Physically Based Deformable Models in Computer Graphics

Andrew Nealen[1], Matthias Müller[2,3], Richard Keiser[3], Eddy Boxerman[4] and Mark Carlson[5]

[1] Discrete Geometric Modeling Group, TU Darmstadt
[2] NovodeX / AGEIA
[3] Computer Graphics Lab, ETH Zürich
[4] Department of Computer Science, University of British Columbia
[5] DNA Productions, Inc.

Abstract

Physically based deformable models have been widely embraced by the Computer Graphics community. Many problems outlined in a previous survey by Gibson and Mirtich [GM97] have been addressed, thereby making these models interesting and useful for both offline and real-time applications, such as motion pictures and video games. In this paper, we present the most significant contributions of the past decade, which produce such impressive and perceivably realistic animations and simulations: finite element/difference/volume methods, mass-spring systems, meshfree methods, coupled particle systems and reduced deformable models based on modal analysis. For completeness, we also make a connection to the simulation of other continua, such as fluids, gases and melting objects. Since time integration is inherent to all simulated phenomena, the general notion of time discretization is treated separately, while specifics are left to the respective models. Finally, we discuss areas of application, such as elastoplastic deformation and fracture, cloth and hair animation, virtual surgery simulation, interactive entertainment and fluid/smoke animation, and also suggest areas for future research.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Physically Based Modeling I.3.7 [Computer Graphics]: Animation and Virtual Reality

1. Introduction

Physically based deformable models have two decades of history in Computer Graphics: since Lasseter's discussion of *squash and stretch* [Las87] and, concurrently, Terzopoulos et. al's seminal paper on elastically deformable models [TPBF87], numerous researchers have partaken in the quest for the visually and physically plausible animation of deformable objects and fluids. This inherently interdisciplinary field elegantly combines newtonian dynamics, continuum mechanics, numerical computation, differential geometry, vector calculus, approximation theory and Computer Graphics (to name a few) into a vast and powerful toolkit, which is being further explored and extended. The field is in constant flux and, thus, active and fruitful, with many visually stunning achievements to account for.

Since Gibson and Mirtich's survey paper [GM97], the field of physically based deformable models in Computer Graphics has expanded tremendously. Significant contributions were made in many key areas, e.g. object modeling, fracture, plasticity, cloth animation, stable fluid simulation, time integration strategies, discretization and numer-

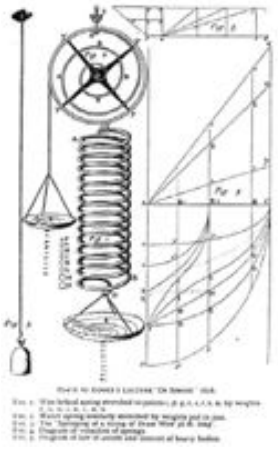

Figure 1: *Hooke's Law, from* De Potentia Restitutiva [1678].

ical solution of PDEs, modal analysis, space-time adaptivity, multiresolution modeling and real-time simulation. Non-physical models, such as parametric curves and surfaces and free-form deformations, are not discussed in this report. The inclined reader is therefore encouraged to browse more recent literature on t-splines [SZBN03] [SCF*04], space-warping [MJBF02] [LKG*03] [BK05] and methods based on differential surface properties [SLCO*04] [YZX*04] [BK04] [LSLCO05] [NSACO05] [IMH05]. For advances in character skinning see e.g. [WP02] [KJP02] [JT05]. Since we are not able to cover basic elasticity theory and continuum mechanics in this report, we would like to point out that a nice review of the history of elasticity theory, starting with the discovery of Hooke's Law in 1660 (Fig. 1) and leading up to the general equations of Navier in 1821, is given in [Lov27]. Furthermore, great introductions to continuum mechanics and dynamics can be found in [WB97] and in general textbooks, such as [Chu96] [Coo95] [BW97] [Gdo93] [BLM00]. For application specific presentations, we refer the reader to a number of recent works. For cloth simulation, there is the text by House and Breen [HB00], as well as the recent, extensive tutorial by Thalmann et al. [MTCK*04]. For hair simulation, there is the (already slightly dated) overview by Thalmann et al. [MTHK00]; the paper by Volino and Thalmann [VMT04] gives a good, more recent overview. Collision detection and haptic force-feedback rendering for deformable objects are other challenging and active areas of research. For a summary of recent work in these fields, we refer the reader to the report by Teschner et al. [TKH*05] and the course notes of Lin and Otaduy [LO05].

In this report we take a *model based* point of view, motivated by the fact that there are many readily available physical models for very similar applications, i.e. we can animate an elastically or plastically deforming solid with many different underlying models, such as mass-spring systems, finite elements or meshfree methods. We furthermore make a distinction between *Lagrangian* methods, where the model consists of a set of points with varying locations and properties, and *Eulerian* methods, where model properties are computed for a set of stationary points. To give a coarse overview, we describe recent developments for

- Lagrangian Mesh Based Methods
  - Continuum Mechanics Based Methods
  - Mass-Spring Systems
- Lagrangian Mesh Free Methods
  - Loosely Coupled Particle Systems
  - Smoothed Particle Hydrodynamics (SPH)
  - Mesh Free Methods for the solution of PDEs
- Reduced Deformation Models and Modal Analysis
- Eulerian and Semi-Lagrangian Methods
  - Fluids and Gases
  - Melting Objects

In each section we present the basic model formulation, recent contributions, benefits and drawbacks, and various areas of application. The section on fluids, gases and melting objects contains an overview of recent work and establishes the connection to the field of physically based deformable models. A complete survey on the animation of fluids and gases would easily fill its own report and is therefore beyond our scope.

Our goal is to provide an up-to-date report to the Computer Graphics community, as an entry point for researchers and developers who are new to the field, thereby complementing the existing survey paper [GM97].

2. Background

2.1. Continuum Elasticity

A deformable object is typically defined by its undeformed shape (also called equilibrium configuration, rest or initial shape) and by a set of material parameters that define how it deforms under applied forces. If we think of the rest shape as a continuous connected subset M of $\mathbb{R}^3$, then the coordinates $\mathbf{m} \in M$ of a point in the object are called *material coordinates* of that point. In the discrete case M is a discrete set of points that sample the rest shape of the object.

When forces are applied, the object deforms and a point originally at location $\mathbf{m}$ (i.e. with material coordinates $\mathbf{m}$) moves to a new location $\mathbf{x}(\mathbf{m})$, the *spatial* or *world coordinates* of that point. Since new locations are defined for all material coordinates $\mathbf{m}$, $\mathbf{x}$ is a vector field defined on M. Alternatively, the deformation can also be specified by the *displacement* vector field $\mathbf{u}(\mathbf{m}) = \mathbf{x}(\mathbf{m}) - \mathbf{m}$ defined on M (see Fig. 2). From $\mathbf{u}(\mathbf{m})$ the elastic strain ε is computed (ε is a dimensionless quantity which, in the (linear) 1D case, is simply $\Delta l/l$). A spatially constant displacement field represents a translation of the object with no strain. Therefore, it becomes clear that strain must be measured in terms of spatial variations of the displacement field $\mathbf{u} = \mathbf{u}(\mathbf{m}) = (u, v, w)^T$. Popular choices in Computer Graphics are

$$\varepsilon_G = \frac{1}{2}(\nabla\mathbf{u} + [\nabla\mathbf{u}]^T + [\nabla\mathbf{u}]^T\nabla\mathbf{u}) \quad (1)$$

$$\varepsilon_C = \frac{1}{2}(\nabla\mathbf{u} + [\nabla\mathbf{u}]^T), \quad (2)$$

where the symmetric tensor $\varepsilon_G \in \mathbb{R}^{3x3}$ is Green's nonlinear strain tensor and $\varepsilon_C \in \mathbb{R}^{3x3}$ its linearization, Cauchy's linear strain tensor. The gradient of the displacement field is a 3 by 3 matrix

$$\nabla\mathbf{u} = \begin{bmatrix} u_{,x} & u_{,y} & u_{,z} \\ v_{,x} & v_{,y} & v_{,z} \\ w_{,x} & w_{,y} & w_{,z} \end{bmatrix}, \quad (3)$$

where the index after the comma represents a spatial derivative.

The material law (or *constitutive law*) is used for the computation of the symmetric internal stress tensor $\sigma \in \mathbb{R}^{3x3}$ for

each material point $\mathbf{m}$ based on the strain ε at that point (σ is measured as force per unit area, where $1Pascal = 1Pa = 1N/m^2$). Most Computer Graphics papers use Hooke's linear material law

$$\sigma = \mathbf{E} \cdot \varepsilon, \tag{4}$$

where $\mathbf{E}$ is a rank four tensor which relates the coefficients of the stress tensor linearly to the coefficients of the strain tensor. For isotropic materials, the coefficients of $\mathbf{E}$ only depend on Young's modulus and Poisson's ratio.

2.2. Time Integration

In order to simulate dynamic deformable solids, we need to know the time dependent world coordinates $\mathbf{x}(\mathbf{m},t)$ of all points in M. Given $\mathbf{x}(\mathbf{m},t)$, we can subsequently display the configurations $\mathbf{x}(0),\mathbf{x}(\Delta t),\mathbf{x}(2\Delta t),..$ resulting in an animation of the object. Here Δt is a fixed time step of the simulation and $\mathbf{x}(t)$ represents the entire vector field at time t.

The unknown vector fields $\mathbf{x}(t)$ are not given directly but implicitly as the solution of a differential equation, namely Newton's second law of motion of the form

$$\ddot{\mathbf{x}} = F(\dot{\mathbf{x}},\mathbf{x},t), \tag{5}$$

where $\ddot{\mathbf{x}}$ and $\dot{\mathbf{x}}$ are the second and first time derivatives of $\mathbf{x}$, respectively and $F()$ a general function given by the physical model of the deformable object. In order to find the solution $\mathbf{x}(t)$, this second order differential equation is often rewritten as a coupled set of two first order equations

$$\dot{\mathbf{x}} = \mathbf{v} \tag{6}$$

$$\dot{\mathbf{v}} = F(\mathbf{v},\mathbf{x},t), \tag{7}$$

where the new quantity $\mathbf{v}$ represents $\dot{\mathbf{x}}$. A discrete set of values $\mathbf{x}(0),\mathbf{x}(\Delta t),\mathbf{x}(2\Delta t),..$ of the unknown vector field $\mathbf{x}$ which is needed for the animation can now be obtained by numerically solving (i.e. integrating) this system of equations. Numerical integration of ordinary differential equations is the subject of many textbooks (e.g. [PTVF92, AP98]). See [HES02] for an excellent overview in the context of deformable modeling in computer graphics. We give a few examples here which appear in subsequent sections.

The simplest scheme is explicit (or forward) Euler integration, where the time derivatives are replaced by finite differences $\dot{\mathbf{v}}(t) = [\mathbf{v}(t+\Delta t) - \mathbf{v}(t)]/\Delta t$ and $\dot{\mathbf{x}}(t) = [\mathbf{x}(t+\Delta t) - \mathbf{x}(t)]/\Delta t$. Substituting these into the above equations and solving for the quantities at the next time step $t+\Delta t$ yields

$$\mathbf{x}(t+\Delta t) = \mathbf{x}(t) + \Delta t\mathbf{v}(t) \tag{8}$$

$$\mathbf{v}(t+\Delta t) = \mathbf{v}(t) + \Delta t F(\mathbf{v}(t),\mathbf{x}(t),t). \tag{9}$$

Time integration schemes are evaluated by two main criteria, their stability and their accuracy. Their accuracy is measured by their convergence with respect to the time step size Δt, i.e. first order $O(\Delta t)$, second order $O(\Delta t^2)$, etc. In the field of physically based animation in Computer Graphics, stability is often much more important than accuracy.

The integration scheme presented above is called *explicit* because it provides explicit formulas for the quantities at the next time step. Explicit methods are easy to implement but they are only conditionally stable, i.e. stable only if Δt is smaller than a stability threshold (see [MHTG05] for a formalization). For stiff objects this threshold can be very small. The instability is due to the fact that explicit methods extrapolate a constant right hand side blindly into the future as the above equations show. For a simple spring and a large Δt, the scheme can overshoot the equilibrium position arbitrarily. At the next time step the restoring forces get even larger resulting in an exponential gain of energy and finally an explosion. This problem can be solved by using an *implicit* scheme that uses quantities at the next time step $t+\Delta t$ on both sides of the equation

$$\mathbf{x}(t+\Delta t) = \mathbf{x}(t) + \Delta t\mathbf{v}(t+\Delta t) \tag{10}$$

$$\mathbf{v}(t+\Delta t) = \mathbf{v}(t) + \Delta t F(\mathbf{v}(t+\Delta t),\mathbf{x}(t+\Delta t),t). \tag{11}$$

The scheme is now called implicit because the unknown quantities are *implicitly* given as the solution of a system of equations. Now, instead of extrapolating a constant right hand side blindly into the future, the right hand side is part of the solution process. Remarkably, the implicit (or backward) Euler scheme is stable for arbitrarily large time steps Δt (There is, however, a lower time step limit which, for practical purposes, poses no problem). This gain comes with the price of having to solve an algebraic system of equations at each time step (linear if $F()$ is linear, non-linear otherwise).

A simple improvement to the forward Euler scheme is to swap the order of the equations and use a forward-backward scheme

$$\mathbf{v}(t+\Delta t) = \mathbf{v}(t) + \Delta t F(\mathbf{v}(t),\mathbf{x}(t),t) \tag{12}$$

$$\mathbf{x}(t+\Delta t) = \mathbf{x}(t) + \Delta t\mathbf{v}(t+\Delta t). \tag{13}$$

The update to $\mathbf{v}$ uses forward Euler, while the update to $\mathbf{x}$ uses backward Euler. Note that the method is still explicit; $\mathbf{v}(t+\Delta t)$ is simply evaluated first. For non-dissipative systems (ie. when forces are independent of velocities), this reduces to the second order accurate Stoermer-Verlet scheme. The forward-backward Euler scheme is more stable than standard forward Euler integration, without any additional computational overhead.

3. Lagrangian Mesh Based Methods

3.1. The Finite Element Method

The Finite Element Method (FEM) is one of the most popular methods in Computational Sciences to solve Partial Differential Equations (PDE's) on irregular grids. In order to use the method for the simulation of deformable objects, the object is viewed as a continuous connected volume as

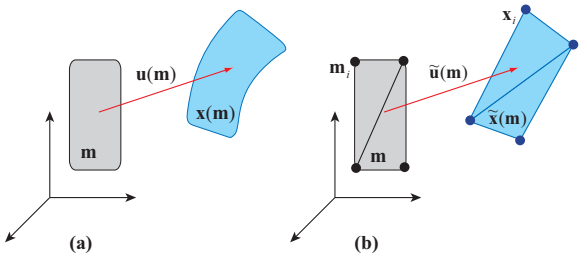

Figure 2: *In the Finite Element method, a continuous deformation (a) is approximated by a sum of (linear) basis functions defined inside a set of finite elements (b).*

in Section 2.1 which is discretized using an irregular mesh. Continuum mechanics, then, provides the PDE to be solved.

The PDE governing dynamic elastic materials is given by

$$\rho\ddot{\mathbf{x}} = \nabla\cdot\sigma + \mathbf{f}, \tag{14}$$

where ρ is the density of the material and $\mathbf{f}$ externally applied forces such as gravity or collision forces. The divergence operator turns the 3 by 3 stress tensor back into a 3 vector

$$\nabla\cdot\sigma = \begin{bmatrix} \sigma_{xx,x} + \sigma_{xy,y} + \sigma_{xz,z} \\ \sigma_{yx,x} + \sigma_{yy,y} + \sigma_{yz,z} \\ \sigma_{zx,x} + \sigma_{zy,y} + \sigma_{zz,z} \end{bmatrix}, \tag{15}$$

representing the internal force resulting from a deformed infinitesimal volume. Eq. 14 shows the equation of motion in *differential form* in contrast to the *integral form* which is used in the Finite Volume method.

The Finite Element Method is used to turn a PDE into a set of algebraic equations which are then solved numerically. To this end, the domain M is discretized into a finite number of disjoint elements (i.e. a mesh). Instead of solving for the spatially continuous function $\mathbf{x}(\mathbf{m},t)$, one only solves for the discrete set of unknown positions $\mathbf{x}_i(t)$ of the nodes of the mesh. First, the function $\mathbf{x}(\mathbf{m},t)$ is approximated using the nodal values by

$$\tilde{\mathbf{x}}(\mathbf{m},t) = \sum_i \mathbf{x}_i(t)\mathbf{b}_i(\mathbf{m}), \tag{16}$$

where $\mathbf{b}_i()$ are fixed nodal basis functions which are 1 at node i and zero at all other nodes, also known as the Kronecker Delta property (see Fig. 2). In the most general case of the Finite Element Method, the basis functions do not have this property. In that case, the unknowns are general parameters which can not be interpreted as nodal values. Substituting $\tilde{\mathbf{x}}(\mathbf{m},t)$ into Eq. 14 results in algebraic equations for the $\mathbf{x}_i(t)$. In the Galerkin approach [Hun05], finding the unknowns $\mathbf{x}_i(t)$ is viewed as an optimization process. When substituting $\mathbf{x}(\mathbf{m},t)$ by the approximation $\tilde{\mathbf{x}}(\mathbf{m},t)$, the infinitely dimensional search space of possible solutions is reduced to a finite dimensional subspace. In general, no function in that subspace can solve the original PDE. The approximation will generate a deviation or residue when substituted into the PDE. In the Galerkin method, the approximation which *minimizes the residue* is sought. In other words, we look for an approximation whose residue is perpendicular to the subspace of functions defined by Eq. 16.

Many papers in Computer Graphics use a simple form of the Finite Element method for the simulation of deformable objects, sometimes called the *explicit* Finite Element Method, which is quite easy to understand and to implement (e.g. [OH99], [DDCB01], [MDM*02]). The explicit Finite Element Method is not to be confused with the standard Finite Element Method being *integrated* explicitly. The explicit Finite Element Method can be integrated either explicitly or implicitly.

In the explicit Finite Element approach, both, the masses and the internal and external forces are lumped to the vertices. The nodes in the mesh are treated like mass points in a mass-spring system while each element acts like a generalized spring connecting all adjacent mass points. The forces acting on the nodes of an element due to its deformation are computed as follows (see for instance [OH99]): given the positions of the vertices of an element and the fixed basis functions, the continuous deformation field $\mathbf{u}(\mathbf{m})$ inside the element can be computed using Eq. 16. From $\mathbf{u}(\mathbf{m})$, the strain field $\varepsilon(\mathbf{m})$ and stress field $\sigma(\mathbf{m})$ are computed. The deformation energy of the element is then given by

$$E = \int_V \varepsilon(\mathbf{m})\cdot\sigma(\mathbf{m})d\mathbf{m}, \tag{17}$$

where the dot represents the componentwise scalar product of the two tensors. The forces can then be computed as the derivatives of the energy with respect to the nodal positions. In general, the relationship between nodal forces and nodal positions is nonlinear. When linearized, the relationship for an element e connecting n_e nodes can simply be expressed as

$$\mathbf{f}_e = \mathbf{K}_e\mathbf{u}_e, \tag{18}$$

where $\mathbf{f}_e \in \mathbb{R}^{3n_e}$ contains the n_e nodal forces and $\mathbf{u}_e \in \mathbb{R}^{3n_e}$ the n_e nodal displacements of an element. The matrix $\mathbf{K}_e \in \mathbb{R}^{3n_e x 3n_e}$ is called the *stiffness matrix* of the element. Because elastic forces coming from adjacent elements add up at a node, a stiffness matrix $\mathbf{K} \in \mathbb{R}^{3nx3n}$ for an enire mesh with n nodes can be formed by assembling the element's stiffness matrices

$$\mathbf{K} = \sum_e \mathbf{K}_e. \tag{19}$$

In this sum, the element's stiffness matrices are expanded to the dimension of $\mathbf{K}$ by filling in zeros at positions related to nodes not adjacent to the element. Using the linearized elastic forces, the linear algebraic equation of motion for an entire mesh becomes ($\mathbf{u} = \mathbf{x} - \mathbf{x}_0$)

$$\mathbf{M}\ddot{\mathbf{u}} + \mathbf{D}\dot{\mathbf{u}} + \mathbf{K}\mathbf{u} = \mathbf{f}_{ext}, \tag{20}$$

where $\mathbf{M} \in \mathbb{R}^{nxn}$ is the mass matrix, $\mathbf{D} \in \mathbb{R}^{nxn}$ the damping

matrix and $\mathbf{f}_{ext} \in \mathbb{R}^n$ externally applied forces. Often, diagonal matrices are used for $\mathbf{M}$ and $\mathbf{D}$, a technique called *mass lumping*. In this case, $\mathbf{M}$ just contains the point masses of the nodes of the mesh on its diagonal. The vectors $\mathbf{x}$ and $\mathbf{x}_0$ contain, respectively, the actual and the rest positions of the nodes.

The Finite Element method only produces a *linear* system of algebraic equations if applied to a *linear* PDE. If a linear strain measure is used and Hooke's law for isotropic materials is substituted into 14, Lamé's linear PDE results:

$$\rho\ddot{\mathbf{x}} = \mu\Delta\mathbf{u} + (\lambda+\mu)\nabla(\nabla\cdot\mathbf{u}), \tag{21}$$

where the material constants λ and μ can be computed directly from Young's modulus and Poisson's ratio. This equation is solved in [DDBC99] in a multiresolution fashion. Using discretized versions of the Laplacian ($\Delta = \nabla^2$) and gradient-of-divergence ($\nabla(\nabla\cdot)$) operators, they solve the Lamé equation on an irregular, multiresolution grid. The system is optimized for limited deformations (linearized strain) and does not support topological changes. Based on Gauss' Divergence Theorem, the discrete operators are further evolved in [DDCB00], which leads to greater accuracy through defined error bounds. Furthermore, the cubic octree hierarchy employed in [DDBC99] is succeeded by a non-nested hierarchy of meshes, in conformance with the redefined operators, which leads to improved shape sampling. In [DDCB01] the previous linearized physical model is replaced by local explicit finite elements and Green's nonlinear strain tensor. To increase stability the simulation is integrated semi-implicitly in time [DSB99].

O'Brien et al. [OH99] and [OBH02] present a FEM based technique for simulating brittle and ductile fracture in connection with elastoplastic materials. They use tetrahedral meshes in connection with linear basis functions $\mathbf{b}_i()$ and Green's strain tensor. The resulting nonlinear equations are solved explicitly and integrated explicitly. The method produces realistic and visually convincing results, but it is not designed for interactive or real-time use. In addition to the strain tensor, they use the so-called *strain rate tensor* (the time derivative of the strain tensor), to compute damping forces. Other studies on the visual simulation of brittle fracture are [SWB00] and [MMDJ01].

Bro-Nielsen and Cotin [BNC96] use linearized finite elements for surgery simulation. They achieve significant speedup by simulating only the visible surface nodes (condensation), similar to the BEM. Many other studies successfully apply the FEM to surgery simulation, such as (but surely not limited to) [GTT89], [CEO*93], [SBMH94], [KGC*96], [CDA99], [CDA00], [PDA00] and [PDA01].

As long as the equation of motion is integrated explicitly in time, nonlinear elastic forces resulting from Green's strain tensor pose no computational problems. The nonlinear formulas for the forces are simply evaluated and used directly to integrate velocities and positions as in [OH99]. As

Figure 3: *The pitbull with its inflated head (left) shows the artifact of linear FEM under large rotational deformations. The correct deformation is shown on the right.*

mentioned earlier (Section 2.2), explicit integration schemes are stable only for small time steps while implicit integration schemes allow arbitrarily large time steps. However, in the latter case, a system of algebraic equations needs to be solved at every time step. Linear PDE's yield linear algebraic systems which can be solved more efficiently and more stably than nonlinear ones. Unfortunately, linearized elastic forces are only valid for small deformations. Large rotational deformations yield highly inaccurate restoring forces (see Fig. 3).

To eliminate these artifacts, Müller et al. extract the rotational part of the deformation for each finite element and compute the forces with respect to the non-rotated reference frame [MG04]. The linear equation 18 for the elastic forces of an element (in this case a tetrahedron) is replaced by

$$\mathbf{f} = \mathbf{R}\mathbf{K}(\mathbf{R}^T\mathbf{x} - \mathbf{x}_0), \tag{22}$$

where $\mathbf{R} \in \mathbb{R}^{12x12}$ is a matrix that contains four 3 by 3 identical rotation matrices along its diagonal. The vector $\mathbf{x}$ contains the actual positions of the four adjacent nodes of the tetrahedron while $\mathbf{x}_0$ contains their rest positions. The rotation of the element used in $\mathbf{R}$ is computed by performing a polar decomposition of the matrix that describes the transformation of the tetrahedron from the configuration $\mathbf{x}_0$ to the configuration $\mathbf{x}$. This yields stable, fast and visually pleasing results. In an earlier approach, they extract the rotational part not per element but per node [MDM*02]. In this case, the global stiffness matrix does not need to be reassembled at each time step but ghost forces are introduced.

Another solution to this problem is proposed in [CGC*02]: each region of the finite element mesh is associated with the bone of a simple skeleton and then locally linearized. The regions are blended in each time step, leading to results which are visually indistinguishable from the nonlinear solution, yet much faster.

An adaptive nonlinear FEM simulation is described by Wu et al. [WDGT01]. Distance, surface and volume preservation for triangular and tetrahedral meshes is outlined in [THMG04]. Grinspun et al. introduce conforming, hierarchical, adaptive refinement methods (CHARMS) for general finite elements [GKS02], refining basis functions instead of elements. Irving et al. [ITF04] present a method which robustly handles large deformation and element inversion by