# Finite Element Modeling in Surgery Simulation

MORTEN BRO-NIELSEN, MEMBER, IEEE

*Invited Paper*

*Modeling the deformation of human organs for surgery simulation systems has turned out to be quite a challenge. Not only is very little known about the physical properties of general human tissue but in addition, most conventional modeling techniques are not applicable because of the timing requirements of simulation systems. To produce a video-like visualization of a deforming organ, the deformation must be determined at rates of 10–20 times/s.*

*In the fields of elasticity and related modeling paradigms, the main interest has been the development of accurate mathematical models. The speed of these models has been a secondary interest. But for surgery simulation systems, the priorities are reversed. The main interest is the speed and robustness of the models, and accuracy is of less concern.*

*Recent years have seen the development of different practical modeling techniques that take into account the reversed priorities and can be used in practice for real-time modeling of deformable organs.*

*This paper discusses some of these new techniques in the reference frame of finite element models. In particular, it builds on the recent work by this author on fast finite element models and discusses the advantages and disadvantages of these models in comparison to previous models.*

***Keywords—*** *Animation, biomechanics, biomedical imaging, computer graphics, finite difference methods, finite element methods, geometric modeling, image analysis, limbs, mechanical engineering, simulation, surgery, virtual reality, X-ray tomography.*

## I. INTRODUCTION

In recent years, we have grown accustomed to seeing highly sophisticated computer graphics in movies such as *Jurassic Park* and *Terminator 2*. The quality and realism of the graphics in these movies is extraordinary and has been an inspiration and challenge to the computer graphics industry. But these movies have only been possible because the movie production process allowed the computer graphics sequences to be created off-line, each of the 20–30 frames/s possibly taking hours to render.

In the field of virtual reality (VR) (which includes simulation of surgical procedures), conditions are different. Virtual environments are interactive and reactive, allowing the user to modify and interact with objects in the virtual scene using virtual tools. Because of the unpredictable nature of the interaction, it is not possible to precompute images for each of the 10–20 frames/s that are needed to provide an immersive VR experience.

Instead, each frame has to be computed on the fly, taking into account the actions of the user, and only limited precomputation is possible. This form of real-time and interactive computer graphics is a significant technical challenge.

Effective surgical simulation is even more difficult. Not only do we need real-time interactive graphics but the objects in the scene should also exhibit physically correct behaviors corresponding to the behaviors of real human organs and tissues.

Unfortunately, human tissue is very complex and often behaves viscoelastically. In addition, human body parts consist of layers of different tissues interlaced with ligaments and fascias. Very complex models are needed to model these objects realistically.

An added practical problem is the acquisition of parameters for the models corresponding to different tissue types. Very little useful data are available describing these parameters, and in practice, it is unlikely that spatially and qualitatively accurate parameters will become available in the foreseeable future.

Last, in practice, the desired realism of the physical models must be balanced against the need for speed. The initial publications in this area have been characterized more by the quest for speed than realism.

To build a general surgery simulator, the following main components are needed.

1) *Computer graphics:* Graphics is needed to render realistic views of the virtual surgery scene and provide the surgeon with a *visual* illusion of reality.
2) *Haptic interface:* This interface is provided to represent the instruments and tools that the surgeon uses to work on the surgery simulator. By tracking the position of these tools and sensing their state, the computer is able to determine the surgeon's actions and provide them as input to the simulation system.

Manuscript received June 21, 1997; revised November 7, 1997.

The author is with HT Medical, Inc., Rockville, MD 20852 USA (e-mail: bro@ht.com).

Publisher Item Identifier S 0018-9219(98)01726-5.

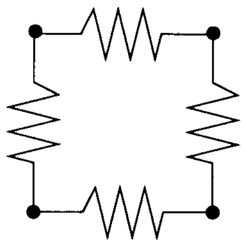


**Fig. 1.** Mass-spring model.

In reaction to these inputs, a haptic interface can provide the surgeon with a physical sensation of touching and sensing objects in the virtual scene using force-feedback techniques. The haptic interface thus closes the loop between action and reaction by providing the *tactile* illusion of reality.

3) *Physical modeling:* Physical models provide the surgeon with a *behavioral* illusion of reality. By modeling the viscoelastic deformation of human skin, the fluid flow of blood from a wound, etc., these models ensure that the virtual scene reflects the behavior of the physical reality.

Computer graphics has been extensively developed over the last few decades and is today deemed by many to be quite sufficient for surgery simulation systems.

Haptic interface devices for minimally invasive surgery simulation can be bought commercially, and although benefits of continued development are expected, the quality is sufficient for many purposes.

Physical models represent the biggest obstacle in surgery simulation. In particular, the development of real-time deformable models of human soft tissue is essential for the continued evolution of surgery simulation systems.

This paper presents a methodology for modeling human soft tissue in real time by applying finite element (FE) modeling techniques in a computationally efficient framework.

## *A. Previous Work*

The demand for real-time performance has forced most researchers to develop or adapt very simplistic models of elastic deformation to the needs of surgery simulation. As an example, instead of using implicit nodal methods, which require the solution of matrix systems, explicit models (e.g., mass-spring models) have frequently been used. Although these models suffer from poor precision and stability problems, they are very easy to implement and yield reasonable speeds.

A mass-spring model consists of a number of nodes connected by springs (see Fig. 1). The mass of the modeled object is concentrated on the nodes; thus the name.

Cover *et al.* [9] were the first to present real-time models for surgery simulation. They used a simple surface-based mass-spring model to simulate deformation of a gallbladder.

Surface models have also been used in the commercial Teleos software developed by HT Medical, Inc.[1] Teleos uses a tubular spline surface controlled by an imbedded particle-based spine to represent deformable objects. It can model simple structures derived from the tubular topology such as arteries, a gallbladder, etc.

[1] Previously known as High Techsplanations, Inc.

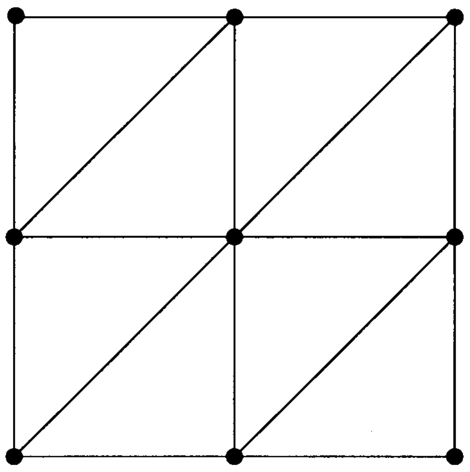


**Fig. 2.** FE mesh model.

A major problem with surface-based models is that they do not allow for volumetric behavior, i.e., if you push on one side of an organ, the other side should move too. This behavior can only be achieved using volumetric models that model the interior of the organ and thereby implicitly connect the opposite sides of the organ model. In addition, the interior of surface models is not defined. It is not possible to make incisions in these models for simulating surgical procedures since the models are hollow shells.

Kühnapfel *et al.* [14], [15] have implemented mass-spring models in their KISMET simulation system. Although they in principle use surface models, they introduce a volumetric behavior by adding interior parent nodes that connect surface nodes on opposite sides of an object.

One of the difficulties of mass-spring models is the ad hoc definition of the geometry. Although mass-spring models can be defined on the basis of FE meshes, springs are usually traced from each mass node to many other mass nodes, often crossing each other. This allows for efficient transfer of forces but complicates the calculation of nodal parameters.

In the work of Deussen *et al.* [10], Kühnapfel's group presented work on determining the optimal positioning of nodes and their masses. This paper reveals how they are forced to use stochastic methods, in this case simulated annealing, to find the optimal configuration of the spring network.

An alternative methodology for elastic deformation is offered by FE models of elasticity. An FE model consists of a mesh defined over a set of nodes (see Fig. 2) describing the geometry of the object and an interpolation function over the mesh approximating the real deformation field.

FE models do not suffer from geometry problems, since the definition of the FE mesh geometry leads to straightforward computation of element parameters. In addition, simple linear models resembling mass spring models can be derived from FE using the FE computation approach to compute spring and nodal parameters.

Implicitly solved finite element systems have been used in the parallel work of this author [2], [4], [5] and Cotin *et al.* [7], [8].

The first work on the use of implicit FE models of linear elastic objects was presented in [2] and [7]. This paper describes later work by this author [4], [5], where a technique called condensation is introduced to reduce the complexity of volumetric FE models. Additional speed-up is achieved by explicitly inverting the stiffness matrix of the resulting linear matrix system and using a selective matrix-vector multiplication method. The resulting models are called fast finite element (FFE) models.

The most advanced use of finite elements has been presented by Sagar *et al.* [17]. Their eye-surgery simulator uses a nonlinear incompressible Mooney–Rivlin [6] elastic FE model of the eye. The FE solution to this kind of model is normally computationally expensive, and it is currently only in such very specialized simulators that it is possible to use these models.

One of the major advantages of methods based on FE models is the scalability of the solution method. With the same mesh structure it is possible to increase the precision and complexity of the model as more computer power becomes available. At the same time, it also allows for graceful degradation of the model by going the opposite direction from an advanced model to a less advanced but faster model.

## II. Fast Finite Element Models

To develop a model for real-time deformation, we need to make a series of decisions. We need to decide on a geometric description of the object, a mathematical model of the elastic deformation, and a solution algorithm, which together allow the solution to be determined quickly and reasonably.

What does *reasonably* mean in this context?

If we were modeling an airplane with the aim of determining the maximum load that it could accept, we would need very high accuracy.

But for many simulation systems, absolute accuracy is less important. Since the tissue composition of each patient that a surgeon meets is slightly different and probably not predictable, for many purposes it does not really matter whether the deformation that the surgeon sees in the virtual environment is accurate as long as it seems realistic. Just as important is that the model is robust and shows a consistent and predictable behavior over time. The model can be inaccurate if it looks right in a consistent way.

In addition, with methods currently available, it is almost impossible to measure material parameters of human tissue completely and accurately. Therefore, even in the case that an accurate model was available, the accuracy would still be controlled by the poor precision of the material parameters.

This discussion leads to the conclusion that the following requirements should have the highest priority:

1) speed;
2) robustness;
3) satisfactory visual result.

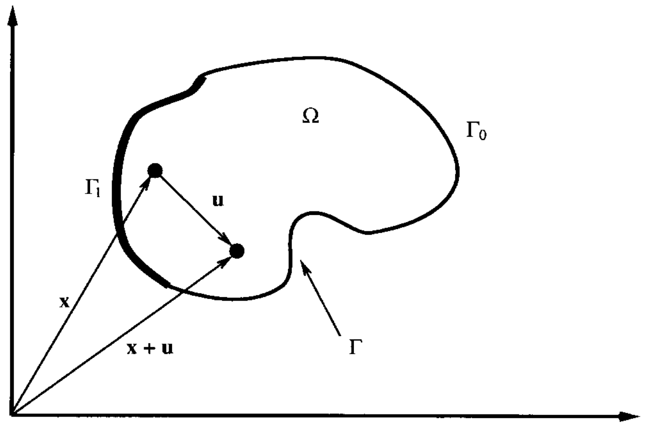


**Fig. 3.** Solid elastic object.

In the following, we will satisfy these requirements by modeling the deformation using an FE approximation of linear elasticity based on a tetrahedral mesh structure.

Using linear elasticity as the basic model involves a number of assumptions regarding the physical material being modeled. Most important, linear elastic models are only valid for very small deformations and strains. They are typically correct for metal beams, building structures, etc. Although they are used extensively in modeling, their accuracy for large deformation modeling is quite poor.

But when used with FE, these models lead to linear matrix systems that are easy to solve, and fast. Since speed is more important to us at this stage, we ignore the associated inaccuracies. Modeling elastic volumetric deformation using FE is only *barely* possible with today's computers. With faster computers in the future, more realistic models, such as incompressible Mooney–Rivlin material models [6], can be used.

## III. Linear Elastic Material Model

We define the organ to be modeled as a three-dimensional (3-D) linear elastic solid $\Omega$. $\Omega$ consists of particles with positions $\boldsymbol{x} = [x, y, z]^T$, where $\boldsymbol{x} \in \Omega$ (see Fig. 3). When forces are applied to $\Omega$, it is deformed into a new shape. The corresponding displacement of $\boldsymbol{x}$ is defined as $\boldsymbol{u}(\boldsymbol{x}) = [u, v, w]^T$ so that the particle $\boldsymbol{x}$ is moved by the deformation to $\boldsymbol{x} + \boldsymbol{u}$.

The boundary of the domain $\Omega$ is defined as $\Gamma = \Gamma_0 \cup \Gamma_1$, $\Gamma_0 \cap \Gamma_1 = \emptyset$, where $\Gamma_0$ is the part of the boundary that has fixed displacements $\boldsymbol{u}(\boldsymbol{x}) = \boldsymbol{u}_0(\boldsymbol{x})$ imposed on it and $\Gamma_1$ is the part where surface forces are applied. The fixed displacements are used to impose constraints on the model.

The *strain* energy [6], [12] of the linear elastic body $\Omega$ is defined as

$$\mathrm{E_{strain}} = \tfrac{1}{2} \int_{\Omega} \boldsymbol{\epsilon}^T \boldsymbol{\sigma} \, d\boldsymbol{x} \tag{1}$$

where $\boldsymbol{\epsilon}$ is the *engineering strain vector* and $\boldsymbol{\sigma}$ is the *engineering stress vector*. Although using the engineering notation is a little out of step with the general notation used in elasticity, it is easier to use for linear elasticity and makes the derivation simpler.

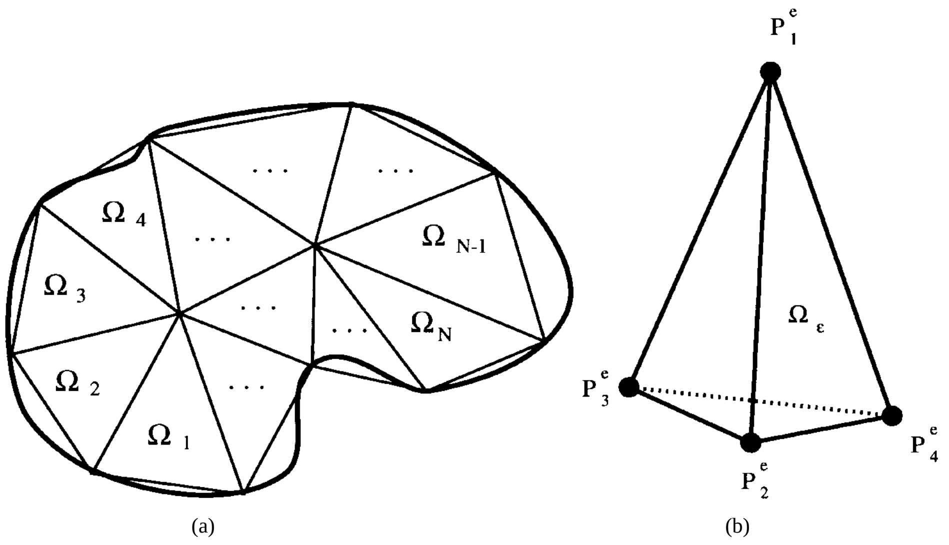


**Fig. 4.** (a) Discretization of the domain into finite elements (two-dimensional illustration). (b) Tetrahedral finite element.

The engineering strain vector is defined as $\boldsymbol{\epsilon} = [\epsilon_x\ \epsilon_y\ \epsilon_z\ \gamma_{xy}\ \gamma_{xz}\ \gamma_{yz}]^T$ and consists of

$$\epsilon_x = \frac{\delta u}{\delta x} \qquad \epsilon_y = \frac{\delta u}{\delta y} \qquad \epsilon_z = \frac{\delta u}{\delta z}$$
$$\gamma_{xy} = \frac{\delta u}{\delta y} + \frac{\delta v}{\delta x} \quad \gamma_{xz} = \frac{\delta u}{\delta z} + \frac{\delta w}{\delta x} \quad \gamma_{yz} = \frac{\delta v}{\delta z} + \frac{\delta w}{\delta y}. \quad (2)$$

We can rewrite this as $\boldsymbol{\epsilon} = \boldsymbol{B}\boldsymbol{u}$ where

$$\boldsymbol{B} = \begin{bmatrix} \frac{\delta}{\delta x} & 0 & 0 \\ 0 & \frac{\delta}{\delta y} & 0 \\ 0 & 0 & \frac{\delta}{\delta z} \\ \frac{\delta}{\delta y} & \frac{\delta}{\delta x} & 0 \\ \frac{\delta}{\delta z} & 0 & \frac{\delta}{\delta x} \\ 0 & \frac{\delta}{\delta z} & \frac{\delta}{\delta y} \end{bmatrix}. \quad (3)$$

The engineering stress vector $\boldsymbol{\sigma}$ is related to the strain vector through Hooke's law [12]

$$\boldsymbol{\sigma} = \boldsymbol{C}\boldsymbol{\epsilon} \quad (4)$$

where $\boldsymbol{C}$ is called the *material matrix*. For a *homogenous and isotropic* material, this matrix is defined by the two Lamé material constants $\lambda$ and $\mu$

$$\boldsymbol{C} = \begin{bmatrix} \lambda+2\mu & \lambda & \lambda & 0 & 0 & 0 \\ \lambda & \lambda+2\mu & \lambda & 0 & 0 & 0 \\ \lambda & \lambda & \lambda+2\mu & 0 & 0 & 0 \\ 0 & 0 & 0 & \mu & 0 & 0 \\ 0 & 0 & 0 & 0 & \mu & 0 \\ 0 & 0 & 0 & 0 & 0 & \mu \end{bmatrix}. \quad (5)$$

Using these relations, we can now rewrite the strain energy and add work done by internal and external forces $\boldsymbol{f}$ and $\boldsymbol{g}$, respectively, to yield the total energy function

$$\mathrm{E}(\boldsymbol{u}) = \tfrac{1}{2} \int_\Omega \boldsymbol{u}^T \boldsymbol{B}^T \boldsymbol{C} \boldsymbol{B} \boldsymbol{u}\, d\boldsymbol{x} - \int_\Omega \boldsymbol{f}^T \boldsymbol{u}\, d\boldsymbol{x} - \int_{\Gamma_1} \boldsymbol{g}^T \boldsymbol{u}\, d\boldsymbol{a}. \quad (6)$$

In the next section, we will discretize this continuous energy function, using a FE model of the displacement field.

## IV. Discretization Using FE Model

We assume that the domain $\Omega$ of the volumetric solid has been discretized into a number of finite elements $\Omega^e$ in the form of tetrahedrons over nodes $P_q$ defined by $\boldsymbol{x}_q = [x_q,\, y_q,\, z_q]^T$ (see Fig. 4). The deformation at each node is specified by the deformation vector $\boldsymbol{u}_q = [u_q,\, v_q,\, w_q]^T$. For notational convenience, we stack these displacement vectors into a compound vector

$$\underline{\boldsymbol{u}} = [\boldsymbol{u}_1^T, \boldsymbol{u}_2^T, \cdots, \boldsymbol{u}_n^T]^T.$$

The nodes of each finite element $\Omega^e$ are denoted $P_i^e$, where $i$ is the local number of the node, unrelated to the global numbering of the nodes. Discretized element factors are denoted with a superscripted $e$. The corresponding compound element displacement vector is

$$\underline{\boldsymbol{u}}^e = [\boldsymbol{u}_1^{eT}, \boldsymbol{u}_2^{eT}, \boldsymbol{u}_3^{eT}, \boldsymbol{u}_4^{eT}]^T. \quad (7)$$

For a point $\boldsymbol{x}$ lying within the element $e$, we can estimate the displacement $\boldsymbol{u}(\boldsymbol{x})$ by making a weighted average of the displacement field at the four nodes of the tetrahedron

$$\boldsymbol{u}(\boldsymbol{x}) = \sum_{i=1}^{4} N_i^e(\boldsymbol{x}) \boldsymbol{u}_i^e. \quad (8)$$

Using linear interpolation, the basis functions $N_i^e(\boldsymbol{x})$ are defined as the so-called natural coordinates $L_i$ of the tetrahedron

$$N_i^e(\boldsymbol{x}) = L_i = \frac{1}{6V^e}(a_i^e + b_i^e x + c_i^e y + d_i^e z), \qquad i = 1,\, 2,\, 3,\, 4. \quad (9)$$

The natural coordinates, the volume $V^e$, and the coefficients $a_i^e$, $b_i^e$, $c_i^e$, $d_i^e$ are all defined in the appendix.

In (6), we used $\boldsymbol{Bu}$ as a notational trick to enable us to write the continuous equation easily. We now need to discretize the equation and thus evaluate $\boldsymbol{Bu}$.

Using the fact that

$$\begin{aligned}\frac{\partial \boldsymbol{u}}{\partial x} &= \sum_{i=1}^{4} \frac{\partial N_i^e(\boldsymbol{x})}{\partial x} \boldsymbol{u}_i^e \\ \frac{\partial \boldsymbol{u}}{\partial y} &= \sum_{i=1}^{4} \frac{\partial N_i^e(\boldsymbol{x})}{\partial y} \boldsymbol{u}_i^e \\ \frac{\partial \boldsymbol{u}}{\partial z} &= \sum_{i=1}^{4} \frac{\partial N_i^e(\boldsymbol{x})}{\partial z} \boldsymbol{u}_i^e \end{aligned} \tag{10}$$

where

$$\begin{aligned}\partial N_i^e(\boldsymbol{x})/\partial x &= \frac{b_i^e}{6V^e} \\ \partial N_i^e(\boldsymbol{x})/\partial y &= \frac{c_i^e}{6V^e} \\ \partial N_i^e(\boldsymbol{x})/\partial z &= \frac{d_i^e}{6V^e}\end{aligned} \tag{11}$$

we find that the discretized strain energy may be written as

$$\mathrm{E}_{\mathrm{strain}}(\underline{\boldsymbol{u}}) = \tfrac{1}{2} \sum_e \int_{\Omega^e} \underline{\boldsymbol{u}}^{eT} \boldsymbol{B}^{eT} \boldsymbol{C} \boldsymbol{B}^e \underline{\boldsymbol{u}}^e \, d\boldsymbol{x} \tag{12}$$

where $\boldsymbol{B}^e$ is a constant matrix defined in the appendix.

The solution to the deformation problem is found when the total energy of the system assumes its minimum value. This happens when the first variation of the functional E vanishes, i.e., when $\delta \mathrm{E}(\boldsymbol{u}) = 0$.

The equilibrium equation $\delta \mathrm{E}(\boldsymbol{u}) = 0$ can be split into element contributions

$$\delta \mathrm{E}(\boldsymbol{u}) = \sum_e \delta \mathrm{E}^e(\boldsymbol{u}) = 0 \tag{13}$$

where

$$\delta \mathrm{E}^e(\boldsymbol{u}) = \sum_{i=1}^{4} \frac{\partial \mathrm{E}^e}{\partial u_i^e} \delta u_i^e + \sum_{i=1}^{4} \frac{\partial \mathrm{E}^e}{\partial v_i^e} \delta v_i^e + \sum_{i=1}^{4} \frac{\partial \mathrm{E}^e}{\partial w_i^e} \delta w_i^e. \tag{14}$$

But since the $u_i^e$, $v_i^e$, and $w_i^e$ are independent, this means that

$$\frac{\partial \mathrm{E}^e}{\partial u_i^e} = \frac{\partial \mathrm{E}^e}{\partial v_i^e} = \frac{\partial \mathrm{E}^e}{\partial w_i^e} = 0, \qquad i = 1, 2, 3, 4. \tag{15}$$

The resulting equilibrium equation for each element thus becomes

$$\boldsymbol{o} = \int_{\Omega^e} \boldsymbol{B}^{eT} \boldsymbol{C} \boldsymbol{B}^e \underline{\boldsymbol{u}}^e \, d\boldsymbol{x} - \underline{\boldsymbol{f}}^e \tag{16}$$

where $\underline{\boldsymbol{f}}^e$ is a generalized discretized element compound force vector.[2]

[2] $\underline{\boldsymbol{u}}^e$ is $12 \times 1$, $\underline{\boldsymbol{f}}^e$ is $12 \times 1$, and $\boldsymbol{B}^e$ is $6 \times 12$.

Because everything inside the integration sign is constant, the equilibrium equation for the finite element reduces to a linear matrix equation

$$\boldsymbol{K}^e \underline{\boldsymbol{u}}^e = \underline{\boldsymbol{f}}^e \tag{17}$$

where

$$\boldsymbol{K}^e = \boldsymbol{B}^{eT} \boldsymbol{C} \boldsymbol{B}^e V^e \tag{18}$$

is called the *element stiffness matrix*[3] and $V^e$ is the volume of the tetrahedron (see the appendix).

The only remaining step is the assembly of the global stiffness matrix and force vector from the element stiffness matrices and element force vectors

$$\begin{aligned}\boldsymbol{K} &= \sum_e \mathrm{global}(\boldsymbol{K}^e) \\ \underline{\boldsymbol{f}} &= \sum_e \mathrm{global}(\underline{\boldsymbol{f}}^e)\end{aligned} \tag{19}$$

where global() is a transfer function from element node numbers to global node numbers.

The result is a large sparse linear matrix system

$$\boldsymbol{K}\underline{\boldsymbol{u}} = \underline{\boldsymbol{f}} \tag{20}$$

where the size of the global stiffness matrix $\boldsymbol{K}$ is $3n \times 3n$, $n$ being the number of nodes in the system. Since the number of nodes is often quite large (possibly in the thousands), this matrix easily becomes nontrivial in size.

### *A. Fixing Nodes*

The stiffness matrix $\boldsymbol{K}$ is singular as it is defined above. Therefore, in order to solve the linear system $\boldsymbol{K}\underline{\boldsymbol{u}} = \underline{\boldsymbol{f}}$, we need to impose a set of constraints on the system that limits the number of degrees of freedom sufficiently for only one solution to exist.

We can interpret this geometrically. Without any constraints the FE mesh floats in space and can occupy infinitely many positions. To solve the system there should be only one possible position. We can ensure this by fixing some of the nodes of the mesh to predetermined positions.

To solve the system, we need to fix the displacement of at least three nodes (or a sufficient smaller number of degrees of freedom).

When some of the nodes are assigned a fixed displacement, the stiffness matrix $\boldsymbol{K}$ is changed. This change can be applied on both the element stiffness matrices and the global stiffness matrix. Only the approach for the global matrix is shown here, but the same procedure can be applied directly to the element matrices.

Without loss of generality, we assume that the displacement of the last node $P_N$ is to be fixed as $\boldsymbol{u}_N = \boldsymbol{u}_N^0 = [u_N^0, \ v_N^0, \ w_N^0]^T$. Since the procedure can be seen as three identical operations modifying rows and columns $3N-2$, $3N-1$, and $3N$ of the linear system, only the modifications of the last row and column $3N$ are described. The modified

[3] $\boldsymbol{K}^e$ is $12 \times 12$.

linear system becomes

$$\begin{bmatrix} k_{1,1} & \cdots & k_{1,3N-1} & 0 \\ \vdots & \vdots & \vdots & \vdots \\ k_{3N-1,1} & \cdots & k_{3N-1,3N-1} & 0 \\ 0 & \cdots & 0 & 1 \end{bmatrix} \underline{\boldsymbol{u}} = \begin{bmatrix} f_1 - w_N^0 k_{1,3N} \\ \vdots \\ f_{3N-1} - w_N^0 k_{3N-1,3N} \\ w_N^0 \end{bmatrix}. \tag{21}$$

The stiffness matrix is modified by setting the columns and rows corresponding to the elements of node $N$ to zero, with ones in the diagonal. The force vector is modified to reflect the fixed displacement of node $N$. Note that the force vector is not changed for fixed displacements that are zero.

## V. Simplifying the System Using Condensation

Since we built the FE model using a 3-D volumetric approach, the linear matrix system $\boldsymbol{K}\underline{\boldsymbol{u}} = \underline{\boldsymbol{f}}$ models the *volumetric* behavior of the object. The model thus includes both surface nodes as well as internal nodes of the model.

For simulation purposes, we are usually only interested in the behavior of the surface nodes, since these are the only *visible* nodes. The computations that we are performing for the internal nodes are therefore partly superfluous. It would be desirable to be able to remove these nodes from the system equation. Fortunately, this is possible using an FE technique called *condensation* [13]. This technique changes the matrix equation to yield a smaller and more compact system of equations.

The matrix equation for the condensed problem has the same size as a system developed for an FE *surface* model. But it has *exactly* the same behavior for the surface nodes as the original *volumetric* system. The condensation method only rearranges terms in the linear matrix system and does not discard information about the interaction between the remaining nodes, even across the interior nodes that have been removed from the equation.

Let us assume that the nodes of the FE model have been ordered with the surface nodes first, followed by the internal nodes. Using this ordering, the linear system can be rewritten as a block matrix system [surface (s)/internal (i)]

$$\begin{bmatrix} \boldsymbol{K}_{ss} & \boldsymbol{K}_{si} \\ \boldsymbol{K}_{is} & \boldsymbol{K}_{ii} \end{bmatrix} \begin{bmatrix} \underline{\boldsymbol{u}}_s \\ \underline{\boldsymbol{u}}_i \end{bmatrix} = \begin{bmatrix} \underline{\boldsymbol{f}}_s \\ \underline{\boldsymbol{f}}_i \end{bmatrix}. \tag{22}$$

By rearranging this block matrix system, a new linear matrix system may be derived that only involves the variables of the *surface* nodes

$$\breve{\boldsymbol{K}}_{ss}\underline{\boldsymbol{u}}_s = \breve{\underline{\boldsymbol{f}}}_s \tag{23}$$

where

$$\breve{\boldsymbol{K}}_{ss} = \boldsymbol{K}_{ss} - \boldsymbol{K}_{si}\boldsymbol{K}_{ii}^{-1}\boldsymbol{K}_{is} \tag{24}$$

$$\breve{\underline{\boldsymbol{f}}}_s = \underline{\boldsymbol{f}}_s - \boldsymbol{K}_{si}\boldsymbol{K}_{ii}^{-1}\underline{\boldsymbol{f}}_i. \tag{25}$$

The modification of the stiffness matrix in (24) transfers displacements from surface nodes to interior nodes ($\boldsymbol{K}_{is}\underline{\boldsymbol{u}}_s$) through the interior ($\boldsymbol{K}_{ii}^{-1}\boldsymbol{K}_{is}\underline{\boldsymbol{u}}_s$) to other surface nodes ($\boldsymbol{K}_{si}\boldsymbol{K}_{ii}^{-1}\boldsymbol{K}_{is}\underline{\boldsymbol{u}}_s$), thus effectively maintaining the volumetric behavior of the general model.

The displacement of the internal nodes can still be calculated using

$$\underline{\boldsymbol{u}}_i = \boldsymbol{K}_{ii}^{-1}(\underline{\boldsymbol{f}}_i - \boldsymbol{K}_{is}\underline{\boldsymbol{u}}_s). \tag{26}$$

In addition, if no forces are applied to internal nodes

$$\breve{\underline{\boldsymbol{f}}}_s\Big|_{\underline{\boldsymbol{f}}_i = \boldsymbol{o}} = \underline{\boldsymbol{f}}_s. \tag{27}$$

In this case, we can use the original force vector without modifications.

Generally, the new stiffness matrix will be dense compared to the sparse structure of the original system. But since the system is to be solved by inverting the stiffness matrix in the precalculation stage, this is not important.

## VI. Solving the Linear Matrix System

Implicit solution of a linear system is normally performed using iterated algorithms such as conjugated gradients (CG's) [1]. This algorithm performs a sparse matrix vector multiplication, three vector updates, and two inner products repeatedly in an iterative loop. The number of iterations is seldom less than five to ten and in practice cannot be predicted. Especially for a real-time simulation system, where the response must come at specific frame rates, unpredictable solution time is very unfortunate.

One of the alternatives, which is used here, is to explicitly invert the stiffness matrix. Usually this is not done when linear systems resulting from FE models are solved. The precision of the result suffers from numerical errors, and the amount of storage needed to store a dense inverted stiffness matrix is quite large compared to the sparse stiffness matrix itself. But, as indicated in the beginning of this paper, precision and memory size are relatively unimportant when compared to speed in a real-time simulation.

Although the time for inversion is considerable, the solution time is small since it only involves a dense matrix vector multiplication

$$\underline{\boldsymbol{u}} = \boldsymbol{K}^{-1}\underline{\boldsymbol{f}}. \tag{28}$$

Numerical tests have been performed using the Meschach library [19] to solve a linear system generated by a 3-D volumetric FE model. These experiments included explicit inversion, CG's with and without preconditioner, Gaussian elimination, and several factorization techniques such as QR and Cholesky. When the precalculation time was ignored, solution by matrix vector multiplication with the inverted stiffness matrix was at least ten times faster than any other method.

The actual numerical results are not provided here since the implementation of the different algorithms in the Meschach library has not been properly optimized. The specific timings therefore could be different for other implementations, although we believe the general result would be the same.

## VII. Simulation

In this section, we will explore different simulation approaches using the static FE systems developed above. We have two linear matrix equations: one containing all the nodes of the FE model and a sparse stiffness matrix; and a condensed version with only the surface nodes and a dense stiffness matrix.

The different approaches are characterized by how their behavior over time is handled.

### A. Dynamic System

To change a static system to a dynamic time-dependent system, mass and damping are added to the governing equation to account for inertia and energy dissipation. We formulate the equation for damped harmonic motion

$$\boldsymbol{M}\ddot{\underline{\boldsymbol{u}}} + \boldsymbol{D}\dot{\underline{\boldsymbol{u}}} + \boldsymbol{K}\underline{\boldsymbol{u}} = \underline{\boldsymbol{f}} \tag{29}$$

where $\boldsymbol{M}$ is the mass, $\boldsymbol{D}$ is the damping, and $\boldsymbol{K}$ is the stiffness matrix. $\boldsymbol{K}$ is calculated as shown above.

Assuming lumped masses at the nodes, we can use diagonal damping and mass element matrices

$$\boldsymbol{M}^e_{ii} = \tfrac{1}{3}\rho V^e \quad \boldsymbol{D}^e_{ii} = \alpha \boldsymbol{M}^e_{ii} \tag{30}$$

where $\rho$ is the mass density and $\alpha$ is a scaling factor.

The global element matrices are assembled into global matrices. Since the mass and damping matrices are diagonal, they are also block diagonal, and the damped harmonic motion equation for the condensed system therefore simply becomes

$$\boldsymbol{M}_{ss}\ddot{\underline{\boldsymbol{u}}}_s + \boldsymbol{D}_{ss}\dot{\underline{\boldsymbol{u}}}_s + \check{\boldsymbol{K}}_{ss}\underline{\boldsymbol{u}}_s = \check{\underline{\boldsymbol{f}}}_{ss}. \tag{31}$$

*1) Time Discretization:* By adding the continuous velocity $\dot{\underline{\boldsymbol{u}}}$ and acceleration $\ddot{\underline{\boldsymbol{u}}}$ to (29), we introduced an added difficulty to the solution process. To determine a solution, we must discretize the time-dependent variables. We use a standard finite difference technique to determine estimates of the continuous variables.

By choosing different combinations of forward and backward finite difference estimates, we can make a choice between an implicit and an explicit solution.

A semiimplicit Euler method is derived using

$$\frac{\boldsymbol{M}}{\Delta t^2}\left[\underline{\boldsymbol{u}}(t+\Delta t) - 2\underline{\boldsymbol{u}}(t) + \underline{\boldsymbol{u}}(t-\Delta t)\right] + \frac{\boldsymbol{D}}{2\Delta t}\left[\underline{\boldsymbol{u}}(t+\Delta t) - \underline{\boldsymbol{u}}(t-\Delta t)\right] + \boldsymbol{K}\underline{\boldsymbol{u}}(t+\Delta t) = \underline{\boldsymbol{f}}(t) \tag{32}$$

or $\hat{\boldsymbol{K}}\underline{\boldsymbol{u}}(t+\Delta t) = \hat{\underline{\boldsymbol{f}}}(t)$ where

$$\begin{aligned}\hat{\boldsymbol{K}} &= \frac{\boldsymbol{M}}{\Delta t^2} + \frac{\boldsymbol{D}}{2\Delta t} + \boldsymbol{K}\\ \hat{\underline{\boldsymbol{f}}}(t) &= \frac{2\boldsymbol{M}}{\Delta t^2}\underline{\boldsymbol{u}}(t) + \left(\frac{\boldsymbol{D}}{2\Delta t} - \frac{\boldsymbol{M}}{\Delta t^2}\right)\underline{\boldsymbol{u}}(t-\Delta t) + \underline{\boldsymbol{f}}(t).\end{aligned} \tag{33}$$

The significance of this method is that we have to solve a large linear matrix equation, e.g., by inverting the stiffness matrix.

The equations for a comparable explicit solution are

$$\frac{\boldsymbol{M}}{\Delta t^2}\left[\underline{\boldsymbol{u}}(t+\Delta t) - 2\underline{\boldsymbol{u}}(t) + \underline{\boldsymbol{u}}(t-\Delta t)\right] + \frac{\boldsymbol{D}}{2\Delta t}\left[\underline{\boldsymbol{u}}(t+\Delta t) - \underline{\boldsymbol{u}}(t-\Delta t)\right] + \boldsymbol{K}\underline{\boldsymbol{u}}(t) = \underline{\boldsymbol{f}}(t) \tag{34}$$

or

$$\underline{\boldsymbol{u}}(t+\Delta t) = \left(\frac{\boldsymbol{M}}{\Delta t^2} + \frac{\boldsymbol{D}}{2\Delta t}\right)^{-1} \cdot \left[2\frac{\boldsymbol{M}}{\Delta t^2}\underline{\boldsymbol{u}}(t) - \left(\frac{\boldsymbol{M}}{\Delta t^2} - \frac{\boldsymbol{D}}{2\Delta t}\right)\underline{\boldsymbol{u}}(t-\Delta t) - \boldsymbol{K}\underline{\boldsymbol{u}}(t) + \underline{\boldsymbol{f}}(t)\right]. \tag{35}$$

Since both $\boldsymbol{M}$ and $\boldsymbol{D}$ are diagonal matrices, they are easily inverted.

The main feature of the explicit solution method is that the stiffness matrix $\boldsymbol{K}$ does not have to be inverted. In practice, this allows us to split the large global equation into simple independent equations for the nodes

$$\begin{aligned}\boldsymbol{u}_q(t+\Delta t) &= \left(\frac{\rho V^e}{3\Delta t^2} + \frac{\alpha\rho V^e}{6\Delta t}\right)^{-1} \cdot \left[2\frac{\rho V^e}{3\Delta t^2}\boldsymbol{u}_q(t) - \left(\frac{\rho V^e}{3\Delta t^2} - \frac{\alpha\rho V^e}{6\Delta t}\right) \cdot \boldsymbol{u}_q(t-\Delta t) - \{\boldsymbol{K}\underline{\boldsymbol{u}}(t)\}_q + \boldsymbol{f}_q(t)\right]\\ &= \left(\frac{1}{\Delta t} + \frac{\alpha}{2}\right)^{-1} \cdot \left[\frac{2}{\Delta t}\boldsymbol{u}_q(t) - \left(\frac{1}{\Delta t} - \frac{\alpha}{2}\right)\boldsymbol{u}_q(t-\Delta t) - \frac{3\Delta t}{\rho V^e}\left(\{\boldsymbol{K}\underline{\boldsymbol{u}}(t)\}_q + \boldsymbol{f}_q(t)\right)\right]\end{aligned} \tag{36}$$

where $\{\boldsymbol{K}\underline{\boldsymbol{u}}(t)\}_q$ is the result of the matrix-vector multiplication related to the node $q$.

The explicit solution method can be compared to the mass-spring models used by Kühnapfel, Deussen, and Kuhn [10], [14], [15], and Waters and Terzopoulos [20], [22].

Mass has been lumped at the nodes, and if we regard the edges of the FE mesh as springs, we are in fact solving the corresponding mass-spring system. The nonzero elements of the rows, associated with the $q$th node of $\boldsymbol{K}$, indicate which nodes are connected to it with springs.

Of the two methods, the semiimplicit solution algorithm is preferred because of better stability and a more direct solution of the system. System response is more global with the implicit approach. Using explicit methods, the response to forces is only spread globally after some iterations.

In addition, the explicit method is considerably more unstable and can explode numerically during simulation. For arbitrary, but still realistic, forces it is generally difficult to predict whether or not the system will stay stable. The general solution to this problem is to make the time steps very small and thus demand many more iterations per frame. The semiimplicit system is much more stable and can take longer time steps.

A drawback with the semiimplicit models is the greater difficulty when modeling topological changes in the FE mesh as the result of cuts, e.g., for modeling surgical incisions. Any change in the stiffness matrix implies that the inverted stiffness matrix has to be updated or recalculated. This is possible but can be quite expensive (see Section X-B). The explicit model is much easier to change in response to cuts since only localized changes to the stiffness matrix are needed.

Explicit and semiimplicit models can be combined by using domain decomposition techniques. The domain is split into smaller subdomains, which use explicit or semiimplicit solution methods, depending on whether or not cuts have been made. See Section X-A for a discussion of domain decomposition.

Although the discussion above has been carried out using the full system, the semiimplicit equations could also be used for the condensed system. The explicit equations could be formulated, but it does not make sense to do so since the condensed system has a dense stiffness matrix. The explicit system depends on the sparse nature of the stiffness matrix to be effective.

### B. Static System and Selective Matrix Vector Multiplication

Generally the force vector $\hat{\underline{\boldsymbol{f}}}(t)$ of the semiimplicit system is a full vector because of the contribution from the previous displacement vectors $\underline{\boldsymbol{u}}(t)$ and $\underline{\boldsymbol{u}}(t-\Delta t)$. In contrast, the original force vector $\underline{\boldsymbol{f}}(t)$ is a sparse vector when forces only are applied to a limited part of the surface. Since this is often the case in simulation, we were inspired to develop an alternative simulation method, which for sparse force vectors is considerably faster.

The idea is to use the static linear system $\boldsymbol{K}\underline{\boldsymbol{u}} = \underline{\boldsymbol{f}}$ (or the condensed version) instead of the dynamic system and exploit the sparse structure of the static force vector.

The cost of this simulation method is the loss of dynamics. Is that a problem? It depends on the nature of the object that is being modeled. If applying forces to the object should result in subsequent vibrations, such as with a gel-like substance, dynamics are very important. But for most materials, the vibrations are negligible and the static solution looks quite realistic.

Formally, solving the system using the inverted stiffness matrix is performed using

$$\underline{\boldsymbol{u}} = \boldsymbol{K}^{-1}\underline{\boldsymbol{f}}. \tag{37}$$

If only a few positions of the force vector are nonzero, clearly standard matrix vector multiplication would involve a large number of superfluous multiplications. Note that

$$\underline{\boldsymbol{u}} = \boldsymbol{K}^{-1}\underline{\boldsymbol{f}} = \sum_i \boldsymbol{K}_{*i}^{-1}\underline{f}_i \tag{38}$$

where $\boldsymbol{K}_{*i}^{-1}$ is the $i$th column vector of $\boldsymbol{K}^{-1}$ and $\underline{f}_i$is the $i$th element of $\underline{\boldsymbol{f}}$. Since the majority of the $\underline{f}_i$ are zero, $i$ is restricted to run through only the positions of $\underline{\boldsymbol{f}}$ for which $\underline{f}_i \neq 0$. If $n$ of the $N$ positions in $\underline{\boldsymbol{f}}$ are nonzero, this will reduce the complexity to $O(n/N)$ times the time of a normal matrix vector multiplication. We call this approach *selective matrix vector multiplication* (SMVM).

In practice, whenever forces are applied to our object, a short list of the nodes with nonzero forces is maintained. The resulting deformation is determined by running through the list and using (38) to add the contributions of the nodes together.

**Table 1** Summary of Possible Modeling Methods

| | Dynamic simulation | | SMVM |
|---|---|---|---|
| | Explicit | Semiimplicit | |
| Full FE model | FX | FI | FS |
| Condensed FE model | na. | CI | CS |

### C. Summary of Simulation Methods

Several methods related to simulation using FE models have been presented. In general, two criteria separate the possible algorithms: full FE model versus condensed FE model and dynamic simulation (explicit/semiimplicit) versus SMVM (see Table 1). The choice of algorithm depends very much on the requirements of the application.

The choice between a full or a condensed model is determined by whether you need to change the topology of the model during the simulation and therefore change the stiffness matrix. This could happen in response to incisions and cuts in the model. Since the condensed stiffness matrix is one step further in refinement than the standard stiffness matrix, it is more difficult to change.

The choice between semiimplicit models and SMVM models is generally a choice based on the size of the problem and the available compute power. It is most realistic to use the semiimplicit dynamic model, but the static SMVM model is considerably faster.

Explicit dynamic modeling is used whenever changes to the model have to be made on the fly, since it is the only model allowing modifications easily.

In some cases, more than one model can be used for the same object when the conditions change during the simulation. As an example, consider the abdomen of a patient.

- Initially the surgeon only examines the abdomen. This involves touching organs and deforming them by pushing with either hands or instruments.
- After finishing the initial exploration, the surgeon decides to make an incision in *a few* of the organs.

Most of the organs in the abdomen are therefore only deformed, and no incisions are made in them. It would be natural to choose a condensed model for these organs, possibly using the SMVM model for the less important organs and the semiimplicit model for the important ones.

For the organ in which the surgeon is going to make the incision, we would use the explicit full model. But initially during the examination, we could also use the condensed semiimplicit model. An adaptive simulation system would

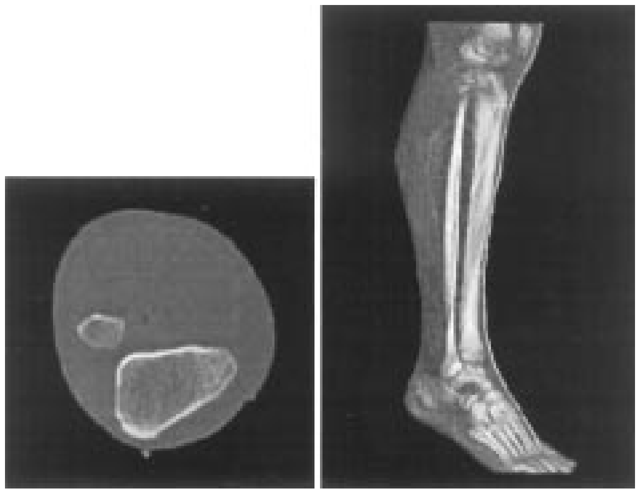


**Fig. 5.** Voxel data from the Visible Human data set.

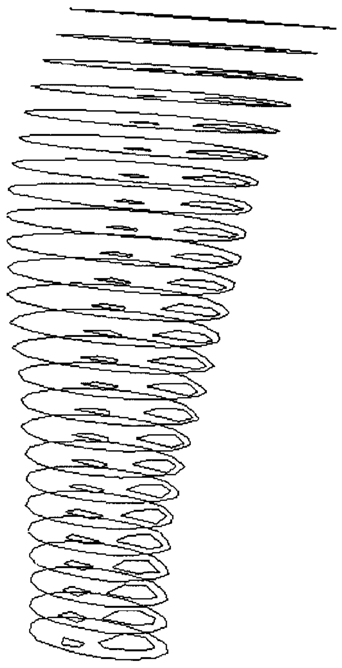


**Fig. 6.** Contours created using Mvox.

use the condensed model initially for all objects and change to the explicit model for an organ when the surgeon starts making incisions in it.

The CS, CI, and FS methods (see Table 1) have been implemented in the simulation system described below. Experiments with the FX method have been carried out in other contexts.

## VIII. Simulation System

This section briefly discusses practical aspects of implementing the simulation methods and generating models from voxel data. Screen shots from a Silicon Graphics (SGI) Performer-based system illustrate the results.

### *A. Mesh Generation Using Mvox and NUAGES*

In addition to a range of simple box-like structures, data from the Visible Human Project [21] have been used to make a model of a lower leg.

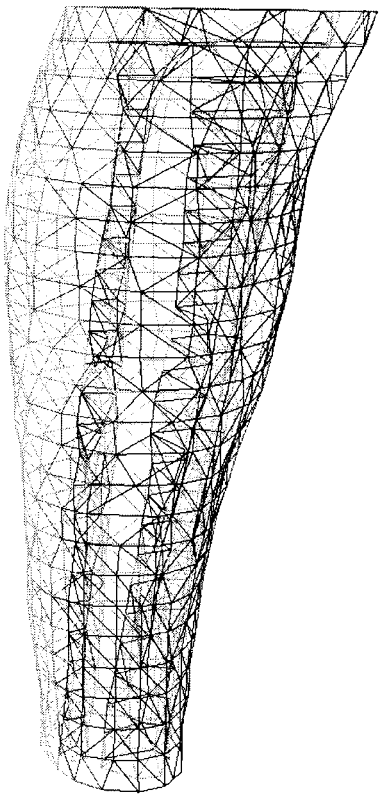


**Fig. 7.** FE mesh created from contours using NUAGES.

Since the Visible Human data set is voxel based (see Fig. 5), it must first be turned into a mesh model. The Mvox software [3] was used to draw contours manually on the boundary of the skin and bone in the voxel data (see Fig. 6). The NUAGES software [11] was then applied to create a 3-D tetrahedral mesh model of the leg. The result for the lower leg is shown in Fig. 7. This model is used in the simulation system described next.

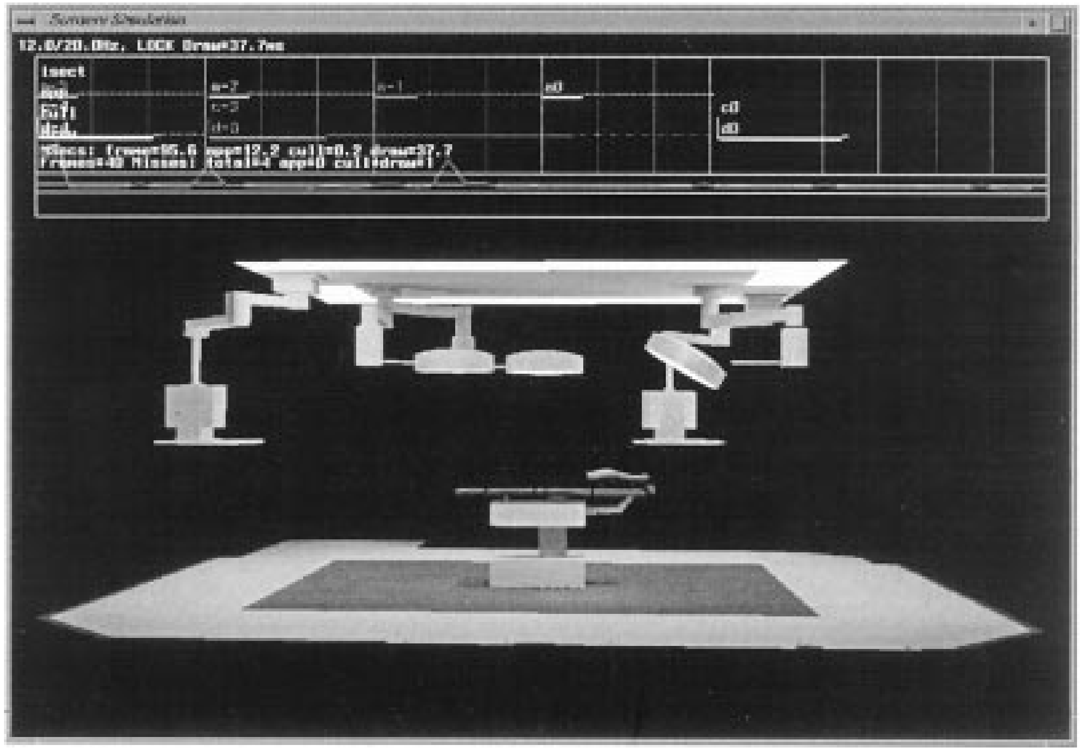
### *B. SGI Performer Parallel Pipelineing System*

The simulation system has been implemented on a Silicon Graphics ONYX with four MIPS R4400 processors using the SGI Performer 1.2 graphics library. SGI Performer allows the programmer to create parallel pipelineing software quite easily by providing the basic tools for communication, shared memory, etc.

Currently, the system runs with three processes (see Fig. 9): the application, culling, and drawing processes. The *application* process handles the actual simulation of the deformable solid, i.e., calculates displacements, etc. The *culling* process analyzes the scene that the simulation process provides and determines which parts are visible in the current window. It then pipes the visible parts to the *drawing* process, which finally renders the scene.

Note that although the entire system is a parallel system, the actual deformation processing still runs on a single processor. The parallel features are only used to separate rendering from simulation.

Fig. 8 shows a screen dump of the virtual surgery room with the leg lying on the operating table. Fig. 10 shows the surface of the FE mesh in the simulator.



```
12.0/20.0Hz, LOCK Draw=37.7ms
MSecs: frame=85.6 app=12.2 cull=0.2 draw=37.7
Frames=40 Misses: total=4 app=0 cull+draw=1
```

**Fig. 8.** Simulation system implemented using SGI Performer.

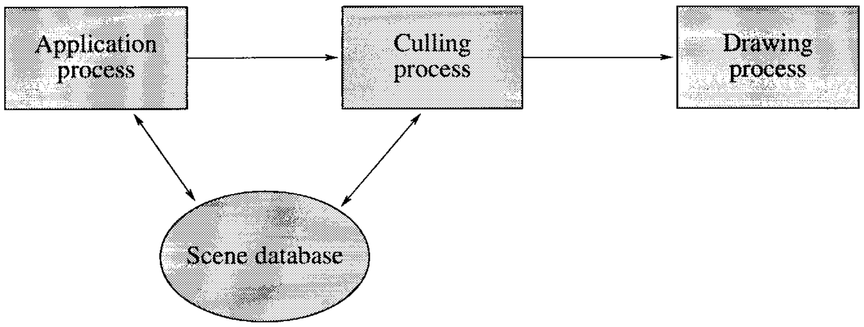


**Fig. 9.** Diagram showing relationship between the processes in the simulation system.

## IX. Results

This paper has described five methods for real-time simulation of elastic deformation of a volumetric solid based on linear elastic finite elements. Examples of deformation (semiimplicit condensed) of a lower leg are shown in Figs. 11 and 12.

Performance of the semiimplicit dynamic simulation methods is determined solely by the size of the linear system. A performance of 20 frames/s has been achieved for models with up to 250 nodes in the system equation. For the full linear system, this includes all nodes, both internally and on the surface. For the condensed system, it only includes the surface nodes. The number of internal nodes of the model does not matter for the condensed system since they have been removed from the system equation.

It is more difficult to predict the performance of the methods using SMVM. The above comments concerning the full versus condensed systems apply here also. In addition, the number of nodes that have forces applied to them is a very important factor.

The example using a leg from the Visible Human data set with 700 system nodes (condensed system with only surface nodes) ran comfortably using only one-third of a frame (20 frames/s) when forces were applied to three nodes. This included calculation of the deformation and also basic processing. So, although both more nodes and more surface nodes with forces applied would increase the time, larger models could be accommodated using the SMVM method.

As a group, we call the optimized models FFE's.

## X. Future Extensions

This section describes two extensions to the basic FFE algorithms that could be used in larger practical systems. Full implementation of these extensions requires considerable data base handling facilities and access to parallel systems with several processors.

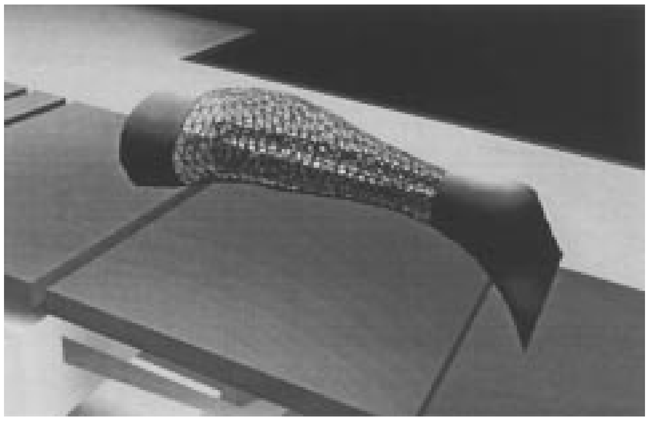
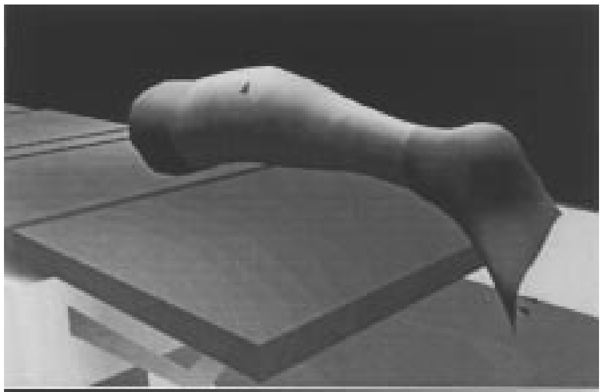


**Fig. 10.** Wire-frame model of lower leg in simulator.

(a)



(b)

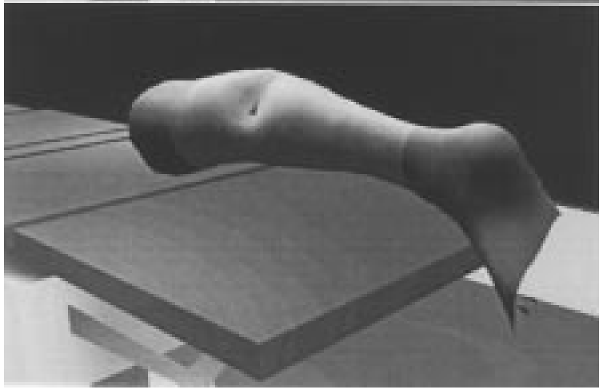


**Fig. 11.** Simulation of pushing on the lower leg. (a) Default shape. (b) Deformation of leg when a push is applied to the black triangle.

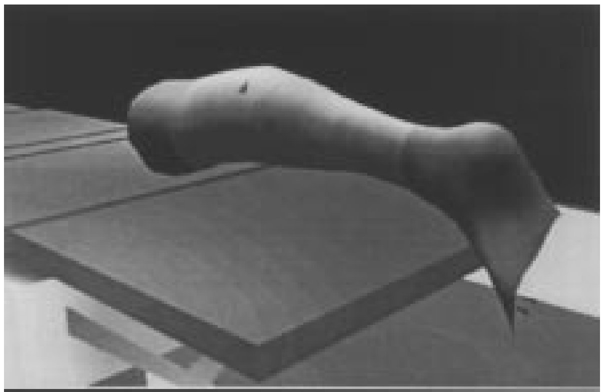
### A. Domain Decomposition

The computational complexity of the finite element models is generally high. Although computational improvements have been presented in this paper, there is still a low limit on the possible size of the deformable model if real-time response is necessary.

Parallel processors are starting to become generally available on, e.g., the Silicon Graphics ONYX series of high-end graphics computers. A natural suggestion is therefore to create parallel versions of the FE models.

Explicit models are easy to parallelize, since the solution is determined locally. In principle, each node can be assigned to one processor in the parallel computer.

(a)



(b)

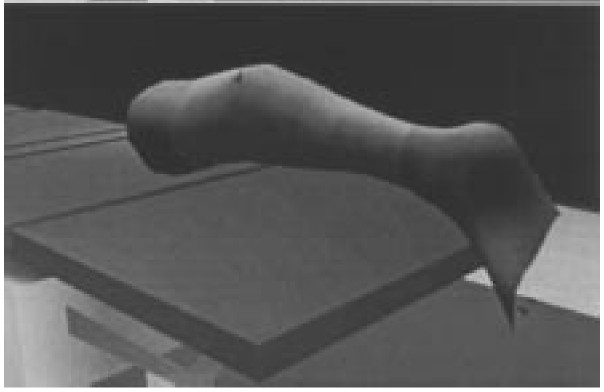


**Fig. 12.** Simulation of pulling on the lower leg. (a) Default shape. (b) Deformation of leg when a pull is applied to the black triangle.

Creating parallel versions of implicit models is more difficult. Fortunately, there is currently a great deal of interest in these problems. *Domain decomposition* techniques are being developed with the specific aim of implementing FE models on parallel systems. Although the field is still in the early stages, there are reasonable methods available.

In the following, the basic idea of domain decomposition is described. Domain decomposition methods can be quite complex, and it is not clear which method should be used for real-time deformable models. A more detailed description is therefore not given here. Refer to the recent book by Smith *et al.* [18] for a more in-depth discussion.

Let us assume that the domain $\Omega$ can be separated into nonoverlapping subdomains $\Omega^{(i)}$ with boundaries $\Gamma^{(i)}$ (see Fig. 13). Adjacent domains $i$ and $j$ only share the common boundary $\Gamma^{(i)} \cap \Gamma^{(j)}$.

Let the linear system for subdomain $i$ be

$$\boldsymbol{K}^{(i)}\underline{\boldsymbol{u}}^{(i)} = \underline{\boldsymbol{f}}^{(i)}. \tag{39}$$

Using the same approach as used for the condensation, the nodes of the subdomain are split into interior ($i$) and common ($c$) nodes. The common nodes are those shared by two or more subdomains.

With these definitions, the equilibrium matrix equation for the subdomain can be written as

$$\begin{bmatrix} \boldsymbol{K}_{cc}^{(i)} & \boldsymbol{K}_{ci}^{(i)} \\ \boldsymbol{K}_{ic}^{(i)} & \boldsymbol{K}_{ii}^{(i)} \end{bmatrix} \begin{bmatrix} \underline{\boldsymbol{u}}_c^{(i)} \\ \underline{\boldsymbol{u}}_i^{(i)} \end{bmatrix} = \begin{bmatrix} \underline{\boldsymbol{f}}_c^{(i)} \\ \underline{\boldsymbol{f}}_i^{(i)} \end{bmatrix}. \tag{40}$$

From this block matrix system, a new linear matrix system may be derived, which only involves the variables of the

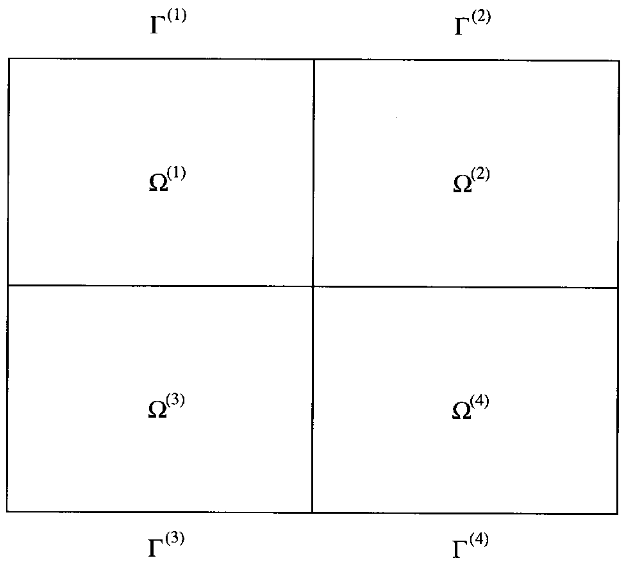


**Fig. 13.** Domain decomposition.

common nodes

$$\breve{\boldsymbol{K}}_{cc}^{(i)}\underline{\boldsymbol{u}}_c^{(i)} = \underline{\breve{\boldsymbol{f}}}_c^{(i)} \tag{41}$$

where

$$\breve{\boldsymbol{K}}_{cc}^{(i)} = \boldsymbol{K}_{cc}^{(i)} - \boldsymbol{K}_{ci}^{(i)}\boldsymbol{K}_{ii}^{(i)-1}\boldsymbol{K}_{ic}^{(i)} \tag{42}$$

$$\underline{\breve{\boldsymbol{f}}}_c^{(i)} = \underline{\boldsymbol{f}}_c^{(i)} - \boldsymbol{K}_{ci}^{(i)}\boldsymbol{K}_{ii}^{(i)-1}\underline{\boldsymbol{f}}_i^{(i)}. \tag{43}$$

The subdomain variables are assembled into a global system

$$\breve{\boldsymbol{K}}_{cc}\underline{\boldsymbol{u}}_c = \underline{\breve{\boldsymbol{f}}}_c \tag{44}$$

where $\breve{\boldsymbol{K}}_{cc}$ is called the *Schur's complement*.

When a solution has been found for the common nodes, the displacements of the internal nodes in the subdomain can be calculated using

$$\underline{\boldsymbol{u}}_i^{(i)} = \boldsymbol{K}_{ii}^{(i)-1}\left[\underline{\boldsymbol{f}}_i^{(i)} - \boldsymbol{K}_{ic}^{(i)}\underline{\boldsymbol{u}}_c^{(i)}\right]. \tag{45}$$

In theory, the following approach could be used to solve the global matrix equation using domain decomposition (*parallel* indicates that the step can be carried out in parallel).

1) The common forces are determined locally in each subdomain using (43) (*parallel*).
2) These forces are assembled into a global force vector, which is used to find the displacements of the common nodes using (44).
3) When the displacements of the common nodes are found, the displacements of the internal nodes can be determined using (45) (*parallel*).

Normal domain decomposition methods do not use this approach directly. Instead, iterated conjugated gradients methods [1] are often used with preconditioners based on the Schur's complement.

The actual application will determine what form of domain decomposition algorithm should be used. In the case of cutting, the Schur's complement is changed, and iterated methods are probably necessary.

Experiments have been carried out to solve the global linear system

$$\boldsymbol{K}\underline{\boldsymbol{u}} = \underline{\boldsymbol{f}} \tag{46}$$

by applying domain decomposition based on the conjugated gradients algorithm.

First, a preconditioner is created from the global stiffness matrix by zeroing the off-diagonal elements, which are related to the common nodes. Since the subdomain blocks of the preconditioner are now independent, the application of the preconditioner in the conjugate gradient algorithm can be carried out in parallel by subdomain processors. The remaining part of the conjugate gradient algorithm is global and can be performed on a specially assigned processor.

Preliminary results have been obtained showing the validity of the algorithm. The conjugate gradient algorithm is slow, however, and the solution of the global system is not fast enough for real-time performance. Further work is necessary to determine an appropriate domain decomposition algorithm.

## B. Cutting in FE Systems

Implementing cutting in FE systems causes two major problems. The first, which we ignore here, is related to the geometric modification of the FE mesh. These modifications should ensure that the cut looks smooth when rendered using computer graphics. In general, it is necessary to refine the mesh around the cut, and the geometric aspects of this are nontrivial.

The second problem concerns the necessary change of the stiffness matrix and the linear system. For explicit methods, this is simple. But for implicit systems, where the stiffness matrix has been inverted, the inverted stiffness matrix needs to be updated.

For simplicity, let us assume that the cutting procedure implies removing the FE number $e$ from the system. The modified stiffness matrix becomes

$$\boldsymbol{K} = \boldsymbol{K} - \text{global}(\boldsymbol{K}^e). \tag{47}$$

Since the size of the element stiffness matrix is $12 \times 12$, this involves changes in 12 rows and 12 columns.

To change the inverted stiffness matrix, the Woodbury formula [16] is used.

Let us assume that the stiffness matrix is updated with a matrix that can be written as the outer product of two $3N\times$ 12 vectors ($N$ is the number of nodes)

$$\boldsymbol{K} \rightarrow \boldsymbol{K} + \boldsymbol{U}\boldsymbol{V}^T. \tag{48}$$

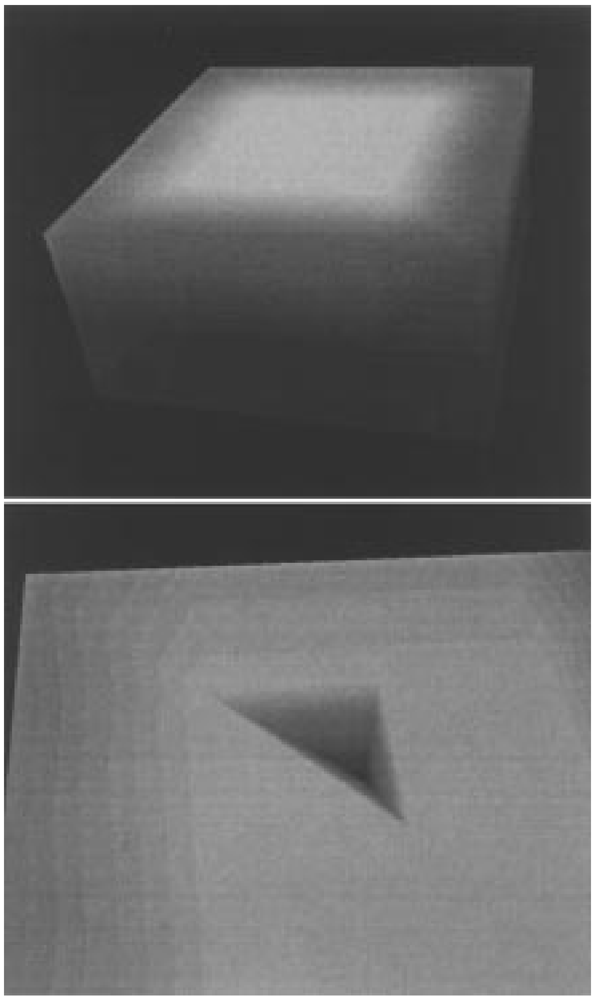


**Fig. 14.** Simple cutting in FE mesh by removing one finite element.

The updated inverted stiffness matrix then becomes

$$(\boldsymbol{K}+\boldsymbol{U}\boldsymbol{V}^T)^{-1} = \boldsymbol{K}^{-1} \tag{49}$$

$$- \left[\boldsymbol{K}^{-1}\boldsymbol{U}(\boldsymbol{I}_{12\times 12}+\boldsymbol{V}^T\boldsymbol{K}^{-1}\boldsymbol{U})^{-1} \cdot \boldsymbol{V}^T\boldsymbol{K}^{-1}\right] \tag{50}$$

where $\boldsymbol{I}_{12\times 12}$ is a 12 by 12 identity matrix.

The thin matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ are easily determined: $\boldsymbol{K}^e$ and the identity matrix are inserted in the rows of $\boldsymbol{U}$ and $\boldsymbol{V}$, respectively, corresponding to the global row/column positions of the element nodes.

The computation consists of the inversion of the 12 by 12 matrix

$$\boldsymbol{I}_{12\times 12}+\boldsymbol{V}^T\boldsymbol{K}^{-1}\boldsymbol{U} \tag{51}$$

as well as six matrix multiplications of different sizes and two matrix additions—in total, $O(N^2)$ operations.

To demonstrate the applicability of this approach, a simple cutting algorithm has been implemented in the simulation system. This algorithm allows individual FE's to be removed. The resulting modification of the linear system is obtained using the algorithm described above.

An example is shown in Fig. 14. This simple FE model has only 75 nodes, and the modification of the linear system is more or less instant. For a more complex mesh with 1125 nodes, the modification of the inverted stiffness matrix takes about one minute. The time consumption is, consequently, too high for general real-time response. But it could be used in specific cases where the waiting time can be accepted or otherwise disguised.

## XI. Conclusion

In this paper, a comprehensive description of the issues facing a developer of real-time deformable FE models has been given. In particular, methods using condensation techniques, direct matrix inversion, and selective matrix vector multiplication were presented. These methods are essential to obtaining acceptable response in surgery simulation systems.

A simulation system developed using SGI Performer was described, and a routine for creating the basic FE mesh was suggested.

Two important extensions of the work were outlined in the final section. These extensions are the use of domain decomposition for parallel implementation of the FE system and a procedure for modification of the system matrix in response to incisions and cuts in the mesh. A simple example of a cut was demonstrated, in which a single FE was removed from the stiffness matrix and a hole appeared.

The implementations of the two extensions are nontrivial and require comprehensive bookkeeping and computer power to work in practice. Both extensions require improvements before they can be used in practice.

In general, there is still a considerable amount of work to be performed. One important issue is the introduction of detailed segmentations of the organs, limbs, etc. to allow different material properties and models to be used.

## Appendix
## Linear Tetrahedral Finite Element

We assume that the nodes of the tetrahedron have been numbered as illustrated in Fig. 4. The *natural coordinates* $L_1$, $L_2$, $L_3$, and $L_4$ of the tetrahedron are related to the

$$\boldsymbol{B}^e = \frac{1}{6V}\begin{bmatrix} b_1 & 0 & 0 & b_2 & 0 & 0 & b_3 & 0 & 0 & b_4 & 0 & 0 \\ 0 & c_1 & 0 & 0 & c_2 & 0 & 0 & c_3 & 0 & 0 & c_4 & 0 \\ 0 & 0 & d_1 & 0 & 0 & d_2 & 0 & 0 & d_3 & 0 & 0 & d_4 \\ c_1 & b_1 & 0 & c_2 & b_2 & 0 & c_3 & b_3 & 0 & c_4 & b_4 & 0 \\ 0 & d_1 & c_1 & 0 & d_2 & c_2 & 0 & d_3 & c_3 & 0 & d_4 & c_4 \\ d_1 & 0 & b_1 & d_2 & 0 & b_2 & d_3 & 0 & b_3 & d_4 & 0 & b_4 \end{bmatrix} \tag{55}$$

global coordinates $x$, $y$, and $z$ by (we ignore element superscripts)

$$\begin{bmatrix}1\\x\\y\\z\end{bmatrix} = \begin{bmatrix}1 & 1 & 1 & 1\\ x_1 & x_2 & x_3 & x_4\\ y_1 & y_2 & y_3 & y_4\\ z_1 & z_2 & z_3 & z_4\end{bmatrix}\begin{bmatrix}L_1\\L_2\\L_3\\L_4\end{bmatrix}. \tag{52}$$

This equation can be inverted to give

$$\begin{bmatrix}L_1\\L_2\\L_3\\L_4\end{bmatrix} = \frac{1}{6V}\begin{bmatrix}a_1 & b_1 & c_1 & d_1\\ a_2 & b_2 & c_2 & d_2\\ a_3 & b_3 & c_3 & d_3\\ a_4 & b_4 & c_4 & d_4\end{bmatrix}\begin{bmatrix}1\\x\\y\\z\end{bmatrix} \tag{53}$$

where

$$6V = \begin{vmatrix}1 & 1 & 1 & 1\\ x_1 & x_2 & x_3 & x_4\\ y_1 & y_2 & y_3 & y_4\\ z_1 & z_2 & z_3 & z_4\end{vmatrix} \quad a_1 = \begin{vmatrix}x_2 & x_3 & x_4\\ y_2 & y_3 & y_4\\ z_2 & z_3 & z_4\end{vmatrix}$$
$$b_1 = -\begin{vmatrix}1 & 1 & 1\\ y_2 & y_3 & y_4\\ z_2 & z_3 & z_4\end{vmatrix} \quad c_1 = \begin{vmatrix}1 & 1 & 1\\ x_2 & x_3 & x_4\\ z_2 & z_3 & z_4\end{vmatrix}$$
$$d_1 = -\begin{vmatrix}1 & 1 & 1\\ x_2 & x_3 & x_4\\ y_2 & y_3 & y_4\end{vmatrix}. \tag{54}$$

The other coefficients are found by cyclic interchange of the indexes.

The $\boldsymbol{B}^e$ matrix becomes (55), shown at the bottom of the preceding page.

## References

[1] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. M. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst, Templates for the solution of linear systems: Building blocks for iterative methods. (1995). [Online]. Available: http://www.netlib.org/templates/templates.ps.

[2] M. Bro-Nielsen and S. Cotin, "Soft tissue modeling in surgery simulation for prediction of results of craniofacial operations & steps toward virtual reality training systems," in *Proc. 3rd Int. Workshop Rapid Prototyping in Medicine & Computer-Assisted Surgery,* Erlangen, Germany, Oct. 19–21, 1995, p. 35.

[3] M. Bro-Nielsen, "Mvox: Interactive 2–4D medical image and graphics visualization software," in *Proceedings of Computer Assisted Radiology (CAR'96),* H. U. Lemke, M. W. Vannier, K. Inamura, and A. G. Farman, Eds. Amsterdam, The Netherlands: Elsevier, 1996, pp. 335–338.

[4] M. Bro-Nielsen and S. Cotin, "Real-time volumetric deformable models for surgery simulation using finite elements and condensation," in *Proc. Computer Graphics Forum, Eurographics'96,* 1996, vol. 15, no. 3, pp. 57–66.

[5] M. Bro-Nielsen, "Medical image registration and surgery simulation," Ph.D. dissertation, Department of Mathematical Modeling, Technical University of Denmark, 1997.

[6] P. G. Ciarlet, *Mathematical Elasticity Vol. 1: Three-Dimensional Elasticity.* Amsterdam, The Netherlands: North-Holland, 1987.

[7] S. Cotin, H. Delingette, M. Bro-Nielsen, N. Ayache, J. M. Clément, V. Tassetti, and J. Marescaux, "Geometric and physical representations for a simulator of hepatic surgery," in *Proc. Medicine Meets Virtual Reality,* San Diego, CA, Jan. 17–20, 1996, pp. 139–151.

[8] S. Cotin, H. Delingette, J. M. Clement, V. Tassetti, J. Marescaux, and N. Ayache, "Volumetric deformable models for simulation of laparoscopic surgery," in *Proceedings of Computer Assisted Radiology (CAR'96),* H. U. Lemke, M. W. Vannier, K. Inamura, and A. G. Farman, Eds. Amsterdam, The Netherlands: Elsevier, 1996, pp. 793–798.

[9] S. A. Cover, N. F. Ezquerra, J. F. O'Brien, R. Rowe, T. Gadacz, and E. Palm, "Interactively deformable models for surgery simulation," *IEEE Comput. Graphics Applicat. Mag.,* pp. 68–75, Nov. 1993.

[10] O. Deussen, L. Kobbelt, and P. Tücke, "Using simulated annealing to obtain good approximations of deformable bodies," in *Proceedings of the EuroGraphics Workshop Computer Animation and Simulation,* D. Terzopoulos and D. Thalmann, Eds. New York: Springer-Verlag, 1995.

[11] B. Geiger, "Three-dimensional modeling of human organs and its application to diagnosis and surgical planning," Institut National de Recherche en Informatique et en Automation, France, INRIA Tech. Rep. 2105, Dec. 1993.

[12] K. H. Huebner, *The Finite Element Method for Engineers.* New York: Wiley, 1975.

[13] H. Kardestuncer, *Finite Element Handbook.* New York: McGraw-Hill, 1987.

[14] U. G. Kühnapfel and B. Neisius, "Realtime graphical computer simulation for endoscopic surgery," in *Proc. Medicine Meets Virtual Reality II,* San Diego, CA, Jan. 27–30, 1994.

[15] C. Kuhn, U. Kühnapfel, H.-G. Krumm, and B. Neisius, "A 'virtual reality' based training system for minimally invasive surgery," in *Proceedings of Computer Assisted Radiology (CAR'96),* H. U. Lemke, M. W. Vannier, K. Inamura, and A. G. Farman, Eds. Amsterdam, The Netherlands: Elsevier, 1996, pp. 764–769.

[16] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C.* Cambridge, U.K.: Cambridge Univ. Press, 1992.

[17] M. A. Sagar, D. Bullivant, G. D. Mallinson, P. J. Hunter, and I. Hunter, "A virtual environment and model of the eye for surgical simulation," in *Proc. ACM SIGGRAPH'94,* Orlando, FL, pp. 205–212.

[18] B. Smith, P. Bjorstad, and W. Gropp, *Domain Decomposition: Parallel Multilevel Methods for Elliptic PDES.* Cambridge, U.K.: Cambridge Univ. Press, 1996.

[19] D. E. Stewart. Meschach: Matrix computations in C. (1992). [Online]. Available FTP: ftpmaths.anu.edu.au/pub/meschach/meschach.html.

[20] D. Terzopoulos and K. Waters, "Techniques for realistic facial modeling and animation," in *Computer Animation.* Berlin: Springer, 1991, pp. 59–73.

[21] National Institutes of Health. The Visible Human Project. (1996). [Online]. Available: http://www.nlm.nih.gov/research/visible/visible_human.html.

[22] K. Waters, "A muscle model for animating three-dimensional facial expression," *Comput. Graphics,* vol. 21, no. 4, pp. 17–24, 1987.

**Morten Bro-Nielsen** (Member, IEEE) received the M.Sc. and Ph.D. degrees in engineering from the Technical University of Denmark in 1992 and 1996, respectively.

As part of the Ph.D. degree, he spent nine months in 1994–1995 at the Epidaure group of the Institut National de Recherche en Informatique et en Automation (INRIA) in France. From 1988 to 1992, he was with IBM Denmark A/S. In 1992–1993, he visited the Toshiba Kansai Research Laboratory in Japan. His research there focused on developing three-dimensional (3-D) computer graphics models based on two-dimensional photographs. From 1995 to 1996, he was with the 3D-Lab at the School of Dentistry, University of Copenhagen, Denmark. During this period, he worked on medical image registration, surgery simulation, and 3-D visualization. In addition, he coordinated the medical imaging research collaboration between the Technical University of Denmark, the School of Dentistry, and the National University Hospital of Denmark. Since 1996, he has been a Senior Scientist with HT Medical, Inc., Rockville, MD. His current interests are surgery simulation, computer graphics, real-time and parallel computing, medical image registration, and 3-D visualization.

Dr. Bro-Nielsen is a member of the Association of Computing Machinery Special Interest Group on Graphics (SIGGRAPH) and EuroGraphics.