{F_y}$, and $\mathbf{F_z}$ are each normalized to yield an orthonormal basis. Note that in practice, coordinate frames are not defined until vertices are used in subsequent steps, so that coordinate frames are not computed for vertices that are not used for skinning. We have presented coordinate-frame definition first for clarity.

After defining coordinate frames, we place all simulation vertices on the surface of the simulation mesh in a kd-tree [55].

For each vertex on the rendering mesh, we then find the nearest *nneighbors* vertices on the surface of the simulation mesh. Higher values for *nneighbors* result in more expensive rendering but more accurate rendering mesh deformation. In practice we generally set *nneighbors* = 5.

For each vertex v_r on the rendering mesh, and each of its nearby vertices v_s on the simulation mesh, we then compute the world-frame offset of v_r relative to v_s, and rotate it into the coordinate frame $\mathbf{F_{vs}}$ defined at v_s:

$$offset_{world}(v_r, v_s) = v_r - v_s$$

$$R_{v_s} = \begin{bmatrix} F_x.x & F_x.y & F_x.z \\ F_y.x & F_y.y & F_y.z \\ F_y.x & F_z.y & F_z.z \end{bmatrix}$$

$$offset_{vertex}(v_r, v_s) = [R_{v_s}]offset_{world}$$

…where $\mathbf{F_x}$, $\mathbf{F_y}$, and $\mathbf{F_z}$ are the components of $\mathbf{F_{vs}}$, as computed above. We store $\mathbf{offset}_{vertex}(v_r,v_s)$ for each of the *nneighbors* v_s's associated with v_r. We also compute, for each **offset**$_{vertex}(v_s,v_r)$, a weighting factor defined by the distance between v_s and v_r (closer vertices should have more influence over v_r). The weighting factor for a particular (v_r,v_s) is computed as:

$$w(v_r, v_s) = \frac{\dfrac{1}{|v_r - v_s|^2}}{\displaystyle\sum_{i=1}^{nneighbors} \frac{1}{|v_{r_i} - v_s|^2}}$$

…where the denominator here is a normalization factor ensuring that the *nneighbors* weights add up to 1.

The indices of all weighted vertices, the weight values, and the offset$_{vertex}$ values are stored for each rendering vertex v_r.

During each frame of interactive rendering, for each vertex v_r, we look up the indices and deformed positions of each weighted vertex v_s. Then to find the position at which v_r should be rendered, we recompute each coordinate frame $\mathbf{F_{vs}}$ exactly as described above (including normalization) using the *deformed* position of v_s, yielding new F_x, F_y, and F_z vectors (which we'll refer to as F_x', F_y', and F_z'). The new position for v_r based on a simulation vertex v_s is then computed as:

$$p(v_r, v_s) = F_x' \bullet offset_{vertex}.x + F_y' \bullet offset_{vertex}.y + F_z' \bullet offset_{vertex}.z$$

The coordinate frame is based on the local surface normal and the local tangent vector (the chosen surface edge), and thus rotates with the space surrounding v_s.

The final position for v_r is the weighted average of the position "votes" from each v_s:

$$p(v_r) = \sum_{i=1}^{nneighbors} w(v_r, v_{s_i}) \bullet p(v_r, v_{s_i})$$

4.3 Implementation and Results

The proposed simulation approach is a reformulation of [16], so we refer to their results for detailed deformation results. The proposed skinning approach was implemented using CHAI [27] for visualization and the ANN library [56] for kd-tree searching. With N=5 and a the simulation and rendering meshes shown in Figure 16 (50,000 rendered faces and 13,000 simulated tetrahedra), simulation proceeds at 200fps, with rendering taking place every 10 simulation frames (20fps).

Skinning results are best communicated by video, so we have made a video of our skinning approach, applied during interactive deformation, available at:

http://cs.stanford.edu/~dmorris/video/dragon_deforming.avi

5. Conclusion and Future Work

We have presented an automated pipeline for interactively deforming an object originally defined as a surface mesh. Pipeline stages included mesh generation, calibration to a constitutive model using simulated annealing, and simulation/rendering.

5.1 Future Work: Mesh Generation

Future work on mesh generation will focus on generating nonuniform meshes that provide more resolution in more detailed regions of the surface model; the AABB hierarchy that we already create during voxelization provides a multiresolution representation of the object that translates naturally into a voxel array. Calibration (Section 3) will compensate for simulation artifacts resulting from nonuniform mesh resolution. Also, simulations that involve topology changes (cuts and fractures) and large deformations may benefit from dynamic background re-meshing, another area for future research.

5.2 Future Work: Calibration

Our calibration procedure is currently naïve to the deformation model and treats each error function evaluation as a black box. Calibration would be greatly sped up by automatically and dynamically generating loads that probe sensitive, high-resolution, or user-highlighted regions of the mesh. Also, error gradients are currently estimated by finite differencing; more sophisticated approaches would adjust constants more efficiently using ad hoc heuristics that predict the effects of parameter changes (for example, higher stiffness constants are likely to reduce overall deformation).

Additionally, a more sophisticated error metric would penalize shape deformation but allow rigid body transformations; the current per-vertex-distance metric penalizes all errors equally. The calibration could also derive a more accurate seed point for optimization by using simple, canonical models (for example, homogeneous cubes or single tetrahedra) to obtain approximate canonical values for deformation constants representing particular material properties.

Non-geometric error metrics that incorporate stress or surface tension would also improve the applicability of our approach to applications that require force information, e.g. simulations incorporating haptics or fracture/cut modeling.

Another application of the presented approach is topology optimization. The ability to find optimal constants for a given topology can be generalized to iteratively adjust topology, to minimize mesh size and simulation complexity while still satisfying a given error bound.

We would also like to generalize our calibration approach to more complex deformation models, particularly incorporating dynamics, nonlinear stress/strain relationships, plasticity, and topology changes.

5.3 Future Work: Parallelization

Finally, all of the approaches presented in this paper lend themselves extremely well to parallelization, and are expected to benefit from parallel implementations. Voxelization can be parallelized across regions at a high level, or across AABB nodes at a finer level. A custom annealing procedure could make use of multiple, simultaneous samples in the parameter space, and would be further optimized by a parallelized version of the simulation itself, as per [46]. The skinning approach presented in Section 4 is particularly well-suited to parallel implementation on graphics hardware, especially when using a simulation technique such as

[46], [47], [17], or [18], which place vertex positions in a GPU-resident render target that can be accessed from a vertex shader used to transform the vertices of the rendering mesh.

Supplemental Material

The mesh generation approach presented in Section 2 is available in binary form at:

http://cs.stanford.edu/~dmorris/voxelizer

A video of our mesh skinning approach is available at:

http://cs.stanford.edu/~dmorris/video/dragon_deforming.avi

Acknowledgements

Support was provided by NIH LM07295 and the AO Foundation.

The "dragon", "bunny", and "happy" models were obtained from the Stanford 3D Scanning Repository [57]. The "gear" model was obtained from the TetGen examples page [58].

References

[1] Mueller M and Teschner M. Volumetric Meshes for Real-Time Medical Simulations. *Proc. BVM '03*, Erlangen, Germany, pp. 279-283, March 2003.

[2] Si H. TetGen, A Quality Tetrahedral Mesh Generator and Three-Dimensional Delaunay Triangulator, v1.3 User's Manual. Weierstrass Institute for Applied Analysis and Stochastics, Technical Report No. 9, 2004.

[3] Bridson R, Teran J, Molino N, Fedkiw R. Adaptive Physics Based Tetrahedral Mesh Generation Using Level Sets. *Engineering with Computers* 21, 2-18 (2005).

[4] Zhang H. Mesh Generation for Voxel-Based Objects, PhD Thesis, West Virginia University, 2005.

[5] Ho-Le K. Finite element mesh generation methods: a review and classification. *Computer Aided Design*, Vol 20(1), 27-38, 1988.

[6] Bhat K, Twigg C, Hodgins J, Khosla P, Popovic Z, Seitz S. Estimating cloth simulation parameters from video. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (SCA) 2003.

[7] Bianchi G, Harders M, Székely G. Mesh Topology Identification for Mass-Spring Models. *Proceedings of Medical Image Computing and Computer-Assisted Intervention* (MICCAI) 2003, Montreal, Canada, November 2003.

[8] Bianchi G, Solenthaler B, Székely G, Harders M. Simultaneous Topology and Stiffness Identification for Mass-Spring Models based on FEM Reference Deformations. *Proceedings of Medical Image Computing and Computer-Assisted Intervention* (MICCAI) 2004. St-Malo, France, November 2004.

[9] Etzmuss O, Gross J, Strasser W. Deriving a Particle System from Continuum Mechanics for the Animation of Deformable Objects. *IEEE Transactions on Visualization and Computer Graphics*, v.9 n.4, p.538-550, October 2003.

[10] Deussen O, Kobbelt L, Tücke P. Using Simulated Annealing to Obtain Good Nodal Approximations of Deformable Bodies. *Proc. Sixth Eurographics Workshop on Simulation and Animation*, September 1995, Maastricht.

[11] Choi K-S, Sun H, Heng P-A, Zou J. Deformable simulation using force propagation model with finite element optimization. *Computers & Graphics* 28(4): 559-568 (2004).

[12] Mosegaard J. Parameter Optimisation for the Behaviour of Elastic Models over Time. *Proceedings of Medicine Meets Virtual Reality 12*, 2004.

[13] Van Gelder A. Approximate Simulation of Elastic Membranes by Triangle Meshes. *Journal of Graphics Tools*,3:21--42, 1998, 178-180.

[14] James D and Twigg CD. Skinning Mesh Animations. *ACM Transactions on Graphics* (*SIGGRAPH 2005*), Vol. 24, No. 3, August 2005.

[15] Kry PG, James DL, Pai DK. EigenSkin: Real Time Large Deformation Character Skinning in Hardware. *Proceedings of ACM SIGGRAPH 2002 Symposium on Computer Animation*.

[16] Teschner M, Heidelberger B, Mueller M, Gross M. A Versatile and Robust Model for Geometrically Complex Deformable Solids. *Proceedings of Computer Graphics International* (CGI'04), Crete, Greece, pp. 312-319, June 16-19, 2004.

[17] Mosegaard J and Sørensen TS. GPU Accelerated Surgical Simulators for Complex Morphology. *Proceedings of IEEE Virtual Reality 2005*.

[18] Mosegaard J, Herborg P, Sørensen TS. A GPU Accelerated Spring Mass System for Surgical Simulation. *Proceedings of Medicine Meets Virtual Reality 13*, 2005.

[19] Nienhuys H-W and van der Stappen AF. A surgery simulation supporting cuts and finite element deformation. Proceedings of MICCAI 2001.

[20] Lindblad AJ, Turkiyyah GM, Weghorst SJ, Berg D. Real-Time Finite Element Based Virtual Tissue Cutting. Proceedings of MMVR 2006, 24-27 January 2006, Long Beach, CA.

[21] Berkley J, Turkiyyah G, Berg D, Ganter M, Weghorst S. Real-Time Finite Element Modeling for Surgery Simulation: An Application to Virtual Suturing. IEEE Transactions on Visualization and Computer Graphics, 10(3), 1-12.

[22] Mendoza C and Laugier C. Simulating soft tissue cutting using finite element models. Proceedings of ICRA 2003: 1109-1114.

[23] James D and Pai D. ARTDEFO: Accurate real time deformable objects. *Proceedings of ACM SIGGRAPH 1999*, 65—72.

[24] Balaniuk R and Salisbury JK. Dynamic Simulation of Deformable Objects Using the Long Elements Method. IEEE Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems 2002: 58-65.

[25] Balaniuk R and Salisbury JK. Soft-Tissue Simulation Using the Radial Elements Method. Surgery Simulation and Soft Tissue Modeling: International Symposium, IS4TM 2003 Juan-Les-Pins, France, June 12-13, 2003.

[26] Morris D. Algorithms and Data Structures for Haptic Rendering: Curve Constraints, Distance Maps, and Data Logging. Stanford University Department of Computer Science Technical Report 2006-06, June 2006.

[27] Conti F, Barbagli F, Morris D, Sewell C. CHAI: An Open-Source Library for the Rapid Development of Haptic Scenes Demo paper presented at *IEEE World Haptics*, Pisa, Italy, March 2005.

[28] Ingber L. Adaptive simulated annealing (ASA): lessons learned. *Control and Cybernetics*, 25, 33–54, 1995.

[29] PhysX, Ageia, http://www.ageia.com/

[30] HavokFX, Havok, Inc., http://www.havok.com/

[31] SimBionix, http://www.simbionix.com

[32] CardioSkills, http://www.cardioskills.com/

[33] Immersion Medical, http://www.immersion.com/medical/

[34] SimSurgery, http://www.simsurgery.com/

[35] Surgical Science, http://www.surgical-science.com/

[36] Xitact, http://www.xitact.com/

[37] Montgomery K, Bruyns C, Wildermuth S, Heinrichs L, Hasser C, Ozenne S, Bailey D. Surgical Simulator for Operative Hysteroscopy. *IEEE Visualization 2001*, San Diego, California, October 14-17, 2001.

[38] Brown J, Montgomery K, Latombe JC, Stephanides M. A Microsurgery Simulation System. *Medical Image Computing and Computer-Assisted Intervention* (MICCAI) 2001, Utrecht, The Netherlands, October 14-17, 2001.

[39] Webster RW, Haluck R, Mohler B, Ravenscroft R, Crouthamel E, Frack T, Terlecki S, Sheaffer J. Elastically deformable 3d organs for haptic surgical simulators. In *Proceedings of Medicine Meets Virtual Reality* (MMVR) 2002.

[40] Webster R, Haluck RS, Zoppetti G, Benson A, Boyd J, Charles N, Reeser J, Sampson S. A Haptic Surgical Simulator for Laparoscopic Cholecystectomy using Real-time Deformable Organs. *Proceedings of the IASTED International Conference on Biomedical Engineering*, June 25-27, 2003 Salzburg, Austria.

[41] Nienhuys H-W and van der Stappen AF. A surgery simulation supporting cuts and finite element deformation. *Proceedings of Medical Image Computing and Computer Assisted Intervention* (MICCAI) 2001.

[42] Brouwer I, Ustin J, Bentley L, Sherman A, Dhruv N, Tendick F. Measuring In Vivo Animal Soft Tissue Properties for Haptic Modeling in Surgical Simulation. *Proceedings of Medicine Meets Virtual Reality* 2001.

[43] Samani A, Bishop J, Luginbuhl C, Plewes DB. Measuring the elastic modulus of ex vivo small tissue samples. *Physics in Medicine and Biology*, 48, July 2003.

[44] Tay B, Stylopoulos N, De S, Rattner DW, Srinivasan MA. Measurement of In-vivo Force Response of Intra-abdominal Soft Tissues for Surgical Simulation. *Proceedings of Medicine Meets Virtual Reality*, pages 514-519, Newport Beach, January 2002.

[45] Nvidia tutorial:
http://developer.nvidia.com/object/skinning.html

[46] Georgii J, Echtler F, Westermann R. Interactive Simulation of Deformable Bodies on GPUs. *SimVis 2005*: 247-258.

[47] Tejada E and Ertl T. Large Steps in GPU-based Deformable Bodies Simulation. *Simulation Theory and Practice*, Special Issue on Special Issue on Programmable Graphics Hardware. 2005.

[48] Bourguignon D and Cani M-P. Controlling Anisotropy in Mass-Spring Systems. *Eurographics Workshop on Computer Animation and Simulation* (EGCAS), pages 113-123, August 2000.

[49] Sederberg TW and Parry SR. Free-Form Deformation of Solid Geometric Models. *Proceedings of SIGGRAPH '86*, Computer Graphics 20, 4 (August 1986), 151-159.

[50] Kirkpatrick S, Gelatt CD, Vecchi MP. Optimization by Simulated Annealing. *Science*, Vol 220, Number 4598, pages 671-680, 1983.

[51] Abaqus/Standard and Abaqus/CAE. Abaqus, Inc., Providence, RI. http://www.hks.com/

[52] Matlab 7.0 (R14). The MathWorks, Inc., Natick, MA.
http://www.mathworks.com/

[53] Sakata S. Asamin 1.30. 2006.
http://www.econ.ubc.ca/ssakata/public_html/software/

[54] Ingber, L. ASA-26.14. 2006.
http://www.ingber.com/#ASA

[55] Bentley JL. Multidimensional binary search trees used for associative searching. *Communications of the ACM* 18, 9 (Sep. 1975), 509-517.

[56] Mount DM and Arya S. ANN: A library for approximate nearest neighbor searching. *CGC 2nd Annual Fall Workshop on Computational Geometry*, 1997. Available at http://www.cs.umd.edu/~mount/ANN .

[57] http://graphics.stanford.edu/data/3Dscanrep/

[58] http://tetgen.berlios.de/fformats.examples.html